Package 'PsyMetricTools'

August 6, 2026

Version 1.2.2

Title Psychometric and Statistical Analysis Tools

Description Provides tools for psychometric and statistical analysis in social sciences, including functions for data preprocessing, factor analysis, reliability testing, descriptive statistics, visualization of Likert-scale items, measurement invariance analysis, and handling of multi-class imbalance. Facilitates advanced data analysis tasks commonly encountered in psychological and educational research. Includes an updated easy_invariance() function for ordinal data following Wu and Estabrook (2016) <doi:10.1037/met0000051>.

License GPL (>= 3)

URL https://github.com/jventural/PsyMetricTools

BugReports https://github.com/jventural/PsyMetricTools/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports lavaan, semTools, psych, dplyr, tidyr, ggplot2, purrr, tibble, stringr, gridExtra, gtable, grid, pbapply, tidyselect, rlang, grDevices, scales, stats, utils

Suggests testthat (>= 3.0.0), EGAnet, bootnet, future, future.apply, progressr, progress, careless, qgraph, semPlot, ggridges, ggpubr, forcats, RColorBrewer, openxlsx, lavaan.mi, blavaan, HDInterval, ggplotify, patchwork, wesanderson, circlize

Config/testthat/edition 3

RoxygenNote 7.3.2

NeedsCompilation no

Author Jose Ventura-Leon [aut, cre] (ORCID: <https://orcid.org/0000-0003-2996-4244>)

Maintainer Jose Ventura-Leon <jventuraleon@gmail.com>

Repository CRAN

Date/Publication 2026-08-06 13:10:21 UTC

Contents

`admis_simple` *Simple Admissibility Check for CFA*

Description

Checks whether a fitted CFA model has admissible solutions by detecting negative residual variances (Heywood cases) and standardized factor loadings with absolute value greater than 1.

Usage

```
admis_simple(fit, latents)
```

Arguments

fit	A fitted lavaan object or NULL.
latents	Character vector of latent variable names to check.

Value

A character string: "OK" if admissible, "NONCONV/FAIL" if the model is NULL, or a description of the problems found.

Examples

```
set.seed(123)
mydata <- data.frame(
  x1 = sample(1:5, 200, replace = TRUE),
  x2 = sample(1:5, 200, replace = TRUE),
  x3 = sample(1:5, 200, replace = TRUE)
)
model <- 'F1 =~ x1 + x2 + x3'
fit <- safe_cfa(model, data = mydata, items = c("x1", "x2", "x3"))
latents <- lavaan::lavNames(fit, type = "lv")
admis_simple(fit, latents)
```

---

agrupar_por_factor *Group Items by Factor Loadings*

Description

Groups items by their factor loadings above a threshold.

Usage

```
agrupar_por_factor(df, item_col = "Items", threshold = 0)
```

Arguments

df	Data frame with items and factor loadings.
item_col	Name of the column containing item identifiers.
threshold	Threshold for factor loading (default 0).

Value

A named list with items grouped by factor.

Examples

```
# Create a data frame with factor loadings
loadings_df <- data.frame(
  Items = c("Item1", "Item2", "Item3", "Item4", "Item5", "Item6"),
  f1 = c(0.75, 0.68, 0.72, 0.10, 0.05, 0.08),
  f2 = c(0.08, 0.12, 0.05, 0.70, 0.65, 0.78)
)

# Group items by factor (using threshold of 0.30)
groups <- agrupar_por_factor(
  df = loadings_df,
  item_col = "Items",
  threshold = 0.30
)
print(groups)
# $f1
# [1] "Item1" "Item2" "Item3"
# $f2
# [1] "Item4" "Item5" "Item6"
```

---

boot_cfa *Bootstrap Confirmatory Factor Analysis*

Description

Performs bootstrap CFA with reliability estimation.

Usage

```
boot_cfa(
  new_df,
  model_string,
  item_prefix,
  seed = NULL,
  n_replications = 1000,
  ordered = TRUE,
  estimator = "WLSMV"
)
```

Arguments

new_df	Data frame with the data.
model_string	Lavaan model specification string.
item_prefix	Prefix for item column names.
seed	Optional random seed for reproducibility (default NULL, no seed is set).
n_replications	Number of bootstrap replications (default 1000).
ordered	Logical indicating if variables are ordinal (default TRUE).
estimator	Estimator to use (default "WLSMV").

Value

Data frame with bootstrap results including fit measures and reliability.

Examples

```
# Create sample data with 9 Likert-type items (3 factors, 3 items each)
set.seed(123)
n <- 300
data_cfa <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE),
  Item7 = sample(1:5, n, replace = TRUE),
  Item8 = sample(1:5, n, replace = TRUE),
  Item9 = sample(1:5, n, replace = TRUE)
)

# Define the CFA model
model <- "
  F1 =~ Item1 + Item2 + Item3
  F2 =~ Item4 + Item5 + Item6
  F3 =~ Item7 + Item8 + Item9
"

# Perform bootstrap CFA with 100 replications (use more in practice)
result <- boot_cfa(
  new_df = data_cfa,
  model_string = model,
  item_prefix = "Item",
  seed = 2023,
  n_replications = 25,
  ordered = TRUE,
  estimator = "WLSMV"
)

# Access fit measures
result$fit_measures1

# Check convergence rates
mean(result$converged1)

# Access reliability estimates (omega)
result[, c("Rel1", "Rel2", "Rel3")]
```

`boot_cfa_density` *Density Plot para Bootstrap CFA*

Description

Visualiza la dispersion de resultados bootstrap mediante graficos de densidad con intervalos de confianza y valores de referencia.

Usage

```
boot_cfa_density(
  df,
  save = FALSE,
  path = NULL,
  dpi = 600,
  exclude_indices = NULL,
  show_ci = TRUE,
  ci_level = 0.95,
  show_reference = TRUE,
  fill_color = "#3498db",
  ci_color = "#e74c3c",
  ...
)
```

Arguments

df	Data frame con resultados de boot_cfa().
save	Logical. Guardar el grafico (default FALSE).
path	Ruta para guardar (default NULL; obligatoria si save = TRUE, e.g. file.path(tempdir(), "Plot_boot_density.jpg")).
dpi	Resolucion (default 600).
exclude_indices	Vector de indices a excluir (e.g., c("RMSEA")).
show_ci	Mostrar intervalo de confianza sombreado (default TRUE).
ci_level	Nivel de confianza (default 0.95).
show_reference	Mostrar lineas de referencia (default TRUE).
fill_color	Color de relleno de densidad (default "#3498db").
ci_color	Color del area de IC (default "#e74c3c").
...	Argumentos adicionales para ggsave.

Value

Un objeto ggplot. La tabla de estadisticos bootstrap se adjunta como atributo "stats" y puede extraerse con attr(x, "stats").

Examples

```
# First run boot_cfa to get bootstrap results
set.seed(123)
n <- 300
```

```
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

model <- "
  F1 =~ Item1 + Item2 + Item3
  F2 =~ Item4 + Item5 + Item6
"

boot_results <- boot_cfa(
  new_df = data,
  model_string = model,
  item_prefix = "Item",
  n_replications = 25
)

# Create density plot with confidence intervals
boot_cfa_density(boot_results,
                 save = TRUE,
                 path = file.path(tempdir(), "bootstrap_density.jpg"),
                 show_ci = TRUE,
                 ci_level = 0.95,
                 show_reference = TRUE)

# Custom colors
boot_cfa_density(boot_results,
                 save = FALSE,
                 fill_color = "#2ecc71",
                 ci_color = "#e74c3c")
```

`boot_cfa_plot` *Bootstrap CFA Plot*

Description

Creates boxplot visualizations for bootstrap CFA results.

Usage

```
boot_cfa_plot(
  df,
  save = FALSE,
  path = NULL,
  dpi = 600,
```

```
  omega_ymin_annot = NULL,
  omega_ymax_annot = NULL,
  comp_ymin_annot = NULL,
  comp_ymax_annot = NULL,
  abs_ymin_annot = NULL,
  abs_ymax_annot = NULL,
  palette = "grey",
  exclude_indices = NULL,
  show_tables = TRUE,
  ...
)
```

Arguments

df	Data frame with results from boot_cfa().
save	Logical. Save the plot (default FALSE).
path	Path to save the plot (default NULL; required when save = TRUE, e.g. file.path(tempdir(), "Plot_boot_cfa.jpg")).
dpi	Resolution in DPI (default 600).
omega_ymin_annot	Y-axis minimum for omega annotation.
omega_ymax_annot	Y-axis maximum for omega annotation.
comp_ymin_annot	Y-axis minimum for comparative indices annotation.
comp_ymax_annot	Y-axis maximum for comparative indices annotation.
abs_ymin_annot	Y-axis minimum for absolute indices annotation.
abs_ymax_annot	Y-axis maximum for absolute indices annotation.
palette	Color palette (default "grey").
exclude_indices	Indices to exclude from plots.
show_tables	Show summary tables in plots (default TRUE).
...	Additional arguments passed to ggsave.

Value

A combined ggplot object (invisibly).

Examples

```
# First run boot_cfa to get bootstrap results
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
```

```
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

model <- "
  F1 =~ Item1 + Item2 + Item3
  F2 =~ Item4 + Item5 + Item6
"

boot_results <- boot_cfa(
  new_df = data,
  model_string = model,
  item_prefix = "Item",
  n_replications = 25
)

# Create boxplot visualization
boot_cfa_plot(boot_results,
              save = TRUE,
              path = file.path(tempdir(), "bootstrap_cfa_results.jpg"),
              palette = "grey",
              show_tables = TRUE)

# Exclude certain indices
boot_cfa_plot(boot_results,
              save = FALSE,
              exclude_indices = c("CRMR"),
              palette = "grey")
```

`boot_cfa_plot_enhanced` *Boot CFA Plot Enhanced (panel A enriquecido)*

Description

Versión mejorada de `boot_cfa_plot()` que mantiene el lenguaje visual base (boxplot azul + tabla descriptiva) pero agrega capas que responden más preguntas de un vistazo:

1. Banda verde (*pasa*) y roja (*falla*) detrás del boxplot.
2. Línea de cutoff (rojo punteado) con etiqueta lateral por columna.
3. Jitter de las réplicas (azul muy claro) detrás del boxplot.
4. Mediana con valor numérico anotado al lado del boxplot.
5. Porcentaje de réplicas que cumplen el criterio (*compliance %*) en verde bold sobre el panel.
6. Tabla descriptiva con cabecera coloreada y bordes APA.

La función original `boot_cfa_plot()` permanece intacta.

Usage

```
boot_cfa_plot_enhanced(
  df,
  save = FALSE,
  path = NULL,
  dpi = 600,
  palette = NULL,
  accent_color = NULL,
  semantic_colors = TRUE,
  pass_palette = c("#bae4b3", "#74c476", "#238b45"),
  fail_palette = c("#fcae91", "#fb6a4a", "#cb181d"),
  pass_accent = "#00441b",
  fail_accent = "#67000d",
  compliance_threshold = 90,
  lang = c("es", "en"),
  table_style = c("iqr_min_max", "mean_sd_ci", "cv_min_max"),
 cutoffs = list(omega = 0.7, CFI = 0.95, TLI = 0.95, RMSEA = 0.08, SRMR = 0.08, CRMR =
    0.08),
 higher_better = list(omega = TRUE, CFI = TRUE, TLI = TRUE, RMSEA = FALSE, SRMR = FALSE,
    CRMR = FALSE),
  show_jitter = TRUE,
  show_compliance = TRUE,
  show_median_label = TRUE,
  show_zone_bands = TRUE,
  show_cutoff_label = TRUE,
  show_tables = TRUE,
  exclude_indices = NULL,
  ...
)
```

Arguments

df	Data frame producido por boot_cfa().
save	Lógico. Guardar el gráfico (default FALSE).
path	Ruta del archivo a guardar (default NULL; obligatoria si save = TRUE, e.g. file.path(tempdir(), "Plot_boot_cfa_enhanced.jpg")).
dpi	Resolución (default 600).
palette	\[Deprecated\] Vector de 3 colores. Si se especifica, se usará ignorando semantic_colors. Se mantiene por compatibilidad hacia atrás.
accent_color	\[Deprecated\] Color de acento. Si se especifica con palette no nulo, se usará. Si semantic_colors=TRUE, se ignora.
semantic_colors	Lógico. Si TRUE (default), el color del boxplot se asigna según el veredicto de compliance: verde si %pasa >= 90, rojo si < 90. Esto hace el plot dialogar visualmente con las bandas pasa/falla del fondo. Si FALSE, usa la paleta única palette.

`pass_palette` Vector de 3 verdes de claro a oscuro para variables que cumplen el criterio (default verdes de RColorBrewer).

`fail_palette` Vector de 3 rojos de claro a oscuro para variables que no cumplen el criterio.

`pass_accent` Color de borde/mediana para boxplots verdes.

`fail_accent` Color de borde/mediana para boxplots rojos.

`compliance_threshold` Umbral de %pasa que separa "pass" de "fail" (default 90).

`lang` Idioma del gráfico: `"es"` (default) o `"en"`. Afecta etiquetas de eje Y, etiqueta de compliance (*X% pasa* / *X% pass*) y demás textos visibles. Los nombres de los índices (CFI, RMSEA, etc.) y "cutoff", "Mdn" se mantienen como notación universal.

`table_style` Estadísticos descriptivos a reportar en la tabla bajo cada sub-panel. Tres opciones:

- `"iqr_min_max"` (default) — IQR + min + max. Coherente con la mediana del boxplot (todos no-paramétricos / robustos).
- `"mean_sd_ci"` — mean + SD + IC 95% percentil bootstrap. Estándar APA, comparable con la literatura clásica.
- `"cv_min_max"` — coeficiente de variación (%) + min + max. Útil para comparar dispersión entre índices con escalas distintas, pero puede ser engañoso cuando la media se acerca a 0 (e.g., RMSEA bajo).

`cutoffs` Lista nombrada con los valores de corte por índice. Default: `list(omega = 0.70, CFI = 0.95, TLI = 0.95, RMSEA = 0.08, SRMR = 0.08, CRMR = 0.08)`.

`higher_better` Lista nombrada lógica indicando si el cutoff es un mínimo (`TRUE`, e.g., CFI >= .95) o un máximo (`FALSE`, e.g., RMSEA <= .08).

`show_jitter` Mostrar jitter de réplicas detrás del boxplot (default `TRUE`).

`show_compliance` Mostrar % compliance arriba (default `TRUE`).

`show_median_label` Mostrar etiqueta `Mdn = X.XXX` al lado del boxplot (default `TRUE`).

`show_zone_bands` Mostrar bandas verde/roja de fondo (default `TRUE`).

`show_cutoff_label` Mostrar etiqueta del valor del cutoff (default `TRUE`).

`show_tables` Mostrar tabla descriptiva debajo del boxplot (default `TRUE`).

`exclude_indices` Vector con índices a excluir (e.g., `c("CRMR")`).

`...` Argumentos adicionales para `ggsave`.

Value

Un objeto `patchwork` listo para imprimirse o componer con `patchwork::wrap_elements()`.

Examples

```
set.seed(123)
n <- 300
df <- as.data.frame(lapply(setNames(1:6, paste0("PHQ", 1:6)),
                           function(i) sample(1:5, n, replace = TRUE)))
m <- "F1 =~ PHQ1 + PHQ2 + PHQ3
      F2 =~ PHQ4 + PHQ5 + PHQ6"
results_boot <- boot_cfa(new_df = df, model_string = m,
                         item_prefix = "PHQ", n_replications = 30)
boot_cfa_plot_enhanced(results_boot)
```

`boot_cfa_raincloud` *Raincloud Plot para Bootstrap CFA*

Description

Crea visualizaciones raincloud elegantes para resultados de bootstrap CFA.

Usage

```
boot_cfa_raincloud(
  df,
  save = FALSE,
  path = NULL,
  dpi = 600,
  exclude_indices = NULL,
  color_scheme = "ocean",
  show_stats = TRUE,
  theme_style = "modern",
  ...
)
```

Arguments

`df`	Data frame con resultados de boot_cfa().
`save`	Logical. Guardar el grafico (default FALSE).
`path`	Ruta para guardar el grafico (default NULL; obligatoria si `save = TRUE`, e.g. `file.path(tempdir(), "Plot_boot_raincloud.jpg")`).
`dpi`	Resolucion en DPI (default 600).
`exclude_indices`	Vector de indices a excluir (e.g., c("RMSEA")).
`color_scheme`	Esquema de color: "ocean", "sunset", "forest", "lavender", "monochrome", "elegant".
`show_stats`	Mostrar estadisticos en el grafico (default TRUE).
`theme_style`	Estilo: "modern", "minimal", "dark" (default "modern").
`...`	Argumentos adicionales para ggsave.

Value

Un objeto ggplot.

