CryptoR

```
library(CryptoR)
```

Introduction

An interface to the CoinGecko cryptocurrency API.

Main site: https://www.coingecko.com/en
API site: https://www.coingecko.com/en/api
V3 API documentation: https://www.coingecko.com/api/documentations/v3

CoinGecko API benefits as of April 2021:

- 100% free crypto API
- No keys required
- Publicly available

Notes:

- Rate limit: 100 requests/minute
- Time stamps returned are UTC

Overview

CryptoR {CryptoR} R Documentation

CryptoR: an interface to the CoinGecko cryptocurrency API.

Description

The CryptoR package provides 3 API and 1 wrapper functions.

API functions

cg_get_ping(). Get CoinGecko API server status.

cg_get_global(). Get global cryptocurrency data.

cg_get_coins_markets(...). Get all supported coins price, market cap, volume, and market related data.

User functions

cg_coins_df(...). Helper for `cg_get_coins_markets(...)`. Builds a custom size data frame given that `cg_get_coins_markets(...)` is limited to returning 250 symbols.

See Also

`cg_get_ping`, `cg_get_global`, `cg_get_coins_markets`, `cg_coins_df`

API Connection

Let's first check the API connection to CoinGecko using `cg_get_ping()`.

This should return: (screenshot)

```
$gecko_says
[1] "(V3) To the Moon!"
```

```
cg_get_ping()
#> $gecko_says
#> [1] "(V3) To the Moon!"
```

With a verified connection we can answer some questions.

(1) What's the broad crypto market like?

We can request a snapshot of the cryptocurrency market via `cg_get_global()`:

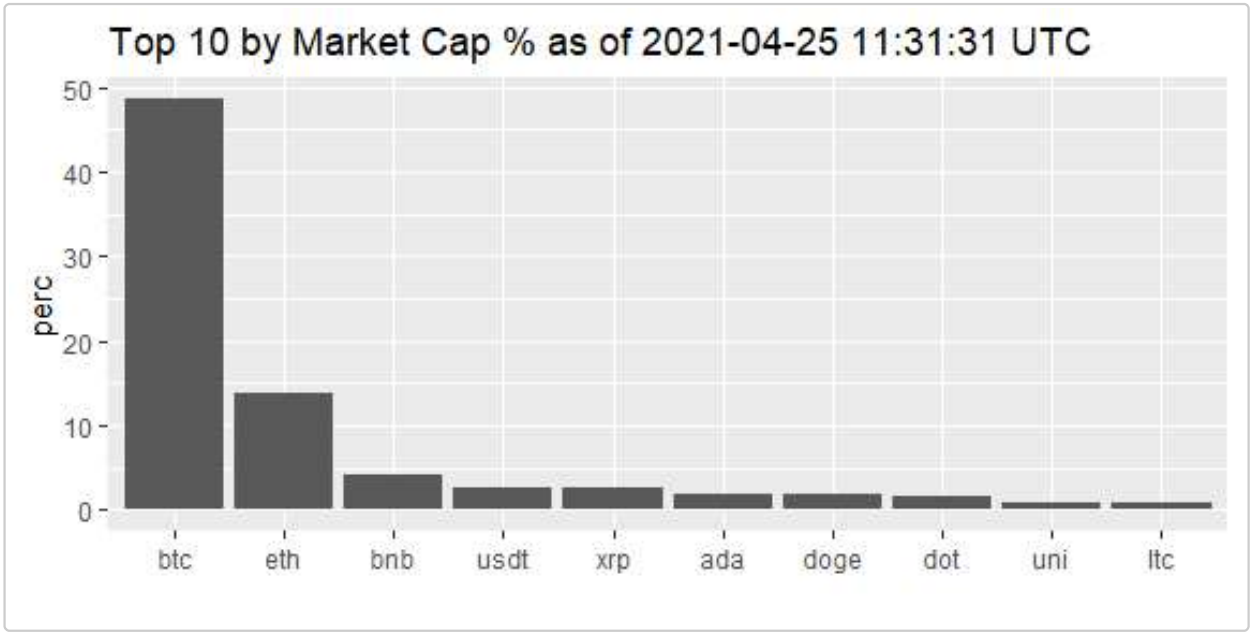
```
global <- cg_get_global()
dateTime <- lubridate::now("UTC")
```

