ACML Project: LSTM Stock Price Predictor

Kudzai Saurombe (2503314)
Riot Ndlovu (2096330)

May 27, 2024

1 Introduction

1.1 Project Overview

In this project, we aim to predict the closing stock price of Apple Inc. (AAPL) using a Long Short-Term Memory (LSTM) model. Accurate stock price prediction is crucial for investors and analysts to make informed decisions. The closing price is the final price at which a stock is traded on a given trading day, and it often reflects the day's performance of a stock.

Apple Inc. (AAPL) is one of the most valuable companies globally, making its stock price an essential indicator for market trends and investor sentiment. The data used for this project includes the historical prices and volumes of Apple stock, which are essential for developing a reliable predictive model.

1.2 Background Information

Stock prices are influenced by various factors, including company performance, investor sentiment, market trends, and economic indicators. The closing price, in particular, is a critical metric as it represents the final price at which the stock was traded on a given day, summarising the day's market activity.

Key Data Components:

- **Open**: The price at which the stock starts trading when the market opens.
- **High**: The highest price at which the stock traded during the day.
- **Low**: The lowest price at which the stock traded during the day.
- **Close**: The final price at which the stock traded when the market closes.
- **Volume**: The total number of shares traded during the day.
- **Dividends**: The distribution of a portion of the company's earnings to shareholders.
- **Stock Splits**: Events where a company increases its number of shares while reducing the price of each share proportionately.

For more information on how stock prices are determined and their significance, you can refer to the following resources:

- Investopedia - How Stock Prices Are Determined
- Apple Inc. Financials

1.3 Objectives

The primary objectives of this project are:

- **Data Collection**: Collect historical stock price data of Apple Inc. using the `yfinance` library.
- **Data Preprocessing**: Clean and preprocess the data to ensure it is suitable for modeling.
- **Model Selection**: Evaluate different models and select the Long Short-Term Memory (LSTM) model for its ability to capture long-term dependencies in time series data.

- **Model Training**: Train the LSTM model on the preprocessed data.
- **Model Evaluation**: Assess the model's performance using appropriate metrics and compare predicted stock prices with actual values.
- **Result Analysis**: Analyse the results to determine the effectiveness of the model and identify areas for improvement.

Why LSTM?

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) designed to overcome the limitations of traditional RNNs, such as the vanishing and exploding gradient problems. LSTMs are particularly effective for time series prediction tasks due to their ability to capture long-term dependencies.

Key components of LSTM:

- **Cell State**: The "memory" of the network that carries information across different time steps.
- **Forget Gate**: Decides which information should be discarded from the cell state.
- **Input Gate**: Determines which new information should be added to the cell state.
- **Output Gate**: Controls the output of the cell state to the next hidden state.

These features make LSTMs more suitable for our task of predicting stock prices, where capturing long-term dependencies and trends is essential.

2 Data Collection

2.1 Data description

The dataset contains the following columns:

- **Date**: The trading date.
- **Open**: The opening price of the stock.
- **High**: The highest price of the stock during the trading day.
- **Low**: The lowest price of the stock during the trading day.
- **Close**: The closing price of the stock.
- **Volume**: The number of shares traded.
- **Dividends**: Dividends paid.
- **Stock Splits**: Stock splits occurred.

2.2 Source of the Data

The historical stock data of Apple Inc. (AAPL) was sourced using the `yfinance` library, which provides easy access to historical market data from Yahoo Finance. This dataset includes crucial financial metrics such as the open, high, low, and close prices, as well as the trading volume. The dataset spans from Apple's initial public offering (IPO) in 1980 to the present day.

We chose this data for several reasons:

1. **Relevance**: Apple Inc. is one of the most valuable and widely traded companies globally, making its stock price a significant indicator for market trends and investor sentiment.
2. **Data Availability**: The `yfinance` library offers comprehensive historical data that is easy to access and use, ensuring we have a robust dataset for training our model.
3. **Rich Features**: The dataset includes essential financial metrics that are critical for stock price prediction models.

2.3 Data Collection Process

The following code snippet, which is part of the `main.py` file, demonstrates how we collected and saved the Apple stock data:

```
import yfinance as yf

# Download Apple stock data
apple = yf.Ticker("AAPL")
apple_data = apple.history(period="max")

# Save the data to a CSV file
apple_data.to_csv("apple_stock_data.csv")
```

2.4 Initial Data Analysis

Once the data was collected, we conducted an initial analysis to understand its structure and key statistics. Below are the outputs from our analysis:

1. **First Few Rows of the Data**:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1980-12-12 00:00:00-05:00	0.099058	0.099488	0.099058	0.099058	469033600	0.0	0.0
1980-12-15 00:00:00-05:00	0.094321	0.094321	0.093890	0.093890	175884800	0.0	0.0
1980-12-16 00:00:00-05:00	0.087429	0.087429	0.086999	0.086999	105728000	0.0	0.0
1980-12-17 00:00:00-05:00	0.089152	0.089582	0.089152	0.089152	86441600	0.0	0.0
1980-12-18 00:00:00-05:00	0.091737	0.092167	0.091737	0.091737	73449600	0.0	0.0

