

Package ‘tidycreel’

October 10, 2026

Title Tidy Interface for Creel Survey Design and Analysis

Version 8.0.0

Description Provides a tidy, pipe-friendly interface for creel survey design, data management, estimation, visualisation, and reporting. A creel survey interviews anglers on site to estimate fishing effort, catch, and harvest for a water body. Built on the 'survey' package for design-based inference, with support for instantaneous, bus-route, ice, camera, and aerial designs. Catch-rate estimators follow Hoenig, Jones, Pollock, Robson and Wade (1997) <[doi:10.2307/2533116](https://doi.org/10.2307/2533116)>; bus-route designs follow Kinloch, McGlennon, Nicoll and Pike (1997) <[doi:10.1016/s0165-7836\(97\)00068-4](https://doi.org/10.1016/s0165-7836(97)00068-4)>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports checkmate, cli, dplyr, generics, ggplot2, lifecycle, rlang, stats, survey, tibble, tidyselect (>= 1.2.0)

Suggests bench, covr, DBI, duckdb, glmmTMB, hedgehog, htmlwidgets, knitr, lintr, lme4, lubridate (>= 1.9.0), pkgdown, pkgload, quickcheck, profvis, readxl (>= 1.4.0), rmarkdown, styler, testthat (>= 3.0.0), withr, writexl (>= 1.5.4), zipcodeR

VignetteBuilder knitr

Config/testthat/edition 3

URL <https://github.com/chrischizinski/tidycreel>,
<https://chrischizinski.com/tidycreel/>

BugReports <https://github.com/chrischizinski/tidycreel/issues>

NeedsCompilation no

Author Christopher Chizinski [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0001-9294-2588>>)

Maintainer Christopher Chizinski <cchizinski2@unl.edu>

Repository CRAN

Date/Publication 2026-10-10 11:20:02 UTC

Contents

add_ages	5
add_catch	6
add_counts	8
add_interviews	13
add_lengths	18
add_sections	20
adjust_nonresponse	22
as.data.frame.creel_data_validation	24
as.data.frame.creel_design_comparison	24
as.data.frame.creel_validation_report	25
as.data.frame.creel_variance_comparison	25
as_creel_svydesign	26
as_hybrid_svydesign	27
attach_count_times	30
audit_strata	31
autoplot.creel_design_comparison	33
autoplot.creel_estimates	34
autoplot.creel_length_distribution	35
autoplot.creel_schedule	36
check_completeness	37
compare_cpue_estimators	38
compare_designs	40
compare_variance	41
compute_angler_effort	43
compute_effort	44
creel_confidence_intervals	45
creel_counts_toy	47
creel_design	48
creel_interviews_toy	53
creel_n_camera	54
creel_n_cpue	56
creel_n_effort	57
creel_palette	59
creel_power	59
creel_schema	61
creel_vocabulary	65
cv_from_n	66
day_length	67
derive_angler_count	69
estimate_angler_n	72
estimate_angler_trips	76

estimate_catch_rate	77
estimate_effort	82
estimate_effort_aerial_glm	85
estimate_effort_per_acre	88
estimate_exploitation_rate	89
estimate_harvest_rate	93
estimate_mr_harvest	97
estimate_release_rate	99
estimate_total_catch	103
estimate_total_harvest	108
estimate_total_release	113
est_age_distribution	117
est_biomass	119
est_compliance	122
est_effort_camera	124
est_effort_camera_mi	128
est_length_distribution	130
est_mean_age	133
est_mean_length	134
example_aerial_counts	136
example_aerial_glm_counts	137
example_aerial_interviews	139
example_ages	140
example_calendar	141
example_camera_counts	142
example_camera_interviews	143
example_camera_timestamps	145
example_catch	146
example_counts	148
example_ice_interviews	149
example_ice_sampling_frame	150
example_interviews	152
example_lengths	153
example_sections_calendar	155
example_sections_counts	156
example_sections_interviews	157
flag_outliers	158
format.creel_completeness_report	160
format.creel_design	160
format.creel_design_report	161
format.creel_estimates	161
format.creel_estimates_diagnostic	162
format.creel_estimates_mor	162
format.creel_schedule	163
format.creel_schema	163
format.creel_season_summary	164
format.creel_tost_validation	164
format.creel_validation	165

generate_bus_schedule	165
generate_count_times	167
generate_progressive_start	169
generate_schedule	171
get_enumeration_counts	173
get_inclusion_probs	174
get_sampling_frame	175
get_site_contributions	176
impute_camera_counts	178
mean_party_size	181
new_creel_schedule	182
optimal_n	183
plot_design	185
power_creel	186
preprocess_camera_timestamps	189
prep_counts_boat_party	190
prep_counts_daily_effort	192
prep_interviews_trips	194
prep_interview_catch	196
print.creel_completeness_report	197
print.creel_data_validation	197
print.creel_design	198
print.creel_design_comparison	198
print.creel_design_report	199
print.creel_estimates	199
print.creel_estimates_diagnostic	200
print.creel_estimates_mor	200
print.creel_hybrid_svydesign	201
print.creel_schedule	201
print.creel_schema	202
print.creel_season_summary	203
print.creel_summary	203
print.creel_tost_validation	204
print.creel_validation	204
print.creel_validation_report	205
print.creel_variance_comparison	205
read_schedule	206
reallocate_strata	207
season_summary	208
simulate_creel_catch	209
simulate_creel_data	211
simulate_strata_collapse	215
standardize_species	216
summarize_boat_composition	218
summarize_by_angler_type	220
summarize_by_county	221
summarize_by_day_type	222
summarize_by_method	223

summarize_by_species_sought	225
summarize_by_trip_length	226
summarize_by_zip	227
summarize_cws_rates	228
summarize_hws_rates	231
summarize_length_freq	233
summarize_refusals	235
summarize_successful_parties	236
summarize_trips	238
summary.creel_estimates	239
theme_creel	240
tidy.creel_estimates	241
validate_creel_data	242
validate_creel_schedule	243
validate_creel_schema	244
validate_design	245
validate_incomplete_trips	247
validation_report	250
write_estimates	251
write_schedule	253

Index 255

add_ages	<i>Attach age data to a creel design</i>
----------	--

Description

add_ages() attaches a data frame of individual fish age records (from scale, fin ray, or otolith samples) to a creel_design object. The age data are linked to interviews via a shared identifier, analogous to [add_lengths\(\)](#).

Usage

```
add_ages(design, data, age_uid, interview_uid, species, age, age_type)
```

Arguments

design	A creel_design object with interviews attached.
data	A data frame of age records. One row per aged fish.
age_uid	Unquoted column in data — the column that holds the interview identifier, linking each age record to its interview (the foreign key; analogous to length_uid in add_lengths()).
interview_uid	Unquoted column in design\$interviews — the interview identifier column in the design, used as the join target.
species	Unquoted column in data — species name or code.
age	Unquoted column in data — estimated age (integer or numeric).
age_type	Unquoted column in data — fate of the fish: "harvest" or "release".

Value

A creel_design object with age data attached in design\$ages and associated column-name slots.

See Also

[add_lengths\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_ages)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
design <- add_ages(design, example_ages,
  age_uid      = interview_id,
  interview_uid = interview_id,
  species      = species,
  age          = age,
  age_type     = age_type
)
head(design$ages)
```

add_catch

Attach species-level catch data to a creel design

Description

Attaches a long-format data frame of species-level catch data to a creel_design object. Each row in data represents a species-catch-type combination for a single interview. Data is validated at attach time and stored on the design for use by downstream summary and estimation functions.

Usage

```
add_catch(design, data, catch_uid, interview_uid, species, count, catch_type)
```

Arguments

design	A creel_design object created by creel_design .
data	A data frame in long format: one row per species per catch type per interview.
catch_uid	<tidyselect> Column in data containing interview IDs (the catch-side join key).
interview_uid	<tidyselect> Column in design\$interviews containing the matching interview IDs.

species	<tidyselect> Column in data containing species names or codes.
count	<tidyselect> Column in data containing fish counts (non-negative integer or numeric).
catch_type	<tidyselect> Column in data containing catch fate: one of "caught", "harvested", or "released". Values are normalized to lowercase before validation.

Details

Catch type model: Each species-interview row carries one of three catch types. "caught" is the total; "harvested" and "released" are subsets. A "caught" row is optional — when absent, total catch is inferred as harvested + released. When a "caught" row is present, caught >= harvested + released is enforced (CATCH-04).

Interview ID validation: Every interview ID appearing in data must appear in design\$interviews[[interview_uid]]. Interviews with no catch rows are valid (anglers who caught nothing need not appear in catch data).

Counts must be known (CATCH-07): count may not contain NA. A *missing row* carries a definite meaning here — none of that disposition, or, for a "caught" row, derive the total from harvested + released — so a row that is present but carries an unknown count is silently read as that same definite thing rather than as an unknown. Fill the missing counts, or drop those rows; note that dropping states something, since a dropped "caught" row changes how the total is derived rather than setting it to zero. Aborts with class `creel_error_na_catch_count`.

Immutability: Returns a new `creel_design` — the input is not modified. Calling `add_catch()` on a design that already has `$catch` is an error.

Value

A new `creel_design` object with `$catch` and associated `$catch*_col` fields attached.

See Also

Other "Survey Design": `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
data(example_calendar)
data(example_interviews)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, trip_duration = trip_duration
)
design <- add_catch(design, example_catch,
  catch_uid = interview_id,
  interview_uid = interview_id,
```

```

    species = species,
    count = count,
    catch_type = catch_type
  )
  print(design)

```

add_counts

Attach count data to a creel design

Description

Attaches count-side effort data to a `creel_design` object and constructs the internal survey design object eagerly. The preferred workflow is to standardize raw count-process data into sampled-day effort rows with `prep_counts_daily_effort()` (or another `prep_counts_*()` helper) before calling `add_counts()`. This keeps survey-specific count reconstruction logic out of the core estimator path.

Compatibility paths for raw-ish count inputs remain supported: `add_counts()` can still aggregate multiple within-day observations via `count_time_col` and can still compute progressive-count daily effort from `circuit_time` and `period_length_col`. Eager construction catches design errors at `add_counts()` time when users have context about what data they are adding.

`period_length_col` applies to instantaneous counts as well as progressive ones. An instantaneous count is a snapshot of the anglers present at one moment, so it becomes effort only once multiplied by the period it was randomised within; supply that column and the estimate is in angler-hours.

Usage

```

add_counts(
  design,
  counts,
  count_col = NULL,
  psu = NULL,
  count_time_col = NULL,
  count_type = "instantaneous",
  circuit_time = NULL,
  period_length_col = NULL,
  unit_cols = NULL,
  allow_invalid = FALSE
)

```

Arguments

<code>design</code>	A <code>creel_design</code> object (created with <code>creel_design()</code>)
<code>counts</code>	Data frame containing count-side effort data. Preferred input: sampled-day effort rows that already have a <code>Date</code> column matching the design's <code>date_col</code> , all strata columns from the design's <code>strata_cols</code> , at least one numeric effort column,

and a PSU column (specified via `psu`, defaults to `date_col`). Use `prep_counts_daily_effort()` to build this form from raw source data.

Compatibility input paths remain available for raw-ish count workflows with multiple within-day observations or progressive counts.

<code>count_col</code>	Tidy selector for the numeric column holding the angler counts. Defaults to <code>NULL</code> , in which case the column is inferred as the only numeric column that is not design metadata (<code>date</code> , <code>strata</code> , <code>PSU</code> , <code>section</code> , <code>count_time</code> , or <code>period_length</code>). If more than one numeric column qualifies, <code>add_counts()</code> aborts and lists the candidates rather than choosing by position — name the column here to resolve it. The resolved name is stored on the design and used by every downstream estimator.
<code>psu</code>	Character string naming the PSU (Primary Sampling Unit) column in the count data. Defaults to <code>NULL</code> , which uses the design's <code>date_col</code> as the PSU (day-as-PSU is the most common creel design). For other designs, specify the PSU column explicitly (e.g., <code>"site_day"</code> for day-site PSUs).
<code>count_time_col</code>	Tidy selector for a column that identifies distinct sub-PSU count observations (e.g., <code>count_time_col = count_time</code> where <code>count_time</code> contains <code>"am"</code> / <code>"pm"</code>). When supplied, multiple rows per PSU are aggregated to a single PSU-level mean ($\bar{C}_d$) and within-day variance components (SS_d , K_d) are stored in <code>design\$within_day_var</code> . When <code>NULL</code> (default), a single row per PSU is expected; duplicate PSU rows emit a CNT-06 warning.
<code>count_type</code>	Character string specifying the count method. Must be <code>"instantaneous"</code> (default) or <code>"progressive"</code> . When <code>"progressive"</code> , <code>circuit_time</code> and <code>period_length_col</code> are required.
<code>circuit_time</code>	Numeric. Circuit duration τ in hours — the time required to complete one roving count circuit of the water body. Required when <code>count_type = "progressive"</code> (CNT-05). Used to compute $\kappa = T_d / \tau$ and $\hat{E}_d = C \times \tau \times \kappa$. Ignored (with a warning) when <code>count_type = "instantaneous"</code> .
<code>period_length_col</code>	Tidy selector for the column containing T_d — the length in hours of the period each count was randomised within. Required when <code>count_type = "progressive"</code> (CNT-05), optional but strongly recommended when <code>count_type = "instantaneous"</code> . Values must be positive and finite. The column is dropped from <code>design\$counts</code> after $\hat{E}_d$ is computed (it must not be passed to <code>estimate_effort()</code> as a count variable).
<code>unit_cols</code>	Optional character vector naming the columns that together identify one sampling unit. When omitted, the unit is inferred from the design: the PSU column plus any <code>strata</code> , <code>section</code> , and <code>site</code> columns. Supply it when the counts table carries a dimension the design does not declare. The commonest case is <code>effort_type</code> , which <code>prep_counts_daily_effort()</code> emits: bank and boat counts on the same day are two units, not one day counted twice. Inference cannot see such a column, so it would treat those rows as repeats — warning about them without <code>count_time_col</code> , and averaging across them with it (GH #162). You are not required to guess when this matters. If aggregation would collapse rows that differ in an undeclared column, <code>add_counts()</code> aborts and names the column rather than silently taking its first value.

For instantaneous counts, supplying this is what makes the estimate angler-hours. A count is a snapshot of how many anglers were present at one moment; effort is that count times the period it was randomised within, $\hat{E}_d = \bar{C}_d \times T_d$ (Hoenig et al. 1993). Without it, `estimate_effort()` expands the counts to the season and returns them unmultiplied, and warns once per session that it has done so.

Not accepted on aerial designs, which carry their period length as `h_open` and scale the count by `h_open / v` when estimating. Supplying both would apply time twice, so this errors rather than choosing one.

`T_d` is a property of the survey protocol — the period set by regulation, access hours, or field practice. It is not astronomical daylight. `day_length()` is for simulation and planning; do not feed it here as a substitute for the period actually surveyed.

The multiplication happens per PSU, before aggregation. Converting after the fact computes $\bar{C} \times \bar{T}$ where the target is the mean of $C \times T$; the two differ by $\text{Cov}(C, T)$, which is positive in practice because anglers fish more on long days, so the collapsed form biases low.

`allow_invalid` Logical flag for validation behavior. If FALSE (default), validation failures abort with detailed error messages. If TRUE, validation failures generate warnings and attach counts anyway (use with caution).

Value

A new `creel_design` object (list) with components:

<code>calendar</code>	Original calendar data frame
<code>date_col</code>	Character name of date column
<code>strata_cols</code>	Character vector of strata column names
<code>site_col</code>	Character name of site column, or NULL
<code>design_type</code>	Character design type
<code>counts</code>	The count data frame (newly attached)
<code>psu_col</code>	Character name of PSU column
<code>survey</code>	Internal <code>survey.design2</code> object (newly constructed)
<code>validation</code>	<code>creel_validation</code> object with Tier 1 results

Immutability

`add_counts()` follows functional programming patterns and returns a new `creel_design` object. The original design object is not modified. This prevents accidental data loss and makes the workflow explicit: `design2 <- add_counts(design, counts)` not `add_counts(design, counts)`

Validation

`add_counts()` performs Tier 1 validation:

- Count data schema (Date column, numeric column) via `validate_count_schema()`

- Design column presence (date_col, strata_cols, PSU in count data)
- No NA values in design-critical columns (date, strata, PSU)
- Survey construction (catches lonely PSU errors, stratification issues)

Party-size expansion carriers

`derive_angler_count()` attaches `expansion_basis`, `expansion_se`, `expansion_group`, and `expansion_of` to the counts table so the party-size sampling error can reach the effort standard error. The four are written together and must travel together: `add_counts()` aborts on a table carrying only some of them, since that state can only come from partial deletion and would otherwise drop the variance component while leaving an `expansion_se` visible in the data.

`add_counts()` also aborts when `count_col` is not the column named in `expansion_of`. The basis is $d(\text{count})/d(\text{party_size})$, so a count transformed after `derive_angler_count()` no longer matches the basis carried beside it, and the variance component would come out understated by exactly the scale factor while still reading as propagated. Supply the untransformed count and `period_length_col`, which scales the count and the basis together.

Dropping all four — which an ordinary `select()` will do — is indistinguishable here from counts that never had them, and is silent.

PSU Specification

Per user decision (clarified 2026-02-08), PSU is specified only in `add_counts()`, not in `creel_design()` constructor. PSU is only meaningful when count data is present, making `add_counts()` the correct abstraction boundary. This design allows the same `creel_design` calendar to be used with different PSU structures.

See Also

Other "Survey Design": `add_catch()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
# Preferred workflow: standardize sampled-day effort before attachment
calendar <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend")
)
design <- creel_design(calendar, date = date, strata = day_type)

raw_counts <- data.frame(
  sample_date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend"),
  effort_kind = c("bank", "bank", "bank", "bank"),
  effort_value = c(15, 23, 45, 52)
)
```

```

counts_ready <- prep_counts_daily_effort(
  raw_counts,
  date = sample_date,
  strata = day_type,
  effort_type = effort_kind,
  daily_effort = effort_value
)

design_with_counts <- add_counts(design, counts_ready)
print(design_with_counts)

# Compatibility path: raw count rows with a custom PSU column
counts_with_site_psu <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend"),
  site_day = paste0("site_", 1:4),
  count = c(15, 23, 45, 52)
)

design2 <- add_counts(design, counts_with_site_psu, psu = "site_day")

# Compatibility path: multiple counts per day (within-day variance via count_time_col)
# Two circuits per day: "am" and "pm"
calendar2 <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend")
)
design3 <- creel_design(calendar2, date = date, strata = day_type)

multi_counts <- data.frame(
  date = as.Date(rep(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04"), each = 2)),
  day_type = rep(c("weekday", "weekday", "weekend", "weekend"), each = 2),
  count_time = rep(c("am", "pm"), 4),
  n_anglers = c(12, 18, 20, 26, 40, 50, 48, 56)
)
design_multi <- add_counts(design3, multi_counts,
  count_time_col = count_time # nolint: object_usage_linter
)

# Compatibility path: progressive count type ( $\hat{E}_d = C \times \text{circuit\_time} \times \text{kappa}$ )
calendar3 <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend")
)
design4 <- creel_design(calendar3, date = date, strata = day_type)

prog_counts <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend"),
  n_anglers = c(15L, 23L, 45L, 52L),
  shift_hours = rep(8, 4)
)

```

```

design_prog <- add_counts(
  design4, prog_counts,
  count_type = "progressive",
  circuit_time = 2,
  period_length_col = shift_hours # nolint: object_usage_linter
)

```

add_interviews

Attach interview data to a creel design

Description

Attaches interview-side trip data (catch, effort, and optionally harvest per fishing trip) to a `creel_design` object and constructs the internal interview survey design object eagerly. The preferred workflow is to standardize raw interview exports into canonical trip rows with `prep_interviews_trips()` before calling `add_interviews()`, and to standardize species-level catch detail with `prep_interview_catch()` before calling `add_catch()`.

`add_interviews()` still supports direct attachment of raw-ish interview data with tidy selectors for catch, effort, and trip metadata. This keeps current workflows working while making the prep-helper seam explicit.

Usage

```

add_interviews(
  design,
  interviews,
  catch,
  effort,
  harvest = NULL,
  trip_status,
  trip_duration = NULL,
  trip_start = NULL,
  interview_time = NULL,
  n_counted = NULL,
  n_interviewed = NULL,
  angler_type = NULL,
  angler_method = NULL,
  species_sought = NULL,
  n_anglers = 1L,
  refused = NULL,
  date_col = NULL,
  interview_type = c("access", "roving"),
  allow_invalid = FALSE
)

```

Arguments

design	A creel_design object (created with <code>creel_design()</code>)
interviews	Data frame containing interview data. Must have: • A Date column matching the design's date_col • Numeric catch column (total fish caught per trip) • Numeric effort column (fishing time per trip, e.g., hours) • Character trip status column ("complete" or "incomplete") • Optional numeric harvest column (fish kept per trip) • Optional numeric trip duration column (hours) OR trip_start + interview_time columns (POSIXct)
catch	Tidy selector for total catch column (required). Use bare column names (e.g., catch = catch_total) or tidyselect helpers.
effort	Tidy selector for fishing effort column (required, e.g., effort = hours_fished). Should represent time spent fishing per trip.
harvest	Tidy selector for harvest (kept fish) column (optional, default NULL). If provided, will be validated for consistency (harvest <= catch).
trip_status	Tidy selector for trip completion status column (required). Must contain "complete" or "incomplete" (case-insensitive). This is essential for downstream incomplete trip estimators.
trip_duration	Tidy selector for trip duration column in hours (optional, default NULL). Provide either trip_duration OR trip_start + interview_time, not both. Duration values must be positive and >= 1/60 hours (1 minute).
trip_start	Tidy selector for trip start time column (optional, default NULL). Must be POSIXct or POSIXlt. Requires interview_time to calculate duration. Use when duration needs to be calculated from timestamps.
interview_time	Tidy selector for interview time column (optional, default NULL). Must be POSIXct or POSIXlt. Requires trip_start to calculate duration. Duration is calculated as interview_time - trip_start in hours.
n_counted	Tidy selector for the count of all anglers observed at the site during the sampling period (required for bus-route designs, ignored for other designs). Values must be non-negative integers. Must satisfy n_counted >= n_interviewed.
n_interviewed	Tidy selector for the count of anglers actually interviewed at the site (required for bus-route designs, ignored for other designs). Values of 0 are valid (no anglers came off the water).
angler_type	Tidy selector for angler type column (optional, default NULL). Use bare column names (e.g., angler_type = angler_type). Common values are "bank" and "boat". Not validated in Phase 28; downstream summary functions use this field.
angler_method	Tidy selector for angler method column (optional, default NULL). Use bare column names (e.g., angler_method = method_code). Records the fishing technique employed (e.g., "fly", "spin", "bait").
species_sought	Tidy selector for species sought column (optional, default NULL). Use bare column names (e.g., species_sought = target_species). Records the species the angler was targeting during the interview.

n_anglers	Number of anglers in the party. Either a bare column name (e.g. <code>n_anglers = party_size</code>) or a single positive number stating a constant party size (e.g. <code>n_anglers = 1</code> when every interview is one angler). A bare number is read as a party size, not as a tidyselect column position, so <code>n_anglers = 1</code> means one angler per party rather than "column 1". Values must be positive and finite; missing and non-integer values warn. When omitted, effort is left un-normalised and a <code>cli_warn()</code> message states the assumption. In that case the rate estimators return quantities per <i>party</i>-hour, and the product totals warn when they multiply one by count-derived angler-hours. Pass <code>n_anglers = 1</code> to state that the interviews really are one angler each; that is a declaration, not a default, and it silences the warning.
refused	Tidy selector for the refused interview flag column (optional, default NULL). Use bare column names (e.g., <code>refused = refused_flag</code>). Values should be logical (TRUE/FALSE) or coercible to logical.
date_col	Character name of date column in interviews (default NULL, which uses the design's <code>date_col</code>). Specify explicitly if interview data uses a different date column name than the design calendar.
interview_type	Character: "access" (complete trips at access point) or "roving" (incomplete trips during fishing). Default is "access". When set to "roving", estimation functions automatically default to using all interviews (complete + incomplete) via the mean-of-ratios (MOR) estimator rather than restricting to complete trips. Override the auto-routing by passing <code>use_trips</code> or <code>estimator</code> explicitly.
allow_invalid	Logical flag for validation behavior. If FALSE (default), validation failures abort with detailed error messages. If TRUE, validation failures generate warnings and attach interviews anyway (use with caution).

Value

A new `creel_design` object (list) with components:

calendar	Original calendar data frame
date_col	Character name of date column
strata_cols	Character vector of strata column names
site_col	Character name of site column, or NULL
design_type	Character design type
counts	Count data frame (if previously attached, or NULL)
interviews	The interview data frame (newly attached)
catch_col	Character name of catch column
effort_col	Character name of effort column
harvest_col	Character name of harvest column, or NULL
trip_status_col	Character name of trip status column
trip_duration_col	Character name of trip duration column, or NULL

trip_start_col Character name of trip start time column, or NULL
 interview_time_col
 Character name of interview time column, or NULL
 interview_type Character interview type
 interview_survey
 Internal survey.design2 object (newly constructed)
 validation creel_validation object with Tier 1 results

Immutability

add_interviews() follows functional programming patterns and returns a new creel_design object. The original design object is not modified. This prevents accidental data loss and makes the workflow explicit: design2 <- add_interviews(design, interviews, ...) not add_interviews(design, interviews, ...)

Validation

add_interviews() performs Tier 1 validation:

- Interview data schema (Date column, numeric columns) via validate_interview_schema()
- Design column presence (date_col exists in interview data)
- No NA values in date column
- Interview dates exist in design calendar
- Catch and effort columns exist and are numeric
- Harvest column exists and is numeric (if provided)
- Harvest <= catch consistency (if harvest provided)
- Trip status valid ("complete" or "incomplete", case-insensitive)
- Trip status has no NA values
- Trip duration/time inputs are mutually exclusive (error if both provided)
- Trip duration is positive, >= 1 minute, warns if > 48 hours
- Trip start + interview_time are POSIXct/POSIXlt, calculate valid duration
- Interview survey construction (catches stratification issues)

Calendar Integration

Interview dates are automatically linked to the design calendar via date matching. Strata from the calendar are inherited by the interview data, enabling stratified estimation of catch rates.

See Also

Other "Survey Design": [add_catch\(\)](#), [add_counts\(\)](#), [add_lengths\(\)](#), [add_sections\(\)](#), [as_creel_svydesign\(\)](#), [as_hybrid_svydesign\(\)](#), [compute_angler_effort\(\)](#), [compute_effort\(\)](#), [creel_design\(\)](#), [creel_schema\(\)](#), [creel_vocabulary\(\)](#), [derive_angler_count\(\)](#), [est_effort_camera\(\)](#), [impute_camera_counts\(\)](#), [mean_party_size\(\)](#), [prep_counts_boat_party\(\)](#), [prep_counts_daily_effort\(\)](#), [prep_interview_catch\(\)](#), [prep_interviews_trips\(\)](#), [validate_creel_schema\(\)](#)

Examples

```
# Preferred workflow: standardize interview rows before attachment
calendar <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend")
)
design <- creel_design(calendar, date = date, strata = day_type)

raw_interviews <- data.frame(
  survey_date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  catch_total = c(5, 3, 7, 2),
  hours_fished = c(2.0, 2.5, 3.0, 1.5),
  trip_status = c("complete", "complete", "incomplete", "complete"),
  trip_duration = c(2.0, 2.5, 1.5, 1.5),
  interview_id = 1:4
)

interviews_ready <- prep_interviews_trips(
  raw_interviews,
  date = survey_date,
  interview_uid = interview_id,
  effort_hours = hours_fished,
  trip_status = trip_status,
  trip_duration = trip_duration,
  catch_total = catch_total
)

design_with_interviews <- add_interviews(
  design, interviews_ready,
  catch = catch_total,
  effort = effort_hours,
  trip_status = trip_status,
  trip_duration = trip_duration
)
print(design_with_interviews)

# Compatibility path: direct attachment with harvest column and timestamps
interviews2 <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  catch_total = c(5, 3, 7, 2),
  catch_kept = c(2, 1, 5, 2),
  hours_fished = c(2.0, 2.5, 3.0, 1.5),
  trip_status = c("complete", "incomplete", "complete", "complete"),
  trip_start = as.POSIXct(c(
    "2024-06-01 08:00", "2024-06-02 09:00",
    "2024-06-03 07:00", "2024-06-04 10:00"
  )),
  interview_time = as.POSIXct(c(
    "2024-06-01 10:00", "2024-06-02 11:30",
    "2024-06-03 10:00", "2024-06-04 11:30"
  ))
)
```

```
design2 <- add_interviews(
  design, interviews2,
  catch = catch_total,
  effort = hours_fished,
  harvest = catch_kept,
  trip_status = trip_status,
  trip_start = trip_start,
  interview_time = interview_time
)
```

add_lengths

Attach fish length frequency data to a creel design

Description

Attaches a long-format data frame of fish length measurements to a `creel_design` object. Supports both individual measurements (harvest) and binned counts (release). Data is validated at attach time and stored on the design for use by downstream summary and estimation functions.

Usage

```
add_lengths(
  design,
  data,
  length_uid,
  interview_uid,
  species,
  length,
  length_type,
  count = NULL,
  release_format = "individual"
)
```

Arguments

<code>design</code>	A <code>creel_design</code> object created by creel_design .
<code>data</code>	A data frame in long format: one row per fish measurement or length bin per species per interview.
<code>length_uid</code>	<tidyselect> Column in data containing interview IDs (the length-side join key).
<code>interview_uid</code>	<tidyselect> Column in <code>design\$interviews</code> containing the matching interview IDs.
<code>species</code>	<tidyselect> Column in data containing species names or codes.

length	<tidyselect> Column in data containing length values. For harvest rows (when <code>release_format = "individual"</code>), must be numeric (mm). For release rows when <code>release_format = "binned"</code> , may be a character bin label such as <code>"300-350"</code> .
length_type	<tidyselect> Column in data containing the measurement fate: one of <code>"harvest"</code> or <code>"release"</code> . Values are normalized to lowercase before validation.
count	<tidyselect> Optional. Column in data containing fish counts for binned release rows. Required when <code>release_format = "binned"</code> and release rows are present. Harvest rows should have NA in this column. Omit (or pass NULL) when all length data are individual measurements.
release_format	Character scalar: <code>"individual"</code> (default) or <code>"binned"</code> . Controls how release rows are validated and how the length range is computed for display.

Details

Mixed column type footgun: The length column may contain both numeric values (harvest rows) and character bin labels (release rows). R will coerce the entire column to character when mixing types in a data frame. `add_lengths()` validates harvest row lengths by subsetting to harvest rows first, then attempting as.numeric() coercion, to avoid errors from the mixed-type column.

Interview ID validation: Every interview ID appearing in data must appear in `design$interviews[[interview_uid]]`. Interviews with no length rows are valid.

Immutability: Returns a new `creel_design` — the input is not modified. Calling `add_lengths()` on a design that already has `$lengths` is an error.

Value

A new `creel_design` object with `$lengths` and associated `$lengths*_col` fields attached.

See Also

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
data(example_calendar)
data(example_interviews)
data(example_lengths)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, trip_duration = trip_duration
)
design <- add_lengths(design, example_lengths,
  length_uid = interview_id,
  interview_uid = interview_id,
```

```

    species = species,
    length = length,
    length_type = length_type,
    count = count,
    release_format = "binned"
  )
  print(design)

```

add_sections

Register spatial sections for a creel survey design

Description

Attaches a sections registry to a `creel_design` object. Once sections are registered, all subsequent calls to `add_counts()` and `add_interviews()` validate that every row's section value matches a registered section name. Unrecognised section values abort with an informative error identifying the bad values and listing valid options.

`add_sections()` is optional for single-section surveys. Call it when your survey covers multiple named sections and you want early detection of mislabelled data (e.g. "NRTH" instead of "NORTH").

Usage

```

add_sections(
  design,
  sections,
  section_col,
  description_col = NULL,
  area_col = NULL,
  shoreline_col = NULL
)

```

Arguments

<code>design</code>	A <code>creel_design</code> object (created with <code>creel_design()</code>).
<code>sections</code>	A data frame with one row per section. Must contain the column identified by <code>section_col</code> . Optional metadata columns are identified by <code>description_col</code> , <code>area_col</code> , and <code>shoreline_col</code> .
<code>section_col</code>	Tidy selector for the column in <code>sections</code> that holds section names or IDs. Must be character or factor. No duplicate values are permitted.
<code>description_col</code>	Optional tidy selector for a free-text description column (e.g. "North inlet", "Main basin"). Stored for reporting only.
<code>area_col</code>	Optional tidy selector for a surface area column (numeric, ha). All values must be strictly positive. Stored now; used in v0.8.0 aerial survey estimation.
<code>shoreline_col</code>	Optional tidy selector for a shoreline length column (numeric, km). All values must be strictly positive. Stored now; used in v0.8.0 aerial survey estimation.

Value

A new `creel_design` object with `$sections` and `$section_col` populated. The input design is not modified.

Validation performed by downstream functions

After `add_sections()` is called, `add_counts()` and `add_interviews()` check that every row's section value is present in `design$sections[[design$section_col]]`. An unrecognised value produces a `cli_abort()` naming the bad values and listing valid section names.

How sections are named in results

Every sectioned estimate reports its sections in a column named after `section_col`, as a design declaring `strata = day_type` reports a `day_type` column. Register sections under reach and the result's first column is reach, so it joins back to your own section table by name. Read it as `est[[design$section_col]]` rather than assuming a fixed name (#282).

The lake-wide aggregate row, where requested, is a *value* in that same column – the reserved name `.lake_total` – not a separate column.

See Also

`creel_design()`, `add_counts()`, `add_interviews()`

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
cal <- data.frame(
  date = as.Date(c(
    "2024-06-01", "2024-06-02",
    "2024-06-03", "2024-06-04"
  )),
  day_type = c("weekday", "weekday", "weekend", "weekend")
)
design <- creel_design(cal, date = date, strata = day_type)

my_sections <- data.frame(
  section      = c("North Inlet", "Main Basin", "South Outlet"),
  description  = c("Tributary inlet", "Open water", "Dam outlet"),
  area_ha      = c(45.0, 820.0, 12.0),
  shoreline_km = c(8.2, 62.1, 3.4)
)

design2 <- add_sections(design, my_sections,
  section_col      = section,
  description_col  = description,
  area_col         = area_ha,
```

```

    shoreline_col = shoreline_km
  )

```

adjust_nonresponse	<i>Adjust a creel design for nonresponse bias</i>
--------------------	---

Description

Applies nonresponse weighting to a `creel_design` object by scaling survey weights within each stratum by the inverse of the observed response rate. The adjustment uses `postStratify` (default) or `calibrate` to update the internal `svydesign` object, so all downstream estimators (`estimate_effort`, `estimate_catch_rate`, etc.) automatically use the corrected weights.

Usage

```

adjust_nonresponse(
  design,
  response_rates,
  method = c("postStratify", "calibrate"),
  stratum_col = "stratum"
)

```

Arguments

<code>design</code>	A <code>creel_design</code> object with at least one survey sub-object already attached (<code>add_counts</code> or <code>add_interviews</code>).
<code>response_rates</code>	A data frame or tibble with one row per stratum, containing at minimum the columns: stratum Character or factor. Stratum identifier matching the values in the design's strata column. n_sampled Integer. Number of units approached for interview in the stratum. n_responded Integer. Number of units that actually responded. Must be $\leq$ <code>n_sampled</code>.
<code>method</code>	Character. Weighting method to apply. Currently only "postStratify" (default) is implemented. It multiplies each observation's sampling weight by the inverse response rate for its stratum. Specifying "calibrate" raises an error; use <code>survey::calibrate</code> directly with population totals from your sampling frame.
<code>stratum_col</code>	Character. Name of the stratum column in <code>response_rates</code> (default: "stratum").

Value

The input `creel_design` with adjusted weights. The updated design includes an attribute "nonresponse_diagnostics" (a tibble with columns `stratum`, `n_sampled`, `n_responded`, `response_rate`, `weight_adjustment`) that can be retrieved with `attr(result, "nonresponse_diagnostics")`.

Adjustment method

For each stratum h :

$$response_rate_h = n_responded_h / n_sampled_h$$

$$weight_adjustment_h = 1 / response_rate_h$$

The original weights are multiplied by `weight_adjustment_h`, which upweights respondents to represent non-respondents (Armstrong & Overton 1977). This assumes that respondents and non-respondents are exchangeable within strata (a missing-at-random assumption). When this is implausible, a sensitivity analysis comparing pre- and post-adjustment estimates is recommended.

References

Armstrong, B.G. and Overton, W.S. 1977. Estimating nonresponse bias in mail surveys. *Journal of Marketing Research* 14:396–402.

Pollock, K.H., Jones, C.M. and Brown, T.L. 1994. *Angler Survey Methods and Their Applications in Fisheries Management*. American Fisheries Society, Bethesda, MD.

See Also

Other "Reporting & Diagnostics": `check_completeness()`, `compare_variance()`, `flag_outliers()`, `season_summary()`, `standardize_species()`, `summarize_boat_composition()`, `summarize_by_angler_type()`, `summarize_by_county()`, `summarize_by_day_type()`, `summarize_by_method()`, `summarize_by_species_sought()`, `summarize_by_trip_length()`, `summarize_by_zip()`, `summarize_cws_rates()`, `summarize_hws_rates()`, `summarize_length_freq()`, `summarize_refusals()`, `summarize_successful_parties()`, `summarize_trips()`, `summary.creel_estimates()`, `tidy.creel_estimates()`, `validate_creel_data()`, `validate_design()`, `validate_incomplete_trips()`, `validation_report()`, `write_estimates()`

Examples

```
data("example_counts", package = "tidycreel")
data("example_interviews", package = "tidycreel")
cal <- unique(example_counts[, c("date", "day_type")])
design <- creel_design(cal, date = date, strata = day_type)
design <- suppressWarnings(add_counts(design, example_counts))
design <- suppressWarnings(add_interviews(
  design, example_interviews,
  catch = catch_total, effort = hours_fished,
  trip_status = trip_status, trip_duration = trip_duration
))

resp <- data.frame(
  stratum = c("weekday", "weekend"),
  n_sampled = c(80L, 60L),
  n_responded = c(72L, 48L)
)
adj_design <- adjust_nonresponse(design, resp)
attr(adj_design, "nonresponse_diagnostics")
```

```
as.data.frame.creel_data_validation
```

Coerce a creel_data_validation to a plain data frame

Description

Strips the creel_data_validation class, returning the underlying data frame of check results.

Usage

```
## S3 method for class 'creel_data_validation'  
as.data.frame(x, ...)
```

Arguments

x	A creel_data_validation object.
...	Ignored.

Value

A plain data.frame.

```
as.data.frame.creel_design_comparison
```

Coerce a creel_design_comparison to a plain data frame

Description

Coerce a creel_design_comparison to a plain data frame

Usage

```
## S3 method for class 'creel_design_comparison'  
as.data.frame(x, ...)
```

Arguments

x	A creel_design_comparison object.
...	Ignored.

Value

A plain data.frame.

```
as.data.frame.creel_validation_report
```

Coerce a `creel_validation_report` to a plain data frame

Description

Strips the `creel_validation_report` class.

Usage

```
## S3 method for class 'creel_validation_report'  
as.data.frame(x, ...)
```

Arguments

x	A <code>creel_validation_report</code> object.
...	Ignored.

Value

A plain data.frame.

```
as.data.frame.creel_variance_comparison
```

Coerce `creel_variance_comparison` to data.frame

Description

Coerce `creel_variance_comparison` to data.frame

Usage

```
## S3 method for class 'creel_variance_comparison'  
as.data.frame(x, ...)
```

Arguments

x	A <code>creel_variance_comparison</code> object.
...	Additional arguments passed to <code>as.data.frame.tbl_df</code> .

Value

A plain data.frame.

<code>as_creel_svydesign</code>	<i>Extract internal survey design object for advanced use</i>
---------------------------------	---

Description

Provides power users with direct access to the internal `survey.design2` object for advanced analysis using survey package functions. This is an escape hatch for workflows not yet wrapped by `tidycreel`. Most users should use `estimate_effort()` instead.

The function issues a once-per-session warning to educate users that this is an advanced feature with risks if the returned object is modified incorrectly.

Usage

```
as_creel_svydesign(design)
```

Arguments

`design` A `creel_design` object with counts attached via [add_counts](#)

Value

A `survey.design2` object (from `survey::svydesign`). Due to R's copy-on-modify semantics, modifications to the returned object will not affect the internal `design$survey` object.

Warning

This function issues a once-per-session warning explaining:

- This is an advanced feature for power users
- Most users should use `estimate_effort()` instead
- Modifying the survey design may produce incorrect variance estimates

See Also

Other "Survey Design": [add_catch\(\)](#), [add_counts\(\)](#), [add_interviews\(\)](#), [add_lengths\(\)](#), [add_sections\(\)](#), [as_hybrid_svydesign\(\)](#), [compute_angler_effort\(\)](#), [compute_effort\(\)](#), [creel_design\(\)](#), [creel_schema\(\)](#), [creel_vocabulary\(\)](#), [derive_angler_count\(\)](#), [est_effort_camera\(\)](#), [impute_camera_counts\(\)](#), [mean_party_size\(\)](#), [prep_counts_boat_party\(\)](#), [prep_counts_daily_effort\(\)](#), [prep_interview_catch\(\)](#), [prep_interviews_trips\(\)](#), [validate_creel_schema\(\)](#)

Examples

```
# Basic workflow
library(survey)
cal <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend")
)
```

```

design <- creel_design(cal, date = date, strata = day_type)

counts <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend"),
  count = c(15, 23, 45, 52)
)

design2 <- add_counts(design, counts)

# Extract survey object for advanced use
svy <- as_creel_svydesign(design2)

# Use with survey package functions
survey::svytotal(~count, svy)
survey::svymean(~count, svy)

```

as_hybrid_svydesign	<i>Combine disjoint count frames into one stratified survey design</i>
---------------------	--

Description

[Experimental]

Usage

```

as_hybrid_svydesign(
  counts,
  frame_col,
  calendar = NULL,
  date_col = "date",
  strata_col = "day_type",
  count_col = "count",
  fraction = NULL,
  trips_disjoint = NULL,
  fpc = TRUE
)

```

Arguments

counts	Data frame of count observations for every frame, in long form: one row per frame per sampled date. Must contain the columns named by date_col, strata_col, count_col and frame_col, with at most one row per frame per date within a stratum.
frame_col	Character scalar. Name of the column in counts that partitions it into disjoint count frames – an angler-type column, for instance. Required, with no default: the column carries the partition the whole design rests on, and a default would let

	a missed argument pick one silently. Must have at least two distinct non-missing values; its values become the frame labels in the returned design.
calendar	Data frame giving the population of days the totals expand to, carrying the columns named by <code>date_col</code> and <code>strata_col</code>. Required: the NULL default is rejected, and exists only so the error can say what is missing. The stratum population size N_h is the number of distinct dates the stratum holds, counted the way <code>creel_design()</code> counts it. One row per day is the natural form, but a duplicated row is tolerated rather than refused, precisely because the count is over distinct dates and a repeat changes nothing. Two things are required. Every sampled date must appear in calendar under the same stratum, or the population is smaller than the sample. And each date must belong to exactly one stratum – a day listed under two lengthens the season by a day in each, and the period total then expands to a calendar larger than the one that exists.
date_col	Character scalar. Name of the date column (shared by counts and calendar). Default "date". Used to cluster observations into PSUs. Must be of class Date, with no missing values in either table.
strata_col	Character scalar. Name of the stratum column (shared by counts and calendar). Default "day_type". Must have no missing values in either table: dates and strata are the join keys, and a missing key matches every other missing key rather than being refused.
count_col	Character scalar. Name of the count column in counts. Default "count".
fraction	Named list of named numeric vectors, one element per frame, named by the frame labels in <code>frame_col</code> . Each element gives the within-day sampling fraction per stratum for that frame: the proportion of the frame the count enumerated on each sampled day, in (0, 1]. Expands a sampled day to a whole day; it is not a fraction of the season and does not drive the finite-population correction. Names of each element must match the stratum values that frame carries.
trips_disjoint	Logical scalar. Required: the NULL default is rejected, and exists only so the error can say what is missing. Set to TRUE to affirm that the frames sample disjoint sets of angler trips, the precondition under which their totals may be added. tidycreel cannot verify this from the data; see the "Disjointness precondition" section above.
fpc	Logical scalar. Apply the day-level finite-population correction n_h/N_h ? Default TRUE. Set to FALSE for the conservative with-replacement variance. NA and vectors of length other than one are refused.

Details

Combines two or more count series covering disjoint parts of one fishery into a single survey: `svydesign` object. The frames are treated as **strata**, each carrying its own within-day sampling fraction, and all expanded to the same population of days, so the design total is the stratified sum of the frame totals over the season.

Estimand. The design estimates a **period total** – the total over every day in calendar, not over the days that happened to be sampled. Two expansions get it there, and both live in the row weight: the within-day fraction expands the part of the frame that the count enumerated to the whole of it,

and N_h / n_h expands the sampled days to the days the stratum holds. Only the second is a stage-1 sampling fraction, so only the second drives the finite-population correction.

Disjointness precondition. Adding the frame totals is valid if and only if the frames sample **disjoint sets of angler trips** – no angler trip may be observed by more than one. What produces that disjointness is a property of the survey protocol (angler type, geography, access mode, or a rule the designer imposes); `tidycreel` cannot infer it from the counts, the dates, the strata, or the frame labels, so you must affirm it with `trips_disjoint = TRUE`. The design cannot be constructed otherwise. A boat angler intercepted on the water by a roving route and again at the ramp on the same trip belongs to two frames, and the total double counts that trip.

What a frame is. A frame is a disjoint part of the fishery, enumerated by its own count. In the protocol this design was built for the frames are angler-type domains – boat anglers, and bank anglers dispersed along a shoreline with no well-defined access site (Malvestuto 1996). Pope et al. (Chapter 17) carry exactly this as an `anglerType` column alongside the stratum, and estimate effort by stratum and angler type; `frame_col` is that column.

The frame is **not** an interview mode. In the creel literature access and roving describe how anglers are *interviewed*: access interviews intercept completed trips as anglers leave, roving interviews intercept incomplete trips while anglers are still fishing, and the two require different catch-rate estimators (Pollock et al. 1994). A survey mixing the two is a **hybrid interview** design. Counts are not described that way at all – they are instantaneous, progressive, bus-route, camera or aerial, the values `creel_schema()` accepts for `survey_type`. `tidycreel` carries the interview axis on `add_interviews()`'s `interview_type` argument, which is where it belongs. Earlier versions of this function named its arguments `access_data` and `roving_data`, which borrowed the interview vocabulary for something that is not an interview mode (GH #248).

Estimation route. The returned object is a `survey.design2`, not a `creel_design()`, so `estimate_effort()` does not accept it. Estimate from it with `survey::svytotal()` and the other survey functions directly, as in the examples below.

Design structure. Rows are stratified on the interaction of `strata_col` and `frame_col`, so each count frame carries its own sampled-day count at its own within-day fraction, and clustered on `date_col`, so the date is the primary sampling unit. The population size is taken from calendar and is shared by every frame: one stratum is one span of the season, whichever frame observed it. A frame that sampled only one date within a stratum leaves that stratum with a single PSU: the design still constructs, but `survey` refuses to compute a variance for it.

One count row per frame-day. A day-level expansion is only defined when a sampled day is one row per frame, so repeated counts on one date within a frame are refused. Two counts on a date are two looks at that date, not two sampled days; summed, they multiply the total by the number of counts, and the day expansion then multiplies that again. Average them to one row per date before constructing the design, or model them on a path that keeps the count time.

PSU alignment requirement: every frame should sample the same date-stratum combinations. A warning is issued when coverage is asymmetric, because the frames should sample the same days. That is a requirement about *when* each frame samples, not *where* – frames covering different water is the condition that makes their sum valid, not a source of bias.

Value

A `survey::svydesign` object with an additional class attribute `"creel_hybrid_svydesign"`. The design data carries the `frame_col` column unchanged, a weight column holding both the within-day and the day-to-season expansion, a `.hybrid_stratum` column holding the stratum-by-frame

interaction the design is stratified on, and a `.pop_days` column holding the stratum population N_h the finite-population correction is taken against. `attr(design, "component_col")` names the frame column.

See Also

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
calendar <- data.frame(
  date = seq(as.Date("2024-06-01"), as.Date("2024-06-30"), by = "day")
)
calendar$day_type <- ifelse(
  format(calendar$date, "%u") %in% c("6", "7"), "weekend", "weekday"
)

counts <- data.frame(
  date = rep(
    as.Date(c("2024-06-03", "2024-06-04", "2024-06-08", "2024-06-09")),
    times = 2
  ),
  day_type = rep(c("weekday", "weekday", "weekend", "weekend"), times = 2),
  angler_type = rep(c("boat", "bank"), each = 4),
  count = c(12L, 15L, 30L, 28L, 8L, 10L, 22L, 25L)
)
design <- as_hybrid_svydesign(
  counts,
  frame_col = "angler_type",
  calendar = calendar,
  fraction = list(
    boat = c(weekday = 0.5, weekend = 0.5),
    bank = c(weekday = 0.4, weekend = 0.4)
  ),
  trips_disjoint = TRUE
)

# estimate_effort() does not accept this object; use survey directly
survey::svytotal(~count, design)
```

Description

Cross-joins a daily schedule produced by [generate_schedule\(\)](#) with a count-time template produced by [generate_count_times\(\)](#), returning a creel_schedule with one row per (date x period x count_window).

Usage

```
attach_count_times(schedule, count_times)
```

Arguments

schedule	A creel_schedule from generate_schedule() . Must have a date column.
count_times	A creel_schedule from generate_count_times() . Must have start_time, end_time, and window_id columns.

Value

A creel_schedule data frame with all columns from schedule plus start_time, end_time, and window_id from count_times. Row count equals nrow(schedule) * nrow(count_times).

See Also

Other "Scheduling": [generate_bus_schedule\(\)](#), [generate_count_times\(\)](#), [generate_progressive_start\(\)](#), [generate_schedule\(\)](#), [new_creel_schedule\(\)](#), [read_schedule\(\)](#), [validate_creel_schedule\(\)](#), [write_schedule\(\)](#)

Examples

```
sched <- generate_schedule(
  start_date = "2024-06-01", end_date = "2024-06-07",
  n_periods = 2, sampling_rate = 0.5, seed = 1
)
ct <- generate_count_times(
  start_time = "06:00", end_time = "14:00",
  strategy = "systematic", n_windows = 3,
  window_size = 30, min_gap = 10, seed = 1
)
attach_count_times(sched, ct)
```

Description

`audit_strata()` is an S3 generic. Two methods are provided:

- `audit_strata.creel_design()` extracts stratum summaries from a completed design object and computes per-stratum RSE, DEFF, and meets-target flag.
- `audit_strata.default()` accepts pilot summary statistics (`N_h`, `n_h`, `ybar_h`, `s2_h`) directly.

Usage

```
audit_strata(x, ...)

## S3 method for class 'creel_design'
audit_strata(x, rse_target = 0.2, ...)

## Default S3 method:
audit_strata(x, n_h, ybar_h, s2_h, rse_target = 0.2, ...)
```

Arguments

<code>x</code>	A <code>creel_design</code> object (for the <code>creel_design</code> method) or a named numeric vector <code>N_h</code> of total available days per stratum (for the <code>default</code> method).
<code>...</code>	Additional arguments passed to methods.
<code>rse_target</code>	Numeric scalar. Target relative standard error threshold. Default 0.20 (20 percent). Must be in (0, 1].
<code>n_h</code>	Named numeric vector of the same length as <code>x</code> . Observed sample counts per stratum. Values must be ≥ 1 .
<code>ybar_h</code>	Numeric vector of the same length as <code>x</code> . Observed mean effort per day per stratum. Values must be ≥ 0 .
<code>s2_h</code>	Numeric vector of the same length as <code>x</code> . Observed variance of effort per day per stratum. Values must be ≥ 0 .

Details

The per-stratum RSE (relative standard error, equivalent to CV) is computed with the finite-population correction (FPC):

$$RSE_h = \sqrt{((1 - n_h / N_h) * s2_h / n_h) / ybar_h}$$

When $n_h = 1$ for any stratum, `var()` cannot be estimated; RSE, DEFF, and `meets_target` are set to NA for those strata and a warning is issued. The function continues processing valid strata.

The per-stratum design effect (DEFF_h) compares the actual stratum variance to the pooled-SRS variance baseline:

$$DEFF_h = ((1 - n_h / N_h) * s2_h / n_h) / ((1 - n / N) * s2_overall / n)$$

where $n = \text{sum}(n_h)$, $N = \text{sum}(N_h)$, and $s2_overall = \text{sum}(N_h * s2_h) / \text{sum}(N_h)$ (N_h -weighted pooled within-stratum variance). The aggregate DEFF stored in `$deff` is `Var_strat / Var_SRS` (Cochran 1977).

Value

A `creel_strata_audit` S3 object. See `audit_strata.default()` for the complete field description.

A `creel_strata_audit` S3 object — a named list with fields:

`$strata` Tibble with columns: `stratum`, `N_h`, `n_h`, `ybar_h`, `s2_h`, `RSE`, `DEFF`, `meets_target`.

`$rse_target` Scalar. The RSE threshold supplied by the caller.

`$n_total` Integer. Total sampled days across all strata.

`$deff` Scalar. Aggregate design effect ($\text{Var}_{\text{strat}} / \text{Var}_{\text{SRS}}$).

References

Cochran, W.G. 1977. Sampling Techniques, 3rd ed. Wiley, New York.

McCormick, J.L. and Quist, M.C. 2017. Sample size estimation for on-site creel surveys. North American Journal of Fisheries Management 37:970-983. doi:10.1080/02755947.2017.1342723

See Also

Other "Planning & Sample Size": [compare_designs\(\)](#), [creel_n_camera\(\)](#), [creel_n_cpue\(\)](#), [creel_n_effort\(\)](#), [creel_power\(\)](#), [cv_from_n\(\)](#), [optimal_n\(\)](#), [power_creel\(\)](#), [reallocate_strata\(\)](#), [simulate_strata_collapse\(\)](#)

Other "Planning & Sample Size": [compare_designs\(\)](#), [creel_n_camera\(\)](#), [creel_n_cpue\(\)](#), [creel_n_effort\(\)](#), [creel_power\(\)](#), [cv_from_n\(\)](#), [optimal_n\(\)](#), [power_creel\(\)](#), [reallocate_strata\(\)](#), [simulate_strata_collapse\(\)](#)

Examples

```
# Two-stratum weekday/weekend pilot example
audit <- audit_strata(
  c(weekday = 65, weekend = 28),
  n_h      = c(weekday = 22, weekend = 14),
  ybar_h   = c(50, 60),
  s2_h     = c(400, 500),
  rse_target = 0.20
)
audit$strata
audit$deff
```

autoplot.creel_design_comparison

Autoplot a creel_design_comparison as a forest plot

Description

Renders a forest plot showing point estimates with confidence intervals for each design, coloured by design name.

Usage

```
## S3 method for class 'creel_design_comparison'
autoplot(object, title = NULL, ...)
```

Arguments

object	A creel_design_comparison object.
title	Optional plot title.
...	Ignored.

Value

A ggplot object.

autoplot.creel_estimates

Plot creel survey estimates with ggplot2

Description

autoplot.creel_estimates() produces a point-and-errorbar plot from a creel_estimates object. For ungrouped estimates a single point with confidence interval is shown. For grouped estimates (when by was supplied to the estimation function) each group level gets its own point, colour-coded and positioned along the x-axis.

Returns a **ggplot2** object, which the package imports, so nothing extra needs installing.

Usage

```
## S3 method for class 'creel_estimates'
autoplot(object, title = NULL, theme = c("default", "creel"), ...)
```

Arguments

object	A creel_estimates object.
title	Optional character string for the plot title. Defaults to a human-readable description of the estimation method.
theme	Character string selecting the plot theme. Use "default" (default) for <code>ggplot2::theme_bw()</code> , or "creel" for <code>theme_creel()</code> and package-standard colours. Neither inherits a theme set with <code>ggplot2::theme_set()</code> ; add your own with + if you need it.
...	Additional arguments (currently unused).

Value

A ggplot object.

