StockSense

Ansruta Bohra, Tarini Bengani, Rhythem Jain

SI 206 - Professor Barbara Ericson

December 16, 2024

Goals

As UX and Business students, we were motivated to create a comprehensive tool that bridges the gap between financial data accessibility and actionable insights. Existing platforms which are similar are often fragmented, focusing either on data presentation or basic analysis, and exclusive to only professionals. Bloomberg for instance focuses on providing historical and real-time stock data but offers limited advanced analysis or behavioral insights, like insider transactions. Bloomberg, on the other hand, is all-encompassing but caters to professionals with expensive subscriptions, making it less accessible for beginners and retail investors. Our goal with StockSense was to combine historical stock data, dividend information, insider activities, and sentiment-driven news analysis in one cohesive interface, accessible to novice investors. Our primary objective was to integrate data collection, storage, analysis, and visualization, allowing users to draw meaningful conclusions about stock performance.

Original Goals

Goal #1: Selected APIs - we initially wanted to use the following APIs to get stock-related data:

a. **Finnhub API**
    i. Recommendation trends (no. of analysts that recommended buy, hold, sell, strong buy, or strong sell)
    ii. Insider sentiment (decimal value based on insider opinions on a company)
    iii. Peers (ticker symbol of other companies in the same industry)
b. **World News API**
    i. News article title
    ii. News article full text
    iii. News article summary

c. **Tisane API**

i. Polarity (analyses text and returns “positive”, “negative”, or “mixed” based on sentiment)

d. **Financial Modeling Prep API**

a. Financials as Reported (gross profit across years)

Goal #2: Data calculations - we planned to do the following calculations using the API data:

a. **Average recommendation trend** - Giving a weight to “buy”, “sell”, “hold”, etc. (e.g. buy = 1, sell = -1) and then adding all these weighted recommendations together and dividing it by the no. of recommendations for each company
b. **Average polarity** - Take 15 news articles for every company and put every article’s text through the sentiment analysis API. Then give a weight to all potential sentiment values (e.g. “positive” - 1, “negative” = -1), and calculate the average of all the sentiment values across the 15 articles
c. **Final score** - Total of average recommendation trend, average of average polarity, and insider sentiment for each company’s stock
d. **Maximum final score** - Best performing stock in an industry according to final score
e. **Minimum final score** - Worst performing stock in an industry according to final score

<u>Achieved Goals</u>

Goal #1: Selected APIs - we ended up using the following APIs to get stock-related data:

a. **Finnhub API** (same API as original plan, but with slightly different data as mentioned below in order to meet project requirements)

i. Insider transactions (transaction dates, price, share, and change)

b. **Financial Modeling Prep API** (same API as original plan, but with slightly different data as mentioned below in order to meet project requirements)
    i. Dividend date & amount
c. **MarketAux API** (used this financial news API in place of World News API & Tisane API because it automatically gave news articles with sentiment analysis which made it easier for us to work with it)
    i. News URL, title, and sentiment
d. **YahooFinance API** (added this new API to include stock price data as thought it would be important information) (library accessed via yfinance)
    i. Stock price date, open, close, high & low values

Goal #2: Data calculations - we ended up doing the following calculations using the API data:

1. Average closing prices per stock: Average, minimum, and maximum prices for each stock.
2. Average news sentiment per stock: Average sentiment scores calculated from news articles.
3. Closing price trends over time: Tracks price trends over time with minimum and maximum values.
4. Total dividends per stock: Total dividends distributed by each stock.
5. Total insider transactions per stock: Number of insider transactions for each stock.

We successfully incorporated these four APIs to gather critical data on stock prices, dividends, insider trading, and market sentiment. By picking these diverse datasets, we were able to create a platform that provides a well-rounded yet simple way to analyze stocks. For the scope of this project, we deliberately excluded hundreds of advanced metrics. This helped us achieve

our goal of keeping the platform beginner-friendly, and making financial analysis more intuitive and approachable.

Problems Faced

As intermediate-level programmers, we encountered several technical and conceptual hurdles while developing StockSense. One of the first difficulties was setting up the development environment as we were not very familiar with it. Installing the required packages and libraries (e.g., yfinance, matplotlib, pandas, requests), as well as troubleshooting version compatibility issues took longer than expected. For example, some packages were already installed but outdated, which caused errors when running certain parts of the code. While these installations were eventually successful, the process delayed our progress in the early stages of the project.

