

Package ‘portfolio’

April 19, 2009

Title Analysing equity portfolios

Version 0.4-4

Date 2008-10-31

Author Jeff Enos <jeff@kanecap.com> and David Kane <dave@kanecap.com>, with contributions from Daniel Gerlanc <daniel@gerlanc.com> and Kyle Campbell <Kyle.W.Campbell@williams.edu>

Description Classes for analysing and implementing equity portfolios.

Maintainer Jeff Enos <jeff@kanecap.com>

Depends R (>= 2.4.0), methods, graphics, grid, lattice, nlme

License GPL (>= 2)

LazyLoad yes

Repository CRAN

Date/Publication 2008-11-02 18:57:11

R topics documented:

portfolio-package	2
assay	3
contribution-class	4
dow.jan.2005	5
exposure-class	5
global.2004	6
map.market	7
matchedPortfolio-class	8
matchedPortfolioCollection-class	9
performance-class	10
portfolio-class	11
portfolioBasic-class	13
tradelist-class	15
trades-class	18
weight	19

portfolio-package *Analysing equity portfolios*

Description

Classes for analysing and implementing equity portfolios.

Details

Package: portfolio
 Version: 0.4-1
 Date: 2007-10-01
 Depends: R (>= 2.4.0), methods, graphics, grid, lattice, nlme
 License: GPL (>= 2)
 LazyLoad: yes
 Built: R 2.6.0; ; 2007-10-02 08:36:11; unix

Index:

assay	Assay Research rankings as of 2004-12-31
contribution-class	Class "contribution"
dow.jan.2005	DJIA for January, 2005
exposure-class	Class "exposure"
global.2004	Security data of large global companies for 2004
map.market	Create a Map of the Market
matchedPortfolio-class	Class "matchedPortfolio"
matchedPortfolioCollection-class	Class "matchedPortfolioCollection"
performance-class	Class "performance"
portfolio-class	Class "portfolio"
portfolio-package	Analysing equity portfolios
portfolioBasic-class	Class "portfolioBasic"
tradelist-class	Class "tradelist"
trades-class	Class "trades"
weight	Calculate Position Weights

Further information is available in the following vignettes:

matching_portfolio	Matching Portfolios (source, pdf)
portfolio	Using the portfolio package (source, pdf)
tradelist	Using the tradelist class (source, pdf)

Author(s)

Jeff Enos <jeff@kanecap.com> and David Kane <dave@kanecap.com>, with contributions from Daniel Gerlanc <daniel@gerlanc.com> and Kyle Campbell <Kyle.W.Campbell@williams.edu>

Maintainer: Jeff Enos <jeff@kanecap.com>

 assay

Assay Research rankings as of 2004-12-31

Description

A universe of the 5000 largest global stocks as of 2004-12-31, and a flag indicating whether a security was ranked by Assay Research at that time.

Special thanks to Assay Research for granting us permission to release this data.

Usage

data(assay)

Format

A data frame with 5000 observations on the following 11 variables.

date A vector containing a single Date: 2004-12-31.

id A character vector of SEDOLs and CUSIPs.

symbol A character vector of symbols.

name A character vector of the names of the companies.

country A factor with levels AUS AUT BEL CHE DEU DNK ESP FIN FRA GBR HKG ITA JPN NLD NOR NZL SGP SWE USA.

currency A factor with levels AUD CHF DKK EUR GBP HKD JPY NOK NZD SEK SGD USD.

price A numeric vector of prices.

sector A factor with levels Communications Conglomerates Cyclical Energy Financials Industrials Materials Staples Technology Utilities

sec An alternative sector specification. This factor has levels CND, CNS, COM, ENE, FIN, HTH, IND, MAT, TEC and UTL.

ind Industry specification. This factor has levels AERDF, AIRLN, AUTOP, AUTOS, BANKS, BEVGS, BIOTC, BUILD, CHEMS, CNENG, CNFIN, CNMAT, COMEQ, COMPT, COMSS, CONGL, CPMKT, DICNS, DISTR, DVFIN, DVTEL, ELEQI, ELEQT, ELUTL, ENEQS, FDPRD, FDRET, GSUTL, HEPSV, HEQSP, HETEC, HOTEL, HSDUR, HSPRD, INSUR, INTSS, IPPET, ITCAT, ITCON, LEISR, LFSCI, LOGIS, MACHN, MEDIA, METAL, MGFIN, MLRET, MLUTL, OFFIC, OILGS, PACKG, PAPER, PHARM, PRPRD, REALE, REDEV, REITS, RRAIL, SEMIP, SEMIS, SHIPS, SMOKE, SOFTW, SPRET, TEXAP, TRADE, TRINF, WIREL and WTUTL

liq A numeric vector of liquidities.

