Stock price Prediction of TCS using LSTM Neural Networks:

```
## import essential libraries

import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib
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 mdates
from sklearn import linear_model
from math import sqrt
from statsmodels.tsa.arima.model import ARIMA
```

Reading data using parse date

As it is timeseries data so we have to read the data by parsing date means making data column as index.

```
## read csv file
df_final =
pd.read_csv("TCS.csv",na_values=['null'],index_col='Date',parse_dates=
True,infer_datetime_format=True)

df_final.head()
```


```
{"summary":"{\n  \"name\": \"df_final\",\n  \"rows\": 4444,\n
\"fields\": [\n    {\n      \"column\": \"Date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2006-04-03 00:00:00\",\n        \"max\": \"2024-03-28 00:00:00\",\n
\"num_unique_values\": 4444,\n        \"samples\": [\n
\"2014-08-22 00:00:00\",\n          \"2022-04-04 00:00:00\",\n
\"2020-02-19 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Open\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1124.70733600944,\n        \"min\":
112.5,\n        \"max\": 4217.5,\n        \"num_unique_values\":
3168,\n        \"samples\": [\n          1935.900024,\n
611.724976,\n          306.25\n        ],\n        \"semantic_type\":
```

\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"High\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1133.8976099828915,\n   \"min\":
116.224998,\n   \"max\": 4254.450195,\n
\"num_unique_values\": 3774,\n   \"samples\": [\n
257.0,\n   3938.399902,\n   2075.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Low\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1114.3839651768114,\n
\"min\": 104.5,\n   \"max\": 4177.0,\n
\"num_unique_values\": 3821,\n   \"samples\": [\n
2251.100098,\n   754.125,\n   1249.550049\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Close\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 1124.1017754542036,\
n   \"min\": 111.287498,\n   \"max\": 4217.5,\n
\"num_unique_values\": 4281,\n   \"samples\": [\n
1748.199951,\n   276.5625,\n   309.237488\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Adj Close\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1141.3659846589962,\n   \"min\": 71.21302,\n   \"max\":
4217.5,\n   \"num_unique_values\": 4366,\n   \"samples\": [\
n   499.283508,\n   118.849922,\n   92.85215\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Volume\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
1030609.1053272161,\n   \"min\": 9598.0,\n   \"max\":
35573548.0,\n   \"num_unique_values\": 4388,\n
\"samples\": [\n   52056.0,\n   132170.0,\n
529064.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"df_final"}

```
df_final.shape
```

```
(4444, 6)
```

```
df_final.describe()
```

{"summary":"{\n \"name\": \"df_final\",\n \"rows\": 8,\n
\"fields\": [\n   {\n   \"column\": \"Open\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1617.6399810382227,\n   \"min\": 112.5,\n   \"max\":
4413.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n
1411.437896494675,\n   1190.0,\n   4413.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"High\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1622.331885813011,\n

\"min\": 116.224998,\n   \"max\": 4413.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
1425.6543149270337,\n   1200.75,\n   4413.0\n   ],\
n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Low\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 1612.9201145130107,\
n   \"min\": 104.5,\n   \"max\": 4413.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
1396.0706073929298,\n   1178.224976,\n   4413.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Close\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
1617.7202686087749,\n   \"min\": 111.287498,\n   \"max\":
4413.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n
1410.645708534104,\n   1190.375,\n   4413.0\n   ],\
n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Adj Close\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1651.7703029764918,\n   \"min\": 71.21302,\n   \"max\":
4413.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n
1296.6614509972808,\n   1050.273926,\n   4413.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Volume\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
12467442.469466617,\n   \"min\": 4413.0,\n   \"max\":
35573548.0,\n   \"num_unique_values\": 8,\n   \"samples\":
[\n   446359.8925900748,\n   184194.0,\n
4413.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

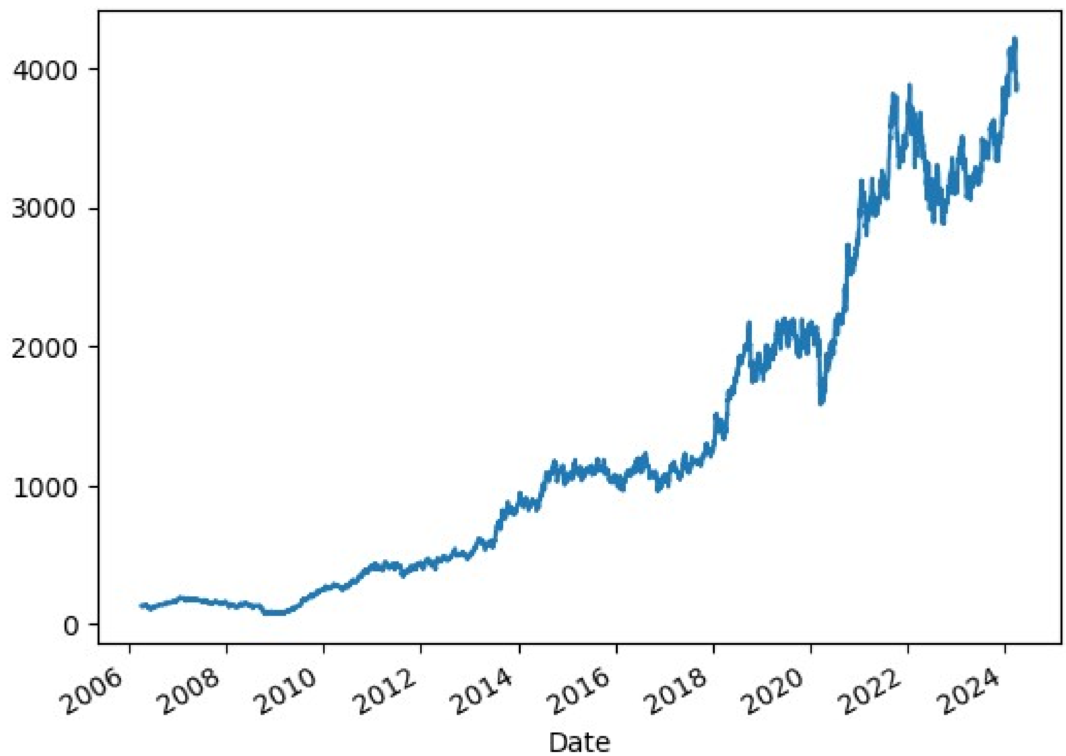
```
df_final.isnull().values.any()
```

```
True
```

```
df_final.dropna(inplace=True)
```

