Task C.1 – Setup

COS30018 – Intelligent Systems

Rahul Raju

Student ID: 105143065

Environment Setup and Requirements

I set up a dedicated virtual environment to keep things organized and ensure the dependencies for the stock prediction task were neatly contained. Using the command ***python3 -m venv venv*** in the terminal, I created a fresh environment, activated it, and installed the necessary Python libraries with ***pip install***. Since the provided P1 codebase shared many dependencies with the earlier v0.1 version, I reused the same virtual environment to maintain consistency and avoid duplicating effort. This approach streamlined the setup process and ensured compatibility across both implementations.

Dependencies for v0.1

- ***numpy:*** For numerical computations and array handling.
- ***matplotlib:*** For generating plots to visualize actual vs. predicted stock prices.
- ***pandas***: For efficient data manipulation and DataFrame operations.
- ***pandas-datareader:*** To support data retrieval (though not used in the final setup).
- ***scikit-learn:*** For data preprocessing, including scaling and train-test splitting.
- ***tensorflow:*** For building and training the LSTM model.
- ***yfinance:*** For fetching historical stock data from Yahoo Finance.

Dependencies for P1

The P1 implementation required the same libraries as v0.1, with one key difference:

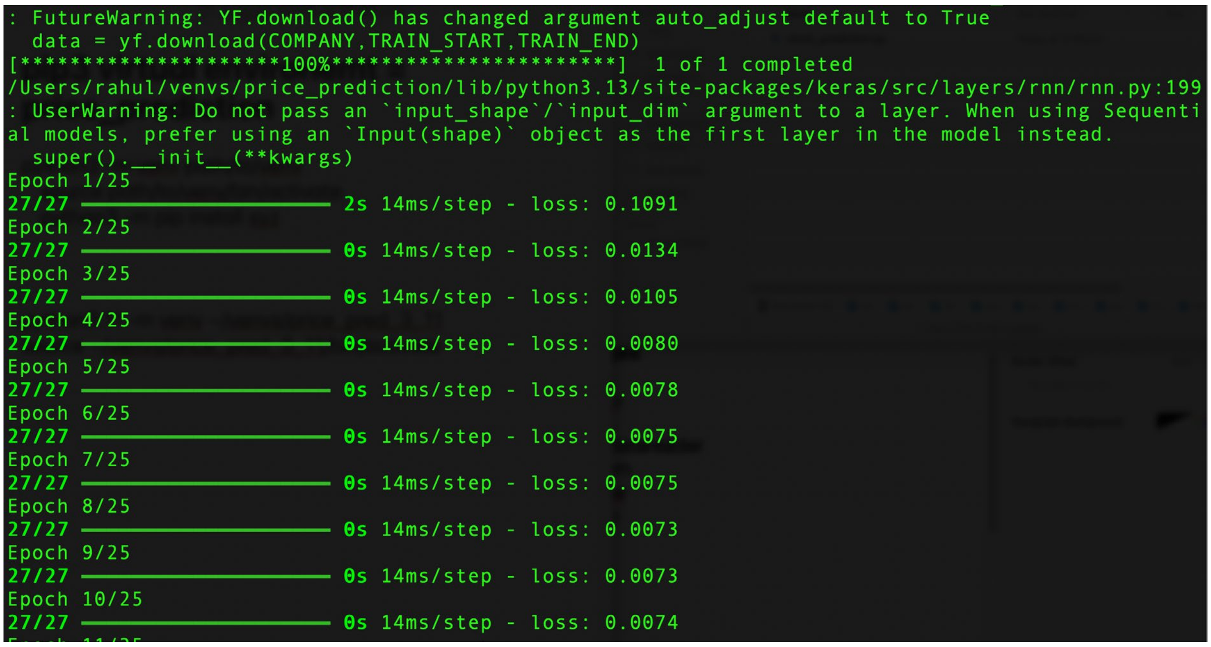
- ***numpy***
- ***matplotlib***
- ***pandas***
- ***pandas-datareader***
- ***scikit-learn***
- ***tensorflow***
- ***yfinance*** (replacing ***yahoo_fin*** from the original P1 code).

Initially, the P1 code relied on ***yahoo_fin*** for data retrieval, but I quickly found it outdated and unreliable. Switching to ***yfinance*** was a critical decision to ensure robust data fetching, aligning with the v0.1 setup and simplifying dependency management.

