

# Stock Market Decoded

Jayagowtham  
ME21B078

May 14, 2025

## Contents

1	Introduction	2
2	Source code	2
3	Version control	2
4	DVC integration	3
5	MLFlow integration	3
6	Model feedback loop	4
7	Frontend and Backend	4
8	Prometheus integration	4
9	Grafana integration	4
10	Dockerization	5
11	Model details and architecture	6
12	Port details	6
13	Remarks	6

# 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.

JG-0212

Removing hanging files of AAPL, MSFT

906a3ba · 2 weeks ago

14 Commits

<div><div></div><div>.dvc</div></div>	Stopped data tracking	2 weeks ago
<div><div></div><div>AAPL</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>GOOG</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>MSFT</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>data</div></div>	Removing hanging files of AAPL, MSFT	2 weeks ago
<div><div></div><div>evaluation</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>mlartifacts/393362287489782283</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>mlruns</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>scripts</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>.dvcignore</div></div>	Stopped data tracking	2 weeks ago
<div><div></div><div>.gitignore</div></div>	Serving automated, Feedback loop executed	2 weeks ago
<div><div></div><div>LICENSE.md</div></div>	Refactored the code to suit pipeline needs	3 weeks ago
<div><div></div><div>README.md</div></div>	Initial commit	3 weeks ago
<div><div></div><div>Targets</div></div>	First working version with mlflow integrated	3 weeks ago
<div><div></div><div>backend_startup.sh</div></div>	Serving automated, Feedback loop executed	2 weeks ago
<div><div></div><div>docker-compose.yml</div></div>	Serving automated, Feedback loop executed	2 weeks ago
<div><div></div><div>evaluator.py</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>evaluator_logs.log</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>feedback.sh</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>frontend.py</div></div>	Logging, try except blocks added	2 weeks ago
<div><div></div><div>frontend_requirements.txt</div></div>	Serving automated, Feedback loop executed	2 weeks ago
<div><div></div><div>frontend_utils.py</div></div>	Serving automated, Feedback loop executed	2 weeks ago
<div><div></div><div>params.yaml</div></div>	Pipeline setup, Works smooth	2 weeks ago
<div><div></div><div>prometheus.yml</div></div>	Monitoring added in frontend	2 weeks ago

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.

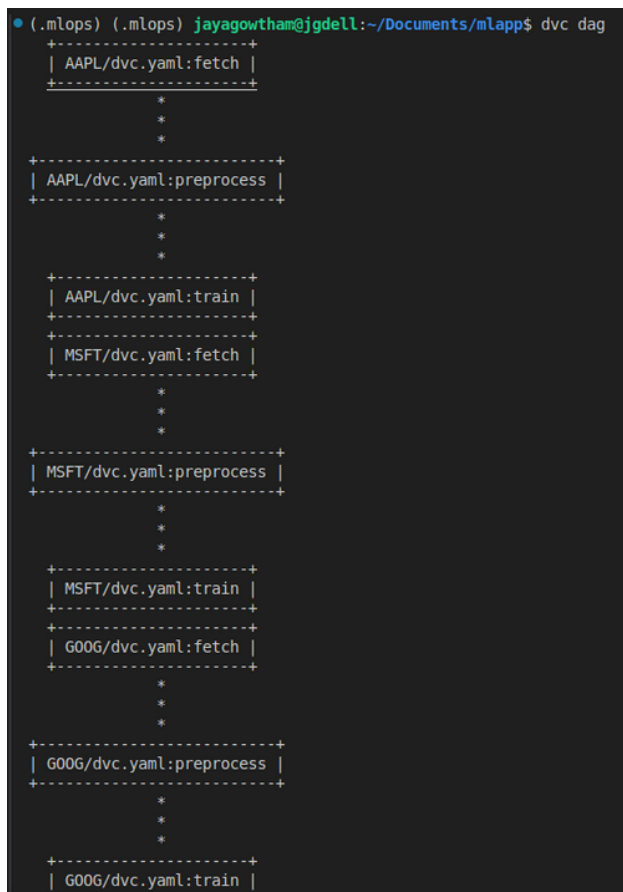


Figure 2: DVC DAG

## 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.

The screenshot shows the MLFlow 'Stock Price Forecasting' interface. At the top, there are tabs for 'Runs', 'Evaluation', 'Experiments', and 'Traces'. Below these is a search bar with the query 'metrics.rmse < 1 and params.model = "tree"'. The main table lists various runs with columns for Run Name, Created, Dataset, Duration, Source, and Models. The runs are for different stock tickers (GOOG, MSFT, AAPL) and dates (2023-05-14 and 2023-04-30). The 'Created' column shows the time since the run was created, ranging from 4 hours ago to 14 days ago. The 'Duration' column shows the time taken for each run, ranging from 1.5min to 40.6s. The 'Source' column shows the source as 'fetch.py'. The 'Models' column shows the model name and version, such as 'GOOGPredModel v10'. A 'Show more columns (11 total)' button is visible on the right side of the table.

