STOCK PRICE PREDICTION

PHASE-2

Problem Definition: The problem is to build a predictive model that forecasts stock prices based on historical market data. The goal is to create a tool that assists investors in making well-informed decisions and optimizing their investment strategies. This project involves data collection, data preprocessing, feature engineering, model selection, training, and evaluation.

Design Thinking:

1. Data Collection:

Collect historical stock market data, including features like date, open price, close price, volume, and other relevant indicators.

Dataset Link: https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	03-13-86	0.08854	0.10156	0.08854	0.09722	0.06255	1E+09
3	03-14-86	0.09722	0.10243	0.09722	0.10069	0.06478	3.1E+08
4	03-17-86	0.10069	0.1033	0.10069	0.10243	0.0659	1.3E+08
5	03-18-86	0.10243	0.1033	0.09896	0.09983	0.06422	6.8E+07
6	03-19-86	0.09983	0.10069	0.09722	0.09809	0.06311	4.8E+07
7	03-20-86	0.09809	0.09809	0.09462	0.09549	0.06143	5.8E+07
8	03-21-86	0.09549	0.09722	0.09115	0.09288	0.05976	6E+07
9	03-24-86	0.09288	0.09288	0.08941	0.09028	0.05808	6.5E+07
10	03-25-86	0.09028	0.09201	0.08941	0.09201	0.0592	3.2E+07
11	03-26-86	0.09201	0.09549	0.09115	0.09462	0.06087	2.3E+07
12	03-27-86	0.09462	0.09635	0.09462	0.09635	0.06199	1.7E+07
13	03-31-86	0.09635	0.09635	0.09375	0.09549	0.06143	1.3E+07
14	04-01-86	0.09549	0.09549	0.09462	0.09462	0.06087	1.1E+07
15	04-02-86	0.09462	0.09722	0.09462	0.09549	0.06143	2.7E+07
16	04-03-86	0.09635	0.09896	0.09635	0.09635	0.06199	2.3E+07
17	04-04-86	0.09635	0.09722	0.09635	0.09635	0.06199	2.7E+07
18	04-07-86	0.09635	0.09722	0.09288	0.09462	0.06087	1.7E+07
19	04-08-86	0.09462	0.09722	0.09462	0.09549	0.06143	1E+07
20	04-09-86	0.09549	0.09809	0.09549	0.09722	0.06255	1.2E+07
21	04-10-86	0.09722	0.09896	0.09549	0.09809	0.06311	1.4E+07
22	04-11-86	0.09896	0.10156	0.09896	0.09983	0.06422	1.7E+07
23	04-14-86	0.09983	0.10156	0.09983	0.10069	0.06478	1.2E+07
24	04-15-86	0.10069	0.10069	0.09722	0.10069	0.06478	9302400
25	04-16-86	0.10069	0.10504	0.09983	0.10417	0.06702	3.2E+07
26	04-17-86	0.10417	0.10504	0.10417	0.10504	0.06758	2.2E+07
27	04-18-86	0.10504	0.10504	0.10069	0.10156	0.06534	2.2E+07
28	04-21-86	0.10156	0.10243	0.09896	0.10156	0.06534	2.3E+07
29	04-22-86	0.10156	0.10156	0.09983	0.09983	0.06422	1.6E+07
30	04-23-86	0.09983	0.10069	0.09896	0.10026	0.0645	1.6E+07
31	04-24-86	0.10026	0.11198	0.09983	0.11024	0.07093	6.2E+07

➔ **First Step of the project is to collect the data from the given website.**

2.Data Preprocessing:

Clean and preprocess the data, handle missing values, and convert categorical features into numerical representations.

Identifying Missing Data

➔ Methods for identifying missing data

` **isnull():**

This function returns a pandas data frame, where each value is a Boolean value True if the value is missing, False otherwise.

notnull():

Similarly, to the previous function, the values for this one are False if either Nan or None value is detected.

isna():

This one is like is null and not null. However, it shows True only when the missing value is Nan type.

