Package 'HighFreq'

August 2, 2016

Type Package

Title High Frequency Data Management

Version 0.1

Date 2015-05-28

Author Jerzy Pawlowski (algoquant)

Maintainer Jerzy Pawlowski <jp3900@nyu.edu>

Description Functions for chaining and joining time series, scrubbing bad data, managing time zones and alligning time indices, converting TAQ data to OHLC format, aggregating data to lower frequency, estimating volatility, skew, and higher moments.

License GPL (>= 2)

Depends rutils,
  quantmod,

Imports TTR,
  caTools,
  lubridate,

Remotes github::algoquant/rutils,

LazyData true

Repository GitHub

URL https://github.com/algoquant/HighFreq

RoxygenNote 5.0.1

R topics documented:

Index   **22**

---

agg_regate	*Calculate the aggregation (weighted sum) of a statistical estimator over a* OHLC *time series.*

Description

Calculate the aggregation (weighted sum) of a statistical estimator over a OHLC time series.

Usage

```
agg_regate(oh_lc, mo_ment = "run_variance", weight_ed = TRUE, ...)
```

Arguments

oh_lc	OHLC time series of prices and trading volumes, in xts format.
mo_ment	character string representing function for estimating the moment.
weight_ed	Boolean should estimate be weighted by the trading volume? (default is TRUE)
...	additional parameters to the mo_ment function.

Details

Calculates a single number representing the volume weighted sum of an estimator over the OHLC time series of prices. By default the sum is trade volume weighted.

Value

Single numeric value equal to the volume weighted sum of an estimator over the time series.

Examples

```
# create time index of one minute intervals over several days
in_dex <- seq(from=as.POSIXct("2015-02-09 09:30:00"),
              to=as.POSIXct("2015-02-28 16:00:00"), by="1 min")
# create synthetic xts of random prices
x_ts <- xts(exp(cumsum(0.001*rnorm(length(in_dex)))), order.by=in_dex)
# add trade Volume data
x_ts <- merge(x_ts,
         volume=sample(x=10*(2:18),
           size=length(in_dex), replace=TRUE))
# aggregate to hours OHLC data
```

```
oh_lc <- to.period(x=x_ts, period="hours")
# calculate time series of daily skew estimates
skew_daily <- apply.daily(x=oh_lc, FUN=agg_regate, mo_ment="run_skew")
```

---

extreme_values	*Identify extreme values in a single-column* xts *time series.*

Description

Identifies extreme values as those that exceed a multiple of the rolling volatility.

Usage

```
extreme_values(x_ts, win_dow = 51, vol_mult = 2)
```

Arguments

x_ts	single-column xts time series.
win_dow	number of data points for estimating rolling volatility.
vol_mult	volatility multiplier.

Details

Calculates the rolling volatility as a quantile of values over a rolling window. Extreme values are those that exceed the product of the volatility multiplier times the rolling volatility. Extreme values are the very tips of the tails when the distribution of values becomes very fat-tailed. The volatility multiplier vol_mult controls the threshold at which values are identified as extreme. Smaller volatility multiplier values will cause more values to be identified as extreme.

Value

Boolean vector with the same number of rows as input time series.

Examples

```
# create xts time series
x_ts <- xts(x=rnorm(1000), order.by=(Sys.time()-3600*(1:1000)))
# scrub extreme values
x_ts <- x_ts[!extreme_values(x_ts, vol_mult=1)]
```

hf_data   *High frequency data sets*

Description

hf_data.RData is a file containing the datasets:

SPY an xts time series containing 1-minute OHLC bar data for the SPY etf, from 2008-01-02 to 2014-05-19. SPY contains 625,425 rows of data, each row contains a single minute bar.

TLT an xts time series containing 1-minute OHLC bar data for the TLT etf, up to 2014-05-19.

VXX an xts time series containing 1-minute OHLC bar data for the VXX etf, up to 2014-05-19.