The object called `global` contains: (screenshot)

global	list [1]	List of length 1
data	list [10]	List of length 10
active_cryptocurrencies	integer [1]	6782
upcoming_icos	integer [1]	0
ongoing_icos	integer [1]	50
ended_icos	integer [1]	3375
markets	integer [1]	597
total_market_cap	list [61]	List of length 61
total_volume	list [61]	List of length 61
market_cap_percentage	list [10]	List of length 10
market_cap_change_percentage_24h_usd	double [1]	-0.5256325
updated_at	integer [1]	1618929444

Let's do a quick plot of the `market_cap_percentage` - it will be the top 10 cryptocurrencies ranked by market capitalization percentage.

```
# pull out the data
mcp1 <- tibble::as_tibble(global$data$market_cap_percentage)
mcp1 <- tidyr::pivot_longer(mcp1, everything(), names_to="symbol",
                            values_to="perc")

# build the plot
ggplot2::ggplot(mcp1, ggplot2::aes(x=reorder(symbol, -perc), y=perc) ) +
  ggplot2::geom_col() +
  ggplot2::labs(title=paste0("Top 10 by Market Cap % as of ",
                             dateTime, " UTC"), x="")
```




```
# format percentage sum
topTenPerc <- round( sum(mcp1$perc), 2)
```

We see that the **top 10** cryptocurrencies account for **78.78** % of the crypto market.

(2) How is the market capitalization distributed?

```
activeCurrencies <- global$data$active_cryptocurrencies
```

CoinGecko tracks a lot of cryptocurrencies: **6846 coins** in total. We can call the function `cg_get_coins_markets(...)` to get supported coins price, market capitalization, volume, and market related data - over 20 data points.

There are 8 parameters and CoinGecko in this instance puts a request limit to 1 page with a maximum of 250 coins or symbols on the page.

`cg_get_coins_markets(...)` (screenshot)

```
cg_get_coins_markets(
  vs_currency = "usd",
  ids = NULL,
  category = NULL,
  order = "market_cap_desc",
  per_page = 10,
  page = 1,
  sparkline = FALSE,
  price_change_percentage = NULL
)
```

We will not request the data for all currencies - just the **first 1000** symbols. We'll have to use a custom function - `cg_coins_df(...)` - it's basically a wrapper for `cg_get_coins_markets(...)`. The function `cg_coins_df(...)` uses a loop to build a custom sized data frame.

```
customDF <- cg_coins_df(per_page = 250, pages = 4)
```

A quick glimpse will see if we received 1000 ordered symbols and also which variable names are available for analysis.

```
dplyr::glimpse(customDF)
#> Rows: 1,000
#> Columns: 27
#> $ id                                <chr> "bitcoin", "ethereum", "binancecoin",~
#> $ symbol                            <chr> "btc", "eth", "bnb", "usdt", "xrp", "~
#> $ name                              <chr> "Bitcoin", "Ethereum", "Binance Coin"~
#> $ image                             <chr> "https://assets.coingecko.com/coins/i~
#> $ current_price                     <dbl> 4.95370e+04, 2.27457e+03, 4.98750e+02~
#> $ market_cap                        <dbl> 925892007976, 262970432850, 768390448~
#> $ market_cap_rank                   <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12~
#> $ fully_diluted_valuation           <dbl> 1.040276e+12, NA, 8.479475e+10, NA, N~
#> $ total_volume                      <dbl> 40804328211, 32017484717, 3341386699,~
#> $ high_24h                          <dbl> 5.10700e+04, 2.29522e+03, 5.09360e+02~
#> $ low_24h                           <dbl> 4.88840e+04, 2.15421e+03, 4.80820e+02~
#> $ price_change_24h                  <dbl> 81.14000000, 56.87000000, 5.20000000,~
#> $ price_change_percentage_24h       <dbl> 0.16406, 2.56417, 1.05368, 0.08407, -~
#> $ market_cap_change_24h             <dbl> 1561010316, 6695508900, 618804115, 11~
#> $ market_cap_change_percentage_24h <dbl> 0.16888, 2.61263, 0.81186, 0.23850, -~
#> $ circulating_supply                <dbl> 1.869094e+07, 1.156131e+08, 1.545337e~
#> $ total_supply                      <dbl> 2.100000e+07, NA, 1.705337e+08, 4.957~
```

