📈 Stock Market Prediction using LSTM (Machine Learning)

Descrption: This program uses an artificial recurrent neural network called Long Short Term Memory(LSTM) to predict the closing stock price of a corporation, using past 60 days stock price.

💡 1. Problem Defination

Train a neural network in such a way that it could able to predict the flow of a time series data that is the closing value of a particular corporate stock.

📁 2. Data

We are going to collect 10 years stock data from a website (https://finance.yahoo.com/ (https://finance.yahoo.com/)) using an api and the python module we are going to use to achive this is `pandas_datareader` .

📄 3. Evaluation

We would be judging our project based on Root Mean Squared Error (RSME) and R-Square Value. And Our Objective would be:-

> Achieving RSME score as low as possible.

> Achieving R-Square score as high as possible.

📌 4. Features

We would be focusing on the `Close` column of the stock data and would pre-process it and train it on the neural network to predict the future outcome.

🎛️ 5. Modelling

To solve the time series prediction the most used Estimator is LSTM which is an extension of RNN. It has some features like:-

- Remembering values over arbitrary time intervals
- Ability to use what it has learn previously as a new input along side with the new set of training data.

Note: Because of these features of LSTM it is trusted and widely used for time series prediction, and we are also choosing this algorithm to achieve our desired goal.

💻 6. Input Required

- This Project requires the name of a stock for example: 'RELIANCE' and the code for that stock 'NS' which means in which stock company it is enlisted (NS is the code of National Stock Exchange).
- For this input it returns an output of 3 graphs in which it shows:

> 10 years stock performance

> Model prediction performance

> comparison of predicted value with the previous day stock value.

In [1]:

```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
// will prevent jupyter notebook to enter into scroll mode when the output is long
```

In [2]:

```
# Import necessary libraries
import pandas_datareader as web
import math
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense,LSTM
import warnings
from sklearn.metrics import r2_score
from datetime import date
from datetime import timedelta
```

In [3]:

```
# from tensorflow import random as tf_random
# tf_random.set_seed(2)
```

In [4]:

```
# setting up graph style
plt.style.use("fivethirtyeight")
```

In [5]:

```
stock=input("Enter stock name:").upper()
code=input("Enter stock code:").upper()
if code=="":
    stock_code=stock
else:
    stock_code=stock+"."+code
stock_code=stock_code
print(f"Collecting the stock data of {stock_code}")
```

```
Enter stock name:reliance
Enter stock code:ns
Collecting the stock data of RELIANCE.NS
```

1. Data Collection and visualization

In [6]:

```
# Fetching data from the internet
df= web.DataReader(stock_code, data_source="yahoo", start='2012-01-01', end='2022-04-0
```

In [7]:

```
# printing the stock DataFrame in tabular format
df
```

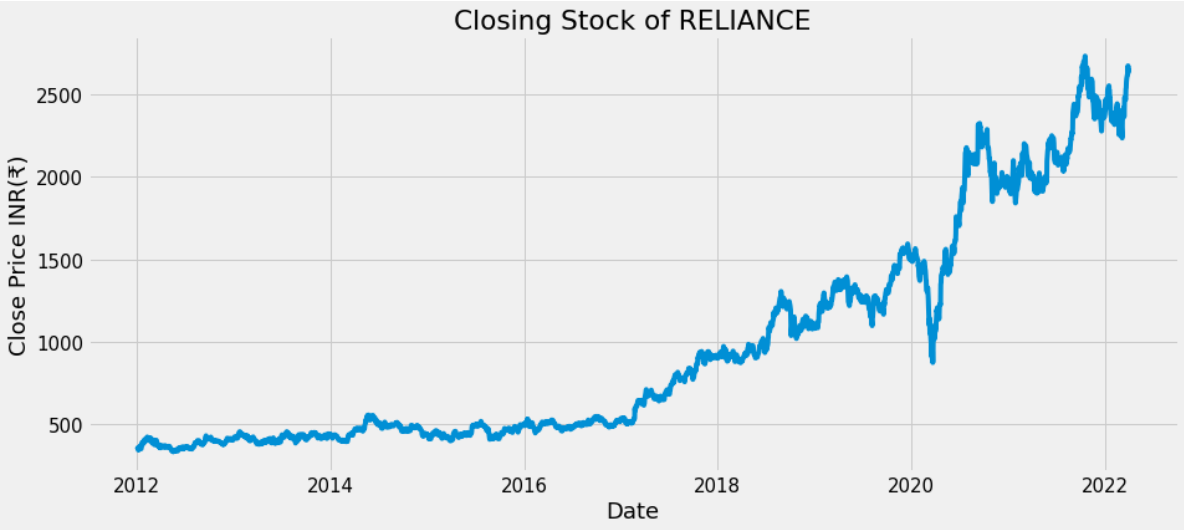
Out[7]:

