Package 'cryptoQuotes'

January 8, 2024

Title A Streamlined Access to OHLC-v Market Data and Sentiment Indicators

Version 1.2.1

Description This high-level API client offers a streamlined access to comprehensive cryptocurrency market data from major exchanges.
It features robust OHLC-V (Open, High, Low, Close, Volume) candle data with flexible granularity, ranging from seconds to months, and includes insightful sentiment indicators.
By aggregating data directly from leading exchanges, this package ensures a reliable and stable flow of market information, eliminating the need for complex, low-level API interactions.

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.2.3

Suggests data.table, knitr, quantmod, rmarkdown, testthat (>= 3.0.0), tidyverse

Config/testthat/edition 3

Imports cli, curl (>= 5.1.0), httr2, lifecycle, magrittr (>= 2.0.3), plotly (>= 4.10.2), RColorBrewer, rlang (>= 1.1.1), TTR, xts (>= 0.13.1), zoo (>= 1.8-12)

Depends R (>= 4.0.0)

LazyData true

VignetteBuilder knitr

URL https://serkor1.github.io/cryptoQuotes/, https://github.com/serkor1/cryptoQuotes

BugReports https://github.com/serkor1/cryptoQuotes/issues

NeedsCompilation no

Author Serkan Korkmaz [cre, aut, ctb, cph] (<https://orcid.org/0000-0002-5052-0982>), Jonas Cuzulan Hirani [ctb] (<https://orcid.org/0000-0002-9512-1993>)

Maintainer Serkan Korkmaz <serkor1@duck.com>

Repository CRAN

Date/Publication 2024-01-08 21:10:02 UTC

R topics documented:

`addBBands` *Add Bollinger Bands to the chart*

Description

[Experimental]

Bollinger Bands provide a visual representation of price volatility and are widely used by traders and investors to assess potential price reversals and trade opportunities in various financial markets, including stocks, forex, and commodities.

Usage

```
addBBands(chart, cols = c("High", "Low", "Close"), ...)
```

Arguments

`chart`	a `kline()` or `ohlc()` chart
`cols`	a vector of column names used to calculate the Bollinger Bands Default values `High`, `Low` and `Close`
`...`	See `TTR::BBands()`

Value

Invisbly returns a plotly object.

See Also

Other chart indicators: addEvents(), addFGIndex(), addLSRatio(), addMACD(), addMA(), addRSI(), addVolume(), chart()

Examples

```
# script: scr_charting
# date: 2023-10-25
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Charting in general
# script start;

# library
library(cryptoQuotes)

# charting klines
# with various indicators
chart(
  chart = kline(
    ATOMUSDT
  ) %>% addVolume() %>% addMA(
    FUN = TTR::SMA,
    n = 7
  ) %>% addMA(
    FUN = TTR::SMA,
    n = 14
  ) %>%
    addBBands() %>%
    addMACD() %>%
    addRSI()

)


# script end;
```

---

addEvents *add eventlines to the chart*

Description

[Experimental]

Common types of event indicators include earnings release dates, dividend payouts, central bank interest rate decisions, chart pattern breakouts, and geopolitical events like elections or geopolitical tensions. The choice of event indicators depends on the trader's or analyst's specific objectives and the factors they believe are most relevant to the asset's price movements.

Usage

```
addEvents(chart, event)
```

Arguments

chart	a kline() or ohlc() chart
event	a data.frame with index, event and colors.

Value

Invisbly returns a plotly object.

Note

The eventlines are drawn using plotly::layout(), so all existing eventlines will be replaced each time you call addEvents().

