Final Project Report

SI 206: Data-Oriented Programming

Ross Pythons: Anuar Baisynov, Tiffany Guo, Chenlin Zhang

Github Repository: https://github.com/zchenlin/Final-Project

Final Project Description

The outbreak of COVID-19 and the subsequent containment measures have brought the economic activity to a near-standstill, completely shutting down major sectors of the economy and leading to mass lay-offs. Therefore, in our final project, we decided to investigate the economic impact of the COVID-19 pandemic in the United States. To evaluate changes on both individual and corporate levels, we analyzed two major indices - the number of weekly initial unemployment claims and the profitability of large-capitalization enterprises in S&P 500 - using them as proxies to the overall state of the US economy. Overall, in spite of the prompt monetary and fiscal response measures, the pandemic has had a severe effect on the economy, leading to record-breaking figures in unemployment filings and a considerable decrease in companies' profitability ratios.

Original Project Goals

As set forth in our project proposal, initially, our team planned to study the entertainment industry, hypothesizing that there would be a correlation between people's preferred music/video genres and the average income in the area where they live. To test this hypothesis, we intended to harvest data from the Spotify website as well as Youtube and US Census Bureau API. With such a project scope, we set the following topic-specific objectives for the three main elements of the assignment:

- **Source.** We wanted to collect data from three main sources: Spotify website, Youtube API, and the US Census Bureau API. Through the first two sources, we planned to access data on people's music and video preferences, and the third source would help us see income levels in different states or even counties.
- **Analyze.** With the collected data, we intended to investigate whether there was a relationship between what people listened or watched and their socio-economic status. Although it was an ambitious goal, we did not really have a plan for how to approach this study and hoped to determine that later.
- **Visualize.** Finally, we wanted to create three visualizations. Two of them would be maps of Michigan (or of the United States) detailing income levels as well as the most popular music/video genres across counties (or states). For the third visualization, we wanted to create a regression model connecting the amount of time users listened to a specific song

on Spotify and the amount of time users watched the official video with that song on Youtube.

Since we did not have a clear strategy for tackling several components of this project, we understood that it was quite likely that we would need to change the topic at some point. Due to this uncertainty, we also decided to set more generic objectives for our work:

- Collect data from three sources
- Create three visualizations
- Use three different types of visualizations
- Demonstrate our project on December 7
- Explore one topic outside the course scope
- Attempt one extra-credit task
- Avoid any point deductions
- Complete the report by the deadline

Achieved Goals

After multiple unfruitful attempts to extract data from the sources specified in our proposal, we came to an unpleasant realization that our intended project could be unfeasible at that point. Overcoming intense frustration, we then decided to switch gears and refocus our efforts. This is how we started our study on the economic aftermath of the virus outbreak. As we reflect on our work, we look back at the objectives that we set for ourselves early on.

Original Goals	Status
Collect data from three sources	Achieved
Create three visualizations	Overachieved
Use three different types of visualizations	Achieved
Demonstrate our project on December 7	Not Achieved
Explore one topic outside the course scope	Achieved
Attempt one extra-credit task	Overachieved
Avoid any point deductions	Not Achieved
Complete the report by the deadline	Achieved

Diving into details, we managed to achieve all essential goals in each element of the assignment:

- **Source.** We collected data on the epidemiological situation in the country from the Covid Tracking Project API, unemployment figures - from the Department of Labor website, and S&P 500 financials - from the Financial Modelling Prep API. In other words, we collected our data from two API and one website, thus improving our understanding of JSON as well as BeautifulSoup and learning the fundamentals of Selenium.
- **Analyze.** From the COVID data, we calculated total positive COVID cases and hospitalizations per month in the United States using the sum of daily increases in cases and hospitalizations. Total positive cases per month increased progressively through the year, with an enormous spike in November and hospitalizations spiking in April and November. Next, using the Department of Labor data, we found the average number of weekly initial unemployment claims for each year since 1967. The weekly mean in 2020 turned out to be many times higher than in any other year, emphasizing the difference between the pandemic-triggered recession and economic downturns of the past. Finally, using the data from Financial Modelling Prep, we calculated changes in companies' gross profit ratios in S&P 500. The result shows that from March to June, nearly 45% of the companies suffered a 0% to 10% decline in gross profit ratios, while more than 50% of the companies experienced a 0% to 10% increase from June to September.
- **Visualize.** Finally, we created six visualizations in total, two for each source:
    - COVID data: two bar charts illustrating the total number of positive COVID-19 cases per month in the United States, as well as the total number of hospitalizations per month in the United States.
    - Unemployment data: a line graph showing the number of unemployment filings and a bar chart comparing the average number of claims for specified years.
    - S&P 500 data: two histograms representing the percent changes in companies' gross profit ratios from March to June and from June to September.

