

Project Assignment

Task C1 – Report

Rajapaksha Mudiyansele Udara Ishan Bandara Embawa
105107155

Task C1

Summaries of Environment Setup

The initial version of the stock prediction tool (v0.1) was originally distributed as a Python script rather than a notebook format, so I migrated the code into a Jupyter Notebook and made several adaptations to streamline future development and usage. For both v0.1 and the current P1 workflow, essential dependencies are handled using commands such as “!pip install”, ensuring that all required packages are available in the working environment. This installation approach is standard practice in Jupyter-based workflows, whether running in cloud-hosted platforms like Google Colab or on a local notebook server with environments such as Anaconda or virtualenv.

Requirements

Platforms & Interfaces

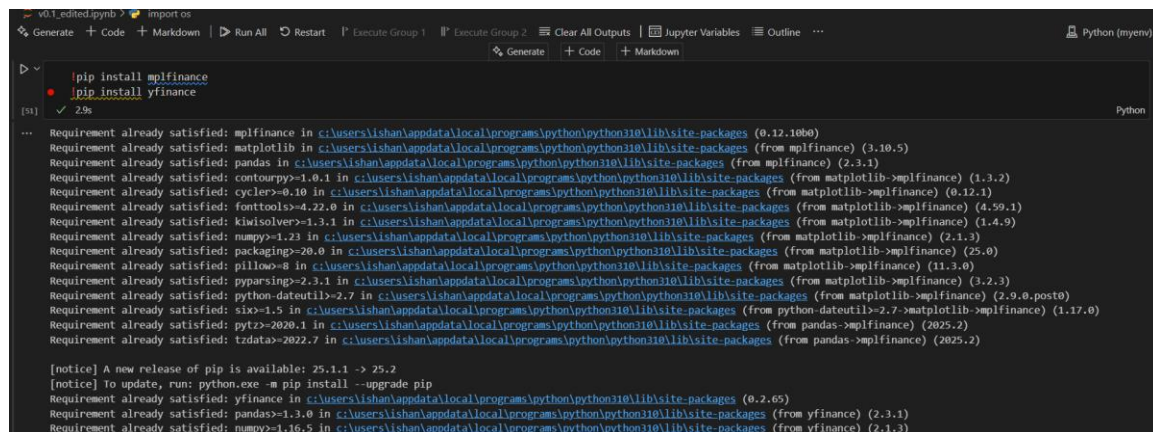
Virtual Environment and Dependencies

The virtual environment for this project is set up to work seamlessly with Jupyter Notebook, particularly within Visual Studio Code using the Jupyter extension. The first set of dependencies (v1) includes essential Python libraries such as numpy and pandas for numerical computations and data manipulation, matplotlib and mplfinance for comprehensive data visualization including candlestick charting, scikit-learn for data preprocessing and evaluation metrics, yfinance for reliable financial data extraction, and tensorflow/keras for building and training deep learning models, notably LSTM networks for time series prediction. Additionally, a second dependency set (p1) is similar but replaces yfinance with yahoo_finance, another library used to fetch historical stock data. This combination ensures a robust and flexible environment for developing, visualizing, and evaluating stock price prediction models in a reproducible and efficient manner.

Installation

Installation Steps

1. Download and Install Required Tools
 - Steps to obtain Anaconda or set up Google Colab.
2. Create New Environment
 - Step-by-step for isolating project requirements.
3. Install Jupyter Notebook & Dependencies
 - Use conda or pip commands to install Jupyter and all libraries.
 - Example: `!pip install <package>`
 - Example: `conda install jupyterlab`



```

v01_edited.ipynb | import os
Generate + Code + Markdown ▶ Run All ⌂ Restart ▶ Execute Group 1 ▶ Execute Group 2 Clear All Outputs Jupyter Variables Outline ... Python (myenv)
pip install mplfinance
pip install yfinance
[51] ✓ 29s
... Requirement already satisfied: mplfinance in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (0.12.10b0)
Requirement already satisfied: matplotlib in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from mplfinance) (3.10.5)
Requirement already satisfied: pandas in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from mplfinance) (2.3.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->mplfinance) (1.3.2)
Requirement already satisfied: cycler>=0.10 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->mplfinance) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->mplfinance) (4.59.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->mplfinance) (1.4.9)
Requirement already satisfied: numpy>=1.23 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->mplfinance) (2.1.3)
Requirement already satisfied: packaging>=20.0 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->mplfinance) (25.0)
Requirement already satisfied: pillow>=8 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->mplfinance) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->mplfinance) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from matplotlib->mplfinance) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from python-dateutil->2.7->matplotlib->mplfinance) (1.17.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from pandas->mplfinance) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from pandas->mplfinance) (2025.2)

[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: yfinance in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (0.2.65)
Requirement already satisfied: pandas>=1.3.0 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from yfinance) (2.3.1)
Requirement already satisfied: numpy>=1.16.5 in c:\users\ishan\appdata\local\programs\python\python310\lib\site-packages (from yfinance) (2.1.3)