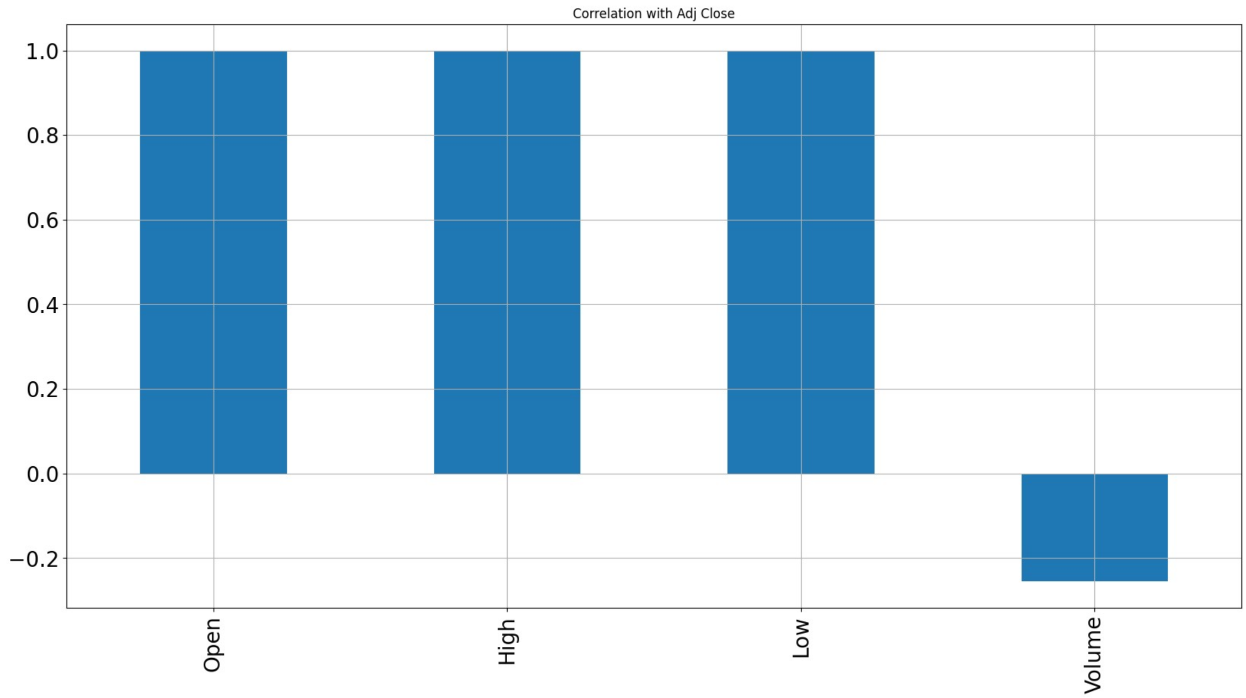
```
df_final['Adj Close'].plot()
```

```
<Axes: xlabel='Date'>
```



Correlation Analysis

```
X=df_final.drop(['Adj Close'],axis=1)
X=X.drop(['Close'],axis=1)

## correlation matrix

X.corrwith(df_final['Adj Close']).plot.bar(
        figsize = (20, 10), title = "Correlation with Adj Close",
fontsize = 20,
        rot = 90, grid = True)

<Axes: title={'center': 'Correlation with Adj Close'}>
```




```python
test = df_final
# Target column
target_adj_close = pd.DataFrame(test['Adj Close'])
display(test.head())
```

{"summary":"{\n \"name\": \"display(test\",\n \"rows\": 5,\n
\"fields\": [\n   {\n   \"column\": \"Date\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"2006-04-03 00:00:00\",\n   \"max\": \"2006-04-10 00:00:00\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n   \"2006-
04-04 00:00:00\",\n   \"2006-04-10 00:00:00\",\n
\"2006-04-05 00:00:00\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Open\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 3.637263738031654,\n   \"min\":
235.625,\n   \"max\": 244.375,\n   \"num_unique_values\":
5,\n   \"samples\": [\n   244.375,\n   235.625,\n
244.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"High\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 3.6881316801338864,\n   \"min\": 239.75,\n
\"max\": 250.0,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   245.975006,\n   239.75,\n
245.375\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Low\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 3.2020196263452227,\n   \"min\": 235.0,\n
\"max\": 241.90625,\n   \"num_unique_values\": 4,\n

```
\"samples\": [\n          241.90625,\n          235.0,\n
240.25\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Close\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3.5176968056226436,\n        \"min\": 236.100006,\n
\"max\": 243.481247,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          243.481247,\n          236.975006,\n
241.243744\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Adj
Close\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.913336254105718,\n        \"min\": 128.41803,\n
\"max\": 132.4328,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          132.4328,\n          128.893967,\n
131.215805\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Volume\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1066226.4222784953,\n        \"min\": 459912.0,\n
\"max\": 2982344.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          553648.0,\n          1666248.0,\n
732216.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```python
# selecting Feature Columns
feature_columns = ['Open', 'High', 'Low', 'Volume']
```