See Also

`estimate_effort()`, `estimate_catch_rate()`, `summary.creel_estimates()`
 Other "Visualisation": `autoplot.creel_length_distribution()`, `autoplot.creel_schedule()`,
`creel_palette()`, `plot_design()`, `theme_creel()`

Examples

```
data(example_calendar)
data(example_counts)
data(example_interviews)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)

est <- estimate_effort(design)
ggplot2::autoplot(est)

est_grp <- estimate_effort(design, by = day_type)
ggplot2::autoplot(est_grp)
```

```
autoplot.creel_length_distribution
```

Plot a weighted length distribution with ggplot2

Description

`autoplot.creel_length_distribution()` renders weighted length-frequency estimates as a histogram-style bar chart. Ungrouped results are shown as a single distribution; grouped results are faceted by the grouping variables.

Usage

```
## S3 method for class 'creel_length_distribution'
autoplot(object, title = NULL, theme = c("default", "creel"), ...)
```

Arguments

<code>object</code>	A <code>creel_length_distribution</code> object returned by <code>est_length_distribution()</code> .
<code>title</code>	Optional plot title. Defaults to a title derived from the estimated fish type (catch, harvest, or release).
<code>theme</code>	Character string selecting the plot theme. Use "default" (default) for <code>ggplot2::theme_bw()</code> , or "creel" for <code>theme_creel()</code> and package-standard colours. Neither inherits a theme set with <code>ggplot2::theme_set()</code> ; add your own with + if you need it.
<code>...</code>	Additional arguments (currently unused).

Value

A ggplot object.

See Also

[est_length_distribution\(\)](#)

Other "Visualisation": [autoplot.creel_estimates\(\)](#), [autoplot.creel_schedule\(\)](#), [creel_palette\(\)](#), [plot_design\(\)](#), [theme_creel\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_catch)
data(example_lengths)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
# Species catch is required to group by species: length totals are scaled
# onto the reported catch, and only this table records it per species.
design <- add_catch(design, example_catch,
  catch_uid = interview_id, interview_uid = interview_id,
  species = species, count = count, catch_type = catch_type
)
design <- add_lengths(design, example_lengths,
  length_uid = interview_id, interview_uid = interview_id,
  species = species, length = length, length_type = length_type,
  count = count, release_format = "binned"
)

ld <- est_length_distribution(design, by = species, bin_width = 25)
ggplot2::autoplot(ld)
```

`autoplot.creel_schedule`

Plot a creel schedule as a ggplot2 tile calendar

Description

`autoplot.creel_schedule()` produces a monthly tile calendar showing sampled dates coloured by day type (weekday / weekend) and unsampled dates in grey. Multiple months are shown as faceted panels.

Returns a **ggplot2** object, which the package imports, so nothing extra needs installing.

Usage

```
## S3 method for class 'creel_schedule'
autoplot(object, title = "Creel Schedule", ...)
```

Arguments

object	A creel_schedule object from generate_schedule() or generate_bus_schedule() .
title	Optional character string for the plot title. Defaults to "Creel Schedule".
...	Additional arguments (currently unused).

Value

A ggplot object.

See Also

[generate_schedule\(\)](#), [print.creel_schedule\(\)](#), [write_schedule\(\)](#)

Other "Visualisation": [autoplot.creel_estimates\(\)](#), [autoplot.creel_length_distribution\(\)](#), [creel_palette\(\)](#), [plot_design\(\)](#), [theme_creel\(\)](#)

Examples

```
sched <- generate_schedule(
  start_date = "2024-06-01", end_date = "2024-07-31",
  n_periods = 1,
  sampling_rate = c(weekday = 0.3, weekend = 0.6),
  seed = 42
)
ggplot2::autoplot(sched)
```

check_completeness	<i>Check post-season data completeness for a creel design</i>
--------------------	---

Description

Dispatches by survey_type to avoid false-positive warnings on aerial and camera designs that do not collect interview data.

Usage

```
check_completeness(design, n_min = 10L)
```

Arguments

design	A creel_design object with counts (and optionally interviews) attached.
n_min	Integer scalar ≥ 1 . Interview threshold below which a stratum is flagged as low-n. Default 10L.

Value

A creel_completeness_report object (S3 list) with:

\$missing_days data.frame of calendar rows with no count data

\$low_n_strata data.frame of strata below n_min, or NULL for aerial/camera

\$refusals creel_summary_refusals object or NULL

\$n_min integer threshold used

\$survey_type character

\$passed logical – TRUE if no missing days and no low-n strata

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_counts)
data(example_interviews)
design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
check_completeness(design)
```

compare_cpue_estimators

Compare CPUE estimators (ROM, MOR, Regression)

Description

Runs all three CPUE estimators — Ratio-of-Means (ROM / CPUE₂), Mean-of-Ratios (MOR / CPUE₁), and OLS regression slope with jackknife SE (CPUE₃) — on the same creel design and returns a combined tibble with a cpue_method column for side-by-side comparison.

This implements the Petrere et al. (2010) Table 1 estimator comparison workflow. When the three estimators yield materially different estimates, the choice of estimator matters; compare_cpue_estimators() makes divergence visible.

Usage

```
compare_cpue_estimators(
  design,
  by = NULL,
  conf_level = 0.95,
  force_origin = TRUE,
  verbose = FALSE
)
```

Arguments

design	A creel_design object with interviews attached. Must have catch_col and angler_effort_col set.
by	Optional tidy selector for grouping variables. Passed to each underlying estimate_catch_rate call.
conf_level	Numeric. Confidence level. Default 0.95.
force_origin	Logical. Force regression through origin. Default TRUE (standard CPUE ₃ formulation).
verbose	Logical. If TRUE, prints a brief message for each estimator run. Default FALSE.

Details

Estimator definitions following Petrere et al. (2010):

ROM (CPUE₂) Ratio of means: total catch / total effort (survey::svyratio). Unbiased for complete trips.

MOR (CPUE₁) Mean of individual ratios: mean(catch_{*i*} / effort_{*i*}) (survey::svymean). Preferred for incomplete trips.

Regression (CPUE₃) OLS slope $\hat{\beta}$ from $C_i = \beta f_i + \varepsilon_i$ with leave-one-out jackknife SE. Most robust when proportionality is violated (non-zero intercept).

Value

A tibble with columns cpue_method (character: "rom", "mor", "regression"), plus estimate, se, ci_lower, ci_upper, n, and any grouping columns when by is specified. The tibble has class c("cpue_comparison", "tbl_df", "tbl", "data.frame").

References

Petrere, M. Jr., Giacomini, H.C. & De Marco, P. Jr. (2010). Catch-per-unit-effort: which estimator is best? *Braz. J. Biol.* 70: 483–491. doi:10.1590/S151969842010005000010

See Also

[estimate_catch_rate](#)

Other "Estimation": [est_age_distribution\(\)](#), [est_biomass\(\)](#), [est_compliance\(\)](#), [est_effort_camera_mi\(\)](#), [est_length_distribution\(\)](#), [est_mean_age\(\)](#), [est_mean_length\(\)](#), [estimate_catch_rate\(\)](#), [estimate_effort\(\)](#), [estimate_effort_aerial_glmm\(\)](#), [estimate_harvest_rate\(\)](#), [estimate_release_rate\(\)](#), [estimate_total_catch\(\)](#), [estimate_total_harvest\(\)](#), [estimate_total_release\(\)](#)

Examples

```

design <- creel_design(example_calendar, date = date, strata = day_type) |>
  add_interviews(example_interviews,
    catch = catch_total, effort = hours_fished,
    trip_status = trip_status, n_anglers = n_anglers
  )

compare_cpue_estimators(design)

```

compare_designs	<i>Compare multiple survey design estimates side by side</i>
-----------------	--

Description**[Experimental]****Usage**

```
compare_designs(designs, metric = "estimate")
```

Arguments

designs	Named list of creel_estimates objects. Names become the design column in the output. At least two elements are required.
metric	Character scalar. Which estimate column to compare. Default "estimate". Must be a column present in every estimates data frame.

Details

Takes a named list of creel_estimates objects (from different survey designs or methods), extracts key precision metrics from each, and returns a tidy comparison tibble. An autoplot() method renders a forest plot of point estimates with confidence intervals.

Value

A creel_design_comparison object – a data frame with columns:

design Design name (from names(designs)).
 estimate Point estimate.
 se Standard error.
 rse Relative standard error ($se / |estimate|$).
 ci_lower Lower confidence interval bound.
 ci_upper Upper confidence interval bound.
 ci_width Width of the confidence interval.
 n Sample size (if present in the estimates frame).

Group columns are retained when all designs share the same by-variable structure.

See Also

`autoplot.creel_design_comparison()`

Other "Planning & Sample Size": `audit_strata()`, `creel_n_camera()`, `creel_n_cpue()`, `creel_n_effort()`, `creel_power()`, `cv_from_n()`, `optimal_n()`, `power_creel()`, `reallocate_strata()`, `simulate_strata_collapse()`

Examples

```
data(example_calendar)
data(example_counts)
data(example_interviews)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)

# Two estimates from the same design: overall, and split by stratum. In
# practice these would come from designs built on different survey types.
est_all <- estimate_effort(design)
est_grp <- estimate_effort(design, by = day_type)

compare_designs(list(overall = est_all, by_day_type = est_grp))
```

compare_variance	<i>Compare Taylor linearization vs. replicate variance for creel estimates</i>
------------------	--

Description

Takes a `creel_estimates` object produced with `variance = "taylor"` and re-estimates using replicate weights (bootstrap or jackknife) to produce a side-by-side comparison of standard errors. A `cli_warn()` is issued for any row where the two SEs diverge by more than `divergence_threshold`.

Usage

```
compare_variance(
  x,
  replicate_method = c("bootstrap", "jackknife"),
  conf_level = 0.95,
  divergence_threshold = 0.1,
  ...
)
```

Arguments

- x** A `creel_estimates` object with `variance_method = "taylor"`. Must have been created with a design stored in `x$design`.
- replicate_method** Character. Replicate variance method to use for comparison. One of "bootstrap" (default) or "jackknife".
- conf_level** Numeric confidence level (default: 0.95). Passed to the replicate estimation call.
- divergence_threshold** Numeric. Fraction by which replicate SE may differ from Taylor SE before a warning is issued (default: 0.10 = 10\ A warning fires for any group where $|se_replicate / se_taylor - 1| > divergence_threshold$.
- ...** Additional arguments passed to the underlying estimator.

Value

A `creel_variance_comparison` S3 object (a tibble subclass) with columns:

se_taylor Taylor linearization SE from the original estimate.

se_replicate Replicate-weight SE from the re-estimation.

divergence_ratio Ratio `se_replicate / se_taylor`. NA when `se_taylor == 0`.

diverges_flag Logical. TRUE when $|divergence_ratio - 1| > divergence_threshold$.

Group columns (if any) are preserved. The full tibble is returned invisibly via `print()`. Use `as.data.frame()` or standard tibble methods for further processing.

Method

The function extracts the Taylor SE from `x$estimates$se`, then calls the same estimator that produced `x` (resolved via `x$method`) with `variance = replicate_method`. The re-estimation uses `x$design` and the grouping variables from `x$by_vars`.

Divergence is computed as:

$$ratio = se_{replicate} / se_{taylor}$$

$$diverges = |ratio - 1| > threshold$$

A ratio substantially different from 1 indicates that the Taylor approximation may be unreliable for this design (e.g., sparse strata, non-linear estimator). Replication-based variance is generally more robust but slower to compute.

References

- Wolter, K.M. 2007. Introduction to Variance Estimation, 2nd ed. Springer.
- Lumley, T. 2010. Complex Surveys: A Guide to Analysis Using R. Wiley.

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data("example_counts", package = "tidycreel")
data("example_interviews", package = "tidycreel")
cal <- unique(example_counts[, c("date", "day_type")])
design <- creel_design(cal, date = date, strata = day_type)
design <- suppressWarnings(add_counts(design, example_counts))
design <- suppressWarnings(add_interviews(
  design, example_interviews,
  catch = catch_total, effort = hours_fished,
  trip_status = trip_status, trip_duration = trip_duration
))
taylor_est <- suppressWarnings(estimate_catch_rate(design))
cmp <- suppressWarnings(compare_variance(taylor_est))
print(cmp)
```

`compute_angler_effort` *Normalize fishing effort to angler-hours*

Description

Multiplies per-trip effort (hours) by party size (number of anglers) to produce angler-hours. This converts party-level effort records to individual-angler units, which are required for CPUE and harvest-rate computations.

This function can be called standalone on a raw data frame or is called internally by [add_interviews](#) when constructing the design object.

Usage

```
compute_angler_effort(data, effort, n_anglers)
```

Arguments

<code>data</code>	A data frame containing the interview records.
<code>effort</code>	Tidy selector for the effort column (numeric, hours per trip).
<code>n_anglers</code>	Number of anglers in the party. Either a bare column name (e.g. <code>n_anglers = party_size</code>) or a single positive number stating a constant party size (e.g. <code>n_anglers = 1</code> for individual-level interviews). A bare number is read as a party size, not as a tidyselect column position. Values must be positive and finite.

Value

The input data frame with an added `.angler_effort` column (numeric, angler-hours). Existing columns are preserved.

See Also

`compute_effort()`, `add_interviews()`

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
parties <- data.frame(effort = c(2.0, 3.0), n_angers = c(2L, 3L))
compute_angler_effort(parties, effort, n_angers)
```

compute_effort	<i>Resolve fishing effort from timestamps or self-reported time</i>
----------------	---

Description

Computes fishing effort (hours) for each interview row using a conditional rule: if the `time_fished` column is present and non-NA for a row, use that value (angler self-reported hours, e.g. after a break); otherwise compute from timestamps as `difftime(interview_time, trip_start, units = "hours")`.

This function can be called standalone on raw data before entering the `add_interviews` workflow, or used to preprocess a column that will be passed as the `effort` argument to `add_interviews()`.

Usage

```
compute_effort(data, trip_start, interview_time, time_fished = NULL)
```

Arguments

- `data` A data frame containing the interview records.
- `trip_start` Tidy selector for the trip start timestamp column (POSIXct).
- `interview_time` Tidy selector for the interview timestamp column (POSIXct).
- `time_fished` Optional tidy selector for a self-reported hours column. When a row has a non-NA value here, it overrides the timestamp calculation. Default is NULL (always compute from timestamps).

Value

The input data frame with an added `.effort` column (numeric, hours). Existing columns are preserved.

See Also

`compute_angler_effort()`, `add_interviews()`

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
trips <- data.frame(
  trip_start      = as.POSIXct(c("2024-06-01 08:00:00", "2024-06-01 09:15:00")),
  interview_time = as.POSIXct(c("2024-06-01 10:30:00", "2024-06-01 12:00:00"))
)
compute_effort(trips, trip_start, interview_time)
```

creel_confidence_intervals

Confidence interval conventions in tidycreel

Description

Estimators in this package do not all build confidence intervals the same way, because the quantities they estimate do not all live on the same scale or carry the same information about their own uncertainty. This topic states the two rules the package follows, so that a new estimator does not have to pick by coin flip.

Bounded quantities

Effort, catch, harvest, release, biomass, abundance and catch rates are all bounded below by zero, and exploitation rate is bounded on both sides. A symmetric Wald interval respects none of that: once the coefficient of variation exceeds roughly 0.51, the lower bound of a 95% interval falls below zero and the estimator reports a value outside the parameter space.

The package handles this in one of two ways, in this order of preference:

1. **Transform**, where a principled transform for the quantity exists. `estimate_exploitation_rate()` builds its interval on the logit scale; `estimate_angler_n()` uses Sardinle's (2009) 0.5 transformed logit interval for Chapman and Petersen $\hat{N}$; the product-total paths accept `ci_type = "log"`. A transformed interval is right-skewed and cannot cross the boundary in the first place.

2. **Clamp** at the feasible limit, where no transform is established for the estimator. This is `pmax(0, ...)` applied to the lower bound, used by the product totals under `ci_type = "symmetric"` and by every bus-route path.

Clamping is the weaker of the two and is deliberately not treated as equivalent. It keeps the symmetric width that produced the excursion and truncates the result at the boundary, so it stops the package reporting an impossible number but does not repair the skew that made the bound negative. A clamped lower bound of exactly zero should be read as a signal that the interval is wide relative to the estimate, not as a precise statement that the quantity could be zero.

Upper bounds are never clamped: none of these quantities has a finite upper limit that the package knows.

Two quantities are bounded below by something other than zero and are therefore left alone: `est_mean_length()` and `est_mean_age()` are bounded by the smallest occupied bin, so a clamp at zero would be the wrong repair.

Quantile choice

Where an estimator has a finite sample size of its own to appeal to, it uses `stats::qt()` on an explicit degrees-of-freedom rule. The product-total paths use total interviews minus the number of strata; the section paths use the number of sections minus one; mark-recapture uses the number of occasions.

Six estimators use `stats::qnorm()` instead — `est_biomass()`, `est_mean_length()`, `est_compliance()`, `est_mean_age()`, and since #310 `est_length_distribution()` and `est_age_distribution()` themselves. This is deliberate, not an oversight.

The first four form a linear combination of the rows of a length or age distribution, and their standard error is propagated from the per-bin standard errors those rows already carry. The two distributions joined them when their totals became two-phase: a bin total is no longer a quantity `svytotal()` returns directly but a delta-method function of the measured bins and the reported total, so its standard error is likewise propagated rather than design-based. (The interval width is unchanged by that switch — survey's `confint()` method for a `svyestat` defaults to `df = Inf`, which is the normal quantile.)

In every case there is no local sample size to key a t-quantile to:

- The number of rows is the number of **bins**, which is a binning choice made by the caller. Keying degrees of freedom to it would make the interval narrow as the bins got finer, with no additional fish measured.
- The row totals are **expanded estimates**, not counts of measured fish, so their sum is an estimated abundance rather than a sample size.

The honest degrees of freedom belong to the design that produced the upstream standard errors, and the large-sample argument is made there. These four estimators inherit that uncertainty rather than sampling afresh, so the normal quantile is the consistent choice at this level.

`estimate_effort_aerial_glm()` is asymptotic by construction and has no finite df to appeal to. The normal quantile inside Sadinle's logit interval is part of the method, not a quantile choice.

References

Sadinle, M. (2009). Transformed logit confidence intervals for small populations in single capture-recapture estimation. *Communications in Statistics - Simulation and Computation*, 38(9), 1909-1924.

See Also

[estimate_effort\(\)](#), [estimate_total_catch\(\)](#), [estimate_harvest_rate\(\)](#), [estimate_exploitation_rate\(\)](#), [estimate_angler_n\(\)](#), [est_biomass\(\)](#), [est_mean_length\(\)](#)

creel_counts_toy	<i>Toy count data for data validation examples</i>
------------------	--

Description

A small creel count data frame designed for demonstrating `validate_creel_data()` and related data-cleaning functions. Contains an intentional NA in the count column to trigger the NA-rate check.

Usage

```
creel_counts_toy
```

Format

A data frame with 6 rows and 4 columns:

date Survey date (Date class).

day_type Day type stratum: "weekday" or "weekend".

section Survey section: "A" or "B".

count Instantaneous angler count; one row is intentionally NA.

Source

Simulated data for package examples and vignettes.

See Also

[creel_interviews_toy](#)

Other "Example Datasets": [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_f](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(creel_counts_toy)
validate_creel_data(counts = creel_counts_toy)
```

creel_design	<i>Create a creel survey design</i>
--------------	-------------------------------------

Description

Constructs a `creel_design` object from calendar data with tidy column selection. This is the entry point for all creel survey analysis workflows. The design object stores the survey structure (date, strata, optional site), validates input data (Tier 1 validation), and serves as the foundation for adding count data and estimating effort.

For bus-route surveys with nonuniform site selection probabilities, use `survey_type = "bus_route"` and supply a `sampling_frame` data frame specifying sites, circuits, and their sampling probabilities.

Usage

```
creel_design(
  calendar,
  date,
  strata,
  site = NULL,
  design_type = "instantaneous",
  survey_type = design_type,
  sampling_frame = NULL,
  p_site = NULL,
  p_period = NULL,
  circuit = NULL,
  effort_type = NULL,
  camera_mode = NULL,
  h_open = NULL,
  visibility_correction = NULL,
  visibility_se = NULL,
  angler_ratio = NULL,
  angler_ratio_se = NULL,
  open_start = NULL
)
```

Arguments

calendar	A data frame containing calendar data with date and strata columns. Must have at least one Date column and one character/factor column (validated via internal schema check).
date	Tidy selector for the date column. Must select exactly one column of class Date. Accepts bare column names or tidyselect helpers (e.g., <code>starts_with("date")</code>).

strata	Tidy selector for strata columns. Can select one or more columns of class character or factor. Accepts bare column names or tidyselect helpers (e.g., <code>c(day_type, season)</code> or <code>starts_with("day")</code>).
site	Optional tidy selector for a site column. For instantaneous designs, selects from calendar. For bus-route designs (<code>survey_type = "bus_route"</code>), selects the site ID column from <code>sampling_frame</code> . Must select exactly one column of class character or factor. Default is NULL (single-site survey for instantaneous designs; required for bus-route designs).
design_type	Character string specifying the survey design type. Default is "instantaneous". Kept for backward compatibility; use <code>survey_type</code> for new code.
survey_type	Character string specifying the survey type. Default inherits from <code>design_type</code> ("instantaneous"). Use "bus_route" for nonuniform probability bus-route surveys (BUSRT-06, BUSRT-07). Both <code>survey_type</code> and <code>design_type</code> refer to the same concept; <code>survey_type</code> is the canonical parameter for new designs.
sampling_frame	Data frame with site, circuit, and probability columns. Required when <code>survey_type = "bus_route"</code> . Each row represents one site-circuit sampling unit with its inclusion probability components (<code>p_site</code> and <code>p_period</code>).
p_site	Tidy selector for the site sampling probability column in <code>sampling_frame</code> . Required when <code>survey_type = "bus_route"</code> . Values must be in $(0, 1]$ and must sum to 1.0 within each circuit (tolerance $1e-6$).
p_period	Tidy selector for the period sampling probability column in <code>sampling_frame</code> , OR a scalar numeric value in $(0, 1]$ that applies globally to all rows. Required when <code>survey_type = "bus_route"</code> .
circuit	Optional tidy selector for the circuit ID column in <code>sampling_frame</code> . A circuit is a route x period combination. If omitted, all rows are treated as belonging to a single unnamed circuit (<code>".default"</code>). Required only for multi-circuit designs.
effort_type	Character string specifying the type of effort measured in ice fishing surveys. Required when <code>survey_type = "ice"</code> . Must be one of "time_on_ice" (total hours the angler was on the ice) or "active_fishing_time" (hours actively fishing, excluding travel/setup). The value controls the column name in <code>estimate_effort()</code> output: <code>total_effort_hr_on_ice</code> or <code>total_effort_hr_active</code> .
camera_mode	Character string specifying the camera sub-mode. Required when <code>survey_type = "camera"</code> . Must be one of "counter" (camera records a daily ingress total) or "ingress_egress" (camera records individual arrival/departure timestamps, which should be preprocessed with <code>preprocess_camera_timestamps()</code> before calling <code>add_counts()</code>).
h_open	Positive numeric scalar specifying the number of hours the fishery is open per day. Required when <code>survey_type = "aerial"</code> . Used as the expansion factor in the aerial effort estimator: $\hat{E} = N_{obs} \times h_{open}/v$.
visibility_correction	Detection probability for the aerial count: the proportion of anglers present that are detected from the aircraft, as a numeric scalar in $(0, 1]$. Required when <code>survey_type = "aerial"</code> ; pass the string "none" to state explicitly that no visibility correction is applied. Used only when <code>survey_type = "aerial"</code> . A value of 0.85 means 85% of anglers are detected; the effort estimate is scaled up by $1/0.85$.

This is the reciprocal of the ratio published by field studies. The standard ground-truthing method reports $r = (\text{ground count}) / (\text{aerial count})$, which is **greater than 1** whenever the aircraft undercounts: Smucker et al. (2010) report $r = 2.69$ for shore anglers. `visibility_correction` is a *probability*, so convert with $v = 1/r$ — $r = 2.69$ becomes `visibility_correction` = $1 / 2.69 = 0.372$. The estimator divides by v , which scales effort up by $1/0.372 = 2.69$ as intended. Passing the published r directly is rejected by the $(0, 1]$ check.

`visibility_se` Optional positive numeric scalar: the standard error of `visibility_correction`, on the same detection-probability scale. Used only when `survey_type = "aerial"`. `visibility_correction` is estimated from paired air-ground counts, not known, and the standard field method reports its SE as routine output (Smucker et al. 2010, equations 6 and 7); supplying it here propagates that uncertainty into the effort SE (GH #135). All-or-none: `visibility_se` requires a numeric `visibility_correction`, and cannot be combined with "none".

To convert an SE published on the ground-truthing-ratio scale, use the delta method for a reciprocal: $SE(v) = SE(r)/r^2$.

When omitted, the correction is treated as supplied without a measured uncertainty and the component is reported as absent — never as zero, which would be indistinguishable from having propagated it.

Note the three distinct claims, which the package keeps separate:

`visibility_correction = "none"` No correction was studied. The point estimate divides by 1 and `visibility_se` is NA, so the reported SE is NA — the uncertainty is unpropagated, not zero.

`visibility_correction = v` **alone** A correction was measured but its SE was not supplied. The component is reported as absent.

`visibility_correction = v`, `visibility_se = 0` The correction is asserted to be known exactly. This is the only way to obtain a numeric SE with no visibility uncertainty, and it must be stated deliberately — e.g. `visibility_correction = 1`, `visibility_se = 0` for a fishery where every angler is genuinely detectable.

`angler_ratio` The proportion of the people recorded in the aerial count that are anglers, as a numeric scalar in $(0, 1]$. **Required** when `survey_type = "aerial"`; pass the string "none" to state explicitly that no such correction is applied.

An aerial count column is a **raw observer count**, and observers cannot reliably tell anglers from non-anglers from the air. Smucker et al. (2010) apply an angler-to-people ratio of 0.404 alongside their visibility correction; omitting it overstates shore effort by roughly 2.5x. The two corrections push in **opposite** directions (0.404 down, 2.69 up), so applying only the visibility correction is not conservative — it is biased in the direction of the correction that was kept (GH #158).

If the count column already records anglers rather than people, say so with `angler_ratio = 1`, `angler_ratio_se = 0`, which asserts the ratio is known exactly. That is a claim about how the data were recorded, and only the surveyor can make it.

For a count of **boats** rather than people, do not use this argument: expand the boat count to anglers with `derive_angler_count()` before `add_counts()`, which attaches the party-size multiplier and its standard error as expansion carrier columns that the estimator reads.

angler_ratio_se	Optional positive numeric scalar: the standard error of angler_ratio. All-or-none — it requires a numeric angler_ratio and cannot be combined with "none". Like the visibility correction, the angler-to-people ratio is estimated from ground observation and is a shared multiplier, so its contribution enters once at the total. Omitted means the component is reported as absent, never as zero.
open_start	Optional non-negative numeric scalar specifying the hour of day (decimal, 24-hour clock) when the fishery opens. Used only when survey_type = "aerial" and only by <code>estimate_effort_aerial_glmm()</code> to anchor the numerical integration window. If NULL (default), the GLMM estimator derives the window start from the earliest observed flight time minus 0.5 hours, with an informational message. Supplying open_start fixes the window across surveys for consistent comparisons. Example: open_start = 5.5 means fishing begins at 5:30 AM.

Value

A creel_design S3 object (list) with components:

calendar	The original calendar data frame
date_col	Character name of the date column
strata_cols	Character vector of strata column names
site_col	Character name of site column, or NULL
design_type	Character design type
counts	NULL (populated by <code>add_counts()</code> in future)
survey	NULL (populated internally during estimation)
bus_route	List with resolved sampling frame data and column mappings, or NULL for non-bus-route designs. Contains: <code>\$data</code> (sampling frame with <code>.pi_i</code> column added), <code>\$site_col</code> , <code>\$circuit_col</code> , <code>\$p_site_col</code> , <code>\$p_period_col</code> , <code>\$pi_i_col</code> (always <code>".pi_i"</code>).

Tier 1 Validation

The constructor performs fail-fast validation:

- Date column is class Date (not character, numeric, POSIXct)
- Date column contains no NA values
- Strata columns are character or factor (not numeric, logical)
- Site column (if provided) is character or factor
- (bus_route only) All p_site and p_period values are in (0, 1]
- (bus_route only) p_site values sum to 1.0 within each circuit (tolerance 1e-6)
- (bus_route only) p_period values are constant within each circuit (tolerance 1e-10)

References

Jones, C. M., & Pollock, K. H. (2012). Recreational survey methods: estimating effort, harvest, and abundance. In A. V. Zale, D. L. Parrish, & T. M. Sutton (Eds.), *Fisheries Techniques* (3rd ed., pp. 883–919). American Fisheries Society. Eq. 19.4 and 19.5 define the bus-route estimators; pp. 883–884 define the inclusion probability $\pi_i = p_{\text{site}} \times p_{\text{period}}$.

See Also

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
# Basic design with single stratum
calendar <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03")),
  day_type = c("weekday", "weekend", "weekend")
)
design <- creel_design(calendar, date = date, strata = day_type)

# Multiple strata
calendar <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02")),
  day_type = c("weekday", "weekend"),
  season = c("summer", "summer")
)
design <- creel_design(calendar, date = date, strata = c(day_type, season))

# With site column for multi-site survey
calendar <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02")),
  day_type = c("weekday", "weekend"),
  lake = c("lake_a", "lake_b")
)
design <- creel_design(calendar, date = date, strata = day_type, site = lake)

# Using tidyselect helpers
calendar <- data.frame(
  survey_date = as.Date(c("2024-06-01", "2024-06-02")),
  day_type = c("weekday", "weekend"),
  day_period = c("morning", "evening")
)
design <- creel_design(
  calendar,
  date = starts_with("survey"),
  strata = starts_with("day")
)
```

```
# Bus-route design with scalar p_period
calendar_br <- data.frame(
  date = as.Date("2024-06-01"),
  day_type = "weekday"
)
sf <- data.frame(
  site = c("A", "B", "C"),
  p_site = c(0.3, 0.4, 0.3),
  p_period = 0.5
)
design_br <- creel_design(
  calendar_br,
  date = date,
  strata = day_type,
  survey_type = "bus_route",
  sampling_frame = sf,
  site = site,
  p_site = p_site,
  p_period = p_period
)
```

creel_interviews_toy *Toy interview data for data validation examples*

Description

A small creel interview data frame with intentional data quality issues for demonstrating `validate_creel_data()` and `standardize_species()`. Includes an empty species string, a negative fish_kept value, and a missing trip_hours value.

Usage

```
creel_interviews_toy
```

Format

A data frame with 6 rows and 5 columns:

date Interview date (Date class).

day_type Day type stratum: "weekday" or "weekend".

species Free-text species name; includes empty string and unrecognised value to demonstrate `standardize_species()` behaviour.

fish_kept Number of fish kept; one row is intentionally negative.

trip_hours Trip duration in hours; one row is intentionally NA.

Source

Simulated data for package examples and vignettes.

See Also

[creel_counts_toy](#)

Other "Example Datasets": [creel_counts_toy](#), [example_aerial_counts](#), [example_aerial_glmm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_f](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(creel_interviews_toy)
validate_creel_data(interviews = creel_interviews_toy)
standardize_species(creel_interviews_toy, species_col = "species")
```

creel_n_camera	<i>Calculate camera-days required to achieve a target CV</i>
----------------	--

Description

Uses the stratified sample size formula from Cochran (1977) to determine how many camera-days are needed to achieve a target coefficient of variation on the camera-effort estimate, given pilot mean and variance estimates per day-type stratum.

Usage

```
creel_n_camera(cv_target, N_h, ybar_h, s2_h)
```

Arguments

cv_target	Numeric scalar. Target coefficient of variation for the camera-effort estimate (e.g., 0.20 for 20 percent). Must be in (0, 1].
N_h	Named numeric vector. Total available days per stratum (e.g., c(weekday = 65, weekend = 28)). Values must be >= 1.
ybar_h	Numeric vector of same length as N_h. Pilot mean camera count per day per stratum. Values must be >= 0.
s2_h	Numeric vector of same length as N_h. Pilot variance of camera counts per day per stratum. Values must be >= 0.

Details

Implements Cochran (1977) equation 5.25 under proportional allocation. The finite-population correction (FPC) factor is intentionally omitted (standard practice for pre-season planning where the goal is to determine how many days to deploy cameras, not to assess precision of a completed survey).

The per-stratum sample sizes n_h are computed from the total n_{total} under proportional allocation: $n_h = \text{ceiling}(n_{\text{total}} * N_h / \text{sum}(N_h))$. Because each stratum is ceiling-ed independently, $\text{sum}(n_h)$ may exceed n_{total} .

Feltz-Middaugh (2025) empirical benchmark. That study reports the camera-day schedules at which a low-frequency time-lapse deployment performed acceptably. Its two headline scenarios are, per **month**:

- *well-performing* (under 20 percent error in at least 80 percent of simulations): 12 weekdays and 7 weekend days, both at 1 count/day;
- *best-performing* (under 10 percent error in at least 80 percent of simulations): 18 weekdays at 2 counts/day and 8 weekend days at 4 counts/day.

These are reported here as design context, not applied as a check. The function cannot judge a computed n_h against them: N_h is the whole survey period rather than a month, nothing here knows the counts per day the schedule assumes, the error bands are fixed by the study rather than taken from `cv_target`, and the simulations measured boat-trailer counts on six Arkansas reservoirs, whereas `ybar_h` and `s2_h` are whatever the caller piloted. Compare against them by hand, after converting to the same units.

Earlier versions warned when n_h fell below 12 or 7, choosing which benchmark to apply by matching the substring "weekday" or "weekend" in the stratum name. That comparison was between a period-scale allocation and a per-month recommendation, so it under-fired by roughly the number of months in the survey (#234). No sample size ever changed: the check only ever emitted a warning.

Value

A named integer vector. Elements named after strata in N_h give the camera-days required per stratum; element "total" gives Cochran's overall sample size *before* proportional allocation, and "allocated" the sum of the per-stratum values actually returned. Budget against "allocated"; see `creel_n_effort()` for why the two differ.

References

Cochran, W.G. 1977. Sampling Techniques, 3rd ed. Wiley, New York.

Feltz, C.J. and Middaugh, C.R. 2025. Improving efficiency of estimating angler effort using low-frequency time-lapse camera data. North American Journal of Fisheries Management 45:322-332.

See Also

`creel_n_effort()` for the equivalent function for angler-contact sampling days.

Other "Planning & Sample Size": `audit_strata()`, `compare_designs()`, `creel_n_cpue()`, `creel_n_effort()`, `creel_power()`, `cv_from_n()`, `optimal_n()`, `power_creel()`, `reallocate_strata()`, `simulate_strata_collapse()`

Examples

```
# Two-stratum weekday/weekend example
creel_n_camera(
  cv_target = 0.20,
  N_h = c(weekday = 65, weekend = 28),
  ybar_h = c(15, 20),
```

```
s2_h = c(625, 900)
)
```

creel_n_cpue

Calculate interviews required to achieve a target CV on CPUE

Description

Determines the number of interviews needed to achieve a target coefficient of variation on a CPUE (catch-per-unit-effort) ratio estimate, using the ratio-estimator variance approximation from Cochran (1977).

Usage

```
creel_n_cpue(cv_catch, cv_effort, rho = 0, cv_target)
```

Arguments

cv_catch	Numeric scalar. Pilot coefficient of variation of catch per interview (the numerator of the CPUE ratio). Must be > 0.
cv_effort	Numeric scalar. Pilot coefficient of variation of effort per interview (the denominator of the CPUE ratio). Must be > 0.
rho	Numeric scalar. Pilot correlation between catch and effort per interview. Must be in [-1, 1]. Default is 0 (conservative; over-estimates required n when catch and effort are positively correlated).
cv_target	Numeric scalar. Target coefficient of variation for the CPUE estimate. Must be in (0, 1].

Details

Implements the ratio-estimator variance approximation from Cochran (1977, Chapter 6) parameterised in terms of coefficients of variation rather than raw variances, which is more natural for pre-season planning:

$$n = \left\lceil \frac{CV_{catch}^2 + CV_{effort}^2 - 2\rho \cdot CV_{catch} \cdot CV_{effort}}{CV_{target}^2} \right\rceil$$

Setting $\rho = 0$ (the default) is conservative: it over-estimates the required sample size when catch and effort are positively correlated. Users with pilot data should supply the observed correlation to obtain a less conservative estimate.

The result is floored at 1L to ensure at least one interview is recommended.

Value

An integer scalar (≥ 1): number of interviews required.

References

Cochran, W.G. 1977. Sampling Techniques, 3rd ed. Wiley, New York. Chapter 6 (ratio estimator variance approximation).

See Also

Other "Planning & Sample Size": [audit_strata\(\)](#), [compare_designs\(\)](#), [creel_n_camera\(\)](#), [creel_n_effort\(\)](#), [creel_power\(\)](#), [cv_from_n\(\)](#), [optimal_n\(\)](#), [power_creel\(\)](#), [reallocate_strata\(\)](#), [simulate_strata_collapse\(\)](#)

Examples

```
# rho = 0 (conservative, no pilot correlation data)
creel_n_cpue(cv_catch = 0.8, cv_effort = 0.5, rho = 0, cv_target = 0.20)

# With known positive correlation (smaller n)
creel_n_cpue(cv_catch = 0.8, cv_effort = 0.5, rho = 0.5, cv_target = 0.20)
```

creel_n_effort	<i>Calculate sampling days required to achieve a target CV on effort</i>
----------------	--

Description

Uses the stratified sample size formula from McCormick & Quist (2017) to determine how many sampling days are needed to achieve a target coefficient of variation on the effort estimate, given pilot variance estimates per day-type stratum.

Usage

```
creel_n_effort(cv_target, N_h, ybar_h, s2_h)
```

Arguments

cv_target	Numeric scalar. Target coefficient of variation for the effort estimate (e.g., 0.20 for 20 percent). Must be in (0, 1].
N_h	Named numeric vector. Total available days per stratum (e.g., c(weekday = 65, weekend = 28)). Values must be >= 1.
ybar_h	Numeric vector of same length as N_h. Pilot mean effort per day per stratum (e.g., angler-hours per day). Values must be >= 0.
s2_h	Numeric vector of same length as N_h. Pilot variance of effort per day per stratum. Values must be >= 0.

Details

Implements Cochran (1977) equation 5.25 under proportional allocation, as applied to creel surveys by McCormick & Quist (2017). The finite-population correction (FPC) factor is intentionally omitted (standard practice for pre-season planning where the goal is to determine how many days to sample, not to assess precision of a completed survey).

The per-stratum sample sizes n_h are computed from the total n_{total} under proportional allocation: $n_h = \text{ceiling}(n_{\text{total}} * N_h / \text{sum}(N_h))$. Because each stratum is ceiling-ed independently, $\text{sum}(n_h)$ may exceed n_{total} .

Value

A named integer vector. Elements named after strata in N_h give the sampling days required per stratum; element "total" gives Cochran's overall sample size *before* proportional allocation, and "allocated" the sum of the per-stratum values actually returned.

"allocated" is the number of sampling days the returned allocation commits to, and is the one to budget against. It is greater than or equal to "total": each stratum is rounded up independently, so the parts can sum to as much as $k - 1$ more than the unallocated optimum for k strata. Rounding up is deliberate – it keeps every stratum at or better than its share of cv_{target} .

References

- McCormick, J.L. and Quist, M.C. 2017. Sample size estimation for on-site creel surveys. North American Journal of Fisheries Management 37:970-983. doi:[10.1080/02755947.2017.1342723](https://doi.org/10.1080/02755947.2017.1342723)
- Cochran, W.G. 1977. Sampling Techniques, 3rd ed. Wiley, New York.

See Also

Other "Planning & Sample Size": [audit_strata\(\)](#), [compare_designs\(\)](#), [creel_n_camera\(\)](#), [creel_n_cpue\(\)](#), [creel_power\(\)](#), [cv_from_n\(\)](#), [optimal_n\(\)](#), [power_creel\(\)](#), [reallocate_strata\(\)](#), [simulate_strata_collapse\(\)](#)

Examples

```
# Two-stratum weekday/weekend example
creel_n_effort(
  cv_target = 0.20,
  N_h = c(weekday = 65, weekend = 28),
  ybar_h = c(50, 60),
  s2_h = c(400, 500)
)
```

creel_palette

Package-standard colour palette for tidycreeel plots

Description

creel_palette() returns a small set of package-standard colours derived from the tidycreeel site palette. Use these colours directly in custom plots or pair them with [theme_creel\(\)](#) for a consistent visual style.

Usage

```
creel_palette(n = NULL)
```

Arguments

n Optional integer number of colours to return. When NULL (default), returns the full named palette.

Value

When n is NULL, a named character vector of hex colours. Otherwise, a character vector of length n recycling through the base palette as needed.

See Also

Other "Visualisation": [autoplot.creeel_estimates\(\)](#), [autoplot.creeel_length_distribution\(\)](#), [autoplot.creeel_schedule\(\)](#), [plot_design\(\)](#), [theme_creel\(\)](#)

Examples

```
creel_palette()
creel_palette(3)
```

creel_power

Estimate statistical power to detect a change in CPUE between seasons

Description

Calculates the probability of detecting a fractional change in CPUE given a target sample size per season, a historical CV, and a significance level. Uses a two-sample normal approximation with equal group sizes.

Usage

```
creel_power(
  n,
  cv_historical,
  delta_pct,
  alpha = 0.05,
  alternative = c("two.sided", "one.sided")
)
```

Arguments

<code>n</code>	Integerish scalar (≥ 1). Number of interviews per season.
<code>cv_historical</code>	Numeric scalar (> 0). Coefficient of variation of CPUE from historical or pilot data.
<code>delta_pct</code>	Numeric scalar (> 0). Fractional change to detect, expressed as a proportion — e.g., 0.20 for a 20 percent change. Note: this is a fraction, not a percentage point.
<code>alpha</code>	Numeric scalar in $(0, 0.5]$. Type I error rate. Default is 0.05.
<code>alternative</code>	Character. Either "two.sided" (default) or "one.sided".

Details

Implements the two-sample normal approximation for power under equal group sizes, parameterised in terms of the CV:

$$ncp = |\delta| \cdot \sqrt{n/2} / CV_{historical}$$

For `alternative = "two.sided"`:

$$power = \Phi(ncp - z_{\alpha/2}) + \Phi(-ncp - z_{\alpha/2})$$

For `alternative = "one.sided"`:

$$power = \Phi(ncp - z_{\alpha})$$

where `delta` is the fractional effect size (`delta_pct`), `n` is the number of interviews per season, and `CV_historical` is the pilot CV of CPUE.

A warning is issued when `delta_pct > 5` because values greater than 5 are almost certainly input in percentage-point form rather than fractional form (e.g., 20 instead of 0.20).

Value

A numeric scalar in $(0, 1)$: estimated statistical power.

References

Cohen, J. 1988. Statistical Power Analysis for the Behavioral Sciences, 2nd ed. Lawrence Erlbaum Associates, Hillsdale, NJ.

See Also

Other "Planning & Sample Size": [audit_strata\(\)](#), [compare_designs\(\)](#), [creel_n_camera\(\)](#), [creel_n_cpue\(\)](#), [creel_n_effort\(\)](#), [cv_from_n\(\)](#), [optimal_n\(\)](#), [power_creel\(\)](#), [reallocate_strata\(\)](#), [simulate_strata_collapse\(\)](#)

Examples

```
# Two-sided power at n = 100, CV = 0.5, 20 percent change
creel_power(n = 100, cv_historical = 0.5, delta_pct = 0.20)

# One-sided test (higher power for same inputs)
creel_power(n = 100, cv_historical = 0.5, delta_pct = 0.20, alternative = "one.sided")
```

creel_schema	<i>Column-mapping contract for tidycreeel data sources</i>
--------------	--

Description

`creel_schema()` constructs a `creel_schema` S3 object that maps canonical tidycreeel column names to actual column and table names in a data source. The schema is the full connection contract consumed by `creel_connect()` and `fetch_*`() functions in the `tidycreeel.connect` companion package.

Construction is permissive — all column arguments default to `NULL`. Use [validate_creel_schema\(\)](#) to check that required columns for the given survey type are mapped.

Usage

```
creel_schema(
  survey_type = c("instantaneous", "bus_route", "ice", "camera", "aerial"),
  interviews_table = NULL,
  counts_table = NULL,
  catch_table = NULL,
  lengths_table = NULL,
  date_col = NULL,
  strata_cols = NULL,
  value_maps = NULL,
  catch_col = NULL,
  effort_col = NULL,
  trip_status_col = NULL,
  count_col = NULL,
  count_time_col = NULL,
  catch_uid_col = NULL,
  interview_uid_col = NULL,
  species_col = NULL,
  catch_count_col = NULL,
  catch_type_col = NULL,
  length_uid_col = NULL,
  length_mm_col = NULL,
```

```

length_bin_col = NULL,
length_count_col = NULL,
length_type_col = NULL,
harvest_col = NULL,
trip_duration_col = NULL,
trip_start_col = NULL,
interview_time_col = NULL,
n_anglers_col = NULL,
n_counted_col = NULL,
n_interviewed_col = NULL,
bank_anglers_col = NULL,
angler_boats_col = NULL,
non_ang_boats_col = NULL,
angler_type_col = NULL,
site_col = NULL,
circuit_col = NULL,
angler_method_col = NULL,
species_sought_col = NULL,
refused_col = NULL,
harvest_lengths_table = NULL,
release_lengths_table = NULL
)

```

Arguments

survey_type	Survey type. One of "instantaneous", "bus_route", "ice", "camera", or "aerial". Validated at construction via <code>match.arg()</code> .
interviews_table	Name of the interviews table in the data source.
counts_table	Name of the counts table in the data source.
catch_table	Name of the catch table in the data source.
lengths_table	Name of the lengths table in the data source. Used for both the harvest and release length fetches unless one of the two below names its own table.
date_col	Column name for survey date.
strata_cols	Stratum columns to carry through from the source, as a named character vector whose names are the columns the design refers to and whose values are the source columns holding them — <code>c(day_type = "DayType")</code> . An unnamed entry, <code>c("day_type")</code> , means the source already uses the design's name. Unlike every other field here, a stratum has no canonical tidycreel name: <code>add_counts()</code> matches <code>design\$strata_cols</code> — the caller's own calendar column names — against the names of the counts frame, so the mapping has to be two-sided. Without it a fetched counts frame reaches <code>add_counts()</code> with no stratum label and any design built with <code>strata = aborts</code> (GH #171).
value_maps	Source vocabularies for the coded columns, as a named list keyed by canonical column — <code>trip_status</code> , <code>catch_type</code> , <code>length_type</code> . Each entry is a fully named character vector mapping the source's own codes to canonical values: <code>c("1" = "complete", "2" = "incomplete")</code> . Names are what the source writes, values what tidycreel means.

Every downstream filter matches the canonical literals, so a source that codes these columns has to declare what its codes mean. Values already canonical pass through untouched; anything neither mapped nor canonical aborts at the fetch, where the source is still in view, rather than being recoded by hand afterwards — a hand recode folds an undeclared third code ("refused", "unknown") into complete or incomplete silently (GH #128).

catch_col	Column name for catch count in interviews.
effort_col	Column name for effort (hours) in interviews.
trip_status_col	Column name for trip status in interviews.
count_col	Column name for total angler count in counts (legacy single-column format).
count_time_col	Column name for the time of a count observation, such as "16:30" or "am". Optional. Map it whenever the source records more than one count per sampled day: the fetched count_time column is what add_counts() 's count_time_col argument groups on, and without it those rows reach the design as separate sampled days rather than as repeat looks at one, which sums the day's effort instead of averaging it and leaves the within-day variance component uncomputed (GH #129). Carried through as character: it is a label that distinguishes observations, not a quantity, and a source may write a clock time in any format.
catch_uid_col	Column name for catch unique identifier.
interview_uid_col	Column name for interview unique identifier.
species_col	Column name for species.
catch_count_col	Column name for catch count in the catch table.
catch_type_col	Column name for catch type (harvest/release).
length_uid_col	Column name for length unique identifier.
length_mm_col	Column name for fish length (mm). Map it only for individually measured fish; a bin label belongs in length_bin_col, whose name does not assert a unit.
length_bin_col	Column name for a length-bin label, such as "300-350". Optional, and mutually exclusive with length_mm_col on any given row: a fish is either measured or binned. Pass the fetched length_bin column as add_lengths() 's length argument together with release_format = "binned" (GH #127).
length_count_col	Column name for the number of fish a binned length row represents. Optional, but required by add_lengths() whenever binned release rows are present: a binned row is frequency-weighted, so dropping the count weights the length distribution by row multiplicity instead of by fish (GH #127). NA on individually measured rows.
length_type_col	Column name for length type.
harvest_col	Column name for harvest count.
trip_duration_col	Column name for trip duration.

trip_start_col	Column name for trip start time.
interview_time_col	Column name for interview time.
n_anglers_col	Column name for number of anglers.
n_counted_col	Column name for number of anglers counted.
n_interviewed_col	Column name for number of anglers interviewed.
bank_anglers_col	Column name for bank (shore) angler count in counts.
angler_boats_col	Column name for boats carrying anglers in counts.
non_ang_boats_col	Column name for boats carrying no anglers in counts. Recorded by some agencies and not others; leave NULL where it is not.
angler_type_col	Column name for angler type.
site_col	Column name for the site an interview was taken at. Bus-route designs need it to join the site inclusion probability; without it <code>add_interviews()</code> cannot build the π_i term (GH #126).
circuit_col	Column name for the bus-route circuit an interview belongs to. Required alongside <code>site_col</code> for the bus-route expansion (GH #126).
angler_method_col	Column name for fishing method.
species_sought_col	Column name for target species.
refused_col	Column name for refused interviews indicator.
harvest_lengths_table	Name of the harvest lengths table, when the source keeps harvest and release lengths in separate tables. Falls back to <code>lengths_table</code> when not given.
release_lengths_table	Name of the release lengths table, on the same terms as <code>harvest_lengths_table</code> .

Value

A `creel_schema` S3 object.

