Package ‘tidySEM’

April 14, 2022

Type Package

Date 2022-4-14

Title Tidy Structural Equation Modeling

Version 0.2.3

Description A tidy workflow for generating, estimating, reporting,
and plotting structural equation models using 'lavaan', 'OpenMx', or
'Mplus'. Throughout this workflow, elements of syntax, results, and graphs
are represented as 'tidy' data, making them easy to customize.

License GPL (>= 3)

URL `https://cjvanlissa.github.io/tidySEM/`

BugReports `https://github.com/cjvanlissa/tidySEM/issues`

Depends R (>= 4.0.0), stats, utils, OpenMx

Imports ggplot2, lavaan, blavaan, MplusAutomation, igraph, psych,
methods, gtable

Suggests testthat, knitr, rmarkdown, dplyr, stringr, covr, tidyLPA,
poLCA, umx, mclust

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.2

LazyData true

NeedsCompilation no

Author Caspar J. van Lissa [aut, cre]
(<`https://orcid.org/0000-0002-0808-5024`>)

Maintainer Caspar J. van Lissa <`c.j.vanlissa@uu.nl`>

Repository CRAN

Date/Publication 2022-04-14 17:50:02 UTC

R topics documented:

`add_paths` *Add paths to an object of class 'tidy_sem'*

Description

Add paths to an object of class `tidy_sem`, or replace existing paths. The paths must be specified as `model.syntax`, and separated by commas.

Usage

```
add_paths(model, ...)
```

Arguments

`model`	An object of class `tidy_sem`.
`...`	Paths to add or substitute, specified in `lavaan{model.syntax}`, and separated by commas.

Details

Currently, only the `lavaan{lavaan}` commands `~`,`~~`,`=~`, and `~1` are parsed.

This function relies on `lavaan` `model.syntax` to convert syntax strings to `lavaan` parameter tables. By default, is uses the arguments `int.ov.free = TRUE,int.lv.free = FALSE,auto.fix.first = TRUE,auto.fix.single = TRUE,auto.var = TRUE,auto.cov.lv.x = TRUE,auto.efa = TRUE,auto.th = TRUE,auto.delta = TRUE,auto.cov.y = TRUE,meanstructure = TRUE`, in a similar way to `sem` and `cfa`.

Value

An object of class `tidy_sem`.

See Also

`model.syntax`

Examples

```
library(lavaan)
df <- iris[, 1:4]
names(df) <- paste0("x_", 1:4)
model <- tidy_sem(df)
model <- measurement(model)
model <- add_paths(model, x =~ a*x_1 + b*x_2 + a*x_3 + b*x_4)
res <- estimate_lavaan(model)
summary(res)
```

---

as_lavaan *Convert tidy_sem to 'lavaan' syntax*

Description

Final stage in the 'tidySEM' workflow for syntax generation: Convert the `tidy_sem` object to `lavaan` syntax in tabular format (see `model.syntax`).

Usage

```
as_lavaan(x, ...)
```

Arguments

`x`	An object of class `tidy_sem`
`...`	Additional parameters to be passed to and from functions.

Value

Character vector.

Examples

```
mod <- list(syntax = structure(list(lhs = "x", op = "~", rhs = "y",
                                    free = TRUE, value = "", label = "",
                                    category = "", aspect = ""),
            class = "data.frame", row.names = c(NA, -1L)))
class(mod) <- "tidy_sem"
as_lavaan(mod)
```

as_mplus — *Convert tidy_sem to 'Mplus' syntax*

Description

Final stage in the 'tidySEM' workflow for syntax generation: Convert the `tidy_sem` object to 'Mplus' syntax.

Usage

```
as_mplus(x, ...)
```

Arguments

`x`	An object of class `tidy_sem`.
`...`	Additional parameters to be passed to and from functions.

Value

Character vector.

Examples

```
mod <- list(syntax = structure(list(lhs = "x", op = "~", rhs = "y",
                                    free = TRUE, value = "", label = "",
                                    category = "", aspect = ""),
            class = "data.frame", row.names = c(NA, -1L)))
class(mod) <- "tidy_sem"
as_mplus(mod)
```

as_ram — *Convert lavaan syntax to RAM specification*

Description

Converts SEM models to RAM models for `OpenMx`.

Usage

```
as_ram(x, ...)
```

Arguments

`x`	An object for which a method exists, such as a `tidy_sem` object, or character vector describing the user-specified model using the lavaan model syntax.
`...`	Parameters passed on to other functions.

Details

For models specified using lavaan syntax, the procedure is as follows:

1. Apply `lavaanify` to the `model`. The default arguments to `lavaanify` correspond to those of the `sem` function.
2. Convert each row of the resulting lavaan parameter table to a `mxPath`.
3. Apply `mxModel` to the `mxPaths` to create an `OpenMx` model using RAM specification

Value

Returns an `mxModel`.

Examples

```
as_ram("y ~ x")
```

BCH — *Estimate an Auxiliary Model using the BCH Method*

Description

Estimate an auxiliary model based on a latent classification by means of mixture modeling (see `mx_mixture`).

The auxiliary model is treated as a multi-group model. All cases are used in all groups, but they are weighted by group-specific BCH weights as described in Bolck, Croon, & Hagenaars, 2004.

Usage

```
BCH(x, model, data, ...)
```

Arguments

`x`	An object for which a method exists.
`model`	An object that can be converted to an `OpenMx` model using `as_ram`.
`data`	A data.frame on which the auxiliary model can be evaluated.
`...`	further arguments to be passed to or from other methods.

Value

An MxModel.

References

Bolck, A., Croon, M., & Hagenaars, J. (2004). Estimating latent structure models with categorical variables: One-step versus three-step estimators. Political Analysis, 12(1), 3–27. <doi:10.2307/25791751>

Examples

```
dat <- data.frame(x = iris$Petal.Length)
mixmod <- mx_profiles(dat,
                      classes = 2)
res <- BCH(mixmod, "y ~ 1", data = data.frame(y = iris$Sepal.Length))
```

`BLRT` *Conduct Bootstrapped Likelihood Ratio Test*

Description

Conduct Bootstrapped Likelihood Ratio Test to compare two mixture models.

Usage

```
BLRT(x, ...)
```

Arguments

`x`	An object for which a method exists.
`...`	further arguments to be passed to or from other methods.

Value

A data.frame.

Examples

```
## Not run:
df <- iris[, 1, drop = FALSE]
names(df) <- "x"
res <- mx_mixture(model = "x ~ m{C}*1
                           x ~~ v{C}*x", classes = 1:2, data = df)
BLRT(res, replications = 4)

## End(Not run)
```

`class_prob` *Obtain latent class probabilities*

Description

Obtain latent class probabilities for an object for which a method exists. See Details.

Usage

```
class_prob(
  x,
  type = c("sum.posterior", "sum.mostlikely", "mostlikely.class", "avg.mostlikely",
    "individual"),
  ...
)
```

Arguments

`x`	An object for which a method exists.
`type`	Character vector, indicating which types of probabilities to extract. See Details.
`...`	Further arguments to be passed to or from other methods.

Details

The following types are available:

- "sum.posterior"A summary table of the posterior class probabilities; this indicates what proportion of your data contributes to each class.
- "sum.mostlikely"A summary table of the most likely class membership, based on the highest posterior class probability. Note that this is subject to measurement error.
- "mostlikely.class"If C is the true class of an observation, and N is the most likely class based on the model, then this table shows the probability P(N==i|C==j). The diagonal represents the probability that observations in each class will be correctly classified.
- "avg.mostlikely"Average posterior probabilities for each class, for the subset of observations with most likely class of 1:k, where k is the number of classes.
- "individual"The posterior probability matrix, with dimensions n (number of cases in the data) x k (number of classes).

Value

A data.frame.

Examples

```
## Not run:
df <- iris[, 1, drop = FALSE]
names(df) <- "x"
res <- mx_mixture(model = "x ~ m{C}*1
                           x ~~ v{C}*x", classes = 1, data = df)
class_prob(res)

## End(Not run)
```

---

conf_int *Format confidence intervals*

Description

Creates 'APA'-formatted confidence intervals, either from an object for which a method exists, or from the arguments `lb` and `ub`. When argument `x` is a numeric vector, it is also possible to construct a confidence interval using the standard error (`se`) and a percentile interval (`ci`).

Usage

```
conf_int(x, digits = 2, se = NULL, lb = NULL, ub = NULL, ci = 95)
```

Arguments

`x`	Optional. An object for which a method exists.
`digits`	Integer. The number of digits to round the confidence boundaries to.
`se`	Optional, numeric. Standard error of the parameters.
`lb`	Optional, numeric. Lower boundary of confidence intervals.
`ub`	Optional, numeric. Upper boundary of confidence intervals.
`ci`	Optional, numeric. What percentage CI to use (only used when computing CI from a numeric vector `x`, and the standard error `se`, based on a normal distribution).

Value

A character vector of formatted confidence intervals.

Author(s)

Caspar J. van Lissa

See Also

table_results est_sig

Other Reporting tools: `est_sig()`, `table_fit()`, `table_prob()`, `table_results()`

Examples

```
conf_int(x = c(1.325, 2.432), se = c(.05336, .00325))
```

---

cors *Generate syntax for correlations*

Description

Generate syntax for correlations between variables.

Usage

```
cors(x, ...)
```

Arguments

x	Object for which a method exists. If x is an object of class tidy_sem, then correlations between all observed and latent variables in the data dictionary of that object are computed, by default. If x is a character vector, all elements of the vector are used.
...	Optional additional character vectors of variables to be correlated. If x is an object of class tidy_sem, then up to two vectors can be provided. If x is a vector, then one more optional vector can be provided. When no additional vectors of variable names are provided, only the correlations among the elements of x are returned.

Value

An object of class tidy_sem.

Examples

```
dict <- tidy_sem(c("bfi_1", "bfi_2", "bfi_3", "bfi_4", "bfi_5"))
cors(dict, c("bfi_1", "bfi_2"))
```

`create_scales` *Create scale scores from observed variables*

Description

This function calculates mean or sum scores from a `data.frame` and a named list describing the items in each scale. It returns the scores, a scale descriptive table, and a scale correlation table. It relies on several functions from the `psych` package.

Usage

```
create_scales(
  x,
  keys.list,
  missing = TRUE,
  impute = "none",
  omega = NULL,
  digits = 2,
  ...
)

## S3 method for class 'tidy_sem'
create_scales(
  x,
  keys.list,
  missing = TRUE,
  impute = "none",
  omega = NULL,
  digits = 2,
  ...
)
```

Arguments

`x`	A `data.frame` containing all variables referenced in the `keys.list`, or an object of class `tidy_sem`.
`keys.list`	A named list, indicating which variables belong to which scale.
`missing`	Whether to use rows with partially missing values. Default: TRUE.
`impute`	Method for handling missing values, Default: 'none'. This default method uses all available data to calculate scale scores, which is acceptable for mean scales, but not for sum scales.
`omega`	Which of McDonald's `omega` coefficients to report. Default: NULL; valid options include: `"omega_h"`, `"omega.lim"`, `"alpha"`, `"omega.tot"`, `"G6"`.
`digits`	Number of digits for rounding, Default: 2
`...`	Additional parameters to pass to and from functions.

Details

For scales with less than 3 items, Cronbach's alpha might not be suitable as an estimate of reliability. For such scales, the Spearman-Brown reliability coefficient for two-item scales is computed, as described in Eisinga, R., Grotenhuis, M. te, & Pelzer, B. (2012). The reliability of a two-item scale: Pearson, Cronbach, or Spearman-Brown? International Journal of Public Health, 58(4), 637–642. <doi:10.1007/s00038-012-0416-3>. These coefficients are marked with "(sb)".

Value

List with elements: `$descriptives`, `$correlations`, and `$scores`.

Examples

```
out <- create_scales(iris, keys.list = list(scalename =
            c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")))
out$descriptives
dict <- tidy_sem(iris, split = "\\.")
create_scales(dict)
```

`curry_mac` *Simulated MAC data*

Description

This simulated dataset, based on Curry et al., 2019, contains data on moral relevance and judgment across the seven domains of the Morality As Cooperation scale.

Usage

```
data(curry_mac)
```

Format

A data.frame with 1392 rows and 42 variables.

Details

sex	`factor`	Self-identified sex of participants, Male, Female, or Transgendered.
age_years	`numeric`	Participants' age in years.
KinshipR	`numeric`	Mean score of moral relevance, kinship subscale.
MutualismR	`numeric`	Mean score of moral relevance, mutualism subscale.
ExchangeR	`numeric`	Mean score of moral relevance, exchange subscale.
HawkR	`numeric`	Mean score of moral relevance, hawk subscale.
DoveR	`numeric`	Mean score of moral relevance, dove subscale.
DivisionR	`numeric`	Mean score of moral relevance, division subscale.
PossessionR	`numeric`	Mean score of moral relevance, possession subscale.
KinshipJ	`numeric`	Mean score of moral judgment, kinship subscale.

MutualismJ	`numeric`	Mean score of moral judgment, mutualism subscale.
ExchangeJ	`numeric`	Mean score of moral judgment, exchange subscale.
HawkJ	`numeric`	Mean score of moral judgment, hawk subscale.
DoveJ	`numeric`	Mean score of moral judgment, dove subscale.
DivisionJ	`numeric`	Mean score of moral judgment, division subscale.
PossessionJ	`numeric`	Mean score of moral judgment, possession subscale.