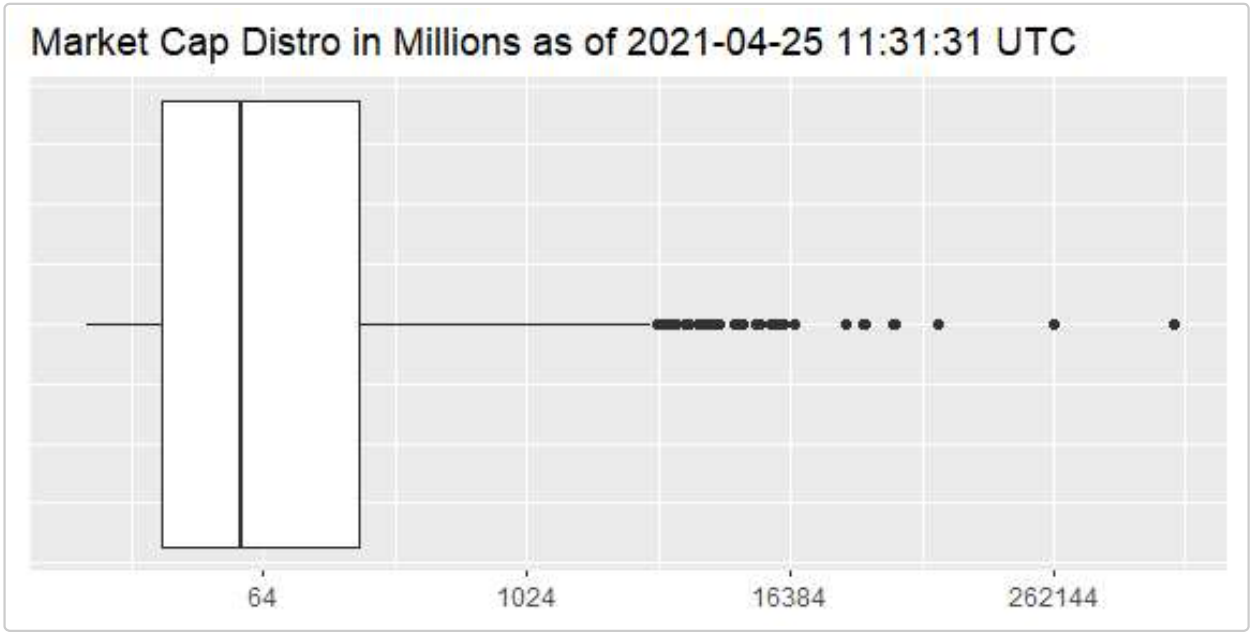
```
#> $ max_supply                <dbl> 21000000.0, NA, 170533651.9, NA, NA, ~
#> $ ath                       <dbl> 6.48050e+04, 2.64037e+03, 6.10060e+02~
#> $ ath_change_percentage     <dbl> -23.70245, -13.99847, -18.49399, -24.~
#> $ ath_date                  <chr> "2021-04-14T11:54:46.763Z", "2021-04-~
#> $ atl                       <dbl> 6.781000e+01, 4.329790e-01, 3.981770e~
#> $ atl_change_percentage     <dbl> 7.281718e+04, 5.243499e+05, 1.248671e~
#> $ atl_date                  <chr> "2013-07-06T00:00:00.000Z", "2015-10-~
#> $ roi                       <df[,3]> <data.frame[26 x 3]>
#> $ last_updated              <chr> "2021-04-25T11:29:53.084Z", "2021-~
#> $ sparkline_in_7d           <df[,1]> <data.frame[26 x 1]>
```

Let's look at the five-number summary for market cap in millions of dollars.

```
summary(customDF$market_cap / 10^6)
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#>      9.9     21.7     50.0   1895.6    174.7 925892.0
```

We can see that the data is heavily skewed. Most people may realize that the market is dominated by a few names. So, we will log transform in order to see a clearer boxplot - again in millions of dollars.

