Project Goals:

- Planned APIs/Websites: The project was designed to work with several APIs to collect diverse data sets. The APIs include a weather data API (such as Open-Meteo) for temperature, precipitation, air quality data, and geocoding data API for pollution metrics, and a financial data API (MarketStack) for stock market data.
- Planned Data to Gather: The goal was to gather and integrate different types of data into a cohesive database. This included weather data (temperature and precipitation), air quality data (PM10, PM2.5, and other pollutants), geocoding (latitude and longitude of cities) and stock market data (opening, closing, high, low prices, and volume).

Achieved Goals:

- APIs/Websites Worked With: Successfully worked with the MarketStack API for stock data, and a weather API for pollution, historical weather, and geocoding data.
- Data Gathered: Collected and stored data in a SQLite database, including temperature, precipitation, air quality indicators (like PM10, and PM2.5), and stock market data. Also gathered geocoding data for various cities to relate the environmental data with specific locations.

Problems Faced:

- API Limitations: Potential limitations in API data availability or rate limits.
- Data Consistency: Challenges in ensuring data consistency and integrity, especially when dealing with data from multiple sources.
- Database Design: Complexity in designing a relational database schema that effectively links weather, air quality, and stock data.
- Data Collection and Processing: Difficulties in automating the collection of data for different dates and handling exceptions or errors in data retrieval.

Calculations from the Database:

- Calculations were performed using SQL queries to aggregate and analyze data, such as average temperature, rainfall, air quality indices, and stock prices. The calculations include average measurements per month, temperature variance, and stock market volatility.
- Screenshot of calculated data output:

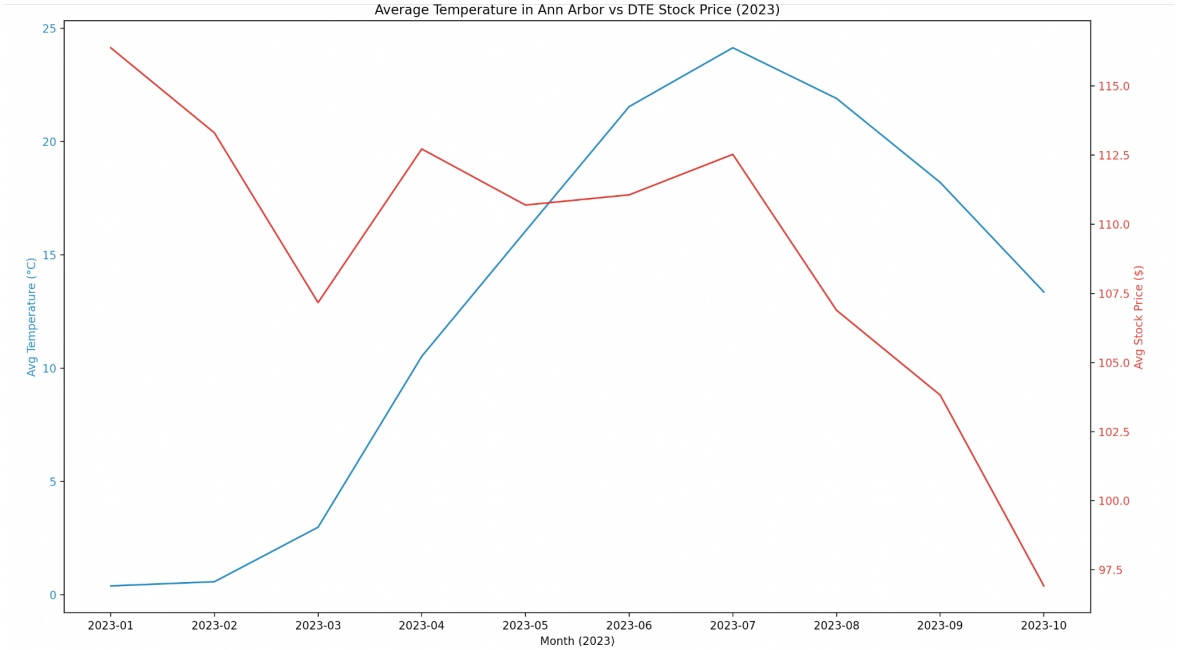
calculated_data.txt

```
Average Temperature in Ann Arbor vs Stock Price (2023)
     Month  AvgTemperature  AvgRainfall  AvgStockPrice
0  2023-01        0.395000     0.220000     116.382447
1  2023-02        0.578947     0.210526     113.305592
2  2023-03        2.982609     0.004348     107.166454
3  2023-04       10.521053     0.157895     112.722171
4  2023-05       16.045455     0.063636     110.693977
5  2023-06       21.538095     0.457143     111.062442
6  2023-07       24.135000     0.350000     112.524313
7  2023-08       21.900000     0.021739     106.889674
8  2023-09       18.195000     0.045000     103.821594
9  2023-10       13.360000     0.140000      96.918098

Air Quality (PM10 & PM2.5) in Ann Arbor and Stock Price (2023)
     Month    AvgPM10    AvgPM25  AvgTemperature  AvgStockPrice
0  2023-01  14.950000  10.370000        0.395000     116.382447
1  2023-02  13.815789   9.542105        0.578947     113.305592
2  2023-03  16.617391  11.521739        2.982609     107.166454
3  2023-04  16.500000  11.340000       10.000000     111.005250
4  2023-05  21.500000  14.950000       26.200000     106.902500
5  2023-06  37.666667  26.285714       21.538095     111.062442
6  2023-07  22.005000  15.290000       24.135000     112.524313
7  2023-08  20.447826  14.173913       21.900000     106.889674
8  2023-09  17.095000  11.800000       18.195000     103.821594
9  2023-10  13.925000   9.645000       13.360000      96.918098

Average Rainfall in Ann Arbor vs. Air Quality Index (2023)
     Month  AvgRainfall    AvgPM10
0  2023-01     0.141935  13.664516
1  2023-02     0.142857  12.650000
2  2023-03     0.009677  15.725806
3  2023-04     0.243750  16.400000
4  2023-05     0.109091  30.818182
5  2023-06     0.320000  34.543333
6  2023-07     0.238710  21.538710
7  2023-08     0.022581  19.738710
8  2023-09     0.030000  17.056667
9  2023-10     0.144444  12.618519

Temperature Variance in Ann Arbor vs. Stock Market Volatility (2023)
     Month  TempVariance  StockVolatility
0  2023-01    170.822528         0.018104
1  2023-02    172.430580         0.018018
2  2023-03    110.121125         0.023105
3  2023-04     35.160524         0.016350
4  2023-05     48.444312         0.015499
5  2023-06     89.340164         0.018481
6  2023-07    131.628155         0.016316
7  2023-08     86.610998         0.014805
8  2023-09     37.720395         0.015984
9  2023-10     27.405722         0.023988
```

