

Package ‘ggdag’

September 12, 2019

Title Analyze and Create Elegant Directed Acyclic Graphs

Version 0.2.0

Description Tidy, analyze, and plot directed acyclic graphs (DAGs). 'ggdag' is built on top of 'dagitty', an R package that uses the 'DAGitty' web tool (<<http://dagitty.net>>) for creating and analyzing DAGs. 'ggdag' makes it easy to tidy and plot 'dagitty' objects using 'ggplot2' and 'ggraph', as well as common analytic and graphical functions, such as determining adjustment sets and node relationships.

License MIT + file LICENSE

URL <https://github.com/malcolmbarrett/ggdag>

BugReports <https://github.com/malcolmbarrett/ggdag/issues>

Depends ggplot2 (>= 3.0.0), R (>= 3.2.0)

Imports dagitty, dplyr, forcats, ggforce, ggraph (>= 2.0.0), ggrepel, igrph, magrittr, pillar, plyr, purrr, stringr, tibble, tidygraph

Suggests knitr, rmarkdown, spelling, testthat

VignetteBuilder knitr

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Author Malcolm Barrett [aut, cre] (<<https://orcid.org/0000-0003-0299-5825>>)

Maintainer Malcolm Barrett <malcolmbarrett@gmail.com>

Repository CRAN

Date/Publication 2019-09-12 05:30:03 UTC

R topics documented:

activate_collider_paths	3
Adjust for variables	3
as.data.frame.tidy_dagitty	5
as.tbl.tidy_dagitty	5
Assess d-separation between variables	6
Assess familial relationships between variables	8
as_tbl_graph	11
Canonicalize DAGs	12
Colliders	13
coordinates	14
Covariate Adjustment Sets	15
dag	16
DAG Edges	17
DAG Labels	20
dagify	21
dplyr	22
Equivalent DAGs and Classes	24
Exogenous Variables	26
expand_plot	27
expand_scale	27
fortify	28
geom_dag_collider_edges	29
geom_dag_edges	30
geom_dag_text	32
ggdag	34
ggdag_classic	35
ggplot.tidy_dagitty	36
ggrepel functions	37
Instrumental Variables	39
is.tidy_dagitty	40
is_confounder	41
Nodes	41
Pathways	43
print.tidy_dagitty	44
Quick Plots for Common DAGs	45
remove_axes	46
scale_adjusted	47
simulate_data	48
tbl_df.tidy_dagitty	49
Test if Variable Is Collider	49
theme_dag_blank	50
theme_dag_grey	51
tidy_dagitty	51
Variable Status	52

`activate_collider_paths`*Activate paths opened by stratifying on a collider*

Description

Stratifying on colliders can open biasing pathways between variables. `activate_collider_paths` activates any such pathways given a variable or set of variables to adjust for and adds them to the `tidy_dagitty`.

Usage

```
activate_collider_paths(.tidy_dag, adjust_for, ...)
```

Arguments

<code>.tidy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>adjust_for</code>	a character vector, the variable(s) to adjust for.
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>

Value

a `tidy_dagitty` with additional rows for collider-activated pathways

See Also

[control_for\(\)](#), [ggdag_adjust\(\)](#), [geom_dag_collider_edges\(\)](#)

Examples

```
dag <- dagify(m ~ x + y, x ~ y)

collided_dag <- activate_collider_paths(dag, adjust_for = "m")
collided_dag
```

Adjust for variables *Adjust for variables and activate any biasing paths that result*

Description

Adjust for variables and activate any biasing paths that result

Usage

```
control_for(.tdy_dag, var, as_factor = TRUE, ...)

adjust_for(.tdy_dag, var, as_factor = TRUE, ...)

ggdag_adjust(.tdy_dag, var = NULL, ..., node_size = 16,
  text_size = 3.88, label_size = text_size, text_col = "white",
  label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
  use_labels = NULL, collider_lines = TRUE)
```

Arguments

<code>.tdy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>var</code>	a character vector, the variable(s) to adjust for.
<code>as_factor</code>	logical. Should the adjusted column be a factor?
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>node_size</code>	size of DAG node
<code>text_size</code>	size of DAG text
<code>label_size</code>	size of label text
<code>text_col</code>	color of DAG text
<code>label_col</code>	color of label text
<code>node</code>	logical. Should nodes be included in the DAG?
<code>stylized</code>	logical. Should DAG nodes be stylized? If so, use <code>geom_dag_nodes</code> and if not use <code>geom_dag_point</code>
<code>text</code>	logical. Should text be included in the DAG?
<code>use_labels</code>	a string. Variable to use for <code>geom_dag_repel_label()</code> . Default is <code>NULL</code> .
<code>collider_lines</code>	logical. Should the plot show paths activated by adjusting for a collider?

Value

a `tidy_dagitty` with a adjusted column for adjusted variables, as well as any biasing paths that arise, or a `ggplot`

Examples

```
dag <- dagify(m ~ a + b, x ~ a, y ~ b)

control_for(dag, var = "m")
ggdag_adjust(dag, var = "m")
```

```
as.data.frame.tidy_dagitty
  Convert a tidy_dagitty object to data.frame
```

Description

Convert a tidy_dagitty object to data.frame

Usage

```
## S3 method for class 'tidy_dagitty'
as.data.frame(x, row.names, optional, ...)
```

Arguments

x	an object of class tidy_dagitty
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional. Note that all of R's base package as.data.frame() methods use optional only for column names treatment, basically with the meaning of data.frame(*, check.names = !optional)
...	optional arguments passed to as.data.frame()

```
as.tbl.tidy_dagitty  Convert a tidy_dagitty object to tbl
```

Description

Convert a tidy_dagitty object to tbl

Usage

```
## S3 method for class 'tidy_dagitty'
as.tbl(x, row.names = NULL, optional = FALSE,
  ...)
```

Arguments

x	an object of class tidy_dagitty
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.

optional	logical. If TRUE, setting row names and converting column names (to syntactic names: see <code>make.names</code>) is optional. Note that all of R's base package <code>as.data.frame()</code> methods use optional only for column names treatment, basically with the meaning of <code>data.frame(*, check.names = !optional)</code>
...	optional arguments passed to <code>as.tbl()</code>

Assess d-separation between variables

D-relationship between variables

Description

D-separation is a key concept in causal structural models. Variables are d-separated if there are no open paths between them. The `node_d*`() functions label variables as d-connected or d-separated. The `ggdag_d*`() functions plot the results. The `*_dconnected()`, `*_dseparated()`, and `*_drelationship()` functions essentially produce the same output and are just different ways of thinking about the relationship. See `dagitty::dseparated()` for details.

Usage

```
node_dconnected(.tdy_dag, from = NULL, to = NULL,
  controlling_for = NULL, as_factor = TRUE, ...)

node_dseparated(.tdy_dag, from = NULL, to = NULL,
  controlling_for = NULL, as_factor = TRUE)

node_drelationship(.tdy_dag, from = NULL, to = NULL,
  controlling_for = NULL, as_factor = TRUE)

ggdag_drelationship(.tdy_dag, from = NULL, to = NULL,
  controlling_for = NULL, ..., edge_type = "link_arc",
  node_size = 16, text_size = 3.88, label_size = text_size,
  text_col = "white", label_col = text_col, node = TRUE,
  stylized = FALSE, text = TRUE, use_labels = NULL,
  collider_lines = TRUE)

ggdag_dseparated(.tdy_dag, from = NULL, to = NULL,
  controlling_for = NULL, ..., edge_type = "link_arc",
  node_size = 16, text_size = 3.88, label_size = text_size,
  text_col = "white", label_col = text_col, node = TRUE,
  stylized = FALSE, text = TRUE, use_labels = NULL,
  collider_lines = TRUE)

ggdag_dconnected(.tdy_dag, from = NULL, to = NULL,
  controlling_for = NULL, ..., edge_type = "link_arc",
  node_size = 16, text_size = 3.88, label_size = text_size,
  text_col = "white", label_col = text_col, node = TRUE,
```

```
stylized = FALSE, text = TRUE, use_labels = NULL,
collider_lines = TRUE)
```

Arguments

<code>.tidy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>from</code>	a character vector, the starting variable (must be in DAG). If <code>NULL</code> , checks DAG for exposure variable.
<code>to</code>	a character vector, the ending variable (must be in DAG). If <code>NULL</code> , checks DAG for outcome variable.
<code>controlling_for</code>	a character vector, variables in the DAG to control for.
<code>as_factor</code>	logical. Should the <code>d_relationship</code> variable be a factor?
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>edge_type</code>	a character vector, the edge geom to use. One of: "link_arc", which accounts for directed and bidirected edges, "link", "arc", or "diagonal"
<code>node_size</code>	size of DAG node
<code>text_size</code>	size of DAG text
<code>label_size</code>	size of label text
<code>text_col</code>	color of DAG text
<code>label_col</code>	color of label text
<code>node</code>	logical. Should nodes be included in the DAG?
<code>stylized</code>	logical. Should DAG nodes be stylized? If so, use <code>geom_dag_nodes</code> and if not use <code>geom_dag_point</code>
<code>text</code>	logical. Should text be included in the DAG?
<code>use_labels</code>	a string. Variable to use for <code>geom_dag_repel_label()</code> . Default is <code>NULL</code> .
<code>collider_lines</code>	logical. Should the plot show paths activated by adjusting for a collider?