Usage

```
data(hf_data)  # not required - data is lazy load
```

Format

Each xts time series contains OHLC data, with each row containing a single minute bar:

Open Open price in the bar

High High price in the bar

Low Low price in the bar

Close Close price in the bar

Volume trading volume in the bar

Source

https://wrds-web.wharton.upenn.edu/wrds/

References

Wharton Research Data Service (WRDS)

Examples

```
# data(hf_data)  # not required - data is lazy load
head(SPY)
chart_Series(x=SPY["2009"])
```

---

price_jumps	*Identify isolated price jumps in a single-column* xts *time series of prices, based on pairs of large neighboring returns of opposite sign.*

Description

Identify isolated price jumps in a single-column xts time series of prices, based on pairs of large neighboring returns of opposite sign.

Usage

```
price_jumps(x_ts, win_dow = 51, vol_mult = 2)
```

Arguments

x_ts	single-column xts time series of prices.
win_dow	number of data points for estimating rolling volatility.
vol_mult	volatility multiplier.

Details

Isolated price jumps are single prices that are very different from neighboring values. Price jumps create pairs of large neighboring returns of opposite sign. The function price_jumps first calculates simple returns from prices. Then it calculates the rolling volatility of returns as a quantile of returns over a rolling window. Jump prices are identified as those where neighboring returns both exceed a multiple of the rolling volatility, but the sum of those returns doesn't exceed it.

Value

Boolean vector with the same number of rows as input time series.

Examples

```
# create xts time series
x_ts <- xts(x=rnorm(1000), order.by=(Sys.time()-3600*(1:1000)))
# scrub jump prices
x_ts <- x_ts[!price_jumps(x_ts, vol_mult=1.0)]
```

---

roll_apply	*Apply an aggregation function over a rolling lookback window and the end points of an* OHLC *time series.*

Description

Apply an aggregation function over a rolling lookback window and the end points of an OHLC time series.

Usage

```
roll_apply(oh_lc, agg_fun = "run_variance", win_dow = 11,
  end_points = (0:NROW(oh_lc)), by_columns = FALSE, ...)
```

Arguments

`oh_lc`	OHLC time series of prices and trading volumes, in `xts` format.
`agg_fun`	`character` string representing an aggregation function to be applied over a rolling lookback window.
`win_dow`	the size of the lookback window, equal to the number of bars of data used for applying the aggregation function.
`end_points`	an integer vector of end points.
`by_columns`	Boolean should the function `agg_fun` be applied column-wise, or should it be applied to all the columns combined? (default is `FALSE`)
`...`	additional parameters to the `agg_fun` function.

Details

Applies an aggregation function over a rolling lookback window and the end points of an OHLC time series.

Performs similar operations to the functions `rollapply()` and `period.apply()` from package xts, and also the function `apply.rolling()` from package PerformanceAnalytics. (The function `rollapply()` isn't exported from the package xts.)

But the function `roll_apply()` is faster because it performs less type-checking and other overhead. Unlike the other functions, `roll_apply()` doesn't produce any leading `NA` values.

The function `roll_apply()` can be called in two different ways, depending on the argument `end_points`. If the argument `end_points` isn't explicitly passed to `roll_apply()`, then the default value is used, and `roll_apply()` performs aggregations over overlapping windows at each point in time. If the argument `end_points` is explicitly passed to `roll_apply()`, then `roll_apply()` performs aggregations over windows spanned by the end_points.

The aggregation function `agg_fun` can return either a single value or a vector of values. If the aggregation function `agg_fun` returns a single value, then `roll_apply()` returns an `xts` time series with a single column. If the aggregation function `agg_fun` returns a vector of values, then `roll_apply()` returns an `xts` time series with multiple columns equal to the length of the vector returned by the aggregation function `agg_fun`.

Value

An `xts` time series with the same number of rows as the argument `oh_lc`.

