GROUP B

Mini Project 02- Use the following dataset to analyze ups and downs in the market and predict future stock price returns based on Indian Market data from 2000 to 2020.

Problem Objective: To implement and analyze performance of algorithm.

Theory:

1. **Introduction to problems**

Stock market prediction involves forecasting the future values of stock prices based on historical data and various financial indicators. Accurate predictions can help investors make informed decisions about buying or selling stocks, making this a critical area of research in finance. The Indian stock market has witnessed significant fluctuations between 2000 and 2020, driven by various factors such as global economic conditions, domestic policy changes, inflation rates, and investor sentiment.

This analysis aims to explore the ups and downs in the Indian stock market between 2000 and 2020 and to predict future stock price returns. We will employ various data analysis techniques, including time series analysis, regression models, and machine learning algorithms.

Objectives:

1. Analyzing Trends: Identify periods of growth and downturns in the Indian market, including major bull and bear phases.

2. Market Volatility: Measure the volatility in stock prices and evaluate factors contributing to major market swings.

3. Stock Price Prediction: Use historical stock price data and machine learning algorithms to predict future stock returns.

4. Feature Selection: Explore which features (macroeconomic factors, technical indicators, etc.) are most impactful in predicting stock returns.

Key Concepts:

1. Historical Stock Price Data: This dataset includes daily stock prices from 2000 to 2020. Key indicators include opening price, closing price, volume, high, and low prices.

2. Technical Indicators: We will compute several key technical indicators like moving averages, Bollinger bands, and Relative Strength Index (RSI) to enhance predictions.

3. Machine Learning Models: We will use models such as Linear Regression, Random Forest, and Long Short-Term Memory (LSTM) networks to forecast future stock prices.

2.APPROACH TO SOLVE PROBLEMS

When approaching a problem, especially in the context of data analysis and prediction, it’s essential to have a structured methodology. Here’s a comprehensive approach to solving problems like predicting stock prices using historical market data:

1. Define the Problem

- Identify the Objective: Clearly outline what you want to achieve. For instance, predicting future stock price returns based on past data.

- Scope: Determine the scope of the analysis (e.g., specific stocks, timeframes, market conditions).

2. Data Collection

- Gather Data: Collect historical stock price data, which may include:

- Daily opening, closing, high, and low prices.

- Trading volume.

- Macroeconomic indicators (inflation rate, GDP growth, etc.).

- Source: Use reliable data sources like stock exchanges, financial websites, or APIs (e.g., Yahoo Finance, Alpha Vantage).

3. Data Preprocessing

- Cleaning: Remove any missing or erroneous data points.

- Normalization: Scale the data if necessary, especially when dealing with different ranges of numerical values.

- Feature Engineering: Create additional features that might help in predictions, such as:

- Moving averages (e.g., 50-day, 200-day).

- Technical indicators (e.g., RSI, MACD).

- Lagged variables (previous days’ prices).

4. Exploratory Data Analysis (EDA)

3. ALGORITHM/ PSEUDOCODE OF PROBLEMS

1. Linear Regression

Description: Linear regression is used to model the relationship between a dependent variable (stock prices) and one or more independent variables (features).

Pseudocode:

csharp

Copy code

```
FUNCTION linearRegression(trainingData, features, target):
    // Step 1: Prepare the data
    X = EXTRACT_FEATURES(trainingData, features)  // Input features
    y = EXTRACT_TARGET(trainingData, target)      // Target variable (stock price)

    // Step 2: Add bias term
    X_b = ADD_BIAS_TERM(X)                        // Add a column of ones to X

    // Step 3: Calculate weights (coefficients)
    weights = INVERT(X_b^T * X_b) * (X_b^T * y)   // Normal Equation

    RETURN weights

FUNCTION predict(testData, weights, features):
    X_test = EXTRACT_FEATURES(testData, features)
    X_test_b = ADD_BIAS_TERM(X_test)
    predictions = X_test_b * weights
    RETURN predictions
```

