SI 206 Final Project Report

Benjamin Jacobson, Hudson Shapiro, and Jake Hirschberg

Github Link: https://github.com/benjacobson36/Final-Project

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather

- The initial goals for the project were to use the Polygon.io API to retrieve information regarding the historical performance of various stock tickers, the SEC EDGAR API to retrieve information regarding the historical finances of those same companies, and the Federal Reserve FRED API to gather historical macroeconomic information. We initially figured we would collect data for some ten-year duration, likely 2013–2022, and get data on a quarterly basis for the financial and macroeconomic data while obtaining monthly data for the performance of the stock tickers. We hoped to gather data as to the closing price of the stocks averaged monthly, which the API expressed that it offered, financial metrics such as assets, accounts receivable, accounts payable, and inventory on a quarterly basis, and macroeconomic metrics such as GDP, unemployment rate, federal funds rate, and CPI on a quarterly time horizon.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather

- We ultimately decided to pivot away from using the Polygon.io API for the historical stock data and used an API from Alpha Vantage instead. We were able to collect monthly historical data, including high, low, open, close, and a column for stock tickers dating back to 1999 by using this resource from Alpha Vantage. The JSON responses were significantly easier to work with, and none of the information remained behind a paywall.
- We remained consistent in our use of the SEC EDGAR API, obtaining data regarding assets, accounts receivable, accounts payable, and inventory for the companies in question on a quarterly basis over a nine-year period from 2014–2022.
- We also remained consistent in our use of the Federal Reserve FRED API and gathered federal macroeconomic data such as GDP, unemployment rate, federal funds rate, and CPI on a quarterly basis from 2014–2022.

3. The problems that you faced

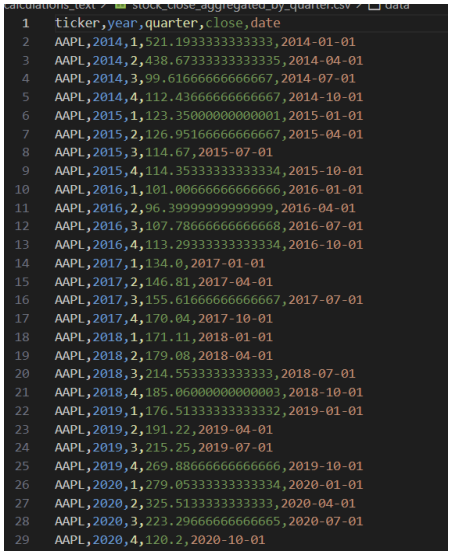
- The first issue we faced was in using the Polygon.io API, which returned JSON formatted in a more complex fashion and did not offer the historical information we were seeking for free. If we attempted to query beyond a two-year time horizon, it required a premium subscription, so we decided to pivot to Alpha Vantage, which offered monthly historical data dating back to 1999 for free.
- The APIs we were using all had rate limits, so we wanted to ensure that we were only querying for the JSON data when absolutely necessary. We developed our functions to retrieve data in such a way that a unique folder was created for each ticker called in the code, and we cached the results of the API calls into those cache files. We then sequentially inserted the data gathered from the API into the database and selected data

for calculations from the database. Just as we learned in class, we understood that we did not want to lose access to the JSON data by reaching our query limits, so we created a mechanism to cache the data if we had already obtained information for a given URL query.

- We encountered another issue with the SEC EDGAR API at first, as it seemed our URL query was correct, but the requests for data were being denied. It turned out that we needed to include headers in the URL to convey that we were real people attempting to access the API data so that it would not get overwhelmed. We created sample credentials to ensure the requests were valid and were then able to easily obtain the information.
- Another issue we had to overcome was developing a structure for the code in our main function for the database to ensure that each time the code was run, fewer than 25 rows of data were added to each table. This required us to query the database to understand where our progress stood in the insertion process and was uniquely more difficult than simply running for loops to push data into the database in bulk.
- For the visuals, we also had to contend with the fact that the phrasing of the metrics in the APIs, and how we inserted them into the database, were not in a great format to be included as labels in visuals. To resolve this, we created dictionaries to map the variables to labels so the visuals would look sleeker and cleaner.

