```
*STOCK* MARKET ANALYSIS AND PREDICTION USING LSTM
```

STOCK MARKET ANALYSIS AND PREDICTION USING LSTM

Dataset consists of following files:

prices.csv: raw, as-is daily prices. Most of data spans from 2010 to the end 2016, for companies new on stock market date range is shorter. There have been approx. 140 stock splits in that time, this set doesn't account for that. prices-split-adjusted.csv: same as prices, but there have been added adjustments for splits.

securities.csv: general description of each company with division on sectors

fundamentals.csv: metrics extracted from annual SEC 10K fillings (2012-2016), should be enough to derive most of popular fundamental indicators.

Importing the basic libraries

```
#basic libs
import pandas as pd
import numpy as np

#visualizing libs
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
#from jupyterthemes import jtplot
#jtplot.style(theme="monokai",context="notebook",ticks=True,grid=True)

#deep learning libs
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,LSTM,Dropout

#stats libs
from scipy import stats

#preprocessing libs
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#warnings
```

```
import warnings
warnings.filterwarnings("ignore")
```

Importing files

```
df=pd.read_csv("/content/prices.csv")
```

Checking the dataset

```
df.head()
```

	date	symbol	open	close	low	high	volume
0	2016-01-05 00:00:00	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06 00:00:00	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07 00:00:00	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
[illegible]	2016-01-08	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
df.shape
```

```
(851264, 7)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 851264 entries, 0 to 851263
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   date    851264 non-null  object 
 1   symbol  851264 non-null  object 
 2   open    851264 non-null  float64
 3   close   851264 non-null  float64
 4   low     851264 non-null  float64
 5   high    851264 non-null  float64
 6   volume  851264 non-null  float64
dtypes: float64(5), object(2)
memory usage: 45.5+ MB
```

```
df.isna().sum()
```

```
date      0
symbol    0
open      0
close     0
low       0
high      0
volume    0
dtype: int64
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	
open	851264.0	7.083699e+01	8.369588e+01	0.85	3.384000e+01	5.277000e+01	7.98
close	851264.0	7.085711e+01	8.368969e+01	0.86	3.385000e+01	5.280000e+01	7.98
low	851264.0	7.011841e+01	8.287729e+01	0.83	3.348000e+01	5.223000e+01	7.91
high	851264.0	7.154348e+01	8.446550e+01	0.88	3.419000e+01	5.331000e+01	8.06
volume	851264.0	5.415113e+06	1.249468e+07	0.00	1.221500e+06	2.476250e+06	5.22

```
# lets check the number of companies in the dataset
len(df["symbol"].unique())
```

```
501
```

Sampling the dataset for the company "APPLE"

At the moment we are precisely iterested in the stock prices variation of "apple".So we will be sampling all the apple datas from the given dataset and proceed further

```
#AAPL is the symbol for apple in NYSE so we will be separating our dataset for apple precisel
df1=df[df["symbol"]=="AAPL"]
df1.head()
```

date	symbol	open	close	low	high	volume

```
df1.describe().T
```

	count	mean	std	min	25%	50%	
open	1762.0	3.130763e+02	1.852995e+02	9.000000e+01	1.152225e+02	3.182300e+02	4.7
close	1762.0	3.129271e+02	1.851471e+02	9.028000e+01	1.151900e+02	3.182400e+02	4.7
low	1762.0	3.098282e+02	1.833839e+02	8.947000e+01	1.140025e+02	3.165450e+02	4.6
high	1762.0	3.159113e+02	1.868982e+02	9.070000e+01	1.163625e+02	3.206000e+02	4.7
volume	1762.0	9.422578e+07	6.020519e+07	1.147590e+07	4.917478e+07	8.050385e+07	1.2

```
df1.shape
```

```
(1762, 7)
```