See Also

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
s <- creel_schema(
  survey_type      = "instantaneous",
  interviews_table = "vwInterviews",
  counts_table     = "vwCounts",
  date_col         = "SurveyDate",
  catch_col        = "TotalCatch",
  effort_col       = "EffortHours",
  trip_status_col  = "TripStatus",
  count_col        = "AnglerCount"
)
print(s)
```

creel_vocabulary

Canonical vocabularies for the coded columns

Description

The exact values `tidycreel` matches on for the three columns whose meaning is a fixed vocabulary rather than a number: `trip_status`, `catch_type` and `length_type`. Every downstream filter compares against these literals, so a source that codes one of these columns has to be translated before its values can be trusted — see the `value_maps` argument of [creel_schema\(\)](#).

Exported because `tidycreel.connect` translates source codes at the fetch and has to check its targets against the same list this package filters on; a second copy of the vocabulary would be free to drift from this one.

Usage

```
creel_vocabulary(column = NULL)
```

Arguments

<code>column</code>	Optional canonical column name. When <code>NULL</code> (default) the whole named list is returned; otherwise the character vector for that column.
---------------------	--

Value

A named list of character vectors, or one character vector when `column` is given.

See Also

Other "Survey Design": [add_catch\(\)](#), [add_counts\(\)](#), [add_interviews\(\)](#), [add_lengths\(\)](#), [add_sections\(\)](#), [as_creel_svydesign\(\)](#), [as_hybrid_svydesign\(\)](#), [compute_angler_effort\(\)](#), [compute_effort\(\)](#), [creel_design\(\)](#), [creel_schema\(\)](#), [derive_angler_count\(\)](#), [est_effort_camera\(\)](#), [impute_camera_counts\(\)](#), [mean_party_size\(\)](#), [prep_counts_boat_party\(\)](#), [prep_counts_daily_effort\(\)](#), [prep_interview_catch\(\)](#), [prep_interviews_trips\(\)](#), [validate_creel_schema\(\)](#)

Examples

```
creel_vocabulary()
creel_vocabulary("trip_status")
```

cv_from_n

Compute the expected CV achievable with a known sample size

Description

Calculates the coefficient of variation attainable given a fixed sample size, acting as the algebraic inverse of `creel_n_effort()` (when `type = "effort"`) or `creel_n_cpue()` (when `type = "cpue"`).

Usage

```
cv_from_n(type = c("effort", "cpue"), n, ...)
```

Arguments

- | | |
|------|---|
| type | Character. Either "effort" or "cpue". Selects the formula branch and required additional arguments. |
| n | Integerish scalar (≥ 1). Available sample size (sampling days for "effort", interviews for "cpue"). |
| ... | Additional arguments passed to the relevant branch:
For type = "effort": <ul style="list-style-type: none"> • N_h — Named numeric vector; total available days per stratum (≥ 1). • ybar_h — Numeric vector; pilot mean effort per day per stratum (≥ 0). • s2_h — Numeric vector; pilot variance of effort per day per stratum (≥ 0). For type = "cpue": <ul style="list-style-type: none"> • cv_catch — Numeric scalar; pilot CV of catch per interview (> 0). • cv_effort — Numeric scalar; pilot CV of effort per interview (> 0). • rho — Numeric scalar; pilot correlation between catch and effort, in [-1, 1]. Default is 0. |

Details

Effort branch (`type = "effort"`):

$$CV = \frac{\sqrt{N \sum_h N_h s_h^2 / n}}{\sum_h N_h \bar{y}_h}$$

where $N = \sum_h N_h$.

This is the inverse of the Cochran (1977) stratified sample-size formula implemented in `creel_n_effort()`.

CPUE branch (`type = "cpue"`):

$$CV = \sqrt{(CV_{catch}^2 + CV_{effort}^2 - 2\rho \cdot CV_{catch} \cdot CV_{effort}) / n}$$

This is the inverse of the ratio-estimator formula implemented in `creel_n_cpue()`.

Because `creel_n_effort()` and `creel_n_cpue()` apply `ceiling()`, the round-trip property is `cv_from_n(type, n = creel_n*(cv, ...), ...) <= cv` (the recovered CV is at or below the target).

Value

A numeric scalar (> 0): the expected CV achievable at sample size n .

References

Cochran, W.G. 1977. Sampling Techniques, 3rd ed. Wiley, New York.

See Also

`creel_n_effort()`, `creel_n_cpue()`

Other "Planning & Sample Size": `audit_strata()`, `compare_designs()`, `creel_n_camera()`, `creel_n_cpue()`, `creel_n_effort()`, `creel_power()`, `optimal_n()`, `power_creel()`, `reallocate_strata()`, `simulate_strata_collapse()`

Examples

```
# Effort round-trip
n_days <- creel_n_effort(0.20,
  N_h = c(weekday = 65, weekend = 28),
  ybar_h = c(50, 60), s2_h = c(400, 500)
)
cv_from_n("effort",
  n = n_days[["total"]],
  N_h = c(weekday = 65, weekend = 28),
  ybar_h = c(50, 60), s2_h = c(400, 500)
)

# CPUE round-trip
n_int <- creel_n_cpue(cv_catch = 0.8, cv_effort = 0.5, rho = 0, cv_target = 0.20)
cv_from_n("cpue", n = n_int, cv_catch = 0.8, cv_effort = 0.5, rho = 0)
```

day_length

Day length for a latitude and date

Description

Computes the number of hours between sunrise and sunset at a given latitude on a given date, using the CBM model of Forsythe et al. (1995). The calculation is a closed form – no lookup tables, network access, or location database is involved.

Only latitude is needed. Longitude and time zone shift *when* sunrise and sunset occur but not the interval between them, so they are not arguments.

Usage

```
day_length(lat, date, horizon = "sunset")
```

Arguments

lat	Numeric latitude in decimal degrees, positive north, in [-90, 90]. Recycled against date.
date	A Date vector (or anything <code>as.Date()</code> accepts). Recycled against lat.
horizon	How far the sun must be below the horizon for the day to count as over. Either one of "sunset" (the default; 0.833 degrees, accounting for the solar disc and atmospheric refraction), "civil" (6), "nautical" (12), "astronomical" (18), or a number giving the depression angle in degrees directly.

Details

Day length is astronomical, and the effort estimators want something else. In Hoenig et al. (1993) and Pope et al. (Ch. 17), the daily expansion factor T_d is the length of the period the counts were *randomised within* – a property of the survey design, set by regulation, access hours, or the field protocol. It is often close to daylight and it is not the same quantity. Use `day_length()` to build simulated or planned surveys, and pass the period your protocol actually used to `add_counts()`.

Above the Arctic and Antarctic circles the sun may not rise or set at all. In those cases the result saturates at 0 or 24 rather than erroring.

Value

A numeric vector of day lengths in hours, the length of the longer of lat and date.

References

Forsythe, W.C., Rykiel, E.J., Stahl, R.S., Wu, H., Schoolfield, R.M. (1995). A model comparison for daylength as a function of latitude and day of year. *Ecological Modelling* 80:87-95. doi:10.1016/03043800(94)00034F

Hoenig, J.M., Robson, D.S., Jones, C.M., Pollock, K.H. (1993). Scheduling counts in the instantaneous and progressive count methods for estimating sportfishing effort. *North American Journal of Fisheries Management* 13:723-736.

See Also

`simulate_creel_data()`, `add_counts()`

Other "Simulation": `simulate_creel_catch()`, `simulate_creel_data()`

Examples

```
# A single day at Kearney, Nebraska
day_length(40.699, as.Date("2024-06-21"))

# A whole season, for use as a simulated expansion factor
season <- seq(as.Date("2024-05-01"), as.Date("2024-08-31"), by = "day")
```

```
summary(day_length(40.699, season))

# Anglers fish into twilight; civil twilight adds roughly an hour in June
day_length(40.699, as.Date("2024-06-21"), horizon = "civil")

# Latitude drives the seasonal swing
day_length(c(25, 45, 65), as.Date("2024-12-21"))
```

derive_angler_count	<i>Derive an angler count from its components</i>
---------------------	---

Description

Builds the single angler-count column that `add_counts()` needs from the columns a creel clerk actually records. Counts are commonly split across bank anglers and boats, and the estimators need one number per count.

Two forms are supported, matching the two ways a boat's anglers get onto the form:

- **Direct counts** — supply `boat_anglers`, the counted number of anglers aboard. Appropriate when the clerk could see and count them.
- **Boat-party expansion** — supply `boat_count` and `party_size`, and the boats are expanded by the mean anglers per boat party. Appropriate when boats were counted but the people aboard could not be counted reliably.

Usage

```
derive_angler_count(
  counts,
  bank = NULL,
  boat_anglers = NULL,
  boat_count = NULL,
  party_size = NULL,
  party_size_se = NULL,
  to = "angler_count"
)
```

Arguments

<code>counts</code>	A data frame of count observations.
<code>bank</code>	Optional tidy selector for the bank (shore) angler count column.
<code>boat_anglers</code>	Optional tidy selector for a directly counted boat-angler column. Mutually exclusive with <code>boat_count</code> .
<code>boat_count</code>	Optional tidy selector for the counted number of boats. Requires <code>party_size</code> . Mutually exclusive with <code>boat_anglers</code> .

party_size	Mean anglers per boat party, used to expand boat_count. One of: a single number; a tidy selector for a numeric column of counts; or a data frame of the kind <code>mean_party_size()</code> returns with <code>by</code> , which is joined onto counts by its non-numeric columns.
party_size_se	Optional standard error of party_size, in the same three shapes. Defaults to the "se" attribute of party_size when it has one, so <code>mean_party_size()</code> output propagates on its own. There is deliberately no zero default; see the section above.
to	Name of the column to write. Defaults to "angler_count".

Details

The two boat forms are separate arguments on purpose. `boat_count` is a count of **hulls**, not people, and adding it to an angler total is a units error that produces a plausible-looking number. Requiring `party_size` alongside it makes that mistake impossible to commit by accident.

Components are added with `na.rm = FALSE`. If bank anglers are missing for a count and boat anglers are 5, the total is unknown, not 5 — a missing count and a count of zero are different observations and are kept different here.

Pooling bank and boat anglers into one count assumes both are detected the same way and are being reported as one quantity. Where detection probabilities or catch rates differ between them, estimate the two as separate domains instead of adding them.

Value

counts with the derived column appended, and the columns consumed to build it (`bank`, `boat_anglers`, `boat_count`) removed — they are superseded by the derived count and, where applicable, by `expansion_basis`. Leaving them in produced a table that varied between sub-counts of one sampling unit, which `add_counts()` cannot distinguish from an undeclared structural dimension (GH #162). The destination column is never dropped, even when it is also one of the inputs.

When a party-size standard error is available, four further columns are appended for the estimators to read: `expansion_basis` (the boat count, which is what the multiplier acts on), `expansion_se`, `expansion_group` (which rows share one estimated multiplier, and so carry perfectly correlated error), and `expansion_of` (the column the basis is the derivative of). `add_counts()` recognises all four and excludes them from count-column detection.

They must travel together and must reach `add_counts()` alongside the column named in `expansion_of`. Transforming that column in between — multiplying a count by a shift length, say — scales the count but not its derivative, and `add_counts()` refuses rather than propagate a component that is understated by exactly the scale factor. Pass the per-day count and let `period_length_col` do the multiplication instead.

All four are written by the package and are not user inputs. In particular, editing `expansion_of` to name a transformed column silences that refusal whether or not the basis was actually rescaled, which re-enables the very defect the check exists to catch. Rescale through `period_length_col`, which scales both together and can be verified, rather than by asserting that you did (GH #148).

The party size is an estimate

A mean party size taken from interviews is itself estimated, and it multiplies the boat component of **every** count. Its error is therefore perfectly correlated across counts and does not shrink as counts

are added – averaging more counts will not reduce it. Left out, the reported effort standard error is too small; the estimate itself is unaffected.

Supply `party_size_se` to carry that term through to the effort standard error. `mean_party_size()` returns it as a "se" attribute, which is picked up automatically when its output is passed as `party_size`, so the usual pipeline propagates the term without any extra argument.

When no standard error is available the term is **omitted rather than set to zero**. A zero would produce a standard error identical to an unpropagated one while looking propagated, which is worse than a documented omission. The returned table carries no expansion columns in that case, and `attr(<estimates>, "se_expansion")` is NULL rather than 0.

See Also

`mean_party_size()`, `add_counts()`, `prep_counts_boat_party()`

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
counts <- data.frame(
  date = as.Date("2024-06-01") + 0:1,
  day_type = c("weekday", "weekend"),
  bank_anglers = c(4L, 9L),
  angler_boats = c(3L, 7L),
  boat_anglers = c(7L, 16L)
)

# Direct counts
derive_angler_count(counts, bank = bank_anglers, boat_anglers = boat_anglers)

# Boat-party expansion with a single mean
derive_angler_count(
  counts,
  bank = bank_anglers,
  boat_count = angler_boats,
  party_size = 2.4
)

# Expansion with a stratum-specific mean
mps <- data.frame(day_type = c("weekday", "weekend"), mean_party_size = c(2.1, 2.8))
derive_angler_count(
  counts,
  bank = bank_anglers,
  boat_count = angler_boats,
  party_size = mps
)
```

estimate_angler_n	<i>Estimate angler population size via closed-population mark-recapture</i>
-------------------	---

Description

Computes a closed-population mark-recapture estimate of total angler population size ($\hat{N}$) using one of three estimators:

- **Chapman** (default, method = "chapman"): A bias-corrected version of the Petersen estimator recommended when recaptures are small. $\hat{N} = \frac{(M+1)(n+1)}{(m+1)} - 1$
- **Petersen** (method = "petersen"): The unadjusted Lincoln-Petersen estimator. Requires at least 7 recaptures ($m \geq 7$) to avoid large positive bias; use Chapman for smaller recapture counts. $\hat{N} = \frac{M \cdot n}{m}$
- **Schnabel** (method = "schnabel"): A multi-occasion weighted estimator for $K \geq 2$ sampling occasions, carrying Chapman's (1952) small-sample correction by default. $\hat{N} = \frac{\sum M_k n_k}{\sum m_k + 1}$ CI uses the Poisson branch when $\sum m_k < 50$ and the normal approximation on $1/\hat{N}$ otherwise, on $K - 1$ degrees of freedom (Hansen & Van Kirk 2018, eq. A.5).
- **Schumacher-Eschmeyer** (method = "schumacher"): The regression alternative to Schnabel for $K \geq 3$ occasions, fitting m_k/n_k against M_k through the origin with slope $1/N$. $\hat{N} = \frac{\sum n_k M_k^2}{\sum m_k M_k}$ Interval from Seber (1982) eq. (4.17) on $K - 2$ degrees of freedom. Also carries Dettloff's (2023) eq. (8) small-sample correction by default.

Usage

```
estimate_angler_n(
  M,
  n,
  m,
  method = "chapman",
  conf_level = 0.95,
  ci_method = c("logit", "delta", "bootstrap"),
  B = 2000L,
  bias_adjust = TRUE
)
```

Arguments

- | | |
|---|--|
| M | integer or numeric. Number of marked animals released (first sample). For method = "schnabel", a vector of cumulative marked-at-large counts before each sampling occasion (M[1] = 0). |
| n | integer or numeric. Number captured in second sample. For Schnabel, a vector of per-occasion catch counts (same length as M). |
| m | integer or numeric. Number of recaptures. Scalar for Chapman and Petersen; vector (same length as M) for Schnabel. |

method	character(1). One of "chapman" (default), "petersen", "schnabel", or "schumacher".
conf_level	numeric. Confidence level for the CI. Default 0.95.
ci_method	character(1). CI construction for the Chapman and Petersen branches: "logit" (default) is Sadinle's (2009) 0.5 transformed logit interval; "delta" is the symmetric Wald interval $\hat{N} \pm t_{\alpha/2, m-1} SE(\hat{N})$ that was the default before 3.0.0; "bootstrap" keeps the "logit" bounds and additionally appends ci_lo_boot and ci_hi_boot columns from a parametric bootstrap via stats::rbinom(), attaching attr(result, "boot_samples"). The Schnabel branch ignores this argument for its analytic bounds — it always inverts Poisson quantiles or uses the t approximation on $1/\hat{N}$, per Hansen & Van Kirk (2018) — but still honours "bootstrap" for the extra columns. Schumacher-Eschmeyer does not support "bootstrap" and refuses it (GH #209). Its interval comes from the weighted regression itself (Seber 1982 eq. 4.17), whose variance is the residual mean square about the fitted line — not the binomial noise in the recaptures that the other methods resample. A bootstrap over m_k alone would report a different quantity under the same column names. "logit" and "delta" both yield that regression interval.
B	integer(1). Number of bootstrap replicates when ci_method = "bootstrap". Default 2000L.
bias_adjust	logical(1). Multi-occasion methods only; ignored by the Chapman and Petersen branches, which carry their own bias handling. TRUE (default, new in 3.0.0) applies the small-sample correction — Chapman's (1952), dividing by $\sum m_k + 1$, for Schnabel, and Dettloff's (2023) eq. (8) for Schumacher-Eschmeyer. FALSE restores the unadjusted forms, which are what fishmethods::schnabel() computes for both and, for Schnabel, the only form available before 3.0.0.

Details

Why the Chapman and Petersen default is not a Wald interval. $\hat{N}$ is a ratio with a small integer denominator, so its sampling distribution is strongly right-skewed and a symmetric interval leaves the parameter space. Evans et al. (1996) measured Wald coverage failing on one side 27.9% and Dettloff (2023) report the same. With $M = 200$, $n = 50$ and $m = 3$ the Wald lower bound is -2124.8; at $m = 5$ it is 48.7, below the 245 individuals actually observed. Chapman is recommended precisely when recaptures are few, so this is the regime the default estimator is chosen for.

The default "logit" interval is Sadinle's (2009) 0.5 transformed logit, built on the 2×2 capture table ($n_{11} = m$, $n_{12} = M - m$, $n_{21} = n - m$) with 0.5 added to each cell. Sadinle compared nine intervals and found it "the best of the intervals reported here", with near-nominal coverage even for small populations and capture probabilities near 0 or 1, where profile-likelihood and Monte Carlo intervals both degrade. Its lower limit is guaranteed never to fall below $n_{11} + n_{12} + n_{21}$, the number of individuals actually seen — the property the Wald interval lacks. It is closed-form and always computable, since the 0.5 continuity correction removes every zero-count division.

One consequence worth knowing. When $m = n$ — every individual in the second sample was already marked — the estimator saturates at $\hat{N} = M$, which is also the observed count. The logit lower limit then sits fractionally *above* $\hat{N}$, because the data imply $N > M$ rather than $N = M$. This is the interval being informative at a boundary, not an error; pass ci_method = "delta" if a bound that brackets the point estimate matters more than coverage.

ci_method = "delta" reproduces the pre-3.0.0 bounds exactly.

Choosing between Schnabel and Schumacher-Eschmeyer, and a warning about how not to.

They use identical field data and differ in how they pool it: Schnabel is a ratio of sums, Schumacher-Eschmeyer a weighted regression through the origin. Seber (1982) expects the regression form "to be robust with regard to departures from the underlying assumptions" and recommends using it "in conjunction with the other methods" — as a cross-check, not a replacement. That is a weaker claim than it is sometimes reported as; Seber neither demonstrates the robustness nor calls it the most robust method. Dettloff (2023) found the two adjusted forms "effectively equivalent at larger sample sizes", with Schumacher-Eschmeyer less variable and Schnabel reaching unbiasedness slightly sooner.

Do not pick whichever gives the narrower interval. Hansen & Van Kirk (2018) computed both and "selected the mark-recapture estimator that produced the smallest 95% the narrower of two intervals after seeing them conditions on the luckier draw, so the reported interval is narrower than its nominal level. tidycreel therefore does not implement the selection rule. Decide between the estimators on design grounds before looking at the answer, or report both.

The Schnabel upper bound at very few recaptures. The Poisson interval inverts the distribution of $\sum m_k$, so it needs the lower quantile $q_{\alpha/2}$ in its denominator. That quantile is zero whenever $\sum m_k \leq 3$ at the 95% Inf. Following Hansen & Van Kirk (2018) eq. (A.4), tidycreel substitutes Iliencko's (2013) continuous Poisson in exactly that case — it has distribution function $\Gamma(x, \lambda)/\Gamma(x)$, is positive there, and so returns a finite bound. The substitution fires only where the discrete quantile is zero; from $\sum m_k \geq 4$ the continuous quantile sits just above the discrete one, so this is a targeted patch rather than a change of method.

Read that bound for what it is. It comes from a continuous interpolation of a discrete distribution at one to three total recaptures, not from the data, and it is wide. It stands in for "the data do not bound this above" rather than measuring anything, which is why the function still warns when it fires. Iliencko's construction is the genuine interpolant — his eq. (1) shows the same expression returns the discrete Poisson CDF at integer x — but interpolating at $\sum m_k = 1$ is still interpolating.

Where the Petersen $m \geq 7$ guard comes from. The threshold is a practical stand-in, not a derivation, and it is worth knowing why no exact one is available. Robson & Regier (1964) give two conditions: Chapman is exactly unbiased when $M + n \geq N$, and its negative bias stays under $2\sqrt{Mn} \geq 2\sqrt{N}$ — the geometric mean of marks and captures at least twice the square root of the population size. Both depend on N , the unknown being estimated. Dettloff (2023) calls this "paradoxical" and treats such rules as "a way of avoiding inaccurate estimates from absurdly small sample sizes based on an educated guess of the order of magnitude" of N . A fixed m threshold is that guess made concrete; it rules out the regime where Petersen's positive bias is severe without pretending to a precision the conditions cannot deliver. Chapman is the better default at any recapture count and is what the error message points to.

Why Schnabel is bias-adjusted by default. Each m_k is approximately Poisson with parameter $M_k n_k / N$, which motivated Chapman's (1952) +1 correction to the recapture total. Dettloff (2023) simulated both forms and found the unadjusted estimator turns biased *high* at moderate sample sizes before settling, whereas the adjusted form has bias that "approaches zero as the sample size increases without ever becoming positive", with lower variance and no cost at large samples; he recommends the adjusted estimators "in place of the originals in all scenarios". The package already defaults to the analogous +1 correction at two occasions (method = "chapman"), and Schnabel reduces exactly to Lincoln-Petersen at $K = 2$, so leaving Schnabel unadjusted made bias handling depend on how many occasions were sampled. The relative shift is $-1/(\sum m_k + 1)$: -33% 500. Pass `bias_adjust = FALSE` for the previous form.

Value

A creel_estimates S3 object with method = "mark-recapture-chapman" (or petersen/schnabel) and an estimates tibble with columns: parameter, estimate, se, ci_lower, ci_upper, n (total recaptures).

References

- Hansen, J. M., & Van Kirk, R. W. (2018). A mark-recapture-based approach for estimating angler harvest. *North American Journal of Fisheries Management*, 38(2), 400–410. doi:10.1002/nafm.10038
- Sadinle, M. (2009). Transformed logit confidence intervals for small populations in single capture-recapture estimation. *Communications in Statistics - Simulation and Computation*, 38(9), 1909–1924. doi:10.1080/03610910903168595
- Evans, M. A., Kim, H.-M., & O'Brien, T. E. (1996). An application of profile-likelihood based confidence interval to capture-recapture estimators. *Journal of Agricultural, Biological, and Environmental Statistics*, 1(1), 131–140. doi:10.2307/1400565
- Dettloff, K. (2023). Assessment of bias and precision among simple closed population mark-recapture estimators. *Fisheries Research*, 265, 106756. doi:10.1016/j.fishres.2023.106756
- Chapman, D. G. (1952). Inverse, multiple and sequential sample censuses. *Biometrics*, 8(4), 286–306. doi:10.2307/3001864
- Robson, D. S., & Regier, H. A. (1964). Sample size in Petersen mark-recapture experiments. *Transactions of the American Fisheries Society*, 93(3), 215–226. doi:10.1577/15488659(1964)93[215:SSIPME]2.0.CO;2
- Iliencko, A. (2013). Continuous counterparts of Poisson and binomial distributions and their properties. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eotvos Nominatae, Sectio Computatorica*, 39, 137–147.
- Schnabel, Z. E. (1938). The estimation of the total fish population of a lake. *The American Mathematical Monthly*, 45(6), 348–352. doi:10.2307/2304025
- Chapman, D. G. (1951). Some properties of the hypergeometric distribution with applications to zoological sample censuses. *University of California Publications in Statistics*, 1(7), 131–160.
- Schumacher, F. X., & Eschmeyer, R. W. (1943). The estimation of fish populations in lakes or ponds. *Journal of the Tennessee Academy of Science*, 18, 228–249.
- Seber, G. A. F. (1982). *The Estimation of Animal Abundance and Related Parameters*, 2nd ed. Macmillan, New York.
- De Lury, D. B. (1958). The estimation of population size by a marking and recapture procedure. *Journal of the Fisheries Research Board of Canada*, 15(1), 19–25. doi:10.1139/f58003

See Also

Other Estimation: [estimate_exploitation_rate\(\)](#), [estimate_mr_harvest\(\)](#)

Examples

```
# Chapman (default) – bias-corrected Petersen
result <- estimate_angler_n(M = 200L, n = 50L, m = 10L)
print(result)
```

```

# Petersen - requires m >= 7
result_p <- estimate_angler_n(M = 200L, n = 50L, m = 10L, method = "petersen")
print(result_p)

# Schnabel - multi-occasion with parallel vectors
result_s <- estimate_angler_n(
  M = c(0L, 47L, 91L, 131L),
  n = c(50L, 50L, 50L, 50L),
  m = c(0L, 4L, 6L, 8L),
  method = "schnabel"
)
print(result_s)

# Schumacher-Eschmeyer - the regression alternative, needs >= 3 occasions
result_se <- estimate_angler_n(
  M = c(0L, 47L, 91L, 131L),
  n = c(50L, 50L, 50L, 50L),
  m = c(0L, 4L, 6L, 8L),
  method = "schumacher"
)
print(result_se)

```

estimate_angler_trips *Estimate angler trips from extrapolated effort*

Description

Computes estimated trips by dividing extrapolated effort by the mean trip length per stratum, with Delta Method variance propagation (Powell 2007). This is a composable estimator: the effort object must be pre-computed via [estimate_effort](#) before calling this function.

The divisor is hours per trip, so the result comes back in whichever actor the effort was measured in: angler-hours give angler trips, party-hours give party trips. The returned unit field records which, and is NA when the effort's own unit was unknown. The method string is "angler-trips" in every case and so is not a guide to the actor.

Usage

```
estimate_angler_trips(effort, design, conf_level = 0.95, ...)
```

Arguments

effort	A creel_estimates object returned by estimate_effort . Must have a numeric estimate column and a se column in effort\$estimates.
design	A creel_design object with interview data containing a trip duration column (set via <code>add_interviews(trip_duration = ...)</code>). Used to compute per-stratum mean trip length.
conf_level	Confidence level for confidence intervals. Default 0.95.
...	Reserved for future arguments.

Value

A `creel_estimates` object with `method = "angler-trips"` and `variance_method = "delta"`. The estimates tibble contains:

by_vars columns Any grouping columns from the effort object (if grouped).

estimate Estimated trips per stratum (effort / mean trip length), in the actor the effort was measured in.

se Standard error via Delta Method variance propagation.

ci_lower Lower confidence interval bound.

ci_upper Upper confidence interval bound.

n Number of interviews contributing to mean trip length per stratum.

For grouped effort, an `.overall` row is appended with `estimate = sum(stratum trips)` and `se` propagated by addition in quadrature.

References

Powell, L. A. (2007). Approximating variance of demographic parameters using the delta method. *Journal of Wildlife Management*, 71(3), 1018-1024.

See Also

[estimate_effort](#), [estimate_exploitation_rate](#)

Examples

```
data(example_calendar)
data(example_counts)
data(example_interviews)
design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, trip_duration = trip_duration
)
effort <- estimate_effort(design)
estimate_angler_trips(effort, design)
```

estimate_catch_rate	<i>Estimate CPUE (Catch Per Unit Effort) from a creel survey design</i>
---------------------	---

Description

Computes CPUE estimates with standard errors and confidence intervals from a creel survey design with attached interview data. Supports both ratio-of-means (for complete trips) and mean-of-ratios (for incomplete trips) estimation methods.

Usage

```
estimate_catch_rate(
  design,
  by = NULL,
  variance = "taylor",
  conf_level = 0.95,
  estimator = NULL,
  use_trips = NULL,
  truncate_at = 0.5,
  targeted = TRUE,
  missing_sections = "warn",
  force_origin = TRUE
)
```

Arguments

design	A creel_design object with interviews attached via add_interviews . The design must have an interview survey object constructed with catch and effort columns.
by	Optional tidy selector for grouping variables. Accepts bare column names (e.g., <code>by = day_type</code>), multiple columns (e.g., <code>by = c(day_type, location)</code>), or tidysselect helpers (e.g., <code>by = starts_with("day")</code>). When NULL (default), computes a single CPUE estimate across all interviews. Two kinds of column are not groupings and are refused: the interview id, as registered by add_catch() , add_lengths() or add_ages() , which holds one value per interview and so leaves no within-group variance to estimate; and columns the package derived rather than the user supplying, such as <code>.angler_effort</code> . A wildcard selector drops the derived columns silently; asking for one specifically is an error. A column of your own is never treated as derived, whatever it is called.
variance	Character string specifying variance estimation method. Options: "taylor" (default, Taylor linearization), "bootstrap" (bootstrap resampling with 500 replicates), or "jackknife" (jackknife resampling, automatic JK _n /JK ₁ selection).
conf_level	Numeric confidence level for confidence intervals (default: 0.95 for 95% confidence intervals). Must be between 0 and 1.
estimator	Character string specifying estimation method. Options: "ratio-of-means" (default, for complete trips), "mor" (mean-of-ratios, for incomplete trips), or "mortr" (truncated mean-of-ratios — same as "mor" but <code>truncate_at</code> is mandatory and defaults to 0.5 h). MOR and MORtr require the <code>trip_status</code> field and error if no incomplete trips are available. See Details.
use_trips	Character string specifying which trip type to use when <code>trip_status</code> field is provided. Options: "complete" uses only complete trips with ratio-of-means estimator; "incomplete" uses only incomplete trips with MOR; "all" uses all interviews (complete + incomplete) with MOR — the correct choice for roving designs; "diagnostic" estimates CPUE using both trip types and returns a comparison table. Default is NULL. When NULL and <code>interview_type =</code>

	"roving" (set via <code>add_interviews</code>), automatically defaults to "all" + MOR. Otherwise defaults to "complete". Parameter is ignored when <code>trip_status</code> field is not provided (backward compatibility). For bus-route and ice designs the set is "complete" (default), "incomplete" or "diagnostic"; "all" is not an estimator there, and the roving auto-route to "all" + MOR does not apply. See Details.
<code>truncate_at</code>	Numeric minimum trip duration (hours) for MOR estimation. Default is 0.5 hours (30 minutes) per Hoenig et al. (1997) to prevent unstable variance from very short trips. Trips with duration < <code>truncate_at</code> are excluded before MOR estimation. Set to NULL to disable truncation (research mode only). Ignored for ratio-of-means estimator.
<code>targeted</code>	Logical. When TRUE (default), all trips are used. When FALSE, zero-catch trips are excluded before estimation — appropriate for non-targeted species where most trips have zero catch. The estimate is then the rate among the trips that were kept, not the fishery-wide rate. A <code>cli_warn()</code> is emitted when more than 70% of trips have zero catch and <code>targeted = TRUE</code> (possible mis-specification). Which estimators read it, and what "zero catch" means to each: • "mor" / "mortr" without <code>by = species</code>: excludes trips with zero <i>total</i> catch. • "mor" / "mortr" with <code>by = species</code>: excludes, per species, the trips that caught none of that species. The 70% total catch, which made them inert on any species request (GH #304). • "regression" with <code>by = species</code>: same per-species exclusion (GH #290). • "regression" without <code>by = species</code>: ignored. • "ratio-of-means": ignored on every path.
<code>missing_sections</code>	Character string controlling behavior when a registered section has no interview observations. "warn" (default) emits a <code>cli_warn()</code> and inserts an NA row with <code>data_available = FALSE</code> . "error" aborts with <code>cli_abort()</code> . Ignored for non-sectioned designs.
<code>force_origin</code>	Logical. When <code>estimator = "regression"</code> , whether to force the regression through the origin ($\text{catch} \sim \text{effort} - 1$). Default TRUE (standard CPUE ₃ formulation per Petrere et al. 2010). Set to FALSE to allow a free intercept. Ignored for other estimators.

Details

Trip Type Selection (`use_trips`): When `trip_status` is provided, the `use_trips` parameter controls which trips are used for estimation. For access-point designs (`interview_type = "access"`, the default), `use_trips` defaults to "complete": interviews are taken at trip end, so complete trips are representative and avoid length-of-stay bias (Pollock et al. 1994). For roving designs (`interview_type = "roving"`), `use_trips` automatically defaults to "all" with the MOR estimator: the clerk intercepts trips mid-stream, so all interviews — complete and incomplete — are valid inputs (Hoenig et al. 1997). Set `use_trips = "incomplete"` to restrict MOR to incomplete trips only. Set `use_trips = "diagnostic"` to run both complete and incomplete trip estimation and return a comparison object with difference metrics. Diagnostic mode requires both trip types. When `trip_status` is not provided, `use_trips` is ignored for backward compatibility with v0.2.0.

Ratio-of-Means (default): CPUE is estimated as the ratio of total catch to total effort. This is the appropriate estimator for complete trip interviews (interview at trip end). The function uses `survey::svyratio()` internally, which correctly accounts for the correlation between catch and effort in variance estimation.

Mean-of-Ratios (MOR): When `estimator = "mor"`, CPUE is estimated as the mean of individual catch/effort ratios. This is the statistically appropriate estimator for incomplete trip interviews (interview during trip). MOR automatically filters to incomplete trips only and requires the `trip_status` field. The function uses `survey::svymean()` on individual ratios.

Trip Truncation: Very short incomplete trips can produce extreme catch/effort ratios that dominate variance estimation. Following Hoenig et al. (1997), the default `truncate_at = 0.5` hours (30 minutes) excludes trips shorter than this threshold before MOR estimation. The survey design is rebuilt with the truncated sample for correct variance computation. Set `truncate_at = NULL` to disable truncation (research mode only). Truncation only applies to MOR estimator; ratio-of-means ignores this parameter.

The function performs sample size validation before estimation: errors if $n < 10$ (ungrouped or any group), warns if $10 \leq n < 30$. For MOR, validation uses the post-truncation sample size. This follows best practices for ratio estimation stability.

When grouped estimation is used (by is not NULL), `survey::svyby()` correctly accounts for domain estimation variance.

Variance estimation methods:

- "taylor" (default): Taylor linearization, computationally efficient and appropriate for smooth statistics like ratios.
- "bootstrap": Bootstrap resampling with 500 replicates. Appropriate for verifying Taylor assumptions.
- "jackknife": Jackknife resampling (automatic JK_n or JK₁ selection based on design). Alternative resampling method.

Value

A `creel_estimates` S3 object (list) with components: `estimates` (tibble with `estimate`, `se`, `ci_lower`, `ci_upper`, `n` columns, plus grouping columns if `by` is specified), `method` (character: names the estimator and the shape of the result. The base names are "ratio-of-means-cpue", "mean-of-ratios-cpue", "mean-of-ratios-truncated-cpue" and "regression-cpue", each gaining a "-sections" suffix on a sectioned design. The "-species" suffix, and the "-per-angler" suffix when normalized, mark a species-level and an angler-normalized result respectively; "regression-cpue-species" is returned for a species request under `estimator = "regression"`), `variance_method` (character: the variance that actually ran, which is the variance argument for every estimator except "regression" – the regression slope carries a leave-one-out jackknife SE and reports "jackknife" whatever variance was set to), `design` (reference to source `creel_design`), `conf_level` (numeric), and `by_vars` (character vector of grouping variable names or NULL). The estimator component records the estimator as you asked for it, "mortr" included, which method cannot: it reports mandatory truncation and the default threshold with the same string.

Package Options

Complete Trip Percentage Threshold: The package option `tidycreel.min_complete_pct` controls the threshold for complete trip percentage warnings (default: 0.10 = 10\ percentage of complete

trips falls below this threshold, a warning is issued referencing Pollock et al. roving-access design best practices. Users can set a custom threshold for their session:

```
options(tidycree1.min_complete_pct = 0.05)
```

The default 10\ scientifically valid estimation. Lowering the threshold is appropriate only for special cases with documented justification. Warnings help ensure data quality and guide users toward diagnostic validation when complete trip samples are insufficient.

Note

When called on a sectioned design, no `.lake_total` row is produced. Catch rates (fish per angler-hour) are not additive across sections. Lake-wide catch rate requires a separate unsectioned call on the full design. See `estimate_total_catch()` for lake-wide total catch estimation.

`estimator = "regression"` on a sectioned design fits one regression per section, on that section's interviews alone. The leave-one-out jackknife standard error therefore rests on the interviews in that section rather than on the whole sample, so section-level regression standard errors are based on fewer points than the unsectioned form and are correspondingly less stable. This is a property of sectioning rather than of the estimator; a section with fewer than three interviews cannot be fitted at all. `species_in` by fits one regression per species, on that species' catch against the same angler effort, with zero-catch interviews retained by default.

See Also

Other "Estimation": `compare_cpue_estimators()`, `est_age_distribution()`, `est_biomass()`, `est_compliance()`, `est_effort_camera_mi()`, `est_length_distribution()`, `est_mean_age()`, `est_mean_length()`, `estimate_effort()`, `estimate_effort_aerial_glmm()`, `estimate_harvest_rate()`, `estimate_release_rate()`, `estimate_total_catch()`, `estimate_total_harvest()`, `estimate_total_release()`

Examples

```
# Basic ungrouped CPUE
calendar <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend")
)
design <- creel_design(calendar, date = date, strata = day_type)

interviews <- data.frame(
  date = as.Date(rep(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04"), each = 10)),
  catch_total = rpois(40, lambda = 3),
  hours_fished = runif(40, min = 1, max = 6),
  trip_status = rep(c("complete", "incomplete"), each = 20),
  trip_duration = runif(40, min = 1, max = 6)
)

design_with_interviews <- add_interviews(design, interviews,
  catch = catch_total,
  effort = hours_fished,
  trip_status = trip_status,
  trip_duration = trip_duration
)
```

```

result <- estimate_catch_rate(design_with_interviews)
print(result)

# Grouped by day_type
result_grouped <- estimate_catch_rate(design_with_interviews, by = day_type)
print(result_grouped)

# Custom confidence level
result_90 <- estimate_catch_rate(design_with_interviews, conf_level = 0.90)

# Bootstrap variance estimation
result_boot <- estimate_catch_rate(design_with_interviews, variance = "bootstrap")

# Mean-of-ratios for incomplete trips
result_mor <- estimate_catch_rate(design_with_interviews, estimator = "mor")

# Mean-of-ratios with custom truncation threshold
result_mor_1h <- estimate_catch_rate(design_with_interviews, estimator = "mor", truncate_at = 1.0)

```

estimate_effort

Estimate total effort from a creel survey design

Description

Computes total effort estimates with standard errors and confidence intervals from a creel survey design with attached count data. Wraps `survey::svytotal()` (ungrouped) or `survey::svyby()` (grouped) with Tier 2 validation and domain-specific output formatting.

Usage

```

estimate_effort(
  design,
  by = NULL,
  variance = "taylor",
  conf_level = 0.95,
  target = c("sampled_days", "stratum_total", "period_total"),
  verbose = FALSE,
  aggregate_sections = TRUE,
  method = "correlated",
  missing_sections = "warn"
)

```

Arguments

design	A <code>creel_design</code> object with counts attached via add_counts . The design must have a survey object constructed.
--------	--

by	Optional tidy selector for grouping variables. Accepts bare column names (e.g., <code>by = day_type</code>), multiple columns (e.g., <code>by = c(day_type, location)</code>), or tidys-elect helpers (e.g., <code>by = starts_with("day")</code>). When NULL (default), computes a single total estimate across all observations.
variance	Character string specifying variance estimation method. Options: "taylor" (default, Taylor linearization), "bootstrap" (bootstrap resampling with 500 replicates), or "jackknife" (jackknife resampling, automatic JKn/JK1 selection).
conf_level	Numeric confidence level for confidence intervals (default: 0.95 for 95% confidence intervals). Must be between 0 and 1.
target	Character string specifying the temporal effort target. Options: "sampled_days" (default, current behavior: total across sampled PSU rows only), "stratum_total" (expand sampled-day means within calendar strata before combining), or "period_total" (full calendar-period total after stratum expansion). For standard stratified count designs, "stratum_total" and "period_total" use the same weighted expansion engine; the distinction is semantic and is recorded on the returned object as <code>effort_target</code> . Expanded targets are currently limited to the standard count-design path and are not yet supported for bus-route, ice, aerial, or sectioned designs.
verbose	Logical. If TRUE, prints an informational message identifying which estimator path was used. Default FALSE for transparent dispatch.
aggregate_sections	Logical. If TRUE (default), a <code>.lake_total</code> row is appended aggregating across all sections. Ignored for non-sectioned designs.
method	Character string specifying how the lake-wide total SE is computed when <code>aggregate_sections = TRUE</code> . "correlated" (default) uses <code>svyby(covmat=TRUE) + svycontrast()</code> for covariance-aware aggregation (recommended for shared-calendar designs). "independent" uses Cochran 5.2 $\sqrt{\sum(SE_h^2)}$ as a documented approximation for genuinely independent section designs. Ignored for non-sectioned designs.
missing_sections	Character string controlling behavior when a registered section has no count observations. "warn" (default) emits a <code>cli_warn()</code> and inserts an NA row with <code>data_available = FALSE</code> . "error" aborts with <code>cli_abort()</code> . Ignored for non-sectioned designs.

Details

The function performs Tier 2 validation before estimation, issuing warnings (not errors) for: zero values in count variables, negative values in count variables, and sparse strata (< 3 observations). When grouped estimation is used (`by` is not NULL), additional warnings are issued for sparse groups (< 3 observations per group level).

Grouped estimation uses `survey::svyby()` internally, which correctly accounts for domain estimation variance. This is different from naive subsetting, which would underestimate variance.

Variance estimation methods:

- "taylor" (default): Taylor linearization, computationally efficient and appropriate for most smooth statistics. This is the recommended default.
- "bootstrap": Bootstrap resampling with 500 replicates. Appropriate for non-smooth statistics or verifying Taylor assumptions. More computationally intensive than Taylor.
- "jackknife": Jackknife resampling (automatic JKn or JK1 selection based on design). Alternative resampling method, deterministic unlike bootstrap.

Value

A `creel_estimates` S3 object (list) with components: `estimates` (tibble with `estimate`, `se`, `se_between`, `se_within`, `ci_lower`, `ci_upper`, `n` columns, plus grouping columns if `by` is specified), `method` (character: "total"), `variance_method` (character: reflects the variance parameter value used), `design` (reference to source `creel_design`), `conf_level` (numeric), and `by_vars` (character vector of grouping variable names or NULL). `se_between` is the between-day standard error from `survey::svytotal()` (equals `se` when a single count is recorded per PSU). `se_within` is the within-day standard error from the Rasmussen two-stage formula; it is zero when a single count is recorded per PSU and nonzero when `count_time_col` is supplied to `add_counts()`. For bus-route designs, a "site_contributions" attribute is also present containing per-site `e_i`, `pi_i`, and `e_i_over_pi_i` columns.

For sectioned designs the tibble carries one row per registered section plus, when `aggregate_sections = TRUE`, a `.lake_total` row, and gains `section`, `prop_of_lake_total`, `se_prop_of_lake_total` and `data_available` columns. `prop_of_lake_total` is the section's share of the lake-wide total and `se_prop_of_lake_total` its standard error; both come from one `survey::svyratio()` call, which accounts for the correlation between a domain total and the overall total containing it. On the `.lake_total` row the share is exactly 1 with a standard error of 0, which is structural rather than an unpropagated component: that row's share of itself was never estimated. A section registered by `add_sections` but absent from the counts reports NA for both, alongside `data_available = FALSE`.

Camera designs are refused

This function estimates instantaneous, bus-route, ice, aerial and sectioned designs. It refuses a camera design, with condition class `creel_error_camera_generic_estimator`.

A camera count is a daily total of arrivals, not an instantaneous count of anglers present, so summing it over days gives arrivals rather than effort. Because camera had no branch in the dispatch below, such a design used to fall through to the instantaneous path and return that sum – a plausible number with a plausible standard error, and no indication that it was not effort.

Use `est_effort_camera`, which calibrates the counts against interview effort and propagates the calibration's uncertainty, or `est_effort_camera_mi` to pool over multiply imputed counts. The same refusal is raised by `estimate_total_catch`, `estimate_total_harvest` and `estimate_total_release`, which build their own effort by this route.

See Also

Other "Estimation": `compare_cpue_estimators()`, `est_age_distribution()`, `est_biomass()`, `est_compliance()`, `est_effort_camera_mi()`, `est_length_distribution()`, `est_mean_age()`, `est_mean_length()`, `estimate_catch_rate()`, `estimate_effort_aerial_glm()`, `estimate_harvest_rate()`, `estimate_release_rate()`, `estimate_total_catch()`, `estimate_total_harvest()`, `estimate_total_release()`

Examples

```
# Basic ungrouped usage
calendar <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend")
)
design <- creel_design(calendar, date = date, strata = day_type)

counts <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend"),
  effort_hours = c(15, 23, 45, 52)
)

design_with_counts <- add_counts(design, counts)
result <- estimate_effort(design_with_counts)
print(result)

# Grouped by day_type
result_grouped <- estimate_effort(design_with_counts, by = day_type)
print(result_grouped)

# Several grouping variables can be combined in `by` when the data carry them

# Custom confidence level
result_90 <- estimate_effort(design_with_counts, conf_level = 0.90)

# Bootstrap variance estimation
result_boot <- estimate_effort(design_with_counts, variance = "bootstrap")
print(result_boot)

# Jackknife variance estimation
result_jk <- estimate_effort(design_with_counts, variance = "jackknife")
print(result_jk)

# Grouped estimation with bootstrap variance
result_grouped_boot <- estimate_effort(design_with_counts, by = day_type, variance = "bootstrap")

# Verbose dispatch message (shows which estimator was used for bus-route designs)
result_verbose <- estimate_effort(design_with_counts, verbose = TRUE)
```

estimate_effort_aerial_glmm

GLMM-based aerial effort estimation with diurnal correction

Description

Estimates total angler effort from aerial creel surveys using a generalized linear mixed model (GLMM), following the approach of Askey et al. (2018). When flights occur at non-random times

of day, simple scaling of instantaneous counts can over- or under-estimate daily effort. This function fits a negative-binomial GLMM (or user-specified family) to model how angler counts change through the day, then integrates the fitted diurnal curve over the fishing day to obtain a bias-corrected effort estimate.

The default model is the quadratic temporal model from Askey (2018): $\text{count} \sim \text{poly}(\text{time_col}, 2) + (1 \mid \text{date})$, fitted via `lme4::glmer.nb()`. Variance is propagated via the delta method (default) or parametric bootstrap (`lme4::bootMer()`).

Usage

```
estimate_effort_aerial_glmm(
  design,
  time_col,
  formula = NULL,
  family = NULL,
  boot = FALSE,
  nboot = 500L,
  conf_level = 0.95,
  target = c("sampled_days", "mean_day")
)
```

Arguments

design	A <code>creel_design()</code> object with <code>design_type == "aerial"</code> and counts attached via <code>add_counts()</code> . The counts data must contain the time-of-flight column specified by <code>time_col</code> .
time_col	Unquoted name of the numeric column in <code>design\$counts</code> recording the hour of each aerial overflight (e.g., <code>time_of_flight</code>).
formula	Optional. A formula for the GLMM, passed directly to <code>lme4::glmer.nb()</code> or <code>lme4::glmer()</code> . If NULL (default), the Askey (2018) quadratic formula is used: $\text{count} \sim \text{poly}(\text{time_col}, 2) + (1 \mid \text{date})$.
family	Optional. A family object or character string specifying the GLM family. If NULL or "negbin" (default), <code>lme4::glmer.nb()</code> is used. Otherwise, <code>lme4::glmer()</code> is called with the specified family.
boot	Logical. If TRUE, use <code>lme4::bootMer()</code> for parametric bootstrap confidence intervals instead of the delta method. Default FALSE.
nboot	Integer. Number of bootstrap replicates when <code>boot = TRUE</code> . Default 500L.
conf_level	Numeric confidence level for the CI. Default 0.95.
target	Character string giving the temporal basis of the returned estimate. "sampled_days" (default) expands the fitted day to every day the design sampled, matching what <code>estimate_effort()</code> returns for the same design so the two are comparable. "mean_day" reports a single average day, the basis this function reported before tidycrrel 8.0.0. It is not identical to the old value: it now carries the retransformation factor the old code omitted, so it is higher by $\exp(\sigma^2 / 2)$. Both are expectations, so both carry the retransformation factor described under Details. Neither expands beyond the sampled days: expanded targets are not supported for aerial designs by <code>estimate_effort()</code> either.

Details

[Experimental]

The fitted curve is a fixed-effects prediction: the day whose random intercept is zero. On a log link that is the *median* day rather than the mean one, so summing it across days would understate the total. Both targets therefore carry a factor of $\exp(\sigma^2 / 2)$, where σ^2 is the day-level intercept variance — 4% on the package's own fixture, and larger where days vary more.

That factor treats σ^2 as known. The reported standard error scales with the expansion but does not carry the uncertainty in the variance component itself, so it is mildly optimistic; quantifying that would need a variance method neither the delta nor the bootstrap path offers today.

Value

A `creel_estimates` object with:

- `estimate`: total angler effort integrated over the fishing day
- `se`: standard error (delta method or bootstrap SD)
- `se_between`: same as `se` (fixed-effect SE component)
- `se_within`: always `NA_real_` — no Rasmussen within-day decomposition is performed for GLMM estimates
- `ci_lower`, `ci_upper`: confidence interval bounds, and `NA_real_` whenever `se` is, on both the delta and bootstrap paths. If the visibility correction or the angler-to-people ratio was declared unknown, the total's uncertainty was never fully propagated, so no unconditional interval exists to report. Reporting the remaining spread would be an interval conditional on the unknown multiplier being exact – indistinguishable from declaring it known with zero uncertainty, which is precisely the confusion `NA` exists to prevent.
- `n`: number of count observations used to fit the model
- `method`: "aerial_glmm_total"

References

Askey, P.J., Ward, H., Godin, T., Boucher, M., and Northrup, S. (2018). Angler effort estimates from instantaneous aerial counts: use of high-frequency time-lapse camera data to inform model-based estimators. *North American Journal of Fisheries Management*, 38, 194-209. doi:[10.1002/nafm.10010](https://doi.org/10.1002/nafm.10010)

See Also

Other "Estimation": `compare_cpue_estimators()`, `est_age_distribution()`, `est_biomass()`, `est_compliance()`, `est_effort_camera_mi()`, `est_length_distribution()`, `est_mean_age()`, `est_mean_length()`, `estimate_catch_rate()`, `estimate_effort()`, `estimate_harvest_rate()`, `estimate_release_rate()`, `estimate_total_catch()`, `estimate_total_harvest()`, `estimate_total_release()`

Examples

```
data(example_aerial_glmm_counts)

aerial_cal <- unique(example_aerial_glmm_counts[, c("date", "day_type")])
```

```

aerial_cal <- aerial_cal[order(aerial_cal$date), ]
design <- creel_design(
  aerial_cal,
  date = date,
  strata = day_type,
  survey_type = "aerial",
  visibility_correction = "none",
  angler_ratio = 1,
  angler_ratio_se = 0,
  h_open = 14
)
design <- add_counts(design, example_aerial_glmm_counts, count_col = n_angers)

# Default Askey quadratic model with delta-method SE
result <- estimate_effort_aerial_glmm(design, time_col = time_of_flight)
print(result)

# Bootstrap CIs. `nboot` is held low here so the example stays fast on a
# check machine; use at least 1000 replicates for real inference. The block
# is wrapped in \donttest{} for runtime alone -- it needs no resource the
# example cannot reach.

result_boot <- estimate_effort_aerial_glmm(
  design,
  time_col = time_of_flight,
  boot = TRUE,
  nboot = 25L
)
print(result_boot)

```

estimate_effort_per_acre

Compute effort density as effort per acre

Description

Divides all effort estimate columns in a pre-computed `creel_estimates` object by a surface area scalar (acres). Standard error propagates linearly because acres is a constant (not a random variable), so no Delta Method is needed: $se_per_acre = se_effort / acres$.

acres is a constant divisor, so the result is whatever the effort was, per acre: the returned unit field composes the effort's own unit ("angler-hours/acre", "party-hours/acre"), and stays NA when the effort's unit was unknown.

This is a composable estimator: the effort object must be pre-computed via [estimate_effort](#) before calling this function.

Usage

```
estimate_effort_per_acre(effort, acres, ...)
```

Arguments

effort	A creel_estimates object returned by estimate_effort . Must contain estimate, se, ci_lower, and ci_upper columns in effort\$estimates.
acres	A single positive numeric scalar giving the total lake surface area in acres. All effort estimate columns are divided by this value.
...	Reserved for future arguments.

Value

A creel_estimates object with method = "effort-per-acre". The estimates tibble has the same rows as the input but with estimate, se, ci_lower, ci_upper (and se_between, se_within when present in the input) all divided by acres. Grouping columns (by_vars) and n are carried through unchanged. variance_method and conf_level are inherited from the input effort object.

See Also

[estimate_effort](#), [estimate_angler_trips](#)

Examples

```
data(example_calendar)
data(example_counts)
data(example_interviews)
design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
effort <- estimate_effort(design)
# Surface area of the water body, in acres.
estimate_effort_per_acre(effort, acres = 120)
```

estimate_exploitation_rate

Estimate exploitation rate using the Pollock et al. moment estimator

Description

Computes the seasonal exploitation rate from a combination of a tagging study and creel-survey harvest data using the moment estimator described in Pollock et al. (1994) and Jones & Pollock (2012, Ch. 19).

When strata is supplied the function takes the stratified path and returns per-stratum estimates plus a T-weighted aggregate (.overall row).

Usage

```
estimate_exploitation_rate(
  T = NULL,
  C = NULL,
  se_C = NULL,
  n = NULL,
  m = NULL,
  conf_level = 0.95,
  reporting_rate = 1,
  reporting_rate_se = NULL,
  strata = NULL,
  by = NULL,
  aggregate = TRUE,
  ci_type = c("symmetric", "logit")
)
```

Arguments

- | | |
|----------------|---|
| T | integer or numeric. Number of tagged fish released at season start. Ignored when strata is supplied. |
| C | <p>Total creel harvest, given either as the estimate_total_harvest result itself or as a bare numeric.</p> <p>Prefer passing the object. $\hat{u}$ measures the fraction of the tagged cohort removed <i>over the whole season</i>, so C must be a period total while T is the full cohort – but estimate_total_harvest defaults to target = "sampled_days". Passing the object lets that be checked; passing a bare number strips the estimand off, and a sampled-day total understates $\hat{u}$ by the sampling fraction while staying inside $[0, 1]$ with a proportionally scaled SE (GH #206).</p> <p>It must also be harvest, not catch: fish caught and released were not removed from the tagged cohort, so a catch total from estimate_total_catch inflates $\hat{u}$. Both totals are counts of fish, so no <i>unit</i> check separates them – passing the object is what makes the substitution detectable.</p> <p>When C is a creel_estimates object its standard error is read from it, and supplying se_C as well is an error. Ignored when strata is supplied.</p> |
| se_C | numeric. Standard error of the harvest estimate. Required when C is a bare number; read from the object instead when C is a creel_estimates, in which case supplying it here is an error. Ignored when strata is supplied. |
| n | integer. Total number of fish inspected in the creel survey (sampled from harvest). Ignored when strata is supplied. |
| m | integer. Number of tagged fish found among the n inspected fish. Ignored when strata is supplied. |
| conf_level | numeric confidence level for the CI. Default 0.95. |
| reporting_rate | <p>numeric in (0, 1]. Tag reporting rate; when less than 1 the function adjusts $\hat{u}$ upward: $\hat{u}_{adj} = \hat{u}/\lambda$. Default 1.0 (full reporting). See Details.</p> <p>Upward, because under-reporting means the recoveries actually observed <i>understate</i> how many tagged fish were removed. Dividing by $\lambda < 1$ restores the</p> |

removals the survey never heard about, so $\hat{u}$ rises: at $\lambda = 0.5$ it doubles. (This paragraph read "downward" until GH #207, contradicting the formula printed beside it.)

Unlike the aerial corrections, this default is left in place: it is a visible, documented default on an exported argument that the caller opts into adjusting, not a value substituted inside an estimator where the caller could not see it. `reporting_rate = 1` states that every tag was reported.

<code>reporting_rate_se</code>	numeric scalar or NULL. The standard error of <code>reporting_rate</code> , on the same $(0, 1]$ scale. In practice λ is estimated rather than known, and supplying its SE adds the third delta term derived in Details. When NULL (default), the component is absent rather than zero — a zero would be indistinguishable from having propagated the reporting rate's error and found none. Note that with the default <code>reporting_rate = 1</code> there is nothing to propagate, which is why an unadjusted analysis is unaffected by this argument.
<code>strata</code>	<code>data.frame</code> or NULL. When non-NULL, the function takes the stratified path. Required columns: <code>stratum</code> , <code>T_h</code> , <code>C_h</code> , <code>se_C_h</code> , <code>n_h</code> , <code>m_h</code> (or whatever column name is given in <code>by</code> for the stratum labels). <code>C_h</code> is per-stratum harvest and carries the same requirement as <code>C</code> .
<code>by</code>	<code>character(1)</code> . Name of the stratum label column in <code>strata</code> . Default <code>"stratum"</code> . Ignored when <code>strata = NULL</code> .
<code>aggregate</code>	logical. If TRUE (the default) an <code>.overall</code> row containing the T-weighted aggregate exploitation rate is appended to the per-stratum estimates. Set to FALSE to suppress it. Ignored when <code>strata = NULL</code> .
<code>ci_type</code>	character. Shape of the confidence interval. <code>"symmetric"</code> (default) gives the standard $\hat{u} \pm z \cdot SE$ interval clamped to $[0, 1]$. <code>"logit"</code> applies a logit-transform so the CI respects the $[0, 1]$ constraint without clamping: $\text{expit}(\text{logit}(\hat{u}) \pm z \cdot SE / (\hat{u}(1 - \hat{u})))$.

Details

Unstratified Formula:

Let:

- $p = m/n$ — proportion of tagged fish among inspected fish,
- $r = C/T$ — ratio of estimated harvest to tagged fish released.

The point estimate is:

$$\hat{u} = r \cdot p = \frac{C \cdot m}{T \cdot n}$$

Delta-method variance:

$$\widehat{\text{Var}}(\hat{u}) \approx r^2 \cdot \frac{p(1-p)}{n} + p^2 \cdot \frac{s_C^2}{T^2}$$

where s_C is `se_C`, the standard error of the creel harvest estimate.

Stratified Formula:

For stratum h , the per-stratum exploitation rate is:

$$\hat{u}_h = \frac{C_h \cdot m_h}{T_h \cdot n_h}$$

with delta-method variance:

$$\widehat{\text{Var}}(\hat{u}_h) \approx r_h^2 \cdot \frac{p_h(1-p_h)}{n_h} + p_h^2 \cdot \frac{s_{C_h}^2}{T_h^2}$$

The T-weighted aggregate over H strata is:

$$\hat{u} = \frac{\sum_h T_h \hat{u}_h}{\sum_h T_h}$$

with variance:

$$\widehat{\text{Var}}(\hat{u}) = \frac{\sum_h T_h^2 \widehat{\text{Var}}(\hat{u}_h)}{(\sum_h T_h)^2}$$

This is the estimator from Jones CM & Pollock KH (2012, Ch. 19) and Pollock KH, Jones CM & Brown TL (1994).

Reporting Rate Adjustment:

When `reporting_rate` = $\lambda < 1$, the adjusted estimate is $\hat{u}_{adj} = \hat{u}/\lambda$ and the variance is divided by λ^2 .

λ is treated as known without error unless `reporting_rate_se` is supplied. When it is, a third delta term is added, since $\partial \hat{u} / \partial \lambda = -\hat{u} / \lambda$:

$$\widehat{\text{Var}}(\hat{u}) = \frac{r^2 \widehat{\text{Var}}(p) + p^2 \widehat{\text{Var}}(r)}{\lambda^2} + \frac{\hat{u}^2}{\lambda^2} \widehat{\text{Var}}(\lambda)$$

That term is added **once**, at the total. λ is a single estimate dividing every stratum, so it is perfectly correlated across them; adding it per stratum and summing in quadrature would treat a shared divisor as independent and understate it. On the stratified path it therefore enters on the aggregate, not inside the per-stratum variances.

Natural mortality between tagging and the creel survey is not corrected for.

Bounds:

Exploitation rate must lie in $[0, 1]$. If the point estimate or CI endpoints fall outside this range, a warning is issued and CI bounds are clamped to $[0, 1]$.

Value

A `creel_estimates` S3 object with `method = "exploitation-rate"` and an `estimates` tibble. For the unstratified path columns are: `estimate`, `se`, `ci_lower`, `ci_upper`, `n`, `T`, `C`, `m`. For the stratified path the stratum label column comes first, followed by the same columns.

References

- Pollock KH, Jones CM & Brown TL (1994). *Angler Survey Methods and Their Applications in Fisheries Management*. AFS Special Publication 25. American Fisheries Society.
- Jones CM & Pollock KH (2012). Recreational survey methods: estimating effort, harvest, and abundance. In Zale AV et al. (eds), *Fisheries Techniques* (3rd ed., Ch. 19). American Fisheries Society.

See Also

Other Estimation: [estimate_angler_n\(\)](#), [estimate_mr_harvest\(\)](#)

Examples

```
# Unstratified exploitation rate
result <- estimate_exploitation_rate(
  T    = 200L,
  C    = 450.0,
  se_C = 42.0,
  n    = 180L,
  m    = 15L
)
print(result)

# Stratified exploitation rate
strata_df <- data.frame(
  stratum = c("weekday", "weekend"),
  T_h     = c(120L, 80L),
  C_h     = c(280.0, 170.0),
  se_C_h  = c(28.0, 22.0),
  n_h     = c(110L, 70L),
  m_h     = c(9L, 6L)
)
result_strat <- estimate_exploitation_rate(
  strata = strata_df,
  by     = "stratum"
)
print(result_strat)
```

estimate_harvest_rate *Estimate harvest (HPUE: Harvest Per Unit Effort) from a creel survey design*

Description

Computes HPUE estimates with standard errors and confidence intervals from a creel survey design with attached interview data. Uses ratio-of-means estimation via `survey::svyratio()` to properly account for ratio variance. HPUE measures the rate of kept fish (harvest) per unit effort, distinguished from total catch rate (CPUE which includes both kept and released fish).

