

Package ‘portfolioBacktest’

June 19, 2019

Title Automated Backtesting of Portfolios over Multiple Datasets

Version 0.1.0

Date 2019-06-19

Description Automated backtesting of multiple portfolios over multiple datasets of stock prices in a rolling-window fashion. Intended for researchers and practitioners to backtest a set of different portfolios, as well as by a course instructor to assess the students in their portfolio design in a fully automated and convenient manner, with results conveniently formatted in tables and plots. Each portfolio design is easily defined as a function that takes as input a window of the stock prices and outputs the portfolio weights. Multiple portfolios can be easily specified as a list of functions or as files in a folder. Multiple datasets can be conveniently extracted randomly from different markets, different time periods, and different subsets of the stock universe. The results can be later assessed and ranked with tables based on a number of performance criteria (e.g., expected return, volatility, Sharpe ratio, drawdown, turnover rate, return on investment, computational time, etc.), as well as plotted in a number of ways with nice barplots and boxplots.

Maintainer Daniel P. Palomar <daniel.p.palomar@gmail.com>

URL <https://CRAN.R-project.org/package=portfolioBacktest>,
<https://github.com/dppalomar/portfolioBacktest>

BugReports <https://github.com/dppalomar/portfolioBacktest/issues>

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 2.10)

Imports doSNOW, evaluate, foreach, PerformanceAnalytics, quantmod,
R.utils, snow, utils, xts, zoo

Suggests CVXR, DT, ggplot2, gridExtra, knitr, prettydoc, readtext,
rmarkdown, R.rsp, stringi, testthat

VignetteBuilder knitr, rmarkdown, R.rsp

NeedsCompilation no

Author Daniel P. Palomar [cre, aut],
Rui Zhou [aut]

Repository CRAN

Date/Publication 2019-06-19 11:40:02 UTC

R topics documented:

portfolioBacktest-package	2
backtestBoxPlot	3
backtestLeaderboard	4
backtestSelector	6
backtestSummary	7
backtestTable	8
dataset10	10
portfolioBacktest	10
SP500_symbols	12
stockDataDownload	13
stockDataResample	14
summaryBarPlot	15
summaryTable	16
Index	18

portfolioBacktest-package

portfolioBacktest: Automated Backtesting of Portfolios over Multiple Datasets.

Description

Automated backtesting of multiple portfolios over multiple datasets of stock prices in a rolling-window fashion. Intended for researchers and practitioners to backtest a set of different portfolios, as well as by a course instructor to assess the students in their portfolio design in a fully automated and convenient manner, with results conveniently formatted in tables and plots. Each portfolio design is easily defined as a function that takes as input a window of the stock prices and outputs the portfolio weights. Multiple portfolios can be easily specified as a list of functions or as files in a folder. Multiple datasets can be conveniently extracted randomly from different markets, different time periods, and different subsets of the stock universe. The results can be later assessed and ranked with tables based on a number of performance criteria (e.g., expected return, volatility, Sharpe ratio, drawdown, turnover rate, return on investment, computational time, etc.), as well as plotted in a number of ways with nice barplots and boxplots.

Functions

[stockDataDownload](#), [stockDataResample](#), [portfolioBacktest](#), [backtestSelector](#), [backtestTable](#), [backtestBoxPlot](#), [backtestLeaderboard](#), [backtestSummary](#), [summaryTable](#), [summaryBarPlot](#)

Data

[dataset10](#), [SP500_symbols](#)

Help

For a quick help see the README file: [GitHub-README](#).

For more details see the vignette: [GitHub-vignette](#).

Author(s)

Daniel P. Palomar and Rui ZHOU

backtestBoxPlot	<i>Create boxplot from backtest results.</i>
-----------------	--

Description

Create boxplot from a portfolio backtest obtained with the function [portfolioBacktest](#). By default the boxplot is based on the package [ggplot2](#) (also plots a dot for each single backtest), but the user can also specify a simple base plot.

Usage

```
backtestBoxPlot(bt, measure = "Sharpe ratio", type = c("ggplot2",
  "simple"), ...)
```

Arguments

bt	Backtest results as produced by the function portfolioBacktest .
measure	String to select a performane measure from "Sharpe ratio", "max drawdown", "annual return", "annual volatility", "Sterling ratio", "Omega ratio", and "ROT bps". Default is "Sharpe ratio".
type	Type of plot. Valid options: "ggplot2", "simple". Default is "ggplot2" (the package ggplot2 must be installed).
...	Additional parameters. For example: mar for margins as in par() (for the case of plot type = "simple"); and alpha for the alpha of each backtest dot (for the case of plot type = "ggplot2"), set to 0 to remove the dots.