Another significant issue was handling duplicate data in our SQLite database, which violated the project instructions. Since the four companies we ran our project for (Apple, Tesla, Microsoft, and Google) were frequently mentioned together in the same news article fetched from Marketaux, the Article Title field often contained duplicates. To resolve this, we removed the Article title column from the table, as it was not necessary for our analysis. We focused on retaining only the sentiment scores, which were essential for calculating the average sentiment for each stock.

API usage posed another major problem. Although we deliberately selected APIs with higher request limits, such as MarketAux and Finnhub, we repeatedly encountered situations where our API keys maxed out. This problem often occurred during testing and debugging, where frequent reruns of the data generation script (data_generator.py) consumed the available request quota.

Lastly, identifying and selecting APIs that were coherent and made sense conceptually was not straightforward. Ensuring the APIs supported both calculation and visualization

requirements called for significant trial-and-error. Additionally, compiling data from multiple APIs into a cohesive database was complex since they all had different data structures.

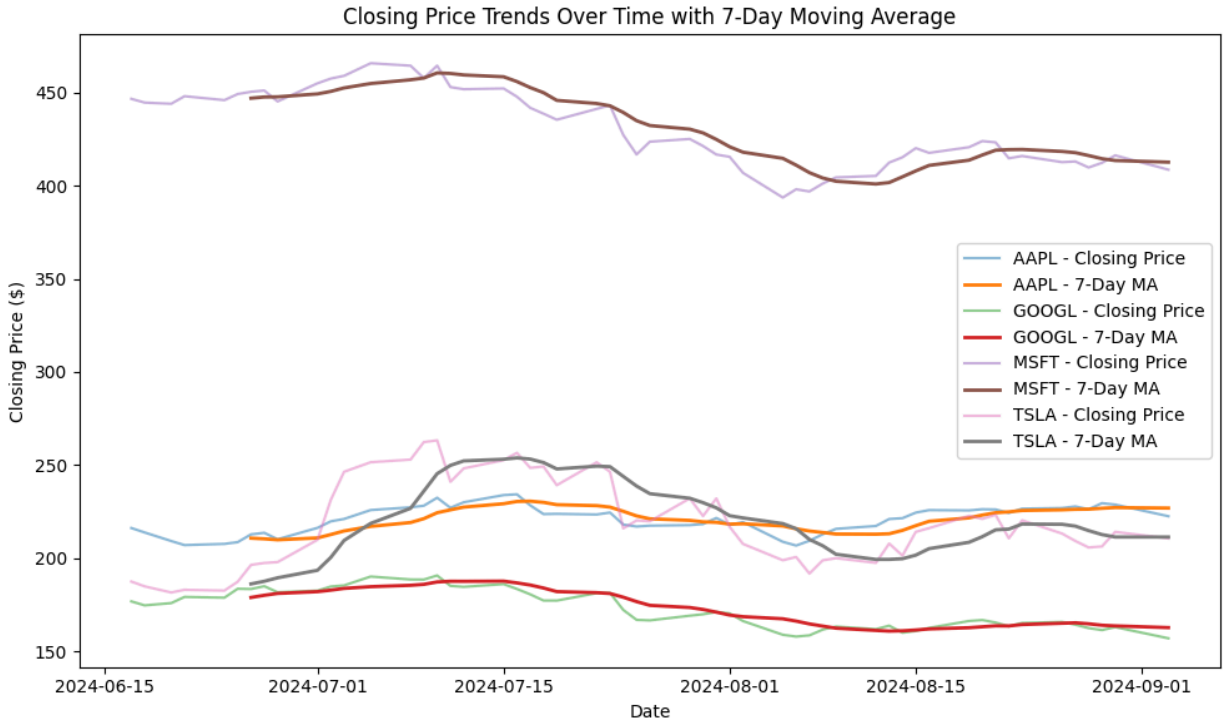
Experiencing and trying to navigate these issues strengthened our understanding of debugging, data-handling, and API integration in real-world projects.