Examples

```
# extract a single day of SPY data
price_s <- SPY["2012-02-13"]
win_dow <- 11
# Calculate the rolling sums of the columns of price_s
agg_regations <- roll_apply(price_s, agg_fun=sum, win_dow=win_dow, by_columns=TRUE)
# Apply a vector-valued aggregation function over a rolling window
agg_function <- function(x_ts)  c(max(x_ts[, 1]), min(x_ts[, 4]))
agg_regations <- roll_apply(price_s, agg_fun=agg_function, win_dow=win_dow)
# define end points at 11-minute intervals (SPY is minutely bars)
end_points <- rutils::end_points(price_s, inter_val=win_dow)
# Calculate the rolling sums of the columns of price_s over end_points
agg_regations <- roll_apply(price_s, agg_fun=sum, win_dow=2, end_points=end_points, by_columns=TRUE)
# Apply a vector-valued aggregation function over the end_points of price_s
agg_regations <- roll_apply(price_s, agg_fun=agg_function, win_dow=2, end_points=end_points)
```

---

roll_hurst	*Calculate the rolling Hurst exponent over a rolling lookback window or the end points of an* OHLC *time series.*

Description

Calculate the rolling Hurst exponent over a rolling lookback window or the end points of an OHLC time series.

Usage

```
roll_hurst(oh_lc, win_dow = 11, off_set = 0, roll_end_points = FALSE)
```

Arguments

oh_lc	an OHLC time series of prices in xts format.
win_dow	the size of the lookback window, equal to the number of bars of data used for aggregating the OHLC prices.
off_set	the number of bars of data in the first, stub window.
roll_end_points	
	Boolean should the Hurst exponent be calculated using aggregations over the end points, or by rolling over a lookback window? (default is FALSE)

Details

Calculates the rolling Hurst exponent in two different ways, depending on the argument roll_end_points.

If roll_end_points is TRUE, then the rolling Hurst exponent is calculated as the logarithm of the ratios of two rolling variance estimates. The Hurst exponent is defined as the logarithm of the ratio of the variance of aggregated returns, divided by the variance of simple returns. The aggregated returns are calculated over non-overlapping windows spanned by the end points, using the function to_period(). The Hurst exponent values are calculated only at the end points. The non-overlapping aggregation windows can be shifted by using the argument off_set, which produces a slightly different series of rolling hurst exponent values.

If roll_end_points is FALSE, then the rolling Hurst exponent is calculated as the logarithm of the ratios of two rolling price range estimates. The Hurst exponent is defined as the logarithm of the ratio of the range of aggregated prices, divided by the average range of prices in each bar. The aggregated prices are calculated over overlapping windows, and the Hurst exponent values are calculated at each point in time.

Value

An xts time series with a single column and the same number of rows as the argument oh_lc.

Examples

```
# calculate rolling Hurst over SPY
hurst_rolling <- roll_hurst(oh_lc=SPY, win_dow=10)
# calculate Hurst over end points of SPY
hurst_rolling <- roll_hurst(oh_lc=SPY, win_dow=10, off_set=0, roll_end_points=TRUE)
# calculate a series of rolling hurst values using argument off_set
hurst_rolling <- lapply(0:9, roll_hurst, oh_lc=SPY, win_dow=10, roll_end_points=TRUE)
```

```
hurst_rolling <- rutils::do_call_rbind(hurst_rolling)
# remove daily warmup periods
hurst_rolling <- hurst_rolling["T09:41:00/T16:00:00"]
chart_Series(x=hurst_rolling["2012-02-13"],
  name=paste(colnames(hurst_rolling), "10-minute aggregations"))
```

---

roll_moment	*Calculate a vector of statistics over an* OHLC *time series, and calculate a rolling mean over the statistics.*

Description

Calculate a vector of statistics over an OHLC time series, and calculate a rolling mean over the statistics.

Usage

```
roll_moment(oh_lc, mo_ment = "run_variance", win_dow = 11,
  weight_ed = TRUE, ...)
```

Arguments

oh_lc	OHLC time series of prices and trading volumes, in xts format.
mo_ment	character string representing a function for estimating statistics of a single bar of OHLC data, such as volatility, skew, and higher moments.
win_dow	the size of the lookback window, equal to the number of bars of data used for calculating the rolling mean.
weight_ed	Boolean should statistic be weighted by trade volume? (default TRUE)
...	additional parameters to the mo_ment function.

Details

Calculates a vector of statistics over an OHLC time series, such as volatility, skew, and higher moments. The statistics could also be any other aggregation of a single bar of OHLC data, for example the High price minus the Low price squared. The length of the vector of statistics is equal to the number of rows of the argument oh_lc. Then it calculates a trade volume weighted rolling mean over the vector of statistics over and calculate statistics.

