Package: golemcrypto (via r-universe)

November 23, 2026

Title Interactive Cryptocurrency Time Series and Market Data Explorer

Version 0.1.0

Maintainer Dimitrios Zacharatos <dzacharatos@yahoo.com>

Description A 'shiny' application for exploring cryptocurrency exchange rates. Plots interactive candlestick and volume charts of historical prices since 2013 for any active cryptocurrency, and shows a table of the top 100 cryptocurrencies by market capitalisation with price changes over 24 hours to 90 days, market dominance and fully diluted market cap, with Bollinger bands and the relative strength index on the price charts. Also compares normalized prices of several coins, on their own or relative to one of them, and shows a market cap treemap, a calendar of monthly returns, their seasonality by day of the week, week of the year, month, year of the halving cycle and calendar effects such as the turn of the month, whether a coin's recent volatility is normal, with a GARCH forecast, and its falls from its all-time high. Market data is retrieved from 'CoinMarketCap'.

Imports shinycssloaders, bslib, cachem, coinmarketcapr, crypto2, curl, DT, htmlwidgets, jsonlite, plotly, shiny, shinycssloaders, shinyWidgets, stats, tools, utils

Suggests htmltools, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

License GPL-3

URL https://sedzinfo.github.io/golemcrypto/, https://github.com/sedzinfo/golemcrypto, https://sedzinfo-golemcrypto.share.connect.posit.cloud/

BugReports https://github.com/sedzinfo/golemcrypto/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev zlib1g-dev

Repository https://sedzinfo.r-universe.dev

Date/Publication 2026-11-23 09:31:33 UTC

RemoteUrl https://github.com/sedzinfo/golemcrypto

RemoteRef HEAD

RemoteSha 35e09cef98b897ed19ad7dcddb0444530e8d0f23

Contents

bollinger_bands . . . 3
drawdown . . . 3
drawdown_analysis . . . 4
drawdown_episodes . . . 5
drawdown_table . . . 6
drawdown_text . . . 6
get_coins . . . 7
get_history . . . 7
get_listings . . . 8
listings_table . . . 9
monthly_returns . . . 9
moving_average . . . 10
normalize_prices . . . 11
plot_compare . . . 11
plot_drawdowns . . . 12
plot_forecast . . . 13
plot_returns_calendar . . . 13
plot_seasonality . . . 14
plot_timeseries . . . 15
plot_treemap . . . 16
plot_volatility . . . 16
relative_prices . . . 17
rsi . . . 18
run_app . . . 19
seasonal_returns . . . 20
seasonality_analysis . . . 21
seasonality_table . . . 22
seasonality_text . . . 23
volatility_analysis . . . 23
volatility_text . . . 25

Index **27**

bollinger_bands *Bollinger bands of prices*

Description

John Bollinger's bands: the moving average of the last `days` closes (see `moving_average()`), and `width` standard deviations of those closes above and below it. The bands widen when the coin swings and narrow when it is calm; a close outside them is far from its recent level. The standard deviation is that of the closes in the window, divided by `days` and not `days - 1`, as Bollinger defined it.

Usage

```
bollinger_bands(close, days = 20, width = 2)
```

Arguments

`close`	Numeric vector of close prices, sorted by date.
`days`	Number of days of the average, 20 by default.
`width`	Number of standard deviations from the average to the bands, 2 by default.

Value

A data frame as long as `close` with `middle`, `upper` and `lower`, `NA` for the first `days - 1` days.

Examples

```
bollinger_bands(c(10, 11, 12, 11, 10, 12, 14), days = 3)
```

drawdown *Drawdown of prices*

Description

How far each close is below the highest close up to that day, the all-time high so far, in percent: 0 on a new high and -50 at half of it.

Usage

```
drawdown(close)
```

Arguments

`close`	Numeric vector of close prices, sorted by date.

Value

A numeric vector as long as `close`, from -100 to 0.

Examples

```
drawdown(c(100, 120, 90, 130, 65))
```

`drawdown_analysis` *Drawdown analysis of a coin*

Description

How far is the coin below its all-time high, and how does that compare with its earlier falls? See `drawdown_episodes()` for how the falls are found.

Now: the last close against the all-time high, the lowest close since that high, the rise the price needs to get back to it, and the earlier falls that went at least as deep as the current one has so far: how many days they took to get back to their high.

The history: the falls of at least `min_depth` percent, the deepest first, the share of days that closed that far below the high, and the days that closed at a new all-time high.

