Amazon Stock Prediction with Linear Regression, LSTM and Prophet

In [ ]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')


import os
for dirname, _, filenames in os.walk('/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Read Data

In [2]:
```
df=pd.read_csv('../input/nyse/prices.csv')
```

Describe the data

In [3]:
```
df.head()
```

Out[3]:

	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
3	2016-01-08 00:00:00	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11 00:00:00	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0

Check for Null Values

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
date      0
symbol    0
open      0
close     0
low       0
high      0
volume    0
dtype: int64
```

No Null values in the entire dataset

Check Datatype of the features

In [5]:
```
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
```

In [6]:
```
df.describe()
```

Out[6]:

	open	close	low	high	volume
count	851264.000000	851264.000000	851264.000000	851264.000000	8.512640e+05
mean	70.836986	70.857109	70.118414	71.543476	5.415113e+06
std	83.695876	83.689686	82.877294	84.465504	1.249468e+07
min	0.850000	0.860000	0.830000	0.880000	0.000000e+00
25%	33.840000	33.849998	33.480000	34.189999	1.221500e+06
50%	52.770000	52.799999	52.230000	53.310001	2.476250e+06
75%	79.879997	79.889999	79.110001	80.610001	5.222500e+06
max	1584.439941	1578.130005	1549.939941	1600.930054	8.596434e+08

Dataframe shape

In [7]:
```
df.shape
```

Out[7]:
```
(851264, 7)
```

In [8]:
```
df['symbol'].nunique()
```

Out[8]:
```
501
```

Extract AMAZON data from the dataset

In [9]:
```
df1=df[df['symbol']=='AMZN']
```

In [10]:
```
df1.head()
```

Out[10]:

	date	symbol	open	close	low	high	volume
284	2010-01-04	AMZN	136.250000	133.899994	133.139999	136.610001	7599900.0
751	2010-01-05	AMZN	133.429993	134.690002	131.809998	135.479996	8851900.0
1219	2010-01-06	AMZN	134.600006	132.250000	131.649994	134.729996	7178800.0
1687	2010-01-07	AMZN	132.009995	130.000000	128.800003	132.320007	11030200.0
2155	2010-01-08	AMZN	130.559998	133.520004	129.029999	133.679993	9830500.0

In [11]:
```
df1.describe()
```

Out[11]:

	open	close	low	high	volume
count	1762.000000	1762.000000	1762.000000	1762.000000	1.762000e+03
mean	337.875664	337.899058	333.969688	341.464438	4.607596e+06
std	189.294231	189.109339	187.654696	190.525796	3.091557e+06
min	105.930000	108.610001	105.800003	111.290001	9.844000e+05
25%	192.962494	193.377506	190.284997	195.532501	2.741550e+06
50%	282.500000	282.915008	279.869995	285.074997	3.890700e+06
75%	398.425003	398.014999	393.799988	402.082496	5.384450e+06
max	845.789978	844.359985	840.599976	847.210022	4.242110e+07

In [12]:
```
np.round(df1.median(),2)
```

Out[12]:
```
open          282.50
close         282.92
low           279.87
high          285.07
volume    3890700.00
dtype: float64
```

***All the Median of all the numeric values are lesser than mean, so the dataset is right skewed

In [13]:
```
df1['date']=pd.to_datetime(df1['date'])
```

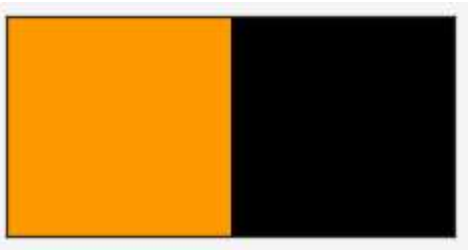
In [14]:
```
print("Minimum date value : {}".format(df1['date'].min()))
print("Maximum date value : {}".format(df1['date'].max()))
```

```
Minimum date value : 2010-01-04 00:00:00
Maximum date value : 2016-12-30 00:00:00
```

we using around 6 years of data

EDA

In [15]:
```
#importing ploting libraries
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
colors = ['#FF9900','#000000']
sns.set(palette=colors, font='Serif', style='white', rc={'axes.facecolor':'whitesmoke', 'figure.facecolor':'whitesmoke'
sns.palplot(colors, size=2)
```

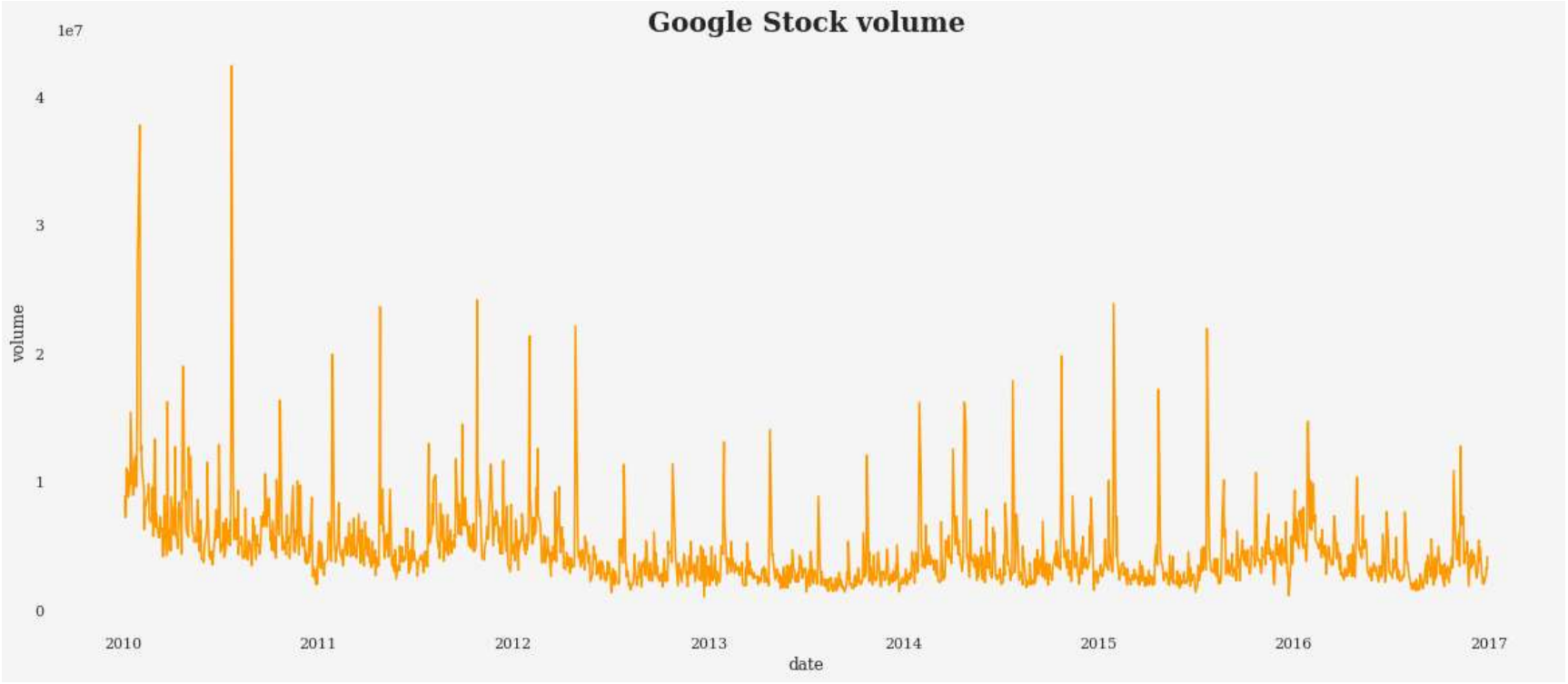


In [16]:
```
fig=plt.figure(figsize=(20,8))
ax=sns.lineplot(data=df1, x='date',y='open')
ax=sns.lineplot(data=df1, x='date',y='close', color=colors[1]);
for s in ['left','right','top','bottom']:
    ax.spines[s].set_visible(False)