Calculations

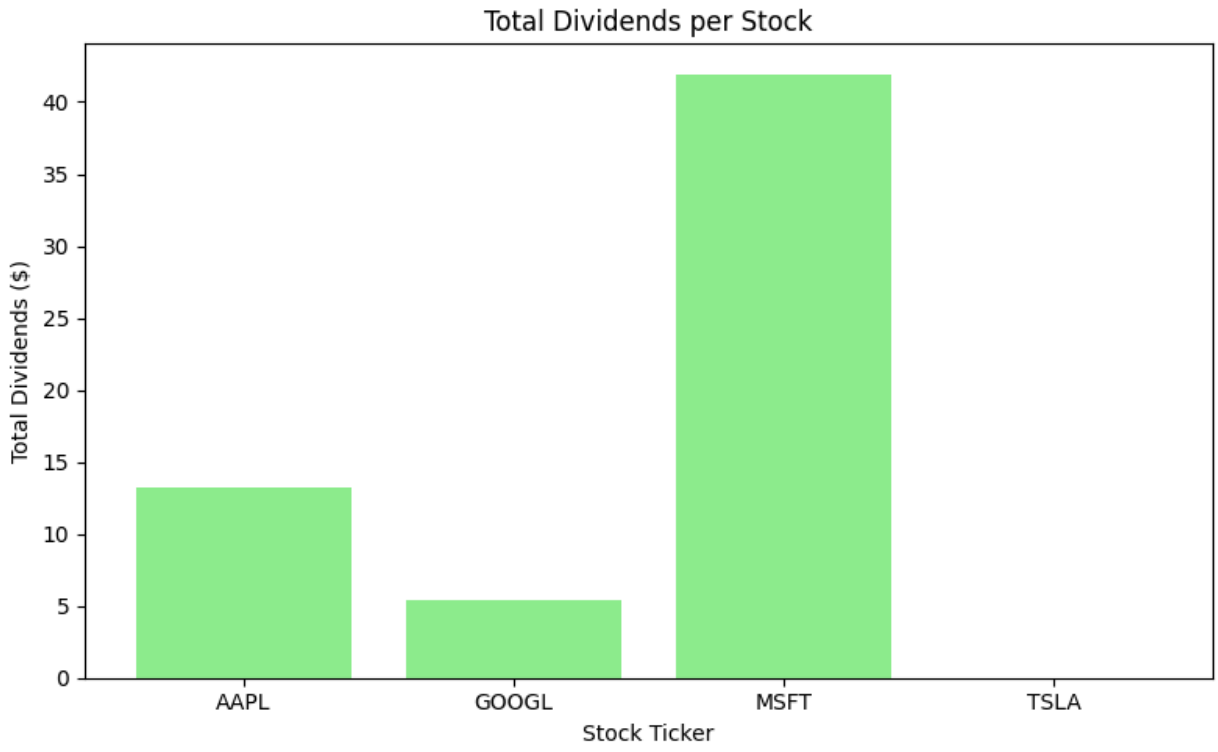
```
================================================
STOCK DATA ANALYSIS REPORT
================================================
Generated on: 2024-12-16 22:14:27



### Total Dividends per Stock ###
 - AAPL: $13.23
 - GOOGL: $5.4
 - MSFT: $41.94
 - TSLA: $0.0

### Total Insider Transactions per Stock ###
 - AAPL: 54 transactions
 - GOOGL: 54 transactions
 - MSFT: 54 transactions
 - TSLA: 48 transactions

### Average Sentiment per Stock ###
 - AAPL: Sentiment Score = 0.31
 - GOOGL: Sentiment Score = 0.4
 - MSFT: Sentiment Score = 0.35
 - TSLA: Sentiment Score = 0.37

### Closing Price Trends Over Time ###
This section shows the minimum and maximum prices, and trends for each stock.

 - AAPL: Min=$206.76292419433594 on 20240806, Max=$234.29074096679688 on 20240716
 - GOOGL: Min=$156.97157287597656 on 20240903, Max=$190.70806884765625 on 20240710
 - MSFT: Min=$393.6510925292969 on 20240805, Max=$465.78643798828125 on 20240705
 - TSLA: Min=$181.57000732421875 on 20240620, Max=$263.260009765625 on 20240710

### Average Closing Prices ###
 - AAPL: Avg=$220.75, Min=$206.76292419433594, Max=$234.29074096679688
 - GOOGL: Avg=$172.76, Min=$156.97157287597656, Max=$190.70806884765625
 - MSFT: Avg=$430.46, Min=$393.6510925292969, Max=$465.78643798828125
 - TSLA: Avg=$218.54, Min=$181.57000732421875, Max=$263.260009765625

================================================
End of Report
================================================
```

