Mercedes_Project_Final_Submission

March 13, 2021

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import math
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.feature_selection import VarianceThreshold
plt.rcParams['figure.figsize'] = (10,10)
```

Step 1. Import Data File

[2]:
```
merc_train_df = pd.read_csv('/home/labsuser/Datasets/merc_train.csv')
merc_train_df.shape
```

[2]:
```
(4209, 378)
```

[3]:
```
merc_test_df = pd.read_csv('/home/labsuser/Datasets/merc_test.csv')
merc_test_df.shape
```

[3]:
```
(4209, 377)
```

Step 2. Data Preprocessing

2.1 Check Top 5 Rows

[4]:
```
merc_train_df.head()
```

[4]:
```
   ID       y  X0 X1  X2 X3 X4 X5 X6 X8  …  X375  X376  X377  X378  X379  \
0   0  130.81   k  v  at  a  d  u  j  o  …     0     0     1     0     0
1   6   88.53   k  t  av  e  d  y  l  o  …     1     0     0     0     0
2   7   76.26  az  w   n  c  d  x  j  x  …     0     0     0     0     0
3   9   80.62  az  t   n  f  d  x  l  e  …     0     0     0     0     0
4  13   78.02  az  v   n  f  d  h  d  n  …     0     0     0     0     0

   X380  X382  X383  X384  X385
0     0     0     0     0     0
```

```
1      0      0      0      0      0
2      0      1      0      0      0
3      0      0      0      0      0
4      0      0      0      0      0

[5 rows x 378 columns]
```

[5]:
```
merc_test_df.head()
```

[5]:
```
   ID  X0 X1  X2 X3 X4 X5 X6 X8  X10  …  X375  X376  X377  X378  X379  X380  \
0   1  az  v   n  f  d  t  a  w    0  …     0     0     0     1     0     0
1   2   t  b  ai  a  d  b  g  y    0  …     0     0     1     0     0     0
2   3  az  v  as  f  d  a  j  j    0  …     0     0     0     1     0     0
3   4  az  l   n  f  d  z  l  n    0  …     0     0     0     1     0     0
4   5   w  s  as  c  d  y  i  m    0  …     1     0     0     0     0     0

   X382  X383  X384  X385
0     0     0     0     0
1     0     0     0     0
2     0     0     0     0
3     0     0     0     0
4     0     0     0     0

[5 rows x 377 columns]
```

2.2 Check Data Stats

[6]:
```
merc_train_df.describe()
```

[6]:
```
                ID            y          X10     X11          X12  \
count  4209.000000  4209.000000  4209.000000  4209.0  4209.000000
mean   4205.960798   100.669318     0.013305     0.0     0.075077
std    2437.608688    12.679381     0.114590     0.0     0.263547
min       0.000000    72.110000     0.000000     0.0     0.000000
25%    2095.000000    90.820000     0.000000     0.0     0.000000
50%    4220.000000    99.150000     0.000000     0.0     0.000000
75%    6314.000000   109.010000     0.000000     0.0     0.000000
max    8417.000000   265.320000     1.000000     0.0     1.000000

               X13          X14          X15          X16          X17  …  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000  …
mean      0.057971     0.428130     0.000475     0.002613     0.007603  …
std       0.233716     0.494867     0.021796     0.051061     0.086872  …
min       0.000000     0.000000     0.000000     0.000000     0.000000  …
25%       0.000000     0.000000     0.000000     0.000000     0.000000  …
50%       0.000000     0.000000     0.000000     0.000000     0.000000  …
75%       0.000000     1.000000     0.000000     0.000000     0.000000  …
```

```
max        1.000000     1.000000     1.000000     1.000000     1.000000  …

                X375         X376         X377         X378         X379  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean      0.318841     0.057258     0.314802     0.020670     0.009503
std       0.466082     0.232363     0.464492     0.142294     0.097033
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       1.000000     0.000000     1.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

                X380         X382         X383         X384         X385
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean      0.008078     0.007603     0.001663     0.000475     0.001426
std       0.089524     0.086872     0.040752     0.021796     0.037734
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     0.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