Testing the Provided Code Base

Testing v0.1 Code

The v0.1 code was straightforward to test once the dependencies were in place. I activated the virtual environment and ran python stock_prediction.py in the terminal. The script used yfinance to pull historical stock data for Commonwealth Bank of Australia (CBA.AX), scaled the data, and trained an LSTM model. After training, it generated a visual comparison of actual vs. predicted stock prices, which was a great starting point to understand the model's behavior. The process was smooth, and the code executed without issues, producing clear outputs for analysis.



```
: FutureWarning: YF.download() has changed argument auto_adjust default to True
  data = yf.download(COMPANY,TRAIN_START,TRAIN_END)
[*********************100%***********************]  1 of 1 completed
/Users/rahul/venvs/price_prediction/lib/python3.13/site-packages/keras/src/layers/rnn/rnn.py:199
: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequenti
al models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Epoch 1/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - loss: 0.1091
Epoch 2/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0134
Epoch 3/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0105
Epoch 4/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0080
Epoch 5/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0078
Epoch 6/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0075
Epoch 7/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0075
Epoch 8/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0073
Epoch 9/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0073
Epoch 10/25
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0074
```

Figure 1 – Training sequence for v0.1

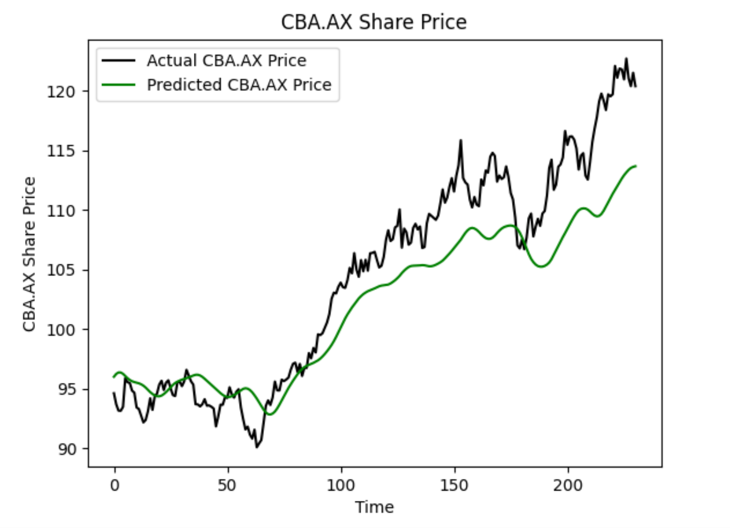


Figure 2 – Prediction diagram (v0.1)

Testing P1 Code

The P1 implementation was designed to be a more modular and flexible upgrade over v0.1, splitting the workflow into separate ***train.py*** and ***test.py*** scripts. However, getting it to run on my system was a journey filled with challenges, primarily due to the original reliance on ***yahoo_fin***. This library proved problematic, as it frequently failed to retrieve data reliably, causing execution errors early in the pipeline.

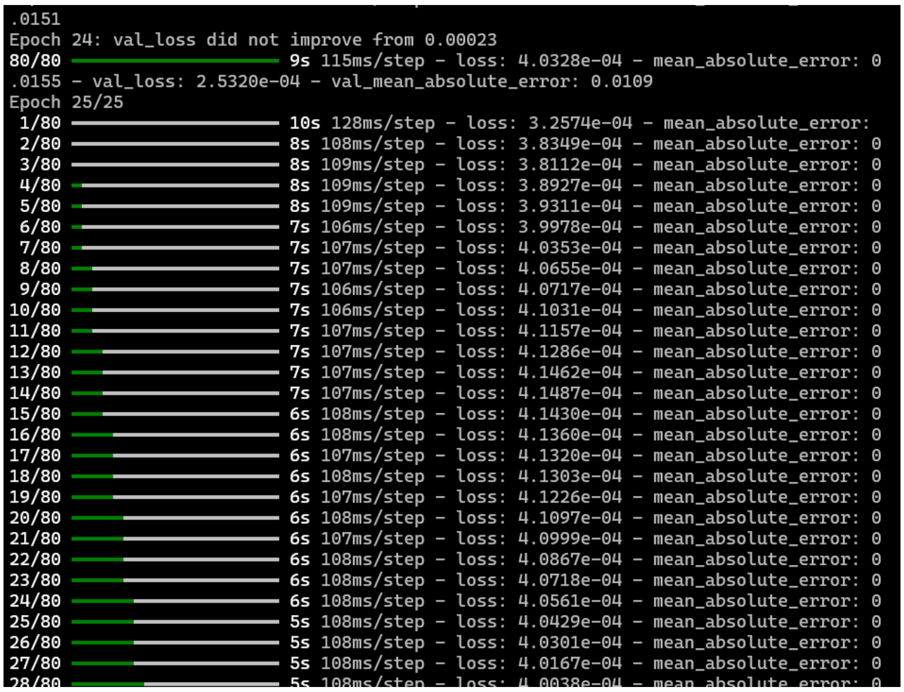
To address this, I replaced *yahoo_fin* with ***yfinance***, which is widely regarded for its reliability and seamless integration with *pandas*. This change wasn't as simple as a one-line swap, though. The *yfinance* library returned data with different column structures, such as capitalized column names (e.g., "Adj Close" instead of "adjclose") and, in some cases, a MultiIndex structure when fetching data for certain tickers. This led to a series of errors, including an AttributeError for attempting to call .lower() on tuple column names and an AssertionError because the expected adjclose column was missing.