Run Name	Created	Dataset	Duration	Source	Models
GOOG_2023-05-14	4 hours ago	-	1.5min	fetch.py	GOOGPredModel v10
MSFT_2023-05-14	4 hours ago	-	1.7min	fetch.py	MSFTPredModel v5
GOOG_2023-05-14	4 hours ago	-	19.0s	fetch.py	-
AAPL_2023-05-14	4 hours ago	-	2.9min	fetch.py	AAPLPredModel v8
AAPL_2023-05-14	5 hours ago	-	11.7s	fetch.py	-
MSFT_2023-05-14	6 hours ago	-	17.2s	fetch.py	-
GOOG_2023-05-14	6 hours ago	-	1.6min	fetch.py	GOOGPredModel v9
AAPL_2023-05-14	6 hours ago	-	2.1min	fetch.py	AAPLPredModel v7
MSFT_2023-05-14	6 hours ago	-	13.2s	fetch.py	-
GOOG_2023-05-14	6 hours ago	-	1.7min	fetch.py	-
AAPL_2023-05-14	6 hours ago	-	12.5s	fetch.py	-
MSFT_2023-05-14	7 hours ago	-	12.6s	fetch.py	-
GOOG_2023-05-14	7 hours ago	-	16.3s	fetch.py	-
AAPL_2023-05-14	7 hours ago	-	19.2s	fetch.py	-
AAPL_2023-04-30	14 days ago	-	1.9min	fetch.py	AAPLPredModel v6
MSFT_2023-04-30	14 days ago	-	40.6s	fetch.py	MSFTPredModel v6

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.

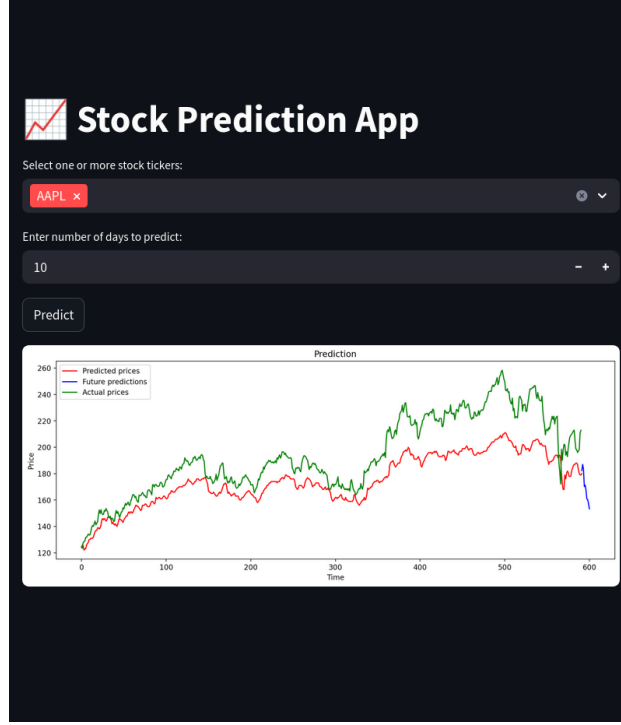


Figure 4: Frontend

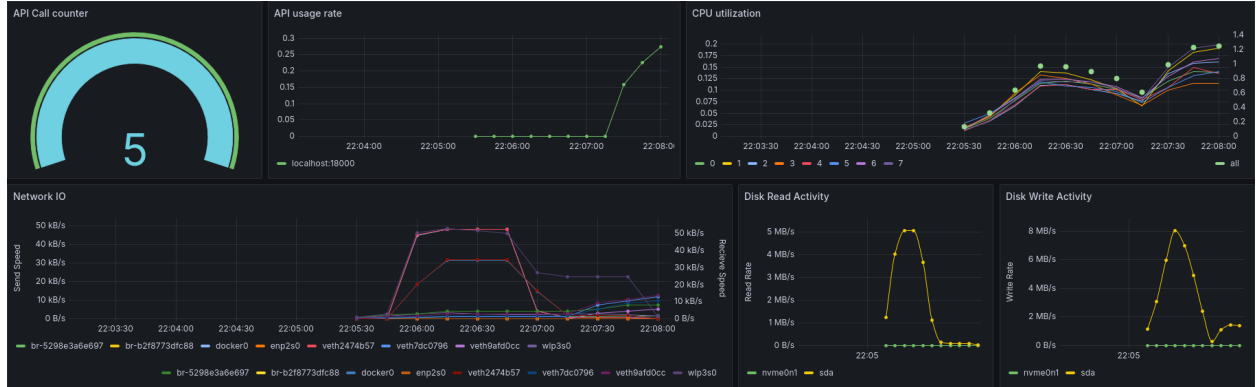


Figure 5: Grafana Dashboard.

## 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.