

Package ‘tidymatrix’

October 8, 2026

Title Tidyverse-Style Operations on Matrices with Row and Column Metadata

Version 0.1.0

Description Provides a unified data structure for matrices with associated row and column metadata, enabling 'tidyverse'-style data manipulation. Following the approach of 'tidygraph', users can activate rows, columns, or the matrix itself and operate on it with familiar 'dplyr' verbs, while the matrix and both metadata tables are kept consistent.

License MIT + file LICENSE

URL <https://raivokolde.github.io/tidymatrix/>,
<https://github.com/raivokolde/tidymatrix>

BugReports <https://github.com/raivokolde/tidymatrix/issues>

Depends R (>= 4.1.0)

Encoding UTF-8

LazyData true

Imports dplyr, rlang, stats, tibble, utils

Suggests testthat (>= 3.0.0), ggplot2, knitr, pheatmap, rmarkdown,
Rtsne, umap

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Raivo Kolde [aut, cre]

Maintainer Raivo Kolde <rkolde@gmail.com>

Repository CRAN

Date/Publication 2026-10-08 11:00:02 UTC

Contents

activate	3
active	4
add_stats	4
analysis-management	5
arrange.tidymatrix	6
as.data.frame.tidymatrix	7
as_tibble.tidymatrix	7
big5	8
big5_countries	10
center	10
center.tidymatrix	11
check_analyses	12
clip_values	13
compute_across	13
compute_anova	15
compute_correlation	17
compute_hclust	18
compute_kmeans	19
compute_kruskal	20
compute_lm	21
compute_lm_simple	23
compute_mds	24
compute_prcomp	25
compute_tsne	26
compute_ttest	27
compute_umap	29
compute_wilcox	30
count.tidymatrix	32
filter.tidymatrix	33
get_analysis	34
groups.grouped_tidymatrix	35
group_by.tidymatrix	35
group_vars.grouped_tidymatrix	36
is_grouped_tidymatrix	37
is_tidymatrix	38
joins	38
list_analyses	41
log_transform	41
matrix-operations	42
mutate.tidymatrix	42
plot_pheatmap	43
print.grouped_tidymatrix	45
print.tidymatrix	46
pull.tidymatrix	46
pull_active	47
relocate.tidymatrix	48

<i>activate</i>	3
remove_analysis	49
rename.tidymatrix	49
scale.tidymatrix	50
select.tidymatrix	51
slice.tidymatrix	52
slice_head.tidymatrix	52
slice_sample.tidymatrix	53
summarize.grouped_tidymatrix	54
t.tidymatrix	55
tally.grouped_tidymatrix	56
tidymatrix	57
to_long	58
transform_matrix	59
ungroup.grouped_tidymatrix	61
Index	62

<i>activate</i>	<i>Activate different components of a tidymatrix</i>
-----------------	--

Description

Switch the active context of a tidymatrix to operate on rows, columns, or the matrix itself. This determines which component will be affected by subsequent dplyr operations.

Usage

```
activate(.data, what)
```

Arguments

- `.data` A tidymatrix object
- `what` Which component to activate. One of "rows", "columns", or "matrix"

Value

A tidymatrix object with the specified component activated

Examples

```
mat <- matrix(rnorm(12), nrow = 4, ncol = 3)
row_data <- data.frame(id = 1:4, group = c("A", "A", "B", "B"))
col_data <- data.frame(id = 1:3, type = c("x", "y", "z"))
tm <- tidymatrix(mat, row_data, col_data)

# Activate rows to filter/mutate row metadata
tm |> activate(rows)

# Activate columns to work with column metadata
```

```
tm |> activate(columns)

# Activate matrix to work with the matrix directly
tm |> activate(matrix)
```

active	<i>Get the active component of a tidymatrix</i>
--------	---

Description

Get the active component of a tidymatrix

Usage

```
active(.data)
```

Arguments

.data A tidymatrix object

Value

A character string indicating the active component

Examples

```
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

active(tm)
active(activate(tm, rows))
```

add_stats	<i>Add statistics to metadata</i>
-----------	-----------------------------------

Description

Compute statistics for the active dimension and add them as columns to the corresponding metadata. Requires rows or columns to be active.

Usage

```
add_stats(.data, ..., .fns = NULL, .names = NULL)
```

Arguments

<code>.data</code>	A tidymatrix object with rows or columns active
<code>...</code>	Functions to compute statistics (unquoted names like mean, var, sd)
<code>.fns</code>	Alternative way to specify functions as a list
<code>.names</code>	Names for the new columns. If NULL, uses function names.

Value

A tidymatrix object with statistics added to metadata

Examples

```
mat <- matrix(rnorm(20), nrow = 4, ncol = 5)
tm <- tidymatrix(mat)

# Add row statistics
tm <- tm |>
  activate(rows) |>
  add_stats(mean, var, sd)

# Now row_data has columns: mean, var, sd

# Add column statistics
tm <- tm |>
  activate(columns) |>
  add_stats(median, min, max)

# Custom names
tm <- tm |>
  activate(rows) |>
  add_stats(mean, var, .names = c("row_mean", "row_var"))
```

Description

Functions to store, retrieve, and manage analysis results attached to tidymatrix objects.

arrange.tidymatrix	<i>Arrange the active component of a tidymatrix</i>
--------------------	---

Description

Reorder rows based on the active metadata component. When rows are active, arranges by row_data and reorders matrix rows accordingly. When columns are active, arranges by col_data and reorders matrix columns accordingly. Cannot arrange when matrix is active.

Usage

```
## S3 method for class 'tidymatrix'  
arrange(.data, ...)
```

Arguments

.data	A tidymatrix object
...	Variables to arrange by

Details

Reordering invalidates any stored analyses (see [get_analysis](#)), since objects such as hclust and prcomp refer to the original row/column positions.

Value

A tidymatrix object with reordered data

Examples

```
library(dplyr, warn.conflicts = FALSE)  
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)  
  
# Order items by trait; the matrix columns are reordered to match  
tm |>  
  activate(columns) |>  
  arrange(trait, position)
```

`as.data.frame.tidymatrix`*Convert tidymatrix metadata to data.frame*

Description

Converts the active metadata component (row_data or col_data) to a data.frame. This method only works when rows or columns are active, not when the matrix is active.

Usage

```
## S3 method for class 'tidymatrix'  
as.data.frame(x, ...)
```

Arguments

x	A tidymatrix object
...	Additional arguments (currently unused)

Value

A data.frame containing the active metadata

Examples

```
mat <- matrix(rnorm(100), nrow = 10, ncol = 10)  
row_data <- data.frame(id = 1:10, group = rep(c("A", "B"), each = 5))  
tm <- tidymatrix(mat, row_data)  
  
# Convert row metadata to data.frame  
df <- tm |>  
  activate(rows) |>  
  compute_prcomp(center = TRUE, scale. = TRUE) |>  
  as.data.frame()
```

`as_tibble.tidymatrix` *Convert tidymatrix metadata to tibble*

Description

Converts the active metadata component (row_data or col_data) to a tibble. This method only works when rows or columns are active, not when the matrix is active.

Usage

```
## S3 method for class 'tidymatrix'  
as_tibble(x, ...)
```

Arguments

`x` A tidymatrix object

`...` Additional arguments (currently unused)

Value

A tibble containing the active metadata

Examples

```
mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
row_data <- data.frame(id = 1:10, group = rep(c("A", "B"), each = 5))
tm <- tidymatrix(mat, row_data)

# Convert row metadata to a tibble, e.g. for plotting
pcs <- tm |>
  activate(rows) |>
  compute_prcomp(center = TRUE, scale. = TRUE) |>
  as_tibble()

if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(pcs, aes(x = row_pca_PC1, y = row_pca_PC2, color = group)) +
    geom_point()
}
```

big5

Simulated Big Five personality survey

Description

A simulated personality questionnaire in which 400 respondents answer 30 Likert items, six for each of the Big Five personality traits. The data come in the three pieces that make up a tidymatrix, and can be combined with `tidymatrix(big5_responses, big5_respondents, big5_items)`.

Usage

```
big5_responses

big5_respondents

big5_items
```

Format

big5_responses An integer matrix with 400 rows (respondents) and 30 columns (items). Values range from 1 (strongly disagree) to 5 (strongly agree). Row and column names are respondent and item IDs.

big5_respondents A data frame with one row per respondent:

respondent_id Respondent ID, e.g. "R001".

age Age in years (18–79).

gender "Female", "Male" or "Non-binary".

education Highest completed education, a factor with levels Basic < Secondary < Bachelor < Master < PhD.

occupation Occupational group, e.g. "Student", "Professional", "Retired".

country Two-letter country code.

life_satisfaction Self-rated life satisfaction, 0–10.

completion_min Time taken to complete the survey, minutes.

big5_items A data frame with one row per item:

item_id Item ID: trait letter and number, e.g. "E1".

trait Big Five trait measured by the item.

reversed TRUE for reverse-keyed items.

item_text Statement shown to the respondent.

position Position of the item in the questionnaire; traits are interleaved.