Value

a `tidy_dagitty` with a `d_relationship` column for variable D relationship or a `ggplot`

Examples

```
dag <- dagify(m ~ x + y)
dag %>% ggdag_drelationship("x", "y")
dag %>% ggdag_drelationship("x", "y", controlling_for = "m")

dag %>%
  node_dseparated("x", "y") %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, shape = adjusted, col = d_relationship)) +
  geom_dag_edges() +
  geom_dag_collider_edges() +
  geom_dag_node() +
  geom_dag_text(col = "white") +
```

```

  theme_dag() + scale_adjusted()

dag %>%
  node_dconnected("x", "y", controlling_for = "m") %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, shape = adjusted, col = d_relationship)) +
  geom_dag_edges() +
  geom_dag_collider_edges() +
  geom_dag_node() +
  geom_dag_text(col = "white") +
  theme_dag() +
  scale_adjusted()

dagify(m ~ x + y, m_jr ~ m) %>%
  tidy_dagitty(layout = "nicely") %>%
  node_dconnected("x", "y", controlling_for = "m_jr") %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, shape = adjusted, col = d_relationship)) +
  geom_dag_edges() +
  geom_dag_collider_edges() +
  geom_dag_node() +
  geom_dag_text(col = "white") +
  theme_dag() +
  scale_adjusted()

```

Assess familial relationships between variables

Familial relationships between variables

Description

Parents and children are those nodes that either directly cause or are caused by the variable, respectively. Ancestors and descendants are those nodes that are on the path to or descend from the variable. The `node_*()` functions label variables depending on their relationship. The `ggdag_*()` functions plot the results. See [dagitty::children](#) for details.

Usage

```

node_children(.tdy_dag, .var, as_factor = TRUE)

node_parents(.tdy_dag, .var, as_factor = TRUE)

node_ancestors(.tdy_dag, .var, as_factor = TRUE)

node_descendants(.tdy_dag, .var, as_factor = TRUE)

node_markov_blanket(.tdy_dag, .var, as_factor = TRUE)

node_adjacent(.tdy_dag, .var, as_factor = TRUE)

ggdag_children(.tdy_dag, .var, ..., edge_type = "link_arc",

```



```

node_size = 16, text_size = 3.88, label_size = text_size,
text_col = "white", label_col = text_col, node = TRUE,
stylized = FALSE, text = TRUE, use_labels = NULL)

ggdag_parents(.tdy_dag, .var, ..., edge_type = "link_arc",
node_size = 16, text_size = 3.88, label_size = text_size,
text_col = "white", label_col = text_col, node = TRUE,
stylized = FALSE, text = TRUE, use_labels = NULL)

ggdag_ancestors(.tdy_dag, .var, ..., edge_type = "link_arc",
node_size = 16, text_size = 3.88, label_size = text_size,
text_col = "white", label_col = text_col, node = TRUE,
stylized = FALSE, text = TRUE, use_labels = NULL)

ggdag_descendants(.tdy_dag, .var, ..., edge_type = "link_arc",
node_size = 16, text_size = 3.88, label_size = text_size,
text_col = "white", label_col = text_col, node = TRUE,
stylized = FALSE, text = TRUE, use_labels = NULL)

ggdag_markov_blanket(.tdy_dag, .var, ..., edge_type = "link_arc",
node_size = 16, text_size = 3.88, label_size = text_size,
text_col = "white", label_col = text_col, node = TRUE,
stylized = FALSE, text = TRUE, use_labels = NULL)

ggdag_adjacent(.tdy_dag, .var, ..., edge_type = "link_arc",
node_size = 16, text_size = 3.88, label_size = text_size,
text_col = "white", label_col = text_col, node = TRUE,
stylized = FALSE, text = TRUE, use_labels = NULL)

```

Arguments

<code>.tdy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>.var</code>	a character vector, the variable to be assessed (must be in DAG)
<code>as_factor</code>	logical. Should the relationship variable be a factor?
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>edge_type</code>	a character vector, the edge geom to use. One of: "link_arc", which accounts for directed and bidirected edges, "link", "arc", or "diagonal"
<code>node_size</code>	size of DAG node
<code>text_size</code>	size of DAG text
<code>label_size</code>	size of label text
<code>text_col</code>	color of DAG text
<code>label_col</code>	color of label text
<code>node</code>	logical. Should nodes be included in the DAG?
<code>stylized</code>	logical. Should DAG nodes be stylized? If so, use <code>geom_dag_nodes</code> and if not use <code>geom_dag_point</code>

text logical. Should text be included in the DAG?

use_labels a string. Variable to use for `geom_dag_repel_label()`. Default is NULL.

Value

a `tidy_dagitty` with an column related to the given relationship for variable D relationship or a `ggplot`

Examples

```
dag <- dagify(y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  w1 ~~ w2)

ggdag_children(dag, "w1")

dag %>%
  node_children("w1") %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, color = children)) +
  geom_dag_edges() +
  geom_dag_node() +
  geom_dag_text(col = "white") +
  geom_dag_label_repel(aes(label = children, fill = children), col = "white", show.legend = FALSE) +
  theme_dag() +
  scale_adjusted() +
  scale_color_hue(breaks = c("parent", "child"))

ggdag_parents(dag, "y")

ggdag_ancestors(dag, "x")

ggdag_descendants(dag, "w1")

dag %>%
  node_parents("y") %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, color = parent)) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text(col = "white") +
  geom_dag_label_repel(aes(label = parent, fill = parent), col = "white", show.legend = FALSE) +
  theme_dag() +
  scale_adjusted() +
  scale_color_hue(breaks = c("parent", "child"))
```

as_tbl_graph	<i>Convert DAGS to tidygraph</i>
--------------	----------------------------------

Description

A thin wrapper to convert `tidy_dagitty` and `dagitty` objects to `tbl_graph`, which can then be used to work in `tidygraph` and `ggraph` directly. See [tidygraph::as_tbl_graph\(\)](#).

Usage

```
## S3 method for class 'tidy_dagitty'  
as_tbl_graph(x, directed = TRUE, ...)
```

```
## S3 method for class 'dagitty'  
as_tbl_graph(x, directed = TRUE, ...)
```

Arguments

<code>x</code>	an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>directed</code>	logical. Should the constructed graph be directed? Default is <code>TRUE</code>
<code>...</code>	other arguments passed to <code>as_tbl_graph</code>

Value

a `tbl_graph`

Examples

```
library(ggraph)  
library(tidygraph)  
butterfly_bias() %>%  
  as_tbl_graph() %>%  
  ggraph() +  
    geom_edge_diagonal() +  
    geom_node_point()
```

Description

Takes an input graph with bidirected edges and replaces every bidirected edge $x \leftrightarrow y$ with a sub-structure $x \leftarrow L \rightarrow y$, where L is a latent variable. See `dagitty::canonicalize()` for details. Undirected edges are not currently supported in `ggdag`.

Usage

```
node_canonical(.dag, ...)

ggdag_canonical(.tidy_dag, ..., edge_type = "link_arc", node_size = 16,
  text_size = 3.88, label_size = text_size, text_col = "white",
  label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
  use_labels = NULL)
```

Arguments

<code>.dag</code> , <code>.tidy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>edge_type</code>	a character vector, the edge geom to use. One of: "link_arc", which accounts for directed and bidirected edges, "link", "arc", or "diagonal"
<code>node_size</code>	size of DAG node
<code>text_size</code>	size of DAG text
<code>label_size</code>	size of label text
<code>text_col</code>	color of DAG text
<code>label_col</code>	color of label text
<code>node</code>	logical. Should nodes be included in the DAG?
<code>stylized</code>	logical. Should DAG nodes be stylized? If so, use <code>geom_dag_nodes</code> and if not use <code>geom_dag_point</code>
<code>text</code>	logical. Should text be included in the DAG?
<code>use_labels</code>	a string. Variable to use for <code>geom_dag_repel_label()</code> . Default is <code>NULL</code> .

Value

a `tidy_dagitty` that includes L or a `ggplot`

Examples

```
dag <- dagify(y ~ x + z, x ~~ z)

ggdag(dag)

node_canonical(dag)
ggdag_canonical(dag)
```

Colliders

Find colliders

Description

Detects any colliders given a DAG. `node_collider` tags colliders and `ggdag_collider` plots all exogenous variables.

Usage

```
node_collider(.dag, as_factor = TRUE, ...)

ggdag_collider(.tdy_dag, ..., edge_type = "link_arc", node_size = 16,
  text_size = 3.88, label_size = text_size, text_col = "white",
  label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
  use_labels = NULL)
```

Arguments

<code>.dag</code> , <code>.tdy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>as_factor</code>	treat collider variable as factor
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>edge_type</code>	a character vector, the edge geom to use. One of: "link_arc", which accounts for directed and bidirected edges, "link", "arc", or "diagonal"
<code>node_size</code>	size of DAG node
<code>text_size</code>	size of DAG text
<code>label_size</code>	size of label text
<code>text_col</code>	color of DAG text
<code>label_col</code>	color of label text
<code>node</code>	logical. Should nodes be included in the DAG?
<code>stylized</code>	logical. Should DAG nodes be stylized? If so, use <code>geom_dag_nodes</code> and if not use <code>geom_dag_point</code>
<code>text</code>	logical. Should text be included in the DAG?
<code>use_labels</code>	a string. Variable to use for <code>geom_dag_repel_label()</code> . Default is NULL.

Value

a tidy_dagitty with a collider column for colliders or a ggplot

Examples

```
dag <- dagify(m ~ x + y, y ~ x)

node_collider(dag)
ggdag_collider(dag)
```

coordinates

Manipulate DAG coordinates

Description

Manipulate DAG coordinates

Usage

```
coords2df(coord_list)

coords2list(coord_df)
```

Arguments

`coord_list` a named list of coordinates
`coord_df` a data.frame with columns x, y, and name

Value

either a list or a data.frame with DAG node coordinates

Examples

```
library(dagitty)
coords <- list(
  x = c(A = 1, B = 2, D = 3, C = 3, F = 3, E = 4, G = 5, H = 5, I = 5),
  y = c(A = 0, B = 0, D = 1, C = 0, F = -1, E = 0, G = 1, H = 0, I = -1)
)
coord_df <- coords2df(coords)
coords2list(coord_df)

x <- dagitty('dag{
  G <-> H <-> I <-> G
  D <- B -> C -> I <- F <- B <- A
  H <- E <- C -> G <- D
}')
coordinates(x) <- coords2list(coord_df)
```

 Covariate Adjustment Sets

Covariate Adjustment Sets

Description

See [dagitty::adjustmentSets\(\)](#) for details.

