Level A

Pick a dataset and objective

Dataset:

- Dataset of Nifty Stock prices of Indian companies. (https://www.kaggle.com/rohanrao/nifty50-stock-market-data)

Problem Statement:

- Creating a Predictive Model using any Algorithm (Deep Learning/Machine Learning) that can predict the stock price(Close column) of ASIANPAINTs.
- A prediction for the year of 2008 using all previous data.
- Show accuracy of the algorithms and explaining a choice of accuracy metric (RMSE/MAE/MAPE,R^2, Adjusted R^2).

This algorithm predits the Close column of the data set for 2008 year using 2000 - 2007 years data.

In [1]:

```
### Loading the libraries required

import pandas as pd        #Loading pandas for creating and adjusting dataframes
import numpy as np         # Loading Numpy for creating and adjusting arrays

import seaborn as sns              # Loading seaborn for visualizations
import matplotlib.pyplot as plt    # Loading matplotlib for visualizations
%matplotlib inline

from sklearn.preprocessing import StandardScaler    # Loading standard scaler to normalize the data

from sklearn.model_selection import GridSearchCV   # Loading gridsearch CV for hyperparameter tuning using Crss Validation
from sklearn.neighbors import KNeighborsRegressor  #Loading KNN regressor to cretae a model
from sklearn.ensemble import RandomForestRegressor #Loading Random Forest regressor to cretae a model
from sklearn.ensemble import AdaBoostRegressor     #Loading Ada Boost regressor to cretae a model
```

```
from tensorflow.keras.models import Sequential     #Loading Sequential model from tensor flow to craete a ANN model
from tensorflow.keras.layers import Dense          #Loading Dense layer from tensor flow to craete a Neural network layers
from tensorflow.keras.layers import Dropout        #Loading Drop out Layer

from tensorflow.keras.optimizers import Adam       #Loading Adam optimizer for NN

from sklearn.metrics import mean_squared_error     #Loading Mean Squared Error to to check the error metrics of each model
from sklearn.metrics import mean_absolute_percentage_error  #Loading Mean absolute percentage error to to check the error metrics
from sklearn.metrics import r2_score       #Loading R^2 (R squared) to check the efficiency metrics of each model
```

In [2]:

```
df = pd.read_csv("ASIANPAINT.csv")   #Loading the data set using read_csv to dataframe
```

In [3]:

```
# Adjusting the "Date" column to date time format and creating a Year column to split the data
df["Year"] = pd.to_datetime(df.Date, format="%d-%m-%Y").dt.year
```

In [4]:

```
df_2008 = df.loc[df["Year"] < 2009] # Splitting the Data to use only data upto 2008
```

In [5]:

```
#Checking the NON null Count of and Data type of the columns
df_2008.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2253 entries, 0 to 2252
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Date                 2253 non-null   object
 1   Symbol               2253 non-null   object
 2   Series               2253 non-null   object
 3   Prev Close           2253 non-null   float64
 4   Open                 2253 non-null   float64
 5   High                 2253 non-null   float64
 6   Low                  2253 non-null   float64
 7   Last                 2253 non-null   float64
 8   Close                2253 non-null   float64
 9   VWAP                 2253 non-null   float64
 10  Volume               2253 non-null   int64
 11  Turnover             2253 non-null   float64
 12  Trades               0 non-null      float64
```

```
 13  Deliverable Volume  1744 non-null   float64
 14  %Deliverble         1744 non-null   float64
 15  Year                2253 non-null   int64
dtypes: float64(11), int64(2), object(3)
memory usage: 299.2+ KB
```

In [6]:

```
# Plotting the "Close" column of the data set

fig = df_2008.plot(x = "Date",y = "Close")
fig.figure.savefig("Close_plot.jpg",bbox_inches='tight')
plt.show()
```

In [7]:

```
#Dropping the unnecessary columns of the data   ("Trade" column has no Data)
df_2008 = df_2008.drop(["Date","Symbol","Series","Trades"],axis=1)
```

In [8]:

```
#Splitting the dataset into train and test
# Train Data for 2000 -2007 years
# Test Data for 2008
train = df_2008.loc[df_2008["Year"] < 2008]
test = df_2008.loc[df_2008["Year"] == 2008]
```

In [9]:

```
# Checking null values count in all the coulmns of the train dataset
```

```
train.isnull().sum()
```

Out[9]:
```
Prev Close             0
Open                   0
High                   0
Low                    0
Last                   0
Close                  0
VWAP                   0
Volume                 0
Turnover               0
Deliverable Volume   509
%Deliverble          509
Year                   0
dtype: int64
```

In [10]:
```
# Checking the skewness of the "Deliverable Volume"  and "%Deliverble" Columns
train["Deliverable Volume"].skew(axis = 0, skipna = True)
```

Out[10]:
```
5.805333236626633
```

In [11]:
```
train["%Deliverble"].skew(axis = 0, skipna = True)
```

Out[11]:
```
-0.5314183852015518
```

In [12]:
```
# Imputing the Columns with mean if not skewed
# Imputing the column with median if skewed
train["Deliverable Volume"].fillna(train["Deliverable Volume"].median(),inplace = True)
train["%Deliverble"].fillna(train["%Deliverble"].mean(),inplace = True)
train.isnull().sum()
```

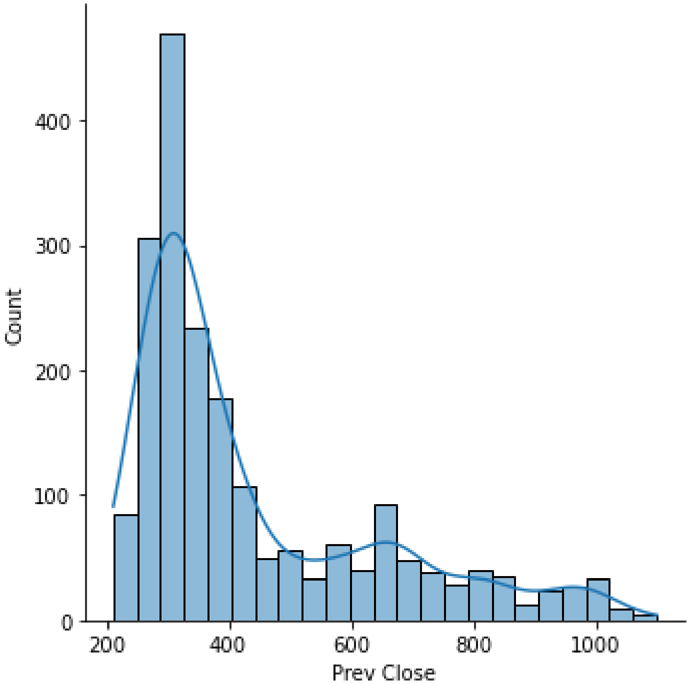
```
C:\Users\cricl\anaconda3\lib\site-packages\pandas\core\generic.py:6392: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  return self._update_inplace(result)
```

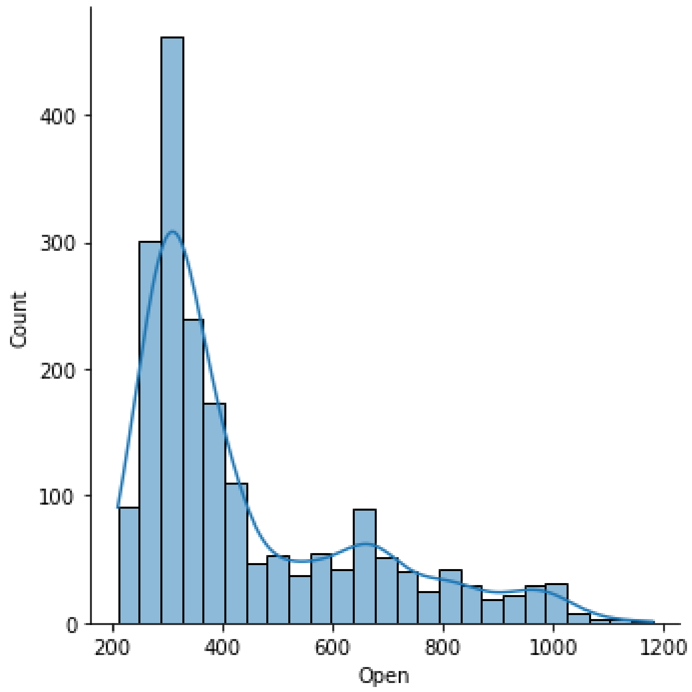
Out[12]:
```
Prev Close           0
Open                 0
High                 0
```

```
Low                   0
Last                  0
Close                 0
VWAP                  0
Volume                0
Turnover              0
Deliverable Volume    0
%Deliverble           0
Year                  0
dtype: int64
```

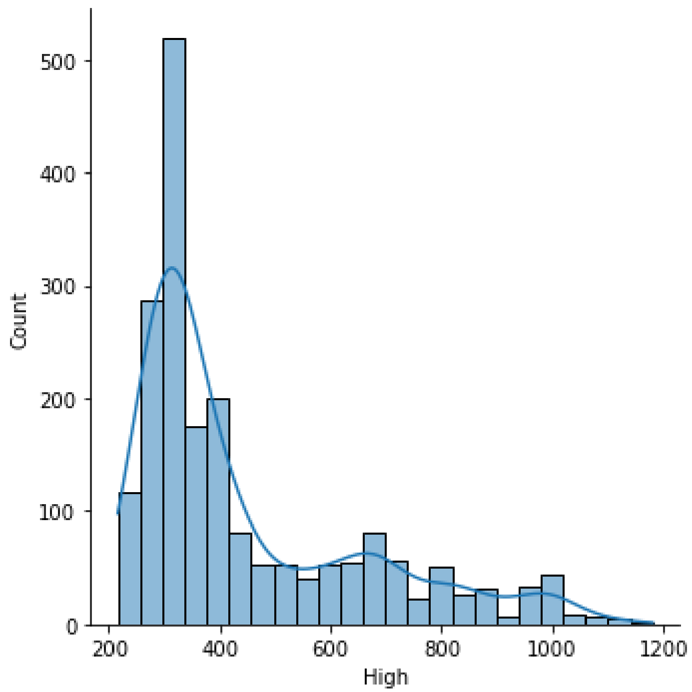
In [13]:

```
#Visualizxing the distributaion of the data for all the independent Columns
for i in range(len(train.columns)):
    x = train.columns[i]
    fig = sns.displot(data=train, x=x, kde=True)
    filename = "{} Histogram.jpg".format(train.columns[i])
    fig.figure.savefig(filename,bbox_inches='tight')
```

