Package: prefviz (via r-universe)

May 31, 2026

Title Visualizes Preferential Data in One and More Contests

Version 0.1.2

Description A visualization toolkit for preferential data, such as ranked-choice election results, tournament outcomes, and survey responses. The package provides methods to visualise the preference distribution of one contest with bar charts and pairwise comparisons of two contestants, as well as methods to visualise multiple contests through 2D and high-dimensional simplex plots both statically and interactively. HD simplex displays are implemented via projection methods using the 'tourr' and 'detourr' packages, enabling dynamic exploration of high-dimensional preference structure. For more details on HD simplex projection, see Wickham et al. (2011) <doi:10.21105/joss.03419>.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports dplyr, lifecycle, prefio, tibble, tidyr, rlang, ggplot2, scales, tidyselect, geozoo

Suggests knitr, rmarkdown, tourr, kableExtra, ggthemes, testthat (>= 3.0.0)

Config/testthat/edition 3

Depends R (>= 4.1.0)

LazyData true

VignetteBuilder knitr

URL https://numbats.github.io/prefviz/

BugReports https://github.com/numbats/prefviz/issues

Config/pak/sysreqs libicu-dev

Repository https://numbats.r-universe.dev

Date/Publication 2026-05-01 00:12:41 UTC

RemoteUrl https://github.com/numbats/prefviz

RemoteRef HEAD

RemoteSha 437b264a890772f74aaf45164a2e0720e57cdac5

Contents

`add_ternary_base` *Draw the 2D ternary simplex*

Description

Draws the boundary of a 2D ternary simplex as an equilateral triangle

Usage

```
add_ternary_base(...)
```

Arguments

`...`	Arguments passed to `ggplot2::geom_polygon()`, such as `colour`, `fill`, `linewidth`, etc.

Value

A ggplot object

Examples

```
library(ggplot2)

# Basic simplex
ggplot() + add_ternary_base()

# Customize appearance
ggplot() + add_ternary_base(colour = "blue", linewidth = 1.5)
```

`add_vertex_labels` *Add vertex labels to ternary plot*

Description

Adds text labels at the vertices of a ternary simplex with automatic positioning adjustments.

Usage

```
add_vertex_labels(
  vertex_labels_df,
  nudge_x = c(-0.02, 0.02, 0),
  nudge_y = c(-0.05, -0.05, 0.05),
  ...
)
```

Arguments

`vertex_labels_df`	A data frame containing vertex coordinates and labels. Should have columns `x1`, `x2`, and `labels`. Can be specified manually or obtained from a ternable object: `ternable_object$simplex_vertices`.
`nudge_x`	Numeric vector of length 3 specifying horizontal nudges for each vertex label.
`nudge_y`	Numeric vector of length 3 specifying vertical nudges for each vertex label.
`...`	Arguments passed to `ggplot2::geom_text()`, such as `size`, `colour`, `fontface`, etc.

Value

A ggplot object

Examples

```
library(ggplot2)

# Create a ternable object
tern <- as_ternable(prefviz::aecdop22_transformed, ALP:Other)

ggplot() +
  add_ternary_base() +
  add_vertex_labels(tern$simplex_vertices, size = 5, fontface = "bold")
```

---

aecdop *Distribution of preferences by candidate by division in the Australian Federal Election (2022 and 2025)*

Description

Provides details on how votes are distributed and transferred among candidates in all count stages of the preferential voting system. All electoral divisions in the Australian Federal Election are included.

Usage

```
aecdop_2022

aecdop_2025
```

Format

A tibble of 14 columns:

StateAb State or territory abbreviation (e.g., "ACT", "NSW", "VIC")

DivisionID Numeric identifier for the electoral division

DivisionNm Name of the electoral division (e.g., "Bean", "Canberra")

CountNumber Round in the counting procedure, starting from 0 (first preference)

BallotPosition Position of the candidate on the ballot paper

CandidateID Unique numeric identifier for the candidate

Surname Candidate's surname

GivenNm Candidate's given name(s)

PartyAb Party abbreviation (e.g., "UAPP", "ALP", "LP")

PartyNm Full party name (e.g., "United Australia Party", "Australian Labor Party")

Elected Whether the candidate was elected: "Y" (yes) or "N" (no)

HistoricElected Whether the candidate was elected in the previous election: "Y" (yes) or "N" (no)

CalculationType Type of calculation:

Preference Count Number of votes received

Preference Percent Percentage of votes received

Transfer Count Number of votes transferred from other candidates

Transfer Percent Percentage of votes transferred from other candidates

CalculationValue Numeric value for the calculation type (votes or percentage)

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 35096 rows and 14 columns.