[8 rows x 370 columns]
```

[7]:
```python
merc_test_df.describe()
```

[7]:
```
                  ID          X10          X11          X12          X13  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean   4211.039202     0.019007     0.000238     0.074364     0.061060
std    2423.078926     0.136565     0.015414     0.262394     0.239468
min       1.000000     0.000000     0.000000     0.000000     0.000000
25%    2115.000000     0.000000     0.000000     0.000000     0.000000
50%    4202.000000     0.000000     0.000000     0.000000     0.000000
75%    6310.000000     0.000000     0.000000     0.000000     0.000000
max    8416.000000     1.000000     1.000000     1.000000     1.000000

                 X14          X15          X16          X17          X18  …  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000  …
mean      0.427893     0.000713     0.002613     0.008791     0.010216  …
std       0.494832     0.026691     0.051061     0.093357     0.100570  …
min       0.000000     0.000000     0.000000     0.000000     0.000000  …
25%       0.000000     0.000000     0.000000     0.000000     0.000000  …
50%       0.000000     0.000000     0.000000     0.000000     0.000000  …
75%       1.000000     0.000000     0.000000     0.000000     0.000000  …
max       1.000000     1.000000     1.000000     1.000000     1.000000  …

                X375         X376         X377         X378         X379  \
```

count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000
mean	0.325968	0.049656	0.311951	0.019244	0.011879
std	0.468791	0.217258	0.463345	0.137399	0.108356
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	0.000000	1.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

	X380	X382	X383	X384	X385
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000
mean	0.008078	0.008791	0.000475	0.000713	0.001663
std	0.089524	0.093357	0.021796	0.026691	0.040752
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

```
[8 rows x 369 columns]
```

2.3 Check for Null Values

[8]:
```
merc_train_df.isna().sum().sum()
#No Null Values Present in Train Data
```

[8]: 0

[9]:
```
merc_test_df.isna().sum().sum()
#No Null Values Present in Test Data
```

[9]: 0

2.4 Check Data Distribution and Unique Values

[10]:
```
# for col in merc_train_df.columns:
#     print("Unique values in {} Column are".format(col), merc_train_df[col].
↪unique())
```

Training Data Observation with above command * ID column is not useful as usual * 8 Columns are Categorical Columns * Most of the Numerical Columns has mix of 0's and 1's * Few of the columns has values only 0's * Target variable is continuous

[11]:
```
# for col in merc_test_df.columns:
#     print("Unique values in {} Column are".format(col), merc_test_df[col].
↪unique())
```

Observations are same as above but Test Data doesn't have the Target Value

2.5 Perform Label Encoding for Categorical Columns

[12]:
```
merc_train_df_1 = merc_train_df.copy()
```

[13]:
```
df_dtypes = merc_train_df_1.dtypes.reset_index()
df_dtypes.columns=['count', 'dtype']
grpd_df = df_dtypes.groupby('dtype').aggregate('count').reset_index()
grpd_df
```

[13]:

	dtype	count
0	int64	369
1	float64	1
2	object	8

[14]:
```
encoding_col_list = df_dtypes[df_dtypes.dtype==object]['count'].values
encoding_col_list
```

[14]:
```
array(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype=object)
```

[15]:
```
le = LabelEncoder()
```

[16]:
```
for column in encoding_col_list:
    merc_train_df_1[column] = le.fit_transform(merc_train_df_1[column])
    print("Training Data - Label Encoding for {} Completed Successfully".
 ↪format(column))
```

```
Training Data - Label Encoding for X0 Completed Successfully
Training Data - Label Encoding for X1 Completed Successfully
Training Data - Label Encoding for X2 Completed Successfully
Training Data - Label Encoding for X3 Completed Successfully
Training Data - Label Encoding for X4 Completed Successfully
Training Data - Label Encoding for X5 Completed Successfully
Training Data - Label Encoding for X6 Completed Successfully
Training Data - Label Encoding for X8 Completed Successfully
```

[17]:
```
merc_train_df_1.head(3)
```

[17]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	…	X375	X376	X377	X378	\
0	0	130.81	32	23	17	0	3	24	9	14	…	0	0	1	0	
1	6	88.53	32	21	19	4	3	28	11	14	…	1	0	0	0	
2	7	76.26	20	24	34	2	3	27	9	23	…	0	0	0	0	

	X379	X380	X382	X383	X384	X385
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	1	0	0	0

[3 rows x 378 columns]

[18]:
```
merc_test_df_1 = merc_test_df.copy()
```

[19]:
```
for column in encoding_col_list:
    merc_test_df_1[column] = le.fit_transform(merc_test_df_1[column])
    print("Testing Data - Label Encoding for {} Completed Successfully".
 ↪format(column))
```