plt.title("AMAZON Stock value changes since 2010", size=20, weight='bold')
```

Out[16]:
```
Text(0.5, 1.0, 'AMAZON Stock value changes since 2010')
```

In [17]:
```
fig=plt.figure(figsize=(20,8))
ax=sns.lineplot(data=df1, x='date',y='volume')
#ax=sns.lineplot(data=df1, x='date',y='close', color=colors[1]);
for s in ['left','right','top','bottom']:
    ax.spines[s].set_visible(False)
plt.title("Google Stock volume", size=20, weight='bold')
```

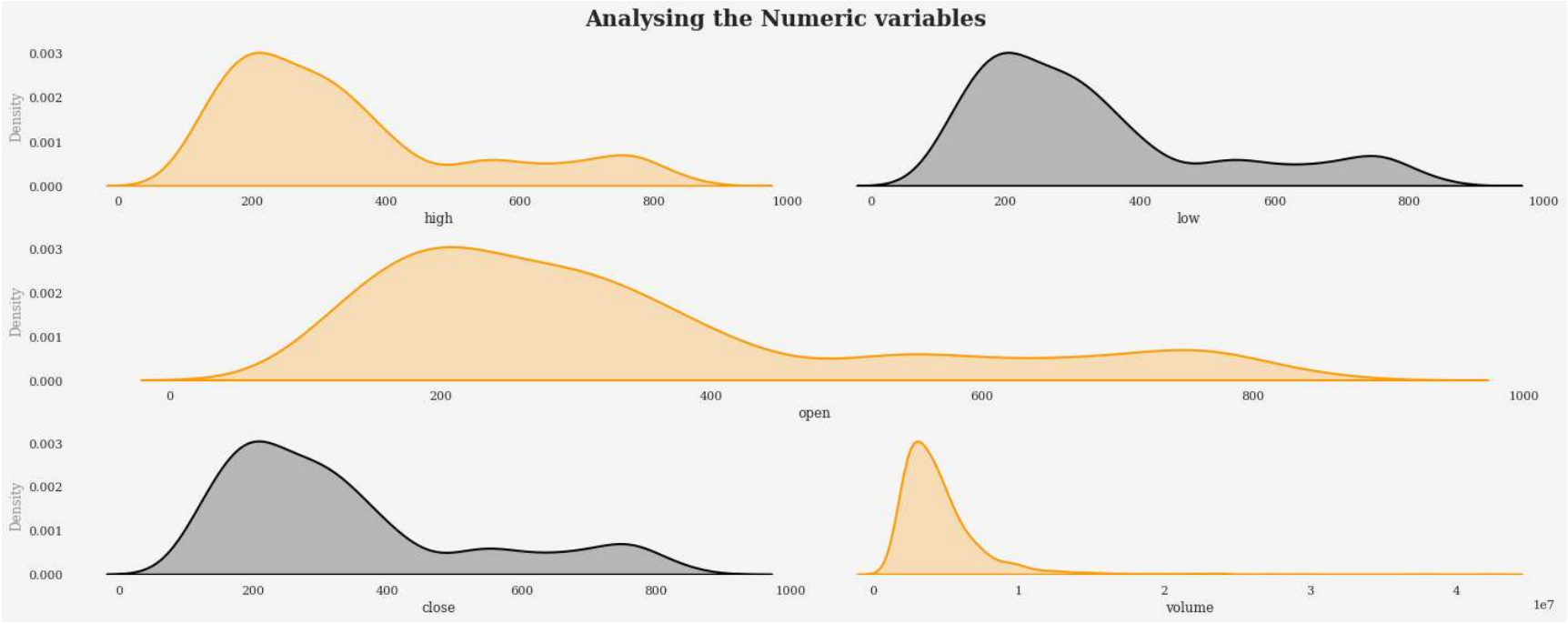
Out[17]:
```
Text(0.5, 1.0, 'Google Stock volume')
```



Univariated Analysis

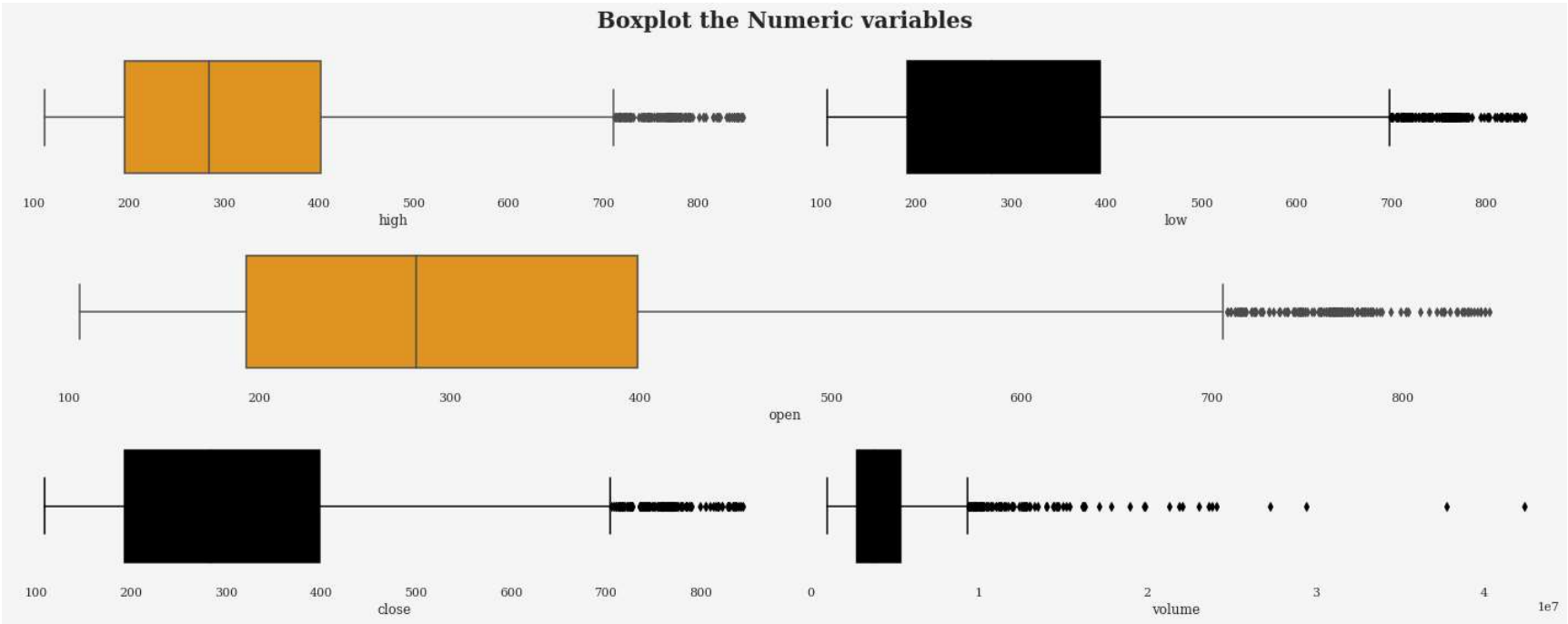
In [18]:
```
#integer columns
fig=plt.figure(figsize=(20,8), tight_layout=True)
plt.suptitle("Analysing the Numeric variables", size=20, weight='bold')
ax=fig.subplot_mosaic("""AB
                         CC
                         DE""")
sns.kdeplot(df1['high'], ax=ax['A'], color=colors[0], fill=True, linewidth=2)
sns.kdeplot(df1['low'], ax=ax['B'], color=colors[1],fill=True, linewidth=2)
sns.kdeplot(df1['open'], ax=ax['C'], color=colors[0],fill=True, linewidth=2)
sns.kdeplot(df1['close'], ax=ax['D'], color=colors[1],fill=True, linewidth=2)
sns.kdeplot(df1['volume'], ax=ax['E'], color=colors[0],fill=True, linewidth=2)
ax['B'].yaxis.set_visible(False)
ax['E'].yaxis.set_visible(False)
ax['A'].yaxis.label.set_alpha(0.5)
ax['C'].yaxis.label.set_alpha(0.5)
ax['A'].yaxis.label.set_alpha(0.5)
ax['C'].yaxis.label.set_alpha(0.5)
```