See Also

Other chart indicators: addBBands(), addFGIndex(), addLSRatio(), addMACD(), addMA(), addRSI(), addVolume(), chart()

Examples

```
# script: scr_addEvents
# date: 2023-12-07
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Describe the usage
# of addEvents
# script start;

# laod library
library(cryptoQuotes)

# 1) Generate random events
# of buys and sells and convert
# to data.frame
#
# Note: tibbles, data.tables are also supported
# but only base R is shown here to avoid
# too many dependencies
set.seed(1903)
event_data <- ATOMUSDT[
  sample(1:nrow(ATOMUSDT), size = 2)
]

# 1.1) Extract the index
# from the event data
index <- zoo::index(
  event_data
)
```

```
# 1.2) Convert the coredata
# into a data.frame
event_data <- as.data.frame(
  zoo::coredata(event_data)
)

# 1.3) Add the index into the data.frame
# case insensitive
event_data$index <- index

# 1.4) add events to the data.
# here we use Buys and Sells.
event_data$event <- rep(
  x = c('Buy', 'Sell'),
  lenght.out = nrow(event_data)
)

# 1.5) add colors based
# on the event; here buy is colored
# darkgrey, and if the position is closed
# with profit the color is green
event_data$color <- ifelse(
  event_data$event == 'Buy',
  yes = 'darkgrey',
  no  = ifelse(
    subset(event_data, event == 'Buy')$Close < subset(event_data, event == 'Sell')$Close,
    yes = 'green',
    no  = 'red'
  )
)

# 1.6) modify the event to add
# closing price at each event
event_data$event <- paste0(
  event_data$event, ' @', event_data$Close
)

# 2) Chart the the klines
# and add the buy and sell events
chart(
  chart = kline(
    ATOMUSDT
  ) %>% addEvents(
    event = event_data
  )
)

# script end;
```

---

addFGIndex *Chart the Fear and Greed Index*

Description

[Experimental]

The fear and greed index is a market sentiment indicator that measures investor emotions to gauge whether they are generally fearful (indicating potential selling pressure) or greedy (indicating potential buying enthusiasm)

Usage

```
addFGIndex(chart, FGI)
```

Arguments

chart	a kline() or ohlc() chart
FGI	The Fear and Greed Index created by getFGIndex()

Details

The Fear and Greed Index goes from 0-100, and can be classifed as follows

- 0-24, Extreme Fear
- 25-44, Fear
- 45-55, Neutral
- 56-75, Greed
- 76-100, Extreme Greed

Value

Invisbly returns a plotly object.

See Also

Other chart indicators: addBBands(), addEvents(), addLSRatio(), addMACD(), addMA(), addRSI(), addVolume(), chart()

Examples

```
# script: Fear and Greed Index
# date: 2023-12-26
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Retrieve and Plot the
# index
# script start;

# 1) get the fear and greed index
# over time
FGI <- try(
  cryptoQuotes::getFGIndex()
)
```

```
# 2) get BTCUSDT-pair on
# daily
BTCUSDT <- try(
  cryptoQuotes::getQuote(
    ticker = 'BTCUSDT',
    interval = '1d',
    futures = FALSE
  )
)

# 3) chart the klines
# of BTCUSDT with
# the Fear and Greed Index
if (!inherits(BTCUSDT, 'try-error') & !inherits(FGI, 'try-error')) {

  cryptoQuotes::chart(
    chart = cryptoQuotes::kline(
      BTCUSDT
    ) %>% cryptoQuotes::addFGIndex(
      FGI = FGI
    ),
    slider = FALSE
  )

}

# script end;
```

`addLSRatio` *Chart the long-short ratios*

Description

[Experimental]

The long-short ratio is a market sentiment indicator on expected price movement.

Usage

```
addLSRatio(chart, LSR)
```

Arguments

`chart`	a `kline()` or `ohlc()` chart
`LSR`	The Fear and Greed Index created by `getLSRatio()`

Value

Invisbly returns a plotly object.

See Also

Other chart indicators: addBBands(), addEvents(), addFGIndex(), addMACD(), addMA(), addRSI(), addVolume(), chart()