Normalizing the data

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
feature_minmax_transform_data =
scaler.fit_transform(test[feature_columns])
feature_minmax_transform = pd.DataFrame(columns=feature_columns,
data=feature_minmax_transform_data, index=test.index)
feature_minmax_transform.head()
```


```
{"summary":"{\n  \"name\": \"feature_minmax_transform\",\n  \"rows\":
4413,\n  \"fields\": [\n    {\n      \"column\": \"Date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2006-04-03 00:00:00\",\n        \"max\": \"2024-03-28 00:00:00\",\n
\"num_unique_values\": 4413,\n        \"samples\": [\n
\"2016-03-14 00:00:00\",\n          \"2016-02-05 00:00:00\",\n
\"2022-05-26 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Open\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.2739847347160633,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 3168,\n
\"samples\": [\n          0.4441900180267966,\n
0.1216138796589525,\n          0.04719853836784409\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"High\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.2740057768738416,\n
\"min\": 0.0,\n        \"max\": 0.9999999999999998,\n
\"num_unique_values\": 3774,\n        \"samples\": [\n
0.034018207153649976,\n          0.9236266085206961,\n
0.47333697630085736\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Low\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.27363633276287574,\n        \"min\": 0.0,\n        \"max\":
1.0,\n        \"num_unique_values\": 3821,\n        \"samples\": [\n
0.5270964022099448,\n          0.15951503990178023,\n
0.2811663717618171\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Volume\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.02897903931726414,\n        \"min\": 0.0,\n        \"max\":
1.0,\n        \"num_unique_values\": 4388,\n        \"samples\": [\n
0.0011938493896206694,\n          0.0034465237972722378,\n
0.014606532738911172\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"feature_minmax_transform"}

display(feature_minmax_transform.head())
print('Shape of features : ', feature_minmax_transform.shape)
print('Shape of target : ', target_adj_close.shape)

# Shift target array because we want to predict the n + 1 day value


target_adj_close = target_adj_close.shift(-1)
validation_y = target_adj_close[-90:-1]
target_adj_close = target_adj_close[:-90]

# Taking last 90 rows of data to be validation set
validation_X = feature_minmax_transform[-90:-1]
feature_minmax_transform = feature_minmax_transform[:-90]
display(validation_X.tail())
display(validation_y.tail())

print("\n -----After process------ \n")
print('Shape of features : ', feature_minmax_transform.shape)
print('Shape of target : ', target_adj_close.shape)
display(target_adj_close.tail())

{"summary":"{\n  \"name\": \"display(target_adj_close\",\n  \"rows\":
5,\n  \"fields\": [\n    {\n      \"column\": \"Date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2006-04-03 00:00:00\",\n        \"max\": \"2006-04-10 00:00:00\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"2006-
04-04 00:00:00\",\n          \"2006-04-10 00:00:00\",\n
```

\"2006-04-05 00:00:00\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Open\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0008860569398371863,\n   \"min\": 0.029993909866017056,\n   \"max\": 0.03212545676004872,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.03212545676004872,\n   0.029993909866017056,\n   0.03203410475030451\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"High\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0008912351321062925,\n   \"min\": 0.02984975348599908,\n   \"max\": 0.03232666073779164,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.03135402299857003,\n   0.02984975348599908,\n   0.03120903185588524\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Low\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0007862540518956933,\n   \"min\": 0.03204419889502763,\n   \"max\": 0.03374002455494168,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   0.03374002455494168,\n   0.03204419889502763,\n   0.03333333333333334\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Volume\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.02998053990848866,\n   \"min\": 0.012662091809261906,\n   \"max\": 0.08358874646938824,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.01529779453632119,\n   0.04658228346401342,\n   0.020318834100261644\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
Shape of features :  (4413, 4)
Shape of target :  (4413, 1)
```

{"summary":"{\n \"name\": \"display(target_adj_close\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"Date\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\": \"2024-03-20 00:00:00\",\n   \"max\": \"2024-03-27 00:00:00\",\n   \"num_unique_values\": 5,\n   \"samples\": [\n   \"2024-03-21 00:00:00\",\n   \"2024-03-27 00:00:00\",\n   \"2024-03-22 00:00:00\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Open\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.011725399840693478,\n   \"min\": 0.921912290133983,\n   \"max\": 0.943605359317905,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.9434591722289891,\n   0.921912290133983,\n   0.9224116930572472\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"High\",\n   \"properties\": {\n   \"dtype\": \"number\",

\"std\": 0.01228443852238587,\n   \"min\": 0.9136102490847599,\n
\"max\": 0.9428619314455349,\n   \"num_unique_values\": 5,\n
\"samples\": [\n   0.9405541669945033,\n
0.9136102490847599,\n   0.9236266085206961\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Low\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.01433844477603234,\n
\"min\": 0.9144383177409454,\n   \"max\": 0.9468385512584409,\n
\"num_unique_values\": 5,\n   \"samples\": [\n
0.9439779246163291,\n   0.9144383177409454,\n
0.9211786372007368\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Volume\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.002528162313880879,\n   \"min\":
0.001555760819593999,\n   \"max\": 0.007488706963090433,\n
\"num_unique_values\": 5,\n   \"samples\": [\n
0.001555760819593999,\n   0.0017389238259529667,\n
0.007488706963090433\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