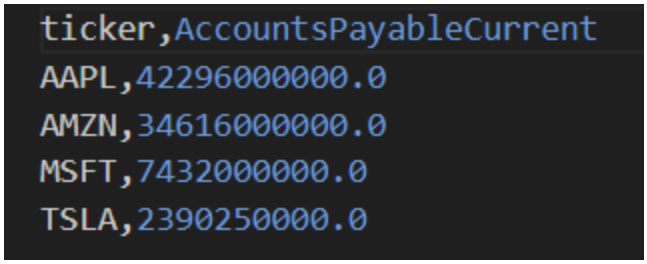
4. The calculations from the data in the database

Calculated the average of each stock's monthly closing price within each quarter and sorted the data by date to plot the time series visual (Below is a snippet)

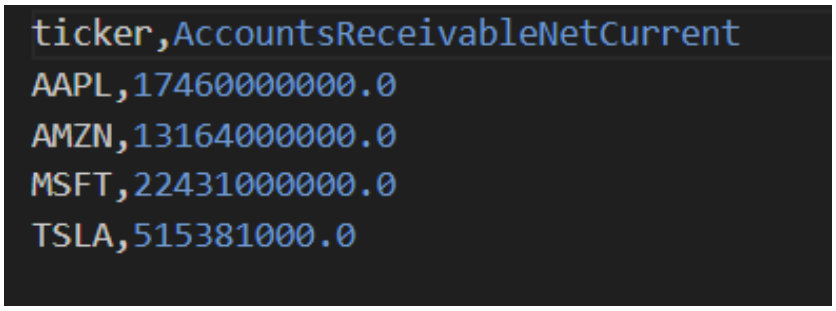


```
ticker,year,quarter,close,date
AAPL,2014,1,521.1933333333333,2014-01-01
AAPL,2014,2,438.67333333333335,2014-04-01
AAPL,2014,3,99.61666666666667,2014-07-01
AAPL,2014,4,112.43666666666667,2014-10-01
AAPL,2015,1,123.35000000000001,2015-01-01
AAPL,2015,2,126.95166666666667,2015-04-01
AAPL,2015,3,114.67,2015-07-01
AAPL,2015,4,114.35333333333334,2015-10-01
AAPL,2016,1,101.00666666666666,2016-01-01
AAPL,2016,2,96.39999999999999,2016-04-01
AAPL,2016,3,107.78666666666668,2016-07-01
AAPL,2016,4,113.29333333333334,2016-10-01
AAPL,2017,1,134.0,2017-01-01
AAPL,2017,2,146.81,2017-04-01
AAPL,2017,3,155.61666666666667,2017-07-01
AAPL,2017,4,170.04,2017-10-01
AAPL,2018,1,171.11,2018-01-01
AAPL,2018,2,179.08,2018-04-01
AAPL,2018,3,214.5533333333333,2018-07-01
AAPL,2018,4,185.06000000000003,2018-10-01
AAPL,2019,1,176.51333333333332,2019-01-01
AAPL,2019,2,191.22,2019-04-01
AAPL,2019,3,215.25,2019-07-01
AAPL,2019,4,269.88666666666666,2019-10-01
AAPL,2020,1,279.05333333333334,2020-01-01
AAPL,2020,2,325.5133333333333,2020-04-01
AAPL,2020,3,223.29666666666665,2020-07-01
AAPL,2020,4,120.2,2020-10-01
```

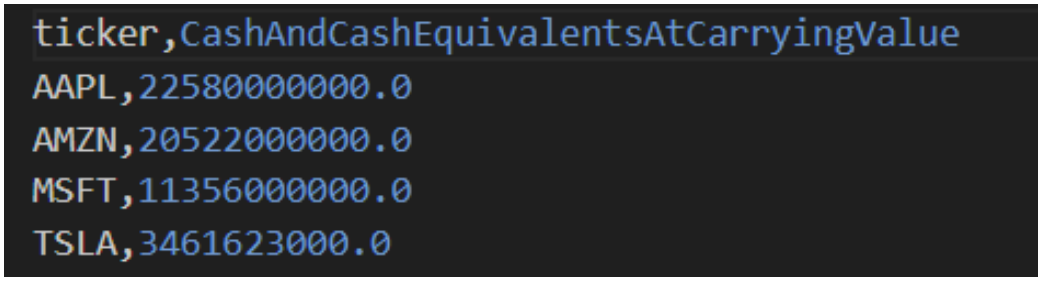
Calculated The Median Value For All Accounts Payable Records For Each Stock From 2014 - 2022