Implementation Problems

The assignment was challenging, and we realized that when we started implementing our original proposal. As we tried to collect data from Spotify, Youtube, and the Census Bureau, we

quickly discovered that these sources, in fact, could not provide us with what we needed. When we decided to change the topic, we experienced similar problems with many other sources, including Uber, Lyft, and Booking API. After a week-long research, we eventually agreed on what sources we would use and what data we would collect. However, that obviously was not the only problem we faced.

- **COVID problems.** The main challenge encountered when implementing the COVID API was finding a way to loop through dates for the request URL's formatting instead of hardcoding date parameters. We researched and discovered the datetime module - which, when imported, would help us loop through dates. However, it was challenging to understand the unique formatting and syntax that came with using datetime objects, as we needed to understand how object types were changed and reformatted back and forth. Another challenge was developing a technique to incorporate the 25 data insertion limit. We discovered that using while and break was the most elegant solution. We also kept in mind that we didn't want to overload requests to the API. Therefore, by checking to see that the database already had the requested date and the data associated with that date before proceeding to call the API, we would avoid overloading the server, and also retrieve 25 data sets from the API at most each time we executed the code.
- **Unemployment problems.** Originally, we wanted to get the unemployment data from the Bureau of Labor Statistics API. However, that approach entailed multiple issues. First, although we managed to decipher the complicated documentation of the API, we soon discovered that the data on weekly unemployment claims had not been updated for a while. We thought we could use the unemployment rate (which was updated more recently) as an alternative, but, as it turned out, the Bureau posted these data only on the monthly basis, leaving us with very few data points for each year. After extensive research, we switched to the Department of Labor website which offered weekly unemployment figures since 1967. However, we then came across another issue: the webpage could not be accessed simply with BeautifulSoup because we first needed to submit a request form specifying the parameters such as start year and end year for the requested data. To automate this process, we needed to install ChromeDriver and learn how to use Selenium with the help of Stack Overflow and other online guides.

- **S&P 500 problems.** Initially, we tried to look at the retail industry and combine the data with COVID and unemployment figures to analyze how the pandemic impacted the retail industry. However, we found it hard to find an API which could provide us with data about the retail companies in uniform metrics. Therefore, we enlarged our range to the whole US economy, as we found an API to extract the data of public companies' financial statements. The first set of data we collected from the Financial Modelling Prep API was companies' quarter revenues, gross profits, and gross profit ratios. However, we found that when we stored these data in the SQL database, the data structure appeared to be confusing, and it was hard to make some meaningful calculations and visualizations. Therefore, we decided to focus on the gross profit ratios, since their expressions are uniform. That said, we extracted the data of companies' gross profit ratios in three quarters in 2020 in S&P 500, an index which has approximately 500 companies from different industries to represent the U.S economy.

Calculation Files

Calculation File 1. COVID-19 data - this is what the JSON output file (calculations.txt) looks like when dates from March 1, 2020 until December 6, 2020 are calculated into total positive cases and total hospitalizations per month.