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 30888 rows and 14 columns.

Details

Two datasets are provided:

- `aecdop_2022`: 2022 Federal Election (35,096 rows)
- `aecdop_2025`: 2025 Federal Election (30,888 rows)

Source

Australian Electoral Commission (AEC) Distribution of Preferences 2022 Distribution of Preferences 2025

Examples

```
# Load the datasets
data(aecdop_2022)
data(aecdop_2025)

# First preferences for Bean division in 2022
aecdop_2022 |>
  dplyr::filter(DivisionNm == "Bean",
         CountNumber == 0,
         CalculationType == "Preference Count")
```

`aecdop_transformed` *Distribution of preferences in wide form for selected parties (2022 and 2025)*

Description

Wide-form versions of the Australian Federal Election distribution-of- preferences data, aggregated to selected parties within each electoral division and couting round. Each row gives the vote share for a set of parties and an "Other" category at a given count stage in a given division. These datasets are derived from `aecdop_2022` and `aecdop_2025` for ease of analysis and visualisation.

Usage

```
aecdop22_transformed

aecdop25_transformed
```

Format

For `aecdop22_transformed`, a tibble with 1,052 rows and 6 columns:

DivisionNm Name of the electoral division (e.g., "Adelaide").

CountNumber Round in the counting procedure, starting from 0 (first preference).

ElectedParty Party abbreviation of the candidate ultimately elected in the division (e.g., "ALP", "LNP").

ALP Proportion of votes for the Australian Labor Party at this count, between 0 and 1.

LNP Proportion of votes for the Coalition grouping at this count, between 0 and 1.

Other Proportion of votes for all other parties and candidates combined at this count, between 0 and 1.

For `aecdop25_transformed`, a tibble with 976 rows and 8 columns:

DivisionNm Name of the electoral division (e.g., "Adelaide").

CountNumber Round in the counting procedure, starting from 0 (first preference).

ElectedParty Party abbreviation of the candidate ultimately elected in the division (e.g., "ALP", "GRN", "LNP", "IND").

ALP Proportion of votes for the Australian Labor Party at this count, between 0 and 1.

GRN Proportion of votes for the Australian Greens at this count, between 0 and 1.

LNP Proportion of votes for the Coalition grouping at this count, between 0 and 1.

Other Proportion of votes for all other parties and candidates combined at this count, between 0 and 1.

IND Proportion of votes for independent candidates at this count, between 0 and 1.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1052 rows and 6 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 976 rows and 8 columns.

Details

Two datasets are provided:

- `aecdop22_transformed`: 2022 Federal Election (1,052 rows), aggregated to Labor (ALP), Coalition (LNP), and Other (Other)
- `aecdop25_transformed`: 2025 Federal Election (976 rows), aggregated to Labor (ALP), Coalition (LNP), Greens (GRN), Independent (IND) and Other (Other)

Within each row, the party columns represent proportions that sum to 1 (up to rounding), giving a compositional view of the distribution of preferences at each count stage. These structures are designed for use in simplex-based visualisations and related methods.

See Also

aecdop_2022, aecdop_2025

Examples

```
data(aecdop22_transformed)
data(aecdop25_transformed)

# Proportions for Adelaide over the count in 2022
aecdop22_transformed |>
  dplyr::filter(DivisionNm == "Adelaide")
```

---

as_ternable *Create a ternable object*

Description

Creates a ternable object, which contains observation coordinates, simplex vertices, and edges necessary for building a ternary plot in both two and higher dimensions.

