Package ‘tidyquant’

February 11, 2019

Type Package

Title Tidy Quantitative Financial Analysis

Version 0.5.5

Date 2018-05-06

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

Description Bringing 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://github.com/business-science/tidyquant

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

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (¿= 3.0.0), lubridate, PerformanceAnalytics, quantmod (¿= 0.4-13), tidyverse

Imports dplyr, ggplot2, httr, lazyeval, magrittr, purrr, Quandl, stringr, tibble, tidyr, timetk, TTR, xml2, xts, rlang

Suggests alphavantager, broom, knitr, rmarkdown, testthat, tibbletime, scales, Rblpapi, XLConnect

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

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

Repository CRAN

Date/Publication 2018-05-09 08:48:07 UTC

R topics documented:

`av_api_key` *Set Alpha Vantage API Key*

Description

Set Alpha Vantage API Key

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:
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(tidyquant)

# coord_x_date
AAPL <- tq_get("AAPL")
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
    coord_x_date(xlim = c(today() - weeks(12), today()),
               ylim = c(100, 130))        # Zoom in


# 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`	Parameters used with `_xy` that consist of column names of variables to be passed to the mutatation function (instead of OHLC functions).
`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()`

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

Description

A dataset containing the daily historical stock prices for the "FANG" tech stocks, "FB", "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

`http://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`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`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 behaviour 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`, and provides additional smoothing in `VMA`.
`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

```
# Load libraries
library(tidyquant)

AAPL <- tq_get("AAPL")

# 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(today() - years(1), today()), ylim = c(80, 130))


# 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(today() - years(1), today()), ylim = c(80, 130))


# 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(today() - years(1), today()), ylim = c(80, 130))
```

---

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, color_up = "darkblue", color_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, color_up = "darkblue", color_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, as a string.
`position`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`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 behaviour from the default plot specification, e.g. `ggplot2::borders()`.
`color_up, color_down`	Select colors to be applied based on price movement from open to close. If close ¿= open, `color_up` is used. Otherwise, `color_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

```
# Load libraries
library(tidyquant)

AAPL <- tq_get("AAPL")

# 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(today() - weeks(6), today()),
                 ylim = c(100, 130))

# 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(today() - weeks(6), today()),
                 ylim = c(100, 130))
```

`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`	Position adjustment, either as a string, or the result of a call to a position adjustment function.
`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 behaviour 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, and provides additional smoothing in VMA.
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
- 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

```
# Load libraries
library(tidyquant)

AAPL <- tq_get("AAPL")

# 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(today() - weeks(12), today()),
               ylim = c(100, 130))                     # 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(today() - weeks(12), today()),
               ylim = c(100, 130))                                  # 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:
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 discrete scale parameters: `name`, `breaks`, `labels`, `na.value`, `limits` and `guide`. See `discrete_scale()` for more details
`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(tidyquant)

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

# Plot for stocks
a <- 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
a +
    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
`base_family`	base font family

See Also

`scale_manual()`

Examples

```
# Load libraries
library(tidyquant)

# Get stock prices
AAPL <- tq_get("AAPL")

# 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(today() - years(1), today()),
                    ylim = c(80, 130)) +
       labs(title = "Apple BBands",
            x = "Date",
            y = "Price") +
       theme_tq()
```

`tidyquant` *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")`

`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()

tq_get_stock_index_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. Wrapper for `quantmod::getSymbols()`.
- `"stock.prices.google"`: DISCONTINUED.
- `"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()`.
- `"financials"`: DISCONTINUED.
- `"key.ratios"`: Get 89 historical growth, profitablity, financial health, efficiency, and valuation ratios that span 10-years from Morningstar.
- `"key.stats"`: DISCONTINUED.
- `"dividends"`: Get the dividends for a stock symbol from Yahoo Finance. Wrapper for `quantmod::getDividends()`.
- `"splits"`: Get the splits for a stock symbol from Yahoo Finance. Wrapper for `quantmod::getSplits()`.
- `"economic.data"`: Get economic data from FRED. rapper for `quantmod::getSymbols.FRE`
- `"metal.prices"`: Get the metal prices from Oanda. Wrapper for `quantmod::getMetals()`.
- `"exchange.rates"`: Get exchange rates from Oanda. Wrapper for `quantmod::getFX()`.
- `"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()`.
- `"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`: Optional for various time series functions in quantmod / quandl packages. A character string representing a start date in YYYY-MM-DD format. No effect on `"key.ratios"`, or `"key.stats"`.
- `to`: Optional for various time series functions in quantmod / quandl packages. A character string representing a end date in YYYY-MM-DD format. No effect on `get = "key.ratios"` or `"key.stats"`.

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 seemlessly 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.
- `av_api_key()` to set the api key for collecting data via the `"alphavantage"` get option.

Examples

```
# Load libraries
library(tidyquant)

# 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("FB", "AMZN"),
                      get  = "stock.prices",
                      from = "2016-01-01",
                      to   = "2017-01-01")

# Multiple gets
mult_gets <- tq_get("AAPL",
                    get = c("stock.prices", "dividends"),
                    from = "2016-01-01",
                    to   = "2017-01-01")
```

---

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_exchange(x)

tq_index_options()

tq_exchange_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.

Details

`tq_index()` returns the stock symbol, company name, weight, and sector of every stock in an index. Nine stock indices are available. The source is www.us.spdrs.com.

`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.

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

```
# Load libraries
library(tidyquant)

# 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)

# 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)
```

---

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(tidyquant)

##### Basic Functionality

fb_stock_prices  <- tq_get("FB",
                           get  = "stock.prices",
                           from = "2016-01-01",
                           to   = "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")
xlk_returns <- tq_get("XLK", from = "2016-01-01", to = "2016-12-31") %>%
    tq_transmute(adjusted, periodReturn, period = "monthly", col_rename = "xlk.returns")
returns_combined <- left_join(fb_returns, xlk_returns, by = "date")
regr_fun <- function(data) {
    coef(lm(fb.returns ~ xlk.returns, data = as_data_frame(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(tidyquant)

# Use FANG data set
data(FANG)

# 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::Returns.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::Returns.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(tidyquant)

# Use FANG data set
data(FANG)

# 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("FB", "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("FB", "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