Usage

```
drawdown_analysis(history, min_depth = 20)
```

Arguments

`history`	Price history of one coin from `get_history()`, with all of it: the highs are those of the history given.
`min_depth`	Smallest fall of the history to list, in percent below the high.

Value

`NULL` when there are fewer than 2 prices, or a list with:

coin, first_date, last_date, min_depth What was analysed.

now `date` and `close` of the last day, `drawdown_pct`, `high_date` and `high_price` of the all-time high, `days_since_high`, and `rise_needed_pct` to get back to it. While the price is below the high also `low_date` and `low_price`, the lowest close since the high, `deepest_pct` how far that is below it, `rank` of the current fall among all the falls of the history by that depth, 1 for the deepest, and `earlier`, the earlier falls at least as deep, as rows of `drawdown_episodes()`. At the high, `previous` holds the last fall of at least `min_depth` percent, or no rows.

episodes The falls of at least `min_depth` percent from `drawdown_episodes()`, with their `rank` in front, 1 for the deepest.

series A row per day: `date`, `close`, `high` (the all-time high so far) and `drawdown_pct` (see `drawdown()`).

share_under Percent of the days that closed at least `min_depth` percent below the high.

new_highs Days that closed at a new all-time high, the first day of the history left out.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
analysis <- drawdown_analysis(btc)
analysis$now$drawdown_pct
analysis$episodes
drawdown_text(analysis)
```

`drawdown_episodes` *Falls of the price from its all-time high*

Description

Splits a price history into its falls, the drawdown episodes. A fall starts on the last close at an all-time high, the peak, goes down to its lowest close, the trough, and ends on the first close back at or above the peak. The last fall is still going on when the price has not got back to its high. The highs are those of the history given: for a coin whose history starts after its real all-time high, the first fall starts on the first day.

Usage

```
drawdown_episodes(close, date, min_depth = 0)
```

Arguments

`close`	Numeric vector of close prices, sorted by date.
`date`	Dates of the closes.
`min_depth`	Smallest fall to keep, in percent below the peak: 20 keeps the falls that went at least 20% down. The default, 0, keeps them all.

Value

A data frame with a row per fall, the deepest first: `peak_date` and `peak_price`, `trough_date` and `trough_price`, `depth_pct` (from -100 to 0), `days_to_trough` (from the peak), `recovery_date` (`NA` while the fall goes on), `days_to_recover` (from the trough), `days_total` (from the peak to the recovery, or to the last date while the fall goes on) and `ongoing`.

Examples

```
drawdown_episodes(c(100, 120, 90, 130, 65, 70), as.Date("2026-01-01") + 0:5)
# only the falls of at least 40%
drawdown_episodes(c(100, 120, 90, 130, 65, 70), as.Date("2026-01-01") + 0:5, min_depth = 40)
```

drawdown_table *Table of the falls of a coin*

Description

A sortable table of the falls of `drawdown_analysis()`, the deepest first, with the numbers of the chart of `plot_drawdowns()`. The fall is shaded red, more strongly the deeper it is, and the current fall, which has not got back to its high, is in bold.

Usage

```
drawdown_table(analysis)
```

Arguments

`analysis`	The result of `drawdown_analysis()`.

Value

A DT datatable.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
drawdown_table(drawdown_analysis(btc))
```

drawdown_text *The drawdown analysis in words*

Description

Sentences built from the numbers of `drawdown_analysis()`, for the Drawdowns tab.

Usage

```
drawdown_text(analysis)
```

Arguments

`analysis`	The result of `drawdown_analysis()`.

Value

A list with `now` and `history`, each with a `headline` and `points`, a character vector of sentences.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
drawdown_text(drawdown_analysis(btc))
```

get_coins *List the active coins*

Description

Downloads the list of the active coins tracked by 'CoinMarketCap' and keeps it on disk for a day (see `disk_cache()`). When the download fails, the kept list is used. Its `name_symbol` column holds the labels, such as "Bitcoin (BTC)", that pick coins in `get_history()`.

Usage

```
get_coins()
```

Value

The `crypto2::crypto_list()` data frame, with a `name_symbol` column.

Examples

```
## Not run:
coins <- get_coins()
head(coins$name_symbol)

## End(Not run)
```

get_history *Get the daily price history of coins*

Description

Returns the daily USD prices of one or more coins from 'CoinMarketCap'. Each coin's history is kept on disk between R sessions, and once it is more than an hour old only the days since its last day are downloaded, so calls are fast and stay below the request limit of 'CoinMarketCap'. When that limit is hit, crypto2 waits a minute and tries again.

Usage