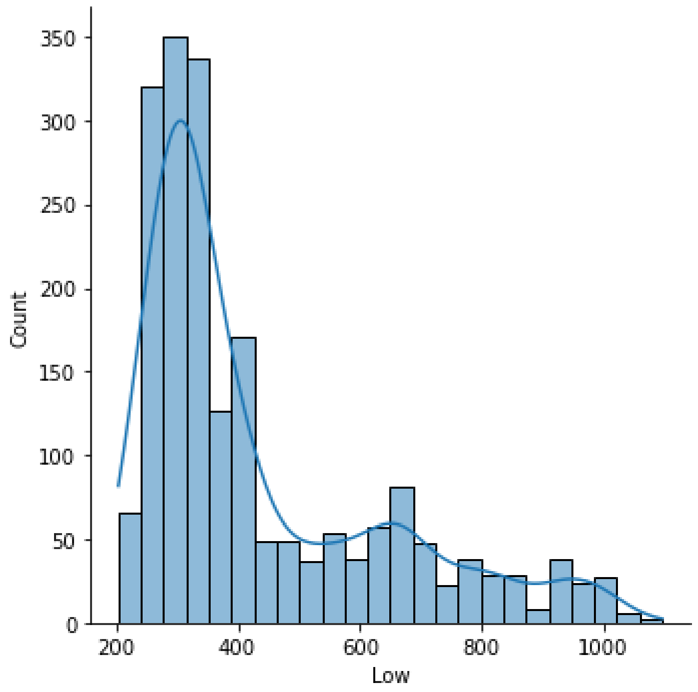


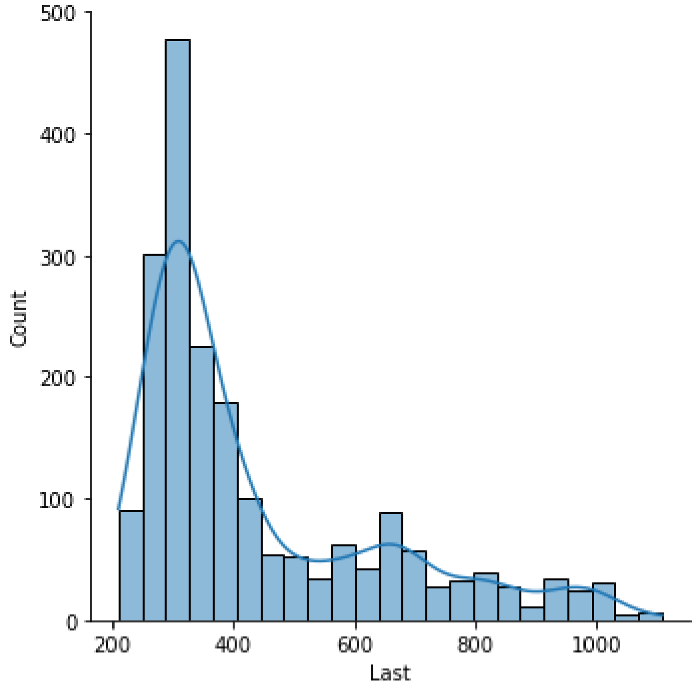


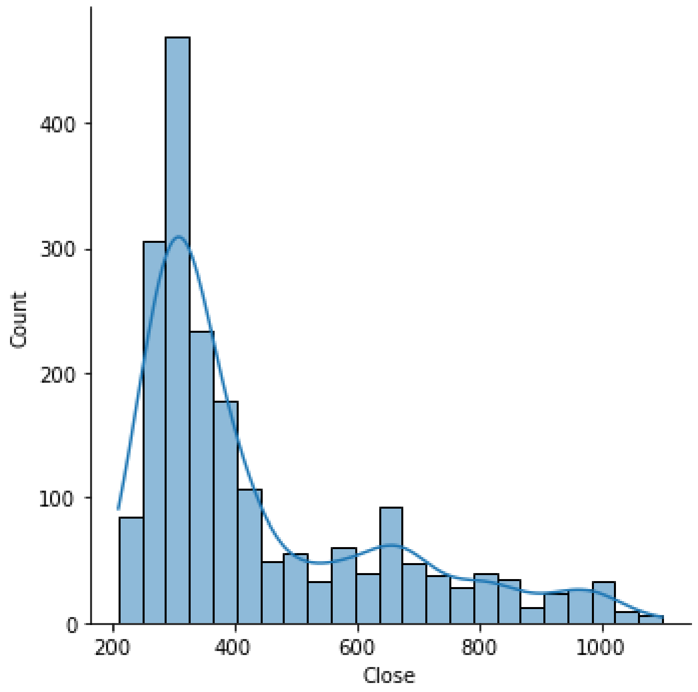
Count
Open
0
100
200
300
400
200
400
600
800
1000
1200

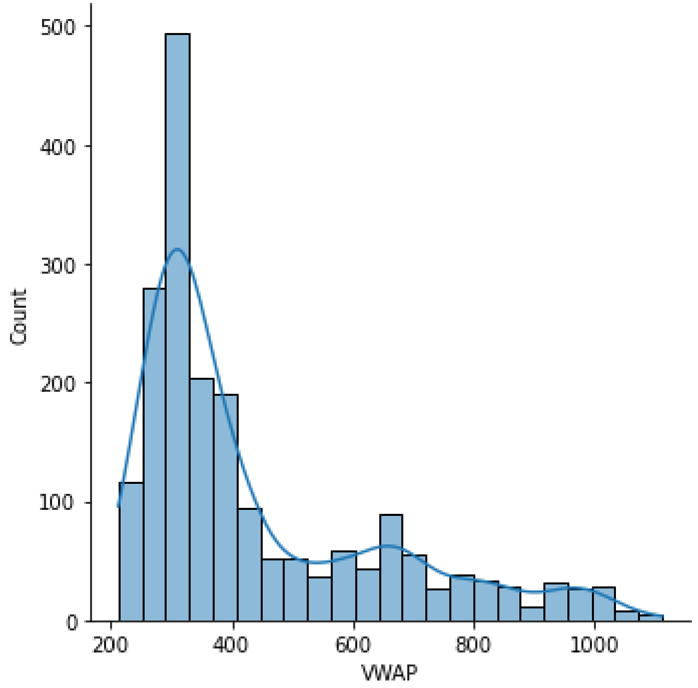


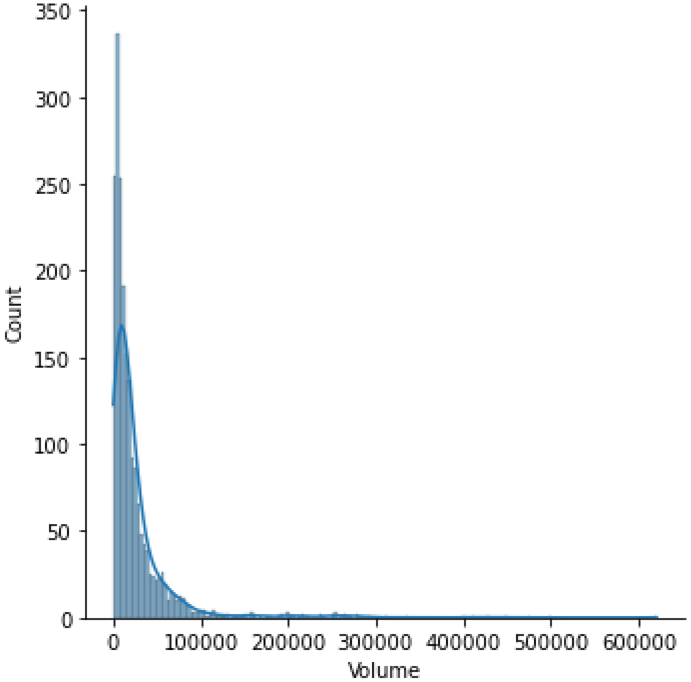
Count
High
0
100
200
300
400
500
200
400
600
800
1000
1200

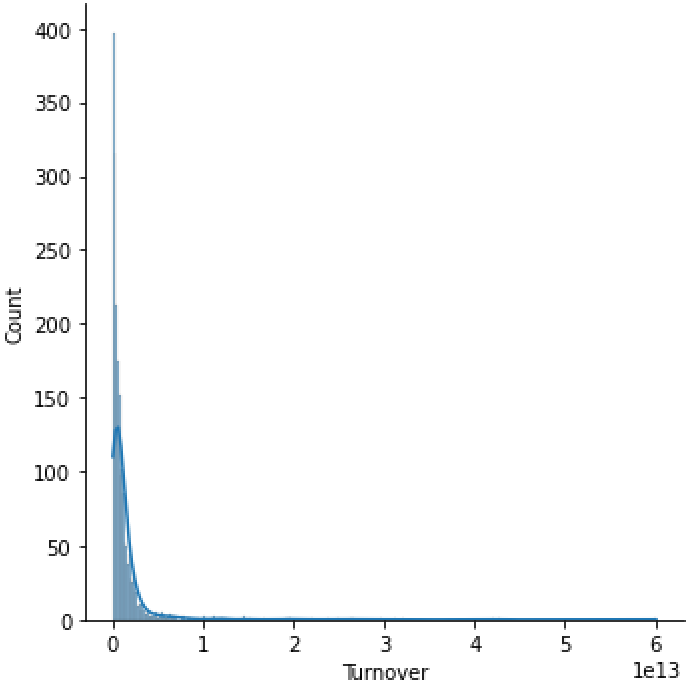

350
300
250
200
150
100
50
0
Count
200
400
600
800
1000
Low

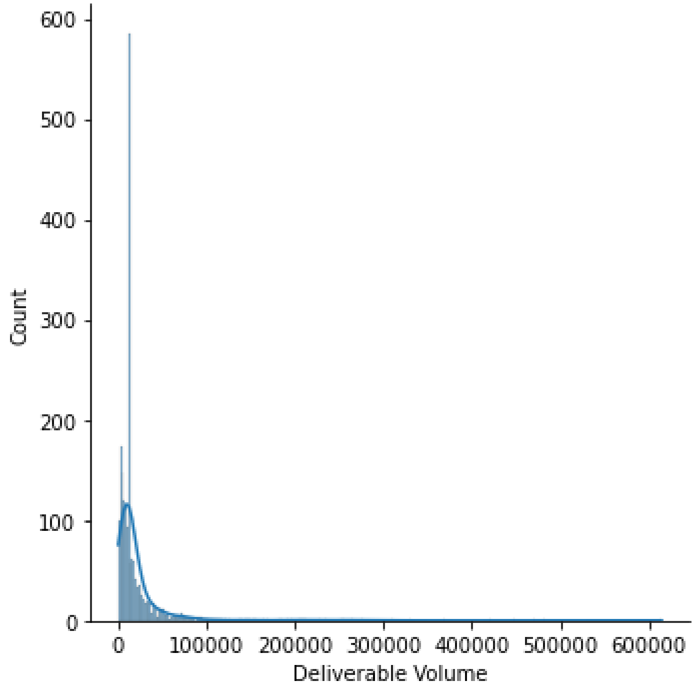

500
400
300
200
100
0
Count
200
400
600
800
1000
Last

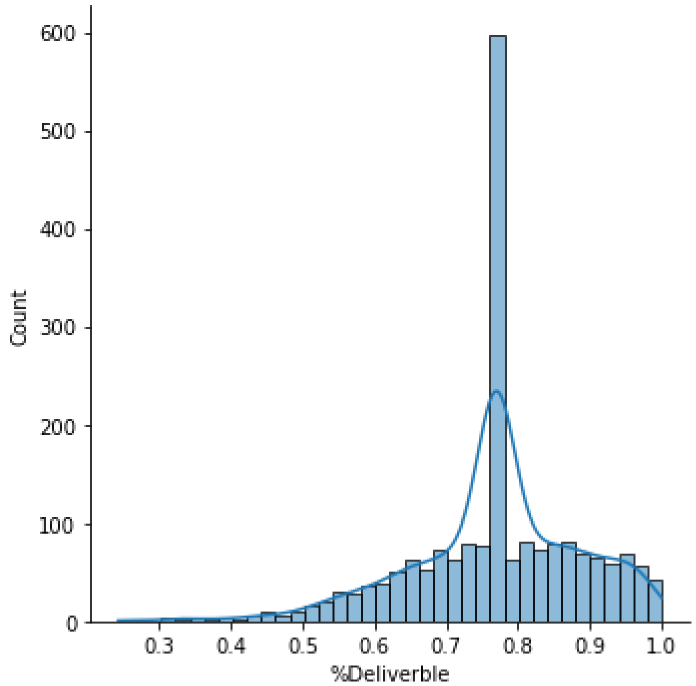

Count
Close

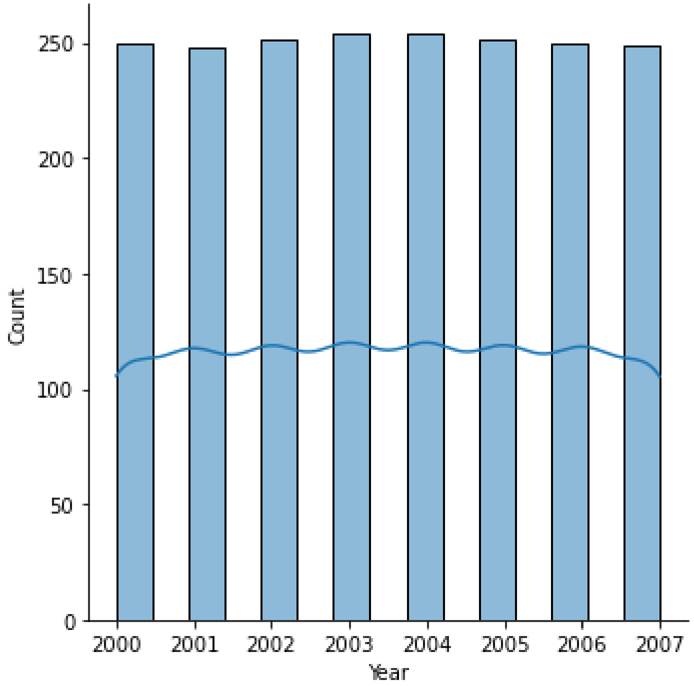

Count
VWAP


350
300
250
200
150
100
50
0
Count
0 100000 200000 300000 400000 500000 600000
Volume


400
350
300
250
200
150
100
50
0
Count
0 1 2 3 4 5 6
1e13
Turnover

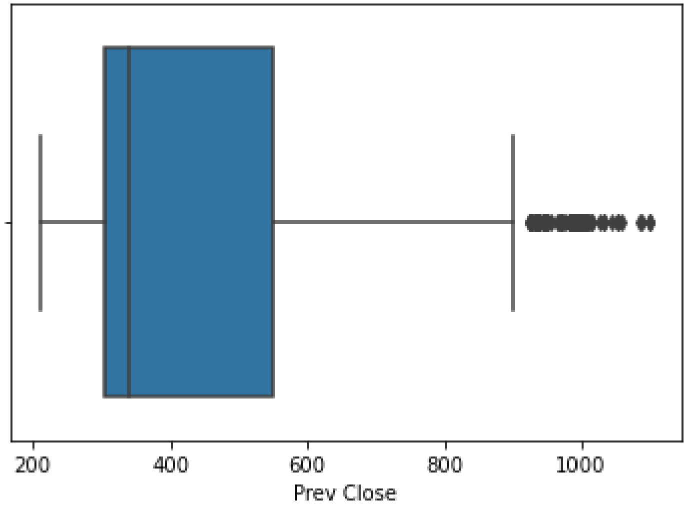

Count
Deliverable Volume
0
100
200
300
400
500
600
100000
200000
300000
400000
500000
600000