Examples

```
# Example on loading
# long-short ratio
# for the last days
# on the 15 minute candle
# wrapped in try to avoid
# failure on Github

# 1) long-short ratio
# on BTCUSDT pair
BTC_LSR <- try(
  expr = cryptoQuotes::getLSRatio(
    ticker = 'BTCUSDT',
    interval = '15m',
    from = Sys.Date() - 1,
    to   = Sys.Date()
  ),
  silent = TRUE
)

# 2) BTCSDT in same period
# as the long-short ratio;
BTCUSDT <- try(
  cryptoQuotes::getQuote(
    ticker = 'BTCUSDT',
    futures = TRUE,
    interval = '15m',
    from = Sys.Date() - 1,
    to   = Sys.Date()
  )
)

if (!inherits(x = BTC_LSR, what = 'try-error') & !inherits(x = BTCUSDT, what = "try-error")) {

  # 3) head the data
  # and display contents
  head(
    BTC_LSR
  )

  # 4) plot BTCUSDT-pair
  # with long-short ratio
  cryptoQuotes::chart(
    chart = cryptoQuotes::kline(
      BTCUSDT
    ) %>% cryptoQuotes::addLSRatio(
      LSR = BTC_LSR
```

```
    )
  )

}

# end of scrtipt;
```

---

addMA *Add various Moving Average indicators to the chart*

Description

[Experimental]

Moving averages are versatile tools used by traders and analysts in various timeframes, from short-term intraday trading to long-term investing. They help smooth out noise in price data and provide valuable information for decision-making in financial markets.

Usage

```
addMA(chart, FUN = TTR::SMA, ...)
```

Arguments

chart	a `kline()` or `ohlc()` chart
FUN	A named function calculating MAs. Has to be explicitly called. See `TTR::SMA()` for more information.
...	See `TTR::SMA()`

Details

The function supports all moving averages calculated by the TTR library. See `TTR::SMA()` for more information.

Value

Invisbly returns a plotly object.

See Also

Other chart indicators: `addBBands()`, `addEvents()`, `addFGIndex()`, `addLSRatio()`, `addMACD()`, `addRSI()`, `addVolume()`, `chart()`

Examples

```
# script: scr_charting
# date: 2023-10-25
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Charting in general
# script start;

# library
library(cryptoQuotes)

# charting klines
# with various indicators
chart(
  chart = kline(
    ATOMUSDT
  ) %>% addVolume() %>% addMA(
    FUN = TTR::SMA,
    n = 7
  ) %>% addMA(
    FUN = TTR::SMA,
    n = 14
  ) %>%
    addBBands() %>%
    addMACD() %>%
    addRSI()

)


# script end;
```

---

addMACD *Add MACD indicators to the chart*

Description

[Experimental]

Traders and investors use the MACD indicator to identify trend changes, potential reversals, and overbought or oversold conditions in the market. It is a versatile tool that can be applied to various timeframes and asset classes, making it a valuable part of technical analysis for many traders.

Usage

```
addMACD(chart, ...)
```

Arguments

chart	a kline() or ohlc() chart
...	See TTR::MACD()

Value

Invisbly returns a plotly object.

See Also

Other chart indicators: addBBands(), addEvents(), addFGIndex(), addLSRatio(), addMA(), addRSI(), addVolume(), chart()

Examples

```
# script: scr_charting
# date: 2023-10-25
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Charting in general
# script start;

# library
library(cryptoQuotes)

# charting klines
# with various indicators
chart(
  chart = kline(
    ATOMUSDT
  ) %>% addVolume() %>% addMA(
    FUN = TTR::SMA,
    n = 7
  ) %>% addMA(
    FUN = TTR::SMA,
    n = 14
  ) %>%
    addBBands() %>%
    addMACD() %>%
    addRSI()

)


# script end;
```

---

addRSI *Add RSI indicators to your chart*

Description

[Experimental]

The RSI can be customized with different look-back periods to suit various trading strategies and timeframes. It is a valuable tool for assessing the momentum and relative strength of an asset, helping traders make more informed decisions in financial markets.

Usage

```
addRSI(chart, ...)
```

Arguments

chart	a kline() or ohlc() chart
...	See TTR::RSI()

Value

Invisbly returns a plotly object.

See Also

Other chart indicators: addBBands(), addEvents(), addFGIndex(), addLSRatio(), addMACD(), addMA(), addVolume(), chart()

Examples

```
# script: scr_charting
# date: 2023-10-25
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Charting in general
# script start;

# library
library(cryptoQuotes)

# charting klines
# with various indicators
chart(
  chart = kline(
    ATOMUSDT
  ) %>% addVolume() %>% addMA(
    FUN = TTR::SMA,
    n = 7
  ) %>% addMA(
    FUN = TTR::SMA,
    n = 14
  ) %>%
    addBBands() %>%
    addMACD() %>%
    addRSI()

)


# script end;
```

addVolume *Add volume indicators to the chart*

Description

[Experimental]

Volume indicators are technical analysis tools used to analyze trading volume, which represents the number of shares or contracts traded in a financial market over a specific period of time. These indicators provide valuable insights into the strength and significance of price movements.