Visualizations Created:

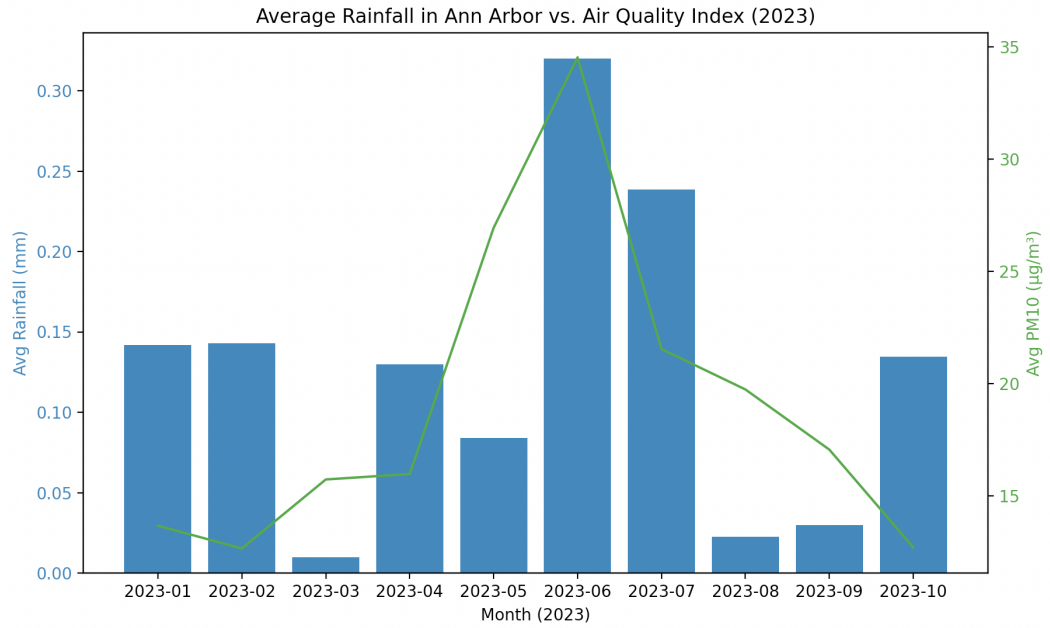
- Four visualizations were created using matplotlib in Python:

- Average Temperature in Ann Arbor vs DTE Stock Price (2023): A line plot showing the relationship between average temperature and average stock price.