```
ax['D'].yaxis.label.set_alpha(0.5)
for s in ['left','right','top','bottom']:
    ax['A'].spines[s].set_visible(False)
    ax['B'].spines[s].set_visible(False)
    ax['C'].spines[s].set_visible(False)
    ax['D'].spines[s].set_visible(False)
    ax['E'].spines[s].set_visible(False)
```



In [19]:
```
#integer columns
fig=plt.figure(figsize=(20,8), tight_layout=True)
plt.suptitle("Boxplot the Numeric variables", size=20, weight='bold')
ax=fig.subplot_mosaic("""AB
                         CC
                         DE""")
sns.boxplot(df1['high'], ax=ax['A'], color=colors[0])
sns.boxplot(df1['low'], ax=ax['B'], color=colors[1])
sns.boxplot(df1['open'], ax=ax['C'], color=colors[0])
sns.boxplot(df1['close'], ax=ax['D'], color=colors[1])
sns.boxplot(df1['volume'], ax=ax['E'], color=colors[1])
ax['B'].yaxis.set_visible(False)
ax['E'].yaxis.set_visible(False)
ax['A'].yaxis.label.set_alpha(0.5)
ax['C'].yaxis.label.set_alpha(0.5)
```

```
ax['A'].yaxis.label.set_alpha(0.5)
ax['C'].yaxis.label.set_alpha(0.5)
ax['D'].yaxis.label.set_alpha(0.5)
for s in ['left','right','top','bottom']:
    ax['A'].spines[s].set_visible(False)
    ax['B'].spines[s].set_visible(False)
    ax['C'].spines[s].set_visible(False)
    ax['D'].spines[s].set_visible(False)
    ax['E'].spines[s].set_visible(False)
```



There seems to be ouliers in the data set, however, those values can't be considered as outlier as they may be extrem values during peak selling days
Date column has been ingored as it is series of numbers

Categorical feature analysis

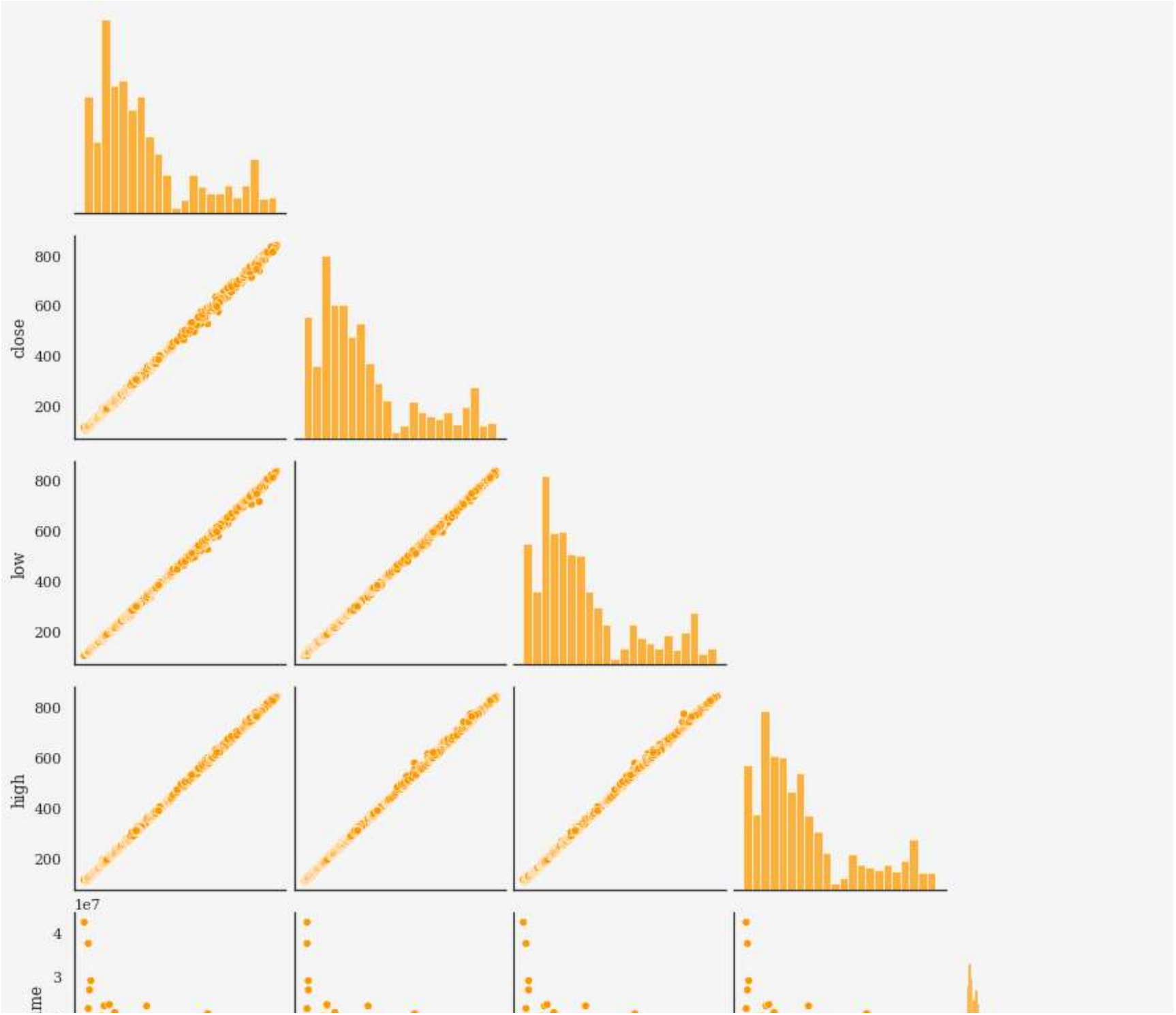
** There is no categorical feature other than Symbol. as we have taken the Google stock as sample data, we will drop the Symbol feature.

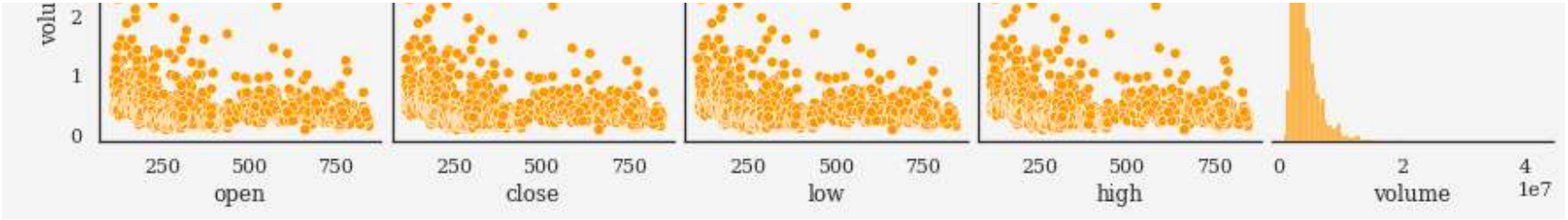
In [20]:
```
df1.drop(['symbol'], axis=1, inplace=True)
```

Bivariated & Multivariated Analysis

In [21]:
```