4.COMPLEXITY ANALYZE FOR ALL CASES

Comparison

Algorithm	Time Complexity (Training)	Time Complexity (Prediction)	Space Complexity
Linear Regression	O(n2)O(n^2)O(n2)	O(n)O(n)O(n)	O(n)O(n)O(n)
Moving Average	O(n·w)O(n \cdot w)O(n·w) (or O(n)O(n)O(n))	O(1)O(1)O(1)	O(n)O(n)O(n)
ARIMA	O(n2)O(n^2)O(n2) or O(n3)O(n^3)O(n3)	O(1)O(1)O(1)	O(n)O(n)O(n)
LSTM	O(t·n·m)O(t \cdot n \cdot m)O(t·n·m)	O(t)O(t)O(t)	O(n·m)O(n \cdot m)O(n·m)
Random Forest Regressor	O(m·n·log(n))O(m \cdot n \cdot \log(n))O(m·n·log(n))	O(m·k)O(m \cdot k)O(m·k)	O(m·d)O(m \cdot d)O(m·d)

5. IMPLEMENTATION OF PROJECT WITH OUTPUT

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

#Importing the Libraries
%matplotlib inline
import matplotlib. pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.layers import LSTM, Dense, Dropout
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.dates as mandates
from sklearn.preprocessing import MinMaxScaler
from sklearn import linear_model
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.models import load_model
from keras.layers import LSTM
from keras.utils.vis_utils import plot_model
from keras.optimizers import Adam
Using TensorFlow backend.
In [2]:
nRowsRead = 1000
df = pd.read_csv('/kaggle/input/Data/532746.BO.csv', delimiter=',', nrows = 
nRowsRead)
```

df.dataframeName = '532746.BO.csv'

nRow, nCol = df.shape

In [3]:

df.head()

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2006-06-12	130.000000	130.000000	92.419998	94.339996	94.339996	8097280
1	2006-06-13	93.059998	94.940002	77.120003	78.540001	78.540001	7475940
2	2006-06-14	80.000000	88.580002	74.209999	82.029999	82.029999	6582600
3	2006-06-19	88.019997	94.720001	82.269997	92.680000	92.680000	6085285
4	2006-06-20	89.940002	92.400002	86.800003	89.839996	89.839996	3506755

In [ ]:

In [4]:

df.isnull().sum()

Out[4]:

Date 0

Open 0

High 0

Low 0

Close 0

Adj Close 0

Volume 0

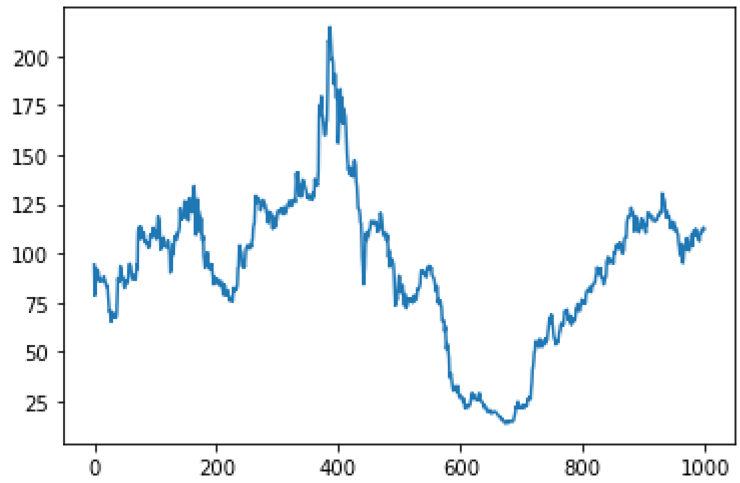
dtype: int64

In [5]:

df["Adj Close"].plot()

Out[5]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fbddfecba10>



In [6]:

#Set Target Variable

output_var = pd.DataFrame(df["Adj Close"])

#Selecting the Features

features = ["Open", 'High', "Low", "Volume"]

In [7]:

#normalising dataset

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

feature_transform = scaler.fit_transform(df[features])

feature_transform= pd.DataFrame(columns=features, data=feature_transform, index=df.index)

feature_transform.head()

Out[7]:

	Open	High	Low	Volume
0	0.589935	0.551098	0.414586	0.644073
1	0.402717	0.383668	0.334313	0.594650
2	0.336526	0.353295	0.319045	0.523593
3	0.377173	0.382617	0.361333	0.484035
4	0.386904	0.371538	0.385100	0.278934

In [8]:

```
from sklearn.model_selection import TimeSeriesSplit
timesplit= TimeSeriesSplit(n_splits=10)
for train_index, test_index in timesplit.split(feature_transform):
        X_train, X_test = feature_transform[:len(train_index)], feature_transform[len(train_index): (len(train_index)+len(test_index))]
        y_train, y_test = output_var[:len(train_index)].values.ravel(), output_var[len(train_index): (len(train_index)+len(test_index))].values.ravel()
```

In [9]:

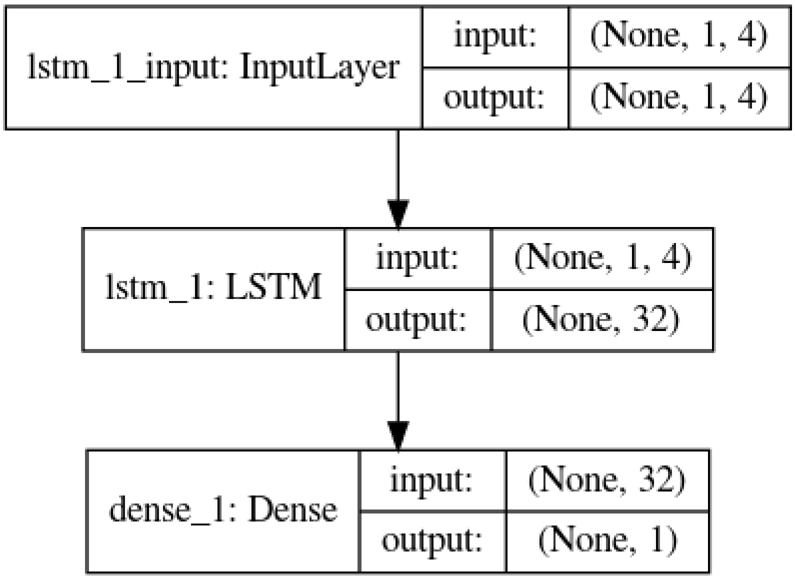
```
#Process the data for LSTM
trainX =np.array(X_train)
testX =np.array(X_test)
X_train = trainX.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = testX.reshape(X_test.shape[0], 1, X_test.shape[1])
```

In [10]:

```
#Building the LSTM Model
lstm = Sequential()
lstm.add(LSTM(32, input_shape=(1, trainX.shape[1]), activation="relu", return_sequences=False))
lstm.add(Dense(1))
lstm.compile(loss="mean_squared_error", optimizer="adam")
```

```
plot_model(lstm, show_shapes=True, show_layer_names=True)
```

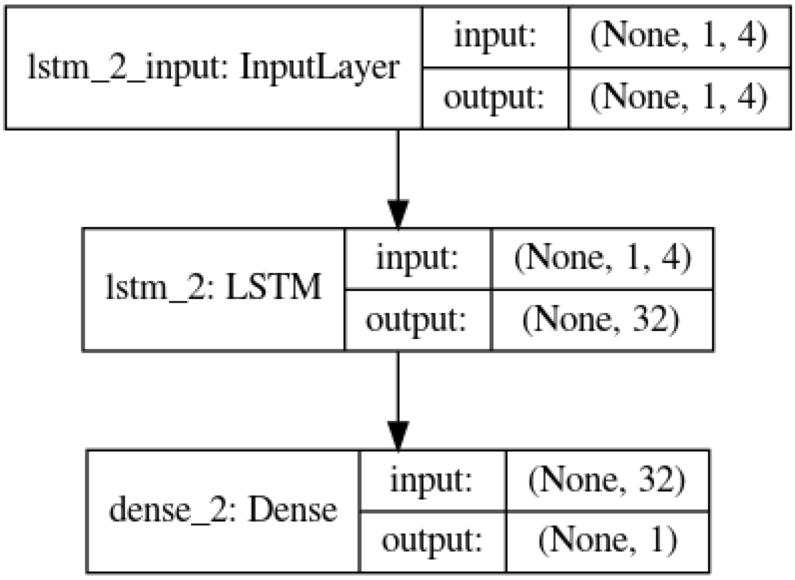
Out[10]:



In [11]:

```
#Building the LSTM Model

lstm = Sequential()

lstm.add(LSTM(32, input_shape=(1, trainX.shape[1]), activation="relu",
return_sequences=False))

lstm.add(Dense(1))

lstm.compile(loss="mean_squared_error", optimizer="adam")

plot_model(lstm, show_shapes=True, show_layer_names=True)
```

Out[11]:



In [12]:

#Model Training

```
history=lstm.fit(X_train, y_train, epochs=100, batch_size=8, verbose=1, shuffle=False)

plt.plot(y_test, label="True Value")

plt.plot(y_pred, label="LSTM Value")

plt.title("Prediction by LSTM")

plt.xlabel("Time Scale")

plt.ylabel("Scaled USD")

plt.legend()

plt.show()
```