Examples

```
# First run boot_cfa to get bootstrap results
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

model <- "
  F1 =~ Item1 + Item2 + Item3
  F2 =~ Item4 + Item5 + Item6
"

boot_results <- boot_cfa(
  new_df = data,
  model_string = model,
  item_prefix = "Item",
  n_replications = 25
)

# Create raincloud plot with ocean color scheme
boot_cfa_raincloud(boot_results,
                   save = TRUE,
                   path = file.path(tempdir(), "bootstrap_raincloud.jpg"),
                   color_scheme = "ocean",
                   show_stats = TRUE,
                   theme_style = "modern")

# Dark theme with sunset colors
boot_cfa_raincloud(boot_results,
                   save = FALSE,
                   color_scheme = "sunset",
                   theme_style = "dark")

# Available color schemes: "ocean", "sunset", "forest", "lavender", "monochrome", "elegant"
```

`boot_cfa_stability` *Bootstrap CFA Stability Analysis*

Description

Performs bootstrap CFA stability analysis across different sample sizes.

Usage

```
boot_cfa_stability(
  modelo,
  data,
  num_replicas,
  estimator,
  seed = NULL,
  n_cores = 2
)
```

Arguments

modelo	Lavaan model specification.
data	Data frame with the data.
num_replicas	Number of bootstrap replications.
estimator	Estimator to use (e.g., "WLSMV").
seed	Optional random seed for reproducibility (default NULL, no seed is set).
n_cores	Number of cores for parallel processing (default 2).

Value

Data frame with fit measures and reliability across sample sizes.

Examples

```
# Create sample data
set.seed(123)
n <- 300
g <- rnorm(n)
t1 <- 0.7 * g + rnorm(n, 0, 0.7)
t2 <- 0.7 * g + rnorm(n, 0, 0.7)
sim_item <- function(t) {
  as.numeric(cut(t + rnorm(length(t), 0, 0.8), c(-Inf, -1, 0, 1, Inf)))
}
data <- data.frame(
  Item1 = sim_item(t1), Item2 = sim_item(t1), Item3 = sim_item(t1),
  Item4 = sim_item(t2), Item5 = sim_item(t2), Item6 = sim_item(t2)
)

# Define CFA model
model <- "
  F1 =~ Item1 + Item2 + Item3
  F2 =~ Item4 + Item5 + Item6
"
```

```
# Run stability analysis (testing 90% to 30% of sample sizes)
stability_results <- boot_cfa_stability(
  modelo = model,
  data = data,
  num_replicas = 10,
  estimator = "WLSMV",
  seed = 2023,
  n_cores = 2
)

# View results - fit indices and reliability at different sample sizes
head(stability_results)

# Plot stability results
plot_cfa_stability(stability_results)
```

`boot_efa` *Bootstrap EFA (Exploratory Factor Analysis)*

Description

Performs bootstrap resampling for Exploratory Factor Analysis using lavaan, calculating fit measures, factor loadings, and McDonald's omega reliability for each replication.

Usage

```
boot_efa(
  data,
  n_factors,
  n_items,
  name_items,
  exclude_items = NULL,
  rotation = "oblimin",
  estimator = "WLSMV",
  ordered = TRUE,
  apply_threshold = FALSE,
  seed = NULL,
  n_replications = 1000
)
```

Arguments

`data`	A data frame containing the item responses.
`n_factors`	Number of factors to extract.
`n_items`	Total number of items in the scale.
`name_items`	Prefix of item names (e.g., "PBA" for PBA1, PBA2, etc.).

exclude_items	Character vector of item names to exclude (default: NULL).
rotation	Rotation method (default: "oblimin").
estimator	Estimator to use (default: "WLSMV").
ordered	Logical; treat indicators as ordered/categorical (default: TRUE). Set to FALSE for continuous indicators with ML estimator.
apply_threshold	Whether to apply threshold to loadings (default: FALSE).
seed	Optional random seed for reproducibility (default: NULL, no seed is set).
n_replications	Number of bootstrap replications (default: 1000).

Value

A list containing:

- Replicaciones: Data frame with all bootstrap results
- fit_summary: Summary statistics of fit measures
- loadings_summary: Summary statistics of factor loadings
- n_converged: Number of converged models
- n_valid: Number of valid models
- parameters: List of parameters used

Examples

```
set.seed(123)
n <- 200
data <- as.data.frame(lapply(setNames(1:6, paste0("PBA", 1:6)),
                             function(i) sample(1:5, n, replace = TRUE)))
results <- boot_efa(
  data = data,
  n_factors = 2,
  n_items = 6,
  name_items = "PBA",
  rotation = "oblimin",
  n_replications = 5
)
```

---

boot_efa_forest_plot *Forest Plot for Bootstrap EFA Results*

Description

Creates a forest plot showing factor loadings with 95 from bootstrap EFA results. Style inspired by Solomon Kurz.

Usage

```
boot_efa_forest_plot(
  boot_efa_results,
  save = FALSE,
  path = NULL,
  dpi = 600,
  title = "Factor Loadings with 95% CI (Bootstrap)",
  threshold_low = 0.4,
  threshold_mid = 0.7,
  ...
)
```

Arguments

boot_efa_results	Results from boot_efa function.
save	Logical, whether to save the plot (default: FALSE).
path	File path to save the plot (default: NULL; required when save = TRUE, e.g. file.path(tempdir(), "Forest_plot_efa.jpg")).
dpi	Resolution in dots per inch (default: 600).
title	Plot title.
threshold_low	Low loading threshold (default: 0.4).
threshold_mid	Mid loading threshold (default: 0.7).
...	Additional arguments passed to ggsave.

Value

Invisibly returns a list with the plot and summary data.

Examples

```
set.seed(123)
n <- 200
data <- as.data.frame(lapply(setNames(1:6, paste0("PBA", 1:6)),
                             function(i) sample(1:5, n, replace = TRUE)))
results_boot_efa <- boot_efa(
  data = data, n_factors = 2, n_items = 6, name_items = "PBA",
  rotation = "oblimin", n_replications = 5
)
boot_efa_forest_plot(results_boot_efa,
                     save = TRUE,
                     path = file.path(tempdir(), "Forest_PBA.jpg"),
                     title = "Factor Loadings PBA - Bootstrap EFA")
```

`boot_efa_plot` *Plot Bootstrap EFA Results*

Description

Creates a three-panel plot showing omega reliability, CFI/TLI, and RMSEA/SRMR/CRMR from bootstrap EFA results.

Usage

```
boot_efa_plot(
  boot_efa_results,
  save = FALSE,
  path = NULL,
  dpi = 600,
  omega_ymin_annot = NULL,
  omega_ymax_annot = NULL,
  comp_ymin_annot = NULL,
  comp_ymax_annot = NULL,
  abs_ymin_annot = NULL,
  abs_ymax_annot = NULL,
  palette = "grey",
  ...
)
```

Arguments

`boot_efa_results`	Results from boot_efa function.
`save`	Logical, whether to save the plot (default: FALSE).
`path`	File path to save the plot (default: NULL; required when `save = TRUE`, e.g. `file.path(tempdir(), "Plot_boot_efa.jpg")`).
`dpi`	Resolution in dots per inch (default: 600).
`omega_ymin_annot`	Y-axis minimum for omega table annotation.
`omega_ymax_annot`	Y-axis maximum for omega table annotation.
`comp_ymin_annot`	Y-axis minimum for CFI/TLI table annotation.
`comp_ymax_annot`	Y-axis maximum for CFI/TLI table annotation.
`abs_ymin_annot`	Y-axis minimum for RMSEA/SRMR/CRMR table annotation.
`abs_ymax_annot`	Y-axis maximum for RMSEA/SRMR/CRMR table annotation.
`palette`	Color palette ("grey" or wesanderson palette name).
`...`	Additional arguments passed to ggsave.

Value

Invisibly returns the plot.

Examples

```
set.seed(123)
n <- 200
data <- as.data.frame(lapply(setNames(1:6, paste0("PBA", 1:6)),
                             function(i) sample(1:5, n, replace = TRUE)))
results_boot_efa <- boot_efa(
  data = data, n_factors = 2, n_items = 6, name_items = "PBA",
  rotation = "oblimin", n_replications = 5
)
boot_efa_plot(results_boot_efa,
              save = TRUE,
              path = file.path(tempdir(), "mi_grafico_efa.jpg"),
              omega_ymin_annot = 0.70,
              omega_ymax_annot = 0.80)
```

---

calculate_omega_J *Omega Coefficient from a Fitted lavaan Model*

Description

Computes McDonald's omega using the R-squared values (communalities) extracted directly from a fitted lavaan object. Accepts either a single character vector of item names or a named list of vectors to compute omega for multiple factors at once.

Usage

```
calculate_omega_J(items, fit, digits = 3)
```

Arguments

items	A character vector of item names *or* a named list of character vectors (one element per factor).
fit	A fitted lavaan object (e.g., from cfa or an EFA specification produced by EFA_modern).
digits	Integer; number of decimal places for rounding (default 3).

Value

When items is a single vector, a numeric scalar (omega). When items is a named list, a named numeric vector with one omega per factor.

Examples

```
library(lavaan)
model <- 'F1 =~ x1 + x2 + x3
          F2 =~ x4 + x5 + x6'
fit <- cfa(model, data = HolzingerSwineford1939)

# Single factor
calculate_omega_J(c("x1", "x2", "x3"), fit)

# Multiple factors at once
calculate_omega_J(
  list(F1 = c("x1", "x2", "x3"),
       F2 = c("x4", "x5", "x6")),
  fit
)
```

---

calculate_per_fit *Calculate Percentage of Fit Indices Meeting Thresholds*

Description

Calculates the percentage of bootstrap replications meeting specified fit thresholds.

Usage

```
calculate_per_fit(df_repli, thresholds_str)
```

Arguments

df_repli	Data frame with bootstrap replications containing fit_measures1.
thresholds_str	String specifying thresholds (e.g., "CFI > 0.90, RMSEA < 0.08").

Value

A data frame with percentages for each fit measure.

Examples

```
# First run boot_cfa to get bootstrap results
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
```

```
)

model <- "
  F1 =~ Item1 + Item2 + Item3
  F2 =~ Item4 + Item5 + Item6
"

boot_results <- boot_cfa(
  new_df = data,
  model_string = model,
  item_prefix = "Item",
  n_replications = 25
)

# Define threshold string
thresholds <- "CFI > 0.95, TLI > 0.95, RMSEA < 0.06, SRMR < 0.08"

# Calculate percentage of bootstrap replications meeting thresholds
fit_percentages <- calculate_per_fit(boot_results, thresholds)
print(fit_percentages)

# Different thresholds (more lenient)
thresholds2 <- "CFI > 0.90, TLI > 0.90, RMSEA < 0.08"
calculate_per_fit(boot_results, thresholds2)
```

`calculate_per_fit_efa` *Calculate Percentage of Bootstrap EFA Models Meeting Fit Thresholds*

Description

Calculates the percentage of bootstrap EFA replications that meet specified fit measure thresholds.

Usage

```
calculate_per_fit_efa(boot_efa_results, thresholds_str)
```

Arguments

`boot_efa_results`	Results from boot_efa function.
`thresholds_str`	A string specifying thresholds in format "MEASURE DIRECTION VALUE" separated by commas. Example: "CFI > 0.95, TLI > 0.95, RMSEA < 0.06, SRMR < 0.08"

Value

A data frame with percentages for each threshold.

Examples

```
set.seed(123)
n <- 200
data <- as.data.frame(lapply(setNames(1:6, paste0("PBA", 1:6)),
                             function(i) sample(1:5, n, replace = TRUE)))
results_boot_efa <- boot_efa(
  data = data, n_factors = 2, n_items = 6, name_items = "PBA",
  rotation = "oblimin", n_replications = 5
)
thresholds <- "CFI > 0.95, TLI > 0.95, RMSEA < 0.06, SRMR < 0.08"
calculate_per_fit_efa(results_boot_efa, thresholds)
```

---

calcula_omega_mcdonald

Calculate McDonald's Omega Coefficient

Description

Calculates McDonald's omega reliability coefficient from factor loadings.

Usage

```
calcula_omega_mcdonald(
  loadings_df,
  groups = NULL,
  item_col = NULL,
  method = c("comunalidad", "sum_loadings")
)
```

Arguments

loadings_df	Data frame with items and factor loadings.
groups	NULL for all items, vector of items, or list of item vectors for groups.
item_col	Name of the column containing item names (default: first column).
method	Either "comunalidad" (h2 = sum of loadings squared) or "sum_loadings".

Value

A list with omega values and item statistics.

Examples

```
# Create a sample loadings data frame (from EFA or CFA)
loadings_df <- data.frame(
  Items = c("Item1", "Item2", "Item3", "Item4", "Item5", "Item6"),
  f1 = c(0.75, 0.68, 0.72, 0.10, 0.05, 0.08),
  f2 = c(0.08, 0.12, 0.05, 0.70, 0.65, 0.78)
)

# Calculate omega for all items
result <- calcula_omega_mcdonald(
  loadings_df = loadings_df,
  groups = NULL,
  method = "comunalidad"
)
result$omegas

# Calculate omega for specific groups of items
result2 <- calcula_omega_mcdonald(
  loadings_df = loadings_df,
  groups = list(
    Factor1 = c("Item1", "Item2", "Item3"),
    Factor2 = c("Item4", "Item5", "Item6")
  ),
  method = "comunalidad"
)
result2$omegas

# View item-level statistics
result2$item_stats$Factor1
```

---

combine_likert_sem *Combine Likert Plot with SEM Path Diagram*

Description

Creates a combined visualization with a Likert plot and SEM path diagram.

Usage

```
combine_likert_sem(
  plot_likert,
  fit_sem,
  sem_args = list(),
  nodeLabels = NULL,
  ncol = 2,
  widths = rep(1, ncol),
  tag_levels = "A",
  tag_suffix = "",
```

```
  tag_pos = c(0.05, 0.95),
  tag_offset_y = 0.02,
  tag_size = 16,
  tag_face = "bold"
)
```

Arguments

plot_likert	A ggplot object with Likert plot.
fit_sem	A lavaan model object for semPaths.
sem_args	List of additional arguments for semPaths.
nodeLabels	Custom node labels for the SEM diagram.
ncol	Number of columns in layout (default: 2).
widths	Relative widths of columns.
tag_levels	Tag levels for plot annotation (default: "A").
tag_suffix	Suffix for tags.
tag_pos	Position of tags as c(x, y).
tag_offset_y	Vertical offset for tags.
tag_size	Font size for tags.
tag_face	Font face for tags.

Value

A combined patchwork plot.

Examples

```
library(lavaan)
library(ggplot2)

# Create sample data
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

# Create Likert plot
likert_plot <- Plot_Likert(
  Data = data,
  name_items = "Item",
  ranges = 1:6
```

```
)

# Fit CFA model
model <- "
  F1 =~ Item1 + Item2 + Item3
  F2 =~ Item4 + Item5 + Item6
"
fit <- cfa(model, data = data, ordered = TRUE, estimator = "WLSMV")

# Combine Likert plot with SEM path diagram
combined_plot <- combine_likert_sem(
  plot_likert = likert_plot,
  fit_sem = fit,
  ncol = 2,
  widths = c(1, 1),
  tag_levels = "A",
  tag_size = 14
)

# Display combined plot
print(combined_plot)

# Save combined plot
ggsave(file.path(tempdir(), "combined_likert_sem.jpg"), combined_plot,
       width = 14, height = 8, dpi = 300)
```

`comparation_metric` *Compare Fit Metrics Across Models*

Description

Compares fit metrics (AIC, BIC, CFI, TLI, RMSEA, etc.) across multiple models.

Usage

```
comparation_metric(...)
```

Arguments

`...`	Named lavaan fit objects to compare.

Value

A data frame with fit metrics for each model.

cor_afe *Extract Factor Correlations from EFA*

Description

Extracts the inter-factor correlation matrix (Phi) from EFA results.

Usage

```
cor_afe(x)
```

Arguments

x	An EFA result object containing factor correlations.

Value

A matrix of factor correlations rounded to 2 decimals.

Examples

```
library(psych)

# Create sample data
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

# Run EFA with psych package
efa_result <- fa(data, nfactors = 2, rotate = "oblimin", fm = "pa")

# Extract factor correlations
factor_cors <- cor_afe(efa_result)
print(factor_cors)
# Output: 2x2 matrix of factor correlations
```

count_excluded_items *Count Excluded Items from Bootstrap Analysis*

Description

Counts how many times each item was excluded across bootstrap iterations.

Usage

```
count_excluded_items(res)
```

Arguments

res	A bootstrap result object containing processed_results.

Value

A tibble with item names and exclusion counts, sorted by frequency.

crearSintaxisCFA *Create CFA Syntax from Data Frame*

Description

Generates lavaan CFA model syntax from a data frame with items and factors.

Usage

```
crearSintaxisCFA(df)
```

Arguments

df	A data frame with Items and Factores columns, or factor loading columns (f1, f2, etc.).

