Package 'tidyfinance'

May 12, 2026

Type Package

Title Tidy Finance Helper Functions

Version 0.5.0

Description Helper functions for empirical research in financial economics, addressing a variety of topics covered in Scheuch, Voigt, and Weiss (2023) <doi:10.1201/b23237>. The package is designed to provide shortcuts for issues extensively discussed in the book, facilitating easier application of its concepts. For more information and resources related to the book, visit <https://www.tidy-finance.org/r/index.html>.

License MIT + file LICENSE

URL https://www.tidy-finance.org/r/, https://github.com/tidy-finance/r-tidyfinance

BugReports https://github.com/tidy-finance/r-tidyfinance/issues

Depends R (>= 4.1)

Imports arrow, cli, dplyr (>= 1.1.4), dbplyr (>= 2.5.0), glue, jsonlite, lifecycle, lubridate (>= 1.9.3), purrr (>= 1.0.2), rlang (>= 1.1.3), slider (>= 0.3.1), stats, stringr, tibble, tidyr (>= 1.3.1), DBI (>= 1.2.2), frenchdata (>= 0.2.0), httr2 (>= 1.0.0), RPostgres (>= 1.4.5), sandwich (>= 3.1-0)

Suggests knitr, rmarkdown, furrr (>= 0.3.1), testthat (>= 3.2.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Christoph Scheuch [aut, cre, cph] (ORCID: <https://orcid.org/0009-0004-0423-6819>), Stefan Voigt [aut, cph] (ORCID: <https://orcid.org/0000-0001-5619-3161>),

Patrick Weiss [aut, cph] (ORCID:
<https://orcid.org/0000-0002-9282-5872>),
Maximilian Mücke [ctb] (ORCID: <https://orcid.org/0009-0000-9432-9795>)

Maintainer Christoph Scheuch <christoph@tidy-intelligence.com>

Repository CRAN

Date/Publication 2026-05-12 15:00:02 UTC

Contents

`add_lagged_columns` *Add Lagged Columns via Join*

Description

Appends lagged versions of specified columns to a data frame using a join-based approach.

Usage

```
add_lagged_columns(
  data,
  cols,
  lag,
  max_lag = lag,
  by = NULL,
  drop_na = FALSE,
  ff_adjustment = FALSE,
  data_options = NULL
)
```

Arguments

`data`	A data frame containing the variables to lag.
`cols`	A character vector specifying the names of the columns to be lagged. Each column produces a new column suffixed with `_lag`.
`lag`	An integer or a `lubridate::periods()` object, e.g., `months(1)`, specifying the minimum lag (inclusive) to apply.
`max_lag`	An integer or a `lubridate::periods()` object specifying the maximum lag (inclusive) to apply. Defaults to `lag` (exact lag).

by	An optional character vector specifying grouping columns (e.g., a stock identifier). Lagged values are matched within groups. Defaults to NULL.
drop_na	A logical value. If TRUE, NA values in the source columns are excluded before matching, so the lookup skips over missing observations. Applied independently per column. Defaults to FALSE.
ff_adjustment	A logical value. If TRUE, only the last observation per year (within each group defined by by) is retained as a source for lagged values, following Fama-French conventions for annual accounting data. Defaults to FALSE.
data_options	A list of class tidyfinance_data_options (created via data_options()) specifying column name mappings. The date element is used to specify the date column. Uses data_options() default if NULL: "date" = "date".

Details

When lag == max_lag (the default), an equi-join is used: source dates are shifted forward by lag and matched exactly. When lag < max_lag, an inequality join is used: for each row, the most recent source value within the window [date - max_lag, date - lag] is selected.

The combination of by and date columns must be unique in data. If by is NULL, dates alone must be unique.

Value

A data frame with the same rows as data and new columns appended, each suffixed with _lag. Unmatched rows receive NA in the lagged columns.

See Also

Other rolling and lagging functions: compute_rolling_value(), join_lagged_values()

Examples

```
set.seed(42)
data <- tibble::tibble(
  permno = rep(1:2, each = 10),
  date = rep(
    seq.Date(as.Date("2023-01-01"), by = "month", length.out = 10),
    2
  ),
  size = runif(20, 100, 200),
  bm = runif(20, 0.5, 1.5)
)

# Exact lag: each row gets the value from exactly 2 months earlier
add_lagged_columns(
  data,
  cols = c("size", "bm"),
  lag = months(2),
  by = "permno"
)
```

```
# Window lag: each row gets the most recent value from 2 to 4 months earlier
add_lagged_columns(
  data,
  cols = "size",
  lag = months(2),
  max_lag = months(4),
  by = "permno"
)
```

`assign_portfolio` *Assign Portfolios Based on Sorting Variable*

Description

[Experimental]

Assigns data points to portfolios based on a specified sorting variable and the selected function to compute breakpoints. Users can specify a function to compute breakpoints. The function must take `data` and `sorting_variable` as the first two arguments. Additional arguments are passed with a named list `breakpoint_options()`. The function needs to return an ascending vector of breakpoints. By default, breakpoints are computed with `compute_breakpoints()`. The default column names can be modified using `data_options()`.

Usage

```
assign_portfolio(
  data,
  sorting_variable,
  breakpoint_options = NULL,
  breakpoint_function = compute_breakpoints,
  data_options = NULL
)
```

Arguments

`data`	A data frame containing the dataset for portfolio assignment.
`sorting_variable`	A string specifying the column name in `data` to be used for sorting and determining portfolio assignments based on the breakpoints.
`breakpoint_options`	An optional named list of arguments passed to `breakpoint_function`.
`breakpoint_function`	A function to compute breakpoints. The default is set to `compute_breakpoints()`.
`data_options`	A list of class `tidyfinance_data_options` (created via `data_options()`) specifying column name mappings. Passed through to `breakpoint_function`. When using the default `compute_breakpoints()`, the `exchange` element is used to specify the exchange column, and `mktcap_lag` is used to specify the market capitalization column. Uses `data_options()` default if `NULL`: `"exchange" = "exchange"` and `"mktcap_lag" = "mktcap_lag"`.

Value

A vector of integer portfolio assignments for each row in the input `data`.

See Also

Other portfolio functions: `breakpoint_options()`, `compute_breakpoints()`, `compute_long_short_returns()`, `compute_portfolio_returns()`, `data_options()`, `filter_options()`, `filter_sorting_data()`, `implement_portfolio_sort()`, `portfolio_sort_options()`

Examples

```
set.seed(42)
data <- data.frame(
  id = 1:100,
  exchange = sample(c("NYSE", "NASDAQ"), 100, replace = TRUE),
  market_cap = 1:100
)

assign_portfolio(data, "market_cap", breakpoint_options(n_portfolios = 5))

assign_portfolio(
  data,
  "market_cap",
  breakpoint_options(
    percentiles = c(0.2, 0.4, 0.6, 0.8),
    breakpoints_exchanges = c("NYSE")
  )
)
```

`breakpoint_options` *Create Breakpoint Options for Portfolio Sorting*

Description

Generates a structured list of options for defining breakpoints in portfolio sorting. It includes parameters for the number of portfolios, percentile thresholds, exchange-specific breakpoints, and smooth bunching, along with additional optional parameters.

Usage

```
breakpoint_options(
  n_portfolios = NULL,
  percentiles = NULL,
  breakpoints_exchanges = NULL,
  smooth_bunching = FALSE,
  breakpoints_min_size_threshold = NULL,
  ...
)
```

Arguments

n_portfolios	Integer, optional. The number of portfolios to create. Must be a positive integer. If not provided, defaults to NULL.
percentiles	Numeric vector, optional. A vector of percentile thresholds for defining breakpoints. Each value must be between 0 and 1. If not provided, defaults to NULL.
breakpoints_exchanges	Character vector, optional. A non-empty vector specifying the exchange from which to compute the breakpoints. If not provided, defaults to NULL.
smooth_bunching	Logical, optional. Indicates whether smooth bunching should be applied. Defaults to FALSE.
breakpoints_min_size_threshold	Numeric, optional. When set to a value between 0 and 1, stocks with market capitalization below this quantile are excluded from breakpoint computation. The quantile is computed among breakpoints_exchanges stocks if specified, otherwise among all stocks. Requires a market capitalization column in the data (see data_options()). Defaults to NULL (no size filtering).
...	Additional optional arguments. These will be captured in the resulting structure as a list.

Value

A list of class "tidyfinance_breakpoint_options" containing the provided breakpoint options, including any additional arguments passed via ....

See Also

Other portfolio functions: assign_portfolio(), compute_breakpoints(), compute_long_short_returns(), compute_portfolio_returns(), data_options(), filter_options(), filter_sorting_data(), implement_portfolio_sort(), portfolio_sort_options()

Examples

```
breakpoint_options(
  n_portfolios = 5,
  percentiles = c(0.2, 0.4, 0.6, 0.8),
  breakpoints_exchanges = "NYSE",
  smooth_bunching = TRUE,
  custom_threshold = 0.5,
  another_option = "example"
)
```

check_supported_type *Check if a Dataset Type is Supported*

Description

Checks if a given dataset type is supported by verifying against a list of all supported dataset types from different domains. If the specified type is not supported, execution stops and an error message listing all supported types is returned.

Usage

```
check_supported_type(type)
```

Arguments

type	A character string specifying the dataset type to check.

Value

Does not return a value; instead, it either passes silently if the type is supported or stops execution with an error message if the type is unsupported.

compute_breakpoints *Compute Breakpoints Based on Sorting Variable*

Description

[Experimental]

Computes breakpoints based on a specified sorting. It can optionally filter the data by exchanges or lagged size quantiles before computing the breakpoints. The function requires either the number of portfolios to be created or specific percentiles for the breakpoints, but not both. The function also optionally handles cases where the sorting variable clusters on the edges, by assigning all extreme values to the edges and attempting to compute equally populated breakpoints with the remaining values.

Usage

```
compute_breakpoints(
  data,
  sorting_variable,
  breakpoint_options,
  data_options = NULL
)
```

Arguments

`data` A data frame containing the dataset for breakpoint computation.

`sorting_variable`
A character string specifying the column name in `data` to be used for determining breakpoints.

`breakpoint_options`
A named list of `breakpoint_options()` for the breakpoints. The arguments include

- `n_portfolios` An optional integer specifying the number of equally sized portfolios to create. This parameter is mutually exclusive with `percentiles`.
- `percentiles` An optional numeric vector specifying the percentiles for determining the breakpoints of the portfolios. This parameter is mutually exclusive with `n_portfolios`.
- `breakpoints_exchanges` An optional character vector specifying exchange names to filter the data before computing breakpoints. Exchanges must be stored in a column given by `data_options` (defaults to `exchange`). If `NULL`, no filtering is applied.
- `smooth_bunching` An optional logical parameter specifying if to attempt smoothing non-extreme portfolios if the sorting variable bunches on the extremes (`TRUE`), or not (`FALSE`, the default). In some cases, smoothing will not result in equal-sized portfolios off the edges due to multiple clusters. If sufficiently large bunching is detected, `percentiles` is ignored and equally-spaced portfolios are returned for these cases with a warning.
- `breakpoints_min_size_threshold` An optional numeric value between 0 and 1 (exclusive). When set, stocks with market capitalization below this quantile are excluded from breakpoint computation. The quantile is computed among `breakpoints_exchanges` stocks if specified, otherwise among all stocks. Requires a market capitalization column in the data (column name determined by `data_options`).

`data_options` A list of class `tidyfinance_data_options` (created via `data_options()`) specifying column name mappings. The `exchange` element is used to specify the exchange column, and `mktcap_lag` is used to specify the market capitalization. Uses `data_options()` default if `NULL`: `"exchange" = "exchange"` and `"mktcap_lag" = "mktcap_lag"`.

Value

A numeric vector of breakpoints of the desired length.

Note

This function will stop and throw an error if both `n_portfolios` and `percentiles` are provided or missing simultaneously.

See Also

Other portfolio functions: assign_portfolio(), breakpoint_options(), compute_long_short_returns(), compute_portfolio_returns(), data_options(), filter_options(), filter_sorting_data(), implement_portfolio_sort(), portfolio_sort_options()

Examples

```
set.seed(42)
data <- data.frame(
  id = 1:100,
  exchange = sample(c("NYSE", "NASDAQ"), 100, replace = TRUE),
  market_cap = 1:100
)

compute_breakpoints(data, "market_cap", breakpoint_options(n_portfolios = 5))
compute_breakpoints(
  data,
  "market_cap",
  breakpoint_options(
    percentiles = c(0.2, 0.4, 0.6, 0.8),
    breakpoints_exchanges = c("NYSE")
  )
 )
```

---

compute_long_short_returns

Compute Long-Short Returns

Description

Calculates long-short returns based on the returns of portfolios. The long-short return is computed as the difference between the returns of the "top" and "bottom" portfolios. The direction of the calculation can be adjusted based on whether the return from the "bottom" portfolio is subtracted from or added to the return from the "top" portfolio.