{"summary":"{\n \"name\": \"display(target_adj_close\",\n \"rows\":
5,\n \"fields\": [\n   {\n   \"column\": \"Date\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"2024-03-20 00:00:00\",\n   \"max\": \"2024-03-27 00:00:00\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n   \"2024-
03-21 00:00:00\",\n   \"2024-03-27 00:00:00\",\n
\"2024-03-22 00:00:00\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Adj Close\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 50.776624672989826,\n
\"min\": 3837.5,\n   \"max\": 3974.050049,\n
\"num_unique_values\": 5,\n   \"samples\": [\n
3913.100098,\n   3883.550049,\n   3877.100098\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
 -----After process------

Shape of features :  (4323, 4)
Shape of target :  (4323, 1)
```

{"summary":"{\n \"name\": \"display(target_adj_close\",\n \"rows\":
5,\n \"fields\": [\n   {\n   \"column\": \"Date\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"2023-11-07 00:00:00\",\n   \"max\": \"2023-11-13 00:00:00\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n   \"2023-
11-08 00:00:00\",\n   \"2023-11-13 00:00:00\",\n
\"2023-11-09 00:00:00\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n

```
\"column\": \"Adj Close\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 30.180627479919625,\n
\"min\": 3331.699951,\n        \"max\": 3399.300049,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          3348.25,\
n          3399.300049,\n          3332.800049\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Train test Split using Timeseriessplit

```
ts_split= TimeSeriesSplit(n_splits=10)
for train_index, test_index in
ts_split.split(feature_minmax_transform):
        X_train, X_test = feature_minmax_transform[:len(train_index)],
feature_minmax_transform[len(train_index): (len(train_index)
+len(test_index))]
        y_train, y_test =
target_adj_close[:len(train_index)].values.ravel(),
target_adj_close[len(train_index): (len(train_index)
+len(test_index))].values.ravel()


X_train.shape

(3930, 4)

X_test.shape

(393, 4)

y_train.shape

(3930,)

y_test.shape

(393,)

def validate_result(model, model_name):
    predicted = model.predict(validation_X)
    RSME_score = np.sqrt(mean_squared_error(validation_y, predicted))
    print('RMSE: ', RSME_score)

    R2_score = r2_score(validation_y, predicted)
    print('R2 score: ', R2_score)

    plt.plot(validation_y.index, predicted,'r', label='Predict')
    plt.plot(validation_y.index, validation_y,'b', label='Actual')
    plt.ylabel('Price')
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-
```

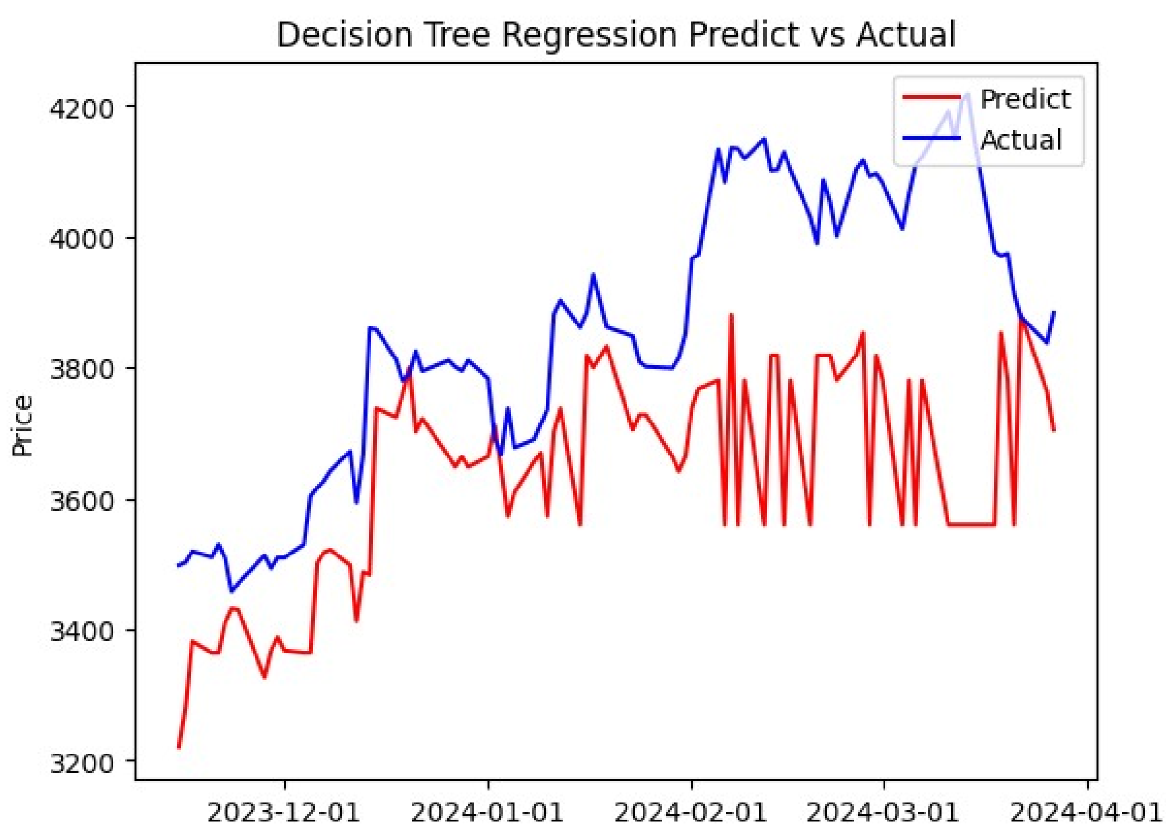
```
%d'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
    plt.title(model_name + ' Predict vs Actual')
    plt.legend(loc='upper right')
    plt.show()
```