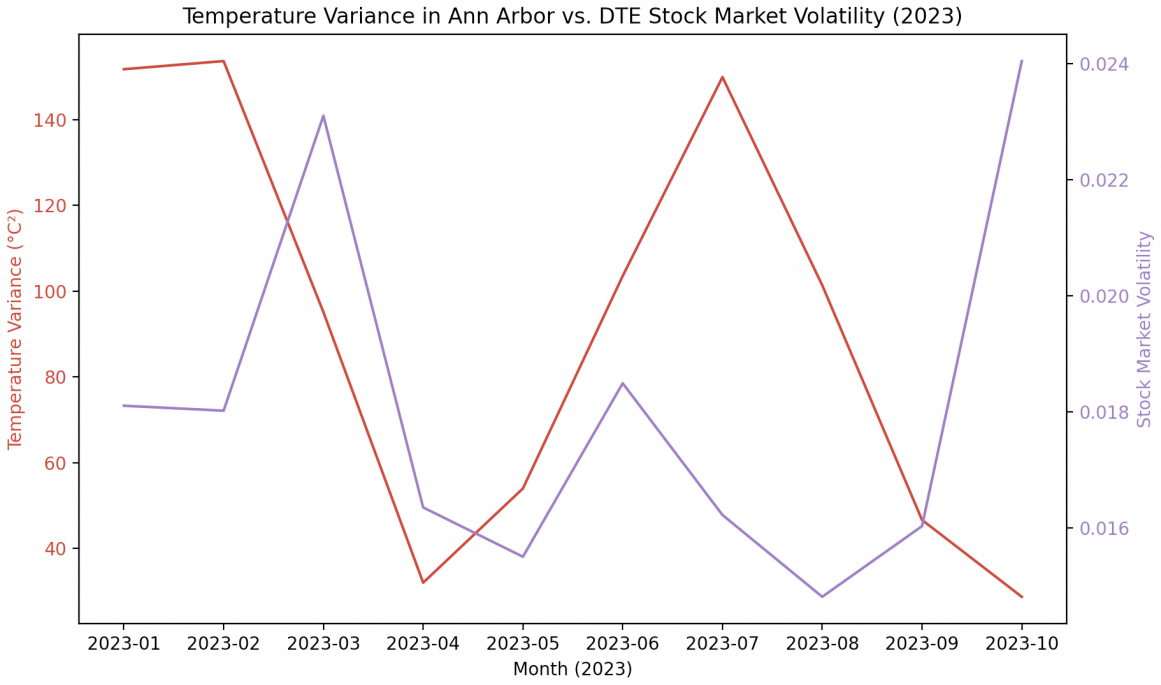


- Air Quality (PM10 & PM2.5) in Ann Arbor and DTE Stock Price (2023): A bar and line plot combining air quality indices and stock price.

- Average Rainfall in Ann Arbor vs. Air Quality Index in Ann Arbor (2023): A bar and line plot showing the relationship between average rainfall and PM10 levels.



- Temperature Variance in Ann Arbor vs. DTE Stock Market Volatility (2023): A plot comparing temperature variance with stock market volatility.



Documentation:

api_scraping.py:

Function: fetch_stock_data_for_date

Fetches stock data for a specific date from the MarketStack API. It queries the end-of-day (EOD) data for a specified stock symbol.
Inputs:

- date (datetime.date): The date for which the stock data is to be fetched.
- symbol (str, optional): The stock symbol to query. Defaults to "DTE".

Outputs:

- Returns a dictionary containing the stock data (open, close, high, low prices, and volume) if the API call is successful and data is available; otherwise, returns None.

Function: fetch_air_quality_data_for_date

Fetches air quality data for a specific date from a designated air quality API.
Inputs:

- date (datetime. date): The date for which air quality data is to be fetched.

Outputs:

- Returns a JSON response containing air quality data (like PM10, PM2.5 levels, etc.) if the API call is successful; otherwise, returns None.

Function: get_existing_dates

Retrieves a set of dates for which temperature data already exists in the database.
Inputs:

- db_connection (sqlite3.Connection): A SQLite database connection object.

