Package 'binancer'

November 3, 2023

Type Package

Title Wrapper for Binance REST API in R

Version 0.1.0

Author Beniamino Sartini

Maintainer Cryptoverser <cryptoverser-project@gmail.com>

Description Provides a comprehensive R interface for the Binance REST API.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)
ggplot2 (>= 3.4.0),

Imports cli (>= 3.4.0),
dplyr (>= 1.1.3),
ggplot2 (>= 3.4.0),
httr (>= 1.4.4),
jsonlite (>= 1.8.4),
lubridate (>= 1.9.2),
purrr (>= 1.0.2),
readr (>= 2.1.2),
stringr (>= 1.5.0),
R6 (>= 2.5.1),
digest (>= 0.6.27),
utils,
stats,
websocket (>= 1.4.1)

Suggests DT,
knitr,
rmarkdown,
xts (>= 0.12.2),
testthat (>= 2.1.0)

RoxygenNote 7.1.1

Roxygen list(markdown = TRUE)

URL http://www.cryptoverser.org

BugReports https://github.com/cryptoverser-project/binancer/issues

VignetteBuilder knitr

R topics documented:

`binanceWebSocket` *R6 Class for Binance Web Socket*

Description

Binance web socket description

Details

Binance web socket details

Active bindings

`stream` Get all streams

`info` Get all streams info

Methods

Public methods:

Method `new()`**:** Initialize a `binanceWebSocket` object.

Usage:

```
binanceWebSocket$new(pair = "BTCUSDT", subscription, interval, update_speed)
```

Arguments:

`pair` pair

`subscription` subscription

`interval` 1m

`update_speed` 1000

Returns: A new `binanceWebSocket` object.

Method `add_depth_snapshot()`**:** Add Depth

Usage:

```
binanceWebSocket$add_depth_snapshot(pair, subscription, update_speed)
```

Arguments:

`pair` pair

`subscription` subscription

`update_speed` 1m

Method `add_trades_snapshot()`**:** Add trades

Usage:

```
binanceWebSocket$add_trades_snapshot(pair, subscription, from, to)
```

Arguments:

`pair` pair

`subscription` subscription

`from` 1000

`to` to

Method `add_klines_snapshot()`**:** Add Klines

Usage:

```
binanceWebSocket$add_klines_snapshot(pair, subscription, interval, from, to)
```

Arguments:

pair pair
subscription subscription
interval 1m
from 1000
to to

Method `subscribe()`**:** Subscribe to a stream

Usage:

```
binanceWebSocket$subscribe(pair, subscription, interval, update_speed)
```

Arguments:

pair pair
subscription subscription
interval 1m
update_speed 1m

Method `unsubscribe()`**:** Unsubscribe from a stream

Usage:

```
binanceWebSocket$unsubscribe(pair, subscription, interval, update_speed)
```

Arguments:

pair pair
subscription subscription
interval 1m
update_speed 1m

Method `connect()`**:** Start the connection

Usage:

```
binanceWebSocket$connect()
```

Method `close()`**:** Close all connection

Usage:

```
binanceWebSocket$close()
```

Method `get_stream()`**:** Unsubscribe from a stream

Usage:

```
binanceWebSocket$get_stream(pair, subscription, interval, update_speed, id)
```

Arguments:

pair pair
subscription subscription
interval 1m
update_speed 1m

Method `get_info()`**:** Unsubscribe from a stream

Usage:

```
binanceWebSocket$get_info(pair, subscription, interval, update_speed, id)
```

Arguments:

pair pair

`subscription` subscription
`interval` 1m
`update_speed` 1m

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
binanceWebSocket$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`binance_account_balance` *Binance Account Balances*

Description

Get current balances of the connected account.

Usage

```
binance_account_balance()
```

Value

A `tibble`.

`binance_account_info` *Binance Account Information*

Description

Get current general Binance account information.

Usage

```
binance_account_info()
```

Value

A `tibble`.

binance_account_trades *Binance Account Trades*

Description

Get trades for a specific account and symbol.

Usage

```
binance_account_trades(pair,
                       from,
                       to,
                       from_id,
                       limit,
                       quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT".
from	Character or POSIXt object. Start time for historical trades. If it is missing, the default, retrieved the last 1000 trades from to date.
to	Character or POSIXt object. End time for historical trades. If it is missing, the default, will be used as end date Sys.time().
from_id	Numeric. The last id from which retrieve the trades. Default is missing.
limit	Integer. The maximum number of trades to retrieve. If missing, the default, will be used the maximum value that is 1000.
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

A tibble.

binance_avg_price *Binance Average Price*

Description

Get the average price of a trading pair in the last 5 minutes.

Usage

```
binance_avg_price(pair, quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT".
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Details

The IP weight for this API call is 2. This function implements the endpoint api/v3/avgPrice of spot API.

Value

Numeric. Average price in last 5 minutes.

Examples

```
# Average price for BTCUSDT
binance_avg_price("BTCUSDT")
# Average price for BNBUSDT
binance_avg_price("BNBUSDT")
```

---

binance_book_ticker *Binance Book Ticker*

Description

Get best ASK and BID price and quantity on the order book.