3.Feature Engineering:

Create additional features that could enhance the predictive power of the model, such as moving averages, technical indicators, and lagged variables.

Feature Engineering helps to derive some valuable features from the existing ones. These extra features sometimes help in increasing the performance of the model significantly and certainly help to gain deeper insights into the data.

Example: In our data set we have date in **month/date/year** format, when it comes to machine learning If we provide more accurate data then only the machine will give an more accurate prediction in our case we can divide the date into month, date, year in separate column by using python

```
splitted = df['Date'].str.split('/', expand=True)


df['day'] = splitted[1].astype('int')

df['month'] = splitted[0].astype('int')

df['year'] = splitted[2].astype('int')


df.head()
```

4.Model Selection:

Choose suitable algorithms for time series forecasting (e.g., ARIMA, LSTM) to predict stock prices.

Time-series forecasting:

Time series forecasting is perhaps one of the most common types of machine learning techniques used in real-world scenarios. time-series forecasting is a significant part of organizations' processes to forecast future revenue and profits.

Summary:

- We split the training dataset into train and test sets and we use the train set to fit the model, and generate a prediction **for each element on the test set**.
- **A rolling forecasting procedure is required given the dependence on observations in prior time steps for differencing and the AR model. To this end, we re-create the ARIMA model after each new observation is received.**
- Finally, we manually keep track of all observations in a list called **history** that is seeded with the training data and to which new observations are appended at each iteration.

5.Model Training:

Train the selected model using the pre-processed data.

Now is the time to train some state-of-the-art machine learning models(Logistic Regression, Support Vector Machine, XGBClassifier), and then based on their performance on the training and validation data we will choose which ML model is serving the purpose at hand better.

For the evaluation metric, we will use the ROC-AUC curve but why this is because instead of predicting the hard probability that is 0 or 1 we would like it to predict soft probabilities that are continuous values between 0 to 1. And with soft probabilities, the ROC-AUC curve is generally used to measure the accuracy of the predictions.

6.Evaluation:

Evaluate the model's performance using appropriate time series forecasting metrics (e.g., Mean Absolute Error, Root Mean Squared Error).

Transforming Into Code :

Step-1: Import Libraries

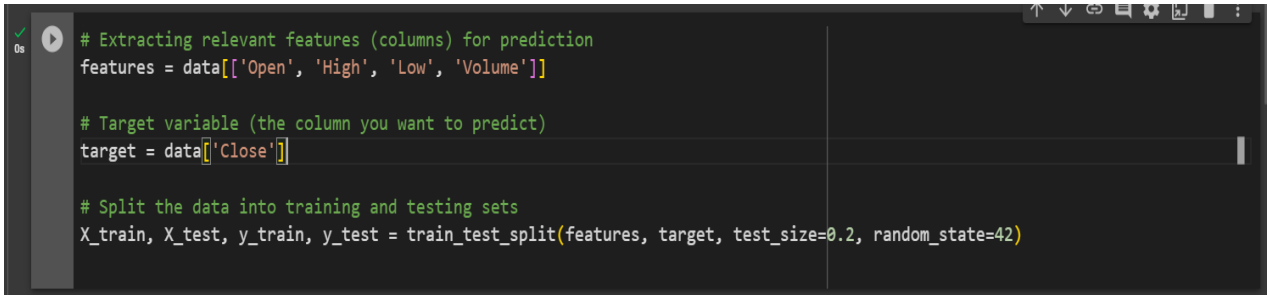
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

Step-2: Load and Exploring Data

```
# Load your dataset here, replace 'your_dataset.csv' with your actual file path
data = pd.read_csv('MSFT.csv')

# Explore the data
print(data.head())
```

