Package ‘ordr’

August 18, 2022

Title A 'tidyverse' Extension for Ordinations and Biplots

Version 0.1.0

Description Ordination comprises several multivariate exploratory and explanatory techniques with theoretical foundations in geometric data analysis; see Podani (2000, ISBN:90-5782-067-6) for techniques and applications and Le Roux & Rouanet (2005) <doi:10.1007/1-4020-2236-0> for foundations.
Greenacre (2010, ISBN:978-84-923846) shows how the most established of these, including principal components analysis, correspondence analysis, multidimensional scaling, factor analysis, and discriminant analysis, rely on eigen-decompositions or singular value decompositions of pre-processed numeric matrix data.
These decompositions give rise to a set of shared coordinates along which the row and column elements can be measured. The overlay of their scatterplots on these axes, introduced by Gabriel (1971) <doi:10.1093/biomet/58.3.453>, is called a biplot.
'ordr' provides inspection, extraction, manipulation, and visualization tools for several popular ordination classes supported by a set of recovery methods. It is inspired by and designed to integrate into 'tidyverse' workflows provided by Wickham et al (2019) <doi:10.21105/joss.01686>.

Depends R (>= 3.3.0), ggplot2

Imports rlang, stringr, tidyselect, scales, generics, magrittr, tibble, broom, tidyr, dplyr, purrr, labeling, ggrepel,

License GPL-3

Encoding UTF-8

LazyData true

Suggests testthat, sessioninfo, gridExtra, MASS, mlpack, vegan, heplots, knitr, rmarkdown

RoxygenNote 7.2.1

Collate 'aaa-.r' 'biplot-key.r' 'biplot.r' 'data.r' 'dplyr-verbs.r' 'utils.r' 'ord-recoverers.r' 'ord-augmentation.r' 'ord-conference.r' 'ord-tbl.r' 'fun-lda.r' 'fun-lra.r' 'fun-wrap.r' 'geom-axis.r' 'geom-intervals.r' 'geom-isoline.r'

'geom-origin.r' 'geom-text-radiate.r' 'geom-unit-circle.r'
'geom-utils.r' 'geom-vector.r' 'list-cancor-tidiers.r'
'list-cmdscale-tidiers.r' 'ord-tidiers.r' 'list-tidiers.r'
'methods-base-eigen.r' 'methods-base-svd.r'
'methods-mass-correspondence.r' 'methods-mass-lda.r'
'methods-mass-mca.r' 'methods-ordr-lra.r'
'methods-stats-cancor.r' 'methods-stats-cmds.r'
'methods-stats-factanal.r' 'methods-stats-kmeans.r'
'methods-stats-lm.r' 'methods-stats-prcomp.r'
'methods-stats-princomp.r' 'ord-annotation.r' 'ord-format.r'
'ord-negation.r' 'ord-plot.r' 'ord-supplementation.r'
'ordinate.r' 'ordr.r' 'stat-center.r' 'stat-chull.r'
'stat-cone.r' 'stat-matrix.r' 'stat-scale.r' 'stat-spantree.r'
'themes.r' 'zzz-biplot-geoms.r' 'zzz-biplot-stats.r' 'zzz.r'

VignetteBuilder knitr

NeedsCompilation no

Author Jason Cory Brunson [aut, cre] (<`https://orcid.org/0000-0003-3126-9494`>),
Emily Paul [ctb]

Maintainer Jason Cory Brunson <`cornelioid@gmail.com`>

Repository CRAN

Date/Publication 2022-08-18 07:40:05 UTC

R topics documented:

`annotation` *Annotate factors of 'tbl_ord' objects*

Description

These functions annotate the matrix factors of tbl_ords with additional variables, and retrieve these annotations.

The unexported `annotation_*()` and `set_annotation_*()` functions assign and retrieve values of the `"*_annotation"` attributes of `x`, which must have the same number of rows as `get_*(x)`.

Arguments

`annot`	A data.frame having the same number of rows as `get_*(x)`.

See Also

augmentation methods that must interface with annotation.

`augmentation` *Augment factors and coordinates of 'tbl_ord' objects*

Description

These functions return data associated with the cases, variables, and coordinates of an ordination object, and attach it to the object.

Usage

```
recover_aug_rows(x)

recover_aug_cols(x)

recover_aug_coord(x)

augment_ord(x, .matrix = "dims")
```

Arguments

`x`	An object of class 'tbl_ord'.
`.matrix`	A character string partially matched (lowercase) to several indicators for one or both matrices in a matrix decomposition used for ordination. The standard values are `"rows"`, `"cols"`, and `"dims"` (for both).

Details

The `recover_aug_*()` S3 methods produce tibbles of values associated with the rows, columns, and artificial coordinates of an object of class 'tbl_ord'. The first field of each tibble is `name`, which contains the row, column, or coordinate names. Additional fields contain information about the rows, columns, or coordinates extracted from the ordination object.

The function `augment_ord()` returns the ordination with either or both matrix factors annotated with the result of `recover_aug_*()`. In this way `augment_ord()` works like `generics::augment()`, as popularized by the **broom** package, by extracting information about the rows and columns, but it differs in returning an annotated 'tbl_ord' rather than a 'tbl_df' object. The advantage of implementing separate methods for the rows, columns, and artificial coordinates is that more information contained in the original object becomes accessible to the user.

Value

The `recover_aug_*()` functions return tibbles having the same numbers of rows as `recover_*()`. `augment_ord()` returns an augmented tbl_ord with the wrapped model unchanged.

See Also

tidiers and annotation methods that interface with augmentation.

Other generic recoverers: `conference`, `recoverers`, `supplementation`

`biplot-geoms` *Convenience geoms for row and column matrix factors*

Description

These geometric element layers (geoms) pair conventional **ggplot2** geoms with `stat_rows()` or `stat_cols()` in order to render elements for one or the other matrix factor of a tbl_ord. They understand the same aesthetics as their corresponding conventional geoms.

Usage

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

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

geom_rows_path(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  arrow = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_cols_path(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  arrow = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_rows_polygon(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  rule = "evenodd",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_cols_polygon(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  rule = "evenodd",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_rows_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
)

geom_cols_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
)

geom_rows_label(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  label.padding = unit(0.25, "lines"),
  label.r = unit(0.15, "lines"),
  label.size = 0.25,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_cols_label(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  nudge_x = 0,
  nudge_y = 0,
  label.padding = unit(0.25, "lines"),
  label.r = unit(0.15, "lines"),
  label.size = 0.25,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_rows_text_repel(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  parse = FALSE,
  ...,
  box.padding = 0.25,
  point.padding = 1e-06,
  min.segment.length = 0.5,
  arrow = NULL,
  force = 1,
  force_pull = 1,
  max.time = 0.5,
  max.iter = 10000,
  max.overlaps = getOption("ggrepel.max.overlaps", default = 10),
  nudge_x = 0,
  nudge_y = 0,
  xlim = c(NA, NA),
  ylim = c(NA, NA),
  na.rm = FALSE,
  show.legend = NA,
  direction = c("both", "y", "x"),
  seed = NA,
  verbose = FALSE,
  inherit.aes = TRUE
)

geom_cols_text_repel(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  parse = FALSE,
  ...,
  box.padding = 0.25,
  point.padding = 1e-06,
  min.segment.length = 0.5,
  arrow = NULL,
  force = 1,
  force_pull = 1,
  max.time = 0.5,
  max.iter = 10000,
  max.overlaps = getOption("ggrepel.max.overlaps", default = 10),
  nudge_x = 0,
  nudge_y = 0,
  xlim = c(NA, NA),
  ylim = c(NA, NA),
  na.rm = FALSE,
  show.legend = NA,
  direction = c("both", "y", "x"),
  seed = NA,
  verbose = FALSE,
  inherit.aes = TRUE
)

geom_rows_label_repel(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  parse = FALSE,
  ...,
  box.padding = 0.25,
  label.padding = 0.25,
  point.padding = 1e-06,
  label.r = 0.15,
  label.size = 0.25,
  min.segment.length = 0.5,
  arrow = NULL,
  force = 1,
  force_pull = 1,
  max.time = 0.5,
  max.iter = 10000,
  max.overlaps = getOption("ggrepel.max.overlaps", default = 10),
  nudge_x = 0,
  nudge_y = 0,
  xlim = c(NA, NA),
  ylim = c(NA, NA),
  na.rm = FALSE,
  show.legend = NA,
  direction = c("both", "y", "x"),
  seed = NA,
  verbose = FALSE,
  inherit.aes = TRUE
)

geom_cols_label_repel(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  parse = FALSE,
  ...,
  box.padding = 0.25,
  label.padding = 0.25,
  point.padding = 1e-06,
  label.r = 0.15,
  label.size = 0.25,
  min.segment.length = 0.5,
  arrow = NULL,
  force = 1,
  force_pull = 1,
  max.time = 0.5,
  max.iter = 10000,
  max.overlaps = getOption("ggrepel.max.overlaps", default = 10),
  nudge_x = 0,
  nudge_y = 0,
  xlim = c(NA, NA),
  ylim = c(NA, NA),
  na.rm = FALSE,
  show.legend = NA,
  direction = c("both", "y", "x"),
  seed = NA,
  verbose = FALSE,
  inherit.aes = TRUE
)

geom_rows_axis(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  axis_labels = TRUE,
  axis_ticks = TRUE,
  axis_text = TRUE,
  by = NULL,
  num = NULL,
  tick_length = 0.025,
  text_dodge = 0.03,
  label_dodge = 0.03,
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_cols_axis(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  axis_labels = TRUE,
  axis_ticks = TRUE,
  axis_text = TRUE,
  by = NULL,
  num = NULL,
  tick_length = 0.025,
  text_dodge = 0.03,
  label_dodge = 0.03,
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

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

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

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

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

geom_rows_isoline(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  isoline_text = TRUE,
  by = NULL,
  num = NULL,
  label_dodge = 0.03,
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_cols_isoline(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  isoline_text = TRUE,
  by = NULL,
  num = NULL,
  label_dodge = 0.03,
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_rows_text_radiate(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_cols_text_radiate(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

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

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

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`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.
`...`	Additional arguments passed to `ggplot2::layer()`.
`na.rm`	Passed to `ggplot2::layer()`.
`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. `borders()`.
`lineend`	Line end style (round, butt, square).
`linejoin`	Line join style (round, mitre, bevel).
`linemitre`	Line mitre limit (number greater than 1).
`arrow`	Arrow specification, as created by `grid::arrow()`.
`rule`	Either `"evenodd"` or `"winding"`. If polygons with holes are being drawn (using the `subgroup` aesthetic) this argument defines how the hole coordinates are interpreted. See the examples in `grid::pathGrob()` for an explanation.

`parse`	If `TRUE`, the labels will be parsed into expressions and displayed as described in `?plotmath`.
`nudge_x, nudge_y`	Horizontal and vertical adjustment to nudge labels by. Useful for offsetting text from points, particularly on discrete scales. Cannot be jointly specified with `position`.
`check_overlap`	If `TRUE`, text that overlaps previous text in the same layer will not be plotted. `check_overlap` happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.
`label.padding`	Amount of padding around label. Defaults to 0.25 lines.
`label.r`	Radius of rounded corners. Defaults to 0.15 lines.
`label.size`	Size of label border, in mm.
`box.padding`	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 `unit(x, "units")`).
`point.padding`	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 `unit(x, "units")`).
`min.segment.length`	Skip drawing segments shorter than this, as unit or number. Defaults to 0.5. (Default unit is lines, but other units can be specified by passing `unit(x, "units")`).
`force`	Force of repulsion between overlapping text labels. Defaults to 1.
`force_pull`	Force of attraction between a text label and its corresponding data point. Defaults to 1.
`max.time`	Maximum number of seconds to try to resolve overlaps. Defaults to 0.5.
`max.iter`	Maximum number of iterations to try to resolve overlaps. Defaults to 10000.
`max.overlaps`	Exclude text labels that overlap too many things. Defaults to 10.
`xlim, ylim`	Limits for the x and y axes. Text labels will be constrained to these limits. By default, text labels are constrained to the entire plot area.
`direction`	direction of stairs: 'vh' for vertical then horizontal, 'hv' for horizontal then vertical, or 'mid' for step half-way between adjacent x-values.
`seed`	Random seed passed to `set.seed`. Defaults to `NA`, which means that `set.seed` will not be called.
`verbose`	If `TRUE`, some diagnostics of the repel algorithm are printed
`axis_labels, axis_ticks, axis_text`	Logical; whether to include labels, tick marks, and text value marks along the axes.
`by, num`	Intervals between elements or number of elements; specify only one.
`tick_length`	Numeric; the length of the tick marks, as a proportion of the minimum of the plot width and height.
`text_dodge`	Numeric; the orthogonal distance of the text from the axis, as a proportion of the minimum of the plot width and height.
`label_dodge`	Numeric; the orthogonal distance of the text from the axis or isoline, as a proportion of the minimum of the plot width and height.
`isoline_text`	Logical; whether to include text value marks along the isolines.

Value

A ggproto layer.

See Also

Other biplot layers: `biplot-stats`, `stat_rows()`

`biplot-stats` *Convenience stats for row and column matrix factors*

Description

These statistical transformations (stats) adapt conventional **ggplot2** stats to one or the other matrix factor of a tbl_ord, in lieu of `stat_rows()` or `stat_cols()`. They accept the same parameters as their corresponding conventional stats.

Usage

```
stat_rows_ellipse(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  ...,
  type = "t",
  level = 0.95,
  segments = 51,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_cols_ellipse(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  ...,
  type = "t",
  level = 0.95,
  segments = 51,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_rows_center(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  fun.data = NULL,
  fun.center = NULL,
  fun.min = NULL,
  fun.max = NULL,
  fun.args = list()
)

stat_cols_center(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  fun.data = NULL,
  fun.center = NULL,
  fun.min = NULL,
  fun.max = NULL,
  fun.args = list()
)

stat_rows_star(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  fun.data = NULL,
  fun.center = NULL,
  fun.args = list()
)

stat_cols_star(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  fun.data = NULL,
  fun.center = NULL,
  fun.args = list()
)

stat_rows_chull(
  mapping = NULL,
  data = NULL,
  geom = "polygon",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

stat_cols_chull(
  mapping = NULL,
  data = NULL,
  geom = "polygon",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

stat_rows_cone(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  origin = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

stat_cols_cone(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  origin = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

```
stat_rows_scale(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  mult = 1
)

stat_cols_scale(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  mult = 1
)

stat_rows_spantree(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  position = "identity",
  engine = "vegan",
  method = "euclidean",
  show.legend = NA,
  inherit.aes = TRUE,
  check.aes = TRUE,
  check.param = TRUE,
  ...
)

stat_cols_spantree(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  position = "identity",
  engine = "vegan",
  method = "euclidean",
  show.legend = NA,
  inherit.aes = TRUE,
  check.aes = TRUE,
  check.param = TRUE,
  ...
)
```

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`geom`	The geometric object to use display the data
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`...`	Additional arguments passed to `ggplot2::layer()`.
`type`	The type of ellipse. The default `"t"` assumes a multivariate t-distribution, and `"norm"` assumes a multivariate normal distribution. `"euclid"` draws a circle with the radius equal to `level`, representing the euclidean distance from the center. This ellipse probably won't appear circular unless `coord_fixed()` is applied.
`level`	The level at which to draw an ellipse, or, if `type="euclid"`, the radius of the circle to be drawn.
`segments`	The number of segments to be used in drawing the ellipse.
`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. `borders()`.
`fun.data, fun.center, fun.min, fun.max, fun.args`	Functions and arguments treated as in `ggplot2::stat_summary()`, with `fun.center`, `fun.min`, and `fun.max` behaving as `fun.y`, `fun.ymin`, and `fun.ymax`.
`origin`	Logical; whether to include the origin with the transformed data. Defaults to `FALSE`.
`mult`	Numeric value used to scale the coordinates.
`engine`	A single character string specifying the package implementation to use; either `"vegan"` or `"mlpack"`.

`method`	Passed to `stats::dist()` if engine is `"vegan"`, ignored if `"mlpack"`.
`check.aes, check.param`	If `TRUE`, the default, will check that supplied parameters and aesthetics are understood by the `geom` or `stat`. Use `FALSE` to suppress the checks.

Value

A ggproto layer.

Ordination aesthetics

The convenience function `ord_aes()` can be used to incorporate all coordinates of the ordination model into a statistical transformation. It maps the coordinates to the custom aesthetics `..coord1`, `..coord2`, etc.

Some transformations, e.g. `stat_center()`, are commutative with projection to the 'x' and 'y' coordinates. If they detect aesthetics of the form `..coord[0-9]+`, then `..coord1` and `..coord2` are converted to `x` and `y` while any remaining are ignored.

Other transformations, e.g. `stat_spantree()`, yield different results in a planar biplot when they are computer before or after projection. If such a stat layer detects these aesthetics, then the lot of them are used in the transformation.

In either case, the stat layer returns a data frame with position aesthetics `x` and `y`.

See Also

Other biplot layers: `biplot-geoms`, `stat_rows()`

Examples

```
# compute row-principal components of scaled iris measurements
iris[, -5] %>%
  prcomp(scale = TRUE) %>%
  as_tbl_ord() %>%
  mutate_rows(species = iris$Species) %>%
  print() -> iris_pca

# row-principal biplot with centroids and confidence elliptical disks
iris_pca %>%
  ggbiplot(aes(color = species)) +
  theme_bw() +
  geom_rows_point() +
  geom_polygon(
    aes(fill = species),
    color = NA, alpha = .25, stat = "rows_ellipse"
  ) +
  geom_cols_vector(color = "#444444") +
  scale_color_brewer(
    type = "qual", palette = 2,
    aesthetics = c("color", "fill")
  ) +
  ggtitle(
    "Row-principal PCA biplot of Anderson iris measurements",
    "Overlaid with 95% confidence disks"
  )
```

---

cancor_tidiers *Tidy* cancor() *list output*

Description

These tidiers handle the output of cancor(), which is effectively an S3 object without a class attribute. It allows **ordr** to enhance the list_tidiers provided by **broom**.

Usage