Value

A character string with lavaan model syntax.

Examples

```
# Example 1: Create CFA syntax from EFA results with factor loadings
# Assume you have EFA results with items and their factor loadings
efa_loadings <- data.frame(
  Items = c("Item1", "Item2", "Item3", "Item4", "Item5", "Item6"),
  f1 = c(0.75, 0.68, 0.72, 0, 0, 0),
  f2 = c(0, 0, 0, 0.70, 0.65, 0.78)
)

# Generate CFA syntax
model <- crearSintaxisCFA(efa_loadings)
cat(model)
# Output:
# f1 =~ Item1 + Item2 + Item3
# f2 =~ Item4 + Item5 + Item6

# Example 2: Create CFA syntax from a data frame with explicit factor assignments
factor_assignments <- data.frame(
  Items = c("Q1", "Q2", "Q3", "Q4", "Q5", "Q6"),
  Factores = c("Anxiety", "Anxiety", "Anxiety", "Depression", "Depression", "Depression")
)

model2 <- crearSintaxisCFA(factor_assignments)
cat(model2)
# Output:
# F1 =~ Q1 + Q2 + Q3
# F2 =~ Q4 + Q5 + Q6
```

---

crear_modelo_lavaan *Create Lavaan Model Syntax*

Description

Creates lavaan model syntax from item prefix and factor specifications.

Usage

```
crear_modelo_lavaan(nombre, ...)
```

Arguments

nombre	Prefix for item names.
...	Named numeric vectors specifying item indices for each factor.

Value

A character string with lavaan model syntax.

Examples

```
# Create a 3-factor CFA model with item prefix "Item"
model <- crear_modelo_lavaan(
  nombre = "Item",
  F1 = 1:4,      # Items 1-4 load on Factor 1
  F2 = 5:8,      # Items 5-8 load on Factor 2
  F3 = 9:12      # Items 9-12 load on Factor 3
)
cat(model)
# Output:
# F1 =~ Item1 + Item2 + Item3 + Item4
# F2 =~ Item5 + Item6 + Item7 + Item8
# F3 =~ Item9 + Item10 + Item11 + Item12

# Create a 2-factor model with non-consecutive items
model2 <- crear_modelo_lavaan(
  nombre = "Q",
  Anxiety = c(1, 3, 5, 7),
  Depression = c(2, 4, 6, 8)
)
cat(model2)
```

---

createInitialModel *Create Initial Model Configuration*

Description

Creates an initial model configuration list for factor analysis.

Usage

```
createInitialModel(n_factors, n_items, name_items, exclude_items = NULL)
```

Arguments

n_factors	Number of factors.
n_items	Number of items.
name_items	Vector of item names or prefix.
exclude_items	Optional vector of items to exclude.

Value

A list with model configuration parameters.

create_groups *Create Item Groups for Factor Analysis*

Description

Creates a list of item index groups for factor definitions.

Usage

```
create_groups(names, values)
```

Arguments

names	Character vector of group/factor names.
values	Numeric vector with number of items per group.

Value

A named list with item indices for each group.

Examples

```
# Create groups for a 3-factor model with different items per factor
groups <- create_groups(
  names = c("Anxiety", "Depression", "Stress"),
  values = c(5, 4, 6)  # 5 items for Anxiety, 4 for Depression, 6 for Stress
)

# View the groups
groups
# $Anxiety
# [1] 1 2 3 4 5
# $Depression
# [1] 6 7 8 9
# $Stress
# [1] 10 11 12 13 14 15

# Use with crear_modelo_lavaan
model <- crear_modelo_lavaan(
  nombre = "Item",
  Anxiety = groups$Anxiety,
  Depression = groups$Depression,
  Stress = groups$Stress
)
cat(model)

# Example for a 2-factor scale
groups2 <- create_groups(
  names = c("Factor1", "Factor2"),
```

```
  values = c(8, 7)
)
```

`easy_invariance` *Analisis de Invarianza Factorial Simplificado*

Description

Realiza analisis de invarianza factorial para datos ordinales utilizando el enfoque de Wu y Estabrook (2016) con estimador WLSMV.

Usage

```
easy_invariance(
  model,
  data,
  estimator,
  ordered,
  ID.cat,
  group,
  levels_of_invariance,
  group.partial = NULL,
  fit_indices = c("CFI", "TLI", "RMSEA", "SRMR")
)
```

Arguments

`model`	Modelo de lavaan en formato texto.
`data`	Data frame con los datos.
`estimator`	Estimador a utilizar (por defecto "WLSMV").
`ordered`	Logical indicando si las variables son ordinales.
`ID.cat`	Metodo de identificacion para variables categoricas (e.g., "Wu.Estabrook.2016").
`group`	Nombre de la variable de agrupacion.
`levels_of_invariance`	Vector con los niveles de invarianza a evaluar: "configural", "threshold", "metric", "strict".
`group.partial`	Vector opcional de parametros a liberar parcialmente.
`fit_indices`	Vector con los indices de ajuste a incluir. Opciones disponibles: "CFI", "TLI", "RMSEA", "SRMR", "CRMR". Por defecto incluye c("CFI", "TLI", "RMSEA", "SRMR"). Para excluir RMSEA (util con pocos grados de libertad), usar c("CFI", "TLI", "SRMR", "CRMR").

Value

Lista con combined_data (tabla resumen) y los modelos ajustados.

Examples

```
set.seed(123)
n <- 400
theta <- rnorm(n)
sim_item <- function(theta) {
  as.numeric(cut(theta + rnorm(length(theta), 0, 0.8),
                 breaks = c(-Inf, -1, 0, 1, Inf)))
}
mydata <- data.frame(
  item1 = sim_item(theta), item2 = sim_item(theta),
  item3 = sim_item(theta), item4 = sim_item(theta),
  sex = rep(c("M", "F"), each = n / 2)
)

# Ejemplo con todos los indices por defecto
res <- easy_invariance(
  model = "F =~ item1 + item2 + item3 + item4",
  data = mydata,
  estimator = "WLSMV",
  ordered = TRUE,
  ID.cat = "Wu.Estabrook.2016",
  group = "sex",
  levels_of_invariance = c("configural", "threshold", "metric", "strict")
)

# Ejemplo excluyendo RMSEA
res <- easy_invariance(
  model = "F =~ item1 + item2 + item3 + item4",
  data = mydata,
  estimator = "WLSMV",
  ordered = TRUE,
  ID.cat = "Wu.Estabrook.2016",
  group = "sex",
  levels_of_invariance = c("configural", "threshold", "metric", "strict"),
  fit_indices = c("CFI", "TLI", "SRMR", "CRMR")
)
```

EFA_modern *Modern Exploratory Factor Analysis*

Description

Performs exploratory factor analysis using lavaan with rotation.

Usage

```
EFA_modern(
  n_factors,
```

```
  n_items,
  name_items,
  data,
  apply_threshold,
  estimator = "WLSMV",
  rotation = "oblimin",
  exclude_items = NULL
)
```

Arguments

n_factors	Number of factors to extract.
n_items	Number of items.
name_items	Prefix for item names.
data	Data frame containing the items.
apply_threshold	Logical, whether to apply threshold to factor loadings.
estimator	Estimator to use (default "WLSMV").
rotation	Rotation method (default "oblimin").
exclude_items	Items to exclude (default NULL).

Value

A list containing fit indices, specifications, interfactor correlations, and pattern matrix.

Examples

```
# Create sample data with 15 Likert-type items
set.seed(123)
n <- 300
data_efa <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE),
  Item7 = sample(1:5, n, replace = TRUE),
  Item8 = sample(1:5, n, replace = TRUE),
  Item9 = sample(1:5, n, replace = TRUE),
  Item10 = sample(1:5, n, replace = TRUE),
  Item11 = sample(1:5, n, replace = TRUE),
  Item12 = sample(1:5, n, replace = TRUE),
  Item13 = sample(1:5, n, replace = TRUE),
  Item14 = sample(1:5, n, replace = TRUE),
  Item15 = sample(1:5, n, replace = TRUE)
)

# Perform EFA with 3 factors using WLSMV estimator
```

```
result <- EFA_modern(
  n_factors = 3,
  n_items = 15,
  name_items = "Item",
  data = data_efa,
  apply_threshold = TRUE,
  estimator = "WLSMV",
  rotation = "oblimin"
)

# Access fit indices
result$Bondades_Original

# Access pattern matrix (factor loadings)
result$result_df

# Access interfactor correlations
result$InterFactor

# Example excluding specific items
result2 <- EFA_modern(
  n_factors = 3,
  n_items = 15,
  name_items = "Item",
  data = data_efa,
  apply_threshold = TRUE,
  estimator = "WLSMV",
  rotation = "oblimin",
  exclude_items = c("Item5", "Item10")
)
```

`EFA_plot` *Plot EFA Factor Loadings*

Description

Creates a visualization of factor loadings from EFA.

Usage

```
EFA_plot(specifications, item_prefix)
```

Arguments

`specifications`	Lavaan model object or a data.frame with columns: est.std, ci.lower, ci.upper, lhs, rhs, op.
`item_prefix`	Prefix for item names.

Value

A ggplot object showing factor loadings.

Examples

```
# Example 1: Using with EFA_modern() result
set.seed(123)
n <- 300
data_efa <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE),
  Item7 = sample(1:5, n, replace = TRUE),
  Item8 = sample(1:5, n, replace = TRUE),
  Item9 = sample(1:5, n, replace = TRUE)
)

# First run EFA
efa_result <- EFA_modern(
  n_factors = 3,
  n_items = 9,
  name_items = "Item",
  data = data_efa,
  apply_threshold = TRUE
)

# Plot factor loadings using the lavaan specification
plot <- EFA_plot(
  specifications = efa_result$Specifications[[3]],
  item_prefix = "Item"
)
print(plot)

# Example 2: Using with a lavaan object directly
library(lavaan)
model <- "
  efa('efa')*f1 +
  efa('efa')*f2 +
  efa('efa')*f3 =~ Item1 + Item2 + Item3 + Item4 + Item5 +
                   Item6 + Item7 + Item8 + Item9
"
fit <- sem(model, data = data_efa, rotation = "oblimin")
EFA_plot(fit, item_prefix = "Item")
```

`efa_with_bootstrap` *Exploratory Factor Analysis with Bootstrap*

Description

Performs EFA with optional bootstrap resampling for stability analysis.

Usage

```
efa_with_bootstrap(
  n_factors,
  n_items,
  name_items,
  data,
  apply_threshold,
  estimator = "WLSMV",
  rotation = "oblimin",
  exclude_items = NULL,
  bootstrap = FALSE,
  n_bootstrap = 1000,
  bootstrap_seed = NULL
)
```

Arguments

n_factors	Number of factors to extract.
n_items	Number of items in the analysis.
name_items	Vector of item names or prefix.
data	Data frame with item responses.
apply_threshold	Logical, whether to apply loading threshold.
estimator	Estimation method (default: "WLSMV").
rotation	Rotation method (default: "oblimin").
exclude_items	Optional vector of items to exclude.
bootstrap	Logical, whether to perform bootstrap (default: FALSE).
n_bootstrap	Number of bootstrap samples (default: 1000).
bootstrap_seed	Optional seed for reproducibility (default: NULL, no seed is set; supply a number to make the bootstrap reproducible).

Value

A list with EFA results and optional bootstrap statistics.

Examples

```
# Create sample data
set.seed(123)
n <- 200
g <- rnorm(n)
t1 <- 0.7 * g + rnorm(n, 0, 0.7)
```

```
t2 <- 0.7 * g + rnorm(n, 0, 0.7)
sim_item <- function(t) {
  as.numeric(cut(t + rnorm(length(t), 0, 0.8), c(-Inf, -1, 0, 1, Inf)))
}
data <- data.frame(
  Item1 = sim_item(t1), Item2 = sim_item(t1), Item3 = sim_item(t1),
  Item4 = sim_item(t2), Item5 = sim_item(t2), Item6 = sim_item(t2)
)

# Basic EFA without bootstrap
result_basic <- efa_with_bootstrap(
  n_factors = 2,
  n_items = 6,
  name_items = "Item",
  data = data,
  apply_threshold = TRUE,
  bootstrap = FALSE
)

# View fit indices
result_basic$Bondades_Original

# View factor loadings
result_basic$result_df

# EFA with bootstrap for stability analysis
result_boot <- efa_with_bootstrap(
  n_factors = 2,
  n_items = 6,
  name_items = "Item",
  data = data,
  apply_threshold = TRUE,
  bootstrap = TRUE,
  n_bootstrap = 5,  # Use 1000 in practice
  bootstrap_seed = 123
)

# View bootstrap summary of fit indices
result_boot$Bootstrap$fit_indices_summary$summary

# View bootstrap summary of factor loadings
result_boot$Bootstrap$loadings_summary

# View global loadings summary
result_boot$Bootstrap$global_loadings_summary
```

---

erp_by_item *Extreme-Response Proportion by Item*

Description

Computes the Extreme-Response Proportion (ERP) for each item in a Likert-type scale. ERP is the proportion of respondents who selected either the lowest or highest response option, which can signal floor/ceiling effects or poor item discrimination.

Usage

```
erp_by_item(data, items, extremes = c(1, 5), threshold = 0.3)
```

Arguments

`data`	A data frame containing the item responses.
`items`	Character vector of column names to analyse.
`extremes`	Numeric vector of length 2 indicating the lowest and highest anchor values (default `c(1, 5)`).
`threshold`	Numeric threshold above which an item is flagged (default 0.30).

Value

A data frame with one row per item containing: `Item`, `n`, `p_low`, `p_high`, `ERP`, `Median`, `IQR`, `Flag_ERP`, and formatted percentage columns. Rows are sorted by descending ERP.

Examples

```
set.seed(123)
df <- data.frame(
  IT1 = sample(1:5, 200, replace = TRUE),
  IT2 = sample(1:5, 200, replace = TRUE),
  IT3 = sample(1:5, 200, replace = TRUE)
)
erp <- erp_by_item(df, c("IT1", "IT2", "IT3"))
print(erp)
```

`export_summary_tables` *Export Summary Tables to Excel*

Description

Exports summary tables to an Excel file with formatted headers.

Usage

```
export_summary_tables(
  summary_Tabla_Total,
  summary_Tabla_EFA,
  summary_Tabla_CFA,
  file_name
)
```

Arguments

summary_Tabla_Total	Summary table for total sample.
summary_Tabla_EFA	Summary table for EFA sample.
summary_Tabla_CFA	Summary table for CFA sample.
file_name	Output file name (no default; supply a full path, e.g. `file.path(tempdir(), "Tablas_Resumenes.xlsx")`).

Value

Invisibly returns the merged data frame.

---

extractAllFitMeasures *Extract All Fit Measures from Bootstrap Results*

Description

Extracts and combines fit measures from all bootstrap replications.

Usage

```
extractAllFitMeasures(res)
```

Arguments

res	A bootstrap result object containing processed_results.

Value

A data frame with fit measures from all replications and iterations.

---

extractAllInterFactors *Extract All Inter-Factor Correlations from Bootstrap Results*

Description

Extracts and combines inter-factor correlations from all bootstrap replications.

Usage

```
extractAllInterFactors(res)
```

Arguments

res	A bootstrap result object containing processed_results.

Value

A data frame with factor correlations from all replications and iterations.

---

extractAllResults *Extract All Results from Bootstrap Analysis*

Description

Extracts and combines all results from bootstrap replications.

Usage

```
extractAllResults(res)
```

Arguments

res	A bootstrap result object containing processed_results.

Value

A data frame with results from all replications and iterations.

---

extracted_items2 *Extract Items by Maximum Loading*

Description

Assigns items to factors based on their highest loading value.

Usage

```
extracted_items2(data)
```

Arguments

data	Data frame with Items column and factor loading columns.

Value

A list with items assigned to each factor.

extract_bondad_boot_AFC

Extract Fit Measures from Bootstrap CFA Results

Description

Extracts and combines fit measures from bootstrap CFA analyses.

Usage

```
extract_bondad_boot_AFC(results)
```

Arguments

results	A list of CFA results containing FitMeasuresDf.

Value

A combined data frame of fit measures with sample and iteration identifiers.

Examples

```
# Bootstrap CFA results contain one FitMeasuresDf element per sample
results_list <- list(
  list(FitMeasuresDf = data.frame(CFI = 0.96, TLI = 0.95, RMSEA = 0.05)),
  list(FitMeasuresDf = data.frame(CFI = 0.97, TLI = 0.96, RMSEA = 0.04))
)

# Extract and combine fit measures across all bootstrap samples
all_fit_measures <- extract_bondad_boot_AFC(results_list)
head(all_fit_measures)
```

extract_bondad_boot_EFA

Extract Fit Measures from Bootstrap EFA Results

Description

Extracts and combines fit measures from bootstrap EFA analyses.

Usage

```
extract_bondad_boot_EFA(resultados_bootstrap)
```

Arguments

resultados_bootstrap	A bootstrap result object with Results list.