Details

The data are simulated, but built to behave like real survey data:

- Answers are driven by latent traits, so items of the same trait are correlated and principal components or clustering recover the five traits once reverse-keyed items are re-scored.
- Two items per trait are reverse-keyed (`reversed = TRUE`); agreeing with them indicates a *low* trait level.
- Demographics are internally consistent: education is only possible from a plausible age onwards, students are young and retirees old, and occupation depends on education.
- Traits depend on demographics: conscientiousness and agreeableness increase with age while neuroticism decreases; women score higher on neuroticism and agreeableness; openness increases with education and is highest in creative occupations.
- Life satisfaction is related to the traits, most strongly (and negatively) to neuroticism.
- About 3\ in a few minutes and either gave the same answer to nearly every item or answered at random.

The script that generates the data is in the `data-raw` folder of the package source.

See Also

[big5_countries](#) for country-level information to join to the respondents.

Examples

```
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)
tm
```

big5_countries

Country information for the Big Five survey

Description

A small country-level table to join to [big5_respondents](#). It is deliberately incomplete: Germany ("DE") has respondents but no row here, and Norway ("NO") has a row but no respondents, which makes it useful for illustrating the different kinds of joins.

Usage

```
big5_countries
```

Format

A data frame with 6 rows and 5 columns:

country Two-letter country code.

country_name Country name.

region "Baltic" or "Nordic".

language Main official language.

population_m Approximate population, millions.

Examples

```
big5_countries
```

center

Center the active dimension of an object

Description

Generic for centering. See [center.tidymatrix](#) for the tidymatrix method.

Usage

```
center(x, ...)
```

Arguments

x An object to center
 ... Arguments passed to methods

Value

An object of the same class as x, centered

Examples

```
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

# Center each item (column) on its mean
tm |>
  activate(columns) |>
  center()
```

center.tidymatrix *Center rows or columns*

Description

Center the active dimension by subtracting the mean. Requires rows or columns to be active.

Usage

```
## S3 method for class 'tidymatrix'
center(x, ...)
```

Arguments

x A tidymatrix object with rows or columns active
 ... Not used

Value

A tidymatrix object with centered matrix

Examples

```
mat <- matrix(rnorm(20, mean = 10), nrow = 4, ncol = 5)
tm <- tidymatrix(mat)

# Center rows (mean of each row = 0)
tm_centered <- tm |>
  activate(rows) |>
  center()
```

```
# Center columns (mean of each column = 0)
tm_centered <- tm |>
  activate(columns) |>
  center()
```

check_analyses	<i>Check analysis validity</i>
----------------	--------------------------------

Description

Check if stored analyses are still valid given the current data dimensions.

Usage

```
check_analyses(x)
```

Arguments

x	A tidymatrix object
---	---------------------

Value

Invisibly returns x. Reports the status of each analysis as a message.

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(matrix(rnorm(100), 10, 10))
tm <- tm |>
  activate(columns) |>
  compute_prcomp(name = "pca")

check_analyses(tm)
# Analysis 'pca': VALID (10 rows x 10 columns)

tm <- tm |> activate(rows) |> filter(row_number() <= 5)
check_analyses(tm)
# Analysis 'pca': dimensions changed (was 10 rows, now 5 rows)
```

clip_values

*Clip matrix values***Description**

Cap matrix values at specified minimum and/or maximum. Requires matrix to be active.

Usage

```
clip_values(.data, min = NULL, max = NULL)
```

Arguments

.data	A tidymatrix object with matrix active
min	Minimum value (values below are set to this)
max	Maximum value (values above are set to this)

Value

A tidymatrix object with clipped matrix values

Examples

```
mat <- matrix(rnorm(20), nrow = 4, ncol = 5)
tm <- tidymatrix(mat)

# Clip to [-2, 2] range
tm_clipped <- tm |>
  activate(matrix) |>
  clip_values(min = -2, max = 2)

# Only set floor
tm_floor <- tm |>
  activate(matrix) |>
  clip_values(min = 0)
```

compute_across

*Apply a function across matrix dimensions with metadata access***Description**

Applies a user-defined function to each row (when rows are active) or each column (when columns are active), providing both the values and the opposite dimension's metadata. This enables complex statistical modeling where you need access to all annotations.

Usage

```
compute_across(
  .data,
  fn,
  add_to_data = FALSE,
  prefix = NULL,
  return_tibble = TRUE,
  ...
)
```

Arguments

<code>.data</code>	A tidymatrix object with rows or columns active (not matrix)
<code>fn</code>	A function with signature <code>function(values, metadata, ...)</code> where: <code>values</code>: Numeric vector of row/column values <code>metadata</code>: Complete <code>col_data</code> (rows active) or <code>row_data</code> (columns active) <code>...</code>: Additional arguments from <code>compute_across()</code> Must return a named list or named vector
<code>add_to_data</code>	Logical. If TRUE, adds results to <code>row_data/col_data</code> and returns modified tidy-matrix. If FALSE (default), returns <code>data.frame</code>
<code>prefix</code>	Character. Optional prefix for result column names when <code>add_to_data = TRUE</code>
<code>return_tibble</code>	Logical. If TRUE (default), returns tibble. If FALSE, returns <code>data.frame</code> . Only applies when <code>add_to_data = FALSE</code>
<code>...</code>	Additional arguments passed to <code>fn</code>

Value

If `add_to_data = FALSE`: `data.frame/tibble` with one row per matrix row/column, containing identifiers and computed statistics. If `add_to_data = TRUE`: modified tidymatrix with results added to metadata.

Examples

```
# T-test example
mat <- matrix(rnorm(100, mean = 10), nrow = 10, ncol = 10)
col_data <- data.frame(
  sample = paste0("S", 1:10),
  condition = rep(c("Control", "Treatment"), each = 5)
)
row_data <- data.frame(gene = paste0("Gene", 1:10))
tm <- tidymatrix(mat, row_data, col_data)

# Run t-test on each row
results <- tm |>
  activate(rows) |>
  compute_across(
    fn = function(vals, meta) {
      test <- t.test(vals ~ meta$condition)
```

```

      list(
        p.value = test$p.value,
        log2fc = log2(mean(vals[meta$condition == "Treatment"]) /
                      mean(vals[meta$condition == "Control"]))
      )
    }
  )

# Linear model with multiple predictors
col_data2 <- data.frame(
  sample = paste0("S", 1:10),
  condition = rep(c("Control", "Treatment"), each = 5),
  batch = factor(rep(1:2, 5)),
  age = rnorm(10, 50, 10)
)
tm2 <- tidymatrix(mat, row_data, col_data2)

lm_results <- tm2 |>
  activate(rows) |>
  compute_across(
    fn = function(vals, meta) {
      fit <- lm(vals ~ condition + batch + age, data = meta)
      summ <- summary(fit)
      coef_summ <- coef(summ)

      list(
        condition_pval = coef_summ["conditionTreatment", "Pr(>|t|)"],
        condition_coef = coef_summ["conditionTreatment", "Estimate"],
        r.squared = summ$r.squared
      )
    }
  )

# Add results to metadata
tm_with_stats <- tm |>
  activate(rows) |>
  compute_across(
    fn = function(vals, meta) {
      test <- t.test(vals ~ meta$condition)
      list(p.value = test$p.value)
    },
    add_to_data = TRUE,
    prefix = "ttest"
  )

```

Description

Performs one-way ANOVA for each row (or column) of a tidymatrix to test for differences across multiple groups.

Usage

```
compute_anova(
  .data,
  group_col,
  adjust = "fdr",
  add_to_data = FALSE,
  prefix = NULL,
  return_tibble = TRUE
)
```

Arguments

<code>.data</code>	A tidymatrix object with rows or columns active
<code>group_col</code>	Character. Name of column in metadata containing group labels
<code>adjust</code>	Character. Method for p-value adjustment. Default "fdr". Use "none" for no adjustment
<code>add_to_data</code>	Logical. If TRUE, adds results to row_data/col_data and returns modified tidymatrix. If FALSE (default), returns data.frame
<code>prefix</code>	Character. Optional prefix for result column names when <code>add_to_data = TRUE</code>
<code>return_tibble</code>	Logical. If TRUE (default), returns tibble. If FALSE, returns data.frame. Only applies when <code>add_to_data = FALSE</code>

Value

A data.frame/tibble with columns: identifiers, f.statistic, p.value, df_between, df_within, and p.adj (if adjustment applied)

Examples

```
mat <- matrix(rnorm(150), nrow = 10, ncol = 15)
col_data <- data.frame(
  sample = paste0("S", 1:15),
  condition = rep(c("A", "B", "C"), each = 5)
)
row_data <- data.frame(gene = paste0("Gene", 1:10))
tm <- tidymatrix(mat, row_data, col_data)

# ANOVA
results <- tm |>
  activate(rows) |>
  compute_anova(group_col = "condition")
```

compute_correlation	<i>Convenience wrapper for correlation tests</i>
---------------------	--

Description

Performs correlation tests between each row (or column) and a continuous variable in the metadata.

Usage

```
compute_correlation(
  .data,
  var,
  method = "pearson",
  adjust = "fdr",
  add_to_data = FALSE,
  prefix = NULL,
  return_tibble = TRUE
)
```

Arguments

.data	A tidymatrix object with rows or columns active
var	Character. Name of continuous variable in metadata to correlate with
method	Character. Correlation method: "pearson" (default), "spearman", or "kendall"
adjust	Character. Method for p-value adjustment. Default "fdr". Use "none" for no adjustment
add_to_data	Logical. If TRUE, adds results to row_data/col_data and returns modified tidy-matrix. If FALSE (default), returns data.frame
prefix	Character. Optional prefix for result column names when add_to_data = TRUE
return_tibble	Logical. If TRUE (default), returns tibble. If FALSE, returns data.frame. Only applies when add_to_data = FALSE

Value

A data.frame/tibble with columns: identifiers, correlation, p.value, and p.adj (if adjustment applied)

Examples