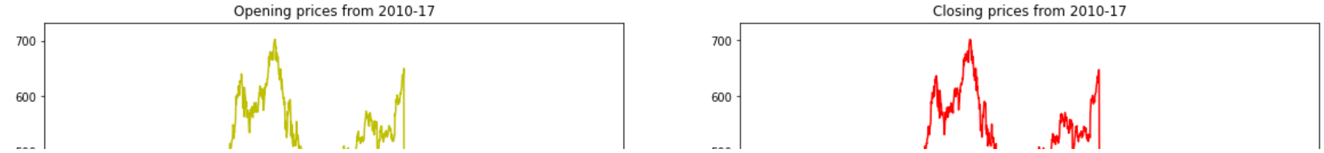
```
#converting date to date time
df1["date"]=pd.to_datetime(df1["date"])
```

Exploratory data analysis

Visualising the stock opening and closing prices over the period of 6 years

```
fig,ax=plt.subplots(1,2,figsize=(20,6))
sns.lineplot(data=df1,x="date",y="open",ax=ax[0],color="y").set_title("Opening prices from 20
sns.lineplot(data=df1,x="date",y="close",ax=ax[1],color="r").set_title("Closing prices from 2
```

```
Text(0.5, 1.0, 'Closing prices from 2010-17')
```



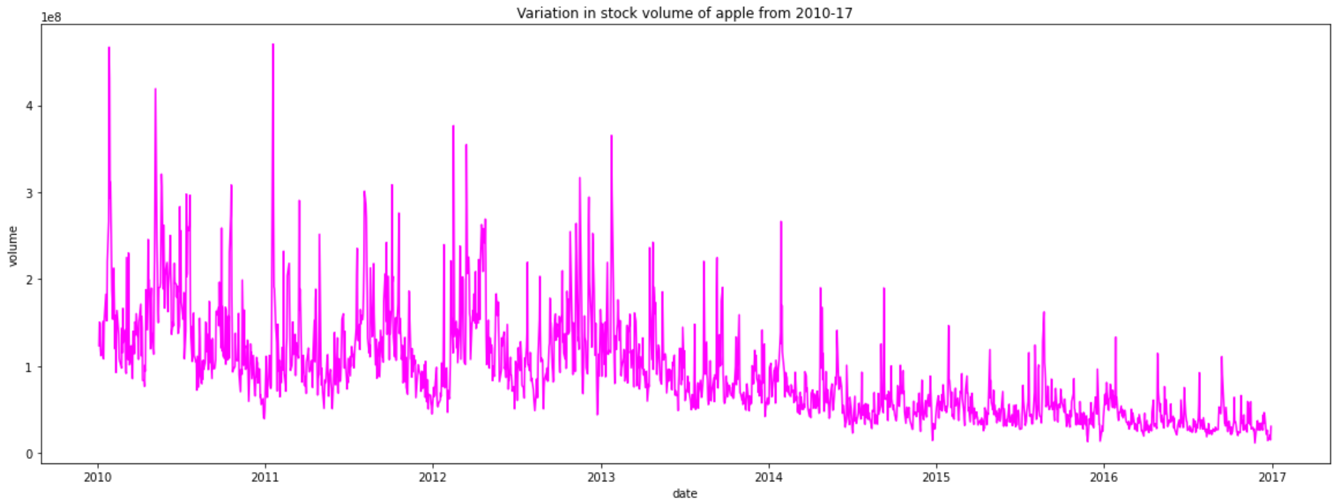
Visualizing the stock volume

Volume is counted as the total number of shares that are actually traded bought and sold) during the trading day or specified set period of time. It is a measure of the total turnover of shares. Each ticket represents a trade and counted towards the total trading volume. While the same shares may be traded back and forth multiple times, the volume is counted on each transaction.