Usage

```
estimate_harvest_rate(
  design,
  by = NULL,
  variance = "taylor",
  conf_level = 0.95,
  verbose = FALSE,
```

```

    use_trips = NULL,
    estimator = NULL,
    truncate_at = 0.5,
    missing_sections = "warn",
    targeted = TRUE
  )

```

Arguments

design	A creel_design object with interviews attached via add_interviews . The design must have an interview survey object constructed with harvest, catch, and effort columns.
by	Optional tidy selector for grouping variables. Accepts bare column names (e.g., <code>by = day_type</code>), multiple columns (e.g., <code>by = c(day_type, location)</code>), or tidys-elect helpers (e.g., <code>by = starts_with("day")</code>). When NULL (default), computes a single HPUE estimate across all interviews. Two kinds of column are not groupings and are refused: the interview id, as registered by add_catch(), add_lengths() or add_ages(), which holds one value per interview and so leaves no within-group variance to estimate; and columns the package derived rather than the user supplying, such as <code>.angler_effort</code>. A wildcard selector drops the derived columns silently; asking for one specifically is an error. A column of your own is never treated as derived, whatever it is called.
variance	Character string specifying variance estimation method. Options: "taylor" (default, Taylor linearization), "bootstrap" (bootstrap resampling with 500 replicates), or "jackknife" (jackknife resampling, automatic JK _n /JK ₁ selection).
conf_level	Numeric confidence level for confidence intervals (default: 0.95 for 95% confidence intervals). Must be between 0 and 1.
verbose	Logical. If TRUE, prints an informational message identifying which estimator path was used. Default FALSE.
use_trips	Character string specifying which interviews to include. For standard (non-bus-route) designs: "complete" (default) restricts to completed trips only; "all" uses all interviews including incomplete trips. "complete" is the statistically preferred default because incomplete-trip HPUE underestimates harvest when anglers keep additional fish after the interview (Hansen & Van Kirk 2010). Fish already in the livewell are directly observable, so "all" remains available for analyses that prefer the larger interview set. For bus-route designs: "complete" (default), "incomplete", or "diagnostic"; "all" is not an estimator there, because pooling the two kinds of trip applies the complete-trip ratio of Horvitz-Thompson totals to numerators that are catch so far. Matching is exact on both paths, and unrecognised values are an error. When <code>trip_status</code> was not provided to add_interviews , this argument has no effect for standard designs.
estimator	Character string selecting the rate estimator: "ratio-of-means" (a ratio of totals), "mor" (the mean of per-interview ratios), or "mortr" ("mor" with truncation made mandatory). Default NULL means "not specified". When <code>use_trips</code> and <code>estimator</code> are <i>both</i> unspecified and the design was built with <code>add_interviews(interview_type</code>

= "roving"), the pair resolves to all-trip truncated MOR; otherwise it resolves to complete-trip ratio-of-means. Specifying either one suppresses the automatic routing.

Hoenig et al. (1997) recommend the truncated mean of ratios for a roving survey because the clerk intercepts trips mid-stream. That argument is about the interview rather than about which fish are counted, so it applies to this rate exactly as it applies to the catch rate. Bus-route and ice designs return before this resolution and are unaffected.

truncate_at	Numeric minimum trip duration in hours for the mean-of-ratios estimator (default 0.5, i.e. 30 minutes). Trips shorter than this are discarded before the mean of ratios is taken. Hoenig et al. (1997) recommend the 30-minute threshold because the untruncated mean-of-ratios estimator has infinite asymptotic variance: $1/L$ has infinite expectation as trip length approaches zero. The threshold applies to elapsed trip duration, not to angler-hours. Set to NULL to disable; the bus-route path warns when it is disabled there, the standard mean-of-ratios path treats it as a documented opt-out and is silent, matching <code>estimate_catch_rate</code> . Ignored under "ratio-of-means". An interview whose duration is missing cannot be shown to meet the threshold, so it is excluded and reported separately from the trips excluded as too short.
missing_sections	Character string controlling behavior when a registered section has no interview observations. "warn" (default) emits a <code>cli_warn()</code> and inserts an NA row with <code>data_available = FALSE</code> . "error" aborts with <code>cli_abort()</code> . Ignored for non-sectioned designs.
targeted	Logical. When TRUE (default), all trips are used. When FALSE, the interviews that recorded none of the species being estimated are excluded before MOR/MORtr estimation, so the result is the rate among trips that harvested it rather than the fishery-wide rate. A <code>cli_warn()</code> names the species and the percentage excluded, and a separate warning fires when more than 70\ species under targeted = TRUE (possible mis-specification). Requires <code>by = species</code>: without one there is no per-species count to test, and the only available test would be "recorded nothing at all", which is a different estimand. <code>targeted = FALSE</code> without <code>by = species</code> is an error rather than a silent no-op (GH #307). Ignored for the ratio-of-means estimator, as for <code>estimate_catch_rate()</code>. Note that the <code>estimate_total_*</code>() functions deliberately do not accept it — see their documentation for why a targeted rate has no matching total.

Details

HPUE is estimated as the ratio of total harvest (kept fish) to total effort (ratio-of-means estimator). This is the appropriate estimator for average harvest rates when trip lengths (effort) vary. The function uses `survey::svyratio()` internally, which correctly accounts for the correlation between harvest and effort in variance estimation.

HPUE will always be less than or equal to CPUE for the same data, since harvest (kept fish) is a subset of total catch.

The function performs sample size validation before estimation: errors if $n < 10$ (ungrouped or any group), warns if $10 \leq n < 30$. This follows best practices for ratio estimation stability.

When grouped estimation is used (by is not NULL), `survey::svyby()` with `svyratio` correctly accounts for domain estimation variance.

Variance estimation methods:

- "taylor" (default): Taylor linearization, computationally efficient and appropriate for smooth statistics like ratios.
- "bootstrap": Bootstrap resampling with 500 replicates. Appropriate for verifying Taylor assumptions.
- "jackknife": Jackknife resampling (automatic JK_n or JK₁ selection based on design). Alternative resampling method.

Value

A `creel_estimates` S3 object (list) with components: `estimates` (tibble with `estimate`, `se`, `ci_lower`, `ci_upper`, `n` columns, plus grouping columns if `by` is specified), `method` (character: "ratio-of-means-hpue", with "-per-angler" suffix when normalized), `variance_method` (character: reflects the variance parameter value used), `design` (reference to source `creel_design`), `conf_level` (numeric), and `by_vars` (character vector of grouping variable names or NULL). The estimator component records the estimator as you asked for it, "mortr" included, which method cannot: it reports mandatory truncation and the default threshold with the same string. For bus-route designs, a "site_contributions" attribute is also present.

Note

Bus-route designs use a different estimator for each trip type, because each is the estimator that trip type supports. `use_trips = "complete"` returns the ratio of the two Horvitz-Thompson totals (Jones & Pollock 2012, Eq. 19.4 and 19.5). `use_trips = "incomplete"` returns a truncated, design-weighted mean of the individual angler rates (Hoenig et al. 1997), whose expectation is the ratio of total harvest to total effort; the ratio of means is biased for anglers intercepted mid-trip, weighting individual rates by the square of completed trip length. Both report fish per angler-hour, so `use_trips = "diagnostic"` compares like with like.

When called on a sectioned design, no `.lake_total` row is produced. Harvest rates (fish per angler-hour) are not additive across sections. Lake-wide harvest rate requires a separate unsectioned call.

This function defaults to using **completed-trip** interviews only for HPUE estimation (`use_trips = "complete"`). Incomplete-trip HPUE underestimates harvest when anglers continue fishing and keep additional fish after being interviewed (Hansen & Van Kirk 2010), so restricting to completed trips is the statistically preferred default. Fish already in the livewell at interview time are directly observable, so `use_trips = "all"` remains available to include incomplete-trip interviews.

See Also

[estimate_catch_rate](#) for total catch rate estimation

Other "Estimation": [compare_cpue_estimators\(\)](#), [est_age_distribution\(\)](#), [est_biomass\(\)](#), [est_compliance\(\)](#), [est_effort_camera_mi\(\)](#), [est_length_distribution\(\)](#), [est_mean_age\(\)](#), [est_mean_length\(\)](#), [estimate_catch_rate\(\)](#), [estimate_effort\(\)](#), [estimate_effort_aerial_glm\(\)](#), [estimate_release_rate\(\)](#), [estimate_total_catch\(\)](#), [estimate_total_harvest\(\)](#), [estimate_total_release\(\)](#)

Examples

```
# Basic ungrouped HPUE
calendar <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = c("weekday", "weekday", "weekend", "weekend")
)
design <- creel_design(calendar, date = date, strata = day_type)

set.seed(123)
interviews <- data.frame(
  date = as.Date(rep(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04"), each = 10)),
  catch_total = rpois(40, lambda = 3),
  hours_fished = runif(40, min = 1, max = 6),
  trip_status = rep(c("complete", "incomplete"), each = 20),
  trip_duration = runif(40, min = 1, max = 6)
)
# Harvest is subset of catch (kept fish)
interviews$catch_kept <- pmax(0, interviews$catch_total - rbinom(40, size = 2, prob = 0.3))

design_with_interviews <- add_interviews(design, interviews,
  catch = catch_total,
  harvest = catch_kept,
  effort = hours_fished,
  trip_status = trip_status,
  trip_duration = trip_duration
)
result <- estimate_harvest_rate(design_with_interviews)
print(result)

# Grouped by day_type
result_grouped <- estimate_harvest_rate(design_with_interviews, by = day_type)
print(result_grouped)

# Custom confidence level
result_90 <- estimate_harvest_rate(design_with_interviews, conf_level = 0.90)

# Bootstrap variance estimation
result_boot <- estimate_harvest_rate(design_with_interviews, variance = "bootstrap")

# Verbose dispatch message (shows which estimator was used for bus-route designs)
result_verbose <- estimate_harvest_rate(design_with_interviews, verbose = TRUE)
```

estimate_mr_harvest	<i>Estimate total harvest from a mark-recapture population estimate</i>
---------------------	---

Description

Computes a total harvest estimate and its uncertainty using the delta method, given a closed-population angler population estimate from [estimate_angler_n](#) and a known harvest rate.

The point estimate is $\hat{H} = \hat{N} \times r$ where r is the harvest rate in fish per angler. The delta-method standard error is $SE(\hat{H}) = r \times SE(\hat{N})$, propagating only the uncertainty in $\hat{N}$ (harvest-rate uncertainty is not propagated in this release).

Usage

```
estimate_mr_harvest(
  angler_n,
  harvest_rate,
  harvest_rate_se = NULL,
  conf_level = 0.95,
  ci_method = c("logit", "delta", "bootstrap")
)
```

Arguments

- | | |
|-----------------|---|
| angler_n | A creel_estimates object returned by estimate_angler_n . |
| harvest_rate | numeric scalar. Harvest per angler, in fish per angler, over the same period angler_n counts anglers for. Must be > 0 . This is a rate, not a proportion, and is not bounded above: a fishery averaging 1.4 fish per angler is a legal value. In the notation of Hansen & Van Kirk (2018) eq. (1), $H = N \cdot D \cdot V$, this argument is the product $D \times V$ — mean days fished per angler times mean daily harvest per angler — not V alone. Supply harvest_rate_se to propagate its uncertainty. |
| harvest_rate_se | <p>numeric scalar or NULL. The standard error of harvest_rate, on the same fish-per-angler scale. When supplied, the total variance is Goodman's (1960) product form, $\hat{N}^2 \sigma_r^2 + r^2 \sigma_{\hat{N}}^2 - \sigma_{\hat{N}}^2 \sigma_r^2$, via the same helper the three estimate_total_*() functions already use.</p> <p>When NULL (default), the reported SE reflects uncertainty in $\hat{N}$ alone and is a lower bound; the function says so at runtime. The component is absent rather than zero, because a zero would be indistinguishable from having propagated the rate's error and found none. Rasmussen et al. (1998) draw the distinction explicitly: the subtractive product formula is the one for terms "estimated from a sample", and differs from the population formula "used when the terms in the product are known, not estimated".</p> <p>Supplying it also changes the confidence interval. The default logit interval scales the endpoints of the $\hat{N}$ interval by harvest_rate, which is exact only while that rate is a known positive constant. Once the rate is estimated those endpoints are themselves random, so the function falls back to a symmetric interval built from the full product SE.</p> |
| conf_level | numeric. Confidence level for the CI. Default 0.95. |
| ci_method | character(1). CI construction method: "delta" (default) uses the analytic delta-method formula; "bootstrap" propagates the bootstrap samples stored in attr(angler_n, "boot_samples") (produced by calling estimate_angler_n(..., ci_method = "bootstrap") first). |

Details

The harvest rate is treated as a known constant. This is a simplification made by this implementation, not by the cited method: Hansen & Van Kirk (2018) estimate both factors of the rate, give each a log-normal sampling distribution, and resample them alongside $\hat{N}$ in the bootstrap that produces their harvest CIs. Holding the rate fixed therefore makes the reported se a lower bound on the true uncertainty. Propagation of harvest-rate uncertainty via a two-source delta method is a planned future extension.

The delta-method interval is also symmetric, which for a mark-recapture estimate is optimistic at the lower end and can place ci_lower below zero when recaptures are few; see the same note under [estimate_angler_n](#).

Value

A creel_estimates S3 object with method = "mark-recapture-harvest" and an estimates tibble with columns: parameter, estimate, se, ci_lower, ci_upper.

References

Hansen, J. M., & Van Kirk, R. W. (2018). A mark-recapture-based approach for estimating angler harvest. *North American Journal of Fisheries Management*, 38(2), 400–410. doi:10.1002/nafm.10038

See Also

Other Estimation: [estimate_angler_n\(\)](#), [estimate_exploitation_rate\(\)](#)

Examples

```
# Step 1: estimate angler population
result <- estimate_angler_n(M = 200L, n = 50L, m = 10L)

# Step 2: compute total harvest
harvest <- estimate_mr_harvest(angler_n = result, harvest_rate = 0.35)
print(harvest)
```

estimate_release_rate	<i>Estimate release rate (RPUE: Released fish Per Unit Effort) from a creel survey design</i>
-----------------------	---

Description

Computes release rate estimates with standard errors and confidence intervals from a creel survey design with attached interview and catch data. Uses ratio-of-means estimation via survey::svyratio(). RPUE measures the rate of released fish per unit effort, analogous to HPUE for harvested fish.

Usage

```
estimate_release_rate(
  design,
  by = NULL,
  variance = "taylor",
  conf_level = 0.95,
  use_trips = NULL,
  estimator = NULL,
  truncate_at = 0.5,
  missing_sections = "warn",
  targeted = TRUE
)
```

Arguments

design	A creel_design object with interviews (via add_interviews) and catch data (via add_catch) attached. The catch data must include records with catch_type = "released".
by	Optional tidy selector for grouping variables. Accepts bare column names (e.g., by = day_type, by = species), multiple columns, or tidysselect helpers. When species grouping is used, per-species release rates are estimated. Two kinds of column are not groupings and are refused: the interview id, as registered by add_catch() , add_lengths() or add_ages() , which holds one value per interview and so leaves no within-group variance to estimate; and columns the package derived rather than the user supplying, such as .angler_effort. A wildcard selector drops the derived columns silently; asking for one specifically is an error. A column of your own is never treated as derived, whatever it is called.
variance	Character string specifying variance estimation method. Options: "taylor" (default), "bootstrap", or "jackknife".
conf_level	Numeric confidence level (default: 0.95).
use_trips	Character string specifying which interviews to include. "complete" (default) restricts to completed trips only; "all" uses all interviews including incomplete trips. "complete" is the statistically preferred default because incomplete-trip RPUE underestimates releases when anglers release additional fish after the interview (Hansen & Van Kirk 2010). "all" remains available for analyses that prefer the larger interview set. When trip_status was not provided to add_interviews , this argument has no effect. For bus-route designs: "complete" (default), "incomplete", or "diagnostic", matching estimate_harvest_rate ; "all" is not an estimator there, and unrecognised values are an error rather than a silent fall-through to the complete-trip path.
estimator	Character string selecting the rate estimator: "ratio-of-means" (a ratio of totals), "mor" (the mean of per-interview ratios), or "mortr" ("mor" with truncation made mandatory). Default NULL means "not specified". When use_trips and estimator are <i>both</i> unspecified and the design was built with <code>add_interviews(interview_type = "roving")</code> , the pair resolves to all-trip truncated MOR; otherwise it resolves

to complete-trip ratio-of-means. Specifying either one suppresses the automatic routing.

Hoenig et al. (1997) recommend the truncated mean of ratios for a roving survey because the clerk intercepts trips mid-stream. That argument is about the interview rather than about which fish are counted, so it applies to this rate exactly as it applies to the catch rate. Bus-route and ice designs return before this resolution and are unaffected.

truncate_at	Numeric minimum trip duration in hours for the mean-of-ratios estimator (default 0.5, i.e. 30 minutes). Trips shorter than this are discarded before the mean of ratios is taken. Hoenig et al. (1997) recommend the 30-minute threshold because the untruncated mean-of-ratios estimator has infinite asymptotic variance: $1/L$ has infinite expectation as trip length approaches zero. The threshold applies to elapsed trip duration, not to angler-hours. Set to NULL to disable; the bus-route path warns when it is disabled there, the standard mean-of-ratios path treats it as a documented opt-out and is silent, matching <code>estimate_catch_rate</code> . Ignored under "ratio-of-means". An interview whose duration is missing cannot be shown to meet the threshold, so it is excluded and reported separately from the trips excluded as too short.
missing_sections	Character string controlling behavior when a registered section has no interview observations. "warn" (default) emits a <code>cli_warn()</code> and inserts an NA row with <code>data_available = FALSE</code> . "error" aborts with <code>cli_abort()</code> . Ignored for non-sectioned designs.
targeted	Logical. When TRUE (default), all trips are used. When FALSE, the interviews that recorded none of the species being estimated are excluded before MOR/MORtr estimation, so the result is the rate among trips that released it rather than the fishery-wide rate. A <code>cli_warn()</code> names the species and the percentage excluded, and a separate warning fires when more than 70\ species under targeted = TRUE (possible mis-specification). Requires <code>by = species</code>: without one there is no per-species count to test, and the only available test would be "recorded nothing at all", which is a different estimand. <code>targeted = FALSE</code> without <code>by = species</code> is an error rather than a silent no-op (GH #307). Ignored for the ratio-of-means estimator, as for <code>estimate_catch_rate()</code>. Note that the <code>estimate_total_*</code>() functions deliberately do not accept it — see their documentation for why a targeted rate has no matching total.

Details

RPUE is estimated as the ratio of total released fish to total effort (ratio-of-means). Release data comes from `add_catch()` records with `catch_type = "released"`. Interviews with no releases contribute 0 to the numerator (zero-fill), ensuring the effort denominator is correct.

Value

A `creel_estimates` S3 object with `method = "ratio-of-means-rpue"`. Estimates tibble has columns: `estimate`, `se`, `ci_lower`, `ci_upper`, `n` (plus any grouping columns). The estimator component records

the estimator as you asked for it, "mortr" included, which method cannot: it reports mandatory truncation and the default threshold with the same string.

Note

Bus-route designs use a different estimator for each trip type, matching `estimate_harvest_rate`. `use_trips = "complete"` returns the ratio of the two Horvitz-Thompson totals (Jones & Pollock 2012, Eq. 19.5 / Eq. 19.4) and reports `method = "ratio-of-means-rpue"`; `use_trips = "incomplete"` returns the truncated, Hajek-weighted mean of per-angler rates (Hoenig et al. 1997) and reports `method = "mean-of-ratios-rpue"`. Both are releases per angler-hour.

When called on a sectioned design, no `.lake_total` row is produced. Release rates (fish per angler-hour) are not additive across sections. Lake-wide release rate requires a separate unsectioned call.

This function defaults to using **completed-trip** interviews only for RPUE estimation (`use_trips = "complete"`). Incomplete-trip RPUE may underestimate releases if anglers release additional fish after the interview (Hansen & Van Kirk 2010), so restricting to completed trips is the statistically preferred default. Released fish counted at interview time are directly observable, so `use_trips = "all"` remains available to include incomplete-trip interviews.

See Also

`estimate_harvest_rate` for harvest rate, `add_catch`

Other "Estimation": `compare_cpue_estimators()`, `est_age_distribution()`, `est_biomass()`, `est_compliance()`, `est_effort_camera_mi()`, `est_length_distribution()`, `est_mean_age()`, `est_mean_length()`, `estimate_catch_rate()`, `estimate_effort()`, `estimate_effort_aerial_glm()`, `estimate_harvest_rate()`, `estimate_total_catch()`, `estimate_total_harvest()`, `estimate_total_release()`

Examples

```
library(tidycrrel)
data(example_calendar)
data(example_counts)
data(example_interviews)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished,
  trip_status = trip_status, trip_duration = trip_duration
)
design <- add_catch(design, example_catch,
  catch_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  count = count,
  catch_type = catch_type
)

# Overall release rate (all species combined)
rpue <- estimate_release_rate(design)
```

```

print(rpue)

# Per-species release rates
rpue_by_species <- estimate_release_rate(design, by = species)
print(rpue_by_species)

```

estimate_total_catch	<i>Estimate total catch by combining effort and CPUE</i>
----------------------	--

Description

Computes total catch estimates by multiplying effort $\times$ CPUE with variance propagation via the delta method. Requires a creel design with both count data (for effort estimation) and interview data (for CPUE estimation).

Usage

```

estimate_total_catch(
  design,
  by = NULL,
  variance = "taylor",
  conf_level = 0.95,
  target = c("sampled_days", "stratum_total", "period_total"),
  use_trips = NULL,
  estimator = NULL,
  truncate_at = 0.5,
  aggregate_sections = TRUE,
  missing_sections = "warn",
  verbose = FALSE,
  ci_method = c("delta", "bootstrap"),
  product_variance = c("goodman", "first_order"),
  ci_type = c("symmetric", "log")
)

```

Arguments

design	A creel_design object with both counts (via add_counts) and interviews (via add_interviews) attached. Both count and interview survey objects must exist.
by	Optional tidy selector for grouping variables. When specified, must match across both effort and CPUE estimates (same calendar strata or interview variables). Accepts bare column names, multiple columns, or tidyselect helpers. Two kinds of column are not groupings and are refused: the interview id registered by add_catch() , which holds one value per interview and so leaves no within-group variance to estimate, and columns the package derived rather than the user supplying, such as <code>.angler_effort</code> . A wildcard selector drops the derived columns silently; naming one is an error. A column of your own is never treated as derived, whatever it is called.

variance	Character string specifying variance estimation method: "taylor" (default), "bootstrap", or "jackknife". Applied to BOTH effort and CPUE estimation, then combined via delta method.
conf_level	Numeric confidence level (default: 0.95)
target	Character string specifying the effort domain supplied to <code>estimate_effort()</code> . Options are "sampled_days" (default), "stratum_total", or "period_total". This controls which effort domain is multiplied by CPUE so total catch stays aligned with the requested temporal target.
use_trips	Character. Which interviews contribute to CPUE: "complete" uses only completed trips, "all" includes incomplete ones. Default NULL means "not specified", which resolves to "complete" – except on a roving design, where it resolves with estimator as described below. Under ratio-of-means, an incomplete trip reports catch-so-far against effort-so-far, so "all" biases the rate downward; under mean-of-ratios that pooling is the intended estimator rather than a bias.
estimator	Character. Rate estimator used for the CPUE component: "ratio-of-means" (a ratio of totals), "mor" (mean of per-interview ratios), or "mortr" ("mor" with truncation made mandatory). Default NULL means "not specified". When use_trips and estimator are <i>both</i> unspecified and the design was built with <code>add_interviews(interview_type = "roving")</code> , the pair resolves to all-trip truncated MOR, matching what <code>estimate_catch_rate()</code> chooses for the same design; otherwise the pair resolves to complete-trip ratio-of-means. Specifying either one suppresses the auto-route. Bus-route and ice designs never auto-route and accept only "ratio-of-means", because their total is a ratio of Horvitz-Thompson totals with no mean-of-ratios form.
truncate_at	Numeric minimum trip duration in hours for MOR, or NULL to disable truncation. Default 0.5 (30 minutes) per Hoenig et al. (1997), who recommend the truncated mean of ratios for the roving catch rate "and hence total catch". Truncation is not a tuning knob: the untruncated mean-of-ratios estimator has infinite variance. Ignored under ratio-of-means. Requires a trip duration column on the design; without one a warning is raised and no truncation is applied. An interview whose duration is missing cannot be shown to meet the threshold, so it is excluded and reported separately from the trips excluded as too short – a missing-data loss and a truncation decision are not the same event.
aggregate_sections	Logical. When the design was created with <code>add_sections</code> , should a .lake_total row be appended that sums the per-section estimates? Default TRUE. Set to FALSE to return only the per-section rows without the lake total.
missing_sections	Character(1). Action when a registered section is absent from either count data or interview data: "warn" (default) inserts an NA row with data_available = FALSE, "error" raises a hard error.
verbose	Logical. If TRUE, prints an informational message identifying which estimator path was used. Default FALSE.
ci_method	character. "delta" (default) returns only delta-method CIs. "bootstrap" additionally returns ci_lo_boot/ci_hi_boot using survey bootstrap resampling. Only applies to bus-route/ice designs.

product_variance	character. Variance formula for the product $E \times C$. "goodman" (default) uses Goodman's (1960) unbiased estimator $E^2 Var(C) + C^2 Var(E) - Var(E)Var(C)$; the cross-term is subtracted because substituting estimates for the unknown means leaves the two-term plug-in biased upward. "first_order" omits it (classical two-term delta method), which is conservative. Both assume E and C are independently estimated. When both components are so imprecise that the subtraction would give a non-positive variance, the first-order value is used as a floor.
ci_type	character. Shape of the confidence interval. "symmetric" (default) gives the standard $\hat{\theta} \pm z \cdot SE$ interval clamped at zero. "log" applies a log-transform so the CI stays positive: $[\hat{\theta}e^{-zSE/\hat{\theta}}, \hat{\theta}e^{zSE/\hat{\theta}}]$.

Details

Total catch is computed as Effort $\times$ CPUE. Variance is propagated using the delta method, which accounts for uncertainty in both estimates. The formula for independent estimates is approximately:

$$Var(E \times C) \approx E^2 \cdot Var(C) + C^2 \cdot Var(E)$$

Variance is computed via a stratified delta-method sum in `compute_stratum_product_sum()`, not via `survey::svycontrast()`.

Sectioned designs: When `add_sections` has been called on the design, each section is estimated independently using its own count survey (via `rebuild_counts_survey`) and interview survey (via `rebuild_interview_survey`). The lake-wide total is the arithmetic sum `sum(TC_i)`, not `E_total * CPUE_pooled`. The lake-wide SE uses the zero-covariance assumption: `sqrt(sum(se_i^2))`. Cross-section covariance between count-based effort and interview-based CPUE designs is not identified and is therefore assumed zero.

`by = <species>` is supported on a sectioned design: catch is apportioned against each section's own whole effort, giving one row per section per species. As with any other grouping, the sectioned result then carries no `.lake_total` row and no `prop_of_lake_total`.

Design compatibility requirements:

- Count data must be attached via `add_counts()` for effort estimation
- Interview data must be attached via `add_interviews()` for CPUE estimation
- Grouped estimation requires identical grouping variables for both estimates
- Calendar stratification must be shared between counts and interviews

Value

A `creel_estimates` S3 object with `method = "product-total-catch"`. The estimator component records the rate estimator this total is a product of, as you asked for it: `method` names the product form and is the same string whichever estimator produced it. For bus-route and ice designs, returns a bus-route HT estimate with `method = "ht-total-catch"` and a `"site_contributions"` attribute. For sectioned designs, returns per-section rows plus (by default) a `.lake_total` row. The lake-wide total is computed as `sum(TC_i)` over sections, never as `E_total * CPUE_pooled`.

For sectioned designs the per-section rows carry `prop_of_lake_total`, the section's share of the lake-wide total, and `se_prop_of_lake_total`, its standard error. The share is a ratio whose numerator is one of its own denominator's terms, and whose numerator and denominator are each products of an effort and a rate estimated from different designs, so the error is derived by delta method from the same section variances and covariance the `.lake_total` row's own standard error is built from. The `.lake_total` row reports `se_prop_of_lake_total = 0`: its share of itself is exactly 1 by construction and was never estimated. A section with no data reports NA for both. Neither column is produced on the grouped path.

Why there is no targeted argument

The rate functions accept `targeted = FALSE`, which restricts the domain to the interviews that recorded some of the species being estimated. The totals deliberately do not, because a total is a rate multiplied by an effort base and the two would no longer describe the same set of trips.

A targeted rate is conditional on having recorded the species; total effort is not. Multiplying one by the other applies a conditional rate to an unconditional base. On the package's own example data one species' rate is 0.48 fish/hr over all 50 trips and 2.00 fish/hr over the 12 that caught it, so expanding the targeted rate by total effort returns roughly 223 fish where 30 were actually caught.

The domain-consistent product — the targeted rate times the effort of the trips that recorded the species — is well defined in the sample but cannot be expanded: it needs the season-wide effort of species-catching trips, which no creel design observes.

So a targeted **rate** is available and a targeted **total** is not, and that is a property of the estimand rather than a gap in the implementation (GH #307).

What the pooled total assumes

Effort comes from the counts, so a total can only be broken down by an attribute the counts classify. When a domain appears in the interviews but not in the counts, the only available total is `E_total * rate_pooled`, where the pooled rate is a ratio of means weighted by the *interview sample's* composition over that domain. Had the domain been classified in the counts it would be a stratum and the total would be `sum(E_h * rate_h)`, which is unbiased whatever the interview composition happens to be.

The two agree only when the interview sample's effort composition matches the true effort composition, and interview selection is not proportional to effort by construction of the standard designs. Access interviews intercept completed trips, over-representing anglers who must return to a fixed point: Malvestuto (1996) notes that it is “usually impossible to sample all angler types proportional to their level of effort”, a particular problem for bank anglers who may be “widely dispersed along the shoreline and not associated with well-defined access sites”. Roving interviews are length-biased toward longer trips. So the mix differs by design rather than by accident, and where levels differ in rate the pooled total inherits that difference.

None of this is verifiable from within the data, because the counts carry no composition to compare against. Where it is detectable — the interviews hold an unclassified categorical domain and the crude rate differs materially across its levels — a warning of class `creel_warning_pooled_domain_mix` is raised. It flags a risk, not a defect. Classifying the domain in the count data is what removes the assumption.

Unit of the total

The reported unit is derived from the two factors, never declared. A total is "fish" only when a per-angler-hour rate multiplies an effort in angler-hours; anything else reports NA_character_, meaning unknown.

Two ways to fail to cancel:

- **The effort unit is unknown.** `design$effort_unit` is NA whenever `add_counts()` received no `period_length_col`, because a bare count column may be an instantaneous head count or effort the caller already expanded, and nothing can tell the two apart. Unknown times known is unknown. Supply `period_length_col` to make the total's unit derivable.
- **The denominators disagree.** A rate per party-hour times an effort in angler-hours is not a count of fish. Pass `n_angers` to `add_interviews()` so the rate is per angler-hour.

The estimate itself is unaffected in both cases – only the label changes. Until version 5.2.0 the unit was the literal "fish" regardless of either factor (GH #213).

See Also

`estimate_effort`, `estimate_catch_rate`

Other "Estimation": `compare_cpue_estimators()`, `est_age_distribution()`, `est_biomass()`, `est_compliance()`, `est_effort_camera_mi()`, `est_length_distribution()`, `est_mean_age()`, `est_mean_length()`, `estimate_catch_rate()`, `estimate_effort()`, `estimate_effort_aerial_glm()`, `estimate_harvest_rate()`, `estimate_release_rate()`, `estimate_total_harvest()`, `estimate_total_release()`

Examples

```
library(tidycree)
data(example_calendar)
data(example_counts)
data(example_interviews)

# Create design with both counts and interviews
design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, n_angers = n_angers,
  trip_status = trip_status, trip_duration = trip_duration
)

# Estimate total catch
total_catch <- estimate_total_catch(design)
print(total_catch)

# Compare components
effort_est <- estimate_effort(design)
cpue_est <- estimate_catch_rate(design)
# The total is close to, but not exactly, effort times CPUE
c(
  total = total_catch$estimates$estimate,
  effort_x_cpue = effort_est$estimates$estimate * cpue_est$estimates$estimate
)
```

```
)

# Grouped estimation needs at least 10 interviews per group, so check
# the sample sizes before grouping
table(design$interviews$day_type)

# Verbose dispatch message (shows which estimator was used for bus-route designs)
result_verbose <- estimate_total_catch(design, verbose = TRUE)
```

```
estimate_total_harvest
```

Estimate total harvest by combining effort and HPUE

Description

Computes total harvest estimates by multiplying effort $\times$ HPUE with variance propagation via the delta method. Requires a creel design with both count data (for effort estimation) and interview data (for HPUE estimation).

Usage

```
estimate_total_harvest(
  design,
  by = NULL,
  variance = "taylor",
  conf_level = 0.95,
  target = c("sampled_days", "stratum_total", "period_total"),
  use_trips = NULL,
  estimator = NULL,
  truncate_at = 0.5,
  aggregate_sections = TRUE,
  missing_sections = "warn",
  ci_method = c("delta", "bootstrap"),
  product_variance = c("goodman", "first_order"),
  ci_type = c("symmetric", "log")
)
```

Arguments

design	A creel_design object with both counts (via add_counts) and interviews (via add_interviews) attached. Both count and interview survey objects must exist. Interview data must include harvest column (specified via harvest parameter in add_interviews).
by	Optional tidy selector for grouping variables. When specified, must match across both effort and HPUE estimates (same calendar strata or interview variables). Accepts bare column names, multiple columns, or tidyselect helpers.

	Two kinds of column are not groupings and are refused: the interview id registered by <code>add_catch()</code> , which holds one value per interview and so leaves no within-group variance to estimate, and columns the package derived rather than the user supplying, such as <code>.angler_effort</code> . A wildcard selector drops the derived columns silently; naming one is an error. A column of your own is never treated as derived, whatever it is called.
variance	Character string specifying variance estimation method: "taylor" (default), "bootstrap", or "jackknife". Applied to BOTH effort and HPUE estimation, then combined via delta method.
conf_level	Numeric confidence level (default: 0.95)
target	Character string specifying the effort domain supplied to <code>estimate_effort()</code> . Options are "sampled_days" (default), "stratum_total", or "period_total". This controls which effort domain is multiplied by HPUE so total harvest stays aligned with the requested temporal target.
use_trips	Character. Which interviews contribute to HPUE. "complete" uses only completed trips; "all" includes incomplete ones. Default NULL means "not specified", which resolves to "complete". An interview taken mid-trip reports the harvest so far against the effort so far, and the two do not scale together over the trip, so "all" gives a length-biased rate and a total built from it. Ignored when the design carries no trip status column. Since GH #271 a roving design routes to all-trip mean-of-ratios here, as it does for <code>estimate_total_catch()</code> , because <code>estimate_harvest_rate()</code> gained the same estimator selection. Both resolve through the same rule, so the total always agrees with its own rate function.
estimator	Character string selecting the rate estimator used for the HPUE component: "ratio-of-means", "mor", or "mortr". Default NULL means "not specified"; see <code>estimate_harvest_rate()</code> for how the pair resolves and when the roving auto-route applies. Bus-route and ice designs accept only "ratio-of-means", because their total is a ratio of Horvitz-Thompson totals with no mean-of-ratios form.
truncate_at	Numeric minimum trip duration in hours for MOR, or NULL to disable truncation. Default 0.5 (30 minutes) per Hoenig et al. (1997). Truncation is not a tuning knob: the untruncated mean-of-ratios estimator has infinite variance. Ignored under ratio-of-means.
aggregate_sections	Logical. When the design was created with <code>add_sections</code> , should a <code>.lake_total</code> row be appended that sums the per-section estimates? Default TRUE. Set to FALSE to return only the per-section rows without the lake total.
missing_sections	Character(1). Action when a registered section is absent from either count data or interview data: "warn" (default) inserts an NA row with <code>data_available = FALSE</code> , "error" raises a hard error.
ci_method	character. "delta" (default) returns only delta-method CIs. "bootstrap" additionally returns <code>ci_lo_boot/ci_hi_boot</code> using survey bootstrap resampling. Only applies to bus-route/ice designs.

product_variance	character. Variance formula for the product $E \times H$. "goodman" (default) uses Goodman's (1960) unbiased estimator $E^2 Var(H) + H^2 Var(E) - Var(E)Var(H)$; "first_order" omits the cross-term, which is conservative. Both assume E and H are independently estimated. When both components are so imprecise that the subtraction would give a non-positive variance, the first-order value is used as a floor.
ci_type	character. Shape of the confidence interval. "symmetric" (default) gives $\hat{\theta} \pm z \cdot SE$ clamped at zero. "log" applies a log-transform for a strictly positive CI.

Details

Total harvest is computed as Effort $\times$ HPUE. Variance is propagated using the delta method, which accounts for uncertainty in both estimates. The formula for independent estimates is approximately:

$$Var(E \times H) \approx E^2 \cdot Var(H) + H^2 \cdot Var(E)$$

Variance is computed via a stratified delta-method sum in `compute_stratum_product_sum()`, not via `survey::svycontrast()`.

Sectioned designs: When `add_sections` has been called on the design, each section is estimated independently. The lake-wide total is `sum(TH_i)`, not `E_total * HPUE_pooled`. The lake-wide SE uses the zero-covariance assumption: `sqrt(sum(se_i^2))`.

`by = <species>` is supported on a sectioned design: catch is apportioned against each section's own whole effort, giving one row per section per species. As with any other grouping, the sectioned result then carries no `.lake_total` row and no `prop_of_lake_total`.

Design compatibility requirements:

- Count data must be attached via `add_counts()` for effort estimation
- Interview data must be attached via `add_interviews()` for HPUE estimation
- Harvest column must be specified in `add_interviews` (harvest parameter)
- Grouped estimation requires identical grouping variables for both estimates
- Calendar stratification must be shared between counts and interviews

Value

A `creel_estimates` S3 object with `method = "product-total-harvest"`. The estimator component records the rate estimator this total is a product of, as you asked for it: `method` names the product form and is the same string whichever estimator produced it. For bus-route and ice designs, returns a bus-route HT estimate with `method = "ht-total-harvest"` and a `"site_contributions"` attribute.

For sectioned designs the per-section rows carry `prop_of_lake_total`, the section's share of the lake-wide total, and `se_prop_of_lake_total`, its standard error. The share is a ratio whose numerator is one of its own denominator's terms, and whose numerator and denominator are each products of an effort and a rate estimated from different designs, so the error is derived by delta method from the same section variances and covariance the `.lake_total` row's own standard error is built from. The `.lake_total` row reports `se_prop_of_lake_total = 0`: its share of itself is exactly 1 by construction and was never estimated. A section with no data reports NA for both. Neither column is produced on the grouped path.

Why there is no targeted argument

The rate functions accept `targeted = FALSE`, which restricts the domain to the interviews that recorded some of the species being estimated. The totals deliberately do not, because a total is a rate multiplied by an effort base and the two would no longer describe the same set of trips.

A targeted rate is conditional on having recorded the species; total effort is not. Multiplying one by the other applies a conditional rate to an unconditional base. On the package's own example data one species' rate is 0.48 fish/hr over all 50 trips and 2.00 fish/hr over the 12 that caught it, so expanding the targeted rate by total effort returns roughly 223 fish where 30 were actually caught.

The domain-consistent product — the targeted rate times the effort of the trips that recorded the species — is well defined in the sample but cannot be expanded: it needs the season-wide effort of species-catching trips, which no creel design observes.

So a targeted **rate** is available and a targeted **total** is not, and that is a property of the estimand rather than a gap in the implementation (GH #307).

What the pooled total assumes

Effort comes from the counts, so a total can only be broken down by an attribute the counts classify. When a domain appears in the interviews but not in the counts, the only available total is $E_total * rate_pooled$, where the pooled rate is a ratio of means weighted by the *interview sample's* composition over that domain. Had the domain been classified in the counts it would be a stratum and the total would be $\sum(E_h * rate_h)$, which is unbiased whatever the interview composition happens to be.

The two agree only when the interview sample's effort composition matches the true effort composition, and interview selection is not proportional to effort by construction of the standard designs. Access interviews intercept completed trips, over-representing anglers who must return to a fixed point: Malvestuto (1996) notes that it is "usually impossible to sample all angler types proportional to their level of effort", a particular problem for bank anglers who may be "widely dispersed along the shoreline and not associated with well-defined access sites". Roving interviews are length-biased toward longer trips. So the mix differs by design rather than by accident, and where levels differ in rate the pooled total inherits that difference.

None of this is verifiable from within the data, because the counts carry no composition to compare against. Where it is detectable — the interviews hold an unclassified categorical domain and the crude rate differs materially across its levels — a warning of class `creel_warning_pooled_domain_mix` is raised. It flags a risk, not a defect. Classifying the domain in the count data is what removes the assumption.

Unit of the total

The reported unit is derived from the two factors, never declared. A total is "fish" only when a per-angler-hour rate multiplies an effort in angler-hours; anything else reports `NA_character_`, meaning unknown.

Two ways to fail to cancel:

- **The effort unit is unknown.** `design$effort_unit` is NA whenever `add_counts()` received no `period_length_col`, because a bare count column may be an instantaneous head count or effort the caller already expanded, and nothing can tell the two apart. Unknown times known is unknown. Supply `period_length_col` to make the total's unit derivable.

- **The denominators disagree.** A rate per party-hour times an effort in angler-hours is not a count of fish. Pass `n_anglers` to `add_interviews()` so the rate is per angler-hour.

The estimate itself is unaffected in both cases – only the label changes. Until version 5.2.0 the unit was the literal "fish" regardless of either factor (GH #213).

See Also

`estimate_effort`, `estimate_harvest_rate`, `estimate_total_catch`

Other "Estimation": `compare_cpue_estimators()`, `est_age_distribution()`, `est_biomass()`, `est_compliance()`, `est_effort_camera_mi()`, `est_length_distribution()`, `est_mean_age()`, `est_mean_length()`, `estimate_catch_rate()`, `estimate_effort()`, `estimate_effort_aerial_glm()`, `estimate_harvest_rate()`, `estimate_release_rate()`, `estimate_total_catch()`, `estimate_total_release()`

Examples

```
library(tidycree)
data(example_calendar)
data(example_counts)
data(example_interviews)

# Create design with both counts and interviews including harvest
design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, harvest = catch_kept, effort = hours_fished,
  n_anglers = n_anglers,
  trip_status = trip_status, trip_duration = trip_duration
)

# Estimate total harvest
total_harvest <- estimate_total_harvest(design)
print(total_harvest)

# Compare components
effort_est <- estimate_effort(design)
hpue_est <- estimate_harvest_rate(design)
# The total is close to, but not exactly, effort times HPUE
c(
  total = total_harvest$estimates$estimate,
  effort_x_hpue = effort_est$estimates$estimate * hpue_est$estimates$estimate
)

# Grouped estimation needs at least 10 interviews per group, so check
# the sample sizes before grouping
table(design$interviews$day_type)
```

estimate_total_release

Estimate total extrapolated release by combining effort and release rate

Description

Computes total release estimates by multiplying effort x RPUE with variance propagation via the delta method. Requires a creel design with count data (for effort estimation), interview data (for effort), and catch data (via [add_catch](#)) containing released records.

Usage

```
estimate_total_release(
  design,
  by = NULL,
  variance = "taylor",
  conf_level = 0.95,
  target = c("sampled_days", "stratum_total", "period_total"),
  use_trips = NULL,
  estimator = NULL,
  truncate_at = 0.5,
  aggregate_sections = TRUE,
  missing_sections = "warn",
  product_variance = c("goodman", "first_order"),
  ci_type = c("symmetric", "log")
)
```

Arguments

design	A creel_design object with counts (via add_counts), interviews (via add_interviews), and catch data (via add_catch) attached. Catch data must include records with catch_type = "released".
by	Optional tidy selector for grouping variables. Accepts bare column names (e.g., by = day_type, by = species), multiple columns, or tidyselect helpers. Two kinds of column are not groupings and are refused: the interview id registered by add_catch() , which holds one value per interview and so leaves no within-group variance to estimate, and columns the package derived rather than the user supplying, such as .angler_effort. A wildcard selector drops the derived columns silently; naming one is an error. A column of your own is never treated as derived, whatever it is called.
variance	Character string specifying variance estimation method: "taylor" (default), "bootstrap", or "jackknife". Applied to BOTH effort and release rate estimation, then combined via delta method.
conf_level	Numeric confidence level (default: 0.95).

target	Character string specifying the effort domain supplied to <code>estimate_effort()</code> . Options are "sampled_days" (default), "stratum_total", or "period_total". This controls which effort domain is multiplied by release rate so total release stays aligned with the requested temporal target.
use_trips	Character. Which interviews contribute to RPUE. "complete" uses only completed trips; "all" includes incomplete ones. Default NULL means "not specified", which resolves to "complete". An interview taken mid-trip reports the releases so far against the effort so far, and the two do not scale together over the trip, so "all" gives a length-biased rate and a total built from it. Ignored when the design carries no trip status column. Since GH #271 a roving design routes to all-trip mean-of-ratios here, as it does for <code>estimate_total_catch()</code> , because <code>estimate_release_rate()</code> gained the same estimator selection. Both resolve through the same rule, so the total always agrees with its own rate function.
estimator	Character string selecting the rate estimator used for the RPUE component: "ratio-of-means", "mor", or "mortr". Default NULL means "not specified"; see <code>estimate_release_rate()</code> for how the pair resolves and when the roving auto-route applies. Bus-route and ice designs accept only "ratio-of-means", because their total is a ratio of Horvitz-Thompson totals with no mean-of-ratios form.
truncate_at	Numeric minimum trip duration in hours for MOR, or NULL to disable truncation. Default 0.5 (30 minutes) per Hoenig et al. (1997). Truncation is not a tuning knob: the untruncated mean-of-ratios estimator has infinite variance. Ignored under ratio-of-means.
aggregate_sections	Logical. When the design was created with <code>add_sections</code> , should a .lake_total row be appended that sums the per-section estimates? Default TRUE. Set to FALSE to return only the per-section rows without the lake total.
missing_sections	Character(1). Action when a registered section is absent from either count data or interview data: "warn" (default) inserts an NA row with data_available = FALSE, "error" raises a hard error.
product_variance	character. Variance formula for the product $E \times R$. "goodman" (default) uses Goodman's (1960) unbiased estimator $E^2 Var(R) + R^2 Var(E) - Var(E) Var(R)$; "first_order" omits the cross-term, which is conservative. Both assume E and R are independently estimated. When both components are so imprecise that the subtraction would give a non-positive variance, the first-order value is used as a floor.
ci_type	character. Shape of the confidence interval. "symmetric" (default) gives $\hat{\theta} \pm z \cdot SE$ clamped at zero. "log" applies a log-transform for a strictly positive CI.

Details

Total release is computed as Effort x RPUE. Variance is propagated using the delta method: $Var(E \times R) = E^2 \cdot Var(R) + R^2 \cdot Var(E)$.

Sectioned designs: When `add_sections` has been called on the design, each section is estimated independently. The lake-wide total is `sum(TR_i)`, not `E_total * RPUE_pooled`. The lake-wide SE uses the zero-covariance assumption: `sqrt(sum(se_i^2))`.

`by = <species>` is supported on a sectioned design: `catch` is apportioned against each section's own whole effort, giving one row per section per species. As with any other grouping, the sectioned result then carries no `.lake_total` row and no `prop_of_lake_total`.

Value

A `creel_estimates` S3 object with `method = "product-total-release"`. The estimator component records the rate estimator this total is a product of, as you asked for it: `method` names the product form and is the same string whichever estimator produced it. Estimates tibble has columns: `estimate`, `se`, `ci_lower`, `ci_upper`, `n` (plus any grouping columns). For bus-route and ice designs, returns a bus-route HT estimate with `method = "ht-total-release"` and a `"site_contributions"` attribute.

For sectioned designs the per-section rows carry `prop_of_lake_total`, the section's share of the lake-wide total, and `se_prop_of_lake_total`, its standard error. The share is a ratio whose numerator is one of its own denominator's terms, and whose numerator and denominator are each products of an effort and a rate estimated from different designs, so the error is derived by delta method from the same section variances and covariance the `.lake_total` row's own standard error is built from. The `.lake_total` row reports `se_prop_of_lake_total = 0`: its share of itself is exactly 1 by construction and was never estimated. A section with no data reports NA for both. Neither column is produced on the grouped path.

Why there is no targeted argument

The rate functions accept `targeted = FALSE`, which restricts the domain to the interviews that recorded some of the species being estimated. The totals deliberately do not, because a total is a rate multiplied by an effort base and the two would no longer describe the same set of trips.

A targeted rate is conditional on having recorded the species; total effort is not. Multiplying one by the other applies a conditional rate to an unconditional base. On the package's own example data one species' rate is 0.48 fish/hr over all 50 trips and 2.00 fish/hr over the 12 that caught it, so expanding the targeted rate by total effort returns roughly 223 fish where 30 were actually caught.

The domain-consistent product — the targeted rate times the effort of the trips that recorded the species — is well defined in the sample but cannot be expanded: it needs the season-wide effort of species-catching trips, which no creel design observes.

So a targeted **rate** is available and a targeted **total** is not, and that is a property of the estimand rather than a gap in the implementation (GH #307).

What the pooled total assumes

Effort comes from the counts, so a total can only be broken down by an attribute the counts classify. When a domain appears in the interviews but not in the counts, the only available total is `E_total * rate_pooled`, where the pooled rate is a ratio of means weighted by the *interview sample's* composition over that domain. Had the domain been classified in the counts it would be a stratum and the total would be `sum(E_h * rate_h)`, which is unbiased whatever the interview composition happens to be.

The two agree only when the interview sample's effort composition matches the true effort composition, and interview selection is not proportional to effort by construction of the standard designs.

Access interviews intercept completed trips, over-representing anglers who must return to a fixed point: Malvestuto (1996) notes that it is “usually impossible to sample all angler types proportional to their level of effort”, a particular problem for bank anglers who may be “widely dispersed along the shoreline and not associated with well-defined access sites”. Roving interviews are length-biased toward longer trips. So the mix differs by design rather than by accident, and where levels differ in rate the pooled total inherits that difference.

None of this is verifiable from within the data, because the counts carry no composition to compare against. Where it is detectable – the interviews hold an unclassified categorical domain and the crude rate differs materially across its levels – a warning of class `creel_warning_pooled_domain_mix` is raised. It flags a risk, not a defect. Classifying the domain in the count data is what removes the assumption.

Unit of the total

The reported unit is derived from the two factors, never declared. A total is “fish” only when a per-angler-hour rate multiplies an effort in angler-hours; anything else reports `NA_character_`, meaning unknown.

Two ways to fail to cancel:

- **The effort unit is unknown.** `design$effort_unit` is NA whenever `add_counts()` received no `period_length_col`, because a bare count column may be an instantaneous head count or effort the caller already expanded, and nothing can tell the two apart. Unknown times known is unknown. Supply `period_length_col` to make the total’s unit derivable.
- **The denominators disagree.** A rate per party-hour times an effort in angler-hours is not a count of fish. Pass `n_anglers` to `add_interviews()` so the rate is per angler-hour.

The estimate itself is unaffected in both cases – only the label changes. Until version 5.2.0 the unit was the literal “fish” regardless of either factor (GH #213).

See Also

`estimate_total_harvest`, `estimate_release_rate`, `add_catch`

Other “Estimation”: `compare_cpue_estimators()`, `est_age_distribution()`, `est_biomass()`, `est_compliance()`, `est_effort_camera_mi()`, `est_length_distribution()`, `est_mean_age()`, `est_mean_length()`, `estimate_catch_rate()`, `estimate_effort()`, `estimate_effort_aerial_glm()`, `estimate_harvest_rate()`, `estimate_release_rate()`, `estimate_total_catch()`, `estimate_total_harvest()`

Examples

```
library(tidycrrel)
data(example_calendar)
data(example_counts)
data(example_interviews)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, n_anglers = n_anglers,
```

```

    trip_status = trip_status, trip_duration = trip_duration
  )
design <- add_catch(design, example_catch,
  catch_uid = interview_id, interview_uid = interview_id,
  species = species, count = count, catch_type = catch_type
)

# Total releases (all species combined)
total_rel <- estimate_total_release(design)
print(total_rel)

# Total releases by species
total_rel_sp <- estimate_total_release(design, by = species)
print(total_rel_sp)

```

est_age_distribution *Estimate a weighted age distribution from creel interview data*

Description

est_age_distribution() estimates a pressure-weighted age-frequency distribution from fish age data attached via [add_ages\(\)](#). Age records are aggregated through the internal interview survey design so the result reflects the survey design rather than only the observed sample.

Ages are discrete integers; each unique observed age is its own class. The estimator returns one row per occupied integer age, with weighted totals, standard errors, confidence intervals, and within-group percentages.

Usage

```

est_age_distribution(
  design,
  by = NULL,
  type = "catch",
  variance = "taylor",
  conf_level = 0.95
)

```

Arguments

design	A creel_design object with interviews and ages attached.
by	Optional tidy selector evaluated against design\$ages. Common choices include by = species.
type	Character string indicating which fish to include. One of "catch" (default; both harvest and release), "harvest", or "release".
variance	Character string specifying variance estimation method. One of "taylor" (default), "bootstrap", or "jackknife".
conf_level	Numeric confidence level for confidence intervals. Default 0.95.

Value

A data.frame with class `c("creel_age_distribution", "data.frame")` and columns: grouping columns (if any), age (integer), estimate, se, ci_lower, ci_upper, percent, cumulative_percent, and n.

percent and cumulative_percent are shares of the group's estimated total, rounded to one decimal for display; cumulative_percent accumulates the unrounded shares, so it reaches 100 rather than drifting. The exception is a group whose estimated total is zero, where there are no shares to take and both columns are 0 rather than reaching 100.

n is the number of **interviews** contributing at least one aged fish to the group. It is therefore constant across every age class of a group, and is neither a per-class sample size nor a count of fish.

Two-phase estimation onto the reported catch

Ages are read from a **subsample** of the catch, exactly as lengths are, so the age-class totals are scaled onto the design-estimated reported total rather than reporting the subsample: $\hat{N}_a = \hat{p}_a \hat{T}$. See [est_length_distribution\(\)](#) for the estimator, its variance, and where $\hat{T}$ comes from. percent and cumulative_percent are unaffected. The call warns when it rescales and aborts when no total is available (GH #310).

