SCHOOL OF INFORMATION
UNIVERSITY OF MICHIGAN

Stocked.AI

Creators:

Adam Ginsberg
Jai Advani

1.0: Goals

1.1: Initial Goals

Our goal was to create a python script that would use the polygon and NASDAQ APIs to collect data on the top 100 stocks by market cap and economic data on inflation. Once the data was collected, the script would calculate the correlation between the change in stock price and the inflation rate. After the calculation was complete, the script would then generate two graphs. The first would be a scatter plot showing the relationship between inflation rates and stock prices. The second would be an individual chart showing the stock prices and inflation rates over time. This would give us a better understanding of how changes in the inflation rate affect the stock market, and help us make more informed investment decisions

1.2: The goals that were achieved

Throughout this project, our initial goals seemed to change slightly and new goals were set. Our initial goals for this project were to gather data on the top 100 stocks by market cap in the S&P 500 and use the NASDAQ API to gather information on inflation rates. However, due to the amount of data that would need to be collected and stored, we were only able to gather data on the top stock by market cap in the S&P 500. Despite this limitation, we were still able to successfully gather and use this data to make several calculations. For instance, we calculated the average stock price for the top stock in the S&P 500 each month and found the percent change in average stock price each month. We then expanded our project to include bitcoin data, which we were able to gather using an additional API. This allowed us to extend our calculations to include percent change in bitcoin prices each month. We also succesfully used the NASDAQ API to gather information on inflation rates, which we used to calculate the percent change in inflation

each month. Within this part of the project, we also decided to set a new goal of using beautiful soup to webscrape for the top asset within the S&P 500 and use that as the basis for our API calls. This is something that we successfully implemented.

With the data we collected, we were able to create several visualizations. We made visualizations for the top stock and bitcoin, showing average volume and average price for the past 10 years. We also created a visualization for the cpi (inflation) over the past 10 years. Finally, we created a single chart showing the change in stock price, change in inflation, and change in bitcoin over the past 10 years, to see how they are related. Overall, we were able to achieve many of our goals despite some limitations, and the data and visualizations we created will be useful for analyzing and understanding the relationship between stock prices, inflation, and bitcoin.

2.0: Problems

2.1: Summary

In this project, our goal was to create a Python script that would use the Polygon and NASDAQ APIs to collect data on the top 100 stocks by market cap and economic data on inflation. We intended to use this data to calculate the correlation between the change in stock price and the inflation rate, and generate two graphs to visualize this relationship.

However, we encountered several problems along the way. The first was that Polygon did not provide the information we needed, and the API was limited in terms of the number of requests we could make without payment. This led us to switch to the market stack API, which allowed us to set a date range for end-of-day prices of a given stock or set of stocks. However, this introduced a new problem: the API only provided data on a daily basis, whereas we needed data on a monthly basis. This meant that we had to change our original plan of using data on the top 100 stocks in the S&P 500. Instead, we focused on just the top stock in the S&P 500 (Apple Inc.), and used beautiful soup to collect the data.

Once we had the data, we wanted to apply the same techniques to cryptocurrency using the tiingo API. However, we found that many of the methods we wanted to use with crypto would not work, as the API was structured differently. This meant that we had to organize the data from tiingo in a way that fit with the data from market stack. We discovered that the easiest way to do this was to essentially break apart the data and resrtucre it in the same way that we did for the market stack api. This allowed us to apply the same functions to the crypto data as we did to the stock data.

Next, we faced problems with creating the database. The main challenge was determining the best way to organize the data, which we ultimately solved by creating a table of month-year pairs that would be represented as IDs on all of the other tables. However, we also had to sort some of the dictionaries to make them properly line up with the month-year IDs. We simply did this by reversing the order that the dictionaries were given to the data base in. This had to happen as the data that came from the crypto API came from oldest to newest, whereas that of the stock API came from newest to oldest.

When it came time to create the visualizations, we encountered several additional problems. The first was that the data was not legible, so we had to learn a technique for turning the labels slightly to make them easier to read. We also had difficulty adding a linear regression line, as the date axis labels were strings rather than numbers. This took us a little while to fix as the error we were given was not something we were familiar with. The error was being provided by numpy. What we finally determined to do was simply make a list of numbers that is equal to the length of the month string list that was initially given to the linear regression formula. This allowed numpy to compare numbers to numbers, which then plotted the lines we desired. Another struggle we had was with organizing our subplots on one page. This we simply fixed through Experimentation. Finally, when graphing all three lines on the same chart, we ran into the problem of having different x-axis lengths for each chart. This was peculiar but we think it is just because of the start data of the data gathered from the economic data API. Since this data is mising the most recent month of data. We fixed this by simply setting the x-axis lists of all of the other variables, stock and crypto, to the length of that of the economic data(CPI).