2. **Summary Statistics**:

	Open	High	Low	Close	Volume	Dividends	Stock Splits
count	10950.000000	10950.000000	10950.000000	10950.000000	1.095000e+04	10950.000000	10950.000000
mean	20.672239	20.894552	20.458996	20.685669	3.191807e+08	0.000756	0.001553
std	43.892892	44.361599	43.454084	43.927929	3.357638e+08	0.011779	0.083846
min	0.038331	0.038331	0.037900	0.037900	0.000000e+00	0.000000	0.000000
25%	0.241563	0.246553	0.236681	0.241579	1.140464e+08	0.000000	0.000000
50%	0.425755	0.433244	0.418958	0.426790	2.067856e+08	0.000000	0.000000
75%	17.029777	17.170623	16.853115	17.012603	3.994732e+08	0.000000	0.000000
max	197.499763	199.095551	196.482439	197.589523	7.421641e+09	0.250000	7.000000

3. **Dataset Information**:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10950 entries, 0 to 10949
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          10950 non-null  object
 1   Open          10950 non-null  float64
 2   High          10950 non-null  float64
 3   Low           10950 non-null  float64
 4   Close         10950 non-null  float64
 5   Volume        10950 non-null  int64
 6   Dividends     10950 non-null  float64
 7   Stock Splits  10950 non-null  float64
dtypes: float64(6), int64(1), object(1)
memory usage: 684.5+ KB
```

4. **Correlation Matrix**: The correlation matrix helped us understand the relationships between different features in the dataset.

   - The **'Open'**, **'High'**, **'Low'**, and **'Close'** prices have very high correlations (close to 1) with each other. This is expected as these prices are interrelated.
   - The **'Volume'** shows a negative correlation with the prices, indicating that higher volumes may be associated with lower prices and vice versa. However, the correlation is relatively weak (-0.25).
   - The **'Dividends'** and **'Stock Splits'** have very low correlations with the stock prices and volume. Specifically, **'Dividends'** have a maximum correlation of approximately 0.099 with the stock prices, and **'Stock Splits'** show almost no correlation with any of the variables.

   Due to the weak correlations of **'Dividends'** and **'Stock Splits'** with the target variable ('Close' price), these columns were excluded from further analysis. This decision helps simplify the model without losing significant predictive power.

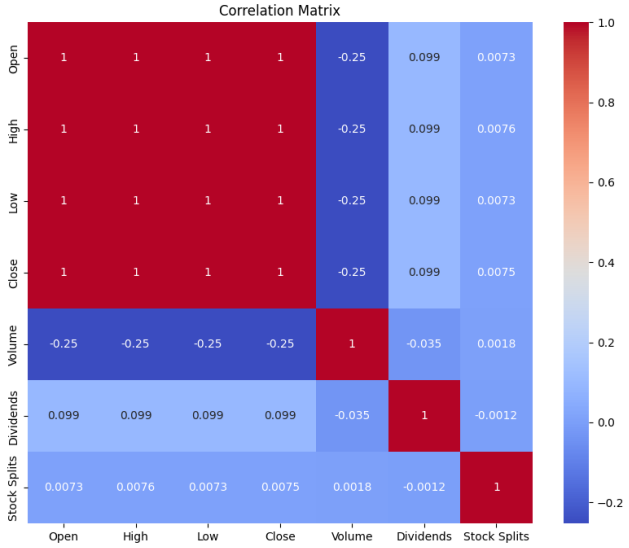

Figure 1: correlation Matrix

5. **Distribution Plots**: Visualising the distribution of each feature helps us understand the data's distribution and identify any potential anomalies.

   From the distribution plots, we observed that the stock prices (Open, High, Low, Close) are highly skewed to the right, with a significant concentration of data points at the lower end of the price spectrum. The volume data also exhibits a right skewed distribution with a long tail. These observations indicate that the features have varying scales and distributions, which necessitates normalisation to ensure effective training of the model. **Normalisation Approach**: Given the skewed nature of the data, we chose to use `MinMaxScaler` for normalisation. The `MinMaxScaler` scales the data to a fixed range, typically 0 to 1, which helps in:

   - Bringing all features to the same scale, facilitating better convergence during model training.
   - Preserving the relationships (ratios) between data points, which is crucial for time series data.
   - Handling skewed distributions effectively by compressing the long tails and expanding the concentrated regions, leading to more balanced input for the model.