```
Testing Data - Label Encoding for X0 Completed Successfully
Testing Data - Label Encoding for X1 Completed Successfully
Testing Data - Label Encoding for X2 Completed Successfully
Testing Data - Label Encoding for X3 Completed Successfully
Testing Data - Label Encoding for X4 Completed Successfully
Testing Data - Label Encoding for X5 Completed Successfully
Testing Data - Label Encoding for X6 Completed Successfully
Testing Data - Label Encoding for X8 Completed Successfully
```

[20]:
```
merc_test_df_1.head(3)
```

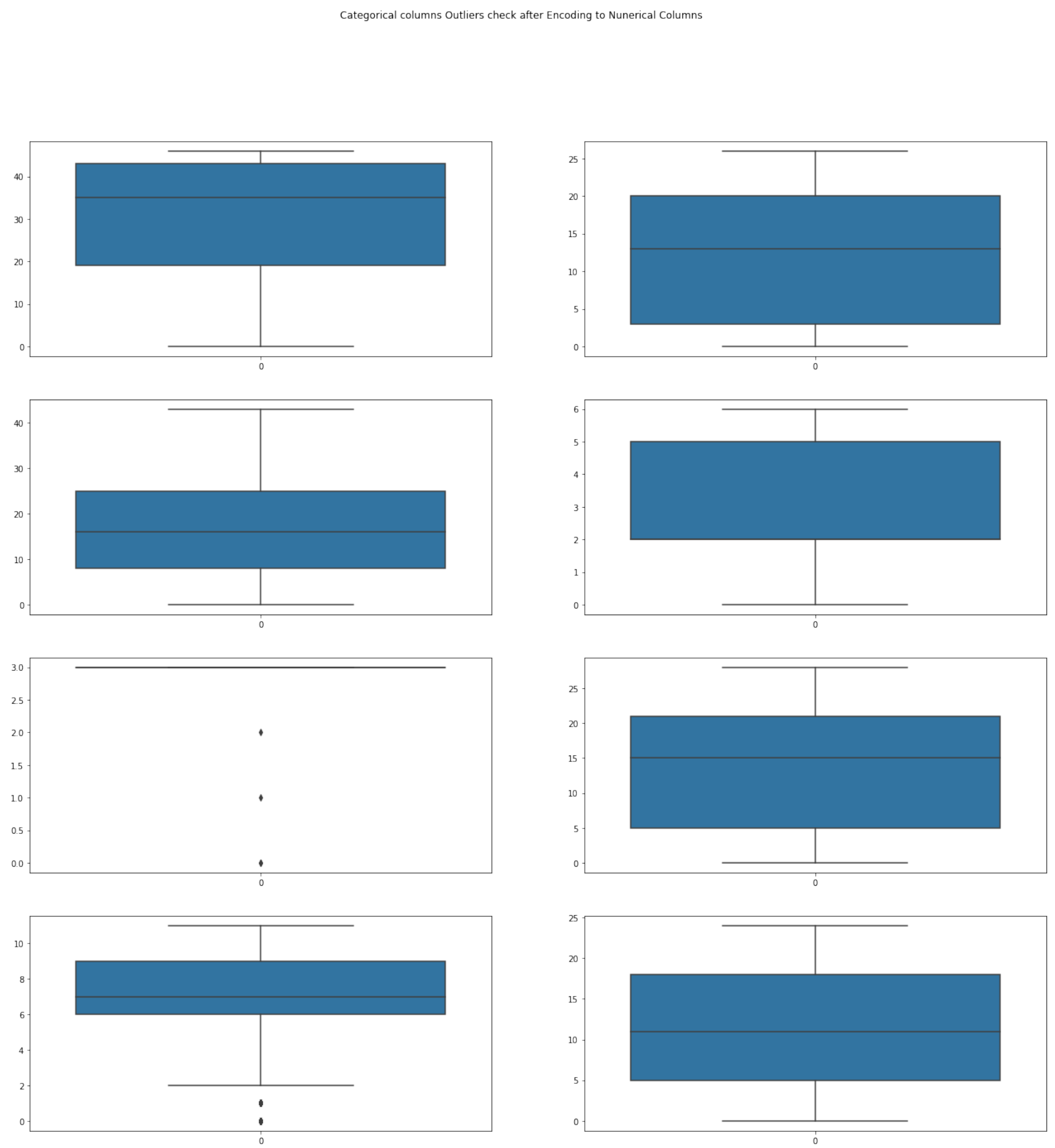
[20]:
```
   ID  X0  X1  X2  X3  X4  X5  X6  X8  X10  …  X375  X376  X377  X378  X379  \
0   1  21  23  34   5   3  26   0  22    0  …     0     0     0     1     0
1   2  42   3   8   0   3   9   6  24    0  …     0     0     1     0     0
2   3  21  23  17   5   3   0   9   9    0  …     0     0     0     1     0

