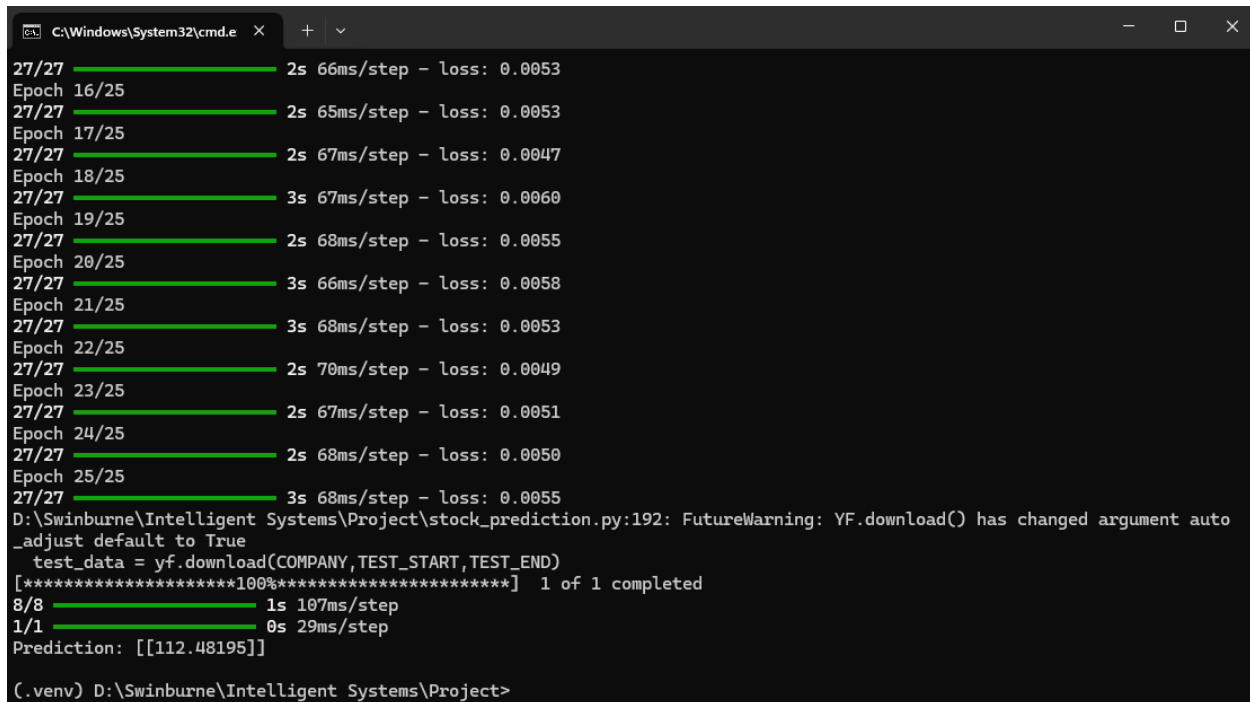


Student Name : Matheesha Ileperuma
Student ID : 104659053

Setting up and running v0.1



```
C:\Windows\System32\cmd.e  X  +  v
27/27 2s 66ms/step - loss: 0.0053
Epoch 16/25
27/27 2s 65ms/step - loss: 0.0053
Epoch 17/25
27/27 2s 67ms/step - loss: 0.0047
Epoch 18/25
27/27 3s 67ms/step - loss: 0.0060
Epoch 19/25
27/27 2s 68ms/step - loss: 0.0055
Epoch 20/25
27/27 3s 66ms/step - loss: 0.0058
Epoch 21/25
27/27 3s 68ms/step - loss: 0.0053
Epoch 22/25
27/27 2s 70ms/step - loss: 0.0049
Epoch 23/25
27/27 2s 67ms/step - loss: 0.0051
Epoch 24/25
27/27 2s 68ms/step - loss: 0.0050
Epoch 25/25
27/27 3s 68ms/step - loss: 0.0055
D:\Swinburne\Intelligent Systems\Project\stock_prediction.py:192: FutureWarning: YF.download() has changed argument auto_
adjust default to True
test_data = yf.download(COMPANY,TEST_START,TEST_END)
[*****100%*****] 1 of 1 completed
8/8 1s 107ms/step
1/1 0s 29ms/step
Prediction: [[112.48195]]
(.venv) D:\Swinburne\Intelligent Systems\Project>
```

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.

