Package 'tidyquant'

February 13, 2025

Type Package

Title Tidy Quantitative Financial Analysis

Version 1.0.11

Description Bringing business and financial analysis to the 'tidyverse'. The 'tidyquant' package provides a convenient wrapper to various 'xts', 'zoo', 'quantmod', 'TTR' and 'PerformanceAnalytics' package
functions and returns the objects in the tidy 'tibble' format. The main advantage is being able to use quantitative functions with the 'tidyverse' functions including 'purrr', 'dplyr', 'tidyr', 'ggplot2', 'lubridate', etc. See the 'tidyquant' website for more information, documentation and examples.

URL https://business-science.github.io/tidyquant/, https://github.com/business-science/tidyquant

BugReports https://github.com/business-science/tidyquant/issues

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0),

Imports dplyr (>= 1.0.0), ggplot2 (>= 3.4.0), httr, httr2, curl, lazyeval, lubridate, magrittr, PerformanceAnalytics, RobStatTM, quantmod (>= 0.4-13), purrr, readr, readxl, stringr, tibble, tidyr (>= 1.0.0), timetk (>= 2.4.0), timeDate, TTR, xts, rlang, zoo, cli

Suggests alphavantager (>= 0.1.2), Quandl, riingo, tibbletime, broom, knitr, forcats, rmarkdown, testthat (>= 2.1.0), scales, Rblpapi, janitor

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Matt Dancho [aut, cre], Davis Vaughan [aut]

Maintainer Matt Dancho <mdancho@business-science.io>

Repository CRAN

Date/Publication 2025-02-13 05:30:02 UTC

Contents

`tidyquant-package` *tidyquant: Integrating quantitative financial analysis tools with the tidyverse*

Description

The main advantage of `tidyquant` is to bridge the gap between the best quantitative resources for collecting and manipulating quantitative data, `xts`, `quantmod` and `TTR`, and the data modeling workflow and infrastructure of the `tidyverse`.

Details

In this package, tidyquant functions and supporting data sets are provided to seamlessly combine tidy tools with existing quantitative analytics packages. The main advantage is being able to use tidy functions with purrr for mapping and tidyr for nesting to extend modeling to many stocks. See the tidyquant website for more information, documentation and examples.

Users will probably be interested in the following:

- **Getting Data from the Web:** tq_get()
- **Manipulating Data:** tq_transmute() and tq_mutate()
- **Performance Analysis and Portfolio Aggregation:** tq_performance() and tq_portfolio()

To learn more about tidyquant, start with the vignettes: browseVignettes(package = "tidyquant")

Author(s)

Maintainer: Matt Dancho <mdancho@business-science.io>

Authors:

- Davis Vaughan <dvaughan@business-science.io>

See Also

Useful links:

- https://business-science.github.io/tidyquant/
- https://github.com/business-science/tidyquant
- Report bugs at https://github.com/business-science/tidyquant/issues

---

av_api_key *Set Alpha Vantage API Key*

Description

Requires the alphavantager packager to use.

Usage

```
av_api_key(api_key)
```

Arguments

api_key	Optionally passed parameter to set Alpha Vantage api_key.

Details

A wrapper for alphavantager::av_api_key()

Value

Returns invisibly the currently set `api_key`

See Also

`tq_get()` `get = "alphavantager"`

Examples

```
## Not run:
if (rlang::is_installed("alphavantager")) {
av_api_key(api_key = "foobar")
}

## End(Not run)
```

`coord_x_date` *Zoom in on plot regions using date ranges or date-time ranges*

Description

Zoom in on plot regions using date ranges or date-time ranges

Usage

```
coord_x_date(xlim = NULL, ylim = NULL, expand = TRUE)

coord_x_datetime(xlim = NULL, ylim = NULL, expand = TRUE)
```

Arguments

`xlim`	Limits for the x axis, entered as character dates in "YYYY-MM-DD" format for date or "YYYY-MM-DD HH:MM:SS" for date-time.
`ylim`	Limits for the y axis, entered as values
`expand`	If `TRUE`, the default, adds a small expansion factor to the limits to ensure that data and axes don't overlap. If `FALSE`, limits are taken exactly from the data or `xlim`/`ylim`.

Details

The `coord_` functions prevent loss of data during zooming, which is necessary when zooming in on plots that calculate `stats` using data outside of the zoom range (e.g. when plotting moving averages with `geom_ma()`). Setting limits using `scale_x_date` changes the underlying data which causes moving averages to fail.

`coord_x_date` is a wrapper for `coord_cartesian` that enables quickly zooming in on plot regions using a date range.

`coord_x_datetime` is a wrapper for `coord_cartesian` that enables quickly zooming in on plot regions using a date-time range.

See Also

ggplot2::coord_cartesian()

Examples

```
# Load libraries
library(dplyr)
library(ggplot2)

# coord_x_date
AAPL <- tq_get("AAPL", from = "2013-01-01", to = "2016-12-31")
AAPL %>%
    ggplot(aes(x = date, y = adjusted)) +
    geom_line() +                          # Plot stock price
    geom_ma(n = 50) +                      # Plot 50-day Moving Average
    geom_ma(n = 200, color = "red") +      # Plot 200-day Moving Average
    # Zoom in
    coord_x_date(xlim = c("2016-01-01", "2016-12-31"))


# coord_x_datetime
time_index <- seq(from = as.POSIXct("2012-05-15 07:00"),
                  to   = as.POSIXct("2012-05-17 18:00"),
                  by   = "hour")
set.seed(1)
value <- rnorm(n = length(time_index))
hourly_data <- tibble(time.index = time_index,
                      value      = value)
hourly_data %>%
    ggplot(aes(x = time.index, y = value)) +
    geom_point() +
    coord_x_datetime(xlim = c("2012-05-15 07:00:00", "2012-05-15 16:00:00"))
```

---

deprecated *Deprecated functions*

Description

A record of functions that have been deprecated.

Usage

```
tq_transform(data, ohlc_fun = OHLCV, mutate_fun, col_rename = NULL, ...)

tq_transform_xy(data, x, y = NULL, mutate_fun, col_rename = NULL, ...)
```

Arguments

data	A tibble (tidy data frame) of data typically from tq_get().
ohlc_fun	Deprecated. Use select.
mutate_fun	The mutation function from either the xts, quantmod, or TTR package. Execute tq_mutate_fun_options() to see the full list of options by package.
col_rename	A string or character vector containing names that can be used to quickly rename columns.
...	Additional parameters passed to the appropriate mutatation function.
x, y	Parameters used with _xy that consist of column names of variables to be passed to the mutatation function (instead of OHLC functions).

Details

- tq_transform() - use tq_transmute()
- tq_transform_xy() - use tq_transmute_xy()
- as_xts() - use timetk::tk_xts()
- as_tibble() - use timetk::tk_tbl()
- summarise_by_time() - Moved to timetk package. Use timetk::summarise_by_time()

---

excel_date_functions *Excel Date and Time Functions*

Description

50+ date and time functions familiar to users coming from an **Excel Background.** The main benefits are:

1. Integration of the amazing lubridate package for handling dates and times
2. Integration of Holidays from timeDate and Business Calendars
3. New Date Math and Date Sequence Functions that factor in Business Calendars (e.g. EOMONTH(), NET_WORKDAYS())

These functions are designed to help users coming from an **Excel background**. Most functions replicate the behavior of Excel:

- Names in most cases match Excel function names
- Functionality replicates Excel
- By default, missing values are ignored (same as in Excel)

Usage

```
AS_DATE(x, ...)

AS_DATETIME(x, ...)

DATE(year, month, day)

DATEVALUE(x, ...)

YMD(x, ...)

MDY(x, ...)

DMY(x, ...)

YMD_HMS(x, ...)

MDY_HMS(x, ...)

DMY_HMS(x, ...)

YMD_HM(x, ...)

MDY_HM(x, ...)

DMY_HM(x, ...)

YMD_H(x, ...)

MDY_H(x, ...)

DMY_H(x, ...)

WEEKDAY(x, ..., label = FALSE, abbr = TRUE)

WDAY(x, ..., label = FALSE, abbr = TRUE)

DOW(x, ..., label = FALSE, abbr = TRUE)

MONTHDAY(x, ...)

MDAY(x, ...)

DOM(x, ...)

QUARTERDAY(x, ...)

QDAY(x, ...)
```

```
DAY(x, ...)

WEEKNUM(x, ...)

WEEK(x, ...)

WEEKNUM_ISO(x, ...)

MONTH(x, ..., label = FALSE, abbr = TRUE)

QUARTER(x, ..., include_year = FALSE, fiscal_start = 1)

YEAR(x, ...)

YEAR_ISO(x, ...)

DATE_TO_NUMERIC(x, ...)

DATE_TO_DECIMAL(x, ...)

SECOND(x, ...)

MINUTE(x, ...)

HOUR(x, ...)

NOW(...)

TODAY(...)

EOMONTH(start_date, months = 0)

EDATE(start_date, months = 0)

NET_WORKDAYS(start_date, end_date, remove_weekends = TRUE, holidays = NULL)

COUNT_DAYS(start_date, end_date)

YEARFRAC(start_date, end_date)

DATE_SEQUENCE(start_date, end_date, by = "day")

WORKDAY_SEQUENCE(start_date, end_date, remove_weekends = TRUE, holidays = NULL)

HOLIDAY_SEQUENCE(
  start_date,
  end_date,
```

```
  calendar = c("NYSE", "LONDON", "NERC", "TSX", "ZURICH")
)

HOLIDAY_TABLE(years, pattern = ".")

FLOOR_DATE(x, ..., by = "day")

FLOOR_DAY(x, ...)

FLOOR_WEEK(x, ...)

FLOOR_MONTH(x, ...)

FLOOR_QUARTER(x, ...)

FLOOR_YEAR(x, ...)

CEILING_DATE(x, ..., by = "day")

CEILING_DAY(x, ...)

CEILING_WEEK(x, ...)

CEILING_MONTH(x, ...)

CEILING_QUARTER(x, ...)

CEILING_YEAR(x, ...)

ROUND_DATE(x, ..., by = "day")

ROUND_DAY(x, ...)

ROUND_WEEK(x, ...)

ROUND_MONTH(x, ...)

ROUND_QUARTER(x, ...)

ROUND_YEAR(x, ...)
```