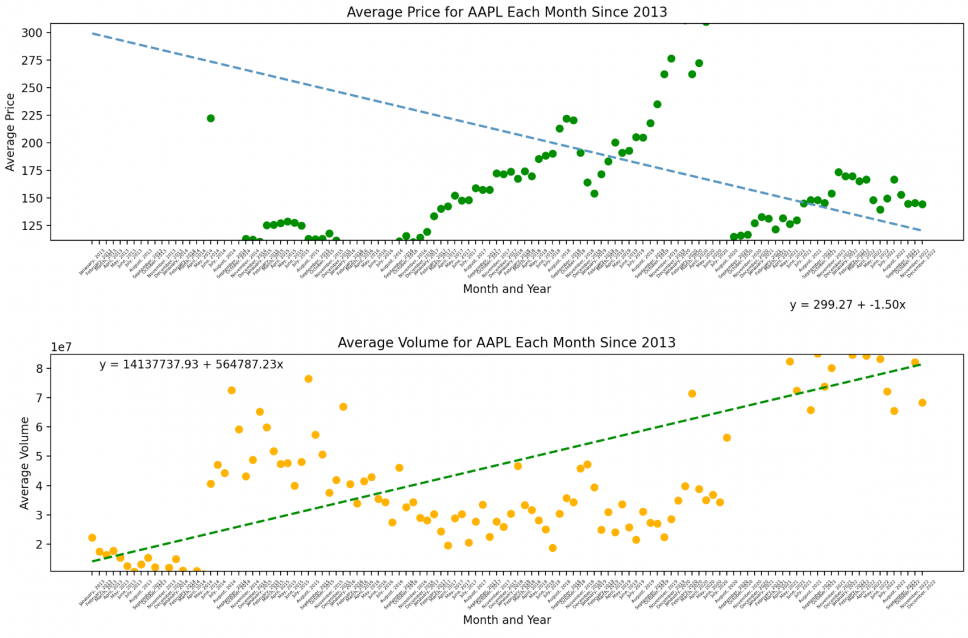
Overall, this project presented us with a number of challenges that we had to overcome in order to successfully collect and analyze the data, and generate useful visualizations. However, despite these difficulties, we were ultimately able to create a Python script that achieves our original goals, and provides insights into the relationship between inflation and stock prices.

4.0: File of Caclulations:

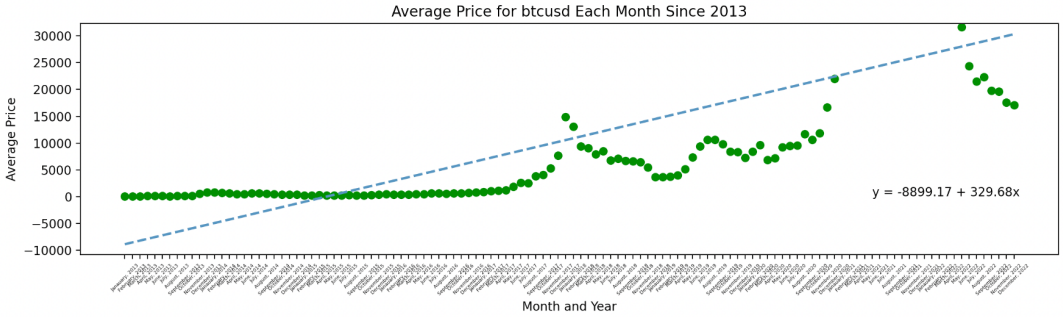
https://github.com/adamjgins/final_project_206/blob/main/calculations.txt

