# Package 'tidyfinance'

January 8, 2026

**Type** Package

**Title** Tidy Finance Helper Functions

**Version** 0.4.5

**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** cli, dplyr (>= 1.1.4), lifecycle, lubridate (>= 1.9.3), purrr (>= 1.0.2), rlang (>= 1.1.3), slider (>= 0.3.1), stats, tibble, tidyr (>= 1.3.1)

**Suggests** DBI (>= 1.2.2), dbplyr (>= 2.5.0), frenchdata (>= 0.2.0), furrr (>= 0.3.1), httr2 (>= 1.0.0), knitr, rmarkdown, RPostgres (>= 1.4.5), sandwich (>= 3.1-0), 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** 2026-01-08 16:50:02 UTC

# Contents

---

`add_lag_columns` *Add Lagged Versions of Columns to a Data Frame*

---

## Description

**[Experimental]**

This function adds lagged versions of specified columns to a data frame. Optionally, the operation can be grouped by another column and allows for flexible handling of missing values. The lag is applied based on the `date` column in the data frame.

## Usage

```
add_lag_columns(
  data,
  cols,
  by = NULL,
  lag,
  max_lag = lag,
  drop_na = TRUE,
  data_options = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame containing the columns to be lagged. |
| `cols` | A character vector specifying the names of the columns to lag. |
| `by` | An optional column by which to group the data when applying the lag. Default is `NULL`, meaning no grouping. |
| `lag` | The number of periods to lag the columns by. Must be non-negative. |
| `max_lag` | An optional maximum lag period. The default is equal to `lag`. |
| `drop_na` | A logical value indicating whether to drop rows with missing values in the lagged columns. Default is `TRUE`. |
| `data_options` | A list of additional options for data processing, such as the `date` column. If `NULL`, defaults are used. |

## Value

A data frame with lagged versions of the specified columns appended, optionally grouped by another column.

## Examples

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

# Add lagged columns for 'bm' and 'size' with a 3-month lag, grouped by 'permno'
data |>
  add_lag_columns(c("bm", "size"), lag = months(3), by = "permno")

# Introduce missing values in the data
data$bm[c(3, 5, 7, 15, 18)] <- NA
data$size[c(2, 4, 8, 13)] <- NA

# Add lagged columns with NA values removed
data |>
  add_lag_columns(c("bm", "size"), lag = months(3), by = permno)
```

---

# assign_portfolio *Assign Portfolios Based on Sorting Variable*

---

## Description

**[Experimental]**

This function 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 named list of data_options with characters, indicating the column names required to run this function. The required column names identify dates. Defaults to `date = date` and `id = permno`. |

## Value

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

## Examples

```
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), breakpoint_exchanges = c("NYSE"))
)
```

---

`breakpoint_options` *Create Breakpoint Options for Portfolio Sorting*

---

## Description

This function 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,
  breakpoint_exchanges = NULL,
  smooth_bunching = FALSE,
  ...
)
```

### 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 should be between 0 and 1. If not provided, defaults to NULL. |
| breakpoint_exchanges | Character, optional. A non-empty string specifying the exchange for which the breakpoints apply. If not provided, defaults to NULL. |
| smooth_bunching | Logical, optional. Indicates whether smooth bunching should be applied. Defaults to FALSE. |
| ... | 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 ....

### Examples

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

---

## check_supported_type *Check if a Dataset Type is Supported*

---

## Description

This function 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, it stops execution and returns an error message listing all supported types.

## Usage

```
check_supported_type(type)
```

## Arguments

| | |
|---|---|
| type | The dataset type to check for support. |

## 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]**

This function computes breakpoints based on a specified sorting. It can optionally filter the data by exchanges 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 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`.
- `breakpoint_exchanges` An optional character vector specifying exchange names to filter the data before computing breakpoints. Exchanges must be stored in a column named `exchange` in `data`. 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, the default), or not (FALSE). 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.

`data_options` A named list of data_options with characters, indicating the column names required to run this function. The required column names identify dates. Defaults to `exchange = exchange`.

## Value

A 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 if neither is provided. Ensure that you only use one of these parameters.

## Examples

```
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), breakpoint_exchanges = c("NYSE"))
 )
```

---

`compute_long_short_returns`

*Compute Long-Short Returns*

---

## Description

This function 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. The portfolio column should indicate different portfolios, and there should be columns for return measurements prefixed with "ret_excess". |
| `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 named list of data_options with characters, indicating the column names required to run this function. The required column names identify dates. Defaults to `date = date`, `portfolio = portfolio` and `ret_excess = ret_excess`. |

## 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.

## Examples