I tackled these issues step-by-step. First, I modified the ***load_data*** function in *stock_prediction.py* to handle column renaming correctly, ensuring that column names were converted to strings and spaces were removed (e.g., "Adj Close" to "adjclose"). I also added a check for MultiIndex columns, as yfinance occasionally returned data in this format, causing further errors. Additionally, I set auto_adjust=False in the yf.download() call to ensure all necessary columns (like "Close" and "Adj Close") were included, matching the feature columns expected by the code.

Another hurdle came when the dataset size was insufficient. The error ValueError: No sequences generated indicated that the downloaded data didn't have enough rows to create sequences given the parameters n_steps=50 and lookup_step=15. To fix this, I added a start="2000-01-01" parameter to ***yf.download()*** to fetch a longer historical dataset, ensuring enough data points to generate sequences.

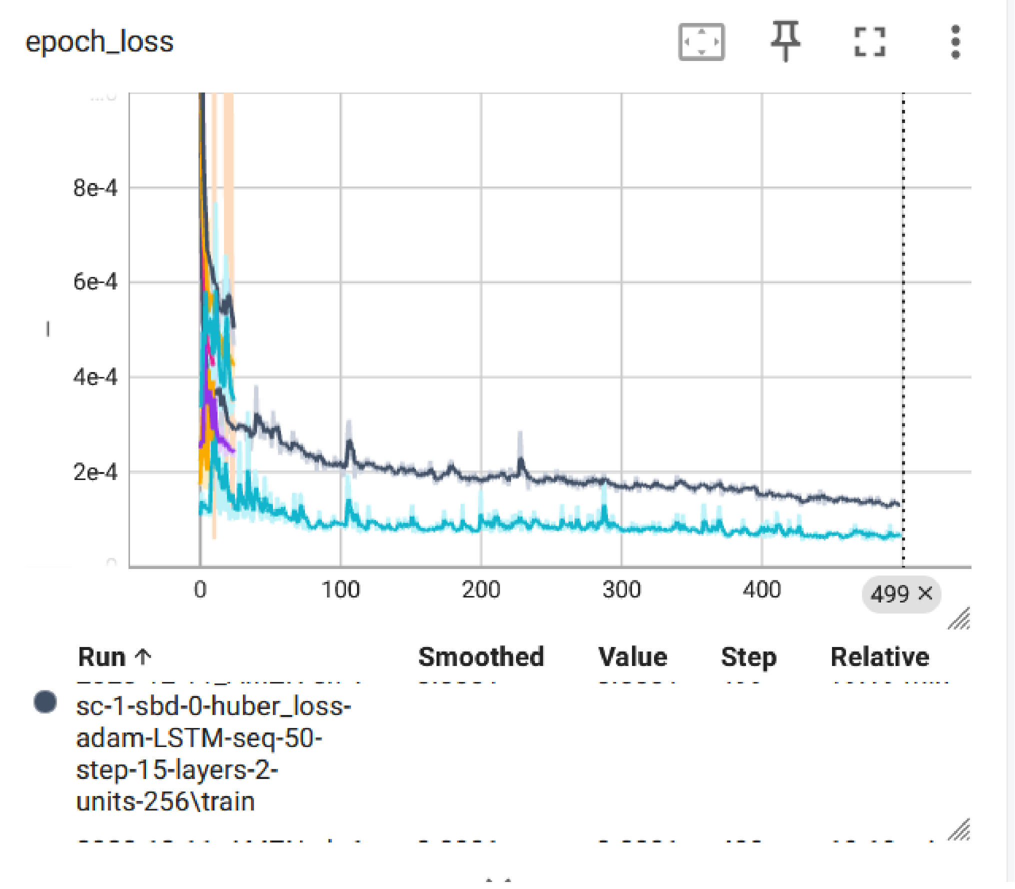
The final challenge was a TensorFlow-related error in the ***create_model*** function, where batch_input_shape was not recognized by the LSTM layer in newer TensorFlow versions. I replaced it with input_shape, a more compatible argument, to define the input dimensions for the model. Additionally, the ModelCheckpoint callback in train.py required the filepath to end in .weights.h5 when save_weights_only=True, so I updated the extension in both train.py and test.py to resolve this.