Usage

```
as_ternable(
  data,
  items = dplyr::everything(),
  group = NULL,
  order_by = NULL,
  decreasing = FALSE,
  na_method = c("drop_na", "drop_group"),
  ...
)
```

Arguments

data	A data frame containing the item (alternative) columns used to construct the ternary plot.
items	<tidy-select> Columns representing the items to be plotted as vertices of the simplex. Default is tidyselect::everything(), which selects all columns. Must select at least 3 columns. All columns must be non-negative and sum to 1.
group	Optional column name indicating the grouping variable. If specified, the data will be grouped by this variable. This is useful for creating paths between observations within each group.
order_by	Optional column name indicating the order variable. If specified, the data will be ordered by this variable. This is useful for creating paths between observations within each group.

decreasing	Logical. If TRUE, paths are ordered in decreasing order of order_by. If FALSE (default), ordering is increasing.
na_method	Character string specifying how to handle missing values in order_by. One of: • "drop_na" (default): drop only rows where order_by is NA; • "drop_group": drop entire groups that contain any NA in order_by.
...	Additional arguments (currently unused, reserved for future extensions).

Value

A ternable object (S3 class) containing:

data	: The validated and normalized data frame
data_coord	: Transformed coordinates for all observations
data_edges	: Edge connections for drawing paths between observations
simplex_vertices	: Vertex coordinates and labels for the simplex
simplex_edges	: Edge connections for drawing the simplex boundary
vertex_labels	: Labels of the vertices, same as names of the selected item columns

Examples

```
# Load and transform the dataset
prefviz::aecdop25_transformed

# Create the ternable object
tern <- as_ternable(prefviz::aecdop25_transformed, items = ALP:IND)
tern
```

---

dop_bar *Bar chart of preference distribution for one contest*

Description

Draws a bar chart showing how votes or preferences are distributed across items (candidates, parties, options) in a single contest or round. Bars are ordered from highest to lowest value.

Usage

```
dop_bar(data, items, value_col = NULL, round_col = "round", at_round = 1)
```

Arguments

data	A data frame in wide or long format. See Details.
items	<tidy-select> Columns to plot. Interpretation depends on format: • **Wide**: A tidy-select expression identifying the item columns to pivot, e.g. ALP:Other or -c(round, winner). • **Long**: A single column containing item names, e.g. PartyAb.
value_col	<tidy-select> **Long format only.** Column containing the numeric values to plot (vote shares or counts). When NULL (default), wide format is assumed and this argument is ignored.
round_col	Character. Name of the column used to identify rounds. Default "round", matching dop_irv() output. Override when your data uses a different column name (e.g., "CountNumber").
at_round	Integer. The round number to display. Default is 1.

Details

dop_bar() accepts data in two formats, detected automatically via value_col:

Wide format (value_col = NULL, the default)

One row per round, one column per item. This is the direct output of dop_irv():

```
round | ALP  | LNP  | Other | winner
    1 | 0.40 | 0.35 | 0.25  | ALP
    2 | 0.52 | 0.48 | 0.00  | ALP
```

Supply the item columns via items (e.g., ALP:Other or -c(round, winner)) and select the round to display with at_round. Use round_col if your round column is named something other than "round" (e.g., round_col = "CountNumber").

Long format (value_col provided)

One row per item, with the item name and its value in separate columns. This is the format of aecdop_2022 and similar raw electoral datasets:

```
DivisionNm | CountNumber | PartyAb | CalculationValue
Adelaide   |           0 | ALP     |            0.40
Adelaide   |           0 | LNP     |            0.35
Adelaide   |           0 | Other   |            0.25
```

Supply the item name column via items, the value column via value_col, and the round to display via at_round. Use round_col if your round column is named something other than "round" (e.g., round_col = "CountNumber").

Value

A ggplot object.

See Also

dop_irv() to generate wide-format input from raw ballot data.