[{"month": "03", "Total Positive Cases": 197906, "Total Hospitalizations": 23796}, {"month": "04", "Total Positive Cases": 875821, "Total Hospitalizations": 98709}, {"month": "05", "Total Positive Cases": 717466, "Total Hospitalizations": 85340}, {"month": "06", "Total Positive Cases": 832778, "Total Hospitalizations": 31925}, {"month": "07", "Total Positive Cases": 1898079, "Total Hospitalizations": 63111}, {"month": "08", "Total Positive Cases": 1452945, "Total Hospitalizations": 61168}, {"month": "09", "Total Positive Cases": 1190470, "Total Hospitalizations": 37445}, {"month": "10", "Total Positive Cases": 1883228, "Total Hospitalizations": 65702}, {"month": "11", "Total Positive Cases": 4289896, "Total Hospitalizations": 92675}, {"month": "12", "Total Positive Cases": 1198265, "Total Hospitalizations": 26001}]

Calculation File 2. Unemployment data - this is what the output file (calcUnemployment.csv) looks like after the fifth execution of the code (all data collected).

Year	Average Number of Claims per Week

2020	1447815
2019	217557
2018	220894
2017	243116
2016	263900
2015	276956
2014	306567
2013	341826
2012	372226
2011	409110
2010	455692
2009	566577
2008	415495
2007	321151
2006	311343
2005	333621
2004	341269
2003	399129
2002	402790
2001	402581
2000	303725
1999	298922
1998	317074
1997	321511
1996	351361
1995	357041
1994	342730
1993	342308
1992	407343
1991	447593
1990	387001
1989	328819
1988	312882
1987	326152
1986	376226
1985	390294
1984	371773
1983	441652

1982	579251
1981	450846
1980	479769
1979	384115
1978	342826
1977	375962
1976	381269
1975	473846
1974	355788
1973	244692
1972	262754
1971	293942
1970	292346
1969	196903
1968	198134
1967	225634

Calculation File 3. S&P 500 data - this is what the first fifty lines of the output file (calculations_revenueDiff.csv) look like after all data is collected.

Difference between Quarter 1 and Quarter 2	**Difference between Quarter 2 and Quarter 3**
-0.00362	0.001609
-0.01105	0.02831
-0.00816	0.000645
-0.00989	0.004157
-0.02326	0.027105
-0.02326	0.027105
-0.0012	0.028093
0.001077	0.032017
-0.00206	-0.00048
-0.00051	0.006965
-0.07342	-0.00126
-0.06219	0.037858
0.031477	-0.00835
0.003107	-0.00514

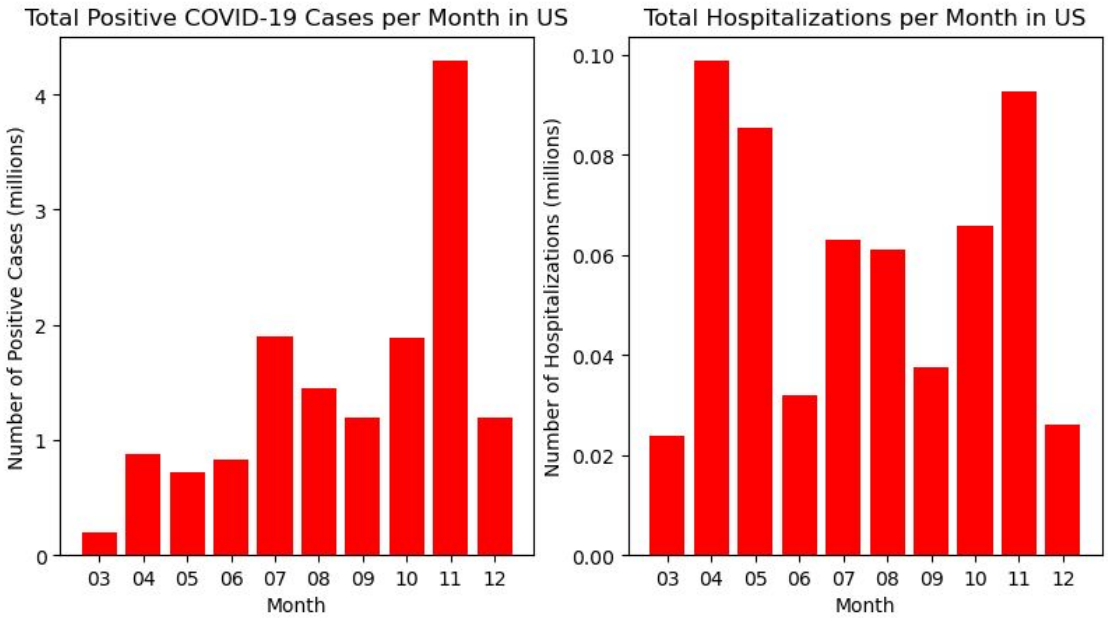
0.034882	0.011567
-0.00857	-0.32771
0.013558	0.00027
-0.01587	0.013214
0.157001	0.000535
0.005908	0.039137
-0.01628	0.00329
0.015526	0.024507
0.036131	0.008502
0.012353	0.000654
-0.00321	-0.00631
-0.02947	0.02022
-0.13066	-0.03544
-0.02061	-0.00271
0.003246	-0.00273
0.04841	0.020056
0.005809	-0.00326
0.006248	-0.00264
0.019543	-0.00351
-0.01484	-0.0088
-0.03357	0.073209
0.103955	-0.03101
-0.00863	0.027088
-0.00195	0.015638
0.015576	-0.00014
-0.06997	0.074795
0.020524	0.10092
0.005623	-0.02396
0.008183	0.00315
0.01187	-0.01209