Author(s)

Daniel P. Palomar and Rui Zhou

Examples

```

library(portfolioBacktest)
data(dataset10) # load dataset

# define your own portfolio function
quintile_portfolio <- function(data) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio), dataset10,
  benchmark = c("uniform", "index"))

# Now we can plot
backtestBoxPlot(bt, "Sharpe ratio")
backtestBoxPlot(bt, "Sharpe ratio", type = "simple")

```

backtestLeaderboard *Leaderboard of portfolios from the backtest results.*

Description

Leaderboard of portfolios according to the backtesting results and a ranking based on the combination of several performance criteria. Since the different performance measures have different ranges and distributions, each is first transformed according to its empirical distribution function (along the empirical distribution of the portfolios being ranked) to obtain percentile scores. After that transformation, each of the measures has an empirical uniform distribution in the interval $[0, 100]$ and can be weighted to obtain the final ranking.

Usage

```
backtestLeaderboard(bt = NA, weights = list(), summary_fun = median,
  show_benchmark = TRUE)
```

Arguments

bt	Backtest results as produced by the function <code>portfolioBacktest</code> .
weights	List of weights for the different performance measures as obtained in <code>backtestSummary()</code> \$performance (i.e., "Sharpe ratio", "max drawdown", "annual return", "annual volatility", "Sterling ratio", "Omega ratio", "ROT bps", "cpu_time", and "failure ratio"), as well as "cpu time" and "failure rate". For example: <code>weights = list("Sharpe ratio" = 8, "ma</code>

summary_fun Summary function to be employed (e.g., median or mean).
 show_benchmark Logical value indicating whether to include benchmarks in the summary (default is TRUE).

Value

List with the following elements:

leaderboard_scores Matrix with the individual scores for the portfolios (as chosen in weights) and the final score.
 leaderboard_performance Matrix with all the performance measures for the portfolios.
 error_summary Error messages generated by each portfolio on each dataset. Useful for debugging and give feedback to the portfolio managers of the different portfolios.

Author(s)

Daniel P. Palomar and Rui Zhou

Examples

```
library(portfolioBacktest)
data(dataset10) # load dataset

# define your own portfolio function
quintile_portfolio <- function(data) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio), dataset10,
  benchmark = c("uniform", "index"))

# see all performance measures available for the ranking
backtestSummary(bt)$performance

# show leaderboard
leaderboard <- backtestLeaderboard(bt, weights = list("Sharpe ratio" = 6,
  "max drawdown" = 1,
  "ROT bps" = 1,
  "cpu time" = 1,
  "failure rate" = 1))

leaderboard$leaderboard_scores
```

backtestSelector *Selector of portfolio backtest results.*

Description

Select the results from a portfolio backtest.

Usage

```
backtestSelector(bt, portfolio_index = NULL, portfolio_name = NULL,
  measures = NULL)
```

Arguments

bt	Backtest results as produced by the function portfolioBacktest .
portfolio_index	Index number of a portfolio, e.g., 1 means to select the performance of the first portfolio recorded in bt.
portfolio_name	String name of a portfolio, e.g., "GMVP" means to select the performance of portfolio with name "GMVP" in bt. Only considered when portfolio_index is not passed.
measures	String vector to select performane measures (default is all) from "Sharpe ratio", "max drawdown", "annual return", "annual volatility", "Sterling ratio", "Omega ratio", and "ROT bps".

Value

List with the following elements:

performance	Performance measures selected by argument measures.
error	Error status (TRUE or FALSE) of portfolio over each dataset (TRUE is when the portfolio function generates an error or the maximum CPU time is exceeded).
error_message	Error messages generated by portfolio function over each dataset. Useful for debugging purposes.
cpu_time	CPU usage by portfolio function over each dataset.
portfolio	Portfolio weights generated by portfolio function over each dataset.
return	Portfolio returns over each dataset.
cumPnL	Portfolio cumulative returns over each dataset.

