# Stock Market Decoded

Jayagowtham
ME21B078

May 14, 2025

## Contents

# 1 Introduction

Stock Market Decoded is your personal expert for all financial predictions. Whether you're curious about how high Google might rise in the next five days, how much Microsoft could dip over the next month, or which stock — Tesla or Google — will come out on top in the coming days, we've got you covered.

# 2 Source code

The base of the project is derived from JordiCorbilla's work. It was morphed completely to a new application compatible project by me.

# 3 Version control

An empty git repository was initialized in the local project directory. A corresponding remote github repository stock_market_decoded was set up to track the local repo. A .gitignore file was created to ignore project ideas and other ignorable files. The commits were frequent and enabled me to checkout between various commits when in doubt.


Figure 1: Github files

# 4 DVC integration

The plan was to create a pipeline consisting of - data fetching, preprocessing and training. 3 separate scripts were written for the three functionalities with inter-dependencies. A dvc.yaml file is created for each of the stock in our portfolio, so that we have independent pipelines for separate stocks. The data generated in these scripts is stored in the *data/* folder. With everything setup, the pipeline can be initiated with `dvc repro`. Now the data files are automatically being tracked by dvc and no longer by git.

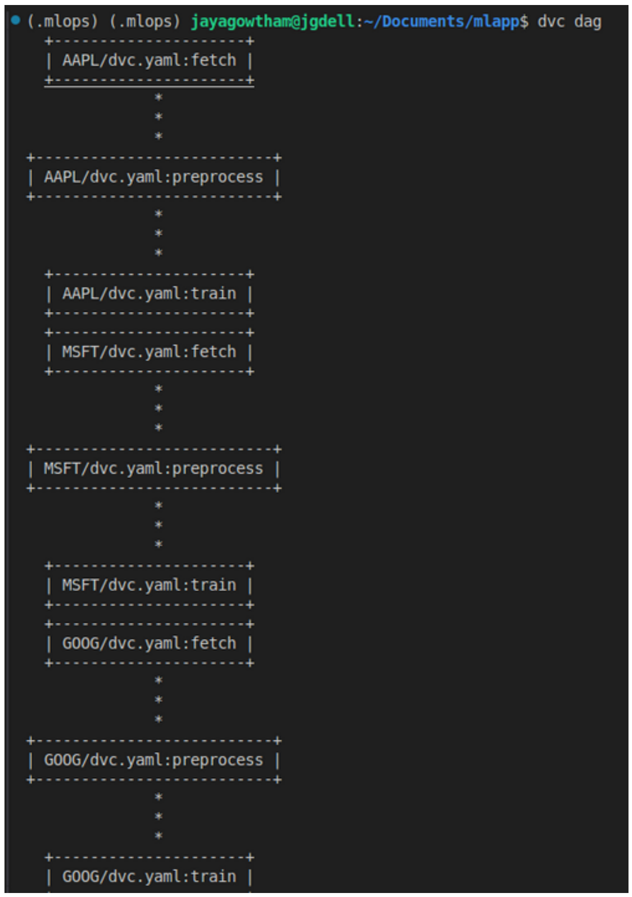


Figure 2: DVC DAG

# 5 MLFlow integration

Next up is the integration of *mlflow* to our project. So, first we set up our tracking server to port 5000 and start the server in the same port. We specify this in our scripts too. Next, we run the pipeline. Throughout the pipeline, we log the parameters, metrics, artifacts (min-max scalers) and the models. For inference, we get the latest version of a specific stock model using MLFlow client and make predictions.