Value

A data frame of fit measures with sample identifiers.

Examples

```
# Bootstrap results contain one Bondades list per sample
resultados_bootstrap <- list(
  Results = list(
    list(CombinedResults = list(Bondades = list(
      data.frame(CFI = 0.96, TLI = 0.95, RMSEA = 0.05)
    ))),
    list(CombinedResults = list(Bondades = list(
      data.frame(CFI = 0.97, TLI = 0.96, RMSEA = 0.04)
    )))
  )
)

# Extract and combine fit measures across all bootstrap samples
fit_measures <- extract_bondad_boot_EFA(resultados_bootstrap)
head(fit_measures)
```

---

extract_ExcludedItems_boot

Extract Excluded Items from Bootstrap Results

Description

Extracts all excluded items across bootstrap samples.

Usage

```
extract_ExcludedItems_boot(resultados_bootstrap)
```

Arguments

resultados_bootstrap	A bootstrap result object with Results list.

Value

A data frame with excluded items and sample IDs.

`extract_fit` *Extract Fit Indices from Factor Analysis*

Description

Extracts fit indices (RMSEA, TLI, CFI, BIC) from psych factor analysis results.

Usage

```
extract_fit(factors_data)
```

Arguments

`factors_data`	A psych factor analysis result object.

Value

A data frame with fit index names and values.

Examples

```
library(psych)

# Create sample data
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

# Run factor analysis with psych package
fa_result <- fa(data, nfactors = 2, rotate = "oblimin", fm = "pa")

# Extract fit indices
fit_indices <- extract_fit(fa_result)
print(fit_indices)

# The result contains: RMSEA, lower_RMSEA, upper_RMSEA,
# confidence, TLI, CFI, null.chisq, objective, BIC
```

`extract_fit_measures` *Extract Fit Measures from Lavaan Specifications*

Description

Extracts fit measures (chi-square, SRMR, WRMR, CFI, TLI, RMSEA) from lavaan models.

Usage

```
extract_fit_measures(Specifications)
```

Arguments

`Specifications` A list of lavaan model objects.

Value

A data frame with fit measures for each factor solution.

Examples

```
# First, run EFA_modern to get specifications
set.seed(123)
n <- 300
g <- rnorm(n)
t1 <- 0.7 * g + rnorm(n, 0, 0.7)
t2 <- 0.7 * g + rnorm(n, 0, 0.7)
sim_item <- function(t) {
  as.numeric(cut(t + rnorm(length(t), 0, 0.8), c(-Inf, -1, 0, 1, Inf)))
}
data_efa <- data.frame(
  Item1 = sim_item(t1), Item2 = sim_item(t1), Item3 = sim_item(t1),
  Item4 = sim_item(t2), Item5 = sim_item(t2), Item6 = sim_item(t2)
)

# Run EFA
efa_result <- EFA_modern(
  n_factors = 2,
  n_items = 6,
  name_items = "Item",
  data = data_efa,
  apply_threshold = TRUE
)

# Extract fit measures from all factor solutions
fit_table <- extract_fit_measures(efa_result$Specifications)
print(fit_table)

# The result contains: Factores, chisq.scaled, df.scaled, srmr, wrmr,
# cfi.scaled, tli.scaled, rmsea.scaled
```

`extract_f_items` *Extract Factor Items for Lavaan Syntax*

Description

Creates lavaan model syntax by extracting items with non-zero loadings per factor.

Usage

```
extract_f_items(data, prefixes)
```

Arguments

`data`	Data frame with Items column and factor loading columns.
`prefixes`	Vector of factor prefixes (e.g., c("f1", "f2")).

Value

A character string with lavaan model syntax.

Examples

```
# Create a data frame with factor loadings (after EFA)
loadings_df <- data.frame(
  Items = c("Item1", "Item2", "Item3", "Item4", "Item5", "Item6"),
  f1 = c(0.75, 0.68, 0.72, 0, 0, 0),
  f2 = c(0, 0, 0, 0.70, 0.65, 0.78)
)

# Extract lavaan syntax for factors f1 and f2
model_syntax <- extract_f_items(
  data = loadings_df,
  prefixes = c("f1", "f2")
)
cat(model_syntax)
# Output:
# f1 =~ Item1 + Item2 + Item3
# f2 =~ Item4 + Item5 + Item6
```

`extract_ModIndex_AFC` *Extract Modification Indices from CFA Results*

Description

Extracts and combines modification indices from multiple CFA analyses.

Usage

```
extract_ModIndex_AFC(results)
```

Arguments

`results`	A list of CFA results containing ModificationsDf.

Value

A combined data frame of modification indices with sample identifiers.

Examples

```
library(lavaan)

# Create sample data
set.seed(123)
n <- 300
g <- rnorm(n)
t1 <- 0.7 * g + rnorm(n, 0, 0.7)
t2 <- 0.7 * g + rnorm(n, 0, 0.7)
sim_item <- function(t) {
  as.numeric(cut(t + rnorm(length(t), 0, 0.8), c(-Inf, -1, 0, 1, Inf)))
}
data <- data.frame(
  Item1 = sim_item(t1), Item2 = sim_item(t1), Item3 = sim_item(t1),
  Item4 = sim_item(t2), Item5 = sim_item(t2), Item6 = sim_item(t2)
)

# Fit the same CFA model on three bootstrap samples
model <- "F1 =~ Item1 + Item2 + Item3
          F2 =~ Item4 + Item5 + Item6"
fit1 <- cfa(model, data = data[sample(n, n, replace = TRUE), ])
fit2 <- cfa(model, data = data[sample(n, n, replace = TRUE), ])
fit3 <- cfa(model, data = data[sample(n, n, replace = TRUE), ])

# Each result contains a ModificationsDf element
results_list <- list(
  list(ModificationsDf = modificationIndices(fit1, sort = TRUE)),
  list(ModificationsDf = modificationIndices(fit2, sort = TRUE)),
  list(ModificationsDf = modificationIndices(fit3, sort = TRUE))
)
```

```
# Extract and combine modification indices
all_mod_indices <- extract_ModIndex_AFC(results_list)
head(all_mod_indices)
```

`extract_summary_modifications` *Summarize Modification Indices Across Replications*

Description

Creates summary statistics for modification indices across bootstrap samples.

Usage

```
extract_summary_modifications(combined_results, min_count = 10)
```

Arguments

`combined_results`	Combined data frame of modification indices.
`min_count`	Minimum count threshold for inclusion (default: 10).

Value

A summary data frame with mean, SD, min, and max MI values.

`extraer_metricas` *Extract Fit Metrics from Model*

Description

Extracts and formats common fit metrics (AIC, BIC, CFI, TLI, RMSEA) from a model.

Usage

```
extraer_metricas(modelo_fit)
```

Arguments

`modelo_fit`	A model fit object with Measure and Value columns.

Value

A data frame with formatted fit metrics.

factors_data_items *Combine Factor Data with Item Information*

Description

Combines factor loadings with item descriptors and applies loading thresholds.

Usage

```
factors_data_items(
  summary_data,
  data_items,
  num_factors,
  apply_threshold = TRUE
)
```

Arguments

summary_data	Factor summary data frame with Items column.
data_items	Item descriptor data frame with Items column.
num_factors	Number of factors in the analysis.
apply_threshold	Logical, whether to apply 0.30 threshold (default: TRUE).

Value

A tibble with items, loadings, and descriptors.

factor_summary *Create Factor Analysis Summary*

Description

Creates a summary table with loadings, communalities, and uniquenesses.

Usage

```
factor_summary(factors_data, num_items, num_factors)
```

Arguments

factors_data	A factor analysis result object with loadings.
num_items	Number of items in the analysis.
num_factors	Number of factors extracted.

Value

A data frame with items, factor loadings, h2, and u2.

Examples

```
library(psych)

# Create sample data
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

# Run EFA with psych package
efa_result <- fa(data, nfactors = 2, rotate = "oblimin", fm = "pa")

# Create summary table
summary_table <- factor_summary(
  factors_data = efa_result,
  num_items = 6,
  num_factors = 2
)
print(summary_table)
# Output:
#   Items   F1    F2   h2   u2
# 1 Item1 0.75  0.10 0.58 0.42
# 2 Item2 0.68  0.05 0.47 0.53
# ...
```

---

filtrar_aberrantes *Filter Aberrant Response Patterns*

Description

Identifies and filters cases with aberrant response patterns using Mahalanobis distance.

Usage

```
filtrar_aberrantes(data, items, plot = FALSE)
```

Arguments

data	Data frame containing the items.
items	Character vector of item names to analyze.
plot	Logical, whether to plot the Mahalanobis distances (default FALSE).

Value

A list with filtered data and a table of aberrant cases.

Examples

```
# Create sample data with some aberrant response patterns
set.seed(123)
n <- 200
data <- data.frame(
  ID = 1:n,
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE)
)

# Add some aberrant patterns (all same responses)
data[1, 2:6] <- rep(1, 5)
data[2, 2:6] <- rep(5, 5)

# Filter aberrant cases
result <- filtrar_aberrantes(
  data = data,
  items = c("Item1", "Item2", "Item3", "Item4", "Item5"),
  plot = FALSE
)

# Access filtered data (without aberrant cases)
clean_data <- result$data_filtrada
nrow(clean_data)

# View table of aberrant cases with Mahalanobis distances
result$tabla_aberrantes

# With visualization of Mahalanobis distances
result_plot <- filtrar_aberrantes(
  data = data,
  items = c("Item1", "Item2", "Item3", "Item4", "Item5"),
  plot = TRUE
)
```

`fit_index_Table` *Create Fit Index Table*

Description

Creates a comprehensive table combining fit indices and reliability measures.

Usage

```
fit_index_Table(resultados)
```

Arguments

`resultados`	A results object with bondades_ajuste and fiabilidad components.

Value

A data frame with model fit indices and omega reliability values.

Examples

```
# First run multi_cfa to get results
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

models <- list(
  "F1 =~ Item1 + Item2 + Item3 + Item4 + Item5 + Item6",
  "F1 =~ Item1 + Item2 + Item3\nF2 =~ Item4 + Item5 + Item6"
)

results <- multi_cfa(modelos = models, data = data,
                     estimator = "WLSMV", ordered = TRUE)

# Create fit index table combining fit measures and reliability
fit_table <- fit_index_Table(results)
print(fit_table)
# Output:
# Model   x2   df  SRMR   CFI   TLI  RMSEA  CRMR omega_F1 omega_F2
# 1      ...  ...  ...   ...   ...   ...    ...   0.78     NA
# 2      ...  ...  ...   ...   ...   ...    ...   0.75     0.80
```

generate_modelos *Generate EFA Models for Multiple Factor Solutions*

Description

Generates lavaan EFA model syntax for 1 to n_factors solutions.

Usage

```
generate_modelos(
  n_factors,
  specific_items = NULL,
  name_items = NULL,
  n_items = NULL,
  exclude_items = NULL
)
```

Arguments

n_factors	Maximum number of factors to test.
specific_items	Optional vector of specific item names.
name_items	Prefix for item names if specific_items is NULL.
n_items	Number of items if specific_items is NULL.
exclude_items	Optional vector of items to exclude.

Value

A list of model syntax strings for each factor solution.

Examples

```
# Generate models for 1 to 4 factor solutions using item prefix
models <- generate_modelos(
  n_factors = 4,
  name_items = "Item",
  n_items = 12
)

# View the 3-factor model syntax
cat(models[[3]])

# Generate models excluding specific items
models2 <- generate_modelos(
  n_factors = 3,
  name_items = "Q",
  n_items = 10,
  exclude_items = c("Q3", "Q7")
```

```
)

# Generate models using specific item names
models3 <- generate_modelos(
  n_factors = 2,
  specific_items = c("Anxiety1", "Anxiety2", "Anxiety3",
                     "Depression1", "Depression2", "Depression3")
)
cat(models3[[2]])
```

`generate_model_lavaan` *Generate Lavaan Model with Thresholds*

Description

Generates complete lavaan model syntax including factor loadings and item thresholds.

Usage

```
generate_model_lavaan(num_items, item_value, t_values, excluded_items = "none")
```

Arguments

`num_items`	Number of items in the model.
`item_value`	Vector of loading values for each item.
`t_values`	Vector of threshold values.
`excluded_items`	Items to exclude (commented out), or "none".

Value

A character string with complete lavaan model syntax.

Examples

```
# Generate a simple 1-factor model with 5 items and 4 thresholds
model <- generate_model_lavaan(
  num_items = 5,
  item_value = c(0.7, 0.8, 0.75, 0.6, 0.65),  # Factor loadings
  t_values = c(-1.5, -0.5, 0.5, 1.5)           # Threshold values
)
cat(model)
# Output:
# F1 =~ 0.7*Item1 + 0.8*Item2 + 0.75*Item3 + 0.6*Item4 + 0.65*Item5
# F1 ~~ 1*F1
# Item1 | -1.5*t1
# Item1 | -0.5*t2
# ...
```

```
# Generate model with some items excluded (commented out)
model2 <- generate_model_lavaan(
  num_items = 5,
  item_value = c(0.7, 0.8, 0.75, 0.6, 0.65),
  t_values = c(-1.5, -0.5, 0.5, 1.5),
  excluded_items = c("Item3t1", "Item3t2")  # Exclude Item3 thresholds 1 and 2
)
cat(model2)
```

`generate_summary` *Generate Summary Statistics*

Description

Generates summary statistics for sociodemographic variables.

Usage

```
generate_summary(data, variables)
```

Arguments

`data`	Data frame containing the variables.
`variables`	Character vector of variable names to summarize.

Value

A list of summary tables for each variable.

`generate_table` *Generate Summary Table*

Description

Converts summary results into a formatted table.

Usage

```
generate_table(summary_results)
```

Arguments

`summary_results`	List of summary results from generate_summary.

Value

A data frame formatted as a summary table.

get_ave_cr *Average Variance Extracted and Composite Reliability*

Description

Computes the Average Variance Extracted (AVE) and Composite Reliability (CR) for each latent factor in a fitted lavaan CFA model, using standardized factor loadings and error variances.

Usage

```
get_ave_cr(fit)
```

Arguments

fit	A fitted lavaan object (e.g., from cfa).

Value

A data frame with one row per latent factor containing: latent, CR, and AVE, both rounded to three decimal places.

Examples

```
library(lavaan)
model <- 'F1 =~ x1 + x2 + x3
          F2 =~ x4 + x5 + x6'
fit <- cfa(model, data = HolzingerSwineford1939)
get_ave_cr(fit)
```

interpret_decision_SSV *Interpret Modification Index Decisions*

Description

Interprets modification indices and power decisions from SEM analysis.

Usage

```
interpret_decision_SSV(MI_Saris)
```

Arguments

MI_Saris	Data frame with modification indices and decision.pow column.

Value

A data frame with interpretations of each decision.

`invertir_items` *Reverse Score Items*

Description

Reverses the scoring of specified items in a data frame.

Usage

```
invertir_items(df, items, num_respuestas, comienza_con_cero = TRUE)
```

Arguments

`df`	Data frame containing the items.
`items`	Character vector of item names to reverse.
`num_respuestas`	Number of response options.
`comienza_con_cero`	Logical, whether responses start with 0 (default TRUE).

Value

Data frame with reversed items.

Examples

```
# Create sample data with 5-point Likert scale (1-5)
set.seed(123)
data <- data.frame(
  Item1 = sample(1:5, 100, replace = TRUE),
  Item2 = sample(1:5, 100, replace = TRUE),
  Item3 = sample(1:5, 100, replace = TRUE),  # Item to reverse
  Item4 = sample(1:5, 100, replace = TRUE),
  Item5 = sample(1:5, 100, replace = TRUE)   # Item to reverse
)

# Reverse score Items 3 and 5 (scale starts at 1)
data_reversed <- invertir_items(
  df = data,
  items = c("Item3", "Item5"),
  num_respuestas = 5,
  comienza_con_cero = FALSE
)

# Check the reversal: 1->5, 2->4, 3->3, 4->2, 5->1
head(data[, c("Item3", "Item5")])
```

```
head(data_reversed[, c("Item3", "Item5")])

# Example with 0-4 scale (starts at 0)
data2 <- data.frame(
  Q1 = sample(0:4, 100, replace = TRUE),
  Q2 = sample(0:4, 100, replace = TRUE)
)

data2_reversed <- invertir_items(
  df = data2,
  items = "Q1",
  num_respuestas = 5,
  comienza_con_cero = TRUE
)
# Check: 0->5, 1->4, 2->3, 3->2, 4->1
```

`Matrix_lambda` *Create Lambda Matrix for CFA*

Description

Creates a binary lambda matrix specifying item-factor relationships for CFA models.

Usage

```
Matrix_lambda(n_items, factores, n_tamanos)
```

Arguments

`n_items`	Total number of items.
`factores`	Number of factors.
`n_tamanos`	Vector specifying the number of items per factor.