```

Understanding of the Initial Codebase

v1 Codebase

The "v1" script is primarily centered on stock price prediction using historical market data. It focuses on predicting the future closing prices of a selected stock ticker by employing a Long Short-Term Memory (LSTM) neural network, a type of recurrent neural network well-suited for time series data like stock prices. The script showcases a typical machine learning pipeline that includes data loading, preprocessing, visualization, model construction, training, and forecasting, with an emphasis on forecasting the price movements of financial assets.

For data loading and preprocessing, the script fetches historical stock price data, which includes columns such as Open, High, Low, Close, and Volume, typically sourced through yfinance. It cleans and processes the data by ensuring the correct format and renaming

columns if necessary (e.g., using adjusted close prices). The Close prices are then scaled using `MinMaxScaler` to normalize the range of values for effective LSTM training. The script also creates sequences from the scaled data by sliding a fixed-size window over it to generate the inputs (X) and expected outputs (y) for the model. This sequence generation captures temporal dependencies essential for predicting future prices.

Visualization is employed through candlestick charts and line plots of high and low prices to provide intuitive market insights. After preprocessing, the LSTM model is built with layers including two recurrent layers and dropout regularization to reduce overfitting. Following training, it predicts stock prices for a defined forecast horizon (e.g., next 10 days), and the script plots actual versus predicted future prices. Additionally, the model's historical data and predictions can be saved as CSV files for further analysis or use. Overall, the script integrates exploratory visualization, rigorous data transformation, and modern deep learning techniques to forecast stock market trends.

p1 Codebase

The “p1” script is a comprehensive stock price prediction model developed specifically for Amazon (ticker: AMZN) that uses historical stock price data as input. It processes decades of price data from the late 1990s to 2025, with rich features including open, high, low, close, adjusted close, and volume. The model focuses on predicting adjusted closing prices 15 days ahead, alongside calculating hypothetical buy and sell profits to measure potential trading profitability. It integrates advanced LSTM (Long Short-Term Memory) neural networks to capture temporal dependencies in stock prices and provides detailed output like predicted prices and profits for trading signals, which show the financial gains from buying or selling based on the model's forecast.

In comparison, the v1 script (shown in the “v0.1_edited.ipynb”) is a more general-purpose implementation designed to work on different stocks, exemplified here with Commonwealth Bank Australia (ticker: CBA.AX). It uses a streamlined LSTM model architecture with two LSTM layers followed by dropout and dense layers, leveraging recent years' stock data (2020-2023) with standard preprocessing like normalization and sequence generation for training and testing. The v1 model includes additional functionalities such as candlestick plotting, high-low price plotting, error metrics (RMSE, MAE) reporting, and options to save predictions and historical data to CSV. The architecture is simpler and focused mainly on close price prediction with an added feature of forecasting 10 days into the future. The innovative difference in “p1” lies in its

extended forecast horizon (15 days) with profit calculations embedded directly into the dataset and a potentially larger historical scope compared to the v1's shorter-term predictive focus and more general modular design for retraining on arbitrary stocks.

Modified v1

The modified v1 introduces several improvements:

1. Visualization Enhancements:

- Actual vs. Predicted prices plotted with clear legends.
- Future predictions displayed as dashed orange lines for interpretability.
- Intuitive chart with titles, labels, and legends.

2. Predictions for Future Dates:

- Model now predicts stock prices for 10 upcoming days.
- These predictions are displayed on the same graph with historical and test predictions.

3. Evaluation Metrics:

- Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are calculated and displayed.

4. Export to CSV:

- Historical stock data is exported to '<COMPANY>_historical.csv'.
- Model predictions are exported to '<COMPANY>_predictions.csv'.