```
tidy_cancor(x, matrix = "xcoef", ...)
```

Arguments

x	A list with components cor, xcoef, ycoef, xcenter, and ycenter as returned by stats::cancor().
matrix	Character specifying which list element should be tidied, matched to the following options. • "xcoef": returns information about the estimated coefficients in the x variables. • "ycoef": returns information about the estimated coefficients in the y variables • "cor": returns information about the canonical correlations.
...	Additional arguments allowed by generics; currently ignored.

Details

cancor() returns a named list of 5 elements. These tidiers rely on this list structure to organize the model output into a tibble.

Value

A tibble.

See Also

generics::tidy() stats::cancor()

Other list tidiers: cmdscale_tidiers, list_tidiers

Examples

```
# savings data
class(LifeCycleSavings)
pop <- LifeCycleSavings[, 2:3]
oec <- LifeCycleSavings[, -(2:3)]
# canonical correlation analysis
savings_cca <- cancor(pop, oec)

# return the tidied canonical coefficients for the left variables
tidy(savings_cca)
# return the tidied canonical coefficients for the right variables
tidy(savings_cca, matrix = "ycoef")
# return the canonical coefficients, with summary statistics
tidy(savings_cca, matrix = "cor")

# scree plot
ggplot(tidy(savings_cca, matrix = "cor"), aes(x = CC, y = percent)) +
  theme_bw() +
  geom_col() +
  labs(x = "Canonical dimension", y = "Percent of variance")
```

`cmdscale_tidiers` *Tidy* `cmdscale()` *list output*

Description

These tidiers handle the output of `cmdscale()`, which under certain conditions is effectively an S3 object without a class attribute. It allows **ordr** to enhance the list_tidiers provided by **broom**.

Usage

```
tidy_cmdscale(x, matrix = "points", ...)

glance_cmdscale(x, ...)
```

Arguments

`x`	A list with components `points`, `eig`, `x`, `ac`, and `GOF` as returned by `stats::cmdscale()`.
`matrix`	Character specifying which list element should be tidied, matched to the following options. • `"points"`: returns information about the coordinates in the representation space. • `"x"`: returns information about the doubly-centered symmetric matrix used in the calculation. • `"eig"`: returns information about the eigenvalues.
`...`	Additional arguments allowed by generics; currently ignored.

Details

When `cmdscale()` is instructed to return any of several optional elements, or when `list. = TRUE`, the output is not the default point coordinate matrix but a 5-element list with a consistent naming scheme (though some elements will be `NULL` if their parameters are not set to `TRUE`). These tidiers rely on this list structure to organize the model output into a tibble.

Value

A tibble.

See Also

`generics::tidy()` `generics::glance()` `stats::cmdscale()`

Other list tidiers: `cancor_tidiers`, `list_tidiers`

Examples

```
# 'dist' object (matrix of road distances) of large American cities
class(UScitiesD)
print(UScitiesD)

# use multidimensional scaling to infer artificial planar coordinates
UScitiesD %>%
  cmdscale(k = 3L, eig = TRUE, x.ret = TRUE) ->
  usa_mds

# glance at the model
glance(usa_mds)
# return the tidied coordinates
tidy(usa_mds)
# return the upper triangle of the doubly-centered distance matrix in tidy form
tidy(usa_mds, matrix = "x")
# return the eigenvalues for the principal coordinates, with summary statistics
tidy(usa_mds, matrix = "eig")
# reorient to conventional compass
tidy(usa_mds) %>%
  ggplot(aes(x = -PCo1, y = -PCo2)) +
  geom_text(aes(label = point), size = 3) +
  ggtitle("MDS biplot of distances between U.S. cities")
```

`conference` *Confer inertia to factors of a 'tbl_ord' object*

Description

Re-distribute inertia between rows and columns in an ordination.

Usage

```
recover_conference(x)

## Default S3 method:
recover_conference(x)

get_conference(x)

revert_conference(x)

confer_inertia(x, p)
```

Arguments

`x`	A tbl_ord.
`p`	Numeric vector of length 1 or 2. If length 1, the proportion of the inertia assigned to the cases, with the remainder `1 - p` assigned to the variables. If length 2, the proportions of the inertia assigned to the cases and to the variables, respectively.

Details

The *inertia* of a singular value decomposition $X = UDV'$ consists in the squares of the singular values (the diagonal elements of D), and represents the variance, likened to the physical inertia, in the directions of the orthogonal singular vectors (the columns of U or of V). Biplots superimpose the projections of the rows and the columns of X onto these coordinate vectors, scaled by some proportion of the total inertia: UD^p and VD^q. A biplot is *balanced* if $p + q = 1$. Read Orlov (2013) for more on conferring inertia in PCA.

`recover_conference()`, like the other recoverers, is an S3 method that is exported for convenience but not intended to be used directly.

Note: In case the `"inertia"` *attribute is a rectangular matrix, one may only be able to confer it entirely to the cases* (`p = 1`) *or entirely to the variables* (`p = 0`)*.*

Value

`recover_conference()` returns the (statically implemented) distribution of inertia between the rows and the columns as stored in the model. `confer_inertia()` returns a tbl_ord with a specified distribution of inertia but the wrapped model unchanged. `get_conference()` returns the distribution currently conferred.

References

Orlov K (2013) *Answer to* "Algebra of LDA. Fisher discrimination power of a variable and Linear Discriminant Analysis". CrossValidated, accessed 2019-07-26. `https://stats.stackexchange.com/a/83114/68743`

See Also

Other generic recoverers: `augmentation`, `recoverers`, `supplementation`

Examples

```
# illustrative ordination: correspondence analysis of hair & eye data
haireye_ca <- ordinate(
  as.data.frame(rowSums(HairEyeColor, dims = 2L)),
  cols = everything(), model = MASS::corresp
)
print(haireye_ca)

# check distribution of inertia
get_conference(haireye_ca)
# confer inertia to rows, then to columns
confer_inertia(haireye_ca, "rows")
confer_inertia(haireye_ca, "columns")
# confer inertia symmetrically
(haireye_ca <- confer_inertia(haireye_ca, "symmetric"))
# check redistributed inertia
get_conference(haireye_ca)
# restore default distribution of inertia
revert_conference(haireye_ca)
```

---

dplyr-verbs — **dplyr** *verbs for tbl_ord factors*

Description

These functions adapt dplyr verbs to the factors of a tbl_ord.

The raw verbs are not defined for tbl_ords; instead, each verb has two analogues, corresponding to the two matrix factors. They each rely on a common workhorse function, which takes the composition of the **dplyr** verb with `annotation_*`, applied to the factor, removes any variables corresponding to coordinates or already annotated, and only then assigns it as the new `"*_annotation"` attribute of `.data` (see annotation). Note that these functions are not generics and so cannot be extended to other classes.

Usage

```
pull_factor(.data, var = -1, .matrix)

pull_rows(.data, var = -1)

pull_cols(.data, var = -1)

rename_rows(.data, ...)

rename_cols(.data, ...)

select_rows(.data, ...)

select_cols(.data, ...)
```

```
mutate_rows(.data, ...)

mutate_cols(.data, ...)

transmute_rows(.data, ...)

transmute_cols(.data, ...)

cbind_rows(.data, ..., elements = "all")

cbind_cols(.data, ..., elements = "all")

left_join_rows(.data, ...)

left_join_cols(.data, ...)
```

Arguments

`.data`	An object of class 'tbl_ord'.
`var`	A variable specified as in `dplyr::pull()`.
`.matrix`	A character string partially matched (lowercase) to several indicators for one or both matrices in a matrix decomposition used for ordination. The standard values are `"rows"`, `"cols"`, and `"dims"` (for both).
`...`	Comma-separated unquoted expressions as in, e.g., `dplyr::select()`.
`elements`	Character vector; which elements of each factor for which to render graphical elements. One of `"all"` (the default), `"active"`, or any supplementary element type defined by the specific class methods (e.g. `"score"` for 'factanal', 'lda_ord', and 'cancord_ord' and `"intraset"` and `"interset"` for 'cancor_ord').

Value

A tbl_ord; the wrapped model is unchanged.

Examples

```
# illustrative ordination: LDA of iris data
(iris_lda <- ordinate(iris, cols = 1:4, lda_ord, grouping = iris$Species))

# extract a coordinate or annotation
head(pull_rows(iris_lda, Species))
pull_cols(iris_lda, LD2)

# rename an annotation
rename_cols(iris_lda, species = name)

# select annotations
select_rows(iris_lda, species = name, .element)
```

```
# create, modify, and delete annotations
mutate_cols(iris_lda, vec.length = sqrt(LD1^2 + LD2^2))
transmute_cols(iris_lda, vec.length = sqrt(LD1^2 + LD2^2))

# bind data frames of annotations
iris_medians <-
  stats::aggregate(iris[, 1:4], median, by = iris[, 5, drop = FALSE])
iris_lda %>%
  # retain '.element' in order to match by `elements`
  select_rows(.element) %>%
  cbind_rows(iris_medians, elements = "active")
```

draw-key *Biplot key drawing functions*

Description

These key drawing functions supplement those built into **ggplot2** for producing legends suitable to biplots.

Usage

```
draw_key_line(data, params, size)

draw_key_crosslines(data, params, size)

draw_key_crosspoint(data, params, size)
```

Arguments

data	A single row data frame containing the scaled aesthetics to display in this key
params	A list of additional parameters supplied to the geom.
size	Width and height of key in mm.

Details

`draw_key_line()` is a horizontal counterpart to `ggplot2::draw_key_vline()`. `draw_key_crosslines()` superimposes these two keys, and `draw_key_crosspoint()` additionally superimposes an oversized `ggplot2::draw_key_point()`.

Value

A grid grob.

See Also

ggplot2::draw_key for key glyphs installed with **ggplot2**.

Examples

```
# scaled PCA of Anderson iris data with ranges and confidence intervals
iris[, -5] %>%
  prcomp(scale = TRUE) %>%
  as_tbl_ord() %>%
  confer_inertia(1) %>%
  augment_ord() %>%
  mutate_rows(species = iris$Species) %>%
  ggbiplot(aes(color = species)) +
  theme_bw() +
  scale_color_brewer(type = "qual", palette = 2) +
  geom_rows_lineranges(fun.data = mean_sdl, size = .75) +
  geom_rows_point(alpha = .5) +
  geom_cols_vector(color = "#444444") +
  geom_cols_text_radiate(aes(label = name), color = "#444444", size = 3) +
  ggtitle(
    "Row-principal PCA biplot of Anderson iris data",
    "Ranges 2 sample standard deviations from centroids"
  )
```

`format` *Format a tbl_ord for printing*

Description

These methods of `base::format()` and `base::print()` render a (usually more) tidy readout of a tbl_ord that is consistent across all original ordination classes.

Usage

```
## S3 method for class 'tbl_ord'
format(
  x,
  width = NULL,
  ...,
  n = NULL,
  max_extra_cols = NULL,
  max_footer_lines = NULL
)

## S3 method for class 'tbl_ord'
print(
  x,
  width = NULL,
  ...,
  n = NULL,
  max_extra_cols = NULL,
  max_footer_lines = NULL
)
```

Arguments

`x`	A tbl_ord.
`width`	Width of text output to generate. This defaults to `NULL`, which means use the `width` option.
`...`	Additional arguments.
`n`	Number of rows to show. If `NULL`, the default, will print all rows if less than the `print_max` option. Otherwise, will print as many rows as specified by the `print_min` option.
`max_extra_cols`	Number of extra columns to print abbreviated information for, if the width is too small for the entire tibble. If `NULL`, the `max_extra_cols` option is used. The previously defined `n_extra` argument is soft-deprecated.
`max_footer_lines`	Maximum number of footer lines. If `NULL`, the `max_footer_lines` option is used.

Details

The `format` and `print` methods for class 'tbl_ord' are adapted from those for class 'tbl_df' and for class 'tbl_graph' from the **tidygraph** package.

Note: The `format()` function is tedius but cannot be easily modularized without invoking recoverers, annotation, and augmentation multiple times, thereby significantly reducing performance.

Value

The `format()` method returns a vector of strings that are more elegantly printed by the `print()` method, which itself returns the tbl_ord invisibly.

`geom_axis` *Axes through the origin*

Description

`geom_axis()` renders lines through the origin and the position of each case or variable.

Usage

```
geom_axis(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  axis_labels = TRUE,
  axis_ticks = TRUE,
  axis_text = TRUE,
  by = NULL,
  num = NULL,
  tick_length = 0.025,
  text_dodge = 0.03,
  label_dodge = 0.03,
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`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.
`axis_labels, axis_ticks, axis_text`	Logical; whether to include labels, tick marks, and text value marks along the axes.
`by, num`	Intervals between elements or number of elements; specify only one.
`tick_length`	Numeric; the length of the tick marks, as a proportion of the minimum of the plot width and height.
`text_dodge`	Numeric; the orthogonal distance of the text from the axis, as a proportion of the minimum of the plot width and height.
`label_dodge`	Numeric; the orthogonal distance of the text from the axis or isoline, as a proportion of the minimum of the plot width and height.
`...`	Additional arguments passed to `ggplot2::layer()`.
`parse`	If `TRUE`, the labels will be parsed into expressions and displayed as described in `?plotmath`.
`check_overlap`	If `TRUE`, text that overlaps previous text in the same layer will not be plotted. `check_overlap` happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

`na.rm`	Passed to `ggplot2::layer()`.
`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. `borders()`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Aesthetics

`geom_axis()` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `colour`
- `alpha`
- `size`
- `linetype`
- `label`
- `center`, `scale`
- `label_colour`, `label_alpha`, `label_size`, `label_angle`, `label_hjust`, `label_vjust`, `label_family`, `label_fontface`
- `tick_colour`, `tick_alpha`, `tick_size`, `tick_linetype`
- `text_colour`, `text_alpha`, `text_size`, `text_angle`, `text_hjust`, `text_vjust`, `text_family`, `text_fontface`
- `group`

The prefixed aesthetics `label_*`, `tick_*`, and `text_*` are used by the text elements and will inherit any values passed to their un-prefixed counterparts, if recognized.

See Also

Other geom layers: `geom_isoline()`, `geom_lineranges()`, `geom_origin()`, `geom_text_radiate()`, `geom_unit_circle()`, `geom_vector()`

Examples

```
# Reaven & Miller overt & chemical diabetes test data and group classification
head(heplots::Diabetes)

# default (standardized) linear discriminant analysis of groups on tests
diabetes_lda <- MASS::lda(group ~ ., heplots::Diabetes)
# bestow 'tbl_ord' class & augment observation, centroid, and variable fields
as_tbl_ord(diabetes_lda) %>%
  augment_ord() %>%
  mutate_rows(discriminant = ifelse(
    .element == "active",
    "centroid", "case"
  )) %>%
  print() -> diabetes_lda
# row-standard biplot
diabetes_lda %>%
  confer_inertia(1) %>%
  ggbiplot() +
  theme_bw() + theme_biplot() +
  geom_rows_point(aes(shape = grouping, size = discriminant), alpha = .5) +
  geom_cols_axis(aes(label = name), color = "#888888", num = 8L,
                 text_size = 2.5, label_dodge = .02) +
  ggtitle(
    "LDA of Reaven & Miller diabetes groups",
    "Row-standard biplot of standardized LDA"
  )

# contribution LDA of groups on tests
diabetes_lda <-
  lda_ord(group ~ ., heplots::Diabetes, axes.scale = "contribution")
# bestow 'tbl_ord' class & augment observation, centroid, and variable fields
as_tbl_ord(diabetes_lda) %>%
  augment_ord() %>%
  mutate_rows(discriminant = ifelse(
    .element == "active",
    "centroid", "case"
  )) %>%
  print() -> diabetes_lda
# symmetric biplot
diabetes_lda %>%
  confer_inertia(.5) %>%
  ggbiplot() +
  theme_bw() + theme_biplot() +
  geom_rows_point(aes(shape = grouping, alpha = discriminant)) +
  geom_cols_axis(color = "#888888", num = 8L,
                 text_size = 2.5, text_dodge = .025) +
  ggtitle(
    "LDA of Reaven & Miller diabetes groups",
    "Symmetric biplot of contribution LDA"
  )
```

`geom_isoline` *Isolines (contour lines)*

Description

`geom_isoline()` renders isolines along row or column axes.

