Package 'cryptoQuotes'

May 31, 2024

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

Version 1.3.1

Description This high-level API client offers a streamlined access to public cryptocurrency market data and sentiment indicators. It features OHLC-V (Open, High, Low, Close, Volume) that comes
with granularity ranging from seconds to months and essential sentiment indicators to develop and backtest trading strategies, or conduct detailed market analysis. By interacting directly with
the major cryptocurrency exchanges this package ensures a reliable, and stable, flow of market information, eliminating the need for complex, low-level API interactions or webcrawlers.

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.3.1

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

Config/testthat/edition 3

Imports cli (>= 3.6.2), curl (>= 5.2.1), jsonlite (>= 1.8.8), lifecycle (>= 1.0.4), plotly (>= 4.10.4), TTR (>= 0.24.4), utils, xts (>= 0.13.2), 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 2024-05-31 21:32:50 UTC

R topics documented:

`alma` *Add Arnaud Legoux Moving Average (ALMA) to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions. The function adds moving average indicators to the main `chart()`.

Usage

```
alma(
 price  = "close",
 n      = 9,
 offset = 0.85,
 sigma  = 6,
 ...
)
```

Arguments

`price`	A character-vector of length 1. "close" by default The name of the vector to passed into TTR::ALMA.
`n`	Number of periods to average over. Must be between 1 and `nrow(x)`, inclusive.
`offset`	Percentile at which the center of the distribution should occur.
`sigma`	Standard deviation of the distribution.
`...`	For internal use. Please ignore.

Value

A `plotly::plot_ly()`-object

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: `add_event()`, `bollinger_bands()`, `chart()`, `dema()`, `donchian_channel()`, `ema()`, `evwma()`, `fgi()`, `hma()`, `lsr()`, `macd()`, `rsi()`, `sma()`, `smi()`, `volume()`, `vwap()`, `wma()`, `zlema()`

Other moving average indicators: `dema()`, `ema()`, `evwma()`, `hma()`, `sma()`, `vwap()`, `wma()`, `zlema()`

Other main chart indicators: `add_event()`, `bollinger_bands()`, `dema()`, `donchian_channel()`, `ema()`, `evwma()`, `hma()`, `sma()`, `vwap()`, `wma()`, `zlema()`

Examples

```
# script start;

cryptoQuotes::chart(
  ticker = BTC,
  main   = kline(),
  indicator = list(
    cryptoQuotes::ema(n = 7),
    cryptoQuotes::sma(n = 14),
    cryptoQuotes::wma(n = 21)
  )
)

# script end;
```

ATOM *USDT Denominated ATOM (ATOMUSDT) 15-Minute Intervals*

Description

This dataset contains time-series data for the ATOM (ATOM) denominated in USDT (Tether), captured in 15-minute intervals. The data spans from December 30 to December 31, 2023.

Usage

```
ATOM
```

Format

An `xts::xts()`-object with 97 rows and 5 columns,

index <POSIXct> The time-index

open <numeric> Opening price

high <numeric> Highest price

low <numeric> Lowest price

close <numeric> Closing price

volume <numeric> Trading volume

Examples

```
# Load the dataset
data("ATOM")

# chart
chart(
  ticker = ATOM,
  main = kline(),
  sub = list(volume())
)
```

`available_exchanges` *Get available exchanges*

Description

[Stable]

Get a vector of all available exchanges passed into the `source` argument of the get-functions.

Usage

```
available_exchanges(
   type = "ohlc"
)
```

Arguments

`type`	character-vector of length 1. One of,

- `"ohlc"` - Available exchanges for Open, High, Low, Close and Volume market data. See the `get_quote()`-function.
- `"lsratio"` - Available exchanges for Long-Short ratios. See the `get_lsratio()`-function.
- `"fundingrate"` - Available exchanges for Funding rates. See the `get_fundingrate()`-function.
- `"interest"` - Available exchanges for Open interest on perpetual contracts on both sides. See the `get_openinterest()`-function.

Details

The endpoints supported by the `available_exchanges()` are not uniform, so exchanges available for, say, `get_lsratio()` is not necessarily the same as those available for `get_quote()`

Value

An `invisible()` character-vector containing available exchanges

Author(s)

Serkan Korkmaz

See Also

Other supported calls: `available_intervals()`, `available_tickers()`

Examples

```
# script start;

# 1) available exchanges
# on ohlc-v endpoint
cryptoQuotes::available_exchanges(
  type = "ohlc"
  )

# 2) available exchanges
# on long-short ratios
cryptoQuotes::available_exchanges(
  type = "lsratio"
)

# script end;
```

`available_intervals` *Get available intervals*

Description

[Stable]

Get available intervals for the `available_tickers()` on the `available_exchanges()`.

Usage

```
available_intervals(
   source = "binance",
   type   = "ohlc",
   futures = TRUE
)
```

Arguments

`source`	A character-vector of length 1. `binance` by default. See `available_exchanges()` for available exchanges.
`type`	character-vector of length 1. One of, • `"ohlc"` - Available exchanges for Open, High, Low, Close and Volume market data. See the `get_quote()`-function. • `"lsratio"` - Available exchanges for Long-Short ratios. See the `get_lsratio()`-function. • `"fundingrate"` - Available exchanges for Funding rates. See the `get_fundingrate()`-function. • `"interest"` - Available exchanges for Open interest on perpetual contracts on both sides. See the `get_openinterest()`-function.
`futures`	A logical-vector of length 1. TRUE by default. Returns futures market if TRUE, spot market otherwise.

Details

The endpoints supported by the available_exchanges() are not uniform, so exchanges available for, say, get_lsratio() is not necessarily the same as those available for get_quote()

Value

An invisible() character-vector containing the available intervals on the exchange, market and endpoint.

Sample output

```
#> i Available Intervals at "bybit" (futures):
#> v 1m, 3m, 5m, 15m, 30m, 1h, 2h, 4h, 6h, 12h, 1d, 1M, 1w
#> [1] "1m"  "3m"  "5m"  "15m" "30m" "1h"
```

Author(s)

Serkan Korkmaz

See Also

Other supported calls: available_exchanges(), available_tickers()

Examples

```
## Not run:
  # script start;

  # available intervals
  # at kucoin futures market
  cryptoQuotes::available_intervals(
    source  = 'kucoin',
    futures = TRUE,
    type    = "ohlc"
  )

  # available intervals
  # at kraken spot market
  cryptoQuotes::available_intervals(
    source  = 'kraken',
    futures = FALSE,
    type    = "ohlc"
  )

  # script end;

## End(Not run)
```

`available_tickers` *Get actively traded cryptocurrency pairs*

Description

[Stable]

Get actively traded cryptocurrency pairs on the `available_exchanges()`.