Output & Analysis

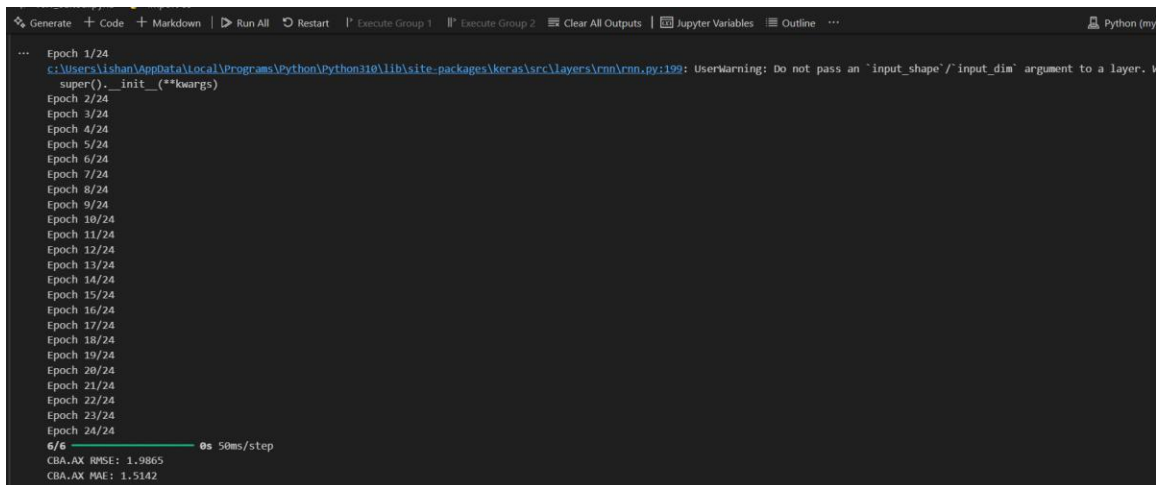
Training Process

The model training process involves preparing historical stock price data, specifically the "Close" prices, which are scaled using MinMaxScaler to normalize values between 0 and 1. The dataset is then split into training and testing sets, typically reserving 20% of the data for testing. The input data is organized into sequences with a specified time step, usually 60 days, where each sequence includes the previous 60 days' scaled prices as features and the next day's price as the target. This sequence preparation allows the model to learn temporal dependencies effectively. The model architecture is a sequential LSTM network composed of two stacked LSTM layers with 50 units each,

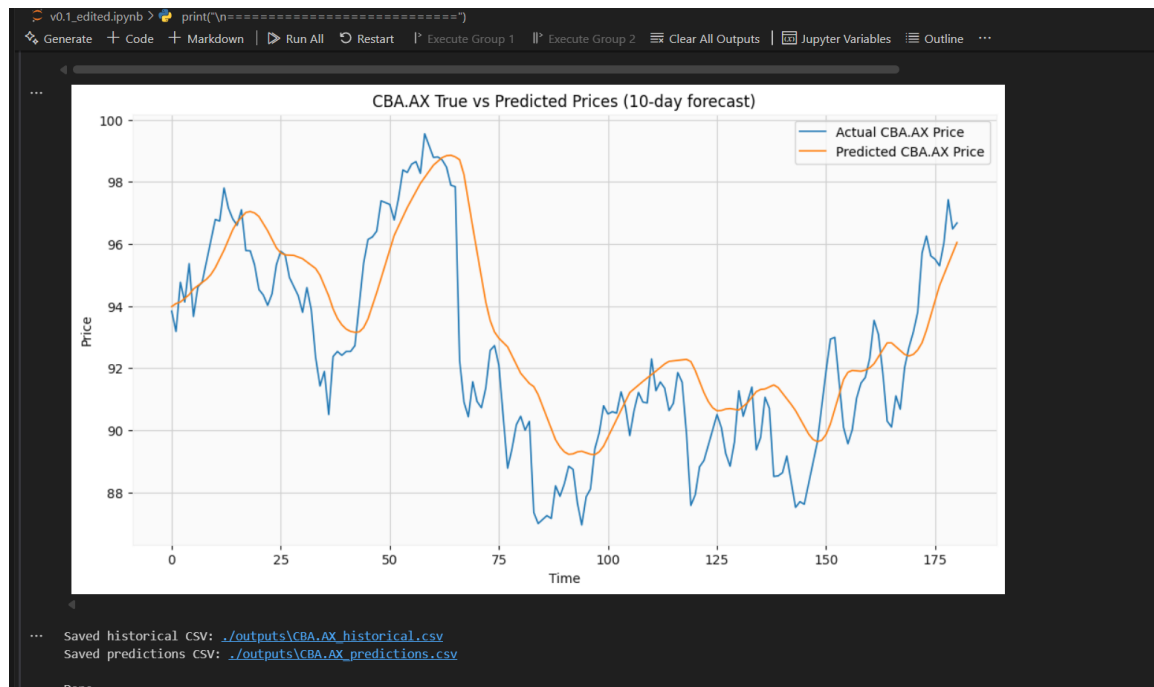
interspersed with dropout layers at 20% to mitigate overfitting. Finally, dense layers reduce output dimensions to a single predicted price. The model is compiled with the Adam optimizer and trained using mean squared error loss.