Usage

```
dag_adjustment_sets(.tidy_dag, exposure = NULL, outcome = NULL, ...)

ggdag_adjustment_set(.tidy_dag, exposure = NULL, outcome = NULL, ...,
  shadow = FALSE, node_size = 16, text_size = 3.88,
  label_size = text_size, text_col = "white", label_col = text_col,
  node = TRUE, stylized = FALSE, text = TRUE, use_labels = NULL,
  expand_x = expand_scale(c(0.25, 0.25)),
  expand_y = expand_scale(c(0.2, 0.2)))
```

Arguments

<code>.tidy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>exposure</code>	a character vector, the exposure variable. Default is <code>NULL</code> , in which case it will be determined from the DAG.
<code>outcome</code>	a character vector, the outcome variable. Default is <code>NULL</code> , in which case it will be determined from the DAG.
<code>...</code>	additional arguments to <code>adjustmentSets</code>
<code>shadow</code>	logical. Show paths blocked by adjustment?
<code>node_size</code>	size of DAG node
<code>text_size</code>	size of DAG text
<code>label_size</code>	size of label text
<code>text_col</code>	color of DAG text
<code>label_col</code>	color of label text
<code>node</code>	logical. Should nodes be included in the DAG?
<code>stylized</code>	logical. Should DAG nodes be stylized? If so, use <code>geom_dag_nodes</code> and if not use <code>geom_dag_point</code>
<code>text</code>	logical. Should text be included in the DAG?
<code>use_labels</code>	a string. Variable to use for <code>geom_dag_repel_label()</code> . Default is <code>NULL</code> .
<code>expand_x, expand_y</code>	Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function <code>expand_scale()</code> to generate the values for the <code>expand_argument</code> .

Value

a `tidy_dagitty` with an adjusted column and set column, indicating adjustment status and DAG ID, respectively, for the adjustment sets or a `ggplot`

Examples

```
dag <- dagify(y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  w1 ~~ w2,
  exposure = "x",
  outcome = "y")

tidy_dagitty(dag) %>% dag_adjustment_sets()

ggdag_adjustment_set(dag)

ggdag_adjustment_set(dagitty::randomDAG(10, .5),
  exposure = "x3",
  outcome = "x5")
```

 dag

Create a dagitty DAG

Description

A convenience wrapper for `dagitty::dagitty("dag...")`

Usage

```
dag(...)
```

Arguments

... a character vector in the style of `dagitty`. See `dagitty::dagitty` for details.

Value

a `dagitty`

Examples

```
dag("{x m} -> y")
```


DAG Edges

*Directed DAG edges***Description**

Directed DAG edges

Usage

```
geom_dag_edges_link(mapping = NULL, data = NULL,
  arrow = grid::arrow(length = grid::unit(5, "pt"), type = "closed"),
  position = "identity", na.rm = TRUE, show.legend = NA,
  inherit.aes = TRUE, ...)
```

```
geom_dag_edges_arc(mapping = NULL, data = NULL, curvature = 0.5,
  arrow = grid::arrow(length = grid::unit(5, "pt"), type = "closed"),
  position = "identity", na.rm = TRUE, show.legend = NA,
  inherit.aes = TRUE, fold = FALSE, n = 100, lineend = "butt",
  linejoin = "round", linemitre = 1, label_colour = "black",
  label_alpha = 1, label_parse = FALSE, check_overlap = FALSE,
  angle_calc = "rot", force_flip = TRUE, label_dodge = NULL,
  label_push = NULL, ...)
```

```
geom_dag_edges_diagonal(mapping = NULL, data = NULL,
  position = "identity", arrow = grid::arrow(length = grid::unit(5,
  "pt"), type = "closed"), na.rm = TRUE, show.legend = NA,
  inherit.aes = TRUE, curvature = 1, n = 100, lineend = "butt",
  linejoin = "round", linemitre = 1, label_colour = "black",
  label_alpha = 1, label_parse = FALSE, check_overlap = FALSE,
  angle_calc = "rot", force_flip = TRUE, label_dodge = NULL,
  label_push = NULL, ...)
```

```
geom_dag_edges_fan(mapping = NULL, data = NULL,
  position = "identity", arrow = grid::arrow(length = grid::unit(5,
  "pt"), type = "closed"), na.rm = TRUE, show.legend = NA,
  inherit.aes = TRUE, spread = 0.7, n = 100, lineend = "butt",
  linejoin = "round", linemitre = 1, label_colour = "black",
  label_alpha = 1, label_parse = FALSE, check_overlap = FALSE,
  angle_calc = "rot", force_flip = TRUE, label_dodge = NULL,
  label_push = NULL, ...)
```

Arguments

mapping Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

<code>data</code>	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> ., and will be used as the layer data.
<code>arrow</code>	specification for arrow heads, as created by <code>arrow()</code>
<code>position</code>	Position adjustment, either as a string, or the result of a call to a position adjustment function.
<code>na.rm</code>	If <code>FALSE</code> (the default), removes missing values with a warning. If <code>TRUE</code> silently removes missing values
<code>show.legend</code>	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
<code>inherit.aes</code>	If <code>FALSE</code> , overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders()</code> .
<code>...</code>	Other arguments passed to <code>ggraph::geom_edge_*</code> ()
<code>curvature</code>	The bend of the curve. 1 approximates a halfcircle while 0 will give a straight line. Negative number will change the direction of the curve. Only used if <code>layout_circular = FALSE</code> .
<code>fold</code>	Logical. Should arcs appear on the same side of the nodes despite different directions. Default to <code>FALSE</code> .
<code>n</code>	The number of points to create along the path.
<code>lineend</code>	Line end style (round, butt, square).
<code>linejoin</code>	Line join style (round, mitre, bevel).
<code>linemitre</code>	Line mitre limit (number greater than 1).
<code>label_colour</code>	The colour of the edge label. If <code>NA</code> it will use the colour of the edge.
<code>label_alpha</code>	The opacity of the edge label. If <code>NA</code> it will use the opacity of the edge.
<code>label_parse</code>	If <code>TRUE</code> , the labels will be parsed into expressions and displayed as described in <code>grDevices::plotmath()</code> .
<code>check_overlap</code>	If <code>TRUE</code> , text that overlaps previous text in the same layer will not be plotted.
<code>angle_calc</code>	Either <code>'none'</code> , <code>'along'</code> , or <code>'across'</code> . If <code>'none'</code> the label will use the angle aesthetic of the geom. If <code>'along'</code> The label will be written along the edge direction. If <code>'across'</code> the label will be written across the edge direction.
<code>force_flip</code>	Logical. If <code>angle_calc</code> is either <code>'along'</code> or <code>'across'</code> should the label be flipped if it is on it's head. Default to <code>TRUE</code> .
<code>label_dodge</code>	A <code>grid::unit()</code> giving a fixed vertical shift to add to the label in case of <code>angle_calc</code> is either <code>'along'</code> or <code>'across'</code>
<code>label_push</code>	A <code>grid::unit()</code> giving a fixed horizontal shift to add to the label in case of <code>angle_calc</code> is either <code>'along'</code> or <code>'across'</code>
<code>spread</code>	Deprecated. Use <code>strength</code> instead.

Aesthetics

`geom_dag_edges_link`, `geom_dag_edges_arc`, `geom_dag_edges_diagonal`, and `geom_dag_edges_fan` understand the following aesthetics. Bold aesthetics are required.

- **x**
- **y**
- **xend**
- **yend**
- `edge_colour`
- `edge_width`
- `edge_linetype`
- `edge_alpha`
- `start_cap`
- `end_cap`
- `label`
- `label_pos`
- `label_size`
- `angle`
- `hjust`
- `vjust`
- `family`
- `fontface`
- `lineheight`

`geom_dag_edges_arc` and `geom_dag_edges_diagonal` also require **circular**, but this is automatically set.

`geom_dag_edges_fan` requires **to** and **from**, but these are also automatically set.

Examples

```
p <- dagify(y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  L ~ w1 + w2) %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_dag_point() +
  geom_dag_text() +
  theme_dag()

p + geom_dag_edges_link()
p + geom_dag_edges_arc()
p + geom_dag_edges_diagonal()
```

Description

Label or otherwise retrieve labels from objects of either class `tidy_dagitty` or `dagitty`

Usage

```
label(x) <- value

## S3 replacement method for class 'dagitty'
label(x) <- value

## S3 replacement method for class 'tidy_dagitty'
label(x) <- value

dag_label(.tdy_dag, labels = NULL)

label(.tdy_dag)

has_labels(.tdy_dag)
```

Arguments

<code>x</code>	an object of either class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>value</code>	a character vector
<code>.tdy_dag</code>	an object of class <code>tidy_dagitty</code>
<code>labels</code>	a character vector

Value

`label` returns the label attribute of `x`

Examples

```
labelled_dag <- dagify(y ~ z, x ~ z) %>%
  tidy_dagitty() %>%
  dag_label(labels = c("x" = "exposure", "y" = "outcome", "z" = "confounder"))

has_labels(labelled_dag)
```

dagify *Create a dagitty DAG using R-like syntax*

Description

dagify() creates dagitty DAGs using a more R-like syntax. It currently accepts formulas in the usual R style, e.g. $y \sim x + z$, which gets translated to $y \leftarrow \{x\ z\}$, as well as using a double tilde ($\sim\sim$) to graph bidirected variables, e.g. $x_1 \sim\sim x_2$ is translated to $x_1 \leftrightarrow x_2$.