```
get_history(labels, start_date = Sys.Date() - 365, coins = get_coins())
```

Arguments

`labels`	Character vector of coin labels, such as "Bitcoin (BTC)", from the `name_symbol` column of `get_coins()`.
`start_date`	Date. The first day to get, a year ago by default.
`coins`	Data frame of coins from `get_coins()`, to look the labels up.

Value

A data frame with one row per coin and day, sorted by coin and date, with the `open`, `high`, `low`, `close` and `volume` columns, `timestamp` as a Date and the `name_symbol` label. Labels that match no coin are left out with a message, and `NULL` is returned when no label matches or nothing could be downloaded.

Examples

```
## Not run:
btc <- get_history("Bitcoin (BTC)", as.Date("2020-01-01"))
plot_timeseries(btc)

history <- get_history(c("Bitcoin (BTC)", "Ethereum (ETH)"))
plot_compare(normalize_prices(history))

## End(Not run)
```

`get_listings` *Get the latest market listings*

Description

Returns the listings of the coins by market cap, with their price, volume, price changes from 24 hours to 90 days, market cap and dominance in USD. The listings are downloaded again when they are older than 10 minutes, and the download is shared by all sessions and app runs in the same R process, so the API is called at most once in that time.

Usage

```
get_listings()
```

Details

The listings need a 'CoinMarketCap' API key in the `COINMARKETCAP_APIKEY` environment variable, see the README. A failed download, for example when the key has used up its credits, is tried again after 10 minutes. Until then the previous listings and their time are kept, so the time stays truthful, and `problem` tells the visitors of the app why, see `listings_problem()`.

Value

A list with `data`, a data frame with one row per coin sorted by market cap, and `time`, when it was downloaded (both `NULL` until a download works), and `problem`, `NULL` when the last download worked and otherwise a sentence on why it failed.

Examples

```
## Not run:
listings <- get_listings()
listings$time
head(listings$data)

## End(Not run)
```

`listings_table` *Table of the market listings*

Description

A sortable table with 100 rows per page and numbers rounded to 2 decimals, showing the price in dollars and the price changes in percent. Losses are shaded red and gains green, more strongly the larger the change, in steps of 0.1 percentile from the 5th to the 95th. Changes near zero keep the table's own background, so the table suits the light and the dark mode.

Usage

```
listings_table(listings)
```

Arguments

`listings`	The data of `get_listings()`.

Value

A DT datatable.

Examples

```
listings_table(get_listings()$data)
```

`monthly_returns` *Monthly returns of a coin*

Description

Each month's return is its last close relative to the last close of the month before, in percent. The first month, and the months still to come, are `NA`.

Usage

```
monthly_returns(history)
```

Arguments

history	Price history of one coin from get_history(), with timestamp and close columns, sorted by date.

Value

A year by month matrix of returns in percent, rounded to 2 decimals, with the years as row names and month.abb as column names.

Examples

```
history <- data.frame(
  timestamp = as.Date(c("2024-01-31", "2024-02-29", "2024-03-31")),
  close = c(100, 110, 121)
)
monthly_returns(history)
```

---

moving_average *Moving average of prices*

Description

The simple moving average: each day's value is the mean of the last days closes, that day included, so the first days - 1 days have none.

Usage

```
moving_average(close, days)
```

Arguments

close	Numeric vector of close prices, sorted by date.
days	Number of days to average, such as 50 or 200.

Value

A numeric vector as long as close, NA for the first days - 1 days, and all NA when there are fewer than days closes, as for a coin younger than the average.

Examples

```
moving_average(c(1, 2, 3, 4, 5), days = 3)
```

normalize_prices *Normalize close prices to 100*

Description

Every coin starts at 100 on the first date on which all of them have data, so their lines can be compared directly: 150 means the price rose by half since that date.

Usage

```
normalize_prices(history)
```

Arguments

history	Price history from get_history(), with name_symbol, timestamp and close columns, sorted by coin and date.

Value

history from the first common date on, with an index column.

Examples

```
history <- data.frame(
  name_symbol = rep(c("Alpha (A)", "Beta (B)"), each = 3),
  timestamp = rep(as.Date("2024-01-01") + 0:2, 2),
  close = c(10, 11, 12, 200, 180, 220)
)
normalize_prices(history)
```

plot_compare *Line chart of normalized close prices*

Description

One line per coin, with buttons that switch the y axis between a linear and a log scale.

Usage