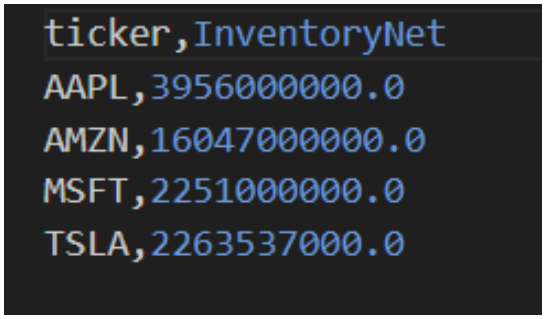
Calculated The Median Value For All Accounts Receivable Records For Each Stock From 2014 - 2022



Calculated The Median Value For All Cash and Cash Equivalents Records For Each Stock From 2014 - 2022



Calculated The Median Value For All Cash and InventoryNet Records For Each Stock From 2014 - 2022

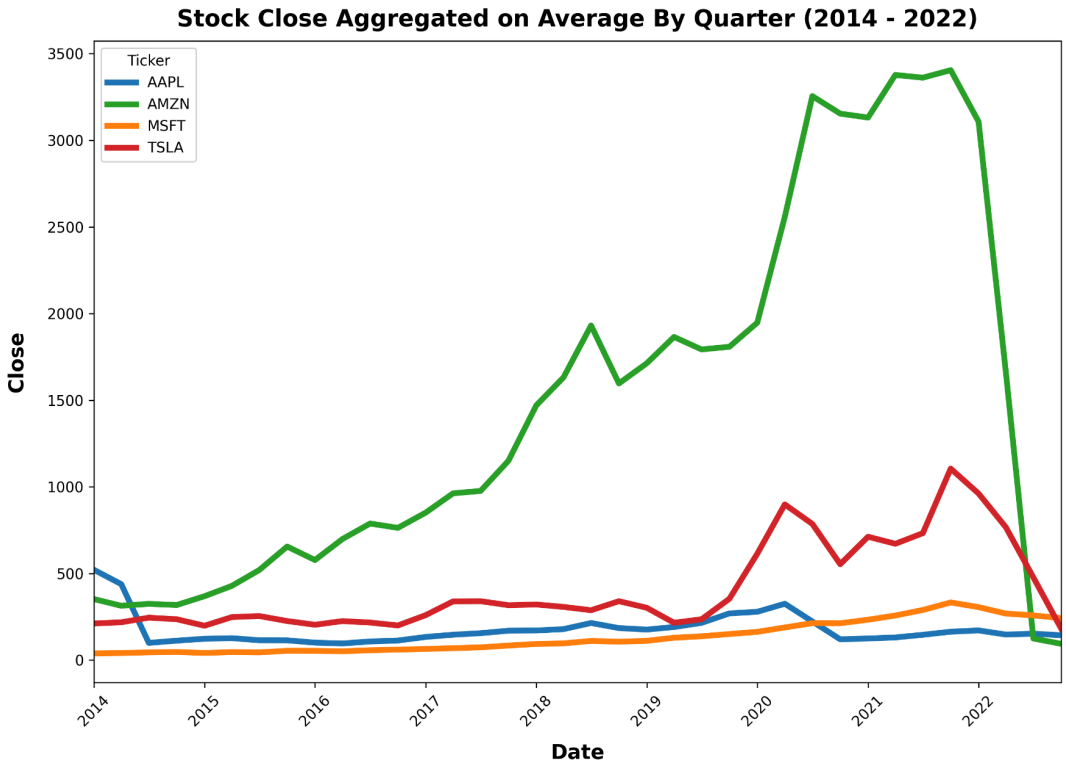


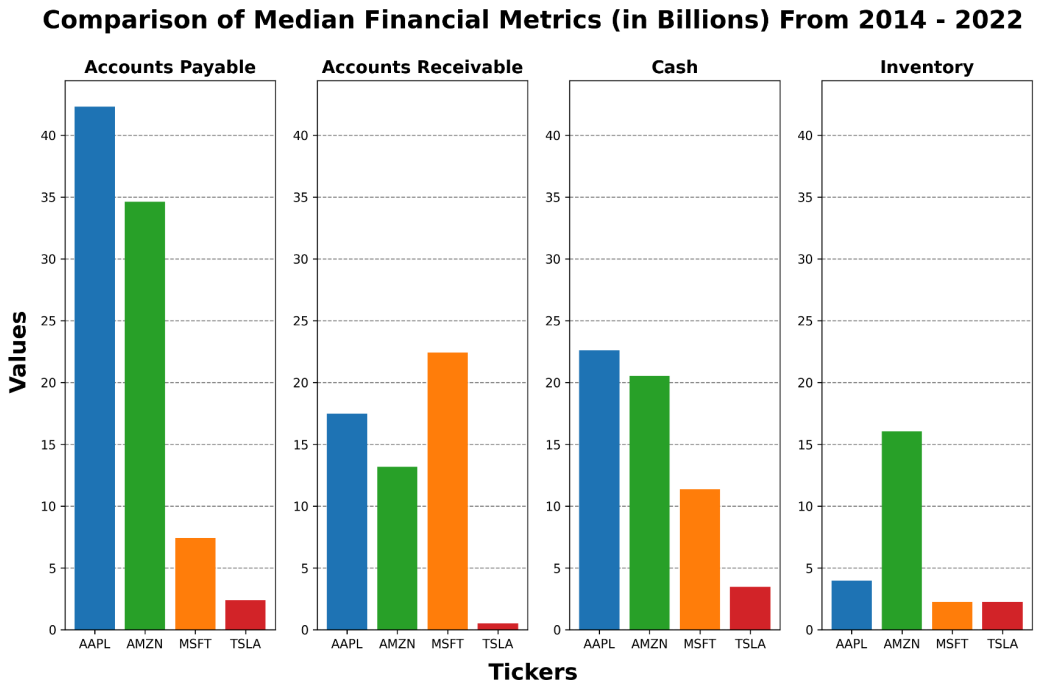
Normalized Data for GDP and CPI on a Quarterly Basis From 2014 - 2022 (Snippet Included)

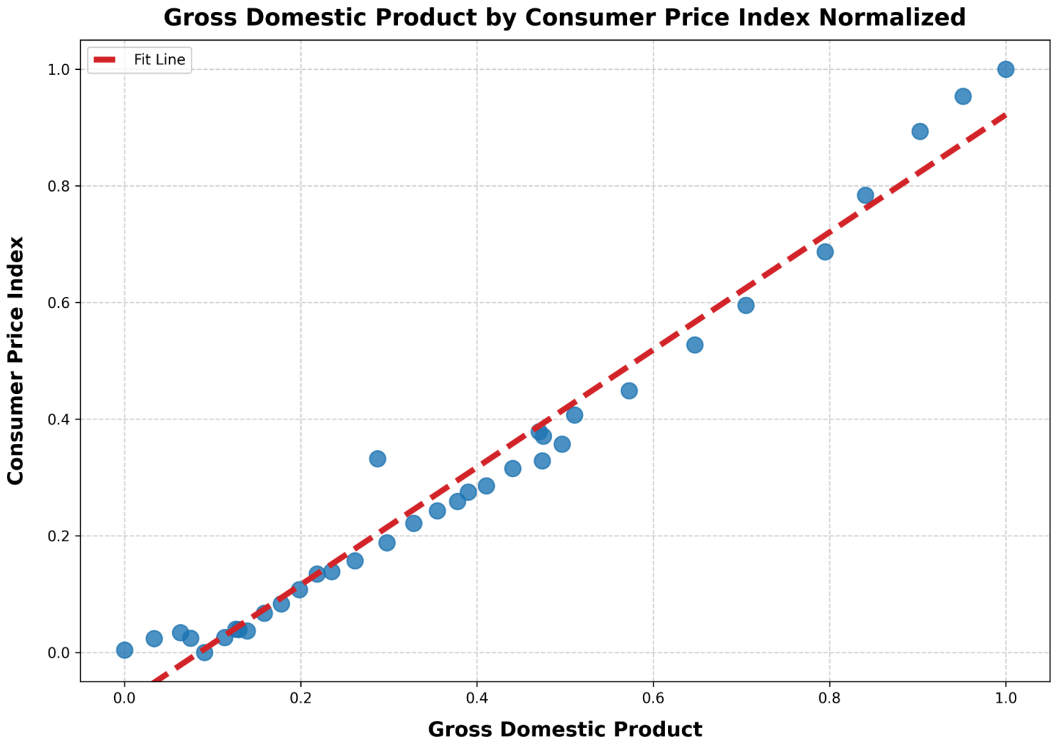
(Normalized formula for each variable: $\frac{value - min}{max - min}$)