```
mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
col_data <- data.frame(
  sample = paste0("S", 1:10),
  age = rnorm(10, 50, 10)
)
row_data <- data.frame(gene = paste0("Gene", 1:10))
tm <- tidymatrix(mat, row_data, col_data)
```

```
# Pearson correlation
results <- tm |>
  activate(rows) |>
  compute_correlation(var = "age", method = "pearson")

# Spearman correlation
results <- tm |>
  activate(rows) |>
  compute_correlation(var = "age", method = "spearman")
```

compute_hclust

Compute hierarchical clustering on tidymatrix

Description

Perform hierarchical clustering on the matrix, adding cluster assignments to metadata and optionally storing the full hclust object.

Usage

```
compute_hclust(
  x,
  k = NULL,
  h = NULL,
  name = NULL,
  store = TRUE,
  method = "complete",
  dist_method = "euclidean",
  ...
)
```

Arguments

x	A tidymatrix object
k	Number of clusters to cut the tree into. If NULL, no cluster assignments are added (only dendrogram is stored).
h	Height at which to cut the tree. Alternative to k.
name	Name for this analysis. Default is "row_hclust" or "column_hclust" depending on active component.
store	If TRUE, stores the full hclust object for later retrieval with get_analysis(). Default is TRUE.
method	Agglomeration method for hclust. Default is "complete". Options: "ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median", "centroid".
dist_method	Distance method for dist(). Default is "euclidean". Options: "euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski".
...	Additional arguments passed to stats::dist()

Details

This function wraps `stats::hclust()` and `stats::dist()`, passing additional parameters directly to them.

Value

A `tidymatrix` object with cluster assignments added to metadata

Examples

```
mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
row_data <- data.frame(id = 1:10, group = rep(c("A", "B"), each = 5))
tm <- tidymatrix(mat, row_data)

# Cluster rows into 3 groups
tm <- tm |>
  activate(rows) |>
  compute_hclust(k = 3, method = "ward.D2")

# Now row_data has row_hclust_cluster column

# Get full hclust object for plotting
hc <- get_analysis(tm, "row_hclust")
plot(hc)

# Multiple clusterings with different k
tm <- tm |>
  activate(rows) |>
  compute_hclust(k = 3, name = "gene_k3") |>
  compute_hclust(k = 5, name = "gene_k5")
```

compute_kmeans

Compute k-means clustering on tidymatrix

Description

Perform k-means clustering on the matrix, adding cluster assignments to metadata and optionally storing the full kmeans object.

Usage

```
compute_kmeans(x, centers, name = NULL, store = TRUE, ...)
```

Arguments

x	A <code>tidymatrix</code> object
centers	Number of clusters (k) or a set of initial cluster centers.

name	Name for this analysis. Default is "row_kmeans" or "column_kmeans" depending on active component.
store	If TRUE, stores the full kmeans object for later retrieval with <code>get_analysis()</code> . Default is TRUE.
...	Additional arguments passed to <code>stats::kmeans()</code> , such as <code>iter.max</code> , <code>nstart</code> , <code>algorithm</code> , etc.

Details

This function wraps `stats::kmeans()`, passing additional parameters directly to it.

Value

A `tidymatrix` object with cluster assignments added to metadata

Examples

```
mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
row_data <- data.frame(id = 1:10)
tm <- tidymatrix(mat, row_data)

# K-means clustering with k=3
tm <- tm |>
  activate(rows) |>
  compute_kmeans(centers = 3, nstart = 25)

# Now row_data has row_kmeans_cluster column

# Get full kmeans object
km <- get_analysis(tm, "row_kmeans")
km$tot.withinss # Total within-cluster sum of squares
```

compute_kruskal

Convenience wrapper for Kruskal-Wallis test

Description

Performs Kruskal-Wallis tests for each row (or column) of a `tidymatrix`. This is a non-parametric alternative to one-way ANOVA.

Usage

```
compute_kruskal(
  .data,
  group_col,
  adjust = "fdr",
  add_to_data = FALSE,
  prefix = NULL,
  return_tibble = TRUE
)
```

Arguments

.data	A tidymatrix object with rows or columns active
group_col	Character. Name of column in metadata containing group labels
adjust	Character. Method for p-value adjustment. Default "fdr". Use "none" for no adjustment
add_to_data	Logical. If TRUE, adds results to row_data/col_data and returns modified tidymatrix. If FALSE (default), returns data.frame
prefix	Character. Optional prefix for result column names when add_to_data = TRUE
return_tibble	Logical. If TRUE (default), returns tibble. If FALSE, returns data.frame. Only applies when add_to_data = FALSE

Value

A data.frame/tibble with columns: identifiers, statistic, p.value, df, and p.adj (if adjustment applied)

Examples

```
mat <- matrix(rnorm(150), nrow = 10, ncol = 15)
col_data <- data.frame(
  sample = paste0("S", 1:15),
  condition = rep(c("A", "B", "C"), each = 5)
)
row_data <- data.frame(gene = paste0("Gene", 1:10))
tm <- tidymatrix(mat, row_data, col_data)

# Kruskal-Wallis test
results <- tm |>
  activate(rows) |>
  compute_kruskal(group_col = "condition")
```

compute_lm

Convenience wrapper for linear models

Description

Fits linear models for each row (or column) of a tidymatrix. Can return statistics for a specific coefficient or overall model fit statistics.

Usage

```
compute_lm(
  .data,
  formula_rhs,
  coef = NULL,
  adjust = "fdr",
  add_to_data = FALSE,
```

```

    prefix = NULL,
    return_tibble = TRUE,
    ...
  )

```

Arguments

<code>.data</code>	A tidymatrix object with rows or columns active
<code>formula_rhs</code>	Right-hand side of formula (e.g., <code>~ condition + batch + age</code>)
<code>coef</code>	Character. Name of coefficient to extract. If NULL, returns overall model statistics
<code>adjust</code>	Character. Method for p-value adjustment. Default "fdr". Use "none" for no adjustment
<code>add_to_data</code>	Logical. If TRUE, adds results to <code>row_data/col_data</code> and returns modified tidymatrix. If FALSE (default), returns <code>data.frame</code>
<code>prefix</code>	Character. Optional prefix for result column names when <code>add_to_data = TRUE</code>
<code>return_tibble</code>	Logical. If TRUE (default), returns tibble. If FALSE, returns <code>data.frame</code> . Only applies when <code>add_to_data = FALSE</code>
<code>...</code>	Additional arguments passed to <code>lm()</code>

Value

A `data.frame/tibble` with model statistics. If `coef` specified: estimate, p.value, se, and p.adj. If `coef = NULL`: r.squared, f.statistic, p.value, and p.adj

Examples

```

mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
col_data <- data.frame(
  sample = paste0("S", 1:10),
  condition = factor(rep(c("Control", "Treatment"), each = 5)),
  batch = factor(rep(1:2, 5)),
  age = rnorm(10, 50, 10)
)
row_data <- data.frame(gene = paste0("Gene", 1:10))
tm <- tidymatrix(mat, row_data, col_data)

# Extract specific coefficient
results <- tm |>
  activate(rows) |>
  compute_lm(
    formula_rhs = ~ condition + batch + age,
    coef = "conditionTreatment"
  )

# Overall model statistics
results <- tm |>
  activate(rows) |>
  compute_lm(formula_rhs = ~ condition + batch + age)

```

compute_lm_simple	<i>Convenience wrapper for simple linear regression</i>
-------------------	---

Description

Performs simple linear regression (one predictor) for each row (or column) of a tidymatrix. For multiple predictors, use `compute_lm()` instead.

Usage

```
compute_lm_simple(  
  .data,  
  predictor,  
  adjust = "fdr",  
  add_to_data = FALSE,  
  prefix = NULL,  
  return_tibble = TRUE  
)
```

Arguments

<code>.data</code>	A tidymatrix object with rows or columns active
<code>predictor</code>	Character. Name of predictor variable in metadata
<code>adjust</code>	Character. Method for p-value adjustment. Default "fdr". Use "none" for no adjustment
<code>add_to_data</code>	Logical. If TRUE, adds results to row_data/col_data and returns modified tidymatrix. If FALSE (default), returns data.frame
<code>prefix</code>	Character. Optional prefix for result column names when <code>add_to_data = TRUE</code>
<code>return_tibble</code>	Logical. If TRUE (default), returns tibble. If FALSE, returns data.frame. Only applies when <code>add_to_data = FALSE</code>

Value

A data.frame/tibble with columns: identifiers, slope, intercept, r.squared, p.value, and p.adj (if adjustment applied)

Examples

```
mat <- matrix(rnorm(100), nrow = 10, ncol = 10)  
col_data <- data.frame(  
  sample = paste0("S", 1:10),  
  age = rnorm(10, 50, 10)  
)  
row_data <- data.frame(gene = paste0("Gene", 1:10))  
tm <- tidymatrix(mat, row_data, col_data)
```

