SI 206 - Data Oriented Programming

Winter 2025

University of Michigan

Group: DOJE

Group members: Jack Bernard (jackber), Oleg Korobkov(olegko), Danielle Schulz (dfaria)

https://github.com/sldanischulz/SI206_FinalProject_WN25/tree/main

Final Project

A. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

The initial goal of our project was to investigate the relationship between market performance and President Donald J. Trump's posts to Truth Social. Our initial plan was to use a small, community based API called TruthBrush to scrape posts from a particular user on TruthSocial, and store their statuses (posts) in a database. We then intended to sum the amount of posts made in a particular day, and compare that to the market performance of the NASDAQ indices (pulled using Polygon API) and the market performance of Bitcoin (pulled using Coinbase API).

B. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

In reality, we were able to achieve some of our goals, but due to limitations of the TruthBrush API, we had to pivot and change our analysis goal to comparing the market activity of high profile tech stocks, and both the NASDAQ index and the Bitcoin market. In addition to the original Polygon API and the Coinbase API, we added the Finage API to pull stock market performance data of individual stock tickers. Due to the limit of the TruthBrush API, we opted to make it our extra credit API instead as it had no limits on the amount of data that could be pulled.

C. The problems that you faced (10 points)

Initially, the TruthBrush API was difficult to install due to its grassroots, community maintained nature. As it is not officially published, the package must be installed directly from the github repo, and was placed in a different folder than most other official package installs. Due to this, an edit must be made to the local PATH environment variable in Windows to allow Python to see and use the package.

After troubleshooting the TruthBrush API, getting it to work was also difficult due to the lack of documentation on the official API calls, leading to a lengthy search through the package's source code to understand the format and data requirements (and even the function names themselves) for the API functions.
After all of this work to debug the truthbrush API and its innerworkings, we discovered that despite there being a date range pull function, the API always pulls from the latest post *up until* posts made on the date specified. Again, there was no way to know this until we discovered it ourselves as there was little documentation. Due to the requirements of the project to only pull and add a max of 25 items from an API per file run, this would not work for one of our standard APIs. We altered the code to use other APIs for our main API calls (in accordance with the rubric) and made a separate option to call the TruthBrush API as our extra credit API (since we did not want to throw away all of this work thus far).

When using the Polygon API, we discovered that despite following the instructions of the API from the documentation exactly, we were hitting rate limits when performing hourly data requests. After searching online through forms and on Stackoverflow, it was discovered that there is an ongoing issue with this function of the API and rate limits, and that the Polygon Team was actively working on solving it. This required us to pivot our project approach yet again from pulling hourly data to instead pulling day to day historical data, to work within the limits of the API.

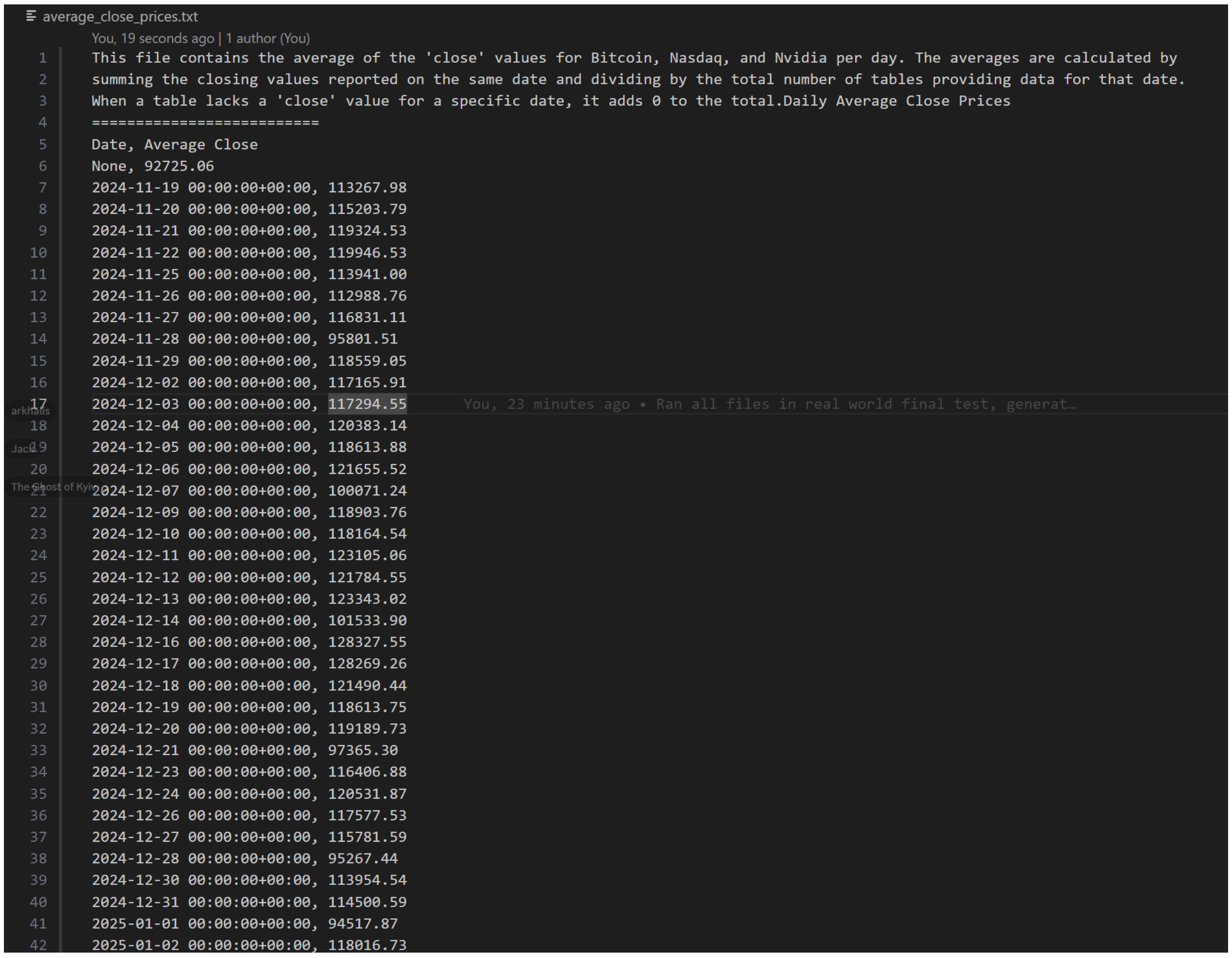
The coinbase API requires a higher level of authentication than any of the other APIs we were using because in addition to accessing historical market trends, the Coinbase API can also be used for Cryptocurrency trading, requiring users to provide a higher level of security to access the API. The API specifically required the creation of a JWT token to access data, something our group had never done before. Through a few calls with experienced programmer friends and colleagues, we successfully learned the requirements to create a JWT key from our API key, and use that to authenticate our calls.

