Experiment 4 NIFTY

```
Step by step Procedure:

1.Data Collection.
    1.Collect the dataset or Create the dataset
2.Data Preprocessing.
    1.Import the Libraries.
    2.Importing the dataset.
    3.Analyse the data
    4.Taking care of Missing Data
    5.Selecting Closing value column for prediction
    6.Data Visualization
    7.Feature Scaling
    8.Splitting Data into Train and Test.
    9.Creating a datasets with a sliding window.

3.Model Building
    1.Import the model building Libraries
    2.Initializing the model
    3.Adding LSTM Layers
    4.Adding Output Layer
    5.Configure the Learning Process
    6.Training the model
4.Model Evaluation
5.Save the Model
6.Test the Model
```

```
!pip install tensorflow
```

```
!pip install keras
```

```
import tensorflow as tf
tf.__version__
```

```
'2.2.0'
```

```
import keras
keras.__version__
```

```
'3.1.1'
```

2.Data Preprocessing.

1.importing necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

2. Importing dataset

1.Since data is in form of excel file we have to use pandas read_excel to load the data 2.After loading it is important to check the complete information of data as it can indication many of the hidden infomation such as null values in a column or a row 3.Check whether any null values are there or not. if it is present then following can be done, a.Imputing data using Imputation method in sklearn b.Filling NaN values with mean, median and mode using fillna() method 4.Describe data --> which can give statistical analysis

```
data=pd.read_csv(r"D:\KMIT\NLP_Lab\Experiments\Tulasi\Dataset\Exp4\MARUTI.csv")
```

3.Analyse the data

```
data.head()
```

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverbl
0	2003-07-09	MARUTI	EQ	125.00	164.90	170.40	155.00	164.0	164.30	165.95	35164283	5.835528e+14	NaN	8537695.0	0.242
1	2003-07-10	MARUTI	EQ	164.30	167.00	168.70	164.50	167.0	167.00	166.74	10464179	1.744820e+14	NaN	4363947.0	0.417
2	2003-07-11	MARUTI	EQ	167.00	167.75	174.85	166.25	173.6	173.35	172.45	11740117	2.024622e+14	NaN	3014852.0	0.256

```
data.tail()
```

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume
4422	2021-04-26	MARUTI	EQ	6676.10	6690.20	6789.00	6600.0	6645.00	6638.90	6678.34	937344	6.259903e+14	74474.0	464999.0
4423	2021-04-27	MARUTI	EQ	6638.90	6669.95	6709.00	6542.0	6552.00	6568.75	6620.68	1610651	1.066360e+15	130986.0	588617.0
4424	2021-04-28	MARUTI	EQ	6568.75	6568.75	6650.00	6545.0	6581.00	6573.80	6598.62	1406270	9.279437e+14	117843.0	672435.0

```
data.describe()
```

	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Tra
count	4427.000000	4427.000000	4427.000000	4427.000000	4427.000000	4427.000000	4427.000000	4.427000e+03	4.427000e+03	2456.000
mean	2923.575085	2927.873074	2962.918432	2889.128066	2924.651604	2925.005094	2926.480642	1.194661e+06	2.395307e+14	55428.511
std	2740.532701	2745.541243	2769.986950	2715.403311	2740.438635	2740.723734	2742.675329	1.637957e+06	2.935761e+14	44405.350
min	125.000000	164.000000	168.700000	155.000000	164.000000	164.300000	165.060000	2.279600e+04	2.131518e+12	1096.000
25%	822.525000	825.100000	840.000000	806.300000	823.025000	822.700000	823.435000	4.263710e+05	6.248277e+13	23089.500
50%	1412.450000	1414.000000	1432.000000	1390.350000	1412.200000	1412.600000	1412.210000	6.909590e+05	1.121591e+14	44031.500
75%	5097.350000	5100.000000	5192.050000	5006.025000	5104.500000	5104.200000	5114.920000	1.208280e+06	3.141731e+14	73714.500

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4427 entries, 0 to 4426
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Date                4427 non-null   object 
 1   Symbol              4427 non-null   object 
 2   Series              4427 non-null   object 
 3   Prev Close          4427 non-null   float64
 4   Open                4427 non-null   float64
 5   High                4427 non-null   float64
 6   Low                 4427 non-null   float64
 7   Last                4427 non-null   float64
 8   Close               4427 non-null   float64
 9   VWAP                4427 non-null   float64
 10  Volume              4427 non-null   int64  
 11  Turnover            4427 non-null   float64
 12  Trades              2456 non-null   float64
 13  Deliverable Volume  4426 non-null   float64
 14  %Deliverble         4426 non-null   float64
dtypes: float64(11), int64(1), object(3)
memory usage: 518.9+ KB
```

Double-click (or enter) to edit

4.Taking care of Missing Data