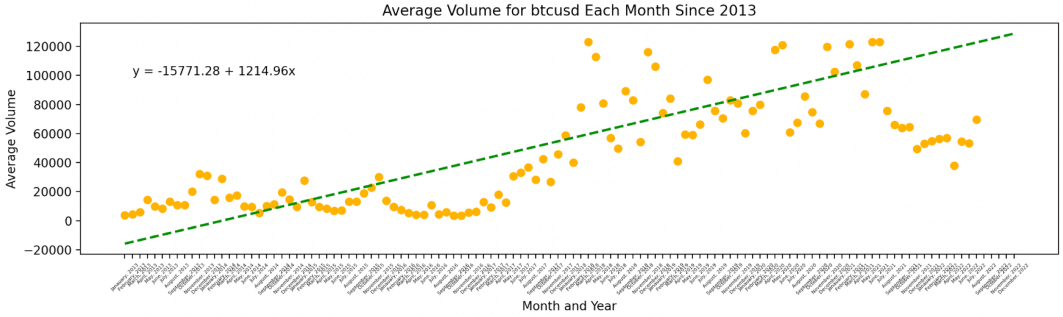
5.0: Images of Charts Created

5.1: Top S&P 500 Stock Visualization

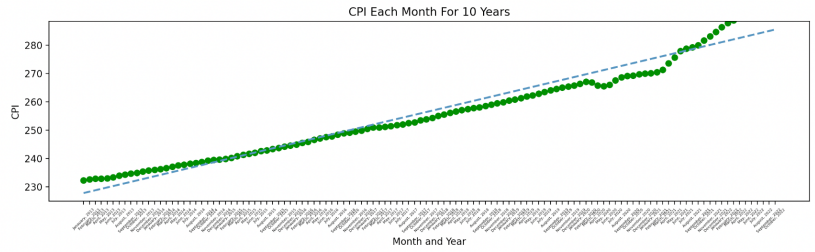


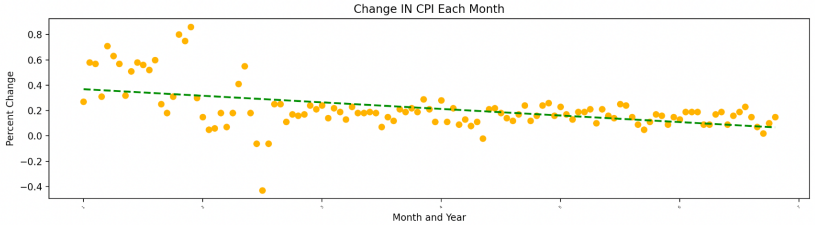
5.2: Bitcoin Visualization

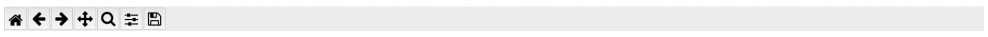




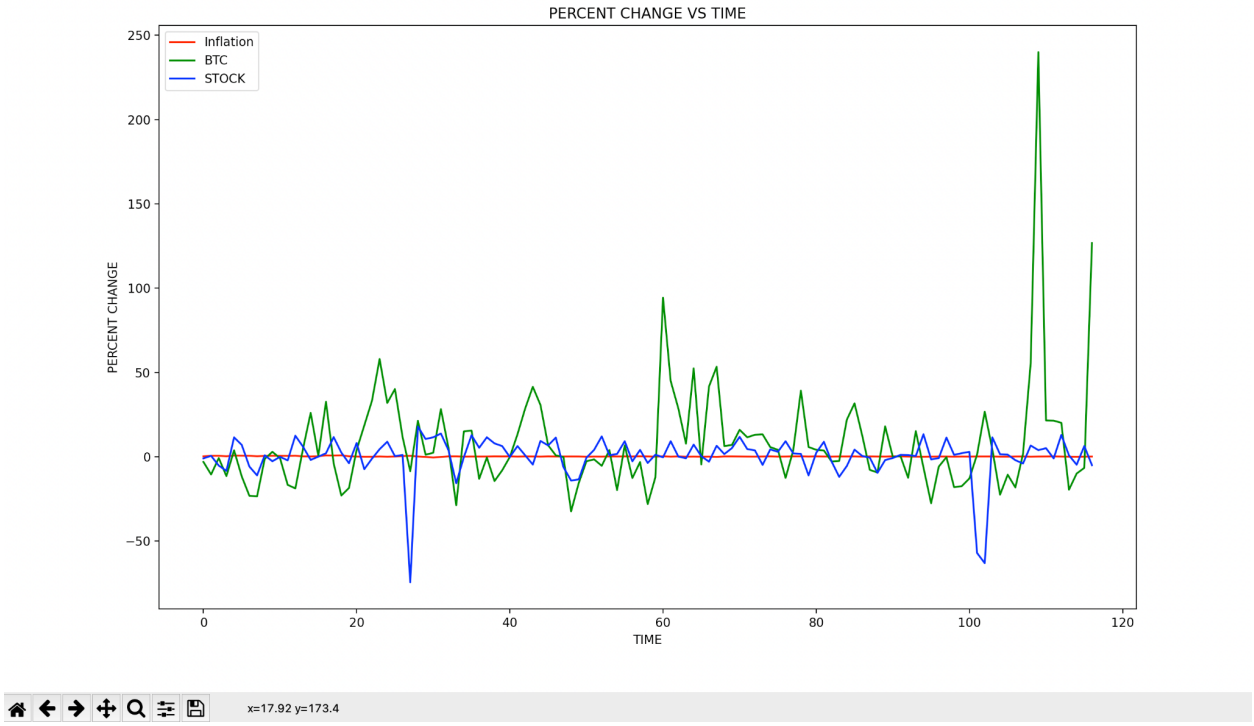
5.3: CPI/Inflation Rate Visualization







5.4: Percent Change For Each Asset



6.0: Instructions For Running This Code

6.1: First Use

In order to run this code you must just simply have a file in the same location as the python document named “final_project.db”. This file will act as the data base for the data that is collected when running the program. Also, ensure the “val” on line 1023 is set to 1.

6.2: What Will Happen

Once you run the program, 4 tables will be created within the “final_project.db”. Each of these table will have 25 data points associated with it. We recommend running the program 5 times in order to collect all 120 data points associated with this project.

6.3: Clearing the Database and Deleting the Output Calculations File:

If you want to run the code fresh, simply change “val” on line 1023 to be anything other than the number 1. This will run a function that clears the database and deletes the calculations.txt file.

7.0: Documentation

7.1: **get_spy_data()**

Input: None

Description:

This function uses beautiful soup to parse through the the website “Stock Market MBA”. Stock market MBA is a website that contains a chart of each of the stocks in the S&P 500. The function selects the contents of the website that contain a “tr” tag. From there it loops through each of the TR tags and gets information about the ticker symbol and market cap for each stock listed on the S&P 500. It uses this information to form a dictionary of stock name and market capitalization. This dictionary is then used to create a list of tuples that is sorted by stock with the largest market capitalization.