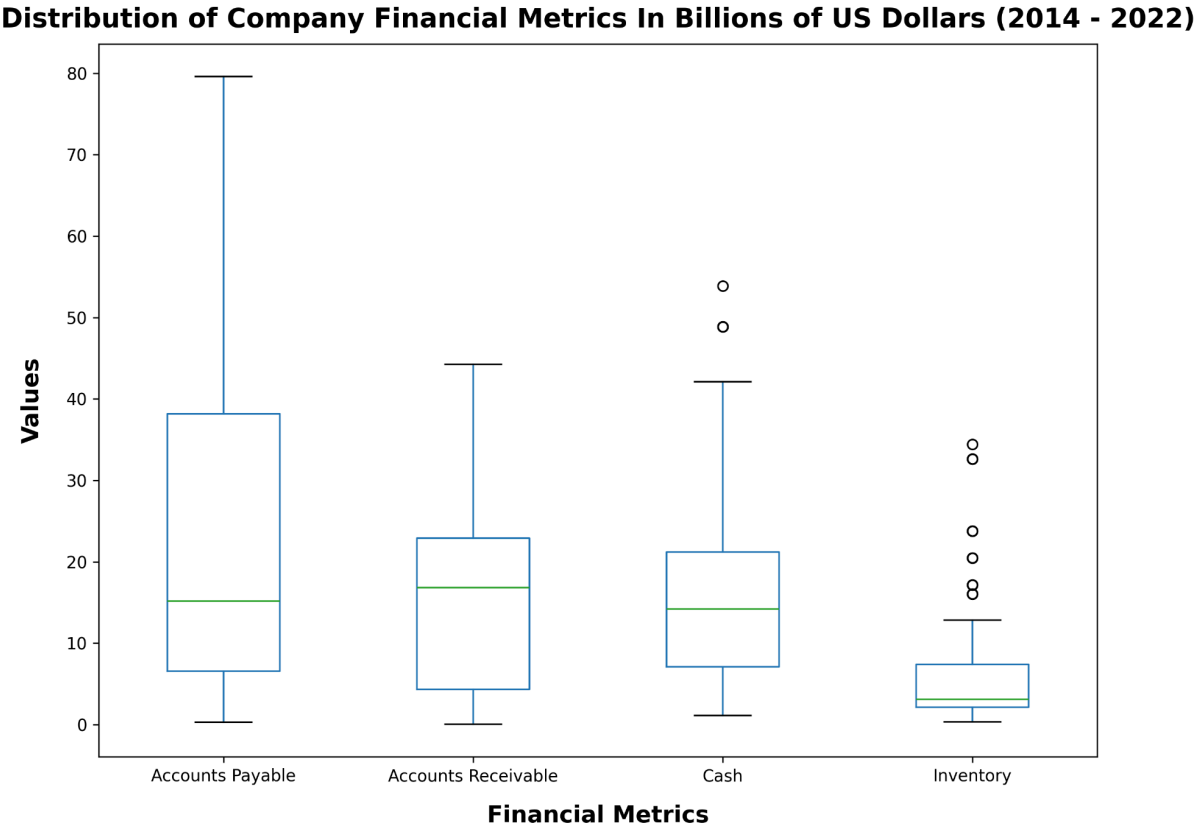
```
date,GDP_normalized,CPIAUCSL_normalized
2014-01-01,0.0,0.004216466410931421
2014-04-01,0.03363589243435176,0.02404653964429548
2014-07-01,0.06359644730651216,0.03365247440002567
2014-10-01,0.07490568643404075,0.024300161684050595
2015-01-01,0.09078670993743095,0.0
2015-04-01,0.11346317568669292,0.025441460862949284
2015-07-01,0.12623950890359692,0.03962844371175856
2015-10-01,0.129753467164555,0.03934311891703389
2016-01-01,0.1392743216380702,0.03699711504929784
2016-04-01,0.15875403015706227,0.06713058364771922
2016-07-01,0.17772705590571156,0.08325143454966254
2016-10-01,0.1983571817826153,0.10747233934628923
2017-01-01,0.2183544784958147,0.13448308658022398
```

5. The visualizations that you created




Comparison of Median Financial Metrics (in Billions) From 2014 - 2022
Accounts Payable
Accounts Receivable
Cash
Inventory
Values
Tickers
AAPL
AMZN
MSFT
TSLA


Gross Domestic Product by Consumer Price Index Normalized
Fit Line
Consumer Price Index
Gross Domestic Product



6. Instructions for running your code

- Our code is relatively simple to run. We have two main functions and files, one for the API and database insertion process and another for visualization creation. Given that you are starting from a completely blank database, you will need to run the API and database main file, 'data_API_main.py', approximately 25 times until 100 rows of data are inserted into each table. Afterward, code for a bulk for loop will insert the rest of the data in one fell swoop when the insertion limit no longer applies.
- Once you have run the file containing the main function for the API and database, you only need to run the visualizations main file, 'visualizations_main.py', one time. The visuals will then be created and saved as individual PNG files in the 'visualizations_figures' folder.