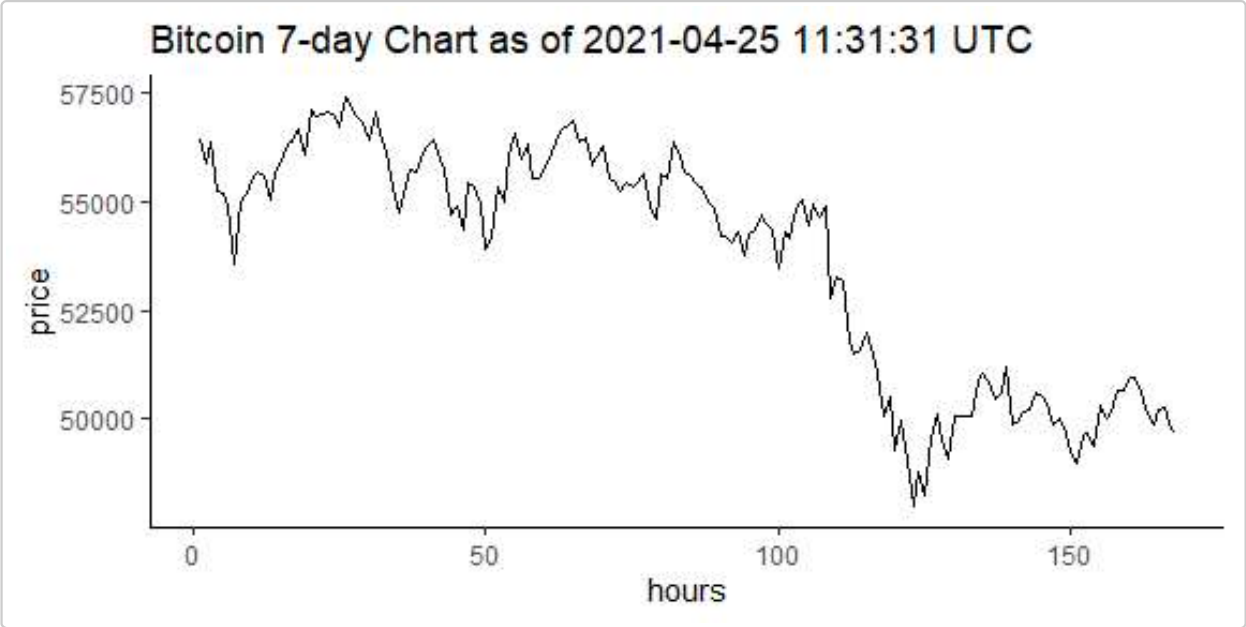
```
# plot building
ggplot2::ggplot(customDF, ggplot2::aes(x=market_cap / 10^6) ) +
  ggplot2::geom_boxplot() +
  ggplot2::scale_x_continuous(trans='log2') +
  ggplot2::theme(axis.title.x=ggplot2::element_blank(),
                 axis.text.y=ggplot2::element_blank(),
                 axis.ticks.y=ggplot2::element_blank()) +
  ggplot2::labs(title=paste0("Market Cap Distro in Millions as of ",
                             dateTime, " UTC") )
```



(3) What about individual coin performance?

Our custom data frame `customDF` includes a variable called `sparkline_in_7d` which provides 7 days of hourly price information or 168 data points. For example, a 7-day sparkline looks like this:

```
# pull out the sparkline data
coin <- dplyr::filter(customDF, name == "Bitcoin")
sparkLine <- tibble::tibble( price = unlist(coin$sparkline_in_7d) )
sparkLine <- tibble::rowid_to_column(sparkLine, "hours")

# build the plot
ggplot2::ggplot(sparkLine, ggplot2::aes(x=hours, y=price ) ) +
  ggplot2::geom_line() +
  ggplot2::theme_classic() +
  ggplot2::labs(title=paste0(coin$name, " 7-day Chart as of ",
                             dateTime, " UTC") )
```



The `sparkline_in_7d` variable allows us to do some unique charting and analysis. We can create a **linear model for each symbol** and then extract the slope to give us a rough idea of the 7-day trend.

First we need to check for valid `sparkline_in_7d` data. CoinGecko can at times return `numeric(0)` for a symbol which is specifically annoying. We'll also check whether a symbol has enough data points; in this case a minimum of 6 days of hourly data.

```
# function def
sparkLen <- function(df){
  return( length(unlist(df$sparkline_in_7d)) )
}

# filter columns and check sparkline data
coins <- dplyr::select(customDF, c(name,sparkline_in_7d) )
coins <- dplyr::mutate(coins, len = apply(coins, 1, sparkLen) )
coins <- dplyr::filter(coins, len > 24*6)
lenMod <- nrow(coins)
```

After filtering the symbol count is now **953**.