Benchmark Model

Decision_Tree_Regressor

```
from sklearn.tree import DecisionTreeRegressor

dt = DecisionTreeRegressor(random_state=0)

benchmark_dt=dt.fit(X_train, y_train)

validate_result(benchmark_dt, 'Decision Tree Regression')
```

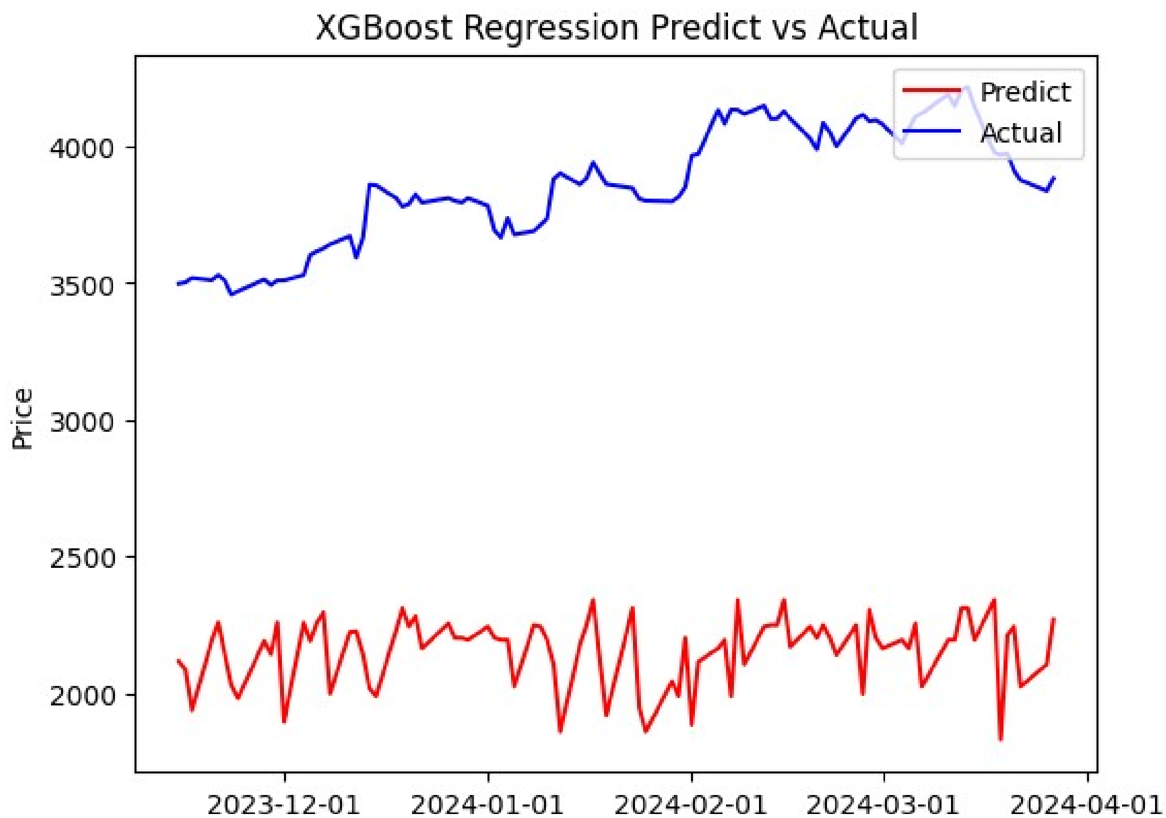
```
RMSE:  283.43805863878754
R2 score:  -0.7198385243711785
```



XG_Boost_Regressor

```
from xgboost import XGBRegressor


xg_reg = XGBRegressor(objective ='reg:squarederror', colsample_bytree
= 0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10,
random_state=0)

xg_reg.fit(X_train, y_train)


validate_result(xg_reg, 'XGBoost Regression')
```

```
RMSE:  1712.360752384338
R2 score:  -61.77140851929769
```



Time_Series_Arima_model

```
# Preprocess features (assuming 'Open', 'High', 'Low', 'Volume' are
your features)
features = df_final[['Open', 'High', 'Low', 'Volume']]
target = df_final['Adj Close']

# Scale features
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
feature_scaled = scaler.fit_transform(features)

# Initialize TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)

# Placeholder for model predictions and actual values
predictions = []
actuals = []

# Cross-validation for time series data
for train_index, test_index in tscv.split(feature_scaled):
    X_train, X_test = feature_scaled[train_index],
feature_scaled[test_index]
    y_train, y_test = target[train_index], target[test_index]

    # Fit ARIMA model (you can replace this with any other model)
    model = ARIMA(y_train, order=(5,1,0))
    model_fit = model.fit()

    # Forecast
    forecast = model_fit.get_forecast(steps=len(test_index))
    predictions.append(forecast.predicted_mean)
    actuals.append(y_test)

# Flatten the list of predictions and actuals
predictions = [item for sublist in predictions for item in sublist]
actuals = [item for sublist in actuals for item in sublist]

# Calculate RMSE over all predictions
rmse = sqrt(mean_squared_error(actuals, predictions))
print('Overall Test RMSE: %.3f' % rmse)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/
tsa_model.py:473: ValueWarning: A date index has been provided, but it
has no associated frequency information and so will be ignored when
e.g. forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
```

```
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
```