7. Documentation for each function that you wrote. This includes describing the input and output for each function

API Requests, Dict conversions, Database Insertions (All are very similar for each API)

- **database.py (Database Functions)**
  - **set_up_database(db_name)**
    - **Description**

                - Takes in a name for a database and creates that database in sqlite
            - **Input**
                - db_name (string): String for a database name
            - **Output**
                - Returns a database cursor and database connection
    - **create_stock_table(cur, conn)**
        - **Description**
            - Takes in a database cursor and connection and creates tables, one for metrics and one for data, to store stock info
        - **Input**
            - cur (cursor): database cursor
            - conn (connection): database connection
        - **Output**
            - Returns None
    - **create_company_table(cur, conn)**
        - **Description**
            - Takes in a database cursor and connection and creates tables, one for metrics and one for data, to store company financial info
        - **Input**
            - cur (cursor): database cursor
            - conn (connection): database connection
        - **Output**
            - Returns None
    - **create_macro_table(cur, conn)**
        - **Description**
            - Takes in a database cursor and connection and creates tables, one for metrics and one for data, to store macroeconomic info
        - **Input**
            - cur (cursor): database cursor
            - conn (connection): database connection
        - **Output**
            - Returns None
- **stock_data.py (Stock Data Functions)**
    - **get_stock_data(ticker)**
        - **Description**
            - This function gets the JSON data from the Alpha Vantage API for the stock that it is called. This function creates a cache file for that data if the data has not already been retrieved and if it has been retrieved it prints a message saying we have that data cached.
        - **Input**

            - ticker (String): Enter the stock ticker as it is formatted on the stock exchange
        - **Output**
            - Returns None
    - **def stock_data_to_dict(ticker, year)**
        - **Description**
            - This takes in the stock ticker and a year value and uses the data derived from the API to format a dictionary that can later be used to insert the data into a database
        - **Input**
            - ticker (String): Enter the stock ticker as it is formatted on the stock exchange
            - year (Int): Enter the year you desire to retrieve data for
        - **Output**
            - Returns Dictionary with date stamp as the keys and a tuple including open price, close price, high price, low price, and volume for that given time period
                - out_dict[date] = (open_price, high, low, close, volume)
    - **def insert_stock_data(ticker, cur, conn, data)**
        - **Description**
            - This function takes data for stocks formatted in the proper dictionary format and inserts it into the database to the proper tables
        - **Input**
            - ticker (string): Enter the stock ticker as it is formatted on the stock exchange
            - cur (cursor): Enter the database cursor
            - conn (connection) : Enter the database connection
            - data: (Dict): Enter the data formatted in the dictionary format from the ‘stock_data_to_dict’ function
        - **Output**
            - Returns None
- **company_data.py (Company Finance Functions)**
    - **get_company_info(ticker, CIK, concept)**
        - **Description**
            - This function gets the JSON data from the SEC EDGAR API for the stock and financial metric it is called for. This function creates a cache file for that data if the data has not already been retrieved and if it has been retrieved it prints a message saying we have that data cached.
        - **Input**
            - ticker (String): Enter the stock ticker as it is formatted on the stock exchange

                - CIK (string): Enter the CIK that for the company you want this data for as formatted on EDGAR website
                - concept (string): Enter string for the financial metric you want data for as formatted on EDGAR website
            - **Output**
                - Returns None
        - **company_data_to_dict(ticker, concept, year)**
            - **Description**
                - This takes in the stock ticker, financial metric, and a year value and uses the data derived from the API to format a dictionary that can later be used to insert the data into a database
            - **Input**
                - ticker (String): Enter the stock ticker as it is formatted on the stock exchange
                - concept (string): Enter string for the financial metric you want data for as formatted on EDGAR website
                - year (Int): Enter the year you desire to retrieve data for
            - **Output**
                - Returns Dictionary with concept as key and dict with year as key and quarterly values in chronological order as values
                    - {concept : {year : (Q1, Q2, Q3, Q4)}}
        - **insert_company_data(ticker, cur, conn, data)**
            - **Description**
                - This function takes data for financial metrics formatted in the proper dictionary format and inserts it into the database to the proper tables
            - **Input**
                - ticker (string): Enter the stock ticker as it is formatted on the stock exchange
                - cur (cursor): Enter the database cursor
                - conn (connection) : Enter the database connection
                - data: (Dict): Enter the data formatted in the dictionary format from the ‘company_data_to_dict’ function
            - **Output**
                - Returns None
- **macro_data.py (Macro Metric Functions)**
    - **get_macro_data(macro_indicator)**
        - **Description**
            - This function gets the JSON data from the Federal Reserve FRED API for the macroeconomic metric it is called for. This function creates a cache file for that data if the data has not already been retrieved and if it has been

retrieved it prints a message saying we have that data cached.