Usage

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

Arguments

`source`	A character-vector of length 1. `binance` by default. See `available_exchanges()` for available exchanges.
`futures`	A logical-vector of length 1. TRUE by default. Returns futures market if TRUE, spot market otherwise.

Details

The naming-conventions across, and within, `available_exchanges()` are not necessarily the same. This function lists all actively traded tickers.

Value

A character-vector of actively traded cryptocurrency pairs on the exchange, and the specified market.

Sample output

```
#> [1] "10000000AIDOGEUSDT" "1000000MOGUSDT"     "10000COQUSDT"
#> [4] "10000LADYSUSDT"     "10000NFTUSDT"       "10000SATSUSDT"
```

Author(s)

Serkan Korkmaz

See Also

Other supported calls: `available_exchanges()`, `available_intervals()`

Examples

```
## Not run:
  # 1) available tickers
  # in Binance spot market
  head(
    cryptoQuotes::available_tickers(
      source  = 'binance',
      futures = FALSE
    )
  )

  # 2) available tickers
  # on Kraken futures market
  head(
    cryptoQuotes::available_tickers(
      source  = 'kraken',
      futures = TRUE
    )
  )

## End(Not run)
```

`bollinger_bands` *Add Bollinger Bands to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with the `TTR::BBands()`-function. The function adds bollinger bands to the main `chart()`.

Usage

```
bollinger_bands(
   n = 20,
   sd = 2,
   maType = "SMA",
   color  = '#4682b4',
   ...
)
```

Arguments

n	Number of periods for moving average.
sd	The number of standard deviations to use.

maType	A function or a string naming the function to be called.
color	A character-vector of length 1. "#4682b4" by default.
...	Other arguments to be passed to the maType function.

Value

An invisible plotly::plot_ly()-object.

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), chart(), dema(), donchian_channel(), ema(), evwma(), fgi(), hma(), lsr(), macd(), rsi(), sma(), smi(), volume(), vwap(), wma(), zlema()

Other main chart indicators: add_event(), alma(), dema(), donchian_channel(), ema(), evwma(), hma(), sma(), vwap(), wma(), zlema()

Examples

```
# script start;

# Charting BTC using
# candlesticks as main
# chart
cryptoQuotes::chart(
  ticker = BTC,
  main   = cryptoQuotes::kline(),
  sub    = list(
    cryptoQuotes::volume()
  )
)

# script end;
```

BTC *USDT Denominated Bitcoin (BTCUSDT) Weekly Intervals*

Description

This dataset contains time-series data for Bitcoin (BTC) denominated in USDT (Tether), captured in weekly intervals. The data spans from January 1, 2023, to December 31, 2023.

Usage

```
BTC
```

Format

An xts::xts()-object with 52 rows and 5 columns,

index <POSIXct> The time-index

open <numeric> Opening price

high <numeric> Highest price

low <numeric> Lowest price

close <numeric> Closing price

volume <numeric> Trading volume

Examples

```
# Load the dataset
data("BTC")

# chart
chart(
  ticker = BTC,
  main = kline(),
  sub = list(volume())
)
```

`calibrate_window` *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

```
calibrate_window(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 utility: remove_bound(), split_window()

Examples

```
# script start;

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

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

  # 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 <- cryptoQuotes::calibrate_window(
  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
stopifnot(
  setequal(
    zoo::index(BTC),
```

```
    zoo::index(FGIndex)
  )
)


# script end;
```

---

chart *Build an interactive financial chart*

Description

[Experimental]

A high-level `plotly::plot_ly()`- and `plotly::subplot()`-wrapper function for building interactive financial charts using the affiliated chart-functions. The chart consists of a main chart, and an optional subchart. The main chart supports overlaying various trading indicators like sma and bollinger_bands.

Usage

```
chart(
  ticker,
  main = list(kline()),
  sub = list(),
  indicator = list(),
  event_data = NULL,
  options = list()
)
```

Arguments

`ticker`	An object with Open, High, Low, Close and Volume columns that can be coerced to a `xts::xts()`-object.
`main`	A `plotly::plot_ly()`-function. `kline()` by default.
`sub`	An optional list of `plotly::plot_ly()`-function(s).
`indicator`	An optional list of `plotly::add_lines()`-function(s).
`event_data`	An optional data.frame with event line(s) to be added to the `chart()`. See `add_event()` for more details.
`options`	An optional list of `chart()`-options. See details below.

Details

Options:

- `dark` A <logical>-value of length 1. TRUE by default. Sets the overall theme of the `chart()`
- `slider` A <logical>-value of length 1. FALSE by default. If TRUE, a `plotly::rangeslider()` is added

- `deficiency` A <logical>-value of length 1. FALSE by default. If TRUE, all `chart()`-elements are colorblind friendly
- `size` A <numeric>-value of length 1. The relative size of the main chart. 0.6 by default. Must be between 0 and 1, non-inclusive

Charting Events:

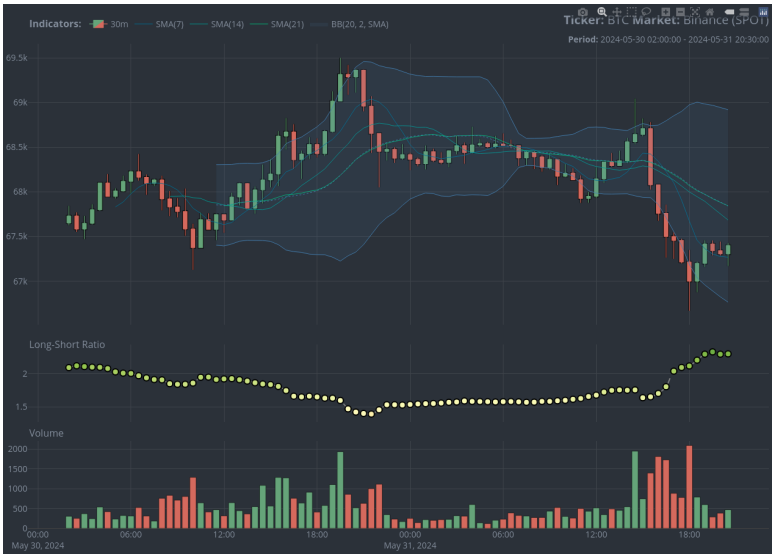
If `event_data` is passed, vertical eventlines with appropriate labels and coloring are added to the `chart()`. This function is rigid, as it will fail if event, label and index columns are not passed.

For more details please see `add_event()`.

Value

A `plotly::plot_ly()` object.

Sample Output



Author(s)

Serkan Korkmaz

See Also

Other chart indicators: `add_event()`, `alma()`, `bollinger_bands()`, `dema()`, `donchian_channel()`, `ema()`, `evwma()`, `fgi()`, `hma()`, `lsr()`, `macd()`, `rsi()`, `sma()`, `smi()`, `volume()`, `vwap()`, `wma()`, `zlema()`

Other price charts: `kline()`, `ohlc()`, `pline()`

Examples

