Build A Cryptocurrency Trading Bot with R

Brad Lindblad Feb 18, 2018 · 7 min read

Photo by Branko Stancevic on Unsplash

*** Note that the API used in this tutorial is no longer in service. This article should be read for illustrative purposes with that in mind.*

We should buy when our reptile brain wants to sell. We should sell when our guts want us to buy more.

It is even more difficult to trade cryptocurrencies with a a critical constitution. The young and emerging markets are flooded with “pump groups” that foster intense FOMO (fear of missing out) which drive prices sky-high before body-slamming them back down to earth. Many novice investors also trade on these markets, investors that possibly never entered a trade on the NYSE. On every trade, there is a maker and a taker, and shrewd crypto investors find it easy to take advantage of the novices flooding the space.

In order to detach my emotions from crypto trading and to take advantage of markets open 24/7, I decided to build a simple trading bot that would follow a simple strategy and execute trades as I slept.

Many “bot traders” as they are called, use the Python programming language to execute these trades. If you were to google, “crypto trading bot,” you would find links to Python code in various Github repositories.

I’m a data scientist, and R is my main tool. I searched for a decent tutorial on using the R language to build a trading bot but found nothing. I was set on building my own package to interface with the GDAX API when I found the package rgdax, which is an R wrapper for the GDAX API. The following is a guide to piecing together a trading bot that you can use to build your own strategies.

The Strategy

In a nutshell, we will be trading the Ethereum — USD pair on the GDAX exchange through their API via the rgdax wrapper. I like trading this pair because Ethereum (ETH) is typically in a bullish stance, which allows this strategy to shine.

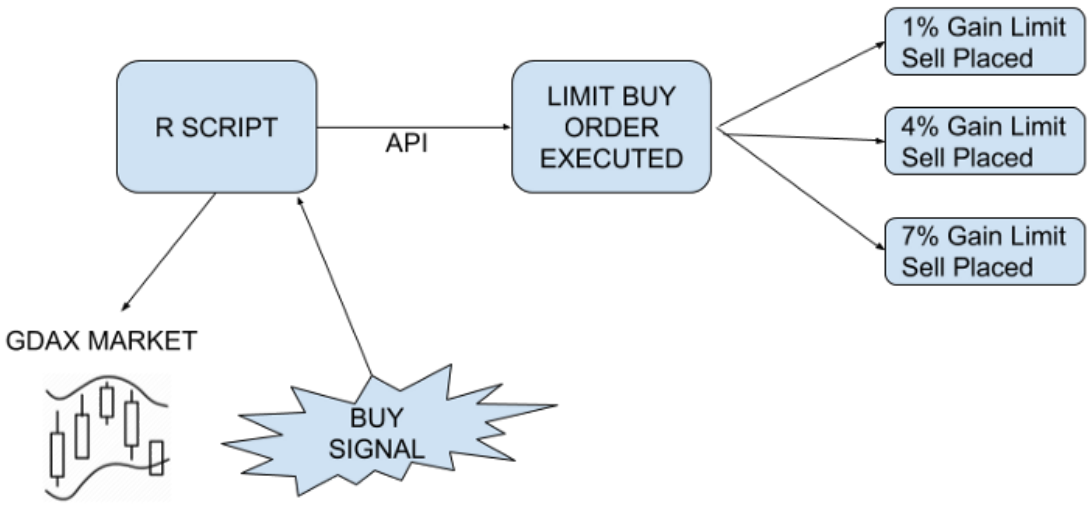
Note: this is a super-simplistic strat that will only make a few bucks in a bull market. For all intents and purposes, use this as a base for building your own strat.

We will be buying when a combination of Relative Strength Index (RSI) indicators point to a temporarily oversold market, with the assumption that the bulls will once again

Once we buy, the bot will enter three limit sell orders: one at 1% profit, another at 4% profit and the last at 7% profit. This allows us to quickly free up funds to enter another trade with the 1st two orders, and the 7% order bolsters our overall profitability.

Software

We will be using Rstudio and Windows task scheduler to execute our R code on a regular basis (every 10 minutes). You will need a GDAX account to send orders to, and a Gmail account to receive trade notifications.