References

Curry, O. S., Jones Chesters, M., & Van Lissa, C. J. (2019). Mapping morality with a compass: Testing the theory of 'morality-as-cooperation' with a new questionnaire. Journal of Research in Personality, 78, 106–124. doi: 10.1016/j.jrp.2018.10.008

`data_mix_ordinal` *Simulated data for mixture model with ordinal indicators*

Description

This simulated dataset, based on the 'Mplus' User's Guide example 7.6, contains four columns of integer data that should be treated as ordinal.

Usage

```
data(data_mix_ordinal)
```

Format

A data frame with 5000 rows and 4 variables.

Details

u1	`integer`	Indicator 1, should be treated as ordinal.
u2	`integer`	Indicator 2, should be treated as ordinal.
u3	`integer`	Indicator 3, should be treated as ordinal.
u4	`integer`	Indicator 4, should be treated as ordinal.

References

Muthén, L.K. and Muthén, B.O. (1998-2017). Mplus User's Guide. Eighth Edition. Los Angeles, CA: Muthén & Muthén

`descriptives` *Describe a dataset*

Description

Provide descriptive statistics for a dataset.

Usage

```
descriptives(x, ...)
```

Arguments

`x`	An object for which a method exists.
`...`	Additional arguments.

Value

A `data.frame` with descriptive statistics for `x`.

Examples

```
descriptives(iris)
```

`dictionary` *Extract dictionary from tidy_sem*

Description

Provides access to the `dictionary` element of a `tidy_sem` object. This can be used to return or assign to the `dictionary` element.

Usage

```
dictionary(x)

dictionary(x) <- value
```

Arguments

`x`	Object of class tidy_sem.
`value`	A valid value for `dictionary(x)`.

Value

data.frame

Examples

```
dict <- tidy_sem(iris, split = "\\.")
dictionary(dict)
```

edges *Extract edges from sem_graph*

Description

Provides access to the edges element of a sem_graph object. This can be used to return or assign to the edges element.

Usage

```
edges(x)

edges(x) <- value
```

Arguments

x	Object of class sem_graph.
value	A valid value for `edges(x)`.

Value

data.frame

Examples

```
edg <- data.frame(from = "x", to = "y")
p <- prepare_graph(edges = edg, layout = get_layout("x", "y", rows = 1))
edges(p)
```

edit_graph *Edit graph elements*

Description

Evaluate an R expression within the environment of the elements of a sem_graph object, and return the modified sem_graph.

Usage

```
edit_graph(x, expr, element = c("edges", "nodes"), ...)

edit_nodes(x, expr, ...)

edit_edges(x, expr, ...)
```

Arguments

`x`	An object of class `sem_graph`.
`expr`	expression to evaluate.
`element`	Character. The element of the `sem_graph` to edit, defaults to `c("edges","nodes")`.
`...`	Arguments passed on to `within`.

Value

An object of class `sem_graph`.

Examples

```
p <- prepare_graph(layout = get_layout("x", rows = 1))
p <- edit_graph(p, {colour = "blue"}, element = "nodes")
plot(p)
```

`empathy` *Simulated empathy data*

Description

This simulated dataset, based on Van Lissa et al., 2014, contains six annual assessments of adolescents' mean scores on the empathic concern and perspective taking subscales of the Interpersonal Reactivity Index (Davis, 1983). The first measurement wave occurred when adolescents were, on average, 13 years old, and the last one when they were 18 years old.

Usage

```
data(empathy)
```

Format

A data frame with 467 rows and 13 variables.

Details

ec1	`numeric`	Mean score of empathic concern in wave 1
ec2	`numeric`	Mean score of empathic concern in wave 2
ec3	`numeric`	Mean score of empathic concern in wave 3
ec4	`numeric`	Mean score of empathic concern in wave 4
ec5	`numeric`	Mean score of empathic concern in wave 5
ec6	`numeric`	Mean score of empathic concern in wave 6
pt1	`numeric`	Mean score of perspective taking in wave 1
pt2	`numeric`	Mean score of perspective taking in wave 2
pt3	`numeric`	Mean score of perspective taking in wave 3
pt4	`numeric`	Mean score of perspective taking in wave 4

pt5	`numeric`	Mean score of perspective taking in wave 5
pt6	`numeric`	Mean score of perspective taking in wave 6
sex	`factor`	Adolescent sex; M = male, F = female.

References

Van Lissa, C. J., Hawk, S. T., Branje, S. J., Koot, H. M., Van Lier, P. A., & Meeus, W. H. (2014). Divergence Between Adolescent and Parental Perceptions of Conflict in Relationship to Adolescent Empathy Development. Journal of Youth and Adolescence, (Journal Article), 1–14. doi: 10.1007/s1096401401525

`estimate_lavaan` *Estimate tidy_sem using 'lavaan'*

Description

This function is a wrapper for the `lavaan` estimating functions. By default, the wrapper uses `sem`, but users can also specify `lavaan`, `cfa`, or `growth`.

Usage

```
estimate_lavaan(x, func = "sem", ...)
```

Arguments

`x`	An object of class `tidy_sem`.
`func`	The `lavaan` modeling function to invoke, Default: 'sem'.
`...`	Additional parameters passed to the estimating function.

Value

An object of class `lavaan`.

Examples

```
library(lavaan)
model <- tidy_sem(iris, "\\.")
model <- measurement(model)
res <- estimate_lavaan(model)
summary(res)
```

estimate_mplus *Estimate tidy_sem using 'Mplus'*

Description

This function is a wrapper for the functions mplusObject and mplusModeler. Using this function requires 'Mplus' to be installed.

Usage

```
estimate_mplus(x, ...)
```

Arguments

x	An object of class tidy_sem.
...	Additional parameters passed to mplusObject and mplusModeler. These arguments are matched to the correct function by name. The arguments rdata, and MODEL cannot be edited, as they are determined from the tidy_sem object.

Details

The arguments dataout, modelout, and run are optional. If these are not specified, the model will be run in tempdir.

Value

An object of class mplusObject.

Examples

```
library(MplusAutomation)
model <- tidy_sem(iris, "\\.")
model <- measurement(model)
if(mplusAvailable() == 0){
  estimate_mplus(model)
}
```

estimate_mx *Estimate tidy_sem using 'OpenMx'*

Description

This function is a wrapper for the as_ram and run_mx functions.

Usage

```
estimate_mx(x, ...)
```

Arguments

`x`	An object of class `tidy_sem`.
`...`	Additional parameters passed to the estimating function.

Value

An object of class `MxModel`.

Examples

```
df <- iris[1:4]
names(df) <- paste0("x_", 1:4)
model <- tidy_sem(df)
model <- measurement(model)
res <- estimate_mx(model)
summary(res)
```

`est_sig` *Add significance asterisks to object*

Description

Takes an object, and adds significance asterisks.

Usage

```
est_sig(x, digits = 2, sig = NULL)
```

Arguments

`x`	An object for which a method exists. This will be treated as numeric by the default method.
`digits`	Integer. The number of digits to round the estimate column to.
`sig`	Optional, a vector of p-values for the default method.

Value

A character vector of formatted estimates.

Author(s)

Caspar J. van Lissa

See Also

table_results

Other Reporting tools: `conf_int()`, `table_fit()`, `table_prob()`, `table_results()`

Examples

```
est_sig(c(.222, .3333), sig = c(.054, .045))
```

`get_data` *Extract data from tidy_sem*

Description

Provides access to the `data` element of a `tidy_sem` object. This can be used to return or assign to the `data` element.

Usage

```
get_data(x)

get_data(x) <- value
```

Arguments

`x`	Object of class tidy_sem.
`value`	A valid value for `get_data(x)`.

Value

data.frame

Examples

```
dict <- tidy_sem(iris, split = "\\.")
get_data(dict)
```

`get_edges` *Extract edges from a SEM model object*

Description

Attempts to extract edges from a SEM model object, where edges are defined as regression paths and covariances between variables (nodes).

Usage

```
get_edges(x, label = "est_sig", ...)
```

Arguments

`x`	A model object of class `mplusObject` or `lavaan`.
`label`	Either a character, indicating which column to use for edge labels, or an expression. See Details. Defaults to `"est_sig"`, which labels edges with the estimated value with significance asterisks, as obtained from `table_results`. See Details and examples for more information.
`...`	Additional parameters passed to `table_results`. For example, users can pass the `digits` argument to control the number of digits in the edge label, or pass the `columns` argument to retain auxiliary columns in the `tidy_edges` `data.frame` for further processing (see Examples).

Details

The function `get_edges` identifies all regression paths, latent variable definitions, and covariances in the model as edges. The output of `table_results` for those paths is used to label the edges.

## Custom labels

One way to create custom edge labels is by passing an expression to `label`. When an expression is passed to `label`, it is evaluated in the context of a `data.frame` containing the results of a call to `table_results` on the `x` argument.

Another way to create custom labels is by requesting auxiliary variables using the `columns` argument (which is passed to `table_results`), and then using these columns to construct a new label. See examples.

Value

An object of class 'tidy_edges'

Examples

```
# Standard use
library(lavaan)
res <- sem("dist ~ speed", cars)
get_edges(res)

# Pass an expression to the 'label' argument for custom labels
get_edges(res, label = paste(est_sig, confint))

# Pass the argument 'columns' to table_results through '...' to retain
# auxiliary columns for further processing
edg <- get_edges(res, columns = c("est_sig", "confint"))
edg
edg <- within(edg, {label <- paste(est_sig, confint)})
edg
```

get_fit *Get fit indices from objects*

Description

Get fit indices from objects for which a method exists.

Usage

```
get_fit(x, ...)
```

Arguments

x	An object for which a method exists.
...	further arguments to be passed to or from other methods.

Value

A data.frame.

Examples

```
## Not run:
df <- iris[, 1, drop = FALSE]
names(df) <- "x"
res <- mx_mixture(model = "x ~ m{C}*1
                           x ~~ v{C}*x", classes = 1, data = df)
table_fit(res)

## End(Not run)
```

get_layout.lavaan *Generate graph layout*

Description

Generate a tidy_layout for a SEM graph.

Usage

```
## S3 method for class 'lavaan'
get_layout(x, ..., layout_algorithm = "layout_as_tree")

get_layout(x, ...)

## Default S3 method:
get_layout(x, ..., rows = NULL)
```

Arguments

`x`	An object for which a method exists; currently, methods exist for `character`, `lavaan`, and `mplus.model` objects.
`...`	Character arguments corresponding to layout elements. Use node names, empty strings (""), or NA values.
`layout_algorithm`	Optional argument for fit model objects. Character string, indicating which `igraph` layout algorithm to apply to position the nodes. Defaults to `"layout_as_tree"`; see details for more options.
`rows`	Numeric, indicating the number of rows of the graph.

Details

There are three ways to generate a layout:

1. Specify the layout in the call to `get_layout()` by providing node names and the number of rows to create a layout matrix. Empty strings (`""`) or `NA` can be used for empty cells. See Example 1.
2. Call `get_layout()` on a model object or `tidy_results` object. It will use the function `layout_as_tree`, or any other layout function from the `igraph` package, to generate a rudimentary layout. See Example 2.
3. Instead of using `get_layout()`, just use a `matrix` or `data.frame` with your layout. For example, specify the layout in a spreadsheet program, and load it into R (see Example 3). Or, copy the layout to the clipboard from your spreadsheet program, and load it from the clipboard (see Example 4)

The layout algorithms imported from `igraph` are: `c("layout_as_star","layout_as_tree","layout_in_circle","layo`
These can be used by specifying the optional argument `layout_algorithm = ""`.

Value

Object of class 'tidy_layout'

Examples

```
# Example 1
get_layout("c", NA,  "d",
           NA,  "e", NA, rows = 2)

# Example 2
library(lavaan)
fit <- cfa(' visual  =~ x1 + x2 + x3 ',
           data = HolzingerSwineford1939[1:50, ])
get_layout(fit)

## Not run:
# Example 3
# Here, we first write the layout to .csv, but you could create it in a
# spreadsheet program, and save the spreadsheet to .csv:
```

```
write.csv(matrix(c("c", "",  "d", "",  "e", ""), nrow = 2, byrow = TRUE),
          file = file.path(tempdir(), "example3.csv"), row.names = FALSE)
# Now, we load the .csv:
read.csv(file.path(tempdir(), "example3.csv"))

# Example 4
# For this example, make your layout in a spreadsheet program, select it, and
# copy to clipboard. Reading from the clipboard works differently in Windows
# and Mac. For this example, I used Microsoft Excel.
# On Windows, run:
read.table("clipboard", sep = "\t")
# On Mac, run:
read.table(pipe("pbpaste"), sep="\t")

## End(Not run)
```

get_nodes *Extract nodes from a SEM model object*

Description

Attempts to extract nodes from a SEM model object, where nodes are defined as observed or latent variables.

Usage

```
get_nodes(x, label = paste2(name, est_sig, sep = "\n"), ...)
```

Arguments

`x`	A model object of class `mplusObject` or `lavaan`.
`label`	Either a character, indicating which column to use for node labels, or an expression. See Details. Defaults to `paste(name,est_sig,sep = "\n"`, which gives the node name followed by the estimated value with significance asterisks.
`...`	Additional parameters passed to `table_results`. For example, users can pass the `digits` argument to control the number of digits in the node label, or pass the `columns` argument to retain auxiliary columns in the `tidy_nodes` `data.frame` for further processing (see Examples).