Usage

```
geom_isoline(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  isoline_text = TRUE,
  by = NULL,
  num = NULL,
  label_dodge = 0.03,
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`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.
`isoline_text`	Logical; whether to include text value marks along the isolines.
`by, num`	Intervals between elements or number of elements; specify only one.

`label_dodge`	Numeric; the orthogonal distance of the text from the axis or isoline, as a proportion of the minimum of the plot width and height.
`...`	Additional arguments passed to `ggplot2::layer()`.
`parse`	If `TRUE`, the labels will be parsed into expressions and displayed as described in `?plotmath`.
`check_overlap`	If `TRUE`, text that overlaps previous text in the same layer will not be plotted. `check_overlap` happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.
`na.rm`	Passed to `ggplot2::layer()`.
`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. `borders()`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Aesthetics

`geom_isoline()` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `colour`
- `alpha`
- `size`
- `linetype`
- `center`, `scale`
- `angle`
- `hjust`
- `vjust`

- family
- fontface
- text_colour, text_alpha, text_size,
- group

The prefixed aesthetics text_* are used by the text elements and will inherit any values passed to their un-prefixed counterparts.

See Also

Other geom layers: geom_axis(), geom_lineranges(), geom_origin(), geom_text_radiate(), geom_unit_circle(), geom_vector()

Examples

```
# Reaven & Miller overt & chemical diabetes test data and group classification
head(heplots::Diabetes)
# default (standardized) linear discriminant analysis of groups on tests
diabetes_lda <- MASS::lda(group ~ ., heplots::Diabetes)

# bestow 'tbl_ord' class & augment observation, centroid, and variable fields
as_tbl_ord(diabetes_lda) %>%
  augment_ord() %>%
  print() -> diabetes_lda

# row-standard biplot
diabetes_lda %>%
  confer_inertia(1) %>%
  ggbiplot(aes(label = name), elements = "active") +
  theme_bw() + theme_biplot() +
  geom_rows_text() +
  geom_cols_vector(subset = c(1, 3, 4)) +
  geom_cols_text_radiate(subset = c(1, 3, 4), size = 3) +
  geom_cols_isoline(subset = c(1, 3, 4), alpha = .25, num = 4L,
                    label_dodge = -.03, text_alpha = .5, text_size = 3) +
  ggtitle(
    "LDA of Reaven & Miller diabetes groups",
    "Row-standard biplot of standardized LDA"
  )
```

---

geom_lineranges *Intervals depicting ranges, usually about center points*

Description

geom_lineranges() renders horizontal and vertical intervals for a specified subject or variable; geom_pointranges() additionally renders a point at their crosshairs.

Usage

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

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

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`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.
`...`	Additional arguments passed to `ggplot2::layer()`.
`na.rm`	Passed to `ggplot2::layer()`.
`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. borders().

Value

A ggproto layer.

Biplot layers

ggbiplot() uses ggplot2::fortify() internally to produce a single data frame with a .matrix column distinguishing the subjects ("rows") and variables ("cols"). The stat layers stat_rows() and stat_cols() simply filter the data frame to one of these two.

The geom layers geom_rows_*() and geom_cols_*() call the corresponding stat in order to render plot elements for the corresponding factor matrix. geom_dims_*() selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Aesthetics

geom_lineranges() and geom_pointranges() understand the following aesthetics (required aesthetics are in bold):

- x
- xmin
- xmax
- y
- ymin
- ymax`
- alpha
- colour
- linetype
- size
- group

See Also

Other geom layers: geom_axis(), geom_isoline(), geom_origin(), geom_text_radiate(), geom_unit_circle(), geom_vector()

Examples

```
# compute log-ratio analysis of Freestone primary class composition measurements
glass %>%
  ordinate(cols = c(SiO2, Al2O3, CaO, FeO, MgO),
           model = lra, compositional = TRUE) %>%
  confer_inertia("rows") %>%
  print() -> glass_lra
```

```
# row-principal biplot with ordinate-wise standard deviations
glass_lra %>%
  ggbiplot(aes(color = Site), sec.axes = "cols", scale.factor = .05) +
  theme_biplot() +
  scale_color_brewer(type = "qual", palette = 6) +
  geom_cols_text(stat = "chull", aes(label = name), color = "#444444") +
  geom_rows_lineranges(fun.data = mean_sdl, size = .75) +
  geom_rows_point(alpha = .5) +
  ggtitle(
    "Row-principal LRA biplot of Freestone glass measurements",
    "Ranges 2 sample standard deviations from centroids"
  )
```

`geom_origin` *Crosshairs or circle at the origin*

Description

`geom_origin()` renders a symbol, either a set of crosshairs or a circle, at the origin.

Usage

```
geom_origin(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  marker = "crosshairs",
  radius = unit(0.04, "snpc"),
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`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.
`marker`	The symbol to be drawn at the origin; matched to `"crosshairs"` or `"circle"`.
`radius`	A `grid::unit()` object that sets the radius of the crosshairs or of the circle.
`...`	Additional arguments passed to `ggplot2::layer()`.
`na.rm`	Passed to `ggplot2::layer()`.
`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. `borders()`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Aesthetics

`geom_origin()` accepts no aesthetics.

See Also

Other geom layers: `geom_axis()`, `geom_isoline()`, `geom_lineranges()`, `geom_text_radiate()`, `geom_unit_circle()`, `geom_vector()`

`geom_text_radiate` *Text radiating outward from the origin*

Description

`geom_text_radiate()` is adapted from `ggbiplot()` in the off-CRAN extensions of the same name (Vu, 2014; Telford, 2017; Gegzna, 2018). It renders text at specified positions and angles that radiate out from the origin.

Usage

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

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`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. Cannot be jointy specified with `nudge_x` or `nudge_y`.
`...`	Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

`parse`	If `TRUE`, the labels will be parsed into expressions and displayed as described in `?plotmath`.
`check_overlap`	If `TRUE`, text that overlaps previous text in the same layer will not be plotted. `check_overlap` happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.
`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. `borders()`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Aesthetics

`geom_text_radiate()` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `label`
- `alpha`
- `angle`
- `colour`
- `family`
- `fontface`
- `hjust`
- `lineheight`
- `size`
- `vjust`
- `group`

References

Vincent Q. Vu (2014). ggbiplot: A 'ggplot2' based biplot. R package version 0.55. https://github.com/vqv/ggbiplot, `experimental` branch

Richard J Telford (2017). ggbiplot: A 'ggplot2' based biplot. R package version 0.6. https://github.com/richardjtelford/ggbiplot (fork), `experimental` branch

Vilmantas Gegzna (2018). ggbiplot: A 'ggplot2' based biplot. R package version 0.55. https://github.com/forked-packages/ggbiplot (fork), `experimental` branch

See Also

Other geom layers: `geom_axis()`, `geom_isoline()`, `geom_lineranges()`, `geom_origin()`, `geom_unit_circle()`, `geom_vector()`

`geom_unit_circle` *Unit circle*

Description

`geom_unit_circle()` renders the unit circle, centered at the origin with radius 1.

Usage

```
geom_unit_circle(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  segments = 60,
  scale.factor = 1,
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`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.
`segments`	The number of segments to be used in drawing the circle.
`scale.factor`	The circle radius; should remain at its default value 1 or passed the same value as `ggbiplot()`. (This is an imperfect fix that may be changed in a future version.)
`...`	Additional arguments passed to `ggplot2::layer()`.
`na.rm`	Passed to `ggplot2::layer()`.
`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. `borders()`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Aesthetics

`geom_unit_circle()` understands the following aesthetics (none required):

- `alpha`
- `colour`
- `linetype`
- `size`

See Also

Other geom layers: `geom_axis()`, `geom_isoline()`, `geom_lineranges()`, `geom_origin()`, `geom_text_radiate()`, `geom_vector()`

Examples

```
# principal components analysis of overt & chemical diabetes test values
heplots::Diabetes[, seq(5L)] %>%
  princomp(cor = TRUE) %>%
  as_tbl_ord() %>%
  cbind_rows(group = heplots::Diabetes$group) %>%
  augment_ord() %>%
  print() -> diabetes_pca

# note that column standard coordinates are unit vectors
rowSums(get_cols(diabetes_pca)^2)

# plot column standard coordinates with a unit circle underlaid
diabetes_pca %>%
  ggbiplot(aes(label = name), sec.axes = "cols", scale.factor = 3) +
  geom_rows_point(aes(color = group), alpha = .25) +
  geom_unit_circle(alpha = .5, scale.factor = 3) +
  geom_cols_vector() +
  geom_cols_text_radiate()
```

`geom_vector` *Vectors from the origin*

Description

`geom_vector()` renders arrows from the origin to points.

Usage

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

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.

`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`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.
`arrow`	Specification for arrows, as created by `grid::arrow()`, or else `NULL` for no arrows.
`...`	Additional arguments passed to `ggplot2::layer()`.
`na.rm`	Passed to `ggplot2::layer()`.
`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. `borders()`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Aesthetics

`geom_vector()` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `alpha`
- `colour`
- `linetype`
- `size`
- `group`

See Also

Other geom layers: `geom_axis()`, `geom_isoline()`, `geom_lineranges()`, `geom_origin()`, `geom_text_radiate()`, `geom_unit_circle()`

Examples

```
# compute unscaled row-principal components of scaled measurements
(iris_pca <- ordinate(iris, cols = 1:4, princomp))

# row-principal biplot with coordinate-wise standard deviations
iris_pca %>%
  ggbiplot(aes(color = Species)) +
  theme_bw() +
  scale_color_brewer(type = "qual", palette = 2) +
  geom_unit_circle() +
  geom_rows_point(alpha = .5) +
  geom_cols_vector(color = "#444444") +
  geom_cols_text_radiate(aes(label = name), color = "#444444") +
  ggtitle("Row-principal unscaled PCA biplot of Anderson iris measurements") +
  expand_limits(y = c(NA, 2))
```

`ggbiplot` *Biplots following the grammar of graphics*

Description

Build a biplot visualization from ordination data wrapped as a tbl_ord object.

Usage

```
ggbiplot(
  ordination = NULL,
  mapping = aes(x = 1, y = 2),
  axis.type = "interpolative",
  xlim = NULL,
  ylim = NULL,
  expand = TRUE,
  clip = "on",
  axis.percents = TRUE,
  sec.axes = NULL,
  scale.factor = NULL,
  scale_rows = NULL,
  scale_cols = NULL,
  ...
)

ord_aes(ordination, ...)
```

Arguments

`ordination`	A tbl_ord.
`mapping`	List of default aesthetic mappings to use for the biplot. The default assigns the first two coordinates to the aesthetics `x` and `y`. Other assignments must be supplied in each layer added to the plot.
`axis.type`	Character, partially matched; whether to build an `"interpolative"` (the default) or a `"predictive"` biplot. The latter requires that `x` and `y` are mapped to shared coordinates, that no other shared coordinates are mapped to, and inertia is conferred entirely onto one matrix factor. **NB:** This option is only implemented for linear techniques (ED, SVD, & PCA).
`xlim`	Limits for the x and y axes.
`ylim`	Limits for the x and y axes.
`expand`	If `TRUE`, the default, adds a small expansion factor to the limits to ensure that data and axes don't overlap. If `FALSE`, limits are taken exactly from the data or `xlim`/`ylim`.
`clip`	Should drawing be clipped to the extent of the plot panel? A setting of `"on"` (the default) means yes, and a setting of `"off"` means no. In most cases, the default of `"on"` should not be changed, as setting `clip = "off"` can cause unexpected results. It allows drawing of data points anywhere on the plot, including in the plot margins. If limits are set via `xlim` and `ylim` and some data points fall outside those limits, then those data points may show up in places such as the axes, the legend, the plot title, or the plot margins.
`axis.percents`	Whether to concatenate default axis labels with inertia percentages.
`sec.axes`	Matrix factor character to specify a secondary set of axes.
`scale.factor`	Numeric value used to scale the secondary axes against the primary axes; ignored if `sec.axes` is not specified.
`scale_rows, scale_cols`	Either the character name of a numeric variable in `get_*(ordination)` or a numeric vector of length `nrow(get_*(ordination))`, used to scale the coordinates of the matrix factors.
`...`	Additional arguments passed to `ggplot2::fortify()`; see `fortify.tbl_ord()`.

Details

`ggbiplot()` produces a ggplot object from a tbl_ord object `ordination`. The baseline object is the default unadorned `"ggplot"`-class object `p` with the following differences from what `ggplot2::ggplot()` returns:

- `p$mapping` is augmented with `.matrix = .matrix`, which expects either `.matrix = "rows"` or `.matrix = "cols"` from the biplot.
- `p$coordinates` is defaulted to `ggplot2::coord_equal()` in order to faithfully render the geometry of an ordination. The optional parameters `xlim`, `ylim`, `expand`, and `clip` are passed to `coord_equal()` and default to its **ggplot2** defaults.
- When `x` or `y` are mapped to coordinates of `ordination`, and if `axis.percents` is `TRUE`, `p$labels$x` or `p$labels$y` are defaulted to the coordinate names concatenated with the percentages of inertia captured by the coordinates.

- p is assigned the class "ggbiplot" in addition to "ggplot". This serves no functional purpose currently.

Furthermore, the user may feed single integer values to the x and y aesthetics, which will be interpreted as the corresponding coordinates in the ordination. Currently only 2-dimensional biplots are supported, so both x and y must take coordinate values.

ord_aes() is a convenience function that generates a full-rank set of coordinate aesthetics ..coord1, ..coord2, etc. mapped to the shared coordinates of the ordination object, along with any additional aesthetics that are processed internally by ggplot2::aes().

The axis.type parameter controls whether the biplot is interpolative or predictive, though predictive biplots are still experimental and limited to linear methods like PCA. Gower & Hand (1996) and Gower, Lubbe, & le Roux (2011) thoroughly explain the construction and interpretation of predictive biplots.

Value

A ggplot object.

Biplot layers

ggbiplot() uses ggplot2::fortify() internally to produce a single data frame with a .matrix column distinguishing the subjects ("rows") and variables ("cols"). The stat layers stat_rows() and stat_cols() simply filter the data frame to one of these two.

The geom layers geom_rows_*() and geom_cols_*() call the corresponding stat in order to render plot elements for the corresponding factor matrix. geom_dims_*() selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

References

Gower JC & Hand DJ (1996) *Biplots*. Chapman & Hall, ISBN: 0-412-71630-5.

Gower JC, Lubbe GS, & le Roux NJ (2011) *Understanding Biplots*. Wiley, ISBN: 978-0-470-01255-0. https://www.wiley.com/go/biplots

See Also

ggplot2::ggplot2(), on which ggbiplot() is built

Examples

```
# compute PCA of Anderson iris measurements
iris[, -5] %>%
  princomp(cor = TRUE) %>%
  as_tbl_ord() %>%
  confer_inertia(1) %>%
  mutate_rows(species = iris$Species) %>%
  mutate_cols(measure = gsub("\\.", " ", tolower(names(iris)[-5]))) %>%
  print() -> iris_pca

# row-principal biplot with rescaled secondary axis
iris_pca %>%
  ggbiplot(aes(color = species), sec.axes = "cols", scale.factor = 2) +
  theme_bw() +
  scale_color_brewer(type = "qual", palette = 2) +
  geom_rows_point() +
  geom_cols_vector(color = "#444444") +
  geom_cols_text_radiate(aes(label = measure), color = "#444444") +
  ggtitle(
    "Row-principal PCA biplot of Anderson iris measurements",
    "Variable loadings scaled to secondary axes"
  ) +
  expand_limits(y = c(-1, 3.5))
# Performance measures can be regressed on the artificial coordinates of
# ordinated vehicle specs. Because the ordination of specs ignores performance,
# these coordinates will probably not be highly predictive. The gradient of each
# performance measure along the artificial axes is visualized by projecting the
# regression coefficients onto the ordination biplot.

# scaled principal components analysis of vehicle specs
mtcars_specs_pca <- ordinate(
  mtcars, cols = c(cyl, disp, hp, drat, wt, vs, carb),
  model = ~ princomp(., cor = TRUE)
)
# data frame of vehicle performance measures
mtcars %>%
  subset(select = c(mpg, qsec)) %>%
  as.matrix() %>%
  print() -> mtcars_perf
# regress performance measures on principal components
lm(mtcars_perf ~ get_rows(mtcars_specs_pca)) %>%
  as_tbl_ord() %>%
  augment_ord() %>%
  print() -> mtcars_pca_lm
# regression biplot
ggbiplot(mtcars_specs_pca, aes(label = name),
         sec.axes = "rows", scale.factor = .5) +
  theme_minimal() +
  geom_rows_text(size = 3) +
  geom_cols_vector(data = mtcars_pca_lm) +
  geom_cols_text_radiate(data = mtcars_pca_lm) +
  expand_limits(x = c(-2.5, 2))

# multidimensional scaling based on a scaled cosine distance of vehicle specs
cosine_dist <- function(x) {
  x <- as.matrix(x)
  num <- x %*% t(x)
  denom_rt <- as.matrix(rowSums(x^2))
  denom <- sqrt(denom_rt %*% t(denom_rt))
  as.dist(1 - num / denom)
}
mtcars %>%
  subset(select = c(cyl, disp, hp, drat, wt, vs, carb)) %>%
  scale() %>%
  cosine_dist() %>%
  cmdscale(list. = TRUE) %>%
  tidy() %>%
  print() -> mtcars_specs_cmds
# regress performance measures on principal coordinates
lm(mtcars_perf ~ as.matrix(mtcars_specs_cmds[, 2:3])) %>%
  as_tbl_ord() %>%
  augment_ord() %>%
  print() -> mtcars_cmds_lm
# multidimensional scaling using `cmdscale_ord()`
mtcars %>%
  subset(select = c(cyl, disp, hp, drat, wt, vs, carb)) %>%
  scale() %>%
  cosine_dist() %>%
  cmdscale_ord() %>%
  as_tbl_ord() %>%
  augment_ord() %>%
  print() -> mtcars_specs_cmds_ord
# regression biplot
ggbiplot(mtcars_specs_cmds_ord, aes(label = name),
         sec.axes = "rows", scale.factor = 3) +
  theme_minimal() +
  geom_rows_text(size = 3) +
  geom_cols_vector(data = mtcars_cmds_lm) +
  geom_cols_text_radiate(data = mtcars_cmds_lm) +
  expand_limits(x = c(-2.25, 1.25), y = c(-2, 1.5))
# PCA of iris data
iris_pca <- ordinate(iris, cols = 1:4, prcomp, scale = TRUE)