# we need to predict the closing price of the stock, lets us consider 'Close' feature as the Target variable.
sns.pairplot(df1,corner=True)
```

Out[21]:
```
<seaborn.axisgrid.PairGrid at 0x79b7d80bc690>
```


close
low
high
1e7
800
600
400
200
4
3



In [22]:
```
df1.corr()['close']
```

Out[22]:
```
open      0.999581
close     1.000000
low       0.999832
high      0.999811
volume   -0.238560
Name: close, dtype: float64
```

Hypothesis test to find the Normality in the Dataset

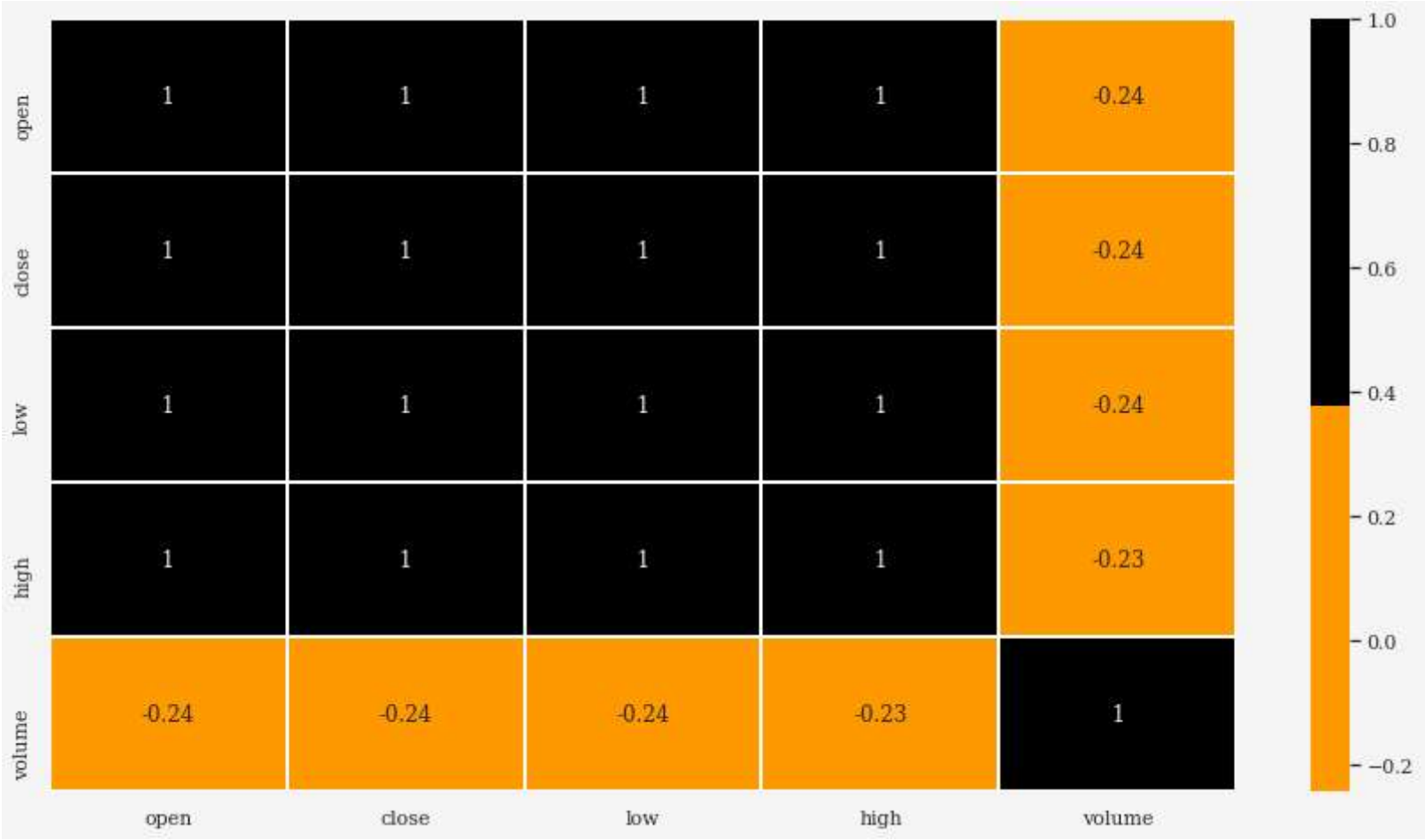
In [23]:
```
from scipy.stats import levene, shapiro
int_cols=df1.select_dtypes(exclude='object').columns.to_list()

for i in int_cols:
    _, p_value=shapiro(df1[i])
    if p_value<0.05:
        print("Feature {} is normaly distributed".format(i))
    else:
        print("Feature {} is not normaly distributed".format(i))

    print("Normalitiy test p_value for featue -  {} is {}".format(i,np.round(p_value,3)))
```

```
Feature date is normaly distributed
Normalitiy test p_value for featue -  date is 0.0
Feature open is normaly distributed
Normalitiy test p_value for featue -  open is 0.0
Feature close is normaly distributed
Normalitiy test p_value for featue -  close is 0.0
Feature low is normaly distributed
Normalitiy test p_value for featue -  low is 0.0
Feature high is normaly distributed
Normalitiy test p_value for featue -  high is 0.0
Feature volume is normaly distributed
Normalitiy test p_value for featue -  volume is 0.0
```

Correlation

In [24]:
```
fig=plt.figure(figsize=(15,8))
sns.heatmap(df1.corr(), annot=True, cmap=[colors[0],colors[1]], linecolor='white', linewidth=2 )
```

Out[24]:
```
<AxesSubplot:>
```



The feature Open, High, Low are highly correlated to Target feature Close. we can use either one of the feature for prediction to avoid multicollinearity

Train Test Split

In [25]:
```
X=df1[['volume','open']]
y=df1['close']
```

In [26]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, shuffle=False, random_state=42)
```