Training is conducted over multiple epochs where the model learns to minimize the prediction error on the training set. Epochs represent how many times the model sees the entire training data, typically adjusted to achieve good convergence without overfitting. Once trained, the model predicts future prices for the test data, and an additional prediction is conducted for an extended forecast horizon (e.g., 10 days ahead) using a sliding window approach where the predicted recent values are fed back as inputs to forecast further ahead. The results are evaluated using typical regression metrics such as mean squared and mean absolute errors, then visualized with plots comparing actual and predicted prices, along with candlestick charts for detailed price movement. The relevant data, including historical prices and predictions, is saved in CSV format for further analysis or validation. This workflow is implemented consistently for both the initial (v1) and later (p1) code versions, reflecting a systematic approach in handling stock price forecasting with LSTM networks.

V1

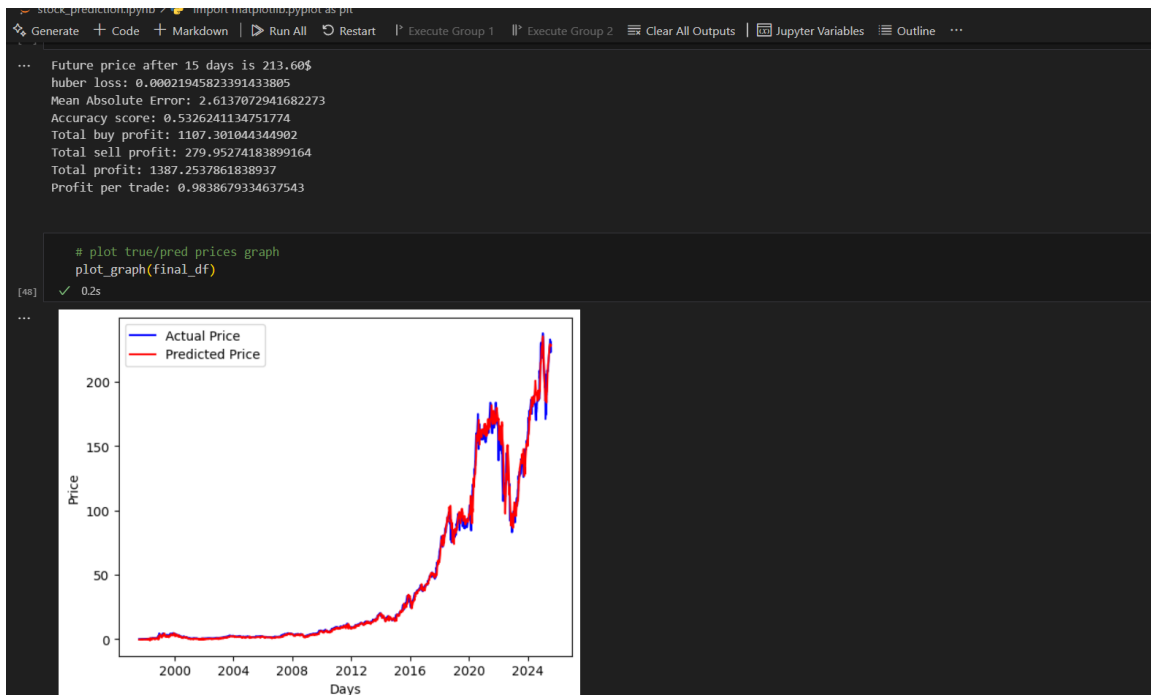


```
... Epoch 1/24
c:\Users\ishan\AppData\Local\Programs\Python\python310\lib\site-packages\keras\src\layers\rnn\rnn.py:199: UserWarning: Do not pass an 'input_shape'/'input_dim' argument to a layer.
super().__init__(**kwargs)
Epoch 2/24
Epoch 3/24
Epoch 4/24
Epoch 5/24
Epoch 6/24
Epoch 7/24
Epoch 8/24
Epoch 9/24
Epoch 10/24
Epoch 11/24
Epoch 12/24
Epoch 13/24
Epoch 14/24
Epoch 15/24
Epoch 16/24
Epoch 17/24
Epoch 18/24
Epoch 19/24
Epoch 20/24
Epoch 21/24
Epoch 22/24
Epoch 23/24
Epoch 24/24
6/6 0s 50ms/step
CBA.AX RMSE: 1.9865
CBA.AX MAE: 1.5142
```