Usage

```
addVolume(chart)
```

Arguments

chart	a kline() or ohlc() chart

Value

Invisbly returns a plotly object.

See Also

Other chart indicators: addBBands(), addEvents(), addFGIndex(), addLSRatio(), addMACD(), addMA(), addRSI(), chart()

Examples

```
# script: scr_charting
# date: 2023-10-25
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Charting in general
# script start;

# library
library(cryptoQuotes)

# charting klines
# with various indicators
chart(
  chart = kline(
    ATOMUSDT
  ) %>% addVolume() %>% addMA(
    FUN = TTR::SMA,
    n = 7
  ) %>% addMA(
    FUN = TTR::SMA,
    n = 14
```

```
  ) %>%
    addBBands() %>%
    addMACD() %>%
    addRSI()

)


# script end;
```

ATOMUSDT *USDT denominated ATOMS with 15m intervals*

Description

A xts object with 15m OHLCV of USDT denominated ATOM with 97 rows and 5 columns, from 2023-01-01 to 2023-01-02.

Usage

```
ATOMUSDT
```

Format

An object of class `xts` (inherits from `zoo`) with 97 rows and 5 columns.

Details

Open Opening price

High Highest price

Low Lowest price

Close Closing price

Volume Volume

See Also

Other data: BTCUSDT, DOGEUSDT, FGIndex

availableExchanges *Get available exchanges*

Description

This function returns all available exchanges as a message in the console.

Usage

```
availableExchanges()
```

Value

Invisbly returns a character vector.

Examples

```
# script:
# date: 2023-10-06
# author: Serkan Korkmaz, serkor1@duck.com
# objective:
# script start;

## return all
## available exchanges
cryptoQuotes::availableExchanges()

# script end;
```

availableIntervals *See all available intervals for the futures and spot markets on the desired exchange*

Description

This function shows all available intervals available from each exchange

Usage

```
availableIntervals(source = "binance", futures = TRUE)
```

Arguments

source	character vector of length one. Must be the name of the supported exchange
futures	logical. TRUE by default. If FALSE, spot market are returned

Value

Invisbly returns a character vector.

Examples

```
# script:
# date: 2023-10-06
# author: Serkan Korkmaz, serkor1@duck.com
# objective:
# script start;

# available intervals
# at kucoin futures market
cryptoQuotes::availableIntervals(
  source = 'kucoin',
  futures = TRUE
)

# available intervals
# at kraken spot market
cryptoQuotes::availableIntervals(
  source = 'kraken',
  futures = FALSE
)

# script end;
```

`availableTickers` *Get all the available tickers on the desired exchange and market*

Description

This function returns all available pairs on the exchanges.

Usage

```
availableTickers(source = "binance", futures = TRUE)
```

Arguments

`source`	a character vector of length 1. The source of the API
`futures`	a logical value. Default TRUE.

Value

Returns a character vector of length N equal to the tradable tickers

Examples

```
## available tickers
## in Binance spot market
head(
  try(
    cryptoQuotes::availableTickers(
      source = 'binance',
      futures = FALSE
    )
  )
)


## available tickers
## in Kraken futures market
head(
  try(
    cryptoQuotes::availableTickers(
      source = 'kraken',
      futures = TRUE
    )
  )
)
```

BTCUSDT *USDT denominated Bitcoin(BTC) with 1 week intervals*

Description

A xts object with weekly OHLCV of USDT denominated Bitcoin with 99 rows and 5 columns, from 2022-02-07 to 2023-12-25.

Usage

```
BTCUSDT
```

Format

An object of class `xts` (inherits from `zoo`) with 99 rows and 5 columns.