We can now use `apply` on each row of the tibble and send data to the `price_model()` function for model creation and slope extraction.

```
# function def and returning just the slope
price_model <- function(df){
  sparkLine <- tibble::tibble( price = unlist(df$sparkline_in_7d) )
  sparkLine <- tibble::rowid_to_column(sparkLine, "hours")
  fit <- lm( price ~ hours, data = sparkLine )
  return(fit$coefficients[2])
}

# for each data frame row, get the sparkline's lm model
coins <- dplyr::mutate(coins, slope = apply( coins, 1, price_model ) )
```

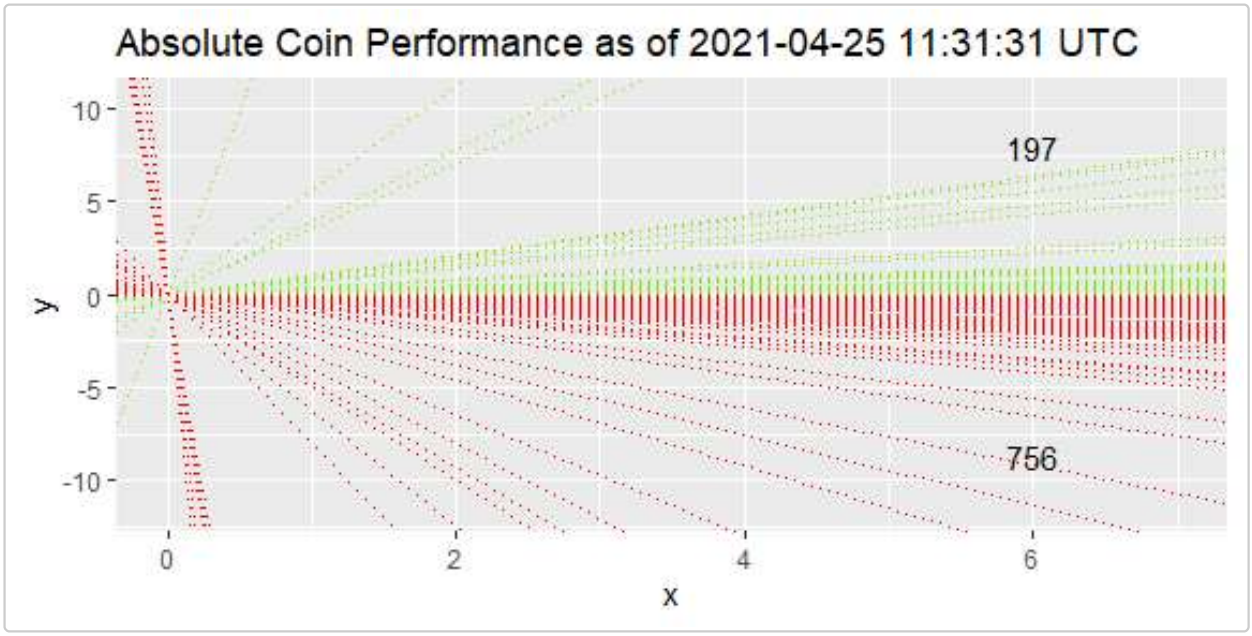
Let's look at some of the extracted slope data.

```
head(coins[-2:-3] )
#>           name         slope
#> 1      Bitcoin -4.750559e+01
#> 2     Ethereum  9.220919e-01
#> 3 Binance Coin  9.354060e-02
#> 4       Tether -1.992397e-06
#> 5          XRP -2.082888e-03
#> 6      Cardano -8.927396e-04
```

We can now plot the slopes on a single chart starting from a common point - intercept at 0 (zero).

```
# for the chart's y-axis boundaries
avg <- mean(coins$slope)
stdDev <- sd(coins$slope)
upperY <- avg+stdDev*2
lowerY <- avg-stdDev*2

slopeGraph <- ggplot2::ggplot() +
  ggplot2::xlim(0, 7) +
  ggplot2::ylim(lowerY, upperY) +
  ggplot2::geom_abline( intercept = 0, slope = coins$slope,
                        linetype = "dotted",
                        color = ifelse(coins$slope<0,"red","chartreuse2") )

# finalize the graph's presentation
posNums <- sum( coins$slope > 0 )
negNums <- sum( coins$slope < 0 )
slopeGraph <- slopeGraph +
  ggplot2::annotate(geom="text", x=6, y=upperY*.75, label=posNums,
                    color="black") +
  ggplot2::annotate(geom="text", x=6, y=lowerY*.75, label=negNums,
                    color="black") +
  ggplot2::ggtitle( paste0("Absolute Coin Performance as of ",
                           dateTime, " UTC") )
```

```
# display it
slopeGraph
```



The graph shows how many coins are up or down on the weekly trend using the `lm` or linear model function.