Expected Output:
[('AAPL', 2242090150920), ('MSFT', 1821649602199), ('GOOG', 1146159520000)...]

7.2: **get_economic_data()**

Input: None

Description:

This function uses the FRED API in order gather information on the CPI (Consumer Price Index) since 2013. The function collects the uncleaned data through an API call to FRED. It collects the data in a JSON format. From there it uses the helper function “make_list_of_dictionaries()” (see helper functions) in order to turn the json data into a list of dictionaries in order for it to be more easily integrated into the data base. Finally, this data is reversed in order to put the most recent year first in the dictionary.

Expected Output:
{2022: {10: [{'date': '2022-10-01', 'value': '299.471'}], 9: [{'date': '2022-09-01', 'value': '298.660'}],...{2013: {12: [{'date': '2013-12-01', 'value': '235.759'}]...

7.3: **get_crypto_data()**

Input: None

Description:

This function uses the tiingo API in order gather information on the pricing and volume of bitcoin each month since 2013. The function collects the uncleaned data through an API call to tiingo. It collects the data in a JSON format. From there it uses the helper function “make_list_of_dictionaries()” (see helper functions) in order to turn the json data into a list of dictionaries in order for it to be more easily integrated into the data base. Although the data is organized properly, there is entirely too much of it. This is because the api call collected data on every single day since 2013. This is where the helper function “find_average_of_list” (see helper functions) is used. This allows for out data output to contain just the averages of each month since 2013. Finally, this data is reversed in order to put the most recent year first in the dictionary. Additionally, the data within each year of the dictionary is reversed in order to have the most recent month (12) first.