Usage

```
compute_long_short_returns(
  data,
  direction = "top_minus_bottom",
  data_options = NULL
)
```

Arguments

data	A data frame containing portfolio returns. The data frame must include columns for the portfolio identifier, date, and return measurements (as specified in data_options).

direction
: A character string specifying the direction of the long-short return calculation. It can be either "top_minus_bottom" or "bottom_minus_top". Default is "top_minus_bottom". If set to "bottom_minus_top", the return will be computed as (bottom - top).

data_options
: A list of class tidyfinance_data_options (created via data_options()) specifying column name mappings. The date element is used to specify the date column, the ret_excess element is used to specify the excess return column, and portfolio is used to specify the assigned portfolio. Uses data_options() default if NULL: "date" = "date", "ret_excess" = "ret_excess", and "portfolio" = "portfolio".

Value

A data frame with columns for date, return measurement types (from the "ret_measure" column), and the computed long-short returns. The data frame is arranged by date and pivoted to have return measurement types as columns with their corresponding long-short returns.

See Also

Other portfolio functions: assign_portfolio(), breakpoint_options(), compute_breakpoints(), compute_portfolio_returns(), data_options(), filter_options(), filter_sorting_data(), implement_portfolio_sort(), portfolio_sort_options()

Examples

```
set.seed(42)
data <- data.frame(
  permno = 1:100,
  date = rep(
    seq.Date(from = as.Date("2020-01-01"), by = "month", length.out = 100),
    each = 10
  ),
  mktcap_lag = runif(100, 100, 1000),
  ret_excess = rnorm(100),
  size = runif(100, 50, 150)
)

portfolio_returns <- compute_portfolio_returns(
  data, "size", "univariate",
  breakpoint_options_main = breakpoint_options(n_portfolios = 5)
)

compute_long_short_returns(portfolio_returns)
```

compute_portfolio_returns *Compute Portfolio Returns*

Description

Computes individual portfolio returns based on specified sorting variables and sorting methods. The portfolios can be rebalanced every period or on an annual frequency by specifying a rebalancing month, which is only applicable at a monthly return frequency. The function supports univariate and bivariate sorts, with the latter supporting dependent and independent sorting methods.

Usage

```
compute_portfolio_returns(
  data,
  sorting_variables,
  sorting_method,
  rebalancing_month = NULL,
  breakpoint_options_main,
  breakpoint_options_secondary = NULL,
  breakpoint_function_main = compute_breakpoints,
  breakpoint_function_secondary = compute_breakpoints,
  min_portfolio_size = 1L,
  cap_weight = 0.8,
  data_options = NULL,
  quiet = FALSE
)
```

Arguments

data	A data frame containing the dataset for portfolio assignment and return computation. The panel data must include individual stock identifiers and the time point. It must contain columns for the sorting variables and excess returns. Additionally, lagged market capitalization is required for value-weighted returns (see parameter data_options).
sorting_variables	A character vector specifying the column names in data to be used for sorting and determining portfolio assignments. For univariate sorts, provide a single variable. For bivariate sorts, provide two variables, where the first string refers to the main variable and the second string refers to the secondary ("control") variable.
sorting_method	A string specifying the sorting method to be used. Possible values are: • "univariate": For a single sorting variable. • "bivariate-dependent": For two sorting variables, where the main sort depends on the secondary variable. • "bivariate-independent": For two independent sorting variables.

For bivariate sorts, the portfolio returns are averaged over the controlling sorting variable (i.e., the second sorting variable), and only portfolio returns for the main sorting variable (given as the first element of `sorting_variables`) are returned.

`rebalancing_month`
: An integer between 1 and 12 specifying the month in which to form portfolios that are held constant for one year. For example, setting it to 7 creates portfolios in July that are held constant until June of the following year. The default `NULL` corresponds to periodic rebalancing.

`breakpoint_options_main`
: A named list of `breakpoint_options()` passed to `breakpoint_function_main` for the main sorting variable.

`breakpoint_options_secondary`
: An optional named list of `breakpoint_options()` passed to `breakpoint_function_secondary`.

`breakpoint_function_main`
: A function to compute the breakpoints for the main sorting variable. The default is set to `compute_breakpoints()`.

`breakpoint_function_secondary`
: A function to compute the breakpoints for the secondary sorting variable. The default is set to `compute_breakpoints()`.

`min_portfolio_size`
: An integer specifying the minimum number of firms required in the reported portfolio cross-section on a given date (default `1L`, i.e. at least one observation per reported portfolio). For both univariate and bivariate sorts the threshold is applied to the firm count per `(portfolio, date)` of the reported cross-section: for univariate sorts that is firms per portfolio-date; for bivariate sorts that is firms per main-portfolio-date summed across the secondary buckets. Cross-sections below the threshold have their returns set to `NA`. A typical value is `5L` (the Fama-French convention). Set to `0L` to deactivate the check entirely.

`cap_weight`
: A numeric value between 0 and 1 specifying the percentile at which market capitalization is capped per date when computing capped value-weighted excess returns (i.e., `ret_excess_vw_capped`). Defaults to `0.8`.

`data_options`
: A list of class `tidyfinance_data_options` (created via `data_options()`) specifying column name mappings. The `id` element is used to specify the entity (i.e., firm), `date` is used to specify the date column, `ret_excess` is used to specify the excess return column, and `mktcap_lag` is used to specify the market capitalization. Uses `data_options()` default if `NULL`: `"id" = "permno"`, `"date" = "date"`, `"ret_excess" = "ret_excess"`, and `"mktcap_lag" = "mktcap_lag"`.

`quiet`
: A logical value indicating whether to suppress informational messages about missing values in the output panel (default is `FALSE`).

Details

The function checks for consistency in the provided arguments. For univariate sorts, a single sorting variable and a corresponding number of portfolios must be provided. For bivariate sorts, two sorting variables and two corresponding numbers of portfolios (or percentiles) are required. The sorting method determines how portfolios are assigned and how returns are computed. The function handles missing and extreme values appropriately based on the specified sorting method and rebalancing frequency.

Value

A data frame with computed portfolio returns as a complete panel (all portfolio-date combinations), containing the following columns:

- `portfolio`: The portfolio identifier.
- `date`: The date of the portfolio return.
- `ret_excess_vw`: The value-weighted excess return of the portfolio (only computed if `data` contains a lagged market capitalization column specified by `data_options()`, which defaults to `mktcap_lag`). `NA` if insufficient observations for that portfolio-date.
- `ret_excess_ew`: The equal-weighted excess return of the portfolio. `NA` if insufficient observations for that portfolio-date.
- `ret_excess_vw_capped`: The capped value-weighted excess return of the portfolio (only computed if `data` contains a lagged market capitalization column specified by `data_options()`, which defaults to `mktcap_lag`). Weights are computed using market capitalization capped at the `cap_weight` percentile per date. `NA` if insufficient observations for that portfolio-date.

Note

Ensure that `data` contains all required columns: the specified sorting variables and excess returns (see options and defaults set in `data_options`). The function will stop and throw an error if any required columns are missing.

See Also

Other portfolio functions: `assign_portfolio()`, `breakpoint_options()`, `compute_breakpoints()`, `compute_long_short_returns()`, `data_options()`, `filter_options()`, `filter_sorting_data()`, `implement_portfolio_sort()`, `portfolio_sort_options()`

Examples

```
set.seed(42)
# Univariate sorting with periodic rebalancing
data <- data.frame(
  permno = 1:500,
  date = rep(
    seq.Date(
      from = as.Date("2020-01-01"),
      by = "month",
      length.out = 100
    ),
    each = 10
  ),
  mktcap_lag = runif(500, 100, 1000),
  ret_excess = rnorm(500),
  size = runif(500, 50, 150)
)

compute_portfolio_returns(
  data, "size", "univariate",
```

```
  breakpoint_options_main = breakpoint_options(n_portfolios = 5)
)

# Bivariate dependent sorting with annual rebalancing
compute_portfolio_returns(
  data, c("size", "mktcap_lag"), "bivariate-dependent", 7,
  breakpoint_options_main = breakpoint_options(n_portfolios = 5),
  breakpoint_options_secondary = breakpoint_options(n_portfolios = 3),
)
```

`compute_rolling_value` *Compute a Rolling Value by Period*

Description

[Experimental]

Applies an arbitrary summary function over rolling time-period windows. Each window spans `periods` units of `period` (e.g., 12 months). Before calling `.f`, rows with any missing values are dropped from the window; if fewer than `min_obs` rows remain, the result is `NA_real_` instead.

Usage

```
compute_rolling_value(
  data,
  .f,
  period = "month",
  periods = 12,
  min_obs = periods,
  data_options = NULL
)
```

Arguments

`data`	A data frame with a date column of class `Date`, named according to `data_options$date` (default `"date"`).
`.f`	A function applied to each window. Receives a data-frame slice (complete cases only) and must return a single scalar value.
`period`	A string specifying the period for rolling windows (e.g., `"month"`, `"quarter"`, `"year"`).
`periods`	Number of periods to include in the rolling window.
`min_obs`	Minimum number of non-missing rows required per window. Defaults to `periods`.
`data_options`	A list of class `tidyfinance_data_options` (created via `data_options()`) specifying column name mappings. The `date` element is used to specify the date column. Uses `data_options()` default if `NULL`: `"date" = "date"`.

Value

A numeric vector aligned with the rows of `data`.

See Also

Other rolling and lagging functions: `add_lagged_columns()`, `join_lagged_values()`

Examples

```
library(dplyr)

# Rolling standard deviation
set.seed(42)
df <- tibble(
  date = seq.Date(
    from = as.Date("2020-01-01"),
    by = "month",
    length.out = 24
  ),
  value = rnorm(24)
)

df |>
  mutate(
    rolling_sd = compute_rolling_value(
      pick(everything()),
      .f = ~ sd(.x$value, na.rm = TRUE),
      period = "month",
      periods = 4,
      min_obs = 2
    )
  )

# Rolling last residual from a regression
set.seed(42)
df_reg <- tibble(
  date = seq.Date(
    from = as.Date("2020-01-01"),
    by = "month",
    length.out = 60
  ),
  ret_excess = rnorm(60, 0, 0.05),
  mkt_excess = rnorm(60, 0, 0.04),
  smb = rnorm(60, 0, 0.03),
  hml = rnorm(60, 0, 0.03)
)

df_reg |>
  mutate(
    residual = compute_rolling_value(
      pick(everything()),
      .f = \(x) {
```

```
        last(lm(ret_excess ~ mkt_excess + smb + hml, data = x)$residuals)
      },
      period = "month",
      periods = 24,
      min_obs = 12
    )
  )

# Rolling cumulative-return-to-SD ratio
set.seed(42)
df_resid <- tibble(
  date = seq.Date(
    from = as.Date("2020-01-01"),
    by = "month",
    length.out = 24
  ),
  int_roll_residual = rnorm(24, 0, 0.02)
)

df_resid |>
  mutate(
    return_to_sd = compute_rolling_value(
      pick(everything()),
      .f = ~ (prod(1 + .x$int_roll_residual) - 1) / sd(.x$int_roll_residual),
      period = "month",
      periods = 12,
      min_obs = 12
    )
  )
```

`create_summary_statistics` *Create Summary Statistics for Specified Variables*

Description

Computes a set of summary statistics for numeric and integer variables in a data frame. It allows users to select specific variables for summarization and can calculate statistics for the whole dataset or within groups specified by the by argument. Additional detail levels for quantiles can be included.

Usage

```
create_summary_statistics(
  data,
  ...,
  by = NULL,
  detail = FALSE,
  drop_na = FALSE
)
```

Arguments

data	A data frame containing the variables to be summarized.
...	Comma-separated list of unquoted variable names in the data frame to summarize. These variables must be either numeric, integer, or logical.
by	An optional unquoted variable name to group the data before summarizing. If NULL (the default), summary statistics are computed across all observations.
detail	A logical flag indicating whether to compute detailed summary statistics, including additional quantiles. Defaults to FALSE, which computes basic statistics (n, mean, sd, min, median, max). When TRUE, additional quantiles (1%, 5%, 10%, 25%, 75%, 90%, 95%, 99%) are computed.
drop_na	A logical flag indicating whether to drop missing values for each variable (default is FALSE).

Details

The function first checks that all specified variables are of type numeric, integer, or logical. If any variables do not meet this criterion, the function stops and returns an error message indicating the non-conforming variables.