Details

The function `get_nodes` identifies all dependent and independent variables in the model as nodes. If a mean structure / intercepts are included in the model, the output of `table_results` for those means / intercepts is used to label the nodes.

## Custom labels

One way to create custom node labels is by passing an expression to `label`, as in the default value of the argument. When an expression is passed to `label`, it is evaluated in the context of a `data.frame` containing the results of a call to table_results on the x argument, with an additional column labeled name, which contains the node names.

Another way to create custom labels is by requesting auxiliary variables using the columns argument (which is passed to table_results), and then using these columns to construct a new label. See examples.

Value

An object of class 'tidy_nodes'

Examples

```
# Standard use extracts node names and shape
# (rect for observed, oval for latent)
library(lavaan)
res <- sem("dist ~ speed", cars)
get_nodes(res)

# To label nodes with mean values, include meanstructure in the model
# Note that it is possible to pass the argument 'digits' to table_results
# through '...'
res <- sem("dist ~ speed", cars, meanstructure = TRUE)
get_nodes(res, digits = 3)

# Pass an expression to the 'label' argument for custom labels
get_nodes(res, label = paste0(name, " ", est_sig, "\n", confint))

# Pass the argument 'columns' to table_results through '...' to retain
# auxiliary columns for further processing
nod <- get_nodes(res, columns = c("est_sig", "confint"))
nod
nod <- within(nod, {label <- paste0(name, " ", est_sig, "\n", confint)})
nod
```

---

graph_sem *Render a graph*

Description

Render a graph based on a layout, and either nodes and edges, or a model object.

Usage

```
graph_sem(...)

## Default S3 method:
graph_sem(
  edges = NULL,
  layout = NULL,
  nodes = NULL,
  rect_width = 1.2,
  rect_height = 0.8,
  ellipses_width = 1,
  ellipses_height = 1,
  variance_diameter = 0.8,
  spacing_x = 2,
  spacing_y = 2,
  text_size = 4,
  curvature = 60,
  angle = NULL,
  fix_coord = FALSE,
  ...
)

## S3 method for class 'lavaan'
graph_sem(model, edges = NULL, layout = NULL, nodes = NULL, ...)

## S3 method for class 'MxModel'
graph_sem(model, edges = NULL, layout = NULL, nodes = NULL, ...)

## S3 method for class 'mplus.model'
graph_sem(model, edges = NULL, layout = NULL, nodes = NULL, ...)

## S3 method for class 'character'
graph_sem(...)

## S3 method for class 'mplusObject'
graph_sem(model, edges = NULL, layout = NULL, nodes = NULL, ...)
```

Arguments

`...`	Additional arguments passed to and from functions.
`edges`	Object of class 'tidy_edges', or a `data.frame` with (at least) the columns `c("from","to")`, and optionally, `c("arrow","label","connect_from","connect_to","curvature")`.
`layout`	A matrix (or data.frame) that describes the layout; see `get_layout`.
`nodes`	Optional, object of class 'tidy_nodes', created with the `get_nodes` function, or a `data.frame` with (at least) the column `c("name")`, and optionally, `c("shape","label")`. If set to `NULL` (the default), nodes are inferred from the `layout` and edges arguments.
`rect_width`	Width of rectangles (used to display observed variables), Default: 1.2
`rect_height`	Height of rectangles (used to display observed variables), Default: 0.8
`ellipses_width`	Width of ellipses (used to display latent variables), Default: 1
`ellipses_height`	Height of ellipses (used to display latent variables), Default: 1
`variance_diameter`	Diameter of variance circles, Default: .8

`spacing_x`	Spacing between columns of the graph, Default: 1
`spacing_y`	Spacing between rows of the graph, Default: 1
`text_size`	Point size of text, Default: 4
`curvature`	Curvature of curved edges. The curve is a circle segment originating in a point that forms a triangle with the two connected points, with angles at the two connected points equal to `curvature`. To flip a curved edge, use a negative value for curvature. Default: 60
`angle`	Angle used to connect nodes by the top and bottom. Defaults to NULL, which means Euclidean distance is used to determine the shortest distance between node sides. A numeric value between 0-180 can be provided, where 0 means that only nodes with the same x-coordinates are connected top-to-bottom, and 180 means that all nodes are connected top-to-bottom.
`fix_coord`	Whether or not to fix the aspect ratio of the graph. Does not work with multi-group or multilevel models. Default: FALSE.
`model`	Instead of the edges argument, it is also possible to use the model argument and pass an object for which a method exists (e.g., `mplus.model` or `lavaan`).

Details

The default interface simply Runs the functions `prepare_graph` and `plot`. The alternative interface first runs `get_nodes` and `get_edges` on the `model` argument.

Value

Object of class 'sem_graph'

Examples

```
library(lavaan)
res <- sem("dist ~ speed", cars)
graph_sem(res)
```

`if_edit` *Conditionally edit a sem_graph object*

Description

This function allows users to conditionally manipulate the edges and nodes of a `sem_graph` object. The generic function `if_edit` applies the expression `expr` to all rows of the `nodes` and `edges` data.frames for which `condition` is `TRUE`.

The wrapper functions documented in the Usage section have a hard-coded `expr` and `condition`; for example, `color_sig(color = "green")` colors all nodes and edges with `pval < .05` green. If no column exists for the assigned aesthetic (e.g., `color`), the wrappers assign the default argument (in this case, `color = "black"`) to all other nodes and edges.

