Package: stocks (via r-universe)

October 25, 2024

Type Package

Title Stock Market Analysis

Version 2.0.0

License GPL-3

LazyData true

URL https://github.com/vandomed/stocks

Date 2020-07-14

Author Dane R. Van Domelen

Maintainer Dane R. Van Domelen <vandomed@gmail.com>

Description Functions for analyzing and visualizing stock market data.
Main features are loading and aligning historical data,
calculating performance metrics for individual funds or
portfolios (e.g. annualized growth, maximum drawdown,
Sharpe/Sortino ratio), and creating graphs.

Depends dplyr, R (>= 3.5.0), rbenchmark, quantmod

Imports data.table, fastmatch, ggplot2, ggrepel, graphics, grDevices,
lubridate, methods, plotly, purrr, Rcpp (>= 0.12.15),
RcppEigen, roll, rvest, scales, stats, tidyr, TTR, xml2, zoo

Suggests knitr, rmarkdown, pander, printr

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 7.1.0

Repository https://vandomed.r-universe.dev

RemoteUrl https://github.com/vandomed/stocks

RemoteRef HEAD

RemoteSha 03244f454ad44f3485263b8267b0ddd6c035f90b

Contents

`beta_trailing50` *Calculate Beta Using Last 50 Daily Gains*

Description

Calculates beta for a ticker symbol based on the previous 50 daily gains.

Usage

```
beta_trailing50(ticker, benchmark = "SPY", ...)
```

Arguments

`ticker`	Character string with ticker symbol that Yahoo! Finance recognizes.
`benchmark`	Character string specifying which fund to use as benchmark.
`...`	Arguments to pass to `load_gains`.

Value

Numeric value.

References

Jeffrey A. Ryan and Joshua M. Ulrich (2019). quantmod: Quantitative Financial Modelling Framework. R package version 0.4-15. https://CRAN.R-project.org/package=quantmod

Examples

```
## Not run:
# Calculate TLT's beta based on the previous 50 daily gains
beta_trailing50("TLT")

## End(Not run)
```

calc_metric *Calculate Performance Metric*

Description

Mainly a helper function for calc_metrics and calc_metrics_overtime, but could also be used independently.

Usage

```
calc_metric(gains, metric = "mean", units.year = 252, benchmark.gains = NULL)
```

Arguments

`gains` Numeric vector.

`metric` Character string specifying metric to calculate. Choices are "mean", "sd", "growth.x" for growth of $x where x is the initial value, "growth" for percent growth, "cagr" for compound annualized growth rate, "mdd" for max drawdown, "sharpe", "sortino", "alpha", "alpha.annualized", "beta", "r.squared", "pearson" or "spearman" for Pearson/Spearman correlation with benchmark, and "auto.pearson" or "auto.spearman" for Pearson/Spearman autocorrelation.

`units.year` Integer value.

`benchmark.gains` Numeric vector.

Value

Numeric value.

Examples

```
## Not run:
# Load daily gains for SPY in 2019 and calculate various metrics
gains <- load_gains(tickers = "SPY", from = "2019-01-01", to = "2019-12-31")
calc_metric(gains$SPY, "growth")
calc_metric(gains$SPY, "cagr")
calc_metric(gains$SPY, "mdd")
calc_metric(gains$SPY, "sharpe")
calc_metric(gains$SPY, "growth.10k")

# Calculate alpha and beta for TLT in 2019, using SPY as a benchmark
gains <- load_gains(tickers = c("SPY", "TLT"), from = "2019-01-01", to = "2019-12-31")
calc_metric(gains = gains$TLT, metric = "alpha", benchmark.gains = gains$SPY)
calc_metric(gains = gains$TLT, metric = "beta", benchmark.gains = gains$SPY)

## End(Not run)
```

calc_metrics *Calculate Performance Metrics*

Description

Useful for comparing funds on one or more metrics.

Usage

```
calc_metrics(
  gains = NULL,
 metrics = c("cagr", "mdd", "mean", "sd", "sharpe", "alpha.annualized", "beta", "r"),
  prices = NULL,
  tickers = NULL,
  ...,
  benchmark = "SPY"
)
```

Arguments

`gains`	Data frame with one column of gains for each investment and a date variable named Date.
`metrics`	Character vector specifying metrics to calculate. Choices are `"cagr"` for compound annualized growth rate, `"mdd"` for max drawdown, `"mean"`, `"sd"`, `"sharpe"`, `"growth.x"` for growth of $x where x is the initial value, `"growth"` for percent growth, `"sortino"`, `"alpha"`, `"alpha.annualized"`, `"beta"`, `"r.squared"`, `"pearson"` or `"spearman"` for Pearson/Spearman correlation with benchmark, and `"auto.pearson"` or `"auto.spearman"` for Pearson/Spearman autocorrelation.
`prices`	Data frame with one column of prices for each investment and a date variable named Date.
`tickers`	Character vector of ticker symbols that Yahoo! Finance recognizes, if you want to download data on the fly.
`...`	Arguments to pass along with `tickers` to `load_gains`.
`benchmark`	Character string specifying which fund to use as a benchmark for metrics that require one.

Value

Data frame with performance metrics for each investment.

Examples

```
## Not run:
# Calculate performance metrics for FANG stocks since the beginning of 2019
calc_metrics(tickers = fang, from = "2019-01-01")
```

```
# Repeat, but use step-by-step approach with piping (need SPY to calculate
# alpha and beta)
c("SPY", fang) %>%
  load_gains(from = "2019-01-01") %>%
  calc_metrics()

## End(Not run)
```

calc_metrics_123 *Calculate Performance Metrics for Any Combination of Individual Funds, 2-Fund Portfolios, and 3-Fund Portfolios*

Description

Integrates `calc_metrics`, `calc_metrics_2funds`, and `calc_metrics_3funds` into a single function, so you can compare strategies of varying complexities.

Usage

```
calc_metrics_123(
  gains = NULL,
  metrics = c("mean", "sd"),
  tickers = NULL,
  ...,
  step = 1,
  prices = NULL,
  benchmark = "SPY"
)
```

Arguments

`gains` Data frame with a date variable named Date and one column of gains for each fund.

`metrics` Character vector specifying metrics to calculate. See `?calc_metrics` for choices.

`tickers` List where each element is a character vector of ticker symbols for a particular fund combination, e.g. `list("BRK-B", c("SPY", "TLT")`. Each set can contain 1-3 funds.

`...` Arguments to pass along with `tickers` to `load_gains`.

`step` Numeric value specifying fund allocation increments.

`prices` Data frame with a date variable named Date and one column of prices for each fund.

`benchmark` Character string specifying which fund to use as a benchmark for metrics that require one.

Value

Data frame with performance metrics for each portfolio at each allocation.

Examples

```
## Not run:
# Calculate CAGR vs. max drawdown for BRK-B, SPY/TLT, and VWEHX/VBLTX/VFINX
df <- calc_metrics_123(
  tickers = list("BRK-B", c("SPY", "TLT"), c("VWEHX", "VBLTX", "VFINX")),
  metrics = c("cagr", "mdd")
)
head(df)

# To plot, just pipe into plot_metrics_123
df %>%
  plot_metrics_123()

# Or bypass calc_metrics_123 altogether
plot_metrics_123(
  formula = cagr ~ mdd,
  tickers = list("BRK-B", c("SPY", "TLT"), c("VWEHX", "VBLTX", "VFINX"))
)

## End(Not run)
```

calc_metrics_2funds *Calculate Performance Metrics for 2-Fund Portfolios with Varying Allocations*

Description

Useful for assessing the characteristics of 2-fund portfolios.

Usage

```
calc_metrics_2funds(
  gains = NULL,
  metrics = c("mean", "sd"),
  tickers = NULL,
  ...,
  prices = NULL,
  benchmark = "SPY",
  ref.tickers = NULL
)
```

Arguments

`gains` Data frame with a date variable named Date and one column of gains for each fund.

`metrics` Character vector specifying metrics to calculate. See `?calc_metrics` for choices.

`tickers` Character vector of ticker symbols, where the first two are are a 2-fund pair, the next two are another, and so on.

`...` Arguments to pass along with `tickers` to `load_gains`.

`prices` Data frame with a date variable named Date and one column of prices for each fund.

`benchmark` Character string specifying which fund to use as a benchmark for metrics that require one.

`ref.tickers` Character vector of ticker symbols to include.

Value

Data frame with performance metrics for each portfolio at each allocation.