```
plot_compare(prices, mode = "light", base = NULL)
```

Arguments

prices	Prices from normalize_prices().
mode	"light" (the default) or "dark", to suit a light or dark page.
base	NULL for the normalized prices, or the name_symbol of the coin that prices, from relative_prices(), are relative to, for the title and a dashed line at 100, where a coin did as well as the base.

Value

A plotly object.

Examples

```
history <- get_history(c("Bitcoin (BTC)", "Ethereum (ETH)", "Solana (SOL)"))
plot_compare(normalize_prices(history))
# against Bitcoin
plot_compare(relative_prices(normalize_prices(history), "Bitcoin (BTC)"), base = "Bitcoin (BTC)")
```

---

plot_drawdowns *Drawdown chart of a coin*

Description

Two panels over the whole history. The upper one is the close on a log scale with the all-time high so far; the lower one is the drawdown, how far each close is below that high (see `drawdown()`), with a dashed line at `min_depth` percent. The falls of the analysis are shaded from their peak to their recovery, the current one in amber, and their troughs carry their rank, as in `drawdown_table()`; hover over a trough for its dates.

Usage

```
plot_drawdowns(analysis, mode = "light")
```

Arguments

`analysis`	The result of `drawdown_analysis()`.
`mode`	`"light"` (the default) or `"dark"`, to suit a light or dark page.

Value

A plotly object.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
plot_drawdowns(drawdown_analysis(btc))
```

plot_forecast *Forecast chart of a coin*

Description

The last closes of a coin and the forecast of its next days from the GARCH model of `volatility_analysis()`: the median close, and the ranges of the middle half and of probability `level`, which start from the last close. Hover over the median line for the ranges of a day. It is the lower panel of `plot_volatility()`, on its own.

Usage

```
plot_forecast(analysis, mode = "light", context_days = 30)
```

Arguments

`analysis`	The result of `volatility_analysis()`, with a forecast: there is none when the GARCH model could not be fitted.
`mode`	`"light"` (the default) or `"dark"`, to suit a light or dark page.
`context_days`	Closes before the forecast.

Value

A plotly object.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
plot_forecast(volatility_analysis(btc))
# the next 14 days after the last 60 closes
plot_forecast(volatility_analysis(btc, forecast_days = 14), context_days = 60)
```

plot_returns_calendar *Heatmap of monthly returns by year*

Description

Months are colored from red to green by their return. The color limit is the 95th percentile of the absolute returns, so a few extreme months do not wash out the rest.

Usage

```
plot_returns_calendar(calendar, label = NULL, mode = "light")
```

Arguments

calendar	Matrix from monthly_returns().
label	Label of the coin for the title, or NULL for none.
mode	"light" (the default) or "dark", to suit a light or dark page.

Value

A plotly object.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
plot_returns_calendar(monthly_returns(btc), "Bitcoin (BTC)")
```

---

plot_seasonality *Seasonality chart of a coin*

Description

Two panels with a bar per group of seasonality_analysis(). The upper one shows the mean return, green above 0 and red below, with the median as a diamond; the lower one shows the share of the returns above 0, with a dashed line at 50%. Hover over a bar for all the numbers of its group.

Usage

```
plot_seasonality(analysis, mode = "light")
```

Arguments

analysis	The result of seasonality_analysis().
mode	"light" (the default) or "dark", to suit a light or dark page.

Value

A plotly object.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
plot_seasonality(seasonality_analysis(btc, "month"))
plot_seasonality(seasonality_analysis(btc, "day"), mode = "dark")
```

`plot_timeseries` *Candlestick chart of a coin's prices*

Description

Plots the daily open, high, low and close prices, with a line through the close, its moving averages, date range buttons and a range slider. Underneath, smaller charts show the trading volume and the drawdown: how far the price is below its all-time high so far. A click on a legend entry hides or shows that line. On request, the Bollinger bands are drawn around the prices, and a panel of the relative strength index, whose axis always runs from 0 to 100, right under them.

The vertical axes follow the days in view: zoomed in, each runs from the lowest to the highest value of those days, so a week's candles fill the chart instead of lying flat on an axis that reaches up to the all-time high. The Linear and Log buttons switch the axis of the prices; on the log axis equal distances are equal percentages, which suits the whole history.

Usage

