In [83]:
```
#pip install nbconvert
```

In [2]:
```
# Import Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
pd.set_option('display.max_rows',None)
pd.set_option('display.max_columns',None)
```

Reading Mercedes Benz Data Set

In [4]:
```
test = pd.read_csv('E:/downloads/MACHINE LEARNING PROJECTS/Mercedes-Benz Greener
train =pd.read_csv('E:/downloads/MACHINE LEARNING PROJECTS/Mercedes-Benz Greener
```

In [5]:
```
test.head()
```

Out[5]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X2
0	1	az	v	n	f	d	t	a	w	0	0	0	0	0	0	0	0	0	0	
1	2	t	b	ai	a	d	b	g	y	0	0	0	0	0	0	0	0	0	1	
2	3	az	v	as	f	d	a	j	j	0	0	0	0	1	0	0	0	0	0	
3	4	az	l	n	f	d	z	l	n	0	0	0	0	0	0	0	0	0	0	
4	5	w	s	as	c	d	y	i	m	0	0	0	0	1	0	0	0	0	0	

In [6]:
```
train.head()
```

Out[6]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17	X18
0	0	130.81	k	v	at	a	d	u	j	o	0	0	0	1	0	0	0	0	1
1	6	88.53	k	t	av	e	d	y	l	o	0	0	0	0	0	0	0	0	1
2	7	76.26	az	w	n	c	d	x	j	x	0	0	0	0	0	0	0	1	0
3	9	80.62	az	t	n	f	d	x	l	e	0	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	0	0	0	0	0	0	0	0	0

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

Out[7]:
```
(4209, 378)
```

In [8]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 378 entries, ID to X385
dtypes: float64(1), int64(369), object(8)
memory usage: 12.1+ MB
```

In [9]:
```
test.shape
```

Out[9]:
```
(4209, 377)
```

In [10]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 377 entries, ID to X385
dtypes: int64(369), object(8)
memory usage: 12.1+ MB
```

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

Out[11]:

	ID	y	X10	X11	X12	X13	X14	
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	420
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	

In [12]:
```
train.nunique()
```

Out[12]:
```
ID      4209
y       2545
X0        47
X1        27
X2        44
X3         7
X4         4
X5        29
X6        12
X8        25
X10        2
X11        1
X12        2
X13        2
X14        2
X15        2
X16        2
X17        2
X18        2
```

In [13]:
```
train['y'].nunique()
```

Out[13]:
```
2545
```

In [14]:
```
print("y is the difference in train test columns, so it is a Target column")
print("Since Target Variable has 2500 unique values, it is a Regression Problem.'
```

```
y is the difference in train test columns, so it is a Target column
Since Target Variable has 2500 unique values, it is a Regression Problem.
```

In [15]:
```
#check the unique value , for that if uniques value is 1 than it has no variance
len(train['X0'].unique())
```

Out[15]:
```
47
```

In [16]:
```
#find all the columns of our data
col = train.columns
col
```

Out[16]:
```
Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=378)
```

Checking columns with zero variance

In [17]:
```
#To check uniques value in all 384 columns sepratly is tuff so we do it by a func
for each in col:
    if len(train[each].unique()) == 1:
        print('column name','    ',   each)
```

```
column name     X11
column name     X93
column name     X107
column name     X233
column name     X235
column name     X268
column name     X289
column name     X290
column name     X293
column name     X297
column name     X330
column name     X347
```

In [18]:
```
#from above we have found that columns X11,X93,X107,X233,X235,X268,X289,X290,X293
#than from probelms objective -If for any column(s), the variance is equal to zer
#we remove
train.drop(['X11','X93','X107','X233','X235','X268','X289','X290','X293','X297',
```

In [19]:
```
train.shape
```

Out[19]:
```
(4209, 366)
```

Checking Null Value in Test & Train Data

In [20]:
```
train.isnull().sum()
```

Out[20]:
```
ID      0
y       0
X0      0
X1      0
X2      0
X3      0
X4      0
X5      0
X6      0
X8      0
X10     0
X12     0
X13     0
X14     0
X15     0
X16     0
X17     0
X18     0
X19     0
```

In [21]:
```
train.isnull().sum().sum()
```

Out[21]:
```
0
```

In [22]:
```
train.isnull().sum().any()
```

Out[22]:
```
False
```

In [23]:
```
test.isnull().sum()
```

Out[23]:
```
ID      0
X0      0
X1      0
X2      0
X3      0
X4      0
X5      0
X6      0
X8      0
X10     0
X11     0
X12     0
X13     0
X14     0
X15     0
X16     0
X17     0
X18     0
X19     0
```

In [24]:
```
test.isnull().sum().sum()
```

Out[24]:
```
0
```

In [25]:
```
test.isnull().sum().any()
```

Out[25]:
```
False
```

Checking columns data types

In [26]:
```
train.dtypes.value_counts()
```

Out[26]:
```
int64      357
object       8
float64      1
dtype: int64
```

In [27]:
```
#to find dtype of all columns we do like this
for each in train.columns:
    print(each, "---",np.dtype(train[each]))
```

```
ID --- int64
y --- float64
X0 --- object
X1 --- object
X2 --- object
X3 --- object
X4 --- object
X5 --- object
X6 --- object
X8 --- object
X10 --- int64
X12 --- int64
X13 --- int64
X14 --- int64
X15 --- int64
X16 --- int64
X17 --- int64
X18 --- int64
X19 --- int64
```