Examples

```
## Not run:
# Calculate CAGR and max drawdown for UPRO/VBLTX
df <- calc_metrics_2funds(
  metrics = c("cagr", "mdd"),
  tickers = c("UPRO", "VBLTX")
)
head(df)

# To plot, just pipe into plot_metrics_2funds
df %>%
  plot_metrics_2funds()

# Or bypass calc_metrics_2funds altogether
plot_metrics_2funds(
  formula = cagr ~ mdd,
  tickers = c("UPRO", "VBLTX")
)

## End(Not run)
```

`calc_metrics_3funds` *Calculate Performance Metrics for 3-Fund Portfolios with Varying Allocations*

Description

Useful for assessing the characteristics of 3-fund portfolios.

Usage

```
calc_metrics_3funds(
  gains = NULL,
  metrics = c("mean", "sd"),
  tickers = NULL,
  ...,
  step = 1,
  prices = NULL,
  benchmark = "SPY",
  ref.tickers = NULL
)
```

Arguments

`gains`	Data frame with a date variable named Date and one column of gains for each fund.
`metrics`	Character vector specifying metrics to calculate. See `?calc_metrics` for choices.
`tickers`	Character vector of ticker symbols, where the first three are are a 3-fund set, the next three are another, and so on.
`...`	Arguments to pass along with `tickers` to `load_gains`.
`step`	Numeric value specifying fund allocation increments.
`prices`	Data frame with a date variable named Date and one column of prices for each fund.
`benchmark`	Character string specifying which fund to use as a benchmark for metrics that require one.
`ref.tickers`	Character vector of ticker symbols to include.

Value

Data frame with performance metrics for each portfolio at each allocation.

Examples

```
## Not run:
# Calculate CAGR and max drawdown for UPRO/VBLTX/VWEHX
df <- calc_metrics_3funds(metrics = c("cagr", "mdd"), tickers = c("UPRO", "VBLTX", "VWEHX"))
head(df)

# To plot, just pipe into plot_metrics_3funds
df %>%
  plot_metrics_3funds()

# Or bypass calc_metrics_3funds altogether
plot_metrics_3funds(formula = cagr ~ mdd, tickers = c("UPRO", "VBLTX", "VWEHX"))
```

```
## End(Not run)
```

`calc_metrics_overtime` *Calculate Performance Metrics over Time*

Description

Useful for assessing how one or two performance metrics vary over time, for one or several funds. Supports fixed-width rolling windows, fixed-width disjoint windows, and disjoint windows on per-month or per-year basis.

Usage

```
calc_metrics_overtime(
  gains = NULL,
  metrics = c("mean", "sd"),
  tickers = NULL,
  ...,
  type = "hop.year",
  minimum.n = 3,
  prices = NULL,
  benchmark = "SPY"
)
```

Arguments

`gains` Data frame with one column of gains for each investment and a date variable named Date.

`metrics` Character vector specifying metrics to calculate. See `?calc_metrics` for choices.

`tickers` Character vector of ticker symbols that Yahoo! Finance recognizes, if you want to download data on the fly.

`...` Arguments to pass along with `tickers` to `load_gains`.

`type` Character string or vector specifying type of calculation. Choices are (1) `"roll.n"` where n is a positive integer; (2) `"hop.n"` where n is a positive integer; (3) `"hop.month"`; (4) `"hop.year"`; and (5) vector of break-point dates, e.g. `c("2019-01-01", "2019-06-01")` for 3 periods. The "roll" and "hop" options correspond to rolling and disjoint windows, respectively.

`minimum.n` Integer value specifying the minimum number of observations per period, e.g. if you want to exclude short partial months at the beginning or end of the analysis period.

`prices` Data frame with a date variable named Date and one column of prices for each investment.

`benchmark` Character string specifying which fund to use as a benchmark for metrics that require one.

Value

Data frame with performance metrics for each investment.

Examples

```
## Not run:
# Calculate annual CAGR's, MDD's, and Sharpe ratios for FANG stocks
calc_metrics_overtime(
  tickers = c("FB", "AAPL", "NFLX", "GOOG"),
  metrics = c("cagr", "mdd", "sharpe"),
  type = "hop.year"
)

## End(Not run)
```

`contango_hedged` *Backtest a Hedged Contango-Based Volatility Trading Strategy*

Description

Implements the following strategy: Each day, hold XIV/SPXU (weighted for zero beta) if contango > `xiv.spxu.cutpoint`, hold VXX/UPRO (weighted for zero beta) if contango < `vxx.upro.cutpoint`, and hold cash otherwise. Perhaps not very useful since XIV closed on Feb. 20, 2018.

Usage

```
contango_hedged(
  contango,
  xiv.spxu.gains = NULL,
  vxx.upro.gains = NULL,
  xiv.spxu.cutpoint = 6.36,
  vxx.upro.cutpoint = 5.45,
  xiv.allocation = 0.46,
  vxx.allocation = 0.46,
  xiv.beta = NULL,
  vxx.beta = NULL,
  initial = 10000
)
```

Arguments

`contango` Numeric vector of contango values at the end of each trading day.

`xiv.spxu.gains` 2-column numeric matrix with gains for XIV and SPXU. Should have the same number of rows as `contango` and be date-shifted one value to the right. For example, the first row should have the XIV and SPXU gains for the day AFTER the first contango value.

`vxx.upro.gains` 2-column numeric matrix with gains for VXX and UPRO. Should have the same number of rows as `contango` and be date-shifted one value to the right. For example, the first row should have the VXX and UPRO gains for the day AFTER the first contango value.

`xiv.spxu.cutpoint`
Numeric value giving the contango cutpoint for XIV/SPXU position. For example, if `xiv.spxu.cutpoint` = 5, XIV/SPXU will be held whenever contango is greater than 5%.

`vxx.upro.cutpoint`
Numeric value giving the contango cutpoint for VXX/UPRO position. For example, if `vxx.upro.cutpoint` = -5, VXX/UPRO will be held whenever contango is less than -5%.

`xiv.allocation` Numeric value specifying XIV allocation for XIV/SPXU position. For example, if set to 0.46, 46% is allocated to XIV and 54% to SPXU when contango > `xiv.spxu.cutpoint`.

`vxx.allocation` Numeric value specifying VXX allocation for VXX/UPRO position. For example, if set to 0.46, 46% is allocated to VXX and 54% to UPRO when contango < `vxx.upro.cutpoint`.

`xiv.beta` Numeric value specifying XIV's beta. If specified, the function figures out what `xiv.allocation` needs to be for zero-beta XIV/SPXU positions. For example, if set to 3.5, then 46.2% XIV/53.8% SPXU achieves zero beta.

`vxx.beta` Numeric value indicating VXX's beta. If specified, the function figures out what `vxx.allocation` needs to be for zero-beta VXX/UPRO positions. For example, if set to -3.5, then 46.2% VXX/53.8% UPRO achieves zero beta.

`initial` Numeric value giving the initial value of the portfolio.

Details

You can find historical contango values from The Intelligent Investor Blog. You can click the first link at `http://investing.kuchita.com/2012/06/28/xiv-data-and-pricing-model-since-vix-futures-availabl` to download a zip file containing an Excel spreadsheet. Then, you will need to calculate whatever version of "contango" you prefer. I typically define contango as what percent higher the second-month VIX futures are acompared to the first-month futures, i.e. dividing the "2nd mth" column by the "1st mth" column, subtracting 1, and then multiplying by 100.

To load daily gains for XIV, SPXU, VXX, and UPRO, you can use `load_gains`, which uses the **quantmod** package to load data from Yahoo! Finance. You will have to specify the `from` and `to` inputs to match the date range for your contango values.

Value

List containing:

1. Character vector named `holdings` indicating what fund was held each day (XIV/SPXU, VXX/UPRO, or cash).
2. Numeric vector named `port.gains` giving the portfolio gain for each day, which will be 0 for days that cash was held and the weighted XIV/SPXU or VXX/UPRO gain for days that one of those positions was held.

3. Numeric vector named `port.balances` giving the portfolio balance each day.
4. Numeric value named `trades` giving the total number of trades executed.

`contango_simple` *Backtest a Simple Contango-Based Volatility Trading Strategy*

Description

Simple strategy: Each day, hold XIV if contango > `xiv.cutpoint`, hold VXX if contango < `vxx.cutpoint`, and hold cash otherwise. Perhaps not very useful since XIV closed on Feb. 20, 2018.