Usage

```
if_edit(data, condition, expr, ...)

if_edges(data, condition, expr, ...)

if_nodes(data, condition, expr, ...)

## S3 method for class 'sem_graph'
if_edit(data, condition, expr, element = c("edges", "nodes"), ...)

all_sig(data, expr, ...)

hide_sig(data, ...)

show_sig(data, ...)

colour_sig(data, colour = "black", ...)

color_sig(data, color = "black", ...)

linetype_sig(data, linetype = 1, ...)

size_sig(data, size = 1, ...)

alpha_sig(data, alpha = 1, ...)

fill_sig(data, fill = "white", ...)

label_colour_sig(data, label_colour = "black", ...)

label_color_sig(data, label_color = "black", ...)

label_fill_sig(data, label_fill = "white", ...)

label_size_sig(data, label_size = 4, ...)

label_alpha_sig(data, label_alpha = 1, ...)

label_family_sig(data, label_family = "sans", ...)

label_fontface_sig(data, label_fontface = "plain", ...)

label_hjust_sig(data, label_hjust = "center", ...)

label_vjust_sig(data, label_vjust = "middle", ...)

label_lineheight_sig(data, label_lineheight = 1, ...)
```

```
label_location_sig(data, label_location = 0.5, ...)

all_nonsig(data, expr, ...)

hide_nonsig(data, ...)

show_nonsig(data, ...)

colour_nonsig(data, colour = "black", ...)

color_nonsig(data, color = "black", ...)

linetype_nonsig(data, linetype = 1, ...)

size_nonsig(data, size = 1, ...)

alpha_nonsig(data, alpha = 1, ...)

fill_nonsig(data, fill = "white", ...)

label_colour_nonsig(data, label_colour = "black", ...)

label_color_nonsig(data, label_color = "black", ...)

label_fill_nonsig(data, label_fill = "white", ...)

label_size_nonsig(data, label_size = 4, ...)

label_alpha_nonsig(data, label_alpha = 1, ...)

label_family_nonsig(data, label_family = "sans", ...)

label_fontface_nonsig(data, label_fontface = "plain", ...)

label_hjust_nonsig(data, label_hjust = "center", ...)

label_vjust_nonsig(data, label_vjust = "middle", ...)

label_lineheight_nonsig(data, label_lineheight = 1, ...)

label_location_nonsig(data, label_location = 0.5, ...)

all_fixed(data, expr, ...)

hide_fixed(data, ...)

show_fixed(data, ...)
```

```
colour_fixed(data, colour = "black", ...)

color_fixed(data, color = "black", ...)

linetype_fixed(data, linetype = 1, ...)

size_fixed(data, size = 1, ...)

alpha_fixed(data, alpha = 1, ...)

fill_fixed(data, fill = "white", ...)

label_colour_fixed(data, label_colour = "black", ...)

label_color_fixed(data, label_color = "black", ...)

label_fill_fixed(data, label_fill = "white", ...)

label_size_fixed(data, label_size = 4, ...)

label_alpha_fixed(data, label_alpha = 1, ...)

label_family_fixed(data, label_family = "sans", ...)

label_fontface_fixed(data, label_fontface = "plain", ...)

label_hjust_fixed(data, label_hjust = "center", ...)

label_vjust_fixed(data, label_vjust = "middle", ...)

label_lineheight_fixed(data, label_lineheight = 1, ...)

label_location_fixed(data, label_location = 0.5, ...)

all_pos(data, expr, ...)

hide_pos(data, ...)

show_pos(data, ...)

colour_pos(data, colour = "black", ...)

color_pos(data, color = "black", ...)

linetype_pos(data, linetype = 1, ...)

size_pos(data, size = 1, ...)
```

```
alpha_pos(data, alpha = 1, ...)

fill_pos(data, fill = "white", ...)

label_colour_pos(data, label_colour = "black", ...)

label_color_pos(data, label_color = "black", ...)

label_fill_pos(data, label_fill = "white", ...)

label_size_pos(data, label_size = 4, ...)

label_alpha_pos(data, label_alpha = 1, ...)

label_family_pos(data, label_family = "sans", ...)

label_fontface_pos(data, label_fontface = "plain", ...)

label_hjust_pos(data, label_hjust = "center", ...)

label_vjust_pos(data, label_vjust = "middle", ...)

label_lineheight_pos(data, label_lineheight = 1, ...)

label_location_pos(data, label_location = 0.5, ...)

all_neg(data, expr, ...)

hide_neg(data, ...)

show_neg(data, ...)

colour_neg(data, colour = "black", ...)

color_neg(data, color = "black", ...)

linetype_neg(data, linetype = 1, ...)

size_neg(data, size = 1, ...)

alpha_neg(data, alpha = 1, ...)

fill_neg(data, fill = "white", ...)

label_colour_neg(data, label_colour = "black", ...)

label_color_neg(data, label_color = "black", ...)
```

```
label_fill_neg(data, label_fill = "white", ...)

label_size_neg(data, label_size = 4, ...)

label_alpha_neg(data, label_alpha = 1, ...)

label_family_neg(data, label_family = "sans", ...)

label_fontface_neg(data, label_fontface = "plain", ...)

label_hjust_neg(data, label_hjust = "center", ...)

label_vjust_neg(data, label_vjust = "middle", ...)

label_lineheight_neg(data, label_lineheight = 1, ...)

label_location_neg(data, label_location = 0.5, ...)

all_var(data, expr, ...)

hide_var(data, ...)

show_var(data, ...)

colour_var(data, colour = "black", ...)

color_var(data, color = "black", ...)

linetype_var(data, linetype = 1, ...)

size_var(data, size = 1, ...)

alpha_var(data, alpha = 1, ...)

label_colour_var(data, label_colour = "black", ...)

label_color_var(data, label_color = "black", ...)

label_fill_var(data, label_fill = "white", ...)

label_size_var(data, label_size = 4, ...)

label_alpha_var(data, label_alpha = 1, ...)

label_family_var(data, label_family = "sans", ...)

label_fontface_var(data, label_fontface = "plain", ...)
```

```
label_hjust_var(data, label_hjust = "center", ...)

label_vjust_var(data, label_vjust = "middle", ...)

label_lineheight_var(data, label_lineheight = 1, ...)

all_cov(data, expr, ...)

hide_cov(data, ...)

show_cov(data, ...)

colour_cov(data, colour = "black", ...)

color_cov(data, color = "black", ...)

linetype_cov(data, linetype = 1, ...)

size_cov(data, size = 1, ...)

alpha_cov(data, alpha = 1, ...)

label_colour_cov(data, label_colour = "black", ...)

label_color_cov(data, label_color = "black", ...)

label_fill_cov(data, label_fill = "white", ...)

label_size_cov(data, label_size = 4, ...)

label_alpha_cov(data, label_alpha = 1, ...)

label_family_cov(data, label_family = "sans", ...)

label_fontface_cov(data, label_fontface = "plain", ...)

label_hjust_cov(data, label_hjust = "center", ...)

label_vjust_cov(data, label_vjust = "middle", ...)

label_lineheight_cov(data, label_lineheight = 1, ...)

label_location_cov(data, label_location = 0.5, ...)

all_reg(data, expr, ...)

hide_reg(data, ...)
```

```
show_reg(data, ...)

colour_reg(data, colour = "black", ...)

color_reg(data, color = "black", ...)

linetype_reg(data, linetype = 1, ...)

size_reg(data, size = 1, ...)

alpha_reg(data, alpha = 1, ...)

label_colour_reg(data, label_colour = "black", ...)

label_color_reg(data, label_color = "black", ...)

label_fill_reg(data, label_fill = "white", ...)

label_size_reg(data, label_size = 4, ...)

label_alpha_reg(data, label_alpha = 1, ...)

label_family_reg(data, label_family = "sans", ...)

label_fontface_reg(data, label_fontface = "plain", ...)

label_hjust_reg(data, label_hjust = "center", ...)

label_vjust_reg(data, label_vjust = "middle", ...)

label_lineheight_reg(data, label_lineheight = 1, ...)

label_location_reg(data, label_location = 0.5, ...)

all_load(data, expr, ...)

hide_load(data, ...)

show_load(data, ...)

colour_load(data, colour = "black", ...)

color_load(data, color = "black", ...)

linetype_load(data, linetype = 1, ...)

size_load(data, size = 1, ...)
```

```
alpha_load(data, alpha = 1, ...)

label_colour_load(data, label_colour = "black", ...)

label_color_load(data, label_color = "black", ...)

label_fill_load(data, label_fill = "white", ...)

label_size_load(data, label_size = 4, ...)

label_alpha_load(data, label_alpha = 1, ...)

label_family_load(data, label_family = "sans", ...)

label_fontface_load(data, label_fontface = "plain", ...)

label_hjust_load(data, label_hjust = "center", ...)

label_vjust_load(data, label_vjust = "middle", ...)

label_lineheight_load(data, label_lineheight = 1, ...)

label_location_load(data, label_location = 0.5, ...)

all_obs(data, expr, ...)

hide_obs(data, ...)

show_obs(data, ...)

colour_obs(data, colour = "black", ...)

color_obs(data, color = "black", ...)

linetype_obs(data, linetype = 1, ...)

size_obs(data, size = 1, ...)

alpha_obs(data, alpha = 1, ...)

fill_obs(data, fill = "white", ...)

label_colour_obs(data, label_colour = "black", ...)

label_color_obs(data, label_color = "black", ...)

label_fill_obs(data, label_fill = "white", ...)
```

```
label_size_obs(data, label_size = 4, ...)

label_alpha_obs(data, label_alpha = 1, ...)

label_family_obs(data, label_family = "sans", ...)

label_fontface_obs(data, label_fontface = "plain", ...)

label_hjust_obs(data, label_hjust = "center", ...)

label_vjust_obs(data, label_vjust = "middle", ...)

label_lineheight_obs(data, label_lineheight = 1, ...)

all_latent(data, expr, ...)

hide_latent(data, ...)

show_latent(data, ...)

colour_latent(data, colour = "black", ...)

color_latent(data, color = "black", ...)

linetype_latent(data, linetype = 1, ...)

size_latent(data, size = 1, ...)

alpha_latent(data, alpha = 1, ...)

fill_latent(data, fill = "white", ...)

label_colour_latent(data, label_colour = "black", ...)

label_color_latent(data, label_color = "black", ...)

label_fill_latent(data, label_fill = "white", ...)

label_size_latent(data, label_size = 4, ...)

label_alpha_latent(data, label_alpha = 1, ...)

label_family_latent(data, label_family = "sans", ...)

label_fontface_latent(data, label_fontface = "plain", ...)

label_hjust_latent(data, label_hjust = "center", ...)
```

```
label_vjust_latent(data, label_vjust = "middle", ...)

label_lineheight_latent(data, label_lineheight = 1, ...)

all_sig_nodes(data, expr, ...)

hide_sig_nodes(data, ...)

show_sig_nodes(data, ...)

colour_sig_nodes(data, colour = "black", ...)

color_sig_nodes(data, color = "black", ...)

linetype_sig_nodes(data, linetype = 1, ...)

size_sig_nodes(data, size = 1, ...)

alpha_sig_nodes(data, alpha = 1, ...)

label_colour_sig_nodes(data, label_colour = "black", ...)

label_color_sig_nodes(data, label_color = "black", ...)

label_fill_sig_nodes(data, label_fill = "white", ...)

label_size_sig_nodes(data, label_size = 4, ...)

label_alpha_sig_nodes(data, label_alpha = 1, ...)

label_family_sig_nodes(data, label_family = "sans", ...)

label_fontface_sig_nodes(data, label_fontface = "plain", ...)

label_hjust_sig_nodes(data, label_hjust = "center", ...)

label_vjust_sig_nodes(data, label_vjust = "middle", ...)

label_lineheight_sig_nodes(data, label_lineheight = 1, ...)

all_nonsig_nodes(data, expr, ...)

hide_nonsig_nodes(data, ...)

show_nonsig_nodes(data, ...)

colour_nonsig_nodes(data, colour = "black", ...)
```

```
color_nonsig_nodes(data, color = "black", ...)

linetype_nonsig_nodes(data, linetype = 1, ...)

size_nonsig_nodes(data, size = 1, ...)

alpha_nonsig_nodes(data, alpha = 1, ...)

label_colour_nonsig_nodes(data, label_colour = "black", ...)

label_color_nonsig_nodes(data, label_color = "black", ...)

label_fill_nonsig_nodes(data, label_fill = "white", ...)

label_size_nonsig_nodes(data, label_size = 4, ...)

label_alpha_nonsig_nodes(data, label_alpha = 1, ...)

label_family_nonsig_nodes(data, label_family = "sans", ...)

label_fontface_nonsig_nodes(data, label_fontface = "plain", ...)

label_hjust_nonsig_nodes(data, label_hjust = "center", ...)

label_vjust_nonsig_nodes(data, label_vjust = "middle", ...)

label_lineheight_nonsig_nodes(data, label_lineheight = 1, ...)

all_fixed_nodes(data, expr, ...)

hide_fixed_nodes(data, ...)

show_fixed_nodes(data, ...)

colour_fixed_nodes(data, colour = "black", ...)

color_fixed_nodes(data, color = "black", ...)

linetype_fixed_nodes(data, linetype = 1, ...)

size_fixed_nodes(data, size = 1, ...)

alpha_fixed_nodes(data, alpha = 1, ...)

label_colour_fixed_nodes(data, label_colour = "black", ...)

label_color_fixed_nodes(data, label_color = "black", ...)
```

```
label_fill_fixed_nodes(data, label_fill = "white", ...)

label_size_fixed_nodes(data, label_size = 4, ...)

label_alpha_fixed_nodes(data, label_alpha = 1, ...)

label_family_fixed_nodes(data, label_family = "sans", ...)

label_fontface_fixed_nodes(data, label_fontface = "plain", ...)

label_hjust_fixed_nodes(data, label_hjust = "center", ...)

label_vjust_fixed_nodes(data, label_vjust = "middle", ...)

label_lineheight_fixed_nodes(data, label_lineheight = 1, ...)

all_pos_nodes(data, expr, ...)

hide_pos_nodes(data, ...)

show_pos_nodes(data, ...)

colour_pos_nodes(data, colour = "black", ...)

color_pos_nodes(data, color = "black", ...)

linetype_pos_nodes(data, linetype = 1, ...)

size_pos_nodes(data, size = 1, ...)

alpha_pos_nodes(data, alpha = 1, ...)

label_colour_pos_nodes(data, label_colour = "black", ...)

label_color_pos_nodes(data, label_color = "black", ...)

label_fill_pos_nodes(data, label_fill = "white", ...)

label_size_pos_nodes(data, label_size = 4, ...)

label_alpha_pos_nodes(data, label_alpha = 1, ...)

label_family_pos_nodes(data, label_family = "sans", ...)

label_fontface_pos_nodes(data, label_fontface = "plain", ...)

label_hjust_pos_nodes(data, label_hjust = "center", ...)
```

```
label_vjust_pos_nodes(data, label_vjust = "middle", ...)

label_lineheight_pos_nodes(data, label_lineheight = 1, ...)

all_neg_nodes(data, expr, ...)

hide_neg_nodes(data, ...)

show_neg_nodes(data, ...)

colour_neg_nodes(data, colour = "black", ...)

color_neg_nodes(data, color = "black", ...)

linetype_neg_nodes(data, linetype = 1, ...)

size_neg_nodes(data, size = 1, ...)

alpha_neg_nodes(data, alpha = 1, ...)

label_colour_neg_nodes(data, label_colour = "black", ...)

label_color_neg_nodes(data, label_color = "black", ...)

label_fill_neg_nodes(data, label_fill = "white", ...)

label_size_neg_nodes(data, label_size = 4, ...)

label_alpha_neg_nodes(data, label_alpha = 1, ...)

label_family_neg_nodes(data, label_family = "sans", ...)

label_fontface_neg_nodes(data, label_fontface = "plain", ...)

label_hjust_neg_nodes(data, label_hjust = "center", ...)

label_vjust_neg_nodes(data, label_vjust = "middle", ...)

label_lineheight_neg_nodes(data, label_lineheight = 1, ...)

all_sig_edges(data, expr, ...)

hide_sig_edges(data, ...)

show_sig_edges(data, ...)

colour_sig_edges(data, colour = "black", ...)
```

```
color_sig_edges(data, color = "black", ...)

linetype_sig_edges(data, linetype = 1, ...)

size_sig_edges(data, size = 1, ...)

alpha_sig_edges(data, alpha = 1, ...)

label_colour_sig_edges(data, label_colour = "black", ...)

label_color_sig_edges(data, label_color = "black", ...)

label_fill_sig_edges(data, label_fill = "white", ...)

label_size_sig_edges(data, label_size = 4, ...)

label_alpha_sig_edges(data, label_alpha = 1, ...)

label_family_sig_edges(data, label_family = "sans", ...)

label_fontface_sig_edges(data, label_fontface = "plain", ...)

label_hjust_sig_edges(data, label_hjust = "center", ...)

label_vjust_sig_edges(data, label_vjust = "middle", ...)

label_lineheight_sig_edges(data, label_lineheight = 1, ...)

all_nonsig_edges(data, expr, ...)

hide_nonsig_edges(data, ...)

show_nonsig_edges(data, ...)

colour_nonsig_edges(data, colour = "black", ...)

color_nonsig_edges(data, color = "black", ...)

linetype_nonsig_edges(data, linetype = 1, ...)

size_nonsig_edges(data, size = 1, ...)

alpha_nonsig_edges(data, alpha = 1, ...)

label_colour_nonsig_edges(data, label_colour = "black", ...)

label_color_nonsig_edges(data, label_color = "black", ...)
```

```
label_fill_nonsig_edges(data, label_fill = "white", ...)

label_size_nonsig_edges(data, label_size = 4, ...)

label_alpha_nonsig_edges(data, label_alpha = 1, ...)

label_family_nonsig_edges(data, label_family = "sans", ...)

label_fontface_nonsig_edges(data, label_fontface = "plain", ...)

label_hjust_nonsig_edges(data, label_hjust = "center", ...)

label_vjust_nonsig_edges(data, label_vjust = "middle", ...)

label_lineheight_nonsig_edges(data, label_lineheight = 1, ...)

all_fixed_edges(data, expr, ...)

hide_fixed_edges(data, ...)

show_fixed_edges(data, ...)

colour_fixed_edges(data, colour = "black", ...)

color_fixed_edges(data, color = "black", ...)

linetype_fixed_edges(data, linetype = 1, ...)

size_fixed_edges(data, size = 1, ...)

alpha_fixed_edges(data, alpha = 1, ...)

label_colour_fixed_edges(data, label_colour = "black", ...)

label_color_fixed_edges(data, label_color = "black", ...)

label_fill_fixed_edges(data, label_fill = "white", ...)

label_size_fixed_edges(data, label_size = 4, ...)

label_alpha_fixed_edges(data, label_alpha = 1, ...)

label_family_fixed_edges(data, label_family = "sans", ...)

label_fontface_fixed_edges(data, label_fontface = "plain", ...)

label_hjust_fixed_edges(data, label_hjust = "center", ...)
```

```
label_vjust_fixed_edges(data, label_vjust = "middle", ...)

label_lineheight_fixed_edges(data, label_lineheight = 1, ...)

all_pos_edges(data, expr, ...)

hide_pos_edges(data, ...)

show_pos_edges(data, ...)

colour_pos_edges(data, colour = "black", ...)

color_pos_edges(data, color = "black", ...)

linetype_pos_edges(data, linetype = 1, ...)

size_pos_edges(data, size = 1, ...)

alpha_pos_edges(data, alpha = 1, ...)

label_colour_pos_edges(data, label_colour = "black", ...)

label_color_pos_edges(data, label_color = "black", ...)

label_fill_pos_edges(data, label_fill = "white", ...)

label_size_pos_edges(data, label_size = 4, ...)

label_alpha_pos_edges(data, label_alpha = 1, ...)

label_family_pos_edges(data, label_family = "sans", ...)

label_fontface_pos_edges(data, label_fontface = "plain", ...)

label_hjust_pos_edges(data, label_hjust = "center", ...)

label_vjust_pos_edges(data, label_vjust = "middle", ...)

label_lineheight_pos_edges(data, label_lineheight = 1, ...)

all_neg_edges(data, expr, ...)

hide_neg_edges(data, ...)

show_neg_edges(data, ...)

colour_neg_edges(data, colour = "black", ...)
```

```
color_neg_edges(data, color = "black", ...)

linetype_neg_edges(data, linetype = 1, ...)

size_neg_edges(data, size = 1, ...)

alpha_neg_edges(data, alpha = 1, ...)

label_colour_neg_edges(data, label_colour = "black", ...)

label_color_neg_edges(data, label_color = "black", ...)

label_fill_neg_edges(data, label_fill = "white", ...)

label_size_neg_edges(data, label_size = 4, ...)

label_alpha_neg_edges(data, label_alpha = 1, ...)

label_family_neg_edges(data, label_family = "sans", ...)

label_fontface_neg_edges(data, label_fontface = "plain", ...)

label_hjust_neg_edges(data, label_hjust = "center", ...)

label_vjust_neg_edges(data, label_vjust = "middle", ...)

label_lineheight_neg_edges(data, label_lineheight = 1, ...)
```

Arguments

`data`	Object to manipulate.
`condition`	Expression that returns a logical vector when evaluated in the environment of `data`.
`expr`	Expression to perform on elements of `data` for which `condition` == `TRUE`.
`...`	Additional arguments passed to and from functions.
`element`	Character vector. The elements of the `sem_graph` to edit, defaults to `c("edges","nodes")`.
`colour`	Atomic character vector, indicating which colour to assign to the selected elements.
`color`	Atomic character vector, indicating which color to assign to the selected elements.
`linetype`	Atomic character vector, indicating which linetype to assign to the selected elements.
`size`	Atomic character vector, indicating which size to assign to the selected elements.
`alpha`	Atomic character vector, indicating which alpha to assign to the selected elements.
`fill`	Atomic character vector, indicating which fill to assign to the selected elements.