- **Input**
  - macro_indicator (string) : Enter the macroeconomic indicator you want to retrieve data for as formatted on the FRED API website
- **Output**
  - Returns None

- **macro_data_to_dict(macro_indicator, year)**
  - **Description**
    - This takes in a macroeconomic indicator and a year value and uses the data derived from the API to format a dictionary that can later be used to insert the data into a database
  - **Input**
    - macro_indicator (string) : Enter the macroeconomic indicator you want to retrieve data for as formatted on the FRED API website
    - year (Int): Enter the year you desire to retrieve data for
  - **Output**
    - Returns Dictionary with macro_indicator as key and dict with year as key and quarterly values in chronological order as value
      - {macro_indicator : {year : (Q1, Q2, Q3, Q4)}}
- **def insert_macro_data(cur, conn, data):**
  - **Description**
    - This function takes data for macroeconomic metrics formatted in the proper dictionary format and inserts it into the database to the proper tables
  - **Input**
    - cur (cursor): Enter the database cursor
    - conn (connection) : Enter the database connection
    - data: (Dict): Enter the data formatted in the dictionary format from the 'macro_data_to_dict' function
  - **Output**
    - Returns None

This file holds the main function to get all the data and insert it into the database

- **data_API_main.py**
  - **main()**
    - **Description**
      - Runs a series of code that requests data from the APIs, converts the JSON data to dictionaries that can be inserted into the database, and inserts data into the database.

There is logic to ensure that for each table's first 100 rows there is never more than 25 rows added in one run of the file and then after this threshold is reached the rest of the data is added in one iteration.

- **Input**
  - None
- **Output**
  - Return: None

This is the calculations file to create data frames based on the calculated information desired to be visualized

- **calculations.py**
  - **stock_time_series_calculations(cur, stock_metric)**
    - **Description**
      - This function takes in a database cursor and stock metric and retrieves the necessary data from the database, takes the median value for all the data for that metric and returns the data in the form of a df
    - **Input**
      - cur (cursor): database cursor object
      - stock_metric (string) : metric for the stock data that desired to be visualized
        - ‘open’, ‘close’, ‘low’, ‘high’, and ‘volume’
    - **Output**
      - Returns df to use to make visualization
  - **stock_finance_calculations(cur, finance_metric)**
    - **Description**
      - This function takes in a database cursor and stock metric and retrieves the necessary data from the database, takes the mean value for each quarterly period, and formats the final df to be used in the time_series visualization
    - **Input**
      - cur (cursor): database cursor object
      - finance_metric (string) : metric for the financial metric data that desired to be visualized
        - "AccountsPayableCurrent", "AccountsReceivableNetCurrent", "CashAndCashEquivalentsAtCarryingValue", "InventoryNet", "Assets"
    - **Output**
      - Returns df to use to make visualization
  - **macro_data_df(cur, metric_one, metric_two)**
    - **Description**

            - This function takes in a database cursor and two macroeconomic metrics and retrieves the necessary data from the database, creates a properly formatted data columns, normalizes the data over the course of the given period, and returns a df to be used in the visualization
        - **Input**
            - cur (cursor): database cursor object
            - Metric_one (string) : metric for the macroeconomic data that desired to be visualized
                - 'GDP', 'FEDFUNDS', 'CPIAUCSL', 'UNRATE'
            - Metric_two (string) : metric for the macroeconomic data that desired to be visualized
                - 'GDP', 'FEDFUNDS', 'CPIAUCSL', 'UNRATE'
        - **Output**
            - Returns df to use to make visualization
    - **stock_finance_pivot(cur)**
        - **Description**
            - This function takes in a database cursor and selects and formats the finance data into a df with proper numericals calling so that the data can be visualized
        - **Input**
            - cur (cursor): database cursor object
        - **Output**
            - Returns df to use to make visualization

This creates the visuals from the calculated data frames

- **visualization_code.py**
    - **stock_time_series_visual(df, agg_metric)**
        - **Description**
            - This is a function that takes in a data frame and then generates a time series visual with line plotting the movement of each stock over time. The function also saves the visual as a file in the ‘visualization_figures’ folder
        - **Input**
            - df (DataFrame): dataframe generated from ‘stock_time_series_calculations’ function
            - agg_metric (String): metric for the stock data that desired to be visualized
            - ‘open’, ‘close’, ‘low’, ‘high’, and ‘volume’
        - **Output**
            - Return: None
    - **finance_bar_chart_visual(cur, finance_metrics_dict)**
        - **Description**

