Package ‘pedquant’

May 5, 2023

Version 0.2.2.1
Title Public Economic Data and Quantitative Analysis
Description Provides an interface to access public economic and financial data for economic research and quantitative analysis. The data sources including NBS, FRED, Sina, Eastmoney and etc. It also provides quantitative functions for trading strategies based on the 'data.table', 'TTR', 'PerformanceAnalytics' and etc packages.
Depends R (>= 4.1.0)
Imports data.table, TTR, zoo, PerformanceAnalytics, curl, httr, rvest, lubridate, stringi, jsonlite, readxl, readr, echarts4r, xefun (> 0.1.3)
Suggests knitr, rmarkdown
License GPL-3
URL https://github.com/ShichenXie/pedquant
BugReports https://github.com/ShichenXie/pedquant/issues
LazyData true
RoxygenNote 7.2.1
Encoding UTF-8
NeedsCompilation no
Author Shichen Xie [aut, cre]
Maintainer Shichen Xie <xie@shichen.name>
Repository CRAN
Date/Publication 2023-05-05 15:00:06 UTC

R topics documented:

dt_banks ........................................................................... 2
dt_ssec ........................................................................... 3
ed_code ........................................................................... 4
ed_fred ........................................................................... 4
dt_banks
dataset of bank stocks in sse

Description
The daily historical data of bank stocks

Usage
dt_banks

Format
A data frame with 7506 rows and 15 variables:
symbol stock ticker symbol
name stock ticker name
date trade date
open stock price at the open of trading
**dt_ssec**

- **high**: stock price at the highest point during trading
- **low**: stock price at the lowest point during trading
- **close**: stock price at the close of trading
- **volume**: number of shares traded
- **amount**: monetary value of shares traded
- **turnover**: rate of shares traded over total
- **close_adj**: adjusted stock price at the close of trading

---

**dt_ssec**  
*dataset of shanghai composite index*

---

**Description**

The daily historical Shanghai Composite Index

**Usage**

```r
dt_ssec
```

**Format**

A data frame with 7506 rows and 15 variables:

- **symbol**: stock ticker symbol
- **name**: stock ticker name
- **date**: trade date
- **open**: stock price at the open of trading
- **high**: stock price at the highest point during trading
- **low**: stock price at the lowest point during trading
- **close**: stock price at the close of trading
- **volume**: number of shares traded
- **amount**: monetary value of shares traded
- **turnover**: rate of shares traded over total
- **close_adj**: adjusted stock price at the close of trading
ed_code

code list by category

Description

ed_code get the code list of country, currency, stock exchange, commodity exchange and administrative district of mainland of China.

Usage

ed_code(cate = NULL)

Arguments

cate The available category values including 'country', 'currency', 'stock_exchange', 'commodity_exchange', 'china_district'.

Examples

## Not run:
# specify the categories
code_list1 = ed_code(cate = c('country', 'currency'))

# interactively return code list
code_list2 = ed_code()

## End(Not run)

ed_fred

query FRED economic data

Description

ed_fred provides an interface to access the economic data provided by FRED (https://fred.stlouisfed.org)

Usage

ed_fred(symbol = NULL, date_range = "10y", from = NULL,
        to = Sys.Date(), na_rm = FALSE, print_step = 1L)
Ed_fred_symbol

Arguments
symbol symbols of FRED economic indicators. It is available via function ed_fred_symbol or its website. Default is NULL, which calls ed_fred_symbol in the back.
date_range date range. Available value includes '1m'-'11m', 'ytd', 'max' and '1y'-'ny'. Default is '10y'.
from the start date. Default is NULL. If it is NULL, then calculate using date_range and end date.
to the end date. Default is the current date.
na.rm logical, whether to remove missing values. Default is FALSE
print_step a non-negative integer, which will print symbol name by each print_step iteration. Default is 1L.

Value
a list of dataframes with columns of symbol, name, date, value, geo, unit. The geo column might be NA according to local internet connection.

Examples

dat = ed_fred(c("A191RL1A225NBEA", "GDPCA"))

Ed_fred_symbol symbol of FRED economic data

Description
Ed_fred_symbol provides an interface to search symbols of economic data from FRED by category or keywords.