   X380  X382  X383  X384  X385
0     0     0     0     0     0
1     0     0     0     0     0
2     0     0     0     0     0

 [3 rows x 377 columns]
```

2.6 Detect Outliers in the Data

2.6.1 Check Encoded Categorical Variables using boxplot

[21]:
```
fig, axes = plt.subplots(4, 2, figsize=(22, 22))
fig.suptitle('Categorical columns Outliers check after Encoding to Nunerical␣
 ↪Columns')
sns.boxplot(ax=axes[0, 0], data = merc_train_df_1['X0'])
sns.boxplot(ax=axes[0, 1], data = merc_train_df_1['X1'])
sns.boxplot(ax=axes[1, 0], data = merc_train_df_1['X2'])
sns.boxplot(ax=axes[1, 1], data = merc_train_df_1['X3'])
sns.boxplot(ax=axes[2, 0], data = merc_train_df_1['X4'])
sns.boxplot(ax=axes[2, 1], data = merc_train_df_1['X5'])
sns.boxplot(ax=axes[3, 0], data = merc_train_df_1['X6'])
sns.boxplot(ax=axes[3, 1], data = merc_train_df_1['X8'])
```

[21]:
```
<AxesSubplot:>
```



[22]:
```
#Checking values for X4 column where outliers are observed
print(merc_train_df_1[merc_train_df_1['X4']==3].shape)
print(merc_train_df_1[merc_train_df_1['X4']<3].shape)
```

```
(4205, 378)
(4, 378)
```

[23]:
```
#Removing 4 Outliers Entries
merc_train_df_1 = merc_train_df_1[merc_train_df_1['X4']==3]
```

[24]:
```
#Checking values for X6 column where outliers are observed
print(merc_train_df_1[merc_train_df_1['X6']>=2].shape)
print(merc_train_df_1[merc_train_df_1['X6']<2].shape)
```

```
(3972, 378)
(233, 378)
```

[25]:
```
#Removing 233 Outiers Entries
merc_train_df_1 = merc_train_df_1[merc_train_df_1['X6']>=2]
```

[26]:
```
merc_train_df_1.shape
```

[26]:
```
(3972, 378)
```

2.6.2 Check Numerical Columns Distribution using bar graph

[27]:
```
num_col_list = merc_train_df_1.iloc[:, 10:].columns
num_col_list
```

[27]:
```
Index(['X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=368)
```

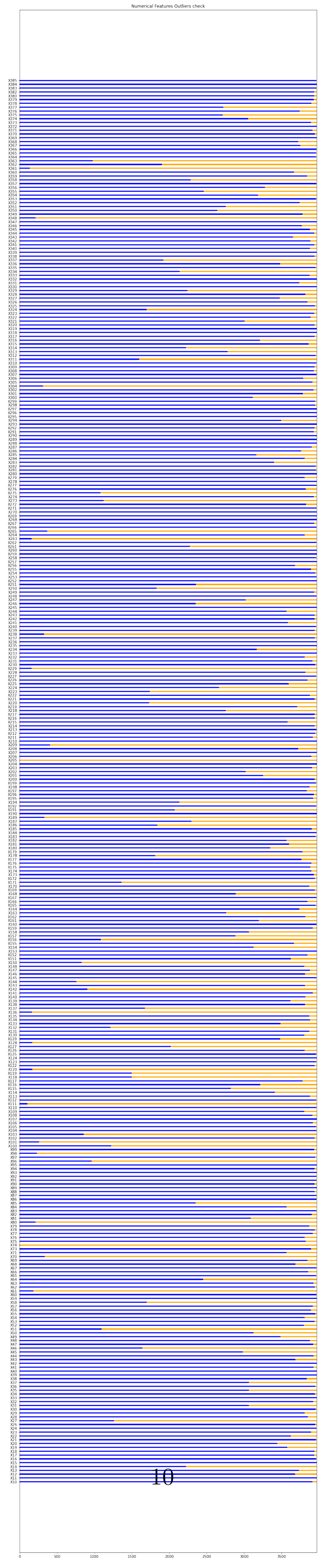
[28]:
```
zeroes_ones_dict = {}
for colname in num_col_list:
    zeroes = merc_train_df_1[merc_train_df_1[colname]==0].shape[0]
    ones = merc_train_df_1[merc_train_df_1[colname]==1].shape[0]
    zeroes_ones_dict[colname] = [zeroes, ones]
zeroes_ones_df = pd.DataFrame(zeroes_ones_dict)
zeroes_ones_df = zeroes_ones_df.T
zeroes_ones_df.columns = ['Zeroes', 'Ones']
zeroes_ones_df =  zeroes_ones_df.reset_index()
zeroes_ones_df
```

[28]:

	index	Zeroes	Ones
0	X10	3916	56
1	X11	3972	0
2	X12	3683	289
3	X13	3733	239
4	X14	2224	1748
..	…	…	…
363	X380	3939	33
364	X382	3940	32
365	X383	3966	6
366	X384	3970	2
367	X385	3967	5