Author(s)

Rui Zhou and Daniel P. Palomar

Examples

```

library(portfolioBacktest)
data("dataset10") # load dataset

# define your own portfolio function
uniform_portfolio <- function(dataset) {
  N <- ncol(dataset$adjusted)
  return(rep(1/N, N))
}

# do backtest
bt <- portfolioBacktest(list("Uniform" = uniform_portfolio), dataset10)

# extract your interested portfolio result
bt_sel <- backtestSelector(bt, portfolio_name = "Uniform")
names(bt_sel)

```

backtestSummary	<i>Summary of portfolio backtest.</i>
-----------------	---------------------------------------

Description

Summarize the results from a portfolio backtest.

Usage

```
backtestSummary(bt, portfolio_indexes = NA, portfolio_names = NA,
  summary_fun = median, show_benchmark = TRUE)
```

Arguments

bt	Backtest results as produced by the function portfolioBacktest .
portfolio_indexes	Numerical vector of portfolio indexes whose performance will be summarized, e.g., c(1, 2) means to summarize the performance of the first and second portfolios recorded in bt.
portfolio_names	String vector of portfolio names whose performance will be summarized, e.g., c("Uniform", "GMVP") means to summarize the performance of portfolios with names "Uniform" and "GMVP" in bt (default is names(bt) except the benchmark names). Only considered when portfolio_indexes is not passed.
summary_fun	Summary function to be employed (e.g., median or mean).
show_benchmark	Logical value indicating whether to include benchmarks in the summary (default is TRUE).

Value

List with the following elements:

performance_summary	Performance criteria: "Sharpe ratio", "max drawdown", "annual return", "annual volatility", "Sterling ratio", "Omega ratio", and "ROT bps". Default is "Sharpe ratio".
failure_rate	Failure rate of each portfolio (failure is when the portfolio function generates an error or the maximum CPU time is exceeded).
cpu_time_summary	Summary of the CPU usage by each portfolio function.
error_message	Error messages generated by each portfolio function over each dataset. Useful for debugging purposes.

Author(s)

Rui Zhou and Daniel P. Palomar

Examples

```
library(portfolioBacktest)
data(dataset10) # load dataset

# define your own portfolio function
uniform_portfolio <- function(dataset) {
  N <- ncol(dataset$adjusted)
  return(rep(1/N, N))
}

# do backtest
bt <- portfolioBacktest(list("Uniform" = uniform_portfolio), dataset10)

# show the summary
bt_sum <- backtestSummary(bt)
names(bt_sum)
bt_sum$performance_summary
```

backtestTable

Table with portfolio backtest results.

Description

Create table with the results from a portfolio backtest.

Usage

```
backtestTable(bt, portfolio_indexes = NA, portfolio_names = NA,
  show_benchmark = TRUE, measures = NULL)
```


Arguments

bt	Backtest results as produced by the function <code>portfolioBacktest</code> .
portfolio_indexes	Numerical vector of portfolio indexes whose performance will be summarized, e.g., <code>c(1, 2)</code> means to summarize the performance of the first and second portfolios recorded in <code>bt</code> .
portfolio_names	String vector of portfolio names whose performance will be summarized, e.g., <code>c("Uniform", "GMVP")</code> means to summarize the performance of portfolios with names "Uniform" and "GMVP" in <code>bt</code> (default is <code>names(bt)</code> except the benchmark names). Only considered when <code>portfolio_indexes</code> is not passed.
show_benchmark	Logical value indicating whether to include benchmarks in the summary (default is TRUE).
measures	String vector to select performane measures (default is all) from "Sharpe ratio", "max drawdown", "annual return", "annual volatility", "Sterling ratio", "Omega ratio", "ROT bps", "error", "cpu_time", and "error_message".

Value

List with the following elements:

<performance criterion>	One item per performance measures as selected by argument <code>measures</code> .
error	Error status (TRUE or FALSE) for each portfolio over each dataset (TRUE is when the portfolio function generates an error or the maximum CPU time is exceeded).
cpu_time	CPU usage by each portfolio function over each dataset.
error_message	Error messages generated by each portfolio function over each dataset. Useful for debugging purposes.

Author(s)

Rui Zhou and Daniel P. Palomar

Examples

```
library(portfolioBacktest)
data(dataset10) # load dataset

# define your own portfolio function
uniform_portfolio <- function(dataset) {
  N <- ncol(dataset$adjusted)
  return(rep(1/N, N))
}

# do backtest
bt <- portfolioBacktest(list("Uniform" = uniform_portfolio), dataset10)

# show the backtest results in table
```

```
bt_tab <- backtestTable(bt)
bt_tab[c("Sharpe ratio", "max drawdown")]
```

dataset10	<i>Ten datasets obtained by resampling the S&P 500.</i>
-----------	---

Description

Ten datasets of stock market data resampled from the S&P 500. Each resample contains a random selection of 50 stocks from the S&P 500 universe and a period of two years with a random initial point.