Usage
Ed_fred_symbol(category = NULL, keywords = NULL, ...)

Arguments
category the category id. If it is NULL, then search symbols from the top categories step by step.
keywords the query text. If it is NULL, the function will search symbols by category.
... ignored parameters
Examples

```r
## Not run:
# search symbols by category
# from top categories
symbol_dt1 = ed_fred_symbol()
# specify the initial categories
symbol_dt2 = ed_fred_symbol(category = 1)

# search symbol by keywords
symbol_dt3 = ed_fred_symbol(keywords = "gdp china")

## End(Not run)
```

Description


Usage

```r
ed_nbs(symbol = NULL, freq = NULL, geo_type = NULL, subregion = NULL,
        date_range = "10y", from = NULL, to = Sys.Date(), na_rm = FALSE,
        eng = FALSE)
```

Arguments

- **symbol**: symbols of NBS indicators. It is available via `ed_nbs_symbol`. Default is NULL.
- **freq**: the frequency of NBS indicators, including 'monthly', 'quarterly', 'yearly'. Default is NULL.
- **geo_type**: geography type in NBS, including 'nation', 'province', 'city'. Default is NULL.
- **subregion**: codes of province or city, which is available via `ed_nbs_subregion`. Default is NULL.
- **date_range**: date range. Available value includes '1m'-'11m', 'ytd', 'max' and '1y'-'ny'. Default is '10y'.
- **from**: the start date. Default is NULL. If it is NULL, then calculate using `date_range` and end date.
- **to**: the end date. Default is the current date.
- **na_rm**: logical. Whether to remove missing values from datasets. Default is FALSE.
- **eng**: logical. The language of the query results is in English or in Chinese. Default is FALSE.
ed_nbs_subregion

Examples

## Not run:
# query NBS data without setting any parameters
dt = ed_nbs()

# specify parameters
dt1 = ed_nbs(geo_type='nation', freq='quarterly', symbol='A010101')
# or using 'n'/ 'q' represents 'nation'/ 'quarterly'
dt2 = ed_nbs(geo_type='n', freq='q', symbol='A010101')

# query data in one province
dt3 = ed_nbs(geo_type='province', freq='quarterly',
             symbol='A010101', subregion='110000')

# query data in all province
dt4 = ed_nbs(geo_type='province', freq='quarterly',
             symbol='A010101', subregion='all')

## End(Not run)

ed_nbs_subregion       subregion code of NBS economic data

Description

ed_nbs_subregion query province or city code from NBS

Usage

ed_nbs_subregion(geo_type = NULL, eng = FALSE)

Arguments

geo_type            geography type in NBS, including 'province', 'city'. Default is NULL.
eng                 logical. The language of the query results is in English or in Chinese. Default is FALSE.

Examples

## Not run:
# province code
prov1 = ed_nbs_subregion(geo_type = 'province')
# or using 'p' represents 'province'
prov2 = ed_nbs_subregion(geo_type = 'p')

# city code in Chinese
# city = ed_nbs_subregion(geo_type = 'c', eng = FALSE)
# city code in English

city = ed_nbs_subregion(geo_type = 'c', eng = TRUE)

## End(Not run)

## ed_nbs_symbol

### symbol of NBS economic data

#### Description

ed_nbs_symbol provides an interface to query symbols of economic indicators from NBS.

#### Usage

ed_nbs_symbol(symbol = NULL, geo_type = NULL, freq = NULL, eng = FALSE)

#### Arguments

- **symbol**: symbols of NBS indicators.
- **geo_type**: geography type in NBS, including 'nation', 'province', 'city'. Default is NULL.
- **freq**: the frequency of NBS indicators, including 'monthly', 'quarterly', 'yearly'. Default is NULL.
- **eng**: logical. The language of the query results is in English or in Chinese. Default is FALSE.

#### Examples

# query symbol interactively
## Not run:
sym = ed_nbs_symbol()
## End(Not run)

### md_bond

#### query bond data

#### Description

md_bond query bond market data from FRED and ChinaBond.