Normalizing the values

In [27]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test =scaler.transform(X_test)
```

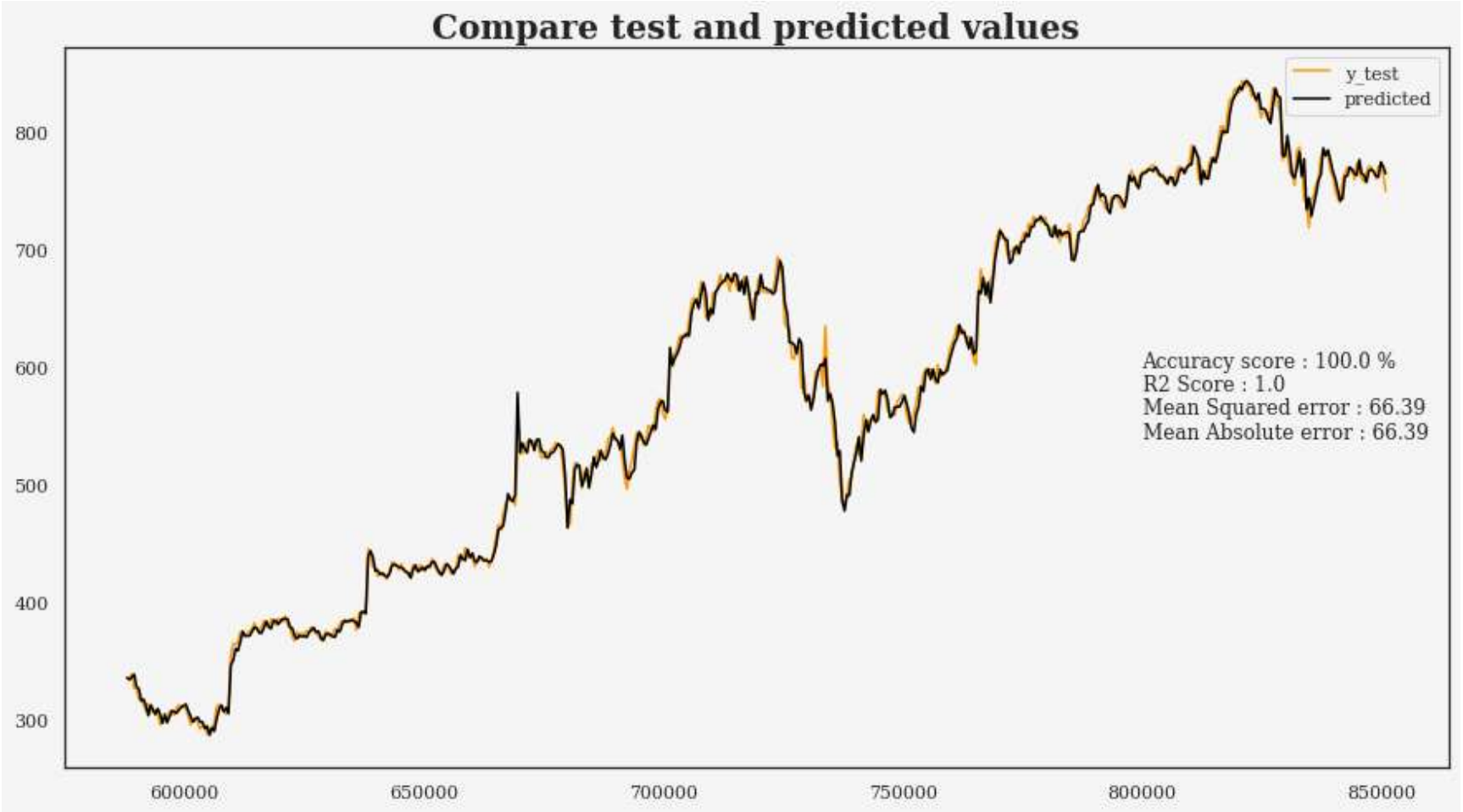
Model Creation

Linear Regression Model

In [28]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn import set_config
model = LinearRegression()
model.fit(X_train,y_train)
set_config(display='diagram')
pred=model.predict(X_test)
sc=np.round(model.score(X_test, y_test),2) * 100
r2=np.round(r2_score(y_test,pred),2)
mse=np.round(mean_squared_error(y_test,pred),2)
mae=np.round(mean_squared_error(y_test,pred),2)
```

In [29]:
```
fig=plt.figure(figsize=(15,8))
p=pd.Series(pred, index=y_test.index)
plt.plot(y_test)
plt.plot(p)
plt.legend(['y_test','predicted'])
plt.title("Compare test and predicted values", size=20, weight='bold')
plt.text(x=800000, y=600,s='Accuracy score : {} %'.format(sc))
plt.text(x=800000, y=580,s='R2 Score : {}'.format(r2))
plt.text(x=800000, y=560,s='Mean Squared error : {}'.format(mse))
plt.text(x=800000, y=540,s='Mean Absolute error : {}'.format(mae))
```

Out[29]:
```
Text(800000, 540, 'Mean Absolute error : 66.39')
```



LSTM Model

In [30]:
```
X=df1[['open','high']]
y=df1['close']
length=100
#from sklearn.model_selection import train_test_split
#X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, shuffle=False, random_state=42)
training_set = X.iloc[:1000].values
test_set = X.iloc[1000:].values
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
```

```
training_set_scaled = sc.fit_transform(training_set)
test_set_scaled=sc.transform(test_set)
# Creating a data structure with 60 time-steps and 1 output
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])
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))

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])
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))
```

In [31]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
model = Sequential()
#Adding the first LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.2))
# Adding a second LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
# Adding a third LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
# Adding a fourth LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
# Adding the output layer
model.add(Dense(units = 1))

# Compiling the RNN
model.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
model.fit(X_train, y_train, validation_data=(X_test,y_test),epochs = 100, batch_size = 32)
```

```
Epoch 1/100
29/29 [==============================] - 9s 74ms/step - loss: 0.0613 - val_loss: 0.4825
Epoch 2/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0080 - val_loss: 0.1162
Epoch 3/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0040 - val_loss: 0.1090
Epoch 4/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0040 - val_loss: 0.0377
Epoch 5/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0039 - val_loss: 0.0403
Epoch 6/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0034 - val_loss: 0.0694
Epoch 7/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0037 - val_loss: 0.0645
Epoch 8/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0039 - val_loss: 0.0570
Epoch 9/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0029 - val_loss: 0.0596
Epoch 10/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0036 - val_loss: 0.0580
Epoch 11/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0029 - val_loss: 0.0228
Epoch 12/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0032 - val_loss: 0.0442
Epoch 13/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0035 - val_loss: 0.0441
Epoch 14/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0027 - val_loss: 0.0700
Epoch 15/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0030 - val_loss: 0.0410
Epoch 16/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0027 - val_loss: 0.0328
Epoch 17/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0029 - val_loss: 0.0363
Epoch 18/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0025 - val_loss: 0.0281
Epoch 19/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0027 - val_loss: 0.0161
Epoch 20/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0027 - val_loss: 0.0171
Epoch 21/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0024 - val_loss: 0.0224
Epoch 22/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0023 - val_loss: 0.0156
Epoch 23/100
```