Usage

```
binance_book_ticker(pair,
                    api,
                    quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT". Multiple pairs are allowed only if api is "spot". If missing, the default, will be returned the book ticker for all the trading pairs.
api	Character. Reference API. If it is missing, the default, will be used "spot". Available options are: • "spot": for endpoint api/v3/ticker/bookTicker. The ip weight is 2 if a symbol is submitted, otherwise is 4. The ip weight depends on the number of pairs requested. The maximum ip weight is 80. • "fapi": for endpoint fapi/v1/ticker/bookTicker. The ip weight is 2 if a symbol is submitted, otherwise is 5. • "dapi": for endpoint dapi/v1/ticker/bookTicker. The ip weight is 2 if a symbol is submitted, otherwise is 5.
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

A data.frame with 8 columns:

- date: POSIXt, time of the snapshot.
- pair: Character, reference trading pair, present only if api is "dapi".
- symbol: Character, trading pair.
- market: Character, reference API.
- ask: Numeric, best ASK price.
- bid: Numeric, best BID price.
- ask_quantity: Numeric, quantity at best ASK price.
- bid_quantity: Numeric, quantity at best BID price.

Examples

```
# Get book ticker for all pairs
binance_book_ticker(api = "spot")
binance_book_ticker(api = "fapi")
binance_book_ticker(api = "dapi")

# Get book ticker for BTCUSDT
binance_book_ticker(pair = "BTCUSDT", api = "spot")
```

---

binance_cancel_order *Binance Cancel Order*

Description

Cancel an active order.

Usage

```
binance_cancel_order(pair,
                     order_id,
                     client_order_id)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT".
order_id	Numeric. Order id that uniquely identify the trade.
client_order_id	Numeric. Client order id that uniquely identify the trade.

binance_credentials *Set Binance API Keys*

Description

Sets the API key and secret to interact with the Binance API

Usage

```
binance_credentials(key, secret)
```

Arguments

key	Character. Api key.
secret	Character. Api secret.

Value

No return values, setting config in the package namespace.

Examples

```
## Not run:
# Add api keys
binance_credentials('foo', 'bar')
# Remove api keys
binance_credentials()

## End(Not run)
```

binance_depth *Binance Order Book Snapshot*

Description

Retrieve a snapshot of the order book for a trading pair.

Usage

```
binance_depth(pair,
              api,
              quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT".
api	Character. Reference API. If it is missing, the default, will be used "spot". Available options are: • "spot": for endpoint api/v3/depth. The ip weight is 250. • "fapi": for endpoint fapi/v1/depth. The ip weight is 20. • "dapi": for endpoint dapi/v1/depth. The ip weight is 20. • "eapi": for endpoint eapi/v1/depth. The ip weight is 1.
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

A tibble with 7 columns:

- last_update_id: Integer, id of the last snapshot.
- date: POSIXt, time of the snapshot.
- market: Character, reference API.
- pair: Character, trading pair.
- side: Character, side of the limit orders in the book. Can be "ASK" or "BID".
- price: Numeric, price level.
- quantity: Numeric, quantity for each price level.

Examples

```
# Get the order book for BTCUSDT
binance_depth(pair = "BTCUSDT", api = "spot")
binance_depth(pair = "BTCUSDT", api = "fapi")

# Get the order book for BTCUSD_PERP
binance_depth(pair = "BTCUSD_PERP", api = "dapi")

# Get the order book for a put option on BTC
binance_depth(pair = "BTC-240628-30000-P", api = "eapi")
```

---

binance_exchange_info *Binance Market Information*

Description

Obtain market information and available trading pairs.

Usage

```
binance_exchange_info(api,
                      permissions,
                      pair,
                      quiet = FALSE)
```

Arguments

api	Character. Reference API. If it is missing, the default, will be used "spot". Available options are: • "spot": for endpoint api/v3/exchangeInfo. The ip weight is 20. • "fapi": for endpoint fapi/v1/exchangeInfo. The ip weight is 1. • "dapi": for endpoint dapi/v1/exchangeInfo. The ip weight is 1. • "eapi": for endpoint eapi/v1/exchangeInfo. The ip weight is 1.
permissions	Character, optional. Used only if api = "spot". Types of trading pairs to return. If "all", the default, all available types of pairs will be returned. Available options are: • "all": trading pairs in all markets. • "spot": trading pairs only in spot markets. • "margin": trading pairs only in margin markets. • "leveraged": trading pairs only in leveraged markets.
pair	Character, optional. Trading pair, e.g. "BTCUSDT". If NULL, the default, all available pairs will be returned.
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

A tibble.

Examples

```
# Get all pairs in all markets
binance_exchange_info(api = "spot", permissions = "all", pair = NULL)