Value

A matrix with 1s indicating item-factor loadings.

`multi_cfa` *Run Multiple CFA Models*

Description

Fits multiple CFA models and extracts fit indices, modification indices, and reliability.

Usage

```
multi_cfa(modelos, data, estimator, ordered = TRUE, orthogonal_indices = NULL)
```

Arguments

`modelos`	List of lavaan model syntax strings.
`data`	Data frame with item responses.
`estimator`	Estimation method (e.g., "WLSMV").
`ordered`	Logical, whether variables are ordered/categorical (default: TRUE).
`orthogonal_indices`	Optional vector of model indices to fit with orthogonal factors.

Value

A list with fits, fit indices, modification indices, correlations, and reliability.

Examples

```
# Create sample data
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE),
  Item7 = sample(1:5, n, replace = TRUE),
  Item8 = sample(1:5, n, replace = TRUE),
  Item9 = sample(1:5, n, replace = TRUE)
)

# Define multiple CFA models to compare
model1 <- "F1 =~ Item1 + Item2 + Item3 + Item4 + Item5 + Item6 + Item7 + Item8 + Item9"

model2 <- "
  F1 =~ Item1 + Item2 + Item3
  F2 =~ Item4 + Item5 + Item6
  F3 =~ Item7 + Item8 + Item9
```

```
"

model3 <- "
  F1 =~ Item1 + Item2 + Item3 + Item4 + Item5
  F2 =~ Item6 + Item7 + Item8 + Item9
"

modelos <- list(model1, model2, model3)

# Run multiple CFAs
results <- multi_cfa(
  modelos = modelos,
  data = data,
  estimator = "WLSMV",
  ordered = TRUE
)

# Compare fit indices across models
results$bondades_ajuste

# Check modification indices for Model 2
results$indices_modificacion[[2]]

# View factor correlations for Model 2
results$correlaciones_factores$Modelo2

# Check reliability (omega) for each model
results$fiabilidad

# Fit model 3 with orthogonal factors
results_orth <- multi_cfa(
  modelos = modelos,
  data = data,
  estimator = "WLSMV",
  ordered = TRUE,
  orthogonal_indices = 3
)
```

`plot_cfa_stability` *Plot CFA Stability Results*

Description

Creates line plots showing CFA fit indices across sample sizes.

Usage

```
plot_cfa_stability(
  resultados,
```

```
  num_factors = 3,
  y_min_omega = 0.7,
  y_max_omega = 1,
  y_breaks_omega = 0.05,
  y_min_cfi = 0.9,
  y_max_cfi = 1,
  y_min_tli = 0.9,
  y_max_tli = 1,
  y_min_rmsea = 0,
  y_max_rmsea = 0.15,
  y_min_srmr = 0,
  y_max_srmr = 0.15,
  y_min_crmr = 0,
  y_max_crmr = 0.15,
  y_min_reliability = 0.5,
  y_max_reliability = 1,
  y_breaks_comparative = 0.02,
  y_breaks_absolutes = 0.05,
  hline_color = "red",
  xlab_size = 12,
  ylab_size = 12
)
```

Arguments

resultados	Data frame with bootstrap CFA results.
num_factors	Number of factors in the model (default 3).
y_min_omega	Minimum y-axis value for omega plot.
y_max_omega	Maximum y-axis value for omega plot.
y_breaks_omega	Y-axis breaks for omega plot.
y_min_cfi	Minimum y-axis value for CFI plot.
y_max_cfi	Maximum y-axis value for CFI plot.
y_min_tli	Minimum y-axis value for TLI plot.
y_max_tli	Maximum y-axis value for TLI plot.
y_min_rmsea	Minimum y-axis value for RMSEA plot.
y_max_rmsea	Maximum y-axis value for RMSEA plot.
y_min_srmr	Minimum y-axis value for SRMR plot.
y_max_srmr	Maximum y-axis value for SRMR plot.
y_min_crmr	Minimum y-axis value for CRMR plot.
y_max_crmr	Maximum y-axis value for CRMR plot.
y_min_reliability	Minimum y-axis value for reliability plot.
y_max_reliability	Maximum y-axis value for reliability plot.

y_breaks_comparative
: Y-axis breaks for comparative indices.

y_breaks_absolutes
: Y-axis breaks for absolute indices.

hline_color
: Color for reference lines (default "red").

xlab_size
: X-axis label size.

ylab_size
: Y-axis label size.

Value

A combined ggplot figure.

Examples

```
# First run boot_cfa_stability to get stability results
set.seed(123)
n <- 300
g <- rnorm(n)
t1 <- 0.7 * g + rnorm(n, 0, 0.7)
t2 <- 0.7 * g + rnorm(n, 0, 0.7)
sim_item <- function(t) {
  as.numeric(cut(t + rnorm(length(t), 0, 0.8), c(-Inf, -1, 0, 1, Inf)))
}
data <- data.frame(
  Item1 = sim_item(t1), Item2 = sim_item(t1), Item3 = sim_item(t1),
  Item4 = sim_item(t2), Item5 = sim_item(t2), Item6 = sim_item(t2)
)

model <- "
  F1 =~ Item1 + Item2 + Item3
  F2 =~ Item4 + Item5 + Item6
"

stability_results <- boot_cfa_stability(
  modelo = model,
  data = data,
  num_replicas = 10,
  estimator = "WLSMV",
  n_cores = 2
)

# Create stability plots
figure <- plot_cfa_stability(
  resultados = stability_results,
  num_factors = 2,
  y_min_omega = 0.6,
  y_max_omega = 1,
  y_min_cfi = 0.85,
  y_max_cfi = 1,
  hline_color = "red"
)
```

```
# Save the figure
ggplot2::ggsave(file.path(tempdir(), "stability_analysis.jpg"), figure,
                width = 12, height = 8, dpi = 300)
```

---

plot_cfa_stability_box *Plot CFA Stability Results (Boxplot)*

Description

Creates boxplots showing CFA fit indices distributions across sample sizes.

Usage

```
plot_cfa_stability_box(
  resultados,
  y_min_cfi = 0.9,
  y_max_cfi = 1,
  y_min_tli = 0.9,
  y_max_tli = 1,
  y_min_rmsea = 0,
  y_max_rmsea = 0.15,
  y_min_srmr = 0,
  y_max_srmr = 0.15,
  y_min_crmr = 0,
  y_max_crmr = 0.15,
  y_min_reliability = 0.5,
  y_max_reliability = 1,
  y_breaks = 0.01,
  y_breaks_reliability = 0.05,
  hline_color = "red",
  xlab_size = 12,
  ylab_size = 12,
  label_size = 10
)
```

Arguments

resultados	Data frame with bootstrap CFA results.
y_min_cfi	Minimum y-axis value for CFI plot.
y_max_cfi	Maximum y-axis value for CFI plot.
y_min_tli	Minimum y-axis value for TLI plot.
y_max_tli	Maximum y-axis value for TLI plot.
y_min_rmsea	Minimum y-axis value for RMSEA plot.

y_max_rmsea	Maximum y-axis value for RMSEA plot.
y_min_srmr	Minimum y-axis value for SRMR plot.
y_max_srmr	Maximum y-axis value for SRMR plot.
y_min_crmr	Minimum y-axis value for CRMR plot.
y_max_crmr	Maximum y-axis value for CRMR plot.
y_min_reliability	Minimum y-axis value for reliability plot.
y_max_reliability	Maximum y-axis value for reliability plot.
y_breaks	Y-axis breaks for fit indices.
y_breaks_reliability	Y-axis breaks for reliability.
hline_color	Color for reference lines (default "red").
xlab_size	X-axis label size.
ylab_size	Y-axis label size.
label_size	Label size.

Value

A combined ggplot figure with boxplots.

---

plot_cfa_stability_intervals

Plot CFA Stability with Confidence Intervals

Description

Creates plots with confidence intervals for CFA fit indices.

Usage

```
plot_cfa_stability_intervals(
  resultados,
  num_factors = 1,
  indices_to_plot = "All",
  indices_to_exclude = NULL,
  fill_color = "skyblue",
  alpha_ribbon = 0.2,
  rename_factors = NULL,
  ...
)
```

Arguments

resultados	Data frame with bootstrap CFA results.
num_factors	Number of factors (default 1).
indices_to_plot	Indices to plot ("All" or vector of names).
indices_to_exclude	Indices to exclude (default NULL).
fill_color	Fill color for ribbons (default "skyblue").
alpha_ribbon	Alpha for ribbons (default 0.2).
rename_factors	Named vector to rename factors (default NULL).
...	Additional arguments.

Value

A ggplot object with faceted confidence intervals.

---

plot_cfa_stability_intervals2 *Plot CFA Stability with Grouped Intervals*

Description

Creates grouped interval plots for CFA fit indices.

Usage

```
plot_cfa_stability_intervals2(
  resultados,
  num_factors = 1,
  alpha_ribbon = 0.2,
  rename_factors = NULL,
  ...
)
```

Arguments

resultados	Data frame with bootstrap CFA results.
num_factors	Number of factors (default 1).
alpha_ribbon	Alpha for ribbons (default 0.2).
rename_factors	Named vector to rename factors (default NULL).
...	Additional arguments.

Value

A list with three ggplot panels for different index groups.

plot_cfa_stability_pointrange
Grafico Pointrange de Estabilidad CFA

Description

Crea graficos pointrange (punto + intervalo de confianza) para visualizar la estabilidad de indices de ajuste y fiabilidad segun tamano muestral.

Usage

```
plot_cfa_stability_pointrange(
  resultados,
  indices = c("cfi.scaled", "tli.scaled", "rmsea.scaled", "srmr"),
  ci_level = 0.95,
  point_size = 2.5,
  line_width = 0.8,
 hline_values = list(cfi.scaled = 0.95, tli.scaled = 0.95, rmsea.scaled = 0.08, srmr =
    0.08, crmr = 0.05),
  hline_color = "red",
  hline_linetype = "dashed",
  facet_scales = "free_y",
  color_palette = NULL,
  theme_style = "minimal"
)
```

Arguments

resultados	Data frame con resultados de boot_cfa_stability.
indices	Vector de indices a graficar. Por defecto c("cfi.scaled", "tli.scaled", "rmsea.scaled", "srmr"). Usar "All" para todos los disponibles.
ci_level	Nivel de confianza para los intervalos (por defecto 0.95).
point_size	Tamano de los puntos (por defecto 2.5).
line_width	Grosor de las lineas del intervalo (por defecto 0.8).
hline_values	Lista nombrada con valores de referencia para lineas horizontales.
hline_color	Color de las lineas de referencia (por defecto "red").
hline_linetype	Tipo de linea de referencia (por defecto "dashed").
facet_scales	Escalas para facetas: "free_y", "fixed", etc. (por defecto "free_y").
color_palette	Paleta de colores. Por defecto usa escala de grises.
theme_style	Estilo del tema: "minimal", "bw", "classic" (por defecto "minimal").

Value

Un objeto ggplot.

Examples

```
set.seed(123)
n <- 300
g <- rnorm(n)
t1 <- 0.7 * g + rnorm(n, 0, 0.7)
t2 <- 0.7 * g + rnorm(n, 0, 0.7)
sim_item <- function(t) {
  as.numeric(cut(t + rnorm(length(t), 0, 0.8), c(-Inf, -1, 0, 1, Inf)))
}
data <- data.frame(
  Item1 = sim_item(t1), Item2 = sim_item(t1), Item3 = sim_item(t1),
  Item4 = sim_item(t2), Item5 = sim_item(t2), Item6 = sim_item(t2)
)
model <- "F1 =~ Item1 + Item2 + Item3
          F2 =~ Item4 + Item5 + Item6"
resultado_stability <- boot_cfa_stability(
  modelo = model, data = data, num_replicas = 10,
  estimator = "WLSMV", n_cores = 2
)
plot_cfa_stability_pointrange(resultado_stability)
plot_cfa_stability_pointrange(resultado_stability, indices = c("cfi.scaled", "rmsea.scaled"))
```

---

plot_cfa_stability_ridgeline

Grafico Ridgeline de Estabilidad CFA

Description

Crea graficos ridgeline (joy plots) para visualizar la distribucion de los indices de ajuste y fiabilidad segun tamano muestral.

Usage

```
plot_cfa_stability_ridgeline(
  resultados,
  indices = c("cfi.scaled", "tli.scaled", "rmsea.scaled", "srmr"),
  scale_ridges = 1.2,
  alpha_fill = 0.7,
  fill_color = NULL,
  gradient_colors = c("steelblue", "darkblue"),
  show_quantiles = TRUE,
  quantile_lines = c(0.025, 0.5, 0.975),
 vline_values = list(cfi.scaled = 0.95, tli.scaled = 0.95, rmsea.scaled = 0.08, srmr =
    0.08, crmr = 0.05),
  vline_color = "red",
  vline_linetype = "dashed",
  theme_style = "minimal",
```

```
  rel_min_height = 0.01,
  lang = c("es", "en")
)
```

Arguments

resultados	Data frame con resultados de boot_cfa_stability.
indices	Vector de indices a graficar. Por defecto c("cfi.scaled", "tli.scaled", "rmsea.scaled", "srmr"). Usar "All" para todos los disponibles.
scale_ridges	Escala de superposicion de las crestas (por defecto 1.2).
alpha_fill	Transparencia del relleno (por defecto 0.7).
fill_color	Color de relleno. Si es NULL, usa gradiente por porcentaje.
gradient_colors	Vector de 2 colores para el gradiente (por defecto c("steelblue", "darkblue")).
show_quantiles	Mostrar lineas de cuantiles (por defecto TRUE).
quantile_lines	Vector de cuantiles a mostrar (por defecto c(0.025, 0.5, 0.975)).
vline_values	Lista nombrada con valores de referencia para lineas verticales.
vline_color	Color de las lineas de referencia (por defecto "red").
vline_linetype	Tipo de linea de referencia (por defecto "dashed").
theme_style	Estilo del tema: "minimal", "bw", "classic" (por defecto "minimal").
rel_min_height	Altura minima relativa para recortar colas (por defecto 0.01).
lang	Idioma de las etiquetas: "es" (por defecto) o "en".

Value

Un objeto ggplot.

Examples

```
set.seed(123)
n <- 300
g <- rnorm(n)
t1 <- 0.7 * g + rnorm(n, 0, 0.7)
t2 <- 0.7 * g + rnorm(n, 0, 0.7)
sim_item <- function(t) {
  as.numeric(cut(t + rnorm(length(t), 0, 0.8), c(-Inf, -1, 0, 1, Inf)))
}
data <- data.frame(
  Item1 = sim_item(t1), Item2 = sim_item(t1), Item3 = sim_item(t1),
  Item4 = sim_item(t2), Item5 = sim_item(t2), Item6 = sim_item(t2)
)
model <- "F1 =~ Item1 + Item2 + Item3
          F2 =~ Item4 + Item5 + Item6"
resultado_stability <- boot_cfa_stability(
  modelo = model, data = data, num_replicas = 10,
  estimator = "WLSMV", n_cores = 2
)
```

```
plot_cfa_stability_ridgeline(resultado_stability)
plot_cfa_stability_ridgeline(resultado_stability, indices = c("cfi.scaled", "rmsea.scaled"))
```

`plot_erp` *Plot Extreme-Response Proportion*

Description

Creates a horizontal bar chart of ERP values per item, with a dashed reference line at the chosen threshold. Items exceeding the threshold are visually distinguished by fill colour.

Usage

```
plot_erp(erp_table, threshold = 0.3)
```

Arguments

`erp_table`	A data frame produced by `erp_by_item`.
`threshold`	Numeric threshold for the reference line (default 0.30).

Value

A `ggplot` object.

Examples

```
set.seed(123)
df <- data.frame(
  IT1 = sample(1:5, 200, replace = TRUE),
  IT2 = sample(1:5, 200, replace = TRUE),
  IT3 = sample(1:5, 200, replace = TRUE)
)
erp <- erp_by_item(df, c("IT1", "IT2", "IT3"))
plot_erp(erp)
```

Plot_Likert *Plot Likert Scale Items*

Description

Creates a visualization for Likert scale items.

Usage

```
Plot_Likert(Data, name_items, ranges, exclude = NULL, text_size = 3)
```

Arguments

Data	Data frame containing the survey data.
name_items	Prefix for item names.
ranges	Range of item numbers to include.
exclude	Items to exclude (default NULL).
text_size	Text size for labels (default 3).

Value

A ggplot object.