Expected Output:
{2022: {12: {'ticker': 'btcusd', 'avg_close': 17058.43, 'avg_vol': 227095.86}, 11: {'ticker': 'btcusd', 'avg_close': 17586.99, 'avg_vol': 350479.18}, 10: {'ticker': 'btcusd', 'avg_close': 19649.01, 'avg_vol': 272248.46},....{2013}

7.3: **get_stock_data()**

Input: tickers list

Description:

This function uses the marketstack API in order gather information on the pricing and volume of the top stock by market cap in the S&P 500 each month since 2013. The function uses the provided input to find the first stock ticker in the provided list. This list is assumed to be collected from the “get_spy_data” function. The function then collects the uncleaned data through a series of 3 API calls to market stack. This was done as we are limited to only 1000 data points on each collection. It collects the data in a JSON format. The data from each of the 3 API calls is then merged together to make one long dataset. From there it uses the helper function “make_list_of_dictionaries()” (see helper functions) in order to turn the json data into a list of dictionaries in order for it to be more easily integrated into the data base. Although the data is organized properly, there is entirely too much of it. This is because the api call collected data on every single day since 2013. This is where the helper function “find_average_of_list” (see helper functions) is used. This allows for out data output to contain just the averages of each month since 2013. The data is then returned.

Expected Output:
{2022: {12: {'ticker': 'AAPL', 'avg_close': 144.49, 'avg_vol': 68306339.14}, 11: {'ticker': 'AAPL', 'avg_close': 145.84, 'avg_vol': 82103164.62},...{2013}

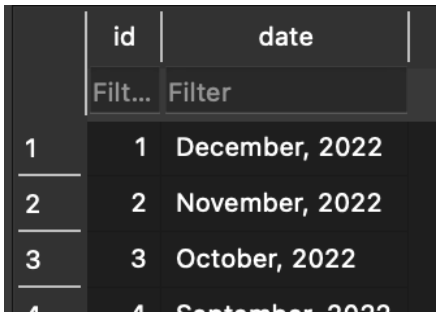
7.4: **create_dates_table()**

Input: stock_dict: dictionary, db_filename: string

Description:

This function takes the dictionary of stocks and the database file name and creates a table of dates. The table is structured so that the each unique key id is matched up to one date. The function begins by creating a table if it does not exist. It then gets the current id that is listed in the table. If the id is none, it begins to populate the table at the first data point in the passed through dictionary. In this case, the stock dictionary. This particular function only uses the “date” data that is within the stock dictionary.

Expected Output:



	id	date
	Filt...	Filter
1	1	December, 2022
2	2	November, 2022
3	3	October, 2022

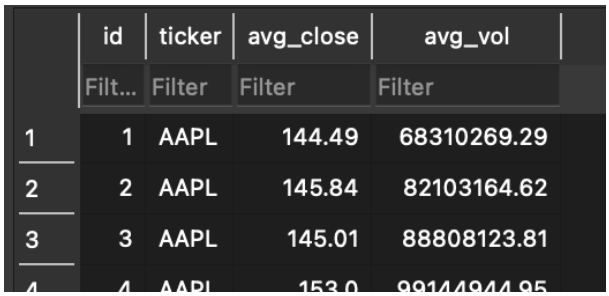
7.5: **create_stock_table(stock_dict,db_filename)**

Input: stock_dict: dictionary, db_filename: string

Description:

This function takes the dictionary of stocks and the database file name and creates a table of id,ticker,avg_close,avg_vol. The table is structured so that the each unique key id is matched up to one date. These are the dates that were stored in the dates database. The function begins by creating a table if it does not exist. It then gets the current id that is listed in the table. If the id is none, it begins to populate the table at the first data point in the passed through dictionary. In this case, the stock dictionary that was created in the api call function. This particular function uses the ticker, avg_close, avg_vol in order to populate the database.

Expected Output:



	id	ticker	avg_close	avg_vol
	Filt...	Filter	Filter	Filter
1	1	AAPL	144.49	68310269.29
2	2	AAPL	145.84	82103164.62
3	3	AAPL	145.01	88808123.81
4	4	AAPL	153.0	99144944.95

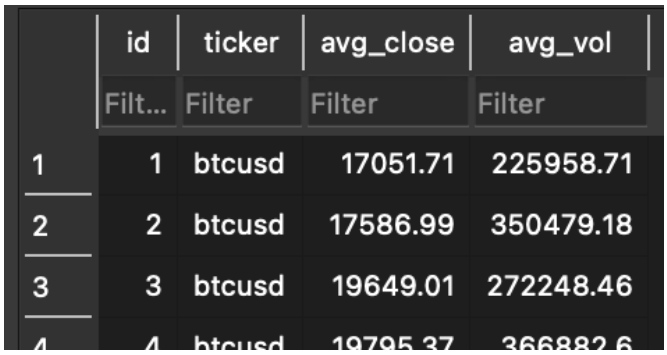
7.6: **create_crypto_table(crypto_dict,db_filename)**