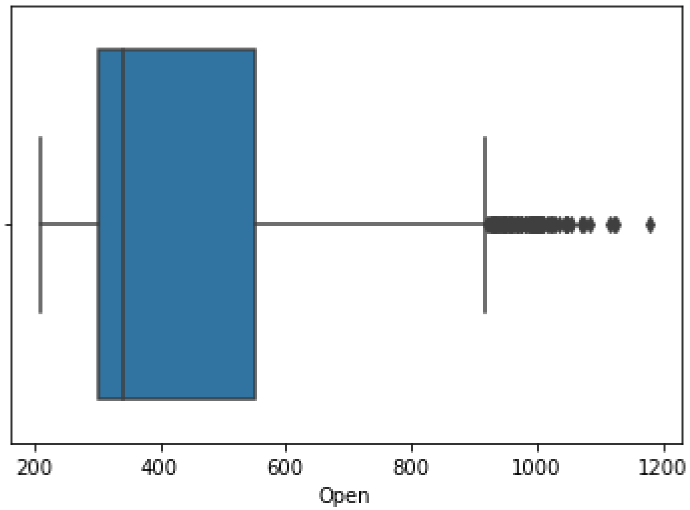

Count
%Deliverble
0
100
200
300
400
500
600
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

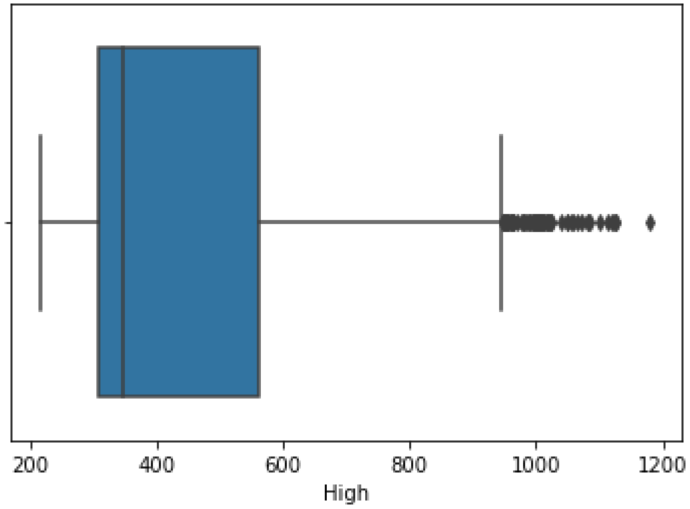


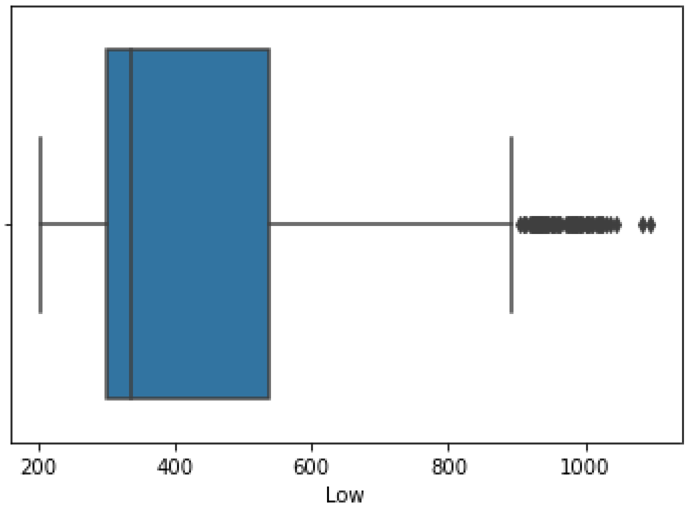
In [14]:

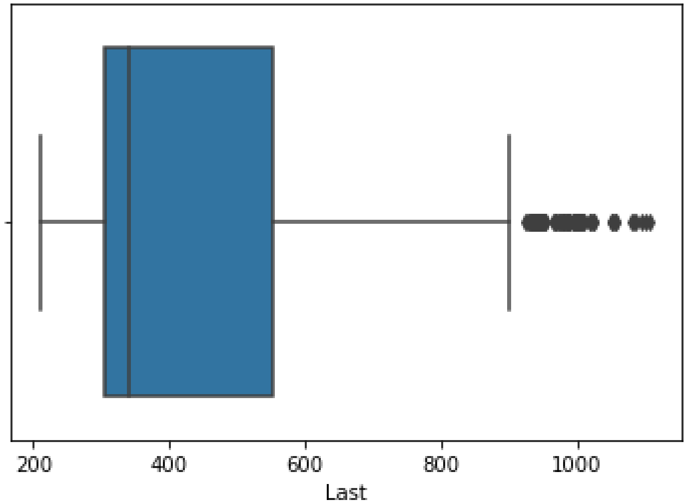
```
# Visualizing th data set for OUTLiers using BOXPLOT
for i in range(train.shape[1]):
    x = train.columns[i]
    fig = sns.boxplot(data=train, x=x)
    filename = "{} Boxplot.jpg".format(train.columns[i])
    fig.figure.savefig(filename,bbox_inches='tight')
    plt.show()
```

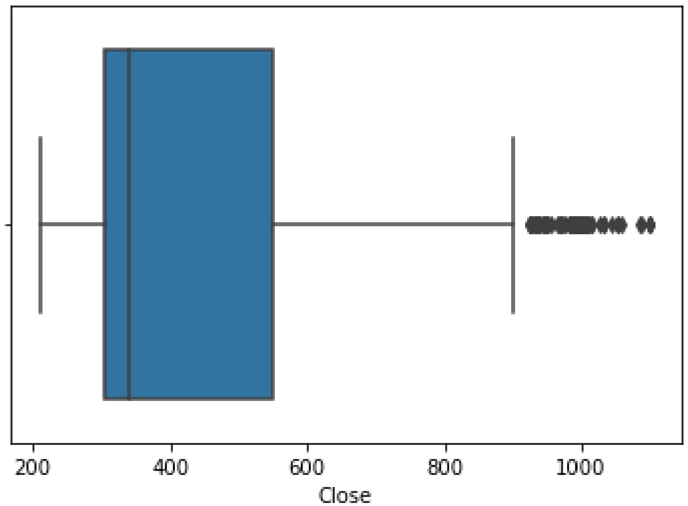

200
400
600
800
1000
Prev Close

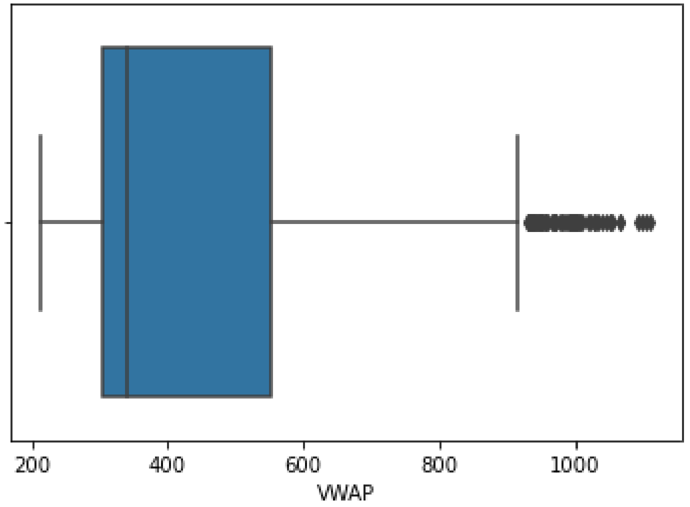

200
400
600
800
1000
1200
Open

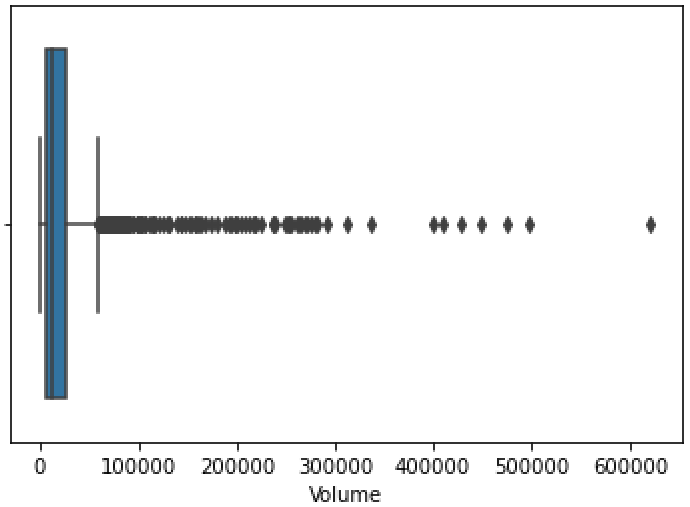

200
400
600
800
1000
1200
High

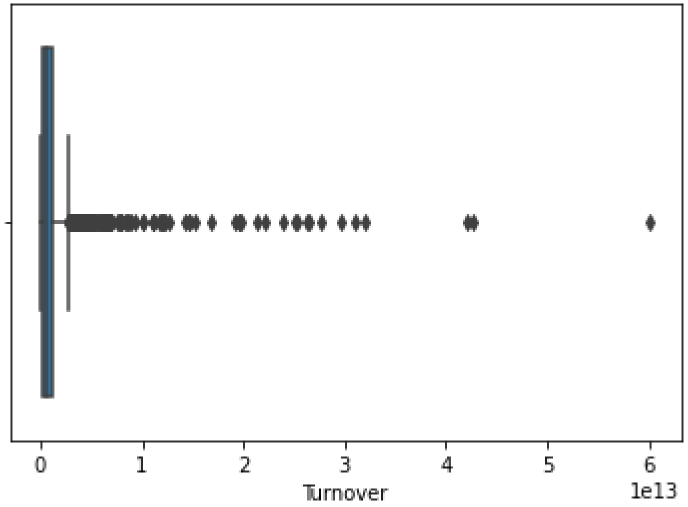

200
400
600
800
1000
Low

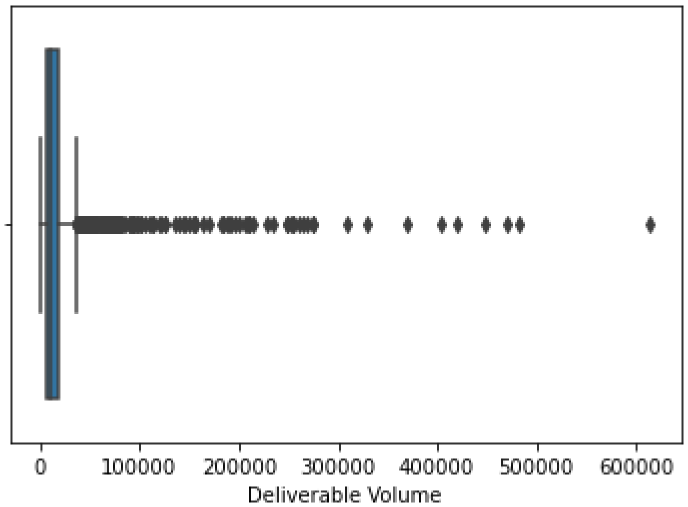

200
400
600
800
1000
Last

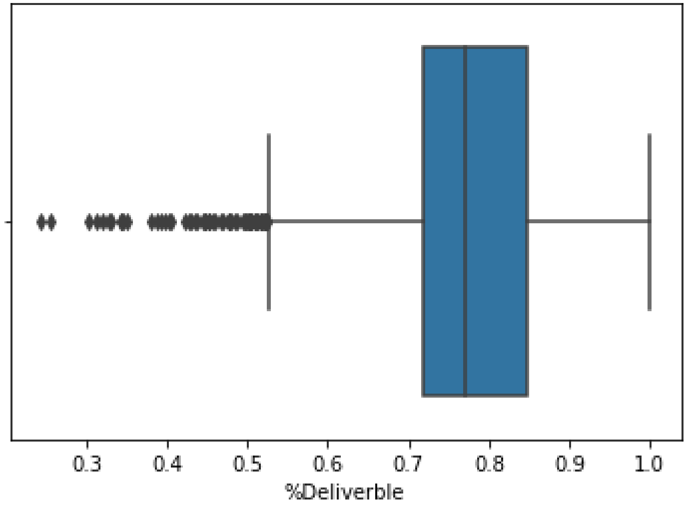

200
400
600
800
1000
Close

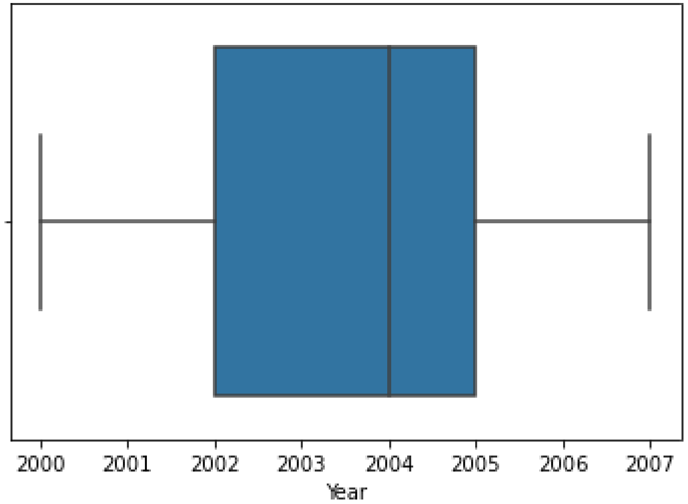

200
400
600
800
1000
VWAP


0
100000
200000
300000
400000
500000
600000
Volume


0
1
2
3
4
5
6
1e13
Turnover


0
100000
200000
300000
400000
500000
600000
Deliverable Volume





In [15]:
```
#SCAling the Train and Test Datasets Using STANDARD SCALER
scaler = StandardScaler()
scaled_array_train = scaler.fit_transform(train)
train = pd.DataFrame(scaled_array_train, columns = train.columns)
scaled_array_test = scaler.fit_transform(test)
test = pd.DataFrame(scaled_array_test, columns = test.columns)
```