Value

An xts time series with a single column and the same number of rows as the argument oh_lc.

Examples

```
# create time index of one minute intervals over several days
in_dex <- seq(from=as.POSIXct("2015-02-09 09:30:00"),
              to=as.POSIXct("2015-02-28 16:00:00"), by="1 min")
# create synthetic xts of random prices
x_ts <- xts(exp(cumsum(0.001*rnorm(length(in_dex)))), order.by=in_dex)
# add trade Volume data
x_ts <- merge(x_ts,
          volume=sample(x=10*(2:18),
```

```
                size=length(in_dex), replace=TRUE))
# aggregate to hours OHLC data
oh_lc <- to.period(x=x_ts, period="hours")
# calculate time series of rolling variance and skew estimates
var_rolling <- roll_moment(oh_lc=oh_lc)
skew_rolling <- roll_moment(oh_lc=oh_lc, mo_ment="run_skew")
skew_rolling <- skew_rolling/(var_rolling)^(1.5)
skew_rolling[1, ] <- 0
skew_rolling <- na.locf(skew_rolling)
```

---

roll_sharpe   *Calculate the rolling Sharpe ratio over a rolling lookback window for*
                             an OHLC *time series.*

Description

Calculate the rolling Sharpe ratio over a rolling lookback window for an OHLC time series.

Usage

```
roll_sharpe(oh_lc, win_dow = 11)
```

Arguments

oh_lc	an OHLC time series of prices in xts format.
win_dow	the size of the lookback window, equal to the number of bars of data used for aggregating the OHLC prices.

Details

Calculates the rolling Sharpe ratio as the ratio of absolute returns over the lookback window (not percentage returns), divided by the average volatility of returns.

Value

An xts time series with a single column and the same number of rows as the argument oh_lc.

Examples

```
# calculate rolling Sharpe ratio over SPY
sharpe_rolling <- roll_sharpe(oh_lc=SPY, win_dow=10)
```

---

roll_vwap   *Calculate the volume-weighted average price of an* OHLC *time series over a rolling window (lookback period).*

Description

Performs the same operation as function VWAP() from package VWAP, but using vectorized functions, so it's a little faster.

Usage

```
roll_vwap(oh_lc, x_ts = Cl(oh_lc), win_dow)
```

Arguments

oh_lc	an OHLC time series of prices in xts format.
x_ts	single-column xts time series.
win_dow	the size of the lookback window, equal to the number of bars of data used for calculating the average price.

Details

Calculates the volume-weighted average closing price, defined as the sum of the prices multiplied by trading volumes in the lookback window, divided by the sum of trading volumes in the window. If the argument x_ts is passed in explicitly, then its volume-weighted average value over time is calculated.

Value

An xts time series with a single column and the same number of rows as the argument oh_lc.

Examples

```
# calculate the volume-weighted average closing price (VWAP)
roll_vwap(oh_lc=get("SPY"), win_dow=11)
# calculate the volume-weighted average returns
roll_vwap(oh_lc=get("SPY"), x_ts=re_turns, win_dow=11)
```

---

run_returns   *Calculate single period returns from either* TAQ *or* OHLC *prices.*

Description

Calculate single period returns from either TAQ or OHLC prices.

Usage

```
run_returns(x_ts)
```

Arguments

x_ts   xts time series of either TAQ or OHLC data.

Details

Calculates single period returns as the ratio of differenced mid prices divided by the time index differences.

Value

A single-column xts time series of returns.