Double-click (or enter) to edit

```
plt.figure(figsize=(20,7))
sns.lineplot(data=df1,x="date",y="volume",color="magenta").set_title("Variation in stock volu
```

```
Text(0.5, 1.0, 'Variation in stock volume of apple from 2010-17')
```

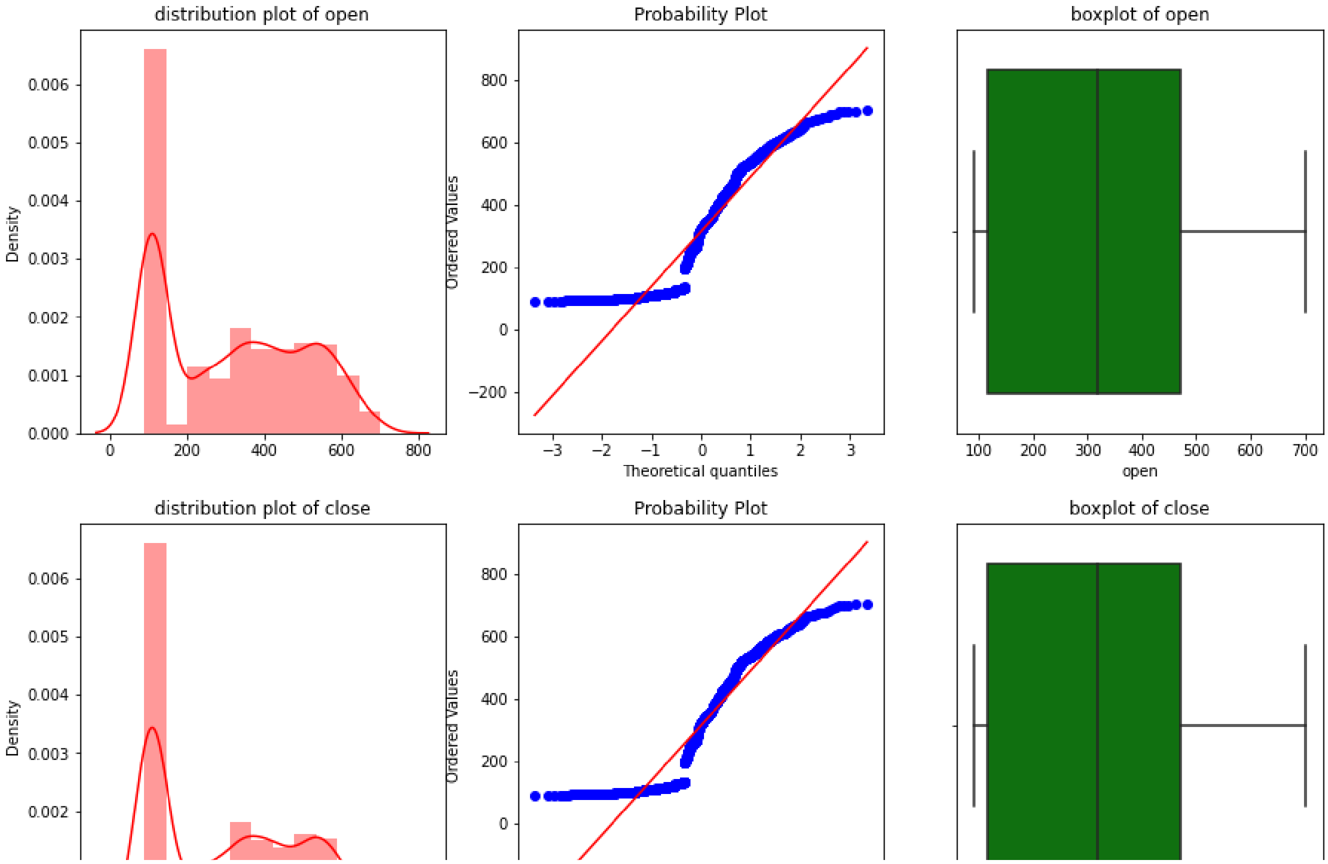


Statistical analysis of the numerical features w.r.t the apple stock data

```
# defining a function to analyse the numerical features statistically
def feature_stats(df,feature):
    #creating the subplots
    fig,ax=plt.subplots(1,3,figsize=(15,5))
    #adding the distribution plot
    sns.distplot(x=df[feature],kde=True,ax=ax[0],color="r").set_title("distribution plot of '
```

```
    sns.distplot(x=df[feature],kde=True,ax=ax[0],color="r").set_title("distribution plot of
    #adding the probability plot
    stats.probplot(x=df[feature],plot=ax[1])
    #adding boxplot
    sns.boxplot(x=df[feature],ax=ax[2],color="g").set_title("boxplot of "+feature)
```