```
# Simple linear regression
results <- tm |>
  activate(rows) |>
  compute_lm_simple(predictor = "age")
```

compute_mds

Compute MDS on tidymatrix

Description

Perform Classical Multidimensional Scaling on the matrix, adding MDS coordinates to metadata and optionally storing the distance matrix and result.

Usage

```
compute_mds(
  x,
  name = NULL,
  k = 2,
  store = TRUE,
  dist_method = "euclidean",
  eig = FALSE,
  ...
)
```

Arguments

<code>x</code>	A tidymatrix object
<code>name</code>	Name for this analysis. Default is "row_mds" or "column_mds" depending on active component.
<code>k</code>	Number of dimensions for MDS embedding. Default is 2.
<code>store</code>	If TRUE, stores the MDS result for later retrieval with <code>get_analysis()</code> . Default is TRUE.
<code>dist_method</code>	Distance method for <code>dist()</code> . Default is "euclidean". Options: "euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski".
<code>eig</code>	If TRUE, return eigenvalues and GOF statistics (passed to <code>cmdscale()</code>).
<code>...</code>	Additional arguments passed to <code>stats::dist()</code> or <code>stats::cmdscale()</code>

Details

This function wraps `stats::cmdscale()` and `stats::dist()`.

Value

A tidymatrix object with MDS coordinates added to metadata

Examples

```

mat <- matrix(rnorm(500), nrow = 50, ncol = 10)
row_data <- data.frame(id = 1:50)
tm <- tidymatrix(mat, row_data)

# MDS on rows
tm <- tm |>
  activate(rows) |>
  compute_mds(k = 2)

# Now row_data has row_mds_1, row_mds_2 columns

# Get MDS result
mds_obj <- get_analysis(tm, "row_mds")

```

compute_prcomp

Compute PCA on tidymatrix

Description

Perform Principal Component Analysis on the matrix, adding PC scores to metadata and optionally storing the full prcomp object.

Usage

```
compute_prcomp(x, name = NULL, n_components = NULL, store = TRUE, ...)
```

Arguments

x	A tidymatrix object
name	Name for this analysis. Default is "row_pca" or "column_pca" depending on active component. Used as prefix for column names.
n_components	Number of PC components to add to metadata. Default is all components. Use a smaller number for large datasets.
store	If TRUE, stores the full prcomp object for later retrieval with get_analysis(). Default is TRUE.
...	Additional arguments passed to stats::prcomp(), such as center, scale., tol, etc.

Details

This function wraps stats::prcomp() and passes all additional parameters directly to it. The PC scores are added as columns to the active metadata (row_data or col_data).

Value

A tidymatrix object with PC scores added to metadata

Examples

```
mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
row_data <- data.frame(id = 1:10, group = rep(c("A", "B"), each = 5))
col_data <- data.frame(id = 1:10, type = rep(c("x", "y"), 5))
tm <- tidymatrix(mat, row_data, col_data)

# PCA on columns (samples)
tm <- tm |>
  activate(columns) |>
  compute_prcomp(center = TRUE, scale. = TRUE)

# Now col_data has column_pca_PC1, column_pca_PC2, etc.

# Custom name and limited components
tm <- tm |>
  activate(rows) |>
  compute_prcomp(name = "gene_pca", n_components = 3, center = TRUE)

# Get full prcomp object
pca_obj <- get_analysis(tm, "gene_pca")
summary(pca_obj)
plot(pca_obj$sdev^2 / sum(pca_obj$sdev^2)) # Variance explained
```

compute_tsne

Compute t-SNE on tidymatrix

Description

Perform t-distributed Stochastic Neighbor Embedding on the matrix, adding t-SNE coordinates to metadata and optionally storing the full Rtsne object.

Usage

```
compute_tsne(x, name = NULL, dims = 2, store = TRUE, perplexity = 30, ...)
```

Arguments

x	A tidymatrix object
name	Name for this analysis. Default is "row_tsne" or "column_tsne" depending on active component.
dims	Number of dimensions for t-SNE embedding. Default is 2.
store	If TRUE, stores the full Rtsne object for later retrieval with <code>get_analysis()</code> . Default is TRUE.
perplexity	Perplexity parameter (default 30). Should be less than the number of samples. Typical values are between 5 and 50.
...	Additional arguments passed to <code>Rtsne::Rtsne()</code> , such as <code>theta</code> , <code>max_iter</code> , <code>verbose</code> , etc.

Details

This function wraps `Rtsne::Rtsne()` and passes all additional parameters directly to it. Note that t-SNE is stochastic, so use `set.seed()` before calling for reproducible results.

Value

A tidymatrix object with t-SNE coordinates added to metadata

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE)) {
  mat <- matrix(rnorm(500), nrow = 50, ncol = 10)
  row_data <- data.frame(id = 1:50)
  tm <- tidymatrix(mat, row_data)

  # t-SNE on rows
  set.seed(42) # For reproducibility
  tm <- tm |>
    activate(rows) |>
    compute_tsne(dims = 2, perplexity = 10)

  # Now row_data has row_tsne_1, row_tsne_2 columns

  # Get full Rtsne object
  tsne_obj <- get_analysis(tm, "row_tsne")
}
```

compute_ttest

Convenience wrapper for t-tests

Description

Performs t-tests comparing two groups for each row (or column) of a tidymatrix. Automatically detects groups and applies multiple testing correction.

Usage

```
compute_ttest(
  .data,
  group_col,
  control = NULL,
  treatment = NULL,
  log2 = TRUE,
  adjust = "fdr",
  add_to_data = FALSE,
  prefix = NULL,
  return_tibble = TRUE,
  ...
)
```

Arguments

<code>.data</code>	A tidymatrix object with rows or columns active
<code>group_col</code>	Character. Name of column in metadata containing group labels
<code>control</code>	Character. Label for the control group. If NULL, it is inferred: when <code>group_col</code> is a factor, the first level present in the data; otherwise the first value in sorted order. If only <code>treatment</code> is given, <code>control</code> is the other group.
<code>treatment</code>	Character. Label for the treatment group. If NULL, the group that is not <code>control</code> . When either group is inferred, a message reports the comparison being made. Set both explicitly to silence it.
<code>log2</code>	Logical. If TRUE (default), computes log2 fold change. If FALSE, computes raw fold change
<code>adjust</code>	Character. Method for p-value adjustment. Default "fdr". See <code>?p.adjust</code> for options. Use "none" for no adjustment
<code>add_to_data</code>	Logical. If TRUE, adds results to <code>row_data/col_data</code> and returns modified tidymatrix. If FALSE (default), returns <code>data.frame</code>
<code>prefix</code>	Character. Optional prefix for result column names when <code>add_to_data = TRUE</code>
<code>return_tibble</code>	Logical. If TRUE (default), returns tibble. If FALSE, returns <code>data.frame</code> . Only applies when <code>add_to_data = FALSE</code>
<code>...</code>	Additional arguments passed to <code>t.test()</code>

Value

A `data.frame`/tibble with columns: `identifiers`, `p.value`, `log2fc` (or `fc`), and `p.adj` (if adjustment applied)

Examples

```
mat <- matrix(rnorm(100, mean = 10), nrow = 10, ncol = 10)
col_data <- data.frame(
  sample = paste0("S", 1:10),
  condition = rep(c("Control", "Treatment"), each = 5)
)
row_data <- data.frame(gene = paste0("Gene", 1:10))
tm <- tidymatrix(mat, row_data, col_data)

# Simple t-test
results <- tm |>
  activate(rows) |>
  compute_ttest(group_col = "condition")

# With specific group labels
results <- tm |>
  activate(rows) |>
  compute_ttest(
    group_col = "condition",
    control = "Control",
    treatment = "Treatment"
  )
```

compute_umap	<i>Compute UMAP on tidymatrix</i>
--------------	-----------------------------------

Description

Perform Uniform Manifold Approximation and Projection on the matrix, adding UMAP coordinates to metadata and optionally storing the full umap object.

Usage

```
compute_umap(
  x,
  name = NULL,
  n_components = 2,
  store = TRUE,
  n_neighbors = 15,
  min_dist = 0.1,
  metric = "euclidean",
  random_state = NULL,
  ...
)
```

Arguments

x	A tidymatrix object
name	Name for this analysis. Default is "row_umap" or "column_umap" depending on active component.
n_components	Number of dimensions for UMAP embedding. Default is 2.
store	If TRUE, stores the full umap object for later retrieval with <code>get_analysis()</code> . Default is TRUE.
n_neighbors	Size of local neighborhood (default 15). Larger values preserve more global structure, smaller values preserve more local structure.
min_dist	Minimum distance between points in low-dimensional space (default 0.1). Smaller values create tighter, more separated clusters.
metric	Distance metric to use (default "euclidean"). Options include "manhattan", "cosine", "correlation", etc.
random_state	Seed for reproducibility (default NULL). Set to an integer for reproducible results.
...	Additional configuration passed via <code>umap.defaults</code>

Details

This function wraps `umap::umap()` and passes additional parameters through the `config` parameter. UMAP is generally faster than t-SNE and better preserves global structure.