```
from sklearn.preprocessing import MinMaxScaler
Select the relevant columns
data = df[['Open', 'High', 'Low', 'Close', 'Volume']]
Normalise the data
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
scaled_df = pd.DataFrame(scaled_data, columns=data.columns, index=data.index)
```

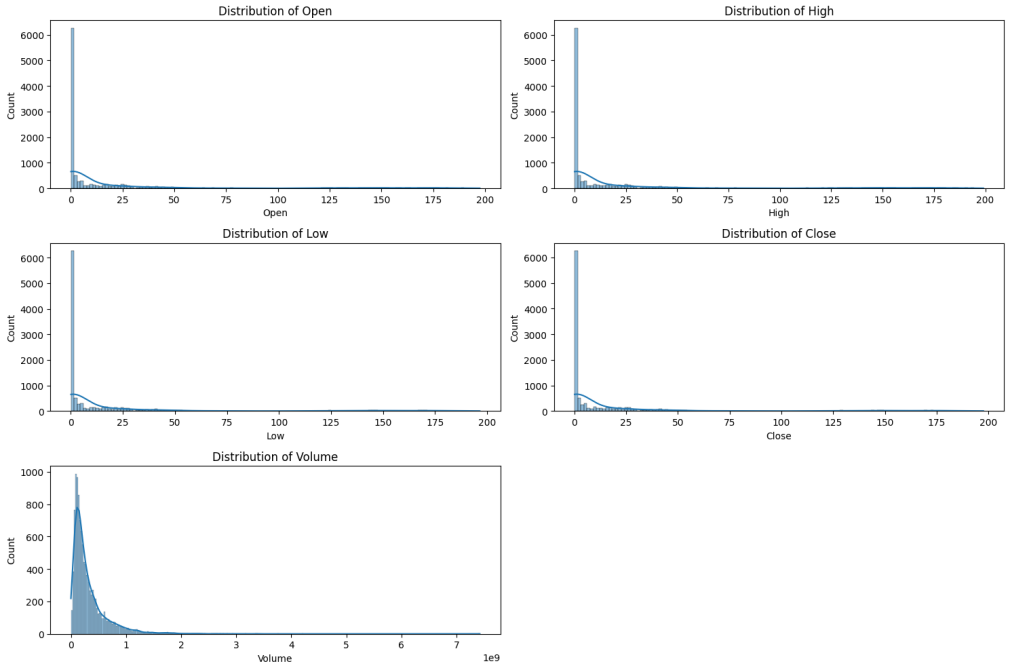

Figure 2: distribution of variables

By conducting this initial analysis, we ensure that our data is clean, well understood, and ready for the next steps in our modelling process.

3 Data Preprocessing

3.1 Loading and Inspecting Data

We began by loading the historical stock data into a Pandas DataFrame. This step was essential for examining the data's structure, identifying any missing values, and generating summary statistics to understand the data better.

```
import pandas as pd

# Load the dataset
file_path = 'apple_stock_data.csv'
df = pd.read_csv(file_path)

# Display the first few rows of the dataframe
print(df.head())

# Display information about the dataset
print(df.info())

# Get summary statistics
print(df.describe())
```

Output:

```
    Date        Open          High       Low     Close    Volume  Dividends  Stock Splits
0  1980-12-12 00:00:00-05:00  0.099058  0.099488  0.099058  0.099058  469033600        0.0           0.0
1  1980-12-15 00:00:00-05:00  0.094321  0.094321  0.093890  0.093890  175884800        0.0           0.0
2  1980-12-16 00:00:00-05:00  0.087429  0.087429  0.086999  0.086999  105728000        0.0           0.0
3  1980-12-17 00:00:00-05:00  0.089152  0.089582  0.089152  0.089152   86441600        0.0           0.0
4  1980-12-18 00:00:00-05:00  0.091737  0.092167  0.091737  0.091737   73449600        0.0           0.0
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10950 entries, 0 to 10949
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          10950 non-null  object
 1   Open          10950 non-null  float64
 2   High          10950 non-null  float64
 3   Low           10950 non-null  float64
 4   Close         10950 non-null  float64
 5   Volume        10950 non-null  int64
 6   Dividends     10950 non-null  float64
 7   Stock Splits  10950 non-null  float64
dtypes: float64(6), int64(1), object(1)
memory usage: 684.5+ KB
```

Open	High	Low	Close	Volume	Dividends	Stock Splits	
count	10950.000000	10950.000000	10950.000000	10950.000000	1.095000e+04	10950.000000	10950.000000
mean	20.672239	20.894552	20.458996	20.685669	3.191807e+08	0.000756	0.001553
std	43.892892	44.361599	43.454084	43.927929	3.357638e+08	0.011779	0.083846
min	0.038331	0.038331	0.037900	0.037900	0.000000e+00	0.000000	0.000000
25%	0.241563	0.246553	0.236681	0.241579	1.140464e+08	0.000000	0.000000
50%	0.425755	0.433244	0.418958	0.426790	2.067856e+08	0.000000	0.000000
75%	17.029777	17.170623	16.853115	17.012603	3.994732e+08	0.000000	0.000000
max	197.499763	199.095551	196.482439	197.589523	7.421641e+09	0.250000	7.000000

3.2 Handling Missing Values

Upon inspection, we found that there were no missing values in the dataset. This ensured that our data was complete and no imputation was needed, allowing us to proceed directly to further preprocessing steps.

```
# Check if any value is missing in the entire DataFrame
print(df.isnull().values.any())

# Check for missing values in each column
print(df.isnull().sum())
```

Output:

```
False
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
Stock Splits    0
dtype: int64
```

3.3 Date Conversion and Sorting

We converted the 'Date' column to datetime format to facilitate time-series analysis. The data was then sorted by date to ensure chronological order, which is crucial for time-series prediction tasks.