```
29/29 [==============================] - 1s 24ms/step - loss: 0.0027 - val_loss: 0.0396
Epoch 24/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0026 - val_loss: 0.0277
Epoch 25/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0025 - val_loss: 0.0238
Epoch 26/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0024 - val_loss: 0.0129
Epoch 27/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0025 - val_loss: 0.0124
Epoch 28/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0023 - val_loss: 0.0148
Epoch 29/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0025 - val_loss: 0.0272
Epoch 30/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0020 - val_loss: 0.0224
Epoch 31/100
29/29 [==============================] - 1s 25ms/step - loss: 0.0018 - val_loss: 0.0312
Epoch 32/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0021 - val_loss: 0.0132
Epoch 33/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0028 - val_loss: 0.0088
Epoch 34/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0022 - val_loss: 0.0130
Epoch 35/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0023 - val_loss: 0.0096
Epoch 36/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0022 - val_loss: 0.0094
Epoch 37/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0021 - val_loss: 0.0174
Epoch 38/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0019 - val_loss: 0.0165
Epoch 39/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0022 - val_loss: 0.0168
Epoch 40/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0020 - val_loss: 0.0106
Epoch 41/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0022 - val_loss: 0.0108
Epoch 42/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0020 - val_loss: 0.0142
Epoch 43/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0019 - val_loss: 0.0127
Epoch 44/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0019 - val_loss: 0.0104
Epoch 45/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0020 - val_loss: 0.0077
```

```
Epoch 46/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0018 - val_loss: 0.0184
Epoch 47/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0019 - val_loss: 0.0126
Epoch 48/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0016 - val_loss: 0.0089
Epoch 49/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0016 - val_loss: 0.0143
Epoch 50/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0019 - val_loss: 0.0098
Epoch 51/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0017 - val_loss: 0.0266
Epoch 52/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0018 - val_loss: 0.0138
Epoch 53/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0018 - val_loss: 0.0102
Epoch 54/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0017 - val_loss: 0.0077
Epoch 55/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0028 - val_loss: 0.0140
Epoch 56/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0015 - val_loss: 0.0180
Epoch 57/100
29/29 [==============================] - 1s 25ms/step - loss: 0.0016 - val_loss: 0.0124
Epoch 58/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0067
Epoch 59/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0015 - val_loss: 0.0100
Epoch 60/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0013 - val_loss: 0.0135
Epoch 61/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0015 - val_loss: 0.0087
Epoch 62/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0014 - val_loss: 0.0065
Epoch 63/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0015 - val_loss: 0.0175
Epoch 64/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0015 - val_loss: 0.0137
Epoch 65/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0014 - val_loss: 0.0098
Epoch 66/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0119
Epoch 67/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0015 - val_loss: 0.0072
Epoch 68/100
```

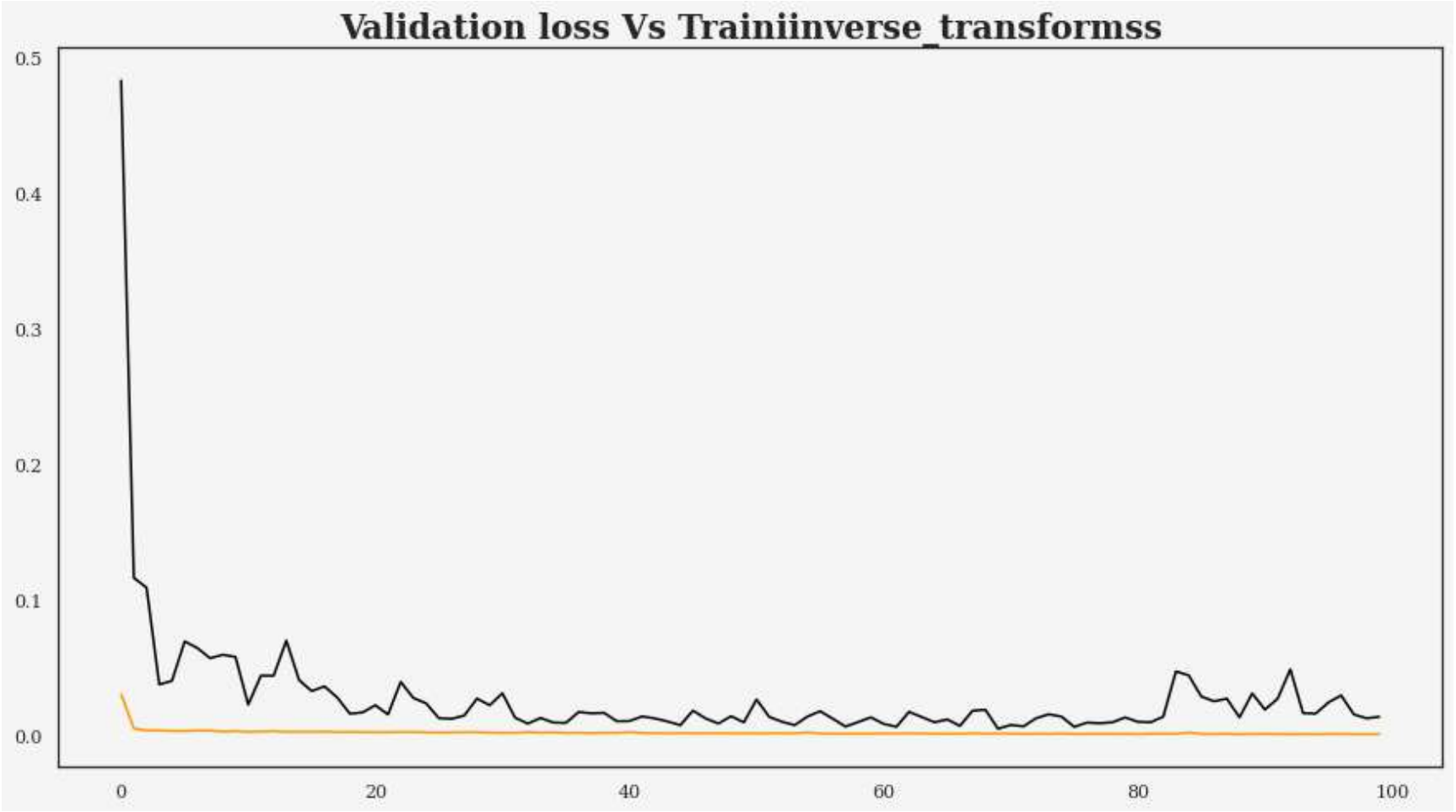
```
29/29 [==============================] - 1s 24ms/step - loss: 0.0018 - val_loss: 0.0184
Epoch 69/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0015 - val_loss: 0.0191
Epoch 70/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0014 - val_loss: 0.0051
Epoch 71/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0015 - val_loss: 0.0078
Epoch 72/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0068
Epoch 73/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0016 - val_loss: 0.0128
Epoch 74/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0158
Epoch 75/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0015 - val_loss: 0.0140
Epoch 76/100
29/29 [==============================] - 1s 25ms/step - loss: 0.0013 - val_loss: 0.0064
Epoch 77/100
29/29 [==============================] - 1s 25ms/step - loss: 0.0012 - val_loss: 0.0096
Epoch 78/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0013 - val_loss: 0.0091
Epoch 79/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0098
Epoch 80/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0134
Epoch 81/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0011 - val_loss: 0.0102
Epoch 82/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0098
Epoch 83/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0013 - val_loss: 0.0140
Epoch 84/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0473
Epoch 85/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0027 - val_loss: 0.0444
Epoch 86/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0014 - val_loss: 0.0288
Epoch 87/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0011 - val_loss: 0.0253
Epoch 88/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0013 - val_loss: 0.0272
Epoch 89/100
29/29 [==============================] - 1s 24ms/step - loss: 9.8388e-04 - val_loss: 0.0134
Epoch 90/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0011 - val_loss: 0.0311
```