In [28]:
```
category_cols = [c for c in train if train[c].dtype ==np.dtype('object')]
category_cols
```

Out[28]:
```
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

Apply Label Encoder

In [29]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [30]:
```
train[category_cols] = train[category_cols].apply(le.fit_transform)
```

In [31]:
```
# Features
train_feature = train.drop(['ID','y'], axis = 1)
# Target
train_target = train['y']
```

In [32]:
```
train_feature.head()
```

Out[32]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X
0	32	23	17	0	3	24	9	14	0	0	1	0	0	0	0	1	0	0	1	
1	32	21	19	4	3	28	11	14	0	0	0	0	0	0	0	1	0	0	0	
2	20	24	34	2	3	27	9	23	0	0	0	0	0	0	1	0	0	0	0	
3	20	21	34	5	3	27	11	4	0	0	0	0	0	0	0	0	0	0	0	
4	20	23	34	5	3	12	3	13	0	0	0	0	0	0	0	0	0	0	0	

In [33]:
```
train_target.head()
```

Out[33]:
```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

In [34]:
```
train_feature.shape
```

Out[34]: (4209, 364)

In [35]:
```
train_target.shape
```

Out[35]: (4209,)

In [36]:
```
train_feature.describe()
```

Out[36]:

	X0	X1	X2	X3	X4	X5	X6
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000
mean	29.760751	11.113566	17.306486	2.919696	2.997862	13.340223	6.807318
std	13.738338	8.531001	10.899914	1.739912	0.073900	8.250832	2.916973
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	19.000000	3.000000	8.000000	2.000000	3.000000	5.000000	6.000000
50%	35.000000	13.000000	16.000000	2.000000	3.000000	15.000000	7.000000
75%	43.000000	20.000000	25.000000	5.000000	3.000000	21.000000	9.000000
max	46.000000	26.000000	43.000000	6.000000	3.000000	28.000000	11.000000

Perfoming Dimensionality Reduction (PCA)

In [37]:
```
print(train_feature.shape)
print(train_target.shape)
```

```
(4209, 364)
(4209,)
```

In [38]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 0.95)
```

In [39]:
```
pca.fit(train_feature,train_target)
```

Out[39]:
```
PCA(n_components=0.95)
```

In [40]:
```
train_feature_pca = pca.fit_transform(train_feature)
```

In [41]:
```
train_feature_pca.shape
```

Out[41]:
```
(4209, 6)
```

Spliting Data

In [42]:
```
from sklearn.model_selection import train_test_split
```

In [43]:
```
train_x,test_x,train_y,test_y = train_test_split(train_feature_pca, train_target,
```

In [44]:
```
train_x.shape
```

Out[44]:
```
(3367, 6)
```

In [45]:
```
train_y.shape
```

Out[45]:
```
(3367,)
```

In [46]:
```
test_x.shape
```

Out[46]:
```
(842, 6)
```

In [47]:
```
test_y.shape
```

Out[47]:
```
(842,)
```

XGboost

In [48]:
```
import xgboost as xgb
```

In [49]:
```
xgb_reg = xgb.XGBRegressor(objective= 'reg:linear', colsample_bytree = 0.3,learni
```

In [50]:
```
model = xgb_reg.fit(train_x,train_y)
```

```
[17:10:26] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.6.
0/src/objective/regression_obj.cu:203: reg:linear is now deprecated in favor of
reg:squarederror.
```

In [51]:
```
y_pred = model.predict(test_x)
y_pre = model.predict(train_x)
```

In [52]:
```
y_pred.shape
```

Out[52]:
```
(842,)
```

In [53]:
```
y_pre.shape
```

Out[53]:
```
(3367,)
```

In [54]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [55]:
```
print('Train_Score=',model.score(train_x, train_y))
print('Test_Score=',model.score(test_x, test_y))
```

```
Train_Score= 0.5701760216727423
Test_Score= 0.25325418172617364
```

In [56]:
```
print(r2_score(test_y, y_pred))
print(r2_score(train_y, y_pre))
```

```
0.25325418172617364
0.5701760216727423
```

In [57]:
```
print('MAE=', mean_absolute_error(test_y, y_pred))
print('MSE=',mean_squared_error(test_y, y_pred))
print('RSME=',np.sqrt(mean_squared_error(test_y, y_pred)))
```