`label_colour`	Atomic character vector, indicating which label_colour to assign to the selected elements.
`label_color`	Atomic character vector, indicating which label_color to assign to the selected elements.
`label_fill`	Atomic character vector, indicating which label_fill to assign to the selected elements.
`label_size`	Atomic character vector, indicating which label_size to assign to the selected elements.
`label_alpha`	Atomic character vector, indicating which label_alpha to assign to the selected elements.
`label_family`	Atomic character vector, indicating which label_family to assign to the selected elements.
`label_fontface`	Atomic character vector, indicating which label_fontface to assign to the selected elements.
`label_hjust`	Atomic character vector, indicating which label_hjust to assign to the selected elements.
`label_vjust`	Atomic character vector, indicating which label_vjust to assign to the selected elements.
`label_lineheight`	Atomic character vector, indicating which label_lineheight to assign to the selected elements.
`label_location`	Atomic character vector, indicating which label_location to assign to the selected elements.

Value

Object of the same class as `data`.

Examples

```
library(lavaan)
res <- sem("dist ~ speed", cars, meanstructure = TRUE)
p <- prepare_graph(res)
out <- if_edit(p, condition = {pval < .05}, expr = {label = "sig"})
out <- if_edges(p, condition = {pval < .05}, expr = {label = "sig"})
out <- if_nodes(p, condition = {pval < .05}, expr = {label = "sig"})
out <- all_sig(p, expr = {label = "sig"})
out <- hide_sig(p)
out <- show_sig(p)
out <- colour_sig(p, { colour = "black" })
out <- color_sig(p, { color = "black" })
out <- linetype_sig(p, { linetype = 1 })
out <- size_sig(p, { size = 1 })
out <- alpha_sig(p, { alpha = 1 })
out <- fill_sig(p, { fill = "white" })
out <- label_colour_sig(p, { label_colour = "black" })
out <- label_color_sig(p, { label_color = "black" })
out <- label_fill_sig(p, { label_fill = "white" })
out <- label_size_sig(p, { label_size = 4 })
out <- label_alpha_sig(p, { label_alpha = 1 })
out <- label_family_sig(p, { label_family = "sans" })
out <- label_fontface_sig(p, { label_fontface = "plain" })
out <- label_hjust_sig(p, { label_hjust = "center" })
out <- label_vjust_sig(p, { label_vjust = "middle" })
out <- label_lineheight_sig(p, { label_lineheight = 1 })
out <- label_location_sig(p, { label_location = .5 })
out <- all_nonsig(p, expr = {label = "sig"})
out <- hide_nonsig(p)
out <- show_nonsig(p)
out <- colour_nonsig(p, { colour = "black" })
out <- color_nonsig(p, { color = "black" })
out <- linetype_nonsig(p, { linetype = 1 })
out <- size_nonsig(p, { size = 1 })
out <- alpha_nonsig(p, { alpha = 1 })
out <- fill_nonsig(p, { fill = "white" })
out <- label_colour_nonsig(p, { label_colour = "black" })
out <- label_color_nonsig(p, { label_color = "black" })
out <- label_fill_nonsig(p, { label_fill = "white" })
out <- label_size_nonsig(p, { label_size = 4 })
out <- label_alpha_nonsig(p, { label_alpha = 1 })
out <- label_family_nonsig(p, { label_family = "sans" })
out <- label_fontface_nonsig(p, { label_fontface = "plain" })
out <- label_hjust_nonsig(p, { label_hjust = "center" })
out <- label_vjust_nonsig(p, { label_vjust = "middle" })
out <- label_lineheight_nonsig(p, { label_lineheight = 1 })
out <- label_location_nonsig(p, { label_location = .5 })
out <- all_fixed(p, expr = {label = "sig"})
out <- hide_fixed(p)
out <- show_fixed(p)
out <- colour_fixed(p, { colour = "black" })
out <- color_fixed(p, { color = "black" })
out <- linetype_fixed(p, { linetype = 1 })
out <- size_fixed(p, { size = 1 })
out <- alpha_fixed(p, { alpha = 1 })
out <- fill_fixed(p, { fill = "white" })
out <- label_colour_fixed(p, { label_colour = "black" })
out <- label_color_fixed(p, { label_color = "black" })
out <- label_fill_fixed(p, { label_fill = "white" })
out <- label_size_fixed(p, { label_size = 4 })
out <- label_alpha_fixed(p, { label_alpha = 1 })
out <- label_family_fixed(p, { label_family = "sans" })
out <- label_fontface_fixed(p, { label_fontface = "plain" })
out <- label_hjust_fixed(p, { label_hjust = "center" })
out <- label_vjust_fixed(p, { label_vjust = "middle" })
out <- label_lineheight_fixed(p, { label_lineheight = 1 })
out <- label_location_fixed(p, { label_location = .5 })
out <- all_pos(p, expr = {label = "sig"})
out <- hide_pos(p)
out <- show_pos(p)
out <- colour_pos(p, { colour = "black" })
out <- color_pos(p, { color = "black" })
out <- linetype_pos(p, { linetype = 1 })
out <- size_pos(p, { size = 1 })
out <- alpha_pos(p, { alpha = 1 })
out <- fill_pos(p, { fill = "white" })
out <- label_colour_pos(p, { label_colour = "black" })
out <- label_color_pos(p, { label_color = "black" })
out <- label_fill_pos(p, { label_fill = "white" })
out <- label_size_pos(p, { label_size = 4 })
out <- label_alpha_pos(p, { label_alpha = 1 })
out <- label_family_pos(p, { label_family = "sans" })
out <- label_fontface_pos(p, { label_fontface = "plain" })
out <- label_hjust_pos(p, { label_hjust = "center" })
out <- label_vjust_pos(p, { label_vjust = "middle" })
out <- label_lineheight_pos(p, { label_lineheight = 1 })
out <- label_location_pos(p, { label_location = .5 })
out <- all_neg(p, expr = {label = "sig"})
out <- hide_neg(p)
out <- show_neg(p)
out <- colour_neg(p, { colour = "black" })
out <- color_neg(p, { color = "black" })
out <- linetype_neg(p, { linetype = 1 })
out <- size_neg(p, { size = 1 })
out <- alpha_neg(p, { alpha = 1 })
out <- fill_neg(p, { fill = "white" })
out <- label_colour_neg(p, { label_colour = "black" })
out <- label_color_neg(p, { label_color = "black" })
out <- label_fill_neg(p, { label_fill = "white" })
out <- label_size_neg(p, { label_size = 4 })
out <- label_alpha_neg(p, { label_alpha = 1 })
out <- label_family_neg(p, { label_family = "sans" })
out <- label_fontface_neg(p, { label_fontface = "plain" })
out <- label_hjust_neg(p, { label_hjust = "center" })
out <- label_vjust_neg(p, { label_vjust = "middle" })
out <- label_lineheight_neg(p, { label_lineheight = 1 })
out <- label_location_neg(p, { label_location = .5 })
out <- all_var(p, expr = {label = "sig"})
out <- hide_var(p)
out <- show_var(p)
out <- colour_var(p, { colour = "black" })
out <- color_var(p, { color = "black" })
out <- linetype_var(p, { linetype = 1 })
out <- size_var(p, { size = 1 })
out <- alpha_var(p, { alpha = 1 })
out <- label_colour_var(p, { label_colour = "black" })
out <- label_color_var(p, { label_color = "black" })
out <- label_fill_var(p, { label_fill = "white" })
out <- label_size_var(p, { label_size = 4 })
out <- label_alpha_var(p, { label_alpha = 1 })
out <- label_family_var(p, { label_family = "sans" })
out <- label_fontface_var(p, { label_fontface = "plain" })
out <- label_hjust_var(p, { label_hjust = "center" })
out <- label_vjust_var(p, { label_vjust = "middle" })
out <- label_lineheight_var(p, { label_lineheight = 1 })
out <- all_cov(p, expr = {label = "sig"})
out <- hide_cov(p)
out <- show_cov(p)
out <- colour_cov(p, { colour = "black" })
out <- color_cov(p, { color = "black" })
out <- linetype_cov(p, { linetype = 1 })
out <- size_cov(p, { size = 1 })
out <- alpha_cov(p, { alpha = 1 })
out <- label_colour_cov(p, { label_colour = "black" })
out <- label_color_cov(p, { label_color = "black" })
out <- label_fill_cov(p, { label_fill = "white" })
out <- label_size_cov(p, { label_size = 4 })
out <- label_alpha_cov(p, { label_alpha = 1 })
out <- label_family_cov(p, { label_family = "sans" })
out <- label_fontface_cov(p, { label_fontface = "plain" })
out <- label_hjust_cov(p, { label_hjust = "center" })
out <- label_vjust_cov(p, { label_vjust = "middle" })
out <- label_lineheight_cov(p, { label_lineheight = 1 })
out <- label_location_cov(p, { label_location = .5 })
out <- all_reg(p, expr = {label = "sig"})
out <- hide_reg(p)
out <- show_reg(p)
out <- colour_reg(p, { colour = "black" })
out <- color_reg(p, { color = "black" })
out <- linetype_reg(p, { linetype = 1 })
out <- size_reg(p, { size = 1 })
out <- alpha_reg(p, { alpha = 1 })
out <- label_colour_reg(p, { label_colour = "black" })
out <- label_color_reg(p, { label_color = "black" })
out <- label_fill_reg(p, { label_fill = "white" })
out <- label_size_reg(p, { label_size = 4 })
out <- label_alpha_reg(p, { label_alpha = 1 })
out <- label_family_reg(p, { label_family = "sans" })
out <- label_fontface_reg(p, { label_fontface = "plain" })
out <- label_hjust_reg(p, { label_hjust = "center" })
out <- label_vjust_reg(p, { label_vjust = "middle" })
out <- label_lineheight_reg(p, { label_lineheight = 1 })
out <- label_location_reg(p, { label_location = .5 })
out <- all_load(p, expr = {label = "sig"})
out <- hide_load(p)
out <- show_load(p)
out <- colour_load(p, { colour = "black" })
out <- color_load(p, { color = "black" })
out <- linetype_load(p, { linetype = 1 })
out <- size_load(p, { size = 1 })
out <- alpha_load(p, { alpha = 1 })
out <- label_colour_load(p, { label_colour = "black" })
out <- label_color_load(p, { label_color = "black" })
out <- label_fill_load(p, { label_fill = "white" })
out <- label_size_load(p, { label_size = 4 })
out <- label_alpha_load(p, { label_alpha = 1 })
out <- label_family_load(p, { label_family = "sans" })
out <- label_fontface_load(p, { label_fontface = "plain" })
out <- label_hjust_load(p, { label_hjust = "center" })
out <- label_vjust_load(p, { label_vjust = "middle" })
out <- label_lineheight_load(p, { label_lineheight = 1 })
out <- label_location_load(p, { label_location = .5 })
out <- all_obs(p, expr = {label = "sig"})
out <- hide_obs(p)
out <- show_obs(p)
out <- colour_obs(p, { colour = "black" })
out <- color_obs(p, { color = "black" })
out <- linetype_obs(p, { linetype = 1 })
out <- size_obs(p, { size = 1 })
out <- alpha_obs(p, { alpha = 1 })
out <- fill_obs(p, { fill = "white" })
out <- label_colour_obs(p, { label_colour = "black" })
out <- label_color_obs(p, { label_color = "black" })
out <- label_fill_obs(p, { label_fill = "white" })
out <- label_size_obs(p, { label_size = 4 })
out <- label_alpha_obs(p, { label_alpha = 1 })
out <- label_family_obs(p, { label_family = "sans" })
out <- label_fontface_obs(p, { label_fontface = "plain" })
out <- label_hjust_obs(p, { label_hjust = "center" })
out <- label_vjust_obs(p, { label_vjust = "middle" })
out <- label_lineheight_obs(p, { label_lineheight = 1 })
out <- all_latent(p, expr = {label = "sig"})
out <- hide_latent(p)
out <- show_latent(p)
out <- colour_latent(p, { colour = "black" })
out <- color_latent(p, { color = "black" })
out <- linetype_latent(p, { linetype = 1 })
out <- size_latent(p, { size = 1 })
out <- alpha_latent(p, { alpha = 1 })
out <- fill_latent(p, { fill = "white" })
out <- label_colour_latent(p, { label_colour = "black" })
out <- label_color_latent(p, { label_color = "black" })
out <- label_fill_latent(p, { label_fill = "white" })
out <- label_size_latent(p, { label_size = 4 })
out <- label_alpha_latent(p, { label_alpha = 1 })
out <- label_family_latent(p, { label_family = "sans" })
out <- label_fontface_latent(p, { label_fontface = "plain" })
out <- label_hjust_latent(p, { label_hjust = "center" })
out <- label_vjust_latent(p, { label_vjust = "middle" })
out <- label_lineheight_latent(p, { label_lineheight = 1 })
out <- all_sig(p, expr = {label = "sig"})
out <- hide_sig(p)
out <- show_sig(p)
out <- colour_sig(p, { colour = "black" })
out <- color_sig(p, { color = "black" })
out <- linetype_sig(p, { linetype = 1 })
out <- size_sig(p, { size = 1 })
out <- alpha_sig(p, { alpha = 1 })
out <- label_colour_sig(p, { label_colour = "black" })
out <- label_color_sig(p, { label_color = "black" })
out <- label_fill_sig(p, { label_fill = "white" })
out <- label_size_sig(p, { label_size = 4 })
out <- label_alpha_sig(p, { label_alpha = 1 })
out <- label_family_sig(p, { label_family = "sans" })
out <- label_fontface_sig(p, { label_fontface = "plain" })
out <- label_hjust_sig(p, { label_hjust = "center" })
out <- label_vjust_sig(p, { label_vjust = "middle" })
out <- label_lineheight_sig(p, { label_lineheight = 1 })
out <- all_nonsig(p, expr = {label = "sig"})
out <- hide_nonsig(p)
out <- show_nonsig(p)
out <- colour_nonsig(p, { colour = "black" })
out <- color_nonsig(p, { color = "black" })
out <- linetype_nonsig(p, { linetype = 1 })
out <- size_nonsig(p, { size = 1 })
out <- alpha_nonsig(p, { alpha = 1 })
out <- label_colour_nonsig(p, { label_colour = "black" })
out <- label_color_nonsig(p, { label_color = "black" })
out <- label_fill_nonsig(p, { label_fill = "white" })
out <- label_size_nonsig(p, { label_size = 4 })
out <- label_alpha_nonsig(p, { label_alpha = 1 })
out <- label_family_nonsig(p, { label_family = "sans" })
out <- label_fontface_nonsig(p, { label_fontface = "plain" })
out <- label_hjust_nonsig(p, { label_hjust = "center" })
out <- label_vjust_nonsig(p, { label_vjust = "middle" })
out <- label_lineheight_nonsig(p, { label_lineheight = 1 })
out <- all_fixed(p, expr = {label = "sig"})
out <- hide_fixed(p)
out <- show_fixed(p)
out <- colour_fixed(p, { colour = "black" })
out <- color_fixed(p, { color = "black" })
out <- linetype_fixed(p, { linetype = 1 })
out <- size_fixed(p, { size = 1 })
out <- alpha_fixed(p, { alpha = 1 })
out <- label_colour_fixed(p, { label_colour = "black" })
out <- label_color_fixed(p, { label_color = "black" })
out <- label_fill_fixed(p, { label_fill = "white" })
out <- label_size_fixed(p, { label_size = 4 })
out <- label_alpha_fixed(p, { label_alpha = 1 })
out <- label_family_fixed(p, { label_family = "sans" })
out <- label_fontface_fixed(p, { label_fontface = "plain" })
out <- label_hjust_fixed(p, { label_hjust = "center" })
out <- label_vjust_fixed(p, { label_vjust = "middle" })
out <- label_lineheight_fixed(p, { label_lineheight = 1 })
out <- all_pos(p, expr = {label = "sig"})
out <- hide_pos(p)
out <- show_pos(p)
out <- colour_pos(p, { colour = "black" })
out <- color_pos(p, { color = "black" })
out <- linetype_pos(p, { linetype = 1 })
out <- size_pos(p, { size = 1 })
out <- alpha_pos(p, { alpha = 1 })
out <- label_colour_pos(p, { label_colour = "black" })
out <- label_color_pos(p, { label_color = "black" })
out <- label_fill_pos(p, { label_fill = "white" })
out <- label_size_pos(p, { label_size = 4 })
out <- label_alpha_pos(p, { label_alpha = 1 })
out <- label_family_pos(p, { label_family = "sans" })
out <- label_fontface_pos(p, { label_fontface = "plain" })
out <- label_hjust_pos(p, { label_hjust = "center" })
out <- label_vjust_pos(p, { label_vjust = "middle" })
out <- label_lineheight_pos(p, { label_lineheight = 1 })
out <- all_neg(p, expr = {label = "sig"})
out <- hide_neg(p)
out <- show_neg(p)
out <- colour_neg(p, { colour = "black" })
out <- color_neg(p, { color = "black" })
out <- linetype_neg(p, { linetype = 1 })
out <- size_neg(p, { size = 1 })
out <- alpha_neg(p, { alpha = 1 })
out <- label_colour_neg(p, { label_colour = "black" })
out <- label_color_neg(p, { label_color = "black" })
out <- label_fill_neg(p, { label_fill = "white" })
out <- label_size_neg(p, { label_size = 4 })
out <- label_alpha_neg(p, { label_alpha = 1 })
out <- label_family_neg(p, { label_family = "sans" })
out <- label_fontface_neg(p, { label_fontface = "plain" })
out <- label_hjust_neg(p, { label_hjust = "center" })
out <- label_vjust_neg(p, { label_vjust = "middle" })
out <- label_lineheight_neg(p, { label_lineheight = 1 })
out <- all_sig(p, expr = {label = "sig"})
out <- hide_sig(p)
out <- show_sig(p)
out <- colour_sig(p, { colour = "black" })
out <- color_sig(p, { color = "black" })
out <- linetype_sig(p, { linetype = 1 })
out <- size_sig(p, { size = 1 })
out <- alpha_sig(p, { alpha = 1 })
out <- label_colour_sig(p, { label_colour = "black" })
out <- label_color_sig(p, { label_color = "black" })
out <- label_fill_sig(p, { label_fill = "white" })
out <- label_size_sig(p, { label_size = 4 })
out <- label_alpha_sig(p, { label_alpha = 1 })
out <- label_family_sig(p, { label_family = "sans" })
out <- label_fontface_sig(p, { label_fontface = "plain" })
out <- label_hjust_sig(p, { label_hjust = "center" })
out <- label_vjust_sig(p, { label_vjust = "middle" })
out <- label_lineheight_sig(p, { label_lineheight = 1 })
out <- all_nonsig(p, expr = {label = "sig"})
out <- hide_nonsig(p)
out <- show_nonsig(p)
out <- colour_nonsig(p, { colour = "black" })
out <- color_nonsig(p, { color = "black" })
out <- linetype_nonsig(p, { linetype = 1 })
out <- size_nonsig(p, { size = 1 })
out <- alpha_nonsig(p, { alpha = 1 })
out <- label_colour_nonsig(p, { label_colour = "black" })
out <- label_color_nonsig(p, { label_color = "black" })
out <- label_fill_nonsig(p, { label_fill = "white" })
out <- label_size_nonsig(p, { label_size = 4 })
out <- label_alpha_nonsig(p, { label_alpha = 1 })
out <- label_family_nonsig(p, { label_family = "sans" })
out <- label_fontface_nonsig(p, { label_fontface = "plain" })
out <- label_hjust_nonsig(p, { label_hjust = "center" })
out <- label_vjust_nonsig(p, { label_vjust = "middle" })
out <- label_lineheight_nonsig(p, { label_lineheight = 1 })
out <- all_fixed(p, expr = {label = "sig"})
out <- hide_fixed(p)
out <- show_fixed(p)
out <- colour_fixed(p, { colour = "black" })
out <- color_fixed(p, { color = "black" })
out <- linetype_fixed(p, { linetype = 1 })
out <- size_fixed(p, { size = 1 })
out <- alpha_fixed(p, { alpha = 1 })
out <- label_colour_fixed(p, { label_colour = "black" })
out <- label_color_fixed(p, { label_color = "black" })
out <- label_fill_fixed(p, { label_fill = "white" })
out <- label_size_fixed(p, { label_size = 4 })
out <- label_alpha_fixed(p, { label_alpha = 1 })
out <- label_family_fixed(p, { label_family = "sans" })
out <- label_fontface_fixed(p, { label_fontface = "plain" })
out <- label_hjust_fixed(p, { label_hjust = "center" })
out <- label_vjust_fixed(p, { label_vjust = "middle" })
out <- label_lineheight_fixed(p, { label_lineheight = 1 })
out <- all_pos(p, expr = {label = "sig"})
out <- hide_pos(p)
out <- show_pos(p)
out <- colour_pos(p, { colour = "black" })
out <- color_pos(p, { color = "black" })
out <- linetype_pos(p, { linetype = 1 })
out <- size_pos(p, { size = 1 })
out <- alpha_pos(p, { alpha = 1 })
out <- label_colour_pos(p, { label_colour = "black" })
out <- label_color_pos(p, { label_color = "black" })
out <- label_fill_pos(p, { label_fill = "white" })
out <- label_size_pos(p, { label_size = 4 })
out <- label_alpha_pos(p, { label_alpha = 1 })
out <- label_family_pos(p, { label_family = "sans" })
out <- label_fontface_pos(p, { label_fontface = "plain" })
out <- label_hjust_pos(p, { label_hjust = "center" })
out <- label_vjust_pos(p, { label_vjust = "middle" })
out <- label_lineheight_pos(p, { label_lineheight = 1 })
out <- all_neg(p, expr = {label = "sig"})
out <- hide_neg(p)
out <- show_neg(p)
out <- colour_neg(p, { colour = "black" })
out <- color_neg(p, { color = "black" })
out <- linetype_neg(p, { linetype = 1 })
out <- size_neg(p, { size = 1 })
out <- alpha_neg(p, { alpha = 1 })
out <- label_colour_neg(p, { label_colour = "black" })
out <- label_color_neg(p, { label_color = "black" })
out <- label_fill_neg(p, { label_fill = "white" })
out <- label_size_neg(p, { label_size = 4 })
out <- label_alpha_neg(p, { label_alpha = 1 })
out <- label_family_neg(p, { label_family = "sans" })
out <- label_fontface_neg(p, { label_fontface = "plain" })
out <- label_hjust_neg(p, { label_hjust = "center" })
out <- label_vjust_neg(p, { label_vjust = "middle" })
out <- label_lineheight_neg(p, { label_lineheight = 1 })
```