#### Usage

md_bond(symbol = NULL, type = "history", date_range = "3y",
from = NULL, to = Sys.Date(), print_step = 1L, ...)

# query symbol interactively
## Not run:
sym = ed_nbs_symbol()
## End(Not run)
**md_forex**

**Arguments**

- **symbol**: bond symbols. Default is NULL.
- **type**: the data type. Default is history.
- **date_range**: date range. Available value includes '1m'-'11m', 'ytd', 'max' and '1y'-'ny'. Default is 3y.
- **from**: the start date. Default is NULL. If it is NULL, then calculate using date_range and end date.
- **to**: the end date. Default is the current date.
- **print_step**: a non-negative integer, which will print symbol name by each print_step iteration. Default is 1L.
- **...**: Additional parameters.

---

**Description**

*md_forex* query forex market data from FRED (history data) or sina (real data).

**Usage**

```R
md_forex(symbol, type = "history", date_range = "3y", from = NULL, to = Sys.Date(), print_step = 1L, ...)```

**Arguments**

- **symbol**: forex symbols. Default is NULL.
- **type**: the data type, available values including history and real. Default is history.
- **date_range**: date range. Available value includes '1m'-'11m', 'ytd', 'max' and '1y'-'ny'. Default is 3y.
- **from**: the start date. Default is NULL. If it is NULL, then calculate using date_range and end date.
- **to**: the end date. Default is the current date.
- **print_step**: a non-negative integer, which will print symbol name by each print_step iteration. Default is 1L.
- **...**: Additional parameters.
### Examples

```r
## Not run:
# history data
dtfx_hist1 = md_forex(c('usdcny', 'usdjpy'))

# real data
dtfx_real = md_forex(c('eurusd', 'usdcny', 'usdjpy'), type = 'real')

## End(Not run)
```

---

### md_future

**query future market data**

### Description


### Usage

```r
md_future(symbol, type = "history", date_range = "max", from = NULL, to = Sys.Date(), freq = "daily", print_step = 1L, ...)```

### Arguments

- `symbol`: future symbols It is available via function `md_future_symbol` or its website.
- `type`: the data type, including history, real and info. Default is history.
- `date_range`: date range. Available value includes "1m"-'11m', 'ytd', 'max' and '1y'-'ny'. Default is max.
- `from`: the start date. Default is NULL. If it is NULL, then calculate using date_range and end date.
- `to`: the end date. Default is the current date.
- `freq`: data frequency, default is daily.
- `print_step`: a non-negative integer, which will print symbol name by each print_step iteration. Default is 1L.
- `...`: Additional parameters.

### Examples

```r
## Not run:
# history data
df_hist = md_future(symbol = c('IF0', 'A0', 'CU0', 'CF0', 'XAU'))

# real data
```
\textbf{md\_future\_symbol}

\begin{verbatim}
df\_real = md\_future(symbol = c('IF0', 'A0', 'CU0', 'CF0', 'XAU'),
                  type = 'real')

## End(Not run)
\end{verbatim}

\textit{md\_future\_symbol} \hspace{1cm} symbol of future market data

\textbf{Description}


\textbf{Usage}

\begin{verbatim}
md\_future\_symbol(...)\end{verbatim}

\textbf{Arguments}

\begin{verbatim}
... ignored parameters\end{verbatim}

\textbf{Examples}

\begin{verbatim}
## Not run:
sybs = md\_future\_symbol()

## End(Not run)
\end{verbatim}

\textbf{md\_money}

\begin{verbatim}
query interbank offered rate
\end{verbatim}

\textbf{Description}

\texttt{md\_money} query libor from FRED or shibor from chinamoney.

\textbf{Usage}

\begin{verbatim}
md\_money(symbol = NULL, date\_range = "3y", from = NULL,
         to = Sys\_Date(), print\_step = 1L)
\end{verbatim}
Arguments

symbol ibor symbols. Default is NULL.
date_range date range. Available value includes '1m'-'11m', 'ytd', 'max' and '1y'-''ny'. Default is 3y.
from the start date. Default is NULL. If it is NULL, then calculate using date_range and end date.
to the end date. Default is the current date.
print_step a non-negative integer, which will print symbol name by each print_step iteration. Default is 1L.