Arguments

x	A vector of date or date-time objects
...	Parameters passed to underlying `lubridate` functions.
year	Used in DATE()
month	Used in DATE()
day	Used in DATE()

label	A logical used for MONTH() and WEEKDAY() Date Extractors to decide whether or not to return names (as ordered factors) or numeric values.
abbr	A logical used for MONTH() and WEEKDAY(). If label = TRUE, used to determine if full names (e.g. Wednesday) or abbreviated names (e.g. Wed) should be returned.
include_year	A logical value used in QUARTER(). Determines whether or not to return 2020 Q3 as 3 or 2020.3.
fiscal_start	A numeric value used in QUARTER(). Determines the fiscal-year starting quarter.
start_date	Used in Date Math and Date Sequence operations. The starting date in the calculation.
months	Used to offset months in EOMONTH() AND EDATE() Date Math calculations
end_date	Used in Date Math and Date Sequence operations. The ending date in the calculation.
remove_weekends	A logical value used in Date Sequence and Date Math calculations. Indicates whether or not weekends should be removed from the calculation.
holidays	A vector of dates corresponding to holidays that should be removed from the calculation.
by	Used to determine the gap in Date Sequence calculations and value to round to in Date Collapsing operations. Acceptable values are: A character string, containing one of "day", "week", "month", "quarter" or "year".
calendar	The calendar to be used in Date Sequence calculations for Holidays from the timeDate package. Acceptable values are: "NYSE", "LONDON", "NERC", "TSX", "ZURICH"
years	A numeric vector of years to return Holidays for in HOLIDAY_TABLE()
pattern	Used to filter Holidays (e.g. pattern = "Easter"). A "regular expression" filtering pattern.

Details

Converters - Make date and date-time from text (character data)

- General String-to-Date Conversion: AS_DATE(), AS_DATETIME()
- Format-Specific String-to-Date Conversion: YMD() (YYYY-MM-DD), MDY() (MM-DD-YYYY), DMY() (DD-MM-YYYY)
- Hour-Minute-Second Conversion: YMD_HMS(), YMD_HM(), and friends.

Extractors - Returns information from a time-stamp.

- Extractors: SECOND(), MINUTE(), HOUR(), DAY(), WEEK(), MONTH(), QUARTER(), YEAR()

Current Time - Returns the current date/date-time based on your locale.

- NOW(), TODAY()

Date Math - Perform popular Excel date calculations

- EOMONTH() - End of Month
- NET_WORKDAYS(), COUNT_DAYS() - Return number of days between 2 dates factoring in working days and holidays
- YEARFRAC() - Return the fractional period of the year that has been completed between 2 dates.

Date Sequences - Return a vector of dates or a Holiday Table (tibble).

- DATE_SEQUENCE(), WORKDAY_SEQUENCE(), HOLIDAY_SEQUENCE - Return a sequence of dates between 2 dates that factor in workdays and timeDate holiday calendars for popular business calendars including NYSE and London stock exchange.

Date Collapsers - Collapse a date sequence (useful in dplyr::group_by() and pivot_table())

- FLOOR_DATE(), FLOOR_DAY(), FLOOR_WEEK(), FLOOR_MONTH(), FLOOR_QUARTER(), FLOOR_YEAR()
- Similar functions exist for CEILING and ROUND. These are wrappers for lubridate functions.

Value

- **Converters** - Date or date-time object the length of x
- **Extractors** - Returns information from a time-stamp.
- **Current Time** - Returns the current date/date-time based on your locale.
- **Date Math** - Numeric values or Date Values depending on the calculation.
- **Date Sequences** - Return a vector of dates or a Holiday Table (tibble).
- **Date Collapsers** - Date or date-time object the length of x

Examples

```
# Libraries
library(lubridate)

# --- Basic Usage ----

# Converters ---
AS_DATE("2011 Jan-01") # General
YMD("2011 Jan-01")     # Year, Month-Day Format
MDY("01-02-20")        # Month-Day, Year Format (January 2nd, 2020)
DMY("01-02-20")        # Day-Month, Year Format (February 1st, 2020)

# Extractors ---
WEEKDAY("2020-01-01")                                  # Labelled Day
WEEKDAY("2020-01-01", label = FALSE)                   # Numeric Day
WEEKDAY("2020-01-01", label = FALSE, week_start = 1)   # Start at 1 (Monday) vs 7 (Sunday)
MONTH("2020-01-01")
QUARTER("2020-01-01")
YEAR("2020-01-01")

# Current Date-Time ---
NOW()
```

```
TODAY()

# Date Math ---
EOMONTH("2020-01-01")
EOMONTH("2020-01-01", months = 1)
NET_WORKDAYS("2020-01-01", "2020-07-01") # 131 Skipping Weekends
NET_WORKDAYS("2020-01-01", "2020-07-01",
             holidays = HOLIDAY_SEQUENCE("2020-01-01", "2020-07-01",
                                         calendar = "NYSE")) # 126 Skipping 5 NYSE Holidays

# Date Sequences ---
DATE_SEQUENCE("2020-01-01", "2020-07-01")
WORKDAY_SEQUENCE("2020-01-01", "2020-07-01")
HOLIDAY_SEQUENCE("2020-01-01", "2020-07-01", calendar = "NYSE")
WORKDAY_SEQUENCE("2020-01-01", "2020-07-01",
                 holidays = HOLIDAY_SEQUENCE("2020-01-01", "2020-07-01",
                                             calendar = "NYSE"))

# Date Collapsers ---
FLOOR_DATE(AS_DATE("2020-01-15"), by = "month")
CEILING_DATE(AS_DATE("2020-01-15"), by = "month")
CEILING_DATE(AS_DATE("2020-01-15"), by = "month") - ddays(1) # EOMONTH using lubridate

# --- Usage with tidyverse ---

# Calculate returns by symbol/year/quarter
FANG %>%
    pivot_table(
        .rows       = c(symbol, ~ QUARTER(date)),
        .columns    = ~ YEAR(date),
        .values     = ~ PCT_CHANGE_FIRSTLAST(adjusted)
    )
```

`excel_financial_math_functions` *Excel Financial Math Functions*

Description

Excel financial math functions are designed to easily calculate Net Present Value (`NPV()`), Future Value of cashflow (`FV()`), Present Value of future cashflow (`PV()`), and more.

These functions are designed to help users coming from an **Excel background**. Most functions replicate the behavior of Excel:

- Names are similar to Excel function names
- By default, missing values are ignored (same as in Excel)

Usage

```
NPV(cashflow, rate, nper = NULL)

IRR(cashflow)

FV(rate, nper, pv = 0, pmt = 0, type = 0)

PV(rate, nper, fv = 0, pmt = 0, type = 0)

PMT(rate, nper, pv, fv = 0, type = 0)

RATE(nper, pmt, pv, fv = 0, type = 0)
```

Arguments

`cashflow`	Cash flow values. When one value is provided, it's assumed constant cash flow.
`rate`	One or more rate. When one rate is provided it's assumed constant rate.
`nper`	Number of periods. When 'nper" is provided, the cashflow values and rate are assumed constant.
`pv`	Present value. Initial investments (cash inflows) are typically a negative value.
`pmt`	Number of payments per period.
`type`	Should payments (`pmt`) occur at the beginning (`type = 0`) or the end (`type = 1`) of each period.
`fv`	Future value. Cash outflows are typically a positive value.

Details

Net Present Value (NPV) Net present value (NPV) is the difference between the present value of cash inflows and the present value of cash outflows over a period of time. NPV is used in capital budgeting and investment planning to analyze the profitability of a projected investment or project. For more information, see Investopedia NPV.