```
MAE= 8.044838375354322
MSE= 121.97190419641204
RSME= 11.044089106685623
```

Prediction on test data set

In [58]:
```
test.head()
```

Out[58]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X2
0	1	az	v	n	f	d	t	a	w	0	0	0	0	0	0	0	0	0	0	
1	2	t	b	ai	a	d	b	g	y	0	0	0	0	0	0	0	0	0	1	
2	3	az	v	as	f	d	a	j	j	0	0	0	0	1	0	0	0	0	0	
3	4	az	l	n	f	d	z	l	n	0	0	0	0	0	0	0	0	0	0	
4	5	w	s	as	c	d	y	i	m	0	0	0	0	1	0	0	0	0	0	

In [59]:
```
test.shape
```

Out[59]: (4209, 377)

In [60]:
```
test.drop(['X11','X93','X107','X233','X235','X268','X289','X290','X293','X297','X
```

In [61]:
```
test.shape
```

Out[61]: (4209, 365)

In [62]:
```
test.head()
```

Out[62]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	X2
0	1	az	v	n	f	d	t	a	w	0	0	0	0	0	0	0	0	0	0	
1	2	t	b	ai	a	d	b	g	y	0	0	0	0	0	0	0	0	1	0	
2	3	az	v	as	f	d	a	j	j	0	0	0	1	0	0	0	0	0	0	
3	4	az	l	n	f	d	z	l	n	0	0	0	0	0	0	0	0	0	0	
4	5	w	s	as	c	d	y	i	m	0	0	0	1	0	0	0	0	0	0	

In [63]:
```
test.isnull().sum().any()
```

Out[63]: False

In [64]:
```
test_feature = test.drop(columns={'ID'})
```

In [65]:
```
test_feature.shape
```

Out[65]: (4209, 364)

In [66]:
```
test_feature.head()
```

Out[66]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X
0	az	v	n	f	d	t	a	w	0	0	0	0	0	0	0	0	0	0	0	
1	t	b	ai	a	d	b	g	y	0	0	0	0	0	0	0	0	1	0	0	
2	az	v	as	f	d	a	j	j	0	0	0	1	0	0	0	0	0	0	0	
3	az	l	n	f	d	z	l	n	0	0	0	0	0	0	0	0	0	0	0	
4	w	s	as	c	d	y	i	m	0	0	0	1	0	0	0	0	0	0	0	

In [67]:
```
test_feature.describe(include='object')
```

Out[67]:

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	49	27	45	7	4	32	12	25
top	ak	aa	as	c	d	v	g	e
freq	432	826	1658	1900	4203	246	1073	274

In [68]:
```
category_cols1 = [c for c in test_feature if test_feature[c].dtype ==np.dtype('ob
category_cols1
```

Out[68]:
```
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

In [69]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [70]:
```
test_feature[category_cols1] = test_feature[category_cols1].apply(le.fit_transfor
```

In [71]:
```
test_feature.head()
```

Out[71]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X
0	21	23	34	5	3	26	0	22	0	0	0	0	0	0	0	0	0	0	0	
1	42	3	8	0	3	9	6	24	0	0	0	0	0	0	0	0	1	0	0	
2	21	23	17	5	3	0	9	9	0	0	0	1	0	0	0	0	0	0	0	
3	21	13	34	5	3	31	11	13	0	0	0	0	0	0	0	0	0	0	0	
4	45	20	17	2	3	30	8	12	0	0	0	1	0	0	0	0	0	0	0	

In [72]:
```
test_feature.dtypes.value_counts()
```

Out[72]:
```
int64    356
int32      8
dtype: int64
```

In [73]:
```
pca.fit(test_feature)
```

Out[73]:
```
PCA(n_components=0.95)
```

In [74]:
```
test_feature_trans = pca.fit_transform(test_feature)
```

In [75]:
```
test_feature_trans.shape
```

Out[75]:
```
(4209, 6)
```

In [76]:
```
test_feature_trans
```

Out[76]:
```
array([[ 14.58336183,  14.16672593,  13.53857566,   2.40835691,
          11.31942221,   6.94220721],
       [-15.25161267,  -7.73675643,  -7.45495068,  -2.66203503,
          11.59379316,   1.15940345],
       [ 11.8564649 ,  -1.68017324,  -9.9896148 ,  14.91886587,
          -1.08886021,  -2.69130553],
       ...,
       [-13.44644008,   3.2885825 ,  -6.85236431,  18.91025575,
          11.32365564,   3.22410016],
       [ 24.92612317,  -4.89888683, -10.16941028,  11.44337736,
           5.90178724,   4.55323232],
       [-15.38430989,  -7.73425491, -15.4930104 ,  -0.5595126 ,
           4.7793639 ,   1.0829113 ]])
```

In [77]:
```
test_pred = model.predict(test_feature_trans)
```

In [78]:
```
test_pred.shape
```

Out[78]:
```
(4209,)
```

In [79]:
```
test_pred
```

Out[79]:
```
array([ 98.24064 ,  99.245674,  99.70339 , ...,  96.97598 , 111.26956 ,
       102.940125], dtype=float32)
```

In [80]:
```
x = pd.DataFrame(test_pred)
```

In [81]:
```
x
```

Out[81]:

	0
0	98.240639
1	99.245674
2	99.703392
3	103.333298
4	106.152199
5	93.206100
6	91.223396
7	98.224739
8	114.123512
9	104.349297
10	114.255074

Thank you