Examples

```
# create time index of one second intervals for a single day
in_dex <- seq(from=as.POSIXct("2015-02-09 09:30:00"), to=as.POSIXct("2015-02-09 16:00:00"), by="1 sec")
# create xts of random TAQ prices
taq_data <- xts(cumsum(rnorm(length(in_dex))), order.by=in_dex)
# create vector of random bid-offer prices
bid_offer <- abs(rnorm(length(in_dex)))/10
# create TAQ data using cbind
taq_data <- cbind(taq_data-bid_offer, taq_data+bid_offer)
# add Trade.Price
taq_data <- cbind(taq_data, taq_data+rnorm(length(in_dex))/10)
# add Volume
taq_data <- cbind(taq_data, sample(x=10*(2:18), size=length(in_dex), replace=TRUE))
colnames(taq_data) <- c("Bid.Price", "Ask.Price", "Trade.Price", "Volume")
re_turns <- run_returns(taq_data)
```

---

run_sharpe   *Calculate time series of Sharpe-like statistics for each bar of a* OHLC *time series.*

Description

Calculate time series of Sharpe-like statistics for each bar of a OHLC time series.

Usage

```
run_sharpe(oh_lc, calc_method = "close")
```

Arguments

oh_lc   an OHLC time series of prices in xts format.

calc_method   character string representing method for estimating the Sharpe-like exponent.

Details

Calculates Sharpe-like statistics for each bar of a OHLC time series. The Sharpe-like statistic is defined as the ratio of the difference between Close minus Open prices divided by the difference between High minus Low prices. This statistic may also be interpreted as something like a Hurst exponent for a single bar of data. The motivation for the Sharpe-like statistic is the notion that if prices are trending in the same direction inside a given time bar of data, then this statistic is close to either 1 or -1.

Value

An xts time series with the same number of rows as the argument oh_lc.

Examples

```
# create time index of one second intervals for a single day
in_dex <- seq(from=as.POSIXct("2015-02-09 09:30:00"),
              to=as.POSIXct("2015-02-09 16:00:00"), by="1 sec")
# create synthetic xts of random prices
x_ts <- xts(cumsum(rnorm(length(in_dex))), order.by=in_dex)
# aggregate to minutes OHLC data
oh_lc <- to.period(x=x_ts, period="minutes")
# calculate running Sharpe ratio
sharpe_running <- run_sharpe(oh_lc)
```

---

run_skew   *Calculate time series of skew estimates from a* OHLC *time series.*

Description

Calculate time series of skew estimates from a OHLC time series.

Usage

```
run_skew(oh_lc, calc_method = "rogers.satchell")
```

Arguments

oh_lc   an OHLC time series of prices in xts format.

calc_method   character string representing method for estimating skew.

Details

Calculates skew estimates from OHLC prices at each point in time (row). The methods include Garman-Klass and Rogers-Satchell.

Value

Time series of skew estimates.

Examples

```
# create time index of one second intervals for a single day
in_dex <- seq(from=as.POSIXct("2015-02-09 09:30:00"),
              to=as.POSIXct("2015-02-09 16:00:00"), by="1 sec")
# create synthetic xts of random prices
x_ts <- xts(cumsum(rnorm(length(in_dex))), order.by=in_dex)
# aggregate to minutes OHLC data
oh_lc <- to.period(x=x_ts, period="minutes")
# calculate skew estimates
sk_ew <- run_skew(oh_lc)
```

run_variance

Calculate a time series of variance estimates for an OHLC *time series.*

Description

Calculates the variance estimates for each bar of OHLC prices at each point in time (row), using the squared differences of OHLC prices at each point in time.

Usage

```
run_variance(oh_lc, calc_method = "garman.klass_yz")
```

Arguments

oh_lc   an OHLC time series of prices in xts format.

calc_method   character string representing method for estimating variance. The methods include:

- "close" close to close,
- "garman.klass" Garman-Klass,
- "garman.klass_yz" Garman-Klass with account for close-to-open price jumps,
- "rogers.satchell" Rogers-Satchell,
- "yang.zhang" Yang-Zhang,

Details

Performs a similar operation as function volatility() from package [TTR](#), but without calculating a running sum using runSum(). It's also a little faster because it performs less data validation. The variance estimation methods "close", "garman.klass_yz", and "yang.zhang" do account for close-to-open price jumps, while the methods "garman.klass" and "rogers.satchell" do not account for close-to-open price jumps.

Value

An xts time series with a single column and the same number of rows as the argument oh_lc.