Examples

```
library(ggplot2)

# Wide format: output of dop_irv()
votes <- prefio::preferences(c("A > B > C", "B > A > C", "C > B > A",
                               "A > B > C", "A > C > B"))
irv_result <- dop_irv(votes)
dop_bar(irv_result, items = -c(round, winner), at_round = 1)

# Long format: pre-filter to desired contest, then plot
long_df <- aecdop_2022 |>
  dplyr::filter(
    CalculationType == "Preference Percent",
    CountNumber == 0,
    DivisionNm == "Adelaide"
  )
dop_bar(long_df, items = PartyAb, value_col = CalculationValue,
       round_col = "CountNumber", at_round = 0)
```

---

dop_irv *Get full distribution of preferences in each instant runoff voting round as percentage*

Description

Compute the preference in each round of instant runoff voting from input data, transforming the results into a tidy format for visualization. Each row represents one round, with columns for each candidate's preference percentage and the election winner.

Usage

```
dop_irv(x, value_type = c("percentage", "count"), ...)
```

Arguments

x	Input data. Accepts the same formats as prefio::pref_irv(): • A preference vector where each element represents one ballot • A data frame with a column for preference
value_type	Character string specifying the output format. Either: • "percentage" (default): Returns vote shares as proportions (0-1) • "count": Returns raw vote counts
...	Additional arguments passed to prefio::pref_irv(), including: • preferences_col: Column name containing preference orderings • frequency_col: Column name containing vote frequencies

Value

A tibble with the following structure:

- round: Integer, the round number (1 to n)
- One column per candidate: Numeric, the percentage of votes (0-1) that candidate received in that round. NA values are replaced with 0 for eliminated candidates.
- winner: Character, the name of the eventual IRV winner (same for all rows)

Examples

```
# Example 1: From preference vector
votes <- prefio::preferences(c("A > B > C", "B > A > C", "C > B > A", "A > B > C"))
dop_irv(votes, value_type = "count")

# Example 2: From data frame with custom column names
vote_data <- tibble::tibble(
  prefs = prefio::preferences(c("A > B > C", "B > C > A", "C > A > B")),
  counts = c(100, 75, 25)
)
dop_irv(vote_data, value_type = "percentage",
        preferences_col = prefs,
        frequency_col = counts)
```

---

dop_transform *Transform AEC distribution of preferences from long to wide format*

Description

Transform AEC distribution of preferences from long to wide format, with optional scaling and normalization. This function is useful for converting all distribution of preference data with similar format into format ready for ternary plots.

Usage

```
dop_transform(
  data,
  key_cols,
  value_col,
  item_col,
  normalize = TRUE,
  scale = 1,
  fill_value = 0,
  winner_col = NULL,
  winner_identifier = "Y"
)
```

Arguments

`data`	A data frame containing preference or vote distribution data, with format similar to AEC Distribution of Preferences 2022
`key_cols`	Columns that identify unique observations, e.g., DivisionNm, CountNumber
`value_col`	Numeric and non-negative. Column containing the numeric values to aggregate, e.g., CalculationValue, Votes.
`item_col`	Column name containing the items (candidates/parties) of the election, e.g., Party, Candidate. This column will become column names in the output wide format.
`normalize`	Logical. If `TRUE` (default), normalizes values within each group to sum to 1. If `FALSE`, returns raw aggregated values.
`scale`	Numeric. If `normalize = FALSE`, divides all values by this scale factor. Default is 1 (no scaling).
`fill_value`	Numeric. Value to use for missing combinations after pivoting. Default is 0.
`winner_col`	Optional character string specifying a column that indicates the winner/elected party. If provided, this column will be joined back to the output based on key columns. Useful for preserving election outcome information. Default is `NULL`.
`winner_identifier`	Optional character string specifying the value in `winner_col` that identifies winning candidates (e.g., "Y", "Elected"). Only used if `winner_col` is specified. Default is "Y".