See Also

Other "Estimation": [compare_cpue_estimators\(\)](#), [est_biomass\(\)](#), [est_compliance\(\)](#), [est_effort_camera_mi\(\)](#), [est_length_distribution\(\)](#), [est_mean_age\(\)](#), [est_mean_length\(\)](#), [estimate_catch_rate\(\)](#), [estimate_effort\(\)](#), [estimate_effort_aerial_glm\(\)](#), [estimate_harvest_rate\(\)](#), [estimate_release_rate\(\)](#), [estimate_total_catch\(\)](#), [estimate_total_harvest\(\)](#), [estimate_total_release\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_ages)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
# Species catch is required to group by species: the totals are scaled onto
# the reported catch, and only this table records it per species.
design <- add_catch(design, example_catch,
  catch_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  count = count,
  catch_type = catch_type
)
design <- add_ages(design, example_ages,
  age_uid = interview_id,
```

```

    interview_uid = interview_id,
    species = species,
    age = age,
    age_type = age_type
)

est_age_distribution(design, by = species)

```

est_biomass

*Estimate total biomass from a creel length distribution***Description**

`est_biomass()` converts a pressure-weighted length-frequency distribution produced by `est_length_distribution()` into a total biomass estimate using the allometric length-weight equation $W = a \cdot L^b$.

Variance is propagated via the delta method, carrying the **full covariance** among the estimated fish counts per length bin and treating the length-weight parameters a and b as known without error unless their standard errors are supplied (see Details). Before GH #311 the bin counts were treated as uncorrelated, which under-estimated the variance.

Since GH #310 the counts supplied by `est_length_distribution()` describe the **reported catch** rather than the measured subsample, so `biomass_estimate` is a catch biomass. It previously described only the fish that were measured.

Usage

```

est_biomass(
  ld,
  a,
  b,
  conf_level = NULL,
  alpha_se = NULL,
  b_se = NULL,
  L0 = NULL
)

```

Arguments

<code>ld</code>	A <code>creel_length_distribution</code> object from <code>est_length_distribution()</code> .
<code>a</code>	Positive numeric allometric coefficient (the a in $W = a \cdot L^b$).
<code>b</code>	Numeric allometric exponent (the b in $W = a \cdot L^b$). Typical values for fish are 2.5–3.5.
<code>conf_level</code>	Numeric confidence level for confidence intervals. Defaults to the level stored in <code>ld</code> (usually 0.95).
<code>alpha_se</code>	Optional standard error of the pivot coefficient $\alpha = a \cdot L_0^b$, i.e. the fitted intercept on the $\log W = \log \alpha + b(\log L - \log L_0)$ scale.

b_se	Optional standard error of the exponent b.
L0	Optional pivot length at which the regression was centred, in the same units as the bin boundaries. Use the geometric mean length of the length-weight calibration sample. These three are all-or-nothing: give all of them to propagate the length-weight regression error, or none to keep the current behaviour. There is no zero default — see Details.

Details

For each length bin h with midpoint $L_h = (\text{bin_lower} + \text{bin_upper})/2$, per-bin biomass is $B_h = a \cdot L_h^b \cdot \hat{N}_h$, where $\hat{N}_h$ is the survey-weighted estimated fish count from `est_length_distribution()`. Total biomass is $B = \sum_h B_h$.

Variance is the quadratic form $\widehat{\text{Var}}(B) = w' \Sigma w$ with $w_h = a \cdot L_h^b$ and Σ the bins' full covariance matrix, carried from the single `svytotal()` that estimated them. Earlier versions used $\sum_h w_h^2 \widehat{\text{SE}}_h^2$ — the same expression with every off-diagonal set to zero — which under-estimated the variance, since the bins partition the same fish and are rescaled onto one reported total.

If Σ is unavailable — the object was produced by an older version, or was subsetting in a way that dropped the attribute carrying it — the independence form is used and a warning says so. An absent covariance is unknown, not zero.

By default a and b are treated as known constants, so `biomass_se` carries no contribution from their estimation error. In practice they are point estimates from a length-weight regression, often one fitted to a different water body or year. Because $a \cdot L_h^b$ multiplies every bin, that error is perfectly correlated across bins and does not shrink as bins are added — unlike the cross-bin term above.

The omission is usually minor relative to count variance: on the example below it adds roughly 2–11% to a coefficient of variation of 40–65%, for regression standard errors spanning well- and poorly-determined fits. It becomes material in two situations — a survey precise enough to bring the count CV near 10%, and a/b borrowed from a system whose fish differ in size from those measured here, since the contribution scales with the distance between the two samples' mean log lengths.

Value

A `data.frame` with class `c("creel_biomass", "data.frame")` and columns: grouping columns (if any), `biomass_estimate`, `biomass_se`, `biomass_ci_lower`, `biomass_ci_upper`.

Propagating the length-weight regression error

Supply `alpha_se`, `b_se`, and `L0` together to carry that term. The allometry is rewritten about a pivot length L_0 :

$$W = \alpha \left(\frac{L}{L_0} \right)^b, \quad \alpha = a L_0^b$$

and the delta method is applied in (α, b) :

$$\widehat{\text{Var}}(B) \approx \sum_h (a L_h^b)^2 \widehat{\text{SE}}_h^2 + \left(\frac{B}{\alpha} \right)^2 \text{Var}(\alpha) + \left(\sum_h B_h \ln \frac{L_h}{L_0} \right)^2 \text{Var}(b)$$

The covariance term is absent by construction rather than by assumption. Fitted on the raw (a, b) scale the two parameters are almost perfectly negatively correlated — typically $\text{cor} < -0.99$ — so dropping their covariance there would **overstate** the variance severalfold, in some cases turning a 2–11% contribution into 5–49%. Centring at L_0 makes them near-orthogonal, so the omitted term is genuinely negligible. Take L_0 as the geometric mean length of the calibration sample, and take alpha_se from the intercept of a regression centred there — not the standard error of a itself.

The contribution grows with $\ln(L_h/L_0)$, so borrowing parameters from a system whose fish differ in size from these is penalised automatically, which is the intended behaviour.

There is deliberately no zero default for these arguments. A zero standard error would produce a biomass_se identical to an unpropagated one while appearing to have been propagated — worse than the documented omission it would replace. When they are absent, `attr(x, "biomass_se_params")` is `NULL` rather than `0`, and biomass_se should be read as a lower bound.

Length and weight units are determined by the user: if lengths are in mm and a is calibrated for mm input, weights are returned in the corresponding unit (e.g., grams).

See Also

Other "Estimation": `compare_cpue_estimators()`, `est_age_distribution()`, `est_compliance()`, `est_effort_camera_mi()`, `est_length_distribution()`, `est_mean_age()`, `est_mean_length()`, `estimate_catch_rate()`, `estimate_effort()`, `estimate_effort_aerial_glm()`, `estimate_harvest_rate()`, `estimate_release_rate()`, `estimate_total_catch()`, `estimate_total_harvest()`, `estimate_total_release()`

Examples

```
data(example_calendar)
data(example_interviews)
data(example_lengths)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
# Species catch is required to group by species: the totals are scaled onto
# the reported catch, and only this table records it per species.
design <- add_catch(design, example_catch,
  catch_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  count = count,
  catch_type = catch_type
)
design <- add_lengths(design, example_lengths,
  length_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  length = length,
  length_type = length_type,
```

```

    count = count,
    release_format = "binned"
  )

ld <- est_length_distribution(design, by = species, bin_width = 25)
est_biomass(ld, a = 0.0088, b = 3.1)

```

est_compliance	<i>Estimate design-weighted size-limit compliance from a creel length distribution</i>
----------------	--

Description

est_compliance() estimates the proportion of fish meeting a minimum size limit from a [est_length_distribution\(\)](#) object. Fish in bins whose lower bound is at or above min_length are classified as legal (conservative: bins straddling the limit are classified as illegal).

Usage

```
est_compliance(ld, min_length, conf_level = NULL)
```

Arguments

ld	A creel_length_distribution object from est_length_distribution() .
min_length	Positive numeric minimum legal length in the same units as the lengths used to build ld.
conf_level	Numeric confidence level for confidence intervals. Defaults to the level stored in ld (usually 0.95).

Details

A bin is legal when $\text{bin_lower} \geq \text{min_length}$. The compliance proportion and its variance use the ratio estimator:

$$P = \frac{\sum_h I_h \hat{N}_h}{\hat{N}}$$

$$\widehat{\text{Var}}(P) = \frac{1}{\hat{N}^2} w' \Sigma w, \quad w_h = I_h - P$$

where $I_h = \mathbf{1}(\text{bin_lower}_h \geq \text{min_length})$ and Σ is the bins' full covariance matrix, carried from the single `svytotal()` that estimated them.

Earlier versions used $\sum_h w_h^2 \widehat{\text{SE}}_h^2$ — the same expression with every off-diagonal set to zero. The bins partition the same fish and are rescaled onto one reported total, so they are strongly dependent, and on the package's own example data that form reported a standard error **32% below** an independently computed survey::svyratio() reference. If Σ is unavailable the independence form is used and a warning says so.

Confidence interval bounds are clamped to $[0, 1]$.

Choose bin_width in [est_length_distribution\(\)](#) smaller than the typical variation near the legal limit to minimise classification error for bins that straddle the threshold.

Value

A data.frame with class `c("creel_compliance", "data.frame")` and columns: grouping columns (if any), `min_length`, `n_legal_est`, `n_total_est`, `compliance_prop`, `compliance_se`, `compliance_ci_lower`, `compliance_ci_upper`. Rows where the total estimated fish is zero or negative return NA for all numeric columns with a warning.

See Also

Other "Estimation": [compare_cpue_estimators\(\)](#), [est_age_distribution\(\)](#), [est_biomass\(\)](#), [est_effort_camera_mi\(\)](#), [est_length_distribution\(\)](#), [est_mean_age\(\)](#), [est_mean_length\(\)](#), [estimate_catch_rate\(\)](#), [estimate_effort\(\)](#), [estimate_effort_aerial_glm\(\)](#), [estimate_harvest_rate\(\)](#), [estimate_release_rate\(\)](#), [estimate_total_catch\(\)](#), [estimate_total_harvest\(\)](#), [estimate_total_release\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_lengths)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
# Species catch is required to group by species: the totals are scaled onto
# the reported catch, and only this table records it per species.
design <- add_catch(design, example_catch,
  catch_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  count = count,
  catch_type = catch_type
)
design <- add_lengths(design, example_lengths,
  length_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  length = length,
  length_type = length_type,
  count = count,
  release_format = "binned"
)

ld <- est_length_distribution(design, by = species, bin_width = 25)
est_compliance(ld, min_length = 356) # 14-inch limit in mm
```

est_effort_camera	<i>Estimate angler effort from camera/time-lapse count data</i>
-------------------	---

Description

Estimates total angler-hours from a camera-based creel survey design. Two estimation modes are supported:

Usage

```
est_effort_camera(
  design,
  interviews = NULL,
  effort_col = "hours_fished",
  n_anglers = NULL,
  intercept_col = NULL,
  h_open = NULL,
  calibration = NULL,
  variance = c("taylor", "replicate"),
  conf_level = 0.95
)
```

Arguments

design	A creel_design object created with creel_design(..., survey_type = "camera") and counts attached via add_counts().
interviews	Optional data frame of angler interview records for ratio calibration. Must contain effort_col and every column in design\$strata_cols: the calibration ratio is estimated within each stratum the design declares, so a missing stratum column is an error rather than a coarser calibration. When NULL, falls back to raw count expansion and h_open is required.
effort_col	Character scalar. Column in interviews containing per-trip effort in hours. Default "hours_fished".
n_anglers	Optional party size for the ratio-calibration path. Either a character scalar naming a column in interviews, or a single positive number stating a constant party size (n_anglers = 1 for individual-level interviews). The calibration ratio cancels the camera counts, so the estimate inherits whatever unit effort_col holds. Supplying n_anglers makes this function perform the party-size multiplication, so the result is angler-hours and is labelled as such. Omitting it leaves the estimate in the unit of the column you supplied, which the package cannot identify: the unit is reported as unknown and a warning names the ambiguity. Default NULL.
intercept_col	Character scalar or NULL. Column in the count data representing the camera count during the interview interception period. Default NULL (auto-detects the first numeric count column).

h_open	Numeric scalar. Fishable hours per day. Required when interviews = NULL. Default NULL.
calibration	Pass the string "none" to run the raw-count expansion path without any calibration. Required to reach that path, because expanding a raw camera count by h_open alone silently assumes each counted object contributes exactly one angler-hour per hour open — a calibration of 1 that was never measured (GH #158). Under the opt-out the point estimate uses that assumption and the reported SE is NA: the calibration component is present-and-unknown rather than absent, because the correction applies and was simply not measured. It is never 0, which would be indistinguishable from having propagated the calibration's uncertainty and found none. Supplying interviews instead uses the ratio-calibration path, which estimates hours of effort per camera count per stratum and propagates that ratio's variance. Prefer it whenever interview data exist.
variance	Character. Variance method: "taylor" (default) or "replicate".
conf_level	Numeric confidence level. Default 0.95.

Details

- **Ratio calibration** (recommended, when interview data are available): Per-stratum calibration ratios (mean interview effort / mean camera count during the interview period) scale raw camera counts to angler-hours. Variance is estimated via Taylor linearisation or replicate weights.
- **Raw count expansion** (fallback): Camera ingress counts are multiplied by h_open (fishable hours per day). Use when no interview data are available.

Value

A creel_estimates object with columns estimate, se, se_between, se_within, ci_lower, ci_upper, n.

Uncertainty the standard error does not cover

Two cases are reported rather than absorbed, because in both the returned standard error would otherwise understate what is known:

- A stratum with a single paired interview/count day gives its calibration ratio no measurable spread. That variance is unknown rather than zero, so it is carried as NA and the combined standard error and confidence interval are NA too; a warning names the stratum. Add a second matched interview day in that stratum to recover a standard error.
- Counts flagged .imputed by `impute_camera_counts()` enter the estimator as observations. The imputation model's prediction uncertainty is not propagated, and model predictions vary less than real counts, so the between-day component is understated as well. A warning reports how many days were imputed; the standard error is a lower bound.

Within-day variance

When counts arrive through `add_counts(count_time_col = )`, several counts on one day are averaged into a daily mean and the within-day components (`ss_d`, `k_d`) are stored on the design. Both paths of this function read them and report the Rasmussen (1998) within-day term as `se_within`, scaling it by the stratum's calibration ratio on the ratio path and by `h_open` on the raw path.

`se_within` is 0 only when there is genuinely nothing to measure – one count per day, where the component is nil by construction rather than unknown. It was previously reported as a literal 0 in every case, while the measured components sat unread on the design, so a design with real within-day spread received the same standard error as one with none.

One count row per day on the calibration path

Ratio calibration pairs each interview day to that day's camera count, so it requires the counts table to hold exactly one row per day (per stratum). A repeated day is refused rather than averaged: two counts on one date are either sub-period snapshots or a data error, and the estimator cannot tell which. Before this was checked, a repeated date entered both sides of the calibration ratio twice and **moved the point estimate**, not merely the standard error.

If the counts are genuine sub-daily observations, pass `count_time_col` to `add_counts()`, which averages them into one row per day and retains the within-day variance. Otherwise remove the repeated rows. Raw count expansion (`interviews = NULL`) does no pairing and is not subject to this requirement.

Where the calibration estimator comes from

The ratio calibration is a **double-sampling ratio estimator**, applied here to camera calibration by this package. It is not a reproduction of a published fisheries estimator, and no paper in the camera literature derives it in this form.

Within each stratum the estimator forms ρ as a ratio of sums – interview hours over camera counts on the paired days – estimates its variance by the ratio-estimator formula on the paired daily residuals, and applies it to that stratum's first-phase count total, combining the two variances by the delta method. The counts are the first-phase sample and the days carrying interviews are the second phase, which is the structure Cochran (1977) Chapter 12 treats; the ratio's variance is Cochran's eq. 2.46 with the finite-population correction omitted.

The **practice** of calibrating camera counts against paired concurrent creel observations is well established – Hartill et al. (2016), van Poorten et al. (2015), Eckelbecker et al. (2022) – but each of those uses a different estimator: a per-day classification proportion, a hierarchical Bayesian model, and a fitted linear correction respectively. Hartill et al. (2020) is a review of camera monitoring and presents no estimator or variance at all.

In particular this is **not** Hartill et al.'s (2016) ρ . Theirs is the dimensionless proportion of observed boats that were fishing, estimated per day from interviews that are a subsample of the camera's own frame, with a bootstrap variance. The ratio here has units of hours per count, corrects counts to effort rather than classifying them, pools over days within a stratum, and pairs the camera against an independent measurement – a different variance structure, which is why a design-based ratio variance is used rather than a bootstrap.

References

- Cochran, W.G. 1977. Sampling Techniques, 3rd ed. Wiley, New York. Section 2.11 gives the ratio estimator and its estimated variance (eq. 2.46), which is the form used here with the finite-population correction omitted. Chapter 12 covers double sampling, and Section 12.9 (p. 343) the ratio estimator applied to a first-phase total.
- Hartill, B.W., Payne, G.W., Rush, N., and Bian, R. 2016. Bridging the temporal gap: continuous and cost-effective monitoring of dynamic recreational fisheries by web cameras and creel surveys. *Fisheries Research* 183:488-497. doi:10.1016/j.fishres.2016.06.002
- van Poorten, B.T., Carruthers, T.R., Ward, H.G.M., and Varkey, D.A. 2015. Imputing recreational angling effort from time-lapse cameras using an hierarchical Bayesian model. *Fisheries Research* 172:265-273. doi:10.1016/j.fishres.2015.07.032
- Eckelbecker, R.W., Coleman, T.S., and Catalano, M.J. 2022. Incorporating time-lapse digital cameras into creel surveys at three Alabama reservoirs. *North American Journal of Fisheries Management* 42:1349-1358. doi:10.1002/nafm.10828
- Hartill, B.W., Taylor, S.M., Keller, K., and Weltersbach, M.S. 2020. Digital camera monitoring of recreational fishing effort: applications and challenges. *Fish and Fisheries* 21:204-215. doi:10.1111/faf.12413

See Also

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
library(tidycreel)
data(example_camera_counts)
data(example_camera_interviews)

cal <- data.frame(
  date      = unique(example_camera_counts$date),
  day_type  = unique(example_camera_counts[, c("date", "day_type")][["day_type"]])
)
design <- creel_design(cal,
  date = date, strata = day_type,
  survey_type = "camera", camera_mode = "counter"
)

# Filter to operational rows
ops <- example_camera_counts[
  example_camera_counts$camera_status == "operational",
]
design <- add_counts(design, ops)

# Ratio calibration using interview hours. `example_camera_interviews` has no
# party-size column, so this warns and reports an unknown unit: the estimate
```

```
# is in whatever unit `hours_fished` holds, which the package cannot tell.
est <- est_effort_camera(design, interviews = example_camera_interviews)
print(est)

# With party sizes the function does the normalisation itself, so the result
# is angler-hours and is labelled as such.
ints <- example_camera_interviews
ints$party_size <- 2
est_ah <- est_effort_camera(design, interviews = ints, n_angers = "party_size")
print(est_ah)
```

est_effort_camera_mi *Pool camera effort estimates across multiply imputed count data sets*

Description

Estimates camera effort once per completed data set produced by `impute_camera_counts()` with $m > 1$, then combines the results with Rubin's (1987) rules.

This exists because a single completed data set structurally cannot carry the uncertainty introduced by imputing. Inside `survey::svytotal()` a predicted count is indistinguishable from an observed one, so the imputation model's own error is dropped; and predictions are smoother than real counts, so the between-day component shrinks as well. The reported SE is therefore biased downward twice over, and can fall *below* the SE of the same design with the outage days simply deleted — reporting more precision from less information (GH #137).

Usage

```
est_effort_camera_mi(design, imputations, ..., conf_level = 0.95)
```

Arguments

design	A <code>creel_design()</code> object of <code>design_type == "camera"</code> without counts attached. Counts come from imputations, one completed set at a time.
imputations	A camera_imputations object from <code>impute_camera_counts()</code> with $m > 1$.
...	Further arguments passed to <code>est_effort_camera()</code> , such as <code>interviews</code> , <code>h_open</code> , or <code>calibration</code> .
conf_level	Numeric confidence level. Default 0.95.

Details

[Experimental]

Value

A `creel_estimates` object with `method = "camera_mi"`. Its `se_components` names the two halves of the pooled variance as `within_imputation` and `between_imputation`, so a reader can see how much of the uncertainty came from imputing. The per-imputation results are attached as `attr(result, "imputations")`.

The pooled variance

With M completed data sets giving estimates Q_m and variances $U_m = SE_m^2$:

$$\begin{aligned}\bar{Q} &= \frac{1}{M} \sum_m Q_m \\ \bar{U} &= \frac{1}{M} \sum_m U_m \\ B &= \frac{M+1}{M(M-1)} \sum_m (Q_m - \bar{Q})^2 \\ T &= \bar{U} + B\end{aligned}$$

$\bar{U}$ is the **within-imputation** variance — the average of what each completed data set reports, and the only part single imputation can produce. B is the **between-imputation** term, and it is the one that is structurally missing today: it measures how much the estimate moves when the outage days are filled differently, which a single filled data set cannot express at all.

This is the pooling in Afrifa-Yamoah et al. (2020) equation (5). Their $(M+1)/(M(M-1))$ factor is the usual Rubin $(1 + 1/M)$ inflation written over the raw sum of squares rather than the sample variance; the two are the same quantity.

Degrees of freedom use Rubin's classic expression $\nu = (M-1)(1 + \bar{U}/B)^2$, which is finite precisely because $B > 0$.

References

Afrifa-Yamoah, E., Taylor, S.M., Fisher, A., and Mueller, U. 2020. Imputation of missing data from time-lapse cameras used in recreational fishing surveys. *ICES Journal of Marine Science* 77(7-8):2984-2994.

Rubin, D.B. 1987. *Multiple Imputation for Nonresponse in Surveys*. Wiley.

See Also

Other "Estimation": [compare_cpue_estimators\(\)](#), [est_age_distribution\(\)](#), [est_biomass\(\)](#), [est_compliance\(\)](#), [est_length_distribution\(\)](#), [est_mean_age\(\)](#), [est_mean_length\(\)](#), [estimate_catch_rate\(\)](#), [estimate_effort\(\)](#), [estimate_effort_aerial_glm\(\)](#), [estimate_harvest_rate\(\)](#), [estimate_release_rate\(\)](#), [estimate_total_catch\(\)](#), [estimate_total_harvest\(\)](#), [estimate_total_release\(\)](#)

Examples

```
data(example_camera_counts)
data(example_camera_interviews)

cal <- data.frame(
  date      = unique(example_camera_counts$date),
  day_type  = unique(example_camera_counts[, c("date", "day_type")][["day_type"]])
)
design <- creel_design(cal,
  date = date, strata = day_type,
```

```

    survey_type = "camera", camera_mode = "counter"
  )
  # No add_counts() here: each imputation supplies its own completed count
  # series, so attaching one of them first would fix the very thing being
  # varied.
  # Outage days are refilled several times over, so the uncertainty about what
  # the camera missed enters the standard error instead of being assumed away.
  imps <- impute_camera_counts(
    example_camera_counts,
    count_col = "ingress_count",
    strata_col = "day_type",
    m = 5L
  )

  ints <- example_camera_interviews
  ints$party_size <- 2
  est_effort_camera_mi(design, imps, interviews = ints, n_anglers = "party_size")

```

est_length_distribution

Estimate a weighted length distribution from creel interview data

Description

`est_length_distribution()` estimates a pressure-weighted length-frequency distribution from fish length data attached via `add_lengths()`. Unlike `summarize_length_freq()`, which reports raw sample frequencies, `est_length_distribution()` aggregates interview-level bin counts through the internal interview survey design so the result reflects the survey design rather than only the observed sample.

The estimator returns one row per occupied length bin, with weighted totals, standard errors, confidence intervals, and within-group percentages.

Lengths are measured on a **subsample** of the catch, so the bin totals are scaled onto the design-estimated reported catch rather than reporting the subsample itself. See the section below; the call warns whenever it rescales, and aborts when the design carries no total to scale to.

Usage

```

est_length_distribution(
  design,
  type = "catch",
  by = NULL,
  bin_width = 1,
  length_col = NULL,
  variance = "taylor",
  conf_level = 0.95
)

```

Arguments

design	A creel_design object with interviews and lengths attached.
type	Character string indicating which fish to include. One of "catch" (default), "harvest", or "release".
by	Optional tidy selector evaluated against design\$lengths. Common choices include by = species.
bin_width	Positive numeric bin width in the same units as the attached length data. Default 1.
length_col	Optional character column name in design\$lengths to use for the length values. Defaults to the column registered by add_lengths() .
variance	Character string specifying variance estimation method. One of "taylor" (default), "bootstrap", or "jackknife".
conf_level	Numeric confidence level for confidence intervals. Default 0.95.

Value

A data.frame with class `c("creel_length_distribution", "data.frame")` and columns: grouping columns (if any), length_bin (ordered factor), bin_lower, bin_upper, estimate, se, ci_lower, ci_upper, percent, cumulative_percent, and n.

percent and cumulative_percent are shares of the group's estimated total, rounded to one decimal for display; cumulative_percent accumulates the unrounded shares, so it reaches 100 rather than drifting. The exception is a group whose estimated total is zero, where there are no shares to take and both columns are 0 rather than reaching 100.

n is the number of **interviews** contributing at least one measured fish to the group. It is therefore constant across every bin of a group, and is neither a per-bin sample size nor a count of fish.

Two-phase estimation onto the reported catch

Lengths are a second-phase sample: interviews report how many fish were caught, and some subset of those fish get measured. Expanding the measured fish through the interview design alone estimates *the total number of fish that happened to be measured*, which is not the catch — on this package's example data it returns 14 against a reported harvest of 77. Because [est_biomass\(\)](#) multiplies these counts by weight-at-length and calls the result total biomass, the error propagated to a headline number (GH #310).

The estimator is therefore two-phase (double sampling, Cochran 1977 §12.9 — the same structure used for the camera calibration ratio). For bin h :

$$\hat{p}_h = \hat{N}_h^{\text{meas}} / \sum_j \hat{N}_j^{\text{meas}}, \quad \hat{N}_h = \hat{p}_h \hat{T}$$

where $\hat{T}$ is the design-estimated reported total for the group. Both parts come from a single `svytotal()` call, so the covariance between a bin and the reported total is estimated rather than assumed away, and the standard error is the delta method over that joint covariance.

What this changes: estimate, se and the confidence bounds now describe the reported catch. percent and cumulative_percent are unchanged — a share is invariant to the subsample size, which is why the *shape* of the distribution was always correct and only its *level* was not.

Where $\hat{T}$ comes from depends on the grouping. A species group can only be scaled by that species' own total, which lives in the table attached by `add_catch()`; grouping by species without it is refused rather than scaled by the all-species total. Any other grouping uses the interview-level column (catch or harvest from `add_interviews()`), with release implied as caught - harvested so that harvest and release sum back to catch.

See Also

Other "Estimation": `compare_cpue_estimators()`, `est_age_distribution()`, `est_biomass()`, `est_compliance()`, `est_effort_camera_mi()`, `est_mean_age()`, `est_mean_length()`, `estimate_catch_rate()`, `estimate_effort()`, `estimate_effort_aerial_glm()`, `estimate_harvest_rate()`, `estimate_release_rate()`, `estimate_total_catch()`, `estimate_total_harvest()`, `estimate_total_release()`

Examples

```
data(example_calendar)
data(example_interviews)
data(example_lengths)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
# Species catch is required to group by species: the totals are scaled onto
# the reported catch, and only this table records it per species.
design <- add_catch(design, example_catch,
  catch_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  count = count,
  catch_type = catch_type
)
design <- add_lengths(design, example_lengths,
  length_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  length = length,
  length_type = length_type,
  count = count,
  release_format = "binned"
)

est_length_distribution(design, by = species, bin_width = 25)
```

est_mean_age

*Estimate design-weighted mean age from a creel age distribution***Description**

est_mean_age() computes the pressure-weighted mean fish age from a [est_age_distribution\(\)](#) object using the ratio estimator $\bar{A} = \sum_a a \hat{N}_a / \sum_a \hat{N}_a$, with delta-method standard error.

Usage

```
est_mean_age(ad, conf_level = NULL)
```

Arguments

ad A creel_age_distribution object from [est_age_distribution\(\)](#).

conf_level Numeric confidence level for confidence intervals. Defaults to the level stored in ad (usually 0.95).

Details

Each integer age a contributes its survey-weighted count $\hat{N}_a$. Mean age is the ratio of total age-weighted count to total count:

$$\bar{A} = \frac{\sum_a a \hat{N}_a}{\hat{N}}$$

Variance is propagated via the delta method for a ratio estimator, treating cross-class covariances as zero:

$$\widehat{\text{Var}}(\bar{A}) \approx \frac{1}{\hat{N}^2} \sum_a (a - \bar{A})^2 \widehat{\text{SE}}_a^2$$

Value

A data.frame with class c("creel_mean_age", "data.frame") and columns: grouping columns (if any), mean_age, mean_age_se, mean_age_ci_lower, mean_age_ci_upper. Rows where the total estimated fish is zero or negative return NA for all numeric columns with a warning.

See Also

Other "Estimation": [compare_cpue_estimators\(\)](#), [est_age_distribution\(\)](#), [est_biomass\(\)](#), [est_compliance\(\)](#), [est_effort_camera_mi\(\)](#), [est_length_distribution\(\)](#), [est_mean_length\(\)](#), [estimate_catch_rate\(\)](#), [estimate_effort\(\)](#), [estimate_effort_aerial_glm\(\)](#), [estimate_harvest_rate\(\)](#), [estimate_release_rate\(\)](#), [estimate_total_catch\(\)](#), [estimate_total_harvest\(\)](#), [estimate_total_release\(\)](#)

Examples

```

data(example_calendar)
data(example_interviews)
data(example_ages)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
# Species catch is required to group by species: the totals are scaled onto
# the reported catch, and only this table records it per species.
design <- add_catch(design, example_catch,
  catch_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  count = count,
  catch_type = catch_type
)
design <- add_ages(design, example_ages,
  age_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  age = age,
  age_type = age_type
)

ad <- est_age_distribution(design, by = species)
est_mean_age(ad)

```

est_mean_length

*Estimate design-weighted mean length from a creel length distribution***Description**

est_mean_length() computes the pressure-weighted mean fish length from a [est_length_distribution\(\)](#) object using the ratio estimator $\bar{L} = \sum_h L_h \hat{N}_h / \sum_h \hat{N}_h$, with delta-method standard error.

Usage

```
est_mean_length(ld, conf_level = NULL)
```

Arguments

ld	A creel_length_distribution object from est_length_distribution() .
conf_level	Numeric confidence level for confidence intervals. Defaults to the level stored in ld (usually 0.95).

Details

Bin midpoints $L_h = (\text{bin_lower} + \text{bin_upper})/2$ serve as representative lengths. Mean length is the ratio of total length-weighted count to total count:

$$\bar{L} = \frac{\sum_h L_h \hat{N}_h}{\hat{N}}$$

Variance is propagated via the delta method for a ratio estimator, using the bins' full covariance matrix Σ :

$$\widehat{\text{Var}}(\bar{L}) = \frac{1}{\hat{N}^2} w' \Sigma w, \quad w_h = L_h - \bar{L}$$

Earlier versions treated the cross-bin covariances as zero, which under-estimated the standard error. If Σ is unavailable the independence form is used and a warning says so.

Value

A data.frame with class `c("creel_mean_length", "data.frame")` and columns: grouping columns (if any), `mean_length`, `mean_length_se`, `mean_length_ci_lower`, `mean_length_ci_upper`. Rows where the total estimated fish is zero or negative return NA for all numeric columns with a warning.

See Also

Other "Estimation": [compare_cpue_estimators\(\)](#), [est_age_distribution\(\)](#), [est_biomass\(\)](#), [est_compliance\(\)](#), [est_effort_camera_mi\(\)](#), [est_length_distribution\(\)](#), [est_mean_age\(\)](#), [estimate_catch_rate\(\)](#), [estimate_effort\(\)](#), [estimate_effort_aerial_glmm\(\)](#), [estimate_harvest_rate\(\)](#), [estimate_release_rate\(\)](#), [estimate_total_catch\(\)](#), [estimate_total_harvest\(\)](#), [estimate_total_release\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_lengths)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
# Species catch is required to group by species: the totals are scaled onto
# the reported catch, and only this table records it per species.
design <- add_catch(design, example_catch,
  catch_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  count = count,
  catch_type = catch_type
)
design <- add_lengths(design, example_lengths,
  length_uid = interview_id,
```

```

    interview_uid = interview_id,
    species = species,
    length = length,
    length_type = length_type,
    count = count,
    release_format = "binned"
  )

ld <- est_length_distribution(design, by = species, bin_width = 25)
est_mean_length(ld)

```

example_aerial_counts *Example aerial angler count dataset*

Description

A dataset of instantaneous angler counts from aerial overflights of a Nebraska reservoir, used to demonstrate aerial survey effort estimation. Contains 16 rows representing one overflight per sampling day across an 8-week summer season (June-July 2024). Weekday and weekend counts vary realistically to produce non-trivial between-day variance in the effort estimate.

Usage

```
example_aerial_counts
```

Format

A data frame with 16 rows and 3 variables:

date Survey date (Date class), June-July 2024.

day_type Day type stratum: "weekday" or "weekend".

n_anglers Instantaneous angler count from one aerial overflight (integer). Weekday counts range 15-40; weekend counts range 40-80.

Source

Simulated for package documentation.

See Also

[example_aerial_interviews](#) for matching interview data, [creel_design\(\)](#), [add_counts\(\)](#), [estimate_effort\(\)](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_glm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_f](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```

data(example_aerial_counts)
head(example_aerial_counts)

# Build a calendar from count dates and construct an aerial design
aerial_cal <- data.frame(
  date = example_aerial_counts$date,
  day_type = example_aerial_counts$day_type,
  stringsAsFactors = FALSE
)
design <- creel_design(
  aerial_cal,
  date = date,
  strata = day_type,
  survey_type = "aerial",
  visibility_correction = "none",
  angler_ratio = 1,
  angler_ratio_se = 0,
  h_open = 14
)
print(design)

```

example_aerial_glmm_counts

Example multi-flight aerial count data for GLMM effort estimation

Description

Simulated instantaneous angler counts from aerial overflights of a Nebraska reservoir, designed to demonstrate GLMM-based effort estimation following Askey (2018). Contains 48 rows: 12 survey days with 4 overflights per day at fixed hours (07:00, 10:00, 13:00, 16:00). Counts follow a diurnal curve (low at dawn, peak mid-morning, lower in afternoon) with day-level Poisson variability and a day random intercept.

Usage

```
example_aerial_glmm_counts
```

Format

A data frame with 48 rows and 4 columns:

date Survey date (Date class), 12 days spaced 3 days apart starting 2024-06-03.

day_type Day type stratum: "weekday" or "weekend", derived from the calendar date.

n_angers Instantaneous angler count from one aerial overflight (integer). Follows a diurnal curve with day-level random effects.

time_of_flight Hour of the aerial overflight (numeric). One of 7.0, 10.0, 13.0, or 16.0.

Source

Simulated data following Askey (2018) NAJFM doi:10.1002/nafm.10010.

References

Askey, P.J., Ward, H., Godin, T., Boucher, M., and Northrup, S. (2018). Angler effort estimates from instantaneous aerial counts: use of high-frequency time-lapse camera data to inform model-based estimators. *North American Journal of Fisheries Management*, 38, 194-209. doi:10.1002/nafm.10010

See Also

[example_aerial_counts](#) for the simple single-flight dataset, [estimate_effort_aerial_glmm\(\)](#) for the GLMM-based estimator, [creel_design\(\)](#), [add_counts\(\)](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_f](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_aerial_glmm_counts)
head(example_aerial_glmm_counts)

# The workflow below fits a GLMM, so it needs lme4 (a Suggests).
if (rlang::is_installed("lme4")) {
  # Build an aerial design and estimate effort with GLMM correction
  aerial_cal <- data.frame(
    date = unique(example_aerial_glmm_counts$date),
    day_type = unique(example_aerial_glmm_counts[, c("date", "day_type")][["day_type"]],
    stringsAsFactors = FALSE
  )
  design <- creel_design(
    aerial_cal,
    date = date,
    strata = day_type,
    survey_type = "aerial",
    visibility_correction = "none",
    angler_ratio = 1,
    angler_ratio_se = 0,
    h_open = 14
  )
  design <- add_counts(design, example_aerial_glmm_counts, count_col = n_anglers)
  result <- estimate_effort_aerial_glmm(design, time_col = time_of_flight)
  print(result)
}
```

example_aerial_interviews

Example angler interview data for aerial creel survey

Description

Angler interview data for an aerial creel survey at a Nebraska reservoir. Contains 48 interviews across 16 sampling days in June-July 2024, with 3 interviews per sampling day. Anglers target walleye and bass. All interviews are complete trips. Dates match [example_aerial_counts](#).

Usage

```
example_aerial_interviews
```

Format

A data frame with 48 rows and 8 variables:

date Interview date (Date class), June-July 2024.

day_type Day type stratum: "weekday" or "weekend".

trip_status Trip completion status: "complete" for all 48 interviews.

hours_fished Numeric trip duration in hours (range 1.0-5.0). This column feeds the mean trip duration ($\bar{L}$) used in [estimate_catch_rate](#).

walleye_catch Integer total walleye caught (kept + released).

walleye_kept Integer walleye harvested; always $\leq$ walleye_catch.

bass_catch Integer total bass caught (kept + released).

bass_kept Integer bass harvested; always $\leq$ bass_catch.

Source

Simulated for package documentation.

See Also

[example_aerial_counts](#) for matching count data, [creel_design\(\)](#), [add_interviews\(\)](#), [estimate_catch_rate\(\)](#), [estimate_total_catch\(\)](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glm_counts](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_aerial_counts)
data(example_aerial_interviews)

# Build an aerial design and add interview data
aerial_cal <- data.frame(
  date = example_aerial_counts$date,
  day_type = example_aerial_counts$day_type,
  stringsAsFactors = FALSE
)
design <- creel_design(
  aerial_cal,
  date = date,
  strata = day_type,
  survey_type = "aerial",
  visibility_correction = "none",
  angler_ratio = 1,
  angler_ratio_se = 0,
  h_open = 14
)
design <- add_counts(design, example_aerial_counts)
design <- suppressWarnings(add_interviews(
  design,
  example_aerial_interviews,
  catch = walleye_catch,
  effort = hours_fished,
  trip_status = trip_status
))
suppressWarnings(estimate_catch_rate(design))
```

example_ages

Example fish age data for creel estimation

Description

A small set of individual fish age records (one row per aged fish) linked to [example_interviews](#). Suitable for use with [add_ages](#).

Usage

```
example_ages
```

Format

A data frame with 18 rows and 4 columns:

interview_id Integer interview identifier. Foreign key to `example_interviews$interview_id`.
species Character. Species name: "walleye", "bass", or "panfish".

age Integer. Estimated age in years (0-6). Each row is a single aged fish.

age_type Character. Fish fate: "harvest" or "release".

Source

Simulated data for package examples.

See Also

[example_interviews](#), [example_lengths](#), [add_ages](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glmm_counts](#), [example_aerial_interviews](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_ages)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, trip_duration = trip_duration
)
design <- add_ages(design, example_ages,
  age_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  age = age,
  age_type = age_type
)
print(design)
```

example_calendar	<i>Example calendar data for creel survey</i>
------------------	---

Description

A sample survey calendar dataset demonstrating the structure required for [creel_design\(\)](#). Contains 14 days (June 1-14, 2024) with weekday/weekend strata, representing a two-week survey period.

Usage

```
example_calendar
```

Format

A data frame with 14 rows and 2 columns:

date Survey date (Date class), June 1-14, 2024

day_type Day type stratum: "weekday" or "weekend"

Source

Simulated data for package examples

See Also

[example_counts](#) for matching count data, [creel_design\(\)](#) to create a design from calendar data

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
# Load and inspect
data(example_calendar)
head(example_calendar)

# Create a creel design
design <- creel_design(example_calendar, date = date, strata = day_type)
print(design)
```

`example_camera_counts` *Example camera counts dataset (counter mode)*

Description

A dataset of daily ingress counts from a remote camera at a boat launch. Contains 10 rows covering non-consecutive sampling days in June 2024. Includes one row with `camera_status = "battery_failure"` and `ingress_count = NA` demonstrating informative gap handling.

Usage

```
example_camera_counts
```

Format

A data frame with 10 rows and 4 variables:

date Survey date (Date class), non-consecutive days in June 2024.

day_type Day type stratum: "weekday" or "weekend".

ingress_count Daily ingress angler count (integer). NA when the camera was not operational.

camera_status Camera operational status. One of "operational", "battery_failure", "memory_full", or "occlusion".

Source

Simulated for package documentation.

See Also

[example_camera_timestamps](#), [example_camera_interviews](#), [creel_design\(\)](#), [add_counts\(\)](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_camera_counts)
head(example_camera_counts)

# Filter to operational rows before adding to a camera design
data(example_calendar)
design <- creel_design(
  example_calendar,
  date = date, strata = day_type,
  survey_type = "camera",
  camera_mode = "counter"
)
counts_clean <- subset(example_camera_counts, camera_status == "operational")
design <- suppressWarnings(add_counts(design, counts_clean))
```

example_camera_interviews

Example interview data for camera-monitored creel survey

Description

Angler interview data for a summer creel survey at a camera-monitored boat launch. Contains 40 interviews across 8 sampling days in June 2024, targeting walleye and bass. All interviews are complete trips. Dates match the date range in [example_camera_counts](#).

Usage

```
example_camera_interviews
```

Format

A data frame with 40 rows and 8 variables:

date Interview date (Date class), June 2024.

day_type Day type stratum: "weekday" or "weekend".

trip_status Trip completion status: "complete" for all 40 interviews.

hours_fished Numeric fishing effort in hours (range 0.5-5.0).

walleye Integer total walleye caught (kept + released).

walleye_kept Integer walleye harvested; always <= walleye.

bass Integer total bass caught (kept + released).

bass_kept Integer bass harvested; always <= bass.

Source

Simulated for package documentation.

See Also

[example_camera_counts](#), [example_camera_timestamps](#), [add_interviews\(\)](#), [estimate_catch_rate\(\)](#), [estimate_total_catch\(\)](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_camera_counts)
data(example_camera_interviews)

# Build a calendar that spans all camera dataset dates
cam_dates <- sort(unique(c(
  example_camera_counts$date,
  example_camera_interviews$date
)))
cam_cal <- data.frame(
  date = cam_dates,
  day_type = ifelse(
    weekdays(cam_dates) %in% c("Saturday", "Sunday"),
    "weekend", "weekday"
  ),
  stringsAsFactors = FALSE
)
```

```
design <- creel_design(  
  cam_cal,  
  date = date, strata = day_type,  
  survey_type = "camera",  
  camera_mode = "counter"  
)  
counts_clean <- subset(example_camera_counts, camera_status == "operational")  
design <- suppressWarnings(add_counts(design, counts_clean))  
design <- suppressWarnings(add_interviews(  
  design, example_camera_interviews,  
  catch = walleye, effort = hours_fished, trip_status = trip_status  
)  
)  
suppressWarnings(estimate_catch_rate(design))
```

example_camera_timestamps

Example camera timestamps dataset (ingress-egress mode)

Description

A dataset of raw ingress and egress timestamps recorded by a remote camera at a boat launch. Contains 14 rows spanning 4 sampling days in June 2024 (3-4 anglers per day). Suitable for use with [preprocess_camera_timestamps](#). One row has a trip duration greater than 8 hours (an unusually long fishing day); all other durations are between 1.5 and 5.5 hours.

Usage

example_camera_timestamps

Format

A data frame with 14 rows and 4 variables:

date Survey date (Date class), June 2024.

day_type Day type stratum: "weekday" or "weekend".

ingress_time Angler arrival time (POSIXct, America/Chicago timezone).

egress_time Angler departure time (POSIXct, America/Chicago timezone). Always later than ingress_time.

Source

Simulated for package documentation.

See Also

`example_camera_counts`, `example_camera_interviews`, `preprocess_camera_timestamps()`

Other "Example Datasets": `creel_counts_toy`, `creel_interviews_toy`, `example_aerial_counts`, `example_aerial_glm_counts`, `example_aerial_interviews`, `example_ages`, `example_calendar`, `example_camera_counts`, `example_camera_interviews`, `example_catch`, `example_counts`, `example_ice_interviews`, `example_ice_sampling_frame`, `example_interviews`, `example_lengths`, `example_sections_calendar`, `example_sections_counts`, `example_sections_interviews`

Examples

```
data(example_camera_timestamps)
head(example_camera_timestamps)

# Preprocess to daily effort hours
daily_effort <- preprocess_camera_timestamps(
  example_camera_timestamps,
  date_col = date,
  ingress_col = ingress_time,
  egress_col = egress_time
)
head(daily_effort)
```

example_catch	<i>Example species catch data for creel survey</i>
---------------	--

Description

Long-format species-level catch data linked to [example_interviews](#). Contains catch, harvest, and release counts per species per interview for 12 of the 22 interviews. Interviews with zero total catch have no rows in this dataset (zero-catch anglers are represented by absence).

Usage

```
example_catch
```

Format

- A data frame with columns:
- interview_id** Integer, foreign key to [example_interviews](#)\$interview_id
 - species** Character species name: "walleye", "bass", or "panfish"
 - count** Integer fish count for this species and catch type
 - catch_type** Character catch disposition: "caught" (total observed), "harvested" (kept), or "released"

Source

Simulated data for package examples

See Also

[example_interviews](#) for the corresponding interview-level data, [prep_interview_catch\(\)](#) to standardize species catch, and [add_catch\(\)](#) to attach species catch to a design

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glmm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_catch)

design <- creel_design(example_calendar, date = date, strata = day_type)
interviews_ready <- prep_interviews_trips(
  example_interviews,
  date = date,
  interview_uid = interview_id,
  effort_hours = hours_fished,
  trip_status = trip_status,
  trip_duration = trip_duration,
  catch_total = catch_total,
  harvest_total = catch_kept
)
design <- add_interviews(design, interviews_ready,
  catch = catch_total,
  effort = effort_hours,
  harvest = harvest_total,
  trip_status = trip_status,
  trip_duration = trip_duration
)

catch_ready <- prep_interview_catch(example_catch,
  interview_uid = interview_id,
  species = species,
  count = count,
  catch_type = catch_type
)
design <- add_catch(design, catch_ready,
  catch_uid = interview_uid,
  interview_uid = interview_uid,
  species = species,
  count = count,
  catch_type = catch_type
)
print(design)
```

example_counts	<i>Example count data for creel survey</i>
----------------	--

Description

Sample daily effort observations matching [example_calendar](#). One row per survey date, suitable for use with [add_counts\(\)](#) and [estimate_effort\(\)](#).

Usage

```
example_counts
```

Format

A data frame with 14 rows and 3 columns:

date Survey date (Date class), matching [example_calendar](#) dates

day_type Day type stratum: "weekday" or "weekend", matching calendar

effort_hours Numeric angler-hours observed on the survey date

Details

The effort column holds angler-hours, not raw angler counts. [estimate_effort\(\)](#) expands whichever column it is given to the season without converting units, so a design built on these data reports angler-hours. Supplying raw instantaneous counts instead would give a total in angler-days.

Source

Simulated data for package examples

See Also

[example_calendar](#) for matching calendar data, [add_counts\(\)](#) to attach counts to a design

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glmm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
# Load and use with a creel design
data(example_calendar)
data(example_counts)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
result <- estimate_effort(design)
```

```
print(result)
```

example_ice_interviews

Example interview data for ice fishing creel survey

Description

Angler interview data for an ice fishing creel survey at Lake McConaughy, Nebraska. Contains 72 interviews across 12 sampling days in January-February 2024. Anglers fish from both open-air setups and enclosed dark-house shelters, targeting walleye and yellow perch. Dates match [example_ice_sampling_frame](#).

Usage

```
example_ice_interviews
```

Format

A data frame with 72 rows and 11 columns:

- date** Interview date (Date class), matching [example_ice_sampling_frame](#)
- n_counted** Integer total number of angler parties counted at the access point during the sampling period
- n_interviewed** Integer number of parties actually interviewed; always <= n_counted
- hours_on_ice** Numeric hours the angler party was physically on the ice (total time-on-ice effort)
- active_fishing_hours** Numeric hours spent actively fishing, excluding travel, setup, and breaks; always <= hours_on_ice
- walleye_catch** Integer total walleye caught (kept + released)
- perch_catch** Integer total yellow perch caught (kept + released)
- walleye_kept** Integer walleye harvested; always <= walleye_catch
- perch_kept** Integer yellow perch harvested; always <= perch_catch
- trip_status** Character trip completion status: "complete" or "incomplete"
- shelter_mode** Character shelter type used by the angler party: "open" (no shelter) or "dark_house" (enclosed shelter). Used to stratify effort estimates by shelter type.

Source

Simulated data based on Nebraska ice fishing survey protocols.

See Also

[example_ice_sampling_frame](#) for the matching sampling frame, [creel_design\(\)](#), [add_interviews\(\)](#), [estimate_effort\(\)](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glmm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_ice_sampling_frame)
data(example_ice_interviews)

# Build an ice fishing design with scalar period sampling probability
design <- creel_design(
  example_ice_sampling_frame,
  date = date,
  strata = day_type,
  survey_type = "ice",
  effort_type = "time_on_ice",
  p_period = 0.5
)

design <- suppressMessages(add_interviews(
  design,
  example_ice_interviews,
  catch = walleye_catch,
  effort = hours_on_ice,
  harvest = walleye_kept,
  trip_status = trip_status,
  n_counted = n_counted,
  n_interviewed = n_interviewed
))
suppressWarnings(estimate_effort(design))
```

example_ice_sampling_frame

Example sampling frame for ice fishing creel survey

Description

A minimal sampling frame for a Nebraska ice fishing creel survey at Lake McConaughy. Contains 12 weekend sampling days across January-February 2024. Ice fishing surveys are a degenerate bus-route design where all access points are sampled with certainty ($p_{\text{site}} = 1.0$), so only the period sampling probability (p_{period}) is specified.

Usage

```
example_ice_sampling_frame
```

Format

A data frame with 12 rows and 3 columns:

date Survey date (Date class), January-February 2024

day_type Day type stratum: "weekday" or "weekend"

p_period Numeric period sampling probability in (0, 1]. The probability that a given period is included in the sample.

Source

Simulated data based on Nebraska ice fishing survey protocols.

See Also

[example_ice_interviews](#) for matching interview data, [creel_design\(\)](#) for ice survey design construction

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_ice_sampling_frame)
head(example_ice_sampling_frame)

# Build an ice fishing design with scalar period sampling probability
design <- creel_design(
  example_ice_sampling_frame,
  date = date,
  strata = day_type,
  survey_type = "ice",
  effort_type = "time_on_ice",
  p_period = 0.5
)
print(design)
```

example_interviews	<i>Example interview data for creel survey</i>
--------------------	--

Description

Sample angler interview data demonstrating the structure required for `add_interviews()`. Contains 22 interviews from June 1-14, 2024, matching the `example_calendar` date range. Each row represents one angler interview with catch, harvest, effort, trip metadata, and extended interview attributes added in v0.5.0.

Usage

```
example_interviews
```

Format

A data frame with 22 rows and 12 columns:

date Interview date (Date class), matching `example_calendar` dates

hours_fished Numeric fishing effort in hours

catch_total Integer total fish caught (kept + released)

catch_kept Integer fish kept (harvest), always $\leq$ catch_total

trip_status Character trip completion status ("complete" or "incomplete")

trip_duration Numeric trip duration in hours

interview_id Integer interview identifier (1 to 22), primary join key for `add_catch()` and future species-level data functions

angler_type Angler party type: "bank" or "boat"

angler_method Fishing method: "bait", "artificial", or "fly"

species_sought Primary target species: "walleye", "bass", or "panfish"

n_anglers Integer number of anglers in party (1 to 4)

refused Logical flag indicating a refused interview (FALSE for all 22 accepted interviews)

Source

Simulated data for package examples

See Also

`example_calendar` for matching calendar data, `example_catch` for species-level catch data, `prep_interviews_trips()` to standardize interview rows, `add_interviews()` to attach interviews to a design

Other "Example Datasets": `creel_counts_toy`, `creel_interviews_toy`, `example_aerial_counts`, `example_aerial_glmm_counts`, `example_aerial_interviews`, `example_ages`, `example_calendar`, `example_camera_counts`, `example_camera_interviews`, `example_camera_timestamps`, `example_catch`, `example_counts`, `example_ice_interviews`, `example_ice_sampling_frame`, `example_lengths`, `example_sections_calendar`, `example_sections_counts`, `example_sections_interviews`

Examples

```
# Load and use with a creel design
data(example_calendar)
data(example_interviews)

design <- creel_design(example_calendar, date = date, strata = day_type)

interviews_ready <- prep_interviews_trips(
  example_interviews,
  date = date,
  interview_uid = interview_id,
  effort_hours = hours_fished,
  trip_status = trip_status,
  trip_duration = trip_duration,
  catch_total = catch_total,
  harvest_total = catch_kept,
  angler_type = angler_type,
  angler_method = angler_method,
  species_sought = species_sought,
  n_anglers = n_anglers,
  refused = refused
)

design <- add_interviews(design, interviews_ready,
  catch = catch_total,
  effort = effort_hours,
  harvest = harvest_total,
  trip_status = trip_status,
  trip_duration = trip_duration,
  angler_type = angler_type,
  angler_method = angler_method,
  species_sought = species_sought,
  n_anglers = n_anglers,
  refused = refused
)
print(design)
```

example_lengths

*Example fish length data for creel survey***Description**

Mixed-format length data containing individual harvest measurements (numeric, in mm) and binned release counts (character bin labels) linked to [example_interviews](#). Suitable for use with [add_lengths](#).

Usage

```
example_lengths
```

Format

A data frame with 20 rows and 5 columns:

interview_id Integer interview identifier. Foreign key to `example_interviews$interview_id`.

species Character. Species name: "walleye", "bass", or "panfish".

length Character. For harvest rows, a numeric length in mm (stored as character due to mixed column). For release rows, a bin label such as "300-350".

length_type Character. Measurement fate: "harvest" or "release".

count Integer. NA_integer_ for harvest rows (individual measurements); positive integer count for release rows (binned format).

Source

Simulated data for package examples.

See Also

[example_interviews](#), [example_catch](#), [add_lengths](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glmm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_sections_calendar](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_lengths)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, trip_duration = trip_duration
)
design <- add_lengths(design, example_lengths,
  length_uid = interview_id,
  interview_uid = interview_id,
  species = species,
  length = length,
  length_type = length_type,
  count = count,
  release_format = "binned"
)
print(design)
```

`example_sections_calendar`*Example calendar for spatially stratified creel survey*

Description

A 12-day survey calendar used to demonstrate the spatially stratified workflow with [add_sections\(\)](#). Contains 6 weekdays and 6 weekends from June 2024. Matches the date range of [example_sections_counts](#) and [example_sections_interviews](#).

Usage

```
example_sections_calendar
```

Format

A data frame with 12 rows and 2 columns:

date Survey date (Date class), June 2024

day_type Day type stratum: "weekday" or "weekend"

Source

Simulated data for package examples

See Also

[example_sections_counts](#), [example_sections_interviews](#), [add_sections\(\)](#), [creel_design\(\)](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_counts](#), [example_sections_interviews](#)

Examples

```
data(example_sections_calendar)
head(example_sections_calendar)
```

```
design <- creel_design(example_sections_calendar, date = date, strata = day_type)
print(design)
```

example_sections_counts

Example effort counts for spatially stratified creel survey

Description

Daily effort observations for a 3-section lake (North, Central, South) covering 12 survey dates. Each section has one row per date (36 rows total). Effort varies materially by section: Central has the highest angler traffic, South the lowest. Use with [add_sections\(\)](#) and [add_counts\(\)](#).

Usage

example_sections_counts

Format

A data frame with 36 rows and 4 columns:

date Survey date (Date class), matching [example_sections_calendar](#)

day_type Day type stratum: "weekday" or "weekend"

section Section identifier: "North", "Central", or "South"

effort_hours Numeric angler-hours observed on the section for that date

Details

As with [example_counts](#), the effort column holds angler-hours rather than raw angler counts; see that dataset for why the distinction matters to [estimate_effort\(\)](#).

Source

Simulated data for package examples

See Also

[example_sections_calendar](#), [example_sections_interviews](#), [add_counts\(\)](#), [add_sections\(\)](#), [estimate_effort\(\)](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glmm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_interviews](#)

Examples

```
data(example_sections_calendar)
data(example_sections_counts)

sections_df <- data.frame(
  section = c("North", "Central", "South"),
  stringsAsFactors = FALSE
)
design <- creel_design(example_sections_calendar, date = date, strata = day_type)
design <- add_sections(design, sections_df, section_col = section)
design <- suppressWarnings(add_counts(design, example_sections_counts))
estimate_effort(design)
```

example_sections_interviews

Example interview data for spatially stratified creel survey

Description

Angler interview data for a 3-section lake (North, Central, South) with 9 interviews per section (27 total). Catch rates differ materially across sections: South has approximately 2.5x the catch rate of North, making this dataset suitable for demonstrating spatially stratified estimation. The `catch_kept` column enables `estimate_total_harvest()` in addition to `estimate_catch_rate()` and `estimate_total_catch()`.

Usage

```
example_sections_interviews
```

Format

A data frame with 27 rows and 9 columns:

date Interview date (Date class), matching `example_sections_calendar`

day_type Day type stratum: "weekday" or "weekend"

section Section identifier: "North", "Central", or "South"

catch_total Integer total fish caught per interview

catch_kept Integer fish harvested (kept); always $\leq$ `catch_total`

hours_fished Numeric fishing effort in hours

trip_status Character trip completion status; "complete" for all 27 interviews

trip_duration Numeric trip duration in hours

interview_id Integer interview identifier (1 to 27)

Source

Simulated data for package examples

See Also

[example_sections_calendar](#), [example_sections_counts](#), [add_interviews\(\)](#), [estimate_catch_rate\(\)](#), [estimate_total_catch\(\)](#)

Other "Example Datasets": [creel_counts_toy](#), [creel_interviews_toy](#), [example_aerial_counts](#), [example_aerial_glmm_counts](#), [example_aerial_interviews](#), [example_ages](#), [example_calendar](#), [example_camera_counts](#), [example_camera_interviews](#), [example_camera_timestamps](#), [example_catch](#), [example_counts](#), [example_ice_interviews](#), [example_ice_sampling_frame](#), [example_interviews](#), [example_lengths](#), [example_sections_calendar](#), [example_sections_counts](#)

Examples

```
data(example_sections_calendar)
data(example_sections_counts)
data(example_sections_interviews)

sections_df <- data.frame(
  section = c("North", "Central", "South"),
  stringsAsFactors = FALSE
)
design <- creel_design(example_sections_calendar, date = date, strata = day_type)
design <- add_sections(design, sections_df, section_col = section)
design <- suppressWarnings(add_counts(design, example_sections_counts))
design <- suppressWarnings(add_interviews(design, example_sections_interviews,
  catch = catch_total, effort = hours_fished,
  harvest = catch_kept,
  trip_status = trip_status, trip_duration = trip_duration
))
estimate_total_catch(design, aggregate_sections = TRUE)
```

flag_outliers

Flag outliers in a creel interview data column

Description

`flag_outliers()` identifies extreme values in a numeric column of a data frame using Tukey's IQR fence method. Flagged rows are annotated with `is_outlier`, `outlier_reason`, `fence_low`, and `fence_high` columns. A cli summary of flagged rows is emitted.