```
# script start;
```

```
# 1) charting weekly
# BTC using candlesticks
# and indicators
cryptoQuotes::chart(
  ticker     = BTC,
  main       = cryptoQuotes::kline(),
  sub        = list(
    cryptoQuotes::volume(),
    cryptoQuotes::macd()
  ),
  indicator = list(
    cryptoQuotes::bollinger_bands(),
    cryptoQuotes::sma(),
    cryptoQuotes::alma()
  ),
  options = list(
    dark       = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

---

dema *Add Double Exponential Moving Average (DEMA) to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions. The function adds moving average indicators to the main `chart()`.

Usage

```
dema(
 price  = "close",
 n      = 10,
 v      = 1,
 wilder = FALSE,
 ratio  = NULL,
 ...
)
```

Arguments

price	A character-vector of length 1. "close" by default. The name of the vector to passed into TTR::DEMA.
n	Number of periods to average over. Must be between 1 and `nrow(x)`, inclusive.

v	The 'volume factor' (a number in [0,1]). See Notes.
wilder	logical; if TRUE, a Welles Wilder type EMA will be calculated; see notes.
ratio	A smoothing/decay ratio. ratio overrides wilder in EMA.
...	For internal use. Please ignore.

Value

A plotly::plot_ly()-object

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), donchian_channel(), ema(), evwma(), fgi(), hma(), lsr(), macd(), rsi(), sma(), smi(), volume(), vwap(), wma(), zlema()

Other moving average indicators: alma(), ema(), evwma(), hma(), sma(), vwap(), wma(), zlema()

Other main chart indicators: add_event(), alma(), bollinger_bands(), donchian_channel(), ema(), evwma(), hma(), sma(), vwap(), wma(), zlema()

Examples

```
# script start;

cryptoQuotes::chart(
  ticker = BTC,
  main   = kline(),
  indicator = list(
    cryptoQuotes::ema(n = 7),
    cryptoQuotes::sma(n = 14),
    cryptoQuotes::wma(n = 21)
  )
)

# script end;
```

DOGE *USDT Denominated DOGECOIN (DOGEUSDT) 1-Minute Intervals*

Description

This dataset contains time-series data for the DOGECOIN (DOGE) denominated in USDT (Tether), captured in 1-minute intervals. The data spans 2022-01-14 07:00:00 CET to 2022-01-14 08:00:00 CET.

Usage

```
DOGE
```

Format

An xts::xts()-object with 61 rows and 5 columns,

index <POSIXct> The time-index

open <numeric> Opening price

high <numeric> Highest price

low <numeric> Lowest price

close <numeric> Closing price

volume <numeric> Trading volume

Examples

```
# Load the dataset
data("DOGE")

# chart
chart(
  ticker = DOGE,
  main = kline(),
  sub = list(volume())
)
```

---

donchian_channel *Add Donchian Channels to the chart*

Description

[Experimental]

A high-level plotly::add_lines()-wrapper function that interacts with the TTR::DonchianChannel()-function. The function adds Donchian Channels to the main chart().

Usage

```
donchian_channel(
 n = 10,
 include.lag = FALSE,
 color = '#4682b4',
 ...
)
```

Arguments

n	Number of periods for moving average.
include.lag	Should values be lagged so that today's prices are not included in the calculation? See Note.
color	A character-vector of length 1. "#4682b4" by default.
...	For internal use. Please ignore.

Value

An invisible plotly::plot_ly()-object.

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), ema(), evwma(), fgi(), hma(), lsr(), macd(), rsi(), sma(), smi(), volume(), vwap(), wma(), zlema()

Other main chart indicators: add_event(), alma(), bollinger_bands(), dema(), ema(), evwma(), hma(), sma(), vwap(), wma(), zlema()

Examples

```
# script start;

# Charting BTC using
# candlesticks as main
# chart
cryptoQuotes::chart(
  ticker = BTC,
  main   = cryptoQuotes::kline(),
  sub    = list(
    cryptoQuotes::volume()
  )
)

# script end;
```

---

ema *Add Exponentially-Weighted Moving Average (EMA) to the chart*

Description

[Experimental]

A high-level plotly::add_lines()-wrapper function that interacts with TTR's moving average family of functions. The function adds moving average indicators to the main chart().

Usage

```
ema(
 price  = "close",
 n      = 10,
 wilder = FALSE,
 ratio  = NULL,
 ...
)
```

Arguments

price	A character-vector of length 1. "close" by default. The name of the vector to passed into TTR::EMA.
n	Number of periods to average over. Must be between 1 and nrow(x), inclusive.
wilder	logical; if TRUE, a Welles Wilder type EMA will be calculated; see notes.
ratio	A smoothing/decay ratio. ratio overrides wilder in EMA.
...	For internal use. Please ignore.

Value

A plotly::plot_ly()-object

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), donchian_channel(), evwma(), fgi(), hma(), lsr(), macd(), rsi(), sma(), smi(), volume(), vwap(), wma(), zlema()

Other moving average indicators: alma(), dema(), evwma(), hma(), sma(), vwap(), wma(), zlema()

Other main chart indicators: add_event(), alma(), bollinger_bands(), dema(), donchian_channel(), evwma(), hma(), sma(), vwap(), wma(), zlema()

Examples

```
# script start;

cryptoQuotes::chart(
  ticker = BTC,
  main   = kline(),
  indicator = list(
    cryptoQuotes::ema(n = 7),
    cryptoQuotes::sma(n = 14),
    cryptoQuotes::wma(n = 21)
  )
)

# script end;
```

evwma *Add Elastic Volume-Weighted Moving Average (EVWMA) to the chart*

Description

[Experimental]

A high-level plotly::add_lines()-wrapper function that interacts with TTR's moving average family of functions. The function adds moving average indicators to the main chart().