A small hiccup in this process was learning that the rest of the Coinbase API (besides the historical data collection calls) uses a different API key than the legacy form of API key that is required for the aspects of the API we are using. Once we discovered this, we requested a new API key in the legacy format, and used that to generate our JWT token.

All of the responses to our chosen APIs came back in different date formats, and also required the input of different date formats (start/end of range), so our group was required to learn more about the datetime package to successfully convert, create and edit timestamps in the same format. Specifically, two of the APIs returned timestamps in Unix format, so we needed to use the datetime package to convert those to a readable format for use in the database.

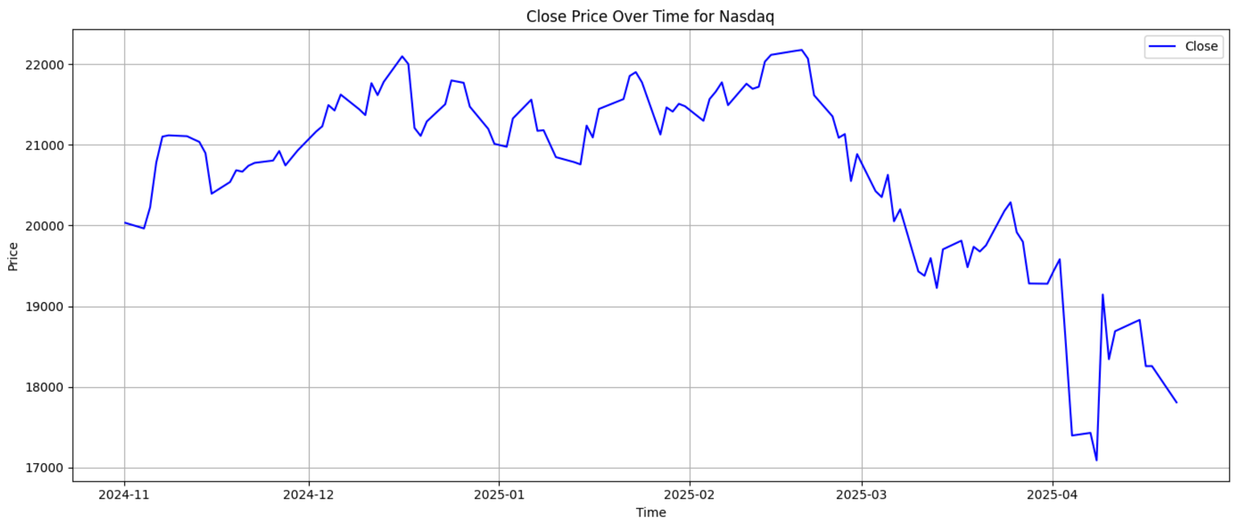
Finally, navigating the ambiguity and poor formatting of the project instructions and rubric was extremely difficult. Multiple times we needed to make significant alterations and changes to our project due to specificities in the instructions or rubric that were not made clear in lecture, or in both of the documents equally. For example, the only place that states we should have a separate file for visualizations/calculations and the initial data collection of the APIs is in the first row of the "Process The Data" section of the rubric. In all of our previous projects, we had kept all function calls and code in a singular file, so we were not expecting to be able to separate the code. This caused a lot of headaches as we attempted to make lots of accommodations to the main file, until realizing through Piazza and conversations with teaching staff that we could separate into multiple *files*. Having this laid out in the instructions, or clarified in lectures/discussions would have been appreciated.