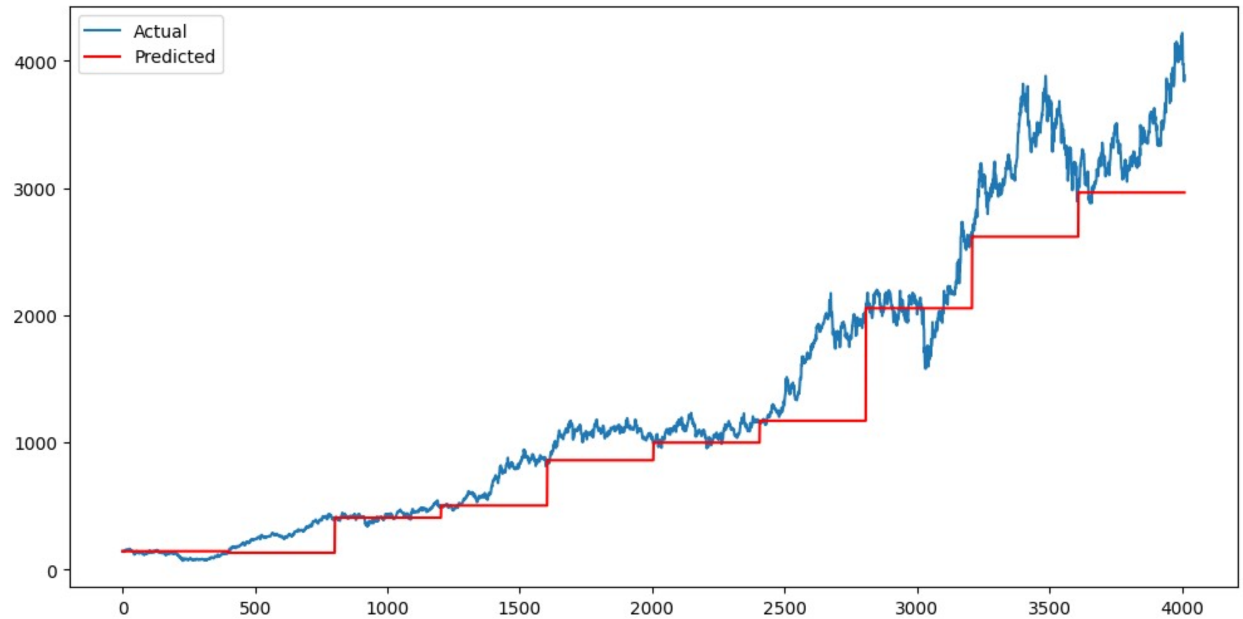
```
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
```

```
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: ValueWarning: No supported index is available. Prediction
results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: FutureWarning: No supported index is available. In the next
```

```
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:473: ValueWarning: A date index has been provided, but it has no
associated frequency information and so will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)

Overall Test RMSE: 360.757

/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/
tsa_model.py:836: ValueWarning: No supported index is available.
Prediction results will be given with an integer index beginning at
`start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model
.py:836: FutureWarning: No supported index is available. In the next
version, calling this method in a model without a supported index will
result in an exception.
  return get_prediction_index(
```

```python
# Plot the predicted vs actual values
plt.figure(figsize=(12,6))
plt.plot(actuals, label='Actual')
plt.plot(predictions, label='Predicted', color='red')
plt.legend()
plt.show()
```



Process the data for LSTM

```
X_train =np.array(X_train)
X_test =np.array(X_test)

X_tr_t = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_tst_t = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])
```