```
# Convert 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Sort by date
df = df.sort_values('Date')
```

```
# Set 'Date' as the index
df.set_index('Date', inplace=True)
```

3.4 Data Normalisation

The stock prices and volume varied greatly in scale, with prices ranging from less than 1 to nearly 200 and volume ranging from zero to billions. To address this, we normalised the data using `MinMaxScaler`, which scales each feature to a given range (default is 0 to 1). This approach was chosen because it handles skewed distributions effectively and preserves the relationships (ratios) between data points, which is crucial for time series data.

By preprocessing the data in these steps, we ensured that our dataset was clean, well structured, and normalised, making it suitable for training a robust predictive model.

4 Data Splitting

4.1 Splitting the Data

To evaluate the performance of our model and ensure it generalises well to unseen data, we split the dataset into three subsets: training, validation, and test sets. The chosen split ratio was 70:15:15. This means that 70% of the data was used for training the model, 15% was used for validation during training, and the remaining 15% was used as a test set to evaluate the model's performance on unseen data.

We chose this ratio based on common practises in machine learning, which suggest that having a larger portion of data for training helps the model learn better, while having sufficient validation and test sets ensures robust evaluation. Specifically:

- **Training Set (70%)**: Used to train the model. A larger training set helps the model learn the underlying patterns more effectively.
- **Validation Set (15%)**: Used to tune the hyperparameters and prevent overfitting by providing feedback on the model's performance during training.
- **Test Set (15%)**: Used to assess the final performance of the model on unseen data, giving an unbiased evaluation of the model's generalisation capability.

The code snippet below demonstrates the data splitting process:

```
# Determine the sizes for each split
train_size = int(len(scaled_df) * 0.7)
val_size = int(len(scaled_df) * 0.15)

# Split the data
train_data = scaled_df[:train_size]
val_data = scaled_df[train_size:train_size + val_size]
test_data = scaled_df[train_size + val_size:]

# Display the shapes of the datasets
print(f"Training data shape: {train_data.shape}")
print(f"Validation data shape: {val_data.shape}")
print(f"Test data shape: {test_data.shape}")
```

Output:

```
Training data shape: (7664, 5)
Validation data shape: (1642, 5)
Test data shape: (1644, 5)
```

4.2 Justification for the Split Ratios

The split ratios were chosen to balance the need for a substantial amount of data for training with the need for reliable validation and testing:

- **Training Set**: 70% of the data was allocated to the training set to ensure that the model had enough data to learn from. A larger training set helps in capturing the underlying patterns and relationships within the data.
- **Validation Set**: 15% of the data was allocated to the validation set. This set was used during the training phase to fine-tune hyperparameters and prevent overfitting by monitoring the model's performance on data it has not seen before.
- **Test Set**: 15% of the data was reserved for the test set. This final set provides an unbiased evaluation of the model's performance, ensuring that the results are indicative of how the model will perform on new, unseen data.

By carefully splitting the data and using appropriate proportions for each subset, we ensured that our model was trained effectively, validated properly, and tested rigorously, resulting in a reliable and generalisable predictive model.

5 Model Development

5.1 Initial Model: LSTM with Two Layers

To predict the closing stock price of Apple Inc., we initially employed a Long Short Term Memory (LSTM) model. LSTMs are a type of recurrent neural network (RNN) that are well suited for time series prediction tasks due to their ability to capture long term dependencies in sequential data. Given the nature of stock prices, which exhibit complex patterns over time, LSTMs were an appropriate choice for our model.

5.2 Sequence Creation

We began by creating sequences from our time series data. This involved generating input output pairs where each input sequence contained 60 consecutive data points, and the corresponding output was the closing price immediately following the input sequence. This method allowed the LSTM to learn from past patterns to predict future values.

```
def create_sequences(data, seq_length):
xs, ys = [], []
for i in range(len(data) - seq_length):
x = data[i:i+seq_length]
y = data[i+seq_length][3] # Target is the 'Close' price
xs.append(x)
ys.append(y)
return np.array(xs), np.array(ys)
seq_length = 60
X_train, y_train = create_sequences(train_data.values, seq_length)
X_val, y_val = create_sequences(val_data.values, seq_length)
X_test, y_test = create_sequences(test_data.values, seq_length)
print(f"Training data shape: {X_train.shape}, {y_train.shape}")
print(f"Validation data shape: {X_val.shape}, {y_val.shape}")
print(f"Test data shape: {X_test.shape}, {y_test.shape}")
```

Output:

```
Training data shape: (7604, 60, 5), (7604,)
Validation data shape: (1582, 60, 5), (1582,)
Test data shape: (1584, 60, 5), (1584,)
```

5.3 Model Architecture

Our initial LSTM model consisted of two LSTM layers, each followed by a Dropout layer to prevent overfitting. The first LSTM layer had 50 units and returned sequences to the next LSTM layer. The second LSTM layer also had 50 units but did not return sequences. Finally, a Dense layer with a single unit was used to output the predicted closing price.