Usage

```
contango_simple(
  contango,
  xiv.gains = NULL,
  vxx.gains = NULL,
  xiv.cutpoint = 0,
  vxx.cutpoint = -Inf,
  initial = 10000
)
```

Arguments

`contango`	Numeric vector of contango values at the end of each trading day.
`xiv.gains`	Numeric vector of gains for XIV. Should be same length as `contango` and date-shifted one value to the right. For example, the first value of `xiv.gains` should be the XIV gain for the day AFTER the first contango value.
`vxx.gains`	Numeric vector of gains for VXX. Should be same length as `contango` and date-shifted one value to the right. For example, the first value of `vxx.gains` should be the VXX gain for the day AFTER the first contango value.
`xiv.cutpoint`	Numeric value giving the contango cutpoint for XIV, in percent.
`vxx.cutpoint`	Numeric value giving the contango cutpoint for VXX, in percent.
`initial`	Numeric value giving the initial value of the portfolio.

Details

You can find historical contango values from The Intelligent Investor Blog. You can click the first link at `http://investing.kuchita.com/2012/06/28/xiv-data-and-pricing-model-since-vix-futures-availabl` to download a zip file containing an Excel spreadsheet. Then, you will need to calculate whatever version of "contango" you prefer. I typically define contango as what percent higher the second-month VIX futures are acompared to the first-month futures, i.e. dividing the "2nd mth" column by the "1st mth" column, subtracting 1, and then multiplying by 100.

I think the most common approach for contango-based volatility strategies is holding XIV (inverse volatility) when contango is above some value (e.g. 0%, 5%, or 10%), and holding cash otherwise.

You can do that with this function by leaving `vxx.cutpoint` as `-Inf`. However, you may also want to hold VXX (volatility) when contango is below some value (e.g. 0%, -5%, -10%), also known as "backwardation". You can implement an XIV-only, VXX-only, or XIV and VXX strategy with this function.

To load daily gains for XIV and/or VXX, you can use `load_gains`, which uses the **quantmod** package to load data from Yahoo! Finance. You will have to specify the `from` and `to` inputs to match the date range for your contango values.

Value

List containing:

1. Character vector named `holdings` indicating what fund was held each day (XIV, VXX, or cash).
2. Numeric vector named `port.gains` giving the portfolio gain for each day, which will be 0 for days that cash was held and the XIV or VXX gain for days that XIV or VXX was held.
3. Numeric vector named `port.balances` giving the portfolio balance each day.
4. Numeric value named `trades` giving the total number of trades executed.

`convert_gain` *Convert Gain from One Time Interval to Another*

Description

For example, you can use this function to figure out that an 8% gain over 70 trading days corresponds to 31.9% annualized.

Usage

```
convert_gain(gain, units.in = 1, units.out = 1)
```

Arguments

`gain`	Numeric vector specifying each gain to convert, e.g. 0.005 for 0.5%.
`units.in`	Numeric value specifying the time period you want to convert from.
`units.out`	Numeric value specifying the time period you want to convert to.

Value

Numeric vector.

Examples

```
# Calculate annualized gain for an 8% gain over a 70-day period
convert_gain(gain = 0.08, units.in = 70, units.out = 252)

# Calculate the annual growth rate of a fund that gains 0.02% per day
convert_gain(gain = 0.0002, units.in = 1, units.out = 252)

# Calculate the annual growth rate of a fund that gains 1% per week
convert_gain(gain = 0.01, units.in = 1, units.out = 52)

# You invest in AAPL and gain 0.5% in 17 business days. Express as a 5-year
# growth rate.
convert_gain(gain = 0.005, units.in = 17, units.out = 252 * 5)

# Your portfolio has tripled in a 13-year period. Calculate your average
# annual gain.
convert_gain(gain = 2, units.in = 13, units.out = 1)
```

---

cum_metric *Calculate Cumulative Performance Metrics*

Description

Mainly a helper function for plot_metrics_overtime. Work in progress.

Usage

```
cum_metric(gains, metric = "mean", units.year = 252, benchmark.gains = NULL)
```

Arguments

gains	Numeric vector.
metric	Character string.
units.year	Integer value.
benchmark.gains	Numeric vector.

Value

Numeric vector.

daily_yearly *Convert Daily Gain to X-year Gain*

Description

For example, you can use this function to calculate that an investment that gains 0.1% each day would gain approximately 28.5% in a year (252 trading days).

Usage

```
daily_yearly(gain, years = 1)
```

Arguments

gain	Numeric vector specifying each gain to convert, e.g. 0.001 for 0.1%.
years	Numeric value.

Value

Numeric value or vector.

Examples

```
# Calculate annual gain for an investment that gains 0.1% per day
daily_yearly(gain = 0.001)

# Calculate 5-year gains corresponding to various daily gains
daily_yearly(gain = seq(0, 0.001, 0.0001), years = 5)
```

diffs *Lagged Differences (Alternate Implementation)*

Description

Calculates differences between subsequent (or lagged) elements of a vector. Very similar to diff, but written in C++.

Usage

```
diffs(x, lag = 1L)
```

Arguments

`x` Numeric vector.

`lag` Numeric value (e.g. 2 for differences between 1st and 3rd element, 2nd and 4th, ...).

Value

Numeric vector.

Examples

```
# Generate 1 million values from Poisson(3) distribution
x <- rpois(100000, 3)

# Calculate vector of differences between subsequent values
y <- diffs(x)

# Could get same result from base R function diff
z <- diff(x)
all.equal(y, z)

# But diffs is faster
benchmark(diffs(x), diff(x), replications = 100)
```

`fang` *Ticker Symbols for FANG Stocks (Facebook Apple, Netflix, Google)*

Description

Ticker Symbols for FANG Stocks (Facebook Apple, Netflix, Google)

`gains_prices` *Convert Sequence of Gains to Sequence of Prices*

Description

Converts sequence of gains and initial balance to sequence of prices for one or more investments.

Usage

```
gains_prices(gains, initial = 10000, date1 = NULL)
```

Arguments

`gains` Numeric vector of gains for one investment, or data frame with one column for each investment and an optional Date variable.

`initial` Numeric value.

`date1` Date to use for initial price.

Value

Numeric vector or data frame.

Examples

```
# Simulate daily gains over a 5-year period
set.seed(123)
gains <- rnorm(n = 252 * 5, mean = 0.001, sd = 0.02)

# Plot balance over time if initial balance is $10,000
prices <- gains_prices(gains)
plot(prices)
```

`gains_rate` *Calculate Growth Rate from Sequence of Gains*

Description

The formula is simply: `prod(gains + 1) - 1`. If `units.out` is specified, then it converts to x-unit growth rate.

Usage

```
gains_rate(gains, units.out = NULL)
```

Arguments

`gains` Data frame with one column of gains for each investment (can be a numeric vector if there is only one).

`units.out` Numeric value specifying the number of units for growth rate calculation, if you want something other than total growth. For annualized growth rate, set to 252 if `gains` has daily gains, 12 if `gains` has monthly gains, etc.

Value

Numeric vector.

Examples

```
# Create vector of daily gains for a hypothetical stock
daily.gains <- c(-0.02, -0.01, 0.01, 0.02, 0.01)

# Overall growth is 0.95%
gains_rate(daily.gains)

# Average daily growth is 0.19%
gains_rate(daily.gains, 1)

# Corresponds to 61.0% annual growth
gains_rate(daily.gains, 252)
```

`get_sp500_tickers` *Get S&P 500 Ticker Symbols as on a Particular Date*

Description

Scrapes ticker symbols from the Wikipedia Revision history https://en.wikipedia.org/wiki/List_of_S%26P_500_companies. Of course, the data may be imperfect.

Usage

```
get_sp500_tickers(date = Sys.Date())
```

Arguments

`date`	Date (or character vector that can be coerced).

Value

Character vector.

Examples

```
## Not run:
# S&P 500 tickers as of today
head(get_sp500_tickers())

# S&P 500 tickers at the beginning of 2019
head(get_sp500_tickers("2019-01-01"))

## End(Not run)
```

`highyield_etfs` *High-Yield ETFs from ETFdb.com*

Description

High-Yield ETFs from ETFdb.com

Source

https://etfdb.com/etfdb-category/high-yield-bonds/

`label_metric` *Convert Label back to Performance Metric*

Description

Mainly a helper function.

Usage

```
label_metric(label)
```

Arguments

`label`	Character string.

Value

Character string.

`largest_etfs` *Largest 100 Market Cap ETFs (as of 3/2/18) and Inception Dates*

Description

Largest 100 Market Cap ETFs (as of 3/2/18) and Inception Dates

Source

http://etfdb.com/compare/market-cap/

load_gains *Download Historical Gains*

Description

Downloads historical gains for specified tickers from Yahoo! Finance, with various options. Relies heavily on the **quantmod** package.