Model building : LSTM

```
from keras.models import Sequential
from keras.layers import Dense
import keras.backend as K
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
from keras.models import load_model
from keras.layers import LSTM
K.clear_session()
model_lstm = Sequential()
model_lstm.add(LSTM(16, input_shape=(1, X_train.shape[1]),
activation='relu', return_sequences=False))
model_lstm.add(Dense(1))
model_lstm.compile(loss='mean_squared_error', optimizer='adam')
early_stop = EarlyStopping(monitor='loss', patience=5, verbose=1)
history_model_lstm = model_lstm.fit(X_tr_t, y_train, epochs=200,
batch_size=8, verbose=1, shuffle=False, callbacks=[early_stop])
```

```
Epoch 1/200
502/502 [==============================] - 6s 5ms/step - loss:
2094184.8750
Epoch 2/200
502/502 [==============================] - 4s 7ms/step - loss:
2048436.7500
Epoch 3/200
502/502 [==============================] - 4s 8ms/step - loss:
1992790.2500
Epoch 4/200
502/502 [==============================] - 2s 5ms/step - loss:
1925644.1250
Epoch 5/200
502/502 [==============================] - 2s 4ms/step - loss:
1849066.7500
Epoch 6/200
502/502 [==============================] - 2s 5ms/step - loss:
1765117.5000
Epoch 7/200
502/502 [==============================] - 3s 6ms/step - loss:
1675649.0000
Epoch 8/200
502/502 [==============================] - 3s 6ms/step - loss:
1582312.3750
Epoch 9/200
502/502 [==============================] - 2s 4ms/step - loss:
1486581.3750
Epoch 10/200
502/502 [==============================] - 1s 3ms/step - loss:
1389788.7500
Epoch 11/200
502/502 [==============================] - 1s 2ms/step - loss:
1293147.8750
Epoch 12/200
502/502 [==============================] - 1s 2ms/step - loss:
1197767.8750
Epoch 13/200
502/502 [==============================] - 1s 2ms/step - loss:
1104655.6250
Epoch 14/200
502/502 [==============================] - 1s 2ms/step - loss:
1014707.1250
Epoch 15/200
502/502 [==============================] - 1s 2ms/step - loss:
928689.6250
Epoch 16/200
502/502 [==============================] - 2s 4ms/step - loss:
847213.0000
Epoch 17/200
502/502 [==============================] - 1s 3ms/step - loss:
```

```
770691.9375
Epoch 18/200
502/502 [==============================] - 1s 2ms/step - loss:
699302.3125
Epoch 19/200
502/502 [==============================] - 1s 2ms/step - loss:
632913.6875
Epoch 20/200
502/502 [==============================] - 1s 2ms/step - loss:
571017.8750
Epoch 21/200
502/502 [==============================] - 1s 2ms/step - loss:
512729.9375
Epoch 22/200
502/502 [==============================] - 1s 2ms/step - loss:
456975.6562
Epoch 23/200
502/502 [==============================] - 1s 2ms/step - loss:
402935.5000
Epoch 24/200
502/502 [==============================] - 1s 3ms/step - loss:
350563.6562
Epoch 25/200
502/502 [==============================] - 2s 4ms/step - loss:
300683.1250
Epoch 26/200
502/502 [==============================] - 2s 3ms/step - loss:
254455.3750
Epoch 27/200
502/502 [==============================] - 1s 2ms/step - loss:
212747.5781
Epoch 28/200
502/502 [==============================] - 1s 2ms/step - loss:
175928.6719
Epoch 29/200
502/502 [==============================] - 2s 3ms/step - loss:
143985.2500
Epoch 30/200
502/502 [==============================] - 1s 2ms/step - loss:
116678.7188
Epoch 31/200
502/502 [==============================] - 1s 2ms/step - loss:
93653.0312
Epoch 32/200
502/502 [==============================] - 1s 2ms/step - loss:
74508.3438
Epoch 33/200
502/502 [==============================] - 1s 2ms/step - loss:
58846.7188
Epoch 34/200
```

```
502/502 [==============================] - 1s 2ms/step - loss:
46267.8867
Epoch 35/200
502/502 [==============================] - 2s 3ms/step - loss:
36338.1211
Epoch 36/200
502/502 [==============================] - 2s 3ms/step - loss:
28601.0117
Epoch 37/200
502/502 [==============================] - 1s 2ms/step - loss:
22632.6172
Epoch 38/200
502/502 [==============================] - 1s 2ms/step - loss:
18070.5801
Epoch 39/200
502/502 [==============================] - 1s 2ms/step - loss:
14608.6709
Epoch 40/200
502/502 [==============================] - 1s 2ms/step - loss:
11984.9541
Epoch 41/200
502/502 [==============================] - 1s 3ms/step - loss:
9976.2451
Epoch 42/200
502/502 [==============================] - 1s 2ms/step - loss:
8399.7324
Epoch 43/200
502/502 [==============================] - 1s 2ms/step - loss:
7115.4697
Epoch 44/200
502/502 [==============================] - 1s 2ms/step - loss:
6028.5142
Epoch 45/200
502/502 [==============================] - 2s 3ms/step - loss:
5087.1914
Epoch 46/200
502/502 [==============================] - 2s 4ms/step - loss:
4274.3872
Epoch 47/200
502/502 [==============================] - 1s 2ms/step - loss:
3592.7478
Epoch 48/200
502/502 [==============================] - 1s 3ms/step - loss:
3049.0271
Epoch 49/200
502/502 [==============================] - 1s 2ms/step - loss:
2638.2642
Epoch 50/200
502/502 [==============================] - 1s 2ms/step - loss:
2338.9436
```

```
Epoch 51/200
502/502 [==============================] - 1s 3ms/step - loss:
2120.9207
Epoch 52/200
502/502 [==============================] - 1s 3ms/step - loss:
1956.1094
Epoch 53/200
502/502 [==============================] - 1s 2ms/step - loss:
1824.1105
Epoch 54/200
502/502 [==============================] - 1s 2ms/step - loss:
1712.3961
Epoch 55/200
502/502 [==============================] - 2s 4ms/step - loss:
1614.2424
Epoch 56/200
502/502 [==============================] - 1s 3ms/step - loss:
1526.5024
Epoch 57/200
502/502 [==============================] - 1s 2ms/step - loss:
1447.9760
Epoch 58/200
502/502 [==============================] - 1s 2ms/step - loss:
1378.3101
Epoch 59/200
502/502 [==============================] - 1s 3ms/step - loss:
1317.3724
Epoch 60/200
502/502 [==============================] - 1s 2ms/step - loss:
1264.9468
Epoch 61/200
502/502 [==============================] - 1s 2ms/step - loss:
1220.6526
Epoch 62/200
502/502 [==============================] - 1s 2ms/step - loss:
1183.9562
Epoch 63/200
502/502 [==============================] - 1s 2ms/step - loss:
1154.1979
Epoch 64/200
502/502 [==============================] - 1s 2ms/step - loss:
1130.6658
Epoch 65/200
502/502 [==============================] - 2s 3ms/step - loss:
1112.6184
Epoch 66/200
502/502 [==============================] - 1s 3ms/step - loss:
1099.3127
Epoch 67/200
502/502 [==============================] - 1s 3ms/step - loss:
```

```
1090.0347
Epoch 68/200
502/502 [==============================] - 1s 2ms/step - loss:
1084.1229
Epoch 69/200
502/502 [==============================] - 1s 3ms/step - loss:
1080.9624
Epoch 70/200
502/502 [==============================] - 1s 2ms/step - loss:
1080.0026
Epoch 71/200
502/502 [==============================] - 1s 2ms/step - loss:
1080.7570
Epoch 72/200
502/502 [==============================] - 2s 4ms/step - loss:
1082.7998
Epoch 73/200
502/502 [==============================] - 1s 2ms/step - loss:
1085.7689
Epoch 74/200
502/502 [==============================] - 2s 3ms/step - loss:
1089.3593
Epoch 75/200
502/502 [==============================] - 2s 4ms/step - loss:
1093.3225
Epoch 75: early stopping
```