Internal Rate of Return (IRR) The internal rate of return (IRR) is a metric used in capital budgeting to estimate the profitability of potential investments. The internal rate of return is a discount rate that makes the net present value (NPV) of all cash flows from a particular project equal to zero. IRR calculations rely on the same formula as NPV does. For more information, see Investopedia IRR.

Future Value (FV) Future value (FV) is the value of a current asset at a future date based on an assumed rate of growth. The future value (FV) is important to investors and financial planners as they use it to estimate how much an investment made today will be worth in the future. Knowing the future value enables investors to make sound investment decisions based on their anticipated needs. However, external economic factors, such as inflation, can adversely affect the future value of the asset by eroding its value. For more information, see Investopedia FV.

Present Value (PV) Present value (PV) is the current value of a future sum of money or stream of cash flows given a specified rate of return. Future cash flows are discounted at the discount rate, and the higher the discount rate, the lower the present value of the future cash flows. Determining the

appropriate discount rate is the key to properly valuing future cash flows, whether they be earnings or obligations. For more information, see Investopedia PV.

Payment (PMT) The Payment `PMT()` function calculates the payment for a loan based on constant payments and a constant interest rate.

Rate (RATE) Returns the interest rate per period of a loan or an investment. For example, use 6%/4 for quarterly payments at 6% APR.

Value

- Summary functions return a single value

Examples

```
NPV(c(-1000, 250, 350, 450, 450), rate = 0.05)

IRR(c(-1000, 250, 350, 450, 450))

FV(rate = 0.05, nper = 5, pv = -100, pmt = 0, type = 0)

PV(rate = 0.05, nper = 5, fv = -100, pmt = 0, type = 0)

PMT(nper = 20, rate = 0.05, pv = -100, fv = 0, type = 0)

RATE(nper = 20, pmt = 8, pv = -100, fv = 0, type = 0)
```

---

excel_if_functions *Excel Summarising "If" Functions*

Description

"IFS" functions are filtering versions of their summarization counterparts. Simply add "cases" that filter if a condition is true. Multiple cases are evaluated as "AND" filtering operations. A single case with | ("OR") bars can be created to accomplish an "OR". See details below.

These functions are designed to help users coming from an **Excel background**. Most functions replicate the behavior of Excel:

- Names are similar to Excel function names
- By default, missing values are ignored (same as in Excel)

Usage

```
SUM_IFS(x, ...)

COUNT_IFS(x, ...)

AVERAGE_IFS(x, ...)
```

```
MEDIAN_IFS(x, ...)

MIN_IFS(x, ...)

MAX_IFS(x, ...)

CREATE_IFS(.f, ...)
```

Arguments

x	A vector. Most functions are designed for numeric data. Some functions like COUNT_IFS() handle multiple data types.
...	Add cases to evaluate. See Details.
.f	A function to convert to an "IFS" function. Use ... in this case to provide parameters to the .f like na.rm = TRUE.

Details

"AND" Filtering: Multiple cases are evaluated as "AND" filtering operations.

"OR" Filtering: Compound single cases with | ("OR") bars can be created to accomplish an "OR". Simply use a statement like x > 10 | x < -10 to perform an "OR" if-statement.

Creating New "Summarizing IFS" Functions: Users can create new "IFS" functions using the CREATE_IFS() function factory. The only requirement is that the output of your function (.f) must be a single value (scalar). See examples below.

Value

- **Summary functions** return a single value

Useful Functions

Summary Functions - Return a single value from a vector

- Sum: SUM_IFS()
- Center: AVERAGE_IFS(), MEDIAN_IFS()
- Count: COUNT_IFS()
- Range: MIN_IFS(), MAX_IFS()

Create your own summary "IFS" function

- CREATE_IFS(): This is a function factory that generates summary "_IFS" functions.

Examples

```
library(dplyr)
library(timetk, exclude = "FANG")
library(stringr)
library(lubridate)

# --- Basic Usage ---

SUM_IFS(x = 1:10, x > 5)

COUNT_IFS(x = letters, str_detect(x, "a|b|c"))

SUM_IFS(-10:10, x > 8 | x < -5)

# Create your own IFS function (Mind blowingly simple)!
Q75_IFS <- CREATE_IFS(.f = quantile, probs = 0.75, na.rm = TRUE)
Q75_IFS(1:10, x > 5)

# --- Usage with tidyverse ---

# Using multiple cases IFS cases to count the frequency of days with
# high trade volume in a given year
FANG %>%
    group_by(symbol) %>%
    summarise(
        high_volume_in_2015 = COUNT_IFS(volume,
                                        year(date) == 2015,
                                        volume > quantile(volume, 0.75))
    )

# Count negative returns by month
FANG %>%
    mutate(symbol = forcats::as_factor(symbol)) %>%
    group_by(symbol) %>%

    # Collapse from daily to FIRST value by month
    summarise_by_time(
        .date_var  = date,
        .by        = "month",
        adjusted   = FIRST(adjusted)
    ) %>%

    # Calculate monthly returns
    group_by(symbol) %>%
    mutate(
        returns = PCT_CHANGE(adjusted, fill_na = 0)
    ) %>%

    # Find returns less than zero and count the frequency
    summarise(
        negative_monthly_returns = COUNT_IFS(returns, returns < 0)
    )
```

`excel_pivot_table` *Excel Pivot Table*

Description

The Pivot Table is one of Excel's most powerful features, and now it's available in R! A pivot table is a table of statistics that summarizes the data of a more extensive table (such as from a database, spreadsheet, or business intelligence program).

These functions are designed to help users coming from an **Excel background**. Most functions replicate the behavior of Excel:

- Names are similar to Excel function names
- Functionality replicates Excel

Usage

```
pivot_table(
  .data,
  .rows,
  .columns,
  .values,
  .filters = NULL,
  .sort = NULL,
  fill_na = NA
)
```

Arguments

`.data`	A `data.frame` or `tibble` that contains data to summarize with a pivot table
`.rows`	Enter one or more groups to assess as expressions (e.g. `~ MONTH(date_column)`)
`.columns`	Enter one or more groups to assess expressions (e.g. `~ YEAR(date_column)`)
`.values`	Numeric only. Enter one or more summarization expression(s) (e.g. `~ SUM(value_column)`)
`.filters`	This argument is not yet in use
`.sort`	This argument is not yet in use
`fill_na`	A value to replace missing values with. Default is `NA`

Details

This summary might include sums, averages, or other statistics, which the pivot table groups together in a meaningful way.

The key parameters are:

- `.rows` - These are groups that will appear as row-wise headings for the summarization, You can modify these groups by applying collapsing functions (e.g. (`YEAR()`).

- `.columns` - These are groups that will appear as column headings for the summarization. You can modify these groups by applying collapsing functions (e.g. (`YEAR()`).
- `.values` - These are numeric data that are summarized using a summary function (e.g. `SUM()`, `AVERAGE()`, `COUNT()`, `FIRST()`, `LAST()`, `SUM_IFS()`, `AVERAGE_IFS()`, `COUNT_IFS()`)

R implementation details.

- The `pivot_table()` function is powered by the `tidyverse`, an ecosystem of packages designed to manipulate data.
- All of the key parameters can be expressed using a functional form:
  - Rows and Column Groupings can be collapsed. Example: `.columns = ~ YEAR(order_date)`
  - Values can be summarized provided a single value is returned. Example: `.values = ~ SUM_IFS(order_volume >= quantile(order_volume, probs = 0.75))`
  - Summarizations and Row/Column Groupings can be stacked (combined) with `c()`. Example: `.rows = c(~ YEAR(order_date), company)`
  - Bare columns (e.g. `company`) don not need to be prefixed with the `~`.
  - **All grouping and summarizing functions MUST BE prefixed with `~`**. Example: `.rows = ~ YEAR(order_date)`

Value

Returns a tibble that has been pivoted to summarize information by column and row groupings

Examples

```
# PIVOT TABLE ----
# Calculate returns by year/quarter
FANG %>%
    pivot_table(
        .rows       = c(symbol, ~ QUARTER(date)),
        .columns    = ~ YEAR(date),
        .values     = ~ PCT_CHANGE_FIRSTLAST(adjusted)
    )
```

`excel_ref_functions` *Excel Reference Functions*

Description

Excel reference functions are used to efficiently lookup values from a data source. The most popular lookup function is "VLOOKUP", which has been implemented in R.

These functions are designed to help users coming from an **Excel background**. Most functions replicate the behavior of Excel:

- Names are similar to Excel function names
- Functionality replicates Excel

Usage

```
VLOOKUP(.lookup_values, .data, .lookup_column, .return_column)
```

Arguments

`.lookup_values`	One or more lookup values.
`.data`	A `data.frame` or `tibble` that contains values to evaluate and return
`.lookup_column`	The column in `.data` containing exact matching values of the `.lookup_values`
`.return_column`	The column in `.data` containing the values to return if a match is found

Details

`VLOOKUP()` **Details**

- Performs exact matching only. Fuzzy matching is not implemented.
- Can only return values from one column only. Use `dplyr::left_join()` to perform table joining.