Description

md_moneycn query chinese benchmark rates

Usage

md_moneycn(symbol = NULL, date_range = "3y", from = NULL, to = Sys.Date(), print_step = 1L)

Arguments

symbol benchmarks, available values including 'rmbx', 'shibor', 'lpr', 'pr', 'yb'. Default is NULL,
date_range date range. Available value includes '1m'-'11m', 'ytd', 'max' and '1y'-''ny'. Default is 3y.
from the start date. Default is NULL. If it is NULL, then calculate using date_range and end date.
to the end date. Default is the current date.
print_step a non-negative integer, which will print symbol name by each print_step iteration. Default is 1L.
Description

`md_stock` provides an interface to query stock or fund data.

Usage

```r
md_stock(symbol, type = "history", date_range = "3y", from = NULL,
          to = Sys.Date(), forward = NULL, print_step = 1L, ...)
```

Arguments

- `symbol` symbols of stock shares.
- `type` the data type, including history, real. Defaults to history.
- `date_range` date range. Available value including '1m'-'11m', 'ytd', 'max' and '1y'-.. Default is '3y'.
- `from` the start date. Default is NULL.
- `to` the end date. Default is current system date.
- `forward` whether to forward adjust the OHLC prices. If it is NULL, return the original data from source, defaults to NULL.
- `print_step` A non-negative integer. Print symbol name by each print_step iteration. Default is 1L.
- `...` Additional parameters.

Examples

```r
## Not run: query history data
# us
FAANG = md_stock(c("META", "AMZN", "AAPL", "NFLX", "GOOG"))

# hkex
TMX = md_stock(c("00700.hk", "03690.hk", "01810.hk"))

# sse/szse
## the symbol without suffix
dt_cn1 = md_stock(c("000001", "000001", "512510"))
## the symbol with suffix
dt_cn2 = md_stock(c("000001.sz", "000001.ss", "512510.ss"))

# Example III: query real prices
# real price for equities
dt_real1 = md_stock(c("META", "AMZN", "AAPL", "NFLX", "GOOG",
```
md_stock_financials

'00700.hk', '03690.hk', '01810.hk',
"000001", "000001", "512510"), type = 'real')

# query company information
dt_info1 = md_stock('600036', type = 'info')

## End(Not run)

md_stock_adjust adjust stock prices

Description

md_stock_adjust adjusts the open, high, low and close stock prices.

Usage

md_stock_adjust(dt, forward = FALSE, ...)

Arguments

dt a list/dataframe of time series datasets that did not adjust for split or dividend.
forward forward adjust or backward adjust, defaults to FALSE.
... Additional parameters.

Examples

data("dt_banks")

dtadj1 = md_stock_adjust(dt_banks, adjust = FALSE)
dtadj2 = md_stock_adjust(dt_banks, adjust = TRUE)

md_stock_financials query financial statements

Description

md_stock_financials provides an interface to query financial statements for all listed companies in SSE and SZSE by specified report date.
**md_stock_financials**

Usage

```r
md_stock_financials(type = NULL, date_range = "1q", from = NULL, to = Sys.Date(), print_step = 1L, ...)
```

Arguments

- **type**
  - the type of financial statements.
- **date_range**
  - date range. Available value including '1m'-'11m', 'ytd', 'max' and '1y'. Default is '3y'.
- **from**
  - the start date. Default is NULL.
- **to**
  - the end date. Default is current system date.
- **print_step**
  - A non-negative integer. Print financial statements name by each print_step iteration. Default is 1L.
- **...**
  - Additional parameters.