on.fl A logical vector indicating presence on the Assay Focus List as of 2004-12-31.

ret.0.1.m A numeric vector of one-month forward returns
ret.0.3.m A numeric vector of three-month forward returns
ret.0.6.m A numeric vector of one-month forward returns
ret.1.0.m A numeric vector of one-month prior returns
ret.6.0.m A numeric vector of six-month prior returns
ret.12.0.m A numeric vector of twelve-month prior returns
mn.dollar.volume.20.d A numeric vector of mean dollar volumes of the past 20 days
md.dollar.volume.120.d A numeric vector of median dollar volumes of the past 120 days
cap.usd A numeric vector of market capitalisation in USD.
cap A numeric vector of market capitalisation in local currency.
sales Annual gross sales of the company.
net.income Annual net income of the company.
common.equity Annual common equity of the company.

Examples

```
data(assay)
```

```
contribution-class Class "contribution"
```

Description

Portfolio contribution of numeric measures (intervals) and categories.

Objects from the Class

Objects can be created by calls of the form `new("contribution", ...)`.

Slots

data: Object of class "list" containing contributions, as data.frame objects. The names of this list correspond to the category variable names.

Methods

plot signature(x = "contribution", y = "missing"): Plot this object.
show signature(object = "contribution"): show this object, briefly.
summary signature(object = "contribution"): display a summary of this object.

Author(s)

Jeff Enos (jeff@kanecap.com)

dow.jan.2005	<i>DJIA for January, 2005</i>
--------------	-------------------------------

Description

Basic descriptive and market data for those securities in the DJIA as of the end of January, 2005.

Usage

```
data(dow.jan.2005)
```

Format

A data frame with 500 observations on the following 15 variables.

symbol a character vector

name a character vector

cap.bil a numeric vector

price a numeric vector

sector a factor with levels Communications Conglomerates Cyclical Energy Financials
 Industrials Materials Staples Technology Utilities

month.ret a numeric vector

Examples

```
data(dow.jan.2005)
```

exposure-class	<i>Class "exposure"</i>
----------------	-------------------------

Description

Portfolio exposures to numeric measures and categories.

Objects from the Class

Objects can be created by calls of the form `new("exposure", ...)`.

Slots

data: Object of class "list" containing exposures, as data.frame objects. The names of this list correspond to the exposure variable names. The special exposure "numeric" contains exposures to all numeric variables.

Methods

plot signature(x = "exposure", y = "missing"): Plot this object.
show signature(object = "exposure"): show the object, briefly.
summary signature(object = "exposure"): display a summary of this object.

Author(s)

Jeff Enos (jeff@kanecap.com)

global.2004

Security data of large global companies for 2004

Description

Contains basic security, category, and return information for a selection of large companies for each month of 2004. While 500 companies are included each month, the set of companies changes each month.

Usage

```
data(global.2004)
```

Format

A data frame with 6000 observations on the following 16 variables.

date a Date
id a character vector
symbol a character vector
name a character vector
country a factor with levels AUS AUT BEL CHE DEU DNK ESP FIN FRA GBR HKG ITA JPN
 NLD NOR SGP SWE USA
currency a factor with levels AUD CHF DKK EUR GBP HKD JPY NOK SEK SGD USD
cap a numeric vector
cap.usd a numeric vector
cap.bil a numeric vector
price a numeric vector
price.usd a numeric vector
round.lot a numeric vector
sector a factor with levels Communications Conglomerates Cyclical Energy Financials
 Industrials Materials Staples Technology Utilities
liquidity a numeric vector
liq.w a numeric vector
volume a numeric vector
avg.volume a numeric vector
ret.0.1.m a numeric vector

Examples

```
data(global.2004)
```

map.market	<i>Create a Map of the Market</i>
------------	-----------------------------------

Description

Utility function for creating a "map of the market" visualization. Creates a treemap where rectangular regions of different size, color, and groupings visualize the stocks in a portfolio.

Usage

```
map.market(id, area, group, color,  
           scale = NULL,  
           lab   = c(TRUE, FALSE),  
           main  = "Map of the Market",  
           print = TRUE)
```

Arguments

id	A vector storing the labels to be used for each stock.
area	A vector storing the values to be used to calculate the areas of rectangles.
group	A vector specifying the group (i.e. country, sector, etc.) to which each stock belongs.
color	A vector storing the values to be used to calculate the color of rectangles.
scale	An object of class <code>numeric</code> indicating the scale to be used in determining colors.
lab	A logical vector of length 2 specifying whether group and stock labels should be drawn. If the two values are the same, the second may be omitted.
main	A title for the plot.
print	An object of class <code>logical</code> indicating whether the map should be drawn.