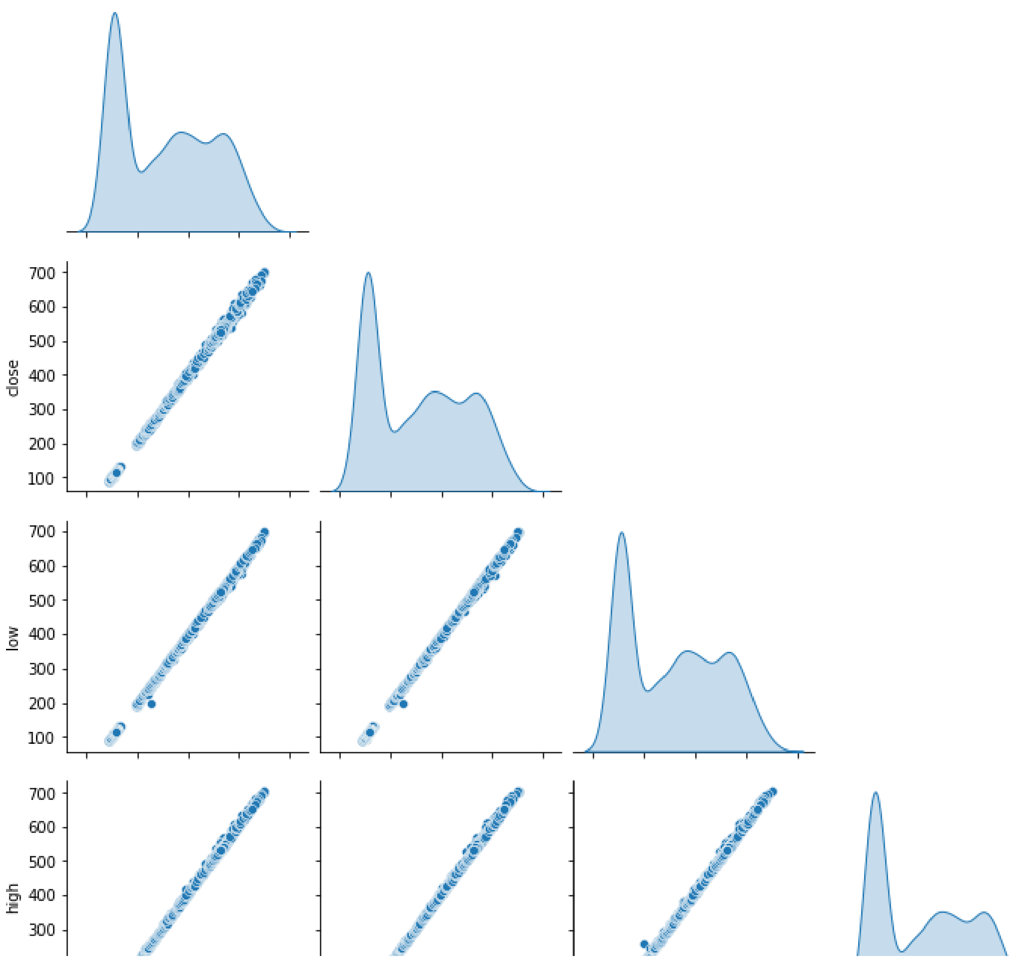
```
# dropping the non-numerical columns
df2=df1.drop("date",axis=1)
df2=df2.drop("symbol",axis=1)
# a for loop to apply the drfined function over all the features in df2
for i in df2.columns:
    feature_stats(df2,i)
    plt.show()
```



Checking the correlation between the variables


```
#generating pairplots betweeen the features to take a birds eye view
sns.pairplot(df2,corner=True,diag_kind="kde")
```

```
<seaborn.axisgrid.PairGrid at 0x7f82d87a7710>
```