Usage

```
flag_outliers(data, col, k = 1.5, na.rm = TRUE)
```

Arguments

data	A data.frame containing the column to check.
col	Bare column name (unquoted) to check for outliers.
k	Numeric IQR multiplier (default: 1.5). Larger values produce wider fences and fewer flags. Tukey's standard values are 1.5 (mild outliers) and 3.0 (extreme outliers).
na.rm	Logical. Remove NA values before computing quantiles (default: TRUE).

Details

Method: Tukey's IQR fence.

$$\text{fence_low} = Q_1 - k \times IQR$$

$$\text{fence_high} = Q_3 + k \times IQR$$

Values below fence_low or above fence_high are flagged. When $n < 4$, there is insufficient data to estimate the IQR reliably; fences are set to NA and no rows are flagged.

Value

The input data with four additional columns appended:

is_outlier	Logical — TRUE if the row is outside the fence.
outlier_reason	Character — brief description of why it is flagged (e.g. "above fence_high (23.5)"), or "" if not flagged.
fence_low	Numeric — lower fence value (same for all rows). NA when $n < 4$.
fence_high	Numeric — upper fence value (same for all rows). NA when $n < 4$.

See Also

[add_interviews\(\)](#), [estimate_catch_rate\(\)](#)

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary_creel_estimates\(\)](#), [tidy_creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
df <- data.frame(
  interview_id = 1:8,
  effort = c(1.0, 1.5, 2.0, 1.8, 1.2, 1.9, 2.1, 15.0)
)
flag_outliers(df, col = effort)
```

format.creel_completeness_report

Format creel_completeness_report for printing

Description

Format creel_completeness_report for printing

Usage

```
## S3 method for class 'creel_completeness_report'  
format(x, ...)
```

Arguments

x	A creel_completeness_report object
...	Additional arguments (currently ignored)

Value

Character vector with formatted output

format.creel_design *Format a creel_design object*

Description

Format a creel_design object

Usage

```
## S3 method for class 'creel_design'  
format(x, ...)
```

Arguments

x	A creel_design object
...	Additional arguments (ignored)

Value

A character vector with the formatted output

`format.creel_design_report`*Format creel_design_report for printing*

Description

Format creel_design_report for printing

Usage

```
## S3 method for class 'creel_design_report'  
format(x, ...)
```

Arguments

x	A creel_design_report object
...	Additional arguments (currently ignored)

Value

Character vector with formatted output

`format.creel_estimates`*Format creel_estimates for printing*

Description

Format creel_estimates for printing

Usage

```
## S3 method for class 'creel_estimates'  
format(x, ...)
```

Arguments

x	A creel_estimates object
...	Additional arguments (currently ignored)

Value

Character vector with formatted output

format.creel_estimates_diagnostic

Format creel_estimates_diagnostic for printing

Description

Format creel_estimates_diagnostic for printing

Usage

```
## S3 method for class 'creel_estimates_diagnostic'  
format(x, ...)
```

Arguments

x	A creel_estimates_diagnostic object
...	Additional arguments (currently ignored)

Value

Character vector with formatted output

format.creel_estimates_mor

Format creel_estimates_mor for printing

Description

Format creel_estimates_mor for printing

Usage

```
## S3 method for class 'creel_estimates_mor'  
format(x, ...)
```

Arguments

x	A creel_estimates_mor object
...	Additional arguments (currently ignored)

Value

Character vector with formatted output

format.creel_schedule *Format a creel_schedule for console printing*

Description

Produces a human-readable ASCII monthly calendar grid for a creel_schedule object. Each sampled date shows the day-type abbreviation (and circuit for bus-route schedules); non-sampled dates show only the day number.

Usage

```
## S3 method for class 'creel_schedule'
format(x, ...)
```

Arguments

x	A creel_schedule object.
...	Currently unused.

Value

A character vector, one element per output line.

format.creel_schema *Format method for creel_schema*

Description

Format method for creel_schema

Usage

```
## S3 method for class 'creel_schema'
format(x, ...)
```

Arguments

x	A creel_schema object.
...	Ignored.

Value

A character vector of formatted lines.

format.creel_season_summary

Format a creel_season_summary object

Description

Format a creel_season_summary object

Usage

```
## S3 method for class 'creel_season_summary'  
format(x, ...)
```

Arguments

x	A creel_season_summary object.
...	Additional arguments (unused).

Value

A character vector.

format.creel_tost_validation

Format creel_tost_validation for printing

Description

Format creel_tost_validation for printing

Usage

```
## S3 method for class 'creel_tost_validation'  
format(x, ...)
```

Arguments

x	A creel_tost_validation object
...	Additional arguments (currently ignored)

Value

Character vector with formatted output

format.creel_validation

Format creed_validation for printing

Description

Format creed_validation for printing

Usage

```
## S3 method for class 'creed_validation'
format(x, ...)
```

Arguments

x	A creed_validation object
...	Additional arguments (currently ignored)

Value

Character vector with formatted output

generate_bus_schedule *Generate a bus-route sampling frame*

Description

Converts a creed schedule calendar and circuit definitions into a sampling frame tibble with inclusion_prob and p_period columns ready for creed_design(survey_type = "bus_route").

Inclusion probability formula: inclusion_prob = p_site * p_period where p_period = crew / n_circuits.

n_circuits is the number of distinct circuit values in sampling_frame (or 1 when circuit is NULL). crew is the number of field crews deployed simultaneously.

Usage

```
generate_bus_schedule(
  schedule,
  sampling_frame,
  site,
  p_site,
  circuit = NULL,
  crew,
  seed = NULL
)
```

Arguments

<code>schedule</code>	A creel_schedule tibble from generate_schedule() . Currently unused in computation but required to ensure the caller has built a valid schedule before constructing the sampling frame.
<code>sampling_frame</code>	A data frame with <code>site</code> and <code>p_site</code> columns (and optionally <code>circuit</code>).
<code>site</code>	Column in <code>sampling_frame</code> giving site identifiers (tidy selector: bare name, quoted string, or tidyselect helper).
<code>p_site</code>	Column in <code>sampling_frame</code> giving per-site selection probability within the circuit. Values must sum to 1.0 per circuit (tolerance 1e-6).
<code>circuit</code>	Optional column giving circuit assignment. If NULL, all sites are treated as a single circuit.
<code>crew</code>	Integer scalar: number of crews in the field simultaneously.
<code>seed</code>	Optional integer seed (reserved for future randomised designs; currently unused as the function is deterministic).

Value

A tibble: `sampling_frame` columns plus `p_period` and `inclusion_prob`. `inclusion_prob = p_site * p_period`.

See Also

Other "Scheduling": [attach_count_times\(\)](#), [generate_count_times\(\)](#), [generate_progressive_start\(\)](#), [generate_schedule\(\)](#), [new_creel_schedule\(\)](#), [read_schedule\(\)](#), [validate_creel_schedule\(\)](#), [write_schedule\(\)](#)

Examples

```
sched <- generate_schedule(
  start_date = "2024-06-01",
  end_date   = "2024-06-14",
  n_periods  = 1,
  sampling_rate = c(weekday = 0.3, weekend = 0.6),
  seed       = 42
)
frame <- data.frame(
  site = c("A", "B", "C"),
  p_site = c(0.4, 0.3, 0.3),
  stringsAsFactors = FALSE
)
generate_bus_schedule(sched, frame, site = site, p_site = p_site, crew = 2)
```

generate_count_times *Generate within-day count time windows*

Description

Generates count time windows for a creel survey day using one of three strategies: random (stratified random placement within equal-width strata), systematic (random start in first stratum with fixed spacing thereafter, preferred per Pollock et al. 1994 and Colorado CPW 2012), or fixed (user-supplied non-overlapping windows).

Usage

```
generate_count_times(
  start_time = NULL,
  end_time = NULL,
  strategy,
  n_windows = NULL,
  window_size = NULL,
  min_gap = NULL,
  fixed_windows = NULL,
  seed = NULL
)
```

Arguments

start_time	Character. Survey-day start time in "HH:MM" format. Required for strategy = "random" and "systematic".
end_time	Character. Survey-day end time in "HH:MM" format. Required for strategy = "random" and "systematic".
strategy	Character scalar. One of "random", "systematic", or "fixed".
n_windows	Positive integer. Number of count time windows. Required for strategy = "random" and "systematic". The total span (end_time - start_time in minutes) must be evenly divisible by n_windows.
window_size	Positive integer. Duration of each count window in minutes. Required for strategy = "random" and "systematic".
min_gap	Non-negative integer. Minimum gap (minutes) between windows. Required for strategy = "random" and "systematic". window_size + min_gap must not exceed the stratum width (total_span / n_windows).
fixed_windows	A data frame with start_time and end_time columns (character "HH:MM"). Required for strategy = "fixed". Windows must be non-overlapping.
seed	Integer seed for reproducible window placement. Passed to <code>withr::with_seed()</code> . Applies to "random" and "systematic" strategies. Has no effect for "fixed" strategy.

Details

Output is a `creel_schedule` data frame compatible with `write_schedule()`.

Random strategy: Each of the `n_windows` strata of equal length $k = \text{total_span} / n_windows$ receives one window with a uniformly random start within $[\text{stratum_start}, \text{stratum_start} + k - \text{window_size}]$.

Systematic strategy (recommended): A single random start `t1` is drawn from $[\text{start_min}, \text{start_min} + k - \text{window_size}]$, all subsequent windows begin at $t1 + (i-1) * k$ for $i = 1, \dots, n_windows$. This is the design described in Pollock et al. (1994) and recommended by Colorado CPW (2012).

Fixed strategy: Windows are taken exactly as supplied after sorting by start time. Overlapping windows trigger an error.

Value

A `creel_schedule` data frame with columns:

- `start_time` (character "HH:MM"): Window start time.
- `end_time` (character "HH:MM"): Window end time.
- `window_id` (integer, 1-based, ordered by start time): Window index.

See Also

Other "Scheduling": `attach_count_times()`, `generate_bus_schedule()`, `generate_progressive_start()`, `generate_schedule()`, `new_creel_schedule()`, `read_schedule()`, `validate_creel_schedule()`, `write_schedule()`

Examples

```
# Random strategy
generate_count_times(
  start_time = "06:00", end_time = "14:00",
  strategy = "random", n_windows = 4, window_size = 30, min_gap = 10,
  seed = 42
)

# Systematic strategy (preferred; Pollock et al. 1994)
generate_count_times(
  start_time = "06:00", end_time = "14:00",
  strategy = "systematic", n_windows = 4, window_size = 30, min_gap = 10,
  seed = 42
)

# Fixed strategy
fw <- data.frame(
  start_time = c("07:00", "09:00", "11:00"),
  end_time = c("07:30", "09:30", "11:30"),
  stringsAsFactors = FALSE
)
generate_count_times(strategy = "fixed", fixed_windows = fw)
```

generate_progressive_start

Schedule progressive count circuit start times

Description

Generates randomised start times for progressive count surveys following Hoenig et al. (1993). Two scheduling strategies are supported:

Usage

```
generate_progressive_start(
  open_start,
  open_end,
  circuit_time,
  strategy = c("discrete", "wraparound"),
  n = 1L,
  seed = NULL
)
```

Arguments

open_start	Character. Survey-day opening time in "HH:MM" format.
open_end	Character. Survey-day closing time in "HH:MM" format. Must be later than open_start.
circuit_time	Positive numeric. Duration of one circuit traversal τ in hours. Must be shorter than the survey period T.
strategy	Character scalar. "discrete" (default) or "wraparound". For "discrete", $T / \text{circuit_time}$ must be a whole number (within 0.001 h tolerance).
n	Positive integer. Number of survey days to schedule. Returns one row per day.
seed	Optional integer. Passed to <code>withr::with_seed()</code> for reproducible scheduling. Has no effect when NULL.

Details

- "discrete" (recommended): The survey period T must be an integer multiple of the circuit time τ . A start time is drawn uniformly from $\{0, \tau, 2\tau, \dots, (k-1)\tau\}$ where $k = T/\tau$. The starting *location* and *direction* of travel must both be randomised; direction is returned in the output and must be recorded in the field protocol.
- "wraparound": A start time is drawn from $U[0, T)$. If the circuit would extend past the end of the survey period it wraps to the beginning of the day (`is_wrapped = TRUE`). Starting location need not be randomised, though direction still must be.

Value

A creel_schedule data frame with columns:

- `circuit_start` (character "HH:MM"): Scheduled circuit start time.
- `circuit_end` (character "HH:MM"): Scheduled circuit end time. For "wraparound" this may be earlier than `circuit_start` when the circuit crosses the end of the survey period.
- `is_wrapped` (logical): TRUE when circuit wraps around the end of the survey period. Always FALSE for "discrete".
- `direction` (character): "forward" or "reverse". Must be implemented in the field protocol for unbiased estimation.

Common scheduling error

Drawing the start time from $U[0, T - \tau]$ is **biased** – it makes the middle of the survey day over-represented, introducing bias toward mid-day effort patterns. Both strategies here avoid this error.

References

Hoenig, J. M., Robson, D. S., Jones, C. M., and Pollock, K. H. (1993). Scheduling counts in the instantaneous and progressive count methods for estimating sportfishing effort. *North American Journal of Fisheries Management*, **13**, 723–736.

See Also

Other "Scheduling": `attach_count_times()`, `generate_bus_schedule()`, `generate_count_times()`, `generate_schedule()`, `new_creel_schedule()`, `read_schedule()`, `validate_creel_schedule()`, `write_schedule()`

Examples

```
# Discrete strategy: T = 10 h, tau = 2 h -> k = 5 valid start times
generate_progressive_start(
  open_start = "06:00", open_end = "16:00",
  circuit_time = 2, strategy = "discrete", n = 5, seed = 42
)

# Wraparound strategy: start drawn from U[0, T)
generate_progressive_start(
  open_start = "06:00", open_end = "16:00",
  circuit_time = 2, strategy = "wraparound", n = 5, seed = 42
)
```

generate_schedule	<i>Generate a creel survey sampling schedule</i>
-------------------	--

Description

Generates a stratified random sampling calendar for a creel survey season. The season is divided into weekday and weekend strata, and days are randomly selected within each stratum. Output is a `creel_schedule` tibble ready to pass to `creel_design()`.

Usage

```
generate_schedule(
  start_date,
  end_date,
  n_periods,
  n_days = NULL,
  sampling_rate = NULL,
  period_labels = NULL,
  expand_periods = TRUE,
  include_all = FALSE,
  ordered_periods = FALSE,
  period_intensity = NULL,
  seed,
  special_periods = NULL
)
```

Arguments

<code>start_date</code>	Character or Date. First day of the survey season (ISO 8601 "YYYY-MM-DD").
<code>end_date</code>	Character or Date. Last day of the survey season (ISO 8601 "YYYY-MM-DD").
<code>n_periods</code>	Integer. Number of sampling periods per day.
<code>n_days</code>	Named integer vector of days to sample per stratum (e.g., <code>c(weekday = 20, weekend = 10)</code>), or a scalar applied uniformly to all strata. Mutually exclusive with <code>sampling_rate</code> .
<code>sampling_rate</code>	Named numeric vector of sampling fractions per stratum (e.g., <code>c(weekday = 0.3, weekend = 0.6)</code>), or a scalar applied uniformly to all strata. Mutually exclusive with <code>n_days</code> .
<code>period_labels</code>	Optional character vector of length <code>n_periods</code> with human-readable period names. When supplied, <code>period_id</code> is character (or ordered factor if <code>ordered_periods = TRUE</code>).
<code>expand_periods</code>	Logical (default <code>TRUE</code>). If <code>TRUE</code> , output has one row per sampled day x period (<code>nrow = sampled_days * n_periods</code>). If <code>FALSE</code> , output has one row per sampled day and <code>period_id</code> is omitted.
<code>include_all</code>	Logical (default <code>FALSE</code>). If <code>TRUE</code> , all season dates are returned with a sampled logical column. If <code>FALSE</code> , only sampled dates are returned.

ordered_periods	Logical (default FALSE). If TRUE and period_labels is supplied, period_id is an ordered factor preserving label order.
period_intensity	Not yet implemented. Must be NULL.
seed	Integer seed for reproducible random day selection. Uses <code>withr::with_seed()</code> to avoid mutating global RNG state.
special_periods	Optional data frame declaring calendar-defined special periods. Must contain start_date, end_date, and label columns, with optional reason. Periods are expanded to day-level assignments before sampling so boundary-crossing periods are split by civil date.

Value

A creel_schedule data frame with columns:

- date (Date): Sampled (or all) dates.
- day_type (character): Baseline "weekday" or "weekend" classification.
- final_stratum (character): Present when special_periods is supplied; gives the final stratum used for day selection.
- special_period_reason (character): Present when special_periods is supplied; gives the optional reason for the special-period assignment.
- period_id (integer, character, or ordered factor): Period within day. Absent when expand_periods = FALSE.
- sampled (logical): Present only when include_all = TRUE.

See Also

Other "Scheduling": `attach_count_times()`, `generate_bus_schedule()`, `generate_count_times()`, `generate_progressive_start()`, `new_creel_schedule()`, `read_schedule()`, `validate_creel_schedule()`, `write_schedule()`

Examples

```
# Basic schedule with stratified sampling rates
sched <- generate_schedule(
  start_date = "2024-06-01",
  end_date = "2024-08-31",
  n_periods = 2,
  sampling_rate = c(weekday = 0.3, weekend = 0.6),
  seed = 42
)

# Use result with creel_design()
creel_design(sched, date = date, strata = day_type)
```

get_enumeration_counts

Get enumeration counts from a bus-route creel design with interviews

Description

Returns the enumeration count data (observed and interviewed angler counts, and the expansion factor) for each interview record in a bus-route creel_design with interviews attached via [add_interviews](#).

The expansion factor $n_{counted}/n_{interviewed}$ accounts for anglers present at a site who were not interviewed. It is used during bus-route effort and harvest estimation (Jones & Pollock (2012) Eq. 19.4 and 19.5).

Usage

```
get_enumeration_counts(design)
```

Arguments

design	A creel_design object with design_type = "bus_route" and interview data attached via add_interviews .
--------	---

Value

A data frame with the site identifier column, the circuit identifier column, n_counted (resolved column name), n_interviewed (resolved column name), and .expansion (n_counted / n_interviewed, NA when n_interviewed = 0).

References

Jones, C. M., & Pollock, K. H. (2012). Recreational survey methods: estimating effort, harvest, and abundance. In A. V. Zale, D. L. Parrish, & T. M. Sutton (Eds.), *Fisheries Techniques* (3rd ed., pp. 883–919). American Fisheries Society. Enumeration expansion factor used in Eq. 19.4 and 19.5 for bus-route effort and harvest estimation.

See Also

[creel_design\(\)](#), [add_interviews\(\)](#), [get_sampling_frame\(\)](#), [get_inclusion_probs\(\)](#)

Other "Bus-Route Helpers": [get_inclusion_probs\(\)](#), [get_sampling_frame\(\)](#), [get_site_contributions\(\)](#)

Examples

```
cal <- data.frame(
  date = as.Date(c("2024-06-03", "2024-06-04", "2024-06-05", "2024-06-06")),
  day_type = "weekday"
)
sf <- data.frame(
  site = c("A", "B"),
  circuit = c("am", "am"),
```

```

    p_site = c(0.6, 0.4),
    p_period = rep(0.5, 2)
  )
  design_br <- creel_design(
    cal,
    date = date, strata = day_type,
    survey_type = "bus_route", sampling_frame = sf,
    site = site, circuit = circuit,
    p_site = p_site, p_period = p_period
  )
  interviews <- data.frame(
    date = as.Date(c("2024-06-03", "2024-06-04")),
    site = c("A", "B"), circuit = c("am", "am"),
    catch_total = c(3L, 2L), hours_fished = c(2.0, 1.5),
    trip_status = c("complete", "complete"),
    trip_duration = c(2.0, 1.5),
    n_counted = c(5L, 4L), n_interviewed = c(3L, 2L)
  )
  design2 <- add_interviews(
    design_br, interviews,
    catch = catch_total, effort = hours_fished,
    trip_status = trip_status, trip_duration = trip_duration,
    n_counted = n_counted, n_interviewed = n_interviewed
  )
  get_enumeration_counts(design2)

```

get_inclusion_probs *Get inclusion probabilities from a bus-route design*

Description

Returns the computed inclusion probabilities ($\pi_i = p_{\text{site}} \times p_{\text{period}}$) for each site-circuit combination in a bus-route creel design. The inclusion probability represents the two-stage sampling probability: the probability that a particular site is visited during a particular sampling period, combining both the site selection probability within the circuit and the circuit (period) selection probability.

Usage

```
get_inclusion_probs(design)
```

Arguments

design A `creel_design()` object created with `survey_type = "bus_route"`.

Value

A data frame with three columns: the site identifier column, the circuit identifier column, and `.pi_i` (the computed inclusion probability $\pi_i = p_{\text{site}} \times p_{\text{period}}$ for each site-circuit unit). Column names for site and circuit match the resolved column names from the original sampling frame (or `.circuit` for designs without an explicit circuit column).

References

Jones, C. M., & Pollock, K. H. (2012). Recreational survey methods: estimating effort, harvest, and abundance. In A. V. Zale, D. L. Parrish, & T. M. Sutton (Eds.), *Fisheries Techniques* (3rd ed., pp. 883–919). American Fisheries Society. Definition of π_i for two-stage bus-route sampling, used in Eq. 19.4 and 19.5.

See Also

[creel_design\(\)](#), [get_sampling_frame\(\)](#)

Other "Bus-Route Helpers": [get_enumeration_counts\(\)](#), [get_sampling_frame\(\)](#), [get_site_contributions\(\)](#)

Examples

```
sf <- data.frame(
  site = c("A", "B", "C"),
  p_site = c(0.3, 0.4, 0.3),
  p_period = rep(0.5, 3),
  stringsAsFactors = FALSE
)
cal <- data.frame(
  date = as.Date("2024-06-01"),
  day_type = "weekday",
  stringsAsFactors = FALSE
)
design <- creel_design(cal,
  date = date, strata = day_type,
  survey_type = "bus_route", sampling_frame = sf,
  site = site, p_site = p_site, p_period = p_period
)
get_inclusion_probs(design)
```

get_sampling_frame	<i>Extract the sampling frame from a bus-route creel design</i>
--------------------	---

Description

Returns the sampling frame data frame stored in a bus-route `creel_design` object. The data frame contains the user's original columns plus a precomputed `.pi_i` column ($\pi_i = p_{\text{site}} * p_{\text{period}}$). Aborts with an informative error for non-bus-route designs.

Usage

```
get_sampling_frame(design)
```

Arguments

`design` A `creel_design` object with `design_type = "bus_route"`.

Value

A data frame: the `sampling_frame` as stored in the design, with the addition of a `.pi_i` column and (if circuit was omitted) a `.circuit` column.

See Also

Other "Bus-Route Helpers": [get_enumeration_counts\(\)](#), [get_inclusion_probs\(\)](#), [get_site_contributions\(\)](#)

Examples

```
sf <- data.frame(
  site      = c("A", "B", "C"),
  p_site    = c(0.3, 0.4, 0.3),
  p_period  = 0.5
)
calendar <- data.frame(
  date      = as.Date("2024-06-01"),
  day_type  = "weekday"
)
design <- creel_design(calendar,
  date = date,
  strata = day_type,
  survey_type = "bus_route",
  sampling_frame = sf,
  site = site,
  p_site = p_site,
  p_period = p_period
)
get_sampling_frame(design)
```

`get_site_contributions`

Extract per-site effort contributions from a bus-route estimate

Description

Returns the per-site calculation table $(e_i, \pi_i, e_i/\pi_i)$ stored as an attribute on effort estimate objects returned by [estimate_effort\(\)](#) for bus-route survey designs. This table enables traceability of the Horvitz-Thompson estimator (Jones & Pollock 2012, Eq. 19.4) and supports validation against published examples (Malvestuto 1996, Box 20.6).

Usage

```
get_site_contributions(x)
```

Arguments

`x` A `creel_estimates` object returned by [estimate_effort\(\)](#) for a bus-route design.

Value

A tibble with columns:

site	Site identifier (from sampling frame)
circuit	Circuit identifier (from sampling frame)
e_i	Enumeration-expanded effort at site i (effort * expansion)
pi_i	Inclusion probability for site i ($p_{\text{site}} * p_{\text{period}}$)
e_i_over_pi_i	Site contribution to Horvitz-Thompson estimate

References

Jones, C. M., & Pollock, K. H. (2012). Recreational survey methods: estimating effort, harvest, and abundance. In A. V. Zale, D. L. Parrish, & T. M. Sutton (Eds.), *Fisheries Techniques* (3rd ed., pp. 883-919). American Fisheries Society.

See Also

[estimate_effort\(\)](#), [get_sampling_frame\(\)](#), [get_inclusion_probs\(\)](#), [get_enumeration_counts\(\)](#)

Other "Bus-Route Helpers": [get_enumeration_counts\(\)](#), [get_inclusion_probs\(\)](#), [get_sampling_frame\(\)](#)

Examples

```
cal <- data.frame(
  date = as.Date(c("2024-06-03", "2024-06-04", "2024-06-05", "2024-06-06")),
  day_type = "weekday"
)
sf <- data.frame(
  site = c("A", "B"),
  circuit = c("am", "am"),
  p_site = c(0.6, 0.4),
  p_period = rep(0.5, 2)
)
design_br <- creel_design(
  cal,
  date = date, strata = day_type,
  survey_type = "bus_route", sampling_frame = sf,
  site = site, circuit = circuit,
  p_site = p_site, p_period = p_period
)
interviews <- data.frame(
  date = as.Date(c("2024-06-03", "2024-06-04")),
  site = c("A", "B"), circuit = c("am", "am"),
  catch_total = c(3L, 2L), hours_fished = c(2.0, 1.5),
  trip_status = c("complete", "complete"),
  trip_duration = c(2.0, 1.5),
  n_counted = c(5L, 4L), n_interviewed = c(3L, 2L)
)
design_br <- add_interviews(
  design_br, interviews,
  catch = catch_total, effort = hours_fished,
```

```

    trip_status = trip_status, trip_duration = trip_duration,
    n_counted = n_counted, n_interviewed = n_interviewed
  )
result <- estimate_effort(design_br)
get_site_contributions(result)

```

impute_camera_counts *Impute missing camera counts using GLM or GLMM*

Description

Fills outage rows in a camera count data frame using a per-stratum model. `strata_col` (typically `day_type`) partitions the data: one model is fitted within each level, from that level's own observed days. The GLM method (default) fits an intercept-only Poisson GLM, so an outage day is filled with its stratum's mean count. The GLMM method fits a negative binomial GLMM and requires the `glmmTMB` package (in `Suggests`).

Outage rows are identified as any row where `status_col != "operational"` AND `count_col` is NA. All rows are returned; imputed rows have `.imputed = TRUE`. The original `status_col` values (e.g., "battery_failure") are preserved in imputed rows for traceability.

Usage

```

impute_camera_counts(
  data,
  count_col,
  strata_col,
  status_col = "camera_status",
  method = "glm",
  m = 1L,
  site_col = NULL
)

```

Arguments

<code>data</code>	A data frame of camera count records. Must have at least one row and must contain the columns named by <code>count_col</code> , <code>strata_col</code> , and <code>status_col</code> .
<code>count_col</code>	Character scalar. Name of the integer count column (e.g., "ingress_count"). Outage rows have NA in this column.
<code>strata_col</code>	Character scalar. Name of the day-type stratum column (e.g., "day_type"). Partitions the data; a separate GLM/GLMM is fitted within each level rather than this column entering a model as a predictor.
<code>status_col</code>	Character scalar. Name of the camera status column. Default "camera_status". Rows where this column is not "operational" and <code>count_col</code> is NA are treated as outages.

method	Character scalar. Imputation model: "glm" (default, Poisson GLM, no extra dependencies) or "glmm" (negative binomial GLMM via glmmTMB, requires glmmTMB in Suggests).
m	Integer scalar. Number of completed data sets to generate. 1L (default) fills each outage row with the fitted mean, reproducing the single-imputation behaviour of earlier versions and returning a plain data frame. m > 1 performs multiple imputation and returns a camera_imputations object for <code>est_effort_camera_mi()</code> to pool. Afrifa-Yamoah et al. (2020) use m = 5 as "an appropriate balance of the bias-variance trade-off". The distinction matters because a single completed data set structurally cannot carry the between-imputation variance. Inside <code>svytotal()</code> a prediction is indistinguishable from an observation, so the imputation model's uncertainty is dropped, and fitted means are smoother than real counts, shrinking the between-day variance a second time (GH #137).
site_col	Character scalar or NULL. When method = "glmm" and site_col is not NULL, a random intercept (1 site_col) is included in the GLMM formula. Default NULL.

Details

[Experimental]

Value

A data frame with the same rows and columns as data, plus a new logical column `.imputed` appended as the last column. Outage rows are filled in `count_col` with model-predicted counts (rounded to integer). The `count_col` storage mode is set to "integer" for schema compatibility with `add_counts()`. Row count equals `nrow(data)`.

Where these imputation models come from

Filling camera outages with a fitted model rather than dropping the days is established practice – Hartill et al. (2016) and Afrifa-Yamoah et al. (2020) both do it – but **neither of the two models offered here is taken from a published creel study**. Both are the package's own choices, and they are deliberately simpler than either paper's.

Hartill et al. (2016) predict the outage ramp's daily count from the counts observed at *two other ramps* on the same day, square-root transformed and fitted as third-order polynomials, given fishing year, season and day-type, selected stepwise with ramp:year interaction terms. They chose a cross-site model precisely because counts on the days either side of an outage were "not considered to be sufficiently representative". The model here has no auxiliary site to borrow from, so it fits the stratum's own observed days.

Afrifa-Yamoah et al. (2020) evaluate nine models in a fully conditional specification multiple-imputation framework – quasi-Poisson, negative binomial, their zero-inflated forms, bootstrap variants and predictive mean matching – with climatic covariates as fixed effects and temporal classifications as random intercepts. Their conclusion does **not** favour the negative binomial: zero-inflated Poisson models "were generally ranked best", and they report the negative binomial fits as slow and cumbersome to converge. The negative binomial offered by method = "glmm" is here as an overdispersion-tolerant alternative to the Poisson default, not as their recommendation, and it falls

back to the Poisson GLM when `glmmTMB` fails outright. A fit that returns while flagging a convergence problem is used as it stands – there is no convergence check beyond the error.

What this function does take from Afrifa-Yamoah et al. (2020) is the multiple-imputation framing itself: that a single completed data set cannot carry the uncertainty of having imputed at all. See `m` below and `est_effort_camera_mi()`.

References

Afrifa-Yamoah, E., Taylor, S.M., Fisher, A., and Mueller, U. 2020. Imputation of missing data from time-lapse cameras used in recreational fishing surveys. *ICES Journal of Marine Science* 77(7-8):2984-2994. doi:[10.1093/icesjms/fsaa180](https://doi.org/10.1093/icesjms/fsaa180) Source of the multiple-imputation framing, not of the negative binomial model offered by `method = "glmm"`.

Hartill, B.W., Payne, G.W., Rush, N., and Bian, R. 2016. Bridging the temporal gap: continuous and cost-effective monitoring of dynamic recreational fisheries by web cameras and creel surveys. *Fisheries Research* 183:488-497. doi:[10.1016/j.fishres.2016.06.002](https://doi.org/10.1016/j.fishres.2016.06.002) Imputes camera outages with a generalised linear model, but a cross-site one; it is not the source of the per-stratum model used here.

See Also

`est_effort_camera()`, `add_counts()`

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
library(tidycreel)
data(example_camera_counts)

# Impute missing counts using the default Poisson GLM
imputed <- impute_camera_counts(
  example_camera_counts,
  count_col = "ingress_count",
  strata_col = "day_type"
)

# Inspect imputed rows
imputed[imputed$.imputed, ]

# Pass imputed data directly into a camera design
cal <- data.frame(
  date = unique(example_camera_counts$date),
  day_type = unique(example_camera_counts[, c("date", "day_type")][["day_type"]])
)
design <- creel_design(cal,
  date = date, strata = day_type,
  survey_type = "camera", camera_mode = "counter"
```

```
)
design <- add_counts(design, imputed)
```

mean_party_size	<i>Mean anglers per boat party from interviews</i>
-----------------	--

Description

Returns the mean number of anglers per boat party, taken from an interviews table. This is the multiplier used to expand a count of boats into a count of anglers when the clerk counted boats rather than the people aboard them.

Boats move, so a count of anglers aboard is often less reliable than a count of hulls. Counting boats and expanding by the interviewed party size trades an unreliable field count for a measured one, at the cost of assuming the interviewed parties are representative of the boats that were counted.

Usage

```
mean_party_size(
  interviews,
  n_anglers,
  angler_type = NULL,
  boat_value = "boat",
  by = NULL
)
```

Arguments

<code>interviews</code>	A data frame of interviews, one row per party.
<code>n_anglers</code>	Tidy selector for the numeric party-size column.
<code>angler_type</code>	Optional tidy selector for the column recording whether a party fished from a boat or the bank. When supplied, only boat parties are used.
<code>boat_value</code>	Value of <code>angler_type</code> marking a boat party. Defaults to "boat". Ignored when <code>angler_type</code> is NULL.
<code>by</code>	Optional tidy selector for one or more grouping columns. When supplied, a mean is returned for each group rather than one overall value.

Details

Each row of interviews is assumed to be one party. A table carrying several rows per party — one per species, say — will weight larger parties more than once; reduce it to one row per party first.

Supply `by` when party size differs across the survey. Weekend parties are commonly larger than weekday parties, and a single season-wide mean applied to both then moves effort in opposite directions in the two strata.

Value

When `by` is `NULL`, a single numeric value. Otherwise a tibble with the grouping columns and a `mean_party_size` column.

Either way the return carries a "se" attribute holding the standard error of the mean ($sd / \sqrt{n}$ over parties), one value per group for the `by` form. `derive_angler_count()` reads it, so the sampling error of the multiplier reaches the effort standard error without being passed by hand.

The standard error is an attribute rather than a column so that the scalar return stays usable directly as a multiplier, and so the `by` form keeps exactly one numeric column and remains valid as a `party_size` lookup.

For the `by` form the attribute is **named by the group key**, and `derive_angler_count()` addresses it by name. Attributes do not follow the rows they describe through a dplyr reordering, so a positional attribute would go stale the moment the lookup were sorted — attributing each stratum's standard error to a different stratum while the means, which join by key, stayed correct. A `by`-form lookup whose "se" attribute has no names is refused rather than matched by row order.

A group with a single party has no estimable standard error and gets `NA_real_`, which propagates to an NA effort standard error rather than being quietly treated as zero uncertainty.

See Also

`derive_angler_count()`, `prep_counts_boat_party()`

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `prep_counts_boat_party()`, `prep_counts_daily_effort()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
interviews <- data.frame(
  day_type = c("weekday", "weekday", "weekend", "weekend"),
  type = c("boat", "bank", "boat", "boat"),
  n_anglers = c(2, 1, 3, 4)
)

# Overall, boat parties only
mean_party_size(interviews, n_anglers, angler_type = type)

# By stratum
mean_party_size(interviews, n_anglers, angler_type = type, by = day_type)
```

<code>new_creel_schedule</code>	<i>Create a <code>creel_schedule</code> S3 object</i>
---------------------------------	---

Description

Constructor for the `creel_schedule` S3 class. Wraps a data frame with the `creel_schedule` class attribute following the tibble-subclass pattern used throughout the package.

Usage

new_creel_schedule(data)

Arguments

data A data frame to wrap as a creel_schedule.

Value

A data frame with class `c("creel_schedule", "data.frame")`.

See Also

Other "Scheduling": [attach_count_times\(\)](#), [generate_bus_schedule\(\)](#), [generate_count_times\(\)](#), [generate_progressive_start\(\)](#), [generate_schedule\(\)](#), [read_schedule\(\)](#), [validate_creel_schedule\(\)](#), [write_schedule\(\)](#)

Examples

```
sched <- new_creel_schedule(data.frame(
  date      = as.Date(c("2024-06-01", "2024-06-08")),
  day_type  = c("weekend", "weekend"),
  sampled   = c(TRUE, TRUE)
))
class(sched)
```

optimal_n	<i>Calculate sampling days required under Neyman-optimal allocation</i>
-----------	---

Description

Determines how many sampling days are needed to achieve a target coefficient of variation on the effort estimate, then allocates those days across strata using Neyman (optimal) allocation. Unlike [creel_n_effort\(\)](#), which distributes days proportionally to stratum size, this function concentrates days in strata with higher between-day variance, minimising total days for a given precision.

Usage

optimal_n(cv_target, N_h, ybar_h, s2_h, cost_ratio = 1)

Arguments

cv_target Numeric scalar. Target coefficient of variation for the effort estimate (e.g., 0.20 for 20 percent). Must be in (0, 1].

N_h Named numeric vector. Total available days per stratum (e.g., `c(weekday = 65, weekend = 28)`). Values must be ≥ 1 .

ybar_h	Numeric vector of same length as N_h. Pilot mean effort per day per stratum (e.g., angler-hours per day). Values must be ≥ 0 .
s2_h	Numeric vector of same length as N_h. Pilot variance of effort per day per stratum. Values must be ≥ 0 .
cost_ratio	Numeric scalar or named numeric vector of same length as N_h. Relative cost of sampling one day in each stratum. Default 1 (equal costs). When costs differ across strata, days are preferentially allocated to cheaper, high-variance strata. Values must be > 0 .

Details

Total sample size uses the cost-generalised Cochran (1977) formula (eq. 5.25 / 5.34 with finite-population correction):

$$n = \left\lceil \frac{A \cdot C}{V_0 + \sum_h N_h s_h^2} \right\rceil$$

where $A = \sum_h N_h s_h / \sqrt{c_h}$, $C = \sum_h N_h s_h \sqrt{c_h}$, $V_0 = (CV_{target} \cdot \hat{E})^2$, $\hat{E} = \sum_h N_h \bar{y}_h$, and $s_h = \sqrt{s_h^2}$. When all $c_h = 1$ (equal costs) this reduces to $(\sum_h N_h s_h)^2 / (V_0 + \sum_h N_h s_h^2)$, which gives the same n_{total} as `creel_n_effort()` (per-stratum allocation differs: Neyman uses $n_h \propto N_h s_h$ vs proportional $n_h \propto N_h$).

Per-stratum allocation uses the cost-adjusted Neyman formula (Cochran 1977 eq. 5.30):

$$n_h = \left\lceil n \cdot \frac{N_h s_h / \sqrt{c_h}}{\sum_h N_h s_h / \sqrt{c_h}} \right\rceil$$

where c_h is the relative sampling cost for stratum h . With equal costs (`cost_ratio = 1`), this reduces to the standard Neyman formula $n_h \propto N_h s_h$.

Because each stratum is ceiling-ed independently, `sum(n_h)` may slightly exceed `n_total`.

Value

A named integer vector. Elements named after strata in `N_h` give the optimal sampling days per stratum; element "total" gives Cochran's overall sample size *before* allocation, and "allocated" the sum of the per-stratum values actually returned. Budget against "allocated"; see `creel_n_effort()` for why the two differ.

References

- Cochran, W.G. 1977. Sampling Techniques, 3rd ed. Wiley, New York.
- McCormick, J.L. and Quist, M.C. 2017. Sample size estimation for on-site creel surveys. North American Journal of Fisheries Management 37:970-983. doi:10.1080/02755947.2017.1342723

See Also

[creel_n_effort\(\)](#) for proportional allocation, [reallocate_strata\(\)](#) to re-allocate a fixed day budget.

Other "Planning & Sample Size": [audit_strata\(\)](#), [compare_designs\(\)](#), [creel_n_camera\(\)](#), [creel_n_cpue\(\)](#), [creel_n_effort\(\)](#), [creel_power\(\)](#), [cv_from_n\(\)](#), [power_creel\(\)](#), [reallocate_strata\(\)](#), [simulate_strata_collapse\(\)](#)

Examples

```
# Two-stratum weekday/weekend example
optimal_n(
  cv_target = 0.20,
  N_h       = c(weekday = 65, weekend = 28),
  ybar_h    = c(50, 60),
  s2_h      = c(400, 500)
)

# Weekend sampling costs twice as much -- shift days toward weekdays
optimal_n(
  cv_target = 0.20,
  N_h       = c(weekday = 65, weekend = 28),
  ybar_h    = c(50, 60),
  s2_h      = c(400, 500),
  cost_ratio = c(weekday = 1, weekend = 2)
)
```

plot_design

Plot a creel survey design

Description

Produces a quick visual summary of a `creel_design` object:

- **Without counts attached** (`design$counts` is `NULL`): a bar chart showing the number of sampled days per stratum.
- **With counts attached**: a jitter + crossbar chart showing the distribution of count values per stratum.

Both variants colour bars/points by stratum for easy differentiation.

Usage

```
plot_design(design, title = NULL, ...)
```

Arguments

<code>design</code>	A <code>creel_design</code> object created by creel_design() .
<code>title</code>	Optional character title. Defaults to "Creel Design Summary" (no counts) or "Count Distribution by Stratum" (with counts).
<code>...</code>	Additional arguments (currently ignored).

Value

A ggplot object.

See Also

[creel_design\(\)](#), [autoplot.creel_schedule\(\)](#)

Other "Visualisation": [autoplot.creel_estimates\(\)](#), [autoplot.creel_length_distribution\(\)](#), [autoplot.creel_schedule\(\)](#), [creel_palette\(\)](#), [theme_creel\(\)](#)

Examples

```
data(example_calendar)
data(example_counts)

# Without counts – stratum sample sizes
design <- creel_design(example_calendar, date = date, strata = day_type)
plot_design(design)

# With counts – count distribution per stratum
design_with_counts <- add_counts(design, example_counts)
plot_design(design_with_counts)
```

power_creel

Unified sample-size and power interface for creel surveys

Description

A single tidy entry point for pre-survey sample-size planning that wraps [creel_n_effort\(\)](#), [creel_n_cpue\(\)](#), and [creel_power\(\)](#) and returns a consistent tibble.

Usage

```
power_creel(
  mode = c("effort_n", "cpue_n", "power"),
  target_rse = NULL,
  strata = NULL,
  N_h = NULL,
  ybar_h = NULL,
  s2_h = NULL,
  cv_catch = NULL,
  cv_effort = NULL,
  rho = 0,
  n = NULL,
  cv_historical = NULL,
  delta_pct = NULL,
  alpha = 0.05,
  alternative = c("two.sided", "one.sided")
)
```

Arguments

mode	Character scalar. One of "effort_n", "cpue_n", or "power". Selects the planning formula.
target_rse	Numeric scalar in (0, 1]. Target relative standard error (= target CV). Required for mode %in% c("effort_n", "cpue_n").
strata	Character vector of stratum names. Required for mode = "effort_n". Length must match N_h, ybar_h, and s2_h.
N_h	Numeric vector. Total sampling days available per stratum. Required for mode = "effort_n".
ybar_h	Numeric vector. Pilot mean effort per day per stratum. Required for mode = "effort_n".
s2_h	Numeric vector. Pilot variance of effort per day per stratum. Required for mode = "effort_n".
cv_catch	Numeric scalar. Pilot CV of catch per interview. Required for mode %in% c("cpue_n", "power").
cv_effort	Numeric scalar. Pilot CV of effort per interview. Required for mode = "cpue_n".
rho	Numeric scalar in [-1, 1]. Pilot correlation between catch and effort. Default 0 (conservative). Used for mode %in% c("cpue_n", "power").
n	Integerish scalar. Sample size (interviews) for mode = "power".
cv_historical	Numeric scalar. Historical CV of CPUE for mode = "power". If NULL, cv_catch is used as a proxy.
delta_pct	Numeric scalar (> 0). Fractional change to detect. Required for mode = "power".
alpha	Numeric scalar in (0, 0.5]. Type I error rate. Default 0.05. Used for mode = "power".
alternative	Character. "two.sided" (default) or "one.sided". Used for mode = "power".

Details

Three mode values are supported:

"effort_n" Required sampling *days* per stratum to achieve target_rse on the effort estimate (calls [creel_n_effort\(\)](#)).

"cpue_n" Required *interviews* to achieve target_rse on the CPUE estimate (calls [creel_n_cpue\(\)](#)).

"power" Statistical power to detect a fractional change in CPUE at a given sample size (calls [creel_power\(\)](#)).

Value

A tibble (data frame) with columns varying by mode:

mode = "effort_n" (one row per stratum, then a "total" row giving Cochran's n before allocation and an "allocated" row giving the sum of the per-stratum rows – budget against "allocated", which is what the returned allocation commits to):

stratum Stratum name.

n_required Sampling days required.
 target_rse The requested target RSE.
 mode = "cpue_n" (one row):
 n_required Interviews required.
 target_rse The requested target RSE.
 cv_catch Input CV of catch.
 cv_effort Input CV of effort.
 rho Input correlation.
 mode = "power" (one row):
 power Estimated statistical power.
 n Input sample size.
 delta_pct Input fractional change.
 cv_historical Historical CV used.
 alpha Input significance level.
 alternative Input test direction.

See Also

[creel_n_effort\(\)](#), [creel_n_cpue\(\)](#), [creel_power\(\)](#)

Other "Planning & Sample Size": [audit_strata\(\)](#), [compare_designs\(\)](#), [creel_n_camera\(\)](#), [creel_n_cpue\(\)](#), [creel_n_effort\(\)](#), [creel_power\(\)](#), [cv_from_n\(\)](#), [optimal_n\(\)](#), [reallocate_strata\(\)](#), [simulate_strata_collapse\(\)](#)

Examples

```

# Effort: sampling days needed for 20 percent RSE
power_creel(
  mode      = "effort_n",
  target_rse = 0.20,
  strata     = c("weekday", "weekend"),
  N_h        = c(65, 28),
  ybar_h     = c(50, 60),
  s2_h       = c(400, 500)
)

# CPUE: interviews needed for 20 percent RSE
power_creel(
  mode      = "cpue_n",
  target_rse = 0.20,
  cv_catch   = 0.8,
  cv_effort  = 0.5
)

# Power: detect a 20 percent change with n = 80 interviews
power_creel(

```

```

mode          = "power",
n             = 80L,
cv_historical = 0.5,
delta_pct     = 0.20
)

```

preprocess_camera_timestamps

Preprocess camera ingress-egress timestamps

Description

Converts paired ingress and egress POSIXct timestamps into a data frame of daily angler-effort hours, suitable for passing to [add_counts](#). Duration for each pair is computed as `difftime(egress_col, ingress_col, units = "hours")`. Pairs where egress precedes ingress (negative duration) are flagged with [cli_warn](#) and excluded from the daily sum (set to NA).

Usage

```
preprocess_camera_timestamps(timestamps, date_col, ingress_col, egress_col)
```

Arguments

<code>timestamps</code>	A data frame containing the ingress-egress records.
<code>date_col</code>	Tidy selector for the date column (Date or POSIXct).
<code>ingress_col</code>	Tidy selector for the ingress timestamp column (POSIXct).
<code>egress_col</code>	Tidy selector for the egress timestamp column (POSIXct).

Details

Preprocess camera ingress-egress timestamps to daily effort hours

Value

A data frame with columns `date` and `daily_effort_hours` (one row per unique date, effort hours summed across all valid pairs for that date).

Examples

```

# Camera timestamps: one row per angler arrival/departure pair.
ts <- data.frame(
  survey_date = rep(as.Date(c("2024-06-01", "2024-06-02")), each = 2L),
  ingress_time = as.POSIXct(
    c("2024-06-01 06:00:00", "2024-06-01 09:00:00",
      "2024-06-02 07:00:00", "2024-06-02 10:30:00"), tz = "UTC"
  ),
  egress_time = as.POSIXct(

```

```

      c("2024-06-01 08:00:00", "2024-06-01 11:00:00",
        "2024-06-02 09:00:00", "2024-06-02 13:00:00"), tz = "UTC"
    )
  )
  preprocess_camera_timestamps(ts, date_col = "survey_date",
                              ingress_col = "ingress_time",
                              egress_col = "egress_time")

```

```
prep_counts_boat_party
```

Standardize boat-party sampled-day effort rows

Description

Converts boat-count rows plus mean anglers-per-boat inputs into canonical sampled-day effort rows for downstream use with `add_counts()`. This helper is intentionally narrow: it handles the common boat-party expansion (`boat_count * mean_party_size`) and leaves broader source-specific reconstruction outside estimator internals.

The returned table always contains canonical columns: `date`, any selected strata columns, `effort_type`, `daily_effort`, `psu`, and `correction_factor`. Optional columns `n_counts`, `within_day_var`, and `source_method` are included when supplied.

Usage

```

prep_counts_boat_party(
  data,
  date,
  strata = NULL,
  boat_count,
  mean_party_size,
  mean_party_size_se = NULL,
  effort_type = "boat",
  correction_factor = 1,
  psu = NULL,
  n_counts = NULL,
  within_day_var = NULL,
  source_method = "boat_count_x_mean_party_size"
)

```

Arguments

<code>data</code>	A data frame containing sampled-day boat-count rows.
<code>date</code>	Tidy selector for the Date column.
<code>strata</code>	Optional tidy selector for one or more strata columns.
<code>boat_count</code>	Tidy selector for the numeric boat count column.

mean_party_size	Tidy selector for the numeric mean anglers-per-boat column.
mean_party_size_se	Optional standard error of mean_party_size. May be a scalar or an expression evaluating to one value per row. Supplying it emits the expansion_* carrier columns, which <code>add_counts()</code> reads so the reported standard error includes the party-size sampling error. NULL (the default) leaves the component absent rather than zero. A zero would enter the variance as "the multiplier is known exactly" and be indistinguishable from never having propagated, so the two states are kept apart. NA is accepted and propagates as unknown. Before tidycrrel 3.4.0 this argument did not exist, and the component was unreachable on this path: the same expansion through <code>derive_angler_count()</code> reported a larger, correct standard error while this one silently omitted the term (GH #143).
effort_type	Effort-type values for output. Defaults to "boat". May be a scalar string/factor or an expression that evaluates to one value per row.
correction_factor	Optional multiplicative correction applied after the boat-party expansion. May be a scalar (defaults to 1) or an expression that evaluates to a numeric vector with one value per row. Values must be finite and strictly positive.
psu	Optional tidy selector for the PSU column. Defaults to the selected date column when omitted.
n_counts	Optional tidy selector for the number of within-day counts each sampled-day estimate is built from (k_d). Required whenever within_day_var is supplied.
within_day_var	Optional tidy selector for the within-day sum of squares of the counts behind each sampled-day estimate, that is $\text{sum}((x - \text{mean}(x))^2)$ per PSU. This is not a variance: the divisor is applied downstream by the estimator, which forms $\text{sum}(ss_d) / (n_sampled * (k_bar - 1))$. Supplying a variance here understates the within-day component by a factor of $k_d - 1$. Must be 0 wherever n_counts is 1, and requires n_counts. Supply it on the raw boat_count values you pass in; it is rescaled into daily_effort squared units on output, multiplied by $(\text{mean_party_size} * \text{correction_factor})^2$. <code>add_counts()</code> reads the emitted within_day_var and n_counts columns into the design, so the reported SE carries a within-day component. Before tidycrrel 2.6.0 both columns were written here and never read, and the SE omitted that component entirely. Do not combine with <code>add_counts(count_time_col =)</code>, which derives the same quantity from raw counts; supplying both is an error.
source_method	Optional source-method values. Defaults to "boat_count_x_mean_party_size". May be a scalar string/factor or an expression that evaluates to one value per row.

Value

A tibble with canonical sampled-day effort columns. Required columns are date, selected strata columns (if any), effort_type, daily_effort, psu, and correction_factor. Optional columns are appended when supplied.

See Also

[prep_counts_daily_effort\(\)](#), [add_counts\(\)](#)

Other "Survey Design": [add_catch\(\)](#), [add_counts\(\)](#), [add_interviews\(\)](#), [add_lengths\(\)](#), [add_sections\(\)](#), [as_creel_svydesign\(\)](#), [as_hybrid_svydesign\(\)](#), [compute_angler_effort\(\)](#), [compute_effort\(\)](#), [creel_design\(\)](#), [creel_schema\(\)](#), [creel_vocabulary\(\)](#), [derive_angler_count\(\)](#), [est_effort_camera\(\)](#), [impute_camera_counts\(\)](#), [mean_party_size\(\)](#), [prep_counts_daily_effort\(\)](#), [prep_interview_catch\(\)](#), [prep_interviews_trips\(\)](#), [validate_creel_schema\(\)](#)

Examples

```
raw <- data.frame(
  sample_date = as.Date(c("2024-06-01", "2024-06-02")),
  day_type    = c("weekend", "weekend"),
  boats       = c(10, 12),
  mean_party  = c(2.5, 2.0)
)
prep_counts_boat_party(raw, date = sample_date, strata = day_type,
  boat_count = boats, mean_party_size = mean_party)
```

```
prep_counts_daily_effort
```

Standardize sampled-day effort rows for count-based workflows

Description

Converts a data frame that already contains sampled-day effort estimates into a canonical tibble for downstream use with [add_counts\(\)](#). This helper is the preferred seam for count-based workflows where raw within-day count schedules, section probabilities, boat-party-size adjustments, camera multipliers, or similar count-side corrections have already been resolved outside the core estimator.

The returned table always contains canonical columns: `date`, any selected strata columns, `effort_type`, `daily_effort`, `psu`, and `correction_factor`. Optional columns `n_counts`, `within_day_var`, and `source_method` are included when supplied.

Usage

```
prep_counts_daily_effort(
  data,
  date,
  strata = NULL,
  effort_type,
  daily_effort,
  correction_factor = 1,
  psu = NULL,
  n_counts = NULL,
  within_day_var = NULL,
  source_method = NULL
)
```

Arguments

<code>data</code>	A data frame containing sampled-day effort rows.
<code>date</code>	Tidy selector for the Date column.
<code>strata</code>	Optional tidy selector for one or more strata columns.
<code>effort_type</code>	Tidy selector for the effort-type column. Common values are "bank" and "boat".
<code>daily_effort</code>	Tidy selector for the numeric sampled-day effort column.
<code>correction_factor</code>	Optional multiplicative correction applied to <code>daily_effort</code> . May be a scalar (defaults to 1) or an expression that evaluates to a numeric vector with one value per row, including a bare column name. Values must be finite and strictly positive.
<code>psu</code>	Optional tidy selector for the PSU column. Defaults to the selected date column when omitted.
<code>n_counts</code>	Optional tidy selector for the number of within-day counts each sampled-day estimate is built from (<code>k_d</code>). Required whenever <code>within_day_var</code> is supplied.
<code>within_day_var</code>	Optional tidy selector for the within-day sum of squares of the counts behind each sampled-day estimate, that is $\sum((x - \text{mean}(x))^2)$ per PSU. This is not a variance: the divisor is applied downstream by the estimator, which forms $\text{sum}(ss_d) / (n_sampled * (k_bar - 1))$. Supplying a variance here understates the within-day component by a factor of $k_d - 1$. Must be 0 wherever <code>n_counts</code> is 1, and requires <code>n_counts</code> . Supply it on the raw <code>daily_effort</code> values you pass in; it is rescaled into <code>daily_effort</code> squared units on output, multiplied by <code>correction_factor^2</code> . <code>add_counts()</code> reads the emitted <code>within_day_var</code> and <code>n_counts</code> columns into the design, so the reported SE carries a within-day component. Before tidycreel 2.6.0 both columns were written here and never read, and the SE omitted that component entirely. Do not combine with <code>add_counts(count_time_col =)</code> , which derives the same quantity from raw counts; supplying both is an error.
<code>source_method</code>	Optional tidy selector for a column describing how the sampled-day effort estimate was derived (e.g. "direct_count", "boat_count_x_mean_party_size", "camera_count_x_detection_correction").

Value

A tibble with canonical sampled-day effort columns. Required columns are `date`, selected strata columns (if any), `effort_type`, `daily_effort`, `psu`, and `correction_factor`. Optional columns are appended when supplied.

See Also

`add_counts()`

Other "Survey Design": `add_catch()`, `add_counts()`, `add_interviews()`, `add_lengths()`, `add_sections()`, `as_creel_svydesign()`, `as_hybrid_svydesign()`, `compute_angler_effort()`, `compute_effort()`, `creel_design()`, `creel_schema()`, `creel_vocabulary()`, `derive_angler_count()`, `est_effort_camera()`, `impute_camera_counts()`, `mean_party_size()`, `prep_counts_boat_party()`, `prep_interview_catch()`, `prep_interviews_trips()`, `validate_creel_schema()`

Examples

```
raw_counts <- data.frame(
  sample_date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-08", "2024-06-09")),
  day_type    = c("weekday", "weekday", "weekend", "weekend"),
  effort_kind = c("bank", "bank", "bank", "bank"),
  effort_value = c(15, 23, 45, 52)
)
prep_counts_daily_effort(raw_counts, date = sample_date, strata = day_type,
  effort_type = effort_kind, daily_effort = effort_value)
```

prep_interviews_trips *Standardize trip/interview rows for interview-based workflows*

Description

Converts raw-ish interview records into a canonical tibble for downstream use with `add_interviews()`. This helper standardizes the trip/interview unit, computes effort from timestamps when needed, normalizes `trip_status`, and emits stable columns for effort, trip duration, angler party size, and optional interview attributes.

The returned table always contains canonical columns: `date`, `interview_uid`, `effort_hours`, `trip_status`, `trip_duration`, `n_anglers`, and `refused`. Optional columns such as `catch_total`, `harvest_total`, `angler_type`, `angler_method`, `species_sought`, and any selected strata are appended when supplied.

Usage

```
prep_interviews_trips(
  data,
  date,
  interview_uid,
  effort_hours = NULL,
  trip_status,
  trip_duration = NULL,
  trip_start = NULL,
  interview_time = NULL,
  catch_total = NULL,
  harvest_total = NULL,
  angler_type = NULL,
  angler_method = NULL,
  species_sought = NULL,
  n_anglers = NULL,
  refused = NULL,
  strata = NULL
)
```

Arguments

<code>data</code>	A data frame containing interview records.
<code>date</code>	Tidy selector for the Date column.
<code>interview_uid</code>	Tidy selector for the unique interview identifier.
<code>effort_hours</code>	Optional tidy selector for an effort-in-hours column. Supply this when hours are already available directly.
<code>trip_status</code>	Tidy selector for the trip-status column. Values are normalized to lowercase and must resolve to "complete" or "incomplete".
<code>trip_duration</code>	Optional tidy selector for a trip duration column in hours. When omitted, the helper uses <code>effort_hours</code> if present or computes duration from <code>trip_start</code> and <code>interview_time</code> .
<code>trip_start</code>	Optional tidy selector for trip start timestamps.
<code>interview_time</code>	Optional tidy selector for interview timestamps. When <code>effort_hours</code> is omitted, <code>trip_start</code> and <code>interview_time</code> are used to compute effort in hours.
<code>catch_total</code>	Optional tidy selector for total catch per trip.
<code>harvest_total</code>	Optional tidy selector for total harvest per trip.
<code>angler_type</code>	Optional tidy selector for angler type (e.g. "bank", "boat").
<code>angler_method</code>	Optional tidy selector for fishing method.
<code>species_sought</code>	Optional tidy selector for the target species field.
<code>n_angers</code>	Optional tidy selector for party size. Defaults to 1L when omitted.
<code>refused</code>	Optional tidy selector for the refused interview flag. Defaults to FALSE when omitted.
<code>strata</code>	Optional tidy selector for one or more strata columns to carry forward into the standardized output.

Value

A tibble with canonical trip/interview columns ready for `add_interviews()`.

See Also

[compute_effort\(\)](#), [add_interviews\(\)](#)

Other "Survey Design": [add_catch\(\)](#), [add_counts\(\)](#), [add_interviews\(\)](#), [add_lengths\(\)](#), [add_sections\(\)](#), [as_creel_svydesign\(\)](#), [as_hybrid_svydesign\(\)](#), [compute_angler_effort\(\)](#), [compute_effort\(\)](#), [creel_design\(\)](#), [creel_schema\(\)](#), [creel_vocabulary\(\)](#), [derive_angler_count\(\)](#), [est_effort_camera\(\)](#), [impute_camera_counts\(\)](#), [mean_party_size\(\)](#), [prep_counts_boat_party\(\)](#), [prep_counts_daily_effort\(\)](#), [prep_interview_catch\(\)](#), [validate_creel_schema\(\)](#)

Examples

```
raw <- data.frame(
  survey_date = as.Date(c("2024-06-01", "2024-06-02")),
  day_type    = c("weekend", "weekend"),
  iid         = c("i1", "i2"),
```

```

    hours      = c(2.5, 3.0),
    status     = c("Complete", "incomplete"),
    duration   = c(2.5, 3.0)
  )
  prep_interviews_trips(raw, date = survey_date, interview_uid = iid,
                        effort_hours = hours, trip_status = status,
                        trip_duration = duration)

```

```
prep_interview_catch
```

Standardize long catch-table rows for interview-based workflows

Description

Converts a long-format catch table into a canonical tibble for downstream use with `add_catch()`. The helper standardizes the interview linkage field, species field, numeric catch counts, and normalized catch-type values while keeping the data in long form.

The returned table always contains canonical columns: `interview_uid`, `species`, `count`, and `catch_type`.

Usage

```
prep_interview_catch(data, interview_uid, species, count, catch_type)
```

Arguments

<code>data</code>	A data frame in long format: one row per interview/species/catch-type combination.
<code>interview_uid</code>	Tidy selector for the interview linkage column.
<code>species</code>	Tidy selector for the species code or name column.
<code>count</code>	Tidy selector for the numeric catch count column.
<code>catch_type</code>	Tidy selector for the catch fate column. Values are normalized to lowercase.

Value

A tibble with canonical columns `interview_uid`, `species`, `count`, and `catch_type`.

See Also

[add_catch\(\)](#)

Other "Survey Design": [add_catch\(\)](#), [add_counts\(\)](#), [add_interviews\(\)](#), [add_lengths\(\)](#), [add_sections\(\)](#), [as_creel_svydesign\(\)](#), [as_hybrid_svydesign\(\)](#), [compute_angler_effort\(\)](#), [compute_effort\(\)](#), [creel_design\(\)](#), [creel_schema\(\)](#), [creel_vocabulary\(\)](#), [derive_angler_count\(\)](#), [est_effort_camera\(\)](#), [impute_camera_counts\(\)](#), [mean_party_size\(\)](#), [prep_counts_boat_party\(\)](#), [prep_counts_daily_effort\(\)](#), [prep_interviews_trips\(\)](#), [validate_creel_schema\(\)](#)

Examples

```
raw <- data.frame(
  iid = c("i1", "i1", "i2"),
  sp  = c("walleye", "walleye", "bass"),
  n   = c(5, 2, 1),
  fate = c("Caught", "HARVESTED", "released")
)
prep_interview_catch(raw, interview_uid = iid, species = sp,
  count = n, catch_type = fate)
```

```
print.creel_completeness_report
      Print creel_completeness_report
```

Description

Print creel_completeness_report

Usage

```
## S3 method for class 'creel_completeness_report'
print(x, ...)
```

Arguments

x	A creel_completeness_report object
...	Additional arguments passed to format

Value

The input object, invisibly

```
print.creel_data_validation
      Print a creel_data_validation result
```

Description

Renders a colour-coded cli summary grouped by table and column. Counts pass/warn/fail verdicts in a header line.

Usage

```
## S3 method for class 'creel_data_validation'
print(x, ...)
```

Arguments

x	A <code>creel_data_validation</code> object returned by <code>validate_creel_data()</code> .
...	Ignored.

Value

x, invisibly.

<code>print.creel_design</code>	<i>Print a <code>creel_design</code> object</i>
---------------------------------	---

Description

Print a `creel_design` object

Usage

```
## S3 method for class 'creel_design'
print(x, ...)
```

Arguments

x	A <code>creel_design</code> object
...	Additional arguments passed to <code>format.creel_design()</code>

Value

Invisibly returns the input object

<code>print.creel_design_comparison</code>	<i>Print a <code>creel_design_comparison</code></i>
--	---

Description

Print a `creel_design_comparison`

Usage

```
## S3 method for class 'creel_design_comparison'
print(x, digits = 3L, ...)
```

Arguments

x	A <code>creel_design_comparison</code> object.
digits	Integer. Number of significant digits. Default 3.
...	Ignored.

Value

x, invisibly.