The basic set of summary statistics includes the count of non-NA values (n), mean, standard deviation (sd), minimum (min), median (q50), and maximum (max). If detail is TRUE, the function also computes the 1st, 5th, 10th, 25th, 75th, 90th, 95th, and 99th percentiles.

Summary statistics are computed for each variable specified in .... If a by variable is provided, statistics are computed within each level of the by variable.

Value

A tibble with summary statistics for each selected variable. If by is specified, the output includes the grouping variable as well. Each row represents a variable (and a group if by is used), and each column contains the computed statistics.

See Also

Other utility functions: get_available_huggingface_files(), list_supported_indexes(), list_supported_types(), list_supported_types_ff(), list_supported_types_ff_legacy(), list_supported_types_macro_predictors(), list_supported_types_other(), list_supported_types_wrds(), list_tidy_finance_chapters(), open_tidy_finance_website(), trim(), validate_dates(), winsorize()

Examples

```
data <- data.frame(
  ret = c(0.01, -0.02, 0.03, NA, 0.005),
  size = c(100, 200, 150, 300, 250),
  group = c("A", "A", "B", "B", "A")
)

# Basic summary across all observations
create_summary_statistics(data, ret, size)
```

```
# Grouped summary
create_summary_statistics(data, ret, size, by = group)

# Detailed quantiles
create_summary_statistics(data, ret, detail = TRUE)
```

`create_wrds_dummy_database` *Create WRDS Dummy Database*

Description

Downloads the WRDS dummy database from the respective Tidy Finance GitHub repository and saves it to the specified path. If the file already exists, the user is prompted before it is replaced.

Usage

```
create_wrds_dummy_database(
  path,
  url = paste0("https://github.com/tidy-finance/website/tree/main/blog/",
    "tidy-finance-dummy-data/data/tidy_finance.sqlite")
)
```

Arguments

`path`	The file path where the SQLite database should be saved.
`url`	The URL where the SQLite database is stored.

Value

Invisible `NULL`. Side effect: downloads a file to the specified path.

See Also

Other WRDS functions: `disconnect_connection()`, `download_data_wrds()`, `download_data_wrds_ccm_links()`, `download_data_wrds_compustat()`, `download_data_wrds_crsp()`, `download_data_wrds_fisd()`, `download_data_wrds_trace_enhanced()`, `get_wrds_connection()`, `set_wrds_credentials()`

Examples

```
path <- paste0(tempdir(), "/tidy_finance_r.sqlite")
create_wrds_dummy_database(path)
```

`data_options` *Create Data Options*

Description

Creates a list of data options of class `tidyfinance_data_options` used for TidyFinance-related functions. These options map the specific data variables to the TidyFinance naming conventions, allowing functions to flexibly work with different datasets by specifying the relevant column names. Additional options can be passed through `...`.

Usage

```
data_options(
  id = "permno",
  date = "date",
  exchange = "exchange",
  mktcap_lag = "mktcap_lag",
  ret_excess = "ret_excess",
  portfolio = "portfolio",
  siccd = "siccd",
  price = "prc_adj",
  listing_age = "listing_age",
  be = "be",
  earnings = "ib",
  ...
)
```

Arguments

`id`	A character string representing the entity variable (defaults to `"permno"`).
`date`	A character string representing the date variable (defaults to `"date"`).
`exchange`	A character string representing the exchange variable (defaults to `"exchange"`).
`mktcap_lag`	A character string representing the market capitalization lag variable (defaults to `"mktcap_lag"`).
`ret_excess`	A character string representing the excess return variable (defaults to `"ret_excess"`).
`portfolio`	A character string representing the portfolio variable (defaults to `"portfolio"`).
`siccd`	A character string representing the Standard Industrial Classification code variable (defaults to `"siccd"`).
`price`	A character string representing the (adjusted) price variable (defaults to `"prc_adj"`).
`listing_age`	A character string representing the listing age variable (defaults to `"listing_age"`).
`be`	A character string representing the book equity variable (defaults to `"be"`).
`earnings`	A character string representing the earnings variable (defaults to `"ib"`, the Compustat income before extraordinary items column).
`...`	Additional arguments to be included in the data options list.

Value

A list of class `tidyfinance_data_options` containing the specified data options.

See Also

Other portfolio functions: `assign_portfolio()`, `breakpoint_options()`, `compute_breakpoints()`, `compute_long_short_returns()`, `compute_portfolio_returns()`, `filter_options()`, `filter_sorting_data()`, `implement_portfolio_sort()`, `portfolio_sort_options()`

Examples

```
data_options(
  id = "permno",
  date = "date",
  exchange = "exchange"
)
```

---

disconnect_connection *Disconnect Database Connection*

Description

Safely disconnects an established database connection using the DBI package.

Usage

```
disconnect_connection(con)
```

Arguments

con	A database connection object created by DBI::dbConnect or any similar function that establishes a connection to a database.

Value

`TRUE`, invisibly. Throws an error if the disconnection fails.

See Also

Other WRDS functions: `create_wrds_dummy_database()`, `download_data_wrds()`, `download_data_wrds_ccm_links()`, `download_data_wrds_compustat()`, `download_data_wrds_crsp()`, `download_data_wrds_fisd()`, `download_data_wrds_trace_enhanced()`, `get_wrds_connection()`, `set_wrds_credentials()`

Examples

```
## Not run:
con <- get_wrds_connection()
disconnect_connection(con)

## End(Not run)
```

`download_data` *Download and Process Data Based on Domain and Dataset*

Description

Downloads and processes data based on the specified domain (e.g., Fama-French factors, Global Q factors, or macro predictors), dataset, and date range. This function checks if the specified domain is supported and then delegates to the appropriate function for downloading and processing the data.

Usage

```
download_data(
  domain = NULL,
  dataset = NULL,
  start_date = NULL,
  end_date = NULL,
  type = deprecated(),
  ...
)
```

Arguments

`domain`	The domain of the dataset to download (e.g., "famafrench", "globalq", "macro_predictors", "wrds", "constituents", "fred", "stock_prices", "osap", "tidyfinance").
`dataset`	Optional. The specific dataset to download within the domain.
`start_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, the full dataset or a subset is returned, depending on the dataset type.
`end_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, the full dataset or a subset is returned, depending on the dataset type.
`type`	**[Deprecated]** Use `domain` and `dataset` instead.
`...`	Additional arguments passed to specific download functions depending on the `domain`. For instance, if `domain` is `"constituents"`, arguments are passed to `download_data_constituents()`. If `domain` is `"tidyfinance"` and `dataset` is `"factor_library"`, arguments are used to filter the portfolio grid (e.g., `sorting_variable`, `rebalancing`, `fill_all`); see `download_data_huggingface()` for details.

Value

A tibble with processed data, including dates and the relevant financial metrics, filtered by the specified date range.

See Also

Other download functions: download_data_constituents(), download_data_factors_ff(), download_data_factors_q(), download_data_fred(), download_data_huggingface(), download_data_macro_pred
download_data_osap(), download_data_risk_free(), download_data_stock_prices()

Examples

```
download_data(
  "famafrench",
  "Fama/French 5 Factors (2x3) [Daily]",
  "2000-01-01",
  "2020-12-31"
)
download_data("macro_predictors", "monthly", "2000-01-01", "2020-12-31")
download_data("constituents", index = "DAX")
download_data("fred", series = c("GDP", "CPIAUCNS"))
download_data("stock_prices", symbols = c("AAPL", "MSFT"))
download_data(
  "tidyfinance",
  "risk_free",
  "2020-01-01",
  "2020-12-31"
)
download_data(
  "tidyfinance",
  "high_frequency_sp500",
  "2007-07-26",
  "2007-07-27"
)
download_data(
  "tidyfinance",
  "factor_library",
  sorting_variable = "52w",
  rebalancing = "annual"
)
```

---

download_data_constituents

Download Constituent Data

Description

Downloads and processes the constituent data for a specified financial index. The data is fetched from a remote CSV file, filtered, and cleaned to provide relevant information about constituents.

Usage

```
download_data_constituents(index)
```

Arguments

index	A character string specifying the name of the financial index for which to download constituent data. The index must be one of the supported indexes listed by list_supported_indexes().

Details

The function retrieves the URL of the CSV file for the specified index from ETF sites, then sends an HTTP GET request to download the CSV file, and processes the CSV file to extract equity constituents.

The approach is inspired by `tidyquant::tq_index()`, which uses a different wrapper around other ETFs.

Value

A tibble with five columns:

symbol The ticker symbol of the equity constituent.

name The name of the equity constituent.

location The location where the company is based.

exchange The exchange where the equity is traded.

currency The currency in which the equity is traded, derived from the exchange.

The tibble is filtered to exclude non-equity entries, blacklisted symbols, empty names, and any entries containing the index name or "CASH".

See Also

Other download functions: download_data(), download_data_factors_ff(), download_data_factors_q(), download_data_fred(), download_data_huggingface(), download_data_macro_predictors(), download_data_osap(), download_data_risk_free(), download_data_stock_prices()

Examples

```
download_data_constituents("DAX")
```

`download_data_factors_ff` *Download and Process Fama-French Factor Data*

Description

Downloads and processes Fama-French factor data based on the specified dataset name and date range. The function requires the `frenchdata` package to download the data. It processes the raw data into a structured format, including date conversion, scaling factor values, and filtering by the specified date range.

Usage

```
download_data_factors_ff(
  dataset = NULL,
  start_date = NULL,
  end_date = NULL,
  type = deprecated()
)
```

Arguments

`dataset`	The name of the Fama-French dataset to download (e.g., "Fama/French 3 Factors").
`start_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, the full dataset is returned.
`end_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, the full dataset is returned.
`type`	**[Deprecated]** Use `dataset` instead.

Details

If there are multiple tables in the raw Fama-French data (e.g., value-weighted and equal-weighted returns), then the function only returns the first table because these are the most popular. Please use the `frenchdata` package directly if you need less commonly used tables.

Value

A tibble with processed factor data, including the date, risk-free rate, market excess return, and other factors, filtered by the specified date range.

References

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1), 3-56. doi:10.1016/0304405X(93)900235

Fama, E. F., & French, K. R. (2015). A five-factor asset pricing model. *Journal of Financial Economics*, 116(1), 1-22. doi:10.1016/j.jfineco.2014.10.010

Carhart, M. M. (1997). On persistence in mutual fund performance. *Journal of Finance*, 52(1), 57-82. doi:10.1111/j.15406261.1997.tb03808.x

See Also

Other download functions: `download_data()`, `download_data_constituents()`, `download_data_factors_q()`, `download_data_fred()`, `download_data_huggingface()`, `download_data_macro_predictors()`, `download_data_osap()`, `download_data_risk_free()`, `download_data_stock_prices()`

Examples

```
download_data_factors_ff(
  "Fama/French 3 Factors", "2000-01-01", "2020-12-31"
)
download_data_factors_ff(
  "10 Industry Portfolios", "2000-01-01", "2020-12-31"
)
```

`download_data_factors_q`

Download and Process Global Q Factor Data

Description

Downloads and processes Global Q factor data based on the specified dataset, date range, and source URL. The processing includes date conversion, renaming variables to a standardized format, scaling factor values, and filtering by the specified date range.

Usage

```
download_data_factors_q(
  dataset = NULL,
  start_date = NULL,
  end_date = NULL,
  type = deprecated(),
  url = "https://global-q.org/uploads/1/2/2/6/122679606/"
)
```

Arguments

`dataset`	The name of the dataset to download (e.g., "q5_factors_daily_2023.csv", "q5_factors_monthly_2023.csv"
`start_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, the full dataset is returned.
`end_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, the full dataset is returned.
`type`	**[Deprecated]** Use `dataset` instead.
`url`	The base URL from which to download the dataset files.

Value

A tibble with processed factor data, including the date, risk-free rate, market excess return, and other factors, filtered by the specified date range.

References

Hou, K., Xue, C., & Zhang, L. (2015). Digesting anomalies: An investment approach. *Review of Financial Studies*, 28(3), 650-705. doi:10.1093/rfs/hhu068

Hou, K., Mo, H., Xue, C., & Zhang, L. (2019). Which factors? *Review of Finance*, 23(1), 1-35. doi:10.1093/rof/rfy032

See Also

Other download functions: `download_data()`, `download_data_constituents()`, `download_data_factors_ff()`, `download_data_fred()`, `download_data_huggingface()`, `download_data_macro_predictors()`, `download_data_osap()`, `download_data_risk_free()`, `download_data_stock_prices()`

Examples

```
download_data_factors_q("q5_factors_daily_2024", "2020-01-01", "2020-12-31")
download_data_factors_q("q5_factors_annual_2024")
```

`download_data_fred` *Download and Process Data from FRED*

Description

Downloads a specified data series from the Federal Reserve Economic Data (FRED) website, processes the data, and returns it as a tibble.