In [16]:
```
# Creating a FUNCTION to remove outliers
```

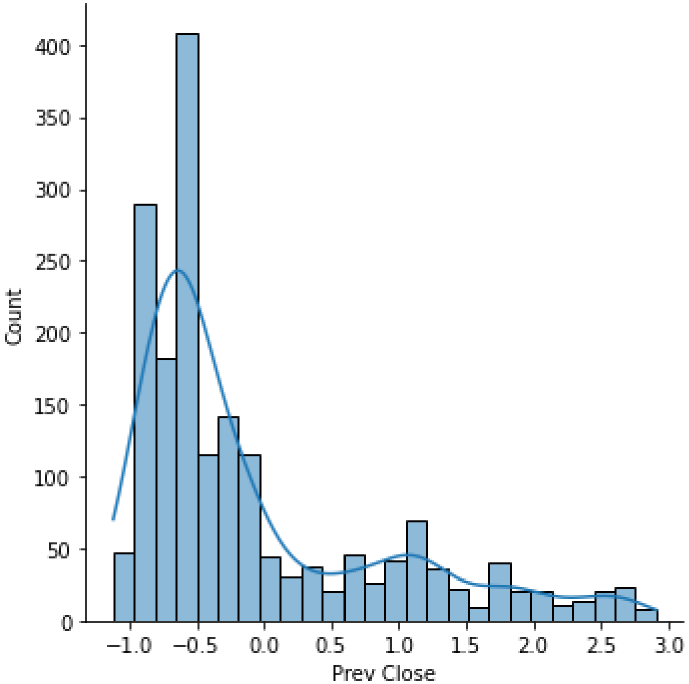
```
def remove_outliers(df):
    for i in range(df.shape[1]):
        col_name = df.columns[i]
        upper_limit = df[col_name].mean() +3*df[col_name].std()
        lower_limit = df[col_name].mean() -3*df[col_name].std()
        df = df[(df[col_name]<upper_limit) & (df[col_name]>lower_limit)]
    return(df)
```

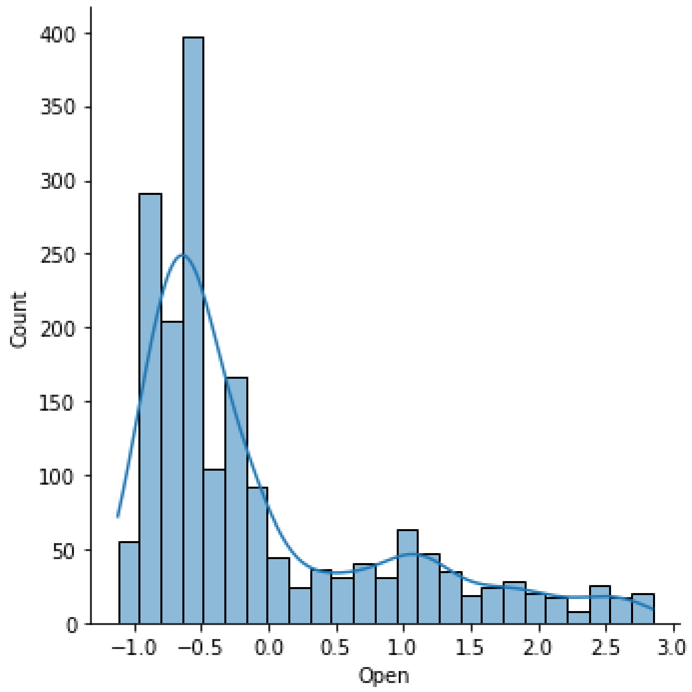
In [17]:

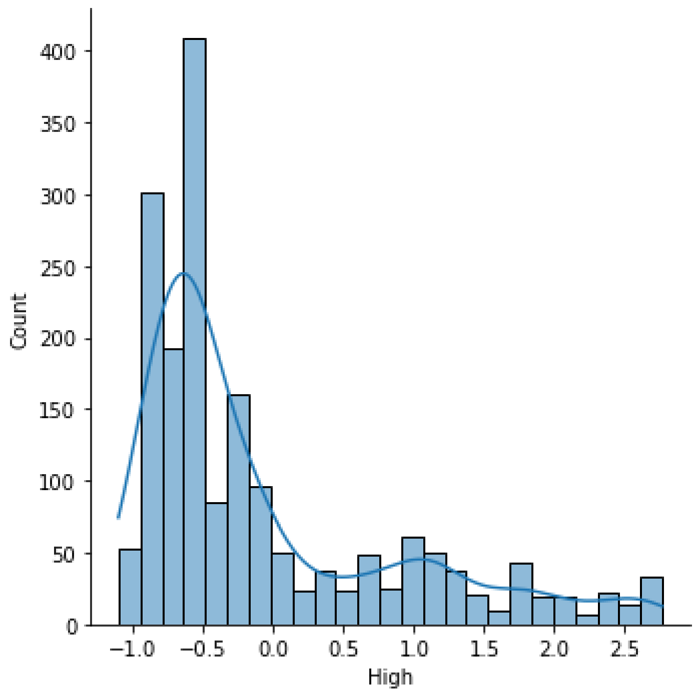
```
#Remove outliers
train = remove_outliers(train)
```

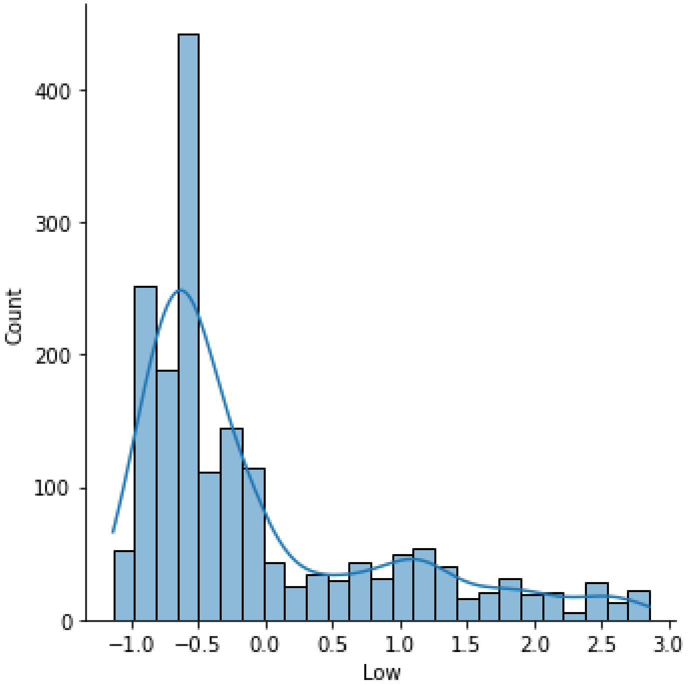
In [18]:

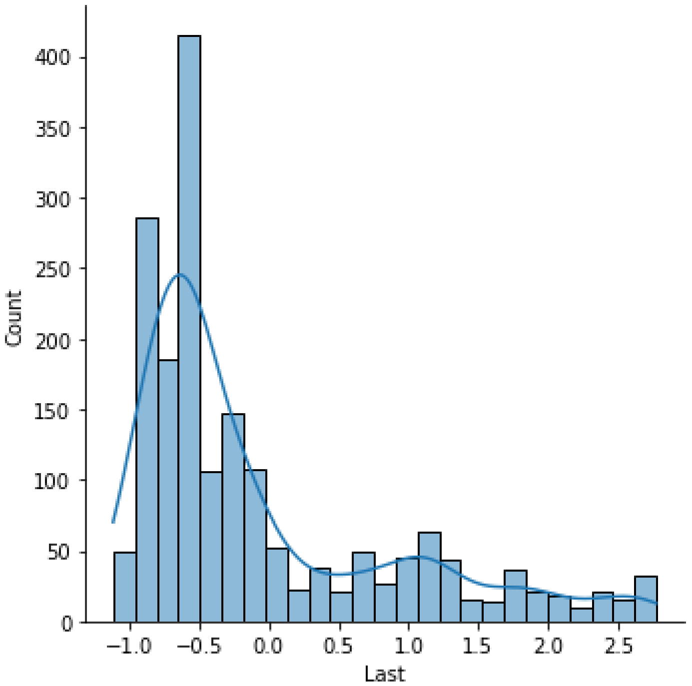
```
# Visualizing the cleaned Data
for i in range(train.shape[1]):
    x = train.columns[i]
    fig = sns.displot(data=train, x=x, kde=True)
    filename = "{} Cleaned_Histogram.jpg".format(train.columns[i])
    fig.figure.savefig(filename,bbox_inches='tight')
```

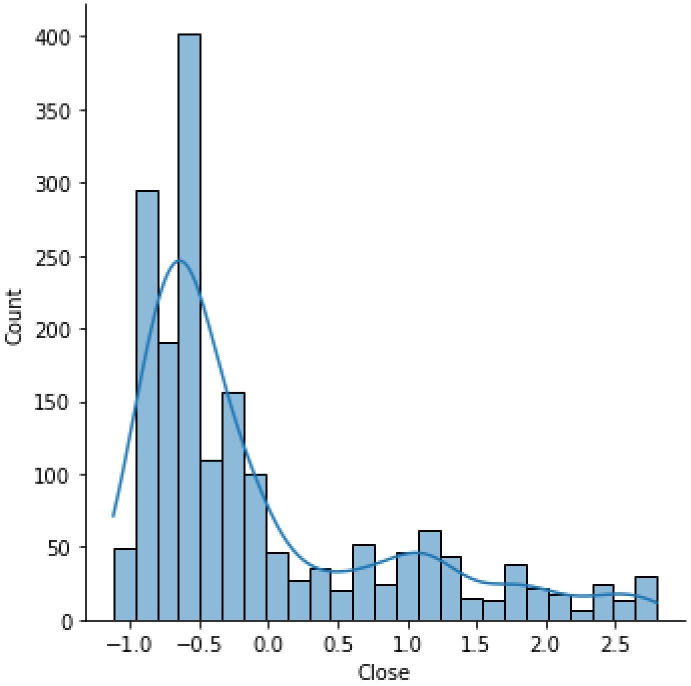


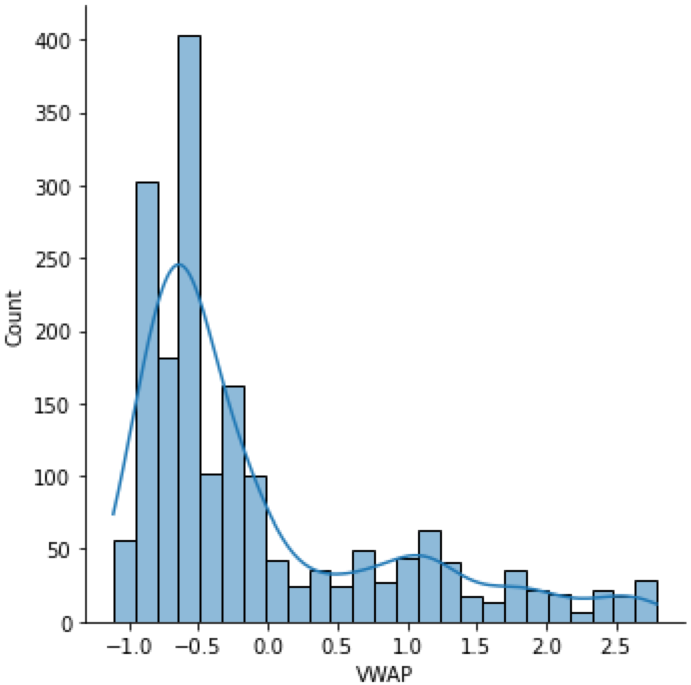

Count
Open
400
350
300
250
200
150
100
50
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