# row-principal predictive biplot
iris_pca %>%
  augment_ord() %>%
  ggbiplot(axis.type = "predictive") +
  theme_bw() +
  scale_color_brewer(type = "qual", palette = 2) +
  geom_cols_axis(aes(label = name, center = center, scale = scale)) +
  geom_rows_point(aes(color = Species), alpha = .5) +
  ggtitle("Predictive biplot of Anderson iris measurements")
```

glass *Glass composition data*

Description

Sites, types, and compositions of glass samples from archaeological sites in Israel.

Usage

```
data(glass)
```

Format

A tibble with 68 cases and 16 variables:

Site site at which sample was found

Anal analysis identifier

Context furnace identifier

Form type of sample

SiO2, TiO2, Al2O3, FeO, MnO, MgO, CaO, Na2O, K2O, P2O5, Cl, SO3 normalized weight percent oxide of each component

Details

Chunks of unformed glass from several furnaces found at the primary Byzantine-era site of Bet Eli'ezer, along with samples from other sites with weaker evidence of glass-making (Apollonia and Dor) and and from an Islamic-era site (Banias), were analyzed using X-ray spectrometry to determine their major components.

Baxter & Freestone (2006) used these data to illustrate log-ratio analysis.

Source

Freestone &al (2000), Table 2.

References

Freestone IC, Gorin-Rosen Y, & Hughes MJ (2000) "Primary glass from Israel and the production of glass in Late Antiquity and the early Islamic period". *La route du verre: Ateliers primaires et secondaires du second millénaire av. J.-C. au Moyen Âge*: 65–83. https://pascal-francis.inist.fr/vibad/index.php?action=getRecordDetail&idt=1158762

Baxter MJ & Freestone IC (2006) "Log-Ratio Compositional Data Analysis in Archaeometry". *Archaeometry*, **48**(3): 511–531. doi: 10.1111/j.14754754.2006.00270.x

Examples

```
# subset glass data to one site and major components
head(glass)
glass_main <- subset(
  glass,
  Site == "Bet Eli'ezer",
  select = c("SiO2", "Na2O", "CaO", "Al2O3", "MgO", "K2O")
)
# format as a data frame with row names
glass_main <- as.data.frame(glass_main)
rownames(glass_main) <- subset(glass, Site == "Bet Eli'ezer")$Anal

# perform log-ratio analysis
glass_lra <- lra(glass_main, compositional = TRUE, weighted = FALSE)
# inspect LRA row and column coordinates
head(glass_lra$row.coords)
glass_lra$column.coords
```

```
# inspect singular values of LRA
glass_lra$sv

# plot samples and measurements in a biplot
biplot(
  x = glass_lra$row.coords %*% diag(glass_lra$sv),
  y = glass_lra$column.coords,
  xlab = "Sample (principal coord.)", ylab = ""
)
mtext("Component (standard coord.)", side = 4L, line = 3L)
```

`lda-ord` *Augmented implementation of linear discriminant analysis*

Description

This function replicates `MASS::lda()` with options to retain elements useful to the tbl_ord class and biplot calculations.

Usage

```
lda_ord(x, ...)

## S3 method for class 'formula'
lda_ord(formula, data, ..., subset, na.action)

## S3 method for class 'data.frame'
lda_ord(x, ...)

## S3 method for class 'matrix'
lda_ord(x, grouping, ..., subset, na.action)

## Default S3 method:
lda_ord(
  x,
  grouping,
  prior = proportions,
  tol = 1e-04,
  method = c("moment", "mle", "mve", "t"),
  CV = FALSE,
  nu = 5,
  ...,
  ret.x = FALSE,
  ret.grouping = FALSE,
  axes.scale = "unstandardized"
)

## S3 method for class 'lda_ord'
```

```
predict(
  object,
  newdata,
  prior = object$prior,
  dimen,
  method = c("plug-in", "predictive", "debiased"),
  ...
)
```

Arguments

`x`	(required if no formula is given as the principal argument.) a matrix or data frame or Matrix containing the explanatory variables.
`...`	arguments passed to or from other methods.
`formula`	A formula of the form `groups ~ x1 + x2 + ...` That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.
`data`	An optional data frame, list or environment from which variables specified in `formula` are preferentially to be taken.
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is for the procedure to fail. An alternative is `na.omit`, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
`grouping`	(required if no formula principal argument is given.) a factor specifying the class for each observation.
`prior`	the prior probabilities of class membership. If unspecified, the class proportions for the training set are used. If present, the probabilities should be specified in the order of the factor levels.
`tol`	A tolerance to decide if a matrix is singular; it will reject variables and linear combinations of unit-variance variables whose variance is less than `tol^2`.
`method`	`"moment"` for standard estimators of the mean and variance, `"mle"` for MLEs, `"mve"` to use `cov.mve`, or `"t"` for robust estimates based on a t distribution.
`CV`	If true, returns results (classes and posterior probabilities) for leave-one-out cross-validation. Note that if the prior is estimated, the proportions in the whole dataset are used.
`nu`	degrees of freedom for `method = "t"`.
`ret.x, ret.grouping`	Logical; whether to retain as attributes the data matrix (`x`) and the class assignments (`grouping`) on which LDA is performed. Methods like `predict()` access these objects by name in the parent environment, and retaining them as attributes prevents errors that arise if these objects are reassigned.
`axes.scale`	Character string indicating how to left-transform the `scaling` value when rendering biplots using `ggbiplot()`. Options include `"unstandardized"`, `"standardized"`, and `"contribution"`.

`object`	object of class `"lda"`
`newdata`	data frame of cases to be classified or, if `object` has a formula, a data frame with columns of the same names as the variables used. A vector will be interpreted as a row vector. If newdata is missing, an attempt will be made to retrieve the data used to fit the `lda` object.
`dimen`	the dimension of the space to be used. If this is less than `min(p, ng-1)`, only the first `dimen` discriminant components are used (except for `method="predictive"`), and only those dimensions are returned in `x`.

Details

Linear discriminant analysis relies on an eigendecomposition of the product $W^{-1}B$ of the inverse of the within-class covariance matrix W by the between-class covariance matrix B. This eigendecomposition can be motivated as the right (V) half of the singular value decomposition of the matrix of *Mahalanobis distances* between the cases after "sphering" (linearly transforming them so that the within-class covariance is the identity matrix). LDA are not traditionally represented as biplots, with some exceptions (Gardner & le Roux, 2005; Greenacre, 2010, p. 109–117).

LDA is implemented as `MASS::lda()` in the **MASS** package, in which the variables are transformed by a sphering matrix S (Venables & Ripley, 2003, p. 331–333). The returned element `scaling` contains the unstandardized *discriminant coefficients*, which define the discriminant scores of the cases and their centroids as linear combinations of the original variables.

The discriminant coefficients constitute one of several possible choices of axes for a biplot representation of the LDA. The slightly modified function `lda_ord()` provides additional options:

- The *standardized discriminant coefficients* are obtained by (re)scaling the coefficients by the variable standard deviations. These coefficients indicate the contributions of the variables to the discriminant scores after controlling for their variances (Orlov, 2013).
- The variables' *contributions* to the Mahalanobis variance along each discriminant axis are obtained by transforming the coefficients by the inverse of the sphering matrix S. Because the contribution biplot derives from the eigendecomposition of the Mahalanobis distance matrix, the projections of the centroids and cases onto the variable axes approximate their variable values after centering and sphering (Greenacre, 2013).

Value

Output from `MASS::lda()` with an additional preceding class 'lda_ord' and up to three attributes:

- the input data `x`, if `ret.x` = TRUE
- the class assignments `grouping`, if `ret.grouping` = TRUE
- if the parameter `axes.scale` is not 'unstandardized', a matrix `axes.scale` that encodes the transformation of the row space

References

Gardner S & le Roux NJ (2005) "Extensions of Biplot Methodology to Discriminant Analysis". *Journal of Classification* **22**(1): 59–86. doi: 10.1007/s0035700500067 `https://link.springer.com/article/10.1007/s00357-005-0006-7`

Greenacre MJ (2010) *Biplots in Practice*. Fundacion BBVA, ISBN: 978-84-923846. https://www.fbbva.es/microsite/multivariate-statistics/biplots.html

Venables WN & Ripley BD (2003) *Modern Applied Statistics with S*, Fourth Edition. Springer Science & Business Media, ISBN: 0387954570, 9780387954578. https://www.mimuw.edu.pl/~pokar/StatystykaMgr/Books/VenablesRipley_ModernAppliedStatisticsS02.pdf

Orlov K (2013) *Answer to* "Algebra of LDA. Fisher discrimination power of a variable and Linear Discriminant Analysis". CrossValidated, accessed 2019-07-26. https://stats.stackexchange.com/a/83114/68743

Greenacre M (2013) "Contribution Biplots". *Journal of Computational and Graphical Statistics*, **22**(1): 107–122. https://amstat.tandfonline.com/doi/full/10.1080/10618600.2012.702494

See Also

MASS::lda(), from which lda_ord() is adapted

Examples

```
# Anderson iris species data centroid
iris_centroid <- t(apply(iris[, 1:4], 2, mean))
# unstandardized discriminant coefficients: the discriminant axes are linear
# combinations of the centered variables
iris_lda <- lda_ord(iris[, 1:4], iris[, 5], axes.scale = "unstandardized")
# linear combinations of centered variables
print(sweep(iris_lda$means, 2, iris_centroid, "-") %*% get_cols(iris_lda))
# discriminant centroids
print(get_rows(iris_lda, elements = "active"))

# unstandardized coefficient LDA biplot
iris_lda %>%
  as_tbl_ord() %>%
  augment_ord() %>%
  mutate_rows(
    species = grouping,
    discriminant = ifelse(.element == "active", "centroid", "case")
  ) %>%
  ggbiplot() +
  theme_bw() +
  geom_rows_point(aes(
    color = grouping,
    size = discriminant, alpha = discriminant
  )) +
  geom_cols_vector(color = "#888888") +
  geom_cols_text_radiate(aes(label = name), size = 3) +
  scale_color_brewer(type = "qual", palette = 2) +
  ggtitle("Unstandardized coefficient biplot of iris LDA") +
  expand_limits(y = c(-3, 5))

# standardized discriminant coefficients: permit comparisons across the
# variables
iris_lda <- lda_ord(iris[, 1:4], iris[, 5], axes.scale = "standardized")
# standardized variable contributions to discriminant axes
```

```
iris_lda %>%
  as_tbl_ord() %>%
  augment_ord() %>%
  fortify(.matrix = "cols") %>%
  dplyr::mutate(variable = name) %>%
  tidyr::gather(discriminant, coefficient, LD1, LD2) %>%
  ggplot(aes(x = discriminant, y = coefficient, fill = variable)) +
  geom_bar(position = "dodge", stat = "identity") +
  labs(y = "Standardized coefficient", x = "Linear discriminant") +
  theme_bw() +
  coord_flip()
# standardized coefficient LDA biplot
iris_lda %>%
  as_tbl_ord() %>%
  augment_ord() %>%
  mutate_rows(
    species = grouping,
    discriminant = ifelse(.element == "active", "centroid", "case")
  ) %>%
  ggbiplot() +
  theme_bw() +
  geom_rows_point(aes(
    color = grouping,
    size = discriminant, alpha = discriminant
  )) +
  geom_cols_vector(color = "#888888") +
  geom_cols_text_radiate(aes(label = name), size = 3) +
  scale_color_brewer(type = "qual", palette = 2) +
  ggtitle("Standardized coefficient biplot of iris LDA") +
  expand_limits(y = c(-2, 3))

# variable contributions (de-sphered discriminant coefficients): recover the
# inner product relationship with the centered class centroids
iris_lda <- lda_ord(iris[, 1:4], iris[, 5], axes.scale = "contribution")
# symmetric square root of within-class covariance
C_W_eig <- eigen(cov(iris[, 1:4] - iris_lda$means[iris[, 5], ]))
C_W_sqrtinv <-
  C_W_eig$vectors %*% diag(1/sqrt(C_W_eig$values)) %*% t(C_W_eig$vectors)
# product of matrix factors (scores and loadings)
print(get_rows(iris_lda, elements = "active") %*% t(get_cols(iris_lda)))
# "asymmetric" square roots of Mahalanobis distances between variables
print(sweep(iris_lda$means, 2, iris_centroid, "-") %*% C_W_sqrtinv)
# contribution LDA biplot
iris_lda %>%
  as_tbl_ord() %>%
  augment_ord() %>%
  mutate_rows(
    species = grouping,
    discriminant = ifelse(.element == "active", "centroid", "case")
  ) %>%
  ggbiplot() +
  theme_bw() +
  geom_rows_point(aes(
    color = grouping,
    size = discriminant, alpha = discriminant
  )) +
  geom_cols_vector(color = "#888888") +
  geom_cols_text_radiate(aes(label = name), size = 3) +
  scale_color_brewer(type = "qual", palette = 2) +
  ggtitle("Contribution biplot of iris LDA") +
  expand_limits(y = c(-2, 3.5))
```

`list_tidiers` *Tidying methods for (additional) lists*

Description

Some unclassed lists of fixed structure, returned by other functions, are recognized by **broom**'s list tidiers. **ordr**'s updated list tidiers append additional checks for the list structures returned by `cmdscale()` and `cancor()`. If such structure is detected, then the new in-package tidiers are called.

Usage

```
## S3 method for class 'list'
tidy(x, ...)

## S3 method for class 'list'
glance(x, ...)
```

Arguments

`x`	A list, potentially representing an object that can be tidied.
`...`	Additionally, arguments passed to the tidying function.

Value

A tibble.

See Also

`generics::tidy()` `generics::glance()`

Other list tidiers: `cancor_tidiers`, `cmdscale_tidiers` `lra-ord` *Log-ratio analysis*

Description

Represent log-ratios between variables based on their values on a population of cases.

Usage

```
lra(x, compositional = FALSE, weighted = TRUE)

## S3 method for class 'lra'
print(x, nd = length(x$sv), n = 6L, ...)

## S3 method for class 'lra'
screeplot(x, main = deparse1(substitute(x)), ...)

## S3 method for class 'lra'
biplot(
  x,
  choices = c(1L, 2L),
  scale = c(0, 0),
  main = deparse1(substitute(x)),
  var.axes = FALSE,
  ...
)

## S3 method for class 'lra'
plot(x, main = deparse1(substitute(x)), ...)
```

Arguments

`x`	A numeric matrix or rectangular data set.
`compositional`	Logical; whether to normalize rows of `x` to sum to 1.
`weighted`	Logical; whether to weight rows and columns by their sums.
`nd`	Integer; number of shared dimensions to include in print.
`n`	Integer; number of rows of each factor to print.
`main, var.axes, ...`	Parameters passed to other plotting methods (in the case of `main`, after being `force()`d.
`choices`	Integer; length-2 vector specifying the components to plot.
`scale`	Numeric; values between 0 and 1 that control how inertia is conferred unto the points: Row (`i = 1L`) and column (`i = 2L`) coordinates are scaled by `sv ^ scale[[i]]`. If a single value `scale` is passed, it is assigned to the rows while `1 - scale` is assigned to the columns.

Details

Log-ratio analysis (LRA) is based on a double-centering of log-transformed data, usually weighted by row and column totals. The technique is suitable for positive-valued variables on a common scale (e.g. percentages). The distances between variables' coordinates (in the full-dimensional space) are their pairwise log-ratios. The distances between cases' coordinates are called their *log-ratio distances*, and the total variance is the weighted sum of their squares.

LRA is not implemented in standard R distributions but is a useful member of the ordination toolkit. This is a minimal implementation following Greenacre's (2010) exposition in Chapter 7.

Value

Given an $n*p$ data matrix and setting $r = min(n,p)$, `lra()` returns a list of class `"lra"` containing three elements:

- svThe $r - 1$ singular values
- row.coordsThe $n * (r - 1)$ matrix of row standard coordinates.
- column.coordsThe $p * (r - 1)$ matrix of column standard coordinates.
- row.weightsThe weights used to scale the row coordinates.
- column.weightsThe weights used to scale the column coordinates.

References

Greenacre MJ (2010) *Biplots in Practice*. Fundacion BBVA, ISBN: 978-84-923846. https://www.fbbva.es/microsite/multivariate-statistics/biplots.html

Examples

```
# U.S. 1973 violent crime arrests
head(USArrests)
# row and column subsets
state_examples <- c("Hawaii", "Mississippi", "North Dakota")
arrests <- c(1L, 2L, 4L)

# pairwise log-ratios of violent crime arrests for two states
arrest_pairs <- combn(arrests, 2L)
arrest_ratios <-
  USArrests[, arrest_pairs[1L, ]] / USArrests[, arrest_pairs[2L, ]]
colnames(arrest_ratios) <- paste(
  colnames(USArrests)[arrest_pairs[1L, ]], "/",
  colnames(USArrests)[arrest_pairs[2L, ]], sep = ""
)
arrest_logratios <- log(arrest_ratios)
arrest_logratios[state_examples, ]

# non-compositional log-ratio analysis
(arrests_lra <- lra(USArrests[, arrests]))
screeplot(arrests_lra)
biplot(arrests_lra, scale = c(1, 0))
```

```
# compositional log-ratio analysis
(arrests_lra <- lra(USArrests[, arrests], compositional = TRUE))
biplot(arrests_lra, scale = c(1, 0))
```

methods-cancor *Functionality for canonical correlations*

Description

These methods extract data from, and attribute new data to, objects of class "cancor_ord". This is a class introduced in this package to identify objects returned by cancor_ord(), which wraps stats::cancor().

Usage

```
## S3 method for class 'cancor_ord'
as_tbl_ord(x)