Input: crypto_dict: dictionary, db_filename: string

Description:

This function takes the dictionary of stocks and the database file name and creates a table of id,ticker,avg_close,avg_vol. The table is structured so that the each unique key id is matched up to one date. These are the dates that were stored in the dates database. The function begins by creating a table if it does not exist. It then gets the current id that is listed in the table. If the id is none, it begins to populate the table at the first data point in the passed through dictionary. In this case, the crypto dictionary that was created in the api call function. This particular function uses the ticker, avg_close, avg_vol in order to populate the database.

Expected Output:



	id	ticker	avg_close	avg_vol
	Filt...	Filter	Filter	Filter
1	1	btcusd	17051.71	225958.71
2	2	btcusd	17586.99	350479.18
3	3	btcusd	19649.01	272248.46
4	4	btcusd	19795.37	366882.6

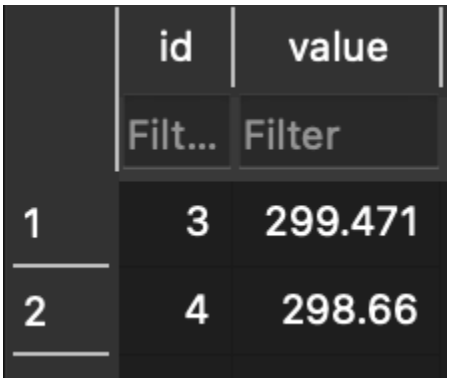
7.7: **create_economic_table(economic_dict,db_filename)**

Input: economic_dict: dictionary, db_filename: string

Description:

This function takes the dictionary of economic datapoints and the database file name and creates a table of id,CPI. The table is structured so that the each unique key id is matched up to one date. These are the dates that were stored in the dates database. The function begins by creating a table if it does not exist. It then gets the current id that is listed in the table. If the id is none, it begins to populate the table at the first data point in the passed through dictionary. In this case, the economic data dictionary. This particular function uses only the CPI data from the passed in function

Expected Output:



	id	value
	Filt...	Filter
1	3	299.471
2	4	298.66

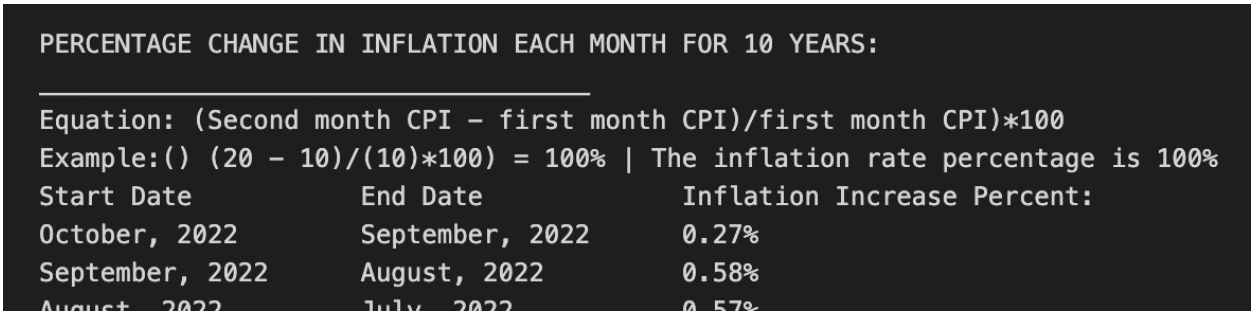
7.8: **calculate_inflation_rate(db_filename)**

Input: db_filename: string

Description:

This function reads through the data base and selects the value data from the economic_info table in the database. From there, it calculates the inflation rate by taking the value at the first given month, subtracting the value of the second given month, and dividing that subtraction value by the value of the second given month. This provides the percent change in inflation between each month, or the inflation rate. This is then written to the text file “calculations.txt”. If the text file does not exist, it will create it. It also outputs a list of inflation increases and returns that list.

Expected Output: [0.27,0.58…..]



