Package ‘cryptoQuotes’

March 12, 2024

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

Version 1.3.0

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), conflicted (>= 1.2.0), curl (>= 5.2.0), jsonlite (>= 1.8.8), lifecycle (>= 1.0.4), paletteer (>= 1.6.0), plotly (>= 4.10.4), rlang (>= 1.1.3), 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 2024-03-12 15:50:10 UTC

R topics documented:

`add_event` *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

```
add_event(event_data, ...)
```

Arguments

`event_data`	a data.frame with index, event and colors.
`...`	arguments pass interally by chart, ignore.

Details

TBA

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 `add_event()`.

See Also

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

Other subcharts: `fgi()`, `lsr()`, `macd()`, `rsi()`, `volume()`

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 <- ATOM[
  sample(1:nrow(ATOM), 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(
  ticker     = ATOM,
  main       = kline(),
  sub        = list(
    volume()
  ),
  indicator = list(
    bollinger_bands()
  ),
  event_data = event_data,
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

`alma` *Add Arnaud Legoux Moving Average to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions.

Usage

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

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.
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	any other passthrough parameters

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

See Also

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

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

`ATOM` *USDT denominated ATOMS with 15m intervals*

Description

A xts object with 15m OHLCV of USDT denominated ATOM with 97 rows and 5 columns, from 2023-12-30 to 2023-12-31.

Usage

```
ATOM
```

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: `BTC`, `DOGE`, `FGIndex`

`availableExchanges` *Get available exchanges*

Description

[Deprecated]

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

Usage

```
availableExchanges(type = "ohlc")
```

Arguments

`type`	character-vector of length 1. See details

Details

Available types:

- ohlc: Open, High, Low, Close and Volume
- lsratio: Long-Short ratio
- fundingrate: Funding rates
- interest: Open perpetual contracts on both sides

Limits:

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 deprecated: `availableIntervals()`, `availableTickers()`, `getFGIndex()`, `getLSRatio()`, `getQuote()`

Examples

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

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

# script end;
```

`availableIntervals` *Get available intervals*

Description

[Deprecated]

Usage

```
availableIntervals(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. See details
`futures`	A logical-vector of length 1. TRUE by default. Returns futures market if TRUE, spot market otherwise.

Details

Available types:

- ohlc: Open, High, Low, Close and Volume
- lsratio: Long-Short ratio
- fundingrate: Funding rates
- interest: Open perpetual contracts on both sides

Limits:

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

Author(s)

Serkan Korkmaz

See Also

Other deprecated: `availableExchanges()`, `availableTickers()`, `getFGIndex()`, `getLSRatio()`, `getQuote()`

Examples

```
## Not run: 
  # script:
  # date: 2023-10-06
  # author: Serkan Korkmaz, serkor1@duck.com
  # objective:
  # script start;

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

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

  # script end;

## End(Not run)
```

`availableTickers` *Get available cryptocurrency pairs*

Description

[Deprecated]

This function returns all available cryptocurrewncy pairs on the available_exchanges

Usage

```
availableTickers(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.

Author(s)

Serkan Korkmaz

See Also

Other deprecated: `availableExchanges`(), `availableIntervals`(), `getFGIndex`(), `getLSRatio`(), `getQuote`()

Examples

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

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

## End(Not run)
```

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

Arguments

`type`	character-vector of length 1. See details

Details

Available types:

- ohlc: Open, High, Low, Close and Volume
- lsratio: Long-Short ratio
- fundingrate: Funding rates
- interest: Open perpetual contracts on both sides

Limits:

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

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

# 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. See details
`futures`	A logical-vector of length 1. TRUE by default. Returns futures market if TRUE, spot market otherwise.

Details

Available types:

- ohlc: Open, High, Low, Close and Volume
- lsratio: Long-Short ratio
- fundingrate: Funding rates
- interest: Open perpetual contracts on both sides

Limits:

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

Author(s)

Serkan Korkmaz

See Also

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

Examples

```
## Not run:
  # script:
  # date: 2023-10-06
  # author: Serkan Korkmaz, serkor1@duck.com
  # objective:
  # script start;

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

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

  # script end;

## End(Not run)
```

`available_tickers` *Get available cryptocurrency pairs*

Description

[Stable]

Get available cryptocurrency pairs

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.

Author(s)

Serkan Korkmaz

See Also

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

Examples

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

  ## 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]

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

```
bollinger_bands(n = 20, sd = 2, maType = "SMA", internal = list(), ...)
```

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.
internal	An empty list. Used for internal purposes. Ignore.
...	Other arguments to be passed to the maType function.

Value

Invisbly returns a plotly object.