```
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

This function 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(
  sorting_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 = 0,
  data_options = NULL
)
```

## Arguments

| | |
|---|---|
| sorting_data | A data frame containing the dataset for portfolio assignment and return computation. Following CRSP naming conventions, the panel data must identify individual stocks with permno and the time point with date. It must contain columns for the sorting variables and ret_excess. Additionally, mktcap_lag is needed for value-weighted returns. |
| sorting_variables | A character vector specifying the column names in sorting_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:<br>• "univariate": For a single sorting variable.<br>• "bivariate-dependent": For two sorting variables, where the main sort depends on the secondary variable.<br>• "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_variable`) 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` for the main sorting variable.

`breakpoint_options_secondary`
: An optional named list of breakpoint_options passed to `breakpoint_function` for the secondary sorting variable.

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

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

`min_portfolio_size`
: An integer specifying the minimum number of portfolio constituents (default is set to `0`, effectively deactivating the check). Small portfolios' returns are set to zero.

`data_options`
: A named list of data_options with characters, indicating the column names required to run this function. The required column names identify dates, the stocks, and returns. Defaults to `date=date`, `id=permno`, and `ret_excess = ret_excess`.

## 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 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, 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 the `sorting_data` contains `mktcap_lag`)
- `ret_excess_ew`: The equal-weighted excess return of the portfolio.

## Note

Ensure that the `sorting_data` contains all the required columns: The specified sorting variables and `ret_excess`. The function will stop and throw an error if any required columns are missing.

## Examples

```
# 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-independent", 7,
  breakpoint_options_main = breakpoint_options(n_portfolios = 5),
  breakpoint_options_secondary = breakpoint_options(n_portfolios = 3),
)
```

---

# 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. This function 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 variabl (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 columns include the computed statistics.

---

## `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.

### Examples

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

---

## data_options *Create Data Options*

---

### Description

This function creates a list of data options used in financial data analysis, specifically for TidyFinance-related functions. It allows users to specify key parameters such as id, date, exchange, mktcap_lag, and ret_excess along with other additional options passed through ....

### Usage

```
data_options(
  id = "permno",
  date = "date",
  exchange = "exchange",
  mktcap_lag = "mktcap_lag",
  ret_excess = "ret_excess",
  portfolio = "portfolio",
  ...
)
```

### Arguments

| | |
|---|---|
| id | A character string representing the identifier variable (e.g., "permno"). |
| date | A character string representing the date variable (e.g., "date"). |
| exchange | A character string representing the exchange variable (e.g., "exchange"). |
| mktcap_lag | A character string representing the market capitalization lag variable (e.g., "mktcap_lag"). |
| ret_excess | A character string representing the excess return variable (e.g., "ret_excess"). |
| portfolio | A character string representing the portfolio variable (e.g., "portfolio"). |
| ... | Additional arguments to be included in the data options list. |

## Value

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

## Examples

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

---

`disconnection_connection` *Disconnect Database Connection*

---

## Description

This function safely disconnects an established database connection using the DBI package.

## Usage

```
disconnection_connection(con)
```

## Arguments

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

## Value

A logical value: `TRUE` if disconnection was successful, `FALSE` otherwise.

---

`download_data` *Download and Process Data Based on Type*

---

## Description

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

## Usage

```
download_data(type, start_date = NULL, end_date = NULL, ...)
```

### Arguments

| | |
|---|---|
| type | The type of dataset to download, indicating either factor data or macroeconomic predictors. |
| 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, dependening 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. |
| ... | Additional arguments passed to specific download functions depending on the type. For instance, if type is "constituents", this might include parameters specific to download_data_constituents. |

### Value

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

### Examples

```
download_data("factors_ff_3_monthly", "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_constituents

*Download Constituent Data*

---

### Description

This function 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 o ther ETFs.

### Value

A tibble with two 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.

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

### Examples

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

---

## `download_data_factors` *Download and Process Factor Data*

---

### Description

Downloads and processes factor data based on the specified type (Fama-French or Global Q), and date range. This function delegates to specific functions based on the type of factors requested: Fama-French or Global Q. It checks if the specified type is supported before proceeding with the download and processing.