Visualizations

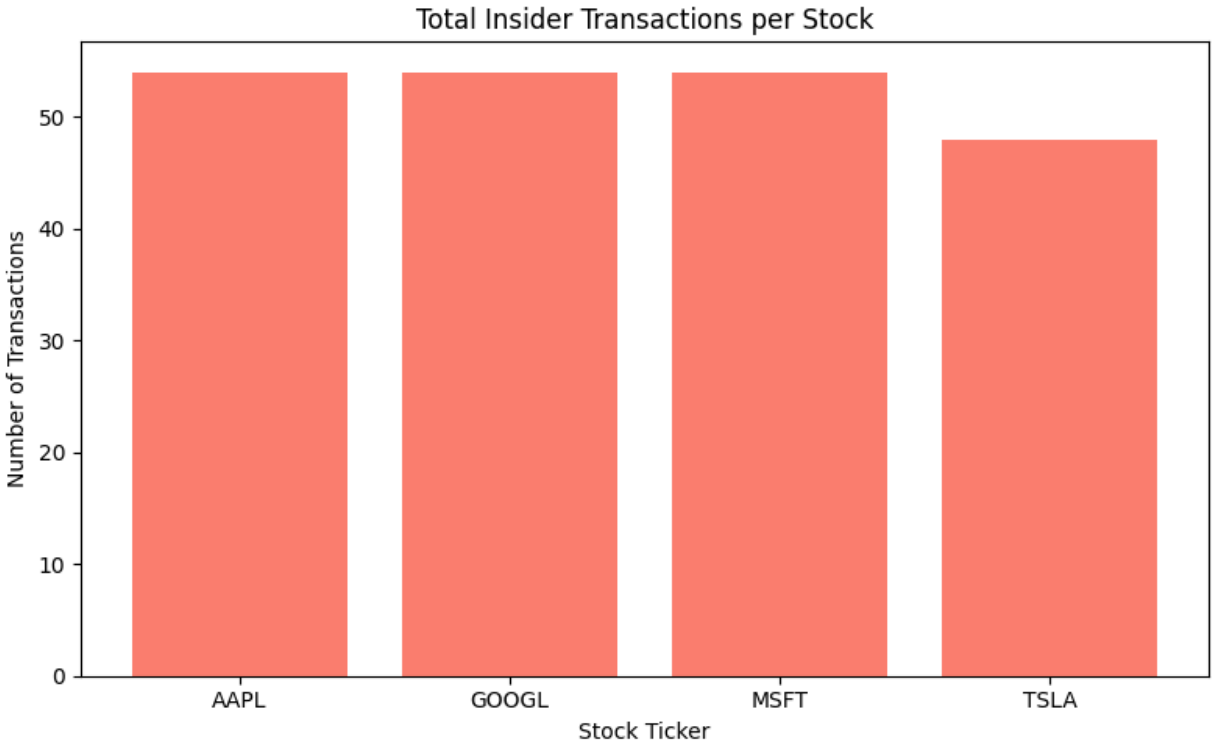
1. Closing price trends over Time, with 7-Day Moving Average: this line graph shows the daily closing prices for the screened stocks over the analysis period, along with their respective 7-day moving averages. The moving average smooths out short-term fluctuations, making it easier to identify trends in stock performance over time. Users can easily identify periods of growth, decline, or stability, and volatility while making investment decisions.