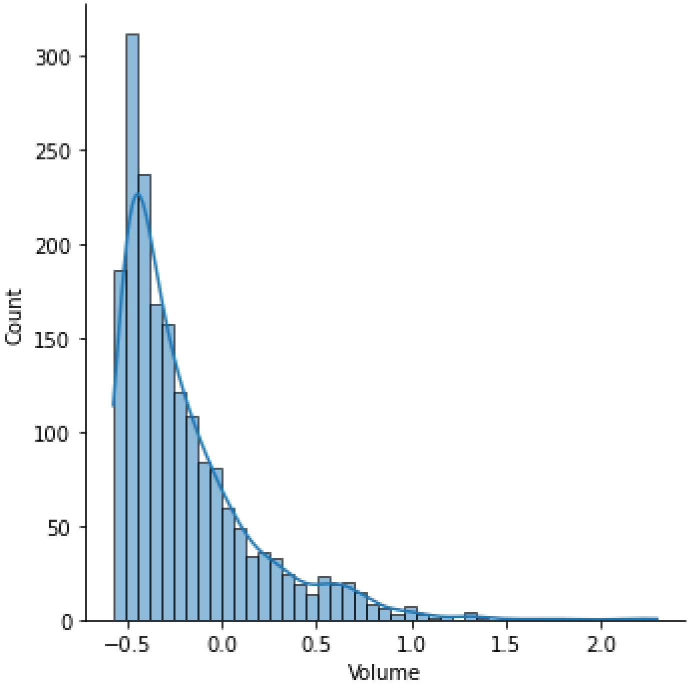

Count
High
400
350
300
250
200
150
100
50
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Count
Low
0
100
200
300
400
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

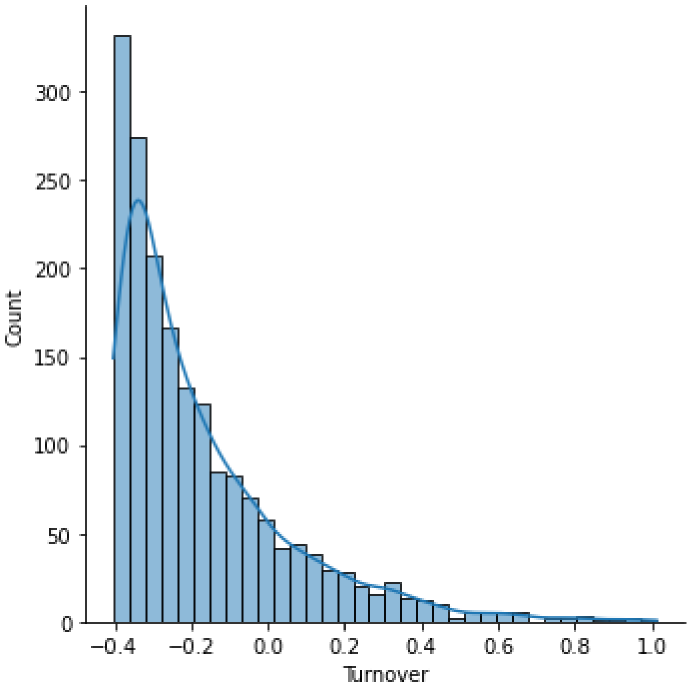

Count
Last
0
50
100
150
200
250
300
350
400
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Count
Close
400
350
300
250
200
150
100
50
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

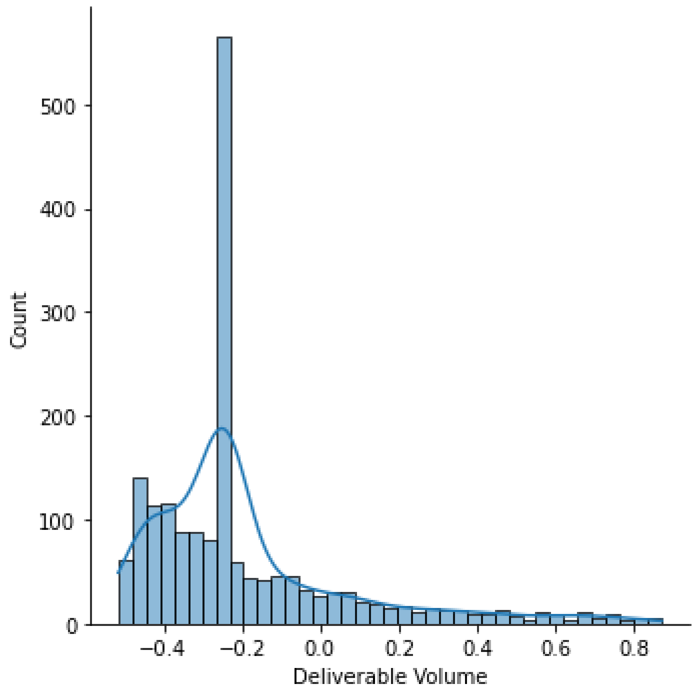

Count
VWAP
400
350
300
250
200
150
100
50
0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

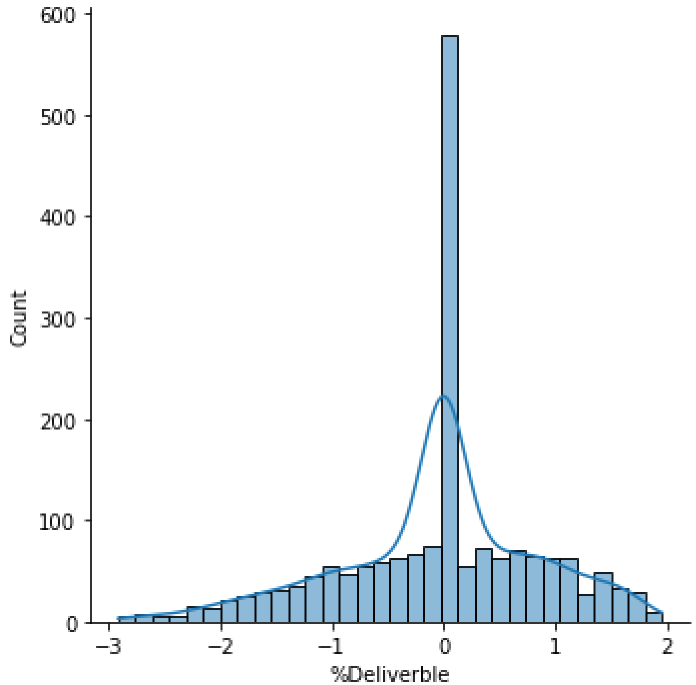


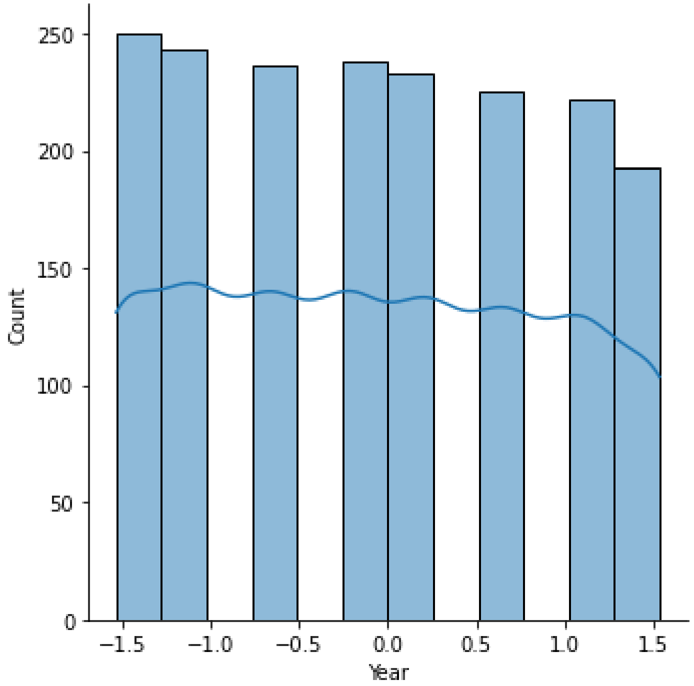
Count
Volume



Count
Turnover

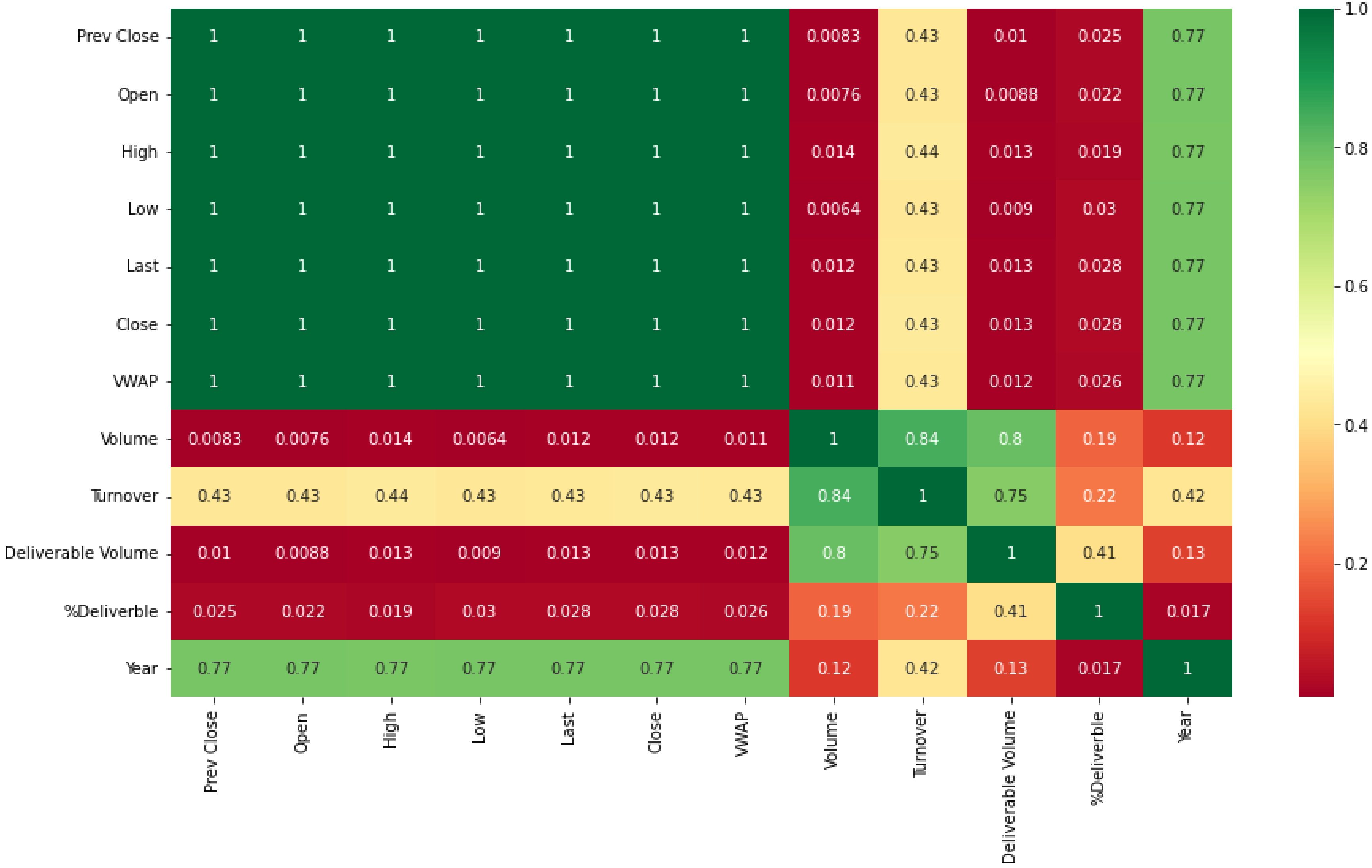

Count
500
400
300
200
100
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
Deliverable Volume


Count
600
500
400
300
200
100
0
−3
−2
−1
0
1
2
%Deliverble



In [19]:

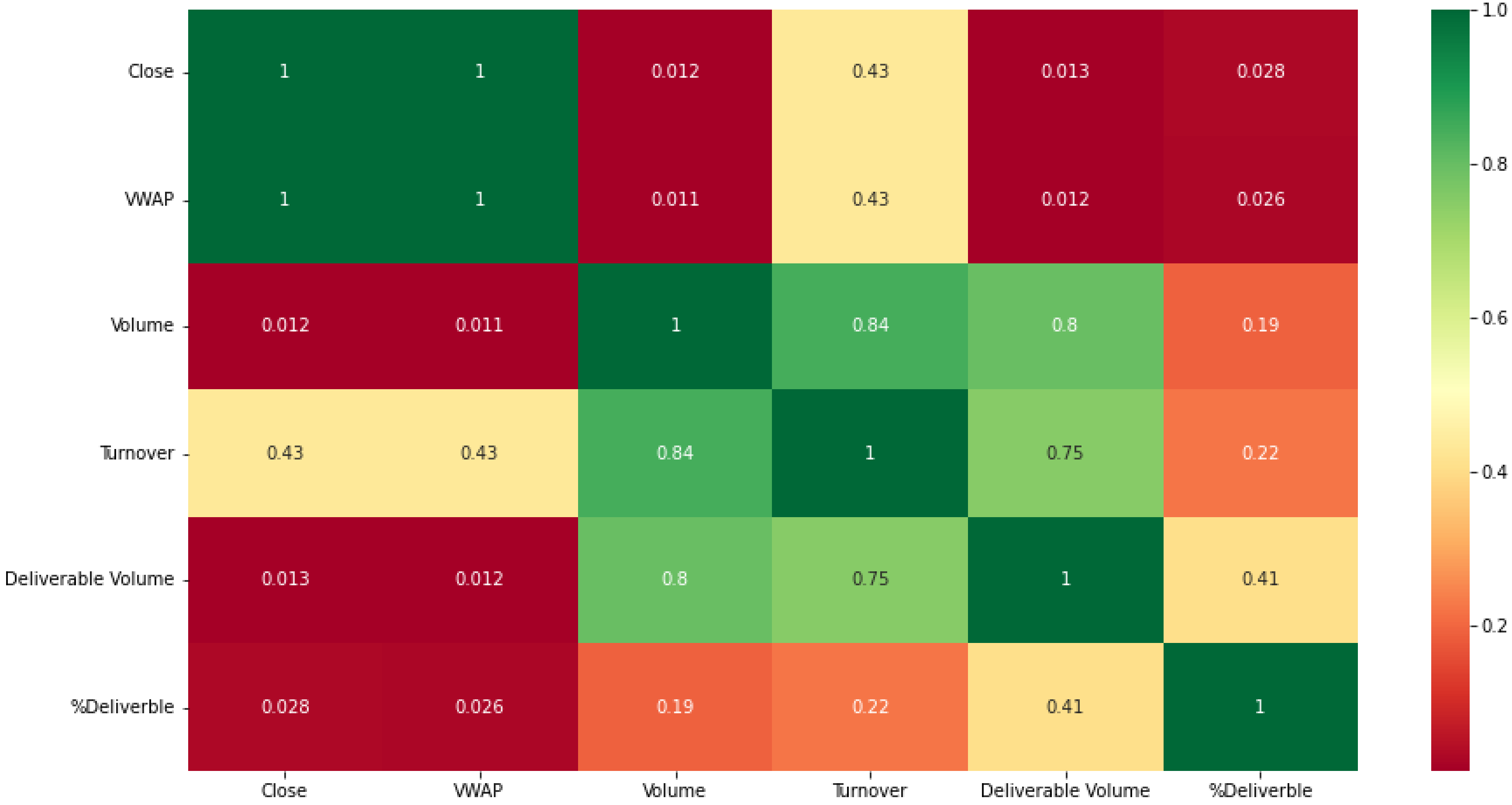
```
# Visualizing the Data for Corelations Using HEAT MAP
corr= train.corr()
plt.figure(figsize=(15,8))
fig = sns.heatmap(corr,annot=True,cmap='RdYlGn')
fig.figure.savefig("Heat map.jpg",bbox_inches='tight')
plt.show()
```



In [20]:
```
#Dropping the Columns with Corelation of 1 as they donot comtribute to the model creation
train = train.drop(["Prev Close", "Open", "High","Low","Last","Year"],axis = 1)
test = test.drop(["Prev Close", "Open", "High","Low","Last","Year"],axis = 1)
```

In [21]:
```
corr= train.corr()
plt.figure(figsize=(15,8))
```

```
fig = sns.heatmap(corr,annot=True,cmap='RdYlGn')
fig.figure.savefig("Feature_selected_Heat map.jpg",bbox_inches='tight')
plt.show()
```



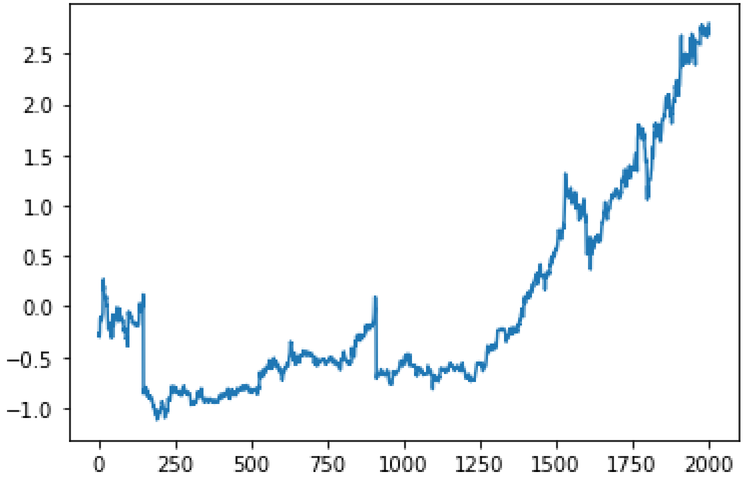
In [22]:

```
#Splitting X_train, Y_train, X_test, Y_test
X_train = train.drop(["Close"],axis=1)
Y_train = train["Close"]

X_test = test.drop(["Close"],axis=1)
Y_test = test["Close"]
```

In [23]:

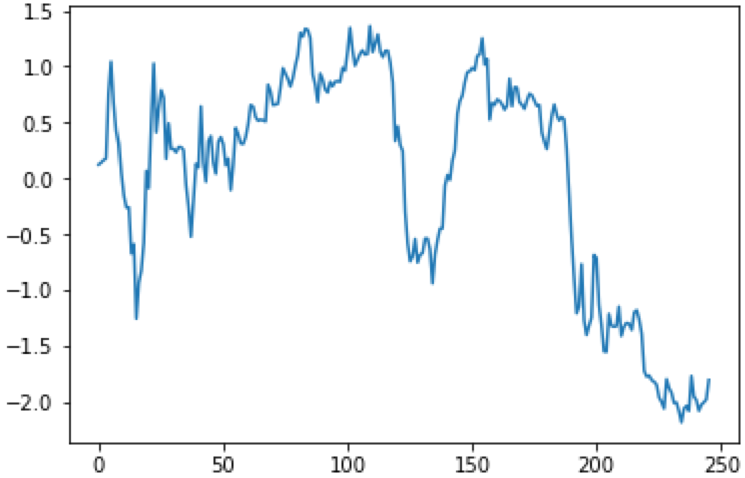
```
Y_train.plot()
```

Out[23]: <AxesSubplot:>



In [24]:
```
Y_test.plot()
```

Out[24]: <AxesSubplot:>



In [26]:
```
#List Hyperparameters that we want to tune.
n_neighbors = list(range(1,15))
```

```
p=[1,2]
#Convert to dictionary
params = dict(n_neighbors=n_neighbors, p=p)
#Create new KNN object
KNN = KNeighborsRegressor()
#Use GridSearch
GSCV = GridSearchCV(KNN, params, cv=10)
#Fit the model
best_model = GSCV.fit(X_train,Y_train)
#Print The value of best Hyperparameters
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best p: 2
Best n_neighbors: 2
```

In [27]:

```
# Assigning the parameters tuned
p = best_model.best_estimator_.get_params()['p']
n_neighbors = best_model.best_estimator_.get_params()['n_neighbors']
```

In [28]:

```
#KNN Model creation and fitting the data
model_KNN = KNeighborsRegressor(n_neighbors=n_neighbors,p = p)
model_KNN.fit(X_train,Y_train)
```

Out[28]:

```
KNeighborsRegressor(n_neighbors=2)
```

In [29]:

```
#Random Forest Model creation and fitting the data
model_RF = RandomForestRegressor()
model_RF.fit(X_train, Y_train)
```

Out[29]:

```
RandomForestRegressor()
```

In [30]:

```
#Ada Boost Model creation and fitting the data
model_Ada = AdaBoostRegressor()
model_Ada.fit(X_train, Y_train)
```

Out[30]:

```
AdaBoostRegressor()
```

In [31]:
```
#Creating the structure of the Neural Network model
# Assigning the num of neurons per layer
hidden_layer1 = 150
hidden_layer2 = 200
hidden_layer3 = 250
# Learnig rate and Input dimensions
learning_rate = 0.01
input_dim = X_train.shape[1]
# Creating a Sequential Nueral network model with 3 Dense and 3 Dropuout layers altenatively
model_NN = Sequential()

model_NN.add(Dense(hidden_layer1, input_dim=input_dim, kernel_initializer='normal', activation='relu'))

model_NN.add(Dropout(0.2))

model_NN.add(Dense(hidden_layer2, kernel_initializer='normal', activation='relu'))

model_NN.add(Dropout(0.2))

model_NN.add(Dense(hidden_layer3, kernel_initializer='normal', activation='relu'))

model_NN.add(Dropout(0.2))

model_NN.add(Dense(1, kernel_initializer='normal', activation='linear'))

#Compilimng the model using Optimizer and learning rate
model_NN.compile(loss='mse',optimizer=Adam(learning_rate=learning_rate),metrics=['mse'])
# train the model
history = model_NN.fit(X_train,Y_train,epochs=300,batch_size=10)
```

Metrics

- MSE -- Mean Squared Error
- RMSE -- Root mean squred error
- R2 -- R Squared
- ADJ_R2 -- Adjusted R Squared
- MAPE -- Mean Absolute Percentage Error

In [33]:
```
# Created a function to evaluate the metrics of all the models
```

```
def Evaluate_models(model):
    y_pred = model.predict(X_test)

    MSE = mean_squared_error(Y_test, y_pred)

    RMSE = mean_squared_error(Y_test, y_pred, squared=False)

    R2 = r2_score(Y_test, y_pred)

    ADJ_R2 = 1 - (1-R2)*(len(Y_train)-1)/(X_train.shape[0]-X_train.shape[1]-1)

    MAPE = mean_absolute_percentage_error(Y_test, y_pred)

    return (MSE,RMSE,R2,ADJ_R2,MAPE)
```

In [34]:

```
# Evaluating all the metrics for K Nearest Neighbour model
Metrics_KNN  = Evaluate_models(model_KNN)
MSE_KNN = Metrics_KNN[0]
RMSE_KNN = Metrics_KNN[1]
R2_KNN = Metrics_KNN[2]
ADJ_R2_KNN = Metrics_KNN[3]
MAPE_KNN = Metrics_KNN[4]
```

In [35]:

```
# Evaluating all the metrics for Random Forest model
Metrics_RF  = Evaluate_models(model_RF)
MSE_RF= Metrics_RF[0]
RMSE_RF = Metrics_RF[1]
R2_RF = Metrics_RF[2]
ADJ_R2_RF = Metrics_RF[3]
MAPE_RF = Metrics_RF[4]
```

In [36]:

```
# Evaluating all the metrics for Ada Boost model
Metrics_Ada  = Evaluate_models(model_Ada)
MSE_Ada= Metrics_Ada[0]
RMSE_Ada = Metrics_Ada[1]
R2_Ada = Metrics_Ada[2]
ADJ_R2_Ada = Metrics_Ada[3]
MAPE_Ada = Metrics_Ada[4]
```

In [37]:

```
# Evaluating all the metrics for Neural Network model
Metrics_NN  = Evaluate_models(model_NN)
MSE_NN= Metrics_NN[0]
RMSE_NN = Metrics_NN[1]
R2_NN = Metrics_NN[2]
ADJ_R2_NN = Metrics_NN[3]
MAPE_NN = Metrics_NN[4]
```

Creating the Dataframe with Metrics corresponding to all the models created

In [54]:

```
Comp={'Models':['KNN','Random Forest','AdaBoost',"Neural Network"],
      'MSE' :[MSE_KNN,MSE_RF,MSE_Ada,MSE_NN],
      'RMSE' :[RMSE_KNN,RMSE_RF,RMSE_Ada,RMSE_NN],
      'R2' :[R2_KNN,R2_RF,R2_Ada,R2_NN],
      'ADJ R2' :[ADJ_R2_KNN,ADJ_R2_RF,ADJ_R2_Ada,ADJ_R2_NN],
      'MAPE' :[MAPE_KNN,MAPE_RF,MAPE_Ada,MAPE_NN]

   }

Compare=pd.DataFrame(Comp)
Compare
```

Out[54]:

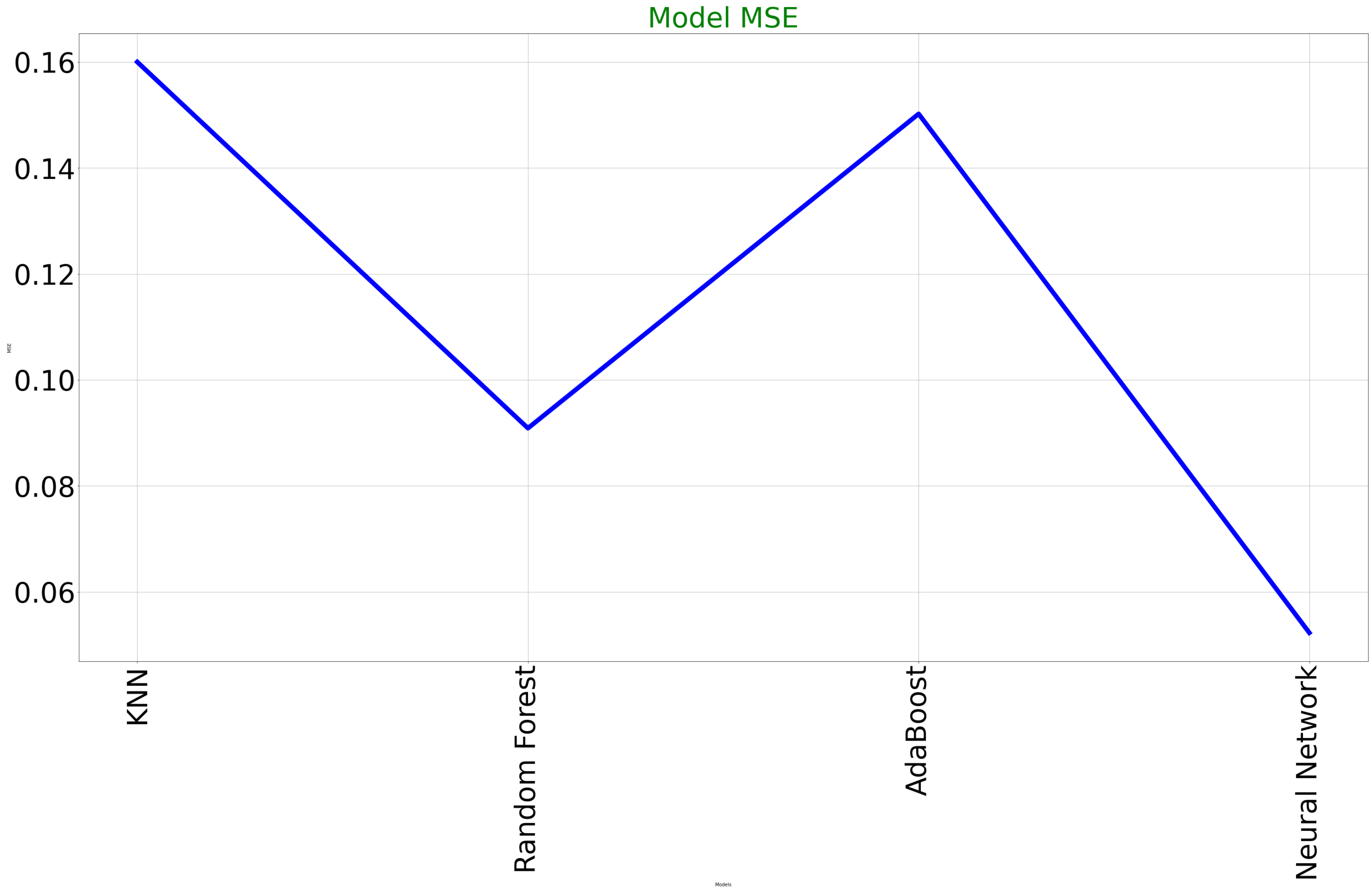
	Models	MSE	RMSE	R2	ADJ R2	MAPE
0	KNN	0.160018	0.400022	0.839982	0.839546	0.358373
1	Random Forest	0.090895	0.301489	0.909105	0.908857	0.225484
2	AdaBoost	0.150219	0.387581	0.849781	0.849372	0.275105
3	Neural Network	0.052283	0.228654	0.947717	0.947575	0.379525

Plotting Model MSE

In [56]:

```
plt.figure(figsize =(50, 25))
```

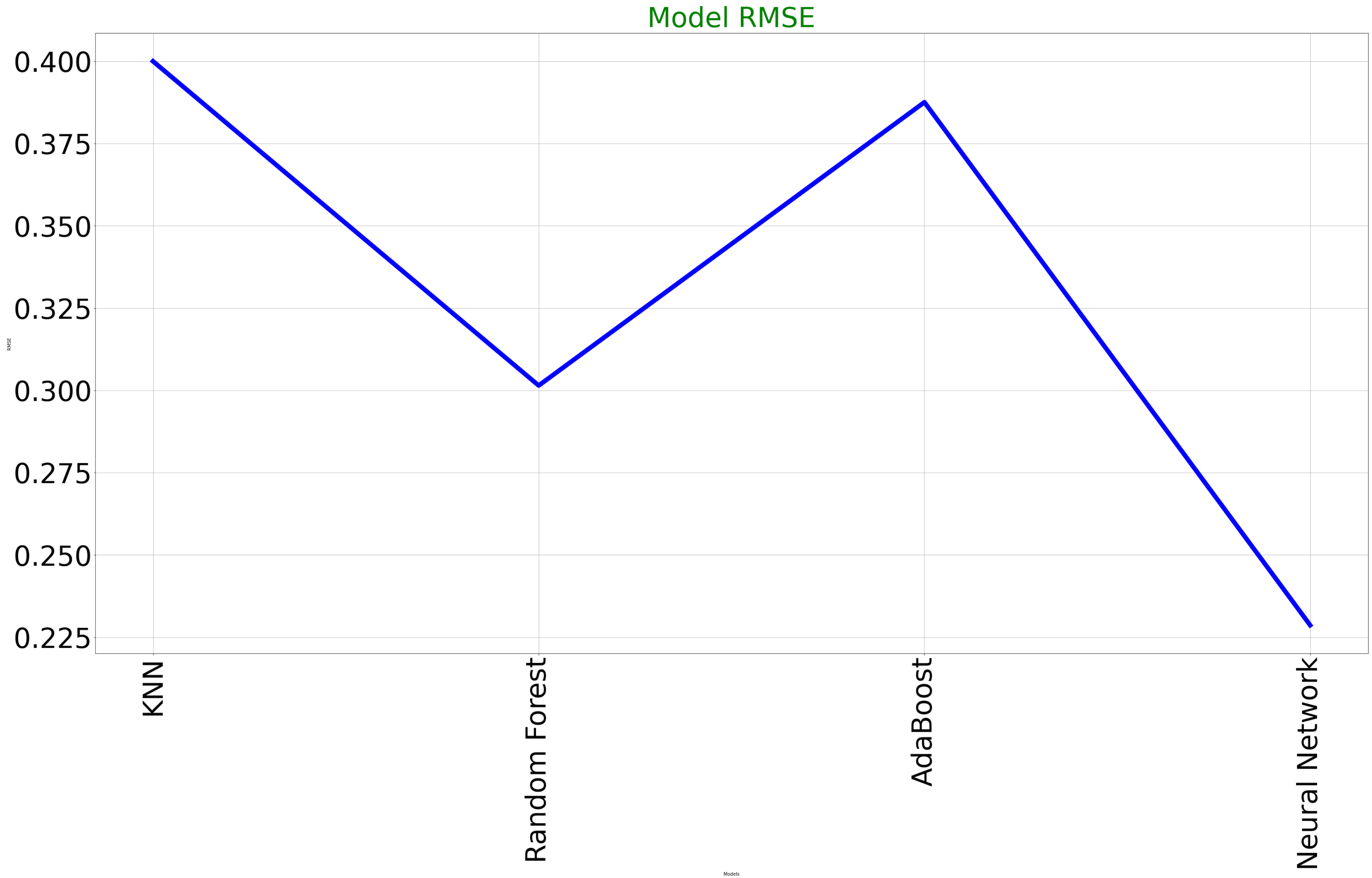
```
plt.plot(Compare['Models'],Compare['MSE'],c='blue', lw=10)

plt.title('Model MSE',fontdict={'fontsize': 60,'fontweight' : 60,'color' : 'g'})
plt.xlabel('Models')
plt.ylabel('MSE')

plt.yticks(fontsize=60)
plt.xticks(rotation=90, fontsize=60)
plt.grid()
plt.savefig("MSE.jpg",bbox_inches='tight')
plt.show()
```



Plotting Model RMSE

In [41]:
```
plt.figure(figsize =(50, 25))
```

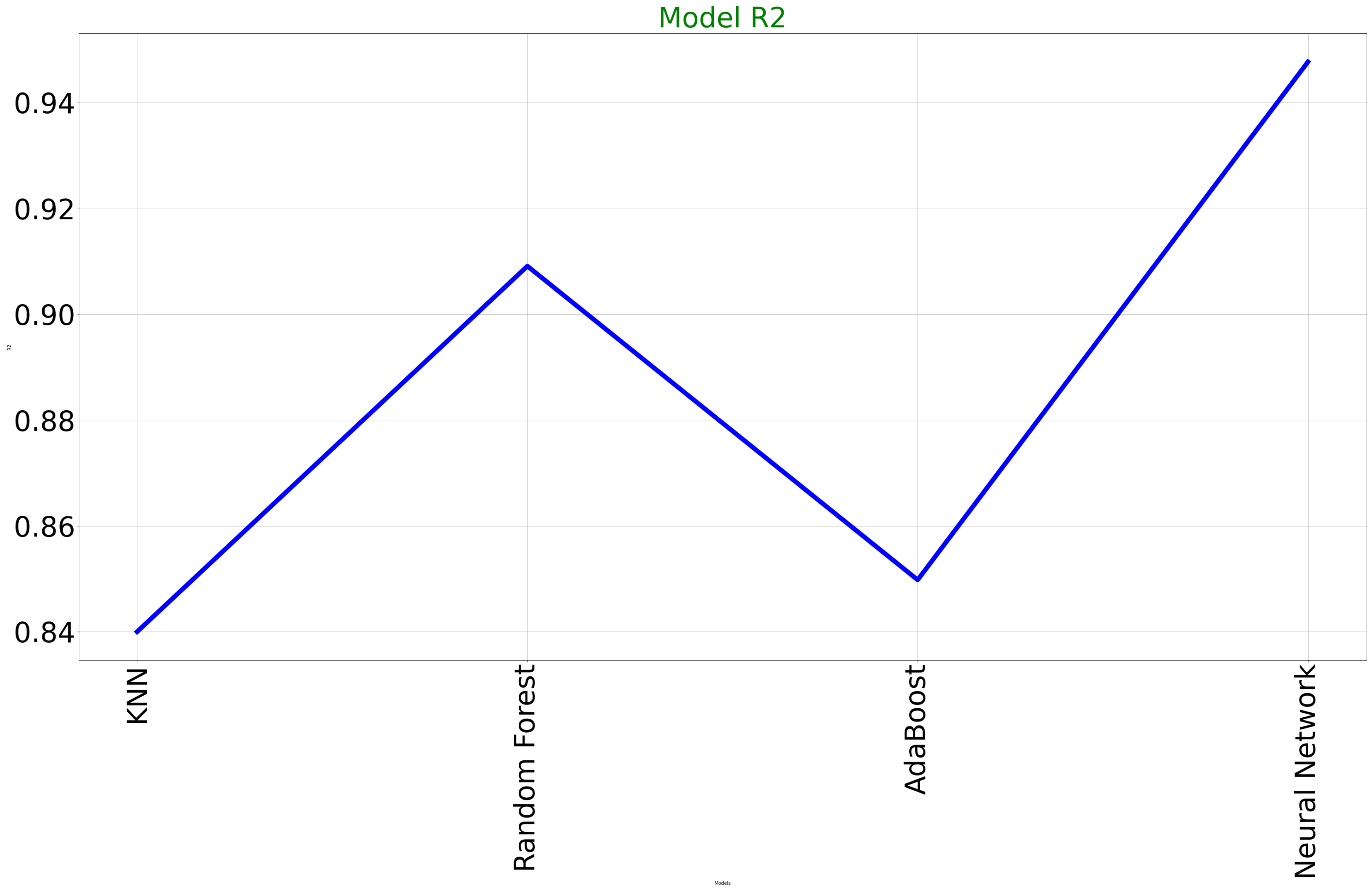
```
plt.plot(Compare['Models'],Compare['RMSE'],c='blue', lw=10)

plt.title('Model RMSE',fontdict={'fontsize': 60,'fontweight' : 60,'color' : 'g'})
plt.xlabel('Models')
plt.ylabel('RMSE')

plt.yticks(fontsize=60)
plt.xticks(rotation=90, fontsize=60)
plt.grid()
plt.savefig("RMSE.jpg",bbox_inches='tight')
plt.show()
```