Evaluation of Model

```
y_pred_test_lstm = model_lstm.predict(X_tst_t)
y_train_pred_lstm = model_lstm.predict(X_tr_t)
print("The R2 score on the Train set is:\
t{:0.3f}".format(r2_score(y_train, y_train_pred_lstm)))
r2_train = r2_score(y_train, y_train_pred_lstm)

print("The R2 score on the Test set is:\
t{:0.3f}".format(r2_score(y_test, y_pred_test_lstm)))
r2_test = r2_score(y_test, y_pred_test_lstm)

13/13 [==============================] - 0s 2ms/step
126/126 [==============================] - 0s 2ms/step
The R2 score on the Train set is:	0.998
The R2 score on the Test set is: 0.907
```

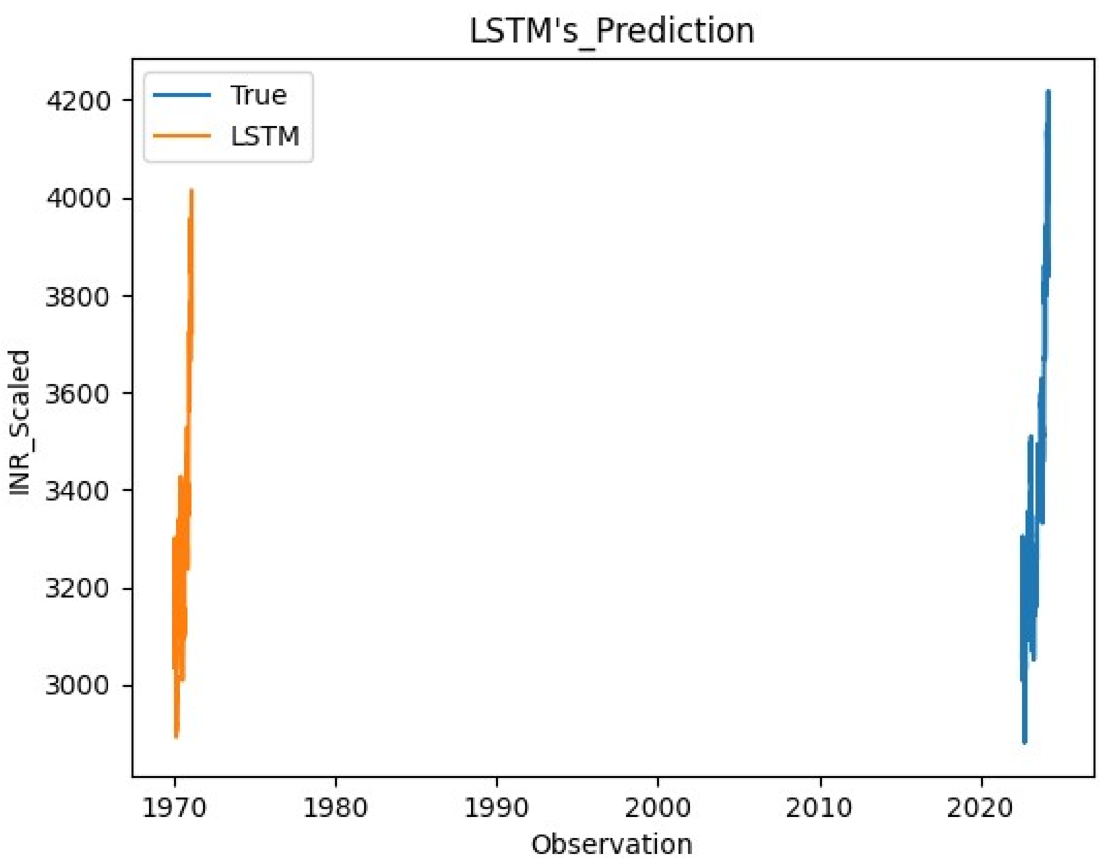
Predictions made by LSTM

```
score_lstm= model_lstm.evaluate(X_tst_t, y_test, batch_size=1)
```

```
401/401 [==============================] - 1s 1ms/step - loss:
8871.6533

print('LSTM: %f'%score_lstm)

LSTM: 8871.653320

y_pred_test_LSTM = model_lstm.predict(X_tst_t)

13/13 [==============================] - 0s 2ms/step
```

LSTM's Prediction Visual

```
## Plot to visualize the accuracy
plt.plot(y_test, label='True')
plt.plot(y_pred_test_LSTM, label='LSTM')
plt.title("LSTM's_Prediction")
plt.xlabel('Observation')
plt.ylabel('INR_Scaled')
plt.legend()
plt.show()
```



Converting Prediction data

In this step I have made the prediction of test data and will convert the dataframe to csv so that we can see the price difference between actual and predicted price.

```
col1 = pd.DataFrame(y_test, columns=['True'])

col2 = pd.DataFrame(y_pred_test_LSTM, columns=['LSTM_prediction'])

col3 = pd.DataFrame(history_model_lstm.history['loss'],
columns=['Loss_LSTM'])
results = pd.concat([col1, col2, col3], axis=1)
results.to_excel('PredictionResults_LSTM_NonShift.xlsx')
```

Conclusion

It is impossible to get a model that can 99% predict the price without any error, there are too many factors can affect the stock prices. So, we cannot hope there is a perfect model, but the general trend of predicted price is in line with the actual data, so the trader could have an indicator to reference, and makes trading decision by himself.

Further, we can improve the model's accuracy by increasing the epochs, trying out different activation functions or even change the model's structure. As exact