See Also

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

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

Description

A xts object with weekly OHLCV of USDT denominated Bitcoin with 52 rows and 5 columns, from 2023-01-01 to 2023-12-31.

Usage

```
BTC
```

Format

An object of class `xts` (inherits from `zoo`) with 52 rows and 5 columns.

Details

Open Opening price

High Highest price

Low Lowest price

Close Closing price

Volume Volume

See Also

Other data: `ATOM`, `DOGE`, `FGIndex`

`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 convinience: `remove_bound()`, `split_window()`

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(BTC),
  zoo::index(FGIndex)
)

# 2) to align the indices,
# we use the convincience functions
# by splitting the FGI by the BTC index.
FGIndex <- split_window(
  xts = 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 <- 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
setequal(
  zoo::index(BTC),
  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 *Build interactive financial charts*

Description

[Experimental]

`chart()` creates interactive financial charts using `plotly::plot_ly()` as backend. It's a high-level function which collects and structures the passed chart elements.

Usage

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

Arguments

`ticker`	A `xts::xts()`-object with Open, High, Low, Close and Volume columns.
`main`	A `plotly::plot_ly()`-object wrapped in `rlang::expr()`. `kline()` by default.
`sub`	An optional list of `plotly::plot_ly()`-object(s) wrapped in `rlang::expr()`.
`indicator`	An optional list of `plotly::plot_ly()`-object(s) wrapped in `rlang::expr()`.
`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.

Value

Returns a `plotly::plot_ly()` object.

Author(s)

Serkan Korkmaz

See Also

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

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

---

dema *Add Double Exponential Moving Average to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions.

Usage

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

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`.
`internal`	An empty list. Used for internal purposes. Ignore.

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

See Also

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

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

DOGE *USDT denominated DOGECOIN in 1m intervals*

Description

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

Usage

```
DOGE
```

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: `ATOM`, `BTC`, `FGIndex`

`ema` *Add Exponentially Weighted Moving Average to the charts*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions.

Usage

```
ema(
  price = "Close",
  n = 10,
  wilder = FALSE,
  ratio = NULL,
  internal = list(),
  ...
)
```

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`.
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	any other passthrough parameters

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

See Also

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

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

`evwma` *Add Elastic Volume-weighted Moving Average to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions.

Usage

```
evwma(price = "Close", n = 10, internal = list(), ...)
```

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.
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	any other passthrough parameters

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

See Also

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

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

fgi — *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

```
fgi(index, internal = list())
```

Arguments

`index`	The Fear and Greed Index created by `getFGIndex()`
`internal`	An empty list. Used for internal purposes. Ignore.

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

Other sentiment indicators: `lsr()`

Other subcharts: `add_event()`, `lsr()`, `macd()`, `rsi()`, `volume()`

Examples

```
## Not run:
  # 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
  # for the last 7 days
  tail(
    fgi <- cryptoQuotes::get_fgindex(
      from = Sys.Date() - 7
    )
  )

  # script end;

## End(Not run)
```

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 2023-01-01 to 2023-12-31

Usage

```
FGIndex
```

Format

An object of class `xts` (inherits from `zoo`) with 364 rows and 1 columns.

Details

FGI Daily Fear and Greed Index Value

See Also

Other data: `ATOM`, `BTC`, `DOGE`

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

Description

[Deprecated]

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 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,

- fgi (numeric): The daily fear and greed index value

Note

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

Author(s)

Serkan Korkmaz

See Also

Other deprecated: `availableExchanges()`, `availableIntervals()`, `availableTickers()`, `getLSRatio()`, `getQuote()`

Examples

```
## Not run:
  # 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
  # for the last 7 days
  tail(
    fgi <- cryptoQuotes::get_fgindex(
      from = Sys.Date() - 7
    )
  )

  # script end;

## End(Not run)
```

getLSRatio *Get the long to short ratio of a cryptocurrency pair*

Description

[Deprecated]

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

Usage

```
getLSRatio(
  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. See `available_intervals()` for available intervals.
`source`	A character-vector of length 1. See `available_exchanges()` for details.
`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.
`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,

- long (numeric) - the share of longs
- short (numeric) - the share of shorts
- ls_ratio (numeric) - the ratio of longs to shorts

Note

Available exchanges:

See `available_exchanges()` with for available exchanges.

Limited return values:

Binance only supports data for the last 30 days. Use other exchanges if you need beyond that.

Author(s)

Jonas Cuzulan Hirani

See Also

Other deprecated: `availableExchanges()`, `availableIntervals()`, `availableTickers()`, `getFGIndex()`, `getQuote()`

Examples