Value

A data frame in wide format with:

- Key columns identifying each observation
- Columns for each item (candidate/party) containing aggregated/normalized values
- Winner column (if `winner_col` was specified)

Examples

```
library(dplyr)
# Convert AEC 2025 Distribution of Preference data to wide format
data(aecdop_2025)

# We are interested in the preferences of Labor, Coalition, Greens and Independent.
# The rest of the parties are aggregated as Other.
aecdop_2025 <- aecdop_2025 |>
 filter(CalculationType == "Preference Percent") |>
 mutate(Party = case_when(
    !(PartyAb %in% c("LP", "ALP", "NP", "LNP", "LNQ")) ~ "Other",
    PartyAb %in% c("LP", "NP", "LNP", "LNQ") ~ "LNP",
   TRUE ~ PartyAb))

dop_transform(
  data = aecdop_2025,
```

```
  key_cols = c(DivisionNm, CountNumber),
  value_col = CalculationValue,
  item_col = Party,
  winner_col = Elected
)
```

`elb_centroid` *Centroids of electoral divisions in the 2025 Australian Federal Election*

Description

Provides the centroids of all electorates in the 2025 Australian Federal Election. The dataset is computed from 2025 Electoral Boundaries data.

Usage

```
elb_centroid
```

Format

A tibble of 5 columns:

id Unique identifier for electorate

elect_div Electoral division name

area_sqkm Area of the electorate in square kilometres

long Longitude of the electoratecentroid

lat Latitude of the electorate centroid

Source

Australian Electoral Commission (AEC) https://www.aec.gov.au/electorates/maps.htm

Examples

```
library(ggplot2)
library(ggthemes)

# Load the dataset
data(elb_centroid)

# Plot the centroids on top of the electoral boundaries
ggplot(elb_map) +
  geom_polygon(
    aes(x = long, y = lat, group = group),
    fill = "grey90", color = "white") +
  geom_point(
    data = elb_centroid,
```

```
  aes(x = long, y = lat),
  size = 1, alpha = 0.8
) +
theme_map()
```

---

elb_map *Electoral boundaries map for the 2025 Australian Federal Election*

Description

Provides the points that make up the boundaries of each electoral division in the 2025 Australian Federal Election.

Usage

```
elb_map
```

Format

A tibble of 8 columns:

long Longitude of point in polygon

lat Latitude of point in polygon

hole Whether the polygon has a hole

piece Polygon piece number

group Polygon group number

order Order of polygon within group

id Unique identifier for polygon

elect_div Electoral division name

Source

Australian Electoral Commission (AEC) https://www.aec.gov.au/electorates/maps.htm

Examples

```
library(ggplot2)
library(ggthemes)

# Load the dataset
data(elb_map)

# Plot the map
ggplot(elb_map) +
  geom_polygon(
    aes(x = long, y = lat, group = group),
    fill = "grey90", color = "white") +
  theme_map()
```

geom_ternary_region *Create polygonal regions in a ternary plot based on a reference point*

Description

geom_ternary_region() and stat_ternary_region() divide the ternary triangle into three polygonal regions centered around a specific reference point.

Geometrically, lines are drawn from the reference point perpendicular to the three edges of the triangle. These lines partition the simplex into three zones, where each zone is associated with the closest vertex (item). This is often used to visualize "winning regions" or catchment areas for each item.

Usage

```
geom_ternary_region(
  mapping = NULL,
  position = "identity",
  show.legend = NA,
  inherit.aes = FALSE,
  x1 = 1/3,
  x2 = 1/3,
  x3 = 1/3,
  vertex_labels = NULL,
  ...
)

stat_ternary_region(
  mapping = NULL,
  data = NULL,
  geom = "polygon",
  position = "identity",
  show.legend = NA,
  inherit.aes = FALSE,
  x1 = 1/3,
  x2 = 1/3,
  x3 = 1/3,
  vertex_labels = NULL,
  ...
)

StatTernaryRegion
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). To map aesthetics to the computed region labels, use ggplot2::after_stat(), e.g., aes(fill = after_stat(vertex_labels)).

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`show.legend` Logical. Should this layer be included in the legends? `NA` (default) includes it if aesthetics are mapped. `FALSE` never includes it; `TRUE` always includes it.

`inherit.aes` If `FALSE`, overrides the default aesthetics rather than combining with them.

`x1, x2, x3` Numeric values defining the reference point in ternary coordinates (proportions). Must sum to 1 (or will be normalized). Default is `c(1/3, 1/3, 1/3)` (the centroid), which divides the space into three equal regions.

`vertex_labels` Character vector of length 3 providing names for the regions. The order must correspond to the three vertices of the ternary plot. If `NULL`, regions are labeled "Region 1", "Region 2", and "Region 3", starting from the rightmost vertex and moving clockwise.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`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 to display the data. Default is `"polygon"`.

Format

An object of class `StatTernaryRegion` (inherits from `Stat`, `ggproto`, `gg`) of length 3.

Value

A ggplot object

Computed variables

`stat_ternary_region()` calculates the following variables which can be accessed with `after_stat()`:

`x`, `y` Cartesian coordinates defining the polygon shapes.

`id` Numeric identifier for the specific geometric points used to build the polygons:

- 1-3: The main vertices of the ternary triangle.
- 4: The reference point (center).
- 5-7: The projection points on the edges.

`group` Integer (1, 2, or 3) identifying which region the polygon belongs to.

`vertex_labels` The label assigned to the region (derived from the `vertex_labels` parameter).

Examples