Usage

```
evwma(
 price  = "close",
 n      = 10,
 ...
)
```

Arguments

price	A character-vector of length 1. "close" by default. The name of the vector to passed into TTR::EVWMA
n	Number of periods to average over. Must be between 1 and nrow(x), inclusive.
...	For internal use. Please ignore.

Value

A plotly::plot_ly()-object

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), donchian_channel(), ema(), fgi(), hma(), lsr(), macd(), rsi(), sma(), smi(), volume(), vwap(), wma(), zlema()

Other moving average indicators: alma(), dema(), ema(), hma(), sma(), vwap(), wma(), zlema()

Other main chart indicators: add_event(), alma(), bollinger_bands(), dema(), donchian_channel(), ema(), hma(), sma(), vwap(), wma(), zlema()

Examples

```
# script start;

cryptoQuotes::chart(
  ticker = BTC,
  main   = kline(),
  indicator = list(
    cryptoQuotes::ema(n = 7),
    cryptoQuotes::sma(n = 14),
    cryptoQuotes::wma(n = 21)
  )
)

# script end;
```

`fgi` *Chart the Fear and Greed Index*

Description

[Experimental]

A high-level `plotly::plot_ly()`-wrapper function. The function adds a subchart with the `fear and greed`-index.

Usage

```
fgi(index, ...)
```

Arguments

`index`	A `xts::xts()`-object. See `get_fgindex()` for more details.
`...`	For internal use. Please ignore.

Details

Classification:

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

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

About the Fear and Greed Index:

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).

Source:

This index is fetched from alternative.me, and can be different from the one provided by coinmarketcap.

Value

An invisible `plotly::plot_ly()`-object.

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: `add_event()`, `alma()`, `bollinger_bands()`, `chart()`, `dema()`, `donchian_channel()`, `ema()`, `evwma()`, `hma()`, `lsr()`, `macd()`, `rsi()`, `sma()`, `smi()`, `volume()`, `vwap()`, `wma()`, `zlema()`

Other sentiment indicators: `lsr()`

Other subchart indicators: `add_event()`, `lsr()`, `macd()`, `rsi()`, `smi()`, `volume()`

Examples

```
## Not run:
  # script start;

  # 1) get the fear and greed index
  # for the last 14 days
  FGIndex <- cryptoQuotes::get_fgindex(
    from = Sys.Date() - 14
  )

  # 2) get the BTC price
  # for the last 14 days
  BTC <- cryptoQuotes::get_quote(
    ticker  = "BTCUSDT",
    source  = "bybit",
    futures = FALSE,
    from    = Sys.Date() - 14
  )

  # 3) chart the daily BTC
  # along side the Fear and
  # Greed Index
  cryptoQuotes::chart(
    ticker = BTC,
    main   = kline(),
    sub    = list(
      fgi(
        FGIndex
      )
    )
  )
```

```
  # script end;

## End(Not run)
```

`FGIndex` *Fear and Greed Index (FGI) values for the cryptocurrency market in daily intervals*

Description

This dataset contains daily values of the Fear and Greed Index for the year 2023, which is used to measure the sentiments of investors in the market. The data spans from January 1, 2023, to December 31, 2023.

Usage

```
FGIndex
```

Format

An `xts::xts()`-object with 364 rows and 1 columns,

index <POSIXct> The time-index

fgi <numeric< The daily fear and greed index value

Details

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

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

Examples

```
# Load the dataset
data("FGIndex")

# Get a summary of index values
summary(FGIndex)
```

`get_fgindex` *Get the daily Fear and Greed Index in the cryptocurrency market*

Description

[Stable]

Get the daily fear and greed index.

Usage

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

Arguments

`from`	An optional character-, date- or POSIXct-vector of length 1. NULL by default.
`to`	An optional character-, date- or POSIXct-vector of length 1. NULL by default.

Details

Classification:

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

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

About the Fear and Greed Index:

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).

Source:

This index is fetched from alternative.me, and can be different from the one provided by coinmarketcap.

Value

An xts-object containing,

`index`	<POSIXct> the time-index
`fgi`	<numeric> the daily fear and greed index value

Sample output

```
#>                     fgi
#> 2024-05-12 02:00:00  56
#> 2024-05-13 02:00:00  57
#> 2024-05-14 02:00:00  66
#> 2024-05-15 02:00:00  64
#> 2024-05-16 02:00:00  70
#> 2024-05-17 02:00:00  74
```

Author(s)

Serkan Korkmaz

See Also

Other get-functions: `get_fundingrate()`, `get_lsratio()`, `get_openinterest()`, `get_quote()`

Examples

```
## Not run:
  # script start;

  # 1) get the fear and greed index
  # for the last 7 days
  tail(
    fgi <- cryptoQuotes::get_fgindex(
      from = Sys.Date() - 7
    )
  )

  # script end;

## End(Not run)
```

---

get_fundingrate *Get the funding rate on futures contracts*

Description

[Stable]

Get the funding rate on a cryptocurrency pair from the `available_exchanges()` in any actively traded `available_tickers()` on the futures markets.

Usage

```
get_fundingrate(
 ticker,
 source   = 'binance',
 from     = NULL,
 to       = NULL
)
```

Arguments

ticker	A character-vector of length 1. See available_tickers() for available tickers.
source	A character-vector of length 1. binance by default. See available_exchanges() for available exchanges.
from	An optional character-, date- or POSIXct-vector of length 1. NULL by default.
to	An optional character-, date- or POSIXct-vector of length 1. NULL by default.

Value

An xts-object containing,

index	<POSIXct> the time-index
funding_rate	<numeric> the current funding rate

Sample output

```
#>                     funding_rate
#> 2024-03-09 17:00:00   0.00026407
#> 2024-03-10 01:00:00   0.00031010
#> 2024-03-10 09:00:00   0.00063451
#> 2024-03-10 17:00:00   0.00054479
#> 2024-03-11 01:00:00   0.00035489
#> 2024-03-11 09:00:00   0.00078428
```

Author(s)

Serkan Korkmaz

See Also

Other get-functions: get_fgindex(), get_lsratio(), get_openinterest(), get_quote()

Examples

```
## Not run:
# script start;

# 1) check available
# exchanges for funding rates
cryptoQuotes::available_exchanges(
```

```
  type = "fundingrate"
  )

# 2) get BTC funding rate
# for the last 7 days
tail(
  BTC <- cryptoQuotes::get_fundingrate(
    ticker = "BTCUSDT",
    source = "binance",
    from   = Sys.Date() - 7
  )
)

# script end;

## End(Not run)
```

`get_lsratio` *Get the long to short ratio of a cryptocurrency pair*

Description

[Stable]

Get the long-short ratio for any `available_tickers()` from the `available_exchanges()`

Usage

```
get_lsratio(
   ticker,
   interval = '1d',
   source   = 'binance',
   from     = NULL,
   to       = NULL,
   top      = FALSE
)
```

Arguments

`ticker`	A character-vector of length 1. See `available_tickers()` for available tickers.
`interval`	A character-vector of length 1. `1d` by default. See `available_intervals()` for available intervals.
`source`	A character-vector of length 1. `binance` by default. See `available_exchanges()` for available exchanges.
`from`	An optional character-, date- or POSIXct-vector of length 1. NULL by default.
`to`	An optional character-, date- or POSIXct-vector of length 1. NULL by default.
`top`	A logical vector. FALSE by default. If TRUE it returns the top traders Long-Short ratios.