```
plot_timeseries(
  history,
  mode = "light",
  averages = c(50, 200),
  show_drawdown = TRUE,
  bollinger = FALSE,
  show_rsi = FALSE
)
```

Arguments

`history`	Price history of one coin from `get_history()`. The averages and the drawdown use the whole history, so pass all of it and zoom with the range buttons.
`mode`	`"light"` (the default) or `"dark"`, to suit a light or dark page.
`averages`	Days of the moving averages to draw (see `moving_average()`), or `NULL` for none.
`show_drawdown`	`TRUE` to add the drawdown chart (see `drawdown()`).
`bollinger`	`TRUE` to add the 20-day Bollinger bands to the prices (see `bollinger_bands()`).
`show_rsi`	`TRUE` to add a chart of the 14-day relative strength index under the prices (see `rsi()`), with dashed lines at 30 and 70.

Value

A plotly object.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
plot_timeseries(btc)
# shorter averages, and no drawdown chart
plot_timeseries(btc, averages = c(20, 100), show_drawdown = FALSE)
# with the Bollinger bands and the RSI
plot_timeseries(btc, bollinger = TRUE, show_rsi = TRUE)
```

---

plot_treemap *Treemap of the top 100 coins*

Description

Each box is sized by market cap and colored from red to green by the 24h price change. The color limits are the 5th and 95th percentiles of the changes, so a few outliers do not wash out the rest. Coins without 24h change are labelled n/a and colored white.

Usage

```
plot_treemap(listings, mode = "light")
```

Arguments

listings	The data of get_listings().
mode	"light" (the default) or "dark", to suit a light or dark page.

Value

A plotly object.

Examples

```
plot_treemap(get_listings()$data)
```

---

plot_volatility *Volatility chart of a coin*

Description

Three panels. The upper one is the volatility of every month_days-day period over the whole history, annualized, with its median and 95th percentile: where the line is now shows how rough the last month was. The middle one shows each of the last days days as a bar of its move in the standard deviations the GARCH model expected for it, red for falls and green for rises, with lines at 2 and 4 standard deviations. The lower one shows the last context_days closes and the forecast of the next days: the median close, and the ranges of the middle half and of probability level (see volatility_analysis()). Without a GARCH model only the upper panel is drawn.

Usage

```
plot_volatility(analysis, mode = "light", days = 90, context_days = 30)
```

Arguments

analysis	The result of volatility_analysis().
mode	"light" (the default) or "dark", to suit a light or dark page.
days	Days of the middle panel.
context_days	Closes before the forecast in the lower panel.

Value

A plotly object.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
plot_volatility(volatility_analysis(btc))
```

---

relative_prices *Normalized prices relative to one of the coins*

Description

The relative strength of each coin against a base coin: the ratio of their normalized prices, so 100 on the first common date, as in normalize_prices(). A line that rises did better than the base, one that falls did worse, and 120 means 20% better since the start: Ethereum against Bitcoin is the ETH/BTC ratio, normalized to 100. On a day the base has no close, its last close before stands in.

Usage

```
relative_prices(prices, base)
```

Arguments

prices	Prices from normalize_prices().
base	The name_symbol of the coin to divide by, one of those of prices.

Value

The rows of prices of the other coins, with index the ratio to the base times 100, the base's own index in base_index, and base.

Examples

```
history <- data.frame(
  name_symbol = rep(c("Alpha (A)", "Beta (B)"), each = 3),
  timestamp = rep(as.Date("2024-01-01") + 0:2, 2),
  close = c(10, 11, 12, 200, 180, 220)
)
relative_prices(normalize_prices(history), "Beta (B)")
```

`rsi` *Relative strength index of prices*

Description

J. Welles Wilder's RSI, from 0 to 100: the share of the average daily rise in the average daily move, `100 - 100 / (1 + average rise / average fall)`. The first averages are the means of the rises and the falls of the first `days` days; after that each day counts for `1 / days` of the average and the day before for the rest, as Wilder smoothed them. Traders read above 70 as overbought, a run up that may pause, and below 30 as oversold.

Usage

```
rsi(close, days = 14)
```

Arguments

`close`	Numeric vector of close prices, sorted by date.
`days`	Number of days of the averages, 14 by default.

Value

A numeric vector as long as `close`, `NA` for the first `days` days, 100 when the coin only rose and 50 when it did not move.

Examples

```
rsi(c(100, 101, 102, 101, 103, 104, 103, 105), days = 3)
```

run_app *Launch the Crypto Time Series Analyzer*

Description

Starts an interactive 'shiny' application for exploring cryptocurrency exchange rates and market data from 'CoinMarketCap'.

Usage