Examples

```
# create time index of one second intervals for a single day
in_dex <- seq(from=as.POSIXct("2015-02-09 09:30:00"),
              to=as.POSIXct("2015-02-09 16:00:00"), by="1 sec")
# create synthetic xts of random prices
x_ts <- xts(cumsum(rnorm(length(in_dex))), order.by=in_dex)
# aggregate to minutes OHLC data
oh_lc <- to.period(x=x_ts, period="minutes")
# calculate variance estimates for oh_lc
var_running <- run_variance(oh_lc)
# calculate variance estimates for SPY
var_running <- run_variance(SPY, calc_method="yang.zhang")
# calculate SPY variance without overnight jumps
var_running <- run_variance(SPY, calc_method="rogers.satchell")
```

---

save_rets   *Load, scrub, aggregate, and rbind multiple days of* TAQ *data for a single symbol. Calculate returns and save them to a single '*.RData' file.*

Description

Load, scrub, aggregate, and rbind multiple days of TAQ data for a single symbol. Calculate returns and save them to a single '*.RData' file.

Usage

```
save_rets(sym_bol, data_dir = "E:/mktdata/sec/",
  output_dir = "E:/output/data/", win_dow = 51, vol_mult = 2,
  period = "minutes", tzone = "America/New_York")
```

Arguments

sym_bol	character string representing symbol or ticker.
data_dir	character string representing directory containing input '*.RData' files.
output_dir	character string representing directory containing output '*.RData' files.
win_dow	number of data points for estimating rolling volatility.
vol_mult	volatility multiplier.
period	aggregation period.
tzone	timezone to convert.

Details

The function save_rets loads multiple days of TAQ data, then scrubs, aggregates, and rbinds them into a OHLC time series. It then calculates returns using function run_returns, and stores them in a variable named 'symbol.rets', and saves them to a file called 'symbol.rets.RData'. The TAQ data files are assumed to be stored in separate directories for each 'symbol'. Each 'symbol' has its own directory (named 'symbol') in the 'data_dir' directory. Each 'symbol' directory contains multiple daily '*.RData' files, each file containing one day of TAQ data.

Value

Time series of returns and volume in xts format.

Examples

```
## Not run:
save_rets("SPY")

## End(Not run)
```

---

save_rets_ohlc | *Load* OHLC *time series data for a single symbol, calculate its returns, and save them to a single '*.RData' file, without aggregation.*

Description

Load OHLC time series data for a single symbol, calculate its returns, and save them to a single '*.RData' file, without aggregation.

Usage

```
save_rets_ohlc(sym_bol, data_dir = "E:/output/data/",
  output_dir = "E:/output/data/")
```

Arguments

sym_bol   character string representing symbol or ticker.

data_dir   character string representing directory containing input '*.RData' files.

output_dir   character string representing directory containing output '*.RData' files.

Details

The function save_rets_ohlc loads OHLC time series data from a single file. It then calculates returns using function run_returns, and stores them in a variable named 'symbol.rets', and saves them to a file called 'symbol.rets.RData'.

Value

Time series of returns and volume in xts format.

Examples

```
## Not run:
save_rets_ohlc("SPY")

## End(Not run)
```

---

save_scrub_agg | *Load, scrub, aggregate, and rbind multiple days of* TAQ *data for a single symbol, and save the* OHLC *time series to a single '*.RData' file.*

Description

Load, scrub, aggregate, and rbind multiple days of TAQ data for a single symbol, and save the OHLC time series to a single '*.RData' file.

Usage

```
save_scrub_agg(sym_bol, data_dir = "E:/mktdata/sec/",
  output_dir = "E:/output/data/", win_dow = 51, vol_mult = 2,
  period = "minutes", tzone = "America/New_York")
```

Arguments

sym_bol	character string representing symbol or ticker.
data_dir	character string representing directory containing input '*.RData' files.
output_dir	character string representing directory containing output '*.RData' files.
win_dow	number of data points for estimating rolling volatility.
vol_mult	volatility multiplier.
period	aggregation period.
tzone	timezone to convert.