0.029196	4.27E-05
0.009752	-0.33478
-0.05173	0.015237
0.015279	0.040131
-0.03223	0.0334
-0.10746	0.17231

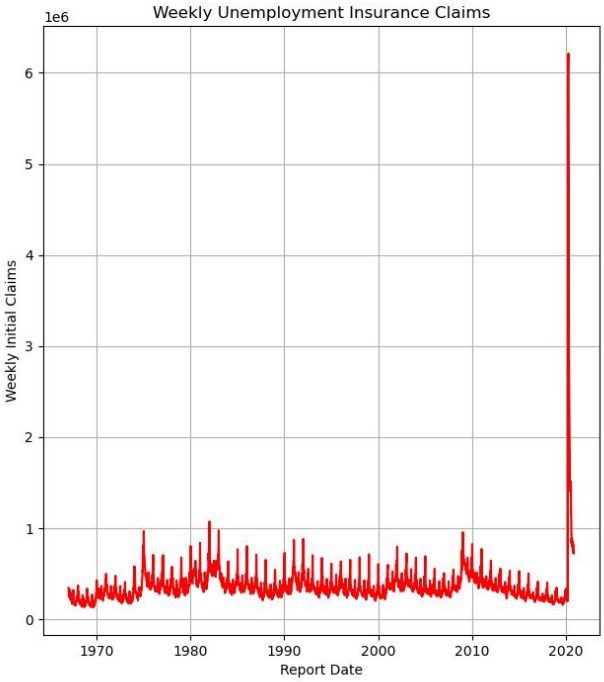
……

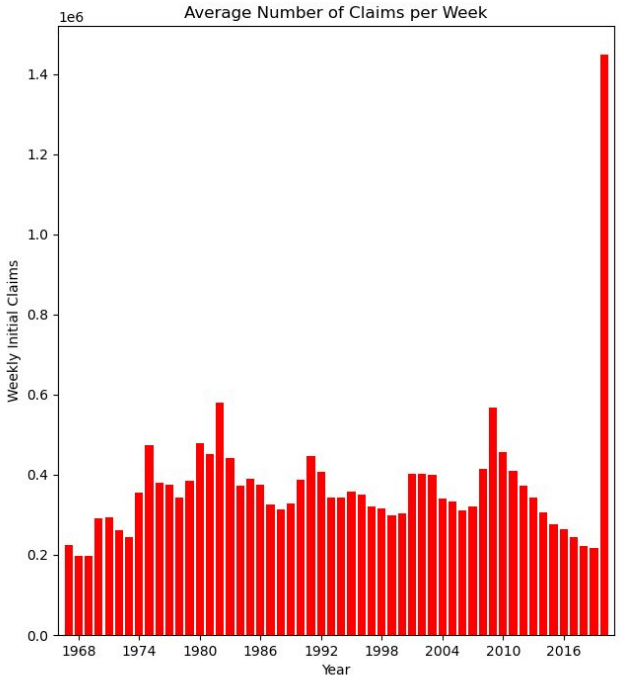
Visualizations

Visualizations 1 and 2. COVID Data (all data collected).