```
run_app(
  api_key = Sys.getenv("COINMARKETCAP_APIKEY"),
  analytics = FALSE,
  keep_alive = TRUE,
  ...
)
```

Arguments

api_key	A 'CoinMarketCap' API key, needed for the market listings. By default it is read from the COINMARKETCAP_APIKEY environment variable, which can be set in .Renviron.
analytics	TRUE to add Google Analytics, through the cookie banner shared with the sites of sedzinfo.github.io, which loads it only after the visitor accepts. Meant for the public deployment (see app.R); off by default, so that running the app yourself sends nothing.
keep_alive	TRUE (the default) to ping the server every 15 seconds while the page is open, so that proxies do not close idle connections. FALSE lets the app go to sleep when nobody uses it, as app.R does on shinyapps.io, where an app that stays awake uses up the active hours of the plan.
...	Further arguments passed to shiny::shinyApp(), for example options = list(port = 8080).

Details

The application has the following tabs:

Timeseries Candlestick chart of daily open, high, low and close prices in USD since 2013 for any active cryptocurrency, with its 50- and 200-day moving averages, optional Bollinger bands and relative strength index, and charts of the volume and of the drawdown (how far the price is below its all-time high) underneath.

Top 100 Table of the top 100 cryptocurrencies by market cap with price, volume, 24h to 90d price changes, market dominance and fully diluted market cap.

Compare Close prices of up to 10 coins, normalized to 100 on the first date all of them have data, or relative to one of them.

Treemap Top 100 coins sized by market cap and colored by 24h change.

Returns calendar Monthly returns by year for the coin selected in the Timeseries tab.

Seasonality The returns of the coin selected in the Timeseries tab by day of the week, week of the year, month or year of the halving cycle, or a calendar effect of the stock market against the rest (the turn of the month, the Santa rally or Sell in May), with a test of whether the differences could be chance.

Volatility Whether the volatility of the last weeks and the move of the last days of the coin are normal, and a GARCH forecast of the next days with the value at risk.

Drawdowns How far the coin is below its all-time high, and how deep its earlier falls went and how long they took to recover.

About What every tab shows, and how its numbers are calculated.

Market listings are downloaded when the app starts and refreshed every 10 minutes. When they cannot be downloaded, for example because the API key has used up its credits, the app still starts, and the Top 100 and Treemap tabs tell the visitors why and when the limit resets. Historical prices are downloaded on demand with `crypto2::crypto_history()` and kept on disk between R sessions, so after the first download only the days since the last one are fetched. An internet connection is required. When 'CoinMarketCap' limits the requests, crypto2 prints "Retrying in 60 seconds" and the chart appears about a minute later.

Value

A `shiny::shinyApp()` object. Printing it, or calling `run_app()` at the console, opens the app in the browser.

Examples

```
if (interactive()) run_app()

# with the API key passed directly, on a fixed port
if (interactive()) run_app(api_key = "your-api-key", options = list(port = 8080))
```

---

seasonal_returns *The return of each complete period of a coin*

Description

A period's return is its last close relative to the last close of the period before, in percent, as in `monthly_returns()`. Only complete periods count: the period and the one before must both have a close on their last day. The first period, which has none before it, the current one, which has not ended, and the periods next to missing days are left out.

Each return is put in a group: the day of the week for days, the ISO 8601 week of the year for weeks (Monday to Sunday), the month for months, and the year of the 4-year cycle of Bitcoin's halvings for years. The halvings came in 2012, 2016, 2020 and 2024, so a year divisible by 4 is a halving year and 2025 is "Halving + 1".

The calendar effects put the returns in two groups: `"turn"` and `"santa"` the returns of the days, in or out of the turn of the month or the Santa rally (see `effect_days()`), and `"half"` the returns of the halves of the year, May to October and November to April, of "Sell in May and go away".

Usage

```
seasonal_returns(
  history,
  period = c("day", "week", "month", "year", "turn", "santa", "half")
)
```

Arguments

history	Price history of one coin from get_history(), with timestamp and close columns.
period	"day", "week", "month" or "year", or a calendar effect: "turn", "santa" or "half".

Value

A data frame with a row per complete period, the oldest first: start and end, the first and last day of the period, group, a factor with the groups in their natural order, and return_pct.

Examples