Usage

```
data(dataset10)
```

Format

List of 10 datasets, each contains two xts objects:

adjusted 505 x 50 xts with the adjusted prices of the 50 stocks

index 505 x 1 xts with the market index prices

Source

Yahoo! Finance

portfolioBacktest	<i>Backtest multiple portfolios over multiple datasets of stock prices in a rolling-window basis.</i>
-------------------	---

Description

Automated backtesting of multiple portfolios over multiple datasets of stock prices in a rolling-window fashion. Each portfolio design is easily defined as a function that takes as input a window of the stock prices and outputs the portfolio weights. Multiple portfolios can be easily specified as a list of functions or as files in a folder. Multiple datasets can be conveniently obtained with the function [stockDataResample](#) that resamples the data downloaded with the function [stockDataDownload](#). The results can be later assessed and arranged with tables and plots. The backtesting can be highly time-consuming depending on the number of portfolios and datasets can be performed with parallel computation over multiple cores. Errors in functions are properly caught and handled so that the execution of the overall backtesting is not stopped (error messages are stored for debugging purposes). See [vignette](#) for a detailed explanation.

Usage

```
portfolioBacktest(portfolio_funs = NULL, dataset_list,
  folder_path = NULL, price_name = "adjusted", paral_portfolios = 1,
  paral_datasets = 1, show_progress_bar = FALSE, benchmark = NULL,
  shortselling = TRUE, leverage = Inf, T_rolling_window = 252,
  optimize_every = 20, rebalance_every = 1, cpu_time_limit = Inf,
  return_portfolio = TRUE, return_returns = TRUE)
```

Arguments

portfolio_funs List of functions (can also be a single function), each of them taking as input a dataset containing a list of xts objects (following the format of each element of the argument `dataset_list`) properly windowed (following the rolling-window approach) and returning the portfolio as a vector of normalized weights. See [vignette](#) for details.

dataset_list List of datasets, each containing a list of xts objects, as generated by the function [stockDataResample](#).

folder_path If `portfolio_funs` is not defined, this should contain the path to a folder containing the portfolio functions saved in files. See [vignette](#) for details.

price_name Name of the xts column in each dataset that contains the prices to be used in the portfolio return computation (default is "adjusted").

paral_portfolios Interger indicating number of portfolios to be evaluated in parallel (default is 1).

paral_datasets Interger indicating number of datasets to be evaluated in parallel (default is 1).

show_progress_bar Logical value indicating whether to show progress bar (default is FALSE).

benchmark String vector indicating the benchmark portfolios to be incorporated, currently supports:

- `uniform` - the uniform portfolio, $w = [1/N, \dots, 1/N]$ with N be number of stocks
- `index` - the market index, requires an xts named 'index' in the datasets.

shortselling Logical value indicating whether shortselling is allowed or not (default is TRUE, so no control for shorselling in the backtesting).

leverage Amount of leverage as in $\|w\|_1 \leq leverage$ (default is Inf, so no control for leverage in the backtesting).

T_rolling_window Length of the lookback rolling window (default is 252).

optimize_every How often the portfolio is to be optimized (default is 20).

rebalance_every How often the portfolio is to be rebalanced (default is 1).

cpu_time_limit Time limit for executing each portfolio function over a single data set (default is Inf, so no time limit).

return_portfolio Logical value indicating whether to return portfolios (default is TRUE).

return_returns Logical value indicating whether to return the portfolio returns (default is TRUE).

Value

List with the portfolio backtest results, see [vignette-result-format](#) for details. It can be accessed directly, but we highly recommend the package specific functions to extract any required information, namely, [backtestSelector](#), [backtestTable](#), [backtestBoxPlot](#), [backtestLeaderboard](#), [backtestSummary](#), [summaryTable](#), [summaryBarPlot](#).

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

[stockDataDownload](#), [stockDataResample](#), [backtestSelector](#), [backtestTable](#), [backtestBoxPlot](#), [backtestLeaderboard](#), [backtestSummary](#), [summaryTable](#), [summaryBarPlot](#).