After these iterative fixes, the P1 code ran successfully. Running python train.py trained the LSTM model on Apple (AAPL) stock data, saving the weights to the results/ directory. Then, python test.py loaded the weights, evaluated the model, and produced performance metrics (loss, mean absolute error, accuracy score, and profit calculations) along with a plot comparing actual vs. predicted prices. The process was now stable and reproducible, relying entirely on live data from yfinance.



```
.0151
Epoch 24: val_loss did not improve from 0.00023
80/80          9s 115ms/step - loss: 4.0328e-04 - mean_absolute_error: 0
.0155 - val_loss: 2.5320e-04 - val_mean_absolute_error: 0.0109
Epoch 25/25
 1/80          10s 128ms/step - loss: 3.2574e-04 - mean_absolute_error:
 2/80          8s 108ms/step - loss: 3.8349e-04 - mean_absolute_error: 0
 3/80          8s 109ms/step - loss: 3.8112e-04 - mean_absolute_error: 0
 4/80          8s 109ms/step - loss: 3.8927e-04 - mean_absolute_error: 0
 5/80          8s 109ms/step - loss: 3.9311e-04 - mean_absolute_error: 0
 6/80          7s 106ms/step - loss: 3.9978e-04 - mean_absolute_error: 0
 7/80          7s 107ms/step - loss: 4.0353e-04 - mean_absolute_error: 0
 8/80          7s 107ms/step - loss: 4.0655e-04 - mean_absolute_error: 0
 9/80          7s 106ms/step - loss: 4.0717e-04 - mean_absolute_error: 0
10/80          7s 106ms/step - loss: 4.1031e-04 - mean_absolute_error: 0
11/80          7s 107ms/step - loss: 4.1157e-04 - mean_absolute_error: 0
12/80          7s 107ms/step - loss: 4.1286e-04 - mean_absolute_error: 0
13/80          7s 107ms/step - loss: 4.1462e-04 - mean_absolute_error: 0
14/80          7s 107ms/step - loss: 4.1487e-04 - mean_absolute_error: 0
15/80          6s 108ms/step - loss: 4.1430e-04 - mean_absolute_error: 0
16/80          6s 108ms/step - loss: 4.1360e-04 - mean_absolute_error: 0
17/80          6s 107ms/step - loss: 4.1320e-04 - mean_absolute_error: 0
18/80          6s 108ms/step - loss: 4.1303e-04 - mean_absolute_error: 0
19/80          6s 107ms/step - loss: 4.1226e-04 - mean_absolute_error: 0
20/80          6s 108ms/step - loss: 4.1097e-04 - mean_absolute_error: 0
21/80          6s 107ms/step - loss: 4.0999e-04 - mean_absolute_error: 0
22/80          6s 108ms/step - loss: 4.0867e-04 - mean_absolute_error: 0
23/80          6s 108ms/step - loss: 4.0718e-04 - mean_absolute_error: 0
24/80          6s 108ms/step - loss: 4.0561e-04 - mean_absolute_error: 0
25/80          5s 108ms/step - loss: 4.0429e-04 - mean_absolute_error: 0
26/80          5s 108ms/step - loss: 4.0301e-04 - mean_absolute_error: 0
27/80          5s 108ms/step - loss: 4.0167e-04 - mean_absolute_error: 0
28/80          5s 108ms/step - loss: 4.0038e-04 - mean_absolute_error: 0
```