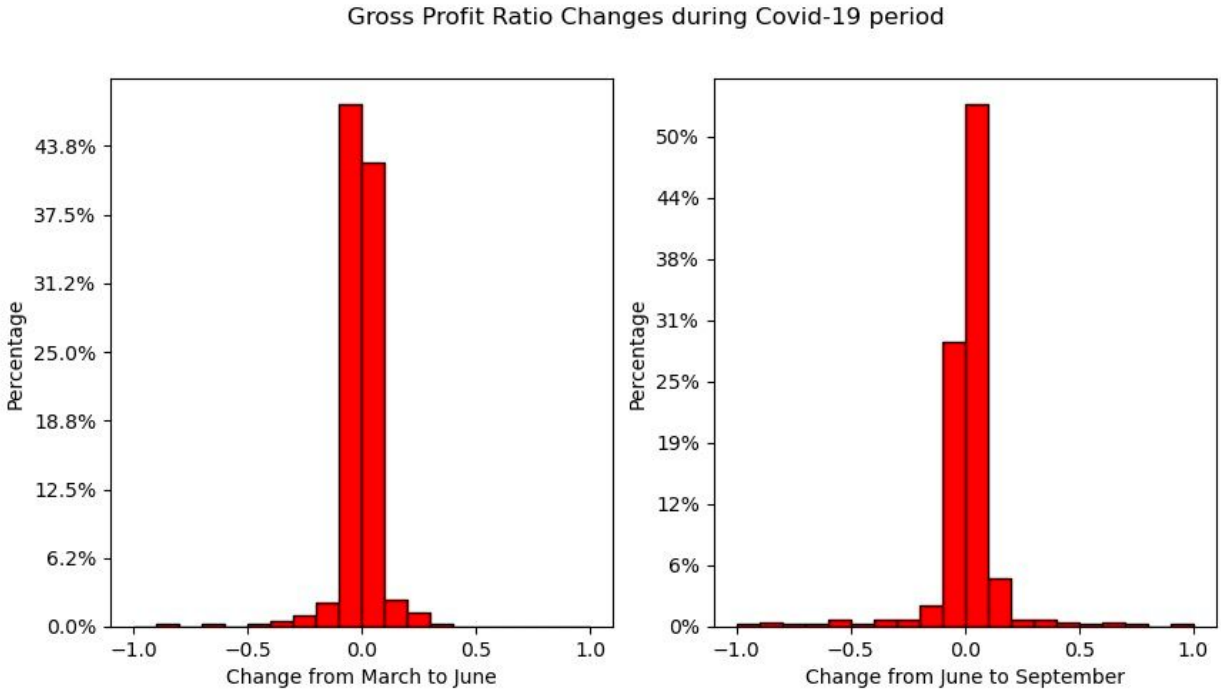


Visualizations 3 and 4. Unemployment Data (all data collected).





Visualizations 5 and 6. S&P 500 Data (all data collected).