Usage

```
load_gains(
  tickers,
  intercepts = NULL,
  slopes = NULL,
  from = "1950-01-01",
  to = Sys.Date(),
  time.scale = "daily",
  preto.days = NULL,
  prefrom.days = NULL,
  mutual.lifetimes = TRUE,
  mutual.start = mutual.lifetimes,
  mutual.end = mutual.lifetimes,
  drop.anyNA = FALSE
)
```

Arguments

`tickers` Character vector of ticker symbols that Yahoo! Finance recognizes, or "^CASH" for cash.

`intercepts` Numeric vector of values to add to daily gains for each fund.

`slopes` Numeric vector of values to multiply daily gains for each fund by. Slopes are multiplied prior to adding intercepts.

`from` Date or character string, e.g. `"2015-01-15"`.

`to` Date or character string, e.g. `"2018-12-31"`.

`time.scale` Character string. Choices are `"daily"`, `"monthly"`, and `"yearly"`.

`preto.days` Numeric value. If specified, function returns gains for `preto.days` trading days prior to `to`. For example, to load the most recent 50 daily gains, leave `to` and `time.scale` as the defaults and set `preto.days = 50`.

`prefrom.days` Numeric value. If specified, function returns gains for `prefrom.days` trading days prior to `from`. Useful when you want to test a trading strategy starting on a particular date, but the strategy requires data leading up to that date (e.g. trailing beta).

`mutual.lifetimes`
Logical value for whether to start on the first day and end on the last day of the funds' mutual lifetimes (within `from` and `to`).

`mutual.start` Logical value for whether to start on the first day of the funds' mutual lifetimes.

`mutual.end` Logical value for whether to end on the last day of the funds' mutual lifetimes.

`drop.anyNA` Logical value for whether to drop dates on which prices are missing for any of the funds.

Value

Data frame with gains for each fund.

References

Jeffrey A. Ryan and Joshua M. Ulrich (2019). quantmod: Quantitative Financial Modelling Framework. R package version 0.4-15. https://CRAN.R-project.org/package=quantmod

Examples

```
## Not run:
# Load gains for Netflix and Amazon over their mutual lifetimes
gains <- load_gains(c("NFLX", "AMZN"))

## End(Not run)
```

`load_prices` *Download Historical Prices*

Description

Downloads historical prices for specified tickers from Yahoo! Finance, with various options. Relies heavily on the **quantmod** package.

Usage

```
load_prices(
  tickers,
  intercepts = NULL,
  slopes = NULL,
  from = "1950-01-01",
  to = Sys.Date(),
  time.scale = "daily",
  preto.days = NULL,
  prefrom.days = NULL,
  initial = NULL,
  mutual.lifetimes = TRUE,
  mutual.start = mutual.lifetimes,
  mutual.end = mutual.lifetimes,
  anchor = FALSE,
  drop.anyNA = FALSE
)
```

Arguments

`tickers` Character vector of ticker symbols that Yahoo! Finance recognizes, or "^CASH" for cash.

`intercepts` Numeric vector of values to add to daily gains for each fund.

`slopes` Numeric vector of values to multiply daily gains for each fund by. Slopes are multiplied prior to adding intercepts.

`from` Date or character string, e.g. `"2015-01-15"`.

`to` Date or character string, e.g. `"2018-12-31"`.

`time.scale` Character string. Choices are `"daily"`, `"monthly"`, and `"yearly"`.

`preto.days` Numeric value. If specified, function returns prices for `preto.days` trading days prior to `to`. For example, to load the most recent 50 closing prices, leave `to` and `time.scale` as the defaults and set `preto.days = 50`.

`prefrom.days` Numeric value. If specified, function returns prices for `prefrom.days` trading days prior to `from`. Useful when you want to test a trading strategy starting on a particular date, but the strategy requires data leading up to that date (e.g. trailing beta).

`initial` Numeric value specifying what value to scale initial prices to.

`mutual.lifetimes`
Logical value for whether to start on the first day and end on the last day of the funds' mutual lifetimes (within `from` and `to`).

`mutual.start` Logical value for whether to start on the first day of the funds' mutual lifetimes.

`mutual.end` Logical value for whether to end on the last day of the funds' mutual lifetimes.

`anchor` Logical value for whether to anchor the starting price for each fund to the price of the longest-running fund on that day. Useful for visualizing funds' entire histories while also fairly comparing them over their mutual lifetimes. Only used if `mutual.start = FALSE`.

`drop.anyNA` Logical value for whether to drop dates on which prices are missing for any of the funds.

Value

Data frame with closing prices for each fund.

References

Jeffrey A. Ryan and Joshua M. Ulrich (2019). quantmod: Quantitative Financial Modelling Framework. R package version 0.4-15. https://CRAN.R-project.org/package=quantmod

Examples

```
## Not run:
# Load prices for Netflix and Amazon over their mutual lifetimes
prices <- load_prices(c("NFLX", "AMZN"))

## End(Not run)
```

mdd *Maximum Drawdown*

Description

Calculates maximum drawdown from vector of closing prices, highs and lows, or gains. Missing values should be removed prior to calling this function.

Usage

```
mdd(prices = NULL, highs = NULL, lows = NULL, gains = NULL, indices = FALSE)
```

Arguments

prices	Numeric vector of daily closing prices.
highs	Numeric vector of daily high prices.
lows	Numeric vector of daily low prices.
gains	Data frame with one column of gains for each investment (extra non-numeric columns are ignored), or numeric vector for one investment.
indices	Logical value for whether to include indices for when the maximum drawdown occurred.

Value

Numeric value, vector, or matrix depending on `indices` and whether there is 1 fund or several.

Examples

```
## Not run:
# Calculate MDD's for FANG stocks in 2018
prices <- load_prices(c("FB", "AAPL", "NFLX", "GOOG"), from = "2018-01-01",
                      to = "2018-12-31")
sapply(prices[-1], mdd)

## End(Not run)
```

`metric_choices` *Performance Metric Choices*

Description

Performance Metric Choices

Source

Original

`metric_decimals` *Get Number of Decimals for Performance Metric*

Description

Mainly a helper function.

Usage

```
metric_decimals(metric)
```

Arguments

`metric`	Character string.

Value

Character string.

`metric_info` *Lookup Table for Performance Metrics*

Description

Lookup Table for Performance Metrics

Source

Original

metric_label *Get Label for Performance Metric*

Description

Mainly a helper function.

Usage

```
metric_label(metric)
```

Arguments

`metric`	Character string.

Value

Character string.

metric_title *Get Title for Performance Metric*

Description

Mainly a helper function.

Usage

```
metric_title(metric)
```

Arguments

`metric`	Character string.

Value

Character string.

`metric_units` *Get Units for Performance Metric*

Description

Mainly a helper function.

Usage

```
metric_units(metric)
```

Arguments

`metric`	Character string.

Value

Character string.

`moving_mean` *Moving Averages*

Description

Calculates moving averages or maximum moving average. For optimal speed, use `integer = TRUE` if `x` is an integer vector and `integer = FALSE` otherwise.

Usage

```
moving_mean(x, window, integer = FALSE, max = FALSE)
```

Arguments

`x`	Integer or numeric vector.
`window`	Integer value specifying window length.
`integer`	Logical value for whether `x` is an integer vector.
`max`	Logical value for whether to return maximum moving average (as opposed to vector of moving averages).

Value

Numeric value or vector depending on `max`.

Examples

```
# 5-unit moving average for integer vector of length 10
x <- rpois(10, lambda = 3)
moving_mean(x, 5)
```

---

pchanges *Lagged Proportion Changes*

Description

Calculates proportion changes between subsequent (or lagged) elements of a vector.

Usage

```
pchanges(x, lag = 1L)
```

Arguments

x	Numeric vector.
lag	Numeric value (e.g. 2 for differences between 1st and 3rd element, 2nd and 4th, ...).

Value

Numeric vector.

Examples

```
# Generate 10 values from N(0, 1)
x <- rnorm(10)

# Calculate vector of proportion changes between subsequent values
(y <- pchanges(x))

# Equivalent base R computation
len <- length(x)
p1 <- x[2: len]
p2 <- x[1: (len - 1)]
y2 <- p1 / p2 - 1
all.equal(y, y2)
```

pdiffs *Lagged Proportion Differences*

Description

Calculates proportion differences between subsequent (or lagged) elements of a vector.

Usage

```
pdiffs(x, lag = 1L)
```

Arguments

x	Numeric vector.
lag	Numeric value (e.g. 2 for differences between 1st and 3rd element, 2nd and 4th, ...).

Value

Numeric vector.