7.9: **calculate_percent_change(db_filename,table_name)**

Input: db_filename: string, table_name:string

Description:

This function reads through the data base and selects the value data from the **table_name** table in the database. For the purposes of this program, this will likely either be the crypto_info table or the stock_info table From there, it calculates the change rate by taking the value at the first given month, subtracting the value of the second given month, and dividing that subtraction value by the value of the second given month. This provides the percent change in asset price between each month, or the percent change. This is then written to the text file “calculations.txt”. If the text file does not exist, it will create it. It also creates a list of percentage increases and returns that list.

Expected Output:[-3.04,-10.49,...]

```
PERCENTAGE INCREASE FOR btcusd EACH MONTH FOR 10 YEARS:
_____________________________________
Equation: (Second month price - first month price)/first month price)*100
Example:() ($20 - $10)/($10)*100) = 100%
Ticker     Start Date           End Date             Percentage Increase
btcusd     December, 2022       November, 2022       -3.04%
btcusd     November, 2022       October, 2022        -10.49%
btcusd     October, 2022        September, 2022      -0.74%
```

```
PERCENTAGE INCREASE FOR AAPL EACH MONTH FOR 10 YEARS:
_____________________________________
Equation: (Second month price - first month price)/first month price)*100
Example:() ($20 - $10)/($10)*100) = 100%
Ticker     Start Date           End Date             Percentage Increase
AAPL       December, 2022       November, 2022       -0.93%
AAPL       November, 2022       October, 2022        0.57%
```

7.10: **calculate_total_average(db_filename,)**

Input: db_filename: string

Description:

This function uses the build in avg in sql to go through each of the tables in the database provided and provide the all time average.

Expected Output:

```
THE AVERAGE COSTS OF AAPL AND btcusd OVER THE PAST 10 YEARS
_____________________________________________
FOUND USING AVG FUNCTION PROVIDED IN SQL
TICKER  AVERAGE COST
AAPL    ....$209.94
btcusd  ..$10717.06
```

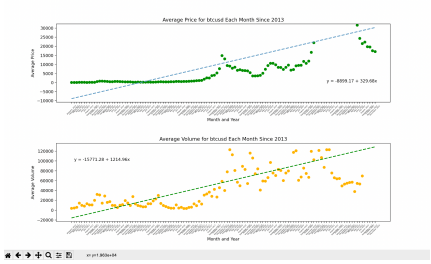
7.11: **price_visualization(db_filename,table_name)**

Input: db_filename: string, table_name: string

Description:

This function takes the table_name provided and searches for that table in the database. From there it selects the ticker, avg_close, and avg_vol. It uses these three metrics in order to make two subplots. The first subplot is a plot of time on the x axis and average price on the y axis. This plot is represented as a scatter plot with a line of best fit. The second subp,ot is a plot of time on the x axis and average volume on the y axis. This plot is also represented a s a scatter plot with a line of best fit. The dates on the x axis are found by using a join when initially selecting the data. It joins the table provided with the dates table. This allows for the ids on the table provided to be turned into actual dates.

Expected Output:



7.12: **inflation_visualization(db_filename,table_name,percentage_change_list)**

Input: db_filename: string, table_name: string, percentage_change_list:lsit

Description:

This function takes the table_name and the percentage change list provided and searches for that table in the database. From there it selects the values from the given table names. It uses the value metric and the percentage change list in order to make two subplots. The first subplot

is a plot of time on the x axis and CPI on the y axis. This plot is represented as a scatter plot with a line of best fit. The second subp,ot is a plot of time on the x axis and percentage change on the y axis. This plot is also represented a s a scatter plot with a line of best fit. The dates on the x axis are found by using a join when initially selecting the data. It joins the table provided with the dates table. This allows for the ids on the table provided to be turned into actual dates.

Expected Output:

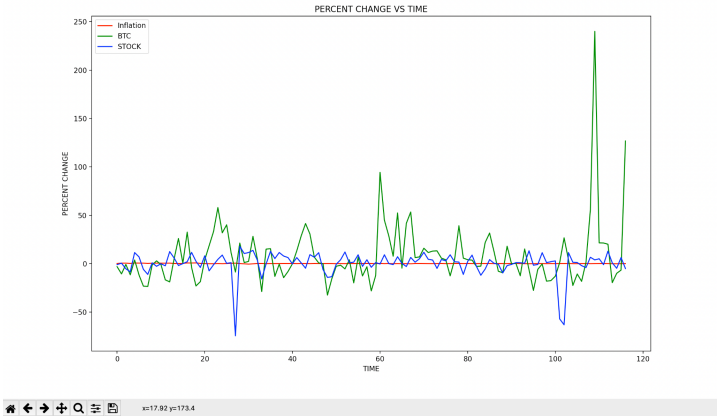
7.13: **asset_nflation_visualization(cpi_list,btc_list,stock_list)**

Input: cpi_list:list, btc_list:list,stock_list:list

Description:

This function takes 3 different lists of change values and plots them on to one singular chart. The x axis is the change over time. The y axis is percentage change. This helps us to visualize how each asset moves in comparison to inflation.