Details

On time-zones and dates:

Values passed to `from` or `to` must be coercible by `as.Date()`, or `as.POSIXct()`, with a format of either `"%Y-%m-%d"` or `"%Y-%m-%d %H:%M:%S"`. By default all dates are passed and returned with `Sys.timezone()`.

On returns:

If only `from` is provided 200 pips are returned up to `Sys.time()`. If only `to` is provided 200 pips up to the specified date is returned.

Value

An xts-object containing,

`index`	<POSIXct> the time-index
`long`	<numeric> the share of longs
`short`	<numeric> the share of shorts
`ls_ratio`	<numeric> the ratio of longs to shorts

Sample output

```
#>                      long  short  ls_ratio
#> 2024-05-12 02:00:00 0.6930 0.3070 2.2573290
#> 2024-05-13 02:00:00 0.6637 0.3363 1.9735355
#> 2024-05-14 02:00:00 0.5555 0.4445 1.2497188
#> 2024-05-15 02:00:00 0.6580 0.3420 1.9239766
#> 2024-05-16 02:00:00 0.4868 0.5132 0.9485581
#> 2024-05-17 02:00:00 0.5102 0.4898 1.0416497
```

Author(s)

Jonas Cuzulan Hirani

See Also

Other get-functions: `get_fgindex()`, `get_fundingrate()`, `get_openinterest()`, `get_quote()`

Examples

```
## Not run:
  # script start;

  LS_BTC <- cryptoQuotes::get_lsratio(
    ticker   = 'BTCUSDT',
    interval = '15m',
    from     = Sys.Date() - 1,
    to       = Sys.Date()
  )
```

```
  # end of scrtipt;

## End(Not run)
```

`get_openinterest` *Get the open interest on perpetual futures contracts*

Description

[Stable]

Get the open interest on a cryptocurrency pair from the `available_exchanges()` in any actively traded `available_tickers()` on the FUTURES markets.

Usage

```
get_openinterest(
 ticker,
 interval = '1d',
 source   = 'binance',
 from     = NULL,
 to       = NULL
)
```

Arguments

`ticker`	A character-vector of length 1. See `available_tickers()` for available tickers.
`interval`	A character-vector of length 1. `1d` by default. See `available_intervals()` for available intervals.
`source`	A character-vector of length 1. `binance` by default. See `available_exchanges()` for available exchanges.
`from`	An optional character-, date- or POSIXct-vector of length 1. NULL by default.
`to`	An optional character-, date- or POSIXct-vector of length 1. NULL by default.

Details

On time-zones and dates:

Values passed to `from` or `to` must be coercible by `as.Date()`, or `as.POSIXct()`, with a format of either `"%Y-%m-%d"` or `"%Y-%m-%d %H:%M:%S"`. By default all dates are passed and returned with `Sys.timezone()`.

On returns:

If only `from` is provided 200 pips are returned up to `Sys.time()`. If only `to` is provided 200 pips up to the specified date is returned.

Value

An xts-object containing,

index	<POSIXct> the time-index
open_interest	<numeric> open perpetual contracts on both both sides

Sample output

```
#>                     open_interest
#> 2024-05-12 02:00:00      70961.07
#> 2024-05-13 02:00:00      69740.49
#> 2024-05-14 02:00:00      71110.33
#> 2024-05-15 02:00:00      67758.06
#> 2024-05-16 02:00:00      73614.70
#> 2024-05-17 02:00:00      72377.85
```

Author(s)

Serkan Korkmaz

See Also

Other get-functions: `get_fgindex()`, `get_fundingrate()`, `get_lsratio()`, `get_quote()`

Examples

```
## Not run:
  # script start;

  # 1) check available
  # exchanges for open interest
  cryptoQuotes::available_exchanges(
    type = 'interest'
    )

  # 2) get BTC funding rate
  # for the last 7 days
  tail(
    BTC <- cryptoQuotes::get_openinterest(
      ticker = "BTCUSDT",
      source = "binance",
      from   = Sys.Date() - 7
    )
  )

  # script end;

## End(Not run)
```

get_quote *Get the Open, High, Low, Close and Volume data on a cryptocurrency pair*

Description

[Stable]

Get a quote on a cryptocurrency pair from the `available_exchanges()` in various `available_intervals()` for any actively traded `available_tickers()`.

Usage

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

Arguments

`ticker`	A character-vector of length 1. See `available_tickers()` for available tickers.
`source`	A character-vector of length 1. `binance` by default. See `available_exchanges()` for available exchanges.
`futures`	A logical-vector of length 1. TRUE by default. Returns futures market if TRUE, spot market otherwise.
`interval`	A character-vector of length 1. `1d` by default. See `available_intervals()` for available intervals.
`from`	An optional character-, date- or POSIXct-vector of length 1. NULL by default.
`to`	An optional character-, date- or POSIXct-vector of length 1. NULL by default.

Details

On time-zones and dates:

Values passed to `from` or `to` must be coercible by `as.Date()`, or `as.POSIXct()`, with a format of either `"%Y-%m-%d"` or `"%Y-%m-%d %H:%M:%S"`. By default all dates are passed and returned with `Sys.timezone()`.

On returns:

If only `from` is provided 200 pips are returned up to `Sys.time()`. If only `to` is provided 200 pips up to the specified date is returned.