Details

The function save_scrub_agg loads multiple days of TAQ data, then scrubs, aggregates, and rbinds them into a OHLC time series, and finally saves it to a single '*.RData' file. The OHLC time series is stored in a variable named 'symbol', and then it's saved to a file named 'symbol.RData' in the 'output_dir' directory. The TAQ data files are assumed to be stored in separate directories for each 'symbol'. Each 'symbol' has its own directory (named 'symbol') in the 'data_dir' directory. Each 'symbol' directory contains multiple daily '*.RData' files, each file containing one day of TAQ data.

Value

OHLC time series in xts format.

Examples

```
## Not run:
save_scrub_agg("SPY")

## End(Not run)
```

---

save_TAQ	*Load and scrub multiple days of* TAQ *data for a single symbol, and save it to multiple* '*.RData' *files.*

Description

Load and scrub multiple days of TAQ data for a single symbol, and save it to multiple '*.RData' files.

Usage

```
save_TAQ(sym_bol, data_dir = "E:/mktdata/sec/",
  output_dir = "E:/output/data/", win_dow = 51, vol_mult = 2,
  tzone = "America/New_York")
```

Arguments

sym_bol	character string representing symbol or ticker.
data_dir	character string representing directory containing input '*.RData' files.
output_dir	character string representing directory containing output '*.RData' files.
win_dow	number of data points for estimating rolling volatility.
vol_mult	volatility multiplier.
tzone	timezone to convert.

Details

The function save_TAQ loads multiple days of TAQ data, scrubs it, and saves it to '*.RData' files. It uses the same file names for output as the input file names. The TAQ data files are assumed to be stored in separate directories for each 'symbol'. Each 'symbol' has its own directory (named 'symbol') in the 'data_dir' directory. Each 'symbol' directory contains multiple daily '*.RData' files, each file containing one day of TAQ data.

Value

TAQ time series in xts format.

Examples

```
## Not run:
save_TAQ("SPY")

## End(Not run)
```

---

scrub_agg	*Scrub a single day of* TAQ *data, aggregate it, and convert to* OHLC *format.*

Description

Scrub a single day of TAQ data, aggregate it, and convert to OHLC format.

Usage

```
scrub_agg(taq_data, win_dow = 51, vol_mult = 2, period = "minutes",
  tzone = "America/New_York")
```

Arguments

taq_data	TAQ xts time series.
win_dow	number of data points for estimating rolling volatility.
vol_mult	volatility multiplier.
period	aggregation period.
tzone	timezone to convert.

Details

The function scrub_agg performs:

- index timezone conversion,

- data subset to trading hours,

- removal of duplicate time stamps,

- scrubbing of quotes with suspect bid-offer spreads,

- scrubbing of quotes with suspect price jumps,

- cbinding of mid prices with volume data,

- aggregation to OHLC using function to.period from package xts,

Valid 'period' character strings include: "minutes", "3 min", "5 min", "10 min", "15 min", "30 min", and "hours". The time index of the output time series is rounded up to the next integer multiple of 'period'.

Value

OHLC time series in xts format.

Examples

```
# create time index of one second intervals for a single day
in_dex <- seq(from=as.POSIXct("2015-02-09 09:30:00"), to=as.POSIXct("2015-02-09 16:00:00"), by="1 sec")
# create xts of random TAQ prices
x_ts <- xts(cumsum(rnorm(length(in_dex))), order.by=in_dex)
# create vector of random bid-offer prices
bid_offer <- abs(rnorm(length(in_dex)))/10
# create TAQ data using cbind
taq_data <- cbind(x_ts-bid_offer, x_ts+bid_offer)
# add Trade.Price
taq_data <- cbind(taq_data, x_ts+rnorm(length(in_dex))/10)
# add Volume
taq_data <- cbind(taq_data, sample(x=10*(2:18), size=length(in_dex), replace=TRUE))
colnames(taq_data) <- c("Bid.Price", "Ask.Price", "Trade.Price", "Volume")
# aggregate to ten minutes OHLC data
ohlc_data <- scrub_agg(taq_data, period="10 min")
chartSeries(ohlc_data, name=SPY, theme=chartTheme("white"))
```

---

scrub_TAQ   *Scrub a single day of* TAQ *data in* xts *format, without aggregation.*

Description

Scrub a single day of TAQ data in xts format, without aggregation.