Usage

```
download_data_fred(series, start_date = NULL, end_date = NULL)
```

Arguments

`series`	A character vector specifying the FRED series ID(s) to download.
`start_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, the full dataset is returned.
`end_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, the full dataset is returned.

Details

Constructs the URL based on the provided FRED series ID, performs an HTTP GET request to download the data in CSV format, and processes it to a tidy tibble format. The resulting tibble includes the date, value, and the series ID.

This approach is inspired by `quantmod::getSymbolsFRED()` which uses a different wrapper around the same FRED download data site. If you want to systematically download FRED data via API, please consider using the `fredr` package.

Value

A tibble containing the processed data with three columns:

date The date corresponding to the data point.

value The value of the data series at that date.

series The FRED series ID corresponding to the data.

See Also

Other download functions: `download_data()`, `download_data_constituents()`, `download_data_factors_ff()`, `download_data_factors_q()`, `download_data_huggingface()`, `download_data_macro_predictors()`, `download_data_osap()`, `download_data_risk_free()`, `download_data_stock_prices()`

Examples

```
download_data_fred("CPIAUCNS")
download_data_fred(c("GDP", "CPIAUCNS"), "2010-01-01", "2010-12-31")
```

---

download_data_huggingface

Download data from a Hugging Face dataset

Description

Downloads data from a supported Hugging Face dataset. For `"high_frequency_sp500"`, parquet files are filtered by date range and row-bound. For `"factor_library"`, portfolio characteristics are selected via `filter_factor_library_grid()` and the matching return data is downloaded.

Usage

```
download_data_huggingface(
  dataset = NULL,
  start_date = "2007-06-27",
  end_date = "2007-07-27",
  type = deprecated(),
  ...
)
```

Arguments

dataset	Character(1). The dataset to download. Supported values are "high_frequency_sp500" and "factor_library".
start_date	Date or character. Start date (inclusive) in "YYYY-MM-DD" format. Only used for "high_frequency_sp500".
end_date	Date or character. End date (inclusive) in "YYYY-MM-DD" format. Only used for "high_frequency_sp500".
type	**[Deprecated]** Use dataset instead.
...	For dataset = "factor_library": named arguments used to filter the portfolio grid. Each argument takes the form column = value, where value may be a vector to match multiple levels. Optionally pass fill_all = TRUE to leave unspecified columns unrestricted (default: FALSE, i.e. unspecified columns are fixed at the defaults listed below). Passing NULL for any parameter removes that filter entirely, returning all values for that column (e.g., min_size_quantile = NULL includes all size groups). Passing an unrecognised column name raises an error listing the supported names. Ignored when dataset != "factor_library". See the Details section for supported columns and their defaults.

Details

Note on dataset = "factor_library" **defaults:** The defaults below reflect one common portfolio construction choice, but may not suit every research question. Always verify that the selected combination matches your intended design.

Supported columns and their defaults for ...:

- sorting_variable: **Required.** The firm characteristic used to sort stocks into portfolios (e.g., "me" for market equity, "bm" for book-to-market). No default is applied.
- min_size_quantile (defaults to 0.2): Fraction of the smallest stocks (by market cap) excluded from the portfolio universe. 0.2 drops the bottom 20%.
- exclude_financials (defaults to FALSE): Whether to drop financial-sector stocks (SIC 6000-6999) from the universe.
- exclude_utilities (default: FALSE): Whether to drop utility-sector stocks (SIC 4900-4999) from the universe.
- exclude_negative_earnings (defaults to FALSE): Whether to drop firms with negative earnings before sorting.
- sorting_variable_lag (defaults to "6m"): Lag applied to the sorting variable before portfolio assignment (e.g., "6m" = 6-month lag).
- rebalancing (defaults to "monthly"): How frequently portfolios are reformed: "monthly" or "annual".
- n_portfolios_main (defaults to 10): Number of quantile groups (e.g., 10 for decile portfolios).
- sorting_method (defaults to "univariate"): Whether portfolios are formed on a single sort ("univariate") or a sequential double sort ("sequential").
- n_portfolios_secondary (defaults to NULL): Number of groups for the secondary sort variable. Required when sorting_method is not "univariate".

- `breakpoints_exchanges` (defaults to: `"NYSE"`): Exchange(s) used to compute breakpoints. `"NYSE"` uses only NYSE-listed stocks to define quantile cutoffs (the conventional Fama-French approach).
- `breakpoints_min_size_threshold` (defaults to `NULL`): Minimum market-cap threshold (in USD) applied when computing breakpoints. `NULL` means no minimum-size screen is applied.
- `weighting_scheme` (defaults to `"VW"`): Return weighting within portfolios: `"VW"` for value-weighted or `"EW"` for equal-weighted.

Value

A tibble with the downloaded data. For `"high_frequency_sp500"`, contains 5-second aggregated orderbook snapshots filtered to the requested date range. For `"factor_library"`, contains portfolio return data joined with the full grid metadata for the matched portfolio IDs.

See Also

Other download functions: `download_data()`, `download_data_constituents()`, `download_data_factors_ff()`, `download_data_factors_q()`, `download_data_fred()`, `download_data_macro_predictors()`, `download_data_osap()`, `download_data_risk_free()`, `download_data_stock_prices()`

Examples

```
## Not run:
  download_data_huggingface(
    "high_frequency_sp500", "2007-07-26", "2007-07-27"
  )
  download_data_huggingface(
    "factor_library",
    sorting_variable = "52w",
    rebalancing = "annual"
  )
  download_data_huggingface(
    "factor_library", sorting_variable = "ag", fill_all = TRUE
  )

## End(Not run)
```

`download_data_macro_predictors`

Download and Process Macro Predictor Data

Description

Downloads and processes macroeconomic predictor data based on the specified dataset (monthly, quarterly, or annual), date range, and source URL. The function downloads the data from a Google Sheets export link. It processes the raw data into a structured format, calculating additional financial metrics and filtering by the specified date range.

Usage

```
download_data_macro_predictors(
  dataset = NULL,
  start_date = NULL,
  end_date = NULL,
  type = deprecated(),
  sheet_id = "1bM7vCWd3WOt95Sf9qjLPZjoiafgF_8EG"
)
```

Arguments

dataset	The dataset to download ("monthly", "quarterly", "annual").
start_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, the full dataset is returned.
end_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, the full dataset is returned.
type	**[Deprecated]** Use `dataset` instead.
sheet_id	The Google Sheets ID from which to download the dataset, with the default "1bM7vCWd3WOt95Sf9qjLPZjoiafgF_8EG".

Value

A tibble with processed data, filtered by the specified date range and including financial metrics.

References

Welch, I., & Goyal, A. (2008). A comprehensive look at the empirical performance of equity premium prediction. *Review of Financial Studies*, 21(4), 1455-1508. doi:10.1093/rfs/hhm014

See Also

Other download functions: `download_data()`, `download_data_constituents()`, `download_data_factors_ff()`, `download_data_factors_q()`, `download_data_fred()`, `download_data_huggingface()`, `download_data_osap()`, `download_data_risk_free()`, `download_data_stock_prices()`

Examples

```
download_data_macro_predictors("monthly")
download_data_macro_predictors("quarterly", "2000-01-01", "2020-12-31")
```

`download_data_osap` *Download and Process Open Source Asset Pricing Data*

Description

Downloads the data from Open Source Asset Pricing from Google Sheets using a specified sheet ID, processes the data by converting column names to snake_case, and optionally filters the data based on a provided date range.

Usage

```
download_data_osap(
  start_date = NULL,
  end_date = NULL,
  sheet_id = "1JyhcF5PRKHcputlioxlu5j5GyLo4JYyY"
)
```

Arguments

`start_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, the full dataset is returned.
`end_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, the full dataset is returned.
`sheet_id`	A character string representing the Google Sheet ID from which to download the data. Default is `"1JyhcF5PRKHcputlioxlu5j5GyLo4JYyY"`.

Value

A tibble containing the processed data. The column names are converted to snake_case, and the data is filtered by the specified date range if `start_date` and `end_date` are provided.

See Also

Other download functions: `download_data()`, `download_data_constituents()`, `download_data_factors_ff()`, `download_data_factors_q()`, `download_data_fred()`, `download_data_huggingface()`, `download_data_macro_pred`
`download_data_risk_free()`, `download_data_stock_prices()`

Examples

```
osap <- download_data_osap(
  start_date = "2020-01-01", end_date = "2020-06-30"
)
```

download_data_risk_free *Download Risk-Free Rate Data*

Description

Downloads pre-processed risk-free rate data from the `tidy-finance/risk-free` dataset on HuggingFace. The dataset is updated monthly via a scheduled GitHub Actions workflow that splices the 3-Month Treasury Bill Secondary Market Rate (pre-2001) with the 4-Week Treasury Bill Secondary Market Rate (from 2001 onwards) sourced from FRED. For monthly data, the monthly TB3MS series is spliced with the daily DTB4WK series aggregated to month-end. For daily data, the daily DTB3 series is spliced with the daily DTB4WK series, both at the business-day frequency provided by FRED.

Usage

```
download_data_risk_free(
  start_date = NULL,
  end_date = NULL,
  frequency = "monthly"
)
```

Arguments

`start_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, the full dataset is returned.
`end_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, the full dataset is returned.
`frequency`	A character string, either `"monthly"` (default) or `"daily"`, specifying the frequency of the returned data. Daily data starts in 1954-01-04 because of availability of DTB3, while monthly data starts in 1934-01-01.

Details

Both series are quoted as annualised bank discount rates on a 360-day basis. Given an annualised discount rate `d` and a T-bill with `n` days to maturity, the holding-period return is `HPR = d * n/360 / (1 - d * n/360)`, which is then converted to the target period length via `(1 + HPR)^(target/source) - 1`.

The series are spliced at 2001-07-01:

- **Pre-2001**: TB3MS (monthly) or DTB3 (daily), 3-month T-bill with n = 90. Monthly conversion uses exponent `1/3`; daily conversion uses exponent `1/63` (approx. trading days per quarter).
- **From 2001**: DTB4WK, 4-week T-bill with n = 28. For monthly data, the last non-NA observation per calendar month is taken and the exponent is `365/(28*12)`. For daily data, observations are used as-is and the exponent is `1/20` (approx. trading days per 4-week period).

Business-day gaps in the daily series (e.g. holidays) are handled by forward-filling the most recent available rate.

Monthly data starts in 1934-01-01 (TB3MS). Daily data starts in 1954-01-04 due to the availability of DTB3.

Value

A tibble with two columns:

date The date of the observation.

risk_free The risk-free rate for the period.

See Also

Other download functions: download_data(), download_data_constituents(), download_data_factors_ff(), download_data_factors_q(), download_data_fred(), download_data_huggingface(), download_data_macro_pred download_data_osap(), download_data_stock_prices()

Examples

```
download_data_risk_free("2020-01-01", "2020-12-31")
download_data_risk_free(
  "2020-01-01", "2020-12-31", frequency = "daily"
)
```

---

download_data_stock_prices

Download Stock Data

Description

Downloads historical stock data from Yahoo Finance for given symbols and date range.

Usage

```
download_data_stock_prices(symbols, start_date = NULL, end_date = NULL)
```

Arguments

symbols	A character vector of stock symbols to download data for. At least one symbol must be provided.
start_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, a one-year subset of the dataset is returned (see validate_dates()).
end_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, a one-year subset of the dataset is returned (see validate_dates()).

Value

A tibble containing the downloaded stock data with columns: symbol, date, volume, open, low, high, close, and adjusted_close.

See Also

Other download functions: download_data(), download_data_constituents(), download_data_factors_ff(), download_data_factors_q(), download_data_fred(), download_data_huggingface(), download_data_macro_pred download_data_osap(), download_data_risk_free()

Examples

```
download_data_stock_prices(c("AAPL", "MSFT"))
download_data_stock_prices("GOOGL", "2021-01-01", "2022-01-01" )
```

---

download_data_wrds *Download Data from WRDS*

Description

Acts as a wrapper to download data from various WRDS datasets including CRSP, Compustat, and CCM links based on the specified dataset. It is designed to handle different datasets by redirecting to the appropriate specific data download function.