```
[368 rows x 3 columns]
```

[29]:
```
width=0.3
plt.figure(figsize=(15,80))
plt.title('Numerical Features Outliers check')
plt.barh(zeroes_ones_df['index'], zeroes_ones_df.Zeroes, width, color='blue')
plt.barh(zeroes_ones_df['index'], zeroes_ones_df.Ones, width,␣
↪left=zeroes_ones_df.Zeroes, color="orange")
plt.show()
```


Numerical Features Outliers check
0
500
1000
1500
2000
2500
3000
3500

2.6.3 Remove Columns with Zero Variance

[30]:
```
train_column_var = merc_train_df_1.var()
var_0_list = list(train_column_var[train_column_var==0.0].index)
var_0_list.append('ID')
var_0_list
```

[30]:
```
['X4',
 'X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X270',
 'X289',
 'X290',
 'X293',
 'X297',
 'X330',
 'X347',
 'ID']
```

[31]:
```
merc_train_df_1.drop(var_0_list,axis=1,inplace=True)
```

[32]:
```
merc_train_df_1.shape
```

[32]:
```
(3972, 363)
```

[ ]:
```

```

[33]:
```
merc_test_df_1.drop(var_0_list,axis=1,inplace=True)
```

[34]:
```
merc_test_df_1.shape
```

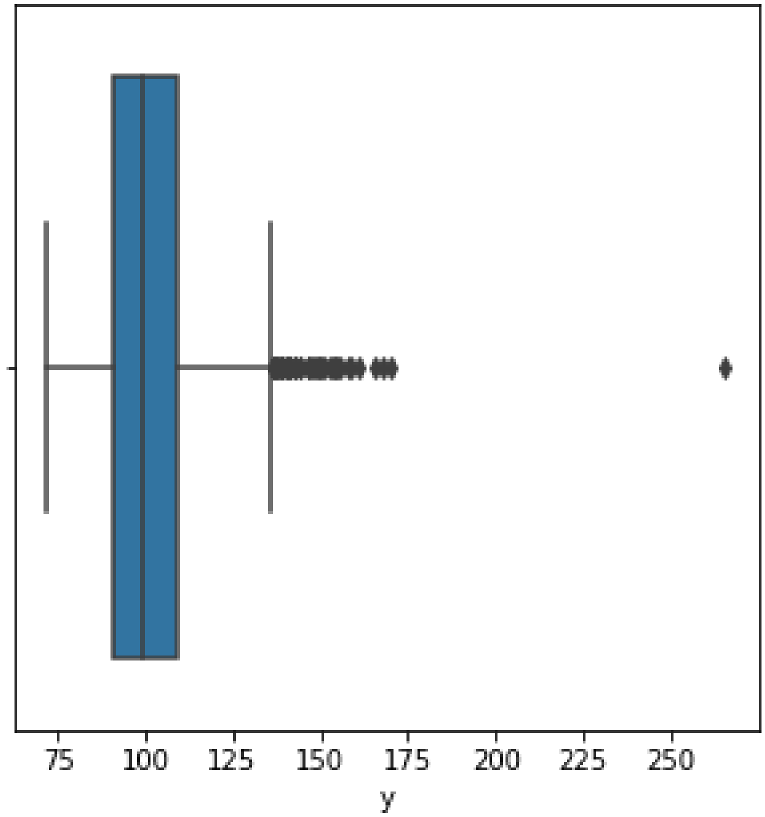
[34]:
```
(4209, 362)
```

[ ]:
```

```

2.6.4 Check Outliers in Dependent Variable

[35]:
```
plt.figure(figsize=(5,5))
sns.boxplot(merc_train_df_1['y'])
```

[35]:
```
<AxesSubplot:xlabel='y'>
```



[ ]:

[36]:
```
def outlier_treatment(datacolumn):
    sorted(datacolumn)
    Q1,Q3 = np.percentile(datacolumn,[25,75])
    IQR = Q3-Q1
    lower_range = Q1 - (1.5*IQR)
    upper_range = Q3 + (1.5*IQR)
    return lower_range,upper_range
```

[37]:
```
l,u = outlier_treatment(merc_train_df_1['y'])
print(l)
print(u)
```