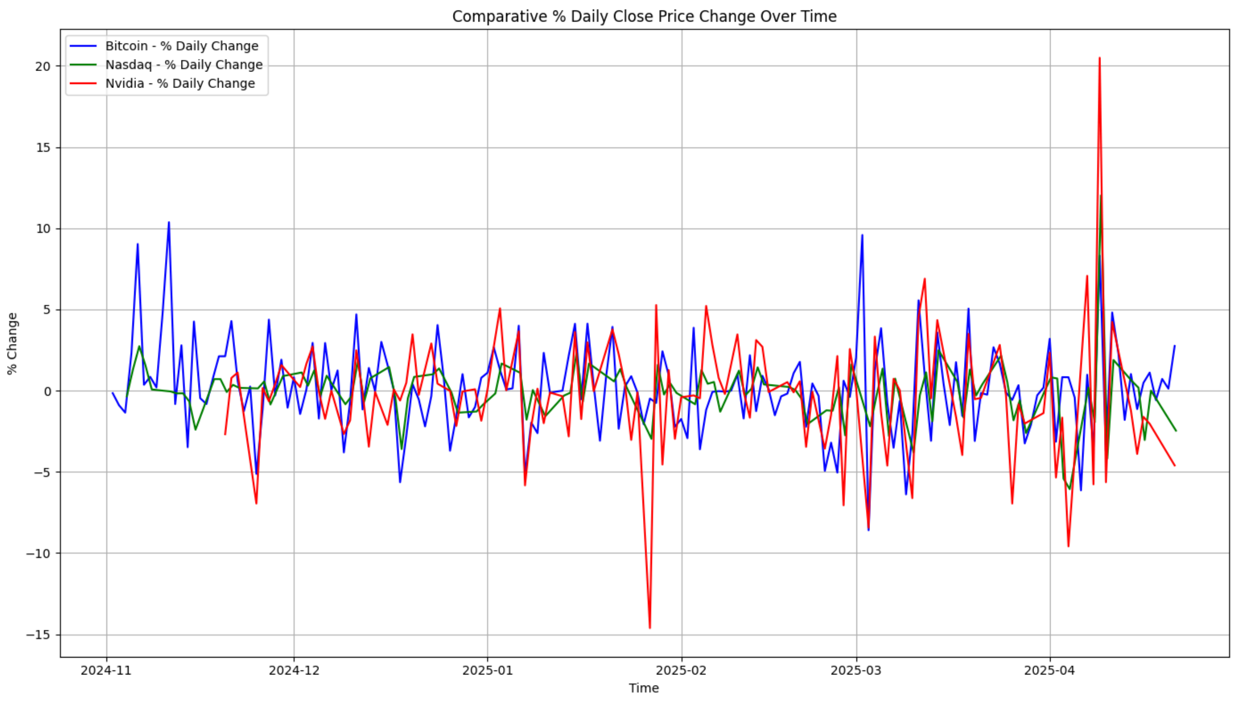
D. The calculations from the data in the database (i.e. a screenshot) (10 points)

