Student Name : Matheesha Ileperuma  
Student ID : 104659053

# A screenshot of a computer AI-generated content may be incorrect.Setting up and running v0.1

To test the v0.1 code, I set up a virtual environment and installed all the required libraries like TensorFlow, Pandas, and Matplotlib. After running the script, everything worked smoothly — no errors during training, and the output was exactly what I expected.

The script:

* Loaded historical stock data for CBA.AX,
* Scaled and shaped the data using MinMaxScaler,
* Trained an LSTM model using 60-day windows,
* Then plotted actual vs predicted prices on a graph.

I could tell the model was working because it successfully trained and generated a clear line graph comparing actual and predicted prices. There were no crashes or warnings, and visually, the predictions followed the trend of the test data decently. It also printed RMSE and R² values, which gave an idea of how accurate the model was.

From a coding perspective, this v0.1 script felt like a solid introduction to time-series forecasting using LSTMs. It was simple and well-commented, which helped me understand how the logic flowed from data loading all the way to prediction. That said, it was a bit outdated in structure — everything was crammed into one script, and there was no modularity or support for model saving/loading. So it worked fine for testing and learning, but it wouldn’t scale well without a lot of cleanup and refactoring.

A screenshot of a computer

AI-generated content may be incorrect.Testing process

# Setting up and running P1

## A screenshot of a computer program AI-generated content may be incorrect.Training the model

training logs over 500 epochs. The logs include both loss and val\_loss values, showing how the model gradually improved. Even though val\_loss stopped improving around epoch 499, the final values still indicate a highly stable and accurate model

## Testing the model

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

This output shows the evaluation results after loading a pre-trained LSTM model. The model successfully predicted the stock price for AAPL (Apple Inc.) and estimated the future price after 15 days to be 223.42 USD. The Mean Absolute Error was extremely low (1.34e-5), which suggests the model performed well.

# Understanding of P1 code

A screenshot of a computer

AI-generated content may be incorrect.

**Stock\_Prediction.py** - This block is responsible for converting time-series data into overlapping sequences of fixed length .A deque (double-ended queue) is used here to efficiently manage a sliding window of data, which allows each new day’s entry to push the oldest one out as the sequence moves forward. This technique is at the heart of how LSTM models are able to “learn from the past.” Since LSTMs are designed to capture patterns over time, feeding them raw data wouldn’t be enough. Instead, these sliding sequences provide temporal context — a look at how values evolve — which allows the model to form meaningful predictions based on historical behaviour.

A screen shot of a computer program

AI-generated content may be incorrect.

**Stock\_Prediction.py** - This block handles data scaling for each selected feature column using MinMaxScaler. It ensures each feature is reshaped to the correct dimensions before scaling and stores the scaler for future use. This is crucial for ensuring consistent scaling across training, testing, and inference stages. Scaling improves model performance and training stability, especially for neural networks like LSTMs. Without it, features with larger numerical ranges (like “volume”) could dominate learning. Also, storing the scaler allows us to reverse the transformation on predictions, which is key to interpreting results in real-world stock prices.

A computer screen shot of text

AI-generated content may be incorrect.

**Test.py** - This section introduces custom lambda functions to simulate buy and sell profits based on the model’s predictions. It compares the model’s suggested future price to both the current and actual future prices, estimating hypothetical financial gains. Accuracy metrics alone don’t always reflect how useful a model is in practice. This block bridges that gap. It interprets predictions through the lens of potential profit — grounding the project in real-world value. It’s a shift from “is this model accurate?” to “would this model help someone make money?”