Value

A tidymatrix object with UMAP coordinates added to metadata

Examples

```
if (requireNamespace("umap", quietly = TRUE)) {
  mat <- matrix(rnorm(500), nrow = 50, ncol = 10)
  row_data <- data.frame(id = 1:50)
  tm <- tidymatrix(mat, row_data)

  # UMAP on rows
  tm <- tm |>
    activate(rows) |>
    compute_umap(n_components = 2, random_state = 42)

  # Now row_data has row_umap_1, row_umap_2 columns

  # Get full umap object
  umap_obj <- get_analysis(tm, "row_umap")
}
```

compute_wilcox

Convenience wrapper for Wilcoxon rank-sum test

Description

Performs Wilcoxon rank-sum tests (Mann-Whitney U test) comparing two groups for each row (or column) of a tidymatrix. This is a non-parametric alternative to the t-test that doesn't assume normal distribution.

Usage

```
compute_wilcox(
  .data,
  group_col,
  control = NULL,
  treatment = NULL,
  log2 = TRUE,
  adjust = "fdr",
  add_to_data = FALSE,
  prefix = NULL,
  return_tibble = TRUE,
  ...
)
```

Arguments

<code>.data</code>	A tidymatrix object with rows or columns active
<code>group_col</code>	Character. Name of column in metadata containing group labels
<code>control</code>	Character. Label for the control group. If NULL, it is inferred: when <code>group_col</code> is a factor, the first level present in the data; otherwise the first value in sorted order. If only <code>treatment</code> is given, <code>control</code> is the other group.
<code>treatment</code>	Character. Label for the treatment group. If NULL, the group that is not <code>control</code> . When either group is inferred, a message reports the comparison being made. Set both explicitly to silence it.
<code>log2</code>	Logical. If TRUE (default), computes log2 fold change. If FALSE, computes raw fold change
<code>adjust</code>	Character. Method for p-value adjustment. Default "fdr". See <code>?p.adjust</code> for options. Use "none" for no adjustment
<code>add_to_data</code>	Logical. If TRUE, adds results to <code>row_data/col_data</code> and returns modified tidymatrix. If FALSE (default), returns <code>data.frame</code>
<code>prefix</code>	Character. Optional prefix for result column names when <code>add_to_data = TRUE</code>
<code>return_tibble</code>	Logical. If TRUE (default), returns tibble. If FALSE, returns <code>data.frame</code> . Only applies when <code>add_to_data = FALSE</code>
<code>...</code>	Additional arguments passed to <code>wilcox.test()</code>

Value

A `data.frame/tibble` with columns: `identifiers`, `p.value`, `log2fc` (or `fc`), `median_diff`, and `p.adj` (if adjustment applied)

Examples

```
mat <- matrix(rnorm(100, mean = 10), nrow = 10, ncol = 10)
col_data <- data.frame(
  sample = paste0("S", 1:10),
  condition = rep(c("Control", "Treatment"), each = 5)
)
row_data <- data.frame(gene = paste0("Gene", 1:10))
tm <- tidymatrix(mat, row_data, col_data)

# Wilcoxon test
results <- tm |>
  activate(rows) |>
  compute_wilcox(group_col = "condition")
```

count.tidymatrix	<i>Count observations by group</i>
------------------	------------------------------------

Description

Count the number of observations in each group, aggregating the matrix. Returns a tidymatrix (not a tibble).

Usage

```
## S3 method for class 'tidymatrix'
count(
  x,
  ...,
  wt = NULL,
  sort = FALSE,
  name = NULL,
  .drop = TRUE,
  .matrix_fn = NULL,
  .matrix_args = list()
)
```

Arguments

x	A tidymatrix object
...	Variables to group by
wt	Frequency weights (not yet implemented)
sort	If TRUE, sort output in descending order of n
name	Name of count column (default: "n")
.drop	Drop groups with zero observations
.matrix_fn	Function to aggregate matrix values. Default is mean for numeric matrices. Required for non-numeric matrices.
.matrix_args	List of additional arguments to pass to .matrix_fn

Value

A tidymatrix object with counts and aggregated matrix

Examples

```
library(dplyr, warn.conflicts = FALSE)
mat <- matrix(rnorm(20), nrow = 10, ncol = 2)
row_data <- data.frame(
  id = 1:10,
  group = rep(c("A", "B"), each = 5),
  subgroup = rep(c("x", "y"), 5)
```

```

)
tm <- tidymatrix(mat, row_data)

# Count by single variable
tm |>
  activate(rows) |>
  count(group)

# Count by multiple variables
tm |>
  activate(rows) |>
  count(group, subgroup)

# Use different aggregation for matrix
tm |>
  activate(rows) |>
  count(group, .matrix_fn = median)

```

filter.tidymatrix	<i>Filter rows or columns of a tidymatrix</i>
-------------------	---

Description

Filter the active component of a tidymatrix object. When rows are active, filters row_data and the corresponding matrix rows. When columns are active, filters col_data and the corresponding matrix columns. Cannot filter when matrix is active.

Usage

```
## S3 method for class 'tidymatrix'
filter(.data, ..., .preserve = FALSE)
```

Arguments

.data	A tidymatrix object
...	Logical predicates for filtering
.preserve	Not used (for compatibility with dplyr)

Value

A tidymatrix object with filtered data

Examples

```

library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

# Keep respondents aged 60 or over; the matrix rows follow
tm |>

```

```

activate(rows) |>
filter(age >= 60)

# Keep only the Extraversion items
tm |>
  activate(columns) |>
  filter(trait == "Extraversion")

```

get_analysis	<i>Get stored analysis object</i>
--------------	-----------------------------------

Description

Retrieve a full analysis object (prcomp, hclust, etc.) that was stored using a compute_* function.

Usage

```
get_analysis(x, name)
```

Arguments

x	A tidymatrix object
name	Name of the analysis to retrieve

Value

The stored analysis object (class depends on analysis type)

Examples

```

tm <- tidymatrix(matrix(rnorm(100), 10, 10))
tm <- tm |>
  activate(columns) |>
  compute_prcomp(name = "pca")

# Get the full prcomp object
pca_obj <- get_analysis(tm, "pca")
summary(pca_obj)
plot(pca_obj$sdev) # Scree plot

```

```
groups.grouped_tidymatrix
```

Get grouping variables

Description

Get grouping variables

Usage

```
## S3 method for class 'grouped_tidymatrix'
groups(x)
```

Arguments

x A grouped_tidymatrix object

Value

A list of grouping variable names

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(rows) |>
  group_by(country, gender) |>
  groups()
```

```
group_by.tidymatrix      Group a tidymatrix by variables in metadata
```

Description

Create a grouped tidymatrix for use with summarize(). Groups are created based on the active metadata component (row_data or col_data).

Usage

```
## S3 method for class 'tidymatrix'
group_by(.data, ..., .add = FALSE, .drop = TRUE)
```

Arguments

<code>.data</code>	A tidymatrix object
<code>...</code>	Variables to group by (unquoted names or expressions)
<code>.add</code>	When FALSE (default), <code>group_by()</code> will override existing groups. When TRUE, add to existing groups.
<code>.drop</code>	Drop groups with zero observations

Value

A `grouped_tidymatrix` object

Examples

```
library(dplyr, warn.conflicts = FALSE)
mat <- matrix(rnorm(20), nrow = 5, ncol = 4)
row_data <- data.frame(
  id = 1:5,
  group = c("A", "A", "B", "B", "C")
)
tm <- tidymatrix(mat, row_data)

# Group by a variable
tm_grouped <- tm |>
  activate(rows) |>
  group_by(group)

# Multiple grouping variables
row_data2 <- data.frame(
  id = 1:5,
  condition = c("ctrl", "ctrl", "treat", "treat", "treat"),
  batch = c(1, 2, 1, 2, 1)
)
tm2 <- tidymatrix(mat, row_data2)
tm_grouped2 <- tm2 |>
  activate(rows) |>
  group_by(condition, batch)
```

```
group_vars.grouped_tidymatrix
  Get grouping variable names
```

Description

Get grouping variable names

Usage

```
## S3 method for class 'grouped_tidymatrix'
group_vars(x)
```

Arguments

x A grouped_tidymatrix object

Value

A character vector of grouping variable names

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(rows) |>
  group_by(country, gender) |>
  group_vars()
```

is_grouped_tidymatrix *Check if object is a grouped tidymatrix*

Description

Check if object is a grouped tidymatrix

Usage

```
is_grouped_tidymatrix(x)
```

Arguments

x An object to test

Value

TRUE if the object is a grouped_tidymatrix, FALSE otherwise

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

is_grouped_tidymatrix(tm)
is_grouped_tidymatrix(tm |> activate(rows) |> group_by(country))
```

is_tidymatrix	<i>Check if an object is a tidymatrix</i>
---------------	---

Description

Check if an object is a tidymatrix

Usage

```
is_tidymatrix(x)
```

Arguments

x An object to test

Value

TRUE if the object is a tidymatrix, FALSE otherwise

Examples

```
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

is_tidymatrix(tm)
is_tidymatrix(big5_responses)
```

joins	<i>Join tidymatrix with another data frame</i>
-------	--

Description

These functions are tidymatrix methods for dplyr's join functions. They join the row_data or col_data (depending on which is active) with an external data.frame, and appropriately update the matrix dimensions.

Usage

```
## S3 method for class 'tidymatrix'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL
```

```

)