Author(s)

Kyle Campbell <kyle.w.campbell@williams.edu> and Jeff Enos <jeff@kanecap.com>

Examples

```
data(dow.jan.2005)  
map.market(id   = dow.jan.2005$symbol,  
           area = dow.jan.2005$price,  
           group = dow.jan.2005$sector,  
           color = 100 * dow.jan.2005$month.ret)
```

```
matchedPortfolio-class
      Class "matchedPortfolio"
```

Description

An object of the class "matchedPortfolio" that contains an object of class "portfolioBasic" and a matrix of weights for portfolios that have been matched to the "portfolioBasic" according to variables specified in a formula.

Objects from the Class

Objects can be created by calls of the form `new("matchedPortfolio", ...)`.

Slots

formula: an object of class `formula` specifying the treatment variable and the covariates on which to match.

original: an object of class "portfolioBasic", the attributes of which will be used for matching.

matches: Object of class "matrix" with a column for each matched portfolio.

Methods

show signature(object = "matchedPortfolio"): prints basic information about the original portfolio and its matches.

summary signature(object = "matchedPortfolio"): prints detailed information about the original portfolio and its matches.

performance signature(object = "matchedPortfolio"): calculates the mean performance across all matched portfolios.

exposure signature(object = "matchedPortfolio", exp.var = "character"): calculates the exposure across each variable in `exp.var`.

contribution signature(object = "matchedPortfolio", contrib.var = "character"): calculates the contribution across each variable in `contrib.var`.

plot signature(x = "matchedPortfolio", y = "missing"): graphs exposure and contribution.

Details

The `matches` matrix contains as many rows as there are stocks in the `data` slot of `original` and as many columns as there are matched portfolios. The row labels of the matrix are the values of `original@data[["id.var"]]` and each column is a matched portfolio. The cell values are the weights of the stock in the portfolio.

Author(s)

Daniel Gerlanc <dgerlanc@gmail.com>

See Also

[portfolioBasic-class](#)

Examples

```
m.p <- new("matchedPortfolio")
```

```
matchedPortfolioCollection-class  
      Class "matchedPortfolioCollection"
```

Description

A collection of objects of class `matchedPortfolio`.

Objects from the Class

Objects can be created by calls of the form `new("matchedPortfolioCollection", ...)`.

Slots

data: Object of class "list" A list of objects of class `matchedPortfolio`.

Methods

No methods defined with class "matchedPortfolioCollection" in the signature.

Author(s)

Jeff Enos <jeff@kanecap.com>

performance-class *Class "performance"*

Description

Return, per-security return, and exposed portfolio object for one period.

Objects from the Class

Objects can be created by calls of the form `new("performance", ...)`.

Slots

ret: Object of class "numeric" containing the total return for the period.

profit: Object of class "numeric" containing the profit for the period, as a monetary amount.

missing.price: Object of class "numeric" containing the number of missing prices encountered during performance calculation.

missing.return: Object of class "numeric" containing the number of missing returns encountered during performance calculation.

ret.detail: Object of class "data.frame" containing the per-security return detail for the period.

t.plus.one: Object of class "portfolioBasic" containing the portfolio at the end of the period.

Methods

plot signature(x = "performance", y = "missing"): Plot this object.

show signature(object = "performance"): show this object, briefly.

summary signature(object = "performance"): display a summary of this object.

Author(s)

Jeff Enos (jeff@kanecap.com)

portfolio-class *Class "portfolio"*

Description

Class "portfolio" extends class "portfolioBasic" to include price and share information. Price information must be included in the supplementary "data" slot.

Objects from the Class

Objects can be created by calls of the form `new ("portfolio", ...)`.

Slots

- equity:** Object of class "numeric" containing the equity for this portfolio.
- file:** Object of class "character" containing the file from which this portfolio was loaded, if applicable.
- price.var:** Object of class "character" containing the name of the column in the "data" slot to be used in share and weight calculations.
- shares:** Object of class "data.frame" containing a data frame of shares for each position. Must have a unique column called "id".
- name:** Object of class "character" containing the name for this portfolio.
- instant:** Object of class "ANY" containing the instant to which the portfolio pertains.
- data:** Object of class "data.frame" containing supplementary information about the positions in this portfolio. Must include a unique "id" column.
- id.var:** Object of class "character" containing the name of the column in the data slot to be used as a unique identifier.
- symbol.var:** Object of class "character" containing the name of the column in the "data" slot to be used as a descriptive symbol.
- in.var:** Object of class "character" containing the name of the column in the "data" slot to be used as a rank vector in calls to `create`.
- weight.var:** Object of class "character" containing the name of the column in the "data" slot to be used as weight overrides in calls to `create`.
- weight.style:** Object of class "character" specifying how to calculate weights in this portfolio. Valid entries are:
 - "sides.separate": The default. Calculate weight of a position with respect to the total market value of positions on the same side.
 - "long.tmv": Calculate weight of a position with respect to the total market value of long positions.
 - "short.tmv": Calculate weight of a position with respect to the (positive) total market value of short positions.