A computer screen shot of text

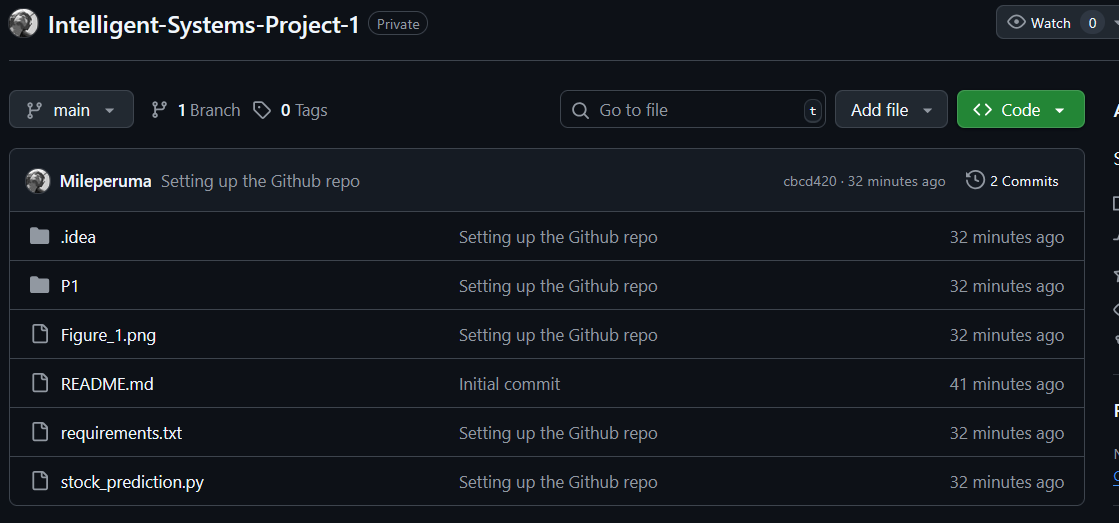
AI-generated content may be incorrect.

**Test.py** - This part of the code checks whether a cached CSV version of the dataset already exists locally. If found, it uses that instead of fetching new data online — reducing network dependency and potential failures. This is a subtle but powerful design choice. Data fetching can break due to internet issues, rate limits, or API changes. By relying on cached data when available, the system becomes more reliable and repeatable. It also saves time, especially during debugging or multiple test runs.

# Setting up the GitHub Repo

A screenshot of a computer program

AI-generated content may be incorrect.



# Comparison

**Model v.01**

This model was designed to predict daily stock prices over a test period. It gives predictions for each day and also shows a graph where you can see how close the predictions are to the real prices. That visual output really helped me understand if the model was learning properly.

Some basic stats were also shown, like how much the predictions differ from actual prices. It’s good for checking how accurate the model is in general.

**Pros**:

* The line graph shows clearly where the prediction and real prices match or differ.
* It gives a solid understanding of the model’s general prediction accuracy.

**Cons**:

* It doesn’t really tell you what to do with those predictions — like when to buy or sell.
* I tried to calculate the trend direction accuracy (whether the model got the ups and downs right), but it gave an error.

**Model P1**

* Stock: AAPL (Apple Inc.)
* Model Type: LSTM (custom configuration with parameters: seq-50, step-15, layers-2, units-256)
* Evaluation Metrics:
* Mean Absolute Error (MAE): 1.34
* Directional Accuracy: 52.38%
* Trading Simulation Metrics:
  + Total Profit: $1174.64
  + Buy Profit: $1018.01
  + Sell Profit: $156.63
  + Profit per Trade: $0.52
* Prediction Output: Forecast of the stock price 15 days into the future

This version was different. It didn’t try to predict every day’s price. Instead, it was trained to predict what the stock price might be 15 days into the future. It also includes some kind of simple trading simulation, where it calculates things like how much profit you could make if you bought or sold based on the model’s forecast.

I didn’t get a graph from this one, but it did give practical info like:

* Future price prediction
* Directional accuracy (whether it got the trend right or not)
* And even estimated profit

**pros**:

* More practical if you're thinking of using it for real-world trading or investing.
* The future forecast helps with decision-making.
* Shows if the trend (going up or down) is correct.
* Gives a rough idea of how much profit a person could have made.

**Not-so-good things**:

* No visual output, so I couldn’t see the predictions versus actual prices easily.
* It didn’t provide the same academic-style accuracy scores as v0.1.