Examples

```
# Create sample survey data with Likert items (1-5 scale)
set.seed(123)
n <- 200
survey_data <- data.frame(
  Q1 = sample(1:5, n, replace = TRUE),
  Q2 = sample(1:5, n, replace = TRUE),
  Q3 = sample(1:5, n, replace = TRUE),
  Q4 = sample(1:5, n, replace = TRUE),
  Q5 = sample(1:5, n, replace = TRUE),
  Q6 = sample(1:5, n, replace = TRUE)
)

# Plot all items Q1 to Q6
plot <- Plot_Likert(
  Data = survey_data,
  name_items = "Q",
  ranges = 1:6
)
print(plot)

# Plot items Q1 to Q4, excluding Q2
plot2 <- Plot_Likert(
  Data = survey_data,
```

```
  name_items = "Q",
  ranges = 1:4,
  exclude = 2,
  text_size = 4
)
print(plot2)
```

`Plot_Likert2` *Plot Likert Scale Items (Diverging Stacked Bar)*

Description

Creates a diverging stacked bar chart visualization for Likert scale items.

Usage

```
Plot_Likert2(
  data,
  prefix = "IS",
  labels = c("Not at all true", "Completely true"),
  palette = "Blues",
  text_size = 3,
  threshold = 5,
  show_text = TRUE,
  item_range = 1:21
)
```

Arguments

`data`	Data frame containing the survey data.
`prefix`	Prefix for item names (default "IS").
`labels`	Labels for the scale endpoints.
`palette`	Color palette name from RColorBrewer (default "Blues").
`text_size`	Text size for labels (default 3).
`threshold`	Threshold for showing text labels (default 5).
`show_text`	Logical, whether to show text labels (default TRUE).
`item_range`	Range of item numbers to include.

Value

A ggplot object.

Examples

```
# Create sample survey data with binary Likert items
set.seed(123)
n <- 200
survey_data <- data.frame(
  IS1 = sample(1:2, n, replace = TRUE),
  IS2 = sample(1:2, n, replace = TRUE),
  IS3 = sample(1:2, n, replace = TRUE),
  IS4 = sample(1:2, n, replace = TRUE),
  IS5 = sample(1:2, n, replace = TRUE)
)

# Create diverging stacked bar chart
plot <- Plot_Likert2(
  data = survey_data,
  prefix = "IS",
  labels = c("Disagree", "Agree"),
  palette = "Blues",
  text_size = 3,
  threshold = 5,
  show_text = TRUE,
  item_range = 1:5
)
print(plot)

# Example with 5-point scale
survey_data2 <- data.frame(
  Q1 = sample(1:5, n, replace = TRUE),
  Q2 = sample(1:5, n, replace = TRUE),
  Q3 = sample(1:5, n, replace = TRUE),
  Q4 = sample(1:5, n, replace = TRUE)
)

plot2 <- Plot_Likert2(
  data = survey_data2,
  prefix = "Q",
  labels = c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"),
  palette = "RdYlBu",
  item_range = 1:4
)
print(plot2)
```

`plot_mediation_chord` *Chord diagram of distal-predictor by mediator associations*

Description

Visualización circular de cuerdas (chord diagram) que muestra las asociaciones bivariadas entre los predictores distales y los indicadores de los mediadores en un modelo de mediación múltiple

paralela. Replica el estilo de la Figura 1 de Wang et al. (2025, BMC Medicine): cada arco perimetral representa una variable agrupada por bloque conceptual (color), y cada cuerda interna representa una asociación con grosor proporcional a $|\beta|$ y color por signo.

Usage

```
plot_mediation_chord(
  fit,
  predictors,
  mediators,
  outcome = NULL,
  node_groups = NULL,
  node_labels = NULL,
  palette = NULL,
  sig_threshold = 0.05,
  show_n_s = TRUE,
  sig_alpha = 0.15,
  positive_color = NULL,
  negative_color = NULL,
  chord_width_range = c(0.5, 6),
  gap_degree = 2,
  big_gap = 8,
  label_cex = 0.85,
  title = NULL
)
```

Arguments

`fit`	Objeto `lavaan` ajustado.
`predictors`	Vector de IDs de los predictores distales (X) tal como aparecen en `rhs` de las regresiones `outcome ~ X`.
`mediators`	Vector de IDs de los mediadores (M) tal como aparecen en `lhs` de las regresiones `M ~ X`.
`outcome`	ID del outcome opcional (para incluir cuerdas X -> Y).
`node_groups`	Vector *named* que mapea cada ID a su bloque conceptual (por ejemplo, `c(sueno_num = "Hábito", DASS_dep = "DASS")`). Si `NULL`, se usa "X" para predictores y "M" para mediadores.
`node_labels`	Vector *named* de etiquetas para mostrar en el arco. Si `NULL`, se usan los IDs.
`palette`	Vector *named* de colores por bloque conceptual.
`sig_threshold`	Umbral de significancia para diferenciar cuerdas (default `0.05`).
`show_n_s`	Lógico. Mostrar también las asociaciones no significativas con transparencia (default `TRUE`).
`sig_alpha`	Transparencia para cuerdas n.s. (default `0.15`).
`positive_color, negative_color`	Colores para asociaciones positivas y negativas. Si `NULL`, se hereda el color del bloque del predictor.

`chord_width_range`
: Rango de grosor de las cuerdas (default `c(0.5, 6)`).

`gap_degree`
: Espacio en grados entre arcos (default `2`).

`big_gap`
: Espacio en grados entre bloques conceptuales (default `8`).

`label_cex`
: Tamaño de las etiquetas perimetrales (default `0.85`).

`title`
: Título del gráfico.

Details

La función calcula los β estandarizados de `lavaan::standardizedSolution()` para cada par (predictor, mediador) y construye una matriz adyacencia que se pasa a `circlize::chordDiagram()`. Requiere el paquete `circlize`.

Value

Invoca `circlize::chordDiagram()` y devuelve invisiblemente la matriz de asociaciones usada para el dibujo.

References

Wang, X., Cao, Z., Yin, S., Duan, T., Sun, T., & Xu, C. (2025). Childhood maltreatment and depression: Mediating role of lifestyle factors, personality traits, adult traumas, and social connections among middle-aged and elderly participants. *BMC Medicine, 23*, 319. doi:10.1186/s12916025-041472

Gu, Z., Gu, L., Eils, R., Schlesner, M., & Brors, B. (2014). circlize implements and enhances circular visualization in R. *Bioinformatics, 30*(19), 2811–2812. doi:10.1093/bioinformatics/btu393

Examples

```
library(lavaan); library(circlize)

set.seed(123)
n <- 300
X1 <- rnorm(n); X2 <- rnorm(n); X3 <- rnorm(n)
M1 <- 0.5 * X1 + 0.3 * X2 + rnorm(n, 0, 0.8)
M2 <- 0.4 * X2 + 0.3 * X3 + rnorm(n, 0, 0.8)
Y  <- 0.4 * M1 + 0.3 * M2 + 0.2 * X1 + rnorm(n, 0, 0.7)
your_data <- data.frame(X1, X2, X3, M1, M2, Y)

mod <- '
  M1 ~ X1 + X2 + X3
  M2 ~ X1 + X2 + X3
  Y  ~ M1 + M2
'
fit <- sem(mod, data = your_data)

plot_mediation_chord(
  fit,
  predictors = c("X1","X2","X3"),
  mediators  = c("M1","M2"),
```

```
  node_groups = c(X1 = "Block A", X2 = "Block A", X3 = "Block B",
                  M1 = "Mediator", M2 = "Mediator"),
  palette = c("Block A" = "#FFE082", "Block B" = "#B0BEC5",
              "Mediator" = "#F4A8A8")
)
```

`plot_mediation_donuts` *Donut panel of mediation proportions per parallel mediator*

Description

Genera un panel de *donuts* (gráficos circulares con centro hueco) que cuantifica visualmente la proporción del efecto total mediado por cada uno de los mediadores paralelos para un predictor distal determinado. Cada donut muestra el porcentaje de la asociación X -> Y atribuible al mediador en cuestión, replicando el estilo de la Figura 2 superior de Wang et al. (2025, BMC Medicine).

Usage

```
plot_mediation_donuts(
  fit,
  predictor,
  mediators,
  outcome = "suic",
  predictor_regression = NULL,
  indirect_prefix = "ind_",
  mediator_colors = NULL,
  remaining_color = "grey85",
  show_total = TRUE,
  ncol = NULL,
  title = NULL,
  subtitle = NULL
)
```

Arguments

`fit`	Objeto `lavaan` ajustado.
`predictor`	Stem del predictor distal (tal como aparece tras los prefijos `ind_` y `tot_` en las definiciones `:=` del modelo). Debe coincidir con uno de los predictores incluidos en el modelo de senderos.
`mediators`	Vector *named* con los IDs de los mediadores donde el nombre es la etiqueta a mostrar y el valor es el ID en `lavaan`. Por ejemplo, `c("Depresión" = "DASS_dep", "Ansiedad" = "DASS_anx")`.
`outcome`	ID del outcome (default `"suic"`).
`predictor_regression`	Nombre del predictor en la regresión `outcome ~ X` (si difiere del stem). Por ejemplo, si el stem es `"sueno"` y la regresión usa `sueno_num`, especificar `"sueno_num"`.

`indirect_prefix`
: Prefijo de las definiciones de efectos indirectos por mediador (default `"ind_"`).

`mediator_colors`
: Vector *named* de colores para cada mediador. Si `NULL`, se usa una paleta pastel por defecto.

`remaining_color`
: Color para la porción no mediada (default `"grey85"`).

`show_total`
: Lógico. Si `TRUE`, añade un panel adicional con la proporción mediada total (suma de todos los mediadores; default `TRUE`).

`ncol`
: Número de columnas en el panel (default es la longitud de `mediators` + opcionalmente uno para el total).

`title`
: Título del panel.

`subtitle`
: Subtítulo del panel.

Details

La función calcula la *proporción mediada* por cada mediador como

$$P_j = \frac{a_j \times b_j}{|a_1 b_1| + |a_2 b_2| + \ldots + |a_m b_m|}$$

donde a_j es el coeficiente `M_j ~ X` y b_j es el coeficiente `Y ~ M_j`, ambos estandarizados. Cada donut representa el valor absoluto de P_j con la fracción restante en gris, acompañado por el porcentaje en el centro y el signo del efecto en la parte inferior.

Requiere los paquetes `ggplot2` y `patchwork`.

Value

Un objeto `patchwork` (composición de `ggplots`) listo para imprimir o guardar con `ggsave()`.

References

Wang, X., Cao, Z., Yin, S., Duan, T., Sun, T., & Xu, C. (2025). Childhood maltreatment and depression: Mediating role of lifestyle factors, personality traits, adult traumas, and social connections among middle-aged and elderly participants. *BMC Medicine, 23*, 319. doi:10.1186/s12916025-041472

Examples

```
library(lavaan); library(patchwork)

set.seed(123)
n <- 300
sueno_num <- rnorm(n)
DASS_dep <- 0.5 * sueno_num + rnorm(n, 0, 0.8)
DASS_anx <- 0.4 * sueno_num + rnorm(n, 0, 0.8)
suic <- 0.4 * DASS_dep + 0.3 * DASS_anx + 0.2 * sueno_num + rnorm(n, 0, 0.7)
dat <- data.frame(sueno_num, DASS_dep, DASS_anx, suic)
```

```
mod <- '
  DASS_dep ~ sueno_num
  DASS_anx ~ sueno_num
  suic ~ DASS_dep + DASS_anx + sueno_num
'
fit <- sem(mod, data = dat)

plot_mediation_donuts(
  fit,
  predictor = "sueno",
  mediators = c("Depresión" = "DASS_dep", "Ansiedad" = "DASS_anx"),
  outcome = "suic",
  predictor_regression = "sueno_num",
  title = "Mediación de la calidad del sueño sobre la conducta suicida"
)
```

`plot_mediation_forest` *Forest plot of direct, indirect, and total effects in a SEM model*

Description

Visualización alternativa al diagrama de senderos clásico cuando el modelo tiene muchos predictores distales. Genera un *forest plot* horizontal que ordena cada predictor por la magnitud absoluta de su efecto β estandarizado y muestra simultáneamente los efectos directos, indirectos ($a \times b$) y totales con intervalos de confianza al 95 %. Sigue la recomendación de Fife y Mendoza (2018) sobre presentar la descomposición de efectos en un panel separado en lugar de saturar el path diagram con los tres tipos de coeficientes.

Usage

```
plot_mediation_forest(
  fit,
  predictors,
  outcome = "suic",
  predictor_regression = NULL,
  indirect_prefix = "ind_",
  total_prefix = "tot_",
  predictor_labels = NULL,
  effect_levels = c("Directo", "Indirecto", "Total"),
 effect_colors = c(Directo = "#5B8DBE", Indirecto = "#E07A5F", Total = "#3D405B"),
  layout = c("facet", "stacked"),
  effect_dodge = 0.55,
  sig_threshold = 0.05,
  sort_by = c("total", "indirect", "direct"),
  show_zero = TRUE,
  show_p_text = TRUE,
  show_beta_text = FALSE,
```

```
  p_text_position = c("right", "panel"),
  point_size = 2.4,
  error_width = 0.7,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x_lim = NULL,
  facet_scales = "fixed"
)
```

Arguments

`fit`	Objeto `lavaan` ajustado con efectos definidos vía `:=` en la sintaxis lavaan (por ejemplo, `ind_X := a*b` para el efecto indirecto y `tot_X := c + ind_X` para el total).
`predictors`	Vector con los *stems* de los predictores distales, tal como aparecen tras los prefijos `ind_` y `tot_` en las definiciones `:=` del modelo. Por ejemplo, si las definiciones son `ind_sueno := a*b` y `tot_sueno := c + ind_sueno`, el stem es `"sueno"`.
`outcome`	ID del outcome (default `"suic"`; usar el nombre del `lhs` de la regresión final del modelo).
`predictor_regression`	Vector *named* que mapea cada `stem` en `predictors` con el nombre exacto de la variable como aparece en la regresión `outcome ~ X`. Útil cuando el modelo usa stems cortos en las definiciones `:=` pero nombres completos en las regresiones (por ejemplo, `ind_sueno` junto con `suic ~ sueno_num`). Si `NULL`, se asume que coinciden.
`indirect_prefix`	Prefijo de los parámetros definidos para los efectos indirectos (default `"ind_"`). Se busca `paste0(prefix, predictor)` dentro de `lavaan::standardizedSolution(fit)` con `op == ":="`.
`total_prefix`	Prefijo de los parámetros definidos para los efectos totales (default `"tot_"`).
`predictor_labels`	Vector *named* para mostrar etiquetas legibles en el eje vertical. Sus nombres deben coincidir con los `stems` de `predictors`.
`effect_levels`	Vector de longitud máximo 3 con las componentes a graficar (default `c("Directo","Indirecto","Tota`). Permite ocultar tipos.
`effect_colors`	Vector con los colores asignados a las tres componentes. Default `c(Directo = "#5B8DBE", Indirecto = "#E07A5F", Total = "#3D405B")`.
`layout`	Disposición visual: `"facet"` (default) coloca cada tipo de efecto en un panel separado lado a lado, evitando el solapamiento de etiquetas p; `"stacked"` reproduce la versión clásica con los tres efectos en un único panel y *dodge* vertical (modo legado).
`effect_dodge`	Distancia vertical entre las tres componentes en modo `"stacked"` (default `0.55`). Ignorado en modo `"facet"`.
`sig_threshold`	Umbral de significancia para destacar coeficientes (default `0.05`).

`sort_by`	Forma de ordenar el eje vertical: `"total"` (default, por $\|\beta\|$ total descendente), `"indirect"` o `"direct"`.
`show_zero`	Lógico. Dibujar la línea vertical de referencia en $\beta = 0$ (default `TRUE`).
`show_p_text`	Lógico. Anotar el valor de p junto a cada punto (default `TRUE`).
`show_beta_text`	Lógico. Anotar el valor de β a la izquierda del IC (default `FALSE`; útil en modo `"facet"` cuando se quiere tabla de coeficientes integrada).
`p_text_position`	Posición horizontal de la etiqueta p: `"right"` (default; al final del IC superior) o `"panel"` (al margen derecho de cada panel; recomendado en modo `"facet"`).
`point_size`	Tamaño de los puntos (default 2.4).
`error_width`	Grosor de las barras de IC (default 0.7).
`title, subtitle, caption`	Cadenas de texto opcionales.
`x_lim`	Vector numérico de longitud 2 con los límites del eje X (default `NULL` = automáticos).
`facet_scales`	Pasado a `ggplot2::facet_wrap()` cuando `layout = "facet"` (default `"fixed"`; usar `"free_x"` si las escalas de los efectos difieren mucho).

Details

La función toma los efectos definidos con `:=` en la sintaxis lavaan y los reorganiza en un *long format* apto para `ggplot2`. Cada predictor genera un bloque vertical con los tres tipos de efecto (cuando están disponibles); las barras representan IC 95 % obtenidos directamente de `lavaan::standardizedSolution()`.

Comparado con el path diagram de `plot_path_mediation`, este formato es preferible cuando ≥ 8 predictores distales saturan visualmente el diagrama de tres columnas, o cuando se necesita comunicar la descomposición *directo* + *indirecto* = *total* con precisión cuantitativa.

Value

Objeto `ggplot` listo para imprimir o guardar con `ggsave()`.