Usage

```
scrub_TAQ(taq_data, win_dow = 51, vol_mult = 2,
  tzone = "America/New_York")
```

Arguments

taq_data	TAQ xts time series.
win_dow	number of data points for estimating rolling volatility.
vol_mult	volatility multiplier.
tzone	timezone to convert.

Details

The function scrub_TAQ performs the same scrubbing operations as scrub_agg, except it doesn't aggregate, and returns the TAQ data in xts format.

Value

TAQ xts time series.

Examples

```
# create time index of one second intervals for a single day
in_dex <- seq(from=as.POSIXct("2015-02-09 09:30:00"),
              to=as.POSIXct("2015-02-09 16:00:00"), by="1 sec")
# create xts of random TAQ prices
x_ts <- xts(cumsum(rnorm(length(in_dex))), order.by=in_dex)
# create vector of random bid-offer prices
bid_offer <- abs(rnorm(length(in_dex)))/10
# create TAQ data using cbind
taq_data <- cbind(x_ts-bid_offer, x_ts+bid_offer)
# add Trade.Price
taq_data <- cbind(taq_data, x_ts+rnorm(length(in_dex))/10)
# add Volume
taq_data <- cbind(taq_data, sample(x=10*(2:18), size=length(in_dex), replace=TRUE))
colnames(taq_data) <- c("Bid.Price", "Ask.Price", "Trade.Price", "Volume")
taq_data <- scrub_TAQ(taq_data)
taq_data <- scrub_TAQ(taq_data, win_dow=11, vol_mult=1)
```

---

season_ality	*Perform seasonality aggregations over a single-column* xts *time series.*

Description

Perform seasonality aggregations over a single-column xts time series.

Usage

```
season_ality(x_ts, in_dex = format(index(x_ts), "%H:%M"))
```

Arguments

x_ts	single-column xts time series.
in_dex	vector of character strings representing points in time, of the same length as the argument x_ts.

Details

Calculates the mean of values observed at the same points in time specified by the argument in_dex. An example of a daily seasonality aggregation is the average price of a stock between 9:30AM and 10:00AM every day, over many days. The argument in_dex is passed into function tapply(), and must be the same length as the argument x_ts.

Value

An xts time series with mean aggregations over the seasonality interval.

Examples

```
# calculate variance of each minutely OHLC bar of data
x_ts <- run_variance(get("SPY"))
# remove overnight variance spikes at "09:31"
in_dex <- format(index(x_ts), "%H:%M")
x_ts <- x_ts[!in_dex=="09:31", ]
# calculate daily seasonality of variance
season_ality(x_ts=x_ts)
```

---

to_period   *Aggregate an* OHLC *time series to a lower periodicity.*

Description

Given an OHLC time series at high periodicity (say seconds), calculates the OHLC prices at lower periodicity (say minutes).

Usage

```
to_period(oh_lc, period = "minutes", k = 1,
  end_points = xts::endpoints(oh_lc, period, k))
```

Arguments

oh_lc	an OHLC time series of prices in xts format.
period	aggregation interval ("seconds", "minutes", "hours", "days", "weeks", "months", "quarters", and "years").
k	number of periods to aggregate over (for example if period="minutes" and k=2, then aggregate over two minute intervals.)
end_points	an integer vector of end points.

Details

#' Performs a similar aggregation as function to.period() from package xts, but has the flexibility to aggregate to a user-specified vector of end points. The function to_period() simply calls the compiled function toPeriod() (from package xts), to perform the actual aggregations. If end_points are passed in explicitly, then the period argument is ignored.

Value

OHLC time series of prices in xts format, with a lower periodicity defined by the end_points.

Examples

```
# define end points at 10-minute intervals (SPY is minutely bars)
end_points <- rutils::end_points(SPY["2009"], inter_val=10)
# aggregate over 10-minute end_points:
to_period(x_ts=SPY["2009"], end_points=end_points)
# aggregate over days:
to_period(x_ts=SPY["2009"], period="days")
# equivalent to:
to.period(x=SPY["2009"], period="days", name=rutils::na_me(SPY))
```

Index