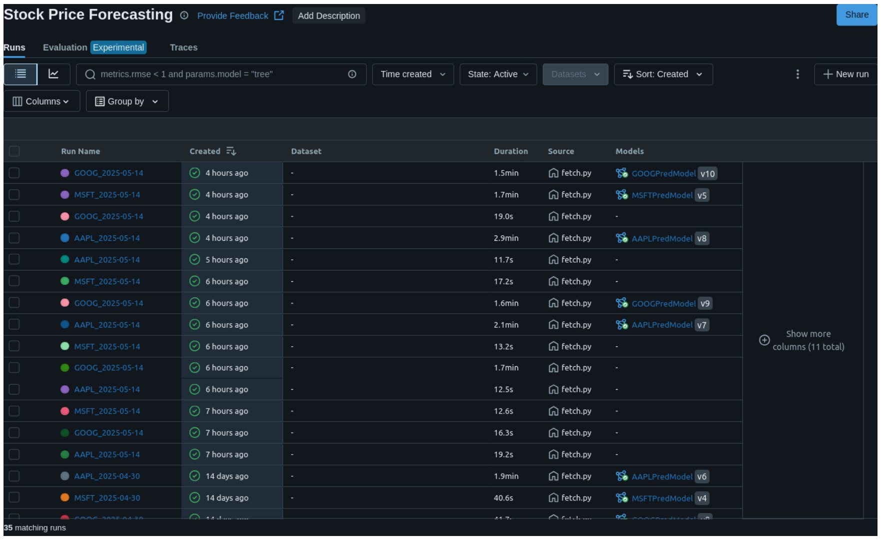


Figure 3: MLFlow experiments

# 6 Model feedback loop

An evaluator script was setup to evaluate the models and compute their rolling average MSE score and store them in text files. A shell script goes over these scores and compares them with a threshold to decide whether to retrain the models. Both the functionalities have separate shell scripts (*evaluator_cron.sh* and *feedback_cron.sh*). They can be setup as separate cron jobs, but care must be taken to separate the 2 jobs by a considerable amount of time owing to rate limits by *yfinance* API.

# 7 Frontend and Backend

The user interface was setup using streamlit. Trained models are hosted on different ports depending on their stock ticker and the frontend "requests" the ports. The frontend was created using Streamlit and is programmed to return a plot upon an API call.

# 8 Prometheus integration

For prometheus monitoring, from the frontend, we post 2 metrics - API call counter based on ip and API runtime sum overall to check health. We publish these metrics at port 18000 and enable Prometheus to fetch it. We also run node exporter to monitor the health of the system. The prometheus monitoring is facilitated by a *prometheus.yaml* file.

# 9 Grafana integration

The prometheus URL was used as the data source for Grafana visualization. The dashboard's JSON file is attached in the repository.