lsub — *Apply pattern replacement over a vector*

Description

`lsub` returns a list of the same length as `replacement`, each element of which is the result of applying `gsub` to `x` using `lapply`.

Usage

```
lsub(x, replacement = NULL, pattern = "{C}", fixed = TRUE, ...)
```

Arguments

`x`	A character vector where matches are sought.
`replacement`	a character vector of length 1 or more. Each element is applied to `x` in turn. Default: NULL
`pattern`	A character string containing a regular expression (or character string when `fixed = TRUE`). Default: `'{C}'`.
`fixed`	logical. If TRUE, pattern is a string to be matched as is. Default: TRUE
`...`	Parameters passed on to `gsub`.

Value

A list of results returned by `gsub`.

Examples

```
lsub("a{C}", 1:3)
```

measurement *Generate syntax for a measurement model*

Description

Generate syntax for a measurement model for latent variables. This function relies on add_paths to generate syntax.

Usage

```
measurement(x, ...)
```

Arguments

x	An object for which a method exists, including tidy_sem (generated using dictionary, or data.frame (for which dictionary will be run first).
...	Additional parameters passed to add_paths.

Value

An object of class tidy_sem.

Examples

```
dict <- tidy_sem(c("bfi_1", "bfi_2", "bfi_3", "bfi_4", "bfi_5"))
measurement(dict)
```

mixture_starts *Automatically set starting values for an OpenMx mixture model*

Description

Automatically set starting values for an OpenMx mixture model. This function was designed to work with mixture models created using tidySEM functions like mx_mixture, and may not work with other mxModels.

Usage

```
mixture_starts(model, splits, ...)
```

Arguments

model	A mixture model of class mxModel.
splits	Optional. A numeric vector of length equal to the number of rows in the mxData used in the model object. The data will be split by this vector. See Details for the default setting and possible alternatives.
...	Additional arguments, passed to functions.

Details

Starting values are derived by the following procedure:

1. The mixture model is converted to a multi-group model.
2. The data are split along `splits`, and assigned to the corresponding groups of the multi-group model.
3. The multi-group model is run, and the final values of each group are assigned to the corresponding mixture component as starting values.
4. The mixture model is returned with these starting values.

If the argument `splits` is not provided, the function will call `kmeans(x = data,centers = classes)$cluster`, where `data` is extracted from the `model` argument.

Sensible ways to split the data include:

- Using Hierarchical clustering: `cutree(hclust(dist(data)),k = classes))`
- Using K-means clustering: `kmeans(x = data,centers = classes)$cluster`
- Using agglomerative hierarchical clustering: `hclass(hc(data = data),G = classes)[,1]`
- Using a random split: `sample.int(n = classes,size = nrow(data),replace = TRUE)`

Value

Returns an `mxModel` with starting values.

References

Shireman, E., Steinley, D. & Brusco, M.J. Examining the effect of initialization strategies on the performance of Gaussian mixture modeling. Behav Res 49, 282–293 (2017). <doi:10.3758/s13428-015-0697-6>

Examples

```
## Not run:
df <- iris[, 1, drop = FALSE]
names(df) <- "x"
mod <- mx_mixture(model = "x ~ m{C}*1
                           x ~~ v{C}*x",
                           classes = 2,
                           data = df,
                           run = FALSE)
mod <- mixture_starts(mod)

## End(Not run)
```

`mplus_expand_names` *Expand abbreviated Mplus variable names*

Description

Expand the Mplus syntax for abbreviating lists of variable names.

Usage

```
mplus_expand_names(x)
```

Arguments

`x`	Atomic character string containing the variable names section of an Mplus syntax file.