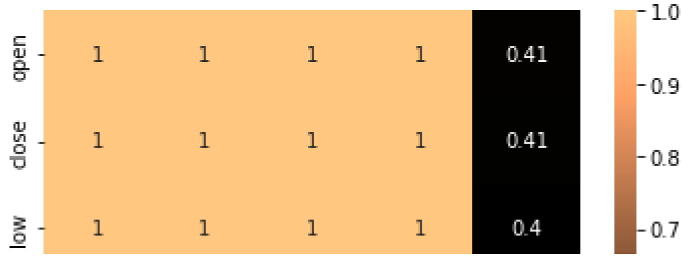
now we want "close" as closing price to be our target variable so lets check our correlation of the other features with the target variable

```
#correlation with features
df2.corr()["close"]
```

```
open      0.999650
close     1.000000
low       0.999834
high      0.999852
volume    0.408547
Name: close, dtype: float64
```

```
#checking correlation among all the variables
sns.heatmap(df2.corr(),annot=True,cmap="copper")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f82d7e73410>
```



In sequential models we use a different approach to create the dataset to feed into LSTM model

Splitting the data

```
#splitting the data into input and target variables
x=df2[["open","volume","high","close"]]
y=df2["close"]
#splitting the data into train set and test set
training_set=x.iloc[:1300].values
test_set=x.iloc[1300:].values
```

Normalization

Double-click (or enter) to edit

```
#scaling the input features with the help of standard scaler
from sklearn.preprocessing import StandardScaler

sc=StandardScaler()
training_set_scaled=sc.fit_transform(training_set)
test_set_scaled=sc.fit_transform(test_set)
```

Creating the input for our model

making a data structure with 60 time steps and one output

```
# the model will look into past 100 timesteps predict the next feature and will continue to c
length=100
#the training set creation
x_train=[]
y_train=[]
for i in range(length,len(training_set)):
    x_train.append(training_set_scaled[i-length:i,0])
    y_train.append(training_set_scaled[i,0])
```

```
#converting train set into array
x_train,y_train=np.array(x_train),np.array(y_train)
x_train=np.reshape(x_train,(x_train.shape[0],x_train.shape[1],1))


# The testing set creation
x_test=[]
y_test=[]
for i in range(length,len(test_set)):
    x_test.append(test_set_scaled[i-length:i,0])
    y_test.append(test_set_scaled[i,0])
# converting test set into array
x_test,y_test=np.array(x_test),np.array(y_test)
x_test=np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))


#checking the shape of the created datastructures
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

```
((1200, 100, 1), (1200,), (362, 100, 1), (362,))
```

Modelling

Fitting the data into a lstm model

The LSTM model will learn a function that maps a sequence of past observations as input to an output observation. ... We can divide the sequence into multiple input/output patterns called samples, where three time steps are used as input and one time step is used as output for the one-step prediction that is being learned

```
model=Sequential()
#adding the first layer with dropout regularization
model.add(LSTM(units=50,return_sequences=True,input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))
#adding second layer to the lstm model with dropout regularization
#model.add(LSTM(units=50,return_sequences=True))
#model.add(Dropout(0.2))
#adding the third layer with dropout regularization
model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.2))
#aedding the 4th layer alongwith dropout regularization
```