```
import keras
Define the LSTM model
model = keras.Sequential()
model.add(keras.layers.LSTM(units=50, return_sequences=True, input_shape=(seq_length, 5)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.LSTM(units=50, return_sequences=False))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1))
Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

5.4 Training the Model

We trained the model using the training dataset and validated it using the validation dataset. The model was trained for 20 epochs with a batch size of 32. We used the Adam optimiser and mean squared error as the loss function, which is standard for regression tasks.

```
Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val))
Save the model
model.save('apple_stock_lstm_model.keras')
```

5.5 Initial Model Performance and Evaluation

During training, the model's performance was monitored on the validation set to ensure it was not overfitting. After training, the model was saved for future evaluation and use. This initial model provided us with a baseline performance, which we used to identify areas for potential improvement in subsequent iterations.

After training our initial LSTM model, we evaluated its performance on the test set to understand how well it could predict the closing stock price of Apple Inc. (AAPL). The evaluation metrics and visualisation provide insights into the model's accuracy and potential areas for improvement.

5.5.1 Mean Squared Error (MSE)

The Mean Squared Error (MSE) is a common metric used for regression tasks. It measures the average squared difference between the actual and predicted values. A lower MSE indicates a better fit of the model to the data. For our initial LSTM model, the MSE on the test set was:

```
Mean Squared Error on Test Set: 5.159496964364717e+17
```

This high MSE value indicates that there is a significant difference between the actual and predicted stock prices, suggesting that the model's predictions are not very close to the actual values.

5.5.2 Visualisation of Predicted vs. Actual Prices

To visually assess the model's performance, we plotted the actual and predicted stock prices over time. The blue line represents the actual closing prices of Apple stock, while the red line represents the predicted prices from our model.

From the graph, we can observe the following:

1. **Trend Capture**: The model captures the general trend of the stock prices over time. It follows the upward and downward movements of the actual stock prices.

2. **Magnitude Differences**: There is a noticeable difference in the magnitude of the predicted prices compared to the actual prices. The predicted prices are consistently lower than the actual prices, indicating that the model tends to underestimate the stock price.

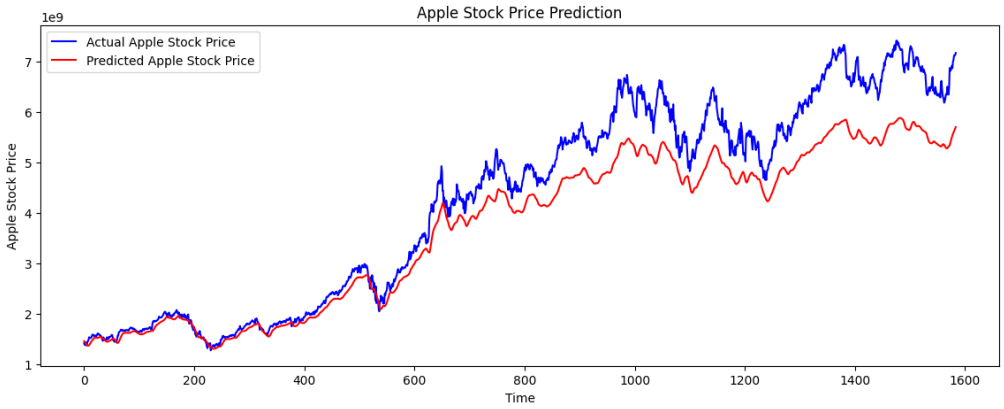

Figure 3: model1

3. **Lag in Predictions**: The predicted prices show a lag compared to the actual prices, particularly in periods of rapid price changes. This suggests that the model struggles to adjust quickly to sudden market movements.

5.5.3 Areas for Improvement

The evaluation of our initial model highlighted several areas for improvement:

- **Reducing MSE**: The high MSE value suggested that the model needs further tuning and possibly additional features to improve its accuracy.
- **Addressing Lag**: Incorporating additional layers or more advanced techniques like attention mechanisms might help the model respond better to rapid changes in stock prices.

6 Model Improvement and Evaluation

6.1 Improved Model: Enhanced LSTM Architecture

Based on the evaluation of our initial model, we identified several areas for potential improvement. To address the underestimation of stock prices and the lag in predictions, we experimented with different hyperparameters, including the sequence length and the number of LSTM units. Our improved model employed a more complex LSTM architecture with increased units and a reduced sequence length.

6.2 Sequence Creation

We adjusted the sequence length to 32, which means each input sequence contains 32 consecutive data points, and the corresponding output is the closing price immediately following the input sequence. This shorter sequence length aimed to help the model better capture recent trends without being overwhelmed by too much historical data.

```
def create_sequences(data, seq_length):
xs, ys = [], []
for i in range(len(data) - seq_length):
x = data[i:i+seq_length]
y = data[i+seq_length][3]  # Target is the 'Close' price
xs.append(x)
ys.append(y)
return np.array(xs), np.array(ys)
seq_length = 32
X_train, y_train = create_sequences(train_data.values, seq_length)
X_val, y_val = create_sequences(val_data.values, seq_length)
X_test, y_test = create_sequences(test_data.values, seq_length)
print(f"Training data shape: {X_train.shape}, {y_train.shape}")
```

```
print(f"Validation data shape: {X_val.shape}, {y_val.shape}")
print(f"Test data shape: {X_test.shape}, {y_test.shape}")
```