References

Fife, D. A., & Mendoza, J. L. (2018). Beyond path diagrams: Enhancing applied structural equation modeling research through data visualization. *Addictive Behaviors, 94*, 232–239. doi:10.1016/j.addbeh.2018.08.030

Preacher, K. J., & Hayes, A. F. (2008). Asymptotic and resampling strategies for assessing and comparing indirect effects in multiple mediator models. *Behavior Research Methods, 40*(3), 879–891. doi:10.3758/BRM.40.3.879

Examples

```
library(lavaan)

set.seed(123)
n <- 300
X1 <- rnorm(n); X2 <- rnorm(n); X3 <- rnorm(n)
```

```
M1 <- 0.5 * X1 + 0.3 * X2 + rnorm(n, 0, 0.8)
M2 <- 0.4 * X2 + 0.3 * X3 + rnorm(n, 0, 0.8)
Y  <- 0.4 * M1 + 0.3 * M2 + 0.2 * X1 + rnorm(n, 0, 0.7)
your_data <- data.frame(X1, X2, X3, M1, M2, Y)

mod <- '
  M1 ~ a1*X1 + a2*X2
  Y  ~ b1*M1 + c1*X1 + c2*X2
  ind_X1 := a1*b1
  ind_X2 := a2*b1
  tot_X1 := c1 + ind_X1
  tot_X2 := c2 + ind_X2
'
fit <- sem(mod, data = your_data)

plot_mediation_forest(
  fit,
  predictors = c("X1","X2"),
  outcome = "Y",
  predictor_labels = c(X1 = "Predictor 1", X2 = "Predictor 2")
)
```

`plot_multi_sem` *Plot Multiple SEM Models*

Description

Creates a panel of SEM path diagrams for multiple models. The panel can be arranged in a single row (default) or in an arbitrary grid via the `nrow` / `ncol` arguments; empty cells are left blank when the number of models does not fill the grid.

Usage

```
plot_multi_sem(
  models,
  titles = NULL,
  layout = "tree2",
  rotation = 2,
  whatLabels = "std",
  residuals = FALSE,
  intercepts = FALSE,
  thresholds = FALSE,
  sizeMan = 10,
  sizeMan2 = 5,
  sizeLat = 12,
  label.cex = 1,
  edge.label.cex = 2,
  edge.color = "grey40",
```

```
  color = list(lat = "grey80", man = "grey90"),
  mar = c(4, 4, 4, 4),
  outerMar = c(1, 1, 3, 1),
  nrow = NULL,
  ncol = NULL,
  show_fit_indices = TRUE,
  fit_indices = c("cfi", "tli", "rmsea", "srmr"),
  use_scaled = FALSE,
  model_descriptions = NULL,
  custom_titles = NULL,
  title.cex = 0.8,
  title.font = 2,
  save_plot = FALSE,
  filename = NULL,
  file_format = "png",
  width_per = 4,
  height = 4,
  dpi = 300,
  units = "in"
)
```

Arguments

models	List of lavaan model objects.
titles	Titles for each model (default NULL).
layout	Layout type (default "tree2").
rotation	Rotation value (default 2).
whatLabels	Labels to show (default "std").
residuals	Show residuals (default FALSE).
intercepts	Show intercepts (default FALSE).
thresholds	Show thresholds (default FALSE).
sizeMan	Manifest variable size (default 10).
sizeMan2	Second manifest size (default 5).
sizeLat	Latent variable size (default 12).
label.cex	Label size multiplier (default 1).
edge.label.cex	Edge label size (default 2).
edge.color	Edge color (default "grey40").
color	Color list for nodes.
mar	Inner margins.
outerMar	Outer margins.
nrow	Number of rows in the panel grid. Default NULL (a single row). If only one of nrow/ncol is supplied, the other is derived from the number of models.
ncol	Number of columns in the panel grid. Default NULL (one column per model when nrow is also NULL).

show_fit_indices
: Show fit indices (default TRUE).

fit_indices
: Which fit indices to show.

use_scaled
: Use scaled indices (default FALSE).

model_descriptions
: Model descriptions.

custom_titles
: Custom titles.

title.cex
: Title size (default 0.8).

title.font
: Title font (default 2).

save_plot
: Save plot to file (default FALSE).

filename
: Output filename without extension (default NULL; required when `save_plot = TRUE`, e.g. `file.path(tempdir(), "sem_plot")`).

file_format
: Output format (png, pdf, tiff, jpeg).

width_per
: Width per plot in inches (per column).

height
: Height in inches (per row).

dpi
: Resolution.

units
: Units for dimensions.

Value

NULL (plots are drawn to device).

Examples

```
library(lavaan)

# Create sample data
set.seed(123)
n <- 300
data <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

# Fit multiple models
model1 <- "F1 =~ Item1 + Item2 + Item3 + Item4 + Item5 + Item6"
model2 <- "F1 =~ Item1 + Item2 + Item3\nF2 =~ Item4 + Item5 + Item6"

fit1 <- cfa(model1, data = data, ordered = TRUE, estimator = "WLSMV")
fit2 <- cfa(model2, data = data, ordered = TRUE, estimator = "WLSMV")

# Plot multiple models side by side (single row, default)
plot_multi_sem(
```

```
  models = list(fit1, fit2),
  model_descriptions = c("1-Factor Model", "2-Factor Model"),
  show_fit_indices = TRUE,
  fit_indices = c("cfi", "tli", "rmsea", "srmr"),
  use_scaled = TRUE
)

# Arrange three models in two rows (2 x 2 grid; last cell left blank)
plot_multi_sem(
  models = list(fit1, fit2, fit1),
  model_descriptions = c("Model 1", "Model 2", "Model 3"),
  nrow = 2, ncol = 2,
  save_plot = TRUE, filename = file.path(tempdir(), "sem_grid"),
  width_per = 6, height = 6
)
```

`plot_path_cfa` *Path diagram for a CFA model (psychometric convention, ggplot2)*

Description

Genera un diagrama de senderos para un modelo CFA ajustado con `lavaan::cfa()` siguiendo la convencion psicometrica estandar: **factores latentes como elipses (circulos)**, **items observados como rectangulos**, y **cargas dibujadas en abanico** desde el borde del circulo del factor hacia cada item. Las correlaciones latentes se representan como curvas dobles entre factores. Por defecto el dibujo es lineal (blanco y negro), tipo diagrama de Brown (2015) o Kline; el usuario puede pedir paleta de color con `bw = FALSE` o pasando `palette`.

Usage

```
plot_path_cfa(
  fit,
  loading_threshold = 0.3,
  show_loading_labels = TRUE,
  loading_label_decimals = 2,
  loading_label_position = 0.55,
  correlate_factors = TRUE,
  show_correlation_labels = TRUE,
  correlation_curvature = 0.45,
  bw = TRUE,
  palette = NULL,
  factor_radius = 0.85,
  item_half_width = 0.65,
  item_half_height = 0.28,
  edge_width_range = c(0.4, 1.6),
  node_label_size = 3,
  edge_label_size = 2.7,
```

```
  item_x = 0,
  factor_x = 8,
  item_spacing = 1,
  group_gap = 1.2,
  fit_indices = c("cfi.scaled", "tli.scaled", "rmsea.scaled", "srmr"),
  show_n = TRUE,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  factor_label_map = NULL
)
```

Arguments

`fit`	Objeto `lavaan` ajustado con `lavaan::cfa()` (o `lavaan::sem()` con bloque de medida).
`loading_threshold`	Numeric. Cargas con $\|\lambda\|$ bajo el umbral se omiten (default 0.30).
`show_loading_labels`	Logical. Imprimir el valor de λ sobre cada flecha (default `TRUE`).
`loading_label_decimals`	Integer. Decimales para etiquetas (default 2).
`loading_label_position`	Numeric en (0, 1). Posicion de la etiqueta a lo largo de la flecha medida desde el factor; default 0.55 (mas cerca del item).
`correlate_factors`	Logical. Dibujar correlaciones latentes ϕ entre factores (default `TRUE`).
`show_correlation_labels`	Logical. Mostrar el valor de ϕ junto a cada curva (default `TRUE`).
`correlation_curvature`	Numeric. Curvatura de las correlaciones (default 0.45).
`bw`	Logical. Si `TRUE` (default), dibujo lineal en blanco y negro al estilo libro de texto. Si `FALSE` usa color: ya sea la `palette` provista o una paleta pastel automatica.
`palette`	Vector con nombres iguales a los factores. Si `NULL` se respeta `bw`: blanco y negro o pastel auto. Pasar `palette` fuerza color (override `bw`).
`factor_radius`	Numeric. Radio (en unidades del eje) de los circulos de factor (default 0.85).
`item_half_width, item_half_height`	Numeric. Medio ancho y media altura de los rectangulos de item (defaults 0.65 y 0.28).
`edge_width_range`	Vector de longitud 2 con el rango de grosor de las flechas (default `c(0.4, 1.6)`).
`node_label_size, edge_label_size`	Tamanos de fuente para los textos (defaults 3.0 y 2.7).
`item_x`	Numeric. Coordenada X de la columna de items (default 0).
`factor_x`	Numeric. Coordenada X de los centros de los circulos de factor (default 8).

`item_spacing` Numeric. Distancia vertical entre items consecutivos del mismo factor (default 1).

`group_gap` Numeric. Espacio adicional entre grupos de items de distintos factores (default 1.2).

`fit_indices` Vector de nombres de `lavaan::fitMeasures()` a incluir en el subtitulo.

`show_n` Logical. Anexar N al subtitulo (default `TRUE`).

`title`, `subtitle`, `caption` Cadenas para los componentes del plot.

`factor_label_map` Lista nombrada para renombrar factores (e.g. `list(F1 = "Cognitive", F2 = "Affective")`).

Details

Toma `lavaan::standardizedSolution()` y filtra:

- `op == "=~"` para construir las cargas factor → item.
- `op == "~~"` con `lhs != rhs` entre factores latentes para las correlaciones ϕ.

Cada flecha sale del **borde** del circulo del factor (no del centro) gracias a un calculo trigonometrico que las distribuye en abanico, evitando la convergencia visual al mismo punto.

Las varianzas residuales se omiten para mantener la lectura limpia. Si las necesitas explicitas, usa `semPlot::semPaths(..., what = "std")`.

Value

Un objeto `ggplot` listo para imprimir, guardar con `ggsave()` o componer con `patchwork`.

References

Brown, T. A. (2015). *Confirmatory factor analysis for applied research* (2nd ed.). Guilford Press.

Kline, R. B. (2023). *Principles and practice of structural equation modeling* (5th ed.). Guilford Press.

Examples

```
library(lavaan)

set.seed(123)
n <- 400
g <- rnorm(n)
sim_f <- function(fscore, k) {
  sapply(seq_len(k), function(i) {
    as.numeric(cut(0.8 * fscore + rnorm(length(fscore), 0, 0.6),
                   c(-Inf, -1, 0, 1, Inf)))
  })
}
F1s <- 0.6 * g + rnorm(n, 0, 0.8)
F2s <- 0.6 * g + rnorm(n, 0, 0.8)
```

```
F3s <- 0.6 * g + rnorm(n, 0, 0.8)
my_data <- data.frame(sim_f(F1s, 5), sim_f(F2s, 5), sim_f(F3s, 5))
names(my_data) <- paste0("ITEM", 1:15)

mod <- '
  F1 =~ ITEM1 + ITEM2 + ITEM3 + ITEM4 + ITEM5
  F2 =~ ITEM6 + ITEM7 + ITEM8 + ITEM9 + ITEM10
  F3 =~ ITEM11 + ITEM12 + ITEM13 + ITEM14 + ITEM15
'
fit <- cfa(mod, data = my_data, ordered = paste0("ITEM", 1:15),
           estimator = "WLSMV", std.lv = TRUE)

# Default: black & white (textbook style)
plot_path_cfa(fit, title = "DERS — final structure")

# With colour palette
plot_path_cfa(fit, bw = FALSE,
  palette = c(F1 = "#90CAF9", F2 = "#F4A8A8", F3 = "#A5D6A7"))
```

`plot_path_mediation` *Path-mediation diagram for SEM models with multiple distal predictors*

Description

Genera un diagrama de senderos en tres columnas (predictores distales → mediadores → outcome) a partir de un objeto `lavaan` ajustado. Las aristas tienen grosor proporcional a $|\beta|$ estandarizado, color por bloque de origen, y se atenúan cuando $p \geq .05$. Los nodos se rellenan con una paleta pastel agrupada por bloque conceptual. Diseñado para modelos con muchos predictores distales (10+) en los que `semPlot`/`lavaanPlot` producen layouts ilegibles.

Usage

```
plot_path_mediation(
  fit,
  nodes,
  edges_extra = NULL,
  x_positions = c(0, 6, 12),
  palette = NULL,
  sig_alpha = 0.18,
  edge_width_range = c(0.3, 2.4),
  sig_threshold = 0.05,
  show_n_s = TRUE,
  show_edge_labels = TRUE,
  edge_label_decimals = 2,
  shrink_dx = 1.2,
  shrink_dy = 0.7,
  node_size = 3.4,
```

```
  node_label_size = 3.4,
  edge_label_size = 2.7,
  fit_indices = c("cfi.robust", "tli.robust", "rmsea.robust", "srmr"),
  show_r2 = TRUE,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
 layer_labels = c("Predictores distales", "Mediadores", "Variable dependiente"),
  show_layer_labels = TRUE
)
```

Arguments

`fit`	Objeto `lavaan` ajustado con `lavaan::sem()` o `lavaan::cfa()`.
`nodes`	Data frame con columnas obligatorias `id` (nombre de la variable como aparece en `fit`), `label` (etiqueta a mostrar; usar "\n" para saltos de línea), `block` (categoría conceptual para el color del nodo), `layer` (1 = distal, 2 = mediador, 3 = outcome) y `y` (posición vertical dentro de la capa). Si `NULL`, la función intenta deducir el layout automáticamente a partir del modelo lavaan.
`edges_extra`	Lista opcional de pares `c(from, to)` adicionales que se quieran trazar más allá de las regresiones detectadas. Default `NULL`.
`x_positions`	Vector numérico de longitud 3 con las coordenadas X de las tres capas. Default `c(0, 6, 12)`.
`palette`	Vector con nombres iguales a las categorías de `block` del data frame `nodes`. Si `NULL` se usa una paleta pastel por defecto basada en `RColorBrewer::Pastel1`.
`sig_alpha`	Número entre 0 y 1 para la transparencia de las aristas no significativas (default `0.18`).
`edge_width_range`	Vector de longitud 2 con el rango de grosor de las aristas (default `c(0.3, 2.4)`).
`sig_threshold`	Umbral de significancia para diferenciar aristas (default `0.05`).
`show_n_s`	Lógico. Si `FALSE`, las aristas no significativas se omiten por completo (default `TRUE`).
`show_edge_labels`	Lógico. Mostrar el valor de β sobre cada arista significativa (default `TRUE`).
`edge_label_decimals`	Número de decimales para los labels de aristas (default 2).
`shrink_dx`, `shrink_dy`	Distancias en X e Y por las que se acortan las aristas para que no se solapen con los nodos (default 1.2 y 0.7).
`node_size`, `node_label_size`, `edge_label_size`	Tamaños de fuente para los textos.
`fit_indices`	Vector con nombres de `lavaan::fitMeasures()` a incluir en el subtítulo (default `c("cfi.robust","tli.robust","rmsea.robust","srmr")`).
`show_r2`	Lógico. Anexar los R^2 de las variables endógenas al subtítulo (default `TRUE`).

`title`, `subtitle`, `caption`
: Cadenas de texto para los componentes correspondientes. Si `subtitle = NULL`, se construye automáticamente con los `fit_indices` y los R^2 (cuando `show_r2 = TRUE`).

`layer_labels`
: Vector de longitud 3 con las etiquetas inferiores de las tres capas (default `c("Predictores distales","Mediadores","Variable dependiente")`).

`show_layer_labels`
: Lógico. Mostrar las etiquetas de capa al pie del gráfico (default `TRUE`).

Details

La función toma los coeficientes estandarizados de `lavaan::standardizedSolution()` y construye el data frame de aristas filtrando las regresiones (`op == "~"`) que conectan nodos del data frame `nodes`. Los efectos indirectos definidos con `:=` en la sintaxis lavaan no se grafican aquí; para visualizarlos en otra capa, use `plot_mediation_forest`.

El layout en tres columnas se inspira en Li et al. (2026, BMC Public Health, figura del modelo de mediación dual de presión académica) y en Bakioglu, Korkmaz y Ercan (2020, IJMHA, modelo de mediación del miedo a la COVID-19).

Value

Un objeto `ggplot` listo para imprimir, guardar con `ggsave()` o componer con `patchwork`.

