Jared Donnelly

Prof. Kevin Ryan

CS 110 – Final

13 December 2020

Web-Scrapping for Special-Purpose Acquisition Company (SPAC) Stock Buzz

**Section One – Overview**

On the internet exists a site which claims to be the front page of it all. This site, Reddit, is home to tens of thousands of communities, one of which we took to web-scrapping. r/SPACs is a community forum dedicated to following the market surrounding Special-Purpose Acquisition Companies, which are a unique set of IPOs who sole purpose is to buy existing private companies to force them to become public. The goal was to index the stock-tickers mentioned and the number of times they were mentioned as well as to see the performance of the “most popular” (most mentioned) SPACs on the stock market.

**Section Two – Target Audience**

The target audience of this program is investors. Investors can potentially find correlations between buzz and stock price, more buzz potentially indicating a well performing or surging stock. Since reddit is a free public platform it allows for a hybrid ground of sentiment and analyst analysis, since the most frequent posters are more knowledgeable on or invested in subjects (such as special-purpose acquisition companies) they fall into a nice middle ground of general public and semi-analysts.

Text

Description automatically generated**Section Three – Program Specifics**

The code starts by importing 5 different modules, including matplotlib, yfinance, and the Reddit API: praw.

Before I began coding, I outlined what the main goal was: an index of Tickers mentioned by the subreddit. To do this I made a class named *Ticker*, which recording the ticker itself, and stored the number of times it had been mentioned during the web-scrapping process. It also has a print line which I call at the end of the program to display all the Tickers referred to more than 5 times.

**Text

Description automatically generated**Next, I out lined my second desired output: a graph of the most mentioned Tickers ( I chose 5, but the body of the code in the *run* function allows for it to be scalable to any amount ) The block of code below (figure 2) displays the use of the module yfinance with stock data ranging from December of 2018 until the most recent data from yahoo finance (December of 2018 was chosen because that is the average cut-off time between IPO and merger date, at which point the stock is no longer a SPAC).

Text

Description automatically generated Then began the process of web-scrapping. Our goal here with *extract\_ticker* was to identify the ticker from the post. This is part of larger function called *search* but is broken apart because it is reference multiple times.

Using a for loop, we analyze the text from a predetermined starting point. Then we go through each character until we reach a break (i.e., spaces, commas, or non-alphabetical characters). Before finally returning the ticker (in uppercase) to the parent function *search*.

Text

Description automatically generated Now we have *search* which consists of two main sections: tickers mentioned with a ‘$’ and tickers mentioned without a ‘$’. Using an if statement we search for a ‘$’ in the text, which is a typical notation for stocks when writing. Then we go one beyond the index of the ‘$’ to begin searching for the stock ticker in the previous *extract\_ticker* function. Once the function identifies what it thinks to be a ticker it checks it against my “blacklist” of words which can easily trip of the program, such as COVID or YOLO. Once it passes the if-statement it adds the word to tic\_tac, which is a list of the occurrence of every single ticker mention before finally checking it against the existing ticker\_list to see if we have any record of the stock yet, if not, it is added to the list.

A similar process is carried out again for stocks which may not be mentioned with a leading ‘$’, using the re module to separate the string into a list of words (re allows for more than one type of character to separate such as spaces, commas, and parentheses). The function *search* does not return anything to where it is called in *run* since it is used to change mutable lists instead of immutable variables.

Text

Description automatically generated Now we begin looking at our final function *run*. This function begins by declaring three lists which will be used throughout the program and again, since lists are mutable, they do not have to be declared global.

Next, we see the use of the Reddit API praw, which has us access reddit to obtain the information we scan, I used my personal account for the project since I do not have a separate developers account. After accessing reddit we then select the subreddit we want to analyze and then select the first 1000 new posts from the subreddit to store in the variable new\_posts.

This for loop is what causes my computer the most grief. It creates two variables count and post both in reference to the enumerate of the new-posts lists. Then we call upon our *search* function, sending the ticker\_list and tic\_tac lists to analyze each title for the 1000 reddit posts we stored. As an insurance that the system is working properly, every 100 posts it goes through it gives us an update to show the number of posts its analyzed and the number of total ticker and unique ticker mentions it has identified.

Text

Description automatically generated Lastly, we have the integration of the graph and the class *Ticker*. Using the *Counter* function that was imported, I store the number of occurrences of each ticker and the ticker name in a list of lists. Each sub-list contains 2 elements, and the over-arching list can contain any amount.

Then we specify the graph we are making, this case we are graphing five stocks by selecting their tickers and storing them in out top\_tic list which will be called on in the final line of the *run* function to graph the returns of the stocks over time.

Next, we have our class integration, which creates a dynamic variable for each object in the class *Ticker* which I then store in a list. This class, as mentioned earlier, is made up of the ticker itself and the number of occurrences. And is subsequently printed to show all the \_\_str\_\_\_ line of the class *Ticker.*

Chart

Description automatically generatedText

Description automatically generatedHere are some sample outputs:

**Section Four – Challenges**

This project saw a numerous number of challenges. Resulting in an entire rewrite of the code starting at 9:00PM on 13 December 2020. The biggest challenge was the class integration as I struggled to find a way to refer to each object of an unknown quantity, as without being able to call upon them again it was essentially useless data. So, after reading through several GitHub and Stack Exchange examples I settled upon using a for loop with a list of lists after finally finding he *Counter* function. Many of my other attempts either had no explanation other than brute force code or include more foreign concepts to me such as dictionaries.

Other challenges included: learning the reddit API, how to use yfinance, and other class related issues.

**Section Five – Further Extensions**

In the future I hope to expand this program to encompass web-scrapping of more data including the body text, comments, and possible any included graphics. I had originally planned to include these, but due to time constraints I could not.

I would also love to integrate packages for sentiment analysis which would allow for this program to be more beneficial to investors as they can then determine if the buzz is to buy or if it is indicating a worry to sell.