Value

Returns a vector the length of the input lookup values

Examples

```
library(dplyr)

lookup_table <- tibble(
    stock   = c("META", "AMZN", "NFLX", "GOOG"),
    company = c("Facebook", "Amazon", "Netflix", "Google")
)

# --- Basic Usage ---

VLOOKUP("NFLX",
        .data = lookup_table,
        .lookup_column = stock,
        .return_column = company)

# --- Usage with tidyverse ---

# Add company names to the stock data
FANG %>%
    mutate(company = VLOOKUP(symbol, lookup_table, stock, company))
```

`excel_stat_mutation_functions` *Excel Statistical Mutation Functions*

Description

15+ common statistical functions familiar to users of Excel (e.g. `ABS()`, `SQRT()`) that **modify / transform** a series of values (i.e. a vector of the same length of the input is returned).

These functions are designed to help users coming from an **Excel background**. Most functions replicate the behavior of Excel:

- Names in most cases match Excel function names
- Functionality replicates Excel
- By default, missing values are ignored (same as in Excel)

Usage

```
ABS(x)

SQRT(x)

LOG(x)

EXP(x)

RETURN(x, n = 1, fill_na = NA)

PCT_CHANGE(x, n = 1, fill_na = NA)

CHANGE(x, n = 1, fill_na = NA)

LAG(x, n = 1, fill_na = NA)

LEAD(x, n = 1, fill_na = NA)

CUMULATIVE_SUM(x)

CUMULATIVE_PRODUCT(x)

CUMULATIVE_MAX(x)

CUMULATIVE_MIN(x)

CUMULATIVE_MEAN(x)

CUMULATIVE_MEDIAN(x)
```

Arguments

x	A vector. Most functions are designed for numeric data.
n	Values to offset. Used in functions like LAG(), LEAD(), and PCT_CHANGE()
fill_na	Fill missing (NA) values with a different value. Used in offsetting functions.

Value

- **Mutation functions** return a mutated / transformed version of the vector

Useful functions

Mutation Functions - Transforms a vector

- Transformation: ABS(), SQRT(), LOG(), EXP()
- Lags & Change (Offsetting Functions): CHANGE(), PCT_CHANGE(), LAG(), LEAD()
- Cumulative Totals: CUMULATIVE_SUM(), CUMULATIVE_PRODUCT()

Examples

```
# Libraries
library(timetk, exclude = "FANG")
library(dplyr)

# --- Basic Usage ----

CUMULATIVE_SUM(1:10)

PCT_CHANGE(c(21, 24, 22, 25), fill_na = 0)

# --- Usage with tidyverse ---

# Go from daily to monthly periodicity,
# then calculate returns and growth of $1 USD
FANG %>%
    mutate(symbol = forcats::as_factor(symbol)) %>%
    group_by(symbol) %>%

    # Summarization - Collapse from daily to FIRST value by month
    summarise_by_time(
        .date_var  = date,
        .by        = "month",
        adjusted   = FIRST(adjusted)
    ) %>%

    # Mutation - Calculate monthly returns and cumulative growth of $1 USD
    group_by(symbol) %>%
    mutate(
        returns = PCT_CHANGE(adjusted, fill_na = 0),
        growth  = CUMULATIVE_SUM(returns) + 1
    )
```

`excel_stat_summary_functions` *Excel Statistical Summary Functions*

Description

15+ common statistical functions familiar to users of Excel (e.g. `SUM()`, `AVERAGE()`). These functions return a **single value** (i.e. a vector of length 1).

These functions are designed to help users coming from an **Excel background**. Most functions replicate the behavior of Excel:

- Names in most cases match Excel function names
- Functionality replicates Excel
- By default, missing values are ignored (same as in Excel)

Usage

```
SUM(x)

AVERAGE(x)

MEDIAN(x)

MIN(x)

MAX(x)

COUNT(x)

COUNT_UNIQUE(x)

STDEV(x)

VAR(x)

COR(x, y)

COV(x, y)

FIRST(x)

LAST(x)

NTH(x, n = 1)

CHANGE_FIRSTLAST(x)
```

```
PCT_CHANGE_FIRSTLAST(x)
```

Arguments

x	A vector. Most functions are designed for numeric data. Some functions like COUNT() handle multiple data types.
y	A vector. Used in functions requiring 2 inputs.
n	A single value used in NTH() to select a specific element location to return.

Details

Summary Functions

- All functions remove missing values (NA). This is the same behavior as in Excel and most commonly what is desired.

Value

- **Summary functions** return a single value

Useful functions

Summary Functions - Return a single value from a vector

- Sum: SUM()
- Center: AVERAGE(), MEDIAN()
- Spread: STDEV(), VAR()
- Range: MIN(), MAX()
- Count: COUNT(), COUNT_UNIQUE()
- Position: FIRST(), LAST(), NTH()
- Change (Summary): CHANGE_FIRSTLAST(), PCT_CHANGE_FIRSTLAST()
- Correlation: COR(), COV()

Examples

```
# Libraries
library(timetk, exclude = "FANG")
library(forcats)
library(dplyr)

# --- Basic Usage ----

SUM(1:10)

PCT_CHANGE_FIRSTLAST(c(21, 24, 22, 25))

# --- Usage with tidyverse ---
```

```
# Go from daily to monthly periodicity,
# then calculate returns and growth of $1 USD
FANG %>%
    mutate(symbol = forcats::as_factor(symbol)) %>%
    group_by(symbol) %>%

    # Summarization - Collapse from daily to FIRST value by month
    summarise_by_time(
        .date_var  = date,
        .by        = "month",
        adjusted   = FIRST(adjusted)
    )
```

FANG *Stock prices for the "FANG" stocks.*

Description

A dataset containing the daily historical stock prices for the "FANG" tech stocks, "META", "AMZN", "NFLX", and "GOOG", spanning from the beginning of 2013 through the end of 2016.

Usage

```
FANG
```

Format

A "tibble" ("tidy" data frame) with 4,032 rows and 8 variables:

symbol stock ticker symbol

date trade date

open stock price at the open of trading, in USD

high stock price at the highest point during trading, in USD

low stock price at the lowest point during trading, in USD

close stock price at the close of trading, in USD

volume number of shares traded

adjusted stock price at the close of trading adjusted for stock splits, in USD

Source

https://www.investopedia.com/terms/f/fang-stocks-fb-amzn.asp

geom_bbands *Plot Bollinger Bands using Moving Averages*

Description

Bollinger Bands plot a range around a moving average typically two standard deviations up and down. The `geom_bbands()` function enables plotting Bollinger Bands quickly using various moving average functions. The moving average functions used are specified in `TTR::SMA()` from the TTR package. Use `coord_x_date()` to zoom into specific plot regions. The following moving averages are available:

- **Simple moving averages (SMA)**: Rolling mean over a period defined by `n`.
- **Exponential moving averages (EMA)**: Includes exponentially-weighted mean that gives more weight to recent observations. Uses `wilder` and `ratio` args.
- **Weighted moving averages (WMA)**: Uses a set of weights, `wts`, to weight observations in the moving average.
- **Double exponential moving averages (DEMA)**: Uses `v` volume factor, `wilder` and `ratio` args.
- **Zero-lag exponential moving averages (ZLEMA)**: Uses `wilder` and `ratio` args.
- **Volume-weighted moving averages (VWMA)**: Requires `volume` aesthetic.
- **Elastic, volume-weighted moving averages (EVWMA)**: Requires `volume` aesthetic.