# Get all pairs only in spot markets
binance_exchange_info(api = "spot", permissions = "spot", pair = NULL)

# Get all pairs only in margin markets
binance_exchange_info(api = "spot", permissions = "margin", pair = NULL)

# Get all pairs only in leveraged market
binance_exchange_info(api = "spot", permissions = "leveraged", pair = NULL)

# Get information only for BTCUSDT in all markets
binance_exchange_info(api = "spot", permissions = "all", pair = "BTCUSDT")

# Get information for multiple pairs in all markets
binance_exchange_info(api = "spot",
                      permissions = "all",
                      pair = c("BTCUSDT", "BNBUSDT"))

# Get information for multiple pairs only in margin and leveraged markets
binance_exchange_info(api = "spot",
                      permissions = c("margin", "leveraged"),
                      pair = c("BTCBUSD", "ETHBUSD"))

# Get all pairs in futures USD-m markets
binance_exchange_info(api = "fapi")
```

```
# Get all pairs in futures COIN-m markets
binance_exchange_info(api = "dapi")

# Get all pairs in options markets
binance_exchange_info(api = "eapi")
```

`binance_futures_stats` *Futures Statistics*

Description

Get the historical statitics for futures.

Usage

```
binance_futures_stats(pair,
                      api,
                      interval,
                      indicator,
                      from,
                      to,
                      quiet = FALSE)
```

Arguments

`pair`	Character. Trading pair, e.g. `"BTCUSDT"` or `"BTCUSD"`.
`api`	Character, reference API. Available options are: • `"fapi"`: for futures USD-m API. • `"dapi"`: for futures COIN-m API.
`interval`	Character. Default is `"1h"`. Time interval for open interest data. Available intervals are: • Minutely: `"5m"`, `"15m"` and `"30m"`. • Hourly: `"1h"`, `"2h"`, `"4h"`, `"6h"`, `"8h"` and `"12h"`. • Daily: `"1d"`.
`indicator`	Character statistics indicator
`from`	Character or `POSIXt` object. Start time for historical data, only last 30 days are available. If it is `missing`, the default, will be used as start date `Sys.time()-lubridate::days(30)`.
`to`	Character or `POSIXt` object. End time for historical data, only last 30 days are available. If it is `missing`, the default, will be used as start date `Sys.time()`.
`quiet`	Logical. Default is `FALSE`. If `TRUE` suppress messages and warnings.

Details

The IP weight for this API call is 1, and the data source is memory. The historical open interest data are only available for the last 30 days.

Value

A `tibble`

Examples

```
# Statistics in USD-m market
binance_futures_stats(pair = "BTCUSDT",
                      api = "fapi",
                      interval = "1d",
                      indicator = "takerlongshortRatio",
                      from = Sys.Date()-2)
binance_futures_stats(pair = "BTCUSDT",
                      api = "fapi",
                      interval = "1d",
                      indicator = "globalLongShortAccountRatio",
                      from = Sys.Date()-2)
binance_futures_stats(pair = "BTCUSDT",
                      api = "fapi",
                      interval = "1d",
                      indicator = "topLongShortPositionRatio",
                      from = Sys.Date()-2)
binance_futures_stats(pair = "BTCUSDT",
                      api = "fapi",
                      interval = "1d",
                      indicator = "topLongShortAccountRatio",
                      from = Sys.Date()-2)

# Statistics in COIN-m market
binance_futures_stats(pair = "BTCUSD",
                      api = "dapi",
                      interval = "1d",
                      indicator = "takerBuySellVol",
                      from = Sys.Date()-2)
binance_futures_stats(pair = "BTCUSD",
                      api = "dapi",
                      interval = "1d",
                      indicator = "globalLongShortAccountRatio",
                      from = Sys.Date()-2)
binance_futures_stats(pair = "BTCUSD",
                      api = "dapi",
                      interval = "1d",
                      indicator = "topLongShortPositionRatio",
                      from = Sys.Date()-2)
binance_futures_stats(pair = "BTCUSD",
                      api = "dapi",
                      interval = "1d",
                      indicator = "topLongShortAccountRatio",
                      from = Sys.Date()-2)
```

---

binance_get_order *Binance Get Order*

Description

Get information about an order.

Usage

```
binance_get_order(pair,
                  order_id,
                  client_order_id)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT".
order_id	Numeric. Order id that uniquely identify the trade.
client_order_id	Numeric. Client order id that uniquely identify the trade.

---

binance_klines *Binance Klines*

Description

Get klines/candlestick data for a trading pair.