```
history <- data.frame(
  timestamp = as.Date("2024-01-01") + 0:120,
  close = 100 * cumprod(c(1, rep(1.002, 120)))
)
seasonal_returns(history, "month")
head(seasonal_returns(history, "day"))
```

---

seasonality_analysis *Seasonality analysis of a coin*

Description

Do some days of the week, weeks of the year, months or years of the halving cycle bring better returns than others? The returns of the complete periods (see seasonal_returns()) are summed up by group, and a Kruskal-Wallis test asks whether the differences between the groups are larger than chance would give. The test compares the ranks of the returns, so a few extreme months do not decide it, but it takes the returns as independent, which crypto returns are only roughly: calm and wild stretches come in runs.

The calendar effects ("turn", "santa" and "half", see seasonal_returns()) have two groups, the returns in the effect and the rest, for which the test is the same as a Wilcoxon rank-sum test.

Usage

```
seasonality_analysis(
  history,
  period = c("day", "week", "month", "year", "turn", "santa", "half")
)
```

Arguments

history	Price history of one coin from get_history(), with all of it.
period	"day", "week", "month" or "year", or a calendar effect: "turn" (the turn of the month), "santa" (the Santa rally) or "half" (Sell in May).

Value

NULL when the history has no complete period with one before it, or a list with:

coin, period What was analysed.

returns The returns of seasonal_returns().

groups A row per group that has returns, in their natural order: group, returns (how many), mean_pct, median_pct, up_pct (the share of the returns above 0), best_pct and worst_pct.

all The same numbers for all the returns together, as a list.

p_value The p-value of the Kruskal-Wallis test, NA when fewer than two groups have returns, or no group has two.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
analysis <- seasonality_analysis(btc, "month")
analysis$groups
analysis$p_value
seasonality_text(analysis)
```

---

seasonality_table *Table of the seasonality of a coin*

Description

A table with a row per group of seasonality_analysis(), in their natural order, with the numbers of the chart of plot_seasonality(). The mean and the median are shaded red below 0 and green above it, and the share up red below 50% and green above it, more strongly the further they are from the middle.

Usage

```
seasonality_table(analysis)
```

Arguments

analysis	The result of seasonality_analysis().

Value

A DT datatable.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
seasonality_table(seasonality_analysis(btc, "month"))
```

---

seasonality_text *The seasonality analysis in words*

Description

Sentences built from the numbers of `seasonality_analysis()`, for the Seasonality tab: the best and the worst group, or for a calendar effect its two groups, and whether the differences between the groups could be chance.

Usage

```
seasonality_text(analysis)
```

Arguments

`analysis`	The result of `seasonality_analysis()`.

Value

A list with `groups` and `chance`, each with a `headline` and `points`, a character vector of sentences.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
seasonality_text(seasonality_analysis(btc, "day"))
```

---

volatility_analysis *Volatility analysis of a coin*

Description

Is the coin's recent volatility normal? The analysis looks at the daily log returns, `log(close(t) / close(t-1))`, in two ways.

The last month (`month_days` days, 30 by default): its volatility, the standard deviation of the daily returns times `sqrt(365)`, against the year before it and against every past period of the same length; an F test and the Fligner-Killeen test, which is robust to a few extreme days, of whether its variance is larger than the year before; its largest daily move, in standard deviations of the year before; and its return against every past period of the same length.

The last days (`recent_days` days): their return against every past period of as many days; the same, in standard deviations of the year before; a GARCH(1,1) model with Student-t errors (see `garch_fit()`), fitted to the `garch_years` years before them, which judges each day against the

volatility the model expected for it given the days before; and each day's volume against the `volume_days` days before it (see `volume_ratio()`), judged by how often the coin's volume was as far from usual.

The next days (`forecast_days` days): the same model, with its variance brought up to the last day, forecasts the volatility and the range of the price (see `garch_forecast()`), and gives the value at risk and the expected shortfall of the next day (see `garch_risk()`). A Kupiec test (see `kupiec_test()`) checks how often the model's value at risk was exceeded over the last year. The value at risk and expected shortfall of the last year's daily returns are given without the model too.

Usage

```
volatility_analysis(
  history,
  month_days = 30,
  recent_days = 2,
  garch_years = 4,
  forecast_days = 7,
  level = 0.95,
  volume_days = 30
)
```

Arguments

`history`	Price history of one coin from `get_history()`, sorted by date. The comparisons use the whole history, so pass all of it.
`month_days`	Days of the volatility window, the "month": 30, or for example 7 for the last week or 90 for the last quarter.
`recent_days`	Days in the recent move, such as 1 for the last day alone.
`garch_years`	Years of history to fit the GARCH model to.
`forecast_days`	Days to forecast.
`level`	Probability of the forecast ranges, and of the value at risk: 0.95 is the 95% range and the worst 5% of days.
`volume_days`	Days before each day whose median volume the day's volume is compared with.