Usage

```
geom_bbands(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  ma_fun = SMA,
  n = 20,
  sd = 2,
  wilder = FALSE,
  ratio = NULL,
  v = 1,
  wts = 1:n,
  color_ma = "darkblue",
  color_bands = "red",
  alpha = 0.15,
  fill = "grey20",
  ...
)
```

```
geom_bbands_(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  ma_fun = "SMA",
  n = 10,
  sd = 2,
  wilder = FALSE,
  ratio = NULL,
  v = 1,
  wts = 1:n,
  color_ma = "darkblue",
  color_bands = "red",
  alpha = 0.15,
  fill = "grey20",
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes() or ggplot2::aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot2::ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See ggplot2::fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
na.rm	If TRUE, silently removes NA values, which typically desired for moving averages.

show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behavior from the default plot specification, e.g. ggplot2::borders().
ma_fun	The function used to calculate the moving average. Seven options are available including: SMA, EMA, WMA, DEMA, ZLEMA, VWMA, and EVWMA. The default is SMA. See TTR::SMA() for underlying functions.
n	Number of periods to average over. Must be between 1 and nrow(x), inclusive.
sd	The number of standard deviations to use.
wilder	logical; if TRUE, a Welles Wilder type EMA will be calculated; see notes.
ratio	A smoothing/decay ratio. ratio overrides wilder in EMA.
v	The 'volume factor' (a number in [0,1]). See Notes.
wts	Vector of weights. Length of wts vector must equal the length of x, or n (the default).
color_ma, color_bands	Select the line color to be applied for the moving average line and the Bollinger band line.
alpha	Used to adjust the alpha transparency for the BBand ribbon.
fill	Used to adjust the fill color for the BBand ribbon.
...	Other arguments passed on to ggplot2::layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

The following aesthetics are understood (required are in bold):

- x, Typically a date
- high, Required to be the high price
- low, Required to be the low price
- close, Required to be the close price
- volume, Required for VWMA and EVWMA
- colour, Affects line colors
- fill, Affects ribbon fill color
- alpha, Affects ribbon alpha value
- group
- linetype
- size

See Also

See individual modeling functions for underlying parameters:

- `TTR::SMA()` for simple moving averages
- `TTR::EMA()` for exponential moving averages
- `TTR::WMA()` for weighted moving averages
- `TTR::DEMA()` for double exponential moving averages
- `TTR::ZLEMA()` for zero-lag exponential moving averages
- `TTR::VWMA()` for volume-weighted moving averages
- `TTR::EVWMA()` for elastic, volume-weighted moving averages
- `coord_x_date()` for zooming into specific regions of a plot

Examples

```
library(dplyr)
library(ggplot2)
library(lubridate)

AAPL <- tq_get("AAPL", from = "2013-01-01", to = "2016-12-31")

# SMA
AAPL %>%
    ggplot(aes(x = date, y = close)) +
    geom_line() +           # Plot stock price
    geom_bbands(aes(high = high, low = low, close = close), ma_fun = SMA, n = 50) +
    coord_x_date(xlim = c(as_date("2016-12-31") - dyears(1), as_date("2016-12-31")),
                 ylim = c(20, 35))


# EMA
AAPL %>%
   ggplot(aes(x = date, y = close)) +
   geom_line() +           # Plot stock price
   geom_bbands(aes(high = high, low = low, close = close),
                  ma_fun = EMA, wilder = TRUE, ratio = NULL, n = 50) +
   coord_x_date(xlim = c(as_date("2016-12-31") - dyears(1), as_date("2016-12-31")),
                ylim = c(20, 35))


# VWMA
AAPL %>%
    ggplot(aes(x = date, y = close)) +
    geom_line() +           # Plot stock price
    geom_bbands(aes(high = high, low = low, close = close, volume = volume),
                   ma_fun = VWMA, n = 50) +
    coord_x_date(xlim = c(as_date("2016-12-31") - dyears(1), as_date("2016-12-31")),
                ylim = c(20, 35))
```

geom_chart *Plot Financial Charts in ggplot2*

Description

Financial charts provide visual cues to open, high, low, and close prices. Use `coord_x_date()` to zoom into specific plot regions. The following financial chart geoms are available:

- **Bar Chart**
- **Candlestick Chart**

Usage

```
geom_barchart(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  colour_up = "darkblue",
  colour_down = "red",
  fill_up = "darkblue",
  fill_down = "red",
  ...
)

geom_candlestick(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  colour_up = "darkblue",
  colour_down = "red",
  fill_up = "darkblue",
  fill_down = "red",
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by `ggplot2::aes()` or `ggplot2::aes_()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default

mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

data — The data to be displayed in this layer. There are three options:

If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot2::ggplot().

A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See ggplot2::fortify() for which variables will be created.

A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data.

stat — The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following:

- A Stat ggproto subclass, for example StatCount.
- A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count".
- For more information and other ways to specify the stat, see the layer stat documentation.

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

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

na.rm — If TRUE, silently removes NA values, which typically desired for moving averages.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behavior from the default plot specification, e.g. ggplot2::borders().

colour_up, colour_down — Select colors to be applied based on price movement from open to close. If close >= open, colour_up is used. Otherwise, colour_down is used. The default is "darkblue" and "red", respectively.

fill_up, fill_down — Select fills to be applied based on price movement from open to close. If close >= open, fill_up is used. Otherwise, fill_down is used. The default is "darkblue" and "red", respectively. Only affects geom_candlestick().

`...` Other arguments passed on to `ggplot2::layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

Aesthetics

The following aesthetics are understood (required are in bold):

- `x`, Typically a date
- `open`, Required to be the open price
- `high`, Required to be the high price
- `low`, Required to be the low price
- `close`, Required to be the close price
- `alpha`
- `group`
- `linetype`
- `size`

See Also

See individual modeling functions for underlying parameters:

- `geom_ma()` for adding moving averages to ggplots
- `geom_bbands()` for adding Bollinger Bands to ggplots
- `coord_x_date()` for zooming into specific regions of a plot

Examples

```
library(dplyr)
library(ggplot2)
library(lubridate)

AAPL <- tq_get("AAPL", from = "2013-01-01", to = "2016-12-31")

# Bar Chart
AAPL %>%
    ggplot(aes(x = date, y = close)) +
    geom_barchart(aes(open = open, high = high, low = low, close = close)) +
    geom_ma(color = "darkgreen") +
    coord_x_date(xlim = c("2016-01-01", "2016-12-31"),
                 ylim = c(20, 30))

# Candlestick Chart
AAPL %>%
    ggplot(aes(x = date, y = close)) +
    geom_candlestick(aes(open = open, high = high, low = low, close = close)) +
    geom_ma(color = "darkgreen") +
    coord_x_date(xlim = c("2016-01-01", "2016-12-31"),
                 ylim = c(20, 30))
```

geom_ma *Plot moving averages*

Description

The underlying moving average functions used are specified in `TTR::SMA()` from the TTR package. Use `coord_x_date()` to zoom into specific plot regions. The following moving averages are available:

- **Simple moving averages (SMA)**: Rolling mean over a period defined by `n`.
- **Exponential moving averages (EMA)**: Includes exponentially-weighted mean that gives more weight to recent observations. Uses `wilder` and `ratio` args.
- **Weighted moving averages (WMA)**: Uses a set of weights, `wts`, to weight observations in the moving average.
- **Double exponential moving averages (DEMA)**: Uses `v` volume factor, `wilder` and `ratio` args.
- **Zero-lag exponential moving averages (ZLEMA)**: Uses `wilder` and `ratio` args.
- **Volume-weighted moving averages (VWMA)**: Requires `volume` aesthetic.
- **Elastic, volume-weighted moving averages (EVWMA)**: Requires `volume` aesthetic.

Usage

```
geom_ma(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
  ma_fun = SMA,
  n = 20,
  wilder = FALSE,
  ratio = NULL,
  v = 1,
  wts = 1:n,
  ...
)

geom_ma_(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = TRUE,
  show.legend = NA,
  inherit.aes = TRUE,
```

```
  ma_fun = "SMA",
  n = 20,
  wilder = FALSE,
  ratio = NULL,
  v = 1,
  wts = 1:n,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes() or ggplot2::aes_(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot2::ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See ggplot2::fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame., and will be used as the layer data.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
na.rm	If TRUE, silently removes NA values, which typically desired for moving averages.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behavior from the default plot specification, e.g. ggplot2::borders().
ma_fun	The function used to calculate the moving average. Seven options are available including: SMA, EMA, WMA, DEMA, ZLEMA, VWMA, and EVWMA. The default is SMA. See TTR::SMA() for underlying functions.
n	Number of periods to average over. Must be between 1 and nrow(x), inclusive.

wilder	logical; if TRUE, a Welles Wilder type EMA will be calculated; see notes.
ratio	A smoothing/decay ratio. ratio overrides wilder in EMA.
v	The 'volume factor' (a number in [0,1]). See Notes.
wts	Vector of weights. Length of wts vector must equal the length of x, or n (the default).
...	Other arguments passed on to ggplot2::layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like color = "red" or size = 3. They may also be parameters to the paired geom/stat.

Aesthetics

The following aesthetics are understood (required are in bold):

- x
- y
- volume, Required for VWMA and EVWMA
- alpha
- colour
- group
- linetype
- linewidth
- size

See Also

See individual modeling functions for underlying parameters:

- TTR::SMA() for simple moving averages
- TTR::EMA() for exponential moving averages
- TTR::WMA() for weighted moving averages
- TTR::DEMA() for double exponential moving averages
- TTR::ZLEMA() for zero-lag exponential moving averages
- TTR::VWMA() for volume-weighted moving averages
- TTR::EVWMA() for elastic, volume-weighted moving averages
- coord_x_date() for zooming into specific regions of a plot

Examples

```
library(dplyr)
library(ggplot2)

AAPL <- tq_get("AAPL", from = "2013-01-01", to = "2016-12-31")

# SMA
AAPL %>%
```

```
    ggplot(aes(x = date, y = adjusted)) +
    geom_line() +                          # Plot stock price
    geom_ma(ma_fun = SMA, n = 50) +                # Plot 50-day SMA
    geom_ma(ma_fun = SMA, n = 200, color = "red") + # Plot 200-day SMA
    coord_x_date(xlim = c("2016-01-01", "2016-12-31"),
                 ylim = c(20, 30))                    # Zoom in

