Implementation of a Long Short-Term Memory (LSTM) network for stock price prediction

1. Importing necessary libraries

We begin by importing the libraries required for our analysis:

- `numpy`: for numerical operations
- `pandas`: for handling and manipulating the data
- `matplotlib`: for plotting and visualising data
- `tensorflow` for building and training the neural network
- `sklearn`: for data preprocessing and model evaluation
- `yfinance`: for downloading historical stock price data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import yfinance as yf
from sklearn.metrics import mean_squared_error
```

2. Downloading stock data

We use the Yahoo Finance API via the `yfinance` library to download historical stock data. Here, we fetch data for Apple Inc. (`AAPL`).

```
stock_data = yf.download('AAPL', start='2010-01-01', end='2023-01-01')

# Displaying the first few rows of the dataset
stock_data.head()
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  1 of 1 completed
```

```
Price       Close     High      Low       Open      Volume
Ticker      AAPL      AAPL      AAPL      AAPL      AAPL
      Date
2010-01-04  6.440331  6.455077  6.391278  6.422877  493729600
2010-01-05  6.451466  6.487879  6.417459  6.458086  601904800
2010-01-06  6.348845  6.477044  6.342224  6.451464  552160000
2010-01-07  6.337108  6.379842  6.291065  6.372318  477131200
2010-01-08  6.379241  6.379843  6.291368  6.328683  447610800
```

3. Preprocessing the data

The `MinMaxScaler` is used to scale the data into the range [0, 1], which improves training performance. We also split the data into training and test sets (80% for training and 20% for testing). Next, the `create_dataset()` function creates sequences of data where each sequence consists of the previous `time_step` stock prices to predict the next one; this is important because LSTMs are good at learning patterns from previous data points.

```
data = stock_data['Close'].values
data = data.reshape(-1, 1)

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

training_data_len = int(np.ceil(0.8 * len(scaled_data)))
train_data = scaled_data[:training_data_len]
test_data = scaled_data[training_data_len:]

def create_dataset(data, time_step=60):
  X, Y = [], []
  for i in range(time_step, len(data)):
    X.append(data[i-time_step:i, 0])
    Y.append(data[i, 0])
  return np.array(X), np.array(Y)

X_train, Y_train = create_dataset(train_data)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
```

```
X_test, Y_test = create_dataset(test_data)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

4. Building the LSTM model

Now, we build the LSTM model using Keras.

- `LSTM` Layers:
  - The first `LSTM` layer has 50 units and returns sequences because the next `LSTM` layer will expect sequence data.
  - The second `LSTM` layer also has 50 units, but it does not return sequences since it is the last `LSTM` layer.
- `Dense` Layer: A fully connected layer with one unit that predicts the stock price.

The model is compiled using the Adam optimiser and mean squared error (MSE) loss function which is commonly used for regression tasks like stock prediction.

```
model = Sequential()

model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=1))

model.compile(optimizer='adam', loss='mean_squared_error')
```

Show hidden output

5. Training the model

The model is trained for 10 epochs with a batch size of 32. We use the test set as the validation data to monitor the model's performance on unseen data.

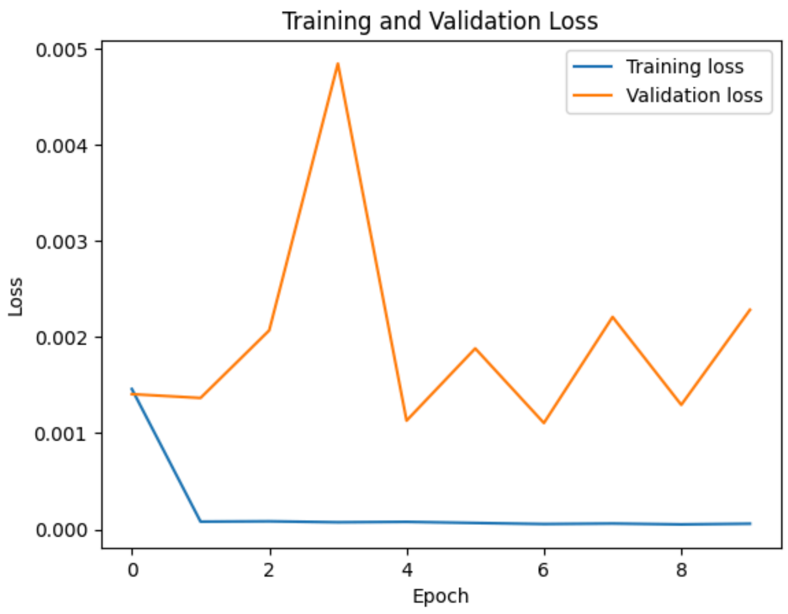
```
history = model.fit(X_train, Y_train, epochs=10, batch_size=32, validation_data=(X_test, Y_test))
```

```
Epoch 1/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 10s 75ms/step - loss: 0.0044 - val_loss: 0.0014
Epoch 2/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 9s 66ms/step - loss: 7.8293e-05 - val_loss: 0.0014
Epoch 3/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 6s 75ms/step - loss: 7.7199e-05 - val_loss: 0.0021
Epoch 4/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 60ms/step - loss: 7.0674e-05 - val_loss: 0.0048
Epoch 5/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 6s 74ms/step - loss: 8.4784e-05 - val_loss: 0.0011
Epoch 6/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 9s 60ms/step - loss: 6.4686e-05 - val_loss: 0.0019
Epoch 7/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 6s 75ms/step - loss: 5.6546e-05 - val_loss: 0.0011
Epoch 8/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 5s 60ms/step - loss: 5.7275e-05 - val_loss: 0.0022
Epoch 9/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 6s 75ms/step - loss: 5.2837e-05 - val_loss: 0.0013
Epoch 10/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 9s 65ms/step - loss: 6.0075e-05 - val_loss: 0.0023
```