Examples

```
# Generate 10 values from N(0, 1)
x <- rnorm(10)

# Calculate vector of proportion differences between subsequent values
(y <- pdiffs(x))

# Equivalent base R computation
len <- length(x)
p1 <- x[2: len]
p2 <- x[1: (len - 1)]
y2 <- (p1 - p2) / (0.5 * (p1 + p2))
all.equal(y, y2)
```

plot_gains *Plot Gains for One Investment vs. Another*

Description

Useful for visualizing how two investments behave relate to each other, or how several investments behave relative to the same benchmark.

Usage

```
plot_gains(
  formula = NULL,
  ...,
  gains = NULL,
  prices = NULL,
  poly_order = 1,
  plotly = FALSE,
  title = NULL,
  base_size = 16,
  return = "plot"
)
```

Arguments

`formula`	Formula, e.g. `SSO + UPRO ~ SPY` to plot gains for SSO and UPRO vs. SPY.
`...`	Arguments to pass along with `tickers` to `load_gains`.
`gains`	Data frame with one column of gains for each investment mentioned in `formula`. If unspecified, function downloads historical gains internally.
`prices`	Data frame with one column of prices for each investment mentioned in `formula`.
`poly_order`	Numeric value specifying the polynomial order for linear regression, e.g. `1` for simple linear regression or `2` for linear regression with first- and second-order terms.
`plotly`	Logical value for whether to convert the `ggplot` to a `plotly` object internally. Note that legend displaying regression estimates will disappear if you choose this option.
`title`	Character string.
`base_size`	Numeric value.
`return`	Character string specifying what to return. Choices are `"plot"`, `"data"`, and `"both"`.

Value

In addition to the graph, a list containing fitted linear regression models returned by `lm` for each investment vs. the benchmark.

References

Jeffrey A. Ryan and Joshua M. Ulrich (2019). quantmod: Quantitative Financial Modelling Framework. R package version 0.4-15. https://CRAN.R-project.org/package=quantmod

Examples

```
## Not run:
# Plot daily gains for SSO and UPRO vs. VFINX
p <- plot_gains(SSO + UPRO ~ VFINX)
```

```
## End(Not run)
```

plot_growth *Plot Investment Growth*

Description

Useful for comparing the performance of several investments, over their full histories or mutual lifetimes.

Usage

```
plot_growth(
  prices = NULL,
  tickers = NULL,
  ...,
  gains = NULL,
  initial = 10000,
  plotly = FALSE,
  title = "Growth Over Time",
  base_size = 16,
  tooltip_size = 20,
  point_size = 1,
  line_size = 1,
  ticklabel_size = 8,
  legend_position = "right",
  return = "plot"
)
```

Arguments

`prices`	Data frame with one column of prices for each investment and a date variable named Date.
`tickers`	Character vector of ticker symbols that Yahoo! Finance recognizes, if you want to download data on the fly.
`...`	Arguments to pass along with `tickers` to `load_gains`.
`gains`	Data frame with one column of gains for each investment and a date variable named Date.
`initial`	Numeric value specifying value to scale initial prices to.
`plotly`	Logical value for whether to convert the `ggplot` to a `plotly` object internally.
`title`	Character string.
`base_size`	Numeric value to pass to `theme_gray`.
`tooltip_size`	Numeric value to pass to `style`.

`point_size` Numeric value to pass to `geom_point`.

`line_size` Numeric value to pass to `geom_line`.

`ticklabel_size` Numeric value to pass to `theme`.

`legend_position` Character string to pass to `theme`.

`return` Character string specifying what to return. Choices are `"plot"`, `"data"`, and `"both"`.

Value

Depending on `return` and `plotly`, a `ggplot`/`plotly` object, a data frame with the source data, or a list containing both.

A `ggplot` object.

Examples

```
## Not run:
# Plot growth of $10k in VFINX and BRK-B
plot_growth(tickers = c("VFINX", "BRK-B"))

## End(Not run)
```

---

plot_metrics

Plot One Performance Metric (Sorted Bar Plot) or One vs. Another (Scatterplot) for a Group of Individual Funds

---

Description

Useful for visualizing the performance of individual funds. For 2- and 3-fund portfolios, see `plot_metrics_2funds` and `plot_metrics_3funds`. To visualize any combination of single funds and 2- and 3-fund portfolios, see `link{plot_metrics_123}`.

Usage

```
plot_metrics(
  metrics = NULL,
  formula = cagr ~ mdd,
  tickers = NULL,
  ...,
  gains = NULL,
  prices = NULL,
  benchmark = "SPY",
  y.benchmark = benchmark,
  x.benchmark = benchmark,
  plotly = FALSE,
  title = NULL,
  base_size = 16,
  label_size = 5,
  ticklabel_size = 8,
  return = "plot"
)
```

Arguments

`metrics` "Long" data frame with Fund column and column for each metric you want to plot. Typically the result of a prior call to `calc_metrics`.

`formula` Formula specifying what to plot, e.g. `cagr ~ mdd` for CAGR vs. MDD, `cagr ~ .` for just CAGR, or `. ~ mdd` for just MDD. See `?calc_metrics` for list of metrics to choose from.

`tickers` Character vector of ticker symbols that Yahoo! Finance recognizes, if you want to download data on the fly.

`...` Arguments to pass along with `tickers` to `load_gains`.

`gains` Data frame with one column of gains for each investment and a date variable named Date.

`prices` Data frame with one column of prices for each investment and a date variable named Date.

`benchmark` Character string specifying which fund to use as a benchmark for metrics that require one.

`y.benchmark` Character string specifying which fund to use as benchmark for y-axis metric.

`x.benchmark` Character string specifying which fund to use as benchmark for x-axis metric.

`plotly` Logical value for whether to convert the `ggplot` to a `plotly` object internally.

`title` Character string.

`base_size` Numeric value.

`label_size` Numeric value.

`ticklabel_size` Numeric value.

`return` Character string specifying what to return. Choices are `"plot"`, `"data"`, and `"both"`.

Value

Depending on `return`, a `ggplot`, a data frame with the source data, or a list containing both.

References

Jeffrey A. Ryan and Joshua M. Ulrich (2019). quantmod: Quantitative Financial Modelling Framework. R package version 0.4-15. https://CRAN.R-project.org/package=quantmod

Examples

```
## Not run:
# Plot Sharpe ratio for FANG stocks
plot_metrics(formula = sharpe ~ ., tickers = fang)

# Create previous plot in step-by-step process with pipes
fang %>%
  load_gains() %>%
  calc_metrics("sharpe") %>%
  plot_metrics(. ~ sharpe)

# Plot CAGR vs. max drawdown for SPY and BRK-B
plot_metrics(formula = cagr ~ mdd, tickers = c("SPY", "BRK-B"))

# Create previous plot in step-by-step process with pipes
c("SPY", "BRK-B") %>%
  load_gains() %>%
  calc_metrics("cagr", "mdd") %>%
  plot_metrics(cagr ~ mdd)

## End(Not run)
```

plot_metrics_123 *Plot One Performance Metric vs. Another for Any Number of Single Funds, 2-Fund Portfolios, and 3-Fund Portfolios*

Description

Integrates plot_metrics, plot_metrics_2funds, and plot_metrics_3funds into a single function, so you can visualize strategies of varying complexities on one figure.

Usage

```
plot_metrics_123(
  metrics = NULL,
  formula = mean ~ sd,
  tickers = NULL,
  ...,
  step = 1,
  gains = NULL,
  prices = NULL,
  benchmark = "SPY",
  y.benchmark = benchmark,
  x.benchmark = benchmark,
  plotly = FALSE,
  title = NULL,
  base_size = 16,
  label_size = 5,
  return = "plot"
)
```

Arguments

metrics Data frame with Fund column and column for each metric you want to plot. Typically the result of a prior call to calc_metrics_123.

formula Formula specifying what to plot, e.g. mean ~ sd, cagr ~ mdd, or sharpe ~ allocation. See ?calc_metrics for list of metrics to choose from ("allocation" is an extra option here). If you specify metrics, default behavior is to use mean ~ sd unless either is not available, in which case the first two performance metrics that appear as columns in metrics are used.

tickers Character vector of ticker symbols, where the first three are are a three-fund set, the next three are another, and so on.

... Arguments to pass along with tickers to load_gains.

step Numeric value specifying fund allocation increments.

gains Data frame with a date variable named Date and one column of gains for each fund.

prices Data frame with a date variable named Date and one column of prices for each fund.

benchmark, y.benchmark, x.benchmark

Character string specifying which fund to use as benchmark for metrics (if you request alpha, alpha.annualized, beta, or r.squared).

plotly Logical value for whether to convert the ggplot to a plotly object internally.

title Character string.

base_size Numeric value.

label_size Numeric value.

return Character string specifying what to return. Choices are "plot", "data", and "both".