Examples

```r
## Not run:
# interactively specify type of financial table
dtfs1 = md_stock_financials(type="fs0_summary", to = '2022-12-31')
dtfs2 = md_stock_financials(type="fs0_summary", to = c('2022-12-31', '2023-03-31'))
dtfs3 = md_stock_financials(type="fs0_summary", from = '2022-12-31', to = Sys.Date())

# all statements
dtfs4 = md_stock_financials(type = "fs", to = '2022-12-31')

# setting column names to Chinese
dtfs5 = md_stock_financials(type="fs0_summary", to = '2022-12-31', colnam_chn = TRUE)

## End(Not run)
```

---

**md_stock_symbol**

**symbol components of exchange**

Description

**md_stock_symbol** returns all stock symbols by exchange

Usage

```r
md_stock_symbol(exchange = NULL, ...)
```

Arguments

- **exchange**
  - the available stock exchanges are sse, szse, hkex, amex, nasdaq, nyse.
- **...**
  - ignored parameters
md_symbol

## Examples

```r
## Not run:
# get stock symbols in a stock exchange
# specify the exchanges
ex_sym1 = md_stock_symbol(exchange = c('sse', 'szse'))

# choose exchanges interactively
ex_sym2 = md_stock_symbol()

## End(Not run)
```

---

**md_symbol**

*symbol of market data*

---

### Description

*md_stock_symbol* returns all symbols by market category, including forex, money, bond, stock, future.

### Usage

```r
md_stock_symbol(market = NULL, ...)
```

### Arguments

- **market**: the market category, including forex, money, bond, stock, future. Default is NULL.
- **...**: ignored parameters

### Examples

```r
## Not run:
syblst = md_stock_symbol()

## End(Not run)
```
Description

pq_addti creates technical indicators using the functions provided in TTR package.

Usage

pq_addti(dt, ...)

Arguments

dt a list/dataframe of time series datasets.

... list of technical indicator parameters: sma = list(n=50), macd = list().

1. There are four types of parameters.
   • set by default and do not required, such as 'OHLC', 'HLC', 'HL' and 'volume'.
   • set by default and can be modified, such as 'price', 'prices', 'x'. Its default value is 'close' or 'value' column.
   • always required, such as 'y', 'w'.
   • numeric parameters, such as 'n', 'sd', 'v', 'nFast', 'nSlow', 'nSig', 'accel'. These parameters should be provided, otherwise using default values in corresponding function.

2. TTR functions are summarized in below. See TTR package’s help document for more detailed parameters.
   • moving averages: SMA, EMA, DEMA, WMA, EVWMA, ZLEMA, VWAP, VMA, HMA, ALMA, GMMA
   • rolling functions: runMin, runMax, runMean, runMedian; runCov, runCor; runVar, runSD, runMAD; runSum, wilderSum
   • bands / channels: BBands, PBands, DonchianChannel
   • SAR, ZigZag
   • trend direction/strength: aroon, CCI, ADX, TDI, VHF, EMV
   • volatility measures: ATR, chaikinVolatility, volatility, SNR
   • money flowing into/out: OBV, chaikinAD, CLV, CMF, MFI, williamsAD
   • rate of change / momentum: ROC, momentum, KST, TRIX
   • oscillator: MACD, DPO, DVI, ultimateOscillator; RSI, CMO; stoch, SMI, WPR

Examples

# load data
data('dt_ssec')
# add technical indicators
dt_ti1 = pq_addti(dt_ssec, sma=list(n=20), sma=list(n=50), macd = list())

dt_ti11 = pq_addti(dt_ssec, sma=list(n=20, x='open'), sma=list(n=50, x='open'))
dt_ti12 = pq_addti(dt_ssec, x='open', sma=list(n=20), sma=list(n=50))

# specify the price column x

dt_ti2 = pq_addti(dt_ssec, sma=list(n=20, x='open'), sma=list(n=50, x='open'),
                 col_kp = c('symbol', 'name'))

dt_ti3 = pq_addti(dt_ssec, sma=list(n=20), sma=list(n=50), col_kp = NULL)

# only technical indicators

dt_ti3 = pq_addti(dt_ssec, sma=list(n=20), sma=list(n=50), macd = list(),
                 col_kp = c('symbol', 'name'))

dt_ti3 = pq_addti(dt_ssec, bias = list(n = 200))

---

pq_addti_funs   

**technical functions**

**Description**

Technical functions provided in TTR package.