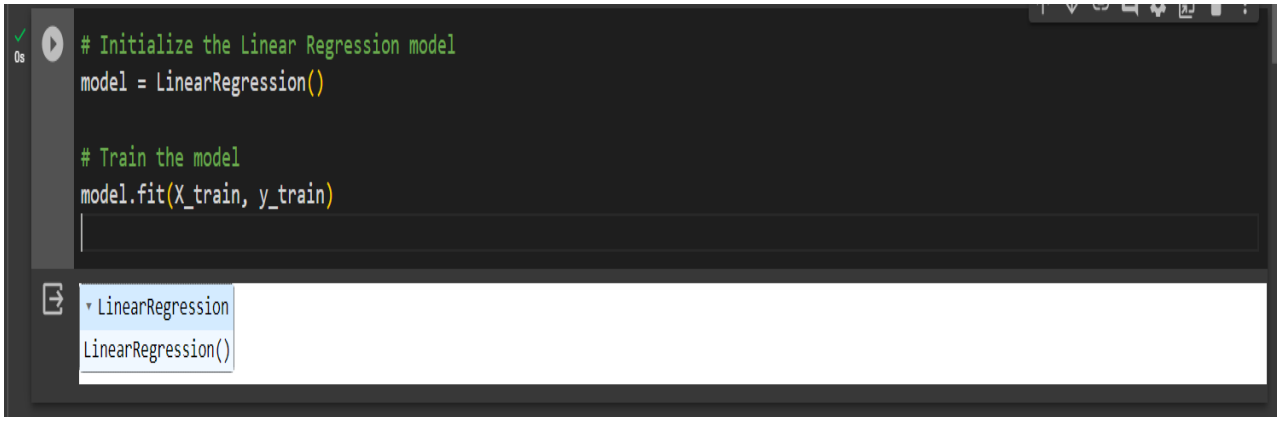
```
         Date      Open      High       Low     Close  Adj Close     Volume
0  1986-03-13  0.088542  0.101563  0.088542  0.097222   0.062549  1031788800
1  1986-03-14  0.097222  0.102431  0.097222  0.100694   0.064783   308160000
2  1986-03-17  0.100694  0.103299  0.100694  0.102431   0.065899   133171200
3  1986-03-18  0.102431  0.103299  0.098958  0.099826   0.064224    67766400
4  1986-03-19  0.099826  0.100694  0.097222  0.098090   0.063107    47894400
```

Step-3: Data Preprocessing



```
# Extracting relevant features (columns) for prediction
features = data[['Open', 'High', 'Low', 'Volume']]

# Target variable (the column you want to predict)
target = data['Close']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

Step-4: Build and Train the Model



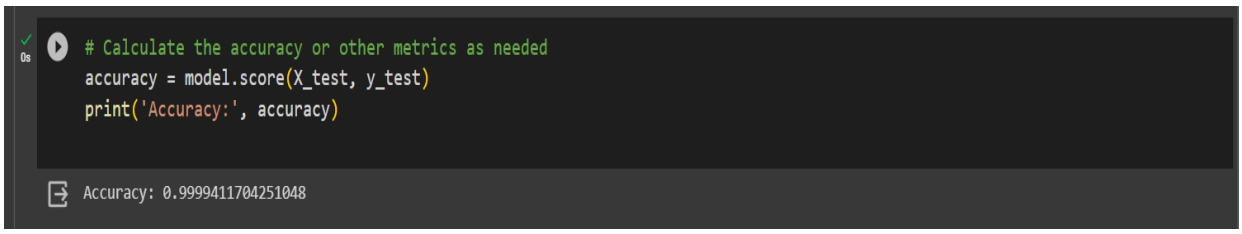
```
# Initialize the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

```
LinearRegression()
```