Details

Open Opening price

High Highest price

Low Lowest price

Close Closing price

Volume Volume

See Also

Other data: ATOMUSDT, DOGEUSDT, FGIndex

`calibrateWindow` *calibrate the time window of a list of xts objects*

Description

[Experimental]

This function is a high-level wrapper of do.call and lapply which modifies each xts object stored in a `list()`.

Usage

```
calibrateWindow(list, FUN, ...)
```

Arguments

`list`	A list of xts objects.
`FUN`	A function applied to each element of the list
`...`	optional arguments passed to `FUN`.

Value

Returns a xts object.

See Also

Other convinience: `removeBound()`, `splitWindow()`

Examples

```
# script: scr_FUN
# date: 2023-12-27
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Demonstrate the use of the convinience
# funtions
# script start;

# by default the Fear and Greed Index
# is given daily. So to align these values
# with, say, weekly candles it has to be aggregated
#
# In this example the built-in data are used

# 1) check index of BTCUSDT and
# the Fear and Greed Index
setequal(
```

```
  zoo::index(BTCUSDT),
  zoo::index(FGIndex)
)

# 2) to align the indices,
# we use the convincience functions
# by splitting the FGI by the BTC index.
FGIndex <- splitWindow(
  xts = FGIndex,
  by  = zoo::index(BTCUSDT),

  # Remove upper bounds of the
  # index to avoid overlap between
  # the dates.
  #
  # This ensures that the FGI is split
  # according to start of each weekly
  # BTC candle
  bounds = 'upper'
)

# 3) as splitWindow returns a list
# it needs to passed into calibrateWindow
# to ensure comparability
FGIndex <- calibrateWindow(
  list = FGIndex,

  # As each element in the list can include
  # more than one row, each element needs to be aggregated
  # or summarised.
  #
  # using xts::first gives the first element
  # of each list, along with its values
  FUN  = xts::first
)


# 3) check if candles aligns
# accordingly
setequal(
  zoo::index(BTCUSDT),
  zoo::index(FGIndex)
)


# As the dates are now aligned
# and the Fear and Greed Index being summarised by
# the first value, the Fear and Greed Index is the opening
# Fear and Greed Index value, at each candle.

# script end;
```

chart *Create an interactive financial chart*

Description

[Stable]

Chart the `kline()` or `ohlc()` with optional indicators.

Usage

```
chart(chart, slider = TRUE)
```

Arguments

chart	a `kline()` or `ohlc()` chart with optional indicators.
slider	A logical value. TRUE by default. Include a slider in the bottom of the chart.

Value

Returns a plotly object

See Also

Other chart indicators: `addBBands()`, `addEvents()`, `addFGIndex()`, `addLSRatio()`, `addMACD()`, `addMA()`, `addRSI()`, `addVolume()`

Other charting: `kline()`, `ohlc()`

Examples

```
# script: scr_charting
# date: 2023-10-25
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Charting in general
# script start;

# library
library(cryptoQuotes)

# charting klines
# with various indicators
chart(
  chart = kline(
    ATOMUSDT
  ) %>% addVolume() %>% addMA(
    FUN = TTR::SMA,
    n = 7
  ) %>% addMA(
    FUN = TTR::SMA,
```

```
    n = 14
  ) %>%
    addBBands() %>%
    addMACD() %>%
    addRSI()

)


# script end;
```

DOGEUSDT *USDT denominated DOGECOIN in 1m intervals*

Description

A xts object with 1m OHLCV of USDT denominated Dogecoin with 61 rows and 5 columns.

Usage

```
DOGEUSDT
```

Format

An object of class `xts` (inherits from `zoo`) with 61 rows and 5 columns.

Details

Open Opening price

High Highest price

Low Lowest price

Close Closing price

Volume Volume

See Also

Other data: `ATOMUSDT`, `BTCUSDT`, `FGIndex`

FGIndex *Fear and Greed Index Values*

Description

A xts object with Fear and Greed Index value. It has 689 rows, and 1 colum. Extracted from 2022-02-07 to 2023-12-27

Usage

```
FGIndex
```

Format

An object of class `xts` (inherits from `zoo`) with 689 rows and 1 columns.