**Usage**

pq_addti_funs()

---

pq_freq   

**converting frequency of daily data**

**Description**

pq_freq convert a daily OHLC dataframe into a specified frequency.
pq_opr

Usage

pq_freq(dt, freq = "monthly", date_type = "eop")

Arguments

dt          a list/dataframe of time series dataset.
freq        the frequency that the input daily data will converted to. It supports weekly, monthly, quarterly and yearly.
date_type   the available date type are eop (end of period) and bop (beginning of period), defaults to the eop.

Examples

data(dt_ssec)
dat1_weekly = pq_freq(dt_ssec, "weekly")

## Not run:
data(dt_banks)
dat2_weekly = pq_freq(dt_banks, "monthly")

## End(Not run)

pq_opr          dataframe operation

Description

It performs arithmetic operation on numeric columns on multiple series.

Usage

pq_opr(dt, opr, x = "close", rm_na = FALSE, ...)

Arguments

dt           a list/dataframe of time series datasets.
opr          operation string.
x            the numeric column names, defaults to close.
rm_na        weather to remove NA values when perform arithmetic.
...          additional parameters.
pq_performance

Examples

data("dt_banks")

dt1 = pq_opr(dt_banks, '601288.SS/601988.SS')
print(dt1)

dt2 = pq_opr(dt_banks, c('(601288.SS+601988.SS)/2', '(601288.SS*601988.SS)^0.5'))
print(dt2)

pq_performance calculating performance metrics

Description

pq_performance calculates performance metrics based on returns of market price or portfolio. The performance analysis functions are calling from PerformanceAnalytics package, which includes many widely used performance metrics.

Usage

pq_performance(dt, Ra, Rb = NULL, perf_fun, ...)

Arguments

dt a list/dataframe of time series datasets.
Ra the column name of asset returns.
Rb the column name of baseline returns, defaults to NULL.
perf_fun performance function from PerformanceAnalytics package, see pq_perf_funs.
... additional parameters, the arguments used in PerformanceAnalytics functions.

Examples

library(pedquant)
library(data.table)

# load data
data(dt_banks)
data(dt_ssec)

# calculate returns
datret1 = pq_return(dt_banks, 'close', freq = 'monthly', rcol_name = 'Ra')
datret2 = pq_return(dt_ssec, 'close', freq = 'monthly', rcol_name = 'Rb')

# merge returns of assets and baseline
datRaRb = merge(
    rbindlist(datret1)[, .(date, symbol, Ra)],
    rbindlist(datret2)[, .(date, Rb)],
    by = 'date', all.x = TRUE
)

# calculate table.CAPM metrics
perf_capm = pq_performance(datRaRb, Ra = 'Ra', Rb = 'Rb', perf_fun = 'table.CAPM')
rbindlist(perf_capm, idcol = 'symbol')

---

**pq_performance_funs**  
*performance functions*

**Description**
A complete list of performance functions from PerformanceAnalytics package.

**Usage**
```
pq_performance_funs()
```

**pq_plot**  
*creating charts for time series*

**Description**
**pq_plot** provides an easy way to create interactive charts for time series dataset based on predefined formats.

**Usage**
```
pq_plot(dt, chart_type = "line", x = "date", y = "close", yb = NULL,
        date_range = "max", yaxis_log = FALSE, title = NULL, addti = NULL,
        nsd_lm = NULL, markline = TRUE, orders = NULL, arrange = list(rows =
        NULL, cols = NULL), theme = "default", ...)
```

**Arguments**
- **dt**  
a list/dataframe of time series dataset
- **chart_type**  
chart type, including line, step, candle.
- **x**  
column name for x axis
- **y**  
column name for y axis
- **yb**  
column name for baseline
**pq_plot**