```
library(ggplot2)
# Get ternable
tern22 <- as_ternable(prefviz::aecdop22_transformed, ALP:Other)

# Draw the ternary plot
ggplot(get_tern_data2d(tern22), aes(x = x1, y = x2)) +
  add_ternary_base() +
  geom_ternary_region(
    vertex_labels = tern22$vertex_labels,
    aes(fill = after_stat(vertex_labels)),
    alpha = 0.3, color = "grey50",
    show.legend = FALSE
  )
```

helmert_transform *Transform compositional data using Helmert matrix*

Description

Transform n-dimension compositional data (all values sum to 1) into an (n-1)-dimensional Euclidean space using the Helmert matrix. This dimension reduction is the geometric basis for plotting points within the simplex.

Usage

```
helmert_transform(data, items = dplyr::everything(), append = FALSE)
```

Arguments

data	A data frame or matrix containing the compositional data.
items	<tidy-select> Columns representing the components of the composition. Default is tidyselect::everything(), which selects all columns. Must select at least 3 columns.
append	(Optional) A logical value indicating whether the transformed data should be appended to the original data frame. Default is FALSE.

Value

A data frame containing the Helmert-transformed coordinates, named x1, x2, ..., x(n-1), where n is the number of items. If append = TRUE, these columns are added to the input data.

Examples

```
# Example 1: Transform a matrix (all columns)
comp_mat <- matrix(c(0.5, 0.3, 0.2,
                     0.4, 0.4, 0.2,
                     0.6, 0.2, 0.2),
                   ncol = 3, byrow = TRUE)
helmert_transform(comp_mat)

# Example 2: Transform specific columns in a data frame
df <- data.frame(
  electorate = c("A", "B", "C"),
  ALP = c(0.5, 0.4, 0.6),
  LNP = c(0.3, 0.4, 0.2),
  Other = c(0.2, 0.2, 0.2)
)
helmert_transform(df, items = c(ALP, LNP, Other))
```

`pairwise_calculator` *Compute pairwise results from ranked preference data*

Description

Computes all pairwise comparisons from a set of ranked preferences using `prefio::adjacency()`. For each pair of items, reports the number of voters who preferred each item and the Two-Candidate Preferred (TCP) ratio. Also identifies the Condorcet winner and loser where they exist.

Usage