Usage

```
download_data_wrds(
  dataset = NULL,
  start_date = NULL,
  end_date = NULL,
  type = deprecated(),
  ...
)
```

Arguments

dataset	A string specifying the dataset to download. Supported values: "crsp_monthly", "crsp_daily" for CRSP data, "compustat_annual", "compustat_quarterly" for Compustat data, "ccm_links" for CCM links data, "fisd" for FISD data, or "trace_enhanced" for TRACE data.
start_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, a subset of the dataset is returned.
end_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, a subset of the dataset is returned.

type	**[Deprecated]** Use dataset instead.
...	Additional arguments passed to specific download functions depending on the dataset.

Value

A data frame containing the requested data, with the structure and contents depending on the specified dataset.

See Also

Other WRDS functions: create_wrds_dummy_database(), disconnect_connection(), download_data_wrds_ccm_link
download_data_wrds_compustat(), download_data_wrds_crsp(), download_data_wrds_fisd(),
download_data_wrds_trace_enhanced(), get_wrds_connection(), set_wrds_credentials()

Examples

```
## Not run:
  crsp_monthly <- download_data_wrds(
    "crsp_monthly", "2020-01-01", "2020-12-31"
  )
  compustat_annual <- download_data_wrds(
    "compustat_annual", "2020-01-01", "2020-12-31"
  )
  ccm_links <- download_data_wrds("ccm_links")
  fisd <- download_data_wrds("fisd")
  trace_enhanced <- download_data_wrds(
    "trace_enhanced", cusips = "00101JAH9"
  )

## End(Not run)
```

---

download_data_wrds_ccm_links

Download CCM Links from WRDS

Description

Downloads data from the WRDS CRSP/Compustat Merged (CCM) links database. It allows users to specify the type of links (linktype) and the primacy of the link (linkprim).

Usage

```
download_data_wrds_ccm_links(linktype = c("LU", "LC"), linkprim = c("P", "C"))
```

Arguments

linktype	A character vector indicating the type of link to download. The default is c("LU", "LC"), where "LU" stands for "Link Up" and "LC" for "Link CRSP".
linkprim	A character vector indicating the primacy of the link. Default is c("P", "C"), where "P" indicates primary and "C" indicates conditional links.

Value

A data frame with the columns permno, gvkey, linkdt, and linkenddt, where linkenddt is the end date of the link, and missing end dates are replaced with today's date.

See Also

Other WRDS functions: create_wrds_dummy_database(), disconnect_connection(), download_data_wrds(), download_data_wrds_compustat(), download_data_wrds_crsp(), download_data_wrds_fisd(), download_data_wrds_trace_enhanced(), get_wrds_connection(), set_wrds_credentials()

Examples

```
## Not run:
  ccm_links <- download_data_wrds_ccm_links(linktype = "LU", linkprim = "P")

## End(Not run)
```

---

download_data_wrds_compustat

Download Data from WRDS Compustat

Description

Downloads financial data from the WRDS Compustat database for a given dataset, start date, and end date. It filters the data according to industry format, data format, and consolidation level, and returns the most current data for each reporting period. Additionally, the annual data also includes the calculated book equity (be), operating profitability (op), and investment (inv) for each company following Fama & French (1993, 2015), as well as income before extraordinary items (ib).

Usage

```
download_data_wrds_compustat(
  dataset = NULL,
  start_date = NULL,
  end_date = NULL,
  type = deprecated(),
  additional_columns = NULL,
  only_us = FALSE
)
```

Arguments

dataset	The dataset to download ("compustat_annual" or "compustat_quarterly").
start_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, a subset of the dataset is returned.
end_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, a subset of the dataset is returned.
type	**[Deprecated]** Use dataset instead.
additional_columns	Additional columns from the Compustat table as a character vector.
only_us	A logical indicating whether only US firms should be returned.

Value

A data frame with financial data for the specified period, including variables for book equity (be), operating profitability (op), investment (inv), and others.

References

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1), 3-56. doi:10.1016/0304405X(93)900235

Fama, E. F., & French, K. R. (2015). A five-factor asset pricing model. *Journal of Financial Economics*, 116(1), 1-22. doi:10.1016/j.jfineco.2014.10.010

See Also

Other WRDS functions: create_wrds_dummy_database(), disconnect_connection(), download_data_wrds(), download_data_wrds_ccm_links(), download_data_wrds_crsp(), download_data_wrds_fisd(), download_data_wrds_trace_enhanced(), get_wrds_connection(), set_wrds_credentials()

Examples

```
## Not run:
download_data_wrds_compustat("compustat_annual", "2020-01-01", "2020-12-31")
download_data_wrds_compustat(
  "compustat_quarterly",
  "2020-01-01",
  "2020-12-31"
)

  # Add additional columns
download_data_wrds_compustat(
  "compustat_annual",
  additional_columns = c("aodo", "aldo")
)

## End(Not run)
```

`download_data_wrds_crsp` *Download Data from WRDS CRSP*

Description

Downloads and processes stock return data from the CRSP database for a specified period. Users can choose between monthly and daily datasets. The function also adjusts returns for delisting and calculates market capitalization and excess returns over the risk-free rate.

Usage

```
download_data_wrds_crsp(
  dataset = NULL,
  start_date = NULL,
  end_date = NULL,
  type = deprecated(),
  batch_size = 500,
  version = "v2",
  additional_columns = NULL,
  add_ccm_links = FALSE,
  adjust_volume = FALSE
)
```

Arguments

`dataset`	A string specifying the CRSP dataset to download: "crsp_monthly" or "crsp_daily".
`start_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, a subset of the dataset is returned.
`end_date`	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, a subset of the dataset is returned.
`type`	**[Deprecated]** Use `dataset` instead.
`batch_size`	An optional integer specifying the batch size for processing daily data, with a default of 500.
`version`	An optional character specifying which CRSP version to use. "v2" (the default) uses the updated second version of CRSP, and "v1" downloads the legacy version of CRSP.
`additional_columns`	Additional columns from the CRSP monthly or daily data as a character vector.
`add_ccm_links`	A logical indicating whether CRSP-Compustat links should be added automatically using `download_data_wrds_ccm_links()`.
`adjust_volume`	A logical indicating whether daily CRSP trading volume data should be adjusted according to Gao & Ritter (2010).

Value

A data frame containing CRSP stock returns, adjusted for delistings, along with calculated market capitalization and excess returns over the risk-free rate. The structure of the returned data frame depends on the selected dataset.

References

Gao, X., & Ritter, J. R. (2010). The marketing of seasoned equity offerings. *Journal of Financial Economics*, 97(1), 33-52. doi:10.1016/j.jfineco.2010.03.007

See Also

Other WRDS functions: `create_wrds_dummy_database()`, `disconnect_connection()`, `download_data_wrds()`, `download_data_wrds_ccm_links()`, `download_data_wrds_compustat()`, `download_data_wrds_fisd()`, `download_data_wrds_trace_enhanced()`, `get_wrds_connection()`, `set_wrds_credentials()`

Examples

```
## Not run:
crsp_monthly <- download_data_wrds_crsp(
  "crsp_monthly",
  "2020-11-01",
  "2020-12-31"
)
crsp_daily <- download_data_wrds_crsp(
  "crsp_daily",
  "2020-12-01",
  "2020-12-31"
)

# Add additional columns
download_data_wrds_crsp(
  "crsp_monthly",
  "2020-11-01",
  "2020-12-31",
  additional_columns = c("mthvol", "mthvolflg")
)

## End(Not run)
```

`download_data_wrds_fisd`

Download Filtered FISD Data from WRDS

Description

Establishes a connection to the WRDS database to download a filtered subset of the FISD (Fixed Income Securities Database). The function filters the `fisd_mergedissue` and `fisd_mergedissuer` tables based on several criteria related to the securities, such as security level, bond type, coupon type, and others, focusing on specific attributes that denote the nature of the securities. It finally returns a data frame with selected fields from the `fisd_mergedissue` table after joining it with issuer information from the `fisd_mergedissuer` table for issuers domiciled in the USA.

Usage

```
download_data_wrds_fisd(additional_columns = NULL)
```

Arguments

`additional_columns`
: Additional columns from the FISD table as a character vector.

Value

A data frame containing a subset of FISD data with fields related to the bond's characteristics and issuer information. This includes complete CUSIP, maturity date, offering amount, offering date, dated date, interest frequency, coupon, last interest date, issue ID, issuer ID, and SIC code of the issuer.

See Also

Other WRDS functions: `create_wrds_dummy_database()`, `disconnect_connection()`, `download_data_wrds()`, `download_data_wrds_ccm_links()`, `download_data_wrds_compustat()`, `download_data_wrds_crsp()`, `download_data_wrds_trace_enhanced()`, `get_wrds_connection()`, `set_wrds_credentials()`

Examples

```
## Not run:
fisd <- download_data_wrds_fisd()
fisd_extended <- download_data_wrds_fisd(
  additional_columns = c("asset_backed", "defeased")
)

## End(Not run)
```

`download_data_wrds_trace_enhanced`

Download Enhanced TRACE Data from WRDS

Description

Establishes a connection to the WRDS database to download the specified CUSIPs trade messages from the Trade Reporting and Compliance Engine (TRACE). The trade data is cleaned as suggested by Dick-Nielsen (2009, 2014).

Usage

```
download_data_wrds_trace_enhanced(cusips, start_date = NULL, end_date = NULL)
```

Arguments

cusips	A character vector specifying the 9-digit CUSIPs to download.
start_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the start date for the data. If not provided, a subset of the dataset is returned.
end_date	Optional. A character string or Date object in "YYYY-MM-DD" format specifying the end date for the data. If not provided, a subset of the dataset is returned.

Value

A data frame containing the cleaned trade messages from TRACE for the selected CUSIPs over the time window specified. Output variables include identifying information (i.e., CUSIP, trade date/time) and trade-specific information (i.e., price/yield, volume, counterparty, and reporting side).

References

Dick-Nielsen, J. (2009). Liquidity biases in TRACE. *Journal of Fixed Income*, 19(2), 43-55. doi:10.3905/jfi.2009.19.2.043

Dick-Nielsen, J. (2014). How to clean enhanced TRACE data. Working Paper. doi:10.2139/ssrn.2337908

See Also

Other WRDS functions: `create_wrds_dummy_database()`, `disconnect_connection()`, `download_data_wrds()`, `download_data_wrds_ccm_links()`, `download_data_wrds_compustat()`, `download_data_wrds_crsp()`, `download_data_wrds_fisd()`, `get_wrds_connection()`, `set_wrds_credentials()`

Examples

```
## Not run:
download_data_wrds_trace_enhanced("00101JAH9", "2019-01-01", "2021-12-31")

## End(Not run)
```

---

estimate_betas *Estimate Rolling Betas*

Description

Estimates rolling betas for a given model using the provided data. It supports parallel processing for faster computation using the `furrr` package.

Usage

```
estimate_betas(
  data,
  model,
  lookback,
  min_obs = NULL,
  use_furrr = FALSE,
  data_options = NULL
)
```

Arguments

data	A data frame containing the data with a date identifier (defaults to date), a stock identifier (defaults to permno), and other variables used in the model.
model	A character string describing the model to be estimated (e.g., "ret_excess ~ mkt_excess + hml + smb").
lookback	A Period object specifying the number of months, days, hours, minutes, or seconds to look back when estimating the rolling model.
min_obs	An integer specifying the minimum number of observations required to estimate the model. Defaults to 80% of lookback.
use_furrr	A logical indicating whether to use the furrr package and its parallelization capabilities. Defaults to FALSE.
data_options	A list of class tidyfinance_data_options (created via data_options()) specifying column name mappings. The id is used to specify the entity (i.e., firm), and the date element is used to specify the date column. Uses data_options() default if NULL: "id" = "permno" and "date" = "date".

Value

A data frame with the estimated betas for each time period.

See Also

Other estimation functions: estimate_fama_macbeth(), estimate_model()

Examples

```
# Estimate monthly betas using monthly return data
set.seed(1234)
data_monthly <- tibble::tibble(
  date = rep(seq.Date(from = as.Date("2020-01-01"),
                      to = as.Date("2020-12-01"), by = "month"), each = 50),
  permno = rep(1:50, times = 12),
  ret_excess = rnorm(600, 0, 0.1),
  mkt_excess = rnorm(600, 0, 0.1),
  smb = rnorm(600, 0, 0.1),
  hml = rnorm(600, 0, 0.1),
)
```

```
estimate_betas(data_monthly,  "ret_excess ~ mkt_excess", months(3))
estimate_betas(
  data_monthly,
  "ret_excess ~ mkt_excess + smb + hml",
  months(6)
)

data_monthly |>
  dplyr::rename(id = permno) |>
  estimate_betas("ret_excess ~ mkt_excess", months(3),
                 data_options = data_options(id = "id"))

# Estimate monthly betas using daily return data and parallelization
data_daily <- tibble::tibble(
  date = rep(seq.Date(from = as.Date("2020-01-01"),
                      to = as.Date("2020-12-31"), by = "day"), each = 50),
  permno = rep(1:50, times = 366),
  ret_excess = rnorm(18300, 0, 0.02),
  mkt_excess = rnorm(18300, 0, 0.02),
  smb = rnorm(18300, 0, 0.02),
  hml = rnorm(18300, 0, 0.02),
)

data_daily <- data_daily |>
  dplyr::mutate(date = lubridate::floor_date(date, "month"))

# Change settings via future::plan(strategy = "multisession", workers = 4)
estimate_betas(
  data_daily,
  "ret_excess ~ mkt_excess",
  lubridate::days(90),
  use_furrr = TRUE
)
```

`estimate_fama_macbeth` *Estimate Fama-MacBeth Regressions*

Description

Estimates Fama-MacBeth regressions (Fama and MacBeth, 1973) by first running cross-sectional regressions for each time period and then aggregating the results over time to obtain average risk premia and corresponding t-statistics.