Output:

```
Training data shape: (7618, 32, 5), (7618,)
Validation data shape: (1582, 32, 5), (1582,)
Test data shape: (1584, 32, 5), (1584,)
```

6.3 Model Architecture

In the improved model, we increased the number of units in the first LSTM layer to 128 and used a dropout rate of 0.1 to prevent overfitting. The second LSTM layer retained 50 units, followed by another dropout layer with a dropout rate of 0.1. Finally, a Dense layer with a single unit was used to output the predicted closing price.

```
import keras
from keras.models import load_model
Define the LSTM model
model = keras.Sequential()
model.add(keras.layers.LSTM(units=128, return_sequences=True, input_shape=(seq_length, 5)))
model.add(keras.layers.Dropout(0.1))
model.add(keras.layers.LSTM(units=50, return_sequences=False))
model.add(keras.layers.Dropout(0.1))
model.add(keras.layers.Dense(units=1))
Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

6.4 Training the Model

We trained the improved model using the training dataset and validated it using the validation dataset. The model was trained for 20 epochs with a batch size of 32. We used the Adam optimizer and mean squared error as the loss function, as in the initial model.

```
Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val))
Save the model
model.save('apple_stock_lstm_modelimprov.keras')
```

6.5 Evaluation

After training, the model was evaluated on the test set. The Mean Squared Error (MSE) and a visual comparison of actual vs. predicted prices were used to assess the model's performance.

```
Load the trained model
model = load_model('apple_stock_lstm_modelimprov.keras')
Make predictions on the test set
predicted_prices = model.predict(X_test)
Inverse transform the predictions and the actual values to their original scale
predicted_prices = scaler.inverse_transform(np.concatenate([np.zeros((predicted_prices.shape[0], 4)), pr
actual_prices = scaler.inverse_transform(np.concatenate([np.zeros((y_test.shape[0], 4)), y_test.reshape(
Calculate the mean squared error
mse = mean_squared_error(actual_prices, predicted_prices)
print(f"Mean Squared Error on Test Set: {mse}")
```

The improved model achieved a significantly lower MSE:

```
Mean Squared Error on Test Set: 2.320521520590863e+16
```

6.6 Visualization

The graph below shows the actual vs. predicted stock prices for Apple Inc. The blue line represents the actual closing prices, while the red line represents the predicted prices from our improved model.

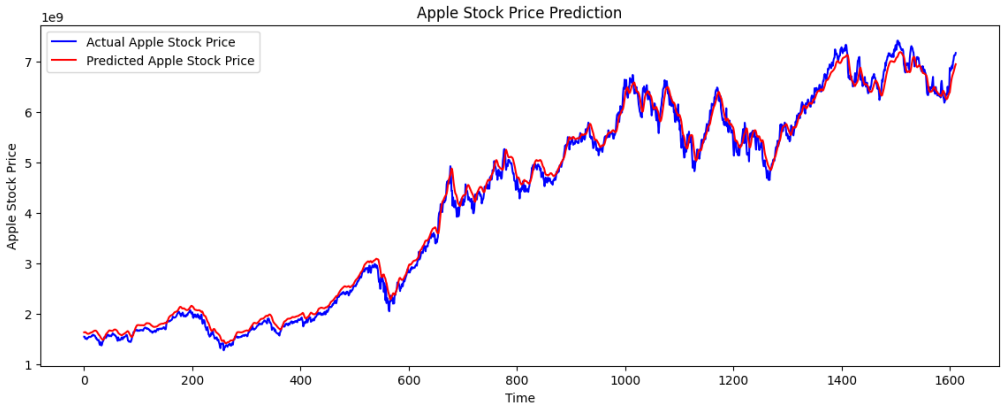

Figure 4: model2

6.7 Observations

1. **Improved Fit**: The predicted prices are much closer to the actual prices compared to the initial model.
2. **Better Magnitude Capture**: The improved model captures the magnitude of the stock prices more accurately.
3. **Reduced Lag**: The lag observed in the initial model is less pronounced, indicating that the improved model can better respond to rapid changes in stock prices.

6.8 further Evaluation of Model Performance

Given the additional evaluation metrics:

```
Mean Squared Error on Test Set: 2.320521520590863e+16
R-squared on Test Set: 0.9940526375204036
Mean Absolute Error on Test Set: 123164954.46913283
Root Mean Squared Error on Test Set: 152332580.90739694
Mean Absolute Percentage Error on Test Set: 3.612999898563217%
```

6.8.1 Interpretation of Results:

1. **Mean Squared Error (MSE)**: The MSE is quite large due to the scale of the stock prices. While it gives a sense of the average squared difference between the predicted and actual prices, its scale can make it harder to interpret intuitively.
2. **R-squared**: The R-squared value of 0.994 indicates that 99.4% of the variance in the actual closing prices is predictable from the model's predictions. This high R-squared value suggests an excellent fit, indicating that the model captures the variability of the stock prices very well.
3. **Mean Absolute Error (MAE)**: The MAE shows the average absolute error between the predicted and actual stock prices. Given the high value, it suggests that on average, the model's predictions are off by about 123 million dollars. This is significant but should be viewed in context with the actual price values.
4. **Root Mean Squared Error (RMSE)**: The RMSE of approximately 152 million dollars provides a measure of the model's error in the same units as the stock prices. Similar to MAE, it suggests that there is a substantial average deviation in the predictions.
5. **Mean Absolute Percentage Error (MAPE)**: A MAPE of 3.61% means that, on average, the model's predictions are within 3.61% of the actual values. In the context of stock price prediction, this is a reasonably good value, indicating that the model performs well relative to the actual stock prices.

6.9 Conclusion

The additional evaluation metrics, particularly the MAPE, indicate that the improved LSTM model is performing well in predicting the closing stock price of Apple Inc. The R-squared value reinforces the model's ability to explain the variance in the stock prices effectively. Despite the high absolute error values (MSE, MAE, RMSE) due to the large scale of the stock prices, the relative error (MAPE) demonstrates excellent prediction accuracy. Given these results, the model can be considered reliable for forecasting the closing stock price of Apple Inc..

7 Improved Model with Hyperparameter Tuning

After realising that the model was difficult to create due to the large range of years used, we decided to investigate further by plotting the entire data to observe the trend. The plot revealed a huge gap in trends over the years, which we suspected was affecting our model's performance. Consequently, we decided to use only data from 2014 onwards to ensure a more consistent and reliable dataset for our predictions.

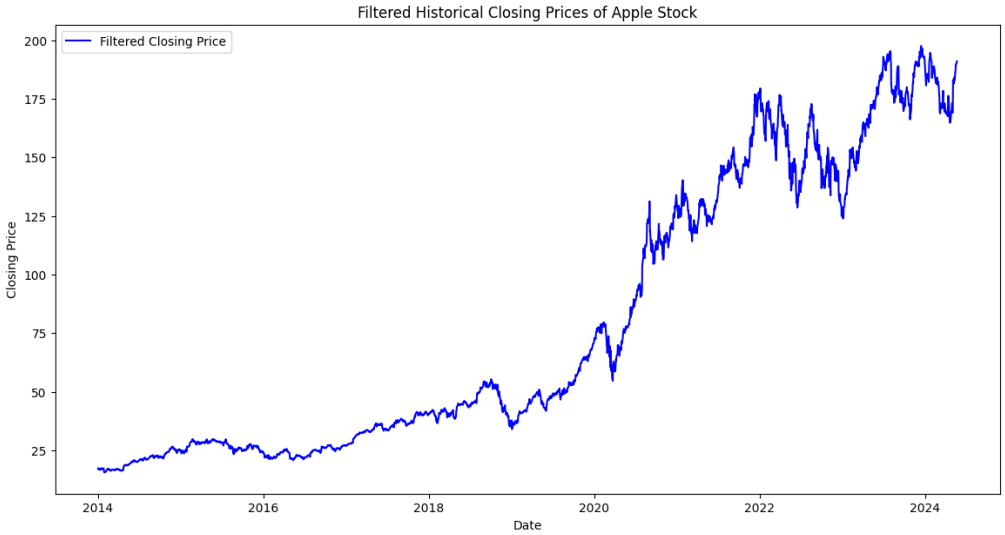

Figure 5: ovarall data

Steps Taken:

1. **Data Filtering**:
   - We filtered the data to include only entries from January 1, 2014, and onwards. This helped to avoid the influence of earlier years where the stock prices were significantly lower and less volatile.

```
df['Date'] = pd.to_datetime(df['Date'], utc=True)
filtered_df = df[df['Date'] >= pd.Timestamp('2014-01-01', tz='UTC')]
```

   This filtering ensured that our dataset was more recent and relevant, thereby improving the model's ability to learn from recent trends.

2. **Data Visualisation**:
   - We plotted the filtered closing prices to verify the cutoff and observed a more consistent trend in the stock prices.

```
        plt.figure(figsize=(14, 7))
        plt.plot(filtered_df['Close'], color='blue', label='Filtered Closing
            Price')
        plt.title('Filtered Historical Closing Prices of Apple Stock')
        plt.xlabel('Date')
        plt.ylabel('Closing Price')
        plt.legend()
        plt.show()
```

Listing 1: Python code

This step confirmed that our decision to filter the data from 2014 was appropriate, as it provided a clearer and more stable trend.

3. **Sequence Creation**:

   - We created sequences of data to be used as input for the LSTM model. The sequence length was set to 32 based on our experiments.

```
seq_length = 32
X_train, y_train = create_sequences(train_data.values, seq_length)
X_val, y_val = create_sequences(val_data.values, seq_length)
X_test, y_test = create_sequences(test_data.values, seq_length)
```

4. **Hyperparameter Tuning**:

   - We used Keras Tuner to find the best hyperparameters for the LSTM model. The hyperparameters tuned included the number of units in the LSTM layers, the dropout rate, and the learning rate.