- `reference.equity`: Calculate weight of a position relative to the reference equity in the equity slot. The equity slot must contain a numeric vector of length 1 for this style.

ret.var: Object of class "character" containing the name of the column in the "data" slot to be used as the return in calls to performance.

type: Object of class "character" containing the type of weight formation to use in calls to create. May be one of "relative", "equal", "linear", "sigmoid", "centroid", or "complex". Defaults to equal.

size: Object of class "characterOrNumeric" containing the size of the portfolio to use in calls to create. May either contain the number of securities per side or one of "decile", "quintile", "quartile", "tercile", or "demile". Defaults to quintile.

weights: Object of class "data.frame" containing the data frame of weights for this portfolio's positions. Must contain a unique column called "id".

Extends

Class "portfolioBasic", directly.

Methods

+ signature(e1 = "portfolio", e2 = "portfolio")

all.equal signature(target = "portfolio", current = "portfolio"): Compare two portfolio objects for "near equality". Two portfolio objects are all.equal iff they are all.equal as portfolioBasic objects, their shares slots contain exactly the same set of securities and shares vectors that are all.equal.

calcShares signature(object = "portfolio"): calculate shares from price and weight information, and store the results in the shares slot.

calcWeights signature(object = "portfolio"): calculate weights from share and price information, and store the results in the weights slot.

create signature(object = "portfolio"): create a portfolio object in the same manner as portfolioBasic, but also compute share amounts.

expandData signature(object = "portfolio"): ...

expose signature(object = "portfolio", trades = "trades"): ...

getYahooData signature(object = "portfolio", symbol.var = "character"): ...

performance signature(object = "portfolio"): ...

securityInfo signature(object = "portfolio", id = "character"): display information about position id within this portfolio.

getYahooData signature(object = "portfolio", symbol.var = "character"): Returns data for P/E Ratio, Book Value, Market Cap, Price/Book, and Price/Sales.

updatePrices signature(object = "portfolio", id = "character", price = "numeric"): ...

Author(s)

Jeff Enos (jeff@kanecap.com)

```
portfolioBasic-class
      Class "portfolioBasic"
```

Description

An object of the lightweight class "portfolioBasic" contains a data frame of weights and a data frame of supplementary information.

Objects from the Class

Objects can be created by calls of the form `new("portfolioBasic", ...)`.

Slots

- name:** Object of class "character" containing the name of this portfolio.
- instant:** Object of class "ANY" containing an instant to which this portfolio pertains.
- data:** Object of class "data.frame" containing supplementary information about the positions in this portfolio. Must include a unique column specified in the `id.var` slot.
- id.var:** Object of class "character" containing the name of the column in the data slot to be used as a unique identifier.
- symbol.var:** Object of class "character" containing the name of the column in the data slot to be used as a descriptive symbol.
- in.var:** Object of class "character" containing the name of the column in the data slot to be used as a rank vector in calls to `create`.
- weight.var:** Object of class "character" containing the name of the column in the data slot to be used as weight overrides in calls to `create`.
- ret.var:** Object of class "character" containing the name of the column in the data slot to be used as the return in calls to `performance`.
- type:** Object of class "character" containing the type of weight formation to use in calls to `create`. May be one of "relative", "equal", "linear", "sigmoid", "centroid", or "complex". Defaults to `equal`.
- size:** Object of class "characterOrNumeric" containing the size of the portfolio to use in calls to `create`. May either contain the number of securities per side or one of "decile", "quintile", "quartile", "tercile", or "demile". Defaults to `quintile`.
- weights:** Object of class "data.frame" containing the data frame of weights for this portfolio's positions. Must contain a unique column called "id".

Methods

- + `signature(e1 = "portfolioBasic", e2 = "portfolioBasic")`
- all.equal** `signature(target = "portfolioBasic", current = "portfolioBasic")`:
Compare two `portfolioBasic` objects for "near equality". Two `portfolioBasic` objects are `all.equal` iff their `weights` slots contain exactly the same set of securities and weight vectors that are `all.equal`.