Let's get the names of the top and bottom 10 cryptocurrencies for the past week.

```
topPerform <- dplyr::slice_max(coins[-2], n=10, order_by=slope)
worstPerform <- dplyr::slice_min(coins[-2], n=10, order_by=slope)
```

The top performers:

```
topPerform
#>                 name len      slope
#> 1     ARC Governance 169 19.8435052
#> 2              Maker 169  5.6227366
#> 3            Olympus 168  3.9405350
#> 4     cVault.finance 168  3.5196258
#> 5               sETH 168  1.0655935
#> 6  Lido Staked Ether 156  1.0345488
#> 7           Ethereum 168  0.9220919
#> 8           Compound 168  0.7955422
#> 9                DEA 162  0.7182992
#> 10   Mirrored Google 161  0.4205602
```

and the worst performers:

```
dplyr::arrange(worstPerform, desc(-slope) )
#>               name len     slope
#> 1     Bounce [old] 165 -79.64747
#> 2    yearn.finance 169 -64.94700
#> 3      pTokens BTC 168 -50.87071
#> 4             tBTC 168 -49.30393
#> 5  Wrapped Bitcoin 168 -47.54822
#> 6          Bitcoin 168 -47.50559
```

```
#> 7        Huobi BTC 169 -47.48117
#> 8            DIGG 169 -46.78225
#> 9          renBTC 169 -46.49538
#> 10           sBTC 169 -44.29277
```

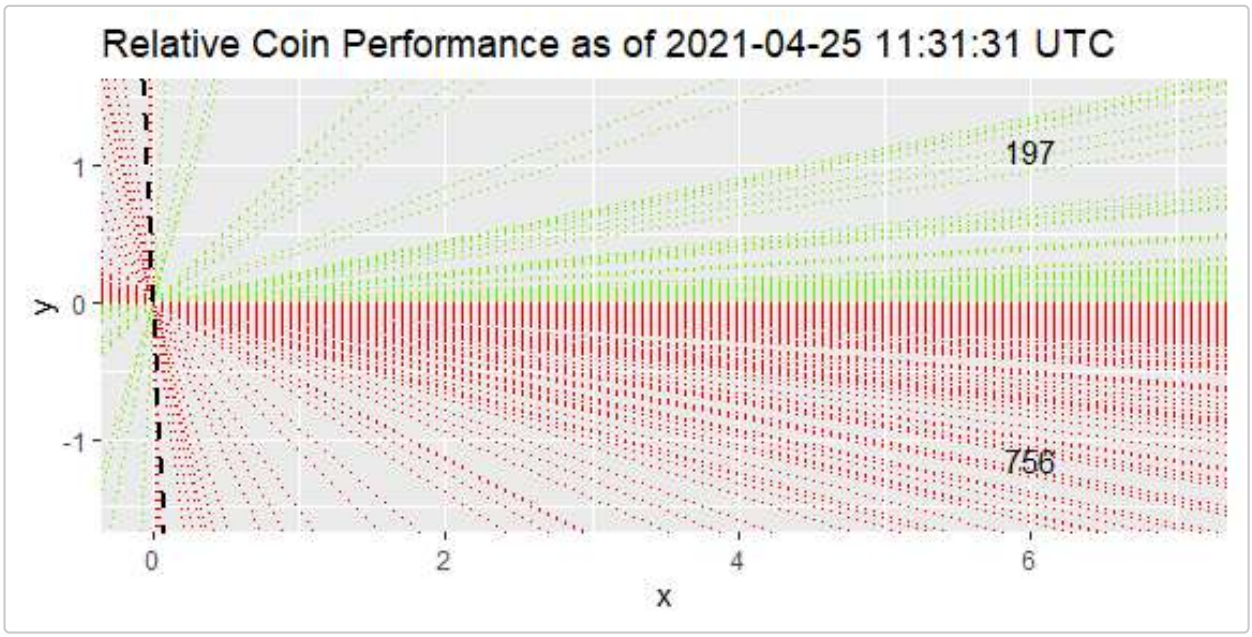
(4) What's the overall crypto market pulse?

