Package 'HighFreq'

June 4, 2016

Type Package

Title High Frequency Data Management

Version 0.1

Date 2015-05-28

Author Jerzy Pawlowski

Maintainer Jerzy Pawlowski <algoquant@algoquants.ch>

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,

LazyData true

Repository GitHub

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

RoxygenNote 5.0.1

R topics documented:

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(ohlc, esti_mator = "vari_ance", weight_ed = TRUE, ...)
```

Arguments

ohlc	OHLC time series of prices and trading volumes.
esti_mator	character string representing function for estimating the moment.
weight_ed	Boolean should estimate be weighted by trade volume (default is TRUE)?
...	additional parameters to esti_mator function.

Details

Calculates a single number representing the 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 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 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
sk_ew <- apply.daily(x=oh_lc, FUN=agg_regate, esti_mator="skew_ohlc")
```

calc_rets *Calculate returns of either* TAQ *or* OHLC *data in* xts *format.*

Description

Calculate returns of either TAQ or OHLC data in xts format.

Usage

```
calc_rets(x_ts)
```

Arguments

x_ts	xts time series of TAQ or OHLC data.

Details

The function calc_rets calculates returns and binds them with volume data.

Value

xts time series of returns and volumes.

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
x_ts <- cbind(x_ts-bid_offer, x_ts+bid_offer)
# add Trade.Price
x_ts <- cbind(x_ts, x_ts+rnorm(length(in_dex))/10)
# add Volume
x_ts <- cbind(x_ts, sample(x=10*(2:18), size=length(in_dex), replace=TRUE))
colnames(x_ts) <- c("Bid.Price", "Ask.Price", "Trade.Price", "Volume")
x_ts <- calc_rets(x_ts)
```

extreme_values *Identify extreme values in a univariate* 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`	univariate `xts` time series.
`win_dow`	number of data points for estimating rolling volatility.
`vol_mult`	volatility multiplier.

Details

Calculates rolling volatility as a quantile of values over a sliding 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:

sym_bol a `string` containing the name "SPY".

SPY an `xts` time series containing `OHLC` minute 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.

Usage

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

Format

an `xts` time series with 625425 rows of 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"])
```

`hurst_ohlc` *Calculate time series of Hurst exponent estimates for a* `OHLC` *time series.*

Description

Calculate time series of Hurst exponent estimates for a `OHLC` time series.

Usage

```
hurst_ohlc(ohlc, calc_method = "rogers.satchell")
```

Arguments

`ohlc`	`OHLC` time series of prices.
`calc_method`	`character` string representing method for estimating Hurst exponent.

Details

Calculates Hurst exponent estimates from `OHLC` prices at each point in time (bar). The methods include Garman-Klass and Rogers-Satchell.

Value

Time series of Hurst exponent 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 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 Hurst exponent
hurst_exp <- hurst_ohlc(oh_lc)
```

open_close — *Adjusts an OHLC time series to make open prices equal to the close prices from the previous period.*

Description

Adjusts an OHLC time series to make open prices equal to the close prices from the previous period.

Usage

```
open_close(x_ts)
```

Arguments

x_ts	an xts time series containing one or more columns of data.

Details

Adds or subtracts a price adjustment to make all open prices equal to the close prices from the previous period. The adjustment preserves the price differences within each bar of OHLC prices, and so preserves open to close returns, variance estimates, etc.

Value

xts OHLC time series with open prices equal to the close prices from the previous period.

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:
open_close(x_ts=SPY["2009"], end_points=end_points)
# aggregate over days:
open_close(x_ts=SPY["2009"], period="days")
# equivalent to:
to.period(x=SPY["2009"], period="days", name=rutils::na_me(SPY))
```

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

Description

Identify isolated price jumps in a univariate 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`	univariate `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 sliding 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_agg` *Calculate the rolling aggregations of a statistical estimator over a `OHLC` time series.*

Description

Calculate the rolling aggregations of a statistical estimator over a `OHLC` time series.