- **date_range**: date range of x axis to display. Available value includes '1m'-'11m', 'ytd', 'max' and '1y'-'ny'. Default is max.
- **yaxis_log**: whether to display y axis values in log. Default is FALSE.
- **title**: chart title. It will added to the front of chart title if it is specified.
- **addti**: list of technical indicators or numerical columns in dt. For technical indicator, it is calculated via pq_addti, which including overlays and indicators.
- **nsd_lm**: number of standard deviation from linear regression fitting values.
- **markline**: whether to display markline. Default is TRUE.
- **orders**: the data frame of transaction orders, which includes symbol, date (required), prices, volumes (required) and type columns.
- **arrange**: a list. Number of rows and columns charts to connect. Default is NULL.
- **theme**: name of echarts theme, see details in e_theme

**Examples**

```r
# single serie
library(data.table)
library(pedquant)
data(dt_ssec)

# line chart (default)
e1 = pq_plot(dt_ssec, chart_type = 'line') # line chart (default)
e1[[1]]

# add technical indicators
e2 = pq_plot(dt_ssec, addti = list(
sma = list(n = 200),
sma = list(n = 50),
volume = list(),
macd = list()
))
e2[[1]]

# linear trend with yaxis in log
e3 = pq_plot(dt_ssec, nsd_lm = c(-0.8, 0, 0.8), markline=FALSE)
e3[[1]]

# multiple series
data(dt_banks)
dt_banksadj = md_stock_adjust(dt_banks)

# linear trend
e4 = pq_plot(dt_banksadj)
e4 = pq_plot(dt_banksadj, arrange = list(rows=1, cols=1))
e4[[1]]

# orders
```
b2 = dt_banks[symbol %in% c('601988.SS', '601398.SS')]
b2orders = b2[sample(.N, 10),.(symbol, date, prices=close,
        type=sample(c('buy','sell'), 10, replace=TRUE))]
e5 = pq_plot(b2, orders=b2orders)
e5[[1]]
e6 = pq_plot(b2, orders=b2orders, arrange = list(rows=1, cols=1))
e6[[1]]

pq_portfolio calculating returns/equity of portfolio

Description

pq_portfolio calculates the weighted returns or the equity of a portfolio assets.

Usage

pq_portfolio(dt, orders, x = "close", dtb = NULL, init_fund = NULL,
        method = "arithmetic", cols_keep = NULL, ...)

Arguments

dt a list/dataframe of price by asset.
orders a data frame of transaction orders, which includes symbol, date, prices, volumes and type columns.
x the column name of adjusted asset price, defaults to close.
dtb a list/dataframe of price base asset.
init_fund initial fund value.
method the method to calculate asset returns, the available values include arithmetic and log, defaults to arithmetic.
cols_keep the columns keep in the return data. The columns of symbol, name and date will always kept if they are exist in the input data.
... ignored

Examples

library(pedquant)

data(dt_banks)
datadj = md_stock_adjust(dt_banks)

# example I
orders = data.frame(
  symbol = c("601288.SS", "601328.SS", "601398.SS", "601939.SS", "601988.SS"),
  volumes = c(100, 200, 300, 300, 100)
)

dtRa = pq_portfolio(datadj, orders=orders)

e1 = pq_plot(dtRa, y = 'cumreturns')
e1[[1]]

# example II

data(dt_ssec)

orders = data.frame(
  symbol = rep(c("601288.SS", "601328.SS", "601398.SS", "601939.SS", "601988.SS"), 3),
  date = rep(c('2009-03-02', '2010-01-04', '2014-09-01'), each = 5),
  volumes = rep(c(100, 200, 300, 300, 100), 3) * rep(c(1, -1, 2), each = 5)
)

dtRab = pq_portfolio(datadj, orders=orders, dtb = dt_ssec, init_fund = 10000)

e2 = pq_plot(dtRab, y = 'cumreturns', yb = 'cumreturns_000001.SS', addti = list(portfolio=list()))
e2[[1]]

# example III

orders = data.frame(symbol = "000001.SS",
  date = c("2009-04-13", "2010-03-24", "2014-08-13", "2015-09-10"),
  volumes = c(400, -400, 300, -300))
dtRa2 = pq_portfolio(dt_ssec, orders=orders, cols_keep = 'all')

e3 = pq_plot(dtRa2, y = 'close', addti = list(cumreturns=list(), portfolio=list()))
e3[[1]]

---

pq_return  

Calculating returns by frequency

Description

pq_return calculates returns for daily series based on specified column, frequency and method type.