The previous discussion was about absolute performance of individual coins. Here we'll consider the relative performance.

Again, we'll use the coins' or symbols' slope measurement but weigh it **by market capitalization.** We can then get a sense of the market cap weighted direction of the coins.

```
# pull out "market cap" data and add it to another using join
marketCapDf <- dplyr::select(customDF, name, market_cap)
coinsWeighting <- dplyr::left_join(coins, marketCapDf, by = "name")

# calculate the slope weighting
marketCapSum <- sum(coinsWeighting$market_cap)
coinsWeighting <- dplyr::mutate(coinsWeighting,
                  slopeWeighted = (market_cap/marketCapSum)*slope )

# the market's overall weighted slope
slopeAll <- sum(coinsWeighting$slopeWeighted)

# what follows - just like above for slopeGraph
# maybe a separate function is better but for now...

# y-axis limits
avg <- mean(coinsWeighting$slopeWeighted)
stdDev <- sd(coinsWeighting$slopeWeighted)
upperY <- avg+stdDev*2
lowerY <- avg-stdDev*2

slopeGraph2 <- ggplot2::ggplot() +
  ggplot2::xlim(0, 7) +
  ggplot2::ylim(lowerY, upperY) +
  ggplot2::geom_abline( intercept = 0, slope = coinsWeighting$slope,
                  linetype = "dotted",
                  color = ifelse(coinsWeighting$slope<0,"red","chartreuse2") )

# prettify the graph
posNums <- sum( coinsWeighting$slopeWeighted > 0 )
negNums <- sum( coinsWeighting$slopeWeighted < 0 )
slopeGraph2 <- slopeGraph2 +
  ggplot2::annotate(geom="text", x=6, y=upperY*.75, label=posNums,
                    color="black") +
  ggplot2::annotate(geom="text", x=6, y=lowerY*.75, label=negNums,
                    color="black") +
  ggplot2::ggtitle(paste0("Relative Coin Performance as of ",
                          dateTime, " UTC") )

# add the overall market slope
```

```
slopeGraph2 <- slopeGraph2 +
  ggplot2::geom_abline(intercept = 0, slope = slopeAll, size = 1,
                       linetype = "dashed", color="black")

# display it
slopeGraph2
```



The **dashed black line** gives the market cap weighted direction of the overall market - or in this case, considering the top 953 cryptos. Slope value equals **-23.430595**.

Lastly, let's look at the 10 cryptocurrencies whose **slopes by weighted market cap** are affecting the market the greatest. This could be **upwards or downwards** pressure! Here we'll just look at the absolute values of weighted slopes.

The top 10 are:

```
coinsWeighting <- dplyr::mutate(coinsWeighting, absVal = abs(slopeWeighted) )
coinsWeighting <- dplyr::arrange(coinsWeighting, by=-absVal)
dplyr::slice_max(coinsWeighting[-2:-6], n=10, order_by=absVal)
#>               name       absVal
#> 1          Bitcoin 23.232646545
#> 2  Wrapped Bitcoin  0.195085909
#> 3         Ethereum  0.128078068
#> 4    yearn.finance  0.050143084
#> 5        Huobi BTC  0.034953622
#> 6     Bounce [old]  0.025635626
#> 7           renBTC  0.014177119
#> 8            Maker  0.010555639
#> 9     Bitcoin Cash  0.008250344
#> 10    Binance Coin  0.003796433
```

Closing

This was a brief introduction into some cyrpto market analysis using the CryptoR package.

API functions

cg_get_ping(). Get CoinGecko API server status.

cg_get_global(). Get global cryptocurrency data.

cg_get_coins_markets(...). Get all supported coins price, market cap, volume, and market related data.

User functions

cg_coins_df(...). Helper for `cg_get_coins_markets(...)`. Builds a custom size data frame given that `cg_get_coins_markets(...)` is limited to returning 250 symbols.

The CoinGecko API exposes much more than what has been implemented here.

But the previously created data frame `customDF` pulls in plenty of data from which further charting, exploring, and analysis can be done.