# EVWMA
AAPL %>%
    ggplot(aes(x = date, y = adjusted)) +
    geom_line() +                                                    # Plot stock price
    geom_ma(aes(volume = volume), ma_fun = EVWMA, n = 50) +   # Plot 50-day EVWMA
    coord_x_date(xlim = c("2016-01-01", "2016-12-31"),
                 ylim = c(20, 30))                                   # Zoom in
```

---

palette_tq *tidyquant palettes for use with scales*

Description

These palettes are mainly called internally by tidyquant `scale_*_tq()` functions.

Usage

```
palette_light()

palette_dark()

palette_green()
```

Examples

```
library(scales)
scales::show_col(palette_light())
```

---

quandl_api_key *Query or set Quandl API Key*

Description

Query or set Quandl API Key

Usage

```
quandl_api_key(api_key)
```

Arguments

api_key	Optionally passed parameter to set Quandl api_key.

Details

A wrapper for Quandl::Quandl.api_key()

Value

Returns invisibly the currently set api_key

See Also

tq_get() get = "quandl"

Examples

```
## Not run:
if (rlang::is_installed("Quandl")) {
quandl_api_key(api_key = "foobar")
}

## End(Not run)
```

---

quandl_search *Search the Quandl database*

Description

Search the Quandl database

Usage

```
quandl_search(query, silent = FALSE, per_page = 10, ...)
```

Arguments

query	Search terms
silent	Prints the results when FALSE.
per_page	Number of results returned per page.
...	Additional named values that are interpretted as Quandl API parameters.

Details

A wrapper for Quandl::Quandl.search()

Value

Returns a tibble with search results.

See Also

tq_get() get = "quandl"

Examples

```
## Not run:
quandl_search(query = "oil")

## End(Not run)
```

---

scale_manual *tidyquant colors and fills for ggplot2.*

Description

The tidyquant scales add colors that work nicely with theme_tq().

Usage

```
scale_color_tq(..., theme = "light")

scale_colour_tq(..., theme = "light")

scale_fill_tq(..., theme = "light")
```

Arguments

...	common parameters for scale_color_manual() or scale_fill_manual(): name, breaks, labels, na.value, limits and guide.
theme	one of "light", "dark", or "green". This should match the theme_tq() that is used with it.

Details

scale_color_tq For use when color is specified as an aes() in a ggplot.

scale_fill_tq For use when fill is specified as an aes() in a ggplot.

See Also

theme_tq()

Examples

```
# Load libraries
library(dplyr)
library(ggplot2)

# Get stock prices
stocks <- c("AAPL", "META", "NFLX") %>%
    tq_get(from = "2013-01-01",
           to   = "2017-01-01")

# Plot for stocks
g <- stocks %>%
    ggplot(aes(date, adjusted, color = symbol)) +
    geom_line() +
    labs(title = "Multi stock example",
         xlab = "Date",
         ylab = "Adjusted Close")

# Plot with tidyquant theme and colors
g +
    theme_tq() +
    scale_color_tq()
```

---

theme_tq *tidyquant themes for ggplot2.*

Description

The `theme_tq()` function creates a custom theme using tidyquant colors.

Usage

```
theme_tq(base_size = 11, base_family = "")

theme_tq_dark(base_size = 11, base_family = "")

theme_tq_green(base_size = 11, base_family = "")
```

Arguments

`base_size`	base font size, given in pts.
`base_family`	base font family

See Also

`scale_manual()`

Examples

```
# Load libraries
library(dplyr)
library(ggplot2)

# Get stock prices
AAPL <- tq_get("AAPL", from = "2013-01-01", to = "2016-12-31")

# Plot using ggplot with theme_tq
AAPL %>% ggplot(aes(x = date, y = close)) +
       geom_line() +
       geom_bbands(aes(high = high, low = low, close = close),
                   ma_fun = EMA,
                   wilder = TRUE,
                   ratio = NULL,
                   n = 50) +
       coord_x_date(xlim = c("2016-01-01", "2016-12-31"),
                 ylim = c(20, 35)) +
       labs(title = "Apple BBands",
            x = "Date",
            y = "Price") +
       theme_tq()
```

---

tidyquant_conflicts *Conflicts between the tidyquant and other packages*

Description

This function lists all the conflicts between packages in the tidyverse and other packages that you have loaded.

Usage

```
tidyquant_conflicts(only = NULL)
```

Arguments

only	Set this to a character vector to restrict to conflicts only with these packages.

Details

There are four conflicts that are deliberately ignored: `intersect`, `union`, `setequal`, and `setdiff` from dplyr. These functions make the base equivalents generic, so shouldn't negatively affect any existing code.

Examples

```
tidyquant_conflicts()
```

tiingo_api_key *Set Tiingo API Key*

Description

Requires the riingo package to be installed.

Usage

```
tiingo_api_key(api_key)
```

Arguments

api_key	Optionally passed parameter to set Tiingo api_key.

Details

A wrapper for riingo::ringo_set_token()

Value

Returns invisibly the currently set api_key

See Also

tq_get() get = "tiingo"

Examples

```
## Not run:
  tiingo_api_key(api_key = "foobar")

## End(Not run)
```

tq_get *Get quantitative data in* tibble *format*

Description

Get quantitative data in tibble format

Usage

```
tq_get(x, get = "stock.prices", complete_cases = TRUE, ...)

tq_get_options()
```

Arguments

x	A single character string, a character vector or tibble representing a single (or multiple) stock symbol, metal symbol, currency combination, FRED code, etc.
get	A character string representing the type of data to get for x. Options include: • "stock.prices": Get the open, high, low, close, volume and adjusted stock prices for a stock symbol from Yahoo Finance (https://finance.yahoo.com/). Wrapper for quantmod::getSymbols(). • "dividends": Get the dividends for a stock symbol from Yahoo Finance (https://finance.yahoo.com/). Wrapper for quantmod::getDividends(). • "splits": Get the split ratio for a stock symbol from Yahoo Finance (https://finance.yahoo.com/). Wrapper for quantmod::getSplits(). • "stock.prices.japan": Get the open, high, low, close, volume and adjusted stock prices for a stock symbol from Yahoo Finance Japan. Wrapper for quantmod::getSymbols.yahooj(). • "economic.data": Get economic data from FRED. rapper for quantmod::getSymbols.FRED(). • "quandl": Get data sets from Quandl. Wrapper for Quandl::Quandl(). See also quandl_api_key(). • "quandl.datatable": Get data tables from Quandl. Wrapper for Quandl::Quandl.datatable(). See also quandl_api_key(). • "tiingo": Get data sets from Tingo (https://www.tiingo.com/). Wrapper for riingo::riingo_prices(). See also tiingo_api_key(). • "tiingo.iex": Get data sets from Tingo (https://www.tiingo.com/). Wrapper for riingo::riingo_iex_prices(). See also tiingo_api_key(). • "tiingo.crypto": Get data sets from Tingo (https://www.tiingo.com/). Wrapper for riingo::riingo_crypto_prices(). See also tiingo_api_key(). • "alphavantager": Get data sets from Alpha Vantage. Wrapper for alphavantager::av_get(). See also av_api_key(). • "rblpapi": Get data sets from Bloomberg. Wrapper for Rblpapi. See also Rblpapi::blpConnect() to connect to Bloomberg terminal (required). Use the argument rblpapi_fun to set the function such as "bdh" (default), "bds", or "bdp".
complete_cases	Removes symbols that return an NA value due to an error with the get call such as sending an incorrect symbol "XYZ" to get = "stock.prices". This is useful in scaling so user does not need to add an extra step to remove these rows. TRUE by default, and a warning message is generated for any rows removed.
...	Additional parameters passed to the "wrapped" function. Investigate underlying functions to see full list of arguments. Common optional parameters include: • from: Standardized for time series functions in quantmod, quandl, tiingo, alphavantager packages. A character string representing a start date in YYYY-MM-DD format. • to: Standardized for time series functions in quantmod, quandl, tiingo, alphavantager packages. A character string representing a end date in YYYY-MM-DD format.

Details

`tq_get()` is a consolidated function that gets data from various web sources. The function is a wrapper for several `quantmod` functions, `Quandl` functions, and also gets data from websources unavailable in other packages. The results are always returned as a `tibble`. The advantages are (1) only one function is needed for all data sources and (2) the function can be seamlessly used with the tidyverse: `purrr`, `tidyr`, and `dplyr` verbs.

`tq_get_options()` returns a list of valid `get` options you can choose from.

`tq_get_stock_index_options()` Is deprecated and will be removed in the next version. Please use `tq_index_options()` instead.

Value

Returns data in the form of a `tibble` object.

See Also

- `tq_index()` to get a ful list of stocks in an index.
- `tq_exchange()` to get a ful list of stocks in an exchange.
- `quandl_api_key()` to set the api key for collecting data via the `"quandl"` get option.
- `tiingo_api_key()` to set the api key for collecting data via the `"tiingo"` get option.
- `av_api_key()` to set the api key for collecting data via the `"alphavantage"` get option.

Examples