Step-5: Make Prediction

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

Step-6: Evaluate the Model



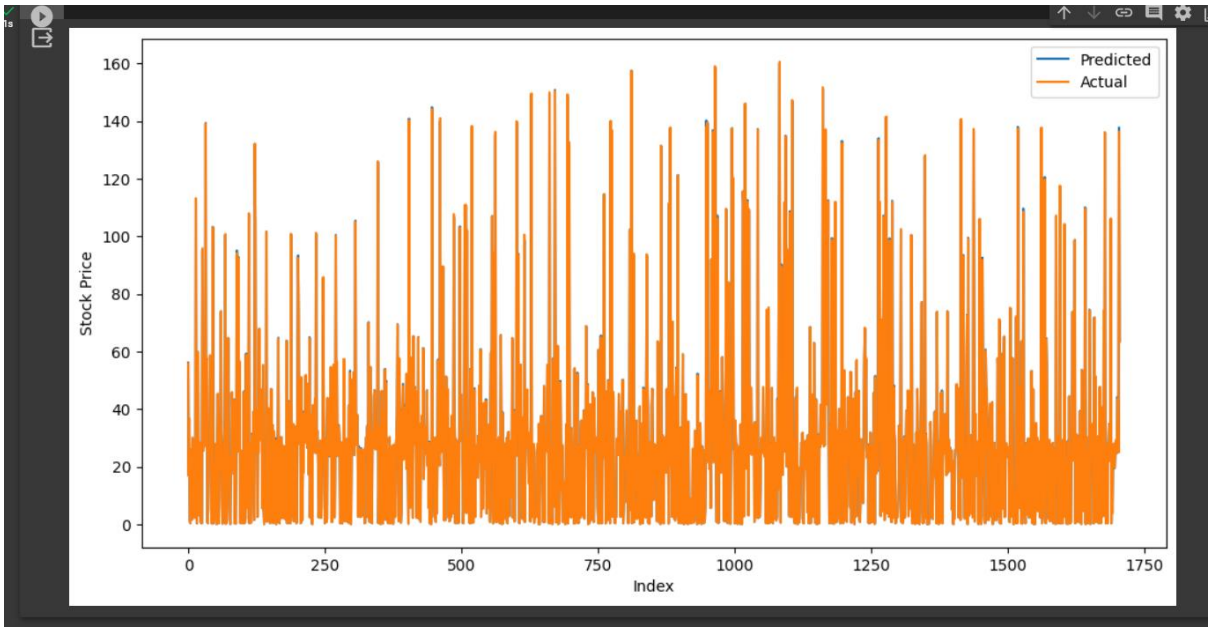
```
# Calculate the accuracy or other metrics as needed
accuracy = model.score(X_test, y_test)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.9999411704251048
```

Step-7: Visualize the Result

```
# Visualize the predicted vs. actual prices
plt.figure(figsize=(12, 6))