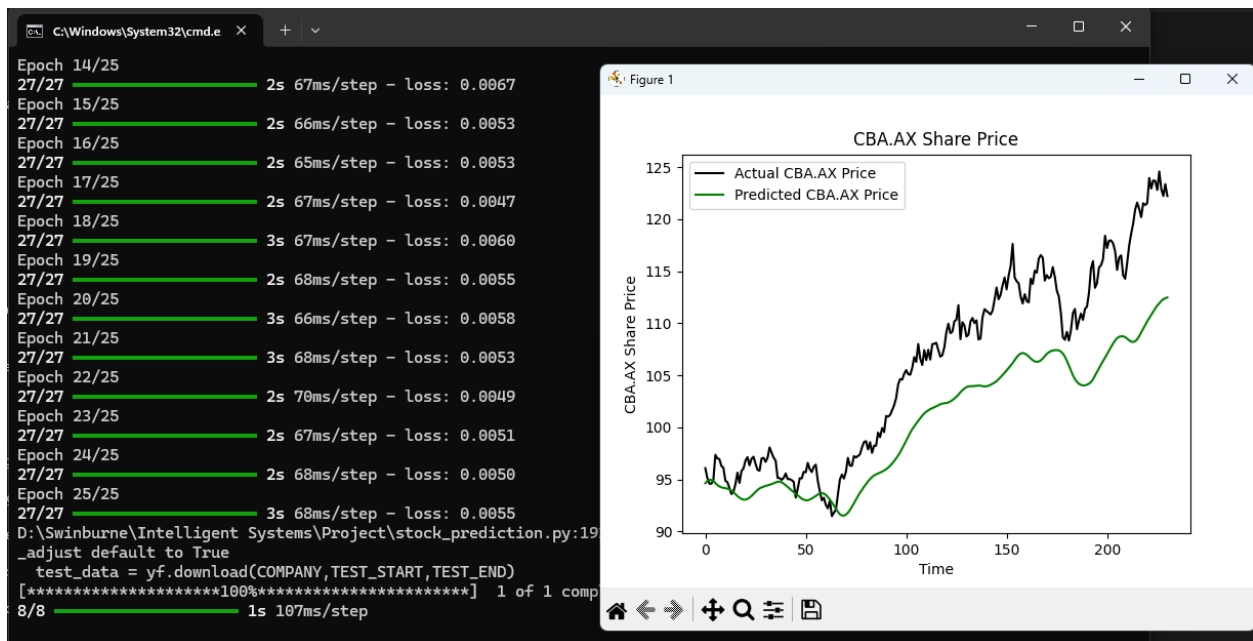
The script:

- 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 R^2 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.