6. Visualising training loss

We plot the training loss and validation loss to understand how well the model is training. The plot shows how the model's loss changes over epochs for both training and validation data. A decrease in loss indicates that the model is learning and improving.

```
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



7. Making predictions and visualising results

After training, we use the model to predict stock prices. We can plot the predicted stock prices against the actual stock prices.

Here, we predict the stock prices using the test data and then inverse transform the results back to the original scale. The plot shows the actual stock prices (blue) and the predicted stock prices (red). This gives us an idea of how well the model is predicting future prices.

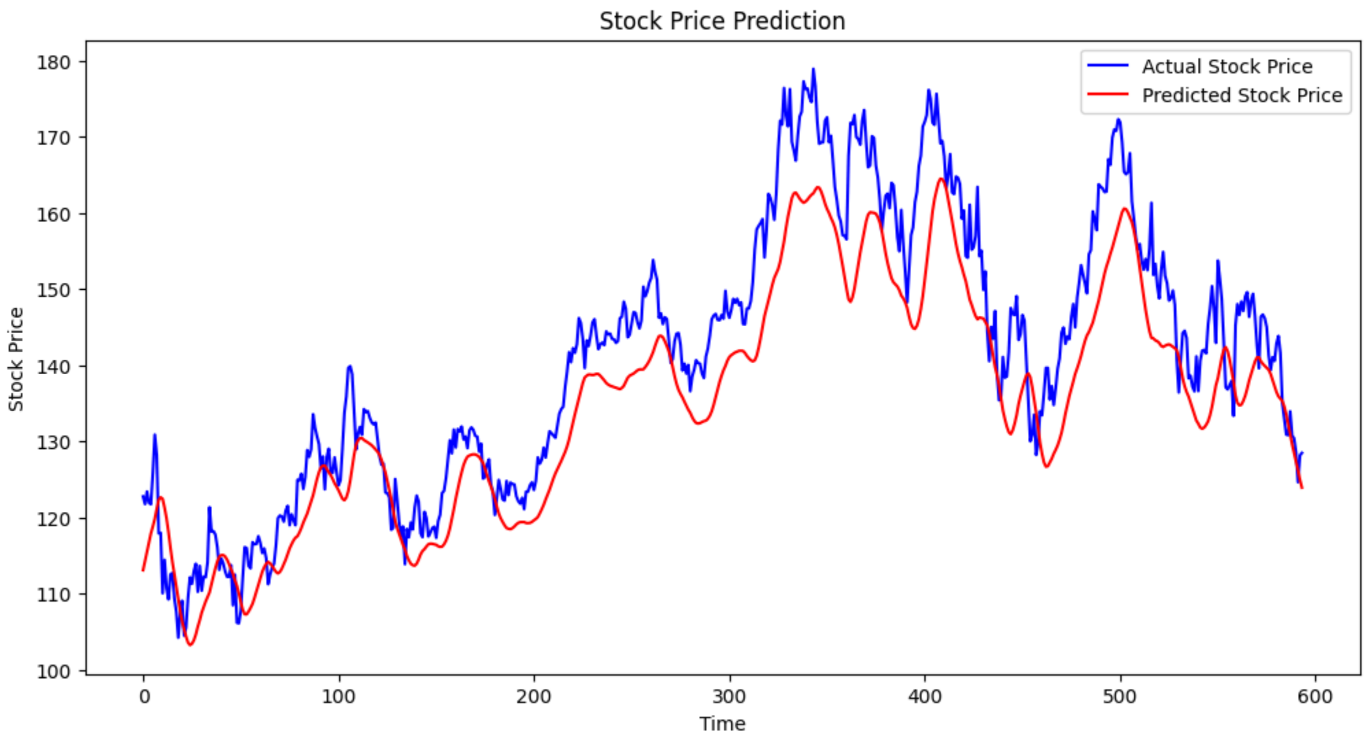
```
predicted_prices = model.predict(X_test)

predicted_prices = scaler.inverse_transform(predicted_prices)

Y_test_rescaled = scaler.inverse_transform(Y_test.reshape(-1, 1))

plt.figure(figsize=(12, 6))
plt.plot(Y_test_rescaled, color='blue', label='Actual Stock Price')
plt.plot(predicted_prices, color='red', label='Predicted Stock Price')
plt.title('Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step
```



8. Evaluating the model

Finally, we evaluate the model by calculating the performance metric Mean Squared Error (MSE). A lower MSE indicates better performance.

```
mse = mean_squared_error(Y_test_rescaled, predicted_prices)
print(f'Mean squared error: {mse}')
```

```
Mean squared error: 68.44536629128947
```