### Usage

```
download_data_factors(type, start_date = NULL, end_date = NULL)
```

### Arguments

| | |
|---|---|
| `type` | The type of dataset to download, indicating the factor model and frequency. |
| `start_date` | The start date for filtering the data, in "YYYY-MM-DD" format. |
| `end_date` | The end date for filtering the data, in "YYYY-MM-DD" format. |

## Value

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

## Examples

```
download_data_factors("factors_ff_3_monthly", "2000-01-01", "2020-12-31")
download_data_factors("factors_ff_3_daily")
download_data_factors("factors_q5_daily", "2020-01-01", "2020-12-31")
```

---

`download_data_factors_ff`

*Download and Process Fama-French Factor Data*

---

## Description

Downloads and processes Fama-French factor data based on the specified type (e.g., "factors_ff_3_monthly"), and date range. The function first checks if the specified type is supported and 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(type, start_date = NULL, end_date = NULL)
```

## Arguments

| | |
|---|---|
| `type` | The type of dataset to download, corresponding to the specific Fama-French model and frequency. |
| `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

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.

## Examples

```
download_data_factors_ff("factors_ff_3_monthly", "2000-01-01", "2020-12-31")
download_data_factors_ff("factors_ff_10_industry_portfolios_monthly", "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 type (daily, monthly, etc.), date range, and source URL. The function first checks if the specified type is supported, identifies the dataset name from the supported types, then downloads and processes the data from the provided 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(
  type,
  start_date = NULL,
  end_date = NULL,
  url = "https://global-q.org/uploads/1/2/2/6/122679606/"
)
```

## Arguments

| | |
|---|---|
| `type` | The type of dataset to download (e.g., "factors_q5_daily", "factors_q5_monthly"). |
| `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. |
| `url` | The base URL from which to download the dataset files, with a specific path for Global Q datasets. |

## 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.

## Examples

```
download_data_factors_q("factors_q5_daily", "2020-01-01", "2020-12-31")
download_data_factors_q("factors_q5_annual")
```

`download_data_fred` *Download and Process Data from FRED*

## Description

This function 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 to download. |
| `start_date` | The start date for filtering the data, in "YYYY-MM-DD" format. |
| `end_date` | The end date for filtering the data, in "YYYY-MM-DD" format. |

## Details

This function 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 `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.

## Examples

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

`download_data_macro_predictors` *Download and Process Macro Predictor Data*

## Description

Downloads and processes macroeconomic predictor data based on the specified type (monthly, quarterly, or annual), date range, and source URL. The function first checks if the specified type is supported, then downloads the data from the provided URL (defaulting to 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(
  type,
  start_date = NULL,
  end_date = NULL,
  sheet_id = "1bM7vCWd3WOt95Sf9qjLPZjoiafgF_8EG"
)
```

## Arguments

| | |
|---|---|
| `type` | The type of dataset to download ("macro_predictors_monthly", "macro_predictors_quarterly", "macro_predictors_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. |
| `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.

## Examples

```
download_data_macro_predictors("macro_predictors_monthly")
```

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

### Description

This function 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 start 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.

### Examples

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

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

### 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

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

### Usage

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

### Arguments

| | |
|---|---|
| type | A string specifying the type of data to download. It should match one of the predefined patterns to indicate the dataset: "wrds_crsp" for CRSP data, "wrds_compustat" for Compustat data, or "wrds_ccm_links" for CCM links 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 subste of the dataset is returned. |

`...` Additional arguments passed to specific download functions depending on the `type`.

## Value

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

## Examples

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

## End(Not run)
```

---

# download_data_wrds_ccm_links

*Download CCM Links from WRDS*

---

## Description

This function 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.

## 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

This function downloads financial data from the WRDS Compustat database for a given type of financial data, 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 calculates book equity (be), operating profitability (op), and investment (inv) for each company.

## Usage

```
download_data_wrds_compustat(
  type,
  start_date = NULL,
  end_date = NULL,
  additional_columns = NULL
)
```

## Arguments

| | |
|---|---|
| `type` | The type of financial data 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. |
| `additional_columns` | Additional columns from the Compustat table as a character vector. |

## Value

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

## Examples

```
## Not run:
  download_data_wrds_compustat("wrds_compustat_annual", "2020-01-01", "2020-12-31")
  download_data_wrds_compustat("wrds_compustat_quarterly", "2020-01-01", "2020-12-31")

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

## End(Not run)
```

---

`download_data_wrds_crsp`

*Download Data from WRDS CRSP*

---

## Description

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

## Usage

```
download_data_wrds_crsp(
  type,
  start_date = NULL,
  end_date = NULL,
  batch_size = 500,
  version = "v2",
  additional_columns = NULL
)
```

## Arguments

| | |
|---|---|
| `type` | A string specifying the type of CRSP data 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. |
| `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. |

## 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 data type.

## Examples

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

  # Add additional columns
  download_data_wrds_crsp("wrds_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, SIC code of the issuer.

## 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).

## Examples

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

## End(Not run)
```

`estimate_betas` *Estimate Rolling Betas*

## Description

This function 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 tibble 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 formula representing the model to be estimated (e.g., `ret_excess ~ mkt_excess + smb + hml`). |
| `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 paralellization capabilities. Defaults to FALSE. |
| `data_options` | A named list of data_options with characters, indicating the column names required to run this function. The required column names identify dates and the stocks. Defaults to `date = date` and `id = permno`. |

## Value

A tibble with the estimated betas for each time period.