Plotting Model R Squared

In [42]:
```
plt.figure(figsize =(50, 25))
```

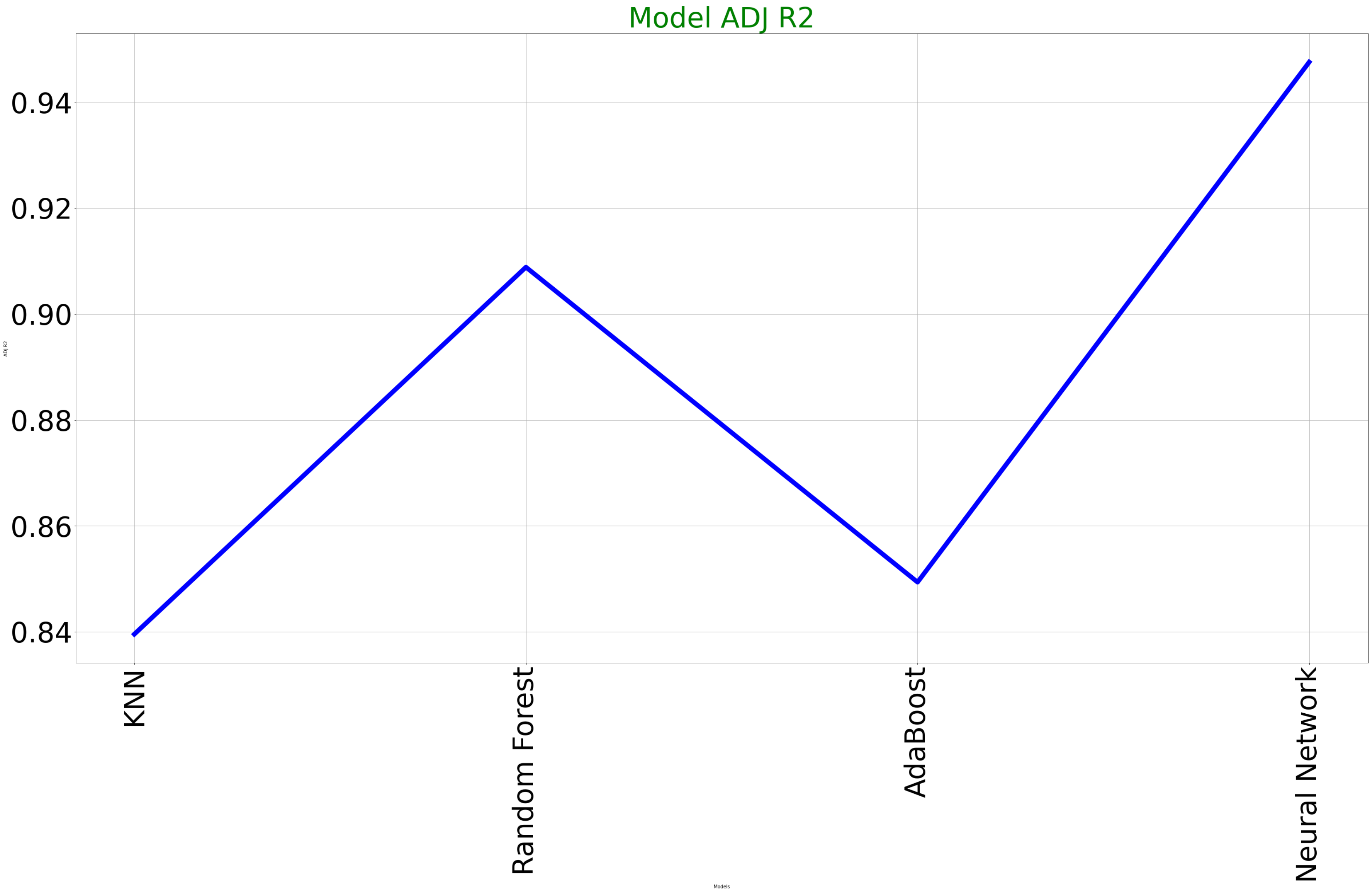
```
plt.plot(Compare['Models'],Compare['R2'],c='blue', lw=10)

plt.title('Model R2',fontdict={'fontsize': 60,'fontweight' : 60,'color' : 'g'})
plt.xlabel('Models')
plt.ylabel('R2')

plt.yticks(fontsize=60)
plt.xticks(rotation=90, fontsize=60)
plt.grid()
plt.savefig("R2.jpg",bbox_inches='tight')
plt.show()
```



Plotting Model Adjusted R Squared

In [43]:
```
plt.figure(figsize =(50, 25))
```

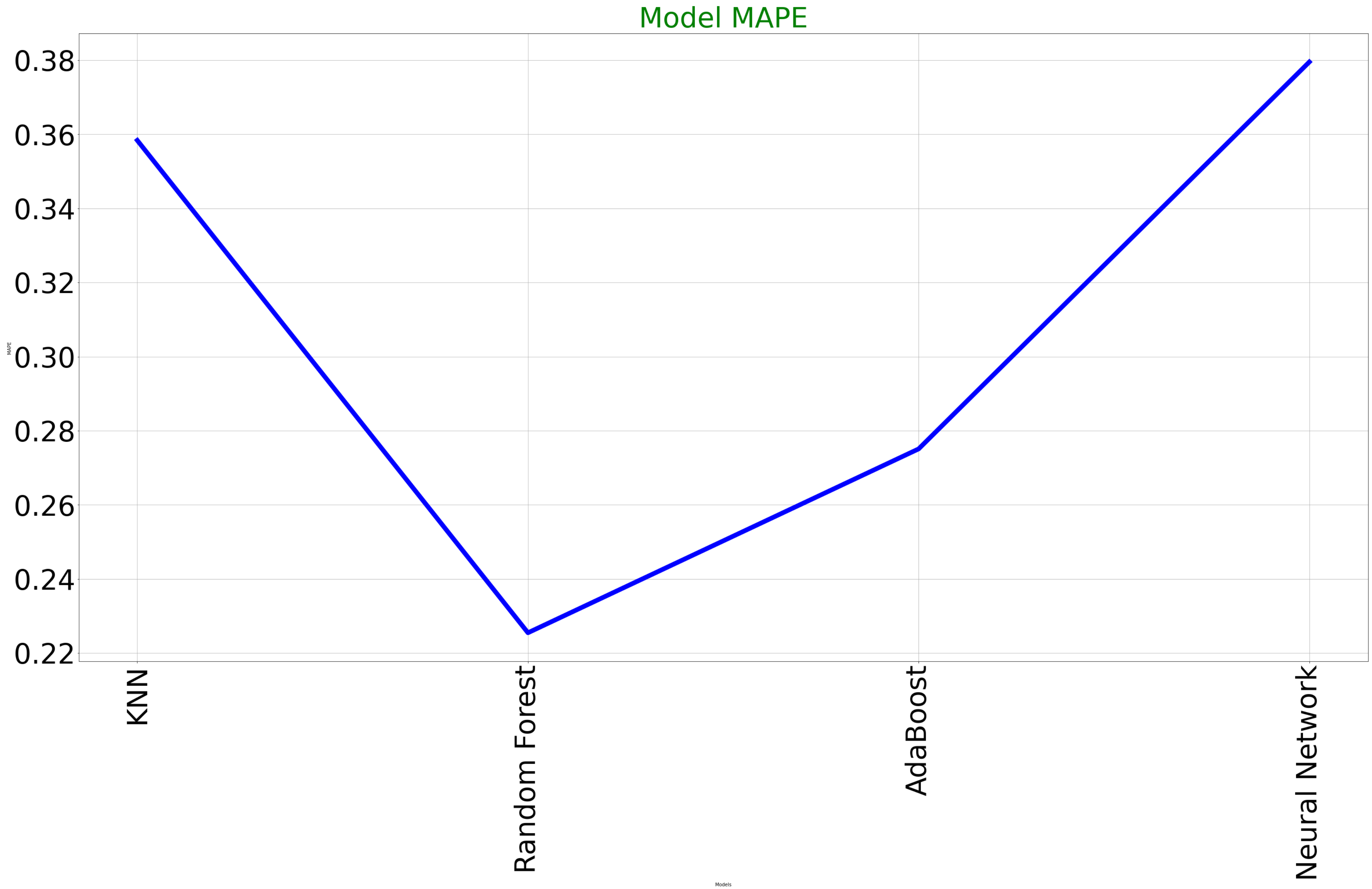
```
plt.plot(Compare['Models'],Compare['ADJ R2'],c='blue', lw=10)

plt.title('Model ADJ R2',fontdict={'fontsize': 60,'fontweight' : 60,'color' : 'g'})
plt.xlabel('Models')
plt.ylabel('ADJ R2')

plt.yticks(fontsize=60)
plt.xticks(rotation=90, fontsize=60)
plt.grid()
plt.savefig("ADJ R2.jpg",bbox_inches='tight')
plt.show()
```



Plotting Model MAPE

In [58]:
```
plt.figure(figsize =(50, 25))
```

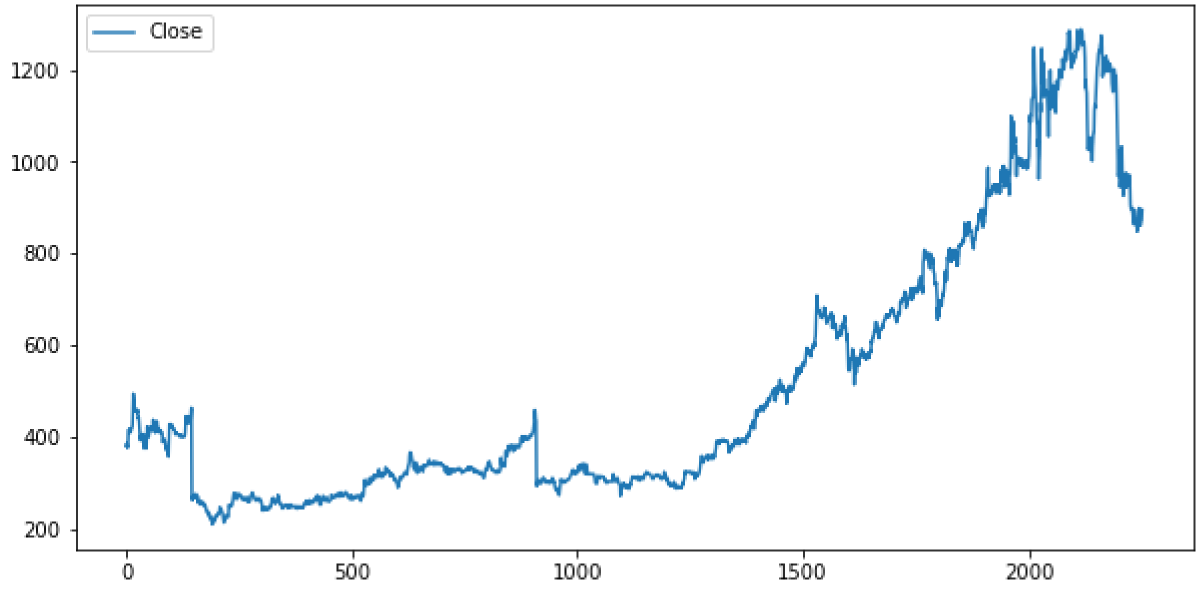
```
plt.plot(Compare['Models'],Compare['MAPE'],c='blue', lw=10)

plt.title('Model MAPE',fontdict={'fontsize': 60,'fontweight' : 60,'color' : 'g'})
plt.xlabel('Models')
plt.ylabel('MAPE')

plt.yticks(fontsize=60)
plt.xticks(rotation=90, fontsize=60)
plt.grid()
plt.savefig("MAPE.jpg",bbox_inches='tight')
plt.show()
```



In [59]:

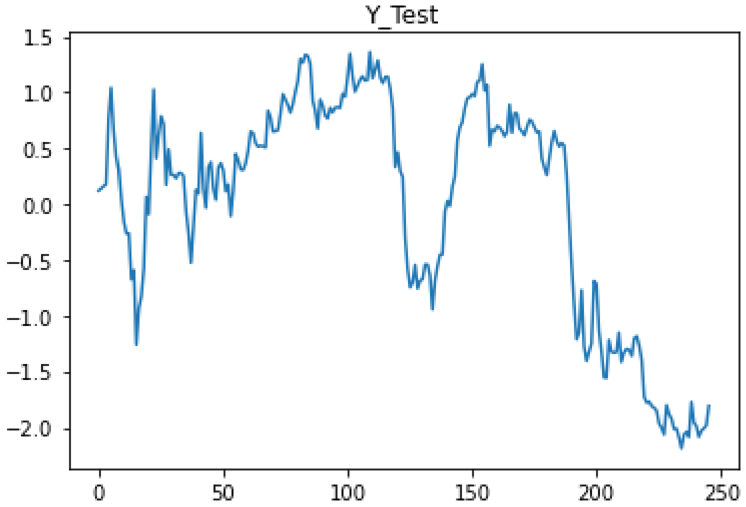
```
# Visualizing the dataset created for "Close" column
df_2008.plot(y = "Close",figsize=(10,5))
```

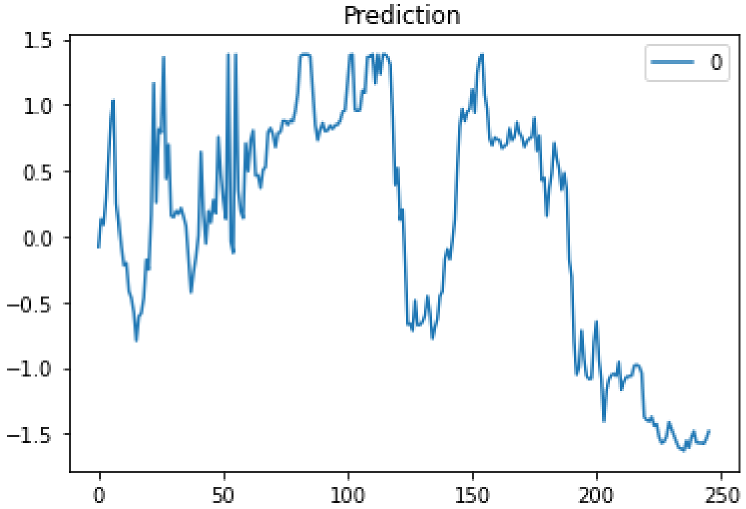
Out[59]: <AxesSubplot:>



In [71]:

```
# Visualizing the plot comparing between Y_test and the predicted values for 2008 year
y_pred = model_NN.predict(X_test)
y_pred = pd.DataFrame(y_pred)
Y_test.plot()
plt.title("Y_Test")
plt.savefig("Y_test.jpg",bbox_inches='tight')
y_pred.plot()
plt.title("Prediction")
plt.savefig("Prediction.jpg",bbox_inches='tight')
```


Y_Test
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
50
100
150
200
250


Prediction
0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
50
100
150
200
250

```
In [ ]:
```