## S3 method for class 'tidymatrix'
right_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL
)

## S3 method for class 'tidymatrix'
inner_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL
)

## S3 method for class 'tidymatrix'
full_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL
)

## S3 method for class 'tidymatrix'
semi_join(x, y, by = NULL, copy = FALSE, ...)

## S3 method for class 'tidymatrix'
anti_join(x, y, by = NULL, copy = FALSE, ...)

```

Arguments

<code>x</code>	A tidymatrix object
<code>y</code>	A data frame or tibble to join with
<code>by</code>	A character vector of variables to join by. If NULL, uses all variables that appear in both tables.
<code>copy</code>	If y is not a data frame or tibble, this controls whether to copy it or not.

suffix	If there are non-joined duplicate variables in x and y, these suffixes will be added to disambiguate them.
...	Additional arguments passed to the corresponding dplyr join function
keep	Control which join keys to preserve in the output (see left_join).

Details

Joins work on the active dimension (rows or columns). Use `activate()` to specify which metadata to join.

When joins add new rows/columns (e.g., `right_join`, `full_join`), the matrix is expanded with NA values for the new entries.

When joins remove rows/columns (e.g., `inner_join`, `semi_join`, `anti_join`), the matrix is subset accordingly.

All joins invalidate stored analyses, as the matrix dimensions may have changed.

Value

A `tidymatrix` object with joined metadata and updated matrix

Examples

```
library(dplyr)

# Create example tidymatrix
mat <- matrix(rnorm(50), nrow = 10, ncol = 5)
row_data <- data.frame(gene_id = paste0("Gene_", 1:10))
col_data <- data.frame(sample_id = paste0("Sample_", 1:5))
tm <- tidymatrix(mat, row_data, col_data)

# Create external annotation data
annotations <- data.frame(
  gene_id = paste0("Gene_", c(1:8, 15:17)),
  pathway = sample(c("A", "B"), 11, replace = TRUE)
)

# Left join - keep all genes from tidymatrix
tm_left <- tm |>
  activate(rows) |>
  left_join(annotations, by = "gene_id")

# Inner join - keep only matching genes
tm_inner <- tm |>
  activate(rows) |>
  inner_join(annotations, by = "gene_id")

# Full join - keep all genes from both
tm_full <- tm |>
  activate(rows) |>
  full_join(annotations, by = "gene_id")
```

list_analyses	<i>List stored analyses</i>
---------------	-----------------------------

Description

Get the names of all analyses stored in a tidymatrix object.

Usage

```
list_analyses(x)
```

Arguments

x	A tidymatrix object
---	---------------------

Value

Character vector of analysis names

Examples

```
tm <- tidymatrix(matrix(rnorm(100), 10, 10))
tm <- tm |>
  activate(columns) |>
  compute_prcomp(name = "pca") |>
  compute_hclust(k = 3, name = "clusters")

list_analyses(tm)
# [1] "pca" "clusters"
```

log_transform	<i>Log transform matrix</i>
---------------	-----------------------------

Description

Apply log transformation to matrix values. Requires matrix to be active. This is a convenience wrapper around [transform_matrix\(\)](#).

Usage

```
log_transform(.data, base = 2, offset = 1)
```

Arguments

.data	A tidymatrix object with matrix active
base	Logarithm base (2, 10, or "natural" for ln)
offset	Value to add before log transform (default 1, for log(x + 1))

Value

A tidymatrix object with log-transformed matrix

Examples

```
mat <- matrix(abs(rnorm(20)), nrow = 4, ncol = 5)
tm <- tidymatrix(mat)

# Log2 transform with pseudocount
tm_log <- tm |>
  activate(matrix) |>
  log_transform(base = 2, offset = 1)

# Natural log
tm_ln <- tm |>
  activate(matrix) |>
  log_transform(base = "natural")
```

matrix-operations	<i>Matrix Operations for tidymatrix</i>
-------------------	---

Description

Functions for transforming and manipulating the matrix component of tidymatrix objects.

mutate.tidymatrix	<i>Mutate the active component of a tidymatrix</i>
-------------------	--

Description

Add or modify columns in the active metadata component. When rows are active, mutates row_data. When columns are active, mutates col_data. Cannot mutate when matrix is active.

Usage

```
## S3 method for class 'tidymatrix'
mutate(.data, ...)
```

Arguments

.data	A tidymatrix object
...	Name-value pairs for new or modified columns

Value

A tidymatrix object with mutated metadata

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(rows) |>
  mutate(age_group = if_else(age < 40, "young", "older"))
```

plot_pheatmap

*Create a pheatmap from tidymatrix***Description**

Generate a heatmap using the pheatmap package, automatically using tidymatrix metadata for annotations and stored clustering results.

Usage

```
plot_pheatmap(
  x,
  row_names = NULL,
  col_names = NULL,
  row_annotation = NULL,
  col_annotation = NULL,
  row_cluster = NULL,
  col_cluster = NULL,
  ...
)
```

Arguments

x	A tidymatrix object
row_names	Column name from row_data to use for row names. If NULL, uses sequential numbers.
col_names	Column name from col_data to use for column names. If NULL, uses sequential numbers.
row_annotation	Character vector of column names from row_data to include as row annotations. If NULL, includes all columns except the name column. Set to FALSE to exclude row annotations.
col_annotation	Character vector of column names from col_data to include as column annotations. If NULL, includes all columns except the name column. Set to FALSE to exclude column annotations.
row_cluster	Name of stored hclust analysis to use for row clustering, or TRUE to let pheatmap cluster, or FALSE for no clustering, or NULL to auto-detect stored clustering. Default NULL (auto-detect).

`col_cluster` Name of stored hclust analysis to use for column clustering, or TRUE to let pheatmap cluster, or FALSE for no clustering, or NULL to auto-detect stored clustering. Default NULL (auto-detect).

`...` Additional arguments passed to `pheatmap::pheatmap()`

Value

A pheatmap object

Examples

```
if (requireNamespace("pheatmap", quietly = TRUE)) {
  mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
  row_data <- data.frame(
    gene = paste0("Gene_", 1:10),
    type = rep(c("A", "B"), each = 5)
  )
  col_data <- data.frame(
    sample = paste0("Sample_", 1:10),
    condition = rep(c("Control", "Treatment"), 5)
  )
  tm <- tidymatrix(mat, row_data, col_data)

  # Basic heatmap (auto-detects stored clustering if available)
  plot_pheatmap(tm, row_names = "gene", col_names = "sample")

  # With stored clustering (auto-detected)
  tm <- tm |>
    activate(rows) |>
    compute_hclust(k = 2, name = "gene_clusters") |>
    activate(columns) |>
    compute_hclust(k = 2, name = "sample_clusters")

  # Auto-detects and uses gene_clusters and sample_clusters
  plot_pheatmap(tm, row_names = "gene", col_names = "sample")

  # Explicitly specify which clustering to use
  plot_pheatmap(tm,
    row_names = "gene",
    col_names = "sample",
    row_cluster = "gene_clusters",
    col_cluster = "sample_clusters"
  )

  # No clustering (explicit)
  plot_pheatmap(tm,
    row_names = "gene",
    col_names = "sample",
    row_cluster = FALSE,
    col_cluster = FALSE
  )
}
```

```
# Automatic pheatmap clustering (not using stored)
plot_pheatmap(tm,
  row_names = "gene",
  col_names = "sample",
  row_cluster = TRUE,
  col_cluster = TRUE
)
}
```

```
print.grouped_tidymatrix
```

Print a grouped tidymatrix

Description

Print a grouped tidymatrix

Usage

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

Arguments

x	A grouped_tidymatrix object
...	Additional arguments (currently unused)

Value

x, invisibly.

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(rows) |>
  group_by(country) |>
  print()
```

print.tidymatrix	<i>Print a tidymatrix object</i>
------------------	----------------------------------

Description

Print a tidymatrix object

Usage

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

Arguments

x	A tidymatrix object
...	Additional arguments (currently unused)

Value

x, invisibly.

Examples

```
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

print(tm)
print(activate(tm, columns))
```

pull.tidymatrix	<i>Extract a column from metadata as a vector</i>
-----------------	---

Description

Extract a single column from the active metadata component as a vector.

Usage

```
## S3 method for class 'tidymatrix'
pull(.data, var = -1, ...)
```

Arguments

.data	A tidymatrix object
var	Column name to extract (can be unquoted)
...	Not used

Value

A vector containing the column values

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(columns) |>
  pull(trait) |>
  table()
```

pull_active	<i>Extract the active component from a tidymatrix</i>
-------------	---

Description

Returns the currently active component of a tidymatrix object. This can be the matrix itself, the row metadata, or the column metadata, depending on what is currently activated.

Usage

```
pull_active(.data)
```

Arguments

.data A tidymatrix object

Value

The active component:

- If "matrix" is active: returns the numeric matrix
- If "rows" is active: returns the row_data data.frame
- If "columns" is active: returns the col_data data.frame

Examples

```
mat <- matrix(rnorm(100), nrow = 10, ncol = 10)
row_data <- data.frame(id = 1:10, group = rep(c("A", "B"), each = 5))
tm <- tidymatrix(mat, row_data)

# Extract matrix
tm |>
  activate(matrix) |>
  pull_active()
```

```
# Extract row metadata, e.g. for plotting
pcs <- tm |>
  activate(rows) |>
  compute_prcomp(center = TRUE, scale. = TRUE) |>
  pull_active()

if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(pcs, aes(x = row_pca_PC1, y = row_pca_PC2, color = group)) +
    geom_point()
}
```

relocate.tidymatrix *Relocate columns in metadata*

Description

Change the order of columns in the active metadata component.