```
## Not run:
  # 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
  ls_ratio <- 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_ratio),
      cryptoQuotes::volume()
    ),
    indicator = list(
      cryptoQuotes::bollinger_bands()
    )
  )

## End(Not run)



# end of scrtipt;
```

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

Description

[Deprecated]

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

Usage

```
## get OHLC-V
getQuote(
 ticker,
 source   = 'binance',
 futures  = TRUE,
 interval = '1d',
 from     = NULL,
 to       = NULL
)
```

Arguments

`ticker`	An 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,

- open (numeric): the opening price
- close (numeric): the closing price
- high (numeric): the highest price
- low (numeric): the lowest price
- volume (numeric): the trading volume

Author(s)

Serkan Korkmaz

See Also

Other deprecated: availableExchanges(), availableIntervals(), availableTickers(), getFGIndex(), getLSRatio()

Examples

```
## Not run:
# script: scr_getQuote
# date: 2024-02-29
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Demonstrate the basic
# usage of the get_quote-function
# script start;


  # 1) Load BTC spot
  # from Kucoin with 30 minute
  # intervals
  BTC <- cryptoQuotes::get_quote(
    ticker   = 'BTCUSDT',
    source   = 'binance',
    interval = '30m',
    futures  = FALSE,
    from     = Sys.Date() - 1
  )

  # 2) chart the spot price
  # using the chart
  # function
  cryptoQuotes::chart(
    ticker = BTC,
    main   = cryptoQuotes::kline(),
    indicator = list(
      cryptoQuotes::volume(),
      cryptoQuotes::bollinger_bands()
    )
  )

# script end;

## End(Not run)
```

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

Description

[Deprecated]

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

```
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

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,

- fgi (numeric): The daily fear and greed index value

Note

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

Author(s)

Serkan Korkmaz

See Also

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

Examples

```
## Not run:
  # 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
  # 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`	An 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,

- funding_rate (numeric): the current funding rate

Author(s)

Serkan Korkmaz

See Also

Other get-function: `get_fgindex()`, `get_lsratio()`, `get_openinterest()`, `get_quote()`

Examples

```
## Not run:
# script: Funding Rate example
# date: 2024-03-01
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Fetch
# funding rate from one of the available
# exchanges
# script start;

# 1) check available
# exchanges for funding rates
available_exchanges(type = "fundingrate")

# 2) get BTC funding rate
# for the last 7 days
tail(
  BTC <- 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

```
## long-short ratio
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. See `available_intervals()` for available intervals.
`source`	A character-vector of length 1. See `available_exchanges()` for details.
`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.
`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,

- long (numeric) - the share of longs
- short (numeric) - the share of shorts
- ls_ratio (numeric) - the ratio of longs to shorts

Note

Available exchanges:
See `available_exchanges()` with for available exchanges.

Limited return values:
Binance only supports data for the last 30 days. Use other exchanges if you need beyond that.

Author(s)

Jonas Cuzulan Hirani

See Also

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

Examples

```
## Not run:
  # 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
  ls_ratio <- 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_ratio),
      cryptoQuotes::volume()
    ),
    indicator = list(
      cryptoQuotes::bollinger_bands()
    )
  )

## End(Not run)



# end of scrtipt;
```

`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

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

Arguments

`ticker`	An 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,

- open_interest (numeric): total open perpetual contracts on both both sides.

Note

Not all exchanges supports this endpoint, check `available_exchanges()` for details.

Author(s)

Serkan Korkmaz

See Also

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

Examples

```
## Not run:
  # script: Open Interest Example
  # date: 2024-03-03
  # author: Serkan Korkmaz, serkor1@duck.com
  # objective: Fetch
  # funding rate from one of the available
  # exchanges
  # script start;

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

  # 2) get BTC funding rate
  # for the last 7 days
  tail(
    BTC <- 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 OHLC-V
get_quote(
 ticker,
 source   = 'binance',
 futures  = TRUE,
 interval = '1d',
 from     = NULL,
 to       = NULL
)
```

Arguments

`ticker`	An 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,

- open (numeric): the opening price
- close (numeric): the closing price
- high (numeric): the highest price
- low (numeric): the lowest price
- volume (numeric): the trading volume

Author(s)

Serkan Korkmaz

See Also

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

Examples

```
## Not run:
# script: scr_getQuote
# date: 2024-02-29
# author: Serkan Korkmaz, serkor1@duck.com
# objective: Demonstrate the basic
# usage of the get_quote-function
# script start;