Value

An xts-object containing,

index	<POSIXct> The time-index
open	<numeric> Opening price
high	<numeric> Highest price
low	<numeric> Lowest price
close	<numeric> Closing price
volume	<numeric> Trading volume

Sample output

```
#>                          open    high     low   close   volume
#> 2024-05-12 02:00:00 60809.2 61849.4 60557.3 61455.8 104043.9
#> 2024-05-13 02:00:00 61455.7 63440.0 60750.0 62912.1 261927.1
#> 2024-05-14 02:00:00 62912.2 63099.6 60950.0 61550.5 244345.3
#> 2024-05-15 02:00:00 61550.5 66440.0 61316.1 66175.4 365031.7
#> 2024-05-16 02:00:00 66175.4 66800.0 64567.0 65217.7 242455.3
#> 2024-05-17 02:00:00 65217.7 66478.5 65061.2 66218.8  66139.1
```

Author(s)

Serkan Korkmaz

See Also

Other get-functions: `get_fgindex()`, `get_fundingrate()`, `get_lsratio()`, `get_openinterest()`

Examples

```
## Not run:
  # script start;

  # get quote on
  # BTCUSDT pair from
  # Binance in 30m
  # intervals from the
  # last 24 hours
  tail(
    BTC <- cryptoQuotes::get_quote(
      ticker   = 'BTCUSDT',
      source   = 'binance',
      interval = '30m',
      futures  = FALSE,
      from     = Sys.Date() - 1
    )
  )

  # script end;
```

```
## End(Not run)
```

---

hma *Add Hull Moving Average (HMA) to the chart*

Description

[Experimental]

A high-level plotly::add_lines()-wrapper function that interacts with TTR's moving average family of functions. The function adds moving average indicators to the main chart().

Usage

```
hma(
 price  = "close",
 n      = 20,
 ...
)
```

Arguments

price	A character-vector of length 1. "close" by default. The name of the vector to passed into TTR::HMA.
n	Number of periods to average over. Must be between 1 and nrow(x), inclusive.
...	For internal use. Please ignore.

Value

A plotly::plot_ly()-object

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), donchian_channel(), ema(), evwma(), fgi(), lsr(), macd(), rsi(), sma(), smi(), volume(), vwap(), wma(), zlema()

Other moving average indicators: alma(), dema(), ema(), evwma(), sma(), vwap(), wma(), zlema()

Other main chart indicators: add_event(), alma(), bollinger_bands(), dema(), donchian_channel(), ema(), evwma(), sma(), vwap(), wma(), zlema()

Examples

```
# script start;

cryptoQuotes::chart(
  ticker = BTC,
  main   = kline(),
  indicator = list(
    cryptoQuotes::ema(n = 7),
    cryptoQuotes::sma(n = 14),
    cryptoQuotes::wma(n = 21)
  )
)

# script end;
```

`kline` *Candlestick Chart*

Description

[Experimental]

A high-level `plotly::plot_ly()`-function for charting Open, High, Low and Close prices.

Usage

```
kline(...)
```

Arguments

`...`	For internal use. Please ignore.

Value

An invisible `plotly::plot_ly()`-object.

Author(s)

Serkan Korkmaz

See Also

Other price charts: `chart()`, `ohlc()`, `pline()`

Examples

```
# script start;

# Charting BTC using
# candlesticks as main
# chart
cryptoQuotes::chart(
  ticker = BTC,
  main   = cryptoQuotes::kline(),
  sub    = list(
    cryptoQuotes::volume()
  )
)

# script end;
```

`lsr` *Chart the long-short ratio*

Description

[Experimental]

A high-level `plotly::plot_ly()`-wrapper function. The function adds a subchart to the chart with `long-short ratio`.

Usage

```
lsr(ratio, ...)
```

Arguments

`ratio`	A `xts::xts()`-object. See `get_lsratio()` for more details.
`...`	For internal use. Please ignore.

Value

An invisible `plotly::plot_ly()`-object.

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: `add_event()`, `alma()`, `bollinger_bands()`, `chart()`, `dema()`, `donchian_channel()`, `ema()`, `evwma()`, `fgi()`, `hma()`, `macd()`, `rsi()`, `sma()`, `smi()`, `volume()`, `vwap()`, `wma()`, `zlema()`

Other sentiment indicators: `fgi()`

Other subchart indicators: `add_event()`, `fgi()`, `macd()`, `rsi()`, `smi()`, `volume()`

Examples

```
## Not run:
  # script start;

  # 1) long-short ratio
  # on BTCUSDT pair
  LS_BTC <- cryptoQuotes::get_lsratio(
    ticker   = 'BTCUSDT',
    interval = '15m',
    from     = Sys.Date() - 1,
    to       = Sys.Date()
  )

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

  # 3) plot BTCUSDT-pair
  # with long-short ratio
  cryptoQuotes::chart(
    ticker = BTC,
    main   = cryptoQuotes::kline(),
    sub    = list(
      cryptoQuotes::lsr(
        ratio = LS_BTC
      )
    )
  )

  # end of scrtipt;

## End(Not run)
```

`macd` *Chart the Moving Average Convergence Divergence (MACD) indicator*

Description

[Experimental]

A high-level `plotly::plot_ly()`- and `plotly::add_lines()`-function that interacts with the `TTR::MACD()`-function. The function adds subchart with a `TTR::MACD()`-indicator.

Usage

```
macd(
 nFast   = 12,
 nSlow   = 26,
 nSig    = 9,
 maType  = "SMA",
 percent = TRUE,
 ...
)
```

Arguments

nFast	Number of periods for fast moving average.
nSlow	Number of periods for slow moving average.
nSig	Number of periods for signal moving average.
maType	Either: 1. A function or a string naming the function to be called. 2. A *list* with the first component like (1) above, and additional parameters specified as *named* components. See Examples.
percent	logical; if TRUE, the percentage difference between the fast and slow moving averages is returned, otherwise the difference between the respective averages is returned.
...	For internal use. Please ignore.

Value

An invisible plotly::plot_ly()-object.

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), donchian_channel(), ema(), evwma(), fgi(), hma(), lsr(), rsi(), sma(), smi(), volume(), vwap(), wma(), zlema()

Other subchart indicators: add_event(), fgi(), lsr(), rsi(), smi(), volume()

Other momentum indicators: rsi(), smi()

Examples

```
# script start;

# 1) charting weekly
# BTC using candlesticks
# and indicators
cryptoQuotes::chart(
```

```
  ticker     = BTC,
  main       = cryptoQuotes::kline(),
  sub        = list(
    cryptoQuotes::volume(),
    cryptoQuotes::macd()
  ),
  indicator = list(
    cryptoQuotes::bollinger_bands(),
    cryptoQuotes::sma(),
    cryptoQuotes::alma()
  ),
  options = list(
    dark       = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

`ohlc` *OHLC Barchart*

Description

[Experimental]

A high-level `plotly::plot_ly()`-function for charting Open, High, Low and Close prices.

Usage