P1

```
[*] ✓ 85m 50.3s
... Epoch 1/500
88/89 → 0s 108ms/step - loss: 0.0059 - mean_absolute_error: 0.0492
Epoch 1: val_loss improved from inf to 0.00046, saving model to results\2025-08-21_AMZN-sh-1-sc-1-sbd-0-huber-adam-lstm-seq-50-step-15-layers-2-units-256.weights.h5
89/89 → 13s 119ms/step - loss: 0.0058 - mean_absolute_error: 0.0488 - val_loss: 4.5804e-04 - val_mean_absolute_error: 0.0155
Epoch 2/500
88/89 → 0s 105ms/step - loss: 7.2669e-04 - mean_absolute_error: 0.0192
Epoch 2: val_loss improved from 0.00046 to 0.00030, saving model to results\2025-08-21_AMZN-sh-1-sc-1-sbd-0-huber-adam-lstm-seq-50-step-15-layers-2-units-256.weights.h5
89/89 → 10s 113ms/step - loss: 7.2480e-04 - mean_absolute_error: 0.0192 - val_loss: 2.9601e-04 - val_mean_absolute_error: 0.0122
Epoch 3/500
88/89 → 0s 106ms/step - loss: 6.9446e-04 - mean_absolute_error: 0.0189
Epoch 3: val_loss did not improve from 0.00030
89/89 → 10s 113ms/step - loss: 6.9487e-04 - mean_absolute_error: 0.0189 - val_loss: 3.2026e-04 - val_mean_absolute_error: 0.0141
Epoch 4/500
88/89 → 0s 105ms/step - loss: 5.8435e-04 - mean_absolute_error: 0.0172
Epoch 4: val_loss did not improve from 0.00030
89/89 → 10s 113ms/step - loss: 5.8558e-04 - mean_absolute_error: 0.0173 - val_loss: 3.4829e-04 - val_mean_absolute_error: 0.0139
Epoch 5/500
88/89 → 0s 105ms/step - loss: 6.4701e-04 - mean_absolute_error: 0.0181
Epoch 5: val_loss did not improve from 0.00030
89/89 → 10s 112ms/step - loss: 6.4621e-04 - mean_absolute_error: 0.0181 - val_loss: 3.5547e-04 - val_mean_absolute_error: 0.0144