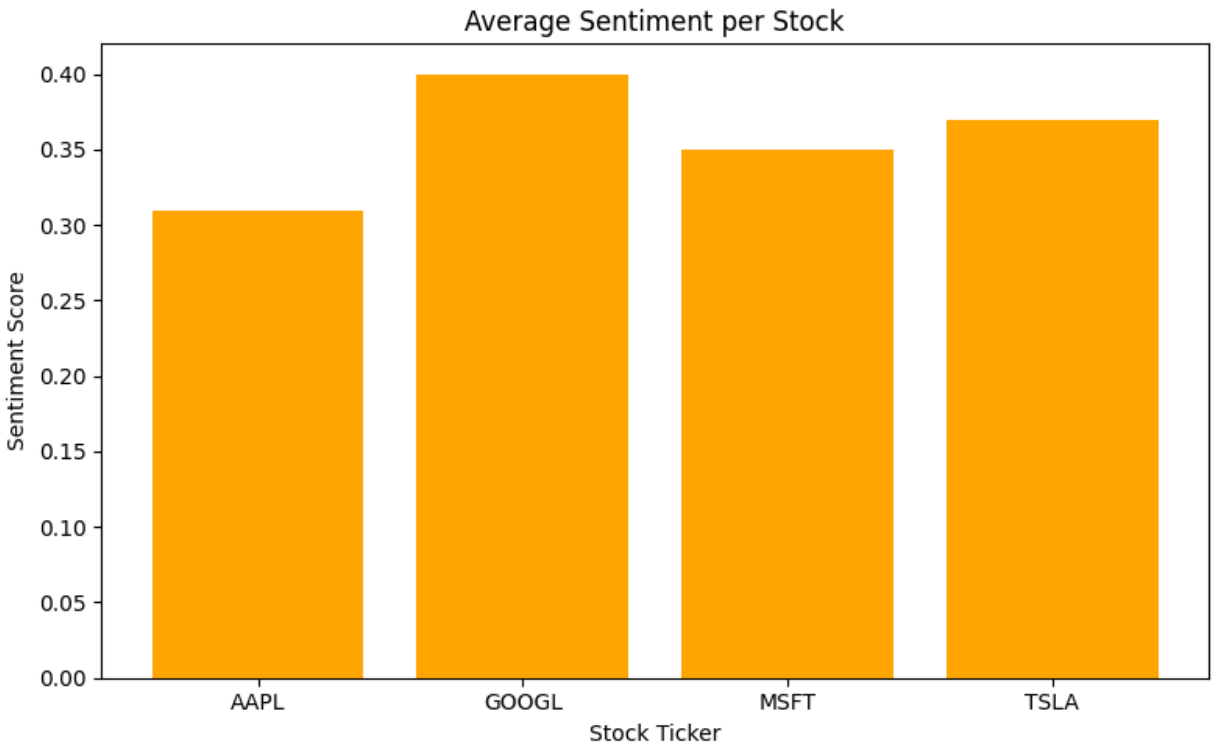
2. Total Dividends per Stock: this bar chart compares the total dividend payouts for each stock during the analysis period. It highlights differences in shareholder returns, with some companies providing regular dividends while others, like Tesla, offering none. The visualization provides insight into which companies provide higher returns to shareholders.



3. Total Insider Transactions per Stock: this bar chart summarizes the total number of insider transactions (e.g., shares bought or sold by company insiders) for each stock. The visualization highlights the level of insider trading activity, which can often serve as a signal of confidence or caution from company stakeholders.



4. Average Sentiment per Stock: this bar chart displays the average sentiment scores for news articles related to each stock. Sentiment scores are positive values ranging from 0.0 (neutral) to values closer to 1.0 (highly positive). The visualization provides insights into the overall tone of financial news coverage for each company.



5. Average Closing Price per Stock: this bar chart displays the average closing prices for the screened stocks over the analyzed period. Each bar represents the average price for a given stock, providing a clear and straightforward comparison of their relative valuations.

Instructions for Running This Project

Step 1: Ensure SQLite3 is installed

For Ubuntu/Debian:

sudo apt update

sudo apt install sqlite3

For macOS (using Homebrew):

brew install sqlite

For Windows:

1. Download the SQLite tools from the SQLite Downloads Page.
2. Extract the files.
3. Add the SQLite executable (`sqlite3.exe`) to your system's PATH.

Step 2: Ensure Python version is correct & relevant Python libraries/modules are installed

A. **Ensure you have the following installed on your computer:**

1. **Python**: Version 3.8 or above

   Verify Python installation:

   python --version

   -

2. **pip**: Python's package manager

   Verify pip installation:

   pip --version

3. **Install Necessary Python Libraries**:

   Open your terminal and navigate to the project folder.

B. Install dependencies:

pip install requests yfinance pandas matplotlib datetime

C. Verify the installation:

pip show requests yfinance pandas matplotlib

Step 3: Setup API Keys

1. **Generate API Keys**: Sign up and generate API keys for the following services as new ones are required each time the project is run due to the API's request limitations:
   - **MarketAux -** https://www.marketaux.com/register
   - **Finnhub** - https://finnhub.io/register
   - **Financial Modeling Prep -** https://site.financialmodelingprep.com/register
2. **Replace Placeholders**:

   Open the file **`data_generator.py`** in a text editor (e.g., VS Code).

   Replace the placeholder API keys with your generated keys:

```
DB_NAME = "stock_data.db"
MARKET_AUX_KEY = "Y7mD3NxUzISxd9IRkvW4anAXAmHuGSMGUhIovM1t"
FINNHUB_KEY = "ctfne0pr01qi0nfdon4gctfne0pr01qi0nfdon50"
FMP_KEY = "6MwrkQYWb9lkTrkJSTm3budgQiAo2Or6"
```

3. **Save the File**.