Usage

```
binance_klines(pair,
               api,
               interval,
               from,
               to,
               contract_type,
               uiKlines = FALSE,
               as_xts = FALSE,
               quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT".
api	Character. Reference API. If it is missing, the default, will be used "spot". Available options are: • "spot": for endpoint api/v3/klines. The ip weight is 2. • "fapi": for endpoint fapi/v1/klines. The ip weight is 10. • "dapi": for endpoint dapi/v1/klines. The ip weight is 10. • "eapi": for endpoint eapi/v1/klines. The ip weight is 1.
interval	Character. Default is "1d". Time interval for klines data. Available intervals are: • Secondly: "1s", available only if api = "spot". • Minutely: "1m", "3m", "5m", "15m" and "30m". • Hourly: "1h", "2h", "4h", "6h", "8h" and "12h". • Daily: "1d" and "3d". • Weekly: "1w". • Monthly: "1M".

from	Character or POSIXt object. Start time for historical data. If it is missing, the default, will be used as start date Sys.time()-lubridate::days(1).
to	Character or POSIXt object. End time for historical data. If it is missing, the default, will be used as end date Sys.time().
contract_type	Character. Used only if api is "fapi" or "dapi". Available contract's types are: • "perpetual": perpetual futures. • "current_quarter": futures with maturity in the current quarter. • "next_quarter": futures with maturity in the next quarter.
uiKlines	Logical. Default is FALSE. If TRUE return data in UIklines format.
as_xts	Logical. Default is FALSE. If TRUE convert the data into an xts object.
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

A tibble with 13 columns:

- date: POSIXt, the opening date of the candle.
- date_close: POSIXt, the closing date of the candle.
- market: Character, API.
- pair: Character, trading pair.
- open: Numeric, open price (price in date).
- high: Numeric, highest price from date up to date_close.
- low: Numeric, lowest price from date up to date_close.
- close: Numeric, close price or price in date_close.
- volume: Numeric, volume in asset value.
- volume_quote: Numeric, volume in quote asset value.
- trades: Numeric, number of trades from date up to date_close.
- taker_buy: Numeric, taker buy volume in asset value.
- taker_buy_quote: Numeric, taker buy volume in quote asset value.

Examples

```
# Get 1-hour OHLC data for BTCUSDT in the spot market
binance_klines(pair = "BTCUSDT", api = "spot", interval = "1h")

# Get 30-minute OHLC data for BTCUSDT in USD-m market
# Perpetual contracts
binance_klines(pair = "BTCUSDT", api = "fapi", interval = "30m", uiKlines = FALSE)
binance_klines(pair = "BTCUSDT", api = "fapi", interval = "30m",
               uiKlines = TRUE, contract_type = "perpetual")
# Futures contracts with maturity in current quarter
binance_klines(pair = "BTCUSDT", api = "fapi", interval = "30m",
               uiKlines = TRUE, contract_type = "current_quarter")
# Futures contracts with maturity in next quarter
binance_klines(pair = "BTCUSDT", api = "fapi", interval = "30m",
               uiKlines = TRUE, contract_type = "next_quarter")

# Get 15-minute OHLC data for BTCUSD in COIN-m market
# Perpetual contracts
```

```
binance_klines(pair = "BTCUSD_PERP", api = "dapi", interval = "15m", uiKlines = FALSE)
binance_klines(pair = "BTCUSD", api = "dapi", interval = "15m",
               uiKlines = TRUE, contract_type = "perpetual")
# Futures contracts with maturity in current quarter
binance_klines(pair = "BTCUSD", api = "dapi", interval = "15m",
               uiKlines = TRUE, contract_type = "current_quarter")
# Futures contracts with maturity in next quarter
binance_klines(pair = "BTCUSD", api = "dapi", interval = "1h",
               uiKlines = TRUE, contract_type = "next_quarter")

# Get 1-hour OHLC data for a put option on BTCUSDT.
# Strike of 30000 and maturity on 2024-06-28.
binance_klines(pair = "BTC-240628-30000-P", api = "eapi", interval = "1h")
```

`binance_last_trades` *Binance Last Trades*

Description

Get the last 1000 trades for a trading pair.

Usage

```
binance_last_trades(pair,
                    api,
                    quiet = FALSE)
```

Arguments

`pair`	Character. Trading pair, e.g. `"BTCUSDT"`.
`api`	Character. Reference API. If it is `missing`, the default, will be used `"spot"`. Available options are: • `"spot"`: for endpoint api/v3/trades. The ip weight is 10. • `"fapi"`: for endpoint fapi/v1/trades. The ip weight is 5. • `"dapi"`: for endpoint dapi/v1/trades. The ip weight is 5. • `"eapi"`: for endpoint eapi/v1/trades. The ip weight is 5.
`quiet`	Logical. Default is `FALSE`. If `TRUE` suppress messages and warnings.

Value

A `tibble` with 7 columns:

- `date`: `POSIXt`, trade execution date.
- `market`: Character, selected API.
- `pair`: Character, trading pair.
- `price`: Numeric, trade price.
- `quantity`: Numeric, trade quantity.
- `side`: Character, trade side. Can be `"BUY"` or `"SELL"`.
- `trade_id`: Integer, trade id.

Examples

```
# Get last 1000 trades for BTCUSDT
binance_last_trades(pair = "BTCUSDT", api = "spot")
binance_last_trades(pair = "BTCUSDT", api = "fapi")

# Get last 1000 trades for BTCUSD_PERP
binance_last_trades(pair = "BTCUSD_PERP", api = "dapi")

# Get last 1000 trades for a put option on BTC
binance_last_trades(pair = "BTC-240628-30000-P", api = "eapi")
```

---

binance_new_order *Create a Spot Order*

Description

Send in a new order in spot market.

Usage