References

Bakioglu, F., Korkmaz, O., & Ercan, H. (2020). Fear of COVID-19 and positivity: Mediating role of intolerance of uncertainty, depression, anxiety and stress. *International Journal of Mental Health and Addiction*. doi:10.1007/s1146902000331y

Li, Y., Wang, J., Luo, W., Zhou, H., Zhang, J., Xu, J., et al. (2026). Academic pressure and suicide risk among adolescents: A dual mediation model of depression and school cohesion. *BMC Public Health, 26*, 335. doi:10.1186/s12889025258053

Examples

```
library(lavaan)

set.seed(123)
n <- 300
X1 <- rnorm(n); X2 <- rnorm(n); X3 <- rnorm(n)
M1 <- 0.5 * X1 + 0.3 * X2 + rnorm(n, 0, 0.8)
M2 <- 0.4 * X2 + 0.3 * X3 + rnorm(n, 0, 0.8)
Y  <- 0.4 * M1 + 0.3 * M2 + 0.2 * X1 + rnorm(n, 0, 0.7)
your_data <- data.frame(X1, X2, X3, M1, M2, Y)

# Modelo de senderos: 3 predictores distales -> 2 mediadores -> 1 outcome
mod <- '
  M1 ~ X1 + X2 + X3
  M2 ~ X1 + X2 + X3
  Y  ~ M1 + M2 + X1 + X2 + X3
'
```

```
fit <- sem(mod, data = your_data, estimator = "MLR")

nodes <- data.frame(
  id    = c("X1","X2","X3","M1","M2","Y"),
  label = c("Estilo 1","Estilo 2","Demogr","Mediador 1","Mediador 2","Outcome"),
  block = c("Habito","Habito","Demogr","Afecto","Afecto","Outcome"),
  layer = c(1,1,1,2,2,3),
  y     = c(2,0,-2,1,-1,0)
)

plot_path_mediation(fit, nodes,
                    title = "Modelo de mediación afectiva",
                    palette = c(Habito = "#FFE082", Demogr = "#B0BEC5",
                                Afecto = "#F4A8A8", Outcome = "#90CAF9"))
```

`Preliminar_table` *Preliminary Response Frequency Table*

Description

Creates a table of response frequencies for items.

Usage

```
Preliminar_table(data)
```

Arguments

`data`	Data frame containing the items.

Value

A tibble with response frequencies as percentages.

Examples

```
# Create sample Likert data (1-5 scale)
set.seed(123)
data <- data.frame(
  Item1 = sample(1:5, 100, replace = TRUE),
  Item2 = sample(1:5, 100, replace = TRUE),
  Item3 = sample(1:5, 100, replace = TRUE)
)

# Get response frequency table
freq_table <- Preliminar_table(data)
print(freq_table)
# Output shows percentage of responses for each response option (1-5)
# Items   1     2     3     4     5
```

```
# 1      20.0  18.0  22.0  21.0  19.0
# 2      19.0  21.0  20.0  22.0  18.0
# 3      21.0  19.0  18.0  20.0  22.0
```

`safe_cfa` *Safe CFA Wrapper*

Description

Fits a CFA model using `cfa` with error handling. Returns `NULL` instead of raising an error when the model fails to converge or encounters estimation problems.

Usage

```
safe_cfa(syntax, data, items, estimator = "WLSMV", stdlv = TRUE)
```

Arguments

`syntax`	A character string with the lavaan model syntax.
`data`	A data frame containing the observed variables.
`items`	Character vector of item names to treat as ordered (categorical).
`estimator`	Estimator to use (default `"WLSMV"`).
`stdlv`	Logical; standardize latent variables (default `TRUE`).

Value

A fitted lavaan object, or `NULL` if the model failed.

Examples

```
set.seed(123)
mydata <- data.frame(
  x1 = sample(1:5, 200, replace = TRUE),
  x2 = sample(1:5, 200, replace = TRUE),
  x3 = sample(1:5, 200, replace = TRUE)
)
model <- 'F1 =~ x1 + x2 + x3'
fit <- safe_cfa(model, data = mydata, items = c("x1", "x2", "x3"))
if (!is.null(fit)) summary(fit)
```

`safe_measures` *Safe Fit Measures Extraction*

Description

Extracts selected fit measures from a lavaan object. Returns a one-row data frame with `NA` values for any measures that could not be retrieved (e.g., when the model is `NULL`).

Usage

```
safe_measures(
  fit,
 wanted = c("chisq.scaled", "df.scaled", "cfi.scaled", "tli.scaled", "rmsea.scaled",
    "srmr", "wrmr")
)
```

Arguments

`fit`	A fitted lavaan object or `NULL`.
`wanted`	Character vector of fit measure names to extract (default includes chi-square, df, CFI, TLI, RMSEA, SRMR, and WRMR, all scaled).

Value

A one-row data frame with the requested fit measures.

Examples

```
set.seed(123)
mydata <- data.frame(
  x1 = sample(1:5, 200, replace = TRUE),
  x2 = sample(1:5, 200, replace = TRUE),
  x3 = sample(1:5, 200, replace = TRUE)
)
model <- 'F1 =~ x1 + x2 + x3'
fit <- safe_cfa(model, data = mydata, items = c("x1", "x2", "x3"))
safe_measures(fit)
```

save_to_excel_table *Save Tables to Excel File*

Description

Saves multiple data frames to an Excel file with separate sheets.

Usage

```
save_to_excel_table(file_name, ..., na.string = "")
```

Arguments

file_name	Path to the output Excel file.
...	Named data frames to save as sheets.
na.string	String to use for NA values (default: "").

Value

Invisibly returns NULL. Writes Excel file to disk.

smote_multiclass *SMOTE Oversampling for Multiclass Data*

Description

Applies SMOTE (Synthetic Minority Over-sampling Technique) for multiclass imbalanced data.

Usage

```
smote_multiclass(data, outcome, perc_maj = 100, k = 5, seed = NULL)
```

Arguments

data	Data frame with features and outcome variable.
outcome	Name or index of the outcome variable.
perc_maj	Percentage of majority class size to target (default: 100).
k	Number of nearest neighbors for SMOTE (default: 5).
seed	Random seed for reproducibility.

Value

A data frame with oversampled minority classes.

Examples

```
# Create imbalanced multiclass data
set.seed(123)
data <- data.frame(
  Item1 = sample(1:5, 200, replace = TRUE),
  Item2 = sample(1:5, 200, replace = TRUE),
  Item3 = sample(1:5, 200, replace = TRUE),
  Group = c(rep("A", 100), rep("B", 70), rep("C", 30))  # Imbalanced classes
)

# Check original class distribution
table(data$Group)
# A   B   C
# 100 70  30

# Apply SMOTE to balance classes
balanced_data <- smote_multiclass(
  data = data,
  outcome = "Group",
  perc_maj = 100,  # Target 100% of majority class size
  k = 5,
  seed = 123
)

# Check balanced class distribution
table(balanced_data$Group)

# Partial balancing (50% of majority)
partial_balanced <- smote_multiclass(
  data = data,
  outcome = "Group",
  perc_maj = 50,
  seed = 456
)
table(partial_balanced$Group)
```

---

specification_models *Fit Multiple Lavaan Model Specifications*

Description

Fits multiple lavaan EFA/CFA models and returns fitted model objects.

Usage

```
specification_models(
  modelos,
  data,
```

```
  estimator,
  rotation = "oblimin",
  ordered = TRUE,
  verbose = FALSE
)
```

Arguments

modelos	List of model syntax strings.
data	Data frame with item responses.
estimator	Estimation method (e.g., "WLSMV").
rotation	Rotation method for EFA (default: "oblimin").
ordered	Logical, whether variables are ordered (default: TRUE).
verbose	Logical, whether to print model output (default: FALSE).

Value

A list of fitted lavaan model objects.

Examples

```
# Create sample data
set.seed(123)
n <- 300
g <- rnorm(n)
t1 <- 0.7 * g + rnorm(n, 0, 0.7)
t2 <- 0.7 * g + rnorm(n, 0, 0.7)
sim_item <- function(t) {
  as.numeric(cut(t + rnorm(length(t), 0, 0.8), c(-Inf, -1, 0, 1, Inf)))
}
data <- data.frame(
  Item1 = sim_item(t1), Item2 = sim_item(t1), Item3 = sim_item(t1),
  Item4 = sim_item(t2), Item5 = sim_item(t2), Item6 = sim_item(t2)
)

# Generate EFA models for 1 to 3 factors
models <- generate_modelos(
  n_factors = 2,
  name_items = "Item",
  n_items = 6
)

# Fit all models
specifications <- specification_models(
  modelos = models,
  data = data,
  estimator = "WLSMV",
  rotation = "oblimin",
  ordered = TRUE
)
```

```
# Access fitted model for 2-factor solution
summary(specifications[[2]])

# Extract fit indices
fit_table <- extract_fit_measures(specifications)
print(fit_table)
```

`split_data_stratified_clustered` *Split Data with Stratified Clustering*

Description

Splits a data frame using stratified and clustered sampling.

Usage

```
split_data_stratified_clustered(
  df,
  strat_var = "Region_reside",
  cluster_var = "Facultad",
  perc_exploratorio = 0.5,
  seed = NULL
)
```

Arguments

`df`	Data frame to split.
`strat_var`	Stratification variable name (default "Region_reside").
`cluster_var`	Cluster variable name (default "Facultad").
`perc_exploratorio`	Proportion for exploratory subset (default 0.5).
`seed`	Random seed for reproducibility (default NULL).

Value

A list with exploratorio and confirmatorio data frames.

Examples

```
# Create sample data with stratification and clustering variables
set.seed(123)
data <- data.frame(
  ID = 1:500,
  Item1 = sample(1:5, 500, replace = TRUE),
```

```
  Item2 = sample(1:5, 500, replace = TRUE),
  Item3 = sample(1:5, 500, replace = TRUE),
  Region_reside = sample(c("Norte", "Sur", "Centro"), 500, replace = TRUE),
  Facultad = sample(c("Ingenieria", "Medicina", "Derecho"), 500, replace = TRUE)
)

# Split with stratification by Region and clustering by Facultad
splits <- split_data_stratified_clustered(
  df = data,
  strat_var = "Region_reside",
  cluster_var = "Facultad",
  perc_exploratorio = 0.5,
  seed = 123
)

# Access subsets
efa_data <- splits$exploratorio
cfa_data <- splits$confirmatorio

# Verify stratification is maintained
table(efa_data$Region_reside, efa_data$Facultad)
table(cfa_data$Region_reside, cfa_data$Facultad)

# Different proportions
splits2 <- split_data_stratified_clustered(
  df = data,
  strat_var = "Region_reside",
  cluster_var = "Facultad",
  perc_exploratorio = 0.6,
  seed = 456
)
```

`split_data_three` *Split Data into Three Subsets*

Description

Splits a data frame into pilot, exploratory and confirmatory subsets.

Usage

```
split_data_three(
  df,
  perc_piloto = 0.1,
  perc_exploratorio = 0.45,
  perc_confirmatorio = 0.45,
  seed = NULL
)
```

Arguments

df	Data frame to split.
perc_piloto	Proportion for pilot subset (default 0.10).
perc_exploratorio	Proportion for exploratory subset (default 0.45).
perc_confirmatorio	Proportion for confirmatory subset (default 0.45).
seed	Random seed for reproducibility (default NULL).

Value

A list with piloto, exploratorio and confirmatorio data frames.

Examples

```
# Create sample data
set.seed(123)
data <- data.frame(
  Item1 = sample(1:5, 1000, replace = TRUE),
  Item2 = sample(1:5, 1000, replace = TRUE),
  Item3 = sample(1:5, 1000, replace = TRUE),
  Item4 = sample(1:5, 1000, replace = TRUE)
)

# Split data into 10% pilot, 45% EFA, 45% CFA
splits <- split_data_three(
  df = data,
  perc_piloto = 0.10,
  perc_exploratorio = 0.45,
  perc_confirmatorio = 0.45,
  seed = 123
)

# Access the subsets
pilot_data <- splits$piloto
efa_data <- splits$exploratorio
cfa_data <- splits$confirmatorio

nrow(pilot_data)  # Should be 100
nrow(efa_data)    # Should be 450
nrow(cfa_data)    # Should be 450

# Custom proportions: 20% pilot, 40% EFA, 40% CFA
splits2 <- split_data_three(
  df = data,
  perc_piloto = 0.20,
  perc_exploratorio = 0.40,
  perc_confirmatorio = 0.40,
  seed = 456
)
```

split_data_two *Split Data into Two Subsets*

Description

Splits a data frame into exploratory and confirmatory subsets.

Usage

```
split_data_two(
  df,
  perc_exploratorio = 0.5,
  perc_confirmatorio = 0.5,
  seed = NULL
)
```

Arguments

df	Data frame to split.
perc_exploratorio	Proportion for exploratory subset (default 0.5).
perc_confirmatorio	Proportion for confirmatory subset (default 0.5).
seed	Random seed for reproducibility (default NULL).

Value

A list with exploratorio and confirmatorio data frames.

Examples

```
# Create sample data
set.seed(123)
data <- data.frame(
  Item1 = sample(1:5, 500, replace = TRUE),
  Item2 = sample(1:5, 500, replace = TRUE),
  Item3 = sample(1:5, 500, replace = TRUE),
  Item4 = sample(1:5, 500, replace = TRUE)
)

# Split data 50/50 for EFA and CFA
splits <- split_data_two(
  df = data,
  perc_exploratorio = 0.5,
  perc_confirmatorio = 0.5,
  seed = 123
)
```

```
# Access the subsets
efa_data <- splits$exploratorio
cfa_data <- splits$confirmatorio

nrow(efa_data)  # Should be 250
nrow(cfa_data)  # Should be 250

# Different proportions: 60% EFA, 40% CFA
splits2 <- split_data_two(
  df = data,
  perc_exploratorio = 0.6,
  perc_confirmatorio = 0.4,
  seed = 456
)
```

`Standardized_solutions` *Standardized Factor Solutions for EFA*

Description

Extracts and formats standardized factor loadings from lavaan EFA.

Usage

```
Standardized_solutions(specification, name_items, apply_threshold = TRUE)
```

Arguments

`specification`	A lavaan model object.
`name_items`	Prefix for item names.
`apply_threshold`	Logical, whether to apply 0.30 threshold (default TRUE).

Value

A data frame with items and factor loadings.

Examples

```
# First run EFA_modern to get the specification object
set.seed(123)
n <- 300
data_efa <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
```

```
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE),
  Item7 = sample(1:5, n, replace = TRUE),
  Item8 = sample(1:5, n, replace = TRUE),
  Item9 = sample(1:5, n, replace = TRUE)
)

# Run EFA
efa_result <- EFA_modern(
  n_factors = 3,
  n_items = 9,
  name_items = "Item",
  data = data_efa,
  apply_threshold = TRUE
)

# Extract standardized solutions from the 3-factor model
loadings <- Standardized_solutions(
  specification = efa_result$Specifications[[3]],
  name_items = "Item",
  apply_threshold = TRUE
)
print(loadings)

# Without threshold (show all loadings)
loadings_full <- Standardized_solutions(
  specification = efa_result$Specifications[[3]],
  name_items = "Item",
  apply_threshold = FALSE
)
print(loadings_full)
```

`Standardized_solutions_cfa` *Standardized Factor Solutions for CFA*

Description

Extracts and formats standardized factor loadings from lavaan CFA.

Usage

```
Standardized_solutions_cfa(specification, name_items, apply_threshold = TRUE)
```

Arguments

`specification`	A lavaan model object.
`name_items`	Prefix for item names.
`apply_threshold`	Logical, whether to apply 0.30 threshold (default TRUE).

Value

A data frame with items and factor loadings.

Examples

```
library(lavaan)

# Create sample data
set.seed(123)
n <- 300
data_cfa <- data.frame(
  Item1 = sample(1:5, n, replace = TRUE),
  Item2 = sample(1:5, n, replace = TRUE),
  Item3 = sample(1:5, n, replace = TRUE),
  Item4 = sample(1:5, n, replace = TRUE),
  Item5 = sample(1:5, n, replace = TRUE),
  Item6 = sample(1:5, n, replace = TRUE)
)

# Define CFA model
model <- "
  f1 =~ Item1 + Item2 + Item3
  f2 =~ Item4 + Item5 + Item6
"

# Fit the model
fit <- cfa(model, data = data_cfa, ordered = TRUE, estimator = "WLSMV")

# Extract standardized solutions with threshold
loadings <- Standardized_solutions_cfa(
  specification = fit,
  name_items = "Item",
  apply_threshold = TRUE
)
print(loadings)

# Without threshold (show all loadings)
loadings_full <- Standardized_solutions_cfa(
  specification = fit,
  name_items = "Item",
  apply_threshold = FALSE
)
print(loadings_full)
```

---

summarise_column *Summarize Column Statistics*

Description

Calculates descriptive statistics for a column, excluding zero values.

Usage

```
summarise_column(data, col_name)
```

Arguments

data	Data frame containing the column.
col_name	Name of the column to summarize.

Value

A data frame with mean, SD, min, and max values.

Index