Value

Character vector of names.

Examples

```
mplus_expand_names("test1-test12")
```

`mx_growth_mixture` *Estimate growth mixture models using OpenMx*

Description

This function is a wrapper around `mx_mixture`, adding the default arguments of `growth` to simplify the specification of growth mixture models. This function is only useful if all the latent variables in the model are growth factors.

Usage

```
mx_growth_mixture(model, classes = 1L, data = NULL, run = TRUE, ...)
```

Arguments

`model`	Syntax for the model; either a character string, or a list of character strings, or a list of `mxModel` objects. See Details.
`classes`	A vector of integers, indicating which class solutions to generate. Defaults to 1L. E.g., `classes = 1:6`, `classes = c(1:4,6:8)`.
`data`	The data.frame to be used for model fitting.
`run`	Logical, whether or not to run the model. If `run = TRUE`, the function calls `mixture_starts` and `run_mx`.
`...`	Additional arguments, passed to functions.

Value

Returns an mxModel.

Examples

```
## Not run:
data("empathy")
df <- empathy[1:6]
mx_growth_mixture(model = "i =~ 1*ec1 + 1*ec2 + 1*ec3 +1*ec4 +1*ec5 +1*ec6
                           s =~ 0*ec1 + 1*ec2 + 2*ec3 +3*ec4 +4*ec5 +5*ec6
                           ec1 ~~ vec1*ec1
                           ec2 ~~ vec2*ec2
                           ec3 ~~ vec3*ec3
                           ec4 ~~ vec4*ec4
                           ec5 ~~ vec5*ec5
                           ec6 ~~ vec6*ec6
                           i ~~ 0*i
                           s ~~ 0*s
                           i ~~ 0*s",
                  classes = 2,
                  data = df) -> res

## End(Not run)
```

---

mx_lca	*Estimate latent class analyses using OpenMx*

Description

This function simplifies the specification of latent class models: models that estimate membership of a categorical latent variable based on binary or ordinal indicators.

Usage

```
mx_lca(data = NULL, classes = 1L, run = TRUE, ...)
```

Arguments

data	The data.frame to be used for model fitting.
classes	A vector of integers, indicating which class solutions to generate. Defaults to 1L. E.g., classes = 1:6,
run	Logical, whether or not to run the model. If run = TRUE, the function calls mxTryHardOrdinal.
...	Additional arguments, passed to functions.

Value

Returns an mxModel.

Examples

```
## Not run:
df <- data_mixture_ordinal
df[1:4] <- lapply(df, ordered)
mx_lca(data = df,
       classes = 2) -> res

## End(Not run)
```

---

mx_mixture *Estimate mixture models using OpenMx*

Description

Dynamically creates a batch of mixture models, with intelligent defaults. See Details for more information.

Usage

```
mx_mixture(model, classes = 1L, data = NULL, run = TRUE, ...)
```

Arguments

`model`	Syntax for the model; either a character string, or a list of character strings, or a list of `mxModel` objects. See Details.
`classes`	A vector of integers, indicating which class solutions to generate. Defaults to 1L. E.g., `classes = 1:6`, `classes = c(1:4,6:8)`.
`data`	The data.frame to be used for model fitting.
`run`	Logical, whether or not to run the model. If `run = TRUE`, the function calls `mixture_starts` and `run_mx`.
`...`	Additional arguments, passed to functions.

Details

Model syntax can be specified in three ways, for ease of use and flexibility:

1. An atomic character string with lavaan syntax. Within this syntax, the character string `{C}` is dynamically substituted with the correct class number using `lsub`, for example to set unique parameter labels for each class, or to specify equality constraints. E.g., `x ~ m{C}*1` will be expanded to `x ~ m1*1` and `x ~ m2*1` when `classes = 2`. The resulting syntax for each class will be converted to an `mxModel` using `as_ram`.
2. A list of character strings with lavaan syntax. Each item of the list will be converted to a class-specific `mxModel` using `as_ram`.
3. A list of `mxModel` objects, specified by the user.

Value

Returns an mxModel.

Examples

```
## Not run:
# Example 1: Dynamic model generation using {C}
df <- iris[, 1, drop = FALSE]
names(df) <- "x"
mx_mixture(model = "x ~ m{C}*1
                    x ~~ v{C}*x", classes = 1, data = df)
# Example 2: Manually specified class-specific models
df <- iris[1:2]
names(df) <- c("x", "y")
mx_mixture(model = list("y ~ a*x",
                        "y ~ b*x"),
                        meanstructure = TRUE,
                        data = df) -> res

# Example 3: Latent growth model
df <- empathy[1:6]
mx_mixture(model = "i =~ 1*ec1 + 1*ec2 + 1*ec3 +1*ec4 +1*ec5 +1*ec6
                    s =~ 0*ec1 + 1*ec2 + 2*ec3 +3*ec4 +4*ec5 +5*ec6",
                    classes = 2,
                    data = df) -> res

## End(Not run)
```

mx_profiles *Estimate latent profile analyses using OpenMx*

Description

This function is a wrapper around mx_mixture to simplify the specification of latent profile models, also known as finite mixture models. By default, the function estimates free means for all observed variables across classes.

Usage

```
mx_profiles(
  data = NULL,
  classes = 1L,
  variances = "equal",
  covariances = "zero",
  run = TRUE,
  ...
)
```

Arguments

`data`	The data.frame to be used for model fitting.
`classes`	A vector of integers, indicating which class solutions to generate. Defaults to 1L. E.g., `classes = 1:6`,
`variances`	Character vector. Specifies which variance components to estimate. Defaults to "equal" (constrain variances across classes); the other option is "varying" (estimate variances freely across classes). Each element of this vector refers to one of the models you wish to run.
`covariances`	Character vector. Specifies which covariance components to estimate. Defaults to "zero" (covariances constrained to zero; this corresponds to an assumption of conditional independence of the indicators); other options are "equal" (covariances between items constrained to be equal across classes), and "varying" (free covariances across classes). `classes = c(1:4,6:8)`.
`run`	Logical, whether or not to run the model. If `run = TRUE`, the function calls `mixture_starts` and `run_mx`.
`...`	Additional arguments, passed to functions.

Value

Returns an `mxModel`.

Examples

```
## Not run:
data("empathy")
df <- empathy[1:6]
mx_profiles(data = df,
            classes = 2) -> res

## End(Not run)
```

nodes *Extract nodes from sem_graph*

Description

Provides access to the `nodes` element of a `sem_graph` object. This can be used to return or assign to the `nodes` element.

Usage

```
nodes(x)

nodes(x) <- value
```

Arguments

`x`	Object of class sem_graph.
`value`	A valid value for `nodes(x)`.

Value

data.frame

Examples

```
edg <- data.frame(from = "x", to = "y")
p <- prepare_graph(edges = edg, layout = get_layout("x", "y", rows = 1))
nodes(p)
```

`paste2` *Concatenate Strings while omitting NA*

Description

Concatenate vectors after converting to character and removing `NA` values. See `paste`.

Usage

```
paste2(..., sep = " ", collapse = NULL, na.rm = TRUE)
```

Arguments

`...`	one or more R objects, to be converted to character vectors.
`sep`	a character string to separate the terms. Not `NA_character_`.
`collapse`	an optional character string to separate the results. Not `NA_character_`.
`na.rm`	logical, indicating whether `NA` values should be stripped before concatenation. Not `NA_character_`.

Value

A character vector of the concatenated values.

Examples

```
paste2("word", NA)
```

plot_bivariate *Create correlation plots for a mixture model*

Description

Creates a faceted plot of two-dimensional correlation plots and unidimensional density plots for a single mixture model.

Usage

```
plot_bivariate(
  x,
  variables = NULL,
  sd = TRUE,
  cors = TRUE,
  rawdata = TRUE,
  bw = FALSE,
  alpha_range = c(0, 0.1),
  return_list = FALSE
)
```

Arguments

x	An object for which a method exists.
variables	Which variables to plot. If NULL, plots all variables that are present in the model.
sd	Logical. Whether to show the estimated standard deviations as lines emanating from the cluster centroid.
cors	Logical. Whether to show the estimated correlation (standardized covariance) as ellipses surrounding the cluster centroid.
rawdata	Logical. Whether to plot raw data, weighted by posterior class probability.
bw	Logical. Whether to make a black and white plot (for print) or a color plot. Defaults to FALSE, because these density plots are hard to read in black and white.
alpha_range	Numeric vector (0-1). Sets the transparency of geom_density and geom_point.
return_list	Logical. Whether to return a list of ggplot objects, or just the final plot. Defaults to FALSE.

Value

An object of class 'ggplot'.

Author(s)

Caspar J. van Lissa

Examples

```
iris_sample <- iris[c(1:5, 145:150), c("Sepal.Length", "Sepal.Width")]
names(iris_sample) <- c("x", "y")
res <- mx_profiles(iris_sample, classes = 2)
plot_bivariate(res, rawdata = FALSE)
```

`plot_density` *Create density plots for mixture models*

Description

Creates mixture density plots. For each variable, a Total density plot will be shown, along with separate density plots for each latent class, where cases are weighted by the posterior probability of being assigned to that class.

Usage

```
plot_density(
  x,
  variables = NULL,
  bw = FALSE,
  conditional = FALSE,
  alpha = 0.2,
  facet_labels = NULL
)
```

Arguments

`x`	Object for which a method exists.
`variables`	Which variables to plot. If NULL, plots all variables that are present in all models.
`bw`	Logical. Whether to make a black and white plot (for print) or a color plot. Defaults to FALSE, because these density plots are hard to read in black and white.
`conditional`	Logical. Whether to show a conditional density plot (surface area is divided among the latent classes), or a classic density plot (surface area of the total density plot is equal to one, and is divided among the classes).
`alpha`	Numeric (0-1). Only used when bw and conditional are FALSE. Sets the transparency of geom_density, so that classes with a small number of cases remain visible.
`facet_labels`	Named character vector, the names of which should correspond to the facet labels one wishes to rename, and the values of which provide new names for these facets. For example, to rename variables, in the example with the 'iris' data below, one could specify: `facet_labels = c("Pet_leng" = "Petal length")`.

Value

An object of class 'ggplot'.

Author(s)

Caspar J. van Lissa

Examples

```
## Not run:
dat <-
  iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")]
names(dat) <- paste0("x", 1:4)
res <- mx_profiles(dat, 1:3)
plot_density(res)

## End(Not run)
```

---

plot_prob *Plot categorical variable probabilities*

Description

Creates a bar chart of categorical variable probabilities with bars reflecting the probability of category membership for each category of the observed variable.

Usage

```
plot_prob(
  x,
  variables = NULL,
  bars = c("Variable", "group", "class"),
  facet = c("group", "class", "Variable"),
  bw = FALSE,
  ...
)
```

Arguments

`x`	An object for which a method exists
`variables`	A character vectors with the names of the variables to be plotted (optional).
`bars`	Atomic character, indicating what separate bars represent. One of 'c("Variable", "group", "class")'.
`facet`	Atomic character, indicating what separate facets represent. One of 'c("group", "class", "Variable")'.
`bw`	Logical. Should the plot be black and white (for print), or color?
`...`	Arguments passed to and from other functions.

Value

An object of class 'ggplot'.

Author(s)

Caspar J. van Lissa

Examples

```
df_plot <- data.frame(Variable = rep(c("u1", "u2"), each = 3),
Category = rep(1:3, 2),
Probability = c(0.3381302605812, 0.148395173612088, 0.513474565806711,
0.472337708760608, 0.118484201496432, 0.40917808974296))
plot_prob(df_plot)
```

`plot_profiles` *Create latent profile plots*

Description

Creates a profile plot (ribbon plot) according to best practices, focusing on the visualization of classification uncertainty by showing:

1. Bars reflecting a confidence interval for the class centroids
2. Boxes reflecting the standard deviations within each class; a box encompasses +/- 64% of the observations in a normal distribution
3. Raw data, whose transparency is weighted by the posterior class probability, such that each observation is most clearly visible for the class it is most likely to be a member of.

Usage

```
plot_profiles(
  x,
  variables = NULL,
  ci = 0.95,
  sd = TRUE,
  add_line = FALSE,
  rawdata = TRUE,
  bw = FALSE,
  alpha_range = c(0, 0.1),
  ...
)

## Default S3 method:
plot_profiles(
  x,
  variables = NULL,
  ci = 0.95,
  sd = TRUE,
  add_line = FALSE,
  rawdata = TRUE,
  bw = FALSE,
  alpha_range = c(0, 0.1),
  ...
)
```

Arguments

x	An object containing the results of a mixture model analysis.
variables	A character vectors with the names of the variables to be plotted (optional).
ci	Numeric. What confidence interval should the error bars span? Defaults to a 95% confidence interval. Set to NULL to remove error bars.
sd	Logical. Whether to display a box encompassing +/- 1SD Defaults to TRUE.
add_line	Logical. Whether to display a line, connecting cluster centroids belonging to the same latent class. Defaults to FALSE, as it is not recommended to imply connectivity between the different variables on the X-axis.
rawdata	Should raw data be plotted in the background? Setting this to TRUE might result in long plotting times.
bw	Logical. Should the plot be black and white (for print), or color?
alpha_range	The minimum and maximum values of alpha (transparency) for the raw data. Minimum should be 0; lower maximum values of alpha can help reduce over-plotting.
...	Arguments passed to and from other functions.

Value

An object of class 'ggplot'.