## S3 method for class 'cancor_ord'
recover_rows(x)

## S3 method for class 'cancor_ord'
recover_cols(x)

## S3 method for class 'cancor_ord'
recover_inertia(x)

## S3 method for class 'cancor_ord'
recover_coord(x)

## S3 method for class 'cancor_ord'
recover_conference(x)

## S3 method for class 'cancor_ord'
recover_supp_rows(x)

## S3 method for class 'cancor_ord'
recover_supp_cols(x)

## S3 method for class 'cancor_ord'
recover_aug_rows(x)

## S3 method for class 'cancor_ord'
recover_aug_cols(x)

## S3 method for class 'cancor_ord'
recover_aug_coord(x)
```

Arguments

`x`	An ordination object.

Details

The canonical coefficients (loadings) are obtained directly from the underlying singular value decomposition and constitute the active elements. If canonical scores are returned, then they and the structure correlations are made available as supplementary elements. **ordr** takes rows and columns from the intraset correlations `$xstructure` and `$ystructure`, on which no intertia is conferred; the interset correlations can be obtained by conferring inertia onto these.

A biplot of the canonical coefficients can be interpreted as approximating the X-Y inner product matrix, inversely weighted by the X and Y variances. The canonical scores and structure coefficients are available as supplementary points if returned by `cancor_ord()`. These can be used to create biplots of the case scores as linear combinations of loadings (the coefficients, in standard coordinates, overlaid with the scores) or of intraset and interset correlations with respect to either data set (the correlations with inertia conferred entirely onto rows or onto columns). Greenacre (1984) and ter Braak (1990) describe these families, though ter Braak recommends against the first.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

References

Greenacre MJ (1984) *Theory and applications of correspondence analysis*. London: Academic Press, ISBN 0-12-299050-1. `http://www.carme-n.org/?sec=books5`

ter Braak CJF (1990) "Interpreting canonical correlation analysis through biplots of structure correlations and weights". *Psychometrika* 55(3), 519–531. doi: 10.1007/BF02294765

See Also

Other methods for singular value decomposition-based techniques: `methods-correspondence`, `methods-lda`, `methods-lra`, `methods-mca`, `methods-prcomp`, `methods-princomp`, `methods-svd`

Examples

```
# data frame of life-cycle savings across countries
class(LifeCycleSavings)
head(LifeCycleSavings)
savings_pop <- LifeCycleSavings[, c("pop15", "pop75")]
savings_oec <- LifeCycleSavings[, c("sr", "dpi", "ddpi")]

# canonical correlation analysis with scores and correlations included
savings_cca <- cancor_ord(savings_pop, savings_oec, scores = TRUE)
savings_cca <- augment_ord(as_tbl_ord(savings_cca))
head(get_cols(savings_cca))
head(get_cols(savings_cca, elements = "score"))
get_rows(savings_cca, elements = "structure")
get_cols(savings_cca, elements = "structure")

# biplot of interset and intraset correlations with the population data
savings_cca %>%
  confer_inertia("cols") %>%
  ggbiplot(aes(label = name, color = .matrix)) +
  theme_bw() + theme_biplot() +
  geom_unit_circle() +
  geom_rows_vector(arrow = NULL, elements = "structure") +
  geom_cols_vector(arrow = NULL, elements = "structure", linetype = "dashed") +
  geom_rows_text(elements = "structure", hjust = "outward") +
  geom_cols_text(elements = "structure", hjust = "outward") +
  scale_color_brewer(limits = c("rows", "cols"), type = "qual") +
  expand_limits(x = c(-1, 1), y = c(-1, 1))

# biplot with scores as supplemental elements
savings_cca %>%
  confer_inertia("rows") %>%
  ggbiplot(aes(label = name), sec.axes = "cols", scale.factor = 5L) +
  theme_biplot() +
  geom_cols_vector(elements = "active") +
  geom_cols_text_radiate(elements = "active") +
  geom_rows_text(elements = "score", subset = seq(50L))
```

methods-cmds *Functionality for classical multidimensional scaling objects*

Description

These methods extract data from, and attribute new data to, objects of class "cmds_ord". This is a class introduced in this package to identify objects returned by cmdscale_ord(), which wraps stats::cmdscale().

Usage

```
## S3 method for class 'cmds_ord'
as_tbl_ord(x)

## S3 method for class 'cmds_ord'
recover_rows(x)

## S3 method for class 'cmds_ord'
recover_cols(x)

## S3 method for class 'cmds_ord'
```

```
recover_inertia(x)

## S3 method for class 'cmds_ord'
recover_coord(x)

## S3 method for class 'cmds_ord'
recover_conference(x)

## S3 method for class 'cmds_ord'
recover_aug_rows(x)

## S3 method for class 'cmds_ord'
recover_aug_cols(x)

## S3 method for class 'cmds_ord'
recover_aug_coord(x)
```

Arguments

`x`	An ordination object.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

See Also

Other methods for eigen-decomposition-based techniques: `methods-eigen`, `methods-factanal`

Examples

```
# 'dist' object (matrix of road distances) of large American cities
class(UScitiesD)
print(UScitiesD)

# use multidimensional scaling to infer artificial planar coordinates
UScitiesD %>%
  cmdscale_ord(k = 2) %>%
  as_tbl_ord() %>%
  print() -> usa_mds

# recover (equivalent) matrices of row and column artificial coordinates
get_rows(usa_mds)
get_cols(usa_mds)
```

```
# augment ordination with point names
(usa_mds <- augment_ord(usa_mds))

# reorient biplot to conventional compass
usa_mds %>%
  negate_ord(c(1, 2)) %>%
  ggbiplot() +
  geom_cols_text(aes(label = name), size = 3) +
  ggtitle("MDS biplot of distances between U.S. cities")
```

`methods-correspondence` *Functionality for correspondence analysis ('correspondence') objects*

Description

These methods extract data from, and attribute new data to, objects of class `"correspondence"` from the **MASS** package.

Usage

```
## S3 method for class 'correspondence'
as_tbl_ord(x)

## S3 method for class 'correspondence'
recover_rows(x)

## S3 method for class 'correspondence'
recover_cols(x)

## S3 method for class 'correspondence'
recover_inertia(x)

## S3 method for class 'correspondence'
recover_conference(x)

## S3 method for class 'correspondence'
recover_coord(x)

## S3 method for class 'correspondence'
recover_aug_rows(x)

## S3 method for class 'correspondence'
recover_aug_cols(x)

## S3 method for class 'correspondence'
recover_aug_coord(x)
```

Arguments

x	An ordination object.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

See Also

Other methods for singular value decomposition-based techniques: `methods-cancor`, `methods-lda`, `methods-lra`, `methods-mca`, `methods-prcomp`, `methods-princomp`, `methods-svd`

Examples

```
# table of hair and eye color data collapsed by sex
data(quine, package = "MASS")
class(quine)
head(quine)

# use correspondence analysis to construct row and column profiles
(quine_ca <- MASS::corresp(~ Age + Eth, data = quine))
(quine_ca <- as_tbl_ord(quine_ca))

# recover row and column profiles
get_rows(quine_ca)
get_cols(quine_ca)

# augment profiles with names, masses, distances, and inertias
(quine_ca <- augment_ord(quine_ca))
```

methods-eigen *Functionality for eigen-decompositions*

Description

These methods extract data from, and attribute new data to, objects of class `"eigen_ord"` returned by `eigen_ord()`.

Usage

```
## S3 method for class 'eigen_ord'
as_tbl_ord(x)

## S3 method for class 'eigen_ord'
recover_rows(x)

## S3 method for class 'eigen_ord'
recover_cols(x)

## S3 method for class 'eigen_ord'
recover_inertia(x)

## S3 method for class 'eigen_ord'
recover_coord(x)

## S3 method for class 'eigen_ord'
recover_conference(x)

## S3 method for class 'eigen_ord'
recover_aug_rows(x)

## S3 method for class 'eigen_ord'
recover_aug_cols(x)

## S3 method for class 'eigen_ord'
recover_aug_coord(x)
```

Arguments

`x`	An ordination object.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the ’tbl_ord’ class. Its methods determine what model classes it is allowed to wrap. It then provides ’tbl_ord’ methods with access to the recoverers and hence to the model components.

See Also

Other methods for eigen-decomposition-based techniques: `methods-cmds`, `methods-factanal`

Examples

```
# subset QS data to rank variables
qs_ranks <- subset(
  qswur_usa,
  complete.cases(qswur_usa),
  select = 8:13
)
head(qs_ranks)
# eigendecomposition of Kendall correlation matrix
qs_ranks %>%
  cor(method = "kendall") %>%
  eigen_ord() %>%
  print() -> qs_eigen

# recover eigenvectors
get_rows(qs_eigen)
identical(get_cols(qs_eigen), get_rows(qs_eigen))

# wrap as a 'tbl_ord' and augment with dimension names
augment_ord(as_tbl_ord(qs_eigen))
# decomposition returns pure eigenvectors
get_conference(qs_eigen)
```

methods-factanal *Functionality for factor analysis ('factanal') objects*

Description

These methods extract data from, and attribute new data to, objects of class `"factanal"` as returned by `stats::factanal()`.

Usage

```
## S3 method for class 'factanal'
as_tbl_ord(x)

## S3 method for class 'factanal'
recover_rows(x)

## S3 method for class 'factanal'
recover_cols(x)

## S3 method for class 'factanal'
recover_inertia(x)

## S3 method for class 'factanal'
recover_coord(x)

## S3 method for class 'factanal'
recover_conference(x)
```

```
## S3 method for class 'factanal'
recover_supp_rows(x)

## S3 method for class 'factanal'
recover_aug_rows(x)

## S3 method for class 'factanal'
recover_aug_cols(x)

## S3 method for class 'factanal'
recover_aug_coord(x)
```

Arguments

`x`	An ordination object.

Details

Factor analysis of a data matrix relies on an an eigendecomposition of its correlation matrix, whose eigenvectors (up to weighting) comprise the variable loadings. For this reason, both row and column recoverers retrieve the loadings and inertia is evenly distributed between them. When computed and returned by `stats::factanal()`, the case scores are accessible as supplementary elements. Redistribution of inertia commutes through both score calculations.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

See Also

Other methods for eigen-decomposition-based techniques: `methods-cmds`, `methods-eigen`

Examples

```
# data frame of Swiss fertility and socioeconomic indicators
class(swiss)
head(swiss)
# perform factor analysis
swiss_fa <- factanal(~ ., factors = 2L, data = swiss, scores = "regression")

# wrap as a 'tbl_ord' object
(swiss_fa <- as_tbl_ord(swiss_fa))

# recover loadings
get_rows(swiss_fa, elements = "active")
```

```
get_cols(swiss_fa)
# recover scores
head(get_rows(swiss_fa, elements = "score"))

# augment column loadings with uniquenesses
(swiss_fa <- augment_ord(swiss_fa))

# symmetric biplot
swiss_fa %>%
  ggbiplot() +
  theme_bw() +
  geom_cols_vector(aes(color = uniqueness)) +
  geom_cols_text_radiate(aes(label = name)) +
  expand_limits(x = c(-2, 2.5), y = c(-1.5, 2))
```

---

methods-kmeans *Functionality for k-means clustering ('kmeans') objects*

Description

These methods extract data from, and attribute new data to, objects of class "kmeans" as returned by `stats::kmeans()`.

Usage

```
## S3 method for class 'kmeans'
as_tbl_ord(x)

## S3 method for class 'kmeans'
recover_rows(x)

## S3 method for class 'kmeans'
recover_cols(x)

## S3 method for class 'kmeans'
recover_coord(x)

## S3 method for class 'kmeans'
recover_aug_rows(x)

## S3 method for class 'kmeans'
recover_aug_cols(x)

## S3 method for class 'kmeans'
recover_aug_coord(x)
```

Arguments

x	An ordination object.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

See Also

Other methods for idiosyncratic techniques: `methods-lm`

Examples

```
# data frame of Anderson iris species measurements
class(iris)
head(iris)
# compute 3-means clustering on scaled iris measurements
set.seed(5601L)
iris %>%
  subset(select = -Species) %>%
  scale() %>%
  kmeans(centers = 3) %>%
  print() -> iris_km

# visualize clusters using PCA
iris %>%
  subset(select = -Species) %>%
  prcomp() %>%
  as_tbl_ord() %>%
  mutate_rows(cluster = iris_km$cluster) %>%
  ggbiplot() +
  geom_rows_point(aes(color = factor(as.character(as.integer(cluster)),
                                     levels = as.character(seq(3L))))) +
  scale_color_brewer(type = "qual", name = "cluster")

# wrap as a 'tbl_ord' object
(iris_km_ord <- as_tbl_ord(iris_km))

# augment everything with names, observations with cluster assignment
(iris_km_ord <- augment_ord(iris_km_ord))

# summarize clusters with standard deviation
iris_km_ord %>%
  tidy() %>%
  transform(sdev = sqrt(withinss / size))

# discriminate between clusters 2 and 3
iris_km_ord %>%
  ggbiplot(aes(x = `2`, y = `3`), color = factor(.cluster)) +
  geom_jitter(stat = "rows", aes(shape = cluster), width = .2, height = .2) +
  geom_cols_axis(aes(color = `1`, label = name),
                 text_size = 2, text_dodge = .1,
                 label_size = 3, label_alpha = .5) +
  scale_x_continuous(expand = expansion(mult = .8)) +
  scale_y_continuous(expand = expansion(mult = .5)) +
  ggtitle(
    "Measurement loadings onto clusters 2 and 3",
    "Color indicates loadings onto cluster 1"
  )
```

`methods-lda` *Functionality for linear discriminant analysis ('lda') objects*

Description

These methods extract data from, and attribute new data to, objects of class `"lda"` and `"lda_ord"` as returned by `MASS::lda()` and `lda_ord()`.

Usage

```
## S3 method for class 'lda'
as_tbl_ord(x)

## S3 method for class 'lda_ord'
as_tbl_ord(x)

## S3 method for class 'lda'
recover_rows(x)

## S3 method for class 'lda_ord'
recover_rows(x)

## S3 method for class 'lda'
recover_cols(x)

## S3 method for class 'lda_ord'
recover_cols(x)

## S3 method for class 'lda'
recover_inertia(x)

## S3 method for class 'lda_ord'
recover_inertia(x)

## S3 method for class 'lda'
recover_coord(x)

## S3 method for class 'lda_ord'
```

```
recover_coord(x)

## S3 method for class 'lda'
recover_conference(x)

## S3 method for class 'lda_ord'
recover_conference(x)

## S3 method for class 'lda'
recover_aug_rows(x)

## S3 method for class 'lda_ord'
recover_aug_rows(x)

## S3 method for class 'lda'
recover_aug_cols(x)

## S3 method for class 'lda_ord'
recover_aug_cols(x)

## S3 method for class 'lda'
recover_aug_coord(x)

## S3 method for class 'lda_ord'
recover_aug_coord(x)

## S3 method for class 'lda'
recover_supp_rows(x)

## S3 method for class 'lda_ord'
recover_supp_rows(x)
```

Arguments

`x`	An ordination object.

Details

See lda-ord for details.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

See Also

Other methods for singular value decomposition-based techniques: `methods-cancor`, `methods-correspondence`, `methods-lra`, `methods-mca`, `methods-prcomp`, `methods-princomp`, `methods-svd`

Examples

```
# data frame of Anderson iris species measurements
class(iris)
head(iris)

# default (unstandardized discriminant) coefficients
lda_ord(iris[, 1:4], iris[, 5]) %>%
  as_tbl_ord() %>%
  print() -> iris_lda

# recover centroid coordinates and measurement discriminant coefficients
get_rows(iris_lda, elements = "active")
head(get_rows(iris_lda, elements = "score"))
get_cols(iris_lda)

# augment ordination with centroid and measurement names
augment_ord(iris_lda)
```

---

methods-lm *Functionality for linear model objects*

Description

These methods extract data from, and attribute new data to, objects of class "`lm`", "`glm`", and "`mlm`" as returned by `stats::lm()` and `stats::glm()`.

Usage

```
## S3 method for class 'lm'
as_tbl_ord(x)

## S3 method for class 'lm'
recover_rows(x)

## S3 method for class 'lm'
recover_cols(x)

## S3 method for class 'lm'
recover_coord(x)

## S3 method for class 'lm'
recover_aug_rows(x)
```

```
## S3 method for class 'lm'
recover_aug_cols(x)

## S3 method for class 'lm'
recover_aug_coord(x)

## S3 method for class 'glm'
recover_aug_rows(x)

## S3 method for class 'mlm'
recover_rows(x)

## S3 method for class 'mlm'
recover_cols(x)

## S3 method for class 'mlm'
recover_coord(x)

## S3 method for class 'mlm'
recover_aug_rows(x)

## S3 method for class 'mlm'
recover_aug_cols(x)

## S3 method for class 'mlm'
recover_aug_coord(x)
```

Arguments

`x`	An ordination object.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

See Also

Other methods for idiosyncratic techniques: `methods-kmeans`

Examples

```
# Motor Trend design and performance data
head(mtcars)
# regression analysis of performance measures on design specifications
```

```
mtcars_centered <- scale(mtcars, scale = FALSE)
mtcars_centered %>%
  as.data.frame() %>%
  lm(formula = mpg ~ wt + cyl) %>%
  print() -> mtcars_lm

# wrap as a 'tbl_ord' object
(mtcars_lm_ord <- as_tbl_ord(mtcars_lm))
# augment everything with names, predictors with observation stats
augment_ord(mtcars_lm_ord)
# calculate influences as the squares of weighted residuals
mutate_rows(augment_ord(mtcars_lm_ord), influence = wt.res^2)