plt.plot(predictions, label='Predicted')
plt.plot(y_test.values, label='Actual')
plt.xlabel('Index')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

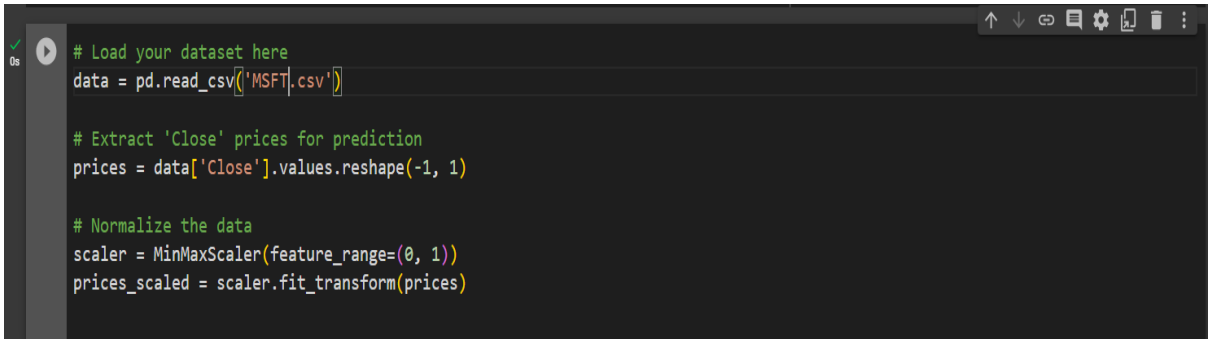
o/p:



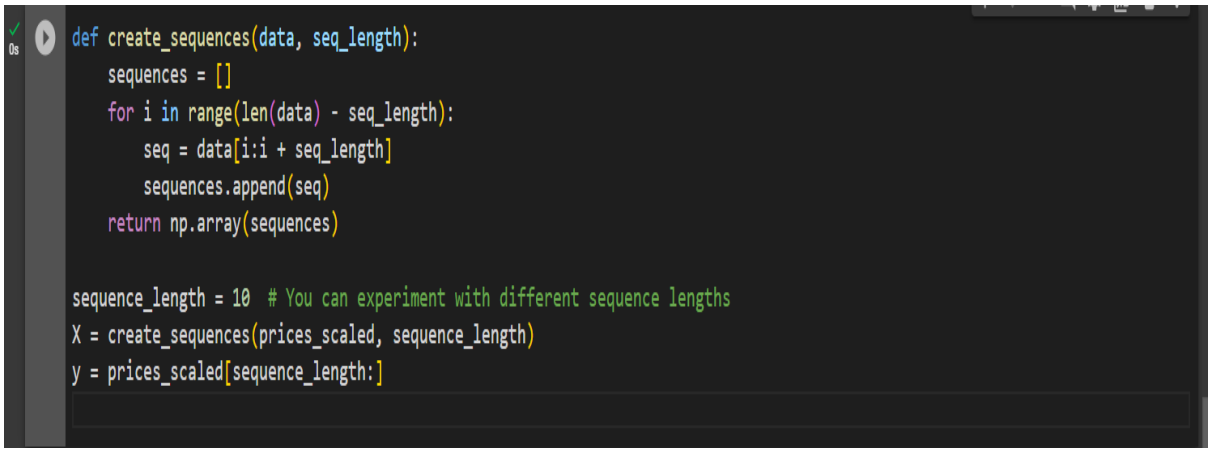
Advanced deep learning techniques like CNN-LSTM

Step-1: Import Libraries

Step-2: Load And Preprocess Data



Step-3: Create Sequences for LSTM



Step 4: Build CNN-LSTM Model with Attention Mechanism