```
model.add(LSTM(units=50))
model.add(Dropout(0.2))
#adding the output layer
model.add(Dense(units=1))
#compiling the recurrent neural network model
model.compile(loss="mean_squared_error",optimizer="adam")
#fitting the model to the training set
model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=100,batch_size=32)
```

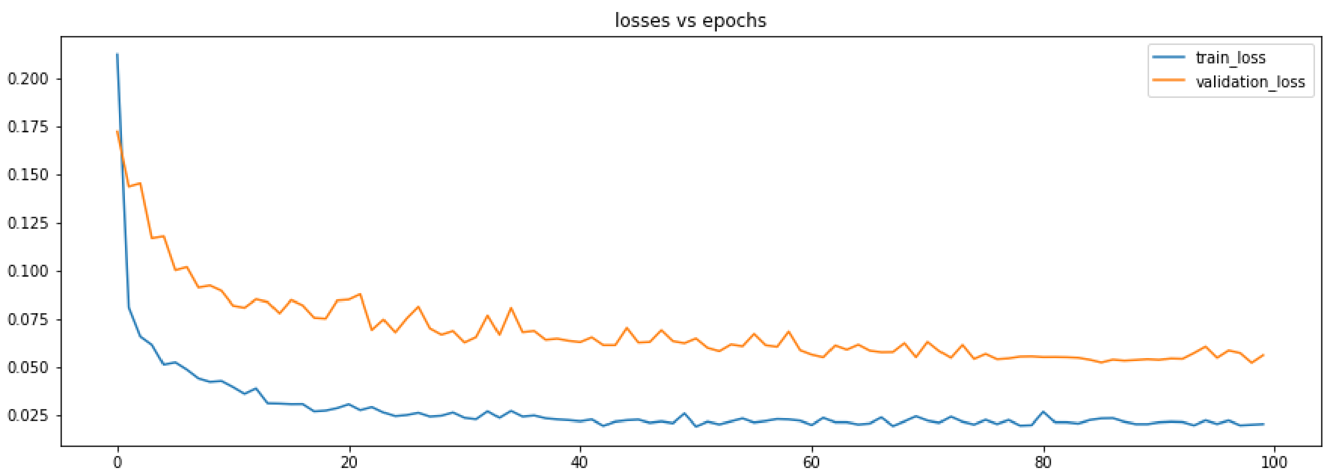
```
Epoch 1/100
38/38 [==============================] - 13s 189ms/step - loss: 0.2119 - val_loss: 0.
Epoch 2/100
38/38 [==============================] - 6s 154ms/step - loss: 0.0808 - val_loss: 0.1
Epoch 3/100
38/38 [==============================] - 6s 153ms/step - loss: 0.0657 - val_loss: 0.1
Epoch 4/100
38/38 [==============================] - 6s 153ms/step - loss: 0.0613 - val_loss: 0.1
Epoch 5/100
38/38 [==============================] - 6s 153ms/step - loss: 0.0510 - val_loss: 0.1
Epoch 6/100
38/38 [==============================] - 6s 151ms/step - loss: 0.0522 - val_loss: 0.1
Epoch 7/100
38/38 [==============================] - 6s 153ms/step - loss: 0.0484 - val_loss: 0.1
Epoch 8/100
38/38 [==============================] - 6s 154ms/step - loss: 0.0438 - val_loss: 0.0
Epoch 9/100
38/38 [==============================] - 6s 156ms/step - loss: 0.0420 - val_loss: 0.0
Epoch 10/100
38/38 [==============================] - 6s 151ms/step - loss: 0.0425 - val_loss: 0.0
Epoch 11/100
38/38 [==============================] - 6s 151ms/step - loss: 0.0393 - val_loss: 0.0
Epoch 12/100
38/38 [==============================] - 6s 152ms/step - loss: 0.0358 - val_loss: 0.0
Epoch 13/100
38/38 [==============================] - 6s 152ms/step - loss: 0.0386 - val_loss: 0.0
Epoch 14/100
38/38 [==============================] - 6s 152ms/step - loss: 0.0309 - val_loss: 0.0
Epoch 15/100
38/38 [==============================] - 6s 151ms/step - loss: 0.0307 - val_loss: 0.0
Epoch 16/100
38/38 [==============================] - 6s 153ms/step - loss: 0.0304 - val_loss: 0.0
Epoch 17/100
38/38 [==============================] - 6s 153ms/step - loss: 0.0305 - val_loss: 0.0
Epoch 18/100
38/38 [==============================] - 6s 155ms/step - loss: 0.0267 - val_loss: 0.0
Epoch 19/100
38/38 [==============================] - 6s 156ms/step - loss: 0.0271 - val_loss: 0.0
Epoch 20/100
38/38 [==============================] - 6s 155ms/step - loss: 0.0284 - val_loss: 0.0
Epoch 21/100
38/38 [==============================] - 6s 154ms/step - loss: 0.0304 - val_loss: 0.0
Epoch 22/100
38/38 [==============================] - 6s 157ms/step - loss: 0.0273 - val_loss: 0.0
Epoch 23/100
38/38 [==============================] - 6s 157ms/step - loss: 0.0289 - val_loss: 0.0
```