Value

`NULL` when the history is not longer than a year and the longer of `month_days` and `recent_days`, or the price did not move at all in the year before the month or the last days, otherwise a list with

- `coin`, `first_date` and `last_date`;
- `month`: a list of the month's measures, see the code;
- `recent`: a list of the measures of the last days, with a `days` data frame of each day's return, GARCH measures, `volume`, `volume_ratio` and `volume_one_in_days` (`NA` without volume);
- `garch`: the fitted model, or `NULL` when it could not be fitted or there are fewer than 500 days before the last days;

- `forecast`: `NULL` without a model, otherwise a list with `days`, a data frame with a row per day ahead: its `date`, `sigma` (the daily standard deviation the model expects, in percent) and the closes `low`, `low_50`, `median`, `high_50` and `high` of the ranges (see `garch_forecast()`); `sigma` and `sigma_long_run`, the daily standard deviation expected for the next day and in the long run; `half_life`, the days for the volatility to get half-way back to the long run; `var` and `es`, the value at risk and expected shortfall of the next day; and `backtest`, the `days`, `exceedances` and `p_value` of the Kupiec test;
- `history_risk`: `var` and `es` of the daily returns of the last `days` (365) days, in percent;
- `series`: a data frame with a row per day and the columns `date`, `close`, `volume`, `return_pct`, `volume_ratio` (see `volume_ratio()`), `volatility_month` (of the `month_days` days up to the day, annualized, in percent), `garch_sigma` (the daily standard deviation the model expected, in percent), `garch_z` (the day's move in those standard deviations) and `garch_var` (the model's value at risk for the day, in percent);
- `settings`: `month_days`, `recent_days`, `garch_years`, `forecast_days`, `level` and `volume_days`.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
analysis <- volatility_analysis(btc)
analysis$month$level
analysis$recent$days
analysis$forecast$days
volatility_text(analysis)
# the volatility of the last week, and the move of the last day
volatility_text(volatility_analysis(btc, month_days = 7, recent_days = 1))$month$headline
```

---

volatility_text *The volatility analysis in words*

Description

Sentences built from the numbers of `volatility_analysis()`, for the Volatility tab.

Usage

```
volatility_text(analysis)
```

Arguments

`analysis`	The result of `volatility_analysis()`.

Value

A list with `month`, `recent` and `forecast`, each with a `headline` and `points`, a character vector of sentences.

Examples

```
btc <- get_history("Bitcoin (BTC)", as.Date("2013-04-29"))
volatility_text(volatility_analysis(btc))
```

Index

∗ **crypto**
    run_app, 19
∗ **timeseries**
    run_app, 19

bollinger_bands, 3
bollinger_bands(), *15*

crypto2::crypto_history(), *20*
crypto2::crypto_list(), *7*

disk_cache(), *7*
drawdown, 3
drawdown(), *4*, *12*, *15*
drawdown_analysis, 4
drawdown_analysis(), *6*, *12*
drawdown_episodes, 5
drawdown_episodes(), *4*
drawdown_table, 6
drawdown_table(), *12*
drawdown_text, 6

effect_days(), *20*

garch_fit(), *23*
garch_forecast(), *24, 25*
garch_risk(), *24*
get_coins, 7
get_coins(), *7*
get_history, 7
get_history(), *4*, *7*, *10, 11*, *15*, *21, 22*, *24*
get_listings, 8
get_listings(), *9*, *16*

kupiec_test(), *24*

listings_problem(), *8*
listings_table, 9

monthly_returns, 9
monthly_returns(), *14*, *20*
moving_average, 10
moving_average(), *3*, *15*

normalize_prices, 11
normalize_prices(), *11*, *17*

plot_compare, 11
plot_drawdowns, 12
plot_drawdowns(), *6*
plot_forecast, 13
plot_returns_calendar, 13
plot_seasonality, 14
plot_seasonality(), *22*
plot_timeseries, 15
plot_treemap, 16
plot_volatility, 16
plot_volatility(), *13*

relative_prices, 17
relative_prices(), *11*
rsi, 18
rsi(), *15*
run_app, 19

seasonal_returns, 20
seasonal_returns(), *21, 22*
seasonality_analysis, 21
seasonality_analysis(), *14*, *22, 23*
seasonality_table, 22
seasonality_text, 23
shiny::shinyApp(), *19, 20*

volatility_analysis, 23
volatility_analysis(), *13*, *16, 17*, *25*
volatility_text, 25
volume_ratio(), *24, 25*