Usage

```
## S3 method for class 'tidymatrix'
relocate(.data, ..., .before = NULL, .after = NULL)
```

Arguments

<code>.data</code>	A tidymatrix object
<code>...</code>	Columns to move
<code>.before</code>	Column to move before
<code>.after</code>	Column to move after

Value

A tidymatrix object with reordered metadata columns

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(rows) |>
  relocate(country, .after = respondent_id)
```

remove_analysis	<i>Remove stored analysis</i>
-----------------	-------------------------------

Description

Remove a stored analysis object from a tidymatrix. This removes the full analysis object but keeps any metadata columns that were added.

Usage

```
remove_analysis(x, name = NULL)
```

Arguments

x	A tidymatrix object
name	Name of the analysis to remove. If NULL, removes all analyses.

Value

A tidymatrix object with the analysis removed

Examples

```
tm <- tidymatrix(matrix(rnorm(100), 10, 10))
tm <- tm |>
  activate(columns) |>
  compute_prcomp(name = "pca")

# Remove specific analysis
tm <- remove_analysis(tm, "pca")

# Remove all analyses
tm <- remove_analysis(tm, NULL)
```

rename.tidymatrix	<i>Rename columns in metadata</i>
-------------------	-----------------------------------

Description

Rename columns in the active metadata component. When rows are active, renames columns in row_data. When columns are active, renames columns in col_data.

Usage

```
## S3 method for class 'tidymatrix'
rename(.data, ...)
```

Arguments

`.data` A tidymatrix object

`...` Name-value pairs for renaming (`new_name = old_name`)

Value

A tidymatrix object with renamed metadata columns

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(columns) |>
  rename(item = item_id)
```

scale.tidymatrix	<i>Scale rows or columns</i>
------------------	------------------------------

Description

Perform z-score scaling (center and scale to unit variance) on the active dimension. Requires rows or columns to be active.

Usage

```
## S3 method for class 'tidymatrix'
scale(x, center = TRUE, scale = TRUE)
```

Arguments

`x` A tidymatrix object with rows or columns active

`center` If TRUE (default), center to mean = 0

`scale` If TRUE (default), scale to sd = 1

Value

A tidymatrix object with scaled matrix

Examples

```
mat <- matrix(rnorm(20, mean = 10, sd = 5), nrow = 4, ncol = 5)
tm <- tidymatrix(mat)

# Scale rows (z-score per row)
tm_scaled <- tm |>
  activate(rows) |>
  scale()

# Scale columns
tm_scaled <- tm |>
  activate(columns) |>
  scale()

# Only center, don't scale
tm_centered <- tm |>
  activate(rows) |>
  scale(center = TRUE, scale = FALSE)
```

select.tidymatrix	<i>Select columns from row or column metadata</i>
-------------------	---

Description

Select columns from the active metadata component. When rows are active, selects from row_data. When columns are active, selects from col_data. Cannot select when matrix is active.

Usage

```
## S3 method for class 'tidymatrix'
select(.data, ...)
```

Arguments

.data	A tidymatrix object
...	Column selection expressions

Value

A tidymatrix object with selected metadata columns

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(rows) |>
  select(respondent_id, age, country)
```

slice.tidymatrix	<i>Slice rows or columns by position</i>
------------------	--

Description

Select rows or columns by their integer positions. When rows are active, slices row_data and the corresponding matrix rows. When columns are active, slices col_data and the corresponding matrix columns.

Usage

```
## S3 method for class 'tidymatrix'
slice(.data, ..., .preserve = FALSE)
```

Arguments

.data	A tidymatrix object
...	Integer positions or expressions to select
.preserve	Not used (for compatibility with dplyr)

Value

A tidymatrix object with sliced data

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(rows) |>
  slice(1:10)
```

slice_head.tidymatrix	<i>Select first or last rows/columns</i>
-----------------------	--

Description

Select first or last rows/columns

Usage

```
## S3 method for class 'tidymatrix'
slice_head(.data, n, prop, ...)

## S3 method for class 'tidymatrix'
slice_tail(.data, n, prop, ...)
```

Arguments

<code>.data</code>	A tidymatrix object
<code>n</code>	Number of rows/columns to select
<code>prop</code>	Proportion of rows/columns to select
<code>...</code>	Not used

Value

A tidymatrix object with selected data

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(rows) |>
  slice_head(n = 5)

tm |>
  activate(columns) |>
  slice_tail(prop = 0.1)
```

slice_sample.tidymatrix

Select a random sample of rows/columns

Description

Select a random sample of rows/columns

Usage

```
## S3 method for class 'tidymatrix'
slice_sample(.data, n, prop, weight_by = NULL, replace = FALSE, ...)
```

Arguments

<code>.data</code>	A tidymatrix object
<code>n</code>	Number of rows/columns to select
<code>prop</code>	Proportion of rows/columns to select
<code>weight_by</code>	Sampling weights (not yet implemented)
<code>replace</code>	Sample with replacement
<code>...</code>	Not used

Value

A tidymatrix object with sampled data

Examples

```
library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

tm |>
  activate(rows) |>
  slice_sample(n = 20)
```

```
summarize.grouped_tidymatrix
      Summarize grouped tidymatrix
```

Description

Aggregate grouped rows or columns, applying summary functions to metadata and aggregating the matrix. For numeric matrices, the default aggregation is `mean()`. For non-numeric matrices, you must specify `.matrix_fn`.

Usage

```
## S3 method for class 'grouped_tidymatrix'
summarize(.data, ..., .matrix_fn = NULL, .matrix_args = list(), .groups = NULL)

## S3 method for class 'grouped_tidymatrix'
summarise(.data, ..., .matrix_fn = NULL, .matrix_args = list(), .groups = NULL)
```

Arguments

<code>.data</code>	A <code>grouped_tidymatrix</code> object
<code>...</code>	Name-value pairs of summary functions for metadata
<code>.matrix_fn</code>	Function to aggregate matrix values within each group. Default is <code>mean</code> for numeric matrices. Required for non-numeric matrices.
<code>.matrix_args</code>	List of additional arguments to pass to <code>.matrix_fn</code> (e.g., <code>list(na.rm = TRUE)</code>)
<code>.groups</code>	Grouping structure of result (same as <code>dplyr::summarize</code>)

Value

An ungrouped tidymatrix object with aggregated data. Stored analysis objects are removed (with a warning), because they describe the data before aggregation; metadata columns are kept.

Examples

```
library(dplyr, warn.conflicts = FALSE)
mat <- matrix(rnorm(20), nrow = 10, ncol = 2)
row_data <- data.frame(
  id = 1:10,
  group = rep(c("A", "B"), each = 5)
)
tm <- tidymatrix(mat, row_data)

# Summarize with default (mean) for numeric matrix
tm |>
  activate(rows) |>
  group_by(group) |>
  summarize(n = n(), avg_id = mean(id))

# Use different aggregation function
tm |>
  activate(rows) |>
  group_by(group) |>
  summarize(n = n(), .matrix_fn = median)

# With additional arguments
mat_na <- mat
mat_na[1, 1] <- NA
tm_na <- tidymatrix(mat_na, row_data)
tm_na |>
  activate(rows) |>
  group_by(group) |>
  summarize(n = n(), .matrix_fn = mean, .matrix_args = list(na.rm = TRUE))
```

t.tidymatrix

*Transpose a tidymatrix***Description**

Transpose the matrix and swap row and column metadata. This operation works regardless of which component is active. Uses base R's `t()` generic, so it works correctly even when tidyverse is loaded.

Usage

```
## S3 method for class 'tidymatrix'
t(x)
```

Arguments

x A tidymatrix object

Value

A tidymatrix object with transposed matrix and swapped metadata

Examples

```
mat <- matrix(1:12, nrow = 3, ncol = 4)
row_data <- data.frame(gene = paste0("G", 1:3))
col_data <- data.frame(sample = paste0("S", 1:4))
tm <- tidymatrix(mat, row_data, col_data)

# Transpose: genes × samples → samples × genes
tm_t <- t(tm)
dim(tm_t$matrix) # Now 4 × 3
```

```
tally.grouped_tidymatrix
```

Count observations in groups

Description

Count observations within existing groups. This is a wrapper around `summarize(n = n())`.

Usage

```
## S3 method for class 'grouped_tidymatrix'
tally(x, wt = NULL, sort = FALSE, name = NULL)
```

Arguments

<code>x</code>	A <code>grouped_tidymatrix</code> object
<code>wt</code>	Frequency weights (not yet implemented)
<code>sort</code>	If TRUE, sort output in descending order of <code>n</code>
<code>name</code>	Name of count column (default: "n")

Details

The matrix is aggregated with `mean()`, which requires a numeric matrix. `dplyr::tally()` takes no `...`, so the aggregation function cannot be overridden here; call `summarize(n = n(), .matrix_fn = ...)` directly instead.

Value

A `tidymatrix` object with counts and aggregated matrix

Examples