Usage

```
estimate_fama_macbeth(
  data,
  model,
```

```
  vcov = "newey-west",
  vcov_options = NULL,
  data_options = NULL,
  detail = FALSE
)
```

Arguments

data	A data frame containing the data for the regression. It must include a column representing the time periods (defaults to date) and the variables specified in the model.
model	A character string describing the model to be estimated in each cross-section (e.g., "ret_excess ~ beta + bm + log_mktcap").
vcov	A character string indicating the type of standard errors to compute. Options are "iid" for independent and identically distributed errors or "newey-west" for Newey-West standard errors. Default is "newey-west".
vcov_options	A list of additional arguments to be passed to the NeweyWest() function when vcov = "newey-west". These can include options such as lag, which specifies the number of lags to use in the Newey-West covariance matrix estimation, and prewhite, which indicates whether to apply a prewhitening transformation. Default is an empty list.
data_options	A list of class tidyfinance_data_options (created via data_options()) specifying column name mappings. The date element is used to specify the date column. Uses data_options() default if NULL: "date" = "date".
detail	A logical value indicating whether to return additional summary statistics. If FALSE (default), the function returns only the coefficient estimates. If TRUE, it returns a list with two elements: coefficients (the usual estimates table) and summary_statistics (a one-row tibble with the average cross-sectional R-squared and the average number of observations per cross-section).

Value

If detail = FALSE (default), a tibble with columns factor, risk_premium, n (number of time periods), standard_error, and t_statistic.

If detail = TRUE, a named list with two elements:

coefficients The same tibble described above.

summary_statistics A one-row tibble with r_squared (mean cross-sectional R-squared) and n_obs (mean cross-sectional observation count).

References

Fama, E. F., & MacBeth, J. D. (1973). Risk, return, and equilibrium: Empirical tests. *Journal of Political Economy*, 81(3), 607-636. doi:10.1086/260061

Newey, W. K., & West, K. D. (1987). A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix. *Econometrica*, 55(3), 703-708. doi:10.2307/1913610

See Also

Other estimation functions: `estimate_betas()`, `estimate_model()`

Examples

```
set.seed(1234)

data <- tibble::tibble(
  date = rep(seq.Date(from = as.Date("2020-01-01"),
                      to = as.Date("2020-12-01"), by = "month"), each = 50),
  permno = rep(1:50, times = 12),
  ret_excess = rnorm(600, 0, 0.1),
  beta = rnorm(600, 1, 0.2),
  bm = rnorm(600, 0.5, 0.1),
  log_mktcap = rnorm(600, 10, 1)
)

estimate_fama_macbeth(data, "ret_excess ~ beta + bm + log_mktcap")
estimate_fama_macbeth(
  data,
  "ret_excess ~ beta + bm + log_mktcap",
  vcov = "iid"
)
estimate_fama_macbeth(
  data,
  "ret_excess ~ beta + bm + log_mktcap",
  vcov = "newey-west",
  vcov_options = list(lag = 6, prewhite = FALSE)
)

# Return detailed output including R-squared and observation counts
estimate_fama_macbeth(
  data,
  "ret_excess ~ beta + bm + log_mktcap",
  detail = TRUE
)

# Use different column name for date
data |>
  dplyr::rename(month = date) |>
  estimate_fama_macbeth(
    "ret_excess ~ beta + bm + log_mktcap",
    data_options = data_options(date = "month")
 )
```

`estimate_model` *Estimate a Linear Model*

Description

[Experimental]

Estimates a linear model specified by one or more independent variables. It checks for the presence of the specified independent variables in the dataset and whether the dataset has a sufficient number of observations. Depending on the `output` parameter, it returns the model's coefficients, t-statistics, residuals, or any combination in a named list.

Usage

```
estimate_model(data, model, min_obs = 1, output = "coefficients")
```

Arguments

`data`	A data frame containing the dependent variable and one or more independent variables.
`model`	A character that describes the model to be estimated (e.g., `"ret_excess ~ mkt_excess + hml + smb"`).
`min_obs`	The minimum number of observations required to estimate the model. Defaults to 1.
`output`	A character vector specifying what to return. Must contain one or more of `"coefficients"` (default), `"residuals"`, and `"tstats"`. If a single value is provided, the corresponding object is returned directly. If multiple values are provided, a named list is returned.

Value

If `output` contains a single value: a data frame of coefficients or t-statistics, or a numeric vector of residuals. If `output` contains multiple values: a named list with the requested elements. Coefficients and t-statistics are returned as data frames with column names corresponding to the model terms. Residuals are returned as a numeric vector of length `nrow(data)` with `NA` for rows with missing data or insufficient observations.

See Also

Other estimation functions: `estimate_betas()`, `estimate_fama_macbeth()`

Examples

```
set.seed(42)
data <- data.frame(
  ret_excess = rnorm(100),
  mkt_excess = rnorm(100),
  smb = rnorm(100),
  hml = rnorm(100)
)

# Estimate model with a single independent variable
estimate_model(data, "ret_excess ~ mkt_excess")
```

```
# Estimate model with multiple independent variables
estimate_model(data, "ret_excess ~ mkt_excess + smb + hml")

# Estimate model without intercept
estimate_model(data, "ret_excess ~ mkt_excess - 1")

# Calculate residuals
estimate_model(data, "ret_excess ~ mkt_excess + smb + hml",
  output = "residuals"
)

# Return t-statistics
estimate_model(data, "ret_excess ~ mkt_excess + smb + hml",
  output = "tstats"
)

# Return coefficients, t-statistics, and residuals
estimate_model(data, "ret_excess ~ mkt_excess + smb + hml",
  output = c("coefficients", "tstats", "residuals")
)
```

`filter_options` *Create Filter Options*

Description

Creates a list of filter options of class `tidyfinance_filter_options` used for sample construction in TidyFinance-related functions. These options control which observations are retained before portfolio sorting.

Usage

```
filter_options(
  exclude_financials = FALSE,
  exclude_utilities = FALSE,
  min_stock_price = NULL,
  min_size_quantile = NULL,
  min_listing_age = NULL,
  exclude_negative_book_equity = FALSE,
  exclude_negative_earnings = FALSE,
  ...
)
```

Arguments

`exclude_financials`
: A logical indicating whether to exclude financial firms (SIC codes 6000–6799). Defaults to `FALSE`.

exclude_utilities
: A logical indicating whether to exclude utility firms (SIC codes 4900–4999). Defaults to FALSE.

min_stock_price
: A single positive numeric specifying the minimum stock price required to include an observation. NULL (the default) applies no price filter.

min_size_quantile
: A single numeric strictly between 0 and 1 specifying the minimum cross-sectional size quantile (based on lagged market cap) required to include an observation. NULL (the default) applies no size quantile filter. The cutoff is computed from NYSE stocks only. This requires an exchange column in the data (as mapped via data_options()); an error is raised if it is missing.

min_listing_age
: A single non-negative integer or numeric specifying the minimum number of months a stock must have been listed in CRSP. NULL (the default) applies no listing age filter.

exclude_negative_book_equity
: A logical indicating whether to exclude observations with non-positive book equity. Defaults to FALSE.

exclude_negative_earnings
: A logical indicating whether to exclude observations with non-positive earnings. Defaults to FALSE.

...
: Additional arguments to be included in the filter options list.

Value

A list of class tidyfinance_filter_options containing the specified filter options.

See Also

Other portfolio functions: assign_portfolio(), breakpoint_options(), compute_breakpoints(), compute_long_short_returns(), compute_portfolio_returns(), data_options(), filter_sorting_data(), implement_portfolio_sort(), portfolio_sort_options()

Examples

```
filter_options(
  exclude_financials = TRUE,
  exclude_utilities = TRUE,
  min_stock_price = 1,
  min_listing_age = 12
)
```

`filter_sorting_data` *Filter Sorting Data*

Description

Applies sample construction filters to a data frame before portfolio sorting. Filters are applied in a fixed order: financials exclusion, utilities exclusion, minimum stock price, minimum size quantile, minimum listing age, positive book equity, and positive earnings. An informational message is emitted for each filter that actually removes at least one observation.

Usage

```
filter_sorting_data(
  data,
  filter_options = NULL,
  data_options = NULL,
  quiet = FALSE
)
```

Arguments

`data`	A data frame containing the stock-level panel data to be filtered.
`filter_options`	A list of class `tidyfinance_filter_options` created by `filter_options()`. If `NULL` (the default), the defaults from `filter_options()` are used (i.e., no filters are applied). The arguments accepted by `filter_options()` include • `exclude_financials` A logical indicating whether to exclude financial firms (SIC codes 6000–6799). Defaults to `FALSE`. • `exclude_utilities` A logical indicating whether to exclude utility firms (SIC codes 4900–4999). Defaults to `FALSE`. • `min_stock_price` A single positive numeric specifying the minimum stock price required to include an observation. `NULL` (the default) applies no price filter. • `min_size_quantile` A single numeric strictly between 0 and 1 specifying the minimum cross-sectional size quantile (based on lagged market cap) required to include an observation. `NULL` (the default) applies no size quantile filter. The cutoff is computed from NYSE stocks only; the `exchange` column (mapped via `data_options()`) must be present in the data or an error is raised. • `min_listing_age` A single non-negative integer or numeric specifying the minimum number of months a stock must have been listed in CRSP. `NULL` (the default) applies no listing age filter. • `exclude_negative_book_equity` A logical indicating whether to exclude observations with non-positive book equity. Defaults to `FALSE`. • `exclude_negative_earnings` A logical indicating whether to exclude observations with non-positive earnings. Defaults to `FALSE`.

data_options	A list of class tidyfinance_data_options (created via data_options()) specifying column name mappings. The siccd element is used to specify the SIC code column, price is used to specify the (adjusted) price column, mktcap_lag is used to specify the market capitalization column, date is used to specify the date column, listing_age is used to specify the listing age column, be is used to specify the book equity column, and earnings is used to specify the earnings column. Uses data_options() default if NULL: "siccd" = "siccd", "price" = "prc_adj", "exchange" = "exchange" "mktcap_lag" = "mktcap_lag", "date" = "date", "listing_age" = "listing_age", "be" = "be", and "earnings" = "ib".
quiet	A logical indicating whether informational messages should be suppressed. Defaults to FALSE.

Value

The filtered data frame, preserving the class and structure of the input.

See Also

Other portfolio functions: assign_portfolio(), breakpoint_options(), compute_breakpoints(), compute_long_short_returns(), compute_portfolio_returns(), data_options(), filter_options(), implement_portfolio_sort(), portfolio_sort_options()

Examples

```
data <- data.frame(
  permno = 1:5,
  date = as.Date("2020-01-01"),
  siccd = c(6100, 2000, 4950, 3000, 6500),
  prc_adj = c(5, 0.5, 15, 20, 10)
)

data |>
  filter_sorting_data(
    filter_options = filter_options(
      exclude_financials = TRUE,
      min_stock_price = 1
    )
  )
```

---

get_available_huggingface_files

List Parquet Files in a Hugging Face Dataset

Description

Query the Hugging Face Datasets API and return a tibble of files with a .parquet suffix. The function follows pagination links returned in the response Link header and returns path, size, and a resolved URL.

Usage

```
get_available_huggingface_files(organization, dataset)
```

Arguments

organization	Character(1). Hugging Face organization or user name.
dataset	Character(1). Dataset name under the organization.

Details

Uses httr2 to perform HTTP requests. Requires internet access and the dataset to be publicly accessible or accessible with appropriate authentication.

Value

A tibble with columns: path (character), size (numeric), and url (character).

