**CODE**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import yfinance as yf

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, LSTM

# Step 1: Download stock data

stock = 'AAPL'  # You can change to 'TSLA', 'GOOGL', etc.

df = yf.download(stock, start='2015-01-01', end='2023-01-01')

# Check the first few rows of the data to ensure 'Close' is present

print("Data Overview:")

print(df[['Close']].head())  # Access only the 'Close' column to display

# Ensure 'Close' column exists and filter it

if 'Close' in df.columns:

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

else:

    print("Error: 'Close' column not found in the data.")

    exit()

# Step 2: Scale the data

scaler = MinMaxScaler(feature\_range=(0, 1))

# Scale the 'Close' prices

scaled\_data = scaler.fit\_transform(data)

# Step 3: Prepare training data

train\_len = int(len(scaled\_data) \* 0.8)

train\_data = scaled\_data[:train\_len]

X\_train, y\_train = [], []

for i in range(60, len(train\_data)):

    X\_train.append(train\_data[i - 60:i, 0])

    y\_train.append(train\_data[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

# Step 4: Build the model

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(LSTM(50))

model.add(Dense(25))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

model.fit(X\_train, y\_train, batch\_size=1, epochs=1)

# Step 5: Prepare testing data

test\_data = scaled\_data[train\_len - 60:]

X\_test, y\_test = [], data[train\_len:].flatten()

for i in range(60, len(test\_data)):

    X\_test.append(test\_data[i - 60:i, 0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Step 6: Predict and unscale

predictions = model.predict(X\_test)

predictions = scaler.inverse\_transform(predictions)

# Step 7: Compare actual vs predicted prices

compare\_df = pd.DataFrame({

    'Actual Price': y\_test,

    'Predicted Price': predictions.flatten()

})

print("\n🔍 Top 10: Actual vs Predicted Prices")

print(compare\_df.head(10))

# # Step 8: Plot the results

# train = df['Close'][:train\_len]

# valid = df['Close'][train\_len:]

# # Convert 'valid' to DataFrame with proper column name

# valid = pd.DataFrame(valid, columns=['Close'])

# # Add predictions to the valid DataFrame

# valid['Predictions'] = predictions

# # Plotting the training data, actual prices, and predicted prices

# plt.figure(figsize=(16, 6))

# plt.title('Stock Price Prediction Using LSTM')

# plt.xlabel('Date')

# plt.ylabel('Stock Price (USD)')

# # Plot the actual prices from the training data and validation data

# plt.plot(train.index, train, label='Training Data')

# plt.plot(valid.index, valid['Close'], label='Actual Price', color='blue')  # Actual prices

# plt.plot(valid.index, valid['Predictions'], label='Predicted Price', color='red')  # Predictions

# plt.legend()

# plt.show()

# Step 8: Plot the results (aligned actual & predicted graph)

# Create a new DataFrame with the same index as original

full\_data = pd.DataFrame(index=df.index)

full\_data['Train'] = df['Close'][:train\_len]                # Actual training prices

full\_data['Actual'] = df['Close'][train\_len:]               # Actual validation prices

full\_data['Predicted'] = np.nan                             # Placeholder

# Insert predicted prices aligned with validation period

full\_data.loc[df.index[train\_len:], 'Predicted'] = predictions.flatten()

# Plot the results

plt.figure(figsize=(16, 6))

plt.title('Stock Price Prediction Using LSTM')

plt.xlabel('Date')

plt.ylabel('Stock Price (USD)')

# Plot each line

# plt.plot(full\_data['Train'], label='Training Data', color='black')

plt.plot(full\_data['Actual'], label='Actual Price', color='blue')

plt.plot(full\_data['Predicted'], label='Predicted Price', color='red')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

**OUTPUT**

Actual vs Predicted Prices

Actual Price Predicted Price

0 122.762154 124.695854

1 122.105614 124.682663

2 121.782234 124.553505

3 122.546593 124.343704

4 121.057121 124.167343

5 123.359901 123.899826

6 123.369705 123.794754

7 124.192825 123.783096

8 124.574959 123.893593

9 123.575485 124.093567