```
print.creel_design_report
```

Print creel_design_report

Description

Print `creel_design_report`

Usage

```
## S3 method for class 'creel_design_report'
print(x, ...)
```

Arguments

x	A <code>creel_design_report</code> object
...	Additional arguments passed to <code>format</code>

Value

The input object, invisibly

```
print.creel_estimates
```

Print creel_estimates

Description

Print `creel_estimates`

Usage

```
## S3 method for class 'creel_estimates'
print(x, ...)
```

Arguments

x	A <code>creel_estimates</code> object
...	Additional arguments passed to <code>format</code>

Value

The input object, invisibly

```
print.creel_estimates_diagnostic
```

Print creel_estimates_diagnostic

Description

Print `creel_estimates_diagnostic`

Usage

```
## S3 method for class 'creel_estimates_diagnostic'  
print(x, ...)
```

Arguments

x	A <code>creel_estimates_diagnostic</code> object
...	Additional arguments passed to <code>format</code>

Value

The input object, invisibly

```
print.creel_estimates_mor
```

Print creel_estimates_mor

Description

Print `creel_estimates_mor`

Usage

```
## S3 method for class 'creel_estimates_mor'  
print(x, ...)
```

Arguments

x	A creel_estimates_mor object
...	Additional arguments passed to format

Value

The input object, invisibly

```
print.creel_hybrid_svydesign
```

Print a creel_hybrid_svydesign

Description

Print a creel_hybrid_svydesign

Usage

```
## S3 method for class 'creel_hybrid_svydesign'
print(x, ...)
```

Arguments

x	A creel_hybrid_svydesign object.
...	Ignored.

Value

x, invisibly.

```
print.creel_schedule
```

Print a creel_schedule as a monthly calendar grid

Description

Prints a formatted ASCII monthly calendar to the console. Sampled dates show day-type abbreviations; bus-route schedules additionally show circuit assignments.

Usage

```
## S3 method for class 'creel_schedule'
print(x, ...)
```

Arguments

`x` A `creel_schedule` object.

`...` Additional arguments passed to `format.creel_schedule()`.

Value

Invisibly returns `x`.

Examples

```
sched <- generate_schedule(
  start_date = "2024-06-01",
  end_date = "2024-07-31",
  n_periods = 1,
  sampling_rate = c(weekday = 0.3, weekend = 0.6),
  seed = 42
)
print(sched)
```

<code>print.creel_schema</code>	<i>Print method for <code>creel_schema</code></i>
---------------------------------	---

Description

Print method for `creel_schema`

Usage

```
## S3 method for class 'creel_schema'
print(x, ...)
```

Arguments

`x` A `creel_schema` object.

`...` Passed to `format.creel_schema()`.

Value

`invisible(x)`.

```
print.creel_season_summary
```

Print a `creel_season_summary` object

Description

Print a `creel_season_summary` object

Usage

```
## S3 method for class 'creel_season_summary'  
print(x, ...)
```

Arguments

<code>x</code>	A <code>creel_season_summary</code> object.
<code>...</code>	Additional arguments passed to <code>format()</code> .

Value

`x`, invisibly.

```
print.creel_summary
```

Print a `creel_summary` object

Description

Print a `creel_summary` object

Usage

```
## S3 method for class 'creel_summary'  
print(x, ...)
```

Arguments

<code>x</code>	A <code>creel_summary</code> object from summary.creel_estimates() .
<code>...</code>	Additional arguments (currently ignored).

Value

The input object, invisibly.

```
print.creel_tost_validation
      Print creel_tost_validation
```

Description

Prints formatted validation results and displays scatter plot comparing complete vs incomplete trip CPUE estimates. Plot includes y=x reference line, confidence interval error bars, and annotations for failed groups.

Usage

```
## S3 method for class 'creel_tost_validation'
print(x, ...)
```

Arguments

x	A creel_tost_validation object
...	Additional arguments passed to format

Value

The input object, invisibly

```
print.creel_validation
      Print creel_validation
```

Description

Print creel_validation

Usage

```
## S3 method for class 'creel_validation'
print(x, ...)
```

Arguments

x	A creel_validation object
...	Additional arguments passed to format

Value

The input object, invisibly

```
print.creel_validation_report
```

Print a `creel_validation_report`

Description

Renders a colour-coded cli summary of the aggregated validation report.

Usage

```
## S3 method for class 'creel_validation_report'
print(x, ...)
```

Arguments

x	A <code>creel_validation_report</code> object returned by <code>validation_report()</code> .
...	Ignored.

Value

x, invisibly.

```
print.creel_variance_comparison
```

Print a `creel_variance_comparison` object

Description

Print a `creel_variance_comparison` object

Usage

```
## S3 method for class 'creel_variance_comparison'
print(x, ...)
```

Arguments

x	A <code>creel_variance_comparison</code> object.
...	Additional arguments (ignored).

Value

x, invisibly.

read_schedule	<i>Read a schedule file into a validated creel_schedule object</i>
---------------	--

Description

Reads a CSV or xlsx schedule file produced by [write_schedule\(\)](#) (or hand-built in Excel) and returns a validated `creel_schedule` object ready for use with [creel_design\(\)](#).

Usage

```
read_schedule(path)
```

Arguments

path	Path to a CSV (.csv) or xlsx (.xlsx, .xls) schedule file. For xlsx files the readxl package must be installed.
------	---

Details

The format is detected from the file extension. All columns are read as text first, then `coerce_schedule_columns()` applies type coercion – the same logic runs regardless of format so that Excel-reformatted dates and serial numbers are handled consistently.

Value

A `creel_schedule` object with columns:

- date (Date)
- day_type (character)
- period_id (integer, if present)
- sampled (logical, if present)

See Also

Other "Scheduling": [attach_count_times\(\)](#), [generate_bus_schedule\(\)](#), [generate_count_times\(\)](#), [generate_progressive_start\(\)](#), [generate_schedule\(\)](#), [new_creel_schedule\(\)](#), [validate_creel_schedule\(\)](#), [write_schedule\(\)](#)

Examples

```
sched <- generate_schedule(
  "2024-06-01", "2024-08-31",
  n_periods = 2,
  sampling_rate = c(weekday = 0.3, weekend = 0.6),
  seed = 42
)
tmp <- tempfile(fileext = ".csv")
write_schedule(sched, tmp)
```

```
sched2 <- read_schedule(tmp)
inherits(sched2, "creel_schedule")
```

reallocate_strata	<i>Compute Neyman-optimal sample allocation across strata</i>
-------------------	---

Description

Given a fixed total sampling budget, allocates days across strata using the Neyman optimal allocation formula (Cochran 1977 eq. 5.24), which assigns more days to strata with larger variability and more calendar days.

Usage

```
reallocate_strata(n_total, N_h, s2_h)
```

Arguments

<code>n_total</code>	Positive integer. Total sampling days available across all strata.
<code>N_h</code>	Named numeric vector. Total available days per stratum. Values must be ≥ 1 .
<code>s2_h</code>	Numeric vector of the same length as <code>N_h</code> . Pilot variance of effort per day per stratum. Values must be ≥ 0 .

Details

Neyman allocation: $n_h = \text{ceiling}(n_{\text{total}} * (N_h * \sqrt{s2_h}) / \sum(N_h * \sqrt{s2_h}))$.

Because each stratum's allocation is ceiling-ed independently, the sum of returned values may slightly exceed `n_total`.

Value

A named integer vector. Elements named after strata in `N_h` give the Neyman-optimal sampling days per stratum.

References

Cochran, W.G. 1977. Sampling Techniques, 3rd ed. Wiley, New York.

See Also

Other "Planning & Sample Size": [audit_strata\(\)](#), [compare_designs\(\)](#), [creel_n_camera\(\)](#), [creel_n_cpue\(\)](#), [creel_n_effort\(\)](#), [creel_power\(\)](#), [cv_from_n\(\)](#), [optimal_n\(\)](#), [power_creel\(\)](#), [simulate_strata_collapse\(\)](#)

Examples

```
reallocate_strata(
  n_total = 36,
  N_h     = c(weekday = 65, weekend = 28),
  s2_h    = c(400, 500)
)
```

season_summary	<i>Assemble pre-computed creel estimates into a report-ready wide tibble</i>
----------------	--

Description

Accepts a named list of pre-computed `creel_estimates` objects (from `estimate_effort()`, `estimate_catch_rate()`, etc.) and joins them into a single wide tibble — one row per stratum with all estimate types as pre-fixed columns.

Usage

```
season_summary(estimates, ...)
```

Arguments

<code>estimates</code>	A named list of <code>creel_estimates</code> objects. Names become column prefixes in the wide tibble (e.g., <code>list(effort = ..., cpue = ...)</code>).
<code>...</code>	Reserved for future arguments.

Details

Note: `season_summary()` performs no re-estimation. All statistical computations must be done before calling this function.

Value

A `creel_season_summary` object (S3 list) with:

- `$table`: A wide tibble — columns prefixed by list element name.
- `$names`: Character vector of input list element names.
- `$n_estimates`: Integer count of estimates assembled.

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```

data(example_calendar)
data(example_counts)
data(example_interviews)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)

result <- season_summary(list(
  effort      = estimate_effort(design),
  catch_rate = estimate_catch_rate(design)
))
result$table
result$n_estimates

```

simulate_creel_catch *Simulate catch counts from a distributional family*

Description

Generates catch observations from one of three distributional families: Negative Binomial ("negbin"), zero-inflated lognormal / Delta ("delta"), or Poisson. Intended for distributional sensitivity analysis, power checks, and Petrere-style estimator comparisons.

Usage

```

simulate_creel_catch(
  n,
  effort = 1,
  family = c("negbin", "delta", "poisson"),
  mu = 5,
  size = 0.5,
  p_zero = 0.4,
  sigma = 1,
  var_structure = c("constant", "proportional", "squared"),
  seed = NULL
)

```

Arguments

n	Integer. Number of catch observations to generate.
effort	Numeric vector of length n or scalar. Effort values (e.g. angler-hours) for each observation. Used to scale catch under var_structure = "proportional" or "squared".

family	Character. Distribution family: "negbin" (default), "delta", or "poisson".
mu	Numeric. Mean catch rate (catch per unit effort). Default 5.
size	Numeric. Negative Binomial dispersion parameter (NB size). Ignored when family = "poisson". Default 0.5.
p_zero	Numeric in [0, 1). Zero-inflation probability for family = "delta". Default 0.40.
sigma	Numeric. Log-scale standard deviation for family = "delta" (positive values only). Default 1.0.
var_structure	Character. Error variance structure. "constant" (default): variance independent of effort; "proportional": $Var \propto f$; "squared": $Var \propto f^2$.
seed	Integer or NULL. Random seed. Default NULL.

Details

Delta distribution (Petrere et al. 2010, Table 1): A mixture of a point mass at zero (probability p_zero) and a lognormal distribution for positive values (parameters mu and sigma on the log scale). Closely matches empirical creel catch distributions.

Variance structures (Petrere et al. 2010):

- constant: $\varepsilon_i \sim N(0, \sigma^2)$
- proportional: $\varepsilon_i \sim N(0, \sigma^2 f_i)$
- squared: $\varepsilon_i \sim N(0, \sigma^2 f_i^2)$

Value

Integer vector of length n containing simulated catch counts.

References

Petrere, M. Jr., Giacomini, H.C. & De Marco, P. Jr. (2010). Catch-per-unit-effort: which estimator is best? Braz. J. Biol. 70: 483–491. doi:10.1590/S151969842010005000010

See Also

[simulate_creel_data](#)

Other "Simulation": [day_length\(\)](#), [simulate_creel_data\(\)](#)

Examples

```
set.seed(1)
# NB catch (default)
catch_nb <- simulate_creel_catch(n = 200, effort = 3.0, mu = 5, size = 0.5)
mean(catch_nb); var(catch_nb)

# Delta distribution (zero-inflated lognormal)
catch_d <- simulate_creel_catch(
  n = 500, effort = 3.0, family = "delta",
  p_zero = 0.45, mu = 1.6, sigma = 0.8
```

```

)
mean(catch_d == 0) # ~0.45

# Proportional variance structure
effort <- rgamma(100, shape = 2.5, rate = 0.57)
catch_prop <- simulate_creel_catch(
  n = 100, effort = effort, mu = 4,
  var_structure = "proportional"
)

```

simulate_creel_data	<i>Simulate a complete creel survey dataset</i>
---------------------	---

Description

Generates realistic synthetic creel data using a three-level hierarchical generative model (day → trip → catch). Caller supplies distributional parameters via `params`; no default data are bundled with the package.

The generative model follows Su & Clapp (2013) for the day and trip levels; the roving-clerk step follows Greene et al. (1995). Only the sampling step is taken from the latter: its own simulated anglers are deterministic – evenly spaced around the shoreline, all starting one hour into an eight-hour day, with trip lengths alternating between 3 and 6 hours – whereas the levels below draw from distributions.

1. **Day level:** Sample `n_sampled_days` days from the season. Each sampled day draws the number of angler trips arriving from a Negative Binomial distribution.
2. **Trip level:** Each trip draws effort (hr) from a Gamma distribution, party size from a zero-truncated Poisson, and trip completion from a Bernoulli draw.
3. **Catch level:** Each complete or incomplete trip draws total catch from a Negative Binomial; harvest is Binomial given total catch.
4. **Roving clerk:** Interview probability is proportional to trip length (length-biased sampling). Incomplete trips contribute elapsed effort, not total effort.

Usage

```

simulate_creel_data(
  params,
  season_days = 100L,
  n_sampled_days = 30L,
  day_types = NULL,
  species = "walleye",
  species_weights = NULL,
  p_complete = 0.75,
  p_zero_catch = 0.4,
  n_anglers_per_day = NULL,
  start_date = Sys.Date(),

```

```

n_counts_per_day = 3L,
lat = NULL,
daylight_hours = NULL,
seed = NULL
)

```

Arguments

params	Named list of distributional parameters. Required. Must include named sub-lists: effort (with gamma_shape, gamma_rate for <code>rgamma()</code>); party (with mean party size for <code>rpois()</code>); catch_per_trip (with mean and nb_size for <code>rnbinom()</code>); harvest (with mean_pct as a percentage, e.g. 35 for 35%); counts (with mean_total_anglers, used when n_anglers_per_day = NULL).
season_days	Integer. Total days in the season. Default 100.
n_sampled_days	Integer. Number of days actually surveyed. Must be <= season_days. Default 30.
day_types	Named numeric vector of stratum proportions, e.g. <code>c(weekday = 5/7, weekend = 2/7)</code> . Names become the day_type values in the output. When NULL (default), uses a single stratum "all". Note: passing a character vector (e.g. <code>c("weekday", "weekend")</code>) will error — the vector must be numeric with named elements.
species	Character vector. Species names for the catch table. Default "walleye". Catch is split proportionally across species using species_weights.
species_weights	Numeric vector. Relative catch weight per species. Must be same length as species. Default equal weights.
p_complete	Numeric in (0, 1]. Probability a trip is complete (intercepted at trip end). Default 0.75.
p_zero_catch	Numeric in [0, 1). Probability a trip has zero total catch (zero-inflation). Default 0.40. Note: this cannot be derived from a survey inventory (API artifact); the default is an empirical estimate from the creel literature.
n_anglers_per_day	Numeric. Mean number of angler parties arriving per sampled day. Overrides <code>params\$counts\$mean_total_anglers</code> when specified. Default NULL (use params).
start_date	Date. First day of the simulated season. Default <code>Sys.Date()</code> .
n_counts_per_day	Integer. Number of instantaneous count observations per sampled day. Default 3.
lat	Numeric latitude in decimal degrees, positive north. When given, the daily fishing period T is computed per date with <code>day_length</code> and the counts table gains daylight_hours and angler_hours columns. Default NULL. Mutually exclusive with daylight_hours.
daylight_hours	Numeric. Length of the daily fishing period T , in hours, used to convert instantaneous counts to angler-hours. Accepts a single value applied to every day, or a named length-12 vector of monthly values (names "1"... "12", or month.abb)

which is matched on the month of each count date. Use this when T is set by regulation or field protocol rather than by daylight. Default NULL. Mutually exclusive with lat.

Supplying neither leaves daylight_hours and angler_hours off the counts table, rather than substituting a latitude the caller never gave.

seed Integer or NULL. Random seed for reproducibility. Default NULL.

Value

A named list with four data frames:

schedule Full-season calendar, one row per day. Columns: date, day_type, sampled (logical). Pass directly to [creel_design](#) as the calendar argument. Unsampled days have day_type assigned proportionally from day_types.

interviews One row per intercepted angler party. Columns: date, day_type, interview_id, trip_status ("complete" or "incomplete"), hours_fished, trip_duration (total trip length; equals hours_fished for complete trips), n_anglers, catch_total, catch_kept, species_sought.

counts One row per instantaneous count. Columns: date, day_type, count_time (integer index 1...n_counts_per_day) and total_anglers, plus daylight_hours (T for that day) and angler_hours (total_anglers * daylight_hours) when lat or daylight_hours was supplied. Pass count_time_col = count_time to [add_counts](#) when n_counts_per_day > 1.

catch Long-format catch table. Columns: interview_id, species, count, catch_type ("caught", "harvested", "released").

Pass angler_hours, not total_anglers, to [add_counts](#). An instantaneous count estimates the mean number of anglers present, not effort; effort is that count multiplied by the length of the period the count was randomised within (Hoenig et al. 1993). `estimate_effort()` expands whatever numeric column it is given and cannot tell the two apart, so handing it total_anglers yields angler-days silently mislabelled as angler-hours. When lat or daylight_hours is supplied the counts table carries three numeric columns and [add_counts](#) will not guess between them: name the one you mean with `add_counts(design, sim$counts, count_col = angler_hours)`.

When n_counts_per_day > 1 you must also **drop the measures you are not using** before attaching. total_anglers differs between the counts taken within one day, so aggregation has no single value to carry forward and would otherwise keep whichever came first. `add_counts()` aborts rather than do that (GH #162); select the columns you need, as the second example below does. daylight_hours is constant within a day and can stay.

Note that [day_length](#) gives astronomical daylight. Where the fishing day is fixed by regulation or access hours instead, pass that period as daylight_hours.

The schedule output can be passed directly to [creel_design](#) as the calendar argument. The interviews and counts outputs are then passed to [add_interviews](#) and [add_counts](#).

References

Su, Z. & Clapp, D.F. (2013). Evaluation of sample design and estimation methods for Great Lakes angler surveys. Trans. Am. Fish. Soc. 142: 234–246. doi:10.1080/00028487.2012.728167

Greene, C.J., Hoenig, J.M., Barrowman, N.J. & Pollock, K.H. (1995). Programs to simulate catch rate estimation in a roving creel survey of anglers. DFO Atlantic Fisheries Research Document 95/99. Department of Fisheries and Oceans, St. John's, NL.

Petrere, M. Jr., Giacomini, H.C. & De Marco, P. Jr. (2010). Catch-per-unit-effort: which estimator is best? Braz. J. Biol. 70: 483–491. doi:10.1590/S151969842010005000010

See Also

[simulate_creel_catch](#)

Other "Simulation": [day_length\(\)](#), [simulate_creel_catch\(\)](#)

Examples

```
my_params <- list(
  effort      = list(gamma_shape = 2.0, gamma_rate = 0.8),
  party       = list(mean = 1.5),
  catch_per_trip = list(mean = 1.8, nb_size = 0.5),
  harvest     = list(mean_pct = 35),
  counts      = list(mean_total_anglers = 10)
)

# Basic simulation (single stratum)
set.seed(42)
sim <- simulate_creel_data(
  params      = my_params,
  season_days = 90,
  n_sampled_days = 20,
  species     = c("walleye", "northern_pike"),
  species_weights = c(0.6, 0.4)
)
head(sim$schedule)
head(sim$interviews)
head(sim$counts)
head(sim$catch)

# Multi-stratum simulation with day_types (named numeric vector).
# `lat` derives the daily fishing period from day_length(), which adds the
# daylight_hours and angler_hours columns to sim2$counts.
set.seed(1)
sim2 <- simulate_creel_data(
  params      = my_params,
  season_days = 90,
  n_sampled_days = 20,
  day_types   = c(weekday = 5/7, weekend = 2/7),
  lat         = 40.699
)

# Round-trip: simulate → creel_design → add_counts → add_interviews
# total_anglers is dropped: it varies between the counts taken within a day,
# so aggregation cannot carry it forward (see the note above).
counts2 <- sim2$counts[, c("date", "day_type", "count_time", "angler_hours")]
```

```

design <- creel_design(sim2$schedule, date = date, strata = day_type) |>
  add_counts(
    counts2,
    count_col = angler_hours, # counts alone are angler-days, not effort
    count_time_col = count_time
  ) |>
  add_interviews(
    sim2$interviews,
    catch      = "catch_total",
    effort     = "hours_fished",
    harvest    = "catch_kept",
    trip_status = "trip_status",
    trip_duration = "trip_duration",
    n_anglers  = "n_anglers",
    interview_type = "roving"
  )

```

simulate_strata_collapse

Simulate the effect of collapsing strata on precision

Description

Compares per-stratum RSE and DEFF before and after merging a set of strata into a single combined stratum.

Usage

```
simulate_strata_collapse(audit, merge_strata)
```

Arguments

audit A `creel_strata_audit` object returned by `audit_strata()`.

merge_strata Character vector. Names of strata to merge (must all appear in `audit$strata$stratum`).

Details

Merged strata are pooled using population-weighted means:

- $N_{\text{merged}} = \sum(N_h[\text{merge_strata}])$
- $n_{\text{merged}} = \sum(n_h[\text{merge_strata}])$
- $ybar_{\text{merged}} = \sum(N_h * ybar_h) / N_{\text{merged}}$ (population-weighted)
- $s2_{\text{merged}} = \sum(n_h * s2_h) / n_{\text{merged}}$ (sample-size-weighted pooled within-stratum variance)

RSE and DEFF for the merged stratum are computed using the same FPC-corrected formulas as `audit_strata()`. Unmerged strata appear identically in both "before" and "after" rows.

Value

A plain tibble (no S3 class) with columns: state ("before" or "after"), stratum, N_h, n_h, RSE, DEFF, meets_target.

References

Cochran, W.G. 1977. Sampling Techniques, 3rd ed. Wiley, New York.

See Also

Other "Planning & Sample Size": [audit_strata\(\)](#), [compare_designs\(\)](#), [creel_n_camera\(\)](#), [creel_n_cpue\(\)](#), [creel_n_effort\(\)](#), [creel_power\(\)](#), [cv_from_n\(\)](#), [optimal_n\(\)](#), [power_creel\(\)](#), [reallocate_strata\(\)](#)

Examples

```
audit <- audit_strata(
  c(early_season = 30, mid_season = 30, late_season = 20),
  n_h = c(early_season = 10, mid_season = 12, late_season = 8),
  ybar_h = c(40, 45, 38),
  s2_h = c(300, 320, 280)
)
simulate_strata_collapse(audit, merge_strata = c("early_season", "mid_season"))
```

standardize_species	<i>Standardize species names to AFS codes</i>
---------------------	---

Description

Maps free-text species names in a data frame to canonical American Fisheries Society (AFS) species codes, appending a species_code column. Matching is case-insensitive and checks both exact common names and comma-separated aliases bundled with the package. Values that already look like a known AFS code (all-uppercase, 3 characters) are passed through directly. Unmatched values are left as NA with a cli warning listing the unrecognised inputs.

Usage

```
standardize_species(
  data,
  species_col = "species",
  lookup = "AFS",
  fuzzy = TRUE,
  keep_original = TRUE,
  custom_codes = NULL
)
```

Arguments

<code>data</code>	A data frame containing a species name column.
<code>species_col</code>	Character scalar naming the column that holds species names. Default "species".
<code>lookup</code>	Character scalar identifying the code system to use. Currently only "AFS" (default) is supported; passing any other value raises an error.
<code>fuzzy</code>	Logical. If TRUE (default), aliases (common abbreviations and alternate names) are also searched. Set FALSE for strict common-name matching only.
<code>keep_original</code>	Logical. If TRUE (default), the original <code>species_col</code> column is preserved unchanged. Set FALSE to drop it.
<code>custom_codes</code>	Named character vector of project-defined overrides applied after the AFS lookup. Names are species name strings (matched case-insensitively); values are the codes to assign. Useful for hybrids, pooled entries, or valid species absent from the default AFS table. Example: <code>c("Wiper" = "WPR", "Crappie" = "CRP-POOL")</code> . NULL (default) applies no overrides.

Details**Handling species not in the AFS table**

The AFS lookup covers common freshwater sport fish but cannot anticipate every project-specific entry. Three common cases require `custom_codes`:

- **Hybrids** (e.g. Wiper = Striped Bass × White Bass) — no universal AFS code; assign a project-defined code.
- **Pooled entries** (e.g. "Crappie" when species was not recorded to species level) — use a code like "CRP-POOL" to signal the aggregated nature of the record.
- **Valid AFS species missing from the built-in table** — supply the correct code via `custom_codes` until the table is updated.

Value

data with an additional `species_code` character column appended. Unmatched rows receive `NA_character_`. When `custom_codes` is supplied, AFS-matched rows are not overwritten; only rows still NA after the AFS pass are candidates for custom matching.

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
interviews <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03")),
  species = c("walleye", "Largemouth Bass", "UNKNOWN"),
  kept = c(2L, 1L, 0L)
)
standardize_species(interviews)

# Override project-specific entries that AFS cannot match
catch <- data.frame(
  species = c("Walleye", "Wiper", "Crappie"),
  stringsAsFactors = FALSE
)
standardize_species(
  catch,
  custom_codes = c("Wiper" = "WPR", "Crappie" = "CRP-POOL")
)
```

```
summarize_boat_composition
```

Tabulate boat composition by month and day type

Description

Computes the percentage of boats that are angler boats from raw count data, grouped by calendar month and day type. Formula: $\text{mean}(\text{angler_boats} / (\text{angler_boats} + \text{non_ang_boats}))$ per group. The day type column is resolved from the design's strata: a stratum named `day_type` when the design declares one, otherwise the first stratum column, which warns when the design declares more than one.

Usage

```
summarize_boat_composition(design, schema, day_type_col = NULL)
```

Arguments

<code>design</code>	A <code>creel_design</code> object with counts attached via add_counts .
<code>schema</code>	A <code>creel_schema</code> object with <code>angler_boats_col</code> and <code>non_ang_boats_col</code> set.
<code>day_type_col</code>	Name of the column holding the day type, as a single string. When <code>NULL</code> (the default) it is resolved from the design's strata as described above.

Details

Count-based summary, not interview-weighted. Rows where `angler_boats + non_ang_boats == 0` are excluded from ratio computation.

Value

A data.frame with class `c("creel_summary_boat_composition", "data.frame")` and columns: month (full month name), day_type, n_events (integer, count events that yielded a share), n_unknown_boats (integer, events excluded because a boat count was not recorded), n_nonpositive_boats (integer, events excluded because the boat total was zero or negative), pct_angler_boats (numeric, 1 decimal, NA when n_events is 0).

Count events that yield no share

A count event contributes an angler-boat share only when the boats were counted and the total is positive. Both exclusions are real – an unrecorded count has no share to give, and a total of zero makes the ratio undefined while a negative one is a data error – and both used to happen with no trace that the event had occurred.

They are now counted in n_unknown_boats and n_nonpositive_boats, and the accounting closes:

```
n_events + n_unknown_boats + n_nonpositive_boats
== count events in that month and day type
```

A month and day type whose every event was excluded keeps its row, reporting NA for pct_angler_boats rather than disappearing.

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary_creel_estimates\(\)](#), [tidy_creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
counts_df <- data.frame(
  date      = as.Date(c("2024-05-01", "2024-05-04",
                        "2024-06-01", "2024-06-08")),
  day_type  = c("weekday", "weekend", "weekday", "weekend"),
  angler_boats = c(3L, 2L, 4L, 1L),
  non_ang_boats = c(1L, 2L, 1L, 3L),
  count      = c(10L, 12L, 9L, 8L)
)
cal <- data.frame(
  date      = counts_df$date,
  day_type  = counts_df$day_type
)
d <- suppressWarnings(
  creel_design(cal, date = date, strata = day_type)
)
d <- suppressWarnings(
  add_counts(d, counts_df, count_col = count)
```

```

)
s <- creel_schema(
  survey_type = "instantaneous",
  angler_boats_col = "angler_boats",
  non_ang_boats_col = "non_ang_boats"
)
summarize_boat_composition(d, s)

```

summarize_by_angler_type

Tabulate interviews by angler type and month

Description

Counts the number of interviews for each angler type within each calendar month. Angler type is taken from the column set via `add_interviews(angler_type = ...)`.

Usage

```
summarize_by_angler_type(design)
```

Arguments

`design` A `creel_design` object with interviews attached and `angler_type` column set via `add_interviews(angler_type = ...)`.

Details

Interview-based summary, not pressure-weighted. This function tabulates raw interview records without applying survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates, use [estimate_catch_rate](#) or [estimate_harvest_rate](#).

Value

A `data.frame` with class `c("creel_summary_angler_type", "data.frame")` and columns: month, angler_type, N, percent.

Unrecorded grouping values

An interview whose grouping value was not recorded is reported under "Unknown", sorted last, rather than dropped. The interview is real and its grouping value is missing, which is not the same as the interview not existing, so `sum(N)` always equals the number of interviews attached to the design. "Unknown" is a label for the absence, never a category anyone selected. This matches [summarize_by_zip](#) and [summarize_by_county](#); the survey-weighted estimators use `<unknown>` instead.

A column holding both unrecorded values and the literal value "Unknown" warns: the two are pooled into one row and cannot be told apart in the output. Missingness is tracked internally, so a category genuinely named "Unknown" keeps its own counts.

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, angler_type = angler_type
)
summarize_by_angler_type(d)
```

summarize_by_county	<i>Tabulate interviews by county of origin</i>
---------------------	--

Description

Maps angler zip codes to county using the **zipcodeR** package, then counts and computes the percent of interviews by county. NA or unmappable zip codes appear as an explicit "Unknown" row.

Usage

```
summarize_by_county(design, zip_col = "zip_code")
```

Arguments

design	A creel_design object with interviews attached via add_interviews .
zip_col	Name of the interview column holding the angler zip code. Defaults to "zip_code".

Details

Interview-based summary, not pressure-weighted. Requires the **zipcodeR** package (listed in Suggests). No state filter is applied; out-of-state anglers receive their actual county name. NA or unmappable zip codes appear as "Unknown" for data quality visibility. Sort order: "Unknown" last; remaining rows sorted by n descending.

Value

A data.frame with class c("creel_summary_county", "data.frame") and columns: county (character), n (integer), pct (numeric, 1 decimal). NA or unmappable zip codes appear as "Unknown".

See Also

Other "Reporting & Diagnostics": `adjust_nonresponse()`, `check_completeness()`, `compare_variance()`, `flag_outliers()`, `season_summary()`, `standardize_species()`, `summarize_boat_composition()`, `summarize_by_angler_type()`, `summarize_by_day_type()`, `summarize_by_method()`, `summarize_by_species_sought()`, `summarize_by_trip_length()`, `summarize_by_zip()`, `summarize_cws_rates()`, `summarize_hws_rates()`, `summarize_length_freq()`, `summarize_refusals()`, `summarize_successful_parties()`, `summarize_trips()`, `summary.creel_estimates()`, `tidy.creel_estimates()`, `validate_creel_data()`, `validate_design()`, `validate_incomplete_trips()`, `validation_report()`, `write_estimates()`

Examples

```
data(example_calendar)
data(example_interviews)

# The shipped interviews carry no zip code, so add one to demonstrate the
# mapping. Two NAs are left in on purpose: an unmappable zip is reported as
# "Unknown" rather than dropped.
interviews_zip <- example_interviews
interviews_zip$zip_code <- rep(
  c("68502", "68508", NA), length.out = nrow(interviews_zip)
)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, interviews_zip,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)

summarize_by_county(design)
```

`summarize_by_day_type` *Tabulate interviews by day type and month*

Description

Counts the number of interviews in each day type stratum (e.g., weekday, weekend) within each calendar month. The day type column is resolved from the design's strata: a stratum named `day_type` when the design declares one, otherwise the first stratum column, which warns when the design declares more than one. Pass `day_type_col` to state the column outright.

Usage

```
summarize_by_day_type(design, day_type_col = NULL)
```

Arguments

<code>design</code>	A <code>creel_design</code> object with interviews attached.
<code>day_type_col</code>	Name of the column holding the day type, as a single string. When <code>NULL</code> (the default) it is resolved from the design's strata as described above.

Details

Interview-based summary, not pressure-weighted. This function tabulates raw interview records without applying survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates, use [estimate_catch_rate](#) or [estimate_harvest_rate](#).

Value

A data.frame with class `c("creel_summary_day_type", "data.frame")` and columns: month, day_type, N, percent.

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary_creel_estimates\(\)](#), [tidy_creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
summarize_by_day_type(d)
```

summarize_by_method	<i>Tabulate interviews by fishing method and month</i>
---------------------	--

Description

Counts the number of interviews for each fishing method within each calendar month. Method is taken from the column set via `add_interviews(angler_method = ...)`.

Usage

```
summarize_by_method(design)
```

Arguments

design	A <code>creel_design</code> object with interviews attached and <code>angler_method</code> column set via <code>add_interviews(angler_method = ...)</code> .
--------	--

Details

Interview-based summary, not pressure-weighted. This function tabulates raw interview records without applying survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates, use [estimate_catch_rate](#) or [estimate_harvest_rate](#).

Value

A data.frame with class `c("creel_summary_method", "data.frame")` and columns: month, method, N, percent.

Unrecorded grouping values

An interview whose grouping value was not recorded is reported under "Unknown", sorted last, rather than dropped. The interview is real and its grouping value is missing, which is not the same as the interview not existing, so `sum(N)` always equals the number of interviews attached to the design. "Unknown" is a label for the absence, never a category anyone selected. This matches [summarize_by_zip](#) and [summarize_by_county](#); the survey-weighted estimators use `<unknown>` instead.

A column holding both unrecorded values and the literal value "Unknown" warns: the two are pooled into one row and cannot be told apart in the output. Missingness is tracked internally, so a category genuinely named "Unknown" keeps its own counts.

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary_creel_estimates\(\)](#), [tidy_creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, angler_method = angler_method
)
summarize_by_method(d)
```

summarize_by_species_sought

Tabulate interviews by species sought and month

Description

Counts the number of interviews for each species sought within each calendar month. Species sought is taken from the column set via `add_interviews(species_sought = ...)`.

Usage

```
summarize_by_species_sought(design)
```

Arguments

`design` A `creel_design` object with interviews attached and `species_sought` column set via `add_interviews(species_sought = ...)`.

Details

Interview-based summary, not pressure-weighted. This function tabulates raw interview records without applying survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates, use [estimate_catch_rate](#) or [estimate_harvest_rate](#).

Value

A `data.frame` with class `c("creel_summary_species_sought", "data.frame")` and columns: month, species, N, percent.

Unrecorded grouping values

An interview whose grouping value was not recorded is reported under "Unknown", sorted last, rather than dropped. The interview is real and its grouping value is missing, which is not the same as the interview not existing, so `sum(N)` always equals the number of interviews attached to the design. "Unknown" is a label for the absence, never a category anyone selected. This matches [summarize_by_zip](#) and [summarize_by_county](#); the survey-weighted estimators use `<unknown>` instead.

A column holding both unrecorded values and the literal value "Unknown" warns: the two are pooled into one row and cannot be told apart in the output. Missingness is tracked internally, so a category genuinely named "Unknown" keeps its own counts.

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#),

```

summarize_length_freq(), summarize_refusals(), summarize_successful_parties(), summarize_trips(),
summary.creel_estimates(), tidy.creel_estimates(), validate_creel_data(), validate_design(),
validate_incomplete_trips(), validation_report(), write_estimates()

```

Examples

```

data(example_calendar)
data(example_interviews)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, species_sought = species_sought
)
summarize_by_species_sought(d)

```

```
summarize_by_trip_length
```

Tabulate interviews by trip length bin

Description

Bins trip durations (in hours) into 1-hour intervals from 0 to 10 hours, with a final bin for trips 10+ hours. Returns counts and percentages for each bin.

Usage

```
summarize_by_trip_length(design)
```

Arguments

design	A creel_design object with interviews attached and trip_duration column set via add_interviews(trip_duration = ...).
--------	--

Details

Interview-based summary, not pressure-weighted. This function tabulates raw interview records without applying survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates, use [estimate_catch_rate](#) or [estimate_harvest_rate](#).

Value

A data.frame with class c("creel_summary_trip_length", "data.frame") and columns: trip_length_bin (ordered factor), N (integer), percent (numeric, 1 decimal). Bins: "[0,1)", "[1,2)", ..., "[9,10)", "10+".

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, trip_duration = trip_duration
)
summarize_by_trip_length(d)
```

summarize_by_zip	<i>Tabulate interviews by zip code of origin</i>
------------------	--

Description

Counts and computes the percent of interviews by angler zip code of origin. NA zip codes appear as an explicit "Unknown" row. Percent denominator is total interviews including NA rows.

Usage

```
summarize_by_zip(design, zip_col = "zip_code")
```

Arguments

design	A creel_design object with interviews attached via add_interviews .
zip_col	Name of the interview column holding the angler zip code. Defaults to "zip_code". Name whatever your source calls it – no organisation's raw field name is assumed.

Details

Interview-based summary, not pressure-weighted. NA zip codes appear as an explicit "Unknown" row for data quality visibility. Sort order: "Unknown" last; remaining rows sorted by n descending.

Value

A data.frame with class `c("creel_summary_zip", "data.frame")` and columns: `zip_code` (character), `n` (integer), `pct` (numeric, 1 decimal). NA zip codes appear as "Unknown". Percent denominator is total interviews including NA.

See Also

Other "Reporting & Diagnostics": `adjust_nonresponse()`, `check_completeness()`, `compare_variance()`, `flag_outliers()`, `season_summary()`, `standardize_species()`, `summarize_boat_composition()`, `summarize_by_angler_type()`, `summarize_by_county()`, `summarize_by_day_type()`, `summarize_by_method()`, `summarize_by_species_sought()`, `summarize_by_trip_length()`, `summarize_cws_rates()`, `summarize_hws_rates()`, `summarize_length_freq()`, `summarize_refusals()`, `summarize_successful_parties()`, `summarize_trips()`, `summary.creel_estimates()`, `tidy.creel_estimates()`, `validate_creel_data()`, `validate_design()`, `validate_incomplete_trips()`, `validation_report()`, `write_estimates()`

Examples

```
data(example_calendar, package = "tidycreel")
data(example_interviews, package = "tidycreel")
example_interviews$zip_code <- rep_len(
  c("68502", "68502", NA, "68508", NA),
  nrow(example_interviews)
)
d <- suppressWarnings(
  creel_design(example_calendar, date = date, strata = day_type)
)
d <- suppressWarnings(
  add_interviews(d, example_interviews,
    catch = catch_total, effort = hours_fished, harvest = catch_kept,
    trip_status = trip_status, trip_duration = trip_duration,
    angler_type = angler_type, angler_method = angler_method,
    species_sought = species_sought, n_anglers = n_anglers, refused = refused
  )
)
summarize_by_zip(d)
```

<code>summarize_cws_rates</code>	<i>Compute caught-while-sought (CWS) rates by group</i>
----------------------------------	---

Description

Computes mean caught-while-sought rates (fish per angler-hour) for anglers targeting each species. For each interview, the rate is: `caught_count` / `angler_effort` where `caught_count` is the total number of fish caught of the species the angler was seeking, and `angler_effort` is angler-hours (effort x `n_anglers`, standardized at design time by `add_interviews`).

Usage

```
summarize_cws_rates(design, by = NULL, conf_level = 0.95)
```

Arguments

design	A creel_design object with interviews attached via add_interviews (with species_sought) and species catch data attached via add_catch .
by	Optional tidy selector for grouping columns from design\$interviews. Common choices: by = species_sought (CWS-03), by = c(month, species_sought) (CWS-02), by = c(month, angler_type, species_sought) (CWS-01). When NULL, returns a single overall rate across all interviews.
conf_level	Numeric confidence level for the t-interval. Default 0.95.

Details

Interview-based summary, not pressure-weighted. This function computes a simple arithmetic mean over sampled interviews. It does NOT apply survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates use [estimate_catch_rate](#).

The catch filter ensures only species the angler was targeting are counted (i.e., rows in design\$catch where catch_type == "caught" and species == species_sought).

Value

A data.frame with class c("creel_summary_cws_rates", "data.frame") and columns: group-ing columns (if any), N (integer, interviews per group that produced a rate), n_unknown_target (integer, interviews excluded because their sought species was not recorded), n_unknown_effort (integer, interviews excluded because their effort was not recorded), n_nonpositive_effort (integer, interviews excluded because their effort was zero or negative), mean_rate (numeric, mean fish/angler-hour, NA when N is 0), se (numeric, standard error), ci_lower, ci_upper.

Unrecorded grouping values

An interview whose value for a by column was not recorded is reported under "Unknown", sorted last, rather than dropped. Dropping it removed the interview from the result entirely, so the remaining groups lost their own members and their rates were computed on the survivors – on the shipped example data that moved one group's mean rate from 0.393 to 0.762 while the table still looked complete.

A group with no interview left to rate – which happens when every one of its members had an unrecorded target, see below – reports NA for mean_rate, se and the interval, and keeps its row rather than disappearing.

Interviews with an unrecorded sought species

These are **excluded** from the rate and counted in n_unknown_target.

The numerator counts fish of the species the party was targeting. With no target recorded nothing in the catch table can match, so such an interview falls through the join exactly as a party that caught none of its target does, and it used to be scored the same way – as a zero. That asserted these parties

caught none of something nobody recorded, and it dragged down every group they belonged to: on the shipped example data, blanking the sought species on 7 of 22 interviews took the boat group's mean rate from 0.393 to 0.254 with N unchanged at 9.

Excluding them makes the estimand **the rate among parties with a known target**. That equals the rate among all parties only if the target went unrecorded independently of what was caught, which is an assumption about the data rather than about the code – so `n_unknown_target` is reported beside every rate and a reader can judge it. A party that genuinely caught none of a *recorded* target is a real zero and still counts, per [add_catch](#).

An interview whose **effort** was not recorded is treated the same way and counted in `n_unknown_effort`. A rate needs an effort to divide by, and one unrecorded effort used to turn the whole group's mean into NA while N went on counting it. The two counts are mutually exclusive, target first, so an interview missing both is counted once.

An effort that is not **positive** cannot produce a rate either, and those interviews are counted in `n_nonpositive_effort`. A zero is a real record – a party interviewed before it started fishing – and a negative one is a data error that [add_interviews](#) already warns about; neither yields a rate. They used to be dropped with no trace at all, so a table could report 20 of 22 interviews with nothing in it to say the other two existed.

N therefore counts the interviews that produced a rate, and the accounting closes:

```
N + n_unknown_target + n_unknown_effort + n_nonpositive_effort
  == interviews in the group
```

The three exclusion counts are mutually exclusive, in that precedence, so an interview missing more than one thing is counted once.

A column holding both unrecorded values and the literal value "Unknown" warns: the two are pooled into one row and cannot be told apart in the output.

See Also

[summarize_hws_rates\(\)](#), [estimate_catch_rate\(\)](#)

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_catch)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, species_sought = species_sought
)
```

```
d <- add_catch(d, example_catch,
  catch_uid = interview_id, interview_uid = interview_id,
  species = species, count = count, catch_type = catch_type
)
summarize_cws_rates(d, by = species_sought)
```

summarize_hws_rates	<i>Compute harvested-while-sought (HWS) rates by group</i>
---------------------	--

Description

Computes mean harvested-while-sought rates (fish per angler-hour) for anglers targeting each species. For each interview, the rate is: `harvested_count / angler_effort` where `harvested_count` is the total number of fish harvested (kept) of the species the angler was seeking, and `angler_effort` is angler-hours (effort x `n_anglers`, standardized at design time by [add_interviews](#)).

Usage

```
summarize_hws_rates(design, by = NULL, conf_level = 0.95)
```

Arguments

<code>design</code>	A <code>creel_design</code> object with interviews attached via add_interviews (with <code>species_sought</code>) and species catch data attached via add_catch .
<code>by</code>	Optional tidy selector for grouping columns from <code>design\$interviews</code> . Common choices: <code>by = species_sought</code> (HWS-03), <code>by = c(month, species_sought)</code> (HWS-02), <code>by = c(month, angler_type, species_sought)</code> (HWS-01). When <code>NULL</code> , returns a single overall rate across all interviews.
<code>conf_level</code>	Numeric confidence level for the t-interval. Default 0.95.

Details

Interview-based summary, not pressure-weighted. This function computes a simple arithmetic mean over sampled interviews. It does NOT apply survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates use [estimate_harvest_rate](#).

The catch filter ensures only species the angler was targeting are counted (i.e., rows in `design$catch` where `catch_type == "harvested"` and `species == species_sought`).

Value

A data.frame with class `c("creel_summary_hws_rates", "data.frame")` and columns: grouping columns (if any), `N` (integer, interviews per group that produced a rate), `n_unknown_target` (integer, interviews excluded because their sought species was not recorded), `n_unknown_effort` (integer, interviews excluded because their effort was not recorded), `n_nonpositive_effort` (integer, interviews excluded because their effort was zero or negative), `mean_rate` (numeric, mean fish/angler-hour, NA when `N` is 0), `se` (numeric, standard error), `ci_lower`, `ci_upper`.

Unrecorded grouping values

An interview whose value for a by column was not recorded is reported under "Unknown", sorted last, rather than dropped. Dropping it removed the interview from the result entirely, so the remaining groups lost their own members and their rates were computed on the survivors – on the shipped example data that moved one group's mean rate from 0.393 to 0.762 while the table still looked complete.

A group with no interview left to rate – which happens when every one of its members had an unrecorded target, see below – reports NA for mean_rate, se and the interval, and keeps its row rather than disappearing.

Interviews with an unrecorded sought species

These are **excluded** from the rate and counted in n_unknown_target.

The numerator counts fish of the species the party was targeting. With no target recorded nothing in the catch table can match, so such an interview falls through the join exactly as a party that caught none of its target does, and it used to be scored the same way – as a zero. That asserted these parties caught none of something nobody recorded, and it dragged down every group they belonged to: on the shipped example data, blanking the sought species on 7 of 22 interviews took the boat group's mean rate from 0.393 to 0.254 with N unchanged at 9.

Excluding them makes the estimand **the rate among parties with a known target**. That equals the rate among all parties only if the target went unrecorded independently of what was caught, which is an assumption about the data rather than about the code – so n_unknown_target is reported beside every rate and a reader can judge it. A party that genuinely caught none of a *recorded* target is a real zero and still counts, per [add_catch](#).

An interview whose **effort** was not recorded is treated the same way and counted in n_unknown_effort. A rate needs an effort to divide by, and one unrecorded effort used to turn the whole group's mean into NA while N went on counting it. The two counts are mutually exclusive, target first, so an interview missing both is counted once.

An effort that is not **positive** cannot produce a rate either, and those interviews are counted in n_nonpositive_effort. A zero is a real record – a party interviewed before it started fishing – and a negative one is a data error that [add_interviews](#) already warns about; neither yields a rate. They used to be dropped with no trace at all, so a table could report 20 of 22 interviews with nothing in it to say the other two existed.

N therefore counts the interviews that produced a rate, and the accounting closes:

```
N + n_unknown_target + n_unknown_effort + n_nonpositive_effort
== interviews in the group
```

The three exclusion counts are mutually exclusive, in that precedence, so an interview missing more than one thing is counted once.

A column holding both unrecorded values and the literal value "Unknown" warns: the two are pooled into one row and cannot be told apart in the output.

See Also

[summarize_cws_rates\(\)](#), [estimate_harvest_rate\(\)](#)

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_catch)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, species_sought = species_sought
)
d <- add_catch(d, example_catch,
  catch_uid = interview_id, interview_uid = interview_id,
  species = species, count = count, catch_type = catch_type
)
summarize_hws_rates(d, by = species_sought)
```

`summarize_length_freq` *Compute length frequency distribution from creel interview data*

Description

Computes length frequency distributions (count, percent, cumulative percent) from fish length data attached via [add_lengths](#). Supports all fish (type = "catch"), harvested fish (type = "harvest"), and released fish (type = "release").