Usage

pq_return(dt, x, freq = "daily", n = 1, date_type = "eop",
  method = "arithmetic", cumreturns = FALSE, rcol_name = NULL,
  cols_keep = NULL, date_range = "max", from = NULL, to = Sys.Date(), ...
)
pq_return

Arguments

dt  | a list/dataframe of adjusted series.
x   | the column name of adjusted asset price.
freq | the frequency of returns. It supports 'daily', 'weekly', 'monthly', 'quarterly', 'yearly' and 'all'. Defaults to daily.
n   | the number of preceding periods used as the base value, defaults to 1, which means based on the previous period value.
date_type | the available date type are eop (end of period) and bop (beginning of period), defaults to the eop.
method | the method to calculate asset returns, the available methods including arithmetic and log, defaults to arithmetic.
cumreturns | logical, whether to return cumulative returns. Defaults to FALSE.
rcol_name | setting the column name of returns, defaults to NULL.
cols_keep | the columns keep in the return data. The columns of symbol, name and date will always kept if they are exist in the input data.
date_range | date range. Available value includes '1m'-11m', 'ytd', 'max' and '1y'-ny'. Default is max.
from | the start date. Default is NULL. If it is NULL, then calculate using date_range and end date.
to | the end date. Default is the current date.
... | ignored

Examples

# load data and adjust
data(dt_banks)
datadj = md_stock_adjust(dt_banks)

# set freq
dts_returns1 = pq_return(datadj, x = 'close_adj', freq = 'all')

# set method
dts_returns2 = pq_return(datadj, x = 'close_adj', method = 'log')

# set cols_keep
dts_returns3 = pq_return(datadj, x = 'close_adj', cols_keep = 'cap_total')

# cumulative returns
dts_cumreturns = pq_return(datadj, x = 'close_adj', from = '2012-01-01', cumreturns = TRUE)
e1 = pq_plot(dts_cumreturns, y = 'cumreturns.daily', title='cumreturns',
arrange = list(rows=1, cols=1))
e1[[1]]
%>% crossover operators

Description

Binary operators which create the upwards or downwards crossover signals.

Usage

x %>% y
x %<% y

Arguments

x, y numeric vectors

Examples

library(data.table)
library(pedquant)
data("dt_banks")
boc = md_stock_adjust(setDT(dt_banks)[symbol=='601988.SS'])
bocti = pq_addti(boc, x='close_adj', sma=list(n=200), sma=list(n=50))
dtorders = copy(bocti[[1]])[,(symbol, name, date, close_adj, sma_50, sma_200)
][sma_50 %>% sma_200, ':='
  type = 'buy', prices = close_adj
][sma_50 %<% sma_200, ':='
  type = 'sell', prices = close_adj
]
, (c('type', 'prices')) := lapply(.SD, shift), .SDcols = c('type', 'prices')]
orders = dtorders[is.na(type)]
head(orders)
e = pq_plot(boc, y='close_adj', addti = list(sma=list(n=200), sma=list(n=50)), orders = orders)
e[[1]]
Index

* datasets
  dt_banks, 2
  dt_ssec, 3
  %x<%(%x>%), 26
  %x>%, 26
  dt_banks, 2
  dt_ssec, 3
  e_theme, 22
  ed_code, 4
  ed_fred, 4
  ed_fred_symbol, 5
  ed_nbs, 6
  ed_nbs_subregion, 7
  ed_nbs_symbol, 8
  md_bond, 8
  md_forex, 9
  md_future, 10
  md_future_symbol, 11
  md_money, 11
  md_moneycn, 12
  md_stock, 13
  md_stock_adjust, 14
  md_stock_financials, 14
  md_stock_symbol, 15
  md_symbol, 16
  pq_addti, 17
  pq_addti_funs, 18
  pq_freq, 18
  pq_opr, 19
  pq_performance, 20
  pq_performance_funs, 21
  pq_plot, 21
  pq_portfolio, 23
  pq_return, 24