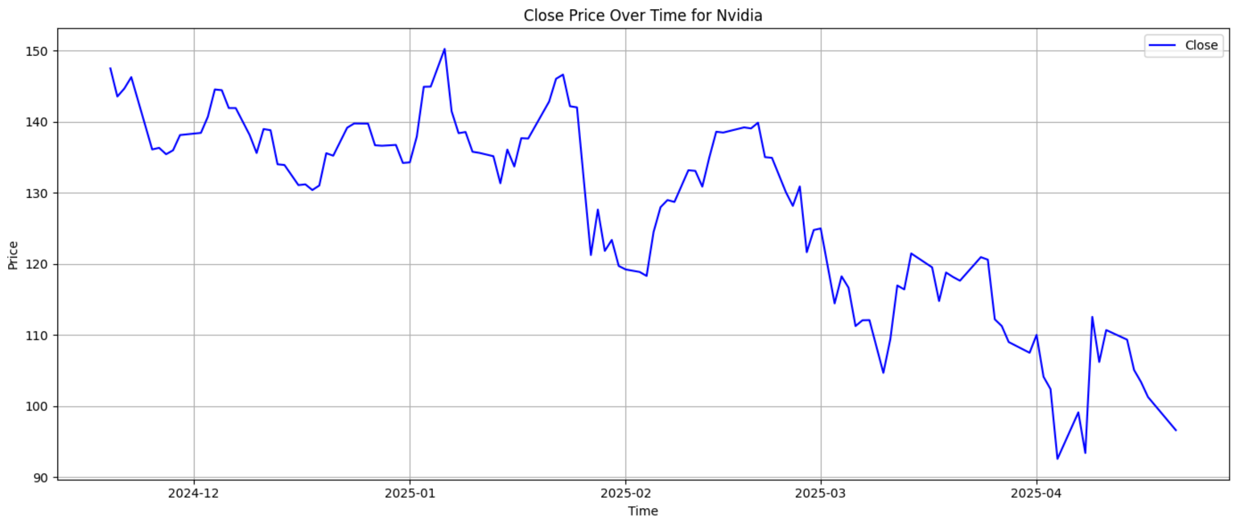


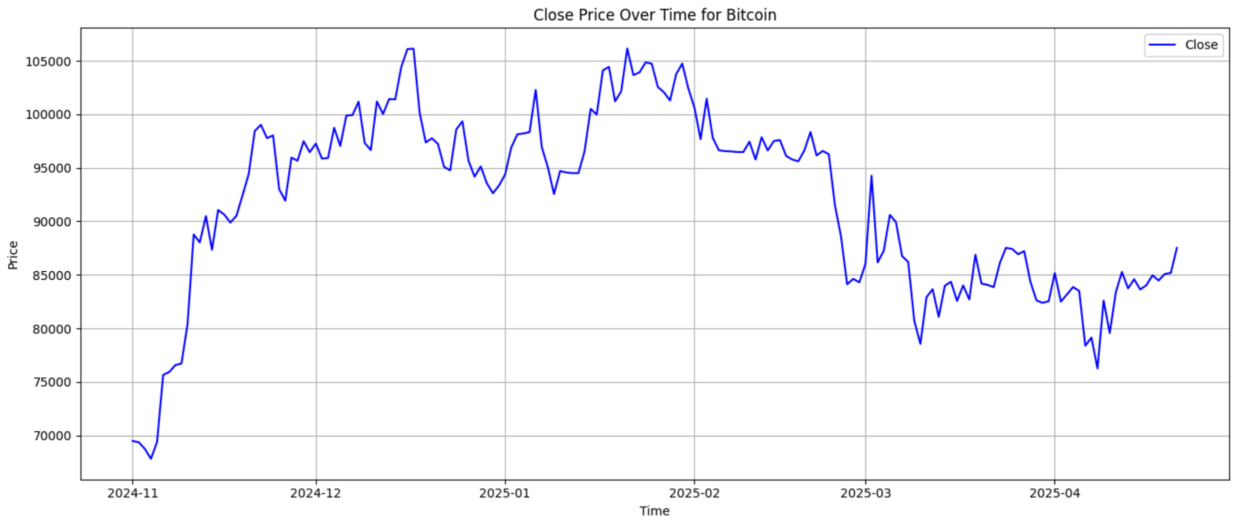
```
This file contains the average of the 'close' values for Bitcoin, Nasdaq, and Nvidia per day. The averages are calculated by
summing the closing values reported on the same date and dividing by the total number of tables providing data for that date.
When a table lacks a 'close' value for a specific date, it adds 0 to the total.Daily Average Close Prices
=========================
Date, Average Close
None, 92725.06
2024-11-19 00:00:00+00:00, 113267.98
2024-11-20 00:00:00+00:00, 115203.79
2024-11-21 00:00:00+00:00, 119324.53
2024-11-22 00:00:00+00:00, 119946.53
2024-11-25 00:00:00+00:00, 113941.00
2024-11-26 00:00:00+00:00, 112988.76
2024-11-27 00:00:00+00:00, 116831.11
2024-11-28 00:00:00+00:00, 95801.51
2024-11-29 00:00:00+00:00, 118559.05
2024-12-02 00:00:00+00:00, 117165.91
2024-12-03 00:00:00+00:00, 117294.55      You, 23 minutes ago • Ran all files in real world final test, generat…
2024-12-04 00:00:00+00:00, 120383.14
2024-12-05 00:00:00+00:00, 118613.88
2024-12-06 00:00:00+00:00, 121655.52
2024-12-07 00:00:00+00:00, 100071.24
2024-12-09 00:00:00+00:00, 118903.76
2024-12-10 00:00:00+00:00, 118164.54
2024-12-11 00:00:00+00:00, 123105.06
2024-12-12 00:00:00+00:00, 121784.55
2024-12-13 00:00:00+00:00, 123343.02
2024-12-14 00:00:00+00:00, 101533.90
2024-12-16 00:00:00+00:00, 128327.55
2024-12-17 00:00:00+00:00, 128269.26
2024-12-18 00:00:00+00:00, 121490.44
2024-12-19 00:00:00+00:00, 118613.75
2024-12-20 00:00:00+00:00, 119189.73
2024-12-21 00:00:00+00:00, 97365.30
2024-12-23 00:00:00+00:00, 116406.88
2024-12-24 00:00:00+00:00, 120531.87
2024-12-26 00:00:00+00:00, 117577.53
2024-12-27 00:00:00+00:00, 115781.59
2024-12-28 00:00:00+00:00, 95267.44
2024-12-30 00:00:00+00:00, 113954.54
2024-12-31 00:00:00+00:00, 114500.59
2025-01-01 00:00:00+00:00, 94517.87
2025-01-02 00:00:00+00:00, 118016.73
```

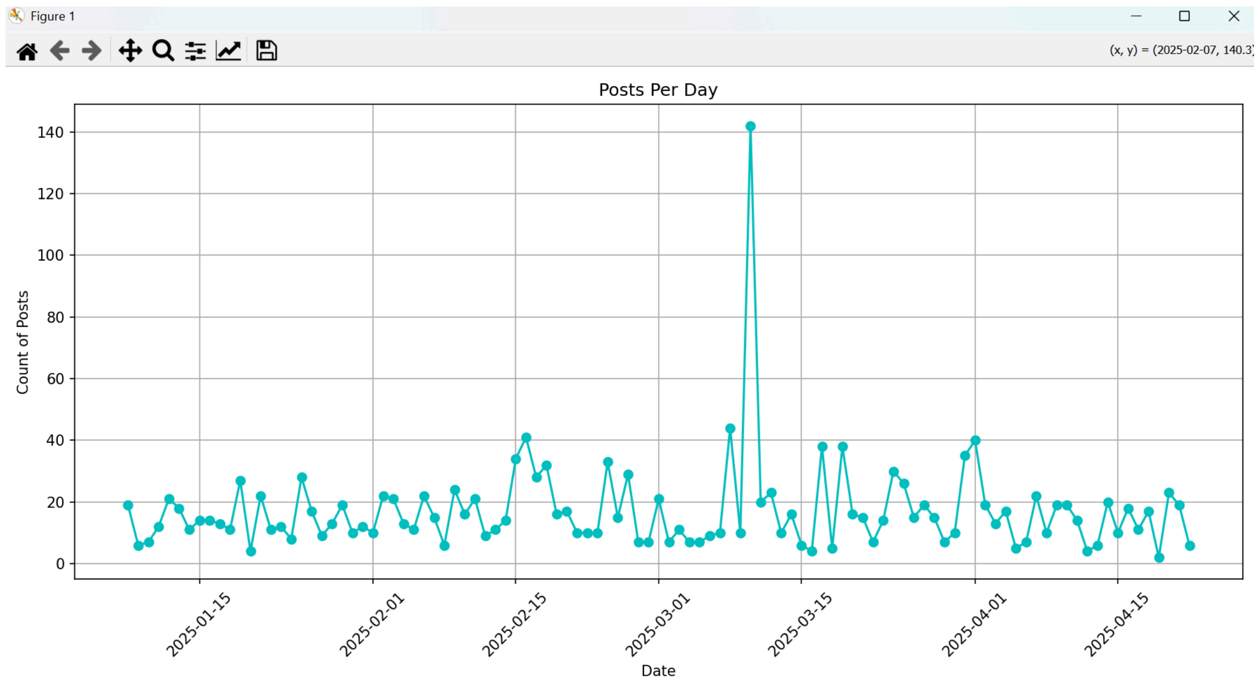
E. The visualization that you created (i.e. screenshot or image file) (10 points)