```
data.isnull().any()
```

```
Date                  False
Symbol                False
Series                False
Prev Close            False
Open                  False
High                  False
```

```
Low                   False
Last                  False
Close                 False
VWAP                  False
Volume                False
Turnover              False
Trades                 True
Deliverable Volume     True
%Deliverble            True
dtype: bool
```

```
data.isnull().sum()
```

```
Date                     0
Symbol                   0
Series                   0
Prev Close               0
Open                     0
High                     0
Low                      0
Last                     0
Close                    0
VWAP                     0
Volume                   0
Turnover                 0
Trades                1971
Deliverable Volume       1
%Deliverble              1
dtype: int64
```

```
data.shape
```

```
(4427, 15)
```

5.Selecting Closing value column for prediction

```
data_Close=data.reset_index()['Close']
```

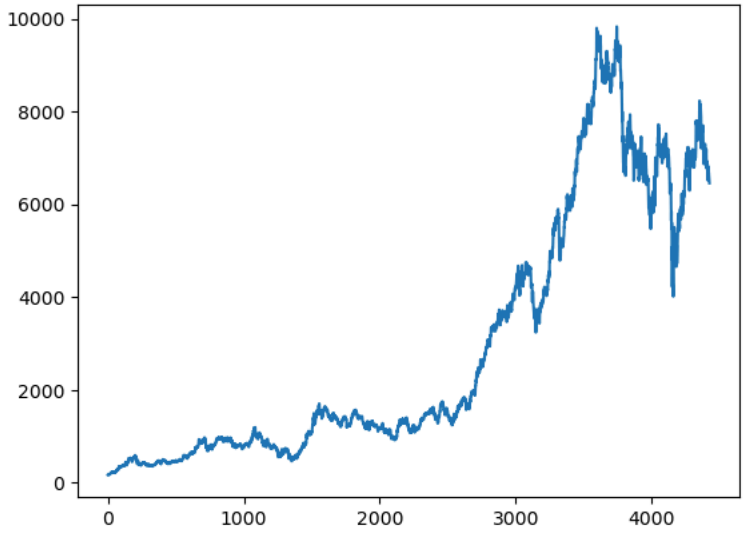
```
data_Close
```

```
0        164.30
1        167.00
2        173.35
3        177.95
4        176.20
          ...
4422    6638.90
4423    6568.75
4424    6573.80
4425    6565.65
4426    6455.65
Name: Close, Length: 4427, dtype: float64
```

7.Data Visualization

```
plt.plot(data_Close)
```

```
[<matplotlib.lines.Line2D at 0x1339ef97590>]
```



8.Feature Scaling

```
### LSTM are sensitive to the scale of the data. so we apply MinMax scaler
```

```
#Featuring Scaling
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
data_Close=scaler.fit_transform(np.array(data_Close).reshape(-1,1))
```

```
print(data_Close)
```

```
[[0.00000000e+00]
 [2.79267492e-04]
 [9.36063259e-04]
 ...
 [6.62949996e-01]
 [6.62107022e-01]
 [6.50729457e-01]]
```

8.Splitting Data into Train and Test.

```
training_size=int(len(data_Close)*0.70)
test_size=len(data_Close)-training_size
train_data,test_data=data_Close[0:training_size,:],data_Close[training_size:len(data_Close),:1]
```

```
training_size,test_size
```

```
(3098, 1329)
```

```
train_data
```

```
array([[0.00000000e+00],
       [2.79267492e-04],
       [9.36063259e-04],
       ...,
       [4.46460802e-01],
       [4.56395484e-01],
       [4.60863764e-01]])
```

```
train_data.shape
```

```
(3098, 1)
```

9.Creating a datasets with a sliding window.

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----9   10
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

```
# reshape into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 10
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

```
print(X_train.shape), print(y_train.shape)
```

```
(3087, 10)
(3087,)
(None, None)
```

```
print(X_test.shape), print(ytest.shape)
```

```
(1318, 10)
(1318,)
(None, None)
```

```
X_train
```

```
array([[0.00000000e+00, 2.79267492e-04, 9.36063259e-04, ...,
        3.20640453e-04, 6.20594426e-05, 3.05125593e-04],
       [2.79267492e-04, 9.36063259e-04, 1.41185232e-03, ...,
        6.20594426e-05, 3.05125593e-04, 1.07569701e-03],
       [9.36063259e-04, 1.41185232e-03, 1.23084561e-03, ...,
        3.05125593e-04, 1.07569701e-03, 1.01363756e-03],
       ...,
       [4.59472598e-01, 4.56731639e-01, 4.59508800e-01, ...,
        4.55092236e-01, 4.52852924e-01, 4.46290138e-01],
       [4.56731639e-01, 4.59508800e-01, 4.54264777e-01, ...,
        4.52852924e-01, 4.46290138e-01, 4.48255354e-01],
       [4.59508800e-01, 4.54264777e-01, 4.60739645e-01, ...,
        4.46290138e-01, 4.48255354e-01, 4.46460802e-01]])
```

```
y_train
```

```
array([0.0010757 , 0.00101364, 0.00096709, ..., 0.44825535, 0.4464608 ,
       0.45639548])
```

```
# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
```