Details

FGI Daily Fear and Greed Index Value

See Also

Other data: `ATOMUSDT`, `BTCUSDT`, `DOGEUSDT`

getFGIndex *Get the daily Fear and Greed Index for the cryptocurrency market*

Description

The fear and greed index is a market sentiment indicator that measures investor emotions to gauge whether they are generally fearful (indicating potential selling pressure) or greedy (indicating potential buying enthusiasm)

Usage

```
getFGIndex(from = NULL, to = NULL)
```

Arguments

`from`	An optional vector of length 1. Can be `Sys.Date()`-class, `Sys.time()`-class or `as.character()` in %Y-%m-%d format.
`to`	An optional vector of length 1. Can be `Sys.Date()`-class, `Sys.time()`-class or `as.character()` in %Y-%m-%d format.

Details

The Fear and Greed Index goes from 0-100, and can be classifed as follows

- 0-24, Extreme Fear
- 25-44, Fear
- 45-55, Neutral
- 56-75, Greed
- 76-100, Extreme Greed

Value

A xts object with the FGI daily score

See Also

Other sentiment: `getLSRatio()`

Examples

```
# script: Fear and Greed Index
# date: 2023-12-26
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Retrieve and Plot the
# index
# script start;

# 1) get the fear and greed index
# over time
FGI <- try(
  cryptoQuotes::getFGIndex()
)

# 2) get BTCUSDT-pair on
# daily
BTCUSDT <- try(
  cryptoQuotes::getQuote(
    ticker = 'BTCUSDT',
    interval = '1d',
    futures = FALSE
  )
)

# 3) chart the klines
# of BTCUSDT with
# the Fear and Greed Index
if (!inherits(BTCUSDT, 'try-error') & !inherits(FGI, 'try-error')) {

  cryptoQuotes::chart(
    chart = cryptoQuotes::kline(
      BTCUSDT
```

```
    ) %>% cryptoQuotes::addFGIndex(
      FGI = FGI
    ),
    slider = FALSE
  )

}

# script end;
```

`getLSRatio` *Get long-short ratios for tickers*

Description

The long-short ratio is a market sentiment indicator on expected price movement

Usage

```
getLSRatio(ticker, interval = "1d", top = FALSE, from = NULL, to = NULL)
```

Arguments

`ticker`	A character vector of length 1. Uppercase. See `availableTickers()` for available tickers.
`interval`	A character vector of length 1. See `availableIntervals()` for available intervals.
`top`	A logical vector. FALSE by default. If TRUE it returns the top traders Long-Short ratios.
`from`	An optional vector of length 1. Can be `Sys.Date()`-class, `Sys.time()`-class or `as.character()` in %Y-%m-%d format.
`to`	An optional vector of length 1. Can be `Sys.Date()`-class, `Sys.time()`-class or `as.character()` in %Y-%m-%d format.

Details

Note! This endpoint only supports intervals between 5 minutes and 1 day.

Value

A xts object with the share of long and short position, and the ratio of the two. If no from and to are supplied the 100 most recent pips are returned.

Author(s)

Jonas Cuzulan Hirani

See Also

Other sentiment: `getFGIndex()`

Examples

```
# Example on loading
# long-short ratio
# for the last days
# on the 15 minute candle
# wrapped in try to avoid
# failure on Github

# 1) long-short ratio
# on BTCUSDT pair
BTC_LSR <- try(
  expr = cryptoQuotes::getLSRatio(
    ticker = 'BTCUSDT',
    interval = '15m',
    from = Sys.Date() - 1,
    to   = Sys.Date()
  ),
  silent = TRUE
)

# 2) BTCSDT in same period
# as the long-short ratio;
BTCUSDT <- try(
  cryptoQuotes::getQuote(
    ticker = 'BTCUSDT',
    futures = TRUE,
    interval = '15m',
    from = Sys.Date() - 1,
    to   = Sys.Date()
  )
)

if (!inherits(x = BTC_LSR, what = 'try-error') & !inherits(x = BTCUSDT, what = "try-error")) {

  # 3) head the data
  # and display contents
  head(
    BTC_LSR
  )

  # 4) plot BTCUSDT-pair
  # with long-short ratio
  cryptoQuotes::chart(
    chart = cryptoQuotes::kline(
      BTCUSDT
    ) %>% cryptoQuotes::addLSRatio(
      LSR = BTC_LSR
    )
```

```
  )

}

# end of scrtipt;
```

---

getQuote *Get a quote on a cryptopair from one of the supported exchanges*

Description

Open, High, Low, Close, and Volume (OHLCV) quotes are essential pieces of information used to analyze the price and trading activity of a financial asset over a specific time frame.