Author(s)

Caspar J. van Lissa

Examples

```
df_plot <- data.frame(Variable = "x1",
Class = "class1",
Classes = 1,
Model = "equal var 1",
Value = 3.48571428571429,
se = 0.426092805342181,
Value.Variances = 3.81265306156537,
se.Variances = 1.17660769119959)
plot_profiles(list(df_plot = df_plot, df_raw = NULL),
ci = NULL, sd = FALSE, add_line = FALSE,
rawdata = FALSE, bw = FALSE)
```

`prepare_graph` *Prepare graph data*

Description

Prepare an object of class `sem_graph`, containing data objects that can be rendered into a SEM graph. Using this function allows users to manually change the default graph specification before plotting it. Input consists of (at least) a layout, and either nodes and edges, or a model object.

Usage

```
prepare_graph(...)

## Default S3 method:
prepare_graph(
  edges = NULL,
  layout = NULL,
  nodes = NULL,
  rect_width = 1.2,
  rect_height = 0.8,
  ellipses_width = 1,
  ellipses_height = 1,
  variance_diameter = 0.8,
  spacing_x = 2,
  spacing_y = 2,
  text_size = 4,
  curvature = 60,
  angle = NULL,
  fix_coord = FALSE,
  ...
)

## S3 method for class 'lavaan'
prepare_graph(
  model,
  edges = get_edges(x = model),
  layout = get_layout(x = model),
  nodes = get_nodes(x = model),
  ...
)

## S3 method for class 'MxModel'
prepare_graph(
  model,
  edges = get_edges(x = model),
  layout = get_layout(x = model),
  nodes = get_nodes(x = model),
  ...
)

## S3 method for class 'character'
prepare_graph(...)

## S3 method for class 'mplus.model'
prepare_graph(
  model,
  edges = get_edges(x = model),
  layout = get_layout(x = model),
  nodes = get_nodes(x = model),
  ...
)

## S3 method for class 'mplusObject'
prepare_graph(
  model,
  edges = get_edges(x = model),
  layout = get_layout(x = model),
  nodes = get_nodes(x = model),
  ...
)
```

Arguments

`...`	Additional arguments passed to and from functions.
`edges`	Object of class 'tidy_edges', or a `data.frame` with (at least) the columns `c("from","to")`, and optionally, `c("arrow","label","connect_from","connect_to","curvature")`.
`layout`	A matrix (or data.frame) that describes the layout; see `get_layout`.
`nodes`	Optional, object of class 'tidy_nodes', created with the `get_nodes` function, or a `data.frame` with (at least) the column `c("name")`, and optionally, `c("shape","label")`. If set to `NULL` (the default), nodes are inferred from the `layout` and `edges` arguments.
`rect_width`	Width of rectangles (used to display observed variables), Default: 1.2
`rect_height`	Height of rectangles (used to display observed variables), Default: 0.8
`ellipses_width`	Width of ellipses (used to display latent variables), Default: 1
`ellipses_height`	Height of ellipses (used to display latent variables), Default: 1
`variance_diameter`	Diameter of variance circles, Default: .8
`spacing_x`	Spacing between columns of the graph, Default: 1
`spacing_y`	Spacing between rows of the graph, Default: 1
`text_size`	Point size of text, Default: 4

`curvature`	Curvature of curved edges. The curve is a circle segment originating in a point that forms a triangle with the two connected points, with angles at the two connected points equal to `curvature`. To flip a curved edge, use a negative value for curvature. Default: 60
`angle`	Angle used to connect nodes by the top and bottom. Defaults to NULL, which means Euclidean distance is used to determine the shortest distance between node sides. A numeric value between 0-180 can be provided, where 0 means that only nodes with the same x-coordinates are connected top-to-bottom, and 180 means that all nodes are connected top-to-bottom.
`fix_coord`	Whether or not to fix the aspect ratio of the graph. Does not work with multi-group or multilevel models. Default: FALSE.
`model`	Instead of the edges argument, it is also possible to use the model argument and pass an object for which a method exists (e.g., `mplus.model` or `lavaan`).

Value

Object of class 'sem_graph'

Examples

```
library(lavaan)
res <- sem("dist ~ speed", cars)
prepare_graph(res)
```

`run_lavaan` *Run as lavaan model*

Description

This convenience function runs objects for which a method exists using lavaan. It is intended for use with `tidySEM`, and passes the `$syntax` and `$data` elements of a `tidy_sem` object on to `lavaan`.

Usage

```
run_lavaan(x, ...)
```

Arguments

`x`	An object for which a method exists.
`...`	Parameters passed on to other functions.

Value

Returns a lavaan object.

Examples

```
df <- iris[1:3]
names(df) <- paste0("X_", 1:3)
run_lavaan(measurement(tidy_sem(df), meanstructure = TRUE))
```

---

run_mx *Run as OpenMx model with sensible defaults*

Description

This convenience function runs objects for which a method exists using OpenMx, with sensible defaults. It is intended for use with `tidySEM`. For instance, it will convert a `tidySEM` object to a `mxModel` and run it, and it will try to ensure convergence for mixture models created using `mx_mixture`. Knowledgeable users may want to run models manually.

Usage

```
run_mx(x, ...)
```

Arguments

`x`	An object for which a method exists.
`...`	Parameters passed on to other functions.

Value

Returns an `mxModel` with free parameters updated to their final values.

Examples

```
df <- iris[1:3]
names(df) <- paste0("X_", 1:3)
run_mx(measurement(tidy_sem(df), meanstructure = TRUE))
```

---

skew_kurtosis *Calculate skew and kurtosis*

Description

Calculate skew and kurtosis, standard errors for both, and the estimates divided by two times the standard error. If this latter quantity exceeds an absolute value of 1, the skew/kurtosis is significant. With very large sample sizes, significant skew/kurtosis is common.

Usage

```
skew_kurtosis(x, verbose = FALSE, se = FALSE, ...)
```

Arguments

`x`	An object for which a method exists.
`verbose`	Logical. Whether or not to print messages to the console, Default: FALSE
`se`	Whether or not to return the standard errors, Default: FALSE
`...`	Additional arguments to pass to and from functions.

Value

A `matrix` of skew and kurtosis statistics for `x`.

Examples

```
skew_kurtosis(datasets::anscombe)
```

`syntax` *Extract syntax from tidy_sem*

Description

Provides access to the `syntax` element of a `tidy_sem` object. This can be used to return or assign to the `syntax` element.

Usage

```
syntax(x)

syntax(x) <- value
```

Arguments

`x`	Object of class tidy_sem.
`value`	A valid value for `syntax(x)`.

Value

data.frame

Examples

```
dict <- tidy_sem(iris, split = "\\.")
dict <- add_paths(dict, Sepal.Width ~~ Sepal.Length)
syntax(dict)
```

table_cors *Extract correlation tables*

Description

Extracts a publication-ready covariance or correlation matrix from an object for which a method exists.

Usage

```
table_cors(x, value_column = "est_sig_std", digits = 2, ...)
```

Arguments

x	An object for which a method exists.
value_column	Character. Name of the column to use to propagate the matrix. Defaults to "est_sig_std", the standardized estimate with significance asterisks.
digits	Number of digits to round to when formatting values.
...	Additional arguments passed to and from methods.

Value

A Matrix or a list of matrices (in case there are between/within correlation matrices).

Author(s)

Caspar J. van Lissa

Examples

```
library(lavaan)
HS.model <- '  visual =~ x1 + x2 + x3
               textual =~ x4 + x5 + x6
               speed   =~ x7 + x8 + x9 '
fit <- cfa(HS.model,
           data = HolzingerSwineford1939,
           group = "school")
table_cors(fit)
```

`table_fit` *Print model fit table formatted for publication*

Description

Takes a model object, extracts model fit information, and formats it as a publication-ready table.

Usage

```
table_fit(x, ...)
```

Arguments

`x`	A model object for which a method exists.
`...`	Arguments passed to other functions.

Value

A data.frame of formatted results.

Author(s)

Caspar J. van Lissa

See Also

Other Reporting tools: `conf_int()`, `est_sig()`, `table_prob()`, `table_results()`

Examples

```
library(lavaan)
HS.model <- '  visual =~ x1 + x2 + x3
               textual =~ x4 + x5 + x6
               speed   =~ x7 + x8 + x9 '
fit <- cfa(HS.model,
           data = HolzingerSwineford1939,
           group = "school")
table_fit(fit)
```

table_prob *Results table in probability scale*

Description

Returns thresholds for ordinal dependent variables in probability scale.

Usage

```
table_prob(x, ...)
```

Arguments

x	An object for which a method exists.
...	Arguments passed to other functions.

Value

A data.frame with results in probability scale.

See Also

Other Reporting tools: conf_int(), est_sig(), table_fit(), table_results()

Examples

```
## Not run:
df <- data_mix_ordinal
df[1:4] <- lapply(df, ordered)
mx_lca(data = df,
       classes = 2) -> res

## End(Not run)
```

table_results *Print results table formatted for publication*

Description

Takes a model object, and formats it as a publication-ready table.

Usage

```
table_results(
  x,
  columns = c("label", "est_sig", "se", "pval", "confint", "group", "level"),
  digits = 2,
  ...
)
```

Arguments

`x`	A model object for which a method exists.
`columns`	A character vector of columns to retain from the results section. If this is set to `NULL`, all available columns are returned. Defaults to `c("label","est_sig","se","pval","confint",` These correspond to 1) the parameter label, 2) estimate column with significance asterisks appended (* <.05, ** < .01, *** < .001); 3) standard error, 4) p-value, 5) a formatted confidence interval, 6) grouping variable (if available), 7) level variable for multilevel models, if available.
`digits`	Number of digits to round to when formatting numeric columns.
`...`	Logical expressions used to filter the rows of results returned.

Value

A data.frame of formatted results.

Author(s)

Caspar J. van Lissa

See Also

Other Reporting tools: `conf_int()`, `est_sig()`, `table_fit()`, `table_prob()`

Examples

```
library(lavaan)
HS.model <- '  visual =~ x1 + x2 + x3
               textual =~ x4 + x5 + x6
               speed   =~ x7 + x8 + x9 '
fit <- cfa(HS.model,
           data = HolzingerSwineford1939,
           group = "school")
table_results(fit)
```

`tidy_sem` *Create a tidy_sem object*

Description

Create an object of class `tidy_sem`, which has the following elements:

- dictionary An overview of the variables in the `tidy_sem` object, and their assignment to scale/latent variables.
- data Optionally, the `data.frame` containing the data referenced in `$dictionary`.
- syntax Optionally, syntax defining a SEM-model by reference to the variables contained in `$data`.

Usage

```
tidy_sem(x, split = "_")
```

Arguments

`x`	An object for which a method exists, e.g., a vector of variable names, or a data.frame.
`split`	Character. Defining the regular expression used by `strsplit` to separate variable names into 1) the name of the scale/construct and 2) the number (or name) of the item.

Details

When `tidy_sem` is called on a `character` string or `data.frame`, it attempts to assign variables to superordinate scale/latent variables based on the variable name and the splitting character defined in the `split` argument. Thus, the function will assign the variable `"scale_01"` to a scale/latent variable called `"scale"` when `split = "_"`. Alternatively, if the variable name is `"construct.1"`, the split character `"\."` separates the `"construct"` name from item number `"1"`. The character `"."` is escaped with a double backslash, because it is a special character in regular expressions.

Value

An object of class "tidy_sem"

Author(s)

Caspar J. van Lissa

Examples

```
tidy_sem(c("bfi_1", "bfi_2", "bfi_3", "bfi_4", "bfi_5",
"macqj_1", "macqj_2", "macqj_3", "macqj_4", "macqj_5", "macqj_6",
"macqj_7", "macqj_8", "macqj_9", "macqj_10", "macqj_11",
"macqj_12", "macqj_13", "macqj_14", "macqj_15", "macqj_16",
"macqj_17", "macqj_18", "macqj_19", "macqj_20", "macqj_21",
"macqr_1", "macqr_2", "macqr_3", "macqr_4", "macqr_5", "macqr_6",
"macqr_7", "macqr_8", "macqr_9", "macqr_10", "macqr_11",
"macqr_12", "macqr_13", "macqr_14", "macqr_15", "macqr_16",
"macqr_17", "macqr_18", "macqr_19", "macqr_20", "macqr_21", "sex"))
tidy_sem(c("bfi_1", "bfi_2", "bfi_3", "bfi_4", "bfi_5",
"mac_q_j_1", "mac_q_j_2", "mac_q_j_3", "mac_q_j_4", "mac_q_j_5", "mac_q_j_6",
"mac_q_j_7", "mac_q_j_8", "mac_q_j_9", "mac_q_j_10", "mac_q_j_11",
"mac_q_j_12", "mac_q_j_13", "mac_q_j_14", "mac_q_j_15", "mac_q_j_16",
"mac_q_j_17", "mac_q_j_18", "mac_q_j_19", "mac_q_j_20", "mac_q_j_21",
"mac_q_r_1", "mac_q_r_2", "mac_q_r_3", "mac_q_r_4", "mac_q_r_5", "mac_q_r_6",
"mac_q_r_7", "mac_q_r_8", "mac_q_r_9", "mac_q_r_10", "mac_q_r_11",
"mac_q_r_12", "mac_q_r_13", "mac_q_r_14", "mac_q_r_15", "mac_q_r_16",
"mac_q_r_17", "mac_q_r_18", "mac_q_r_19", "mac_q_r_20", "mac_q_r_21"))
```

Index