```
ohlc(...)
```

Arguments

`...`	For internal use. Please ignore.

Value

An invisible `plotly::plot_ly()`-object.

Author(s)

Serkan Korkmaz

See Also

Other price charts: `chart()`, `kline()`, `pline()`

Examples

```
# script start;

# Charting BTC using
# OHLC-bars as main
# chart
cryptoQuotes::chart(
  ticker = BTC,
  main   = cryptoQuotes::ohlc(),
  sub    = list(
    cryptoQuotes::volume()
  )
)

# script end;
```

`pline` *Line Chart*

Description

[Experimental]

A high-level `plotly::plot_ly()`-function for charting Open, High, Low and Close prices.

Usage

```
pline(price = "close", ...)
```

Arguments

`price`	A character-vector of length 1. "close" by default.
`...`	For internal use. Please ignore.

Value

An invisible `plotly::plot_ly()`-object.

Author(s)

Serkan Korkmaz

See Also

Other price charts: `chart()`, `kline()`, `ohlc()`

Examples

```
# script start;

# Charting BTC using
# line charts with closing price
# as main chart
cryptoQuotes::chart(
  ticker = BTC,
  main   = cryptoQuotes::pline(),
  sub    = list(
    cryptoQuotes::volume()
  )
)

# script end;
```

`remove_bound` *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

```
remove_bound(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 utility: `calibrate_window()`, `split_window()`

Examples

```
# script start;

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

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

  # 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 <- cryptoQuotes::calibrate_window(
  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
stopifnot(
  setequal(
    zoo::index(BTC),
    zoo::index(FGIndex)
  )
)
```

```
# script end;
```

`rsi` *Chart the Relative Strength Index (RSI)*

Description

[Experimental]

A high-level `plotly::plot_ly()`- and `plotly::add_lines()`-function that interacts with the `TTR::RSI()`-function. The function adds a subchart with a `TTR::RSI()`-indicator.

Usage

```
rsi(
 price       = "close",
 n           = 14,
 maType      = "SMA",
 upper_limit = 80,
 lower_limit = 20,
 color       = '#4682b4',
 ...
)
```

Arguments

`price`	Price series that is coercible to xts or matrix.
`n`	Number of periods for moving averages.
`maType`	Either: 1. A function or a string naming the function to be called. 2. A *list* with the first component like (1) above, and additional parameters specified as *named* components. See Examples.
`upper_limit`	A numeric-vector of length 1. 80 by default. Sets the upper limit of the TTR::RSI.
`lower_limit`	A numeric-vector of length 1. 20 by default. Sets the lower limit of the TTR::RSI.
`color`	A character-vector of length 1. "#4682b4" by default.
`...`	For internal use. Please ignore.

Value

An invisible `plotly::plot_ly()`-object.

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), donchian_channel(), ema(), evwma(), fgi(), hma(), lsr(), macd(), sma(), smi(), volume(), vwap(), wma(), zlema()

Other subchart indicators: add_event(), fgi(), lsr(), macd(), smi(), volume()

Other momentum indicators: macd(), smi()

Examples

```
# script start;

# 1) charting weekly
# BTC using candlesticks
# and indicators
cryptoQuotes::chart(
  ticker     = BTC,
  main       = cryptoQuotes::kline(),
  sub        = list(
    cryptoQuotes::volume(),
    cryptoQuotes::macd()
  ),
  indicator = list(
    cryptoQuotes::bollinger_bands(),
    cryptoQuotes::sma(),
    cryptoQuotes::alma()
  ),
  options = list(
    dark       = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

---

sma *Add Simple Moving Average (SMA) indicators to the chart*

Description

[Experimental]

A high-level plotly::add_lines()-wrapper function that interacts with TTR's moving average family of functions. The function adds moving average indicators to the main chart().

Usage

```
sma(
 price  = "close",
 n      = 10,
 ...
)
```

Arguments

price	A character-vector of length 1. "close" by default. The name of the vector to passed into TTR::SMA.
n	Number of periods to average over. Must be between 1 and nrow(x), inclusive.
...	For internal use. Please ignore.

Value

A plotly::plot_ly()-object

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), donchian_channel(), ema(), evwma(), fgi(), hma(), lsr(), macd(), rsi(), smi(), volume(), vwap(), wma(), zlema()

Other moving average indicators: alma(), dema(), ema(), evwma(), hma(), vwap(), wma(), zlema()

Other main chart indicators: add_event(), alma(), bollinger_bands(), dema(), donchian_channel(), ema(), evwma(), hma(), vwap(), wma(), zlema()

Examples

```
# script start;

cryptoQuotes::chart(
  ticker = BTC,
  main   = kline(),
  indicator = list(
    cryptoQuotes::ema(n = 7),
    cryptoQuotes::sma(n = 14),
    cryptoQuotes::wma(n = 21)
  )
)

# script end;
```

---

smi *Chart the Stochastic Momentum Index (SMI)*

Description

[Experimental]

A high-level plotly::plot_ly()- and plotly::add_lines()-function that interacts with the TTR::SMI()-function. The function adds a subchart with a TTR::SMI()-indicator.

Usage

```
smi(
 nFastK = 14,
 nFastD = 3,
 nSlowD = 3,
 maType,
 bounded = TRUE,
 smooth = 1,
 upper_limit = 40,
 lower_limit = -40,
 color = "#4682b4",
 ...
)
```

Arguments

nFastK	Number of periods for fast %K (i.e. the number of past periods to use).
nFastD	Number of periods for fast %D (i.e. the number smoothing periods to apply to fast %K).
nSlowD	Number of periods for slow %D (i.e. the number smoothing periods to apply to fast %D).
maType	Either: 1. A function or a string naming the function to be called. 2. A *list* with the first component like (1) above, and additional parameters specified as *named* components. See Examples.
bounded	Logical, should current period's values be used in the calculation?
smooth	Number of internal smoothing periods to be applied before calculating FastK. See Details.
upper_limit	A numeric-vector of length 1. 40 by default. Sets the upper limit of the TTR::SMI.
lower_limit	A numeric-vector of length 1. -40 by default. Sets the lower limit of the TTR::SMI.
color	A character-vector of length 1. "#4682b4" by default.
...	For internal use. Please ignore.

Value

An invisible `plotly::plot_ly()`-object.

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), donchian_channel(), ema(), evwma(), fgi(), hma(), lsr(), macd(), rsi(), sma(), volume(), vwap(), wma(), zlema()

Other subchart indicators: add_event(), fgi(), lsr(), macd(), rsi(), volume()

Other momentum indicators: macd(), rsi()

Examples

```
# script start;