Expected Output:



7.14: **restart(db_filename)**

Input: db_filename

Description:

This function simply erases all the tables in the given database and deletes the output file “calculations.txt”.

Expected Output: NONE

7.15: **Helper Functions**

7.15.1: organize_by_year_month(data)

Input: data: dictionary of dictionarys

Description:

This function takes a list of dictionaryss where one value is a date. It then splits this date component and organizes a new dictionary where the values are ordered by {year:{month:{value}}}

Expected Output: ({year:{month:{value},year:{month:{value}...}})

7.15.2: make_list_of_dictionaries(data)

Input: data: dictionary of dictionarys

Description:

This function makes a list of dictionaries based on the dictionary provided to the function. It uses the organize_by_year month in order to do this.

Expected Output: ({year:{month:{value},year:{month:{value}...}})

7.15.3:find_average_of_list(data)

Input: list:list

Description:

This function takes a list and returns the average value of the list

Expected Output: ex. 10.98

Date	Issue Description	Resource Used	Result
Dec 5	Issue with Regex getting match objext	https://www.w3schools.com/python/gloss_python_regex_match.asp	Works
Dec 5	Table alter	https://www.w3schools.com/sql/sql_alter.asp	Didnt need
Dec 6	Error with data	**https://data.nasdaq.com/search?filters=%5B%22Cryptocurrency%22%5D**	Changed to other api: straight from FRED
Dec 6	Wrong number of bindings error?	https://stackoverflow.com/questions/16856647/sqlite3-programmingerror-incorrect-number-of-bindings-supplied-the-current-sta	Figured out the mismatch was the problem
Dec 7	Using this data to get only aapl	https://twelvedata.com/docs#stocks-list	Differnet API
Dec 7	Using this data to get historical prices	https://twelvedata.com/docs#stocks-list	Diff API
Dec 7	Seeing if there is a way to bypass limits?	https://twelvedata.com/docs#stocks-list	Diff API
Dec 7	Retireve crypto data from a certain time span	https://api.cryptorank.io/docs#operation/CurrenciesController_get	Works
Dec 7	Retrieve crypto data	https://api.cryptorank.	works

	for only one currency	io/docs#operation/CurrenciesController_get	
Dec 7	Crypto data - get volume?	https://api.cryptorank.io/docs#operation/CurrenciesController_get	works
Dec 8	Issue with retrieving data from tiingo	https://api.tiingo.com/documentation/crypto	works
Dec 8	Dont know how to reverse a row	https://www.programiz.com/python-programming/methods/list/reverse	works
Dec 8	Creating best fit line	https://www.statology.org/line-of-best-fit-python/	works
Dec 8	Signature matching type error with linear regression line?	https://stackoverflow.com/questions/44527956/python-ufunc-add-did-not-contain-a-loop-with-signature-matching-types-dtype	Realized problem was with the mismatch of data
Dec 8	Cursor object	https://stackoverflow.com/questions/56937330/cursor-returns-sqlite3-cursor-object-at-0x033a21e0-instead-of-returning-the-ob	Had to fetchall()
Dec 9	Confirming that data types will work in SQL	https://www.w3schools.com/sql/sql_datatypes.asp	works
Dec 10	Need only first 25 to be added	https://stackoverflow.com/questions/11768127/use-limit-in-a-mysql-insert	Didnt use this method
Dec 10	Need only first 25 to be addded	https://stackoverflow.com/questions/7971618/return-first-n-keyvalue-pairs-from-dict	Didnt use this method
Dec 11	Need to see if something was	https://stackoverflow.com/questions/16029	works

	already added to a dictionary	34/check-if-a-given-key-already-exists-in-a-dictionary	
Dec 12	Dont know how to delete document in python	https://stackoverflow.com/questions/6996603/how-do-i-delete-a-file-or-folder-in-python	Yes