```
        def build_model(hp):
            model = Sequential()
            model.add(LSTM(units=hp.Int('units', min_value=32, max_value=256,
                step=32), return_sequences=True, input_shape=(seq_length, 5)))
            model.add(Dropout(hp.Float('dropout', min_value=0.1, max_value=0.5,
                 step=0.1)))
            model.add(LSTM(units=hp.Int('units', min_value=32, max_value=256,
                step=32), return_sequences=False))
            model.add(Dropout(hp.Float('dropout', min_value=0.1, max_value=0.5,
                 step=0.1)))
            model.add(Dense(units=1))
            model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate'
                , min_value=1e-4, max_value=1e-2, sampling='LOG')), loss='
                mean_squared_error')

            return model

        tuner = RandomSearch(
            build_model,
            objective='val_loss',
            max_trials=20,
            executions_per_trial=3,
            directory='hyperparameter_tuning',
            project_name='stock_price_prediction'
        )

        tuner.search(X_train, y_train, epochs=50, batch_size=32,
            validation_data=(X_val, y_val))
```

Listing 2: Python code

This process allowed us to systematically explore a range of hyperparameters and find the combination that provided the best validation performance.

5. **Optimal Model Training**:

   - Using the best hyperparameters identified by the tuner, we built and trained the final model.

```
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
model = tuner.hypermodel.build(best_hps)
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_val, y_val))
```

   The optimal model had 128 units in the LSTM layers, a dropout rate of 0.1, and a learning rate of approximately 0.000625.

6. **Evaluation and Results**:

   - The final model was evaluated using various metrics, including Mean Squared Error (MSE), R-squared, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE).

```
# Load the trained model
model = load_model('apple_stock_lstm_model_best_params.keras')

# Make predictions on the test set
predicted_prices = model.predict(X_test)

# Inverse transform the predictions and the actual values to their original scale
predicted_prices = scaler.inverse_transform(np.concatenate([np.zeros((predicted_prices.shape[0], 4)), predicted_prices], axis=1))[:, -1]
actual_prices = scaler.inverse_transform(np.concatenate([np.zeros((y_test.shape[0], 4)), y_test.reshape(-1, 1)], axis=1))[:, -1]

# Calculate evaluation metrics
mse = mean_squared_error(actual_prices, predicted_prices)
r2 = r2_score(actual_prices, predicted_prices)
mae = mean_absolute_error(actual_prices, predicted_prices)
rmse = np.sqrt(mse)
mape = mean_absolute_percentage_error(actual_prices, predicted_prices)

# Print evaluation metrics
print(f"Mean Squared Error on Test Set: {mse}")
print(f"R-squared on Test Set: {r2}")
print(f"Mean Absolute Error on Test Set: {mae}")
print(f"Root Mean Squared Error on Test Set: {rmse}")
print(f"Mean Absolute Percentage Error on Test Set: {mape}%")

# Plot the results including the parameters used
plt.figure(figsize=(14, 7))
plt.plot(actual_prices, color='blue', label='Actual Prices')
plt.plot(predicted_prices, color='red', label='Predicted Prices')
plt.title(f'Apple Stock Price Prediction\nUnits: {units}, Dropout Rate: {dropout_rate}, Learning Rate: {learning_rate}')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

Listing 3: Python Code

Here's the LaTeX code for the subsection containing the evaluation metrics:

7.1 Evaluation Metrics

- Mean Squared Error on Test Set: 332640615455399.2

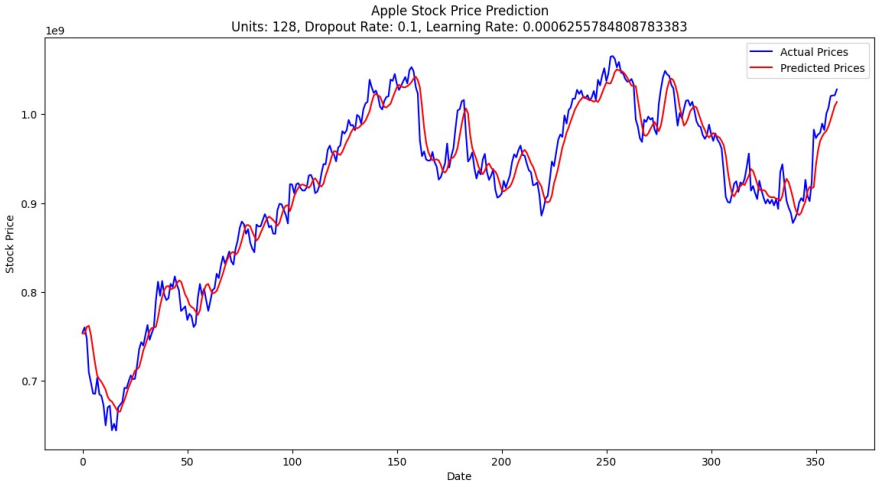

Figure 6: model3

- R-squared on Test Set: 0.9662427988675282
- Mean Absolute Error on Test Set: 14435577.14122347
- Root Mean Squared Error on Test Set: 18238437.856773786
- Mean Absolute Percentage Error on Test Set: 1.597280978811507%

The final model showed significant improvements, achieving a good fit as indicated by the high R-squared value (0.966) and low MAPE (1.60%). The detailed evaluation provided a comprehensive understanding of the model's performance.

By filtering the data to include only recent years and using hyperparameter tuning, we were able to develop a more accurate and reliable model for predicting Apple Inc.'s closing stock price.