# 1) charting weekly
# BTC using candlesticks
# and indicators
cryptoQuotes::chart(
  ticker     = BTC,
  main       = cryptoQuotes::kline(),
  sub        = list(
    cryptoQuotes::volume(),
    cryptoQuotes::macd()
  ),
  indicator = list(
    cryptoQuotes::bollinger_bands(),
    cryptoQuotes::sma(),
    cryptoQuotes::alma()
  ),
  options = list(
    dark       = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

---

split_window *split xts object iteratively in lists of desired intervals*

Description

[Experimental]

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

Usage

```
split_window(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 utility: calibrate_window(), remove_bound()

Examples

```
# script start;

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

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

  # 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 <- cryptoQuotes::calibrate_window(
  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
stopifnot(
  setequal(
    zoo::index(BTC),
    zoo::index(FGIndex)
  )
)


# script end;
```

`volume` *Chart the trading volume*

Description

[Experimental]

A high-level `plotly::plot_ly()`-function. The function adds a subchart with the trading `trading`.

Usage

```
volume(...)
```

Arguments

`...`	For internal use. Please ignore.

Value

An invisible `plotly::plot_ly()`-object.

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: `add_event()`, `alma()`, `bollinger_bands()`, `chart()`, `dema()`, `donchian_channel()`, `ema()`, `evwma()`, `fgi()`, `hma()`, `lsr()`, `macd()`, `rsi()`, `sma()`, `smi()`, `vwap()`, `wma()`, `zlema()`

Other subchart indicators: `add_event()`, `fgi()`, `lsr()`, `macd()`, `rsi()`, `smi()`

Examples

```
# script start;

# 1) charting weekly
# BTC using candlesticks
# and indicators
cryptoQuotes::chart(
  ticker     = BTC,
  main       = cryptoQuotes::kline(),
  sub        = list(
    cryptoQuotes::volume(),
    cryptoQuotes::macd()
  ),
  indicator = list(
    cryptoQuotes::bollinger_bands(),
    cryptoQuotes::sma(),
    cryptoQuotes::alma()
  ),
  options = list(
    dark       = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

---

vwap *Add Volume-Weighted Moving Average (VWAP) to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions. The function adds moving average indicators to the main `chart()`.

Usage

```
vwap(
 price  = "close",
 n      = 10,
 ratio = NULL,
 ...
)
```

Arguments

price	A character-vector of length 1. "close" by default. The name of the vector to passed into TTR::VWAP

n	Number of periods to average over. Must be between 1 and nrow(x), inclusive.
ratio	A smoothing/decay ratio. ratio overrides wilder in EMA.
...	For internal use. Please ignore.

Value

A plotly::plot_ly()-object

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), donchian_channel(), ema(), evwma(), fgi(), hma(), lsr(), macd(), rsi(), sma(), smi(), volume(), wma(), zlema()

Other moving average indicators: alma(), dema(), ema(), evwma(), hma(), sma(), wma(), zlema()

Other main chart indicators: add_event(), alma(), bollinger_bands(), dema(), donchian_channel(), ema(), evwma(), hma(), sma(), wma(), zlema()

Examples

```
# script start;

cryptoQuotes::chart(
  ticker = BTC,
  main   = kline(),
  indicator = list(
    cryptoQuotes::ema(n = 7),
    cryptoQuotes::sma(n = 14),
    cryptoQuotes::wma(n = 21)
  )
)

# script end;
```

---

wma *Add Weighted Moving Average (WMA) to the chart*

Description

[Experimental]

A high-level plotly::add_lines()-wrapper function that interacts with TTR's moving average family of functions. The function adds moving average indicators to the main chart().

Usage

```
wma(
 price  = "close",
 n      = 10,
 wts    = 1:n,
 ...
)
```

Arguments

price	A character-vector of length 1. "close" by default. The name of the vector to passed into TTR::WMA.
n	Number of periods to average over. Must be between 1 and nrow(x), inclusive.
wts	Vector of weights. Length of wts vector must equal the length of x, or n (the default).
...	For internal use. Please ignore.

Value

A plotly::plot_ly()-object

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: add_event(), alma(), bollinger_bands(), chart(), dema(), donchian_channel(), ema(), evwma(), fgi(), hma(), lsr(), macd(), rsi(), sma(), smi(), volume(), vwap(), zlema()

Other moving average indicators: alma(), dema(), ema(), evwma(), hma(), sma(), vwap(), zlema()

Other main chart indicators: add_event(), alma(), bollinger_bands(), dema(), donchian_channel(), ema(), evwma(), hma(), sma(), vwap(), zlema()

Examples

```
# script start;

cryptoQuotes::chart(
  ticker = BTC,
  main   = kline(),
  indicator = list(
    cryptoQuotes::ema(n = 7),
    cryptoQuotes::sma(n = 14),
    cryptoQuotes::wma(n = 21)
  )
)

# script end;
```

`zlema` *Add Zero Lag Exponential Moving Average (ZLEMA) to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions. The function adds moving average indicators to the main `chart()`.

Usage

```
zlema(
 price  = "close",
 n      = 10,
 ratio = NULL,
 ...
)
```

Arguments

`price`	A character-vector of length 1. "close" by default. The name of the vector to passed into TTR::ZLEMA.
`n`	Number of periods to average over. Must be between 1 and `nrow(x)`, inclusive.
`ratio`	A smoothing/decay ratio. `ratio` overrides `wilder` in `EMA`.
`...`	For internal use. Please ignore.

Value

A `plotly::plot_ly()`-object

Author(s)

Serkan Korkmaz

See Also

Other chart indicators: `add_event()`, `alma()`, `bollinger_bands()`, `chart()`, `dema()`, `donchian_channel()`, `ema()`, `evwma()`, `fgi()`, `hma()`, `lsr()`, `macd()`, `rsi()`, `sma()`, `smi()`, `volume()`, `vwap()`, `wma()`

Other moving average indicators: `alma()`, `dema()`, `ema()`, `evwma()`, `hma()`, `sma()`, `vwap()`, `wma()`

Other main chart indicators: `add_event()`, `alma()`, `bollinger_bands()`, `dema()`, `donchian_channel()`, `ema()`, `evwma()`, `hma()`, `sma()`, `vwap()`, `wma()`

Examples

```
# script start;

cryptoQuotes::chart(
  ticker = BTC,
  main   = kline(),
  indicator = list(
    cryptoQuotes::ema(n = 7),
    cryptoQuotes::sma(n = 14),
    cryptoQuotes::wma(n = 21)
  )
)

# script end;
```

Index