Date	High	Low	Open	Close	Volume	Adj Close
2012-01-02	351.542725	340.348846	345.128540	349.957764	8679938.0	323.036743
2012-01-03	360.037201	351.839905	352.780975	358.922760	9455771.0	331.312134
2012-01-04	362.043182	353.325836	360.284851	354.712677	8557084.0	327.425934
2012-01-05	359.071350	343.791199	354.143066	346.465851	13364666.0	319.813477
2012-01-06	358.600830	345.054230	345.252350	355.406097	9495456.0	328.065979
...	...	...	...	...	...	...
2022-03-28	2629.750000	2586.500000	2610.000000	2621.949951	4564891.0	2621.949951
2022-03-29	2638.000000	2607.399902	2638.000000	2622.550049	4007695.0	2622.550049
2022-03-30	2688.000000	2617.100098	2639.899902	2672.949951	7297028.0	2672.949951
2022-03-31	2669.699951	2628.600098	2664.949951	2634.750000	6102744.0	2634.750000
2022-04-01	2665.149902	2622.000000	2636.000000	2655.850098	3656408.0	2655.850098

2527 rows × 6 columns

In [8]:

```
# Plotting and visualizing the Closing price of stock in graphical format
plt.rcParams.update({'font.size': 15})
# plt.figure(figsize=(14,6))
fig,ax=plt.subplots(figsize=(14,6))
ax.plot(df["Close"])
plt.title("Closing Stock of "+stock)
plt.xlabel("Date")
plt.ylabel("Close Price INR(₹)");
fig.savefig("./Images_exported/Closing Stock.png",bbox_inches = 'tight')
```



2. Pre-Processing Data

2.1 Feature Engineering

In [9]:

```
np.random.seed(42)
# Create a dataframe only with closing columns
data=df.filter(["Close"])
# Convert the data into a numpy array
dataset=data.values
# get the number of row to train the model on
training_data_len=math.ceil(len(dataset)*0.8)
training_data_len
```

Out[9]:

2022

In [10]:

```
np.random.seed(42)
# scale the data
scaler=MinMaxScaler(feature_range=(0,1))
scaled_data= scaler.fit_transform(dataset)
scaled_data
```

Out[10]:

```
array([[0.00629212],
       [0.01003225],
       [0.00827584],
       ...,
       [0.97542729],
       [0.95949056],
       [0.96829336]])
```

2.2 Creating Training Dataset

In [11]:

```
np.random.seed(42)
# Creating Scaled Training dataset
train_data= scaled_data[0:training_data_len,:]
# Split the data into x_train and y_train for training dataset
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])
    if i<=60:
        print(x_train)
        print(y_train)
        print()
```

```
[array([0.00629212, 0.01003225, 0.00827584, 0.00483532, 0.00856513,
       0.00623012, 0.01241893, 0.01490891, 0.01256357, 0.01150972,
       0.00761461, 0.01362775, 0.02141799, 0.02263716, 0.02408362,
       0.01972357, 0.02224453, 0.02373234, 0.02981781, 0.02451755,
       0.0291359 , 0.03204949, 0.03164654, 0.03351662, 0.03242145,
       0.03480811, 0.03767003, 0.03672983, 0.03453948, 0.03577931,
       0.03570698, 0.03317567, 0.02817504, 0.0293632 , 0.03460147,
       0.0325041 , 0.03378526, 0.02972482, 0.02176927, 0.02487917,
       0.02989013, 0.02786509, 0.028516  , 0.02508581, 0.02075675,
       0.01770885, 0.02018849, 0.02530278, 0.02972482, 0.02859865,
       0.02491016, 0.01988888, 0.01632438, 0.01745055, 0.01900033,
       0.01244992, 0.01404102, 0.01115844, 0.01135474, 0.01024923])]
[0.010032254891035236]
```

In [12]:

```
np.random.seed(42)
x_train,y_train=np.array(x_train),np.array(y_train)
x_train.shape,y_train.shape
```

Out[12]:

```
((1962, 60), (1962,))
```

2.3 Reshaping Data

Note: An LSTM Algo expect the data to be in 3D format- `[(no. of sample), (no. of time step), (no. of features)]`

In [13]:

```
np.random.seed(42)
# Reshape the data
np.random.seed(42)
x_train=np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_train.shape
```

Out[13]:

```
(1962, 60, 1)
```

3. Modelling

3.1 Setting up LSTM model

In [14]:

```
np.random.seed(42)
# Build the LSTM Model
np.random.seed(42)
model=Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

In [15]:

```
np.random.seed(42)
# compile the model
np.random.seed(42)
model.compile(optimizer='adam', loss='mean_squared_error')
```

In [16]:

```
%%time
np.random.seed(42)
# Train the model
model.fit(x_train,y_train,batch_size=1,epochs=1);
```

```
1962/1962 [==============================] - 53s 25ms/step - loss: 6.7369e-0
4
Wall time: 53 s
```

Out[16]:

```
<keras.callbacks.History at 0x1e43b03bfd0>
```

3.2 Setting up test data

In [17]:

```
np.random.seed(42)
# create the test dataset
# create a new array containing scaled values from index= training data has ended, to t
test_data= scaled_data[training_data_len-60:,:]
# create the dataset x_test, y_test
x_test=[]
y_test=dataset[training_data_len:,:]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i,0])
```

In [18]:

```
# convert the data into numpy array
x_test,y_test=np.array(x_test),np.array(y_test)
```

In [19]:

```
# Reshape the data
x_test = np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))
```

In [20]:

```
# Get the model predicted price value
predictions= model.predict(x_test)
predictions= scaler.inverse_transform(predictions)
```

4. Model Evaluation

4.1 Calculating Loss/Error using Root Mean Squared Error (RMSE)

In [21]:

```
rmse=round(np.sqrt(np.mean(predictions-y_test)**2),4)
rmse
```

Out[21]:

```
28.1226
```

4.2 Calculating Accuracy using R-Square metrics

In [22]:

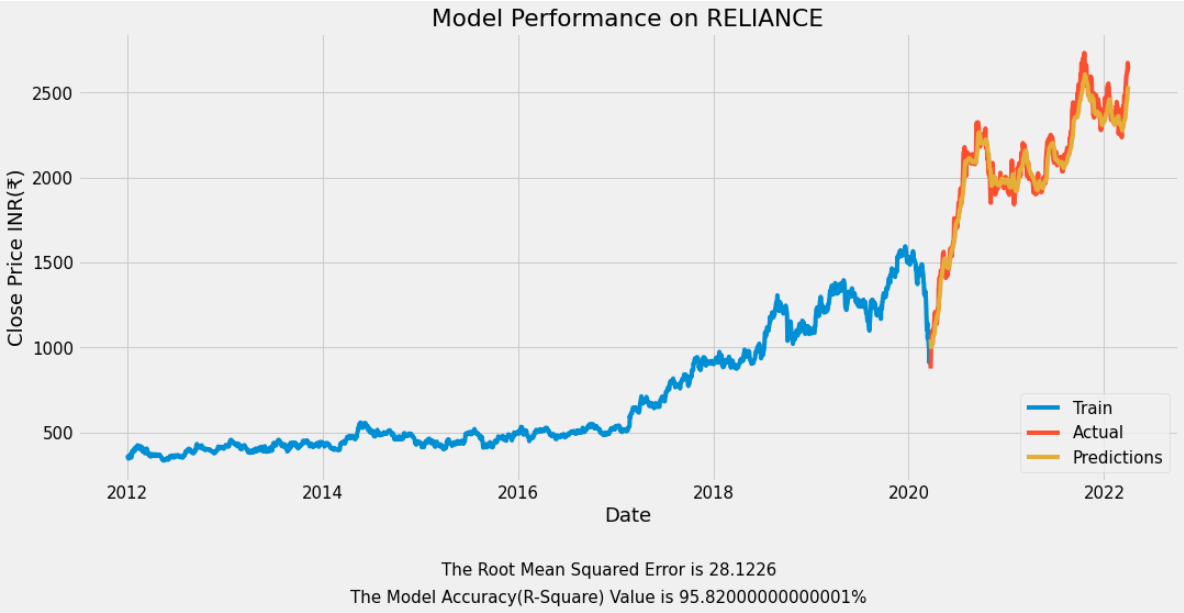
```
# Model Accuracy
r2=round(r2_score(y_test,predictions),4)
print(f"Model Accuracy accroding to R square is {r2*100}%")
```

```
Model Accuracy accroding to R square is 95.82000000000001%
```

4.3 Visualizing model performance

In [23]:

```
warnings.filterwarnings("ignore")
# plot the data
train=data[:training_data_len]
valid=data[training_data_len:]
valid["predictions"]=predictions