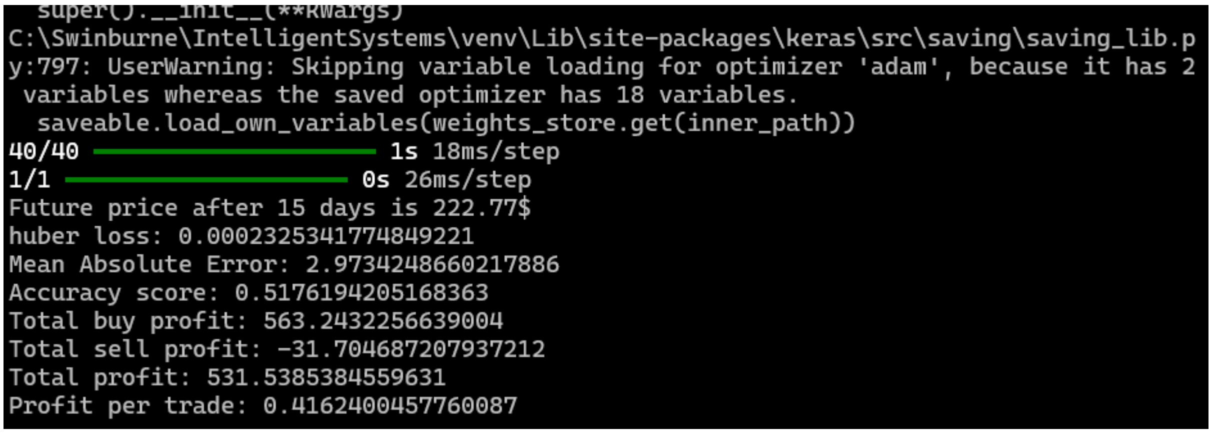
Figure 3 – Training P1 using train.py



Run ↑ Smoothed Value Step Relative

● sc-1-sbd-0-huber_loss-adam-LSTM-seq-50-step-15-layers-2-units-256\train

Figure 4 – Validation loss curve using tensorboard –logdir "logs"



```
  super().__init__(**kwargs)
C:\Swinburne\IntelligentSystems\venv\Lib\site-packages\keras\src\saving\saving_lib.p
y: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))
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
Future price after 15 days is 222.77$
huber loss: 0.0002325341774849221
Mean Absolute Error: 2.9734248660217886
Accuracy score: 0.5176194205168363
Total buy profit: 563.2432256639004
Total sell profit: -31.704687207937212
Total profit: 531.5385384559631
Profit per trade: 0.4162400457760087
```