Examples

```
library(portfolioBacktest)
data(dataset10) # load dataset

# define your own portfolio function
uniform_portfolio <- function(dataset) {
  N <- ncol(dataset$adjusted)
  return(rep(1/N, N))
}

# do backtest
bt <- portfolioBacktest(list("Uniform" = uniform_portfolio), dataset10)

# check your result
names(bt)
backtestSelector(bt, portfolio_name = "Uniform", measures = c("Sharpe ratio", "max drawdown"))
backtestTable(bt, measures = c("Sharpe ratio", "max drawdown"))
bt_summary <- backtestSummary(bt)
summaryTable(bt_summary)
```

SP500_symbols

Stock symbols of the S&P 500 constituents.

Description

Stock symbols of the S&P 500 constituents.

Usage

```
data(SP500_symbols)
```

Format

String vector of stock symbols of the S&P 500 constituents. The market index symbol is concluded as the attribute "index_symbol".

Source

[Yahoo! Finance](#)

stockDataDownload	<i>Download stock data from the Internet.</i>
-------------------	---

Description

This function is basically a robust wrapper for [quantmod:getSymbols](#) to download stock data from the internet. It will return 6 xts objects of the same dimensions named 'open', 'high', 'low', 'close', 'volume', 'adjusted' and 'index'. Additionally, it can return an xts object with an index. If the download for some stock fails after a few attempts they will be ignored and reported. Also, stocks with missing values can be optionally removed.

Usage

```
stockDataDownload(stock_symbols, index_symbol = NULL, from, to,
  rm_stocks_with_na = TRUE, ...)
```

Arguments

stock_symbols	String vector containing the symbols of the stocks to be downloaded. User can pass the market index symbol as its attribute 'index_symbol' (only considered when argument 'index_symbol' is not passed).
index_symbol	String of the market index symbol.
from	String as the starting date, e.g., "2017-08-17".
to	String as the ending date (not included), e.g., "2017-09-17".
rm_stocks_with_na	Logical value indicating whether to remove stocks with missing values (ignoring leading missing values). Default is TRUE.
...	Additional arguments to be passed to quantmod:getSymbols .

Value

List of 7 xts objects named 'open', 'high', 'low', 'close', 'volume', 'adjusted' and 'index'. Note that 'index' will only be returned when correct index symbols is passed.

Author(s)

Rui Zhou and Daniel P. Palomar

See Also[stockDataResample](#)**Examples**

```
library(portfolioBacktest)
data(SP500_symbols)

# download data from internet
SP500_data <- stockDataDownload(stock_symbols = SP500_symbols,
                                from = "2008-12-01", to = "2009-12-01")
```

stockDataResample	<i>Generate random resamples from stock data.</i>
-------------------	---

Description

This function resamples the stock data downloaded by [stockDataDownload](#) to obtain many datasets for a subsequent backtesting with [portfolioBacktest](#). Given the original data, each resample is obtained by randomly choosing a subset of the stock names and randomly choosing a time period over the available long period.

Usage

```
stockDataResample(X, N_sample = 50, T_sample = 2 * 252,
                  num_datasets = 10, check_stocks_with_na = TRUE)
```

Arguments

X	List of xts objects matching the structure returned by the function stockDataDownload .
N_sample	Number of stocks in each resample.
T_sample	Length of each resample (consecutive samples with a random initial time).
num_datasets	Number of resampled datasets (chosen randomly among the stock universe).
check_stocks_with_na	Logical value indicating whether to quit in case of stocks with missing values (ignoring leading missing values). Default is TRUE.

Value

List of datasets resampled from X.

Author(s)

Rui Zhou and Daniel P. Palomar

See Also

[stockDataDownload](#), [portfolioBacktest](#)

Examples

```
library(portfolioBacktest)
data(SP500_symbols)

# download data from internet
SP500_data <- stockDataDownload(stock_symbols = SP500_symbols,
                                from = "2008-12-01", to = "2011-12-01")
# resample from downloaded, each with 50 stocks and 2-year continuous data
my_dataset_list <- stockDataResample(SP500_data, N = 50, T = 2*252, num_datasets = 10)
```

summaryBarPlot	<i>Create barplot from backtest summary.</i>
----------------	--

Description

After performing a backtest with [portfolioBacktest](#) and obtaining a summary of the performance measures with [backtestSummary](#), this function creates a barplot from the summary. By default the plot is based on the package `ggplot2`, but the user can also specify a simple base plot.