See Also

Other utility functions: create_summary_statistics(), list_supported_indexes(), list_supported_types(), list_supported_types_ff(), list_supported_types_ff_legacy(), list_supported_types_macro_predictors(), list_supported_types_other(), list_supported_types_wrds(), list_tidy_finance_chapters(), open_tidy_finance_website(), trim(), validate_dates(), winsorize()

Examples

```
## Not run:
get_available_huggingface_files("voigtstefan", "sp500")

## End(Not run)
```

---

get_wrds_connection *Establish a Connection to the WRDS Database*

Description

Establishes a connection to the Wharton Research Data Services (WRDS) database using the RPostgres package. It requires that the RPostgres package be installed and that valid WRDS credentials be set as environment variables.

Usage

```
get_wrds_connection()
```

Details

The function checks if the RPostgres package is installed before attempting to establish a connection. It uses the host, dbname, port, and sslmode as fixed parameters for the connection. Users must set their WRDS username and password as environment variables WRDS_USER and WRDS_PASSWORD, respectively, before using this function.

Value

An object of class DBIConnection representing the connection to the WRDS database. This object can be used with other DBI-compliant functions to interact with the database.

See Also

RPostgres::Postgres(), DBI::dbDisconnect() for more information on managing database connections.

Other WRDS functions: create_wrds_dummy_database(), disconnect_connection(), download_data_wrds(), download_data_wrds_ccm_links(), download_data_wrds_compustat(), download_data_wrds_crsp(), download_data_wrds_fisd(), download_data_wrds_trace_enhanced(), set_wrds_credentials()

Examples

```
## Not run:
  # Before using this function, set your WRDS credentials:
  # Sys.setenv(WRDS_USER = "your_username", WRDS_PASSWORD = "your_password")

  # con <- get_wrds_connection()
  # Use `con` with DBI-compliant functions to interact with the WRDS database
  # Remember to disconnect after use:
  # disconnect_connection(con)

## End(Not run)
```

---

implement_portfolio_sort

Implement Portfolio Sort

Description

A convenience wrapper that combines sample construction filtering and portfolio return computation into a single call. Equivalent to calling filter_sorting_data() followed by compute_portfolio_returns().

Usage

```
implement_portfolio_sort(
  data,
  sorting_variables,
  sorting_method,
```

```
  rebalancing_month = NULL,
  portfolio_sort_options,
  breakpoint_function_main = compute_breakpoints,
  breakpoint_function_secondary = compute_breakpoints,
  min_portfolio_size = 1L,
  cap_weight = 0.8,
  data_options = NULL,
  quiet = FALSE
)
```

Arguments

`data` A data frame containing the stock-level panel data.

`sorting_variables`
A character vector of one or two column names to sort portfolios on.

`sorting_method` A string specifying the sorting method: `"univariate"`, `"bivariate-dependent"`, or `"bivariate-independent"`.

`rebalancing_month`
An optional integer specifying the month in which portfolios are rebalanced annually. `NULL` (the default) means monthly rebalancing.

`portfolio_sort_options`
A list of class `tidyfinance_portfolio_sort_options` created by `portfolio_sort_options()`, bundling filter and breakpoint specifications. The arguments accepted by `portfolio_sort_options()` include

- `filter_options` A list of class `tidyfinance_filter_options` created by `filter_options()`, or `NULL` (the default, which applies no filters). Options include `exclude_financials`, `exclude_utilities`, `min_stock_price`, `min_size_quantile`, `min_listing_age`, `exclude_negative_book_equity`, and `exclude_negative_earnings`.
- `breakpoint_options_main` A list of class `tidyfinance_breakpoint_options` created by `breakpoint_options()`, specifying breakpoints for the primary sorting variable. Options include `n_portfolios`, `percentiles`, `breakpoints_exchanges`, `smooth_bunching`, and `breakpoints_min_size_threshold`.
- `breakpoint_options_secondary` A list of class `tidyfinance_breakpoint_options` created by `breakpoint_options()`, specifying breakpoints for the secondary sorting variable, or `NULL` (the default) for univariate sorts. Options are the same as for `breakpoint_options_main`.

`breakpoint_function_main`
The function used to compute breakpoints for the main sorting variable. Defaults to `compute_breakpoints()`.

`breakpoint_function_secondary`
The function used to compute breakpoints for the secondary sorting variable. Defaults to `compute_breakpoints()`.

`min_portfolio_size`
An integer specifying the minimum number of firms in the reported portfolio cross-section on a given date. Defaults to `1L` (at least one observation per reported portfolio). For univariate sorts that is firms per portfolio-date; for bivariate sorts that is firms per main-portfolio-date summed across the secondary

	buckets. Cross-sections below the threshold have their returns set to NA. Set to 0L to deactivate the check. See compute_portfolio_returns().
cap_weight	A numeric between 0 and 1 specifying the quantile at which portfolio weights are capped for the capped value-weighted return. Defaults to 0.8.
data_options	A list of class tidyfinance_data_options (created via data_options()) specifying column name mappings. All elements are forwarded to filter_sorting_data() and compute_portfolio_returns(). Uses data_options() default if NULL: "id" = "permno", "date" = "date", "exchange" = "exchange", "mktcap_lag" = "mktcap_lag", "ret_excess" = "ret_excess", "siccd" = "siccd", "price" = "prc_adj", "listing_age" = "listing_age", "be" = "be", and "earnings" = "ib".
quiet	A logical indicating whether informational messages should be suppressed. Defaults to FALSE.

Value

A data frame of portfolio returns as returned by compute_portfolio_returns().

See Also

Other portfolio functions: assign_portfolio(), breakpoint_options(), compute_breakpoints(), compute_long_short_returns(), compute_portfolio_returns(), data_options(), filter_options(), filter_sorting_data(), portfolio_sort_options()

Examples

```
set.seed(123)
data <- data.frame(
  permno = 1:500,
  date = rep(
    seq.Date(
      from = as.Date("2020-01-01"),
      by = "month",
      length.out = 100
    ),
    each = 10
  ),
  mktcap_lag = runif(500, 100, 1000),
  ret_excess = rnorm(500),
  prc_adj = runif(500, 0.5, 50),
  size = runif(500, 50, 150)
)
implement_portfolio_sort(
  data = data,
  sorting_variables = "size",
  sorting_method = "univariate",
  portfolio_sort_options = portfolio_sort_options(
    filter_options = filter_options(
      min_stock_price = 1
    ),
```

```
    breakpoint_options_main = breakpoint_options(n_portfolios = 5)
  )
)
```

`join_lagged_values` *Join Lagged Variable Values over a Date Range*

Description

[Experimental]

Joins lagged values of selected variables from one dataset (`new_data`) into another (`original_data`), based on date ranges defined by `min_lag` and `max_lag`. Unlike `add_lagged_columns()`, this function supports joining across data frames with different date grids (e.g., monthly source data into quarterly target data).

Usage

```
join_lagged_values(
  original_data,
  new_data,
  id_keys,
  min_lag,
  max_lag,
  ff_adjustment = FALSE,
  data_options = NULL
)
```

Arguments

`original_data`	A data frame containing the target panel data.
`new_data`	A data frame containing the source variables to lag and merge. All columns besides `id_keys` and the date column will be lagged and joined.
`id_keys`	A character vector specifying the identifier column(s).
`min_lag`	A `lubridate::Period` specifying the lower lag bound (inclusive).
`max_lag`	A `lubridate::Period` specifying the upper lag bound (inclusive).
`ff_adjustment`	Logical; if `TRUE`, keeps only the last observation per identifier and year before lagging (Fama-French convention). Defaults to `FALSE`.
`data_options`	A list of class `tidyfinance_data_options` (created via `data_options()`) specifying column name mappings. The `date` element is used to identify the date column. Uses `data_options()` default if `NULL`: `"date" = "date"`.

Value

A data frame with all columns from `original_data` plus the lagged columns from `new_data` (keeping their original names).

See Also

Other rolling and lagging functions: `add_lagged_columns()`, `compute_rolling_value()`

Examples

```
set.seed(42)
library(dplyr)
library(lubridate)

df1 <- tibble(
  id = rep(1:2, each = 6),
  date = rep(seq(as.Date("2020-01-01"), by = "month", length.out = 6), 2)
)

df2 <- df1 |>
  mutate(x = rnorm(n()))

join_lagged_values(
  original_data = df1,
  new_data = df2,
  id_keys = "id",
  min_lag = months(1),
  max_lag = months(3)
)
```

---

list_supported_indexes

List Supported Indexes

Description

Returns a tibble containing information about supported financial indexes. Each index is associated with a URL that points to a CSV file containing the holdings of the index. Additionally, each index has a corresponding `skip` value, which indicates the number of lines to skip when reading the CSV file.

Usage

```
list_supported_indexes()
```

Value

A tibble with three columns:

index The name of the financial index (e.g., "DAX", "S&P 500").

url The URL to the CSV file containing the holdings data for the index.

skip The number of lines to skip when reading the CSV file.

See Also

Other utility functions: create_summary_statistics(), get_available_huggingface_files(), list_supported_types(), list_supported_types_ff(), list_supported_types_ff_legacy(), list_supported_types_macro_predictors(), list_supported_types_other(), list_supported_types_wrds(), list_tidy_finance_chapters(), open_tidy_finance_website(), trim(), validate_dates(), winsorize()

Examples

```
supported_indexes <- list_supported_indexes()
print(supported_indexes)
```

---

list_supported_types *List All Supported Dataset Types*

Description

Aggregates and returns a comprehensive tibble of all supported dataset types from different domains. It includes various datasets across different frequencies (daily, weekly, monthly, quarterly, annual) and models (e.g., q5 factors, Fama-French 3 and 5 factors, macro predictors).

Usage

```
list_supported_types(domain = NULL, as_vector = FALSE)
```

Arguments

domain	A character vector to filter for domain-specific types (e.g., c("WRDS", "Fama-French"))
as_vector	Logical indicating whether types should be returned as a character vector instead of a data frame.

Value

A tibble aggregating all supported dataset types with columns: type (the type of dataset), dataset_name (a descriptive name or file name of the dataset), and domain (the domain to which the dataset belongs, e.g., "Global Q", "Fama-French", "Goyal-Welch").

See Also

Other utility functions: create_summary_statistics(), get_available_huggingface_files(), list_supported_indexes(), list_supported_types_ff(), list_supported_types_ff_legacy(), list_supported_types_macro_predictors(), list_supported_types_other(), list_supported_types_wrds(), list_tidy_finance_chapters(), open_tidy_finance_website(), trim(), validate_dates(), winsorize()

Examples

```
# List all supported types as a data frame
list_supported_types()

# Filter by domain
list_supported_types(domain = "WRDS")

# List supported types as a vector
list_supported_types(as_vector = TRUE)
```

`list_supported_types_ff`

List Supported Fama-French Dataset Types

Description

Returns a tibble with the supported Fama-French dataset types, including their names and frequencies (daily, weekly, monthly). Each dataset type is associated with a specific Fama-French model (e.g., 3 factors, 5 factors). Additionally, it annotates each dataset with the domain "Fama-French".

Usage

```
list_supported_types_ff()
```

Value

A tibble with columns: `type` (the type of dataset), `dataset_name` (a descriptive name of the dataset), and `domain` (the domain to which the dataset belongs, always "Fama-French").

See Also

Other utility functions: `create_summary_statistics()`, `get_available_huggingface_files()`, `list_supported_indexes()`, `list_supported_types()`, `list_supported_types_ff_legacy()`, `list_supported_types_macro_predictors()`, `list_supported_types_other()`, `list_supported_types_wrds()`, `list_tidy_finance_chapters()`, `open_tidy_finance_website()`, `trim()`, `validate_dates()`, `winsorize()`

list_supported_types_ff_legacy

List Supported Legacy Fama-French Dataset Types

Description

Returns a tibble with the legacy names of initially supported Fama-French dataset types, including their names and frequencies (daily, weekly, monthly). Each dataset type is associated with a specific Fama-French model (e.g., 3 factors, 5 factors). Additionally, it annotates each dataset with the domain "Fama-French". Not included in the exported `list_supported_types()` function.

Usage

```
list_supported_types_ff_legacy()
```

Value

A tibble with columns: `type` (the type of dataset), `dataset_name` (a descriptive name of the dataset), and `domain` (the domain to which the dataset belongs, always "Fama-French").

See Also

Other utility functions: `create_summary_statistics()`, `get_available_huggingface_files()`, `list_supported_indexes()`, `list_supported_types()`, `list_supported_types_ff()`, `list_supported_types_macro`
`list_supported_types_other()`, `list_supported_types_wrds()`, `list_tidy_finance_chapters()`, `open_tidy_finance_website()`, `trim()`, `validate_dates()`, `winsorize()`

list_supported_types_macro_predictors

List Supported Macro Predictor Dataset Types

Description

Returns a tibble with the supported macro predictor dataset types provided by Goyal-Welch, including their frequencies (monthly, quarterly, annual). All dataset types reference the same source file, "PredictorData2022.xlsx" for the year 2022. Additionally, it annotates each dataset with the domain "Goyal-Welch".