```
Epoch 24/100
38/38 [==============================] - 6s 155ms/step - loss: 0.0261 - val_loss: 0.0
Epoch 25/100
38/38 [==============================] - 6s 157ms/step - loss: 0.0242 - val_loss: 0.0
Epoch 26/100
38/38 [==============================] - 6s 152ms/step - loss: 0.0248 - val_loss: 0.0
Epoch 27/100
38/38 [==============================] - 6s 153ms/step - loss: 0.0259 - val_loss: 0.0
Epoch 28/100
38/38 [==============================] - 6s 155ms/step - loss: 0.0240 - val_loss: 0.0
Epoch 29/100
```

Plotting the validation loss vs epochs

```
#creating the dataframe contaning the validation loss and  train_inver_transform
loss=pd.DataFrame(model.history.history)
plt.figure(figsize=(15,5))
plt.plot(loss)
plt.title("losses vs epochs")
plt.legend(["train_loss","validation_loss"])
```

```
<matplotlib.legend.Legend at 0x7f82d3d60f90>
```



predicting the stock prices

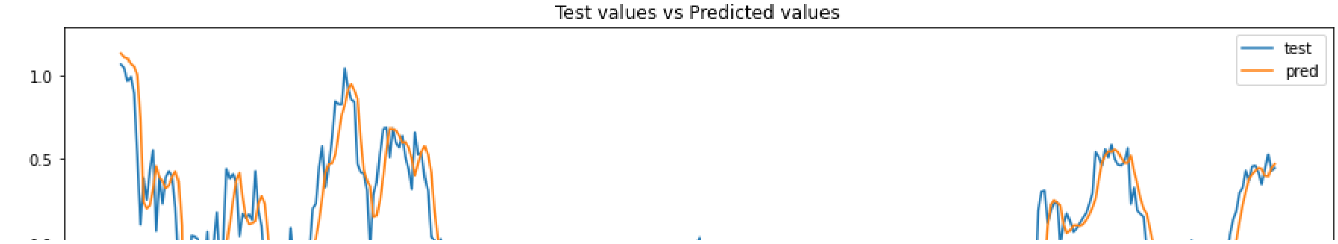
```
#creating a dataframe that consists of the test data and predicted data side by side in two c
pred=model.predict(x_test)
test=pd.DataFrame(columns=["test","pred"])
test["test"]=y_test
test["pred"]=pred.flatten()
#checking the dataframne created
test.head(10)
```

	test	pred
0	1.060551	1.127028
1	1.039555	1.104060
2	0.963786	1.097784
3	0.989346	1.063792
4	0.888930	1.048402
5	0.516474	0.998230
6	0.108417	0.739020
7	0.384107	0.244205
8	0.257217	0.204053
9	0.435228	0.224250

Plotting the test values vs the LSTM predicted values

```
from sklearn.metrics import mean_squared_error,r2_score
#the lineplot
plt.figure(figsize=(15,7))
plt.plot(test)
plt.title("Test values vs Predicted values")
plt.legend(["test","pred"])
#calculating the losses
r2=np.round(r2_score(y_test,pred),2)
mse=np.round(mean_squared_error(y_test,pred),2)
#incorporating the losses in the plot
plt.text(x=320,y=-1.0,s="R2 score:{}".format(r2))
plt.text(x=320,y=-1.25,s="MSE:{}".format(mse))
```

```
Text(320, -1.25, 'MSE:0.06')
```



the model is clearly overfitting and as a consequence the accuracy of the model gets affected,since we can't provide more company data,we take our chances and reduce the number of layers in the lstm model and check if th e model performace increases

going for further accuracy enhancements