```
binance_new_order(pair,
                  side,
                  type,
                  time_in_force,
                  quantity,
                  price,
                  stop_price,
                  trailing_delta,
                  iceberg_qty,
                  test = TRUE,
                  quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT".
side	Character. Side of the trade. Can be "BUY" or "SELL".
type	Character. Type of order. Available orders's types are: • "MARKET": A Market Order is an order to buy or sell immediately at the current market price. It ensures swift execution but may not guarantee the exact price you see at the moment of placing the order, especially during periods of high volatility. • "LIMIT" or "LIMIT_MAKER": A Limit order is an order to buy or sell at a specific price. It will only execute at the specified price or a more favorable one. This type of order allows traders to set a target price and wait for the market to reach it. • "STOP_LOSS" or "TAKE_PROFIT": A Stop Market Order is similar to the Stop Limit Order, but once the stop price is reached, it becomes a market order, and the trade is executed at the prevailing market price. This ensures execution but may not guarantee the exact price.

- `"STOP_LOSS_LIMIT"` or `"TAKE_PROFIT_LIMIT"`: A Stop Limit Order combines elements of a stop order and a limit order. You set a stop price and a limit price. When the stop price is triggered, it becomes a limit order, and it will only execute at or better than the limit price. This order type is useful for entering or exiting positions once a certain price level is reached.

`time_in_force` Character. Time in force, specify the conditions under which the trade expiry. The default `"GTC"`. More details can be found on Binance Academy. Available time in force are:

- `"GTC"`: **Good 'til canceled** orders stipulate that a trade should be kept open until it's either executed or manually canceled.
- `"IOC"`: **Immediate or cancel** orders stipulate that any part of the order that isn't immediately filled must be canceled.
- `"FOK"`: **Fill or kill** orders are either filled immediately, or they're canceled.

`quantity` Numeric. Quantity of the asset to be bought or sold. For example when `pair = "BTCUSDT"` and `quantity = 1`, if `side = "BUY"` we are sending an order to buy 1 BTC, otherwise if `side = "SELL"` we are sending an order to sell 1 BTC.

`price` Numeric, optional. Limit price, used only for limit orders.

`stop_price` Numeric, optional. Stop price, used only for stop loss and take profit orders. Can be specified a stop price or a trailing delta, if specified both will be used trailing delta by default.

`trailing_delta` Numeric, optional. Trailing delta, used only for stop loss and take profit orders. Can be specified a stop price or a trailing delta, if specified both will be used trailing delta by default.

`iceberg_qty` Numeric, iceberg quantity.

`test` Logical. If `TRUE`, the default, the order will be a test order.

`quiet` Logical. Default is `FALSE`. If `TRUE` suppress messages and warnings.

`binance_open_interest` *Binance Open Interest*

Description

Get the current open interest data for a trading pair.

Usage

```
binance_open_interest(pair,
                      api,
                      expiration,
                      quiet = FALSE)
```

Arguments

`pair` Character. Trading pair, e.g. `"BTCUSDT"`.

`api` Character. Reference API. If it is `missing`, the default, will be used `"fapi"`. Available options are:

- `"fapi"`: for endpoint fapi/v1/openInterest. The ip weight is 1.

- "dapi": for endpoint dapi/v1/openInterest. The ip weight is 1.
- "eapi": for endpoint eapi/v1/openInterest. The ip weight is 1.

expiration	Character or POSIXt object. Used only if api = "eapi". Expiration date for options contracts. If it is missing, the default, will be used Sys.time().
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

A tibble with 4 columns:

- date: POSIXt, observation date.
- market: Character, API.
- pair: Character, trading pair.
- open_interest: Numeric, open interest in base currency.

Examples

```
# Get the open interest data for BTCUSDT
binance_open_interest(pair = "BTCUSDT", api = "fapi")

# Get the open interest data for BTCUSD_PERP
binance_open_interest(pair = "BTCUSD_PERP", api = "dapi")

# Get the open interest data for options on BTC
binance_open_interest(pair = "BTC", api = "eapi", expiration = Sys.Date() + 1)
```

---

binance_open_interest_hist

Binance Historical Open Interest

Description

Get the historical open interest for a trading pair. The data are only available for the last 30 days.

Usage

```
binance_open_interest_hist(pair,
                           api,
                           interval,
                           from,
                           to,
                           contract_type,
                           quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT" or "BTCUSD".
api	Character. Reference API. Available options are: • "fapi": for endpoint fapi/v1/openInterestHist. The ip weight is 1. • "dapi": for endpoint dapi/v1/openInterestHist. The ip weight is 1.
interval	Character. Time interval for open interest data. Default is "1h". Available intervals are: • Minutely: "5m", "15m" and "30m". • Hourly: "1h", "2h", "4h", "6h", "8h" and "12h". • Daily: "1d".
from	Character or POSIXt object. Start time for historical data, only last 30 days are available. If it is missing, the default, will be used as start date Sys.time()-lubridate::days(30).
to	Character or POSIXt object. End time for historical data, only last 30 days are available. If it is missing, the default, will be used as start date Sys.time().
contract_type	Character. Used only if api = "dapi". Available contract's types are: • "all": for all types of contracts. • "perpetual": for perpetual futures. • "current_quarter": for futures with maturity in the current quarter. • "next_quarter": for futures with maturity in the next quarter.
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

A tibble with 5 columns:

- date: POSIXt, observation date.
- market: Character, API.
- pair: Character, trading pair.
- open_interest: Numeric, open interest in base currency.
- open_interest_usd: Numeric, open interest in USD.

Examples