Testing process



Setting up and running P1

Training the model

```
C:\Windows\System32\cmd.e  X  +  v
ber-adam-LSTM-seq-50-step-15-layers-2-units-256.weights.h5
15/15 7s 445ms/step - loss: 2.8089e-04 - mean_absolute_error: 0.0177 - val_loss: 2.0212e-04
- val_mean_absolute_error: 0.0158
Epoch 498/500
15/15 0s 395ms/step - loss: 2.7913e-04 - mean_absolute_error: 0.0180
Epoch 498: val_loss did not improve from 0.00020
15/15 10s 442ms/step - loss: 3.0126e-04 - mean_absolute_error: 0.0185 - val_loss: 2.4363e-04
- val_mean_absolute_error: 0.0168
Epoch 499/500
15/15 0s 389ms/step - loss: 3.0023e-04 - mean_absolute_error: 0.0179
Epoch 499: val_loss did not improve from 0.00020
15/15 7s 437ms/step - loss: 3.1498e-04 - mean_absolute_error: 0.0184 - val_loss: 2.5816e-04
- val_mean_absolute_error: 0.0179
Epoch 500/500
15/15 0s 387ms/step - loss: 2.8417e-04 - mean_absolute_error: 0.0177
Epoch 500: val_loss did not improve from 0.00020
15/15 7s 435ms/step - loss: 2.8086e-04 - mean_absolute_error: 0.0178 - val_loss: 2.4948e-04
- val_mean_absolute_error: 0.0173

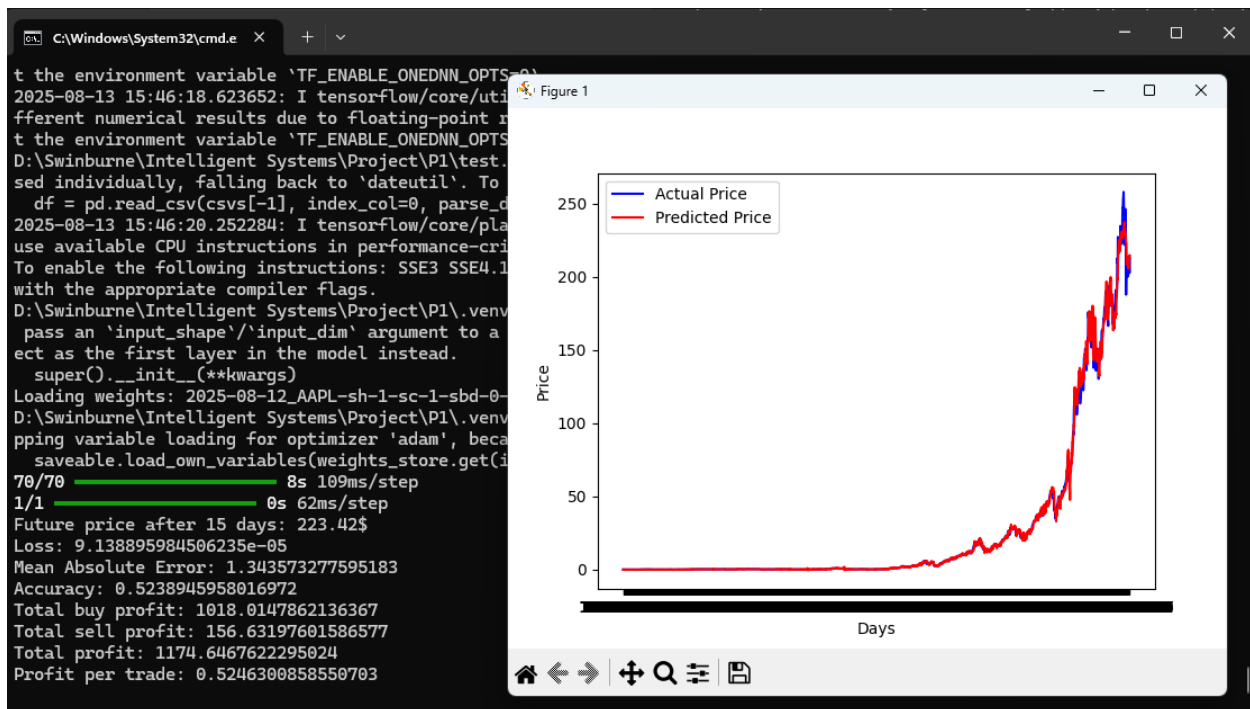
(.venv) D:\Swinburne\Intelligent Systems\Project\P1>py prediction.py
C:\Users\milep\AppData\Local\Programs\Python\Python312\python.exe: can't open file 'D:\Swinburne\Intelligent Systems\Project\P1\prediction.py': [Errno 2] No such file or directory

(.venv) D:\Swinburne\Intelligent Systems\Project\P1>py stock_prediction.py
2025-08-12 13:11:54.371341: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-08-12 13:12:00.769053: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
```

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

```
Loading weights: 2025-08-12_AAPL-sh-1-sc-1-sbd-0-huber-adam-LSTM-seq-50-step-15-layers-2-units-256.weights.h5
D:\Swinburne\Intelligent Systems\Project\P1\venv\Lib\site-packages\keras\src\saving\saving_lib.py:797: UserWarning: Skipping variable loading for optimizer 'adam', because it has 2 variables whereas the saved optimizer has 18 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
70/70 8s 106ms/step
1/1 0s 68ms/step
Future price after 15 days: 223.42$
Loss: 9.138895984506235e-05
Mean Absolute Error: 1.343573277595183
Accuracy: 0.5238945958016972
Total buy profit: 1018.0147862136367
Total sell profit: 156.63197601586577
Total profit: 1174.6467622295024
Profit per trade: 0.5246300858550703
Saved: csv-results\2025-08-12_AAPL-sh-1-sc-1-sbd-0-huber-adam-LSTM-seq-50-step-15-layers-2-units-256.csv
```



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 (1.34×10^{-5}), which suggests the model performed well.

Understanding of P1 code

```
stock_prediction.py  train.py  test.py  P1\stock_prediction.py  parameters.py

143  def load_data(
144      future_sequence = np.array(df[feature_columns].iloc[lookback_step:]),
145
146      # Drop rows with NaNs from the shift
147      df.dropna(inplace=True)
148
149      # Build sequences
150      sequence_data = []
151      sequences = deque(maxlen=n_steps)
152      for entry, target in zip(df[feature_columns + ["date"]].values, df["future"].values):
153          sequences.append(entry)
154          if len(sequences) == n_steps:
155              sequence_data.append((np.array(sequences), target))
```

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.

```
# --- Scaling (safe shapes) ---
column_scaler = {}
if scale:
    for c in feature_columns:
        scaler = preprocessing.MinMaxScaler()
        col = df[c].to_numpy()
        # Ensure exactly 2D (n,1)
        if col.ndim == 1:
            col = col.reshape(-1, 1)
        else:
            col = np.asarray(col).reshape(len(col), -1)[: , :1]
        df[c] = scaler.fit_transform(col).ravel() # back to 1D
        column_scaler[c] = scaler
    result["column_scaler"] = column_scaler
```

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.