Instructions for Code Execution

- To run **covid.py**, run the file in the terminal. At the first execution, the terminal will output 25 dates with corresponding data from the API. A pop up will automatically appear with two generated visualizations using the first 25 dates. Open the generated database, finalproj.db, and there will be two tables - Covid and Hospitalized - with data loaded into them. Run the code 3 more times to generate 100 data points total. After these executions, one can run code for all the remaining dates until December 6, 2020 at once by commenting out lines 176 and 177 (removing the 25 data insertion limit). If one does not comment out these two lines, the code will continue to output and insert 25 data points at each execution. Run the code with the commented out lines, and the resultant visualizations will be complete and the database will be fully populated with dates until December 6, 2020.
- Since the code in **dol.py** uses Selenium, one needs to install a browser-specific Driver (e.g., ChromeDriver) to run it. Otherwise, the code will most likely return an error. With Driver installed, running the code is quite simple. First, a browser window controlled by the automated algorithm should immediately open. It first displays a page for setting parameters to request data from the Department of Labor website. The code automatically sets the necessary parameters and goes to the next page. Then, it may take between 10 seconds and 2 minutes until the browser loads all the data and closes the browser window. After that, assuming all the required packages are installed, a window with two visualizations appears. Each of the first four executions of the code only collects 25 items, gradually incrementing the data points used in the visualizations. On the fifth execution of the code, all the remaining data are collected from the source and used in the visualizations.
- To run the **sp500.py**, run the code in the terminal, and it will take approximately 10 seconds to insert 25 rows of data to the SQL database. At the same time, a visualization of the current data in the database will appear and the calculation of the data inserted will be written into the file. After doing this for four times, which will result in 100 rows in the database, the next execution will insert the rest of the data in one time, and visualize the full picture of the data.

Code Documentation

Code Documentation 1. COVID Data - covid.py

Function Name	Docstring Explanation
create_covid_table	'''This is a function that creates a table in the database titled 'Covid' with columns Date, Positives, PositiveInc, Negatives, and Deaths - all type INTEGER. It takes a database cursor and connector as parameters and returns them.'''
create_hospitalized_table	'''This is a function that creates a table in the database titled 'Hospitalized' with columns Date, Hospitalized, HospitalizedInc, ICU, and Ventilator - all type INTEGER. It takes a database cursor and connector as parameters and returns them.'''
create_request_url	'''This is a function that creates the API request URL. The API url is formatted in 'https://api.covidtracking.com/v1/us/{}.json' where the date changes for each call. It takes a date (which must be in the integer format eg. 20200311) as a parameter and returns a formatted url.'''
get_add_data	'''This is a function that retrieves the data from the API and JSON and adds the data to the database finalproj.db. It takes a database cursor and connector as well as a date as parameters. It calls create_request_url with the date parameter and sets it to a variable url. Then, it checks if the date exists in the database already to account for duplicates in the database before even grabbing data from API by using a SELECT statement with the date. If that SELECT statement fetches something, then the data already exists in the database, so the function returns 0. Else, it uses a try-and-except statement to retrieve data from the API. The try succeeds when the API JSON data are retrieved and loaded into json_results. The exception happens when there is an error grabbing data from the API, the function and returns None. If we are still within the function (meaning the data retrieval succeeds) then we set the data's variables to our variables. Then, the function inserts the data into the Covid table and returns 1.'''
jointables	'''This function joins the Covid and Hospitalized tables together, and performs sum calculations on the increases in COVID-19 cases and increases in hospitalizations. It takes a database cursor and connector as parameters and returns data on these calculations.'''
createcalcfile	'''This function creates a file from the passed data made from the

	jointables function, called 'calculations'. It does not return anything.'"
visualization	'"This is a function that uses matplotlib to create two visualizations: a bar graph of total COVID-19 cases per month in the US; and a bar graph of total hospitalizations per month in the US. First, it creates lists for months, positive cases, and hospitalizations using the passed data from the calculated data in the database. Then, it initializes two plots, passes through lists in corresponding plot axes, and labels title and axes. The function does not return anything.'"

Code Documentation 2. Unemployment Data - dol.py

Function Name	Docstring Explanation
get_data	'"The function extracts data on weekly initial unemployment claims since 1967 from the Department of Labor website. It requires a database cursor and connector as well as a url as parameters. The function uses Selenium and ChromeDriver to automatically set parameters for the request (namely, the start year) and submit the request form in order to access the webpage with the data. It then reads values from the webpage starting with the most recent ones and stores them in the unemployment_dict dictionary. That dictionary is ultimately returned by the function. The first four executions of the code are limited to just 25 values, and the fifth execution extracts all the remaining values from the webpage at once.'"
create_unemployment_table	'"The function takes a database cursor and connector as parameters and creates a table where unemployment data is stored. It does not return anything.'"
check_unemployment_table	'"The function takes a database cursor and connector as well as a date as parameters. It checks whether the specified date already exists in the database. It returns True if the date is in the database and False otherwise.'"
check_table_size	'"The function takes a database cursor and connector as parameters. It checks if the table has reached 100 items in which case it returns True. Otherwise, the function returns False.'"
populate_unemployment_table	'"The function takes a database cursor and connector as well as a dictionary as parameters. It then fills the Unemployment table in the database with values from the dictionary. It does not return anything.'"