            - This function takes in a database cursor object and a dictionary mapping metrics to labels to be used in the final visual. The function calculates the median values for each finance metric for each company and then creates a bar chart visual. The function also saves the visual as a file in the ‘visualization_figures’ folder
        - **Input**
            - cur (cursor): database cursor object
            - Finance_metrics_dict (dict): dictionary mapping each finance metric to a label to be used in the visual
        - **Output**
            - Return: None
    - **macro_scatter_visual(df, metric_one, metric_two, macro_labels_dict)**
        - **Description**
            - This is a function that takes in a data frame and then generates a scatterplot with a line of best fit comparing the relationship between the two variables. The function also saves the visual as a file in the ‘visualization_figures’ folder
        - **Input**
            - df (DataFrame): dataframe generated from ‘macro_data_df’ function
            - Metric_one (String): metric for macro data that is desired to be visualized
                - 'GDP', 'FEDFUNDS', 'CPIAUCSL', 'UNRATE'
            - Metric_two (String): metric for macro data that is desired to be visualized
                - 'GDP', 'FEDFUNDS', 'CPIAUCSL', 'UNRATE'
        - **Output**
            - Return: None
    - **stock_finance_box_plots_visual(df, finance_metrics_dict)**
        - **Description**
            - This is a function that takes in a data frame and then generates a plot of boxplots for the various financial metrics so that one can compare the distribution of the different metrics in the database
        - **Input**
            - df (DataFrame): dataframe generated from ‘stock_finance_pivot’ function
            - Finance_metrics_dict (dict): dictionary mapping each finance metric to a label to be used in the visual
        - **Output**
            - Return: None

This file holds the main function to perform all the calculations and make all the visualizations

- **visualizations_main.py**
  - **main()**
    - **Description**
      - Runs code to actually perform the necessary calculations for the visuals and then create the visuals to be used in the final report. All of the calculations are saved in the 'calculations_text' folder and all of the visualizations are stored in the 'visualization_figures' folder.
    - **Input**
      - None
    - **Output**
      - Return: None

8. You must also clearly document all resources you used. The documentation should be of the following form

Date	Issue Description	Location of Resource	Result
11/26/24	Needed to understand how to create files based on the name of the stock tickers to store the JSON data in a cache in an organized fashion	Looked into some of the os documentation and learned how to use the listdir and makedirs functions	Wrote logic to properly create the folders and check if they are created and then also create template file names to store the cache data
11/27/14	The company data API was not letting us originally access the data due to a lack of authentication credentials	We looked up some key terminology in the error message on the EDGAR API website and figured out we needed to provide user credentials to authenticate ourselves as real people	We added header decorators to the URL and was able to retrieve the data
11/30/24	Wanted to perform aggregations of data from the dataframe and wanted to use Pandas to achieve this	Consulted Pandas documentation especially the groupby, merge, and pivot functions	Performed dataframe manipulations on the information selected from the database and used the finalized data frames

			in the visualizations
11/30/24	Realized that in order to graph the data sequentially we needed to create date time stamps for the data aggregated on a quarterly basis	Consulted Pandas documentation to see how to use the DateTime objects and to_datetime function to create the date column	Created a date column and was easily to plot the data on a time series for our visualizations
12/2/24	There were a variety of intricacies we wanted to tweak to make our visuals look as good as possible such as label padding and text weight and other things along those lines	We consulted the matplotlib documentation extensively to understand how to manipulate the arguments inside many of the title and label arguments to make the visual aesthetically pleasing to look at	Generated visuals with a lot of customization and details
12/3/24	Wanted to figure out how to calculate the line of best fit for the scatter plot	Looked up how to do so online and found that the numpy polyfit and linspace functions enable us to get the appropriate data to know the x-values, slope, and intercept.	Plotted a line of fit on top of the scatter plot visual.
12/3/24	Wanted to change the color pallet for the different stock tickers in the visuals away from the standard library of colors provided by matplotlib	Looked online for color pallets that we thought worked well together and selected those color codes	Created a dictionary mapping each stock ticker to an appropriate color code for consistency across visualizations
12/4/24	Wanted to figure out how to save visuals to files	Referred back to in-class assignments and researched how to use the plt.savefig() function	Saved visuals to a folder in our directory effectively