Step 4: Enter Stocks

- By default, the script fetches data for four stocks: **AAPL, GOOGL, MSFT, and TSLA**.
- If you want to customize this, update the stock tickers in the `main()` function of the **`data_generator.py`** file:

```
def main():
    symbols = ["AAPL", "GOOGL", "MSFT", "TSLA"]

    conn = sqlite3.connect(DB_NAME)

    for symbol in symbols:
        print(f"Fetching data for {symbol}...")

        stock_prices = fetch_stock_prices_yahoo(symbol, conn, max_rows)
        fetch_news_marketaux(symbol, conn, max_rows)
        store_data_in_db(conn, symbol, stock_prices)
        dividends = fetch_dividends_fmp(symbol, conn, max_rows)
        fetch_insider_transactions_finnhub(symbol, conn, max_rows)

        print(f"Data for {symbol} stored successfully.\n")

    conn.close()


if __name__ == "__main__":
    main()
```

Save the File after updating.

Step 5: Run the Project

A. Generate the Database

Run the following command:

```
python3 data_generator.py
```

- **What this does**:
    - Fetches data from APIs and the `yfinance` library.
    - Creates an SQLite database file named **`stock_data.db`**.
    - Populates the database tables with stock prices, dividends, insider transactions, and news sentiment.
- **Expected Output**:
    - A file named `stock_data.db` will be created in your project directory.
    - You can view the database using a database viewer like **DB Browser for SQLite**.
- **Tip**: Run this script **6-7 times** to populate the tables with at least **100 rows** of data.

B. Generate Insights and Visualizations

Run the following command:

python3 visualiser.py

- **What this does**:
    - Reads data from the **`stock_data.db`** database.
    - Generates visualizations (saved as `.png` images in the project folder).
    - Creates a formatted report summarizing the data insights in a new file **`output_log.txt`**.

Generated Outputs

1. **Database**:
    - `stock_data.db` (SQLite database populated with stock-related data).
2. **Visualizations**:
    - PNG files for each graph:
        - average_closing_prices.png
        - total_dividends.png
        - total_insider_transactions.png
        - average_sentiment.png
        - closing_price_trends_with_avg.png
3. **Formatted Report**:
    - output_log.txt summarizing insights, including:
        - Average stock prices

- Total dividends
- Insider transactions
- Sentiment analysis
- Closing price trends with 7-day moving averages

Next Steps

1. Use the generated **database**, **report**, and **visualizations** to analyze stock performance.
2. Identify trends, track insider activities, and assess sentiment to make informed investment decisions.

Run Project Summary

Run Data Generation:

```
python3 data_generator.py
```

Run Visualization and Report Generation:

```
python3 visualiser.py
```

Code Documentation

data_generator.py file

1. setup_database()

- **Purpose**: Initializes the SQLite database and creates required tables if they do not already exist.
- **Input**: None.
- **Output**: None (creates/updates tables in the SQLite database).

Tables Created:

- Stocks: Stores unique stock tickers.
- StockPrices: Stores historical stock price data.
- Dividends: Stores dividend data.
- InsiderTransactions: Tracks insider transactions.
- News: Stores URLs and titles of news articles.
- Sentiments: Stores sentiment scores for news articles related to specific stocks.

2. fetch_stock_prices_yahoo(symbol, conn, max_rows)

- **Purpose**: Fetches stock price data for the last 6 months using Yahoo Finance API and prepares it for insertion into the database.
- **Input**:
    - symbol (str): Stock ticker symbol (e.g., "AAPL").
    - conn (sqlite3.Connection): Database connection object.
    - max_rows (int): Maximum number of rows to fetch and store.

- **Output**:
    - A list of dictionaries, each containing:
        - date: Date as an integer (e.g., 20240430).
        - open: Opening price.
        - close: Closing price.
        - high: High price.
        - low: Low price.
- **Side Effects**: Prints the number of fetched stock price entries.

3. fetch_insider_transactions_finnhub(symbol, conn, max_rows)

- **Purpose**: Fetches insider transaction data from the Finnhub API and inserts it into the database.
- **Input**:
    - symbol (str): Stock ticker symbol.
    - conn (sqlite3.Connection): Database connection object.
    - max_rows (int): Maximum number of rows to fetch.
- **Output**: None.
- **Side Effects**:
    - Inserts insider transaction data into the InsiderTransactions table.
    - Prints the number of records stored or any errors.

Fields Fetched:

- filing_date: Date when the transaction was filed.
- transaction_date: Date when the transaction occurred.

- transaction_price: Transaction price (converted to cents).
- share: Number of shares.
- change: Change in share ownership.