3.Model Building

Create the Stacked LSTM model

```
#tensorflow :open source used for both ML and DL for computation
from tensorflow.keras.models import Sequential#it is a plain stack of layers
from tensorflow.keras.layers import Dense#Dense layer is the regular deeply connected neural network layer
from tensorflow.keras.layers import LSTM #Long Short Trem Memory
```

```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(10,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

```
C:\Users\sritu\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` ar
  super().__init__(**kwargs)
```

```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 10, 50)	10,400
lstm_1 (LSTM)	(None, 10, 50)	20,200
lstm_2 (LSTM)	(None, 50)	20,200
dense (Dense)	(None, 1)	51

```
 Total params: 50,851 (198.64 KB)
 Trainable params: 50,851 (198.64 KB)
 Non-trainable params: 0 (0.00 B)
```

```
#Training the model
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=50,batch_size=64,verbose=1)
```

```
Epoch 1/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 9s 48ms/step - loss: 0.0075 - val_loss: 9.2287e-04
Epoch 2/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - loss: 7.6529e-05 - val_loss: 9.9503e-04
Epoch 3/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - loss: 5.2937e-05 - val_loss: 8.9124e-04
Epoch 4/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 4.7823e-05 - val_loss: 0.0011
Epoch 5/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 4.9597e-05 - val_loss: 0.0020
Epoch 6/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 4.9881e-05 - val_loss: 0.0016
Epoch 7/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 4.9877e-05 - val_loss: 0.0016
Epoch 8/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 5.2346e-05 - val_loss: 0.0013
Epoch 9/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 4.9056e-05 - val_loss: 0.0019
Epoch 10/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 4.7544e-05 - val_loss: 0.0024
Epoch 11/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 4.8404e-05 - val_loss: 0.0022
Epoch 12/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 4.6095e-05 - val_loss: 0.0021
Epoch 13/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 4.7162e-05 - val_loss: 0.0015
Epoch 14/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 4.5545e-05 - val_loss: 0.0014
Epoch 15/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 5.1477e-05 - val_loss: 0.0013
Epoch 16/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 4.5822e-05 - val_loss: 0.0035
Epoch 17/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 6.0006e-05 - val_loss: 0.0012
Epoch 18/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - loss: 4.6423e-05 - val_loss: 0.0015
Epoch 19/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - loss: 4.8288e-05 - val_loss: 0.0012
Epoch 20/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 4.5883e-05 - val_loss: 0.0012
Epoch 21/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 4.5656e-05 - val_loss: 0.0012
Epoch 22/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 4.7039e-05 - val_loss: 0.0016
Epoch 23/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 4.4439e-05 - val_loss: 0.0023
Epoch 24/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 4.6910e-05 - val_loss: 0.0015
Epoch 25/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 4.2210e-05 - val_loss: 0.0020
Epoch 26/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 4.8272e-05 - val_loss: 0.0012
Epoch 27/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 4.3019e-05 - val_loss: 0.0013
Epoch 28/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - loss: 4.1394e-05 - val_loss: 0.0017
Epoch 29/50
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 4.1895e-05 - val_loss: 0.0025
```

```
## Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

```
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

4.Model Evaluation

```
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

```
### Test Data RMSE
math.sqrt(mean_squared_error(ytest,test_predict))
```

Start coding or generate with AI.

Predict the train and test data and plot the output

```
### Plotting
# shift train predictions for plotting
look_back=10
trainPredictPlot = np.empty_like(data_Close)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = np.empty_like(data_Close)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(data_Close)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(data_Close))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

5.Save the model

```
model.save("nifty.h5")
```

6.Test the model

```
#prediction for next 10 days
```

```
len(test_data)
```

```
x_input=test_data[1319:].reshape(1,-1)
x_input.shape
```

```
test_data[1319:].reshape(1,-1)
```

```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

```
temp_input
```

```
len(temp_input)
```

Predict the future 10 days and plot the graph

```
# demonstrate prediction for next 10 days
from numpy import array

lst_output=[]
n_steps=10
i=0
while(i<10):

    if(len(temp_input)>10):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
```