## 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

This function estimates Fama-MacBeth regressions 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
)
```

## 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 formula representing the regression model to be estimated in each cross-section. |
| 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 named list of data_options with characters, indicating the column names required to run this function. The required column names identify dates. Defaults to date = date. |

## Value

A data frame with the estimated risk premiums, the number of observations, standard errors, and t-statistics for each factor in the 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))

# 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 Model Coefficients*

## Description

**[Experimental]**

This function estimates the coefficients of 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. It returns the model's coefficients as either a numeric value (for a single independent variable) or a data frame (for multiple independent variables).

## Usage

```
estimate_model(data, model, min_obs = 1)
```

## Arguments

| | |
|---|---|
| `data` | A data frame containing the dependent variable and one or more independent variables. |
| `model` | A character that describes the model to estimate (e.g. `"ret_excess ~ mkt_excess + hmb + sml"`). |
| `min_obs` | The minimum number of observations required to estimate the model. Defaults to 1. |

## Value

A data frame with a row for each coefficient and column names corresponding to the independent variables.

## See Also

`stats::lm()` for details on the underlying linear model fitting used.

## Examples

```
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")
```

---

`get_random_user_agent` *Get a Random User Agent*

---

## Description

This internal function selects and returns a random user agent string from a predefined list. The list contains user agents for various operating systems and browsers, including Windows, macOS, Linux, Android, iPhone, Chrome, Safari, Firefox, and Edge.

## Usage

```
get_random_user_agent()
```

## Value

A character string representing a randomly selected user agent.

---

`get_wrds_connection` *Establish a Connection to the WRDS Database*

---

## Description

This function establishes a connection to the Wharton Research Data Services (WRDS) database using the `RPostgres` package. It requires that the `RPostgres` package is installed and that valid WRDS credentials are 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

Postgres, dbDisconnect for more information on managing database connections.

## 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)
```

---

# lag_column *Lag a Column Based on Date and Time Range*

---

## Description

**[Experimental]**

This function generates a lagged version of a given column based on a date variable, with the ability to specify a range of lags. It also allows for the optional removal of NA values.

## Usage

```
lag_column(column, date, lag, max_lag = lag, drop_na = TRUE)
```

## Arguments

| | |
|---|---|
| column | A numeric vector or column to be lagged. |
| date | A vector representing dates corresponding to the column. This should be in a date or datetime format. |
| lag | An integer specifying the minimum lag (in days, hours, etc.) to apply to column. |
| max_lag | An integer specifying the maximum lag (in days, hours, etc.) to apply to column. Defaults to lag. |
| drop_na | A logical value indicating whether to drop NA values from the resulting lagged column. Defaults to TRUE. |

## Value

A vector of the same length as column, containing the lagged values. If no matching dates are found within the lag window, NA is returned for that position.

## Examples

```
# Basic example with a vector
dates <- as.Date("2023-01-01") + 0:9
values <- rnorm(10)
lagged_values <- lag_column(values, dates, lag = 1, max_lag = 3)

# Example using a tibble and dplyr::group_by
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)
)

data |>
  dplyr::group_by(permno) |>
  dplyr::mutate(
    across(c(size, bm),
           \(x) lag_column(x, date, months(3), months(6), drop_na = TRUE))
  ) |>
dplyr::ungroup()
```

---

`list_supported_indexes` *List Supported Indexes*

---

## Description

This function 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.

## Examples

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

---

`list_supported_types` *List All Supported Dataset Types*

---

## Description

This function 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 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").

## 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

This function 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").

## list_supported_types_ff_legacy

*List Supported Legacy Fama-French Dataset Types*

### Description

This function 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").

## list_supported_types_macro_predictors

*List Supported Macro Predictor Dataset Types*

### Description

This function 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").

## 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` and `dataset_name`, where `type` indicates the code used to specify the data source and `dataset_name` provides the name of the data source.

## list_supported_types_q

*List Supported Global Q Dataset Types*

### Description

This function 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

This function 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").

## 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. This function 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.

### 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.

### Examples

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

---

## set_wrds_credentials *Set WRDS Credentials*

---

### Description

This function 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.

### 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 length of the vector, 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 between [0, 0.5]. |

### Value

A numeric vector with the extreme values removed.

### Examples

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

---

## 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 inside [0, 0.5]. |

### Value

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

### Examples

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

# Index