Close Price Over Time for Nvidia
Close
Price
Time
150
140
130
120
110
100
90
2024-12
2025-01
2025-02
2025-03
2025-04


Close Price Over Time for Bitcoin
Close
Price
Time
105000
100000
95000
90000
85000
80000
75000
70000
2024-11
2024-12
2025-01
2025-02
2025-03
2025-04



F. Instructions for running your code (10 points)

RECOMMENDED ORDER TO RUN FILES:

1. final _project.py (21x)
2. pull_truth.py
3. extra_credit.py
4. visualizations.py
5. truth_posts_visualization.py

Pre-requisites:

Packages needed for final_project.py:

- Http.client
- Os
- Sqlite3
- Datetime
- Json
- Polygon
- Jwt
- Cryptography.hazmat.primitives
- Time
- Secrets
- Pandas
- Matplotlib

Packages needed for pull_truth.py:

- To run extra credit API function, install the TruthBrush package by running `pip install git+https://github.com/stanfordio/truthbrush.git`
- **YOU WILL NEED TO PROVIDE YOUR OWN TRUTH SOCIAL USERNAME AND PASSWORD TO THE API INSIDE THE PACKAGE CODE OR AS ENV VARIABLE, PER THE DOCUMENTATION**

Pulling Data & Creating Database (final_project.py):

First, run the file get_data as many times as desired to pull data from the APIs based on the specified date range.
1. First, select which API you would like to pull data from on this run by typing options 1 - 3.
2. Then, input the start date and end dates to specify the range to pull data from. Note: You can pull a maximum of 25 days at one time.

To get a full 100 rows of data, the file will need to be ran **21 times,** 7 times per API, with 3 APIs. Here are the recommended date ranges to pull at least 100 market trading days up until 4/17/25. Note that the end date is not inclusive, and that these ranges will need to be pulled for **EACH API.**

Date Range 1: 2024-11-01 *to* 2024-11-26

Date Range 2: 2024-11-26 *to* 2024-12-21

Date Range 3: 2024-12-21 *to* 2025-01-15

Date Range 4: 2025-01-15 *to* 2025-02-09

Date Range 5: 2025-02-09 *to* 2025-03-06

Date Range 6: 2025-03-06 *to* 2025-03-31

Date Range 7: 2025-03-31 *to* 2025-04-22

The data returned in the file call will automatically be made into a json and added to the master database.

Pulling Extra Credit Data, Creating Database, and EC Calc + Viz (pull_truth.py & extra_credit.py):

Then, make sure to have the proper API packages installed from the prerequisites session, and run file pull_truth.py. This file will generate a json file, which is used by the file “extra_credit.py” to input all of the data into a database, and calculate the number of posts per day. Run “extra_credit.py”, which will also generate a CSV file with the output of the calculated data, obtained from the table “PostsPerDay”.

Creating Visualizations & Making Calculations on All Data (visualizations.py):

Run the file to create all needed visualizations and calculations. The file should output PNGs for each visualization, and average_close_price.txt. Note: Matplotlib is set to show visualizations when running the code. These will pop up in a separate window. **You must close this window in order for the program to continue and render the next visualization.**

G. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

truth_user_lookup (only used for debugging)
Inputs: One API Package, One User Handle (str)
Output: Returns the data pulled from the API and creates json
Desc: Pulls a user's basic information from Truthsocial and stores it in a .json file

truth_pull_posts(only used for extra credit)
Inputs: One API Package, One User Handle (str), one start date (str or datetime obj in yyy-mm-dd format)
Output: Returns the pulled posts in a dictionary, and creates json
Desc: Pulls a user's posts from Truthsocial and stores it in a .json file. Pulls posts from most recent to end date specified

> **Note:** Polygon class was created as we initially expected to make multiple calls to the Polygon API, so having API key aval as class variable would be helpful. Didn’t end up happening, but kept the code anyways

Polygon.__init__
Inputs: self
Output: None
Desc: Initializes Polygon API with API key provided in file

Polygon.get_stonks
Inputs: self, start date (str in in yyy-mm-dd format), end_date (str in yyy-mm-dd format)
Output: Creates json file of historical stock market data for the NASDAQ indice. Will provide open, high, low, and close data inside the json.
Desc: Initializes Polygon API with API key provided in file

get_token
Inputs: request_path (str, optional), request_method (str, optional), request_host (str, optional)
Output: returns JWT token generated with API token

Desc: Uses defined Coinbase API key to make a token request on the provided path, and return that token for use in other functions. If path, method and host are not defined, defaults to needed values to get token for Coinbase Candles.