Our Process

Part 1: Call Libraries and Build Functions

We will begin by calling several libraries:

```
8  # To separate entries in logfiles
```

```
library(curl)
library(xts)
library(TTR)
```

The package rgdax provides the interface to the GDAX api, mailR is used to send us email updates with a Gmail account, stringi helps us parse numbers from JSON and TTR allows us to perform technical indicator calculations.

Function: *curr_bal_usd* & *curr_bal_eth*

You will use your api key, secret and passphrase that are generated from GDAX in the API section. These functions query your GDAX account for the most recent balance which we will use repeatedly in our trading:

```
# Build functions
curr_bal_usd <- function(x){
  m <-  accounts(api.key = "your_api_key", secret = "your_secret", passphrase = "your_passphrase")#[3,3]
  m <-  subset(m$available, m$currency == 'USD')
  m
}
curr_bal_eth <- function(x){
  n <- accounts(api.key = "your_api_key", secret = "your_secret", passphrase = "your_passphrase")
  n <- subset(n$available, n$currency == 'ETH')
  n
}
```

Function: RSI

We will use the RSI or Relative Strength Index as our main indicators for this strategy. *Curr_rsi14_api* pulls in the value of the most recent 14 period RSI, using 15 minute candles. *RSI14_api_less_one* and so forth pull in the RSI for the periods prior:

```
curr_rsi14_api <- function(x){
  df <- rgdax::public_candles(product_id = "ETH-USD",
                              granularity = 900)
  rsi_gdax <- tail(TTR::RSI(df[,5],
                            n = 14),
                   n = 1)
  rsi_gdax
}
# v.2
rsi14_api_less_one <- function(x){
  df <- rgdax::public_candles(product_id = "ETH-USD",
                              granularity = 900)
  rsi_gdax_less_one <- head(tail(TTR::RSI(df[,5],
                                          n = 14),
                                 n = 2),n=1)
  rsi_gdax_less_one
```

```
  rsi_gdax_less_two <- head(tail(TTR::RSI(df[,5],
                                          n = 14),
                                 n = 3),n=1)
  rsi_gdax_less_two
}
rsi14_api_less_three <- function(x){
  df <- rgdax::public_candles(product_id = "ETH-USD",
                              granularity = 900)
  rsi_gdax_less_three <- head(tail(TTR::RSI(df[,5],
                                            n = 14),
                                   n = 4),n=1)
  rsi_gdax_less_three
}
rsi14_api_less_four <- function(x){
  df <- rgdax::public_candles(product_id = "ETH-USD",
                              granularity = 900)
  rsi_gdax_less_four <- head(tail(TTR::RSI(df[,5],
                                           n = 14),
                                  n = 5),n=1)
  rsi_gdax_less_four
}
```

Function: *bid & ask*

Next, we will need the current bid and ask prices for our strategy:

```
bid <- function(x){
  bid <- public_orderbook(product_id = "ETH-USD", level = 1)
  bid <- bid$bids[1]
  bid
}
ask <- function(x){
  ask <- public_orderbook(product_id = "ETH-USD", level = 1)
  ask <- ask$asks[1]
  ask
}
```

Function: *usd_hold, eth_hold* and *cancel_orders*

In order for us to place limit orders in an iterative fashion we need to be able to pull in the current status of our orders already placed, and be able to cancel orders that have moved too far down the order book to be filled. We will use the “holds” function of the rgdax package to do this for the former, and “cancel_order” for the latter:

```
usd_hold <- function(x){
  holds(currency = "USD", "your_api_key", "your_secret", "your_passphrase")
}
eth_hold <- function(x){
  holds <- holds(currency = "ETH", "your_api_key", "your_secret", "your_passphrase")
  holds
```

```
101  }
```

Function: *buy_exe*

This is the big-daddy function that actual executes our limit orders. There are several steps that this function works through.