```
input_shape = (X.shape[1], X.shape[2])

input_layer = Input(shape=input_shape)

# Convolutional layer
conv_layer = Conv1D(filters=64, kernel_size=3, activation='relu')(input_layer)
pooling_layer = MaxPooling1D(pool_size=2)(conv_layer)

# LSTM layer
lstm_layer = LSTM(50, return_sequences=True)(pooling_layer)

# Attention Mechanism
attention_weights = Dense(1, activation='softmax')(lstm_layer)
attention_output = Multiply()([lstm_layer, attention_weights])

# Flatten and Dense layers
flatten_layer = Flatten()(attention_output)
output_layer = Dense(1)(flatten_layer)

model = Model(inputs=input_layer, outputs=output_layer)

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

Step 5: Train the Model

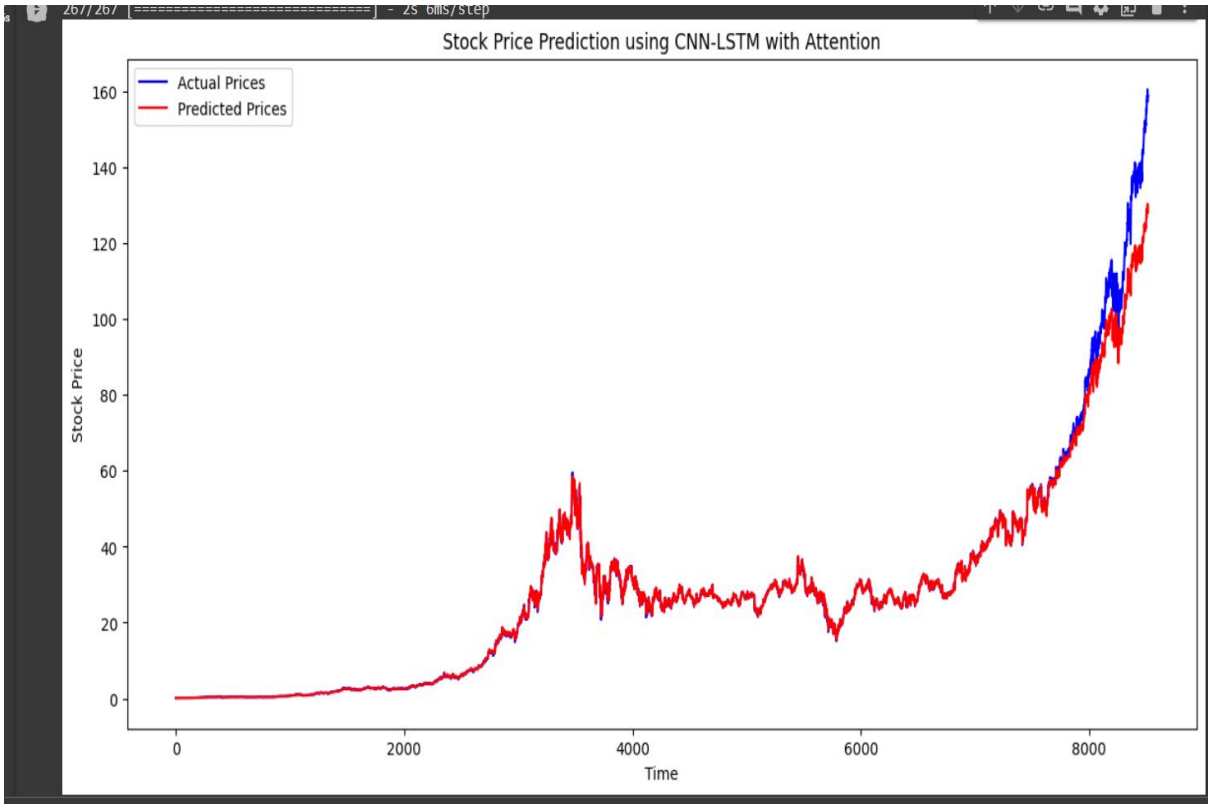
```
model.fit(X, y, epochs=10, batch_size=32, validation_split=0.1)
```

```
Epoch 1/10
240/240 [==============================] - 3s 14ms/step - loss: 1.7013e-05 - val_loss: 0.0039
Epoch 2/10
240/240 [==============================] - 3s 14ms/step - loss: 1.6567e-05 - val_loss: 0.0042
Epoch 3/10
240/240 [==============================] - 1s 6ms/step - loss: 1.7714e-05 - val_loss: 0.0048
Epoch 4/10
240/240 [==============================] - 2s 6ms/step - loss: 1.4846e-05 - val_loss: 0.0042
Epoch 5/10
240/240 [==============================] - 2s 7ms/step - loss: 1.7127e-05 - val_loss: 0.0044
Epoch 6/10
240/240 [==============================] - 1s 6ms/step - loss: 1.5118e-05 - val_loss: 0.0052
Epoch 7/10
240/240 [==============================] - 1s 6ms/step - loss: 1.5762e-05 - val_loss: 0.0044
Epoch 8/10
240/240 [==============================] - 2s 8ms/step - loss: 1.4230e-05 - val_loss: 0.0062
Epoch 9/10
240/240 [==============================] - 2s 9ms/step - loss: 1.5874e-05 - val_loss: 0.0047
Epoch 10/10
240/240 [==============================] - 2s 7ms/step - loss: 1.5635e-05 - val_loss: 0.0055
<keras.src.callbacks.History at 0x7e77e7be6f50>
```

Step 6:Vizualizing Data

```
import matplotlib.pyplot as plt

# Predict using the model
predicted_prices = model.predict(X)

# Inverse transform the predictions to the original scale
predicted_prices = scaler.inverse_transform(predicted_prices)
actual_prices = scaler.inverse_transform(y)

# Visualize the results
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('Stock Price Prediction using CNN-LSTM with Attention')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

O/P:


267/267 [==============================] - 2s 6ms/step
Stock Price Prediction using CNN-LSTM with Attention
Actual Prices
Predicted Prices
Stock Price
Time
0
20
40
60
80
100
120
140
160
0
2000
4000
6000
8000