```
library(dplyr, warn.conflicts = FALSE)
mat <- matrix(rnorm(20), nrow = 10, ncol = 2)
row_data <- data.frame(
  id = 1:10,
  group = rep(c("A", "B"), each = 5)
)
tm <- tidymatrix(mat, row_data)

# Tally within groups
tm |>
  activate(rows) |>
  group_by(group) |>
  tally()
```

tidymatrix

*Create a tidymatrix object***Description**

A tidymatrix combines a matrix with row and column metadata, enabling tidyverse-style manipulation. The object stores three components: the matrix data, row annotations, and column annotations.

Usage

```
tidymatrix(matrix, row_data = NULL, col_data = NULL)
```

Arguments

matrix	A numeric matrix
row_data	A data.frame with row metadata. Must have same number of rows as the matrix. If NULL, a data.frame with row indices is created.
col_data	A data.frame with column metadata. Must have same number of rows as the matrix has columns. If NULL, a data.frame with column indices is created.

Value

A tidymatrix object

Examples

```
# Create a simple tidymatrix
mat <- matrix(rnorm(12), nrow = 4, ncol = 3)
row_data <- data.frame(
  person_id = 1:4,
  age = c(25, 30, 35, 40),
  gender = c("M", "F", "F", "M")
)
```

```
col_data <- data.frame(
  question_id = 1:3,
  type = c("numeric", "categorical", "numeric")
)
tm <- tidymatrix(mat, row_data, col_data)
```

to_long	<i>Convert tidymatrix to long-format data.frame</i>
---------	---

Description

Converts a tidymatrix into a long-format data.frame where each row represents a single matrix cell with its associated row and column metadata. This is useful for plotting individual data points or performing analyses that require long-format data.

Usage

```
to_long(.data, return_tibble = TRUE)
```

Arguments

.data	A tidymatrix object
return_tibble	Logical. If TRUE (default), returns tibble. If FALSE, returns data.frame.

Details

The conversion always processes the entire matrix regardless of which component is active. If column names conflict between row_data and col_data, all row metadata columns are prefixed with "row." and all column metadata columns are prefixed with "col." to avoid ambiguity.

Matrix values are unwrapped in column-major order (R's default), meaning all values from column 1, then all values from column 2, etc.

Value

A data.frame/tibble with $m \times n$ rows (where m and n are matrix dimensions) containing:

- All row_data columns (with "row." prefix if name conflicts exist)
- All col_data columns (with "col." prefix if name conflicts exist)
- A "value" column containing the matrix values

Examples

```
# Basic conversion to long format
mat <- matrix(1:12, nrow = 4, ncol = 3)
row_data <- data.frame(
  gene_id = paste0("Gene", 1:4),
  gene_type = c("A", "A", "B", "B")
)
col_data <- data.frame(
  sample_id = paste0("Sample", 1:3),
  condition = c("Control", "Treatment", "Control")
)
tm <- tidymatrix(mat, row_data, col_data)

long <- to_long(tm)
head(long)

# Use in ggplot2 workflow
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  tm |>
    to_long() |>
    ggplot(aes(x = sample_id, y = value, color = condition)) +
    geom_point() +
    facet_wrap(~gene_id)
}

# Statistics added to the metadata carry over to the long format
big5 <- tidymatrix(big5_responses, big5_respondents, big5_items) |>
  activate(columns) |>
  compute_ttest(
    group_col = "gender", control = "Female", treatment = "Male",
    add_to_data = TRUE
  )
long_big5 <- to_long(big5)
head(long_big5[long_big5$p.adj < 0.05, ])
```

transform_matrix

Apply a function to the matrix

Description

Applies a function to the matrix with behavior determined by the active component:

- `activate(matrix)`: `fn` is applied to the entire matrix at once. Use functions like `log`, `exp`, `sqrt`, or anonymous functions like `\(x) x^2`.
- `activate(rows)`: `fn` is applied independently to each row vector and must return a vector of the same length.
- `activate(columns)`: `fn` is applied independently to each column vector and must return a vector of the same length.

Usage

```
transform_matrix(.data, fn, ...)
```

Arguments

<code>.data</code>	A tidymatrix object
<code>fn</code>	A function to apply. When matrix is active, receives the full matrix. When rows or columns are active, receives one row or column vector at a time. Must return values with the same dimensions.
<code>...</code>	Additional arguments passed to <code>fn</code> . With rows active they may refer to columns of <code>col_data</code> ; with columns active, to columns of <code>row_data</code> . Avoid argument names that partially match <code>fn</code> (such as <code>f</code>), as R would match them to <code>fn</code> .

Details

Additional arguments in `...` are passed on to `fn`. With rows or columns active they are evaluated with the metadata of the *other* dimension as a data mask, in the same way as in `dplyr::mutate()`. A row vector has one element per matrix column, so with rows active a column of `col_data` lines up element by element with the vector that `fn` receives (and likewise for columns and `row_data`). This makes it possible to transform values depending on their metadata, e.g. to reverse-score some questionnaire items (see examples). Use `.env$x` to refer to a variable `x` in the calling environment when a metadata column has the same name. With the matrix active, arguments are evaluated normally.

Value

A tidymatrix object with the transformed matrix

Examples

```
mat <- matrix(1:12, nrow = 3, ncol = 4)
tm <- tidymatrix(mat)

# Element-wise: apply log to entire matrix
tm |>
  activate(matrix) |>
  transform_matrix(log)

# Row-wise: rank values within each row
tm |>
  activate(rows) |>
  transform_matrix(rank)

# Column-wise: min-max normalize each column to [0, 1]
tm |>
  activate(columns) |>
  transform_matrix(\(x) (x - min(x)) / (max(x) - min(x)))

# Passing extra arguments: round to 2 decimal places
tm |>
```

```

activate(matrix) |>
transform_matrix(round, digits = 2)

# Arguments can use the metadata of the other dimension. Reverse-score
# the questionnaire items flagged in the column metadata (1 <-> 5):
big5 <- tidymatrix(big5_responses, big5_respondents, big5_items)
big5 |>
  activate(rows) |>
  transform_matrix(\(x, flip) ifelse(flip, 6L - x, x), flip = reversed)

```

ungroup.grouped_tidymatrix

Remove grouping from a grouped tidymatrix

Description

Remove grouping from a grouped tidymatrix

Usage

```

## S3 method for class 'grouped_tidymatrix'
ungroup(x, ...)

```

Arguments

x	A grouped_tidymatrix object
...	Not used

Value

A tidymatrix object (ungrouped)

Examples

```

library(dplyr, warn.conflicts = FALSE)
tm <- tidymatrix(big5_responses, big5_respondents, big5_items)

grouped <- tm |>
  activate(rows) |>
  group_by(country)

ungroup(grouped)

```

Index

- * **datasets**
 - big5, 8
 - big5_countries, 10
- activate, 3
- active, 4
- add_stats, 4
- analysis-management, 5
- anti_join.tidymatrix (joins), 38
- arrange.tidymatrix, 6
- as.data.frame.tidymatrix, 7
- as_tibble.tidymatrix, 7
- big5, 8
- big5_countries, 9, 10
- big5_items (big5), 8
- big5_respondents, 10
- big5_respondents (big5), 8
- big5_responses (big5), 8
- center, 10
- center.tidymatrix, 10, 11
- check_analyses, 12
- clip_values, 13
- compute_across, 13
- compute_anova, 15
- compute_correlation, 17
- compute_hclust, 18
- compute_kmeans, 19
- compute_kruskal, 20
- compute_lm, 21
- compute_lm_simple, 23
- compute_mds, 24
- compute_prcomp, 25
- compute_tsne, 26
- compute_ttest, 27
- compute_umap, 29
- compute_wilcox, 30
- count.tidymatrix, 32
- filter.tidymatrix, 33
- full_join.tidymatrix (joins), 38
- get_analysis, 6, 34
- group_by.tidymatrix, 35
- group_vars.grouped_tidymatrix, 36
- groups.grouped_tidymatrix, 35
- inner_join.tidymatrix (joins), 38
- is_grouped_tidymatrix, 37
- is_tidymatrix, 38
- joins, 38
- left_join, 40
- left_join.tidymatrix (joins), 38
- list_analyses, 41
- log_transform, 41
- matrix-operations, 42
- mutate.tidymatrix, 42
- plot_pheatmap, 43
- print.grouped_tidymatrix, 45
- print.tidymatrix, 46
- pull.tidymatrix, 46
- pull_active, 47
- relocate.tidymatrix, 48
- remove_analysis, 49
- rename.tidymatrix, 49
- right_join.tidymatrix (joins), 38
- scale.tidymatrix, 50
- select.tidymatrix, 51
- semi_join.tidymatrix (joins), 38
- slice.tidymatrix, 52
- slice_head.tidymatrix, 52
- slice_sample.tidymatrix, 53
- slice_tail.tidymatrix
 - (slice_head.tidymatrix), 52

summarise.grouped_tidymatrix
 (summarize.grouped_tidymatrix),
 [54](#)
summarize.grouped_tidymatrix, [54](#)

t_tidymatrix, [55](#)
tally.grouped_tidymatrix, [56](#)
tidymatrix, [57](#)
to_long, [58](#)
transform_matrix, [41](#), [59](#)

ungroup.grouped_tidymatrix, [61](#)