Usage

```
dagify(..., exposure = NULL, outcome = NULL, latent = NULL,
       labels = NULL, coords = NULL)
```

Arguments

...	formulas, which are converted to dagitty syntax
exposure	a character vector for the exposure (must be a variable name in the DAG)
outcome	a character vector for the outcome (must be a variable name in the DAG)
latent	a character vector for any latent variables (must be a variable name in the DAG)
labels	a named character vector, labels for variables in the DAG
coords	coordinates for the DAG nodes. Can be a named list or a data.frame with columns x, y, and name

Value

a dagitty DAG

See Also

[dag\(\)](#), [coords2df\(\)](#), [coords2list\(\)](#)

Examples

```
dagify(y ~ x + z, x ~ z)

coords <- list(
  x = c(A = 1, B = 2, D = 3, C = 3, F = 3, E = 4, G = 5, H = 5, I = 5),
  y = c(A = 0, B = 0, D = 1, C = 0, F = -1, E = 0, G = 1, H = 0, I = -1)
)

dag <- dagify(G ~~ H,
             G ~~ I,
             I ~~ G,
             H ~~ I,
             D ~ B,
```

```
C ~ B,  
I ~ C + F,  
F ~ B,  
B ~ A,  
H ~ E,  
C ~ E + G,  
G ~ D, coords = coords)  
  
dagitty::is.dagitty(dag)  
  
ggdag(dag)  
  
dag2 <- dagify(y ~ x + z2 + w2 + w1,  
              x ~ z1 + w1,  
              z1 ~ w1 + v,  
              z2 ~ w2 + v,  
              w1 ~~ w2,  
              exposure = "x",  
              outcome = "y")  
  
ggdag(dag2)
```

dplyr

Dplyr verb methods for tidy_dagitty objects

Description

Dplyr verb methods for tidy_dagitty objects.

Usage

```
## S3 method for class 'tidy_dagitty'  
select(.data, ...)  
  
## S3 method for class 'tidy_dagitty'  
filter(.data, ...)  
  
## S3 method for class 'tidy_dagitty'  
mutate(.data, ...)  
  
## S3 method for class 'tidy_dagitty'  
summarise(.data, ...)  
  
## S3 method for class 'tidy_dagitty'  
distinct(.data, ..., .keep_all = FALSE)  
  
## S3 method for class 'tidy_dagitty'  
arrange(.data, ...)
```

```
## S3 method for class 'tidy_dagitty'  
group_by(.data, ...)  
  
## S3 method for class 'tidy_dagitty'  
ungroup(.data, ...)  
  
## S3 method for class 'tidy_dagitty'  
transmute(.data, ...)  
  
## S3 method for class 'tidy_dagitty'  
distinct(.data, ..., .keep_all = FALSE)  
  
## S3 method for class 'tidy_dagitty'  
full_join(x, y, by = NULL, copy = FALSE,  
  suffix = c(".x", ".y"), ...)  
  
## S3 method for class 'tidy_dagitty'  
inner_join(x, y, by = NULL, copy = FALSE,  
  suffix = c(".x", ".y"), ...)  
  
## S3 method for class 'tidy_dagitty'  
left_join(x, y, by = NULL, copy = FALSE,  
  suffix = c(".x", ".y"), ...)  
  
## S3 method for class 'tidy_dagitty'  
right_join(x, y, by = NULL, copy = FALSE,  
  suffix = c(".x", ".y"), ...)  
  
## S3 method for class 'tidy_dagitty'  
anti_join(x, y, by = NULL, copy = FALSE, ...)  
  
## S3 method for class 'tidy_dagitty'  
semi_join(x, y, by = NULL, copy = FALSE, ...)  
  
## S3 method for class 'tidy_dagitty'  
slice(.data, ..., .dots = list())  
  
## S3 method for class 'tidy_dagitty'  
select_(.data, ..., .dots = list())  
  
## S3 method for class 'tidy_dagitty'  
filter_(.data, ..., .dots = list())  
  
## S3 method for class 'tidy_dagitty'  
mutate_(.data, ..., .dots = list())  
  
## S3 method for class 'tidy_dagitty'
```

```

summarise_(.data, ..., .dots = list())

## S3 method for class 'tidy_dagitty'
arrange_(.data, ..., .dots = list())

## S3 method for class 'tidy_dagitty'
slice_(.data, ..., .dots = list())

```

Arguments

```

.data          data object of class tidy_dagitty
...           other arguments passed to the dplyr function
.dots, x, y, by, copy, suffix, .keep_all
              see corresponding function in package dplyr

```

Examples

```

library(dplyr)
tidy_dagitty(m_bias()) %>%
  group_by(name) %>%
  summarize(n = n())

```

Equivalent DAGs and Classes

Generating Equivalent Models

Description

Returns a set of complete partially directed acyclic graphs (CPDAGs) given an input DAG. CPDAGs are Markov equivalent to the input graph. See `dagitty::equivalentDAGs()` for details. `node_equivalent_dags()` returns a set of DAGs, while `node_equivalent_class()` tags reversible edges. `ggdag_equivalent_dags()` plots all equivalent DAGs, while `ggdag_equivalent_class()` plots all reversible edges as undirected.

Usage

```

node_equivalent_dags(.dag, n = 100, layout = "auto", ...)

ggdag_equivalent_dags(.tdy_dag, ..., node_size = 16, text_size = 3.88,
  label_size = text_size, text_col = "white", label_col = text_col,
  node = TRUE, stylized = FALSE, text = TRUE, use_labels = NULL)

node_equivalent_class(.dag, layout = "auto")

ggdag_equivalent_class(.tdy_dag, expand_x = expand_scale(c(0.1, 0.1)),
  expand_y = expand_scale(c(0.1, 0.1)), breaks = ggplot2::waiver(),
  ..., node_size = 16, text_size = 3.88, label_size = text_size,
  text_col = "white", label_col = text_col, node = TRUE,
  stylized = FALSE, text = TRUE, use_labels = NULL)

```


Arguments

<code>.dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>n</code>	maximal number of returned graphs.
<code>layout</code>	a layout available in <code>ggraph</code> . See <code>ggraph::create_layout()</code> for details.
<code>...</code>	optional arguments passed to <code>ggraph::create_layout()</code>
<code>.tidy_dag</code>	an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>node_size</code>	size of DAG node
<code>text_size</code>	size of DAG text
<code>label_size</code>	size of label text
<code>text_col</code>	color of DAG text
<code>label_col</code>	color of label text
<code>node</code>	logical. Should nodes be included in the DAG?
<code>stylized</code>	logical. Should DAG nodes be stylized? If so, use <code>geom_dag_nodes</code> and if not use <code>geom_dag_point</code>
<code>text</code>	logical. Should text be included in the DAG?
<code>use_labels</code>	a string. Variable to use for <code>geom_dag_repel_label()</code> . Default is <code>NULL</code> .
<code>expand_x</code>	Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function <code>expand_scale()</code> to generate the values for the expand argument.
<code>expand_y</code>	Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function <code>expand_scale()</code> to generate the values for the expand argument.
<code>breaks</code>	One of: <ul style="list-style-type: none"> • <code>NULL</code> for no breaks • <code>waiver()</code> for the default breaks computed by the transformation object • A numeric vector of positions • A function that takes the limits as input and returns breaks as output

Value

a `tidy_dagitty` with at least one DAG, including a `dag` column to identify graph set for equivalent DAGs or a `reversible` column for equivalent classes, or a `ggplot`

Examples

```
g_ex <- dagify(y ~ x + z, x ~ z)
g_ex %>% node_equivalent_class()
g_ex %>% ggdag_equivalent_dags()
```

 Exogenous Variables *Find Exogenous Variables*

Description

`node_exogenous` tags exogenous variables given an exposure and outcome. `ggdag_exogenous` plots all exogenous variables. See [`dagitty::exogenousVariables\(\)`](#) for details.

Usage

```
node_exogenous(.dag, ...)

ggdag_exogenous(.tdy_dag, ..., node_size = 16, text_size = 3.88,
  edge_type = "link_arc", label_size = text_size, text_col = "white",
  label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
  use_labels = NULL)
```

Arguments

<code>.dag</code> , <code>.tdy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>
<code>node_size</code>	size of DAG node
<code>text_size</code>	size of DAG text
<code>edge_type</code>	a character vector, the edge geom to use. One of: "link_arc", which accounts for directed and bidirected edges, "link", "arc", or "diagonal"
<code>label_size</code>	size of label text
<code>text_col</code>	color of DAG text
<code>label_col</code>	color of label text
<code>node</code>	logical. Should nodes be included in the DAG?
<code>stylized</code>	logical. Should DAG nodes be stylized? If so, use <code>geom_dag_nodes</code> and if not use <code>geom_dag_point</code>
<code>text</code>	logical. Should text be included in the DAG?
<code>use_labels</code>	a string. Variable to use for <code>geom_dag_repel_label()</code> . Default is <code>NULL</code> .

Value

a `tidy_dagitty` with an exogenous column for exogenous variables or a `ggplot`

Examples

```
dag <- dagify(y ~ x1 + x2 + x3, b ~ x1 + x2)
ggdag_exogenous(dag)
node_exogenous(dag)
```

`expand_plot`*Quickly scale the size of a ggplot*

Description

`expand_plot()` is a convenience function that expands the scales of a `ggplot`, as the large node sizes in a DAG will often get clipped in themes that don't have DAGs in mind.

Usage

```
expand_plot(expand_x = expand_scale(c(0.1, 0.1)),
            expand_y = expand_scale(c(0.1, 0.1)))
```

Arguments

`expand_x`, `expand_y`

Vector of range expansion constants used to add some padding around the data, to ensure that they are placed some distance away from the axes. Use the convenience function `expand_scale()` to generate the values for the `expand` argument.