```
# Historical open interest for BTCUSDT
binance_open_interest_hist(pair = "BTCUSDT",
                           api = "fapi",
                           interval = "1d",
                           from = Sys.time()-lubridate::days(15),
                           to = Sys.time()-lubridate::days(5))

# Historical open interest for BTCUSD
binance_open_interest_hist(pair = "BTCUSD",
                           api = "dapi",
                           interval = "1d",
                           from = Sys.time()-lubridate::days(15),
                           to = NULL)
```

binance_ping *Ping to Binance REST API*

Description

Check the connection to the Binance API.

Usage

```
binance_ping(api)
```

Arguments

api	Character. Reference API. If it is missing, the default, will be used "spot". Available options are: • "spot": for endpoint api/v3/ping. The ip weight is 1. • "fapi": for endpoint fapi/v1/ping. The ip weight is 1. • "dapi": for endpoint dapi/v1/ping. The ip weight is 1. • "eapi": for endpoint eapi/v1/ping. The ip weight is 1.

Value

A logical value. It is TRUE if the connection was successful, otherwise it is FALSE.

Examples

```
# Test connection to spot api
binance_ping("spot")

# Test connection to futures usd-m api
binance_ping("fapi")

# Test connection to futures coin-m api
binance_ping("dapi")

# Test connection to options api
binance_ping("eapi")
```

binance_query *Query to Binance REST API*

Description

Execute a query to Binance REST API.

Usage

```
binance_query(api,
              path = NULL,
              query = list(),
              method = 'GET',
              sign = FALSE,
              use_base_path = TRUE,
              quiet = FALSE)
```

Arguments

api	Character. Reference API. If it is missing, the default, will be used "spot". Available options are: • "spot": for spot API. • "fapi": for futures USD-m API. • "dapi": for futures Coin-m API. • "eapi": for options API. • "sapi": for wallet API.
path	Character vector. API path, NULL or NA elements will be excluded.
query	Named list. Query parameters for the API call, NULL or NA elements will be excluded.
method	Character. A method between "GET", "POST" and "DELETE".
sign	Logical. Default is FALSE. TRUE if signature is required.
use_base_path	Logical. When TRUE, the default, to path argument will be added a base_bath based on the selected API. Available base_bath are: • "spot": base path is "api/v3". • "fapi": base path is "fapi/v1". • "dapi": base path is "dapi/v1". • "eapi": base path is "eapi/v1". • "sapi": base path is "sapi/v1".
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

An R object parsed as character. See more on content

Examples

```
# Execute a call to spot API with base path
binance_query(api = "spot",
              path = "time",
              query = NULL,
              use_base_path = TRUE)

# Execute a call to spot API without base path
binance_query(api = "spot",
              path = c("api", "v3", "time"),
              query = NULL,
              use_base_path = FALSE)
```

binance_ticker24h *Binance 24-Hour Ticker Statistics*

Description

Get 24-hour ticker statistics for a specified trading pair from the selected reference API.

Usage

```
binance_ticker24h(pair,
                  api,
                  type,
                  quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT". Multiple pairs are allowed only if api is "spot". If missing, the default, will be returned the 24hr ticker for all the trading pairs.
api	Character. Reference API. If it is missing, the default, will be used "spot". Available options are: • "spot": for endpoint api/v3/ticker/24hr. The ip weight depends on the number of pairs requested. The maximum ip weight is 80. • "fapi": for endpoint fapi/v1/ticker/24hr. The ip weight is 1. • "dapi": for endpoint dapi/v1/ticker/24hr. The ip weight is 1. • "eapi": for endpoint eapi/v1/ticker. The ip weight is 5.
type	Character. Type of ticker data. Used only if api = "spot". Default is "full". Available options are: • "mini": data without ask and bid prices and quantities. • "full": complete ticker data.
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

A tibble with 13 columns containing 24-hour ticker statistics, including: open, high, low, close prices, volume, and more.

Examples

```
# Get full 24-hour ticker for all pairs
binance_ticker24h(api = "spot")
binance_ticker24h(api = "fapi")
binance_ticker24h(api = "dapi")
binance_ticker24h(api = "eapi")

# Get full 24-hour ticker for BTCUSDT
binance_ticker24h(pair = "BTCUSDT", api = "spot", type = "full")

# Get mini 24-hour ticker for BTCUSDT
binance_ticker24h(pair = "BTCUSDT", api = "spot", type = "mini")
```

```
# Get 24-hour ticker for BTCUSDT
binance_ticker24h(pair = "BTCUSDT", api = "fapi")

# Get 24-hour ticker for BTCUSD_PERP
binance_ticker24h(pair = "BTCUSD_PERP", api = "dapi")

# Get 24-hour ticker for a put option on BTCUSDT
binance_ticker24h(pair = "BTC-240628-30000-P", api = "eapi")
```

`binance_ticker_price` *Binance Book Ticker*

Description

Get last price for a trading pair.