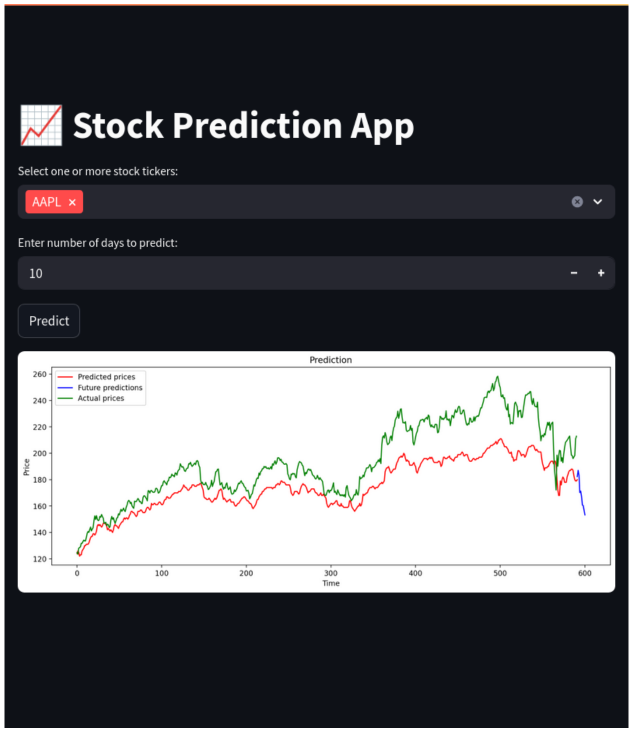


Figure 4: Frontend

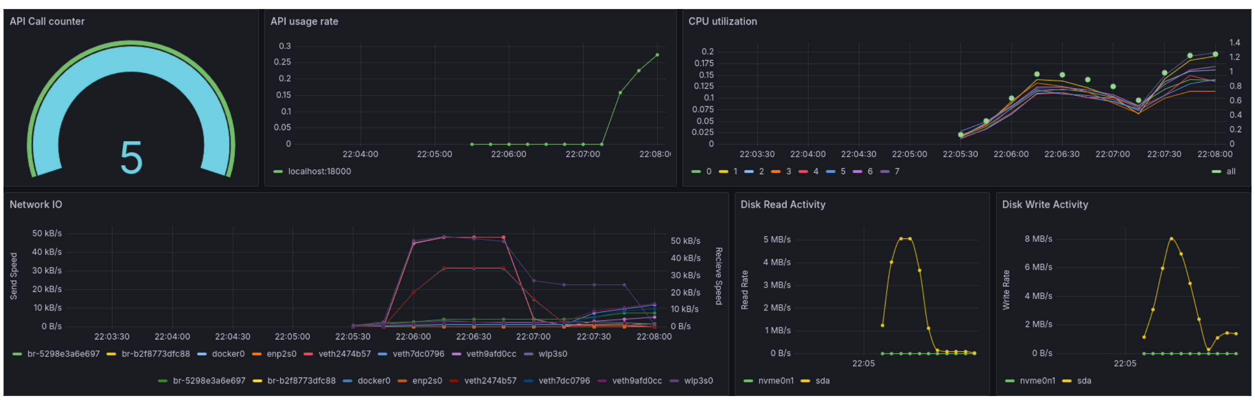


Figure 5: Grafana Dashboard.

# 10 Dockerization

Finally, given the discretization of the application, the frontend and backend were composed as separate services, with MLFlow server as the third service in our Docker Compose file. We start frontend only when backend is healthy. For health check, we ping the ports with dummy inputs using curl till we get an output.

# 11 Model details and architecture

For our stock price prediction, we use an LSTM Model with early stopping. Batch size, epochs, patience and time steps are our hyper-parameters. At present, we do not tune them but it's easier to extend. We also train models only for 3 stocks - GOOG, AAPL, MSFT. It can also be easily extended. The model architecture is shown below.

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| lstm (LSTM) | (None, 3, 100) | 40,800 |
| dropout (Dropout) | (None, 3, 100) | 0 |
| lstm_1 (LSTM) | (None, 3, 50) | 30,200 |
| dropout_1 (Dropout) | (None, 3, 50) | 0 |
| lstm_2 (LSTM) | (None, 3, 50) | 20,200 |
| dropout_2 (Dropout) | (None, 3, 50) | 0 |
| lstm_3 (LSTM) | (None, 50) | 20,200 |
| dropout_3 (Dropout) | (None, 50) | 0 |
| dense (Dense) | (None, 1) | 51 |

**Total params:** 111,451 (435.36 KB)
**Trainable params:** 111,451 (435.36 KB)
**Non-trainable params:** 0 (0.00 B)

Figure 6: Model architecture.

# 12 Port details

| Port | Designation |
|---|---|
| 5000 | MLFlow server |
| 8000 | AAPL best model |
| 8001 | GOOG best model |
| 8002 | MSFT best model |
| 18000 | Prometheus client publisher |
| 9000 | Prometheus server |
| 9090 | Node exporter |
| 5000 | Grafana |
| 8501 | Streamlit UI |

Table 1: Port details

# 13 Remarks

Good software engineering principles were followed. Codebase was regularly committed. Log files are setup for pipeline and the application. Try-except blocks are present almost everywhere to catch exceptions.