`expand_scale`*Generate expansion vector for scales.*

Description

This is a convenience function for generating scale expansion vectors for the `expand` argument of `scale*_continuous()` and `scale*_discrete()`. The expansion vectors are used to add some space between the data and the axes.

Usage

```
expand_scale(mult = 0, add = 0)
```

Arguments

`mult`

vector of multiplicative range expansion factors. If length 1, both the lower and upper limits of the scale are expanded outwards by `mult`. If length 2, the lower limit is expanded by `mult[1]` and the upper limit by `mult[2]`.

`add`

vector of additive range expansion constants. If length 1, both the lower and upper limits of the scale are expanded outwards by `add` units. If length 2, the lower limit is expanded by `add[1]` and the upper limit by `add[2]`.

Examples

```
# No space below the bars but 10% above them
ggplot(mtcars) +
  geom_bar(aes(x = factor(cyl))) +
  scale_y_continuous(expand = expand_scale(mult = c(0, .1)))

# Add 2 units of space on the left and right of the data
ggplot(subset(diamonds, carat > 2), aes(cut, clarity)) +
  geom_jitter() +
  scale_x_discrete(expand = expand_scale(add = 2))

# Reproduce the default range expansion used
# when the 'expand' argument is not specified
ggplot(subset(diamonds, carat > 2), aes(cut, price)) +
  geom_jitter() +
  scale_x_discrete(expand = expand_scale(add = .6)) +
  scale_y_continuous(expand = expand_scale(mult = .05))
```

fortify

Fortify a tidy_dagitty object for ggplot2

Description

Fortify a tidy_dagitty object for ggplot2

Usage

```
## S3 method for class 'tidy_dagitty'
fortify(model, data = NULL, ...)

## S3 method for class 'dagitty'
fortify(model, data = NULL, ...)
```

Arguments

model	an object of class tidy_dagitty or dagitty
data	(not used)
...	(not used)

 geom_dag_collider_edges

Edges for paths activated by stratification on colliders

Description

Adjusting for a collider activates pathways between the parent of the collider. This geom adds a curved edge between any such parent nodes.

Usage

```
geom_dag_collider_edges(mapping = NULL, data = NULL,
  stat = "identity", position = "identity", ..., size = 0.6,
  curvature = 0.5, angle = 90, ncp = 5, arrow = NULL,
  lineend = "butt", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE)
```

Arguments

mapping	Set of aesthetic mappings created by aes() or aes_() . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to ggplot() . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).
stat	The statistical transformation to use on the data for this layer, as a string.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	Other arguments passed on to layer() . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired geom/stat.
size	a numeric vector of length 1. Edge width
curvature	A numeric value giving the amount of curvature. Negative values produce left-hand curves, positive values produce right-hand curves, and zero produces a straight line.
angle	A numeric value between 0 and 180, giving an amount to skew the control points of the curve. Values less than 90 skew the curve towards the start point and values greater than 90 skew the curve towards the end point.

ncp	The number of control points used to draw the curve. More control points creates a smoother curve.
arrow	specification for arrow heads, as created by <code>arrow()</code> .
lineend	Line end style (round, butt, square).
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders()</code> .

Examples

```
library(dagitty)
dagify(m ~ a + b, x ~ a, y ~ b) %>%
  tidy_dagitty() %>%
  control_for("m") %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend, shape = adjusted)) +
    geom_dag_edges() +
    geom_dag_collider_edges() +
    geom_dag_point() +
    geom_dag_text() +
    theme_dag() +
    scale_adjusted()
```

geom_dag_edges

Directed and bidirected DAG edges

Description

Directed and bidirected DAG edges

Usage

```
geom_dag_edges(mapping = NULL, data_directed = filter_direction("<->"),
  data_bidirected = filter_direction("<->"), curvature = 0.3,
  arrow_directed = grid::arrow(length = grid::unit(5, "pt"), type =
  "closed"), arrow_bidirected = grid::arrow(length = grid::unit(5, "pt"),
  ends = "both", type = "closed"), position = "identity", na.rm = TRUE,
  show.legend = NA, inherit.aes = TRUE, fold = FALSE, ...)
```

Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data_directed, data_bidirected	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> ., and will be used as the layer data.
curvature	The bend of the curve. 1 approximates a halfcircle while 0 will give a straight line. Negative number will change the direction of the curve. Only used if <code>layout.circular = FALSE</code> .
arrow_directed, arrow_bidirected	specification for arrow heads, as created by <code>arrow()</code>
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If <code>FALSE</code> (the default), removes missing values with a warning. If <code>TRUE</code> silently removes missing values
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders()</code> .
fold	Logical. Should arcs appear on the same side of the nodes despite different directions. Default to <code>FALSE</code> .
...	Other arguments passed to <code>ggraph::geom_edge_*</code> ()

Aesthetics

`geom_dag_edges` understand the following aesthetics. Bold aesthetics are required.

- **x**
- **y**
- **xend**
- **yend**
- edge_colour
- edge_width
- edge_linetype
- edge_alpha
- start_cap

- end_cap
- label
- label_pos
- label_size
- angle
- hjust
- vjust
- family
- fontface
- lineheight

geom_dag_edges also uses geom_dag_edges_arc, which requires the **circular** aesthetic, but this is automatically set.

Examples

```
dagify(y ~ x + z2 + w2 + w1,
       x ~ z1 + w1,
       z1 ~ w1 + v,
       z2 ~ w2 + v,
       w1 ~~ w2) %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_dag_edges() +
  geom_dag_point() +
  geom_dag_text() +
  theme_dag()
```

geom_dag_text

Node text

Description

Node text

Usage

```
geom_dag_text(mapping = NULL, data = NULL, stat = "identity",
              position = "identity", ..., parse = FALSE, nudge_x = 0,
              nudge_y = 0, check_overlap = FALSE, na.rm = FALSE,
              show.legend = NA, inherit.aes = TRUE)
```


Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).
stat	The statistical transformation to use on the data for this layer, as a string.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	Other arguments passed on to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired geom/stat.
parse	If <code>TRUE</code> , the labels will be parsed into expressions and displayed as described in <code>?plotmath</code> .
nudge_x	Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales.
nudge_y	Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales.
check_overlap	If <code>TRUE</code> , text that overlaps previous text in the same layer will not be plotted.
na.rm	If <code>FALSE</code> , the default, missing values are removed with a warning. If <code>TRUE</code> , missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders()</code> .

Aesthetics

`geom_dag_text` understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **label**
- alpha
- angle

- colour
- family
- fontface
- group
- hjust
- lineheight
- size
- vjust

Examples

```
g <- dagify(m ~ x + y, y ~ x)
g %>%
  tidy_dagitty() %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
    geom_dag_point() +
    geom_dag_edges() +
    geom_dag_text() +
    theme_dag()
```

 ggdag

Quickly plot a DAG in ggplot2

Description

ggdag() is a wrapper to quickly plot DAGs.

Usage

```
ggdag(.tdy_dag, ..., edge_type = "link_arc", node_size = 16,
      text_size = 3.88, label_size = text_size, text_col = "white",
      label_col = "black", node = TRUE, stylized = FALSE, text = TRUE,
      use_labels = NULL)
```

Arguments

.tdy_dag	input graph, an object of class tidy_dagitty or dagitty
...	additional arguments passed to tidy_dagitty()
edge_type	a character vector, the edge geom to use. One of: "link_arc", which accounts for directed and bidirected edges, "link", "arc", or "diagonal"
node_size	size of DAG node
text_size	size of DAG text
label_size	size of label text
text_col	color of DAG text

label_col	color of label text
node	logical. Should nodes be included in the DAG?
stylized	logical. Should DAG nodes be stylized? If so, use geom_dag_nodes and if not use geom_dag_point
text	logical. Should text be included in the DAG?
use_labels	a string. Variable to use for geom_dag_repel_label(). Default is NULL.

Value

a ggplot

See Also

[ggdag_classic\(\)](#)

Examples

```
dag <- dagify(y ~ x + z2 + w2 + w1,
             x ~ z1 + w1,
             z1 ~ w1 + v,
             z2 ~ w2 + v,
             w1 ~~ w2)

ggdag(dag)
ggdag(dag) + theme_dag_blank()

ggdag(dagitty::randomDAG(5, .5))
```

ggdag_classic

Quickly plot a DAG in ggplot2

Description

ggdag_classic() is a wrapper to quickly plot DAGs in a more traditional style.

Usage

```
ggdag_classic(.tdy_dag, ..., size = 8, label_rect_size = NULL,
             text_label = "name", text_col = "black")
```

Arguments

.tidy_dag input graph, an object of class tidy_dagitty or dagitty
 ... additional arguments passed to tidy_dagitty()
 size text size, with a default of 8.
 label_rect_size specify the fontsize argument in ggraph::label_rect; default is NULL, in which case it is scaled relative to size
 text_label text variable, with a default of "name"
 text_col text color, with a default of "black"

Value

a ggplot

See Also

[ggdag\(\)](#)

Examples

```
dag <- dagify(y ~ x + z2 + w2 + w1,
  x ~ z1 + w1,
  z1 ~ w1 + v,
  z2 ~ w2 + v,
  w1 ~~ w2)

ggdag_classic(dag)
ggdag_classic(dag) + theme_dag_blank()

ggdag_classic(dagitty::randomDAG(5, .5))
```

ggplot.tidy_dagitty *Create a new ggplot*

Description

Create a new ggplot

Usage

```
## S3 method for class 'tidy_dagitty'
ggplot(data = NULL, mapping = aes(), ...)

## S3 method for class 'dagitty'
ggplot(data = NULL, mapping = aes(), ...)
```

Arguments

data	Default dataset to use for plot. If not already a data.frame, will be converted to one by <code>fortify()</code> . If not specified, must be supplied in each layer added to the plot.
mapping	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
...	Other arguments passed on to methods. Not currently used.

ggrepel functions *Repulsive textual annotations*

Description

These functions are minor modifications of those in the `ggrepel` package. `geom_dag_text_repel` adds text directly to the plot. `geom_dag_label_repel` draws a rectangle underneath the text, making it easier to read. The text labels repel away from each other and away from the data points.