- balance** signature(object = "portfolioBasic", in.var = "character"): balances the positions in portfolio object to be neutral to the categories specified by column in.var in the data slot.
- contribution** signature(object = "portfolioBasic", contrib.var = "character"): returns one data.frame with contribution analysis for each element of contrib.var. All results are returned in a list.
- create** signature(object = "portfolioBasic"): use this object's creation parameters (such as in slots size and type) to create and return a new object of class portfolioBasic.
- exposure** signature(object = "portfolioBasic", exp.var = "character"): returns one data.frame with exposure analysis for each element of contrib.var. All results are returned in a list.
- matching** signature(object = "portfolioBasic", covariates = "character"): returns a matchedPortfolio object containing n.matches matched portfolios. object is the portfolioBasic to be matched. covariates is a character vector of the attributes on which to match.
- performance** signature(object = "portfolioBasic"): returns a list containing performance results.
- plot** signature(x = "portfolioBasic", y = "missing"): Plot this object.
- portfolioDiff** signature(object = "portfolioBasic", x = "portfolioBasic"): computes the difference, as a portfolioBasic object, between two portfolios.
- scaleWeights** signature(object = "portfolioBasic"): scale weights to the weights supplied in the target parameter. To restrict the set of positions whose weights are scaled, use the condition argument.
- show** signature(object = "portfolioBasic"): display this object, briefly.
- summary** signature(object = "portfolioBasic"): display descriptive information about this portfolio.
- initialize** signature(object = "portfolioBasic"): initialize the portfolio by calling create.
- mapMarket** signature(object = "portfolioBasic"): create a map of the market plot of the portfolio.

Matched portfolios

The matching method allows one to benchmark a portfolio against a similar portfolio formed from other stocks in the universe. The universe consists of all the stocks in the data slot of original.

matching calculates a propensity score for each stock in the universe. covariates determines which attributes are used to calculate the propensity score. covariates must refer to the names of columns in the data slot of original.

Matching accepts an optional argument, method, which sets the algorithm for determining the best match for each stock. There are 2 available algorithms, "greedy" and "sample". "greedy" is the default and generates 1 matched portfolio. "sample" randomly matches each stock in original with one of the stocks in the universe. Although the matching is random, stocks in original are most likely to be matched with stocks having similar propensity scores.

`n.matches` is another optional argument to `matching` which determines the number of matched portfolios to generate. Requesting more than 1 matched portfolio. (`n.matches > 1`) while using `greedy` is not allowed. When using `sample`, there is no bound on `n.matches`.

Author(s)

Jeff Enos (jeff@kanecap.com) with contributions from Daniel Gerlanc (dgerlanc@gmail.com)

Examples

```
data(dow.jan.2005)

p <- new("portfolioBasic",
        id.var = "symbol",
        in.var = "price",
        sides = "long",
        ret.var = "month.ret",
        data = dow.jan.2005)

summary(p)

exposure(p, exp.var = c("price", "sector"))
performance(p)
contribution(p, contrib.var = c("cap.bil", "sector"))

p <- new("portfolioBasic",
        id.var = "symbol",
        in.var = "price",
        type = "linear",
        sides = c("long", "short"),
        ret.var = "month.ret",
        data = dow.jan.2005)

summary(p)

exposure(p, exp.var = c("price", "sector"))
performance(p)
contribution(p, contrib.var = c("cap.bil", "sector"))
```

tradelist-class *Class "tradelist"*

Description

Note: This class is a rough first pass and will change drastically in future releases.

An object of the class "tradelist" containing a data frame of trades and a data frame of supplementary information.

Objects from the Class

Objects can be created by calls of the form `new("tradelist", orig, target, ...)`.

Slots

type: Object of class "character" specifying the type of the tradelist. Must be "all" or "ranks".

id.var: Object of class "character" containing the name of the column in the data slot to be used as a unique identifier.

candidates: Object of class "data.frame" containing one row for each candidate trade.

ranks: Object of class "data.frame" where candidate trades have been interleaved by trade type (B,S,X,C) and assigned a unique rank, "rank.t".

chunks: Object of class "data.frame" that contains one row for each chunk, a smaller portion of an order.

swaps: Object of class "data.frame" where buys and sells have been matched with other shorts and covers of similar market value and desirability.

swaps.actual: Object of class "data.frame" where the least desirable chunks that would exceed if "turnover" if ordered have been removed.

actual: Object of class "data.frame" where the chunks have been rolled up into one row/order per security.

final: Object of class "trades" containing the most basic information on the set of trades in the tradelist.

chunks.actual: Object of class "data.frame" where "swaps" have been turned back into chunks and each chunk has its own row.