4. fetch_dividends_fmp(symbol, conn, max_rows=10)

- **Purpose**: Fetches historical dividend data using the Financial Modeling Prep API.
- **Input**:
    - symbol (str): Stock ticker symbol.
    - conn (sqlite3.Connection): Database connection object.
    - max_rows (int): Maximum number of records to fetch (default is 10).
- **Output**: None.
- **Side Effects**:
    - Inserts dividend data into the Dividends table.
    - Prints the number of fetched dividend records.

Fields Fetched:

- date: Date of the dividend.
- dividend: Dividend value (converted to cents).

5. fetch_news_marketaux(symbol, conn, max_rows)

- **Purpose**: Fetches news articles and associated sentiment scores for a given stock ticker using the MarketAux API.
- **Input**:
    - symbol (str): Stock ticker symbol.

    - conn (sqlite3.Connection): Database connection object.
    - max_rows (int): Maximum number of articles to fetch.
- **Output**: None.
- **Side Effects**:
    - Inserts news articles into the News table.
    - Inserts corresponding sentiment scores into the Sentiments table.
    - Prints the number of articles fetched and any skipped duplicates.

Fields Fetched:

- title: Title of the news article.
- url: URL of the article.
- entities: Contains sentiment scores for entities in the article.

6. store_data_in_db(conn, symbol, stock_prices)

- **Purpose**: Inserts fetched stock price data into the StockPrices table.
- **Input**:
    - conn (sqlite3.Connection): Database connection object.
    - symbol (str): Stock ticker symbol.
    - stock_prices (list): List of stock price dictionaries returned by fetch_stock_prices_yahoo().
- **Output**: None.
- **Side Effects**:
    - Inserts stock prices into the StockPrices table.
    - Ensures the stock exists in the Stocks table.

7. main()

- **Purpose**: The main driver function that orchestrates the entire workflow:
    - Sets up the database.
    - Fetches stock prices, news articles, dividends, and insider transactions for multiple stock tickers.
    - Inserts fetched data into the database.
- **Input**: None.
- **Output**: None.
- **Side Effects**:
    - Calls all the data-fetching functions for predefined stock tickers (AAPL, GOOGL, MSFT, TSLA). (the user can edit these stocks to their liking)
    - Prints status messages for each step.

8. Script Execution

- **If the script is executed directly** (if __name__ == "__main__":):
    - The main() function is called, which sets up the database and fetches data for specified stock tickers.

Summary of Workflow:

1. The setup_database() function initializes the database.
2. The main() function iterates through stock tickers and:
    - Fetches stock prices (fetch_stock_prices_yahoo).

    - Fetches news and sentiment data (fetch_news_marketaux).
    - Fetches dividend data (fetch_dividends_fmp).
    - Fetches insider transaction data (fetch_insider_transactions_finnhub).
    - Inserts all the fetched data into respective tables.
3. The script prints progress at every step, ensuring transparency during execution.

visualiser.py file

1. fetch_data(query)

- **Purpose**: Executes a SQL query on the stock_data.db SQLite database and returns the results as a Pandas DataFrame.
- **Input**:
    - query (str): SQL query to execute.
- **Output**:
    - data (pd.DataFrame): Result of the query.
- **Side Effects**: Opens and closes a database connection.

2. calculate_summary()

- **Purpose**: Generates a summary report of stock data and writes it to a text file (stock_summary.txt).
- **Input**: None.
- **Output**:
    - Four Pandas DataFrames:

        - avg_close_data: Average, minimum, and maximum closing prices for each stock.
        - total_dividends_data: Total dividends for each stock.
        - transactions_data: Number of insider transactions per stock.
        - avg_sentiment_data: Average sentiment scores for each stock.
- **Side Effects**:
    - Writes a formatted stock data analysis report to stock_summary.txt.
    - Prints confirmation message for report generation.

Key Calculations and Queries:

- **Average Closing Prices**: Average, minimum, and maximum prices for each stock.
- **Total Dividends**: Total dividends distributed by each stock.
- **Total Insider Transactions**: Number of insider transactions for each stock.
- **Average Sentiment**: Average sentiment scores calculated from news articles.
- **Closing Price Trends**: Tracks price trends over time with minimum and maximum values.

3. plot_avg_closing_prices(avg_close_data)

- **Purpose**: Generates a bar chart visualizing the average closing prices for each stock.
- **Input**:
    - avg_close_data (pd.DataFrame): DataFrame containing tickers and their average closing prices.
- **Output**: None.
- **Side Effects**:

    - Saves the chart as average_closing_prices.png.
    - Displays the bar chart.