Epoch 6/500
88/89 → 0s 106ms/step - loss: 5.9911e-04 - mean_absolute_error: 0.0180
Epoch 6: val_loss did not improve from 0.00030
89/89 → 10s 113ms/step - loss: 5.9881e-04 - mean_absolute_error: 0.0180 - val_loss: 3.0621e-04 - val_mean_absolute_error: 0.0124
Epoch 7/500
...
Epoch 500/500
88/89 → 0s 116ms/step - loss: 3.5564e-04 - mean_absolute_error: 0.0156
Epoch 500: val_loss did not improve from 0.00022
89/89 → 11s 124ms/step - loss: 3.5500e-04 - mean_absolute_error: 0.0156 - val_loss: 2.2122e-04 - val_mean_absolute_error: 0.0107
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```



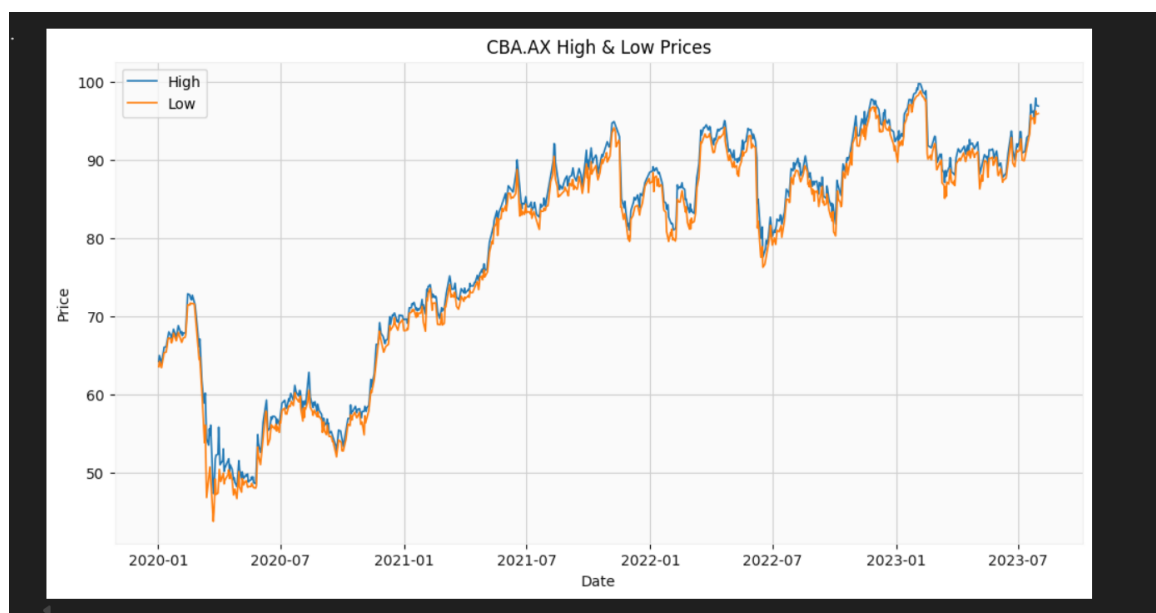
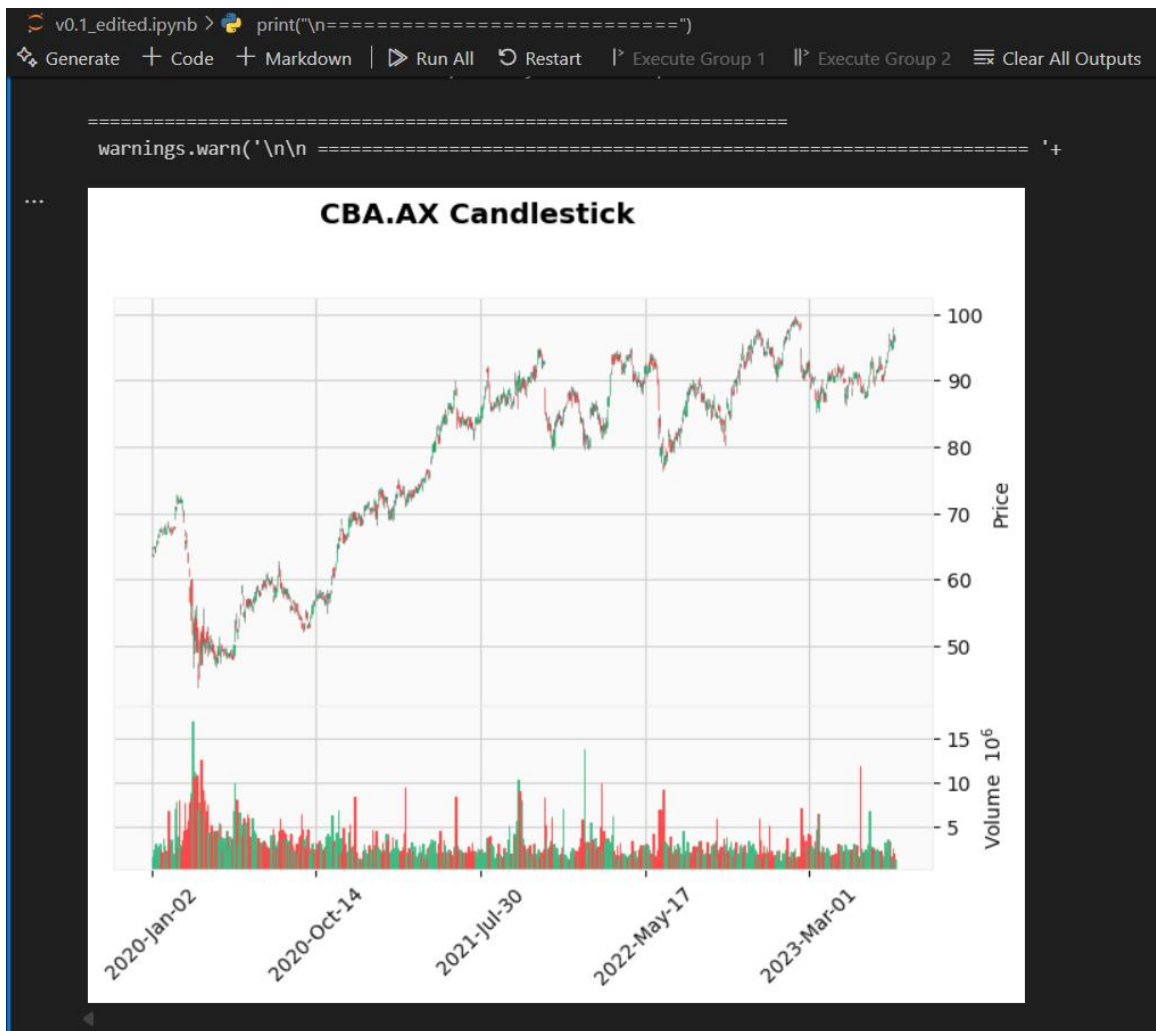
Graphical Outputs

- Historical Data Visualization

Historical data visualization in stock analysis involves plotting price and volume data over time to observe trends, volatility, and trading activity. Common plots include:

- Price plots such as candlestick charts, which show open, high, low, and close prices to reveal trading patterns and market sentiment.
- Line plots of high and low prices highlighting price ranges over a period.
- Volume plots showing the number of shares traded, useful for identifying the strength of price movements.

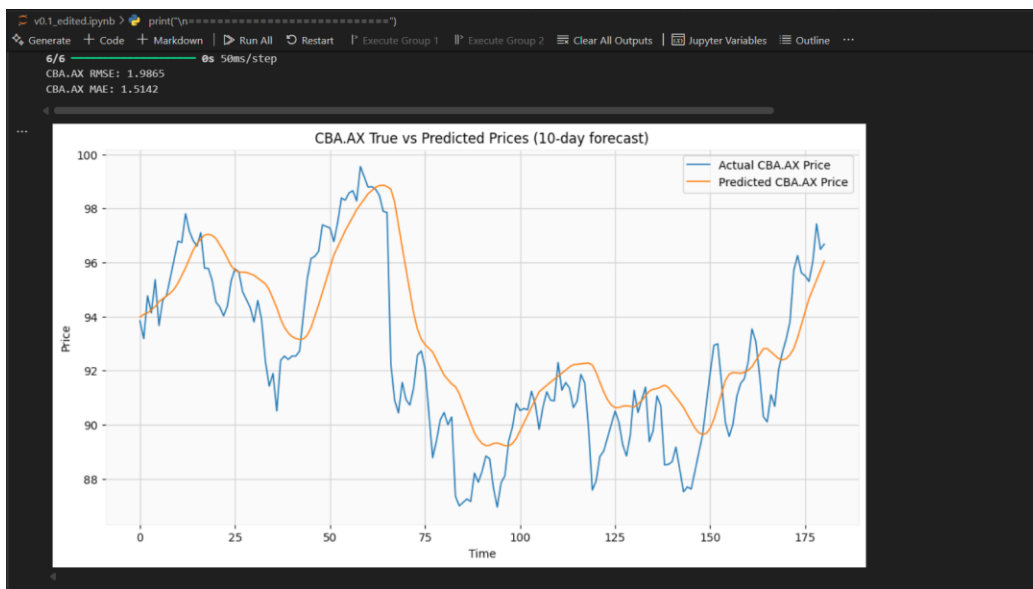
These visualizations help traders and analysts understand market behavior, identify entry and exit points, and assess stock performance historically. The code you provided uses `mplfinance` and `matplotlib` to plot candlestick charts, high/low price lines, and volume in conjunction with price data for comprehensive historical data visualization.