Details

If you prefer to have complete control over the plotting, you can set return = "data" to just get the source data.

Value

Depending on return, a ggplot object, a data frame, or a list containing both.

Examples

```
## Not run:
# Plot CAGR vs. max drawdown for BRK-B, SPY/TLT, and VWEHX/VBLTX/VFINX
plot_metrics_123(
  formula = cagr ~ mdd,
  tickers = list("BRK-B", c("SPY", "TLT"), c("VWEHX", "VBLTX", "VFINX"))
)

## End(Not run)
```

plot_metrics_2funds *Plot One Performance Metric vs. Another for 2-Fund Portfolios*

Description

Useful for visualizing the behavior of 2-fund portfolios, e.g. by plotting a measure of growth vs. a measure of volatility.

Usage

```
plot_metrics_2funds(
  metrics = NULL,
  formula = mean ~ sd,
  tickers = NULL,
  ...,
  points = seq(0, 100, 10),
  gains = NULL,
  prices = NULL,
  benchmark = "SPY",
  y.benchmark = benchmark,
  x.benchmark = benchmark,
  ref.tickers = NULL,
  plotly = FALSE,
  title = NULL,
  base_size = 16,
  label_size = 5,
  return = "plot"
)
```

Arguments

`metrics` Data frame with Fund column and column for each metric you want to plot. Typically the result of a prior call to `calc_metrics_2funds`.

`formula` Formula specifying what to plot, e.g. `mean ~ sd`, `cagr ~ mdd`, or `sharpe ~ allocation`. See `?calc_metrics` for list of metrics to choose from (`"allocation"` is an extra option here). If you specify `metrics`, default behavior is to use `mean ~ sd` unless either is not available, in which case the first two performance metrics that appear as columns in `metrics` are used.

`tickers` Character vector of ticker symbols, where the first two are are a two-fund pair, the next two are another, and so on.

`...` Arguments to pass along with `tickers` to `load_gains`.

`points` Numeric vector specifying allocations to include as points on the curve. Set to `NULL` for none (0 and 100 will still be included).

`gains` Data frame with a date variable named Date and one column of gains for each fund.

`prices` Data frame with a date variable named Date and one column of prices for each fund.

`benchmark`, `y.benchmark`, `x.benchmark`
Character string specifying which fund to use as benchmark for metrics (if you request `alpha`, `alpha.annualized`, `beta`, or `r.squared`).

`ref.tickers` Character vector of ticker symbols to include on the plot.

`plotly` Logical value for whether to convert the `ggplot` to a `plotly` object internally.

`title` Character string.

`base_size` Numeric value.

`label_size` Numeric value.

`return` Character string specifying what to return. Choices are `"plot"`, `"data"`, and `"both"`.

Value

Depending on `return`, a `ggplot` object, a data frame, or a list containing both.

Examples

```
## Not run:
# Plot mean vs. SD for UPRO/VBLTX, and compare to SPY
plot_metrics_2funds(
  formula = mean ~ sd,
  tickers = c("UPRO", "VBLTX")
)

# Plot CAGR vs. max drawdown for AAPL/GOOG and FB/TWTR
plot_metrics_2funds(
  formula = cagr ~ mdd,
  tickers = c("AAPL", "GOOG", "FB", "TWTR")
)

# Plot Sharpe ratio vs. allocation for SPY/TLT
plot_metrics_2funds(
  formula = sharpe ~ allocation,
  tickers = c("SPY", "TLT")
)

## End(Not run)
```

plot_metrics_3funds *Plot One Performance Metric vs. Another for 3-Fund Portfolios*

Description

Useful for visualizing the behavior of one or several 3-fund portfolios, e.g. by plotting a measure of growth vs. a measure of volatility.

Usage

```
plot_metrics_3funds(
  metrics = NULL,
  formula = mean ~ sd,
  tickers = NULL,
  ...,
  step = 2.5,
  gains = NULL,
  prices = NULL,
  benchmark = "SPY",
  y.benchmark = benchmark,
  x.benchmark = benchmark,
  ref.tickers = NULL,
  plotly = FALSE,
  title = NULL,
  base_size = 16,
  label_size = 5,
  return = "plot"
)
```

Arguments

`metrics` Data frame with Fund column and column for each metric you want to plot. Typically the result of a prior call to `calc_metrics_3funds`.

`formula` Formula specifying what to plot, e.g. `mean ~ sd`, `cagr ~ mdd`, or `sharpe ~ allocation`. See `?calc_metrics` for list of metrics to choose from (`"allocation"` is an extra option here). If you specify `metrics`, default behavior is to use `mean ~ sd` unless either is not available, in which case the first two performance metrics that appear as columns in `metrics` are used.

`tickers` Character vector of ticker symbols, where the first three are are a 3-fund set, the next three are another, and so on.

`...` Arguments to pass along with `tickers` to `load_gains`.

`step` Numeric value specifying fund allocation increments.

`gains` Data frame with a date variable named Date and one column of gains for each fund.

`prices` Data frame with a date variable named Date and one column of prices for each fund.

`benchmark`, `y.benchmark`, `x.benchmark`
Character string specifying which fund to use as benchmark for metrics (if you request `alpha`, `alpha.annualized`, `beta`, or `r.squared`).

`ref.tickers` Character vector of ticker symbols to include on the graph.

`plotly` Logical value for whether to convert the `ggplot` to a `plotly` object internally.

`title` Character string.

`base_size` Numeric value.

`label_size` Numeric value.

`return` Character string specifying what to return. Choices are `"plot"`, `"data"`, and `"both"`.

Value

Depending on `return`, a `ggplot` object, a data frame, or a list containing both.

Examples

```
## Not run:
# Plot mean vs. SD for UPRO/VBLTX/VWEHX
plot_metrics_3funds(
  formula = mean ~ sd,
  tickers = c("UPRO", "VBLTX", "VWEHX")
)

# Plot CAGR vs. max drawdown for FB/AAPL/NFLX and SPY/TLT/JNK
plot_metrics_3funds(
  formula = cagr ~ mdd,
  tickers = c("FB", "AAPL", "NFLX", "SPY", "TLT", "JNK")
)

# Plot Sharpe ratio vs. allocation for the same sets
plot_metrics_3funds(
  formula = sharpe ~ allocation,
  tickers = c("FB", "AAPL", "NFLX", "SPY", "TLT", "JNK")
)

## End(Not run)
```

`plot_metrics_overtime` *Plot One Performance Metric over Time or One vs. Another over Time*

Description

Useful for assessing how one or two performance metrics vary over time, for one or several funds. Supports fixed-width rolling windows, fixed-width disjoint windows, and disjoint windows on per-month or per-year basis.

Usage

```
plot_metrics_overtime(
  metrics = NULL,
  formula = cagr ~ .,
  type = "hop.year",
  minimum.n = 3,
  tickers = NULL,
  ...,
  gains = NULL,
  prices = NULL,
  benchmark = "SPY",
  y.benchmark = benchmark,
  x.benchmark = benchmark,
  plotly = FALSE,
  title = NULL,
  base_size = 16,
  return = "plot"
)
```

Arguments

`metrics` "Long" data frame with Fund column, Date column, and column for each metric you want to plot. Typically the result of a prior call to `calc_metrics_overtime`.

`formula` Formula specifying what to plot, e.g. `cagr ~ mdd` for CAGR vs. MDD or `cagr ~ .` for CAGR over time. See `?calc_metrics` for list of performance metrics to choose from.

`type` Character string or vector specifying type of calculation. Choices are (1) `"roll.n"` where n is a positive integer; (2) `"hop.n"` where n is a positive integer; (3) `"hop.month"`; (4) `"hop.year"`; and (5) vector of break-point dates, e.g. `c("2019-01-01", "2019-06-01")` for 3 periods. The "roll" and "hop" options correspond to rolling and disjoint windows, respectively.

`minimum.n` Integer value specifying the minimum number of observations per period, e.g. if you want to exclude short partial months at the beginning or end of the analysis period.

`tickers` Character vector of ticker symbols that Yahoo! Finance recognizes, if you want to download data on the fly.

`...` Arguments to pass along with `tickers` to `load_gains`.

`gains` Data frame with a date variable named Date and one column of gains for each investment.

`prices` Data frame with a date variable named Date and one column of prices for each investment.

`benchmark`, `y.benchmark`, `x.benchmark`
Character string specifying which fund to use as benchmark for metrics (if you request `alpha`, `alpha.annualized`, `beta`, or `r.squared`).

`plotly` Logical value for whether to convert the `ggplot` to a `plotly` object internally.