Usage

```
roll_agg(ohlc, esti_mator = "vari_ance", n = 20, N = 260,
  weight_ed = TRUE, ...)
```

Arguments

`ohlc`	`OHLC` time series of prices and trading volumes.
`esti_mator`	`character` string representing function for estimating the moment.
`n`	`numeric` number of periods for averaging of estimates.
`N`	`numeric` number of periods in a year (to annualize the estimates).
`weight_ed`	`Boolean` should estimate be weighted by trade volume (default `TRUE`)?
`...`	additional parameters to esti_mator function.

Details

Calculates a time series of rolling aggregations of an estimator over a `OHLC` time series of prices or returns, etc. By default the estimates are trade volume weighted.

Value

numeric time series of rolling aggregations, with the same number of rows as the input xts 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 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
vari_ance <- roll_agg(ohlc=oh_lc)
sk_ew <- roll_agg(ohlc=oh_lc, esti_mator="skew_ohlc")
sk_ew <- sk_ew/(vari_ance)^(1.5)
sk_ew[1, ] <- 0
sk_ew <- na.locf(sk_ew)
```

---

roll_sum *Calculate the rolling sum of an xts time series over a sliding window (lookback period).*

Description

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

Usage

```
roll_sum(x_ts, win_dow)
```

Arguments

x_ts	an xts time series containing one or more columns of data.
win_dow	an integer specifying the number of lookback periods.

Details

For example, if win_dow=3, then the rolling sum at any point is equal to the sum of x_ts values for that point plus two preceding points. The initial values of roll_sum() are equal to cumsum() values, so that roll_sum() doesn't return any NA values.

Value

xts time series with the same dimensions as the input series.

Examples

```
# create xts time series
x_ts <- xts(x=rnorm(1000), order.by=(Sys.time()-3600*(1:1000)))
roll_sum(x_ts=get("SPY"), win_dow=3)
```

---

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 `calc_rets`, 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 `calc_rets`, 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=sym_bol, 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 daily, weekly, monthly, and yearly seasonality aggregations over a univariate* xts *time series.*

Description

Perform daily, weekly, monthly, and yearly seasonality aggregations over a univariate xts time series.

Usage

```
season_ality(x_ts)
```

Arguments

x_ts	univariate xts time series.

Details

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.

Value

`xts` time series with aggregations over the seasonality interval.

Examples

```
season_ality(x_ts=get("SPY"))
```

---

skew_ohlc *Calculate time series of skew estimates from a* `OHLC` *time series.*

Description

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

Usage

```
skew_ohlc(ohlc, calc_method = "rogers.satchell")
```

Arguments

`ohlc`	`OHLC` time series of prices.
`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 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 <- skew_ohlc(oh_lc)
```

to_period *Aggregates an OHLC time series to 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(x_ts, period = "minutes", k = 1,
  end_points = xts::endpoints(x_ts, period, k))
```

Arguments

`x_ts`	an `xts` time series containing one or more columns of data.
`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

`xts` `OHLC` time series of lower periodicity defined by 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))
```

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

Description

Calculates 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

```
vari_ance(ohlc, calc_method = "garman.klass_yz")
```

Arguments

`ohlc`	`OHLC` time series of prices.
`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 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.

Value

A single-column `xts` time series of variance estimates, with the same number of rows as the input 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 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
vari_ance <- vari_ance(oh_lc)
# calculate variance estimates for SPY
vari_ance <- vari_ance(SPY, calc_method="yang.zhang")
```

v_wap — *Calculate the volume-weighted average price of an* OHLC *time series over a sliding 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

```
v_wap(x_ts, win_dow)
```

Arguments

x_ts	an xts time series containing one or more columns of data.
win_dow	an integer specifying the number of lookback periods.

Details

The volume-weighted average price (VWAP) over a period is defined as the sum of the prices multiplied by trading volumes, divided by the total trading volume in that period.

Value

xts time series with a single column containing the VWAP of the close prices, with the same number of rows as the input xts series.

Examples

```
# create xts time series
x_ts <- xts(x=rnorm(1000), order.by=(Sys.time()-3600*(1:1000)))
v_wap(x_ts=get("SPY"), win_dow=11)
```

Index