Coin_candles
Inputs: coin (str), start_date (datetime), end_date (datetime)
Output: Creates json file of historical cryptocurrency market data for the coin specified. Will provide open, high, low, and close data inside the json.
Desc: Initializes Coinbase API with token provided from get_token, and makes request. Outputs data into json and saves.

get_stonks_finage
Inputs: stock (str), start_date (str in in yyy-mm-dd format), end_date (str in in yyy-mm-dd format)
Output: Creates json file of historical stock ticker data for the stock specified. Stock should be in ticker format (Ex: NVDA) Will provide open, high, low, and close data inside the json.
Desc: Initializes Finage API with key provided, and makes request. Outputs data into json and saves.

get_json_content
Input: filename: name of file to be opened
Output: json dictionary OR an empty dict if the file could not be opened
Desc: opens file file, loads content as json object.

set_up_database
Parameters: db_name: str (The name of the SQLite database)
Returns: Tuple (cursor, connection)
Desc: Sets up a SQLite database connection and cursor.

set_up_market_coin_table
Parameters: cur: SQL Cursor, conn: SQL Connector
Returns: Nothing
Desc: Sets up a table for each of the APIs we are using, and each specific column.

set_up_posts_table
Parameters: cur: SQL Cursor, conn: SQL Connector
Returns: Nothing
Desc: Sets up a table for each of the content of the posts, as well as a table for measure of engagement and one referring to the post count per day.

add_posts_to_table
Parameters: data: dictionary of json file from Truthbrush API, cur: SQL Cursor, conn: SQL Connector
Returns: Nothing
Desc: Sets up a table for each of the content of the posts, as well as a table for measure of engagement and one referring to the post count per day.

add_criptodata_to_table
Parameters: coin: dictionary of json file from Bitcoin API, cur: SQL Cursor, conn: SQL Connector
Returns: Nothing
Desc: For each item of the dictionary with the data, adds data on each row, using the date of the data (YYYYMMDD format) as Integer Primary Key, and converts the Unix timestamp into a readable date to be entered into the database.

add_nvdadata_to_table
Parameters: coin: dictionary of json file from NVIDIA Stock API, cur: SQL Cursor, conn: SQL Connector
Returns: Nothing
Desc: For each item of the dictionary with the data, adds data on each row, using the date of the data (YYYYMMDD format) as Integer Primary Key, and converts the Unix timestamp into a readable date to be entered into the database.

add_stockdata_to_table
Parameters: coin: dictionary of json file from NASDAQ Stock API, cur: SQL Cursor, conn: SQL Connector
Returns: Nothing
Desc: For each item of the dictionary with the data, adds data on each row, using the date of the data (YYYYMMDD format) as Integer Primary Key, and converts the Unix timestamp into a readable date to be entered into the database

menu_apis
Parameters: Nothing
Returns: Selected option from menu
Desc: Function that prompts the user to select an API to be used by the file. Returns one of the described options as a string to be used by upcoming functions.

get _dates
Parameters: Nothing
Returns: start_date: datetime object that contains the beginning of specified timeframe, end_date: datetime object that contains the end of specified timeframe
Desc: Function that prompts the user to provide a date (year, then month, then day) to start and end the timeframe of data that must be collected from the API, including some accuracy tests, such as ensuring that start_date is before end_date, that end_date is not past today's date, and to ensure that the longest period we will observe is 25 days, in order to comply with assignment guidelines.

plot_individual_table
Parameters: dataframe object, table_name from the SQL database

Returns: Viz
Desc: This function generates individual plots for each selected table

plot_comparative_tables
Parameters: connection to the SQL database, a list of tables from the SQL database
Returns: Viz
Desc: This function generates a comparative plot for the daily percent change of close prices across multiple tables. It retrieves the data from each table, calculates the daily percent change, and plots them on the same graph.

plot_tables
Parameters: connection to the SQL database, a list of tables from the SQL database
Returns: Viz
Desc: This function retrieves data from the specified tables in the SQLite database, processes it, and generates plots. It retrieves the data from each table, processes the timestamps, and generates individual plots for each table. It also generates a comparative plot for the daily percent change of close prices across all tables. It then saves the plots as PNG files and displays them.

calculate_daily_averages
Parameters: connection to the SQL database
Returns: pd.read_sql_query(query, connection)
Desc: This function calculates the daily average close prices for Bitcoin, Nasdaq, and Nvidia. It retrieves the data from the database, joins the tables on the date, and calculates the average close price for each date.

write_averages_to_file
Parameters: dataframe object, filename of a text file
Returns: a text file with written changes
Desc: This function writes the daily average close prices to a text file. It includes a header explaining the contents of the file and formats the data for readability.

visualize_averages
Parameters: a dataframe object
Returns: Viz
Desc: This function visualizes the daily average close prices using a line plot. It ensures the date column is in datetime format, handles missing values, and plots the data.

fetch_post_data
Parameters: connection to the SQL database
Returns: a dataframe object
Desc: This function establishes a connection with a database that contains truths counts, reads it, and creates the appropriate dataframe object

visualize_posts_per_day
Parameters: a dataframe object

Returns: Viz
Desc: This function is used to visualize the posts per day graph with the number of truth posts on the Y axis and the timeline on the X axis

main (final_project)
Parameters: Nothing
Returns: Nothing
Desc: Sets up database and tables, then inputs data from json generated from "pull_truth.py" into a dedicated table.

main (extra_credit.py)
Parameters: Nothing
Returns: Nothing
Desc: Sets up database and tables, then interacts with users, to then identify which API to pull data from and for which period of time, then inputs data from generated json into a dedicated table.

main (visualizations.py)
Parameters: Nothing
Returns: Nothing
Desc: Establishes a connection to a database, iterates through tables in the table list to create individual tables and a comparative table, creates a new text file where it writes down the calculations, visualizes a graph with the calculations and then terminates the connection with a database

You must also clearly document all resources you used. The documentation should be of the following form (20 points)