  # 1) Load BTC spot
  # from Kucoin with 30 minute
  # intervals
  BTC <- cryptoQuotes::get_quote(
    ticker   = 'BTCUSDT',
    source   = 'binance',
    interval = '30m',
    futures  = FALSE,
    from     = Sys.Date() - 1
  )
  # 2) chart the spot price
  # using the chart
  # function
  cryptoQuotes::chart(
    ticker = BTC,
    main   = cryptoQuotes::kline(),
    indicator = list(
      cryptoQuotes::volume(),
      cryptoQuotes::bollinger_bands()
    )
  )

# script end;

## End(Not run)
```

`hma` *Add Hull Moving Average to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions.

Usage

```
hma(price = "Close", n = 20, internal = list(), ...)
```

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.
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	any other passthrough parameters

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

See Also

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

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

`kline` *Candlestick chart*

Description

[Experimental]

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(internal = list())
```

Arguments

`internal`	An empty list. Used for internal purposes. Ignore.

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

Author(s)

Serkan Korkmaz

See Also

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

`lsr` *Chart the long-short ratios*

Description

[Experimental]

The `lsr()`-function adds a scatter plot as a subplot to the chart colored by ratio size.

Usage

```
lsr(ratio, internal = list())
```

Arguments

`ratio`	A `xts::xts()`-object with the column LSRatio. See `get_lsratio()` for more details.
`internal`	An empty `list()`. This is an internal helper-argument, ignore.

Details

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

Value

A plotly::plot_ly()-object wrapped in rlang::expr()

See Also

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

Other sentiment indicators: fgi()

Other subcharts: add_event(), fgi(), macd(), rsi(), volume()

Examples

```
## Not run:
  # 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
  ls_ratio <- 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_ratio),
      cryptoQuotes::volume()
    ),
    indicator = list(
      cryptoQuotes::bollinger_bands()
    )
  )

## End(Not run)



# end of scrtipt;
```

---

macd *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

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

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.
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	Other arguments to be passed to the `maType` function in case (1) above.

Value

Invisbly returns a plotly object.

See Also

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

Other subcharts: add_event(), fgi(), lsr(), rsi(), volume()

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

ohlc — *OHLC chart*

Description

[Experimental]

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

```
ohlc(internal = list())
```

Arguments

`internal`	An empty list. Used for internal purposes. Ignore.

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

Author(s)

Serkan Korkmaz

See Also

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# 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 convinience: `calibrate_window()`, `split_window()`

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(BTC),
  zoo::index(FGIndex)
)

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

rsi — *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

```
rsi(
    n = 14,
    maType = "SMA",
    upper_limit = 80,
    lower_limit = 20,
    internal = list(),
    ...
  )
```

Arguments

`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.
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	Other arguments to be passed to the `maType` function in case (1) above.

Value

Invisbly returns a plotly object.

See Also

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

Other subcharts: `add_event()`, `fgi()`, `lsr()`, `macd()`, `volume()`

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

sma *Add Simple Moving Averages to the charts*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions.

Usage

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

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.
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	any other passthrough parameters

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

See Also

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

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  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 convinience: `calibrate_window()`, `remove_bound()`

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(BTC),
  zoo::index(FGIndex)
)

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

`volume` *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

```
volume(internal = list())
```

Arguments

`internal`	An empty list. Used for internal purposes. Ignore.

Value

Invisbly returns a plotly object.

See Also

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

Other subcharts: `add_event()`, `fgi()`, `lsr()`, `macd()`, `rsi()`

Examples

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

## charting the klines
## with indicators as
## subcharts
```

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

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

`vwap` *Add Volume-weighted Moving Average to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions.

Usage

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

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`.
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	any other passthrough parameters

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

See Also

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main      = macd(),
  sub       = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

wma — *Add Weighted Moving Average to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions.

Usage

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

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).
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	any other passthrough parameters

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

See Also

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

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

zlema — *Add Zero Lag Exponential Moving Average to the chart*

Description

[Experimental]

A high-level `plotly::add_lines()`-wrapper function that interacts with TTR's moving average family of functions.

Usage

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

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`.
`internal`	An empty list. Used for internal purposes. Ignore.
`...`	any other passthrough parameters

Value

A `plotly::plot_ly()`-object wrapped in `rlang::expr()`.

See Also

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

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

Examples

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

## charting the klines
## with indicators as
## subcharts
chart(
  ticker     = BTC,
  main       = kline(),
  sub        = list(
    volume(),
    macd()
  ),
  indicator = list(
    bollinger_bands(),
    sma(),
    alma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

## charting the MACD-indicator
## with klines as subcharts
chart(
  ticker     = BTC,
  main       = macd(),
  sub        = list(
    volume(),
    kline()
  ),
  indicator = list(
    bollinger_bands(),
    sma()
  ),
  options = list(
    dark = TRUE,
    deficiency = FALSE
  )
)

# script end;
```

Index