# Visualizing the data
fig,ax=plt.subplots(figsize=(16,7))
plt.title('Model Performance on '+stock)
plt.xlabel("Date")
plt.ylabel("Close Price INR(₹)")
ax.plot(train["Close"])
ax.plot(valid[["Close", "predictions"]])
plt.legend(["Train","Actual","Predictions"], loc="lower right");
# plt.text(1,1,"kncjbdasdnbvsvlsn dvlns dv")
rmse1="The Root Mean Squared Error is "+str(rmse)
r21="The Model Accuracy(R-Square) Value is "+str(r2*100)+"%"
fig.text(0.5,-0.1,rmse1,horizontalalignment='center')
fig.text(0.5,-0.15,r21,horizontalalignment='center')
fig.savefig("./Images_exported/Model_performance.png",bbox_inches = 'tight')
```



5. Model Deployment

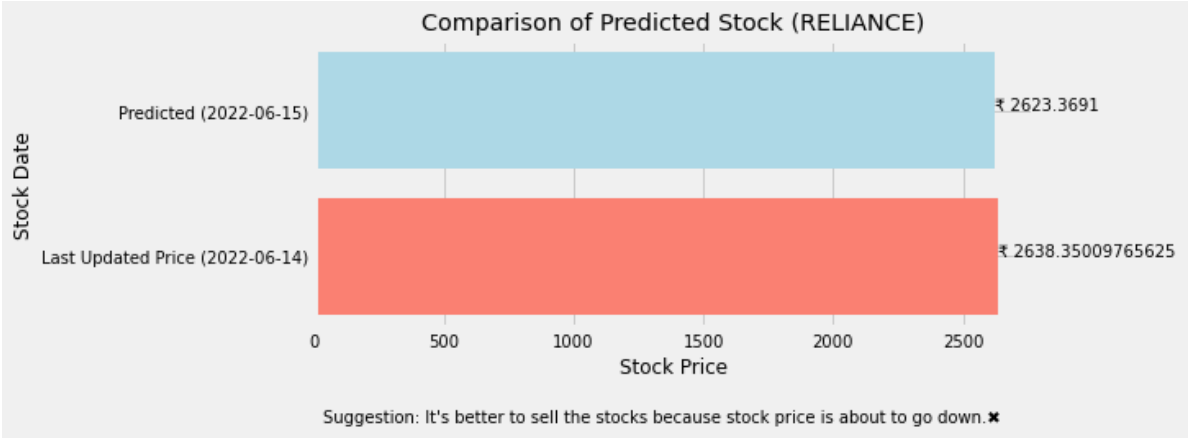
In [24]:

```
# getting todays date
today = date.today()
print("Today's date:", today)

#Get the quote

df_quote=web.DataReader(stock_code, data_source='yahoo', start='2012-01-01', end=today)

#Create a new dataframe

new_df = df_quote.filter(['Close'])

#Get teh last 60 day closing price values and convert the dataframe to an array

last_60_days = new_df[-60:].values

#Scale the data to be values between 0 and 1

last_60_days_scaled = scaler.transform(last_60_days)

#Create an empty list

X_test=[]

#Append teh past 60 days

X_test.append(last_60_days_scaled)

#Convert the X_test data set to a numpy array

X_test = np.array(X_test)

#Reshape the data

X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

#Get the predicted scaled price

pred_price = model.predict(X_test)

#undo the scaling

pred_price = scaler.inverse_transform(pred_price)

print("Predicted Stock Price is: ₹",pred_price[0][0])
```

```
Today's date: 2022-06-14
Predicted Stock Price is: ₹ 2623.3691
```

In [26]:

```
1
2  # setting up values
3  p=pred_price[0][0]
4  r=df_quote["Close"][-1:].values[0]
5  a=(df_quote["Close"].index[-1])
6  ri="Last Updated Price ("+str(df_quote["Close"].index[-1])[0:10]+")"
7  pi="Predicted ("+str(a+timedelta(days=1))[0:10]+")"
8  plt.rcParams.update({'font.size': 10})
9  # creating the dictionary
10 compare={ri:r, pi:p}
11 # plotting the dictionary
12 fig,ax=plt.subplots(figsize=(7,3))
13 ax.barh(list(compare.keys()),list(compare.values()), color=["salmon","lightblue"])
14 for index, value in enumerate(compare.values()):
15     plt.text(value, index,
16              "₹ "+str(value))
17 ax.set(title="Comparison of Predicted Stock ("+ stock+")",
18       xlabel="Stock Price",
19       ylabel="Stock Date");
20 print()
21 if r>p:
22     fig.text(0.5,-0.2,"Suggestion: It's better to sell the stocks because stock price i
23 else:
24     fig.text(0.5,-0.2,"Suggestion: It's better to invest the stocks because stock price
25 fig.savefig("./Images_exported/Comparison_of_Prediction_Stock.png",bbox_inches = 'tight
```



In [ ]:

```
1
```