# regression biplot with performance isolines
mtcars_lm_ord %>%
  augment_ord() %>%
  mutate_cols(center = attr(mtcars_centered, "scaled:center")[name]) %>%
  mutate_rows(influence = wt.res^2) %T>% print() %>%
  ggbiplot(aes(x = wt, y = cyl, intercept = `(Intercept)`)) +
  #theme_biplot() +
  geom_origin(marker = "circle", radius = unit(0.02, "snpc")) +
  geom_rows_point(aes(color = influence)) +
  geom_cols_vector() +
  geom_cols_isoline(aes(center = center), by = .5, hjust = -.1) +
  ggtitle(
    "Weight isolines with data colored by importance",
    "Regressing gas mileage onto weight and number of cylinders"
  )
```

methods-lra *Functionality for log-ratio analysis ('lra') objects*

Description

These methods extract data from, and attribute new data to, objects of class "lra", a class introduced in this package to organize the singular value decomposition of a double-centered log-transformed data matrix output by lra().

Usage

```
## S3 method for class 'lra'
as_tbl_ord(x)

## S3 method for class 'lra'
recover_rows(x)

## S3 method for class 'lra'
recover_cols(x)
```

```
## S3 method for class 'lra'
recover_inertia(x)

## S3 method for class 'lra'
recover_coord(x)

## S3 method for class 'lra'
recover_conference(x)

## S3 method for class 'lra'
recover_aug_rows(x)

## S3 method for class 'lra'
recover_aug_cols(x)

## S3 method for class 'lra'
recover_aug_coord(x)
```

Arguments

`x`	An ordination object.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

See Also

Other methods for singular value decomposition-based techniques: `methods-cancor`, `methods-correspondence`, `methods-lda`, `methods-mca`, `methods-prcomp`, `methods-princomp`, `methods-svd`

Examples

```
# data frame of violent crime arrests in the United States
class(USArrests)
head(USArrests)
# get state abbreviation data
state <- data.frame(
  name = state.name,
  abb = state.abb
)

# compute (non-compositional, unweighted) log-ratio analysis
USArrests %>%
  subset(select = -UrbanPop) %>%
  lra() %>%
  as_tbl_ord() %>%
  print() -> arrests_lra

# augment log-ratio profiles with names and join state abbreviations
arrests_lra %>%
  augment_ord() %>%
  left_join_rows(state, by = "name") %>%
  print() -> arrests_lra

# recover state and arrest profiles
head(get_rows(arrests_lra))
get_cols(arrests_lra)
# initially, inertia is conferred on neither factor
get_conference(arrests_lra)

# row-principal biplot
arrests_lra %>%
  confer_inertia("rows") %>%
  ggbiplot(aes(color = .matrix), sec.axes = "cols", scale.factor = 1/20) +
  scale_color_manual(values = c("tomato4", "turquoise4")) +
  theme_bw() +
  geom_rows_text(aes(label = abb), size = 3, alpha = .75) +
  geom_cols_polygon(fill = NA, linetype = "dashed") +
  geom_cols_text(aes(label = name, size = weight), fontface = "bold") +
  scale_size_area(guide = "none") +
  ggtitle(
    "Non-compositional LRA of violent crime arrest rates",
    "United States, 1973"
  ) +
  expand_limits(x = c(-.35)) +
  guides(color = "none")
```

methods-mca *Functionality for multiple correspondence analysis ('mca') objects*

Description

These methods extract data from, and attribute new data to, objects of class "mca" from the **MASS** package.

Usage

```
## S3 method for class 'mca'
as_tbl_ord(x)

## S3 method for class 'mca'
recover_rows(x)

## S3 method for class 'mca'
```

```
recover_cols(x)

## S3 method for class 'mca'
recover_inertia(x)

## S3 method for class 'mca'
recover_conference(x)

## S3 method for class 'mca'
recover_coord(x)

## S3 method for class 'mca'
recover_supp_rows(x)

## S3 method for class 'mca'
recover_aug_rows(x)

## S3 method for class 'mca'
recover_aug_cols(x)

## S3 method for class 'mca'
recover_aug_coord(x)
```

Arguments

`x`	An ordination object.

Details

Multiple correspondence analysis (MCA) relies on a singular value decomposition of the indicator matrix X of a table of several categorical variables, scaled by its column totals. `MASS::mca()` returns the SVD factors UD and V as the row weights `$fs`, on which the inertia is conferred, and the column coordinates `$cs`. The row coordinates `$rs` are obtained as XV and accessible as supplementary elements.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

See Also

Other methods for singular value decomposition-based techniques: `methods-cancor`, `methods-correspondence`, `methods-lda`, `methods-lra`, `methods-prcomp`, `methods-princomp`, `methods-svd`

Examples

```
# table of admissions and rejections from UC Berkeley
class(UCBAdmissions)
ucb_admissions <- as.data.frame(UCBAdmissions)
ucb_admissions <-
  ucb_admissions[rep(seq(nrow(ucb_admissions)), ucb_admissions$Freq), -4L]
head(ucb_admissions)
# perform multiple correspondence analysis
ucb_admissions %>%
  MASS::mca() %>%
  as_tbl_ord() %>%
  # augment profiles with names, masses, distances, and inertias
  augment_ord() %>%
  print() -> admissions_mca

# recover row and column coordinates and row weights
head(get_rows(admissions_mca, elements = "score"))
get_cols(admissions_mca)
head(get_rows(admissions_mca))

# column-standard biplot of factor levels
admissions_mca %>%
  ggbiplot() +
  theme_bw() + theme_biplot() +
  geom_origin() +
  #geom_rows_point(stat = "unique") +
  geom_cols_point(aes(color = factor, shape = factor)) +
  geom_cols_text_repel(aes(label = level, color = factor),
                       show.legend = FALSE) +
  scale_color_brewer(palette = "Dark2") +
  scale_size_area(guide = "none") +
  labs(color = "Factor level", shape = "Factor level")
```

---

methods-prcomp *Functionality for principal components analysis ('prcomp') objects*

Description

These methods extract data from, and attribute new data to, objects of class `"prcomp"` as returned by `stats::prcomp()`.

Usage

```
## S3 method for class 'prcomp'
as_tbl_ord(x)

## S3 method for class 'prcomp'
recover_rows(x)
```

```
## S3 method for class 'prcomp'
recover_cols(x)

## S3 method for class 'prcomp'
recover_inertia(x)

## S3 method for class 'prcomp'
recover_coord(x)

## S3 method for class 'prcomp'
recover_conference(x)

## S3 method for class 'prcomp'
recover_aug_rows(x)

## S3 method for class 'prcomp'
recover_aug_cols(x)

## S3 method for class 'prcomp'
recover_aug_coord(x)
```

Arguments

x	An ordination object.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

Author(s)

Emily Paul

See Also

Other methods for singular value decomposition-based techniques: `methods-cancor`, `methods-correspondence`, `methods-lda`, `methods-lra`, `methods-mca`, `methods-princomp`, `methods-svd`

Examples

```
# data frame of Anderson iris species measurements
class(iris)
head(iris)

# compute scaled row-principal components of scaled measurements
```

```
iris[, -5] %>%
  prcomp(scale = TRUE) %>%
  as_tbl_ord() %>%
  print() -> iris_pca

# recover observation principal coordinates and measurement standard coordinates
head(get_rows(iris_pca))
get_cols(iris_pca)

# augment measurements with names and scaling parameters
(iris_pca <- augment_ord(iris_pca))
```

`methods-princomp` *Functionality for principal components analysis ('princomp') objects*

Description

These methods extract data from, and attribute new data to, objects of class `"princomp"` as returned by `stats::princomp()`.

Usage

```
## S3 method for class 'princomp'
as_tbl_ord(x)

## S3 method for class 'princomp'
recover_rows(x)

## S3 method for class 'princomp'
recover_cols(x)

## S3 method for class 'princomp'
recover_inertia(x)

## S3 method for class 'princomp'
recover_coord(x)

## S3 method for class 'princomp'
recover_conference(x)

## S3 method for class 'princomp'
recover_aug_rows(x)

## S3 method for class 'princomp'
recover_aug_cols(x)

## S3 method for class 'princomp'
recover_aug_coord(x)
```

Arguments

x	An ordination object.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

Author(s)

Emily Paul

See Also

Other methods for singular value decomposition-based techniques: `methods-cancor`, `methods-correspondence`, `methods-lda`, `methods-lra`, `methods-mca`, `methods-prcomp`, `methods-svd`

Examples

```
# data frame of Anderson iris species measurements
class(iris)
head(iris)

# compute unscaled row-principal components of scaled measurements
iris[, -5] %>%
  princomp() %>%
  as_tbl_ord() %>%
  print() -> iris_pca

# recover observation principal coordinates and measurement standard coordinates
head(get_rows(iris_pca))
get_cols(iris_pca)

# augment measurement coordinates with names and scaling parameters
(iris_pca <- augment_ord(iris_pca))
```

`methods-svd` *Functionality for singular value decompositions*

Description

These methods extract data from, and attribute new data to, objects of class `"svd_ord"` returned by `svd_ord()`.

Usage

```
## S3 method for class 'svd_ord'
as_tbl_ord(x)

## S3 method for class 'svd_ord'
recover_rows(x)

## S3 method for class 'svd_ord'
recover_cols(x)

## S3 method for class 'svd_ord'
recover_inertia(x)

## S3 method for class 'svd_ord'
recover_coord(x)

## S3 method for class 'svd_ord'
recover_conference(x)

## S3 method for class 'svd_ord'
recover_aug_rows(x)

## S3 method for class 'svd_ord'
recover_aug_cols(x)

## S3 method for class 'svd_ord'
recover_aug_coord(x)
```

Arguments

`x`	An ordination object.

Value

The recovery generics `recover_*()` return core model components, distribution of inertia, supplementary elements, and intrinsic metadata; but they require methods for each model class to tell them what these components are.

The generic `as_tbl_ord()` returns its input wrapped in the 'tbl_ord' class. Its methods determine what model classes it is allowed to wrap. It then provides 'tbl_ord' methods with access to the recoverers and hence to the model components.

See Also

Other methods for singular value decomposition-based techniques: `methods-cancor`, `methods-correspondence`, `methods-lda`, `methods-lra`, `methods-mca`, `methods-prcomp`, `methods-princomp`

Examples

```
# matrix of U.S. personal expenditure data
```

```
class(USPersonalExpenditure)
print(USPersonalExpenditure)
# singular value decomposition into row and column coordinates
USPersonalExpenditure %>%
  svd_ord() %>%
  as_tbl_ord() %>%
  print() -> spend_svd

# recover matrices of row and column coordinates
get_rows(spend_svd)
get_cols(spend_svd)

# augment with row and column names
augment_ord(spend_svd)
# initial matrix decomposition confers no inertia to coordinates
get_conference(spend_svd)
```

`negation` *Negation of ordination axes*

Description

Negate the coordinates of a subset of ordination axes in both row and column singular vectors.

Usage

```
get_negation(x)

revert_negation(x)

negate_ord(x, negation = NULL)

negate_to_first_orthant(x, .matrix)
```

Arguments

`x`	A tbl_ord.
`negation`	Integer vector of coordinates to negate.
`.matrix`	A character string partially matched (lowercase) to several indicators for one or both matrices in a matrix decomposition used for ordination. The standard values are `"rows"`, `"cols"`, and `"dims"` (for both).

Details

For purposes of comparison and visualization, it can be useful to negate the (already artificial) coordinates of an ordination, either by fixed criteria or to better align with another basis (matrix) of coordinates. `negate_ord()` allows the user to negate specified coordinates of an ordination.

`get_negation()` accesses the negations of an ordination, an integer vector of `1`s and `-1`s stored as a `"negate"` attribute.

Value

`negate_ord()` and `negate_to_first_orthant()` return a tbl_ord with certain axes negated but the wrapped model unchanged. `get_negation()` returns the current negations. `revert_negation()` returns the tbl_ord without any manual negations.

A tbl_ord; the wrapped model is unchanged.

Examples

```
(pca <- ordinate(iris, cols = 1:4, prcomp))
ggbiplot(pca) + geom_rows_point() + geom_cols_vector()

# manually negate second coordinate
(pca_neg <- negate_ord(pca, 2))
ggbiplot(pca_neg) + geom_rows_point() + geom_cols_vector()

# NB: 'prcomp' method takes precedence; negations are part of the wrapper
biplot(pca)
biplot(pca_neg)

# negate to the first orthant
(pca_orth <- negate_to_first_orthant(pca, "v"))
get_negation(pca_orth)
```

`ordinate` *Fit an ordination model to a data object*

Description

This is a convenience function to fit an ordination model to a data object, wrap the result as a tbl_ord, and annotate this output with metadata from the model and possibly from the data.

Usage

```
ordinate(x, model, ...)

## Default S3 method:
ordinate(x, model, ...)

## S3 method for class 'array'
ordinate(x, model, ...)

## S3 method for class 'table'
ordinate(x, model, ...)

## S3 method for class 'data.frame'
ordinate(x, model, cols, augment, ...)

## S3 method for class 'dist'
ordinate(x, model, ...)
```

Arguments

`x`	A data object to be passed to the `model`, such as an array, table, data.frame, or stats::dist.
`model`	An ordination function whose output is coercible to class 'tbl_ord', or a symbol or character string (handled by `match.fun()`). Alternatively, a formula `~ fun(., ...)` where `fun` is such a function and other arguments are explicit, which will be evaluated with `x` in place of `.`.
`...`	Additional arguments passed to `model`.
`cols`	<`tidy-select`> If `x` is a data frame, columns to pass to `model`. If missing, all columns are used.
`augment`	<`tidy-select`> If `x` is a data frame, columns to augment to the row data of the ordination. If missing, all columns not included in `cols` will be augmented.

Details

The default method fits the specified model to the provided data object, wraps the result as a tbl_ord, and augments this output with any intrinsic metadata from the model via `augment_ord()`.

The default method is used for most classes, though this may change in future. The data.frame method allows the user to specify what columns to include in the model and what columns with which to annotate the output.

Value

An augmented tbl_ord.

Examples

```
# LRA of arrest data
ordinate(USArrests, cols = c(Murder, Rape, Assault), lra)

# CMDS of inter-city distance data
ordinate(UScitiesD, cmdscale_ord, k = 3L)

# PCA of iris data
ordinate(iris, princomp, cols = -Species, augment = c(Sepal.Width, Species))
ordinate(iris, cols = 1:4, ~ prcomp(., center = TRUE, scale. = TRUE))

# CA of hair & eye color data
haireye <- as.data.frame(rowSums(HairEyeColor, dims = 2L))
ordinate(haireye, MASS::corresp, cols = everything())

# FA of Swiss social data
ordinate(swiss, model = factanal, factors = 2L, scores = "Bartlett")

# LDA of iris data
ordinate(iris, ~ lda_ord(.[, 1:4], .[, 5], ret.x = TRUE))

# CCA of savings data
ordinate(
  LifeCycleSavings[, c("pop15", "pop75")],
  # second data set must be handled as an additional parameter to `model`
  y = LifeCycleSavings[, c("sr", "dpi", "ddpi")],
  model = cancor_ord, scores = TRUE
)
```

---

ordr — **ordr** *package*

Description

This is a **tidyverse** extension for handling, manipulating, and visualizing ordination models with consistent conventions and in a tidy workflow.

Details

This package is designed to integrate ordination analysis and biplot visualization into a **tidyverse** workflow. It is inspired in particular by the extensions **ggbiplot** and **tidygraph**.

The package consists in several modules:

- the 'tbl_ord' class, a wrapper for various ordination object classes
- extracting augmentation for the factors of an ordination
- using dplyr-verbs to add annotation to the factors
- adjusting the conference of inertia between the factors
- methods of the above generics for several widely-used object classes
- convenient formatting of ordination objects
- `ggbiplot()`, a **ggplot2** extension for rendering biplots
- additional stats and geoms for biplots

Ordinations and biplots

Ordination encompasses a variety of techniques for data compression, dimension reduction, feature extraction, and visualization. Well-known ordination techniques are predominantly unsupervised and include principal components analysis, multidimensional scaling, and correspondence analyis (Podani, 2000, Chapter 7; Palmer, n.d.). These methods are theoretically grounded in geometric data analysis (Le Roux & Rouanet, 2004) and powered by the matrix factorizations described below. A variety of other techniques may also be viewed, or treated using the same tools, as ordination, including linear regression, linear discriminant analysis, k-means clustering, and non-negative matrix factorization.

Biplots are two-layered scatterplots widely used to visualize unsupervised SVD-based ordinations. Gabriel (1971) introduced biplots to represent the scores and loadings of PCA on a single set of axes. They have also been used to visualize generalized linear regression and linear discriminant analysis (Greenacre, 2010) and can adapted to any 2-factor matrix decomposition.

Singular value decomposition

The most popular ordination techniques use singular value decomposition (SVD) to factor a data matrix X into a product $X = UDV'$ of two orthogonal (rotation) matrices U and V and a diagonal (scaling) matrix D, with V' the transpose of V. In most cases, the data matrix X is transformed from an original data matrix, e.g. by centering, scaling, double-centering, or log-transforming. The SVD introduces a set of shared orthogonal coordinates in which U encodes the rows of X and V encodes the columns of X. The singular values in D are the variances of X along each of these coordinates, and they proceed in decreasing order, so that the first r (for "rank") columns of U and of V produce a geometrically optimized approximation to X.

Biplots of SVD-based ordinations usually plot the rows and columns of X on these r coordinate axes. For an SVD-based biplot to be truly geometric, the total variance contained in D must be conferred onto U or V, or distributed over both (Orlov, 2015). When D is conferred onto U, the rows of X are represented by the rows of UD, and their distances in the biplot approximate their distances in the original column space of X. Meanwhile, the columns of X are represented by the rows of V. These are unit vectors in the full space of shared coordinates, so their squared lengths in the biplot indicate the proportion of their variance captured by the biplot axes and their cosines with each other approximate the correlations between the columns. Finally, the projection of a row's coordinates (point) onto a column's coordinates (vector) approximates the corresponding entry of X.

References

Podani J (2000) "Ordination". *Introduction to the Exploration of Multivariate Biological Data* Chapter 7, 215–284. Backhuys Publishers, ISBN 90-5782-067-6. `https://web.archive.org/web/20200221000313/http://ramet.elte.hu/~podani/books.html`

Palmer M *Ordination Methods for Ecologists*. Website, accessed 2019-07-12. `http://ordination.okstate.edu/`

Le Roux B & Rouanet H (2004) *Geometric Data Analysis: From Correspondence Analysis to Stsructured Data Analysis*. Springer Dordrecht, ISBN: 978-1-4020-2236-4. doi: 10.1007/14020-22360 `https://link.springer.com/book/10.1007/1-4020-2236-0`

Gabriel KR (1971) "The biplot graphic display of matrices with application to principal component analysis". *Biometrika* 58(3), 453–467. doi: 10.1093/biomet/58.3.453

Greenacre MJ (2010) *Biplots in Practice*. Fundacion BBVA, ISBN: 978-84-923846. `https://www.fbbva.es/microsite/multivariate-statistics/biplots.html`

Orlov K (2015) *Answer to* "PCA and Correspondence analysis in their relation to Biplot". Cross-Validated, accessed 2019-07-12. `https://stats.stackexchange.com/a/141755/68743`

`ordr-ggproto` *ggproto classes created and adapted for ordr*

Description

In addition to geometric element layers (geoms) based on base-**ggplot2** layers like `geom_point()` but specified to matrix factors as `geom_row_point()`, **ordr** introduces ggproto classes for some additional geometric elements commonly used in biplots. The factor-specific geoms invoke the statistical transformation layers (stats) `stat_rows()` and `stat_cols()`, which specify the matrix factor. Because each ggplot layer consists of only one stat and one geom, this necessitates that ggproto classes for new stats must also come in `*Rows` and `*Cols` flavors.

See Also

`ggplot2::ggplot2-ggproto` and ggplot2::ggproto for explanations of base ggproto classes in **ggplot2** and how to create new ones.

`plot.tbl_ord` *Plot and biplot methods for 'tbl_ord' objects*

Description

Adapt **stats** 'prcomp' and 'princomp' methods for `plot()`, `screeplot()`, and `biplot()` generics to 'tbl_ord' objects.

Usage