Usage

```
getQuote(
 ticker,
 source   = 'binance',
 futures  = TRUE,
 interval = '1d',
 from     = NULL,
 to       = NULL
)
```

Arguments

ticker	A character vector of length 1. Uppercase. See availableTickers() for available tickers.
source	A character vector of length 1. See availableExchanges() for available exchanges.
futures	A logical value. Returns futures market if TRUE, spot market otherwise.
interval	A character vector of length 1. See availableIntervals() for available intervals.
from	An optional vector of length 1. Can be Sys.Date()-class, Sys.time()-class or as.character() in %Y-%m-%d format.
to	An optional vector of length 1. Can be Sys.Date()-class, Sys.time()-class or as.character() in %Y-%m-%d format.

Details

If only from is provided 100 pips are returned up to Sys.time().

If only to is provided 100 pips up to the specified date is returned.

If from and to are both NULL 100 pips returned up to Sys.time()

Value

an xts object with Open, High, Low, Close and Volume. If `futures = TRUE` the prices are last prices.

Examples

```
# 1) Load BTC spot
# from Kucoin with 30 minute
# intervals
BTC <- try(
  cryptoQuotes::getQuote(
    ticker   = 'BTC-USDT',
    source   = 'kucoin',
    interval = '30m',
    futures  = FALSE,
    from     = Sys.Date() - 1
  )
)

# 2) chart the spot price
# using the chart
# function
if (!inherits(BTC, 'try-error')){

  cryptoQuotes::chart(
    chart = cryptoQuotes::kline(BTC) %>%
      cryptoQuotes::addVolume() %>%
      cryptoQuotes::addBBands()
  )

}

# script end;
```

`kline` *Chart the OHLC prices using candlesticks*

Description

[Stable]

Candlestick charts are highly visual and provide a quick and intuitive way to assess market sentiment and price action. Traders and analysts use them in conjunction with other technical analysis tools to make informed trading decisions. These charts are particularly useful for identifying key support and resistance levels, trend changes, and potential entry and exit points in financial markets.

Usage

```
kline(quote, deficiency = FALSE, slider = TRUE)
```

Arguments

quote	A cryptoQuote in xts/zoo format.
deficiency	Logical. FALSE by default, if TRUE color defiency compliant colors are used.
slider	Logical. TRUE by default. If FALSE, no slider will be included.

Value

Invisbly returns a plotly object.

See Also

Other charting: chart(), ohlc()

---

ohlc *chart quote using ohlc bars*

Description

[Stable]

Traders and analysts use OHLC bar charts to analyze price action, identify trends, support and resistance levels, and potential reversal patterns. They are especially useful for assessing the relationship between the opening and closing prices within a given time frame, which can offer insights into market sentiment and potential future price movements.

Usage

```
ohlc(quote, deficiency = FALSE, slider = TRUE)
```

Arguments

quote	A cryptoQuote in xts/zoo format.
deficiency	Logical. FALSE by default, if TRUE color defiency compliant colors are used.
slider	Logical. TRUE by default. If FALSE, no slider will be included.

Value

Invisbly returns a plotly object.

See Also

Other charting: chart(), kline()

Examples

```
# script: scr_charting
# date: 2023-10-25
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Charting in general
# script start;

# library
library(cryptoQuotes)

# charting klines
# with various indicators
chart(
  chart = kline(
    ATOMUSDT
  ) %>% addVolume() %>% addMA(
    FUN = TTR::SMA,
    n = 7
  ) %>% addMA(
    FUN = TTR::SMA,
    n = 14
  ) %>%
    addBBands() %>%
    addMACD() %>%
    addRSI()

)


# script end;
```

`removeBound` *remove upper and lower bounds from an XTS object*

Description

[Experimental]

The `stats::window()`-function has inclusive upper and lower bounds, which in some cases is an undesirable feature. This high level function removes the bounds if desired

Usage

```
removeBound(xts, bounds = c("upper"))
```

Arguments

`xts`	A xts-object that needs its bounds modified.
`bounds`	A character vector of length 1. Has to be one of `c('upper','lower','both')`. Defaults to Upper.