sorts: Object of class "optionalList" which may be interpreted as a list of key-value pairs. The key is the name of the sort and must exist as a column in the "data" slot. The numeric value expresses the relative weight of the sort.

rank.sorts: Object of class "list", where the names of the elements are the names of the sorts defined in the "sorts" list and the elements are data frames, each of which contains a ranking of the candidate trades created by applying an individual sort.

regions: Object of class "character"

chunk.usd: Object of class "numeric" that expresses the minimum unsigned market value in US dollars of a chunk. Defaults to \$10,000.

trade.usd.min: Object of class "numeric" that expresses the minimum unsigned market value a trade must have in order to be placed. Trades of lower market value are removed from the "candidates" data.frame and appended to the "restricted" data frame.

restrictions: Object of class "data.frame" with 1 row for each trade and three columns, "id", "type", and "reason". "id" uniquely identifies the trade, "type" accepts a value of B, S, C, or X (buy, sell, cover, or short), expressing the type of trade that is prohibited, and "reason" is a label expressing why the restriction exists.

restricted: Object of class "data.frame" that contains one row for every trade for which a restriction exists

- to.equity:** Object of class "logical" expressing whether or not the algorithm should trade towards the value of `target.equity`
- turnover:** Object of class "numeric" that expresses the maximum unsigned market value of all trades effected in one session.
- tca:** Object of class "character" expressing whether or not to use trade cost adjustment.
- target.equity:** Object of class "numeric" expressing the unsigned market value of the target portfolio.
- mv.long.orig:** Object of class "numeric" The market value of the long side of the original portfolio.
- mv.short.orig:** Object of class "numeric" The unsigned market value of the short side of the original portfolio
- unrestricted:** Object of class "logical" specifying whether any restrictions should be applied, including checks for price and volume.
- data:** Object of class "data.frame" containing supplementary information about the "tradelist". Must contain an "id" column, a "price.usd" column, a "volume" column, and a column named after each element listed in "sorts".

Methods

- actualCols** signature(object = "tradelist"): Returns a vector with the following elements: "id", "side", "shares", "mv", `names(object@sorts)`, and "rank.t"
- calcActual** signature(object = "tradelist"): Rolls up the chunks calculated in `calcChunksActual` into single orders and stores the result as a data frame in the "actual" slot.
- calcCandidates** signature(object = "tradelist", orig = "portfolio", target = "portfolio"): Builds a data frame of candidate trades with one row per trade by determining which positions have different numbers of shares in the original and target portfolios. Removes trades in the "restrictions" data frame, trades with a market value below "trade.usd.min", and trades that would cause a side change in one session, and appends these trades to the "restricted data frame."
- calcChunksActual** signature(object = "tradelist"): Turns the swaps calculated in `calcSwapsActual` back into chunks and stores the results in a data frame in the "actual.chunks" slot.
- calcChunks** signature(object = "tradelist"): Examines the data frame stored in the "ranks" slot, breaks the candidate trades into chunks of size "chunk.usd" or smaller, and stores the results in the "chunks" slot
- calcSwapsActual** signature(object = "tradelist"): Examines the data frame stored in the "swaps" slot and removes swaps, which had they been processed as orders, would have exceeded the specified "turnover" of the tradelist. Stores the results as a data frame in the "swaps.actual" slot.
- calcSwaps** signature(object = "tradelist"): Using the "chunks" data frame created by the `calcChunks` method, pairs attractive chunks with other attractive chunks of similar market value and stores the results as a data frame in the swaps slot.
- calcRanks** signature(object = "tradelist"): Using information from the `calcCandidates` data frame, interleaves the trades and calculates an absolute rank for each trade, "rank.t". Stores the results in the ranks slot

- candidatesCols** signature(object = "tradelist"): Returns a vector of class character containing the following elements: "id", "orig", "target", "side", "shares", "mv"
- chunksCols** signature(object = "tradelist"): Returns a vector of class character containing the following elements: rankCols(object), "tca.rank", "chunk.shares", "chunk.mv", "chunk".
- dummyChunks** signature(object = "tradelist"): Creates a data frame of dummy chunks for a given side and dollar amount (total.usd). The supplied dollar amount, together with the tradelist object's chunk size, determines the number of rows in the resulting data frame.
- initialize** signature(.Object = "tradelist"): Transparently calls calcCandidates, calcRanks, calcChunks, calcSwaps, calcSwapsActual, calcChunksActual, calcActual to construct the tradelist object.
- ranksCols** signature(object = "tradelist"): Returns a vector of class character containing the following elements: "id", "orig", "target", "side", "shares", "mv", names(object@sorts), "rank.t"
- restrictedCols** signature(object = "tradelist"): Returns a vector of class character containing the following elements: candidatesCols(object), "reason".
- securityInfo** signature(object = "tradelist", id = "character"): Returns detailed information regarding a security in the tradelist.
- show** signature(object = "tradelist"): Prints a detailed summary of tradelist attributes.
- trimSide** signature(object = "tradelist"): If the market value of the side passed as the "side" parameter to this function is greater than market value of the side as specified by the "value" parameter, excises the least desirable trades on that side until the the market value of that side is less than value. Returns a copy of the data frame stored in the "actual" slot with the trades that meet the forementioned conditions removed.