```
## S3 method for class 'tbl_ord'
plot(x, main = deparse(substitute(x)), ...)
```

Arguments

`x`	A 'tbl_ord' object.
`main`	A main title for the plot, passed to other methods (included to enable parsing of object name).
`...`	Additional arguments passed to other methods.

Details

These methods defer to any `plot()` and `biplot()` methods for the original, underlying model classes of 'tbl_ord' objects. If none are found: Following the examples of `stats::plot.prcomp()` and `stats::plot.princomp()`, `plot.tbl_ord()` calls on `stats::screeplot()` to produce a scree plot of the decomposition of variance in the singular value decomposition. Similarly following `stats::biplot.prcomp()` and `stats::biplot.princomp()`, `biplot.tbl_ord()` produces a biplot of both rows and columns, using text labels when available and markers otherwise, with rows and columns distinguished by color and no additional annotation (e.g. vectors). The biplot confers inertia according to `get_conference()` unless the proportions do not sum to 1, in which case it produces a symmetric biplot (inertia conferred equally to rows and columns).

Value

Nothing, but a plot is produced on the current graphics device.

Examples

```
# note: behavior depends on installed packages with class-specific methods

# class 'prcomp'
iris_pca <- prcomp(iris[, -5L], scale = TRUE)
iris_pca_ord <- as_tbl_ord(iris_pca)
plot(iris_pca)
plot(iris_pca_ord)
screeplot(iris_pca)
screeplot(iris_pca_ord)
biplot(iris_pca)
biplot(iris_pca_ord)

# class 'correspondence'
haireye_ca <- MASS::corresp(rowSums(HairEyeColor, dims = 2L), nf = 2L)
haireye_ca_ord <- as_tbl_ord(haireye_ca)
plot(haireye_ca)
plot(haireye_ca_ord)
# no `screeplot()` method for class 'correspondence'
screeplot(haireye_ca_ord)
biplot(haireye_ca)
biplot(haireye_ca_ord)
```

`qswur_usa` *U.S. university rankings*

Description

Classifications and rankings of U.S. universities for the years 2017–2020.

Usage

```
data(qswur_usa)
```

Format

A tibble of 13 variables on 612 cases:

year year of rankings

institution institution of higher learning

size size category of institution

focus subject range of institution

res research intensity of institution

age age classification of institution

status status of institution

rk_academic rank by academic reputation

rk_employer rank by employer reputation

rk_ratio rank by faculty–student ratio

rk_citations rank by citations per faculty

rk_intl_faculty rank by international faculty ratio

rk_intl_students rank by international student ratio

Details

Ranking data were obtained from the public QS website.

Source

Quacquarelli Symonds (2021).

References

Quacquarelli Symonds (2021) "University Rankings". TopUniversities.com https://www.topuniversities.com/university-rankings.

Examples

```
# subset QS data to rank variables
head(qswur_usa)
qs_ranks <- subset(
  qswur_usa,
  complete.cases(qswur_usa),
  select = 8:13
)
# calculate Kendall correlation matrix
qs_cor <- cor(qs_ranks, method = "kendall")

# calculate eigendecomposition
qs_eigen <- eigen_ord(qs_cor)
# view correlations as cosines of biplot vectors
biplot(x = qs_eigen$vectors, y = qs_eigen$vectors, col = c(NA, "black"))
```

---

recoverers *Access factors, coordinates, and metadata from ordination objects*

Description

These functions return information about the matrix factorization underlying an ordination.

Usage

```
recover_rows(x)

recover_cols(x)

## Default S3 method:
recover_rows(x)

## Default S3 method:
recover_cols(x)

## S3 method for class 'data.frame'
recover_rows(x)

## S3 method for class 'data.frame'
recover_cols(x)

get_rows(x, elements = "all")

get_cols(x, elements = "all")

## S3 method for class 'tbl_ord'
as.matrix(x, ..., .matrix, elements = "all")

recover_inertia(x)

## Default S3 method:
recover_inertia(x)

recover_coord(x)

## Default S3 method:
recover_coord(x)

## S3 method for class 'data.frame'
recover_coord(x)

get_coord(x)

get_inertia(x)

## S3 method for class 'tbl_ord'
dim(x)
```

Arguments

`x`	An object of class 'tbl_ord'.
`elements`	Character vector; which elements of each factor for which to render graphi-

	cal elements. One of "all" (the default), "active", or any supplementary element type defined by the specific class methods (e.g. "score" for 'factanal', 'lda_ord', and 'cancord_ord' and "intraset" and "interset" for 'cancor_ord').
...	Additional arguments from base::as.matrix(); ignored.
.matrix	A character string partially matched (lowercase) to several indicators for one or both matrices in a matrix decomposition used for ordination. The standard values are "rows", "cols", and "dims" (for both).

Details

The recover_*() S3 methods extract one or both of the row and column matrix factors that constitute the original ordination. These are interpreted as the case scores (rows) and the variable loadings (columns). The get_*() functions optionally (and by default) include any supplemental observations (see supplementation).

The recover_*() functions are generics that require methods for each ordination class. They are not intended to be called directly but are exported so that users can query methods("recover_*").

get_coord() retrieves the names of the coordinates shared by the matrix factors on which the original data were ordinated, and get_inertia() retrieves a vector of the inertia with these names. dim() retrieves the dimensions of the row and column factors, which reflect the dimensions of the matrix they reconstruct—**not** the original data matrix. (This matters for techniques that rely on eigendecomposition, for which the decomposed matrix is square.)

Value

The recover_*() functions are generics whose methods return base R objects retrieved from the model wrapped in the 'tbl_ord' class:

- rows: the row matrix as stored in the model
- cols: the column matrix as stored in the model
- inertia: the vector of eigen-values or squared singular values, often known by other names depending on the model
- coord: names for the artificial axes, from the model if available The get_*() functions (which are not generics) return modifications of these objects:
- rows: the recovered rows, adjusted according to any negation of axes or conference of inertia
- cols: the recovered columns, adjusted according to any negation of axes or conference of inertia
- inertia: the recovered inertia, named by the recovered coordinates
- coord: the recovered coordinates (unmodified) dim() returns the dimensions of the decomposed matrix, i.e. the numbers of rows of recover_rows() and of recover_cols().

See Also

Other generic recoverers: augmentation, conference, supplementation

Examples

```
# example ordination: LRA of U.S. arrests data
arrests_lra <- ordinate(USArrests, cols = c(Murder, Rape, Assault), lra)

# extract matrix factors
as.matrix(arrests_lra, .matrix = "rows")
as.matrix(arrests_lra, .matrix = "cols")
# special named functions
get_rows(arrests_lra)
get_cols(arrests_lra)
# get dimensions of underlying matrix factorization (not of original data)
dim(arrests_lra)

# get names of artificial / latent coordinates
get_coord(arrests_lra)
# get distribution of inertia
get_inertia(arrests_lra)
```

---

stat_center *Compute geometric centers and spreads for ordination factors*

Description

Compute geometric centers and spreads for ordination factors

Usage

```
stat_center(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  fun.data = NULL,
  fun.center = NULL,
  fun.min = NULL,
  fun.max = NULL,
  fun.args = list()
)

stat_star(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  fun.data = NULL,
  fun.center = NULL,
  fun.args = list()
)
```

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`geom`	The geometric object to use display the data
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`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. `borders()`.
`...`	Additional arguments passed to `ggplot2::layer()`.
`fun.data, fun.center, fun.min, fun.max, fun.args`	Functions and arguments treated as in `ggplot2::stat_summary()`, with `fun.center`, `fun.min`, and `fun.max` behaving as `fun.y`, `fun.ymin`, and `fun.ymax`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Ordination aesthetics

The convenience function `ord_aes()` can be used to incorporate all coordinates of the ordination model into a statistical transformation. It maps the coordinates to the custom aesthetics `..coord1`, `..coord2`, etc.

Some transformations, e.g. `stat_center()`, are commutative with projection to the 'x' and 'y' coordinates. If they detect aesthetics of the form `..coord[0-9]+`, then `..coord1` and `..coord2` are converted to `x` and `y` while any remaining are ignored.

Other transformations, e.g. `stat_spantree()`, yield different results in a planar biplot when they are computer before or after projection. If such a stat layer detects these aesthetics, then the lot of them are used in the transformation.

In either case, the stat layer returns a data frame with position aesthetics `x` and `y`.

See Also

Other stat layers: `stat_chull()`, `stat_cone()`, `stat_scale()`, `stat_spantree()`

Examples

```
# scaled PCA of Anderson iris measurements
iris[, -5] %>%
  princomp(cor = TRUE) %>%
  as_tbl_ord() %>%
  mutate_rows(species = iris$Species) %>%
  print() -> iris_pca

# row-principal biplot with centroid-based stars
iris_pca %>%
  ggbiplot(aes(color = species)) +
  theme_bw() +
  scale_color_brewer(type = "qual", palette = 2) +
  stat_rows_star(alpha = .5, fun.center = "mean") +
  geom_rows_point(alpha = .5) +
  stat_rows_center(fun.center = "mean", size = 4, shape = 1L) +
  ggtitle(
    "Row-principal PCA biplot of Anderson iris measurements",
    "Segments connect each observation to its within-species centroid"
  )
```

`stat_chull` *Restrict geometric data to boundary points for its convex hull*

Description

As used in a **ggplot2** vignette, this stat layer restricts a dataset with `x` and `y` variables to the points that lie on its convex hull. The biplot extension restricts each matrix factor to its own hull.

Usage

```
stat_chull(
  mapping = NULL,
  data = NULL,
  geom = "polygon",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

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.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`geom`	The geometric object to use display the data
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`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. `borders()`.
`...`	Additional arguments passed to `ggplot2::layer()`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Ordination aesthetics

The convenience function `ord_aes()` can be used to incorporate all coordinates of the ordination model into a statistical transformation. It maps the coordinates to the custom aesthetics `..coord1`, `..coord2`, etc.

Some transformations, e.g. `stat_center()`, are commutative with projection to the 'x' and 'y' coordinates. If they detect aesthetics of the form `..coord[0-9]+`, then `..coord1` and `..coord2` are converted to `x` and `y` while any remaining are ignored.

Other transformations, e.g. `stat_spantree()`, yield different results in a planar biplot when they are computer before or after projection. If such a stat layer detects these aesthetics, then the lot of them are used in the transformation.

In either case, the stat layer returns a data frame with position aesthetics `x` and `y`.

See Also

Other stat layers: `stat_center()`, `stat_cone()`, `stat_scale()`, `stat_spantree()`

Examples

```
# correspondence analysis of combined female and male hair and eye color data
HairEyeColor %>%
  rowSums(dims = 2L) %>%
  MASS::corresp(nf = 2L) %>%
  as_tbl_ord() %>%
  augment_ord() %>%
  print() -> hec_ca
# inertia across artificial coordinates (all singular values < 1)
get_inertia(hec_ca)

# in row-principal biplot, row coordinates are weighted averages of columns
hec_ca %>%
  confer_inertia("rows") %>%
  ggbiplot(aes(color = .matrix, fill = .matrix, shape = .matrix)) +
  theme_bw() +
  stat_cols_chull(alpha = .1) +
  geom_cols_point() +
  geom_rows_point() +
  ggtitle("Row-principal CA of hair & eye color")
# in column-principal biplot, column coordinates are weighted averages of rows
hec_ca %>%
  confer_inertia("cols") %>%
  ggbiplot(aes(color = .matrix, fill = .matrix, shape = .matrix)) +
  theme_bw() +
  stat_rows_chull(alpha = .1) +
  geom_rows_point() +
  geom_cols_point() +
  ggtitle("Column-principal CA of hair & eye color")
```

stat_cone *Restrict geometric data to boundary points for its conical hull*

Description

This stat layer restricts a dataset with x and y variables to the points that lie on its conical hull (other than the origin).

Usage

```
stat_cone(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  origin = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

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.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, 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 data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
origin	Logical; whether to include the origin with the transformed data. Defaults to FALSE.
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. `borders()`.
`...`	Additional arguments passed to `ggplot2::layer()`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Ordination aesthetics

The convenience function `ord_aes()` can be used to incorporate all coordinates of the ordination model into a statistical transformation. It maps the coordinates to the custom aesthetics `..coord1`, `..coord2`, etc.

Some transformations, e.g. `stat_center()`, are commutative with projection to the 'x' and 'y' coordinates. If they detect aesthetics of the form `..coord[0-9]+`, then `..coord1` and `..coord2` are converted to `x` and `y` while any remaining are ignored.

Other transformations, e.g. `stat_spantree()`, yield different results in a planar biplot when they are computer before or after projection. If such a stat layer detects these aesthetics, then the lot of them are used in the transformation.

In either case, the stat layer returns a data frame with position aesthetics `x` and `y`.

See Also

Other stat layers: `stat_center()`, `stat_chull()`, `stat_scale()`, `stat_spantree()`

Examples

```
# centered principal components analysis of U.S. personal expenditure data
USPersonalExpenditure %>%
  prcomp() %>%
  as_tbl_ord() %>%
  augment_ord() %>%
  # allow radiating text to exceed plotting window
  ggbiplot(aes(label = name), clip = "off",
           sec.axes = "cols", scale.factor = 50) +
  geom_rows_label(size = 3) +
  geom_cols_vector() +
  # omit labels in the conical hull without the origin
  stat_cols_cone(linetype = "dotted") +
  geom_cols_text_radiate(stat = "cone") +
  ggtitle(
    "U.S. Personal Expenditure data, 1940-1960",
    "Row-principal biplot of centered PCA"
  )
```

---

stat_rows *Render plot elements for one matrix of an ordination*

Description

These stats merely tell `ggplot2::ggplot()` which factor of an ordination to pull data from for a plot layer. They are invoked internally by the various `geom_*_*()` layers.

Usage

```
stat_rows(
  mapping = NULL,
  data = data,
  geom = "point",
  position = "identity",
  subset = NULL,
  elements = "all",
  ...,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_cols(
  mapping = NULL,
  data = data,
  geom = "axis",
  position = "identity",
  subset = NULL,
  elements = "all",
  ...,
  show.legend = NA,
  inherit.aes = TRUE
)
```

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.

`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`geom`	The geometric object to use display the data
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`subset`	An integer, logical, or character vector indicating a subset of rows or columns for which to render graphical elements. NB: Internally, the `subset` will be taken from the rows of the fortified 'tbl_ord' comprising rows from only one of the matrix factors. It is still possible to pass a formula to the `data` parameter, but it will act on the fortified data *before* it has been restricted to one matrix factor.
`elements`	Character vector; which elements of each factor for which to render graphical elements. One of `"all"` (the default), `"active"`, or any supplementary element type defined by the specific class methods (e.g. `"score"` for 'factanal', 'lda_ord', and 'cancord_ord' and `"intraset"` and `"interset"` for 'cancor_ord').
`...`	Additional arguments passed to `ggplot2::layer()`.
`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. `borders()`.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

See Also

Other biplot layers: `biplot-geoms`, `biplot-stats`

Examples

```
# FA of Swiss social data
swiss_fa <-
  ordinate(swiss, model = factanal, factors = 2L, scores = "regression")
# active and supplementary elements
get_rows(swiss_fa, elements = "active")
head(get_rows(swiss_fa, elements = "score"))

# biplot using element filters and selection
# (note that filter precedes selection)
ggbiplot(swiss_fa) +
  geom_rows_point(elements = "score") +
  geom_rows_text(aes(label = name), elements = "score", subset = c(1, 4, 18)) +
  scale_alpha_manual(values = c(0, 1), guide = "none") +
  geom_cols_vector() +
  geom_cols_text_radiate(aes(label = name))
```

---

stat_scale *Multiply artificial coordinates by a scale factor*

Description

Multiply artificial coordinates by a scale factor

Usage