Value

Returns an xts-class object with its bounds removed.

See Also

Other convinience: `calibrateWindow()`, `splitWindow()`

Examples

```
# script: scr_FUN
# date: 2023-12-27
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Demonstrate the use of the convinience
# funtions
# script start;

# by default the Fear and Greed Index
# is given daily. So to align these values
# with, say, weekly candles it has to be aggregated
#
# In this example the built-in data are used

# 1) check index of BTCUSDT and
# the Fear and Greed Index
setequal(
  zoo::index(BTCUSDT),
  zoo::index(FGIndex)
)

# 2) to align the indices,
# we use the convincience functions
# by splitting the FGI by the BTC index.
FGIndex <- splitWindow(
  xts = FGIndex,
  by  = zoo::index(BTCUSDT),

  # Remove upper bounds of the
  # index to avoid overlap between
  # the dates.
  #
  # This ensures that the FGI is split
  # according to start of each weekly
  # BTC candle
  bounds = 'upper'
)

# 3) as splitWindow returns a list
# it needs to passed into calibrateWindow
# to ensure comparability
FGIndex <- calibrateWindow(
  list = FGIndex,
```

```
  # As each element in the list can include
  # more than one row, each element needs to be aggregated
  # or summarised.
  #
  # using xts::first gives the first element
  # of each list, along with its values
  FUN  = xts::first
)


# 3) check if candles aligns
# accordingly
setequal(
  zoo::index(BTCUSDT),
  zoo::index(FGIndex)
)


# As the dates are now aligned
# and the Fear and Greed Index being summarised by
# the first value, the Fear and Greed Index is the opening
# Fear and Greed Index value, at each candle.

# script end;
```

`splitWindow` *split xts object iteratively in lists of desired intervals*

Description

[Experimental]

The `splitWindow()`-function is a high level wrapper of the `stats::window()`-function which restricts the intervals between the first and second index value iteratively

Usage

```
splitWindow(xts, by, bounds = "upper")
```

Arguments

`xts`	A xts-object that needs needs to be split.
`by`	A reference `zoo::index()`-object, to be split by.
`bounds`	A character vector of length 1. Has to be one of `c('upper','lower','both')`. Defaults to Upper.

Value

Returns a list of iteratively restricted xts objects

See Also

Other convinience: calibrateWindow(), removeBound()

Examples

```
# script: scr_FUN
# date: 2023-12-27
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Demonstrate the use of the convinience
# funtions
# script start;

# by default the Fear and Greed Index
# is given daily. So to align these values
# with, say, weekly candles it has to be aggregated
#
# In this example the built-in data are used

# 1) check index of BTCUSDT and
# the Fear and Greed Index
setequal(
  zoo::index(BTCUSDT),
  zoo::index(FGIndex)
)

# 2) to align the indices,
# we use the convincience functions
# by splitting the FGI by the BTC index.
FGIndex <- splitWindow(
  xts = FGIndex,
  by  = zoo::index(BTCUSDT),

  # Remove upper bounds of the
  # index to avoid overlap between
  # the dates.
  #
  # This ensures that the FGI is split
  # according to start of each weekly
  # BTC candle
  bounds = 'upper'
)

# 3) as splitWindow returns a list
# it needs to passed into calibrateWindow
# to ensure comparability
FGIndex <- calibrateWindow(
  list = FGIndex,

  # As each element in the list can include
  # more than one row, each element needs to be aggregated
  # or summarised.
  #
```

```
  # using xts::first gives the first element
  # of each list, along with its values
  FUN  = xts::first
)


# 3) check if candles aligns
# accordingly
setequal(
  zoo::index(BTCUSDT),
  zoo::index(FGIndex)
)


# As the dates are now aligned
# and the Fear and Greed Index being summarised by
# the first value, the Fear and Greed Index is the opening
# Fear and Greed Index value, at each candle.

# script end;
```

Index