```
pairwise_calculator(x, preferences_col = NULL, frequency_col = NULL)
```

Arguments

`x`	A `preferences` object, or a data frame / tibble containing a `preferences`-typed column.
`preferences_col`	<tidy-select> When `x` is a data frame, the column containing the preferences. Passed directly to `prefio::adjacency()`.
`frequency_col`	<tidy-select> Optional column containing ballot frequencies. Passed directly to `prefio::adjacency()`.

Details

TCP ratio (`tcp_a`, `tcp_b`) is computed as `wins / (wins_a + wins_b)`. The denominator is the number of voters who expressed a preference between the specific pair, not the total number of voters. This correctly handles partial rankings where some voters did not rank all items.

Condorcet winner: the item that beats every other item head-to-head (tcp > 0.5 in all pairs it appears in). At most one can exist.

Condorcet loser: the item that loses to every other item head-to-head (tcp < 0.5 in all pairs it appears in). At most one can exist. Neither winner nor loser may exist when preference cycles are present.

Value

An S3 object of class `"pairwise"` with four components:

`pairwise_matrix` N×N integer matrix from `prefio::adjacency()`.

`two_candidate_preferred` Tibble with one row per pair, columns: `item_a`, `item_b`, `wins_a`, `wins_b`, `total`, `tcp_a`, `tcp_b`, `h2h_winner`.

`condorcet_winner` Name of the Condorcet winner, or `NA` if none.

`condorcet_loser` Name of the Condorcet loser, or `NA` if none.

See Also

`pairwise_heatmap()` to visualise the results.

Examples

```
library(prefio)

prefs <- data.frame(
  A = c(1, 1, 1, 2, 2),
  B = c(2, 2, 3, 1, 3),
  C = c(3, 3, 2, 3, 1)
) |>
  wide_preferences(col = vote, ranking_cols = A:C)

result <- pairwise_calculator(prefs, preferences_col = vote)
print(result)
```

---

pairwise_heatmap *Heatmap of pairwise results*

Description

Plots a full N×N heatmap of pairwise results from a `pairwise_calculator()` object. Each cell shows how the row item performed against the column item. Color always encodes the TCP ratio (green = win, red = lose, white = 50/50).

Usage

```
pairwise_heatmap(x, value = c("tcp", "count"))
```

Arguments

`x`	A `pairwise` object returned by `pairwise_calculator()`.
`value`	`"tcp"` (default) or `"count"`. Controls the tile annotation: • `"tcp"`: shows the TCP percentage, e.g. `"56.8%"`. • `"count"`: shows the raw vote count, e.g. `"2841"`.

Value

A ggplot object.

See Also

`pairwise_calculator()` to compute the input object.

Examples

```
library(prefio)

prefs <- data.frame(
  A = c(1, 1, 1, 2, 2),
  B = c(2, 2, 3, 1, 3),
  C = c(3, 3, 2, 3, 1)
) |>
  wide_preferences(col = vote, ranking_cols = A:C)

result <- pairwise_calculator(prefs, preferences_col = vote)
pairwise_heatmap(result, value = "tcp")
pairwise_heatmap(result, value = "count")
```

---

stat_ordered_path *Reorder observations for a path geom*

Description

`stat_ordered_path()` reorders observations along each path using a user-supplied ordering aesthetic (`order_by`) before drawing the path. The statistic can be used to ensure that paths are drawn in a consistent order even when the input data are not pre-sorted. This is equivalent to reordering the data before passing it to `geom_path()`.

Usage

```
stat_ordered_path(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  decreasing = TRUE,
  na_method = c("drop_na", "drop_group"),
  ...
)

StatOrderedPath
```

Arguments

`mapping`	Set of aesthetic mappings created by `ggplot2::aes()`. Must at least supply `x`, `y`, and `order_by`. The `group` aesthetic can be used to define separate paths.
`data`	Data frame to be used for this layer. If `NULL`, the default, the data is inherited from the plot.

geom	The geometric object to use to draw the paths. Defaults to "path".
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
show.legend	Logical or NA. Should this layer be included in the legends? NA, the default, includes the layer if any aesthetics are mapped.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them.
decreasing	Logical. If TRUE, paths are ordered in decreasing order of order_by. If FALSE (default), ordering is increasing.
na_method	Character string specifying how to handle missing values in order_by. One of: • "drop_na" (default): drop only rows where order_by is NA; • "drop_group": drop entire groups that contain any NA in order_by.
...	Additional parameters passed on to the underlying geom.

Format

An object of class StatOrdered (inherits from Stat, ggproto, gg) of length 4.

Details

The statistic expects an order_by aesthetic that supplies the variable used to order observations within each group.

Missing values order_by are handled according to na_method. If na_method is "drop_na", missing values are dropped from the data, and the path is still drawn, but might skipping steps due to missing values. If na_method is "drop_group", the entire group whose missing values belong is dropped from the data, and the path is not drawn.

Ties in order_by are allowed but trigger a warning and preserve the original row order for tied values.

Duplicates in order_by are dropped and a warning is issued.

Grouping is controlled via the usual group aesthetic. If a group column is present in the data, reordering is performed independently within each group; otherwise, the entire data is treated as a single path.

Value

A ggplot object

A ggplot2 layer that can be added to a plot object.

Aesthetics

stat_ordered_path() understands the following aesthetics (required are in bold).

- **x**
- **y**
- **order_by**
- group
- alpha, colour, linewidth, linetype, etc. (inherited from ggplot2::geom_path())

Examples