```
model=Sequential()
#adding the first layer with dropout regularization
model.add(LSTM(units=50,return_sequences=True,input_shape = (x_train.shape[1], 1)))
model.add(Dropout(0.2))
#adding the next layer(2nd layer)
model.add(LSTM(units=50))
model.add(Dropout(0.2))
#adding the output layer
model.add(Dense(units=1))
#compiling the recurrent neural network model
model.compile(loss="mean_squared_error",optimizer="adam")
#fitting the model to the training set
model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=100,batch_size=30)
```

```
Epoch 1/100
40/40 [==============================] - 7s 100ms/step - loss: 0.2004 - val_loss: 0.1
Epoch 2/100
40/40 [==============================] - 3s 79ms/step - loss: 0.0608 - val_loss: 0.10
Epoch 3/100
40/40 [==============================] - 3s 78ms/step - loss: 0.0537 - val_loss: 0.09
Epoch 4/100
40/40 [==============================] - 3s 78ms/step - loss: 0.0487 - val_loss: 0.09
Epoch 5/100
40/40 [==============================] - 3s 77ms/step - loss: 0.0441 - val_loss: 0.08
Epoch 6/100
40/40 [==============================] - 3s 79ms/step - loss: 0.0396 - val_loss: 0.08
Epoch 7/100
40/40 [==============================] - 3s 79ms/step - loss: 0.0388 - val_loss: 0.07
Epoch 8/100
40/40 [==============================] - 3s 78ms/step - loss: 0.0405 - val_loss: 0.08
Epoch 9/100
40/40 [==============================] - 3s 78ms/step - loss: 0.0345 - val_loss: 0.08
Epoch 10/100
40/40 [==============================] - 3s 77ms/step - loss: 0.0353 - val_loss: 0.06
Epoch 11/100
40/40 [==============================] - 3s 77ms/step - loss: 0.0330 - val_loss: 0.06
```

```
Epoch 12/100
40/40 [==============================] - 3s 77ms/step - loss: 0.0313 - val_loss: 0.06
Epoch 13/100
40/40 [==============================] - 3s 78ms/step - loss: 0.0321 - val_loss: 0.06
Epoch 14/100
40/40 [==============================] - 3s 78ms/step - loss: 0.0310 - val_loss: 0.06
Epoch 15/100
40/40 [==============================] - 3s 77ms/step - loss: 0.0297 - val_loss: 0.06
Epoch 16/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0265 - val_loss: 0.06
Epoch 17/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0246 - val_loss: 0.06
Epoch 18/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0256 - val_loss: 0.05
Epoch 19/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0265 - val_loss: 0.05
Epoch 20/100
40/40 [==============================] - 3s 75ms/step - loss: 0.0231 - val_loss: 0.06
Epoch 21/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0251 - val_loss: 0.05
Epoch 22/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0262 - val_loss: 0.05
Epoch 23/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0269 - val_loss: 0.05
Epoch 24/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0263 - val_loss: 0.05
Epoch 25/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0216 - val_loss: 0.05
Epoch 26/100
40/40 [==============================] - 3s 78ms/step - loss: 0.0247 - val_loss: 0.05
Epoch 27/100
40/40 [==============================] - 3s 76ms/step - loss: 0.0232 - val_loss: 0.05
Epoch 28/100
40/40 [==============================] - 3s 77ms/step - loss: 0.0210 - val_loss: 0.05
```

```
#creating a dataframe that consists of the test data and predicted data side by side in two c
pred=model.predict(x_test)
test=pd.DataFrame(columns=["test","pred"])
test["test"]=y_test
test["pred"]=pred.flatten()
#the lineplot
plt.figure(figsize=(15,7))
plt.plot(test)
plt.title("Test values vs Predicted values")
plt.legend(["test","pred"])
#calculating the losses
r2=np.round(r2_score(y_test,pred),2)
mse=np.round(mean_squared_error(y_test,pred),2)
#incorporating the losses in the plot
plt.text(x=320,y=-1.0,s="R2 score:{}".format(r2))
plt.text(x=320,y=-1.25,s="MSE:{}".format(mse))
```

```
Text(320, -1.25, 'MSE:0.05')
```