Outputs:

- Returns a set of datetime.date objects representing dates for which data already exists in the database.

Function: calculate_sample_dates

Calculates a set of dates for data sampling throughout a specified year range, excluding dates for which data already exists in the database.
Inputs:

- start_date (datetime.date): The start date of the range.
- end_date (datetime.date): The end date of the range.
- total_samples (int): The total number of samples needed.
- existing_dates (set of datetime.date): A set of dates for which data already exists.

Outputs:

- Returns a list of datetime.date objects representing the dates to be sampled.

Function: main

Serves as the main entry point of the script. It establishes a connection to the SQLite database, creates necessary tables if they don't exist, calculates sample dates, and fetches weather, air quality, and stock data for those dates. It then inserts this data into the database.
Inputs:

- None.

Outputs:

- None. (This function performs operations like database creation, data fetching, and insertion but does not return any value.)

location_api_scraping.py

Function: get_lat_lon

Fetches latitude and longitude for a given city name using a geocoding API.
Inputs:

- city_name (str): The name of the city for which to find latitude and longitude.

Outputs:

- Returns a tuple containing latitude and longitude (both float) if the city is found; otherwise, returns (None, None).

Function: insert_city_data

Inserts data for a city into the locations table in the SQLite database. It first calls get_lat_lon to get the latitude and longitude of the city.
Inputs:

- city_name (str): The name of the city to be inserted into the database.
- cursor (sqlite3.Cursor): A SQLite cursor object to execute database operations.

Outputs:

- Returns True if the city data is successfully inserted; otherwise, returns False.

Function: main

Serves as the main entry point of the script. It connects to a SQLite database, creates a locations table if it does not exist, and then iterates through a list of city names. For each city, it checks if the city already exists in the database. If not, it attempts to insert the city data into the locations table. The script processes up to 25 cities.
Inputs:

- None.

Outputs:

- None. The function performs operations like database connection, table creation, and data insertion but does not return any value.

calc_visualizations.py

Function: main

The main function serves as the central point for executing a series of data analysis and visualization tasks. It connects to a SQLite database, retrieves data through SQL queries, and then uses this data to create several visualizations. These visualizations relate to temperature, precipitation, air quality, and stock prices. Additionally, it writes the query results to a text file.
Inputs:

- None.

Operations:

Database Connection: Opens a connection to the SQLite database weather_data_final.db.
Data Retrieval: Executes SQL queries to fetch data about temperature, precipitation, air quality, and stock prices. The data is grouped by month for the year 2023.
Visualization Creation:

- Visualization 1: Plots average temperature against average stock price for each month.
- Visualization 2: Creates a bar and line plot showing average air quality indices (PM10, PM2.5) and stock prices.
- Visualization 3: Displays a bar and line plot comparing average rainfall with air quality index (PM10) for each month.
- Visualization 4: Illustrates the relationship between temperature variance and stock market volatility.

File Writing: Writes the data used in the visualizations to a text file calculated_data.txt.

Outputs:

- Four matplotlib visualizations are displayed on the screen.
- A text file calculated_data.txt containing the data used in the visualizations.

Usage Notes:

- Ensure that the SQLite database weather_data_final.db is present and correctly formatted with the necessary tables and data.
- The function uses matplotlib for plotting, so an appropriate setup for matplotlib is required (e.g., running in an environment that supports plotting).
- The function writes to a text file in the working directory, so write permissions are needed.
- This function is designed to run as a standalone script and assumes that the required libraries (matplotlib, pandas, numpy, sqlite3) are installed.

Documents:

2023-012-01	How to fetch stock data using MarketStack	MarketStack API Documentation https://marketstack.com/documentation	Successfully fetched stock data
2023-12-3	Parsing JSON in Python	Stack Overflow Thread on JSON Parsing https://stackoverflow.com/questions/7771011/how-can-i-parse-read-and-use-json-in-python	Resolved JSON parsing issue
2023-12-03	Handling API rate limits	Reddit Discussion on API Rate Limiting https://www.reddit.com/r/SoftwareEngineering/comments/16jsl8q/rate_limiting_an_api_properly/	Implemented rate limiting handling
2023-12-06	Visualizing data with matplotlib	Matplotlib Official Tutorials https://matplotlib.org/stable/index.html	Created visualizations as planned
2023-12-11	Storing API data in SQLite	SQLite Python Tutorial https://docs.python.org/3/library/sqlite3.html	Data stored successfully in SQLite