Usage

```
list_supported_types_macro_predictors()
```

Value

A tibble with columns: `type` (the type of dataset), `dataset_name` (the file name of the dataset, which is the same for all types), and `domain` (the domain to which the dataset belongs, always "Goyal-Welch").

See Also

Other utility functions: create_summary_statistics(), get_available_huggingface_files(), list_supported_indexes(), list_supported_types(), list_supported_types_ff(), list_supported_types_ff_le list_supported_types_other(), list_supported_types_wrds(), list_tidy_finance_chapters(), open_tidy_finance_website(), trim(), validate_dates(), winsorize()

---

list_supported_types_other

List Supported Other Data Types

Description

Returns a tibble listing the supported other data types and their corresponding dataset names.

Usage

```
list_supported_types_other()
```

Value

A tibble with columns: type (the type of dataset), dataset_name (the name of the data source), and domain (the domain to which the dataset belongs).

See Also

Other utility functions: create_summary_statistics(), get_available_huggingface_files(), list_supported_indexes(), list_supported_types(), list_supported_types_ff(), list_supported_types_ff_le list_supported_types_macro_predictors(), list_supported_types_wrds(), list_tidy_finance_chapters(), open_tidy_finance_website(), trim(), validate_dates(), winsorize()

---

list_supported_types_q

List Supported Global Q Dataset Types

Description

Returns a tibble with the supported Global Q dataset types, including their names and frequencies (daily, weekly, weekly week-to-week, monthly, quarterly, annual). Each dataset type is associated with the Global Q model, specifically the q5 factors model for the year 2023. Additionally, it annotates each dataset with the domain "Global Q".

Usage

```
list_supported_types_q()
```

Value

A tibble with columns: `type` (the type of dataset), `dataset_name` (the file name of the dataset), and `domain` (the domain to which the dataset belongs, always "Global Q").

`list_supported_types_wrds`

List Supported WRDS Dataset Types

Description

Returns a tibble with the supported dataset types provided via WRDS. Additionally, it annotates each dataset with the domain "WRDS".

Usage

```
list_supported_types_wrds()
```

Value

A tibble with columns: `type` (the type of dataset), `dataset_name` (the file name of the dataset), and `domain` (the domain to which the dataset belongs, always "WRDS").

See Also

Other utility functions: `create_summary_statistics()`, `get_available_huggingface_files()`, `list_supported_indexes()`, `list_supported_types()`, `list_supported_types_ff()`, `list_supported_types_ff_le`
`list_supported_types_macro_predictors()`, `list_supported_types_other()`, `list_tidy_finance_chapters()`, `open_tidy_finance_website()`, `trim()`, `validate_dates()`, `winsorize()`

`list_tidy_finance_chapters`

List Chapters of Tidy Finance

Description

Returns a character vector containing the names of the chapters available in the Tidy Finance resource. It provides a quick reference to the various topics covered.

Usage

```
list_tidy_finance_chapters()
```

Value

A character vector where each element is the name of a chapter available in the Tidy Finance resource. These names correspond to specific chapters in Tidy Finance with R.

See Also

Other utility functions: create_summary_statistics(), get_available_huggingface_files(), list_supported_indexes(), list_supported_types(), list_supported_types_ff(), list_supported_types_ff_le list_supported_types_macro_predictors(), list_supported_types_other(), list_supported_types_wrds(), open_tidy_finance_website(), trim(), validate_dates(), winsorize()

Examples

```
list_tidy_finance_chapters()
```

---

open_tidy_finance_website

Open Tidy Finance Website or Specific Chapter in Browser

Description

Opens the main Tidy Finance website or a specific chapter within the site in the user's default web browser. If a chapter is specified, the function constructs the URL to access the chapter directly.

Usage

```
open_tidy_finance_website(chapter = NULL)
```

Arguments

chapter	An optional character string specifying the chapter to open. If NULL (the default), the function opens the main page of Tidy Finance with R. If a chapter name is provided (e.g., "beta-estimation"), the function opens the corresponding chapter's page (e.g., "beta-estimation.html"). If the chapter name does not exist, then the function opens the main page.

Value

Invisible NULL. The function is called for its side effect of opening a web page.

See Also

Other utility functions: create_summary_statistics(), get_available_huggingface_files(), list_supported_indexes(), list_supported_types(), list_supported_types_ff(), list_supported_types_ff_le list_supported_types_macro_predictors(), list_supported_types_other(), list_supported_types_wrds(), list_tidy_finance_chapters(), trim(), validate_dates(), winsorize()

Examples

```
open_tidy_finance_website()
open_tidy_finance_website("beta-estimation")
```

portfolio_sort_options *Create Portfolio Sort Options*

Description

Creates a list of options of class `tidyfinance_portfolio_sort_options` that bundles sample construction filters and breakpoint specifications for use with `implement_portfolio_sort()`.

Usage

```
portfolio_sort_options(
  filter_options = NULL,
  breakpoint_options_main,
  breakpoint_options_secondary = NULL,
  ...
)
```

Arguments

`filter_options` A list of class `tidyfinance_filter_options` created by `filter_options()`, or `NULL` (the default, which applies no filters). The arguments accepted by `filter_options()` include

- `exclude_financials` A logical indicating whether to exclude financial firms (SIC codes 6000–6799). Defaults to `FALSE`.
- `exclude_utilities` A logical indicating whether to exclude utility firms (SIC codes 4900–4999). Defaults to `FALSE`.
- `min_stock_price` A single positive numeric specifying the minimum stock price required to include an observation. `NULL` (the default) applies no price filter.
- `min_size_quantile` A single numeric strictly between 0 and 1 specifying the minimum cross-sectional size quantile (based on lagged market cap) required to include an observation. `NULL` (the default) applies no size quantile filter.
- `min_listing_age` A single non-negative integer or numeric specifying the minimum number of months a stock must have been listed in CRSP. `NULL` (the default) applies no listing age filter.
- `exclude_negative_book_equity` A logical indicating whether to exclude observations with non-positive book equity. Defaults to `FALSE`.
- `exclude_negative_earnings` A logical indicating whether to exclude observations with non-positive earnings. Defaults to `FALSE`.

`breakpoint_options_main`
A list of class `tidyfinance_breakpoint_options` created by `breakpoint_options()`, specifying breakpoints for the primary sorting variable. The arguments accepted by `breakpoint_options()` include

- `n_portfolios` An optional integer specifying the number of equally sized portfolios to create. This parameter is mutually exclusive with `percentiles`.
- `percentiles` An optional numeric vector specifying the percentiles for determining the breakpoints of the portfolios. This parameter is mutually exclusive with `n_portfolios`.
- `breakpoints_exchanges` An optional character vector specifying exchange names to filter the data before computing breakpoints. Exchanges must be stored in a column given by `data_options` (defaults to `exchange`). If `NULL`, no filtering is applied.
- `smooth_bunching` An optional logical parameter specifying if to attempt smoothing non-extreme portfolios if the sorting variable bunches on the extremes (`TRUE`), or not (`FALSE`, the default).
- `breakpoints_min_size_threshold` An optional numeric value between 0 and 1 (exclusive). When set, stocks with market capitalization below this quantile are excluded from breakpoint computation.

`breakpoint_options_secondary`
: A list of class `tidyfinance_breakpoint_options` created by `breakpoint_options()`, specifying breakpoints for the secondary sorting variable, or `NULL` (the default) for univariate sorts. The arguments accepted by `breakpoint_options()` are the same as for `breakpoint_options_main`.

`...`
: Additional arguments to be included in the options list.

Value

A list of class `tidyfinance_portfolio_sort_options` containing the specified options.

See Also

Other portfolio functions: `assign_portfolio()`, `breakpoint_options()`, `compute_breakpoints()`, `compute_long_short_returns()`, `compute_portfolio_returns()`, `data_options()`, `filter_options()`, `filter_sorting_data()`, `implement_portfolio_sort()`

Examples

```
portfolio_sort_options(
  filter_options = filter_options(exclude_financials = TRUE),
  breakpoint_options_main = breakpoint_options(n_portfolios = 10)
)
```

`set_wrds_credentials` *Set WRDS Credentials*

Description

Prompts the user to input their WRDS (Wharton Research Data Services) username and password, and stores these credentials in a `.Renviron` file. The user can choose to store the `.Renviron` file in either the project directory or the home directory. If the `.Renviron` file already contains WRDS credentials, the user will be asked if they want to overwrite the existing credentials. Additionally, the user has the option to add the `.Renviron` file to the .gitignore file to prevent it from being tracked by version control.

Usage

```
set_wrds_credentials()
```

Value

Invisibly returns `TRUE`. Displays messages to the user based on their input and actions taken.

See Also

Other WRDS functions: `create_wrds_dummy_database()`, `disconnect_connection()`, `download_data_wrds()`, `download_data_wrds_ccm_links()`, `download_data_wrds_compustat()`, `download_data_wrds_crsp()`, `download_data_wrds_fisd()`, `download_data_wrds_trace_enhanced()`, `get_wrds_connection()`

Examples

```
## Not run:
set_wrds_credentials()

## End(Not run)
```

`trim` *Trim a Numeric Vector*

Description

Removes the values in a numeric vector that are beyond the specified quantiles, effectively trimming the distribution based on the `cut` parameter. This process reduces the vector's length by excluding extreme values from both tails of the distribution.

Usage

```
trim(x, cut)
```

Arguments

`x`	A numeric vector to be trimmed.
`cut`	The proportion of data to be trimmed from both ends of the distribution. For example, a `cut` of 0.05 will remove the lowest and highest 5% of the data. Must be in [0, 0.5].

Value

A numeric vector with the extreme values removed.

See Also

Other utility functions: create_summary_statistics(), get_available_huggingface_files(), list_supported_indexes(), list_supported_types(), list_supported_types_ff(), list_supported_types_ff_le list_supported_types_macro_predictors(), list_supported_types_other(), list_supported_types_wrds(), list_tidy_finance_chapters(), open_tidy_finance_website(), validate_dates(), winsorize()

Examples

```
set.seed(123)
data <- rnorm(100)
trimmed_data <- trim(x = data, cut = 0.05)
```

---

validate_dates *Validate and Coerce Date Range Arguments*

Description

Checks that start_date and end_date are a valid pair, coerces them to Date, and handles the case where both are NULL. When both are NULL and use_default_range = TRUE, a two-year default window ending one year ago is applied and reported to the user.

Usage

```
validate_dates(start_date, end_date, use_default_range = FALSE)
```

Arguments

start_date	A scalar coercible to Date via as.Date(), or NULL.
end_date	A scalar coercible to Date via as.Date(), or NULL.
use_default_range	A logical scalar. If TRUE and both date arguments are NULL, a default one-year range is used instead of returning NULL. Defaults to FALSE.

Value

A named list with elements start_date and end_date, both of class Date (or NULL when no dates are provided and use_default_range = FALSE).

See Also

Other utility functions: create_summary_statistics(), get_available_huggingface_files(), list_supported_indexes(), list_supported_types(), list_supported_types_ff(), list_supported_types_ff_le list_supported_types_macro_predictors(), list_supported_types_other(), list_supported_types_wrds(), list_tidy_finance_chapters(), open_tidy_finance_website(), trim(), winsorize()

`winsorize` *Winsorize a Numeric Vector*

Description

Replaces the values in a numeric vector that are beyond the specified quantiles with the boundary values of those quantiles. This is done for both tails of the distribution based on the `cut` parameter.

Usage

```
winsorize(x, cut)
```

Arguments

`x`	A numeric vector to be winsorized.
`cut`	The proportion of data to be winsorized from both ends of the distribution. For example, a `cut` of 0.05 will winsorize the lowest and highest 5% of the data. Must be in [0, 0.5].

Value

A numeric vector with the extreme values replaced by the corresponding quantile values.

See Also

Other utility functions: `create_summary_statistics()`, `get_available_huggingface_files()`, `list_supported_indexes()`, `list_supported_types()`, `list_supported_types_ff()`, `list_supported_types_ff_le` `list_supported_types_macro_predictors()`, `list_supported_types_other()`, `list_supported_types_wrds()`, `list_tidy_finance_chapters()`, `open_tidy_finance_website()`, `trim()`, `validate_dates()`

Examples

```
set.seed(123)
data <- rnorm(100)
winsorized_data <- winsorize(data, 0.05)
```

Index