```
Epoch 91/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0014 - val_loss: 0.0192
Epoch 92/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0010 - val_loss: 0.0272
Epoch 93/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0013 - val_loss: 0.0489
Epoch 94/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0166
Epoch 95/100
29/29 [==============================] - 1s 24ms/step - loss: 9.8453e-04 - val_loss: 0.0162
Epoch 96/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0012 - val_loss: 0.0245
Epoch 97/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0011 - val_loss: 0.0296
Epoch 98/100
29/29 [==============================] - 1s 24ms/step - loss: 9.5980e-04 - val_loss: 0.0156
Epoch 99/100
29/29 [==============================] - 1s 24ms/step - loss: 9.3465e-04 - val_loss: 0.0128
Epoch 100/100
29/29 [==============================] - 1s 24ms/step - loss: 0.0011 - val_loss: 0.0139
```

Out[31]:
```
<tensorflow.python.keras.callbacks.History at 0x79b7981594d0>
```

In [32]:
```
loss = pd.DataFrame(model.history.history)
fig=plt.figure(figsize=(15,8))
plt.title("Validation loss Vs Trainiinverse_transformss", size=20, weight='bold')
plt.plot(loss)
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x79b72819b590>,
 <matplotlib.lines.Line2D at 0x79b728134ad0>]
```



In [33]:
```
pred=model.predict(X_test)
test=pd.DataFrame(columns=['test','pred'])
test['test']=y_test
test['pred']=pred.flatten()
test
```

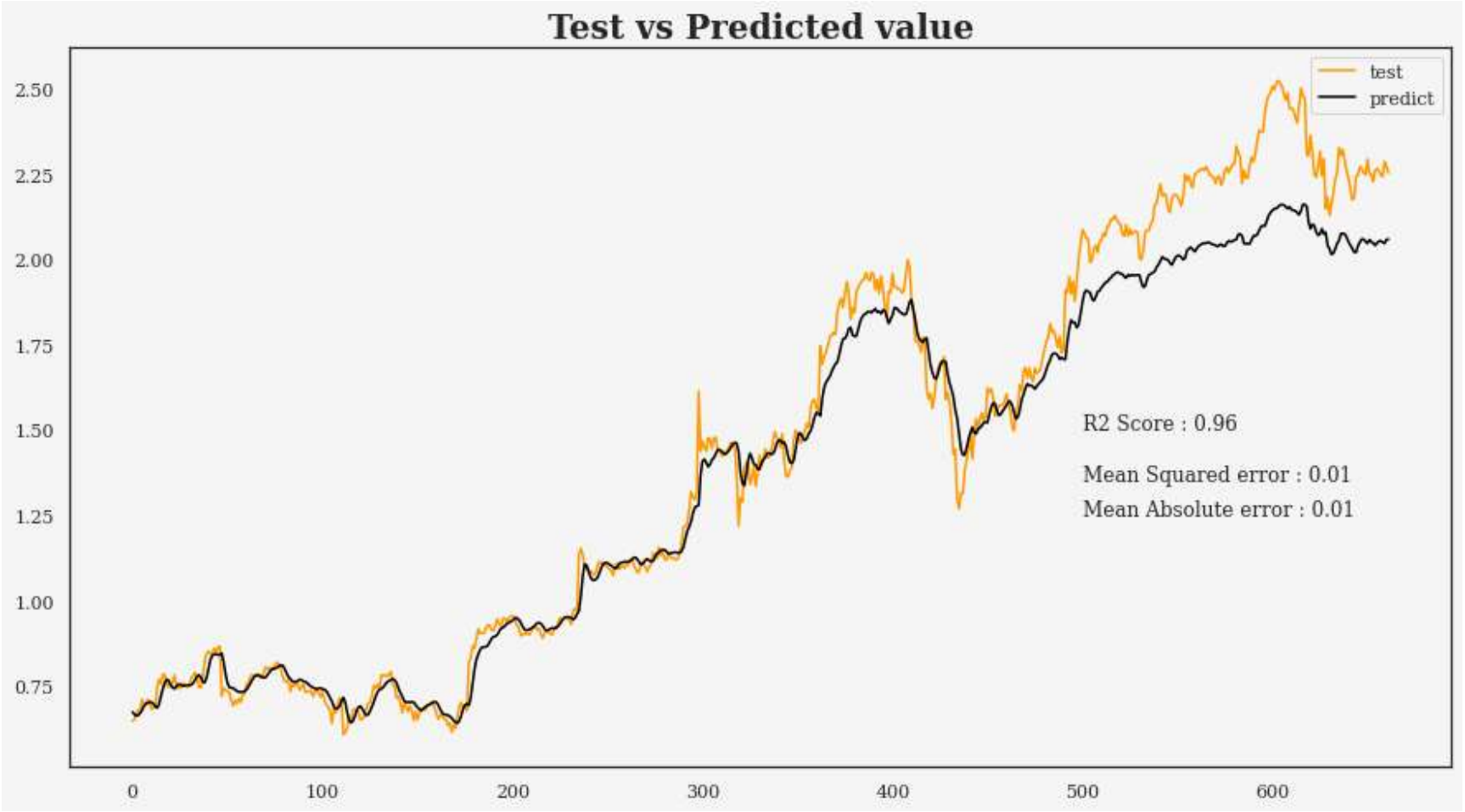
Out[33]:

	test	pred
0	0.647729	0.674112
1	0.652302	0.667890
2	0.669738	0.663177
3	0.679428	0.664529
4	0.680827	0.671369
...	...	...
657	2.247313	2.054365
658	2.243389	2.050452
659	2.287235	2.047239
660	2.274098	2.058147
661	2.253864	2.059852

662 rows × 2 columns

In [34]:
```
fig=plt.figure(figsize=(15,8))
plt.title("Test vs Predicted value", size=20, weight='bold')
plt.plot(test)
plt.legend(['test','predict'])
r2=np.round(r2_score(y_test,pred),2)
mse=np.round(mean_squared_error(y_test,pred),2)
mae=np.round(mean_squared_error(y_test,pred),2)
plt.text(x=500, y=1.5,s='R2 Score : {}'.format(r2))
plt.text(x=500, y=1.35,s='Mean Squared error : {}'.format(mse))
plt.text(x=500, y=1.25,s='Mean Absolute error : {}'.format(mae))
```

Out[34]:
```
Text(500, 1.25, 'Mean Absolute error : 0.01')
```



FaceBook Prophet Model

In [35]:

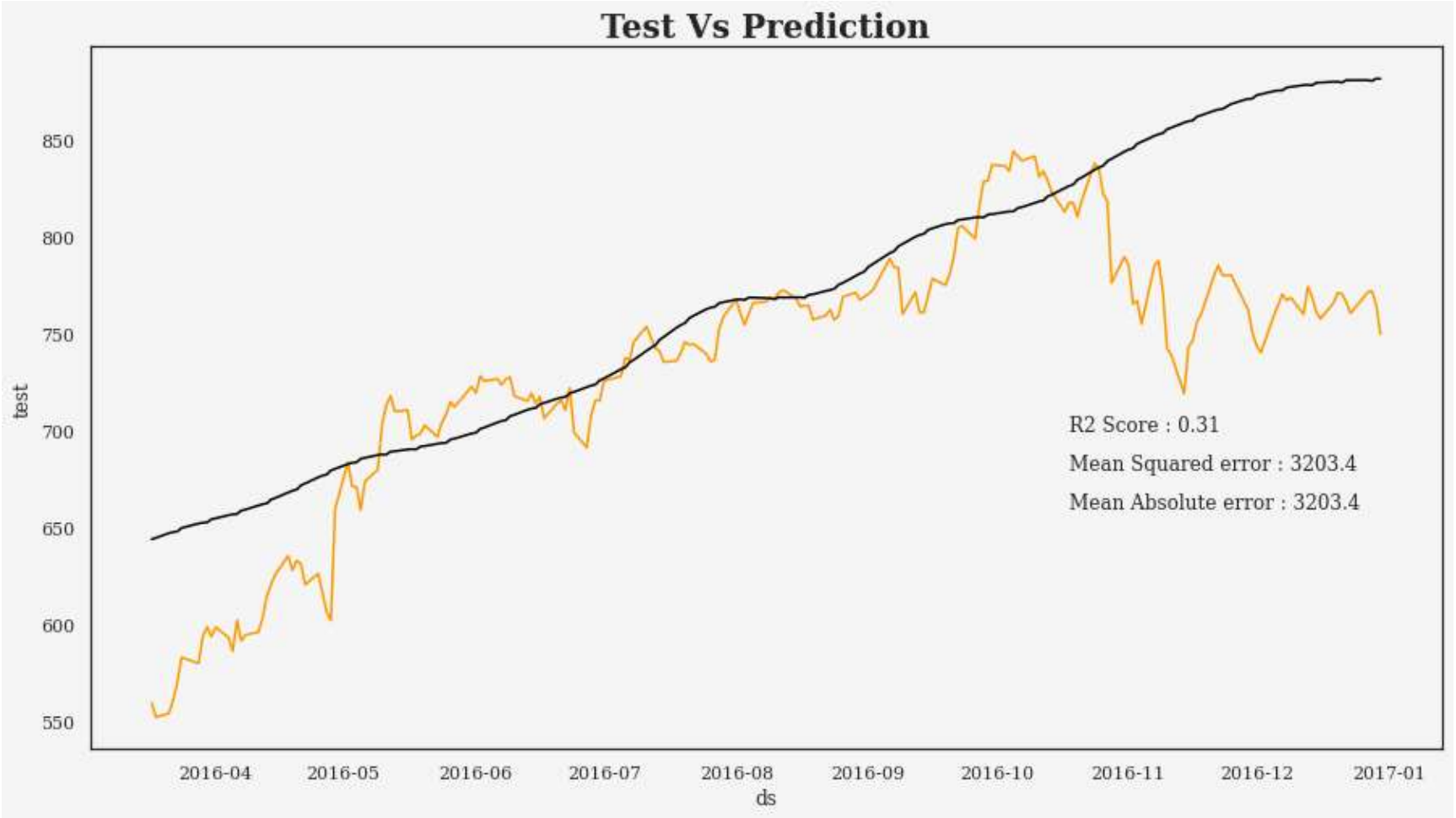
```
from fbprophet import Prophet
df_p = df1[['date','close']]
df_p.columns=['ds','y']