Usage

```
binance_ticker_price(pair,
                     api,
                     quiet = FALSE)
```

Arguments

`pair`	Character. Trading pair, e.g. `"BTCUSDT"`. Multiple pairs are allowed only if `api` is `"spot"`. If `missing`, the default, will be returned the book ticker for all the trading pairs.
`api`	Character. Reference API. If it is `missing`, the default, will be used `"spot"`. Available options are: • `"spot"`: for endpoint api/v3/ticker/price. The ip weight is 2 if a symbol is submitted, otherwise is 4. The ip weight depends on the number of pairs requested. The maximum ip weight is 80. • `"fapi"`: for endpoint fapi/v1/ticker/price. The ip weight is 1 if a symbol is submitted, otherwise is 2. • `"dapi"`: for endpoint dapi/v1/ticker/price. The ip weight is 1if a symbol is submitted, otherwise is 2.
`quiet`	Logical. Default is `FALSE`. If `TRUE` suppress messages and warnings.

Value

A `tibble` with 8 columns:

- `date`: `POSIXt`, time of the snapshot.
- `pair`: Character, reference trading pair, present only if `api` is `"dapi"`.
- `symbol`: Character, trading pair.
- `market`: Character, reference API.
- `ask`: Numeric, best ASK price.
- `bid`: Numeric, best BID price.
- `ask_quantity`: Numeric, quantity at best ASK price.
- `bid_quantity`: Numeric, quantity at best BID price.

Examples

```
# Get book ticker for all pairs
binance_ticker_price(api = "spot")
binance_ticker_price(api = "fapi")
binance_ticker_price(api = "dapi")

# Get book ticker for BTCUSDT
binance_ticker_price(pair = "BTCUSDT", api = "spot")
```

`binance_time` *Binance Server Time*

Description

Get the current server time from Binance API.

Usage

```
binance_time(api, quiet = FALSE)
```

Arguments

`api`	Character. Reference API. If it is `missing`, the default, will be used `"spot"`. Available options are: • `"spot"`: for endpoint api/v3/time. The ip weight is 1. • `"fapi"`: for endpoint fapi/v3/time. The ip weight is 1. • `"dapi"`: for endpoint dapi/v3/time. The ip weight is 1. • `"eapi"`: for endpoint eapi/v3/time. The ip weight is 1.
`quiet`	Logical. Default is `FALSE`. If `TRUE` suppress messages and warnings.

Details

The IP weight for this API call is 1, and the data source is memory.

Value

A `POSIXt` object. The server time for the reference API.

Examples

```
# Get the server time
binance_time("spot")
binance_time("fapi")
binance_time("dapi")
binance_time("eapi")
```

binance_trades *Binance Aggregated Historical Trades*

Description

Get aggregated historical trades data for a trading pair.

Usage

```
binance_trades(pair,
               api,
               from,
               to,
               quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g. "BTCUSDT".
api	Character. Reference API. If it is missing, the default, will be used "spot". Available options are: • "spot": for endpoint api/v3/aggTrades. • "fapi": for endpoint fapi/v1/aggTrades. • "dapi": for endpoint dapi/v1/aggTrades. • "eapi": for endpoint eapi/v1/aggTrades.
from	Character or POSIXt object. Start time for historical data. If it is missing, the default, will be used as start date Sys.time()-lubridate::minutes(10).
to	Character or POSIXt object. End time for historical data. If it is missing, the default, will be used as end date Sys.time().
quiet	Logical. Default is FALSE. If TRUE suppress messages and warnings.

Value

A tibble with 9 columns:

- date: POSIXt, trade execution date;
- market: Character, API.
- pair: Character, trading pair.
- price: Numeric, trade price.
- quantity: Numeric, trade quantity.
- side: Character, trade side. Can be "BUY" or "SELL".
- agg_id: Integer, aggregated trade id.
- first_id: Integer, first trade id for aggregation.
- last_id: Integer, last trade id for aggregation.

Examples

```
# Get trades in last 10 minutes for BTCUSDT
binance_trades(pair = "BTCUSDT", api = "spot", from = NULL, to = NULL)

# Get trades in last 10 minutes for LAZIOUSDT
binance_trades(pair = "LAZIOUSDT", api = "spot", from = "2023-01-01", to = "2023-01-02")

# Get trades in last 10 minutes for BTCUSDT
binance_trades(pair = "BTCUSDT", api = "fapi", from = NULL, to = NULL)

# Get trades in last 10 minutes for BTCUSD_PERP
binance_trades(pair = "BTCUSD_PERP", api = "dapi", from = NULL, to = NULL)
```

---

binance_vision_klines *Historical Klines data from Binance Vision*

Description

Historical Kline data for a trading pair from the Binance Vision database.

Usage

