Runtime Terror

Shahbab Ahmed
Amirul Miah

Github: https://github.com/riz734/SI206FinalProject

Alpha Vantage API

Necessary background about stocks: Stocks have a few different types of weekly price data, which includes low and high. Low price is the lowest price that a particular stock cost during a certain week, and similarly high is the highest price.

Note: My program uses data for a random sample of (# times you run Stock_final_api.py x 20) stocks. So the database, visualizations, and calculation file will most likely be somewhat different for instructors when they run my program compared to the files I submitted. But the conclusions reached by analyzing the data should be the same (with slight standard deviation).

Original Goals:

- Use the API from Alpha Vantage to gather data about the price of stocks(both high and low) from the first week of 2020 and the latest week of 2020 in order to compare the two.

Achieved Goals:

- Was able to utilize data to determine that the prices of most of the stocks reduced when compared from the first week of 2020 vs. the latest week of 2020. This could be due a myriad of factors, one of which may include the effect of the pandemic.
- I found that there is a correlation between net high prices (latest week - first week of 2020) and net low prices (latest week - first week of 2020) of most stocks

Problems:

- Alpha Vantage has a cool down time for non-premium users (5 stocks per minute, 500 per day), which made fetching data very slow since one of their required parameter is a stock symbol, which meant I'd have to fetch data for one stock at a time
- Only a handful of the full list of stocks I had available existed within Alpha Vantage, which made fetching data even more difficult. And Alpha Vantage does not provide the public with a full list of stocks that they do have available.
- According to users on stackoverflow, Alpha Vantage is also very slow compared to other APIs such as IEX. I did not realize this until I was done with the creating and storing data process because I only used around 50 stocks at a time for testing purposes, so I did not realize how slow Alpha Vantage was until I put in the full list of 100s of stocks.

Visualizations: Attached png files to report

- Scatterplot: This plot shows that there is a positive correlation between net high price and net low price in relation to the change in price (from the latest week of 2020 and first week of 2020)
- Two Pie Charts: The pie charts are used to support the correlation created using the scatterplot. Shows that the # of net high counts vs # of net low price counts are similar, indicating a correlation.

Instructions:

1) Go to https://www.alphavantage.co/support/#api-key to create an API Key. Copy and paste this API into the prompt each time you run Stock_final_api.py. If any problems occur with entering the API Key, please change the API_KEY variable to just your API key as a string (although I haven't had any problems so it should run okay).
2) Have the following files in one folder: Stock_final_api.py, stock_api_output.py, and nasdaqlisted.txt
3) Cd into that folder (i had to do this multiple times on some full test runs so if any problems with directories/file locations occur please try this step again. Please also do this if stock_data.json seem to be overriding the data on each run, but I added code to avoid this problem so it should not occur)
4) Run Stock_final_api.py 5 times, each run may take some time due to Alpha Vantage's issues I stated previously
5) Run stock_api_output.py

Code Documentation:

Stock_final_api.py file

- Read_cache: Takes in a file name. First, it changes the directory to the current directory that the file exists. Then it reads from the JSON cache file and returns a dictionary from the cache data. If the file doesn't exist, it returns an empty dictionary.
- Write_cache: Takes in a file name, and a dictionary. This function encodes the cache dictionary (CACHE_DICT) into JSON format and writes the JSON to the cache file (CACHE_FNAME) to save the search results.
- Create_request_url: Takes in a stock symbol as an input, creates the proper URL for an API request on Alpha Vantage and returns the URL as a string
- Get_weekly_price_data: It takes in a random sample of stock symbols as a list, and a file name. Uses read_cache to either create a new dictionary called new_dict or open the cached dictionary. Loops through the list of symbols, if the symbol is not in the dictionary, then it creates the url for fetching the data for that stock and assigns it to request_url. Then it tries to fetch the data and load it into the variable data. The next few lines check to see if "Weekly Time Series" is a key (required to fetch correct data), if so then it loops through the values (which are dates of every week for the past decade +). If the year is 2020, then it grabs the data and adds it to the new_dict in the following format: {symbol: highest price of stock for the latest week of 2020, highest price of stock for the first week of 2020, lowest price of stock for the latest week of 2020, lowest price of stock for the first week of 2020). Then it calls write_cache to add the new data to the

saved json file. Once the number of stocks saved reaches 20, then it breaks out of the loop and you must restart to get another 20. It returns the dictionary that is saved.

- Create_db_high: It takes in a dictionary and name for the database. It opens a connection from python to sql. Then it creates a table in SQL called Stocks_High (Symbol, LatestWeekPrice, OldestWeekPrice) if it doesn't exist. Then it selects all symbols from Stocks_High, and assigns all of them to rows. Then it loops through rows (where each element is a tuple) and appends the symbol (if it doesn't already exist) as a string to the list rows_lst. Then it loops through rows_lst, if the symbol does not exist, then it inserts the following data into Stocks_High: Symbol, highest price of stock for the latest week of 2020, highest price of stock for the first week of 2020. I used rows_lst to prevent duplicates. Nothing is returned. Final line closes sql connection.
- Create_db_low: It takes in the same dictionary and name for the database as create_high_db. It does the exact same thing as create_db_high for creating the second table, with the difference being that it stores the lowest price of stock for the latest week of 2020, lowest price of stock for the first week of 2020 instead. Nothing is returned. The final line closes the connection to SQL.
- Create_symbols: Changes directory to where file is. Takes in file name as parameter (string). Opens the file, saves content into lines, closes the file. Randomizes the list in order to avoid going through stock data alphabetically and instead get a randomized sample of stocks. Loops through lines, splits it into proper format to get the symbols as strings, appends the symbols to list called symbols. Returns the list of symbols.