Usage

```
summarize_length_freq(design, type = "catch", by = NULL, bin_width = 1)
```

Arguments

<code>design</code>	A <code>creel_design</code> object with length data attached via add_lengths .
<code>type</code>	Character string specifying which fish to include. One of "catch" (all fish, harvest + release combined), "harvest" (kept fish only), or "release" (released fish only). Default "catch".

<code>by</code>	Optional tidy selector for grouping columns from <code>design\$lengths</code> . Common choice: <code>by = species</code> . When <code>NULL</code> , returns a single overall distribution.
<code>bin_width</code>	Positive numeric specifying the width of each length bin in the same units as the length data (typically mm). Default 1.

Details

Interview-based summary, not pressure-weighted. This function tabulates raw length measurements from sampled interviews without applying survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates use [estimate_catch_rate](#) or [estimate_harvest_rate](#).

Pre-binned release format: When length data was attached with `release_format = "binned"`, release rows have character bin labels (e.g., "350-400") and a count column. This function parses each bin label into a numeric midpoint and expands by count before applying `bin_width` binning. This allows a consistent `bin_width` to be applied to both individual and pre-binned data.

Value

A data.frame with class

A data.frame with class `c("creel_summary_length_freq", "data.frame")` and columns: grouping columns (if any), `length_bin` (ordered factor), `N` (integer, fish count per bin), `percent` (numeric, percent of group total), `cumulative_percent` (numeric, within group). Only bins with `N > 0` are returned. Percent values are rounded to 1 decimal place.

Unrecorded grouping values

A length record whose value for a `by` column was not recorded is reported under "Unknown", sorted last, rather than dropped. Dropping it removed the record from the distribution entirely, taking its weight with it – and a binned release row carries a *count* rather than one fish, so six dropped rows cost eleven fish on the shipped example data. The ungrouped total was never affected, which is what kept this invisible.

"Unknown" labels the absence; nothing is imputed and it is never a category anyone recorded. `sum(N)` equals the number of fish the lengths frame describes, grouped or not.

See Also

[add_lengths\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#)

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary_creel_estimates\(\)](#), [tidy_creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
```

```

data(example_lengths)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)
d <- add_lengths(d, example_lengths,
  length_uid = interview_id, interview_uid = interview_id,
  species = species, length = length,
  length_type = length_type, count = count,
  release_format = "binned"
)
summarize_length_freq(d, type = "harvest", by = species, bin_width = 25)
summarize_length_freq(d, type = "release", by = species)
summarize_length_freq(d, type = "catch")

```

summarize_refusals	<i>Tabulate refused vs accepted interviews by month</i>
--------------------	---

Description

Counts the number of refused and accepted interviews in each calendar month. Refusals are recorded as TRUE in the refused column set via [add_interviews](#).

Usage

```
summarize_refusals(design)
```

Arguments

design	A creel_design object with interviews attached and refused column set via <code>add_interviews(refused = ...)</code> .
--------	--

Details

Interview-based summary, not pressure-weighted. This function tabulates raw interview records without applying survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates, use [estimate_catch_rate](#) or [estimate_harvest_rate](#).

Value

A data.frame with class `c("creel_summary_refusals", "data.frame")` and columns: month (full month name), participation ("accepted" or "refused"), N (integer count), percent (numeric, rounded to 1 decimal, percent within month).

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, refused = refused
)
summarize_refusals(d)
```

```
summarize_successful_parties
```

Tabulate successful parties by angler type and species sought

Description

A party is "successful" when its total catch of the species it was seeking (`species_sought`) is greater than zero. That total follows the model [add_catch](#) documents: the pair's "caught" row when it has one, and otherwise harvested + released, because a "caught" row is optional. Returns counts of successful and total parties for each angler type x species sought combination.

Usage

```
summarize_successful_parties(design)
```

Arguments

<code>design</code>	A <code>creel_design</code> object with interviews attached (including <code>angler_type</code> and <code>species_sought</code> columns) and catch data attached via add_catch .
---------------------	--

Details

A pair that records its own "caught" row keeps it even when that row is zero — a recorded catch of none is data, not an absence, and does not fall back to the dispositions.

Interview-based summary, not pressure-weighted. This function tabulates raw interview records without applying survey weighting by sampling effort or effort stratum. For pressure-weighted extrapolated estimates, use [estimate_catch_rate](#) or [estimate_harvest_rate](#).

Value

A data.frame with class `c("creel_summary_successful_parties", "data.frame")` and columns: `angler_type`, `species_sought`, `N_successful` (integer, NA where success is not determinable), `N_total` (integer), `percent` (numeric, 1 decimal, NA likewise).

Unrecorded grouping values

An interview whose `angler_type` or `species_sought` was not recorded is reported under "Unknown", sorted last, rather than dropped, so `sum(N_total)` always equals the number of interviews attached to the design.

The two are not equivalent. A party is successful when it caught some of the species it *sought*, so where the sought species is unrecorded there is nothing to compare the catch against and success is **not determinable**: those rows report NA for `N_successful` and `percent`, never 0, which would assert that the parties failed. An unrecorded *angler type* leaves success perfectly determinable – only the reporting group is unknown – so those rows carry real counts. So does a sought species genuinely *recorded* as "Unknown": that is a real answer, not a missing one, and it keeps its own counts.

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
data(example_catch)
d <- creel_design(example_calendar, date = date, strata = day_type)
d <- add_interviews(d, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status, angler_type = angler_type,
  species_sought = species_sought
)
d <- add_catch(d, example_catch,
  catch_uid = interview_id, interview_uid = interview_id,
  species = species, count = count, catch_type = catch_type
)
summarize_successful_parties(d)
```

summarize_trips	<i>Summarize trip metadata for interview data</i>
-----------------	---

Description

Provides a diagnostic summary of trip completion status and duration statistics for interview data attached to a creel design. Useful for inspecting data quality before estimation.

Usage

```
summarize_trips(design)
```

Arguments

design A creel_design object with interviews attached via [add_interviews\(\)](#).

Value

A list (class "creel_trip_summary") with components:

n_total Total number of interviews

n_complete Number of complete trip interviews

n_incomplete Number of incomplete trip interviews

pct_complete Percentage of complete trips

pct_incomplete Percentage of incomplete trips

duration_stats Data frame with duration statistics by trip status

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_interviews)
```

```
design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_interviews(design, example_interviews,
  catch = catch_total,
  effort = hours_fished,
  harvest = catch_kept,
```

```

    trip_status = trip_status,
    trip_duration = trip_duration
  )
summary <- summarize_trips(design)
print(summary)

```

summary.creel_estimates

Summarise creel survey estimates as a formatted table

Description

summary.creel_estimates() converts a creel_estimates object into a creel_summary table with human-readable column names, suitable for display or export.

Usage

```

## S3 method for class 'creel_estimates'
summary(object, digits = 4L, ...)

```

Arguments

object	A creel_estimates object returned by estimate_effort(), estimate_catch_rate(), estimate_harvest_rate(), estimate_total_catch(), or estimate_total_harvest().
digits	Integer number of significant digits for numeric columns (default: 4).
...	Additional arguments (currently ignored).

Value

A creel_summary S3 object (a list) with components:

table	A data.frame with columns: any grouping variables, Estimate, SE, CI Lower, CI Upper, N.
method	Character string — the estimation method.
variance_method	Character string — the variance method.
conf_level	Numeric confidence level (e.g. 0.95).

See Also

[estimate_effort\(\)](#), [estimate_catch_rate\(\)](#), [estimate_harvest_rate\(\)](#)

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
data(example_calendar)
data(example_counts)
data(example_interviews)

design <- creel_design(example_calendar, date = date, strata = day_type)
design <- add_counts(design, example_counts)
design <- add_interviews(design, example_interviews,
  catch = catch_total, effort = hours_fished, harvest = catch_kept,
  trip_status = trip_status
)

est <- estimate_effort(design)
summary(est)
as.data.frame(summary(est))
```

 theme_creel

Package-standard ggplot2 theme for tidycreel plots

Description

theme_creel() applies a light, publication-friendly theme aligned with the package website colours. It is designed to be a stable default for tidycreel examples, vignettes, and autoplot() methods.

Usage

```
theme_creel(base_size = 11, base_family = "sans")
```

Arguments

base_size	Base text size. Default 11.
base_family	Base font family. Default "sans".

Value

A ggplot2 theme object.

See Also

Other "Visualisation": [autoplot.creel_estimates\(\)](#), [autoplot.creel_length_distribution\(\)](#), [autoplot.creel_schedule\(\)](#), [creel_palette\(\)](#), [plot_design\(\)](#)

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
    ggplot2::geom_point(colour = creel_palette()[["primary"]]) +
    theme_creel()
}
```

`tidy.creel_estimates` *Tidy a creel_estimates object into a flat tibble*

Description

Tidy a `creel_estimates` object into a flat tibble

Usage

```
## S3 method for class 'creel_estimates'
tidy(x, ...)
```

Arguments

`x` A `creel_estimates` object.

`...` Unused; reserved for future arguments.

Value

A tibble with one row per estimate. All columns from `x$estimates` are returned, plus `n` padded to `NA_integer_` when the estimator does not produce a sample size (e.g. mark-recapture harvest). Guaranteed columns: `estimate`, `se`, `ci_lower`, `ci_upper`, `n`.

Lossy for uncertainty components. Named components – the `x$se_components` list, and the `se_expansion` slot it mirrors – stay on the object and are deliberately not returned as columns. The contract distinguishes a component that does not apply or was never propagated (an absent name) from one that applies but is unknown (NA), and a tibble column collapses both to NA – reintroducing exactly the ambiguity the absent name was chosen to prevent. Read them from the object (`x$se_components`) or from `print()`, which also states each component’s relationship to `se`.

Note also that `se_between` and `se_within`, where present, do not reconstruct `se` on designs carrying a party-size expansion: the expansion term is a third contribution and is not among the visible columns.

See Also

[write_estimates](#)

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#),

```

summarize_by_species_sought(), summarize_by_trip_length(), summarize_by_zip(), summarize_cws_rates(),
summarize_hws_rates(), summarize_length_freq(), summarize_refusals(), summarize_successful_parties(),
summarize_trips(), summary_creel_estimates(), validate_creel_data(), validate_design(),
validate_incomplete_trips(), validation_report(), write_estimates()

```

validate_creel_data	<i>Validate creel survey data frames</i>
---------------------	--

Description

Runs field-level schema and quality checks on counts and/or interview data frames, returning a tidy results tibble with a pass/warn/fail verdict per column check. A print method renders a colour-coded cli summary.

Usage

```

validate_creel_data(
  counts = NULL,
  interviews = NULL,
  na_threshold = 0.1,
  date_range = c(as.Date("1970-01-01"), as.Date("2100-12-31"))
)

```

Arguments

counts	A data frame of count (effort) observations, or NULL to skip.
interviews	A data frame of interview observations, or NULL to skip.
na_threshold	Numeric scalar in $[0, 1]$. Columns with an NA rate above this threshold receive a "warn" status. Default 0.10.
date_range	A length-2 Date vector giving the earliest and latest plausible dates. Default <code>c(as.Date("1970-01-01"), as.Date("2100-12-31"))</code> .

Details

Checks performed for **every** column:

- Type check - column class is reported.
- NA rate - warns if $> na_threshold$ (default 0.10) of values are NA.

Additional checks based on detected column role:

- **Date columns** - values must fall within `date_range` (defaults to 1970-01-01 - 2100-12-31); warns on future dates.
- **Numeric columns** - warns if any value is negative (effort/count should be ≥ 0).
- **Character/factor columns** - warns if any value is an empty string.

Value

An object of class `creel_data_validation` - a tibble with columns:

`table` Which input was checked: "counts" or "interviews".

`column` Column name.

`check` Short check label (e.g. "na_rate", "negative_values", "type").

`status` "pass", "warn", or "fail".

`detail` Human-readable detail string.

See Also

[validate_creel_schedule\(\)](#) for schedule-specific validation.

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```
counts <- data.frame(
  date      = as.Date(c("2024-06-01", "2024-06-02")),
  day_type  = c("weekday", "weekend"),
  count     = c(10L, NA_integer_)
)
interviews <- data.frame(
  date      = as.Date(c("2024-06-01", "2024-06-02")),
  fish_kept = c(2L, -1L),
  species   = c("walleye", "")
)
res <- validate_creel_data(counts, interviews)
print(res)
```

`validate_creel_schedule`

Validate a creel_schedule object

Description

Checks that a `creel_schedule` (or plain data frame intended for use with [creel_design\(\)](#)) has the required columns, correct types, and sensible values. Called by [read_schedule\(\)](#) after coercion and available for users to validate hand-constructed schedules.

Usage

```
validate_creel_schedule(data)
```

Arguments

data A data frame to validate.

Value

Invisibly returns the input data frame on success. Aborts with an informative error message on validation failure.

See Also

Other "Scheduling": [attach_count_times\(\)](#), [generate_bus_schedule\(\)](#), [generate_count_times\(\)](#), [generate_progressive_start\(\)](#), [generate_schedule\(\)](#), [new_creel_schedule\(\)](#), [read_schedule\(\)](#), [write_schedule\(\)](#)

Examples

```
sched <- generate_schedule(
  start_date = "2024-06-01",
  end_date   = "2024-06-14",
  n_periods  = 1,
  sampling_rate = c(weekday = 0.3, weekend = 0.6),
  seed       = 42
)
validate_creel_schedule(sched)
```

validate_creel_schema *Validate a creel_schema object*

Description

Checks that all columns required for the schema's survey_type are mapped (non-NULL). Aborts with an informative cli_abort() listing each missing column and its table.

Usage

```
validate_creel_schema(schema)
```

Arguments

schema A creel_schema object created by [creel_schema\(\)](#).

Value

invisible(schema) if all required columns are mapped.

See Also

Other "Survey Design": [add_catch\(\)](#), [add_counts\(\)](#), [add_interviews\(\)](#), [add_lengths\(\)](#), [add_sections\(\)](#), [as_creel_svydesign\(\)](#), [as_hybrid_svydesign\(\)](#), [compute_angler_effort\(\)](#), [compute_effort\(\)](#), [creel_design\(\)](#), [creel_schema\(\)](#), [creel_vocabulary\(\)](#), [derive_angler_count\(\)](#), [est_effort_camera\(\)](#), [impute_camera_counts\(\)](#), [mean_party_size\(\)](#), [prep_counts_boat_party\(\)](#), [prep_counts_daily_effort\(\)](#), [prep_interview_catch\(\)](#), [prep_interviews_trips\(\)](#)

Examples

```
# A schema names the source columns for each table its survey type needs.
schema <- creel_schema(
  survey_type      = "instantaneous",
  interview_uid_col = "interview_id",
  date_col         = "date",
  trip_status_col  = "trip_status",
  effort_col       = "hours_fished",
  catch_col        = "catch_total",
  catch_uid_col    = "catch_id",
  species_col      = "species",
  catch_count_col  = "count",
  catch_type_col   = "catch_type",
  length_uid_col   = "length_id",
  length_mm_col    = "length",
  length_type_col  = "length_type",
  count_time_col   = "count_time",
  bank_anglers_col = "bank_anglers",
  count_col        = "angler_count"
)
validate_creel_schema(schema)

# An incomplete schema is refused here rather than failing later at a join.
try(validate_creel_schema(creel_schema(survey_type = "instantaneous")))
```

 validate_design

 Validate a proposed creel survey design against sample size targets

Description

Pre-season design check: runs [creel_n_effort\(\)](#) and [creel_n_cpue\(\)](#) per stratum and returns a pass/warn/fail status report.

Usage

```
validate_design(
  N_h,
  ybar_h,
  s2_h,
  n_proposed,
```

```

    cv_target,
    type = c("effort", "cpue"),
    cv_catch = NULL,
    cv_effort = NULL,
    rho = 0
  )

```

Arguments

<code>N_h</code>	Named numeric vector. Total available sampling days per stratum.
<code>ybar_h</code>	Numeric vector (same length as <code>N_h</code>). Pilot mean effort per day per stratum.
<code>s2_h</code>	Numeric vector (same length as <code>N_h</code>). Pilot variance of effort per stratum.
<code>n_proposed</code>	Named integer vector (same length as <code>N_h</code>). Proposed sampling days per stratum.
<code>cv_target</code>	Numeric scalar. Target CV for the effort estimate.
<code>type</code>	Character. One of "effort" or "cpue". Default "effort".
<code>cv_catch</code>	Numeric scalar. Required when <code>type = "cpue"</code> .
<code>cv_effort</code>	Numeric scalar. Required when <code>type = "cpue"</code> .
<code>rho</code>	Numeric scalar. Correlation between catch and effort. Default 0.

Value

A `creel_design_report` object (S3 list) with:

\$results tibble with columns `stratum`, `status`, `n_proposed`, `n_required`, `cv_actual`, `cv_target`, `message`

\$passed logical – TRUE if all strata status == "pass"

\$survey_type character

See Also

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_incomplete_trips\(\)](#), [validation_report\(\)](#), [write_estimates\(\)](#)

Examples

```

validate_design(
  N_h      = c(weekday = 65L, weekend = 28L),
  ybar_h    = c(weekday = 50, weekend = 60),
  s2_h      = c(weekday = 400, weekend = 500),
  n_proposed = c(weekday = 20L, weekend = 12L),
  cv_target  = 0.15
)

```

`validate_incomplete_trips`*Validate incomplete trip estimates using TOST equivalence testing*

Description

Performs Two One-Sided Tests (TOST) to determine if incomplete trip CPUE estimates are statistically equivalent to complete trip estimates within a specified threshold. Returns validation results with recommendations for whether incomplete trips are appropriate for estimation in the given dataset.

Usage

```
validate_incomplete_trips(  
  design,  
  catch,  
  effort,  
  by = NULL,  
  variance = "taylor",  
  conf_level = 0.95,  
  truncate_at = 0.5  
)
```

Arguments

<code>design</code>	A <code>creel_design</code> object with interviews attached via add_interviews . Must include <code>trip_status</code> field to distinguish complete from incomplete trips.
<code>catch</code>	Bare column name for catch data (supports tidy evaluation)
<code>effort</code>	Bare column name for effort data (supports tidy evaluation)
<code>by</code>	Optional tidy selector for grouping variables. When provided, performs TOST for each group independently. Overall validation passes only if overall test AND all group tests pass equivalence.
<code>variance</code>	Character string specifying variance estimation method. Options: "taylor" (default), "bootstrap", or "jackknife". Passed to estimate_catch_rate .
<code>conf_level</code>	Numeric confidence level for confidence intervals (default: 0.95). Passed to estimate_catch_rate .
<code>truncate_at</code>	Numeric minimum trip duration (hours) for incomplete trip estimation. Default is 0.5 hours (30 minutes). Passed to estimate_catch_rate for MOR estimator.

Details

The function estimates CPUE separately for complete trips (using ratio-of-means) and incomplete trips (using mean-of-ratios) via [estimate_catch_rate](#). It then performs TOST to test equivalence within the specified threshold.

Equivalence bounds are calculated as $\pm \text{threshold} * \text{complete_trip_estimate}$. The difference variance is estimated using the delta method: $\text{Var}(\text{complete} - \text{incomplete}) = \text{Var}(\text{complete}) + \text{Var}(\text{incomplete})$, assuming independence between the two samples.

Sample size requirements: At least 10 complete trips AND 10 incomplete trips are required for stable variance estimation and TOST. The function errors if either sample size is insufficient.

Value

A `creel_tost_validation` S3 object (list) with components:

- `overall_test`: List with TOST results for overall comparison (`p_lower`, `p_upper`, `equivalence_passed`, `diff_estimate`, `equivalence_bounds`)
- `group_tests`: Data frame with per-group TOST results (only present for grouped estimation)
- `equivalence_threshold`: Numeric threshold used (e.g., 0.20 for $\pm 20\%$)
- `passed`: Logical indicating if overall AND all groups (if applicable) passed equivalence
- `recommendation`: Character string with usage recommendation
- `metadata`: List with complete and incomplete trip estimates, standard errors, confidence intervals, and sample sizes

Package Options

The package option `tidycreel.equivalence_threshold` controls the equivalence threshold (default: 0.20 = $\pm 20\%$ bounds for equivalence as $\pm \text{threshold} * \text{complete_trip_estimate}$. For example, with the default 20% equivalence bounds are 1.6 to 2.4 fish/hour. Users can set a custom threshold:

```
options(tidycreel.equivalence_threshold = 0.15)
```

TOST Equivalence Testing

TOST (Two One-Sided Tests) is the statistically appropriate method for proving similarity between two estimates. Unlike traditional hypothesis testing which tests for difference, TOST tests the null hypothesis that estimates differ by MORE than the threshold. Equivalence is concluded when both one-sided tests reject the null (both p-values < 0.05).

The two tests are:

1. $H_0: \text{complete} - \text{incomplete} \leq -\text{threshold}$ vs $H_1: \text{complete} - \text{incomplete} > -\text{threshold}$
2. $H_0: \text{complete} - \text{incomplete} \geq \text{threshold}$ vs $H_1: \text{complete} - \text{incomplete} < \text{threshold}$

Both tests must reject ($p < 0.05$) for equivalence. This ensures estimates are "close enough" to be considered equivalent for practical purposes.

Grouped Validation

When `by` is provided, the function performs TOST for each group independently AND for the overall (ungrouped) data. The validation passes only if ALL tests pass equivalence. This conservative approach prevents overlooking group-specific bias that could be masked by overall equivalence.

See Also

Other "Reporting & Diagnostics": `adjust_nonresponse()`, `check_completeness()`, `compare_variance()`, `flag_outliers()`, `season_summary()`, `standardize_species()`, `summarize_boat_composition()`, `summarize_by_angler_type()`, `summarize_by_county()`, `summarize_by_day_type()`, `summarize_by_method()`, `summarize_by_species_sought()`, `summarize_by_trip_length()`, `summarize_by_zip()`, `summarize_cws_rates()`, `summarize_hws_rates()`, `summarize_length_freq()`, `summarize_refusals()`, `summarize_successful_parties()`, `summarize_trips()`, `summary.creel_estimates()`, `tidy.creel_estimates()`, `validate_creel_data()`, `validate_design()`, `validation_report()`, `write_estimates()`

Examples

```
# Create design with both complete and incomplete trips
calendar <- data.frame(
  date = as.Date(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04")),
  day_type = rep(c("weekday", "weekend"), each = 2)
)
design <- creel_design(calendar, date = date, strata = day_type)

set.seed(123)
interviews <- data.frame(
  date = as.Date(rep(c("2024-06-01", "2024-06-02", "2024-06-03", "2024-06-04"), each = 25)),
  catch_total = rpois(100, lambda = 6),
  hours_fished = runif(100, min = 2, max = 4),
  trip_status = rep(c("complete", "incomplete"), each = 50),
  trip_duration = runif(100, min = 2, max = 4)
)

design_with_interviews <- add_interviews(design, interviews,
  catch = catch_total,
  effort = hours_fished,
  trip_status = trip_status,
  trip_duration = trip_duration
)

# Validate incomplete trips
result <- validate_incomplete_trips(design_with_interviews,
  catch = catch_total,
  effort = hours_fished
)
print(result)

# Grouped validation
result_grouped <- validate_incomplete_trips(design_with_interviews,
  catch = catch_total,
  effort = hours_fished,
  by = day_type
)
print(result_grouped)

# Custom equivalence threshold, restoring the previous option afterwards
old_opts <- options(tidycreelequivalence_threshold = 0.15) # 15% threshold
```

```

result_custom <- validate_incomplete_trips(design_with_interviews,
  catch = catch_total,
  effort = hours_fished
)
options(old_opts)

```

validation_report	<i>Generate a validation summary report</i>
-------------------	---

Description

Runs `validate_creel_data()` on counts and/or interviews, aggregates the results into a human-readable summary tibble (one row per table x check type), and optionally detects unrecognised species values via `standardize_species()`.

Usage

```

validation_report(
  counts = NULL,
  interviews = NULL,
  species_col = NULL,
  na_threshold = 0.1,
  date_range = c(as.Date("1970-01-01"), as.Date("2100-12-31"))
)

```

Arguments

<code>counts</code>	A data frame of count (effort) observations, or NULL.
<code>interviews</code>	A data frame of interview observations, or NULL.
<code>species_col</code>	Character scalar. If non-NULL and interviews is provided, calls <code>standardize_species()</code> on this column and appends a <code>species_coverage</code> row showing the fraction of rows successfully matched to an AFS code. Default NULL (no species check).
<code>na_threshold</code>	Passed to <code>validate_creel_data()</code> . Default 0.10.
<code>date_range</code>	Passed to <code>validate_creel_data()</code> . Default <code>c(as.Date("1970-01-01"), as.Date("2100-12-31"))</code> .

Details

The returned object is a `creel_validation_report` - a data frame with a custom print method that renders a colour-coded cli summary. It can be exported with `write_estimates()`.

Value

An object of class `creel_validation_report` - a data frame with columns:

`table` Source table: "counts", "interviews", or "species".
`check` Check type (e.g. "na_rate", "date_range").

n_pass Number of columns with "pass" status.

n_warn Number of columns with "warn" status.

n_fail Number of columns with "fail" status.

detail Comma-separated list of flagged columns, or "all ok".

See Also

[write_estimates\(\)](#)

Other "Reporting & Diagnostics": [adjust_nonresponse\(\)](#), [check_completeness\(\)](#), [compare_variance\(\)](#), [flag_outliers\(\)](#), [season_summary\(\)](#), [standardize_species\(\)](#), [summarize_boat_composition\(\)](#), [summarize_by_angler_type\(\)](#), [summarize_by_county\(\)](#), [summarize_by_day_type\(\)](#), [summarize_by_method\(\)](#), [summarize_by_species_sought\(\)](#), [summarize_by_trip_length\(\)](#), [summarize_by_zip\(\)](#), [summarize_cws_rates\(\)](#), [summarize_hws_rates\(\)](#), [summarize_length_freq\(\)](#), [summarize_refusals\(\)](#), [summarize_successful_parties\(\)](#), [summarize_trips\(\)](#), [summary.creel_estimates\(\)](#), [tidy.creel_estimates\(\)](#), [validate_creel_data\(\)](#), [validate_design\(\)](#), [validate_incomplete_trips\(\)](#), [write_estimates\(\)](#)

Examples

```
counts <- data.frame(
  date      = as.Date(c("2024-06-01", "2024-06-02")),
  day_type  = c("weekday", "weekend"),
  count     = c(10L, NA_integer_)
)
interviews <- data.frame(
  date      = as.Date(c("2024-06-01", "2024-06-02")),
  fish_kept = c(2L, -1L),
  species   = c("walleye", "")
)
rpt <- validation_report(counts, interviews, species_col = "species")
print(rpt)
```

write_estimates

Export creel survey estimates to a file

Description

Writes a `creel_estimates` or `creel_summary` object to a CSV or xlsx file. For CSV, a three-line comment block is prepended containing the estimation method, variance method, confidence level, and generation timestamp. For xlsx, the data are written directly (Excel does not support comment rows).

Usage

```
write_estimates(
  x,
  path,
  format = c("auto", "csv", "xlsx"),
  overwrite = FALSE,
  ...
)
```

Arguments

<code>x</code>	A <code>creel_estimates</code> or <code>creel_summary</code> object.
<code>path</code>	File path for the output. The extension (<code>.csv</code> or <code>.xlsx</code>) determines the format; alternatively, use the <code>format</code> argument to override.
<code>format</code>	One of <code>"csv"</code> (default) or <code>"xlsx"</code> . When <code>"csv"</code> , a comment header is prepended. When <code>"xlsx"</code> , <code>writexl::write_xlsx()</code> is used behind an <code>rlang::check_installed()</code> guard.
<code>overwrite</code>	Logical; if <code>FALSE</code> (default) an error is raised when path already exists.
<code>...</code>	Currently unused; reserved for future arguments.

Details

CSV format — The output file begins with comment lines starting with `#` that record survey meta-data:

```
# Survey estimates - tidycree1
# Method: Total Effort | Taylor linearization | 95% CI
# Generated: 2024-06-15 09:32:11 UTC
Estimate,SE,CI Lower,CI Upper,N
372.5,13.18,343.8,401.2,14
```

These lines can be skipped when reading back with `utils::read.csv(path, comment.char = "#")`.

xlsx format — The data are written without a comment header since Excel does not natively support comment rows. Row 1 will be the column headers.

Value

`path`, returned invisibly.

See Also

`summary.cree1_estimates()`, `write_schedule()`

Other "Reporting & Diagnostics": `adjust_nonresponse()`, `check_completeness()`, `compare_variance()`, `flag_outliers()`, `season_summary()`, `standardize_species()`, `summarize_boat_composition()`, `summarize_by_angler_type()`, `summarize_by_county()`, `summarize_by_day_type()`, `summarize_by_method()`, `summarize_by_species_sought()`, `summarize_by_trip_length()`, `summarize_by_zip()`, `summarize_cws_rates()`,

```

summarize_hws_rates(), summarize_length_freq(), summarize_refusals(), summarize_successful_parties(),
summarize_trips(), summary.creel_estimates(), tidy.creel_estimates(), validate_creel_data(),
validate_design(), validate_incomplete_trips(), validation_report()

```

Examples

```

data("example_counts")
data("example_interviews")
cal <- unique(example_counts[, c("date", "day_type")])
design <- suppressWarnings(
  creel_design(cal, date = date, strata = day_type) # nolint
)
design <- suppressWarnings(add_counts(design, example_counts))
design <- suppressWarnings(
  add_interviews(
    design, example_interviews,
    catch = catch_total, effort = hours_fished, n_anglers = n_anglers,
    trip_status = trip_status
  )
)
eff <- suppressWarnings(estimate_effort(design))

tmp <- tempfile(fileext = ".csv")
write_estimates(eff, tmp)

# Read back (skipping comment lines)
out <- utils::read.csv(tmp, comment.char = "#")
out

```

write_schedule	<i>Write a creel schedule to a CSV or xlsx file</i>
----------------	---

Description

Exports a `creel_schedule` object to disk. The default format is CSV using base R (no extra dependencies). The "xlsx" format requires the **writexl** package; an informative error is raised if it is not installed.

Usage

```
write_schedule(schedule, path, format = c("csv", "xlsx"), overwrite = FALSE)
```

Arguments

schedule	A <code>creel_schedule</code> object (or plain data frame) to export.
path	File path for the output file.

format	One of "csv" (default) or "xlsx". When "csv", the file is written with <code>utils::write.csv()</code> (no row names). When "xlsx", <code>writexl::write_xlsx()</code> is used behind an <code>rlang::check_installed()</code> guard.
overwrite	Logical. If FALSE (default), aborts with an error when path already exists. Set TRUE to replace an existing file.

Value

path, returned invisibly.

See Also

Other "Scheduling": `attach_count_times()`, `generate_bus_schedule()`, `generate_count_times()`, `generate_progressive_start()`, `generate_schedule()`, `new_creel_schedule()`, `read_schedule()`, `validate_creel_schedule()`

Examples

```
sched <- generate_schedule(
  "2024-06-01", "2024-08-31",
  n_periods = 2,
  sampling_rate = c(weekday = 0.3, weekend = 0.6),
  seed = 42
)
tmp <- tempfile(fileext = ".csv")
write_schedule(sched, tmp)
```

Index

* Estimation

estimate_angler_n, 72
estimate_exploitation_rate, 89
estimate_mr_harvest, 97

* datasets

creel_counts_toy, 47
creel_interviews_toy, 53
example_aerial_counts, 136
example_aerial_glm_counts, 137
example_aerial_interviews, 139
example_ages, 140
example_calendar, 141
example_camera_counts, 142
example_camera_interviews, 143
example_camera_timestamps, 145
example_catch, 146
example_counts, 148
example_ice_interviews, 149
example_ice_sampling_frame, 150
example_interviews, 152
example_lengths, 153
example_sections_calendar, 155
example_sections_counts, 156
example_sections_interviews, 157

add_ages, 5, 140, 141

add_ages(), 78, 94, 100, 117

add_catch, 6, 11, 16, 19, 21, 26, 30, 44, 45,
52, 64, 65, 71, 100, 102, 113, 116,
127, 180, 182, 192, 193, 195, 196,
229–232, 236, 245

add_catch(), 78, 94, 100, 103, 109, 113, 132,
196

add_counts, 7, 8, 16, 19, 21, 22, 26, 30, 44,
45, 52, 64, 65, 71, 82, 103, 108, 113,
127, 180, 182, 189, 192, 193, 195,
196, 213, 218, 245

add_counts(), 20, 21, 50, 62, 63, 68, 70, 71,
86, 126, 136, 138, 143, 148, 156,
179, 180, 191–193

add_interviews, 7, 11, 13, 19, 21, 22, 26, 30,
43–45, 52, 64, 65, 71, 78, 94, 100,
103, 108, 113, 127, 173, 180, 182,
192, 193, 195, 196, 213, 221,
227–232, 235, 245, 247

add_interviews(), 20, 21, 29, 44, 45, 107,
112, 116, 132, 139, 144, 150, 152,
158, 159, 173, 195, 238

add_lengths, 7, 11, 16, 18, 21, 26, 30, 44, 45,
52, 64, 65, 71, 127, 153, 154, 180,
182, 192, 193, 195, 196, 233, 245

add_lengths(), 5, 6, 63, 78, 94, 100, 130,
131, 234

add_sections, 7, 11, 16, 19, 20, 26, 30, 44,
45, 52, 64, 65, 71, 84, 104, 105, 109,
110, 114, 115, 127, 180, 182, 192,
193, 195, 196, 245

add_sections(), 155, 156

adjust_nonresponse, 22, 38, 43, 159, 208,
217, 219, 221–225, 227, 228, 230,
233, 234, 236–239, 241, 243, 246,
249, 251, 252

as.data.frame.creel_data_validation,
24

as.data.frame.creel_design_comparison,
24

as.data.frame.creel_validation_report,
25

as.data.frame.creel_variance_comparison,
25

as.Date(), 68

as_creel_svydesign, 7, 11, 16, 19, 21, 26,
30, 44, 45, 52, 64, 65, 71, 127, 180,
182, 192, 193, 195, 196, 245

as_hybrid_svydesign, 7, 11, 16, 19, 21, 26,
27, 44, 45, 52, 64, 65, 71, 127, 180,
182, 192, 193, 195, 196, 245

attach_count_times, 30, 166, 168, 170, 172,
183, 206, 244, 254

- `audit_strata`, 31, 41, 55, 57, 58, 61, 67, 185, 188, 207, 216
- `autoplot.creel_design_comparison`, 33
- `autoplot.creel_design_comparison()`, 41
- `autoplot.creel_estimates`, 34, 36, 37, 59, 186, 240
- `autoplot.creel_length_distribution`, 35, 35, 37, 59, 186, 240
- `autoplot.creel_schedule`, 35, 36, 36, 59, 186, 240
- `autoplot.creel_schedule()`, 186
- `calibrate`, 22
- `check_completeness`, 23, 37, 43, 159, 208, 217, 219, 221–225, 227, 228, 230, 233, 234, 236–239, 241, 243, 246, 249, 251, 252
- `cli_warn`, 189
- `compare_cpue_estimators`, 38, 81, 84, 87, 96, 102, 107, 112, 116, 118, 121, 123, 129, 132, 133, 135
- `compare_designs`, 33, 40, 55, 57, 58, 61, 67, 185, 188, 207, 216
- `compare_variance`, 23, 38, 41, 159, 208, 217, 219, 221–225, 227, 228, 230, 233, 234, 236–239, 241, 243, 246, 249, 251, 252
- `compute_angler_effort`, 7, 11, 16, 19, 21, 26, 30, 43, 45, 52, 64, 65, 71, 127, 180, 182, 192, 193, 195, 196, 245
- `compute_angler_effort()`, 45
- `compute_effort`, 7, 11, 16, 19, 21, 26, 30, 44, 45, 52, 64, 65, 71, 127, 180, 182, 192, 193, 195, 196, 245
- `compute_effort()`, 44, 195
- `creel_confidence_intervals`, 45
- `creel_counts_toy`, 47, 54, 136, 138, 139, 141–144, 146–148, 150–152, 154–156, 158
- `creel_design`, 6, 7, 11, 16, 18, 19, 21, 26, 30, 44, 45, 48, 64, 65, 71, 127, 180, 182, 192, 193, 195, 196, 213, 245
- `creel_design()`, 8, 14, 20, 21, 28, 29, 86, 128, 136, 138, 139, 141–143, 150, 151, 155, 171, 173–175, 185, 186, 206, 243
- `creel_interviews_toy`, 47, 53, 136, 138, 139, 141–144, 146–148, 150–152, 154–156, 158
- `creel_n_camera`, 33, 41, 54, 57, 58, 61, 67, 185, 188, 207, 216
- `creel_n_cpue`, 33, 41, 55, 56, 58, 61, 67, 185, 188, 207, 216
- `creel_n_cpue()`, 66, 67, 186–188
- `creel_n_effort`, 33, 41, 55, 57, 57, 61, 67, 185, 188, 207, 216
- `creel_n_effort()`, 55, 66, 67, 183–188
- `creel_palette`, 35–37, 59, 186, 240
- `creel_power`, 33, 41, 55, 57, 58, 59, 67, 185, 188, 207, 216
- `creel_power()`, 186–188
- `creel_schema`, 7, 11, 16, 19, 21, 26, 30, 44, 45, 52, 61, 65, 71, 127, 180, 182, 192, 193, 195, 196, 245
- `creel_schema()`, 29, 65, 244
- `creel_vocabulary`, 7, 11, 16, 19, 21, 26, 30, 44, 45, 52, 64, 65, 71, 127, 180, 182, 192, 193, 195, 196, 245
- `cv_from_n`, 33, 41, 55, 57, 58, 61, 66, 185, 188, 207, 216
- `day_length`, 67, 210, 212–214
- `day_length()`, 10
- `derive_angler_count`, 7, 11, 16, 19, 21, 26, 30, 44, 45, 52, 64, 65, 69, 127, 180, 182, 192, 193, 195, 196, 245
- `derive_angler_count()`, 11, 50, 182, 191
- `est_age_distribution`, 39, 81, 84, 87, 96, 102, 107, 112, 116, 117, 121, 123, 129, 132, 133, 135
- `est_age_distribution()`, 133
- `est_biomass`, 39, 81, 84, 87, 96, 102, 107, 112, 116, 118, 119, 123, 129, 132, 133, 135
- `est_biomass()`, 47, 131
- `est_compliance`, 39, 81, 84, 87, 96, 102, 107, 112, 116, 118, 121, 122, 129, 132, 133, 135
- `est_effort_camera`, 7, 11, 16, 19, 21, 26, 30, 44, 45, 52, 64, 65, 71, 84, 124, 180, 182, 192, 193, 195, 196, 245
- `est_effort_camera()`, 128, 180
- `est_effort_camera_mi`, 39, 81, 84, 87, 96, 102, 107, 112, 116, 118, 121, 123, 128, 132, 133, 135
- `est_effort_camera_mi()`, 179, 180

- `est_length_distribution`, 39, 81, 84, 87, 96, 102, 107, 112, 116, 118, 121, 123, 129, 130, 133, 135
- `est_length_distribution()`, 35, 36, 118–120, 122, 134
- `est_mean_age`, 39, 81, 84, 87, 96, 102, 107, 112, 116, 118, 121, 123, 129, 132, 133, 135
- `est_mean_length`, 39, 81, 84, 87, 96, 102, 107, 112, 116, 118, 121, 123, 129, 132, 133, 134
- `est_mean_length()`, 47
- `estimate_angler_n`, 72, 93, 97–99
- `estimate_angler_n()`, 47
- `estimate_angler_trips`, 76, 89
- `estimate_catch_rate`, 22, 39, 77, 84, 87, 95, 96, 101, 102, 107, 112, 116, 118, 121, 123, 129, 132, 133, 135, 139, 220, 223–226, 229, 234–236, 247
- `estimate_catch_rate()`, 35, 104, 139, 144, 157–159, 230, 239
- `estimate_effort`, 22, 39, 76, 77, 81, 82, 87–89, 96, 102, 107, 112, 116, 118, 121, 123, 129, 132, 133, 135
- `estimate_effort()`, 26, 29, 35, 47, 86, 104, 109, 114, 136, 148, 150, 156, 176, 177, 239
- `estimate_effort_aerial_glm`, 39, 81, 84, 85, 96, 102, 107, 112, 116, 118, 121, 123, 129, 132, 133, 135
- `estimate_effort_aerial_glm()`, 51, 138
- `estimate_effort_per_acre`, 88
- `estimate_exploitation_rate`, 75, 77, 89, 99
- `estimate_exploitation_rate()`, 47
- `estimate_harvest_rate`, 39, 81, 84, 87, 93, 100, 102, 107, 112, 116, 118, 121, 123, 129, 132, 133, 135, 220, 223–226, 231, 234–236
- `estimate_harvest_rate()`, 47, 109, 233, 239
- `estimate_mr_harvest`, 75, 93, 97
- `estimate_release_rate`, 39, 81, 84, 87, 96, 99, 107, 112, 116, 118, 121, 123, 129, 132, 133, 135
- `estimate_release_rate()`, 114
- `estimate_total_catch`, 39, 81, 84, 87, 90, 96, 102, 103, 112, 116, 118, 121, 123, 129, 132, 133, 135
- `estimate_total_catch()`, 47, 109, 114, 139, 144, 157, 158
- `estimate_total_harvest`, 39, 81, 84, 87, 90, 96, 102, 107, 108, 116, 118, 121, 123, 129, 132, 133, 135
- `estimate_total_harvest()`, 157
- `estimate_total_release`, 39, 81, 84, 87, 96, 102, 107, 112, 113, 118, 121, 123, 129, 132, 133, 135
- `example_aerial_counts`, 47, 54, 136, 138, 139, 141–144, 146–148, 150–152, 154–156, 158
- `example_aerial_glm_counts`, 47, 54, 136, 137, 139, 141–144, 146–148, 150–152, 154–156, 158
- `example_aerial_interviews`, 47, 54, 136, 138, 139, 141–144, 146–148, 150–152, 154–156, 158
- `example_ages`, 47, 54, 136, 138, 139, 140, 142–144, 146–148, 150–152, 154–156, 158
- `example_calendar`, 47, 54, 136, 138, 139, 141, 141, 143, 144, 146–148, 150–152, 154–156, 158
- `example_camera_counts`, 47, 54, 136, 138, 139, 141, 142, 142, 143, 144, 146–148, 150–152, 154–156, 158
- `example_camera_interviews`, 47, 54, 136, 138, 139, 141–143, 143, 146–148, 150–152, 154–156, 158
- `example_camera_timestamps`, 47, 54, 136, 138, 139, 141–144, 145, 147, 148, 150–152, 154–156, 158
- `example_catch`, 47, 54, 136, 138, 139, 141–144, 146, 146, 148, 150–152, 154–156, 158
- `example_counts`, 47, 54, 136, 138, 139, 141–144, 146, 147, 148, 150–152, 154–156, 158
- `example_ice_interviews`, 47, 54, 136, 138, 139, 141–144, 146–148, 149, 151, 152, 154–156, 158
- `example_ice_sampling_frame`, 47, 54, 136, 138, 139, 141–144, 146–150, 150, 152, 154–156, 158
- `example_interviews`, 47, 54, 136, 138–144, 146–148, 150, 151, 152, 153–156,

- [158](#)
- `example_lengths`, [47](#), [54](#), [136](#), [138](#), [139](#),
[141–144](#), [146–148](#), [150–152](#), [153](#),
[155](#), [156](#), [158](#)
- `example_sections_calendar`, [47](#), [54](#), [136](#),
[138](#), [139](#), [141–144](#), [146–148](#),
[150–152](#), [154](#), [155](#), [156–158](#)
- `example_sections_counts`, [47](#), [54](#), [136](#), [138](#),
[139](#), [141–144](#), [146–148](#), [150–152](#),
[154](#), [155](#), [156](#), [158](#)
- `example_sections_interviews`, [47](#), [54](#), [136](#),
[138](#), [139](#), [141–144](#), [146–148](#),
[150–152](#), [154–156](#), [157](#)
- `flag_outliers`, [23](#), [38](#), [43](#), [158](#), [208](#), [217](#),
[219](#), [221–225](#), [227](#), [228](#), [230](#), [233](#),
[234](#), [236–239](#), [241](#), [243](#), [246](#), [249](#),
[251](#), [252](#)
- `format.creel_completeness_report`, [160](#)
- `format.creel_design`, [160](#)
- `format.creel_design()`, [198](#)
- `format.creel_design_report`, [161](#)
- `format.creel_estimates`, [161](#)
- `format.creel_estimates_diagnostic`, [162](#)
- `format.creel_estimates_mor`, [162](#)
- `format.creel_schedule`, [163](#)
- `format.creel_schedule()`, [202](#)
- `format.creel_schema`, [163](#)
- `format.creel_schema()`, [202](#)
- `format.creel_season_summary`, [164](#)
- `format.creel_tost_validation`, [164](#)
- `format.creel_validation`, [165](#)
- `generate_bus_schedule`, [31](#), [165](#), [168](#), [170](#),
[172](#), [183](#), [206](#), [244](#), [254](#)
- `generate_bus_schedule()`, [37](#)
- `generate_count_times`, [31](#), [166](#), [167](#), [170](#),
[172](#), [183](#), [206](#), [244](#), [254](#)
- `generate_count_times()`, [31](#)
- `generate_progressive_start`, [31](#), [166](#), [168](#),
[169](#), [172](#), [183](#), [206](#), [244](#), [254](#)
- `generate_schedule`, [31](#), [166](#), [168](#), [170](#), [171](#),
[183](#), [206](#), [244](#), [254](#)
- `generate_schedule()`, [31](#), [37](#), [166](#)
- `get_enumeration_counts`, [173](#), [175–177](#)
- `get_enumeration_counts()`, [177](#)
- `get_inclusion_probs`, [173](#), [174](#), [176](#), [177](#)
- `get_inclusion_probs()`, [173](#), [177](#)
- `get_sampling_frame`, [173](#), [175](#), [175](#), [177](#)
- `get_sampling_frame()`, [173](#), [175](#), [177](#)
- `get_site_contributions`, [173](#), [175](#), [176](#),
[176](#)
- `ggplot2::theme_bw()`, [34](#), [35](#)
- `ggplot2::theme_set()`, [34](#), [35](#)
- `impute_camera_counts`, [7](#), [11](#), [16](#), [19](#), [21](#), [26](#),
[30](#), [44](#), [45](#), [52](#), [64](#), [65](#), [71](#), [127](#), [178](#),
[182](#), [192](#), [193](#), [195](#), [196](#), [245](#)
- `impute_camera_counts()`, [125](#), [128](#)
- `lme4::bootMer()`, [86](#)
- `lme4::glmer()`, [86](#)
- `lme4::glmer.nb()`, [86](#)
- `mean_party_size`, [7](#), [11](#), [16](#), [19](#), [21](#), [26](#), [30](#),
[44](#), [45](#), [52](#), [64](#), [65](#), [71](#), [127](#), [180](#), [181](#),
[192](#), [193](#), [195](#), [196](#), [245](#)
- `mean_party_size()`, [70](#), [71](#)
- `new_creel_schedule`, [31](#), [166](#), [168](#), [170](#), [172](#),
[182](#), [206](#), [244](#), [254](#)
- `optimal_n`, [33](#), [41](#), [55](#), [57](#), [58](#), [61](#), [67](#), [183](#),
[188](#), [207](#), [216](#)
- `plot_design`, [35–37](#), [59](#), [185](#), [240](#)
- `postStratify`, [22](#)
- `power_creel`, [33](#), [41](#), [55](#), [57](#), [58](#), [61](#), [67](#), [185](#),
[186](#), [207](#), [216](#)
- `prep_counts_boat_party`, [7](#), [11](#), [16](#), [19](#), [21](#),
[26](#), [30](#), [44](#), [45](#), [52](#), [64](#), [65](#), [71](#), [127](#),
[180](#), [182](#), [190](#), [193](#), [195](#), [196](#), [245](#)
- `prep_counts_boat_party()`, [71](#), [182](#)
- `prep_counts_daily_effort`, [7](#), [11](#), [16](#), [19](#),
[21](#), [26](#), [30](#), [44](#), [45](#), [52](#), [64](#), [65](#), [71](#),
[127](#), [180](#), [182](#), [192](#), [192](#), [195](#), [196](#),
[245](#)
- `prep_counts_daily_effort()`, [8](#), [9](#), [192](#)
- `prep_interview_catch`, [7](#), [11](#), [16](#), [19](#), [21](#), [26](#),
[30](#), [44](#), [45](#), [52](#), [64](#), [65](#), [71](#), [127](#), [180](#),
[182](#), [192](#), [193](#), [195](#), [196](#), [245](#)
- `prep_interview_catch()`, [13](#), [147](#)
- `prep_interviews_trips`, [7](#), [11](#), [16](#), [19](#), [21](#),
[26](#), [30](#), [44](#), [45](#), [52](#), [64](#), [65](#), [71](#), [127](#),
[180](#), [182](#), [192](#), [193](#), [194](#), [196](#), [245](#)
- `prep_interviews_trips()`, [13](#), [152](#)
- `preprocess_camera_timestamps`, [145](#), [189](#)
- `preprocess_camera_timestamps()`, [146](#)
- `print.creel_completeness_report`, [197](#)

- `print.creel_data_validation`, 197
- `print.creel_design`, 198
- `print.creel_design_comparison`, 198
- `print.creel_design_report`, 199
- `print.creel_estimates`, 199
- `print.creel_estimates_diagnostic`, 200
- `print.creel_estimates_mor`, 200
- `print.creel_hybrid_svydesign`, 201
- `print.creel_schedule`, 201
- `print.creel_schedule()`, 37
- `print.creel_schema`, 202
- `print.creel_season_summary`, 203
- `print.creel_summary`, 203
- `print.creel_tost_validation`, 204
- `print.creel_validation`, 204
- `print.creel_validation_report`, 205
- `print.creel_variance_comparison`, 205

- `read_schedule`, 31, 166, 168, 170, 172, 183, 206, 244, 254
- `read_schedule()`, 243
- `reallocate_strata`, 33, 41, 55, 57, 58, 61, 67, 185, 188, 207, 216
- `reallocate_strata()`, 185
- `rgamma()`, 212
- `rlang::check_installed()`, 252, 254
- `rnbinom()`, 212
- `rpois()`, 212

- `season_summary`, 23, 38, 43, 159, 208, 217, 219, 221–225, 227, 228, 230, 233, 234, 236–239, 241, 243, 246, 249, 251, 252
- `simulate_creel_catch`, 68, 209, 214
- `simulate_creel_data`, 68, 210, 211
- `simulate_creel_data()`, 68
- `simulate_strata_collapse`, 33, 41, 55, 57, 58, 61, 67, 185, 188, 207, 215
- `standardize_species`, 23, 38, 43, 159, 208, 216, 219, 221–225, 227, 228, 230, 233, 234, 236–239, 241, 243, 246, 249, 251, 252
- `summarize_boat_composition`, 23, 38, 43, 159, 208, 217, 218, 221–225, 227, 228, 230, 233, 234, 236–239, 241, 243, 246, 249, 251, 252
- `summarize_by_angler_type`, 23, 38, 43, 159, 208, 217, 219, 220, 222–225, 227, 228, 230, 233, 234, 236–239, 241, 243, 246, 249, 251, 252
- `summarize_by_county`, 23, 38, 43, 159, 208, 217, 219–221, 221, 223–225, 227, 228, 230, 233, 234, 236–239, 241, 243, 246, 249, 251, 252
- `summarize_by_day_type`, 23, 38, 43, 159, 208, 217, 219, 221, 222, 222, 224, 225, 227, 228, 230, 233, 234, 236–239, 241, 243, 246, 249, 251, 252
- `summarize_by_method`, 23, 38, 43, 159, 208, 217, 219, 221–223, 223, 225, 227, 228, 230, 233, 234, 236–239, 241, 243, 246, 249, 251, 252
- `summarize_by_species_sought`, 23, 38, 43, 159, 208, 217, 219, 221–224, 225, 227, 228, 230, 233, 234, 236–239, 242, 243, 246, 249, 251, 252
- `summarize_by_trip_length`, 23, 38, 43, 159, 208, 217, 219, 221–225, 226, 228, 230, 233, 234, 236–239, 242, 243, 246, 249, 251, 252
- `summarize_by_zip`, 23, 38, 43, 159, 208, 217, 219–225, 227, 227, 230, 233, 234, 236–239, 242, 243, 246, 249, 251, 252
- `summarize_cws_rates`, 23, 38, 43, 159, 208, 217, 219, 221–225, 227, 228, 228, 233, 234, 236–239, 242, 243, 246, 249, 251, 252
- `summarize_cws_rates()`, 233, 234
- `summarize_hws_rates`, 23, 38, 43, 159, 208, 217, 219, 221–225, 227, 228, 230, 231, 234, 236–239, 242, 243, 246, 249, 251, 253
- `summarize_hws_rates()`, 230, 234
- `summarize_length_freq`, 23, 38, 43, 159, 208, 217, 219, 221–224, 226–228, 230, 233, 233, 236–239, 242, 243, 246, 249, 251, 253
- `summarize_length_freq()`, 130
- `summarize_refusals`, 23, 38, 43, 159, 208, 217, 219, 221–224, 226–228, 230, 233, 234, 235, 237–239, 242, 243, 246, 249, 251, 253
- `summarize_successful_parties`, 23, 38, 43, 159, 208, 217, 219, 221–224,

[226–228](#), [230](#), [233](#), [234](#), [236](#), [236](#),
[238](#), [239](#), [242](#), [243](#), [246](#), [249](#), [251](#),
[253](#)
[summarize_trips](#), [23](#), [38](#), [43](#), [159](#), [208](#), [217](#),
[219](#), [221–224](#), [226–228](#), [230](#), [233](#),
[234](#), [236](#), [237](#), [238](#), [239](#), [242](#), [243](#),
[246](#), [249](#), [251](#), [253](#)
[summary.creel_estimates](#), [23](#), [38](#), [43](#), [159](#),
[208](#), [217](#), [219](#), [221–224](#), [226–228](#),
[230](#), [233](#), [234](#), [236–238](#), [239](#), [242](#),
[243](#), [246](#), [249](#), [251](#), [253](#)
[summary.creel_estimates\(\)](#), [35](#), [203](#), [252](#)
[svydesign](#), [22](#)

[theme_creel](#), [35–37](#), [59](#), [186](#), [240](#)
[theme_creel\(\)](#), [34](#), [35](#), [59](#)
[tidy.creel_estimates](#), [23](#), [38](#), [43](#), [159](#), [208](#),
[217](#), [219](#), [221–224](#), [226–228](#), [230](#),
[233](#), [234](#), [236–239](#), [241](#), [243](#), [246](#),
[249](#), [251](#), [253](#)
[tidyselect](#), [6](#), [7](#), [18](#), [19](#)

[utils::write.csv\(\)](#), [254](#)

[validate_creel_data](#), [23](#), [38](#), [43](#), [159](#), [208](#),
[217](#), [219](#), [221–224](#), [226–228](#), [230](#),
[233](#), [234](#), [236–239](#), [242](#), [242](#), [246](#),
[249](#), [251](#), [253](#)
[validate_creel_schedule](#), [31](#), [166](#), [168](#),
[170](#), [172](#), [183](#), [206](#), [243](#), [254](#)
[validate_creel_schedule\(\)](#), [243](#)
[validate_creel_schema](#), [7](#), [11](#), [16](#), [19](#), [21](#),
[26](#), [30](#), [44](#), [45](#), [52](#), [64](#), [65](#), [71](#), [127](#),
[180](#), [182](#), [192](#), [193](#), [195](#), [196](#), [244](#)
[validate_creel_schema\(\)](#), [61](#)
[validate_design](#), [23](#), [38](#), [43](#), [159](#), [208](#), [217](#),
[219](#), [221–224](#), [226–228](#), [230](#), [233](#),
[234](#), [236–239](#), [242](#), [243](#), [245](#), [249](#),
[251](#), [253](#)
[validate_incomplete_trips](#), [23](#), [38](#), [43](#),
[159](#), [208](#), [217](#), [219](#), [221–224](#),
[226–228](#), [230](#), [233](#), [234](#), [236–239](#),
[242](#), [243](#), [246](#), [247](#), [251](#), [253](#)
[validation_report](#), [23](#), [38](#), [43](#), [159](#), [208](#),
[217](#), [219](#), [221–224](#), [226–228](#), [230](#),
[233](#), [234](#), [236–239](#), [242](#), [243](#), [246](#),
[249](#), [250](#), [253](#)

[withr::with_seed\(\)](#), [167](#), [169](#), [172](#)