```
binance_vision_klines(pair, api, interval, from, to, quiet = FALSE)
```

Arguments

pair	Character. Trading pair, e.g., "BTCUSDT".
api	Character. Reference API. If it is missing, the default, will be used "spot". Available options are: • "spot": for spot API. • "fapi": for futures USD-m API. • "dapi": for futures COIN-m API.
interval	Character. Default is "1d". Time interval for klines data. Available intervals are: • Secondly: "1s", available only if api = "spot". • Minutely: "1m", "3m", "5m", "15m" and "30m". • Hourly: "1h", "2h", "4h", "6h", "8h" and "12h". • Daily: "1d".
from	Character or an object of class "POSIXt", the start time for data retrieval. If NULL, the default is Sys.Date()-lubridate::days(4).
to	Character or an object of class "POSIXt", the end time for data retrieval. If NULL, the default is Sys.Date()-lubridate::days(2).
quiet	Logical, suppress informational messages if TRUE. Default FALSE.

Value

A tibble.

Examples

```
from <- "2023-01-01"
to <- "2023-01-01"
interval <- "1m"
binance_vision_klines("BTCUSDT", api = "spot",
                      interval = interval, from = from, to = from)
binance_vision_klines("BTCUSDT", api = "fapi",
                      interval = interval, from = from, to = from)
binance_vision_klines("BTCUSD_PERP", api = "dapi",
                      interval = interval, from = from, to = from)
```

`candleChart` *Candlestick Plot*

Description

Create a candlestick chart to visualize price movements over a specified time frame.

Usage

```
candleChart(
  data,
  from = NULL,
  to = NULL,
  title = NULL,
  col_body_up = "green",
  col_body_dw = "red",
  col_wick = "black"
)
```

Arguments

`data`	A data frame containing time-series data with columns: 'date', 'open', 'close', 'low', 'high', 'pair', and 'interval'.
`from`	Character or an object of class `"POSIXt"`, the start date for the plot. If specified, only data from this date on wards will be included in the chart.
`to`	Character or an object of class `"POSIXt"`, the end date for the plot. If specified, only data up to this date will be included in the chart.
`title`	Character, optional title for the chart.
`col_body_up`	Character, color for candlestick bodies when the close price is higher than the open price. Default is "green".
`col_body_dw`	Character, color for candlestick bodies when the close price is lower than the open price. Default is "red".
`col_wick`	Character, color for candlestick wicks. Default is "black".

Value

A ggplot2 object representing the candlestick chart.

`geom_candlechart` *Candlestick plot*

Description

Candlestick plot for all time frames within the ggplot2 framework

Usage

```
geom_candlechart(mapping = NULL,
                 data = NULL,
                 stat = "candle",
                 position = "identity",
                 linejoin = "mitre",...,
                 na.rm = FALSE,
                 show.legend = NA,
                 bargap = 6,
                 method = "candle",
                 col_up = "green",
                 col_dw = "red",
                 inherit.aes = TRUE)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`stat`	The statistical transformation to use on the data for this layer, either as a ggproto Geom subclass or as a string naming the stat stripped of the stat_ prefix (e.g. "candle" rather than "stat_candle" or "heikin_ashi" rather than "stat_heikin_ashi")
`position`	Position adjustment, either as a string naming the adjustment (e.g. `"jitter"` to use `position_jitter`), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
`linejoin`	Line join style (round, mitre, bevel).
`...`	Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.
`na.rm`	If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
bargap	Numeric, positive number to regulate the distance between candles. Increasing the "bargap" reduce the distance between candles. Default is 6.
col_up	Character, color of the candle when open price is greater than close price.
col_dw	Character, color of the candle when open price is lower than close price.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

OrderBook *Order Book*

Description

Create and structure an order book from depth data.

Usage

```
OrderBook(data = NULL,
          min_price = NULL,
          max_price = NULL,
          levels = NULL,
          trades = NULL,
          as_datatable = FALSE)
```

Arguments

data	A data.frame with at least 3 columns (price, quantity and side).
min_price	Numeric. Minimum price for the aggregation of order book.
max_price	Numeric. Maximum price for the aggregation of order book.
levels	Integer. Number of levels for the aggregated order book.
trades	A data.frame containing trades data.
as_datatable	Logical, if TRUE return the order book as a datatable. Default is FALSE.

Value

A tibble or datatable object. Depth data aggregated for a specified price range.

Examples

```
# Generate an order book for BTCUSDT
depth_data <- binance_depth("BTCUSDT", api = "spot")
depth <- 0.01
best_ask <- min(depth_data[depth_data$side == "ASK",]$price)
best_bid <- max(depth_data[depth_data$side == "BID",]$price)
OrderBook(data = depth_data,
          min_price = best_bid*(1 - depth),
          max_price = best_ask*(1 + depth),
```

```
levels = 10,
trades = NULL,
as_datatable = FALSE)
```

Index