- Model Predictive Performance

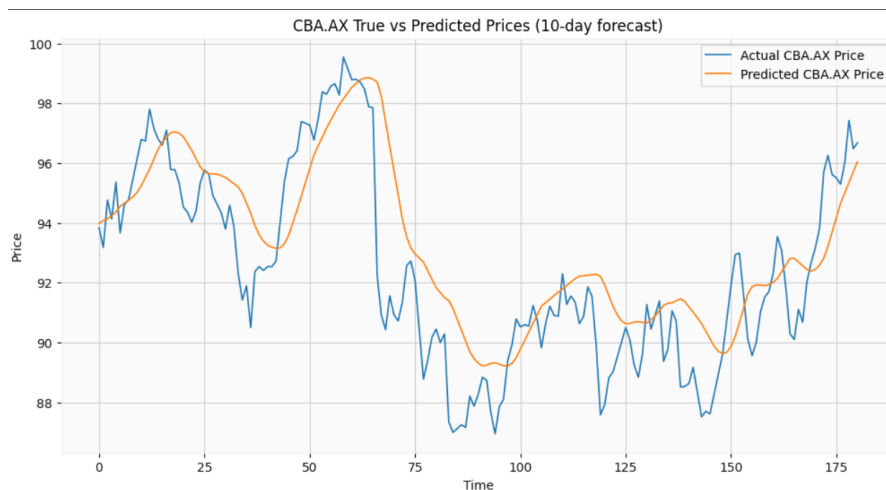
The model's predictive performance was evaluated with a Root Mean Squared Error and a Mean Absolute Error, indicating the prediction accuracy of the stock price forecast.

A plot was created to visually compare the actual versus predicted prices for a 10-day forecast period, providing a clear illustration of the model's effectiveness in capturing price trends. This visualization helps assess how closely the predicted prices matched the real market prices over that timeframe.



- Future Predictions / Forecast

Visualize forecasts for upcoming days.



Exported/Generated Files

During the analysis, the following CSV data files were saved:

- A CSV file containing the historical stock prices for the analyzed company (ticker symbol given as COMPANY), saved as "{ticker}_historical.csv"
- A CSV file containing the predicted stock prices for the test dataset period, saved as "{ticker}_predictions.csv"

These files include detailed price data and prediction outputs used in the stock price forecasting process. The saving paths depend on the configured save directory, which can be either a local folder or a Google Drive folder if enabled.

This allows for external review and further use of both historical data and forecasted future price values generated by the model during analysis.