```
stat_scale(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...,
  mult = 1
)
```

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.
data	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, 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 `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`geom`	The geometric object to use display the data
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`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. `borders()`.
`...`	Additional arguments passed to `ggplot2::layer()`.
`mult`	Numeric value used to scale the coordinates.

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Ordination aesthetics

The convenience function `ord_aes()` can be used to incorporate all coordinates of the ordination model into a statistical transformation. It maps the coordinates to the custom aesthetics `..coord1`, `..coord2`, etc.

Some transformations, e.g. `stat_center()`, are commutative with projection to the 'x' and 'y' coordinates. If they detect aesthetics of the form `..coord[0-9]+`, then `..coord1` and `..coord2` are converted to `x` and `y` while any remaining are ignored.

Other transformations, e.g. `stat_spantree()`, yield different results in a planar biplot when they are computer before or after projection. If such a stat layer detects these aesthetics, then the lot of them are used in the transformation.

In either case, the stat layer returns a data frame with position aesthetics `x` and `y`.

See Also

Other stat layers: `stat_center()`, `stat_chull()`, `stat_cone()`, `stat_spantree()`

stat_spantree *Calculate a minimum spanning tree among cases or variables*

Description

This stat layer identifies the $n - 1$ pairs among n points that form a minimum spanning tree, then calculates the segments between these poirs in the two dimensions x and y.

Usage

```
stat_spantree(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  position = "identity",
  engine = "vegan",
  method = "euclidean",
  show.legend = NA,
  inherit.aes = TRUE,
  check.aes = TRUE,
  check.param = TRUE,
  ...
)
```

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.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, 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 data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
engine	A single character string specifying the package implementation to use; either "vegan" or "mlpack".
method	Passed to stats::dist() if engine is "vegan", ignored if "mlpack".

`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. `borders()`.
`check.aes, check.param`	If `TRUE`, the default, will check that supplied parameters and aesthetics are understood by the `geom` or `stat`. Use `FALSE` to suppress the checks.
`...`	Additional arguments passed to `ggplot2::layer()`.

Details

A minimum spanning tree (MST) on the point cloud X is a minimal connected graph on X with the smallest possible sum of distances (or dissimilarities) between linked points. These layers call `stats::dist()` to calculate a distance/dissimilarity object and `vegan::spantree()` to calculate the MST. The result is formatted with position aesthetics readable by `ggplot2::geom_segment()`.

An MST calculated on `x` and `y` reflects the distances among the points in X in the reduced-dimension plane of the biplot. In contrast, one calculated on the full set of coordinates reflects distances in higher-dimensional space. Plotting this high-dimensional MST on the 2-dimensional biplot provides a visual cue as to how faithfully two dimensions can encapsulate the "true" distances between points (Jolliffe, 2002).

Value

A ggproto layer.

Biplot layers

`ggbiplot()` uses `ggplot2::fortify()` internally to produce a single data frame with a `.matrix` column distinguishing the subjects (`"rows"`) and variables (`"cols"`). The stat layers `stat_rows()` and `stat_cols()` simply filter the data frame to one of these two.

The geom layers `geom_rows_*()` and `geom_cols_*()` call the corresponding stat in order to render plot elements for the corresponding factor matrix. `geom_dims_*()` selects a default matrix based on common practice, e.g. points for rows and arrows for columns.

Ordination aesthetics

The convenience function `ord_aes()` can be used to incorporate all coordinates of the ordination model into a statistical transformation. It maps the coordinates to the custom aesthetics `..coord1`, `..coord2`, etc.

Some transformations, e.g. `stat_center()`, are commutative with projection to the 'x' and 'y' coordinates. If they detect aesthetics of the form `..coord[0-9]+`, then `..coord1` and `..coord2` are converted to `x` and `y` while any remaining are ignored.

Other transformations, e.g. `stat_spantree()`, yield different results in a planar biplot when they are computer before or after projection. If such a stat layer detects these aesthetics, then the lot of them are used in the transformation.

In either case, the stat layer returns a data frame with position aesthetics `x` and `y`.

References

Jolliffe IT (2002) *Principal Component Analysis*, Second Edition. Springer Series in Statistics, ISSN 0172-7397. doi: 10.1007/b98835 https://link.springer.com/book/10.1007/b98835

See Also

Other stat layers: `stat_center()`, `stat_chull()`, `stat_cone()`, `stat_scale()`

Examples

```
# classical multidimensional scaling of road distances between European cities
euro_mds <- ordinate(eurodist, cmdscale_ord, k = 11)

# biplot with minimal spanning tree based on full-dimensional distances
# (as implemented in {mlpack})
euro_mds %>%
  negate_ord("PCo2") %>%
  ggbiplot() +
  stat_cols_spantree(
    ord_aes(euro_mds), check.aes = FALSE, engine = "mlpack",
    alpha = .5, linetype = "dotted"
  ) +
  geom_cols_text(aes(label = name), size = 3) +
  ggtitle(
    "MDS biplot of road distances between European cities",
    "Dotted segments constitute the minimal spanning tree"
  )
```

`supplementation` *Supplement 'tbl_ord' objects with new data*

Description

These functions attach supplementary rows or columns to an ordination object.

Usage

```
recover_supp_rows(x)

## Default S3 method:
recover_supp_rows(x)

recover_supp_cols(x)

## Default S3 method:
recover_supp_cols(x)
```

Arguments

`x`	An object of class 'tbl_ord'.

Details

The `recover_supp_*()` S3 methods produce matrices of supplemental rows or columns of a tbl_ord object from the object itself. The motivating example is linear discriminant analysis, which produces a natural biplot of class discriminant centroids and variable axes but is usually supplemented with case discriminant scores. The supplementary values are augmented with an `.element` column whose value indicates their source and can be incorporated into a tidied form. If no supplementary rows of a factor are produced, the functions return `NULL`.

Value

Matrices having the same numbers of columns as returned by `recover_rows()` and `recover_cols()`, or else `NULL`.

See Also

Other generic recoverers: `augmentation`, `conference`, `recoverers`

`tbl_ord` *A unified ordination object class*

Description

These functions wrap ordination objects in the class tbl_ord, create tbl_ords directly from matrices, and test for the class and basic structure.

Usage

```
as_tbl_ord(x)

## S3 method for class 'tbl_ord'
as_tbl_ord(x)

make_tbl_ord(rows = NULL, cols = NULL, ...)

is_tbl_ord(x)

is.tbl_ord(x)

valid_tbl_ord(x)

un_tbl_ord(x)
```

Arguments

`x`	An ordination object.
`rows, cols`	Matrices to be used as factors of a tbl_ord.
`...`	Additional elements of a custom tbl_ord.

Details

The tbl_ord class wraps around a range of ordination classes, making available a suite of ordination tools that specialize to each original object class. These tools include `format()` and `fortify()` methods, which facilitate the `print()` method and the `ggbiplot()` function.

No default method is provided for `as_tbl_ord()`, despite most defined methods being equivalent (simply appending 'tbl_ord' to the vector of object classes). This prevents objects for which other methods are not defined from being re-classed as tbl_ords.

The function `make_tbl_ord()` creates a tbl_ord structured as a list of two matrices, `u` and `v`, which must have the same number of columns and the same column names.

`is_tbl_ord()` checks an object `x` for the tbl_ord class; `valid_tbl_ord()` additionally checks for consistency between `recover_coord(x)` and the columns of `recover_rows(x)` and `recover_cols(x)`, using the recoverers. `un_tbl_ord()` removes attributes associated with the tbl_ord class in order to restore an object that was originally passed to `as_tbl_ord`.

Value

A tbl_ord (`as*()`, `make*()`), an S3-class model object that can be wrapped as one (`un*()`), or a logical value (`is*()`, `value*()`).

Examples

```
# illustrative ordination: FA of Swiss social data
swiss_fa <- factanal(swiss, factors = 3L, scores = "regression")
print(swiss_fa)

# add the 'tbl_ord' wrapper
swiss_fa_ord <- as_tbl_ord(swiss_fa)
# inspect wrapped model
is_tbl_ord(swiss_fa_ord)
print(swiss_fa_ord)
valid_tbl_ord(swiss_fa_ord)
# unwrap the model
un_tbl_ord(swiss_fa_ord)

# create a 'tbl_ord' directly from row and column factors
# (missing inertia & other attributes)
swiss_fa_ord2 <- make_tbl_ord(rows = swiss_fa$scores, cols = swiss_fa$loadings)
# inspect wrapped factors
is_tbl_ord(swiss_fa_ord2)
print(swiss_fa_ord2)
valid_tbl_ord(swiss_fa_ord2)
# unwrap factors
un_tbl_ord(swiss_fa_ord2)
```

`theme_biplot` *Biplot theme*

Description

Omit coordinate visual aids from biplots.

Usage

```
theme_biplot()
```

Details

Because the artificial axes often go uninterpreted, biplots may omit the visual aids (tick marks and labels, grid lines) used to recover the artificial coordinates of the row and column markers The biplot (partial) theme removes these elements from the current theme. This can be especially helpful when plotting axes or isolines.

Value

A ggplot theme.

`tidiers` *Tidiers for 'tbl_ord' objects*

Description

These functions return tibbles that summarize an object of class 'tbl_ord'. `tidy()` output contains one row per artificial coordinate and `glance()` output contains one row for the whole ordination.

Usage

```
## S3 method for class 'tbl_ord'
tidy(x, ...)

## S3 method for class 'tbl_ord'
glance(x, ...)

## S3 method for class 'tbl_ord'
fortify(model, data, ..., .matrix = "dims", elements = "all")
```

Arguments

`x, model`	An object of class 'tbl_ord'.
`...`	Additional arguments allowed by generics; currently ignored.
`data`	Passed to generic methods; currently ignored.
`.matrix`	A character string partially matched (lowercase) to several indicators for one or both matrices in a matrix decomposition used for ordination. The standard values are `"rows"`, `"cols"`, and `"dims"` (for both).
`elements`	Character vector; which elements of each factor for which to render graphical elements. One of `"all"` (the default), `"active"`, or any supplementary element type defined by the specific class methods (e.g. `"score"` for 'factanal', 'lda_ord', and 'cancord_ord' and `"intraset"` and `"interset"` for 'cancor_ord').

Details

Three generics popularized by the **ggplot2** and **broom** packages make use of the augmentation methods:

- The `generics::tidy()` method summarizes information about model components, which here are the artificial coordinates created by ordinations. The output can be passed to `ggplot2::ggplot()` to generate scree plots. The returned columns are
  - `name`: (the name of) the coordinate
  - other columns extracted from the model, usually a single additional column of the singular or eigen values
  - `inertia`: the multidimensional variance
  - `prop_var`: the proportion of inertia
  - `quality`: the cumulative proportion of variance
- The `generics::glance()` method reports information about the entire model, here always treated as one of a broader class of ordination models. The returned columns are
  - `rank`: the rank of the ordination model, i.e. the number of ordinates
  - `n.row,n.col`: the dimensions of the decomposed matrix
  - `inertia`: the total inertia in the ordination
  - `prop.var.*`: the proportion of variance in the first 2 ordinates
  - `class`: the class of the wrapped model object
- The `ggplot2::fortify()` method augments and collapses row and/or column data, depending on `.matrix` and `.element`, into a single tibble, in preparation for `ggplot2::ggplot()`. Its output resembles that of `generics::augment()`, though rows in the output may correspond to rows, columns, or both of the original data. If `.matrix` is passed `"rows"`, `"cols"`, or `"dims"` (for both), then `fortify()` returns a tibble whose fields are obtained, in order, via `get_*()`, `recover_aug_*()`, and `annotation_*()`.

The tibble is assigned a `"coordinates"` attribute whose value is obtained via `get_coord()`. This facilitates some downstream functionality that relies on more than those coordinates used as position aesthetics in a biplot, in particular `stat_spantree()`.

Value

A tibble.

See Also

augmentation methods that must interface with tidiers.

Examples

```
# illustrative ordination: PCA of iris data
iris_pca <- ordinate(iris, ~ prcomp(., center = TRUE, scale. = TRUE), seq(4L))

# use `tidy()` to summarize distribution of inertia
tidy(iris_pca)
# this facilitates scree plots
tidy(iris_pca) %>%
  ggplot(aes(x = name, y = prop_var)) +
  geom_col() +
  scale_y_continuous(labels = scales::percent) +
  labs(x = NULL, y = "Proportion of variance")

# use `fortify()` to prepare either matrix factor for `ggplot()`
fortify(iris_pca, .matrix = "V") %>%
  ggplot(aes(x = name, y = PC1)) +
  geom_col() +
  coord_flip() +
  labs(x = "Measurement")
iris_pca %>%
  fortify(.matrix = "U") %>%
  ggplot(aes(x = PC1, fill = Species)) +
  geom_histogram() +
  labs(y = NULL)
# ... or to prepare both for `ggbiplot()`
fortify(iris_pca)

# use `glance()` to summarize the model as an ordination
glance(iris_pca)
# this enables comparisons to other models
rbind(
  glance(ordinate(subset(iris, Species == "setosa"), prcomp, seq(4L))),
  glance(ordinate(subset(iris, Species == "versicolor"), prcomp, seq(4L))),
  glance(ordinate(subset(iris, Species == "virginica"), prcomp, seq(4L)))
)
```

`wrap-ord` *Wrappers for lossy ordination methods*

Description

These *_ord functions wrap core R functions with modifications for use with 'tbl_ord' methods. Some parameters are hidden from the user and set to settings required for these methods, some matrix outputs are given row or column names to be used by them, and new '*_ord' S3 class attributes are added to enable them.

Usage

```
eigen_ord(x, symmetric = isSymmetric.matrix(x))

svd_ord(x, nu = min(dim(x)), nv = min(dim(x)))

cmdscale_ord(d, k = 2, add = FALSE)

cancor_ord(x, y, xcenter = TRUE, ycenter = TRUE, scores = FALSE)
```

Arguments

`x`	a numeric or complex matrix whose spectral decomposition is to be computed. Logical matrices are coerced to numeric.
`symmetric`	if `TRUE`, the matrix is assumed to be symmetric (or Hermitian if complex) and only its lower triangle (diagonal included) is used. If `symmetric` is not specified, `isSymmetric(x)` is used.
`nu`	the number of left singular vectors to be computed. This must between `0` and `n` = `nrow(x)`.
`nv`	the number of right singular vectors to be computed. This must be between `0` and `p` = `ncol(x)`.
`d`	a distance structure such as that returned by `dist` or a full symmetric matrix containing the dissimilarities.
`k`	the maximum dimension of the space which the data are to be represented in; must be in $\{1, 2, \ldots, n-1\}$.
`add`	logical indicating if an additive constant $c*$ should be computed, and added to the non-diagonal dissimilarities such that the modified dissimilarities are Euclidean.
`y`	numeric matrix ($n \times p_2$), containing the y coordinates.
`xcenter`	logical or numeric vector of length p_1, describing any centering to be done on the x values before the analysis. If `TRUE` (default), subtract the column means. If `FALSE`, do not adjust the columns. Otherwise, a vector of values to be subtracted from the columns.
`ycenter`	analogous to `xcenter`, but for the y values.
`scores`	Logical; whether to return canonical scores and structure correlations.

Details

The following table summarizes the wrapped functions:

Original function	Hide params	New params	Add names	New class
`base::eigen()`	Yes	No	Yes	Yes
`base::svd()`	Yes	No	Yes	Yes
`stats::cmdscale()`	Yes	No	No	Yes
`stats::cancor()`	No	Yes	No	Yes

By default, `cancor_ord()` returns the same data as `stats::cancor()`: the canonical correlations (`cor`), the canonical coefficients (`$xcoef` and `$ycoef`), and the variable means (`$xcenter`, `$ycenter`). If `scores = TRUE`, then `cancor_ord()` also returns the scores `$xscores` and `$yscores` calculated from the (appropriately centered) data and the coefficients and the intraset structure correlations `$xstructure` and `$ystructure` between these and the data. These modifications are inspired by the `cancor()` function in **candisc**, though two caveats should be noted: First, the canonical coefficients (hence the canonical scores) are scaled by $n - 1$ compared to these, though the intraset structure correlations are the same. Second, the *interset* structure correlations are not returned, as these may be obtained by conferring inertia unto the intraset ones.

Value

Objects slightly modified from the outputs of the original functions, with new '*-ord' classes.

Examples

```
# glass composition data from one furnace
glass_banias <- subset(
  glass,
  Context == "L.15;B.166",
  select = c("SiO2", "Na2O", "CaO", "Al2O3", "MgO", "K2O")
)
# eigendecomposition of a covariance matrix
(glass_cov <- cov(glass_banias))
eigen_ord(glass_cov)
# singular value decomposition of a data matrix
svd_ord(glass_banias)
# classical multidimensional scaling of a distance matrix
cmdscale_ord(dist(glass_banias))

# canonical correlation analysis with trace components
glass_banias_minor <- subset(
  glass,
  Context == "L.15;B.166",
  select = c("TiO2", "FeO", "MnO", "P2O5", "Cl", "SO3")
)
# impute half of detection threshold
glass_banias_minor$TiO2[[1L]] <- 0.5
cancor_ord(glass_banias, glass_banias_minor)

# calculate canonical scores and structure correlations
glass_cca <-
  cancor_ord(glass_banias[, 1:3], glass_banias_minor[, 1:3], scores = TRUE)
# scores
```

```
glass_cca$xscores
# intraset correlations
glass_cca$xstructure
# interset correlations
glass_cca$xstructure %*% diag(glass_cca$cor)
```

Index