Author(s)

Daniel Gerlanc <daniel@gerlanc.com>

trades-class

Class "trades"

Description

An object of the class "trades" contains a data frame with columns "id", "side", and "shares" describing a simple list of trades to be performed.

Objects from the Class

Objects can be created by calls of the form `new("trades", ...)`.

Slots

trades: Object of class "data.frame" with columns "id", "side", and "shares".

Methods

No methods defined with class "trades" in the signature.

Author(s)

Kyle Campbell and Daniel Gerlanc

See Also

[tradelist-class](#)

Examples

```
df <- data.frame(id = c(1,2,3), side = c("B","X","C"), shares = c(10,20,30))
t <- new("trades", trades = df)
```

weight

Calculate Position Weights

Description

Compute position weights of various types from an input variable.

Usage

```
weight(x, in.var, type, size, sides,
       weight.var = NULL, verbose = FALSE)
```

Arguments

x	A data.frame containing the columns in.var and weight.var (if necessary).
in.var	Character vector specifying the column in x that contains a ranking from which weights can be computed.
type	Character vector specifying the method to use for weight creation. Can be one of c("relative", "equal", "linear", "sigmoid", "centroid", "complex").
size	Character or numeric vector specifying the number of desired non-na weights per side in the result. Can either be a positive number or one of "all", "decile", "quintile", "tercile", "demile").
sides	Character vector specifying the sides for which to create weights. May be any nonempty subset of "long", "short".

`weight.var` Numeric vector containing specifying the column in `x` that contains weight overrides. Overrides are applied after weights have been computed. Defaults to `NULL`.

`verbose` Be verbose. Defaults to `FALSE`.

Value

A numeric vector of weights the same length as `x`.

Author(s)

Jeff Enos (jeff@kanecap.com)

Examples

```
data <- data.frame(in.var = 1:50, weight.var = NA)
data$in.var <- as.numeric(data$in.var)

weight(data, in.var = "in.var", type = "linear", size = "quintile",
        sides = c("long", "short"))

data$weight.var[25] <- -0.05
weight(data, in.var = "in.var", type = "linear", size = "quintile",
        sides = c("long", "short"), weight.var = "weight.var")
```

Index

*Topic **classes**

contribution-class, 4
exposure-class, 5
matchedPortfolio-class, 8
matchedPortfolioCollection-class, 9
performance-class, 10
portfolio-class, 11
portfolioBasic-class, 13
tradelist-class, 15
trades-class, 18

*Topic **datasets**

assay, 2
dow.jan.2005, 5
global.2004, 6

*Topic **file**

map.market, 7
weight, 19

*Topic **package**

portfolio-package, 1

+, portfolio, portfolio-method
(*portfolio-class*), 11
+, portfolioBasic, portfolioBasic-method
(*portfolioBasic-class*), 13

actualCols, tradelist-method
(*tradelist-class*), 15

all.equal, portfolio, portfolio-method
(*portfolio-class*), 11

all.equal, portfolioBasic, portfolioBasic-method
(*portfolioBasic-class*), 13

assay, 2

balance (*portfolioBasic-class*), 13

balance, portfolioBasic, character-method
(*portfolioBasic-class*), 13

calcActual, tradelist-method
(*tradelist-class*), 15

calcCandidates, tradelist, portfolio, portfolio-method
(*tradelist-class*), 15

calcChunks, tradelist-method
(*tradelist-class*), 15

calcChunksActual, tradelist-method
(*tradelist-class*), 15

calcRanks, tradelist-method
(*tradelist-class*), 15

calcShares (*portfolio-class*), 11

calcShares, portfolio-method
(*portfolio-class*), 11

calcSwaps, tradelist-method
(*tradelist-class*), 15

calcSwapsActual, tradelist-method
(*tradelist-class*), 15

calcWeights (*portfolio-class*), 11

calcWeights, portfolio-method
(*portfolio-class*), 11

candidatesCols, tradelist-method
(*tradelist-class*), 15

chunksCols, tradelist-method
(*tradelist-class*), 15

contribution
(*portfolioBasic-class*), 13

contribution, matchedPortfolio, character-method
(*matchedPortfolio-class*), 8

contribution, portfolio, character-method
(*portfolio-class*), 11

contribution, portfolioBasic, character-method
(*portfolioBasic-class*), 13

contribution-class, 4

create (*portfolioBasic-class*), 13
create, portfolio-method
(*portfolio-class*), 11