Figure 5 – Results of running test.py

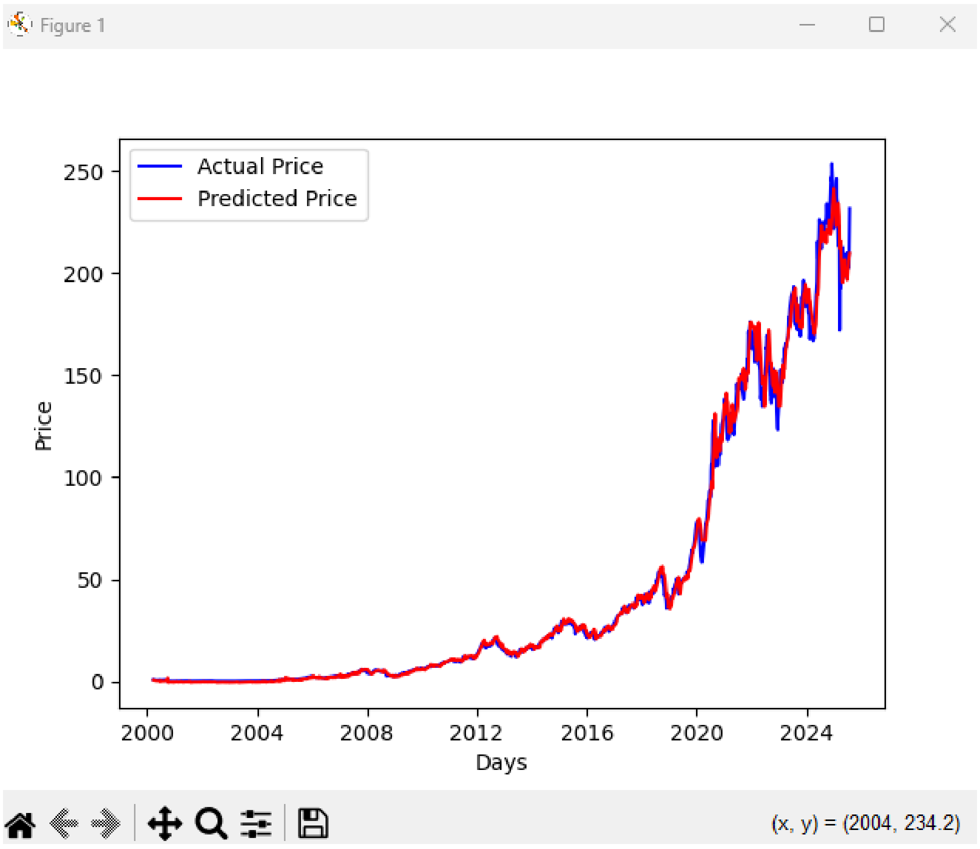


Figure 6 – P1 predicted price vs actual price result

Understanding of Initial Code Base (v0.1)

The v0.1 code followed a standard machine learning pipeline, which provided a solid foundation for understanding the P1 implementation:

1. **Data Collection**: Originally designed to fetch data via the Yahoo Finance API, it was updated to use yfinance for reliable data retrieval.
2. **Data Preprocessing**: Stock prices (specifically closing prices) were scaled using MinMaxScaler to normalize values between 0 and 1. The code created sliding windows of 50 days (n_steps) to form training sequences.
3. **Model Construction**: A stacked LSTM network with two layers, 256 units each, and 40% dropout was built using TensorFlow/Keras to prevent overfitting.
4. **Training**: The model was trained for 25 epochs with a batch size of 64, a reduction from the typical 500 epochs to balance speed and performance.
5. **Testing**: Predictions were generated and compared against actual prices, visualized in a plot.
6. **Next-Day Prediction**: The model used the most recent 50 days of data to predict the stock price 15 days ahead (lookup_step).

While v0.1 was functional, it had limitations:

- Dependence on live API calls made it vulnerable to network or API issues.
- It offered limited flexibility for experimenting with different features or parameters.
- It lacked detailed evaluation metrics beyond visual plots, making it hard to quantify performance.

These insights guided the transition to P1, which aimed to improve modularity, configurability, and robustness through separate training and testing scripts and more comprehensive evaluation metrics.

Final Working Setup

Getting the P1 code to run on my system required persistence, but the final setup was both reliable and efficient. Here's how it came together:

1. **Preparation**:
   - I set up a virtual environment using python3 -m venv venv and installed all required dependencies (numpy, matplotlib, pandas, pandas-datareader, scikit-learn, tensorflow, yfinance) using pip install.
   - I replaced yahoo_fin with yfinance in stock_prediction.py to ensure reliable data retrieval.
2. **Code Adjustments**:
   - Updated the load_data function to handle yfinance's column naming (e.g., converting "Adj Close" to "adjclose" and handling MultiIndex columns).
   - Added start="2000-01-01" to yf.download() to fetch sufficient historical data for AAPL, addressing the insufficient data error.
   - Modified the create_model function to use input_shape instead of batch_input_shape for TensorFlow compatibility.
   - Changed the ModelCheckpoint filepath extension to .weights.h5 in train.py and test.py to comply with TensorFlow's requirements.
3. **Execution of Training**:
   - Ran python train.py in the terminal, which downloaded AAPL stock data, preprocessed it, trained the LSTM model for 25 epochs, and saved the weights to results/<model_name>.weights.h5. The script also logged training progress to the logs/ directory for analysis.
4. **Execution of Testing**:
   - Ran python test.py, which loaded the trained model weights, evaluated the model on the test split, and generated predictions. The script outputted metrics (loss, mean absolute error, accuracy score, buy/sell profits) and saved a CSV file with results to csv-results/<model_name>.csv. A plot comparing actual vs. predicted prices was also displayed.
5. **Verification**:
   - Confirmed that the scripts executed without errors, producing consistent outputs across runs. The use of yfinance ensured fresh data, while the fixed parameters (n_steps=50, lookup_step=15, test_size=0.2) guaranteed reproducibility.