write_calculations	'"The function takes a database cursor and connector as parameters. First, it SELECTs data from the specified database table and calculates the total number of items as well as the average of the values in different years. Then, the function creates a file (calcUnemployment) in the same directory and writes the results of the calculations (averages of values in different years).'"
visualization	'"The function takes a database cursor and connector as parameters. First, it SELECTs data from the specified database table and calculates the total number of items as well as the average of the values in different years. The function then creates two visualizations: a line graph depicting weekly unemployment claims and a bar chart showing the average number of claims per week in the years for which data has been collected.'"

Code Documentation 3. S&P 500 Data - sp500.py

Function Name	Docstring Explanation
request_url	'"This function formats and returns the url which helps extract the information about companies' revenues. It takes a company name as a parameter and returns a formatted url'"
get_etf_company	'"This function formats and returns a list of companies in the etf index. It takes an ETF index as a parameter.'"
setupDatabase	'"This function directs the path file and sets up the database for SQL. It takes a filename as a parameter and returns a database cursor and connector.'"
setupIncomeDataTable	'"This function first creates an IncomeData table, which is the table storing all the information. Next, it extracts information and inserts it into the table, 25 companies at a time. It takes a database cursor and connector as well as a list of companies as parameters and does not return anything.'"
dataCalculation	'"When the database has more than 100 data rows, this function extracts some of the data from the database and calculates the difference between two quarters for each company. Then, it writes the results to a file. It takes a database cursor and connector as parameters.'"
visualization	'"This function visualizes the results in two histograms. It takes two lists as parameters and does not return anything.'"

Resource Documentation

Date	Issue Description	Resource Location	Result
COVID Data			
11/26	Way to check if data existed in a database before adding it to database	Link (Stack Overflow)	Resolved
11/26	How to create a table only if it did not exist in the first place so that I wouldn't create duplicate tables with each code run.	Link (Stack Overflow)	Resolved
11/26	How to use JSON loads and dumps	Link (Stack Overflow)	Resolved
11/26	How to use the basics of the datetime module	Link (Python.org)	Resolved
12/1	How to format datetime objects	Link (Programiz)	Resolved
12/2	How to insert into database x number of data points at a time	Link (Stack Overflow)	Unresolved (didn't seem very useful in the context of my file)
12/2	How to use break and continue to limit the database insertion to 25 items at a time	Link (Programiz)	Resolved (used while instead of continue)
Unemployment Data			
12/4	What parameters to specify in the request url for the Bureau of Labor Statistics API	Link 1 (Blog) Link 2 (BLS)	Resolved (not used in the final version)
12/4	How to convert a string variable to the date format in Python	Link (Stack Overflow)	Resolved
12/5	How to select an item in a drop-down menu on a webpage with Python	Link (All Things Selenium)	Resolved
12/5	How to click a Submit button on a webpage with Python	Link (Stack Overflow)	Resolved
12/5	How to install Selenium and ChromeDriver	Link (Blog)	Resolved

12/6	How to set limits for axes labels in matplotlib	Link (matplotlib)	Resolved
12/6	How to prevent scientific notation (offset) in matplotlib	Link (Stack Overflow)	Resolved
S&P 500 Data			
12/1	How to fetch data from SQL database	Link (Microsoft)	Resolved
12/5	How to make histograms in Python	Link (Real Python)	Resolved
12/5	How to make histograms in Python	Link (Matplotlib)	Resolved
12/6	How to adjust the y-axis of a histogram to percentage expression	Link (Stack Overflow)	Resolved
12/6	How to adjust the y-axis of a histogram to percentage expression	Link (Stack Overflow)	Resolved