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
4/9/25	Truthbrush API package installed but not being recognized by Python	Over Discord with Programmer Friend & stackoverflow thread	Edited PATH Windows environment variable to reflect path of package (not with other packages) and Python could recognize the file, yes
4/9/25	Truthbrush API requiring	ChatGPT to discuss how to import and	ChatGPT showed me how the datetime package worked and how to use it inside of my code

	datetime object	use datetime package	to correctly make the API call, yes
4/14/25	Polygon API returns timestamps in Unix, needs to be datetime	Asked ChatGPT to help me figure out what format timestamp was in, and how to conv to DT	ChatGPT helped me correctly identify the timestamp as Unix, and then suggested how I could convert the unix timestamp to datetime, so yes
4/14/25	Had to add a timezone offset of a few hours to account for timezone	Asked ChatGPT how to add a timezone offset to an existing datetime package	ChatGPT suggested how I account for offsets in datetime objects, by manually adding hours, so yes
4/14/25	Did not know syntax for SQL database setup and json reference	Used code in HW6 and HW7 for reference	Get_json_content and setup_database from HW6 and HW7 to reference
4/16/25	Coinbase API required a special kind of authenticatio n that I had not used before, JWT token.	Watched a YouTube video on the package to try and understand how to authenticate	YouTube video did explain how to make a few function calls, and clarified a few questions I had about the structure of the data, but ultimately did not provide the info I needed as it was not using JWT token auth, so **NO**
4/16/25	Coinbase API required a special kind of authenticatio	Called my experienced Python Developer friend to	Friend helped me understand how to make and test calls using Postman instead of Python, taught me how to create a JWT token, and helped me write a

	n that I had not used before, JWT token.	understand how to make the JWT token, used Postman to test API calls	basic auth function using the documentation, so yes
4/17/25	Discovered that Truthbrush API would not pull posts from any other point beyond the most recent post, limiting the amount of time we can go back in time due to account rate limits	Dove into truthbrush API documentation and source code	Tried to modify code ourselves to no avail, so **NO**, ended up abandoning Truthbrush API for the 3 main APIs, saved it for extra credit
4/17/25	Discovered that there was a bug in the Polygon API that would make separate API calls for each hour, instead of one call getting all hours of the day.	Used Polygon API docs and Polygon Community Knowledge Base (was on Discord, no specific issue link)	Discovered that what we were experiencing was an ongoing bug that users are facing, and the team is working on it, Not a problem for anyone but free users. Ended up pivoting to pulling day intervals instead so yes
4/22/25	Was not sure how to perform specific join functions	Used ChatGPT to debug and suggest code snippets.	Ultimately suggested a fix and helped understand how to query the SQL database appropriately.

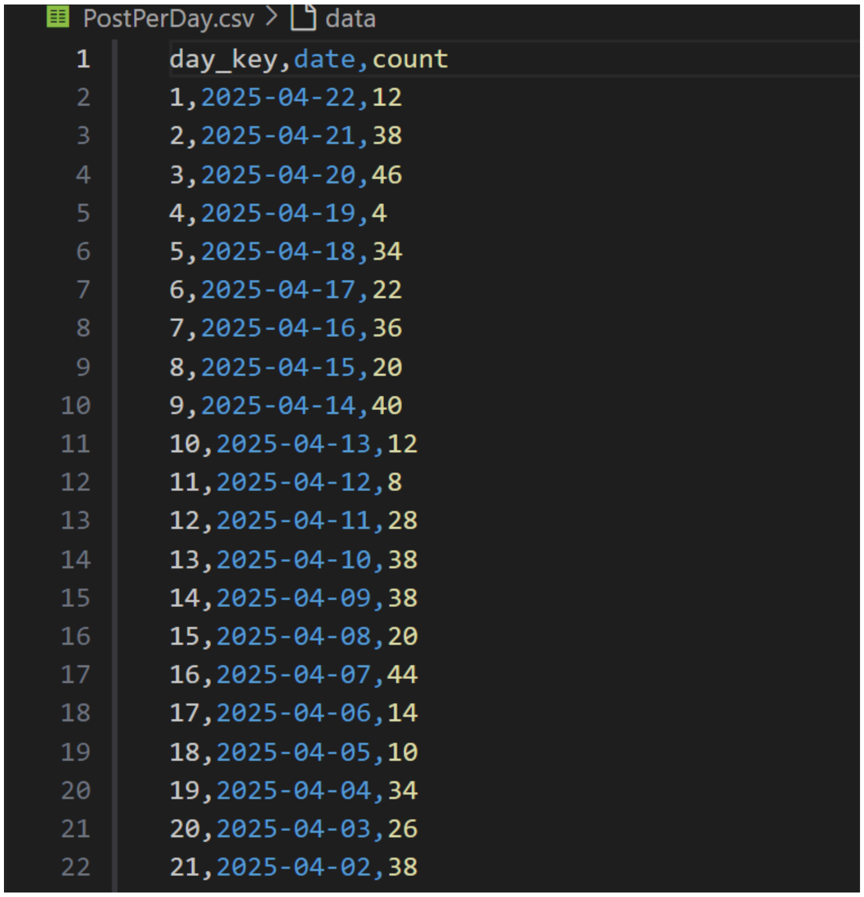
	based on datetime formatted objects inside database		
4/22/25	Didn't know how to calculate number of rows in SQL database with certain variable (eg, how many rows contain the timestamp of "2025-04-18" ?	Used ChatGPT to debug and suggest code snippets	Got code to check if date exists in table, and then add to counter, so YES