Usage

```
geom_dag_text_repel(mapping = NULL, data = NULL, parse = FALSE, ...,
  box.padding = 0.35, point.padding = 1.5, segment.color = "#666666",
  fontface = "bold", segment.size = 0.5, arrow = NULL, force = 1,
  max.iter = 2000, nudge_x = 0, nudge_y = 0, na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE)
```

```
geom_dag_label_repel(mapping = NULL, data = NULL, parse = FALSE, ...,
  box.padding = grid::unit(0.35, "lines"),
  label.padding = grid::unit(0.25, "lines"),
  point.padding = grid::unit(1.5, "lines"), label.r = grid::unit(0.15,
  "lines"), label.size = 0.25, segment.color = "grey50",
  segment.size = 0.5, arrow = NULL, force = 1, max.iter = 2000,
  nudge_x = 0, nudge_y = 0, na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE)
```

Arguments

mapping	Set of aesthetic mappings created by <code>aes</code> or <code>aes_</code> . If specified and <code>inherit.aes = TRUE</code> (the default), is combined with the default mapping at the top level of the plot. You only need to supply <code>mapping</code> if there isn't a mapping defined for the plot.
data	A data frame. If specified, overrides the default data frame defined at the top level of the plot.
parse	If <code>TRUE</code> , the labels will be parsed into expressions and displayed as described in <code>?plotmath</code>

...	other arguments passed on to <code>layer</code> . There are three types of arguments you can use here: <ul style="list-style-type: none"> • Aesthetics: to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code>. • Other arguments to the layer, for example you override the default stat associated with the layer. • Other arguments passed on to the stat.
<code>box.padding</code>	Amount of padding around bounding box, as unit or number. Defaults to 0.25. (Default unit is lines, but other units can be specified by passing <code>unit(x, "units")</code>).
<code>point.padding</code>	Amount of padding around labeled point, as unit or number. Defaults to 0. (Default unit is lines, but other units can be specified by passing <code>unit(x, "units")</code>).
<code>segment.color</code>	Colour of the line segment. Defaults to the same colour as the text. In the unlikely event you specify both US and UK spellings of colour, the US spelling will take precedence.
<code>fontface</code>	A character vector. Default is "bold"
<code>segment.size</code>	Width of line segment connecting the data point to the text label, in mm.
<code>arrow</code>	specification for arrow heads, as created by <code>arrow</code>
<code>force</code>	Force of repulsion between overlapping text labels. Defaults to 1.
<code>max.iter</code>	Maximum number of iterations to try to resolve overlaps. Defaults to 2000.
<code>nudge_x</code>	Horizontal and vertical adjustments to nudge the starting position of each text label.
<code>nudge_y</code>	Horizontal and vertical adjustments to nudge the starting position of each text label.
<code>na.rm</code>	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
<code>show.legend</code>	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
<code>inherit.aes</code>	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders</code> .
<code>label.padding</code>	Amount of padding around label, as unit or number. Defaults to 0.25. (Default unit is lines, but other units can be specified by passing <code>unit(x, "units")</code>).
<code>label.r</code>	Radius of rounded corners, as unit or number. Defaults to 0.15. (Default unit is lines, but other units can be specified by passing <code>unit(x, "units")</code>).
<code>label.size</code>	Size of label border, in mm.

Examples

```
g <- dagify(m ~ x + y,
y ~ x,
exposure = "x",
outcome = "y",
latent = "m",
```

```

labels = c("x" = "Exposure", "y" = "Outcome", "m" = "Collider")

g %>% tidy_dagitty() %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
    geom_dag_edges() +
    geom_dag_point() +
    geom_dag_text_repel(aes(label = name), show.legend = FALSE) +
    theme_dag()

g %>% tidy_dagitty() %>%
  dag_label(labels = c("x" = "This is the exposure",
    "y" = "Here's the outcome",
    "m" = "Here is where they collide")) %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
    geom_dag_edges() +
    geom_dag_point() +
    geom_dag_text() +
    geom_dag_label_repel(aes(label = label, fill = label),
      col = "white", show.legend = FALSE) +
    theme_dag()

```

Instrumental Variables

Find Instrumental Variables

Description

`node_instrumental` tags instrumental variables given an exposure and outcome. `ggdag_instrumental` plots all instrumental variables. See [dagitty::instrumentalVariables\(\)](#) for details.

Usage

```

node_instrumental(.dag, exposure = NULL, outcome = NULL, ...)

ggdag_instrumental(.tdy_dag, exposure = NULL, outcome = NULL, ...,
  node_size = 16, text_size = 3.88, label_size = text_size,
  text_col = "white", label_col = text_col, node = TRUE,
  stylized = FALSE, text = TRUE, use_labels = NULL)

```

Arguments

<code>.dag</code> , <code>.tdy_dag</code>	input graph, an object of class <code>tidy_dagitty</code> or <code>dagitty</code>
<code>exposure</code>	character vector of length 1, name of exposure variable. Default is <code>NULL</code> , in which case it will check the input DAG for exposure.
<code>outcome</code>	character vector of length 1, name of exposure variable. Default is <code>NULL</code> , in which case it will check the input DAG for exposure.
<code>...</code>	additional arguments passed to <code>tidy_dagitty()</code>

node_size	size of DAG node
text_size	size of DAG text
label_size	size of label text
text_col	color of DAG text
label_col	color of label text
node	logical. Should nodes be included in the DAG?
stylized	logical. Should DAG nodes be stylized? If so, use geom_dag_nodes and if not use geom_dag_point
text	logical. Should text be included in the DAG?
use_labels	a string. Variable to use for geom_dag_repel_label(). Default is NULL.

Value

a tidy_dagitty with an instrumental column for instrumental variables or a ggplot

Examples

```
library(dagitty)

node_instrumental(dagitty("dag{ i->x->y; x<->y }"), "x", "y")
ggdag_instrumental(dagitty("dag{ i->x->y; i2->x->y; x<->y }"), "x", "y")
```

is.tidy_dagitty *Test for object class for tidy_dagitty*

Description

Test for object class for tidy_dagitty

Usage

```
is.tidy_dagitty(x)
```

Arguments

x object to be tested

is_confounder	<i>Assess if a variable confounds a relationship</i>
---------------	--

Description

Assess if a variable confounds a relationship

Usage

```
is_confounder(.tdy_dag, z, x, y, direct = FALSE)
```

Arguments

.tdy_dag	input graph, an object of class tidy_dagitty or dagitty
z	a character vector, the potential confounder
x, y	a character vector, the variables z may confound.
direct	logical. Only consider direct confounding? Default is FALSE

Value

Logical. Is the variable a confounder?

Examples

```
dag <- dagify(y ~ z, x ~ z)

is_confounder(dag, "z", "x", "y")
is_confounder(dag, "x", "z", "y")
```

Nodes	<i>DAG Nodes</i>
-------	------------------

Description

geom_dag_node and geom_dag_point are very similar to [ggplot2::geom_point](#) but with a few defaults changed. geom_dag_node is slightly stylized and includes an internal white circle, while geom_dag_point plots a single point.

Usage

```
geom_dag_node(mapping = NULL, data = NULL, position = "identity",
  ..., na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

geom_dag_point(mapping = NULL, data = NULL, position = "identity",
  ..., na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	Other arguments passed on to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired <code>geom/stat</code> .
na.rm	If <code>FALSE</code> , the default, missing values are removed with a warning. If <code>TRUE</code> , missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders()</code> .

Aesthetics

`geom_dag_node` and `geom_dag_point` understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- alpha
- colour
- fill
- shape
- size
- stroke
- filter

`geom_dag_node` also accepts:

- internal_colour

Examples

```

g <- dagify(m ~ x + y, y ~ x)
p <- g %>%
  tidy_dagitty() %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
    geom_dag_edges() +
    theme_dag()

p +
  geom_dag_node() +
  geom_dag_text()

p +
  geom_dag_point() +
  geom_dag_text()

```

Pathways

*Find Pathways Between Variables***Description**

`node_paths` finds the pathways between a given exposure and outcome. `ggdag_paths` plots all pathways. See `dagitty::paths()` for details.

Usage

```

dag_paths(.dag, from = NULL, to = NULL, adjust_for = NULL,
  directed = FALSE, paths_only = FALSE, ...)

```

```

ggdag_paths(.tdy_dag, from = NULL, to = NULL, adjust_for = NULL,
  directed = FALSE, shadow = FALSE, ..., node_size = 16,
  text_size = 3.88, label_size = text_size, text_col = "white",
  label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
  use_labels = NULL)

```

```

ggdag_paths_fan(.tdy_dag, from = NULL, to = NULL, adjust_for = NULL,
  directed = FALSE, ..., spread = 0.7, node_size = 16,
  text_size = 2, label_size = text_size, text_col = "white",
  label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
  use_labels = NULL)

```

Arguments

`.dag`, `.tdy_dag` input graph, an object of class `tidy_dagitty` or `dagitty`

`from` character vector of length 1, name of exposure variable. Default is `NULL`, in which case it will check the input DAG for exposure.

`to` character vector of length 1, name of exposure variable. Default is `NULL`, in which case it will check the input DAG for exposure.

adjust_for	character vector, a set of variables to control for. Default is NULL.
directed	logical. Should only directed paths be shown?
paths_only	logical. Should only open paths be returned? Default is FALSE, which includes every variable and edge in the DAG regardless if they are part of the path.
...	additional arguments passed to tidy_dagitty()
shadow	logical. Show edges not in path? Ignored if paths_only is TRUE
node_size	size of DAG node
text_size	size of DAG text
label_size	size of label text
text_col	color of DAG text
label_col	label color
node	logical. Should nodes be included in the DAG?
stylized	logical. Should DAG nodes be stylized? If so, use geom_dag_nodes and if not use geom_dag_point
text	logical. Should text be included in the DAG?
use_labels	a string. Variable to use for geom_dag_repel_label(). Default is NULL.
spread	the width of the fan spread

Value

a tidy_dagitty with a path column for path variables and a set grouping column or a ggplot

Examples