`title` Character string. Only really useful if you're going to set `plotly = TRUE`, otherwise you can change the title, axes, etc. afterwards.

`base_size` Numeric value.

`return` Character string specifying what to return. Choices are `"plot"`, `"data"`, and `"both"`.

Value

Depending on `return`, a `ggplot`, a data frame with the source data, or a list containing both.

Examples

```
## Not run:
# Plot net growth each year for BRK-B and SPY
plot_metrics_overtime(formula = growth ~ ., type = "hop.year", tickers = c("BRK-B", "SPY"))

# Create previous plot in step-by-step process with pipes
c("BRK-B", "SPY") %>%
  load_gains() %>%
  calc_metrics_overtime("growth", type = "hop.year") %>%
  plot_metrics_overtime(growth ~ .)

# Plot betas from 100-day disjoint intervals for a 2x daily (SSO) and 3x
# daily (UPRO) leveraged ETF
plot_metrics_overtime(formula = beta ~ ., type = "hop.100", tickers = c("SSO", "UPRO"))

# Create previous plot in step-by-step process with pipes
c("SPY", "SSO", "UPRO") %>%
  load_gains() %>%
  calc_metrics_overtime(metrics = "beta", type = "hop.100") %>%
  plot_metrics_overtime(formula = beta ~ .)

# Plot 50-day rolling alpha vs. beta for SSO and UPRO during 2018
plot_metrics_overtime(
  formula = alpha ~ beta,
  type = "roll.50",
  tickers = c("SSO", "UPRO"),
  from = "2018-01-01", to = "2018-12-31"
)

# Create previous plot in step-by-step process with pipes
c("SPY", "SSO", "UPRO") %>%
  load_gains(from = "2018-01-01", to = "2018-12-31") %>%
  calc_metrics_overtime(metrics = c("alpha", "beta"), type = "roll.50") %>%
  plot_metrics_overtime(alpha ~ beta)
```

```
## End(Not run)
```

prices_gains *Convert Sequence of Prices to Sequence of Gains*

Description

Converts sequence of prices to sequence of gains for one or more investments.

Usage

```
prices_gains(prices)
```

Arguments

prices	Numeric vector of prices for one investment or data frame with one column for each investment and an optional Date variable.

Value

Numeric vector or data frame.

Examples

```
## Not run:
# Load 2017 prices for Netflix and Amazon, and calculate growth of $10k
prices <- load_prices(c("NFLX", "AMZN"), initial = 1000)

# Calculate gains
gains <- prices_gains(prices)

## End(Not run)
```

`prices_rate` *Calculate Growth Rate From a Vector of Prices*

Description

The formula is simply: `prices[length(prices)] / prices[1] - 1`. If `units.rate` is specified, then it converts to x-unit growth rate.

Usage

```
prices_rate(prices, units.rate = NULL)
```

Arguments

`prices`	Numeric vector of prices or data frame with one column for each investment.
`units.rate`	Numeric value specifying the number of units for growth rate calculation, if you want something other than total growth. For annualized growth rate, set to 252 if `prices` has daily prices, 12 if `prices` has monthly prices, etc.

Value

Numeric value or vector.

Examples

```
## Not run:
# Load historical prices for SPY and TLT and then calculate growth rate
prices <- load_prices(tickers = c("SPY", "TLT"), mutual.start = TRUE)
prices_rate(prices)
# Plot mean vs. SD for UPRO/VBLTX/VWEHX
plot_metrics_3funds(mean ~ sd, tickers = c("UPRO", "VBLTX", "VWEHX"))

# Plot CAGR vs. MDD for FB/AAPL/NFLX and SPY/TLT/JNK
plot_metrics_3funds(cagr ~ mdd, tickers = c("FB", "AAPL", "NFLX", "SPY", "TLT", "JNK"))

# Plot Sharpe ratio vs. allocation for the same sets
plot_metrics_3funds(sharpe ~ allocation, tickers = c("FB", "AAPL", "NFLX", "SPY", "TLT", "JNK"))

## End(Not run)

# Create vector of daily closing prices for a hypothetical stock
prices <- c(100.4, 98.7, 101.3, 101.0, 100.9)

# Overall growth is 0.50%
prices_rate(prices)

# Average daily growth is 0.12%
prices_rate(prices, 1)
```

```
# Corresponds to 36.7% annualized growth
prices_rate(prices, 252)
```

ratios *Ratios of Subsequent Elements in a Vector*

Description

Calculates vector of ratios of a vector, i.e. ratio of `x[2]` to `x[1]`, ratio of `x[3]` to `x[2]`, and so forth.

Usage

```
ratios(x)
```

Arguments

x	Numeric vector.

Value

Numeric vector.

Examples

```
# Generate 10 values from N(0, 1)
x <- rnorm(10)

# Calculate vector of ratios
(y <- ratios(x))

# Slower base R computation
len <- length(x)
y2 <- x[2: len] / x[1: (len - 1)]
all.equal(y, y2)
```

rolling_metric *Calculate Moving-Window Performance Metrics*

Description

Mainly a helper function for plot_metrics_overtime.

Usage

```
rolling_metric(
  gains,
  metric = "mean",
  width = 50,
  units.year = 252,
  benchmark.gains = NULL
)
```

Arguments

gains	Numeric vector.
metric	Character string.
width	Integer value.
units.year	Integer value.
benchmark.gains	Numeric vector.

Value

Numeric vector.

rrr *Risk-Return Ratio*

Description

Calculates risk-return ratio, defined as growth rate divided by maximum drawdown.

Usage

```
rrr(prices = NULL, gains = NULL)
```

Arguments

prices	Numeric vector of prices.
gains	Numeric vector of gains.

Value

Numeric value.

Examples

```
# Simulate daily gains over a 5-year period
set.seed(123)
stock.gains <- rnorm(252 * 5, 0.0005, 0.01)

# Convert to daily balances assuming an initial balance of $10,000
daily.balances <- gains_prices(stock.gains + 1)

# Total return is about 1.23
daily.balances[length(daily.balances)] / daily.balances[1] - 1

# Maximum drawdown is about 0.19
mdd(prices = daily.balances)

# Ratio of these two is about 6.48
(daily.balances[length(daily.balances)] / daily.balances[1] - 1) /
mdd(daily.balances)

# Easier to calculate using rrr
rrr(daily.balances)
```

`sector_spdr_etfs` *Sector SPDR ETFs*

Description

Sector SPDR ETFs

Source

http://www.sectorspdr.com/sectorspdr/sectors/performance

`sharpe` *Sharpe Ratio*

Description

Calculates Sharpe ratio from vector of gains or prices. The formula is: `(mean(gains) - rf) / sd(gains)`, where `rf` is some risk-free rate of return.

Usage

```
sharpe(gains = NULL, prices = NULL, rf = 0)
```

Arguments

`gains`	Numeric vector of gains.
`prices`	Numeric vector of prices.
`rf`	Numeric value.

Value

Numeric value.

Examples

```
# Simulate daily gains over a 5-year period
set.seed(123)
stock.gains <- rnorm(252 * 5, 0.0005, 0.01)

# Calculate Sharpe ratio using risk-free return of 0
sharpe(stock.gains)
```

`sortino` *Sortino Ratio*

Description

Calculates Sortino ratio from vector of gains or prices. The formula is: `(mean(gains) - rf) / sd(gains[gains < 0])`, where `rf` is some risk-free rate of return.

Usage

```
sortino(gains = NULL, prices = NULL, rf = 0)
```

Arguments

`gains`	Numeric vector of gains.
`prices`	Numeric vector of prices.
`rf`	Numeric value.

Value

Numeric value.

Examples

```
# Simulate daily gains over a 5-year period
set.seed(123)
stock.gains <- rnorm(252 * 5, 0.0005, 0.01)

# Calculate Sortino ratio using risk-free return of 0
sortino(stock.gains)
```

`sp500_dates` *Lookup Table for Wikipedia S&P 500 Pages*

Description

Lookup Table for Wikipedia S&P 500 Pages

Source

Wikipedia

`stocks` *Stock Market Analysis*

Description

Functions for analyzing and visualizing stock market data. Main features are loading and aligning historical data, calculating performance metrics for individual funds or portfolios (e.g. annualized growth, maximum drawdown, Sharpe/Sortino ratio), and creating graphs.

Details

Package:	stocks
Type:	Package
Version:	2.0.0
Date:	2020-07-14
License:	GPL-3

See CRAN documentation for full list of functions and the GitHub page for an overview of the package with some examples.