Bonus Opportunities

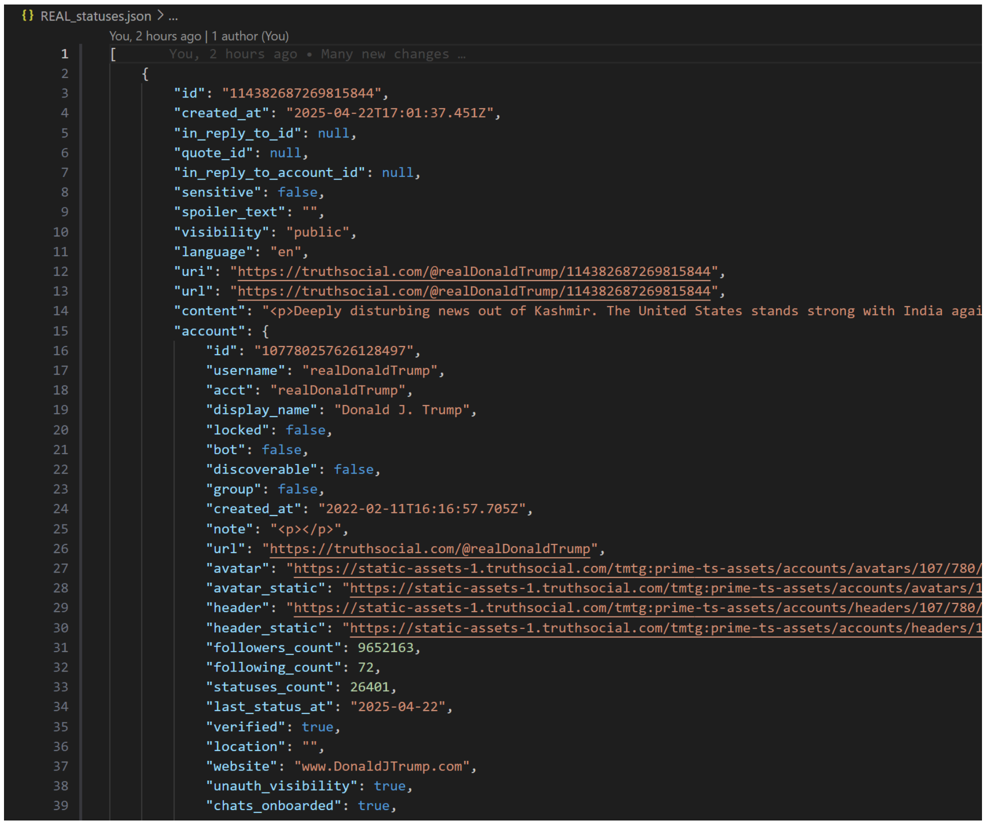
- Bonus A: Additional API sources (Max 30 points)

  Earn up to 30 points for an additional API. You have to gather 100 items from the API and store it in the database. You must calculate something from the data in the database. You must write out the calculation in a file.

    To accomplish the extra credit option, we used the information of posts pulled from the TruthBrush API and computed the number of posts from Donald Trump for each day from the 8th of January to the current day (April 22nd).


PostPerDay.csv > data

```
day_key,date,count
1,2025-04-22,12
2,2025-04-21,38
3,2025-04-20,46
4,2025-04-19,4
5,2025-04-18,34
6,2025-04-17,22
7,2025-04-16,36
8,2025-04-15,20
9,2025-04-14,40
10,2025-04-13,12
11,2025-04-12,8
12,2025-04-11,28
13,2025-04-10,38
14,2025-04-09,38
15,2025-04-08,20
16,2025-04-07,44
17,2025-04-06,14
18,2025-04-05,10
19,2025-04-04,34
20,2025-04-03,26
21,2025-04-02,38
```



{} REAL_statuses.json > ...

You, 2 hours ago | 1 author (You)

```
[        You, 2 hours ago • Many new changes …
    {
        "id": "114382687269815844",
        "created_at": "2025-04-22T17:01:37.451Z",
        "in_reply_to_id": null,
        "quote_id": null,
        "in_reply_to_account_id": null,
        "sensitive": false,
        "spoiler_text": "",
        "visibility": "public",
        "language": "en",
        "uri": "https://truthsocial.com/@realDonaldTrump/114382687269815844",
        "url": "https://truthsocial.com/@realDonaldTrump/114382687269815844",
        "content": "<p>Deeply disturbing news out of Kashmir. The United States stands strong with India agai
        "account": {
            "id": "107780257626128497",
            "username": "realDonaldTrump",
            "acct": "realDonaldTrump",
            "display_name": "Donald J. Trump",
            "locked": false,
            "bot": false,
            "discoverable": false,
            "group": false,
            "created_at": "2022-02-11T16:16:57.705Z",
            "note": "<p></p>",
            "url": "https://truthsocial.com/@realDonaldTrump",
            "avatar": "https://static-assets-1.truthsocial.com/tmtg:prime-ts-assets/accounts/avatars/107/780/
            "avatar_static": "https://static-assets-1.truthsocial.com/tmtg:prime-ts-assets/accounts/avatars/1
            "header": "https://static-assets-1.truthsocial.com/tmtg:prime-ts-assets/accounts/headers/107/780/
            "header_static": "https://static-assets-1.truthsocial.com/tmtg:prime-ts-assets/accounts/headers/1
            "followers_count": 9652163,
            "following_count": 72,
            "statuses_count": 26401,
            "last_status_at": "2025-04-22",
            "verified": true,
            "location": "",
            "website": "www.DonaldJTrump.com",
            "unauth_visibility": true,
            "chats_onboarded": true,
```

- Bonus B: Additional visualizations (Max 30 points)
  Earn up to 15 points for each additional visualization.