4. plot_total_dividends()

- **Purpose**: Generates a bar chart showing total dividends for each stock, including those with zero dividends.
- **Input**: None.
- **Output**: None.
- **Side Effects**:
    - Fetches data using a query for total dividends.
    - Saves the chart as total_dividends.png.
    - Displays the bar chart.

5. plot_total_transactions()

- **Purpose**: Generates a bar chart visualizing the number of insider transactions per stock.
- **Input**: None.
- **Output**: None.
- **Side Effects**:
    - Fetches data using a query for insider transactions.
    - Saves the chart as total_insider_transactions.png.
    - Displays the bar chart.

6. plot_avg_sentiment()

- **Purpose**: Generates a bar chart showing the average sentiment score for each stock.

- **Input**: None.
- **Output**: None.
- **Side Effects**:
    - Fetches data using a query for average sentiment scores.
    - Saves the chart as average_sentiment.png.
    - Displays the bar chart.

7. plot_closing_price_trend()

- **Purpose**: Plots closing price trends for all stocks over time, including a 7-day moving average.
- **Input**: None.
- **Output**: None.
- **Side Effects**:
    - Fetches data for closing prices over time.
    - Saves the chart as closing_price_trends_with_avg.png.
    - Displays the time-series line chart.

Visualization:

- Closing prices are plotted as individual time-series lines.
- A 7-day moving average (rolling mean) is overlaid on each line for trend analysis.

8. main()

- **Purpose**: Orchestrates the entire workflow:
    - Calculates the stock summary and generates a report.

    - Generates visualizations for stock data:
        - Average closing prices
        - Total dividends
        - Total insider transactions
        - Average sentiment
        - Closing price trends with moving averages.
- **Input**: None.
- **Output**: None.
- **Side Effects**:
    - Calls all functions for calculations and visualizations.
    - Prints progress messages at each step.

9. Script Execution

- **If the script is executed directly** (if __name__ == "__main__":):
    - The main() function is called.
    - A summary report and all visualizations are generated and saved.

Key Files Generated:

1. **Text File**:
    - stock_summary.txt: Contains the stock data analysis report.
2. **Visualization Files**:
    - average_closing_prices.png: Bar chart of average closing prices.

- total_dividends.png: Bar chart of total dividends.
- total_insider_transactions.png: Bar chart of total insider transactions.
- average_sentiment.png: Bar chart of average sentiment scores.
- closing_price_trends_with_avg.png: Line chart of closing prices with a 7-day moving average.

Workflow Overview:

1. Fetch stock data using SQL queries.
2. Generate summary statistics and store them in a text file.
3. Plot various visualizations:
    - Bar charts for closing prices, dividends, transactions, and sentiment scores.
    - Line charts for closing price trends.
4. Save all outputs to text and image files for easy analysis and presentation.

Resources Used

Date	Issue Description	Location of Resource	Result (did we resolve the issue?)
11/25	Struggling to setup python environments and verify their version compatibility	https://note.nkmk.me/en/python-pip-install-requirements/	Yes, we successfully installed requests, yfinance, pandas, and matplotlib using pip.
11/28	Too many news articles getting returned using MarketAux API	https://www.marketaux.com/documentation	We looked through documentation and found that there is a "limit" parameter, so we could put a limit on number of articles being returned, and were able to resolve this issue.
12/03	Having difficulties with duplicate strings while using insider sentiment data	https://finnhub.io/docs/api	We reviewed documentation and found new data (insider transactions) that better suits project purpose as it can easily remove duplicate strings. By using this new data, we resolved the issue.
12/08	The fetch_news_marketaux function was not handling duplicate news articles effectively	ChatGPT	Helped with debugging the function and refined our logic to avoid duplicates, therefore solving the problem.
12/09	Found inconsistency in returned data types from APIs (e.g., JSON fields missing or unexpected nulls)	https://stackoverflow.com/questions/tagged/python	Implemented .get() method with default values to handle missing fields without crashing the script.
12/10	Unsure how stock price data is formatted	yfinance.Ticker(symbol).history(period="6mo") (via yfinance library)	Understood data formatting for the json

12/12	Did not know how to create the stock price line graph with multiple companies	ChatGPT	We got the guidance we needed on how to start the visualization, and were able to complete it successfully.
12/15	Unsure what is included in dividend data	https://financialmodelingprep.com/api/v3/historical-price-full/stock_dividend/{symbol}	Better grasped different components of dividend data