Usage

```
summaryBarPlot(bt_summary, measures = NULL, type = c("ggplot2",
  "simple"), ...)
```

Arguments

<code>bt_summary</code>	Backtest summary as obtained from the function <code>backtestSummary</code> .
<code>measures</code>	String vector to select performane measures (default is all) from ‘Sharpe ratio’, ‘max drawdown’, ‘annual return’, ‘annual volatility’, ‘Sterling ratio’, ‘Omega ratio’, and ‘ROT bps’.
<code>type</code>	Type of plot. Valid options: “ggplot2”, “simple”. Default is “ggplot2” (the package <code>ggplot2</code> must be installed).
<code>...</code>	Additional parameters (only used for plot type = “simple”); for example: <code>mar</code> for margins as in <code>par()</code> , <code>inset</code> for the legend inset as in <code>legend()</code> , <code>legend_loc</code> for the legend location as in <code>legend()</code> .

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

[summaryTable](#)

Examples

```
library(portfolioBacktest)
data(dataset10) # load dataset

# define your own portfolio function
quintile_portfolio <- function(data) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio), dataset10,
  benchmark = c("uniform", "index"))

# now we can obtain the table
bt_summary_median <- backtestSummary(bt)
summaryBarPlot(bt_summary_median, measures = c("max drawdown", "annual volatility"))
summaryBarPlot(bt_summary_median, measures = c("max drawdown", "annual volatility"),
  type = "simple")
```

summaryTable

Create table from backtest summary.

Description

After performing a backtest with [portfolioBacktest](#) and obtaining a summary of the performance measures with [backtestSummary](#), this function creates a table from the summary. By default the table is a simple matrix, but if the user has installed the package DT or grid, nicer tables can be generated.

Usage

```
summaryTable(bt_summary, measures = NULL, type = c("simple", "DT",
  "grid.table"), order_col = NULL, order_dir = c("asc", "desc"))
```


Arguments

bt_summary	Backtest summary as obtained from the function backtestSummary.
measures	String vector to select performane measures (default is all) from ‘Sharpe ratio’, ‘max drawdown’, ‘annual return’, ‘annual volatility’, ‘Sterling ratio’, ‘Omega ratio’, and ‘ROT bps’.
type	Type of table. Valid options: "simple", "DT", "grid.table". Default is "simple" and generates a simple matrix (with the other choices the corresponding package must be installed).
order_col	Column number of the performance measure to be used to sort the rows (only used for table type = "DT"). By default the last column will be used.
order_dir	Direction to be used to sort the rows (only used for table type = "DT"). Valid options: "asc", "desc". Default is "asc".

Author(s)

Daniel P. Palomar and Rui Zhou

See Also

[summaryBarPlot](#)

Examples

```
library(portfolioBacktest)
data(dataset10) # load dataset

# define your own portfolio function
quintile_portfolio <- function(data) {
  X <- diff(log(data$adjusted))[-1]
  N <- ncol(X)
  ranking <- sort(colMeans(X), decreasing = TRUE, index.return = TRUE)$ix
  w <- rep(0, N)
  w[ranking[1:round(N/5)]] <- 1/round(N/5)
  return(w)
}

# do backtest
bt <- portfolioBacktest(list("Quintile" = quintile_portfolio), dataset10,
  benchmark = c("uniform", "index"))

# now we can obtain the table
bt_summary_median <- backtestSummary(bt)
summaryTable(bt_summary_median, measures = c("max drawdown", "annual volatility"))
summaryTable(bt_summary_median, measures = c("max drawdown", "annual volatility"), type = "DT")
```

Index

*Topic **SP500_symbols**

SP500_symbols, [12](#)

*Topic **datasets**

dataset10, [10](#)

backtestBoxPlot, [3](#), [3](#), [12](#)

backtestLeaderboard, [3](#), [4](#), [12](#)

backtestSelector, [3](#), [6](#), [12](#)

backtestSummary, [3](#), [4](#), [7](#), [12](#), [15](#), [16](#)

backtestTable, [3](#), [8](#), [12](#)

dataset10, [3](#), [10](#)

portfolioBacktest, [3](#), [4](#), [6](#), [7](#), [9](#), [10](#), [14–16](#)

portfolioBacktest-package, [2](#)

quantmod:getSymbols, [13](#)

SP500_symbols, [3](#), [12](#)

stockDataDownload, [3](#), [10](#), [12](#), [13](#), [14](#), [15](#)

stockDataResample, [3](#), [10–12](#), [14](#), [14](#)

summaryBarPlot, [3](#), [12](#), [15](#), [17](#)

summaryTable, [3](#), [12](#), [16](#), [16](#)