create, portfolioBasic-method
(*portfolioBasic-class*), 13

dow.jan.2005, 5

dummyChunks, tradelist-method
(*tradelist-class*), 15

- expandData (*portfolio-class*), 11
- expandData, portfolio-method (*portfolio-class*), 11
- expose (*portfolio-class*), 11
- expose, portfolio, trades-method (*portfolio-class*), 11
- exposure (*portfolioBasic-class*), 13
- exposure, matchedPortfolio, character-method (*matchedPortfolio-class*), 8
- exposure, portfolioBasic, character-method (*portfolioBasic-class*), 13
- exposure-class, 5
- getYahooData (*portfolio-class*), 11
- getYahooData, portfolio, character-method (*portfolio-class*), 11
- global.2004, 6
- initialize, performance-method (*performance-class*), 10
- initialize, portfolio-method (*portfolio-class*), 11
- initialize, portfolioBasic-method (*portfolioBasic-class*), 13
- initialize, tradelist-method (*tradelist-class*), 15
- map.market, 7
- mapMarket (*portfolioBasic-class*), 13
- mapMarket, portfolioBasic-method (*portfolioBasic-class*), 13
- matchedPortfolio-class, 8
- matchedPortfolioCollection-class, 9
- matching (*portfolioBasic-class*), 13
- matching, data.frame-method (*matchedPortfolioCollection-class*), 9
- matching, portfolioBasic-method (*portfolioBasic-class*), 13
- performance (*portfolioBasic-class*), 13
- performance, matchedPortfolio-method (*matchedPortfolio-class*), 8
- performance, portfolio-method (*portfolio-class*), 11
- performance, portfolioBasic-method (*portfolioBasic-class*), 13
- performance-class, 10
- plot, contribution, missing-method (*contribution-class*), 4
- plot, exposure, missing-method (*exposure-class*), 5
- plot, matchedPortfolio, missing-method (*matchedPortfolio-class*), 8
- plot, matchedPortfolioCollection, missing-method (*matchedPortfolioCollection-class*), 9
- plot, performance, missing-method (*performance-class*), 10
- plot, portfolioBasic, missing-method (*portfolioBasic-class*), 13
- portfolio (*portfolio-package*), 1
- portfolio-class, 11
- portfolio-package, 1
- portfolioBasic-class, 9
- portfolioBasic-class, 13
- portfolioBasicOrNull-class (*portfolioBasic-class*), 13
- portfolioDiff (*portfolioBasic-class*), 13
- portfolioDiff, portfolio, portfolio-method (*portfolio-class*), 11
- portfolioDiff, portfolioBasic, portfolioBasic-method (*portfolioBasic-class*), 13
- portfolioOrNull-class (*portfolio-class*), 11
- ranksCols, tradelist-method (*tradelist-class*), 15
- restrictedCols, tradelist-method (*tradelist-class*), 15
- scaleWeights (*portfolioBasic-class*), 13
- scaleWeights, portfolioBasic-method (*portfolioBasic-class*), 13
- securityInfo (*portfolio-class*), 11
- securityInfo, portfolio, character-method (*portfolio-class*), 11
- securityInfo, tradelist, character-method (*tradelist-class*), 15
- show, contribution-method (*contribution-class*), 4

show, exposure-method
 (*exposure-class*), 5

show, matchedPortfolio-method
 (*matchedPortfolio-class*), 8

show, performance-method
 (*performance-class*), 10

show, portfolioBasic-method
 (*portfolioBasic-class*), 13

show, tradelist-method
 (*tradelist-class*), 15

summary, contribution-method
 (*contribution-class*), 4

summary, exposure-method
 (*exposure-class*), 5

summary, matchedPortfolio-method
 (*matchedPortfolio-class*), 8

summary, matchedPortfolioCollection-method
 (*matchedPortfolioCollection-class*),
 9

summary, performance-method
 (*performance-class*), 10

summary, portfolio-method
 (*portfolio-class*), 11

summary, portfolioBasic-method
 (*portfolioBasic-class*), 13

tradelist (*tradelist-class*), 15

tradelist-class, 19

tradelist-class, 15

trades-class, 18

trimSide, tradelist-method
 (*tradelist-class*), 15

updatePrices (*portfolio-class*), 11

updatePrices, portfolio, character, numeric-method
 (*portfolio-class*), 11

weight, 19