```
63.710000000000036
136.28999999999996
```

[38]:
```
merc_train_df_1[(merc_train_df_1['y'] > u) | (merc_train_df_1['y'] < l)].shape
```

[38]: (48, 363)

[39]:
```
#Removing the Outliers Observed
merc_train_df_1.drop(merc_train_df_1[(merc_train_df_1['y'] > u) |
 ↪(merc_train_df_1['y'] < l)].index,inplace=True)
merc_train_df_1.shape
```

[39]: (3924, 363)

2.7 Assign features (Independent) and target (Dependent) Variable

Train Data

[40]:
```
merc_train_df_1.head(2)
```

[40]:
```
        y  X0  X1  X2  X3  X5  X6  X8  X10  X12  …  X375  X376  X377  X378  \
0  130.81  32  23  17   0  24   9  14    0    0  …     0     0     1     0
1   88.53  32  21  19   4  28  11  14    0    0  …     1     0     0     0

   X379  X380  X382  X383  X384  X385
0     0     0     0     0     0     0
1     0     0     0     0     0     0

[2 rows x 363 columns]
```

[41]:
```
features = merc_train_df_1.drop('y',axis=1)
```

[42]:
```
features.shape
```

[42]: (3924, 362)

[43]:
```
target = merc_train_df_1['y']
target.shape
```

[43]: (3924,)

Test Data

[44]:
```
merc_test_df_1.head(2)
```

[44]:
```
   X0  X1  X2  X3  X5  X6  X8  X10  X12  X13  …  X375  X376  X377  X378  \
0  21  23  34   5  26   0  22    0    0    0  …     0     0     0     1
1  42   3   8   0   9   6  24    0    0    0  …     0     0     1     0

   X379  X380  X382  X383  X384  X385
0     0     0     0     0     0     0
1     0     0     0     0     0     0

[2 rows x 362 columns]
```

[45]:
```
features_test = merc_test_df_1.copy()
```

[46]:
```python
features_test.shape
```

[46]: (4209, 362)

2.8 Perform Train Test Split of the data

[47]:
```python
from sklearn.model_selection import train_test_split
```

[48]:
```python
X_tr_train, X_tr_test, y_tr_train, y_tr_test =␣
 ↪train_test_split(features,target, test_size=0.30, random_state=42)
```

[49]:
```python
print(X_tr_train.shape)
print(X_tr_test.shape)
print(y_tr_train.shape)
print(y_tr_test.shape)
```

```
(2746, 362)
(1178, 362)
(2746,)
(1178,)
```

2.9 Perform Standardization using Standard Scaler

[50]:
```python
from sklearn.preprocessing import StandardScaler
```

[51]:
```python
scaler = StandardScaler()
```

Train Data With Train Test Split

[52]:
```python
X_tr_train_scaled = scaler.fit_transform(X_tr_train)
```

[53]:
```python
X_tr_test_scaled = scaler.fit_transform(X_tr_test)
```

[54]:
```python
print(X_tr_train_scaled.shape)
print(X_tr_test_scaled.shape)
```