Author(s)

Dane R. Van Domelen
`<vandomed@gmail.com>`

References

Jeffrey A. Ryan and Joshua M. Ulrich (2019). quantmod: Quantitative Financial Modelling Framework. R package version 0.4-15. https://CRAN.R-project.org/package=quantmod

---

targetall *Backtest a Fixed-Allocation Trading Strategy*

Description

Implements a trading strategy aimed at maintaining a fixed allocation to each of several funds, rebalancing when the effective allocations deviate too far from the targets.

Usage

```
targetall(
  tickers = NULL,
  intercepts = NULL,
  slopes = NULL,
  ...,
  tickers.gains = NULL,
  target.alls = NULL,
  tol = 0.05,
  rebalance.cost = 0,
  initial = 10000
)
```

Arguments

`tickers`	Character vector specifying 2 ticker symbols that Yahoo! Finance recognizes, if you want to download data on the fly.
`intercepts`	Numeric vector of values to add to daily gains for each fund.
`slopes`	Numeric vector of values to multiply daily gains for each fund by. Slopes are multiplied prior to adding intercepts.
`...`	Arguments to pass along with `tickers` to `load_gains`.
`tickers.gains`	Data frame with one column of gains for each investment and a date variable named Date.
`target.alls`	Numeric vector specifying target allocations to each fund. If unspecified, equal allocations are used (e.g. 1/3, 1/3, 1/3 if there are 3 funds).
`tol`	Numeric value indicating how far the effective allocations can drift away from the targets before rebalancing.
`rebalance.cost`	Numeric value specifying total cost of each rebalancing trade.
`initial`	Numeric value specifying what value to scale initial prices to.

Value

List containing:

1. Numeric matrix named `fund.balances` giving fund balances over time.
2. Numeric value named `rebalance.count` giving the number of rebalancing trades executed.

Examples

```
## Not run:
# Backtest equal-allocation UPRO/VBLTX/VWEHX strategy
port <- targetall(tickers = c("UPRO", "VBLTX", "VWEHX"))
plot(port$fund.balances[, "Portfolio"])

## End(Not run)
```

targetbeta_twofunds *Backtest a Two-Fund Strategy that Targets a Certain Beta*

Description

Implements a two-fund strategy where allocations to each fund are adjusted to maintain some user-specified portfolio beta. For example, you could back-test a zero-beta (i.e. market neutral) UPRO/VBLTX strategy using this function.

Usage

```
targetbeta_twofunds(
  tickers = NULL,
  intercepts = NULL,
  slopes = NULL,
  ...,
  benchmark.ticker = NULL,
  reference.tickers = NULL,
  tickers.gains = NULL,
  benchmark.gains = NULL,
  reference.gains = NULL,
  target.beta = 0,
  tol = 0.15,
  window.units = 50,
  failure.method = "closer",
  maxall.tol = tol - 0.05,
  initial = 10000
)
```

Arguments

`tickers` Character vector specifying 2 ticker symbols that Yahoo! Finance recognizes, if you want to download data on the fly.

`intercepts` Numeric vector of values to add to daily gains for each fund.

`slopes` Numeric vector of values to multiply daily gains for each fund by. Slopes are multiplied prior to adding intercepts.

`...` Arguments to pass along with `tickers` to `load_gains`.

`benchmark.ticker`
Character string specifying ticker symbol for benchmark index for calculating beta. If unspecified, the first fund in `tickers` is used as the benchmark.

`reference.tickers`
Character vector of ticker symbols to include on graph as data points for comparative purposes.

`tickers.gains` Data frame with one column of gains for each investment and a date variable named Date.

`benchmark.gains`
Numeric vector of gains for the benchmark index for calculating beta. If unspecified, the first fund in `tickers.gains` is used as the benchmark.

`reference.gains`
Numeric vector or matrix of gains for funds to include on graph as data points for comparative purposes.

`target.beta` Numeric value.

`tol` Numeric value specifying how far the effective portfolio beta has to deviate from `target.beta` to trigger a rebalancing trade.

`window.units` Numeric value specifying the width of the trailing moving window used to estimate each fund's beta.

`failure.method` Character string or vector specifying method(s) to use when fund betas are such that the target portfolio beta cannot be achieved. Choices are `"cash"`, `"fund1"`, `"fund2"`, `"fund1.maxall"`, `"fund2.maxall"`, `"inverse1"`, `"inverse2"`, and `"closer"`. See Details.

`maxall.tol` Numeric value specifying tolerance to use when implementing the `"fund1.maxall"` or `"fund2.maxall"` failure method. To illustrate, if `target.beta = 0`, fund 1 has a current beta of 1, fund 2 has a current beta of 0.25, `failure.method = "fund2.maxall"`, and `maxall.tol = 0.1`, a trade will be triggered that results in 40% fund 2 and 60% cash. The portfolio beta is `0.4 * 0.25 = 0.1`. The reason you might want `maxall.tol` to be less than `tol` is to avoid frequently triggering another trade on the very next day, as fund 2's beta changes a little and moves the portfolio beta outside of `[target.beta - tol, target.beta + tol]`.

`initial` Numeric value specifying what value to scale initial prices to.

Details

The general implementation is as follows. Beta for each of the two funds is estimated based on the first `window.units` gains. Initial allocations are selected to achieve portfolio beta of `target.beta`.

If that is not possible - for example, if target.beta = 0 and both funds have positive beta - then the action taken depends on what method is selected through the failure.method input (details below).

Assuming the target beta is attainable, the function moves over 1 day, and applies each fund's gains for that day. It then re-calculates each fund's beta based on the window.units-width interval, and determines the effective portfolio beta based on fund allocations and betas. If the effective beta is outside of [target.beta - tol, target.beta + tol], a rebalancing trade is triggered. As before, if the target beta cannot be achieved, certain actions are taken depending on the selected method.

When outside of a trade because the target beta could not be achieved, the function attempts to rebalance each time it shifts over to a new day, regardless of the effective portfolio beta.

When failure.method = "cash", the entire portfolio balance is allocated to cash when the target beta cannot be achieved.

When failure.method = "fund1" (or "fund2"), the entire portfolio balance is allocated to the first (or second) fund when the target beta cannot be achieved.

When failure.method = "fund1.maxall" (or "fund2.maxall"), when the target beta cannot be achieved, fund 1 (or fund 2) is combined with cash, with the fund 1 (fund 2) allocation as high as possible while staying within maxall.tol of target.beta.

When failure.method = "inverse1" (or "inverse2"), an inverse version of the first (or second) fund is used when the target beta cannot be achieved. In many cases where the target beta cannot be achieved with the two funds, it can be achieved with an inverse version of one and the other. If the target beta still cannot be achieved, the entire portfolio balance is allocated to cash.

When failure.method = "closer", the entire portfolio balance is allocated to whichever fund has a beta closer to target.beta.

Value

For each method, a 4-element list containing:

1. Numeric matrix named fund.balances giving fund balances over time.
2. Numeric matrix named fund.betas giving fund betas over time.
3. Numeric vector named effective.betas giving effective portfolio beta over time.
4. Numeric value named trades giving the total number of trades executed.

Examples

```
## Not run:
# Backtest zero-beta UPRO/VBLTX strategy
beta0 <- targetbeta_twofunds(tickers = c("UPRO", "VBLTX"), target.beta = 0)
plot(beta0$fund.balances$Portfolio)

## End(Not run)
```

ticker_dates *Get Yahoo! Finance Start/End Dates for Tickers*

Description

Useful for figuring out a time period over which to compare several funds.

Usage

```
ticker_dates(tickers, from = "1950-01-01", to = Sys.Date())
```

Arguments

`tickers`	Character vector with ticker symbols that Yahoo! Finance recognizes.
`from`	Date or character string (e.g. `"2015-01-15"`.
`to`	Date or character string (e.g. `"2016-01-30"`).

Value

Data frame with start and end dates for each fund.

Examples

```
## Not run:
# See what dates are available for AAPL and AMZN
ticker_dates(c("AAPL", "AMZN"))

## End(Not run)
```

title_metric *Convert Title back to Performance Metric*

Description

For internal use only.

Usage

```
title_metric(title)
```

Arguments

`title`	Character string.

Value

Character string.

`vanguard_etfs` *Vanguard ETF's*

Description

Vanguard ETF's

Source

https://investor.vanguard.com/mutual-funds/list#/mutual-funds/asset-class/month-end-returns

---

`vanguard_funds` *Vanguard Mutual Funds*

Description

Vanguard Mutual Funds

Source

https://investor.vanguard.com/mutual-funds/list#/mutual-funds/asset-class/month-end-returns

---

`vanguard_products` *Vanguard Products*

Description

Vanguard Products

Source

https://investor.vanguard.com/mutual-funds/list#/mutual-funds/asset-class/month-end-returns

Index