> *1. Order_size function calculates how much eth we can buy, because we want to buy as much as possible each time, less 0.005 eth to account for rounding errors*
>
> *2. Our WHILE function places limit orders while we still have zero ETH.*
>
> *3. An order is added at the bid() price, the system sleeps 17 seconds to allow the order to be filled, and then checks to see if the order was filled. If it wasn't then the process repeats.*

Part 2: Store Variables

Next, we need to store some our our RSI indicator variables as objects so the trading loop runs faster and so that we don't exceed the rate limit of the API:

```
136  # Store variables so don't exceed rate limit of API
137  curr_rsi14_api <- curr_rsi14_api()
138  Sys.sleep(2)
139  rsi14_api_less_one <- rsi14_api_less_one()
140  Sys.sleep(2)
141  rsi14_api_less_two <- rsi14_api_less_two()
142  Sys.sleep(2)
143  rsi14_api_less_three <- rsi14_api_less_three()
144  Sys.sleep(2)
145  rsi14_api_less_four <- rsi14_api_less_four()
146
```

Part 3: Trading Loop Executes

Up until now, we have just been preparing our functions and variables in order to execute the trading loop. The following is a verbal walk through of the actual trading loop:

If the current balance of our account in USD is greater than $20, we will start the loop. Next, if the current RSI is greater than or equal to 30 AND the RSI in the previous period

Next, we save this buy price to a CSV file.

Then, we send an email to ourselves to alert us of the buy action.

The loop then prints “buy” so we can track that in our log file.

The system then sleeps for 3 seconds.

```
# Actual Trading Loop
if(curr_bal_usd() >= 20){    # if have more than $20 USD start loop
  if(curr_rsi14_api >= 30 &    # and current rsi >= 35  # v.2
     rsi14_api_less_one <= 30 &    # previous close RSI <= 35  # v.2
     rsi14_api_less_two < 30 | rsi14_api_less_three < 30 | rsi14_api_less_four < 30) {    # i-2, i-3 or i-4 RSI < 35  # v.2

    # 1 Buy
    buy_exe()

    Sys.sleep(180)

    # 2 save buy price in csv on desktop
    position <- write.csv(bid(), file = "C:/R_Directory/position.csv")

    # 3 send email

    send.mail(from = "your_username@gmail.com",
              to = c("your_username@gmail.com"),
              replyTo = c("Reply to someone else <your_username@gmail.com>"),
              subject = "GDAX ETH Test - Buy",
              body = paste("Your model says buy right now at price",bid()),
              smtp = list(host.name = "smtp.gmail.com", port = 465, user.name = "your_username", passwd = "your_password", ssl = TRUE),
              authenticate = TRUE,
              send = TRUE)
    # 4 print for logs
    print("buy")
    Sys.sleep(3)
    # v.5
    # 5 Enter tiered limit sell orders
```

Now, we enter 3 tiered limit sell orders to take profits.

Our first limit sell order takes profit at a 1% gain, the next takes profit at a 4% gain, and the last takes profit at a 7% gain:

```
    # 5 Enter tiered limit sell orders

    # Order 1: take 1/3 profits at 1% gain
    order_size_tiered3 <- round(curr_bal_eth()/3,3)
    order_price_tiered3 <- round(bid() * 1.01,2)
    Sys.sleep(1)
    add_order(product_id = "ETH-USD", api.key = "your_api_key", secret = "your_secret", passphrase = "your_passphrase",
              type="limit", price = order_price_tiered3, side = "s",  size = order_size_tiered3 )
    Sys.sleep(20)
    # Order 2: take 1/3 profits at 4% gain
    order_size_tiered5 <- round(curr_bal_eth()/2,3)
    order_price_tiered5 <- round(bid() * 1.04,2)
    add_order(product_id = "ETH-USD", api.key = "your_api_key", secret = "your_secret", passphrase = "your_passphrase",
              type="limit", price = order_price_tiered5, side = "s",  size = order_size_tiered5)
    Sys.sleep(20)
    # Order 3: take 1/3 profits at 7% gain
    order_size_tiered8 <- round(curr_bal_eth(),3)
    order_price_tiered8 <- round(bid() * 1.07,2)
    add_order(product_id = "ETH-USD", api.key = "your_api_key", secret = "your_secret", passphrase = "your_passphrase",
              type="limit", price = order_price_tiered8, side = "s",  size = order_size_tiered8 )
    # v.5
```