```
confounder_triangle(x_y_associated = TRUE) %>%
  ggdag_paths(from = "x", to = "y")
```

```
print.tidy_dagitty    Print a tidy_dagitty
```

Description

Print a tidy_dagitty

Usage

```
## S3 method for class 'tidy_dagitty'
print(x, ...)
```

Arguments

x an object of class tidy_dagitty
 ... optional arguments passed to print()

Quick Plots for Common DAGs

Quickly create a DAGs with common structures of bias

Description

base functions create an object of class dagitty; `ggdag_*` functions are wrappers that also call `ggdag()` on the dagitty object.

Usage

```
m_bias(x = NULL, y = NULL, a = NULL, b = NULL, m = NULL,
       x_y_associated = FALSE)
```

```
butterfly_bias(x = NULL, y = NULL, a = NULL, b = NULL, m = NULL,
              x_y_associated = FALSE)
```

```
confounder_triangle(x = NULL, y = NULL, z = NULL,
                   x_y_associated = FALSE)
```

```
collider_triangle(x = NULL, y = NULL, m = NULL,
                  x_y_associated = FALSE)
```

```
mediation_triangle(x = NULL, y = NULL, m = NULL,
                   x_y_associated = FALSE)
```

```
ggdag_m_bias(x = NULL, y = NULL, a = NULL, b = NULL, m = NULL,
             x_y_associated = FALSE, edge_type = "link_arc", node_size = 16,
             text_size = 3.88, label_size = text_size, text_col = "white",
             label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
             use_labels = NULL)
```

```
ggdag_butterfly_bias(x = NULL, y = NULL, a = NULL, b = NULL,
                    m = NULL, x_y_associated = FALSE, edge_type = "link_arc",
                    node_size = 16, text_size = 3.88, label_size = text_size,
                    text_col = "white", label_col = text_col, node = TRUE,
                    stylized = FALSE, text = TRUE, use_labels = NULL)
```

```
ggdag_confounder_triangle(x = NULL, y = NULL, z = NULL,
                          x_y_associated = FALSE, edge_type = "link_arc",
                          node_size = 16, text_size = 3.88, label_size = text_size,
                          text_col = "white", label_col = text_col, node = TRUE,
                          stylized = FALSE, text = TRUE, use_labels = NULL)
```

```
ggdag_collider_triangle(x = NULL, y = NULL, m = NULL,
                        x_y_associated = FALSE, edge_type = "link_arc",
                        node_size = 16, text_size = 3.88, label_size = text_size,
                        text_col = "white",
```

```

label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
use_labels = NULL)

ggdag_mediation_triangle(x = NULL, y = NULL, m = NULL,
  x_y_associated = FALSE, edge_type = "link_arc", node_size = 16,
  text_size = 3.88, label_size = text_size, text_col = "white",
  label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
  use_labels = NULL)

```

Arguments

`x, y, a, b, m, z` Character vector. Optional label. Default is NULL

`x_y_associated` Logical. Are `x` and `y` associated? Default is FALSE.

`edge_type` a character vector, the edge geom to use. One of: "link_arc", which accounts for directed and bidirected edges, "link", "arc", or "diagonal"

`node_size` size of DAG node

`text_size` size of DAG text

`label_size` size of label text

`text_col` color of DAG text

`label_col` color of label text

`node` logical. Should nodes be included in the DAG?

`stylized` logical. Should DAG nodes be stylized? If so, use `geom_dag_nodes` and if not use `geom_dag_point`

`text` logical. Should text be included in the DAG?

`use_labels` a string. Variable to use for `geom_dag_repel_label()`. Default is NULL.

Value

a DAG of class `dagitty` or a `ggplot`

Examples

```

m_bias() %>% ggdag_adjust("m")
ggdag_confounder_triangle()

```

remove_axes

Quickly remove plot axes and grids

Description

`remove_axes()` and `remove_grid()` are convenience functions that removes the axes and grids from a `ggplot`, respectively. This is useful when you want to use an existing theme, e.g. those included in `ggplot2`, for a DAG.

Usage

```
remove_axes()
```

```
remove_grid()
```

Examples

```
ggdag(confounder_triangle()) +  
  theme_bw() +  
  remove_axes()
```

scale_adjusted

Common scale adjustments for DAGs

Description

`scale_adjusted()` is a convenience function that implements ways of visualizing adjustment for a variable. By convention, a square shape is used to indicate adjustment and a circle when not adjusted. Arrows out of adjusted variables are often eliminated or de-emphasized, and `scale_adjusted()` uses a lower alpha for these arrows. When adjusting a collider, a dashed line is sometimes used to demarcate opened pathways, and `scale_adjusted()` does this whenever `geom_dag_collider_edges()` is used. `scale_dag()` is deprecated in favor of `scale_adjusted()`.

Usage

```
scale_adjusted()
```

```
scale_dag(breaks = ggplot2::waiver())
```

Arguments

breaks

One of:

- NULL for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output

 simulate_data

Simulate Data from Structural Equation Model

Description

This is a thin wrapper for the `simulateSEM()` function in `dagitty` that works with tidied `dagitty` objects. It treats the input DAG as a structural equation model, generating random path coefficients and simulating corresponding data. See `dagitty::simulateSEM()` for details.

Usage

```
simulate_data(.tdy_dag, b.default = NULL, b.lower = -0.6,
             b.upper = 0.6, eps = 1, N = 500, standardized = TRUE)
```

Arguments

<code>.tdy_dag</code>	the input DAG, which can be a <code>tidy_dagitty</code> or <code>dagitty</code> object.
<code>b.default</code>	default path coefficient applied to arrows for which no coefficient is defined in the model syntax.
<code>b.lower</code>	lower bound for random path coefficients, applied if <code>b.default = NULL</code> .
<code>b.upper</code>	upper bound for path coefficients.
<code>eps</code>	residual variance (only meaningful if <code>standardized=FALSE</code>).
<code>N</code>	number of samples to generate.
<code>standardized</code>	whether a standardized output is desired (all variables have variance 1).

Value

a `tbl` with `N` values for each variable in `.tdy_dag`

Examples

```
dagify(y ~ z, x ~ z) %>%
  tidy_dagitty() %>%
  simulate_data()
```

tbl_df.tidy_daggity *Convert a tidy_dagitty object to tbl_df*

Description

Convert a tidy_dagitty object to tbl_df

Usage

```
tbl_df.tidy_daggity(.tdy_dag)
```

Arguments

.tdy_dag an object of class tidy_dagitty

Test if Variable Is Collider

Detecting colliders in DAGs

Description

Detecting colliders in DAGs

Usage

```
is_collider(.dag, .var, downstream = TRUE)
```

```
is_downstream_collider(.dag, .var)
```

Arguments

.dag an input graph, an object of class tidy_dagitty or dagitty

.var a character vector of length 1, the potential collider to check

downstream Logical. Check for downstream colliders? Default is TRUE.

Value

Logical. Is the variable a collider or downstream collider?

Examples

```
dag <- dagify(m ~ x + y, m_jr ~ m)
is_collider(dag, "m")
is_downstream_collider(dag, "m_jr")

# a downstream collider is also treated as a collider
is_collider(dag, "m_jr")

# but a direct collider is not treated as a downstream collider
is_downstream_collider(dag, "m")
```

theme_dag_blank	<i>Minimalist DAG themes</i>
-----------------	------------------------------

Description

Minimalist DAG themes

Usage

```
theme_dag_blank(base_size = 12, base_family = "", ...)  
theme_dag(base_size = 12, base_family = "", ...)  
theme_dag_grid(base_size = 12, base_family = "", ...)
```

Arguments

base_size	base font size
base_family	base font family
...	additional arguments passed to theme()

Examples

```
ggdag(m_bias()) + theme_dag_blank() # the default
```

theme_dag_grey	<i>Simple grey themes for DAGs</i>
----------------	------------------------------------

Description

Simple grey themes for DAGs

Usage

```
theme_dag_grey(base_size = 12, base_family = "", ...)
```

```
theme_dag_gray(base_size = 12, base_family = "", ...)
```

```
theme_dag_grey_grid(base_size = 12, base_family = "", ...)
```

```
theme_dag_gray_grid(base_size = 12, base_family = "", ...)
```

Arguments

base_size	base font size
base_family	base font family
...	additional arguments passed to theme()

Examples

```
ggdag(m_bias()) + theme_dag_grey()
```

tidy_dagitty	<i>Tidy a dagitty object</i>
--------------	------------------------------

Description

Tidy a dagitty object

Usage

```
tidy_dagitty(.dagitty, seed = NULL, layout = "nicely", ...)
```

Arguments

.dagitty	a dagitty
seed	a numeric seed for reproducible layout generation
layout	a layout available in ggraph. See ggraph::create_layout() for details.
...	optional arguments passed to ggraph::create_layout()

Value

a `tidy_dagitty` object

Examples