split_data = df_p.index.max()-100000
train = df_p.loc[df_p.index<=split_data].copy()
test=df_p.loc[df_p.index>split_data].copy()
train.set_index('ds',inplace=True)
test.set_index('ds',inplace=True)
train.reset_index(inplace=True)
```

```
test.reset_index(inplace=True)

#Model creation
model=Prophet()
model.fit(train)

#model prediction
pred=model.predict(test)
```

In [36]:
```
test_pred=pd.DataFrame(columns=['ds','test','predict','pred_lower','pred_high'], index=test.index)
test_pred['test']=test['y']
test_pred['ds']=test['ds']
test_pred['predict']=pred['yhat']
test_pred['pred_lower']=pred['yhat_lower']
test_pred['pred_high']=pred['yhat_upper']
```

In [37]:
```
#plotting Test vs Predicted
fig=plt.figure(figsize=(15,8))
plt.title("Test Vs Prediction", size=20, weight='bold')
sns.lineplot(data=test_pred,x='ds',y='test')
sns.lineplot(data=test_pred,x='ds',y='predict')

r2=np.round(r2_score(test_pred['test'],test_pred['predict']),2)
mse=np.round(mean_squared_error(test_pred['test'],test_pred['predict']),2)
mae=np.round(mean_squared_error(test_pred['test'],test_pred['predict']),2)
plt.text(x=mdates.datestr2num('2016-10'), y=700,s='R2 Score : {}'.format(r2))
plt.text(x=mdates.datestr2num('2016-10'), y=680,s='Mean Squared error : {}'.format(mse))
plt.text(x=mdates.datestr2num('2016-10'), y=660,s='Mean Absolute error : {}'.format(mae))
```

Out[37]:
```
Text(17092.0, 660, 'Mean Absolute error : 3203.4')
```



In [38]:

```
fig=plt.figure(figsize=(15,8))
plt.title("Test Vs Prediction", size=20, weight='bold')
sns.lineplot(data=train,x='ds',y='y')
sns.lineplot(data=test_pred,x='ds',y='predict')
sns.lineplot(data=test_pred,x='ds',y='test', alpha=0.5, ls='--', color='black')

r2=np.round(r2_score(test_pred['test'],test_pred['predict']),2)
mse=np.round(mean_squared_error(test_pred['test'],test_pred['predict']),2)
mae=np.round(mean_squared_error(test_pred['test'],test_pred['predict']),2)
plt.text(x=mdates.datestr2num('2015'), y=300,s='R2 Score : {}'.format(r2))
plt.text(x=mdates.datestr2num('2015'), y=260,s='Mean Squared error : {}'.format(mse))
plt.text(x=mdates.datestr2num('2015'), y=220,s='Mean Absolute error : {}'.format(mae))
plt.legend(['Train','Predict','Test'])
```

Out[38]: <matplotlib.legend.Legend at 0x79b30607c590>