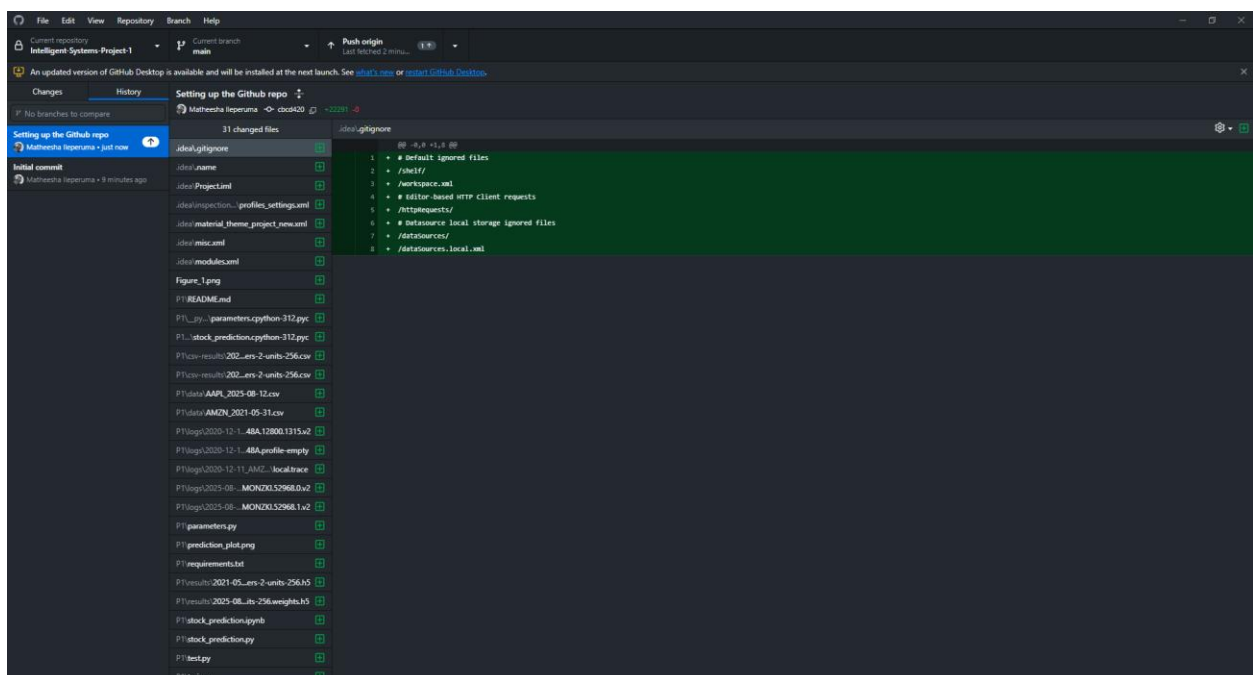
```
def get_final_df(model, data):
    buy_profit = lambda c, pf, tf: (tf - c) if pf > c else 0
    sell_profit = lambda c, pf, tf: (c - tf) if pf < c else 0
    X_test, y_test = data["X_test"], data["y_test"]
    y_pred = model.predict(X_test)
    if SCALE:
        y_test = np.squeeze(data["column_scaler"]["adjclose"].inverse_transform(np.expand_dims(y_test, axis=0)))
        y_pred = np.squeeze(data["column_scaler"]["adjclose"].inverse_transform(y_pred))
    test_df = data["test_df"].copy()
    test_df[f"adjclose_{LOOKUP_STEP}"] = y_pred
    test_df[f"true_adjclose_{LOOKUP_STEP}"] = y_test
    test_df.sort_index(inplace=True)
    test_df["buy_profit"] = list(map(buy_profit, test_df["adjclose"], test_df[f"adjclose_{LOOKUP_STEP}"], test_df[f"true_adjclose_{LOOKUP_STEP}"]))
    test_df["sell_profit"] = list(map(sell_profit, test_df["adjclose"], test_df[f"adjclose_{LOOKUP_STEP}"], test_df[f"true_adjclose_{LOOKUP_STEP}"]))
    return test_df
```

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?”

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



main 1 Branch 0 Tags

Go to file

Add file

Code

Mileperuma Setting up the Github repo

cbcd420 · 32 minutes ago 2 Commits

.idea	Setting up the Github repo	32 minutes ago
P1	Setting up the Github repo	32 minutes ago
Figure_1.png	Setting up the Github repo	32 minutes ago
README.md	Initial commit	41 minutes ago
requirements.txt	Setting up the Github repo	32 minutes ago
stock_prediction.py	Setting up the Github repo	32 minutes ago

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.