```
library(dagitty)
library(ggplot2)

dag <- dagitty("dag {
  Y <- X <- Z1 <- V -> Z2 -> Y
  Z1 <- W1 <-> W2 -> Z2
  X <- W1 -> Y
  X <- W2 -> Y
  X [exposure]
  Y [outcome]
}")

tidy_dagitty(dag)

tidy_dagitty(dag, layout = "fr") %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_dag_node() +
  geom_dag_text() +
  geom_dag_edges() +
  theme_dag()
```

Variable Status

Find variable status

Description

Detects variable status given a DAG (exposure, outcome, latent). See [`dagitty::VariableStatus\(\)`](#) for details.

Usage

```
node_status(.dag, as_factor = TRUE, ...)

ggdag_status(.tdy_dag, ..., edge_type = "link_arc", node_size = 16,
  text_size = 3.88, label_size = text_size, text_col = "white",
  label_col = text_col, node = TRUE, stylized = FALSE, text = TRUE,
  use_labels = NULL)
```

Arguments

`.dag`, `.tdy_dag` input graph, an object of class `tidy_dagitty` or `dagitty`
`as_factor` treat status variable as factor
`...` additional arguments passed to `tidy_dagitty()`

<code>edge_type</code>	a character vector, the edge geom to use. One of: "link_arc", which accounts for directed and bidirected edges, "link", "arc", or "diagonal"
<code>node_size</code>	size of DAG node
<code>text_size</code>	size of DAG text
<code>label_size</code>	size of label text
<code>text_col</code>	color of DAG text
<code>label_col</code>	color of label text
<code>node</code>	logical. Should nodes be included in the DAG?
<code>stylized</code>	logical. Should DAG nodes be stylized? If so, use <code>geom_dag_nodes</code> and if not use <code>geom_dag_point</code>
<code>text</code>	logical. Should text be included in the DAG?
<code>use_labels</code>	a string. Variable to use for <code>geom_dag_repel_label()</code> . Default is NULL.

Details

`node_collider` tags variable status and `ggdag_collider` plots all variable statuses.

Value

a `tidy_dagitty` with a status column for variable status or a `ggplot`

Examples

```
dag <- dagify(l ~ x + y,  
             y ~ x,  
             exposure = "x",  
             outcome = "y",  
             latent = "l")  
  
node_status(dag)  
ggdag_status(dag)
```

Index

activate_collider_paths, 3
Adjust for variables, 3
adjust_for (Adjust for variables), 3
aes, 37
aes(), 29, 33, 42
aes_, 37
aes_(), 29, 33, 42
anti_join.tidy_dagitty (dplyr), 22
arrange.tidy_dagitty (dplyr), 22
arrange_.tidy_dagitty (dplyr), 22
arrow, 38
as.data.frame.tidy_dagitty, 5
as.tbl.tidy_dagitty, 5
as_tbl_graph, 11
Assess d-separation between variables, 6
Assess familial relationships between variables, 8

borders, 38
borders(), 30, 33, 42
butterfly_bias (Quick Plots for Common DAGs), 45

Canonicalize DAGs, 12
collider_triangle (Quick Plots for Common DAGs), 45
Colliders, 13
confounder_triangle (Quick Plots for Common DAGs), 45
control_for (Adjust for variables), 3
control_for(), 3
coordinates, 14
coords2df (coordinates), 14
coords2df(), 21
coords2list (coordinates), 14
coords2list(), 21
Covariate Adjustment Sets, 15

dag, 16

DAG Edges, 17
DAG Labels, 20
dag(), 21
dag_adjustment_sets (Covariate Adjustment Sets), 15
dag_label (DAG Labels), 20
dag_paths (Pathways), 43
dagify, 21
dagitty, 16
dagitty::adjustmentSets(), 15
dagitty::canonicalize(), 12
dagitty::children, 8
dagitty::dseparated(), 6
dagitty::equivalentDAGs(), 24
dagitty::exogenousVariables(), 26
dagitty::instrumentalVariables(), 39
dagitty::paths(), 43
dagitty::simulateSEM(), 48
dagitty::VariableStatus(), 52
distinct.tidy_dagitty (dplyr), 22
dplyr, 22

Equivalent DAGs and Classes, 24
Exogenous Variables, 26
expand_plot, 27
expand_scale, 27

filter.tidy_dagitty (dplyr), 22
filter_.tidy_dagitty (dplyr), 22
fortify, 28
fortify(), 29, 33, 37, 42
full_join.tidy_dagitty (dplyr), 22

geom_dag_collider_edges, 29
geom_dag_collider_edges(), 3, 47
geom_dag_edges, 30
geom_dag_edges_arc (DAG Edges), 17
geom_dag_edges_diagonal (DAG Edges), 17
geom_dag_edges_fan (DAG Edges), 17
geom_dag_edges_link (DAG Edges), 17

- geom_dag_label_repel (ggrepel functions), 37
- geom_dag_node (Nodes), 41
- geom_dag_point (Nodes), 41
- geom_dag_text, 32
- geom_dag_text_repel (ggrepel functions), 37
- ggdag, 34
- ggdag(), 36
- ggdag_adjacent (Assess familial relationships between variables), 8
- ggdag_adjust (Adjust for variables), 3
- ggdag_adjust(), 3
- ggdag_adjustment_set (Covariate Adjustment Sets), 15
- ggdag_ancestors (Assess familial relationships between variables), 8
- ggdag_butterfly_bias (Quick Plots for Common DAGs), 45
- ggdag_canonical (Canonicalize DAGs), 12
- ggdag_children (Assess familial relationships between variables), 8
- ggdag_classic, 35
- ggdag_classic(), 35
- ggdag_collider (Colliders), 13
- ggdag_collider_triangle (Quick Plots for Common DAGs), 45
- ggdag_confounder_triangle (Quick Plots for Common DAGs), 45
- ggdag_dconnected (Assess d-separation between variables), 6
- ggdag_descendants (Assess familial relationships between variables), 8
- ggdag_drelationship (Assess d-separation between variables), 6
- ggdag_dseparated (Assess d-separation between variables), 6
- ggdag_equivalent_class (Equivalent DAGs and Classes), 24
- ggdag_equivalent_dags (Equivalent DAGs and Classes), 24
- ggdag_exogenous (Exogenous Variables), 26
- ggdag_instrumental (Instrumental Variables), 39
- ggdag_m_bias (Quick Plots for Common DAGs), 45
- ggdag_markov_blanket (Assess familial relationships between variables), 8
- ggdag_mediation_triangle (Quick Plots for Common DAGs), 45
- ggdag_parents (Assess familial relationships between variables), 8
- ggdag_paths (Pathways), 43
- ggdag_paths_fan (Pathways), 43
- ggdag_status (Variable Status), 52
- ggplot(), 29, 33, 42
- ggplot.dagitty (ggplot.tidy_dagitty), 36
- ggplot.tidy_dagitty, 36
- ggplot2::geom_point, 41
- ggraph::create_layout(), 25, 51
- ggrepel functions, 37
- grDevices::plotmath(), 18
- grid::unit(), 18
- group_by.tidy_dagitty (dplyr), 22
- has_labels (DAG Labels), 20
- inner_join.tidy_dagitty (dplyr), 22
- Instrumental Variables, 39
- is.tidy_dagitty, 40
- is_collider (Test if Variable Is Collider), 49
- is_confounder, 41
- is_downstream_collider (Test if Variable Is Collider), 49
- label (DAG Labels), 20
- label<- (DAG Labels), 20
- layer, 38
- layer(), 29, 33, 42
- left_join.tidy_dagitty (dplyr), 22
- m_bias (Quick Plots for Common DAGs), 45
- mediation_triangle (Quick Plots for Common DAGs), 45
- mutate.tidy_dagitty (dplyr), 22
- mutate_.tidy_dagitty (dplyr), 22

- node_adjacent (Assess familial relationships between variables), 8
- node_ancestors (Assess familial relationships between variables), 8
- node_canonical (Canonicalize DAGs), 12
- node_children (Assess familial relationships between variables), 8
- node_collider (Colliders), 13
- node_dconnected (Assess d-separation between variables), 6
- node_descendants (Assess familial relationships between variables), 8
- node_drelationship (Assess d-separation between variables), 6
- node_dseparated (Assess d-separation between variables), 6
- node_equivalent_class (Equivalent DAGs and Classes), 24
- node_equivalent_dags (Equivalent DAGs and Classes), 24
- node_exogenous (Exogenous Variables), 26
- node_instrumental (Instrumental Variables), 39
- node_markov_blanket (Assess familial relationships between variables), 8
- node_parents (Assess familial relationships between variables), 8
- node_status (Variable Status), 52
- Nodes, 41
- Pathways, 43
- print.tidy_dagitty, 44
- Quick Plots for Common DAGs, 45
- remove_axes, 46
- remove_grid (remove_axes), 46
- right_join.tidy_dagitty (dplyr), 22
- scale_adjusted, 47
- scale_dag (scale_adjusted), 47
- select.tidy_dagitty (dplyr), 22
- select_.tidy_dagitty (dplyr), 22
- semi_join.tidy_dagitty (dplyr), 22
- simulate_data, 48
- slice.tidy_dagitty (dplyr), 22
- slice_.tidy_dagitty (dplyr), 22
- summarise.tidy_dagitty (dplyr), 22
- summarise_.tidy_dagitty (dplyr), 22
- tbl_df.tidy_dagitty, 49
- Test if Variable Is Collider, 49
- theme_dag (theme_dag_blank), 50
- theme_dag_blank, 50
- theme_dag_gray (theme_dag_gray), 51
- theme_dag_gray_grid (theme_dag_gray), 51
- theme_dag_grey, 51
- theme_dag_grey_grid (theme_dag_grey), 51
- theme_dag_grid (theme_dag_blank), 50
- tidy_dagitty, 51
- tidygraph::as_tbl_graph(), 11
- transmute.tidy_dagitty (dplyr), 22
- ungroup.tidy_dagitty (dplyr), 22
- Variable Status, 52