```
(2746, 362)
(1178, 362)
```

[ ]:
```python

```

Train Data Without Train Test Split

[55]:
```python
features_train_scaled = scaler.fit_transform(features)
features_train_scaled.shape
```

[55]: (3924, 362)

[ ]:
```python

```

Test Data

```
[56]: features_test_scaled = scaler.fit_transform(features_test)
features_test_scaled.shape
```

```
[56]: (4209, 362)
```

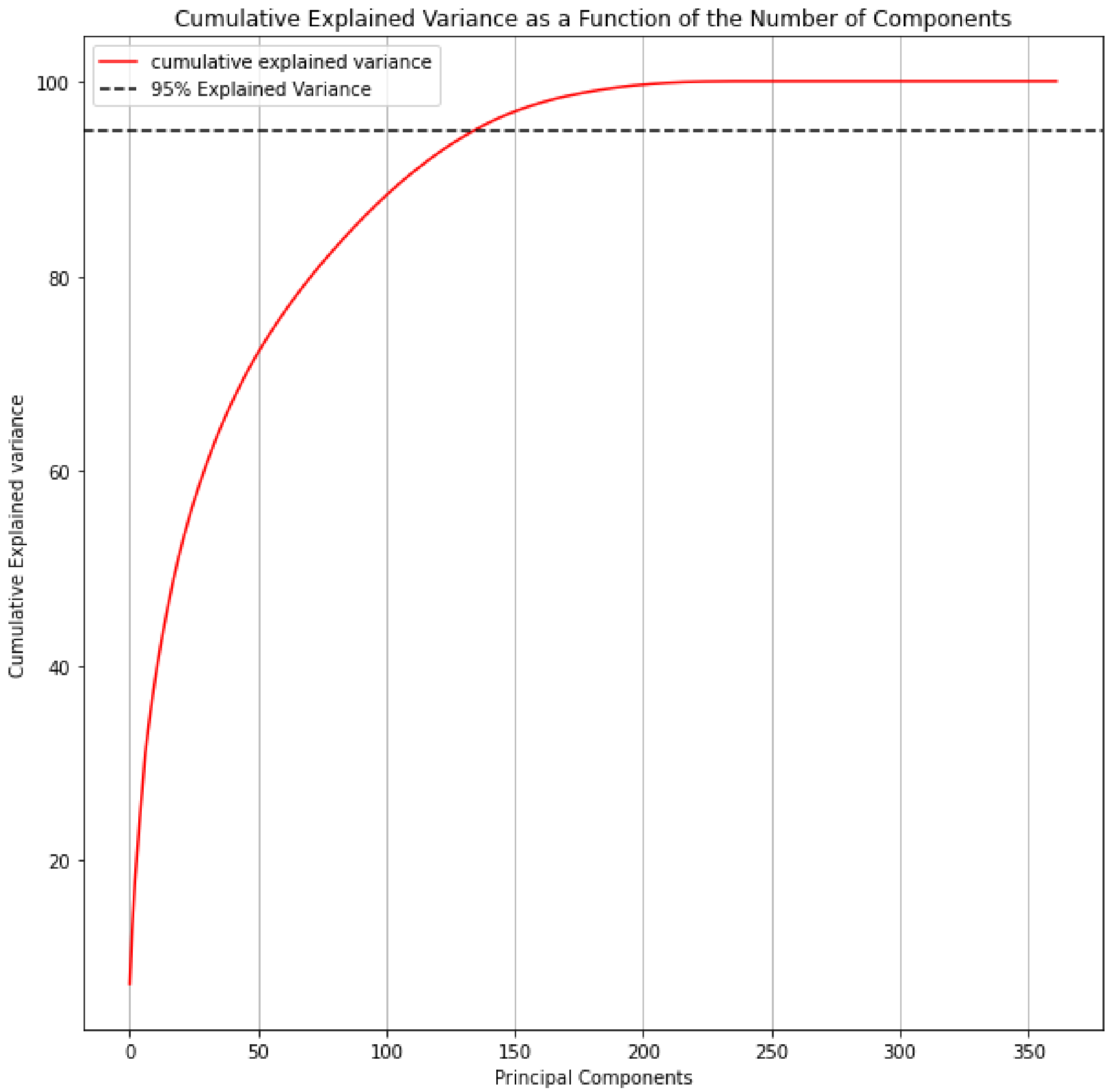
```
[ ]:
```

Step 3. Feature Engineering

Applying Principal Component Analysis (PCA)

