**STOCK MARKET ANALYSIS AND PREDICTION**

**Import the libraries:**

|  |
| --- |
| **# Data Manipulation and Analysis**  import pandas as pd  import numpy as np  **# Data Visualization**  import matplotlib.pyplot as plt  **# Statistical Testing**  from statsmodels.tsa.stattools import adfuller # For stationarity testing  **# Time Series Forecasting**  from statsmodels.tsa.arima.model import ARIMA # For ARIMA modeling  **# Evaluation Metrics**  from sklearn.metrics import mean\_squared\_error # For evaluating model performance  **# Deep Learning Framework**  import tensorflow as tf  from sklearn.preprocessing import MinMaxScaler # For feature scaling  from tensorflow.keras.models import Sequential # To define the neural network model  from tensorflow.keras.layers import Dense, LSTM # For deep learning layers |

**Load and Preprocess Data:**

|  |
| --- |
| **# Load the stock market data**  data = pd.read\_csv('Tesla.csv') # Replace with your actual dataset  data['Date'] = pd.to\_datetime(data['Date'])  data.set\_index('Date', inplace=True)  **# Explore the dataset**  print(data.head())  print(data.info())  print(data.describe())  **# Plot the closing price**  plt.figure(figsize=(10, 6))  plt.plot(data['Close'])  plt.title('Stock Closing Price Over Time')  plt.xlabel('Date')  plt.ylabel('Price')  plt.show()  **# Check for missing values**  print(data.isnull().sum())  data.fillna(method='ffill', inplace=True) # Forward fill missing values |

**Stationarity Testing:**

|  |
| --- |
| **# ADF Test for stationarity**  result = adfuller(data['Close'])  print(f'ADF Statistic: {result[0]}')  print(f'p-value: {result[1]}')  if result[1] <= 0.05:  print("The data is stationary.")  else:  print("The data is not stationary.") |

**ARIMA Modeling:**

|  |
| --- |
| **# Ensure the index is datetime and has a frequency**  data.index = pd.to\_datetime(data.index) # Convert index to datetime if not already  data = data.asfreq('D') # Set frequency to daily  **# Train ARIMA model**  model = ARIMA(data['Close'], order=(1, 1, 1))  arima\_result = model.fit()  print(arima\_result.summary())  **# Forecasting the next 30 days**  forecast = arima\_result.get\_forecast(steps=30)  forecast\_mean = forecast.predicted\_mean  forecast\_ci = forecast.conf\_int()  **# Plot actual vs forecasted values**  plt.figure(figsize=(12, 6))  plt.plot(data['Close'], label='Actual', color='blue')  plt.plot(forecast\_mean.index, forecast\_mean, label='Forecast', color='red')  plt.fill\_between(forecast\_ci.index, forecast\_ci.iloc[:, 0], forecast\_ci.iloc[:, 1], color='pink', alpha=0.3)  plt.title('ARIMA Model - Stock Price Forecast')  plt.xlabel('Date')  plt.ylabel('Price')  plt.legend()  plt.show() |

**LSTM Neural Network for Prediction:**

**Data Preparation**

|  |
| --- |
| # Scale data to [0,1] for LSTM  scaler = MinMaxScaler(feature\_range=(0, 1))  scaled\_data = scaler.fit\_transform(data['Close'].values.reshape(-1, 1))  # Create training and test datasets  train\_size = int(len(scaled\_data) \* 0.8)  train\_data = scaled\_data[:train\_size]  test\_data = scaled\_data[train\_size:]  # Create sequences for LSTM  def create\_sequences(data, seq\_length):  X, y = [], []  for i in range(len(data) - seq\_length):  X.append(data[i:i + seq\_length])  y.append(data[i + seq\_length])  return np.array(X), np.array(y)  seq\_length = 60 # Use last 60 days to predict the next day  X\_train, y\_train = create\_sequences(train\_data, seq\_length)  X\_test, y\_test = create\_sequences(test\_data, seq\_length) |

**Build and Train the LSTM Model**

|  |
| --- |
| **# Define the improved LSTM model**  model = Sequential([  # First LSTM layer with Dropout  LSTM(units=100, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)),  Dropout(0.2), # 20% dropout to prevent overfitting    # Second LSTM layer with Dropout  LSTM(units=100, return\_sequences=True),  Dropout(0.2),    # Third LSTM layer with Dropout  LSTM(units=100),  Dropout(0.2),    # Output layer  Dense(units=1) # Predict a single value (next day's stock price)  ])  **# Compile the model**  model.compile(optimizer=Adam(learning\_rate=0.001), loss='mean\_squared\_error')  # Callbacks to improve training  callbacks = [  ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=5, min\_lr=1e-5, verbose=1), # Reduce learning rate on plateau  EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True, verbose=1) # Stop early if no improvement  ]  **# Train the model**  history = model.fit(  X\_train, y\_train,  epochs=50, # Increase epochs for better learning  batch\_size=32, # Moderate batch size  validation\_split=0.2, # Use 20% of training data for validation  callbacks=callbacks,  verbose=1  )  **# Predict using the improved model**  predicted\_prices = model.predict(X\_test)  predicted\_prices = scaler.inverse\_transform(predicted\_prices)  **# Plot the predictions**  plt.figure(figsize=(10, 6))  plt.plot(data.index[train\_size + seq\_length:], scaler.inverse\_transform(test\_data[seq\_length:]), label='Actual Price')  plt.plot(data.index[train\_size + seq\_length:], predicted\_prices, label='Predicted Price', color='red')  plt.title('Improved LSTM Stock Price Prediction')  plt.xlabel('Date')  plt.ylabel('Price')  plt.legend()  plt.show() |

**Evaluate the Model:**

|  |
| --- |
| mse = mean\_squared\_error(scaler.inverse\_transform(y\_test.reshape(-1, 1)), predicted\_prices)  rmse = np.sqrt(mse)  print(f'Mean Squared Error: {mse}')  print(f'Root Mean Squared Error: {rmse}') |

**Statistical Summary**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Statistic** | **Open** | **High** | **Low** | **Close** | **Volume** | **Adj. Close** |
| Count | 1692 | 1692 | 1692 | 1692 | 1692e+03 | 1692 |
| Mean | 132.44 | 134.77 | 129.99 | 132.43 | 4270741 | 132.43 |
| Std Dev | 94.31 | 95.69 | 92.86 | 94.31 | 4295971 | 94.31 |
| Min | 16.14 | 16.63 | 14.98 | 15.80 | 118500 | 15.80 |
| 25th Percentile | 30.00 | 30.65 | 29.21 | 29.88 | 1194350 | 29.88 |
| Median | 156.33 | 162.37 | 153.15 | 158.16 | 3180700 | 158.16 |
| 75th Percentile | 220.56 | 224.10 | 217.12 | 220.02 | 5662100 | 220.02 |
| Max | 287.67 | 291.42 | 280.40 | 286.04 | 37163900 | 286.04 |

****