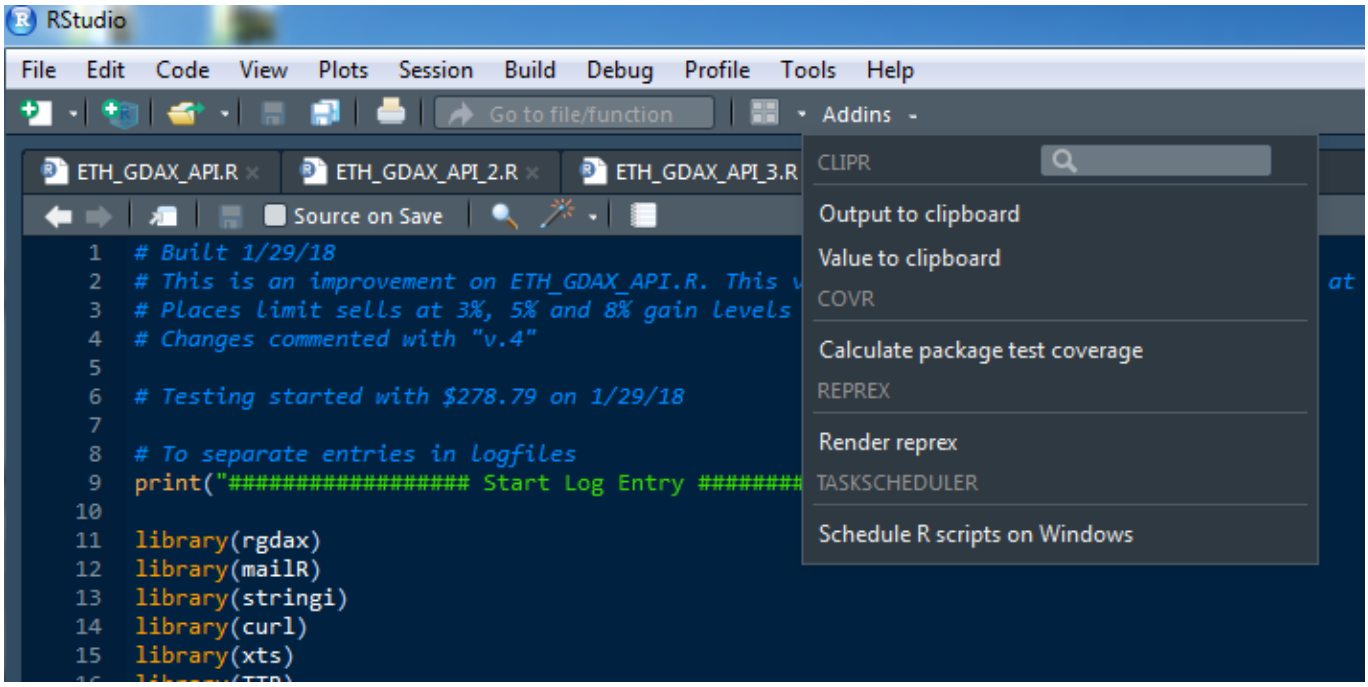
That's it, that's the entire script.

Part 4: Using Windows Task Scheduler to Automate the Script

The whole purpose of this bot is to take the human error out of the trade, and to allow us to enter trades without having to be present at a screen. We will use Windows Task Scheduler to accomplish this.

Schedule script with Rstudio addin

Use the handy Rstudio add in to easily schedule the script:



Modify the scheduled task with Task Scheduler

Navigate to the task created by the Rstudio add in and adjust the trigger to fire at the interval you wish. In my case I choose every 10 minutes indefinitely.

Keep an eye on your task with the log file

Every time your script runs it will make an entry in a text log file, which allows you to troubleshoot errors in your script:

You can see how the “START LOG ENTRY” and “END LOG ENTRY” print function comes in handy to separate our entries.

Make it Your Own

You can modify this script to make it as simple or as complex as you want. I’m working on improving this script with the addition of neural networks from the Keras module from Tensorflow for Rstudio. These neural networks add an exponentially more complex element to the script, but are incredibly powerful for finding hidden patterns in the data.

In addition, the TTR package provides us with a large number of financial functions and technical indicators that can be used to improve your model.

With all this being said, do not play with more money that you can afford to lose. The markets are not a game and you can and will lose your shirt.

Link to Full Source Code on Github