```
Saved historical CSV: ./outputs\CBA.AX\_historical.csv  
Saved predictions CSV: ./outputs\CBA.AX\_predictions.csv
```

```
Done.
```

Issues & Debugging

During this assignment, I encountered challenges primarily related to installing the required Python libraries, such as `mplfinance` and `yfinance`, due to version mismatches and compatibility errors. Additionally, I faced issues with fetching stock data for certain ticker symbols using `yfinance`, where sometimes no data was returned or columns were misnamed, requiring data cleaning and renaming for consistent processing. I overcame these problems by carefully managing package versions, ensuring proper installation, and adding checks and cleanup steps in the data fetching code to handle missing or inconsistent data gracefully. These solutions allowed the assignment to progress smoothly and the models to train and predict successfully.

```

PS D:\Intelligent Systems> & "D:/Intelligent Systems/venv/Scripts/Activate.ps1"
(venv) PS D:\Intelligent Systems> python stock_prediction.py
2025-08-22 02:09:54.718933: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
omputation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-08-22 02:09:56.399765: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
omputation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
Failed to get ticker 'CBA.AX' reason: Expecting value: line 1 column 1 (char 0)
[*****100%*****] 1 of 1 completed

1 Failed download:
['CBA.AX']: Exception('%ticker%: No timezone found, symbol may be delisted')
Traceback (most recent call last):
  File "D:\Intelligent Systems\stock_prediction.py", line 65, in <module>
    scaled_data = scaler.fit_transform(data[PRICE_VALUE].values.reshape(-1, 1))
  File "D:\Intelligent Systems\venv\lib\site-packages\sklearn\utils\_set_output.py", line 316, in wrapped
    data_to_wrap = f(self, X, *args, **kwargs)
  File "D:\Intelligent Systems\venv\lib\site-packages\sklearn\base.py", line 894, in fit_transform
    return self.fit(X, **fit_params).transform(X)
  File "D:\Intelligent Systems\venv\lib\site-packages\sklearn\preprocessing\_data.py", line 454, in fit
    return self.partial_fit(X, y)
  File "D:\Intelligent Systems\venv\lib\site-packages\sklearn\base.py", line 1365, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "D:\Intelligent Systems\venv\lib\site-packages\sklearn\preprocessing\_data.py", line 494, in partial_fit
    x = validate_data(
  File "D:\Intelligent Systems\venv\lib\site-packages\sklearn\utils\validation.py", line 2954, in validate_data
    out = check_array(X, input_name="X", **check_params)
  File "D:\Intelligent Systems\venv\lib\site-packages\sklearn\utils\validation.py", line 1128, in check_array
    raise ValueError(
ValueError: Found array with 0 sample(s) (shape=(0, 1)) while a minimum of 1 is required by MinMaxScaler.
(venv) PS D:\Intelligent Systems> python check.py

```