stock_api_output.py file

- Join_database: Takes in a string. Changes path directory to correct path. Creates a connection to sql and creates a database, using the passed string as the name (unless a database with the same name already exists then it opens that db instead). Creates a cursor called cur, selects and joins Stocks_High and Stocks_Low using Symbol. Returns a list of tuples with (Symbol, Latest High Price, Oldest High Price, Symbol, Latest Low Price, Oldest Low Price).
- Calculate_net_price: Takes in a list of tuples, creates a new lst, loops through passed list, appends to new list in the the following format: [(Symbol, latest high stock price - oldest high stock price, latest low stock price - oldest low stock price)]. Returns the new list of tuples.
- Write_csv: Takes in a list of tuples, and a string name for the file. Changes directory to correct path. Opens/Creates a csv file using the passed string name, writes headers, loops through passed in list of tuples, writes value of symbol under the "Symbol" row, value of net high price under "Net High Price" row, and value of net low price under "Net Low Price" row. Nothing is returned.
- Create_scatterplot_high_low: Takes in a list of tuples Creates a scatterplot using net low prices as the y-axis and net high prices as the x-axis. Nothing is returned.
- Create_pie_chart: Takes in a list of tuples, and a string. The passed string is used to decide which net prices to create a pie chart for: "high" or "low". Creates a pie chart using net negative counts (# of stocks with net price < 0), net positive counts (# of stocks

with net price > 0), and net equal counts (# of stocks with net price = 0) for either net high prices or net low prices. Nothing is returned.

Documentations:

- Alpha Vantage API: https://www.alphavantage.co/documentation/
- Stock list obtained from: ftp://ftp.nasdaqtrader.com/SymbolDirectory (file called nasdaqlisted.txt)

Webpage: 100 Most Popular Stocks
(https://robinhood.com/collections/100-most-popular)

Original Goals:

- Use the data from the web page to create a database that tracks only specific information such as the name, symbol, price, daily performance and ratings
- Use that information to figure out how many popular stocks are affordable for college students like me that don't have a lot of money to invest. I do this by dividing up the stocks by price ranges and use a pie chart to visualize the distribution
- Use historical data to see the overall performance of the stocks over a period of time to assess each individual stock's stability

Achieved Goals:

- I was able to collect the necessary information from the live webpage and use the collected information to determine how many of the stocks can be afforded by me. This distribution is displayed by the pie chart that is created by running the output file

Problems:

- In the beginning, it was very difficult to index into the specific source code within the inspect element tool which would give me the data I was looking for. The class and attribute names were not very comprehensive and the code was overall very difficult to make sense of. I managed to find the lines of code I needed after running the blocks of code through a beautifier
- I was unable to get the historical data because I was unable to figure out which method I should use to get the historical data of each stock. I was considering using a webscraper extension that has a feature to get old copies of a webpage, but I did not know how to utilize this tool to its fullest potential

Visualization:

- The visualization I decided to go with is a simple pie chart. The pie chart makes it easy to comprehend how many of the stocks are affordable and how many aren't

Instructions:

- Have these files in one directory: final_project_website.py and Website_output.py
- Cd into the directory containing these files
- Run final_project_website.py 5 times
- Run Website_output.py

Code Documentation:

- setUp_db: Takes in a database name and table name as inputs. Creates a table in the database if it doesn't exist. If it does exist, create a list of symbols that are already in the table so that when new data is added, there are no duplicates. Limits newly added data to the database to 20 per run.
- robinhood_scraper: Takes in a url as an input. Reads the html code of the webpage and indexes into the block of code that contains the information of the stocks on the page. It then finds the data that has information on the stocks' name, symbol, price, daily performance, and rating and add it all to a dictionary.
- stock_lst: Takes in the name of a database and table as inputs. From the table, it only retrieves the symbol and price which are appended to a list as a tuple. Convert the price to a float to be able to use it in calculations in the future. Also, round the float to two decimal points.
- get_price_ranges: Takes in a list of tuples as input. Use the price value of a stock (using the list of tuples returned by stock_lst) to determine if the stock is "Low-Price", "Medium-Price", "Expensive", or "Uber-Expensive". Then add that to the tuple of each stock. Then return the new tuple list.
- write_csv: Takes in a list of tuples as input. Create and write into a csv file that has the headers "Symbol, Price, Affordability". Then iterate through the given list and write the symbol, price, and affordability element of each stock to the csv file
- create_pie_chart: Takes in a list of tuples as input. Uses the tuples in the list to create a dictionary that keeps count of the number of "Low-Price" (< $15), "Medium-Price" (<$50), "Expensive" (< $100), and "Uber-Expensive" (> $100) stocks there are. Then use that information to use matplotlib to create a pie chart that represents these distributions. Explode the "Low-Price" and "Medium-Price" proportions to highlight the stocks I may be interested in. Save and show the pie chart img.

Documentations & Resources:

- Robinhood web page: https://robinhood.com/collections/100-most-popular
  - The web page where I retrieved the data from
- Beautiful soup documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
  - Used this as reference for collecting the proper tags and attributes
- Html beautifier: https://www.freeformatter.com/html-formatter.html

- I used this to help me read the messy html that appears using the inspect element tool