```
# Load libraries

# Get the list of `get` options
tq_get_options()

# Get stock prices for a stock from Yahoo
aapl_stock_prices <- tq_get("AAPL")

# Get stock prices for multiple stocks
mult_stocks <- tq_get(c("META", "AMZN"),
                      get  = "stock.prices",
                      from = "2016-01-01",
                      to   = "2017-01-01")


## Not run:

# --- Quandl ---
if (rlang::is_installed("quandl")) {
quandl_api_key('<your_api_key>')
tq_get("EIA/PET_MTTIMUS1_M", get = "quandl", from = "2010-01-01")
}


# Energy data from EIA
```

```
# --- Tiingo ---
if (rlang::is_installed("riingo")) {
tiingo_api_key('<your_api_key>')

# Tiingo Prices (Free alternative to Yahoo Finance!)
tq_get(c("AAPL", "GOOG"), get = "tiingo", from = "2010-01-01")

# Sub-daily prices from IEX ----
tq_get(c("AAPL", "GOOG"),
       get = "tiingo.iex",
       from   = "2020-01-01",
       to     = "2020-01-15",
       resample_frequency = "5min")

# Tiingo Bitcoin Prices ----
tq_get(c("btcusd", "btceur"),
       get    = "tiingo.crypto",
       from   = "2020-01-01",
       to     = "2020-01-15",
       resample_frequency = "5min")


}

# --- Alpha Vantage ---

if (rlang::is_installed("alphavantager")) {
av_api_key('<your_api_key>')

# Daily Time Series
tq_get("AAPL",
       get        = "alphavantager",
       av_fun     = "TIME_SERIES_DAILY_ADJUSTED",
       outputsize = "full")

# Intraday 15 Min Interval
tq_get("AAPL",
       get        = "alphavantage",
       av_fun     = "TIME_SERIES_INTRADAY",
       interval   = "15min",
       outputsize = "full")
# FX DAILY
tq_get("USD/EUR", get = "alphavantage", av_fun = "FX_DAILY", outputsize = "full")

# FX REAL-TIME QUOTE
tq_get("USD/EUR", get = "alphavantage", av_fun = "CURRENCY_EXCHANGE_RATE")

}

## End(Not run)
```

tq_index *Get all stocks in a stock index or stock exchange in* tibble *format*

Description

Get all stocks in a stock index or stock exchange in tibble format

Usage

```
tq_index(x, use_fallback = FALSE)

tq_index_options()

tq_exchange(x)

tq_exchange_options()

tq_fund_holdings(x, source = "SSGA")

tq_fund_source_options()
```

Arguments

x	A single character string, a character vector or tibble representing a single stock index or multiple stock indexes.
use_fallback	A boolean that can be used to return a fallback data set last downloaded when the package was updated. Useful if the website is down. Set to FALSE by default.
source	The API source to use.

Details

tq_index() returns the stock symbol, company name, weight, and sector of every stock in an index.

tq_index_options() returns a list of stock indexes you can choose from.

tq_exchange() returns the stock symbol, company, last sale price, market capitalization, sector and industry of every stock in an exchange. Three stock exchanges are available (AMEX, NASDAQ, and NYSE).

tq_exchange_options() returns a list of stock exchanges you can choose from. The options are AMEX, NASDAQ and NYSE.

tq_fund_holdings() returns the the stock symbol, company name, weight, and sector of every stock in an fund. The source parameter specifies which investment management company to use. Example: source = "SSGA" connects to State Street Global Advisors (SSGA). If x = "SPY", then SPDR SPY ETF holdings will be returned.

tq_fund_source_options(): returns the options that can be used for the source API for tq_fund_holdings().

Value

Returns data in the form of a `tibble` object.

See Also

`tq_get()` to get stock prices, financials, key stats, etc using the stock symbols.

Examples

```
# Stock Indexes:

# Get the list of stock index options
tq_index_options()

# Get all stock symbols in a stock index
## Not run:
tq_index("DOW")

## End(Not run)

# Stock Exchanges:

# Get the list of stock exchange options
tq_exchange_options()

# Get all stocks in a stock exchange
## Not run:
tq_exchange("NYSE")

## End(Not run)

# Mutual Funds and ETFs:

# Get the list of stock exchange options
tq_fund_source_options()

# Get all stocks in a fund
## Not run:
tq_fund_holdings("SPY", source = "SSGA")

## End(Not run)
```

---

tq_mutate *Mutates quantitative data*

Description

`tq_mutate()` adds new variables to an existing tibble; `tq_transmute()` returns only newly created columns and is typically used when periodicity changes

Usage

```
tq_mutate(
  data,
  select = NULL,
  mutate_fun,
  col_rename = NULL,
  ohlc_fun = NULL,
  ...
)

tq_mutate_(data, select = NULL, mutate_fun, col_rename = NULL, ...)

tq_mutate_xy(data, x, y = NULL, mutate_fun, col_rename = NULL, ...)

tq_mutate_xy_(data, x, y = NULL, mutate_fun, col_rename = NULL, ...)

tq_mutate_fun_options()

tq_transmute(
  data,
  select = NULL,
  mutate_fun,
  col_rename = NULL,
  ohlc_fun = NULL,
  ...
)

tq_transmute_(data, select = NULL, mutate_fun, col_rename = NULL, ...)

tq_transmute_xy(data, x, y = NULL, mutate_fun, col_rename = NULL, ...)

tq_transmute_xy_(data, x, y = NULL, mutate_fun, col_rename = NULL, ...)

tq_transmute_fun_options()
```

Arguments

data	A tibble (tidy data frame) of data typically from tq_get().
select	The columns to send to the mutation function.
mutate_fun	The mutation function from either the xts, quantmod, or TTR package. Execute tq_mutate_fun_options() to see the full list of options by package.
col_rename	A string or character vector containing names that can be used to quickly rename columns.
ohlc_fun	Deprecated. Use select.
...	Additional parameters passed to the appropriate mutatation function.
x, y	Parameters used with _xy that consist of column names of variables to be passed to the mutatation function (instead of OHLC functions).

Details

tq_mutate and tq_transmute are very flexible wrappers for various xts, quantmod and TTR functions. The main advantage is the results are returned as a tibble and the function can be used with the tidyverse. tq_mutate is used when additional columns are added to the return data frame. tq_transmute works exactly like tq_mutate except it only returns the newly created columns. This is helpful when changing periodicity where the new columns would not have the same number of rows as the original tibble.

select specifies the columns that get passed to the mutation function. Select works as a more flexible version of the OHLC extractor functions from quantmod where non-OHLC data works as well. When select is NULL, all columns are selected. In Example 1 below, close returns the "close" price and sends this to the mutate function, periodReturn.

mutate_fun is the function that performs the work. In Example 1, this is periodReturn, which calculates the period returns. The ... are additional arguments passed to the mutate_fun. Think of the whole operation in Example 1 as the close price, obtained by select = close, being sent to the periodReturn function along with additional arguments defining how to perform the period return, which includes period = "daily" and type = "log". Example 4 shows how to apply a rolling regression.

tq_mutate_xy and tq_transmute_xy are designed to enable working with mutatation functions that require two primary inputs (e.g. EVWMA, VWAP, etc). Example 2 shows this benefit in action: using the EVWMA function that uses volume to define the moving average period.

tq_mutate_, tq_mutate_xy_, tq_transmute_, and tq_transmute_xy_ are setup for Non-Standard Evaluation (NSE). This enables programatically changing column names by modifying the text representations. Example 5 shows the difference in implementation. Note that character strings are being passed to the variables instead of unquoted variable names. See vignette("nse") for more information.

tq_mutate_fun_options and tq_transmute_fun_options return a list of various financial functions that are compatible with tq_mutate and tq_transmute, respectively.

Value

Returns mutated data in the form of a tibble object.

See Also

tq_get()

Examples

```
# Load libraries
library(dplyr)

##### Basic Functionality

fb_stock_prices  <- tidyquant::FANG %>%
    filter(symbol == "META") %>%
        filter(
            date >= "2016-01-01",
            date <= "2016-12-31"
```

```
        )

goog_stock_prices  <- FANG %>%
    filter(symbol == "GOOG") %>%
        filter(
            date >= "2016-01-01",
            date <= "2016-12-31"
        )

# Example 1: Return logarithmic daily returns using periodReturn()
fb_stock_prices %>%
    tq_mutate(select = close, mutate_fun = periodReturn,
              period = "daily", type = "log")

# Example 2: Use tq_mutate_xy to use functions with two columns required
fb_stock_prices %>%
    tq_mutate_xy(x = close, y = volume, mutate_fun = EVWMA,
                 col_rename = "EVWMA")