```
library(ggplot2)
library(dplyr)
# Data prep
input_df <- prefviz::aecdop22_transformed |>
   filter(DivisionNm %in% c("Higgins", "Monash"))
tern22 <- as_ternable(input_df, ALP:Other)

# Base plot
p <- get_tern_data2d(tern22) |>
  ggplot(aes(x = x1, y = x2)) +
  add_ternary_base() +
  geom_ternary_region(
    aes(fill = after_stat(vertex_labels)),
    vertex_labels = tern22$vertex_labels,
    alpha = 0.3, color = "grey50",
    show.legend = FALSE
  ) +
  geom_point(aes(color = ElectedParty)) +
  add_vertex_labels(tern22$simplex_vertices) +
  scale_color_manual(
    values = c("ALP" = "red", "LNP" = "blue", "Other" = "grey70"),
    aesthetics = c("fill", "colour")
  )

# Add ordered paths
p +
  stat_ordered_path(
    aes(group = DivisionNm, order_by = CountNumber, color = ElectedParty))
```

---

ternary_getters *Getter functions to extract components from ternable object for ternary plots*

Description

Performs additional transformations on ternable object components, making it ready for both 2D ternary plot with ggplot2 and high-dimensional ternary plots with tourr.

Usage

```
get_tern_data2d(ternable)

get_tern_datahd(ternable)

get_tern_edges(ternable, include_data = FALSE)

get_tern_labels(ternable)
```

Arguments

ternable	A ternable object created by as_ternable().
include_data	Logical. Only in get_tern_edges(). If TRUE, return data edges, along with simplex edges. If FALSE, only return simplex edges.

Details

These functions are designed to work together for creating animated tours of high-dimensional ternary data:

- get_tern_datahd() provides both the point coordinates and observation labels
- get_tern_edges() provides the simplex structure

Value

- get_tern_data2d(): A data frame augmenting the original data with its ternary coordinates (x1, x2). Used as input data for 2D ternary plot with ggplot2.
- get_tern_datahd(): A data frame combining the simplex vertices with the original data and ternary coordinates. The labels column contains labels for the vertexes and "" for data rows. Pass the coordinate columns (dplyr::select(starts_with("x"))) to tourr and use the labels column directly for obs_labels.
- get_tern_edges(): A matrix of simplex edge connections for drawing the simplex boundary.
  - If include_data = FALSE, the matrix contains only the simplex edges. Equivalent to ternable$simplex_edges.
  - If include_data = TRUE, the matrix combines the simplex edges with the data edges. Used when you want to draw lines between the data points.

Deprecated

get_tern_labels() is deprecated as of version 0.1.2. Use get_tern_datahd(ternable)[["labels"]] instead.

See Also

as_ternable() for creating ternable objects

Examples

```
library(ggplot2)
# Create a ternable object
tern <- as_ternable(aecdop22_transformed, ALP:Other)

# Use with tourr (example)
## Not run:
tourr_data <- get_tern_datahd(tern)
tourr::animate_xy(
  dplyr::select(tourr_data, starts_with("x")),
  edges = get_tern_edges(tern),
  obs_labels = tourr_data[["labels"]],
  axes = "bottomleft")

## End(Not run)

# Use with ggplot2 (example)
ggplot(get_tern_data2d(tern), aes(x = x1, y = x2)) +
  add_ternary_base() +
  geom_point(aes(color = ElectedParty))
```

Index