# Example 3: Using tq_mutate to work with non-OHLC data
tq_get("DCOILWTICO", get = "economic.data") %>%
    tq_mutate(select = price, mutate_fun = lag.xts, k = 1, na.pad = TRUE)

# Example 4: Using tq_mutate to apply a rolling regression
fb_returns <- fb_stock_prices %>%
    tq_transmute(adjusted, periodReturn, period = "monthly", col_rename = "fb.returns")
goog_returns <- goog_stock_prices %>%
    tq_transmute(adjusted, periodReturn, period = "monthly", col_rename = "goog.returns")
returns_combined <- left_join(fb_returns, goog_returns, by = "date")
regr_fun <- function(data) {
    coef(lm(fb.returns ~ goog.returns, data = as_tibble(data)))
}
returns_combined %>%
    tq_mutate(mutate_fun = rollapply,
              width      = 6,
              FUN        = regr_fun,
              by.column  = FALSE,
              col_rename = c("coef.0", "coef.1"))

# Example 5: Non-standard evaluation:
# Programming with tq_mutate_() and tq_mutate_xy_()
col_name <- "adjusted"
mutate <- c("MACD", "SMA")
tq_mutate_xy_(fb_stock_prices, x = col_name, mutate_fun = mutate[[1]])
```

---

tq_performance — *Computes a wide variety of summary performance metrics from stock or portfolio returns*

Description

Asset and portfolio performance analysis is a deep field with a wide range of theories and methods for analyzing risk versus reward. The `PerformanceAnalytics` package consolidates many of the most widely used performance metrics as functions that can be applied to stock or portfolio returns. `tq_performance` implements these performance analysis functions in a tidy way, enabling scaling analysis using the split, apply, combine framework.

Usage

```
tq_performance(data, Ra, Rb = NULL, performance_fun, ...)

tq_performance_(data, Ra, Rb = NULL, performance_fun, ...)

tq_performance_fun_options()
```

Arguments

`data`	A `tibble` (tidy data frame) of returns in tidy format (i.e long format).
`Ra`	The column of asset returns
`Rb`	The column of baseline returns (for functions that require comparison to a baseline)
`performance_fun`	The performance function from `PerformanceAnalytics`. See `tq_performance_fun_options()` for a complete list of integrated functions.
`...`	Additional parameters passed to the `PerformanceAnalytics` function.

Details

Important concept: Performance is based on the statistical properties of returns, and as a result this function uses stock or portfolio returns as opposed to stock prices.

`tq_performance` is a wrapper for various `PerformanceAnalytics` functions that return portfolio statistics. The main advantage is the ability to scale with the `tidyverse`.

`Ra` and `Rb` are the columns containing asset and baseline returns, respectively. These columns are mapped to the `PerformanceAnalytics` functions. Note that `Rb` is not always required, and in these instances the argument defaults to `Rb = NULL`. The user can tell if `Rb` is required by researching the underlying performance function.

`...` are additional arguments that are passed to the `PerformanceAnalytics` function. Search the underlying function to see what arguments can be passed through.

`tq_performance_fun_options` returns a list of compatible `PerformanceAnalytics` functions that can be supplied to the `performance_fun` argument.

Value

Returns data in the form of a `tibble` object.

See Also

- tq_transmute() which can be used to calculate period returns from a set of stock prices. Use mutate_fun = periodReturn with the appropriate periodicity such as period = "monthly".
- tq_portfolio() which can be used to aggregate period returns from multiple stocks to period returns for a portfolio.
- The PerformanceAnalytics package, which contains the underlying functions for the performance_fun argument. Additional parameters can be passed via ....

Examples

```
# Load libraries
library(dplyr)

# Use FANG data set

# Get returns for individual stock components grouped by symbol
Ra <- FANG %>%
    group_by(symbol) %>%
    tq_transmute(adjusted, periodReturn, period = "monthly", col_rename = "Ra")

# Get returns for SP500 as baseline
Rb <- "^GSPC" %>%
    tq_get(get  = "stock.prices",
           from = "2010-01-01",
           to   = "2015-12-31") %>%
    tq_transmute(adjusted, periodReturn, period = "monthly", col_rename = "Rb")

# Merge stock returns with baseline
RaRb <- left_join(Ra, Rb, by = c("date" = "date"))

##### Performance Metrics #####

# View options
tq_performance_fun_options()

# Get performance metrics
RaRb %>%
    tq_performance(Ra = Ra, performance_fun = SharpeRatio, p = 0.95)

RaRb %>%
    tq_performance(Ra = Ra, Rb = Rb, performance_fun = table.CAPM)
```

---

tq_portfolio *Aggregates a group of returns by asset into portfolio returns*

Description

Aggregates a group of returns by asset into portfolio returns

Usage

```
tq_portfolio(
  data,
  assets_col,
  returns_col,
  weights = NULL,
  col_rename = NULL,
  ...
)

tq_portfolio_(
  data,
  assets_col,
  returns_col,
  weights = NULL,
  col_rename = NULL,
  ...
)

tq_repeat_df(data, n, index_col_name = "portfolio")
```

Arguments

data	A tibble (tidy data frame) of returns in tidy format (i.e long format).
assets_col	The column with assets (securities)
returns_col	The column with returns
weights	Optional parameter for the asset weights, which can be passed as a numeric vector the length of the number of assets or a two column tibble with asset names in first column and weights in second column.
col_rename	A string or character vector containing names that can be used to quickly rename columns.
...	Additional parameters passed to PerformanceAnalytics::Return.portfolio
n	Number of times to repeat a data frame row-wise.
index_col_name	A renaming function for the "index" column, used when repeating data frames.

Details

tq_portfolio is a wrapper for PerformanceAnalytics::Return.portfolio. The main advantage is the results are returned as a tibble and the function can be used with the tidyverse.

assets_col and returns_col are columns within data that are used to compute returns for a portfolio. The columns should be in "long" format (or "tidy" format) meaning there is only one column containing all of the assets and one column containing all of the return values (i.e. not in "wide" format with returns spread by asset).

weights are the weights to be applied to the asset returns. Weights can be input in one of three options:

- Single Portfolio: A numeric vector of weights that is the same length as unique number of assets. The weights are applied in the order of the assets.
- Single Portfolio: A two column tibble with assets in the first column and weights in the second column. The advantage to this method is the weights are mapped to the assets and any unlisted assets default to a weight of zero.
- Multiple Portfolios: A three column tibble with portfolio index in the first column, assets in the second column, and weights in the third column. The tibble must be grouped by portfolio index.

`tq_repeat_df` is a simple function that repeats a data frame `n` times row-wise (long-wise), and adds a new column for a portfolio index. The function is used to assist in Multiple Portfolio analyses, and is a useful precursor to `tq_portfolio`.

Value

Returns data in the form of a `tibble` object.

See Also

- `tq_transmute()` which can be used to get period returns.
- `PerformanceAnalytics::Return.portfolio()` which is the underlying function that specifies which parameters can be passed via `...`

Examples

```
# Load libraries
library(dplyr)

# Use FANG data set

# Get returns for individual stock components
monthly_returns_stocks <- FANG %>%
    group_by(symbol) %>%
    tq_transmute(adjusted, periodReturn, period = "monthly")

##### Portfolio Aggregation Methods #####

# Method 1: Use tq_portfolio with numeric vector of weights

weights <- c(0.50, 0.25, 0.25, 0)
tq_portfolio(data = monthly_returns_stocks,
             assets_col = symbol,
             returns_col = monthly.returns,
             weights = weights,
             col_rename = NULL,
             wealth.index = FALSE)

# Method 2: Use tq_portfolio with two column tibble and map weights

# Note that GOOG's weighting is zero in Method 1. In Method 2,
# GOOG is not added and same result is achieved.
```

```
weights_df <- tibble(symbol = c("META", "AMZN", "NFLX"),
                     weights = c(0.50, 0.25, 0.25))
tq_portfolio(data = monthly_returns_stocks,
             assets_col = symbol,
             returns_col = monthly.returns,
             weights = weights_df,
             col_rename = NULL,
             wealth.index = FALSE)

# Method 3: Working with multiple portfolios

# 3A: Duplicate monthly_returns_stocks multiple times
mult_monthly_returns_stocks <- tq_repeat_df(monthly_returns_stocks, n = 4)

# 3B: Create weights table grouped by portfolio id
weights <- c(0.50, 0.25, 0.25, 0.00,
             0.00, 0.50, 0.25, 0.25,
             0.25, 0.00, 0.50, 0.25,
             0.25, 0.25, 0.00, 0.50)
stocks <- c("META", "AMZN", "NFLX", "GOOG")
weights_table <- tibble(stocks) %>%
    tq_repeat_df(n = 4) %>%
    bind_cols(tibble(weights)) %>%
    group_by(portfolio)

# 3C: Scale to multiple portfolios
tq_portfolio(data = mult_monthly_returns_stocks,
             assets_col = symbol,
             returns_col = monthly.returns,
             weights = weights_table,
             col_rename = NULL,
             wealth.index = FALSE)
```

Index