```
[57]: from sklearn.decomposition import PCA
pca = PCA()
pca.fit(X_tr_train_scaled)
cumsum = np.cumsum(pca.explained_variance_ratio_)*100
d = [n for n in range(len(cumsum))]
plt.figure(figsize=(10, 10))
plt.plot(d,cumsum, color = 'red',label='cumulative explained variance')
plt.title('Cumulative Explained Variance as a Function of the Number of␣
 ↪Components')
plt.ylabel('Cumulative Explained variance')
plt.grid(axis='x')
plt.xlabel('Principal Components')
plt.axhline(y = 95, color='k', linestyle='--', label = '95% Explained Variance')
plt.legend(loc='best')
```

```
[57]: <matplotlib.legend.Legend at 0x7f0bda8daa50>
```



[58]:
```
pca = PCA(n_components=0.95)
```

Applying PCA for Train Data Split

[59]:
```
pca.fit(X_tr_train_scaled)
```

[59]:
```
PCA(copy=True, iterated_power='auto', n_components=0.95, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

[60]:
```
pca.explained_variance_ratio_.size
```

[60]:
```
136
```

Train Data Transform - With Train Test Split

[61]:
```
X_tr_train_pca = pca.transform(X_tr_train_scaled)
X_tr_train_pca.shape
```

[61]: (2746, 136)

[62]:
```
X_tr_test_pca = pca.transform(X_tr_test_scaled)
X_tr_test_pca.shape
```

[62]: (1178, 136)

[ ]:

Test Data Transform

[63]:
```
features_test_pca = pca.transform(features_test_scaled)
features_test_pca.shape
```

[63]: (4209, 136)

[ ]:

Step 4. Build Models

#1: Apply Multiple Linear Regression on Reduced Dimesions

[64]:
```
model_lr = LinearRegression()
```

[65]:
```
model_lr.fit(X_tr_train_pca,y_tr_train)
```

[65]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

[66]:
```
y_pred_lr = model_lr.predict(X_tr_test_pca)
```

[67]:
```
r2_score(y_tr_test, y_pred_lr)
```

[67]: 0.5801537981789575

#2: Apply Lasso Regression on Reduced Dimesions

[68]:
```
from sklearn.linear_model import Lasso
```

[69]:
```
model_lasso = Lasso(alpha=0.01)
```

[70]:
```
model_lasso.fit(X_tr_train_pca,y_tr_train)
```

[70]:
```
Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

[71]:
```
y_pred_lasso = model_lasso.predict(X_tr_test_pca)
```

[72]:
```
r2_score(y_tr_test, y_pred_lasso)
```

[72]:
```
0.5815549312956898
```

[ ]:
```

```

#3 Apply K Nearest Neighbors Regressor Algorithm

[73]:
```
from sklearn.neighbors import KNeighborsRegressor
```

[74]:
```
model_knn = KNeighborsRegressor(n_neighbors=5)
```

[75]:
```
model_knn.fit(X_tr_train_pca,y_tr_train)
```

[75]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

[76]:
```
y_pred_knn = model_knn.predict(X_tr_test_pca)
```

[77]:
```
r2_score(y_tr_test, y_pred_knn)
```

[77]:
```
0.45545247129565414
```

[ ]:
```

```

#4 Apply Decision Tree Regressor Algorithm

[78]:
```
from sklearn.tree import DecisionTreeRegressor
```

[79]:
```
model_dtr = DecisionTreeRegressor(random_state=2)
```

[80]:
```
model_dtr.fit(X_tr_train_pca,y_tr_train)
```

[80]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=2, splitter='best')
```

[81]:
```
y_pred_dtr = model_dtr.predict(X_tr_test_pca)
```

[82]:
```
r2_score(y_tr_test, y_pred_dtr)
```

[82]:
```
-0.020689970825636506
```

[ ]:

Step 5. Ensemble Techniques

#1 Apply Random Forest Regressor Algorithm

[83]:
```
from sklearn.ensemble import RandomForestRegressor
```

[84]:
```
model_rfr = RandomForestRegressor(n_estimators=100, random_state=2)
```

[85]:
```
model_rfr.fit(X_tr_train_pca,y_tr_train)
```

[85]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=2, verbose=0, warm_start=False)
```

[86]:
```
y_pred_rfr = model_rfr.predict(X_tr_test_pca)
```

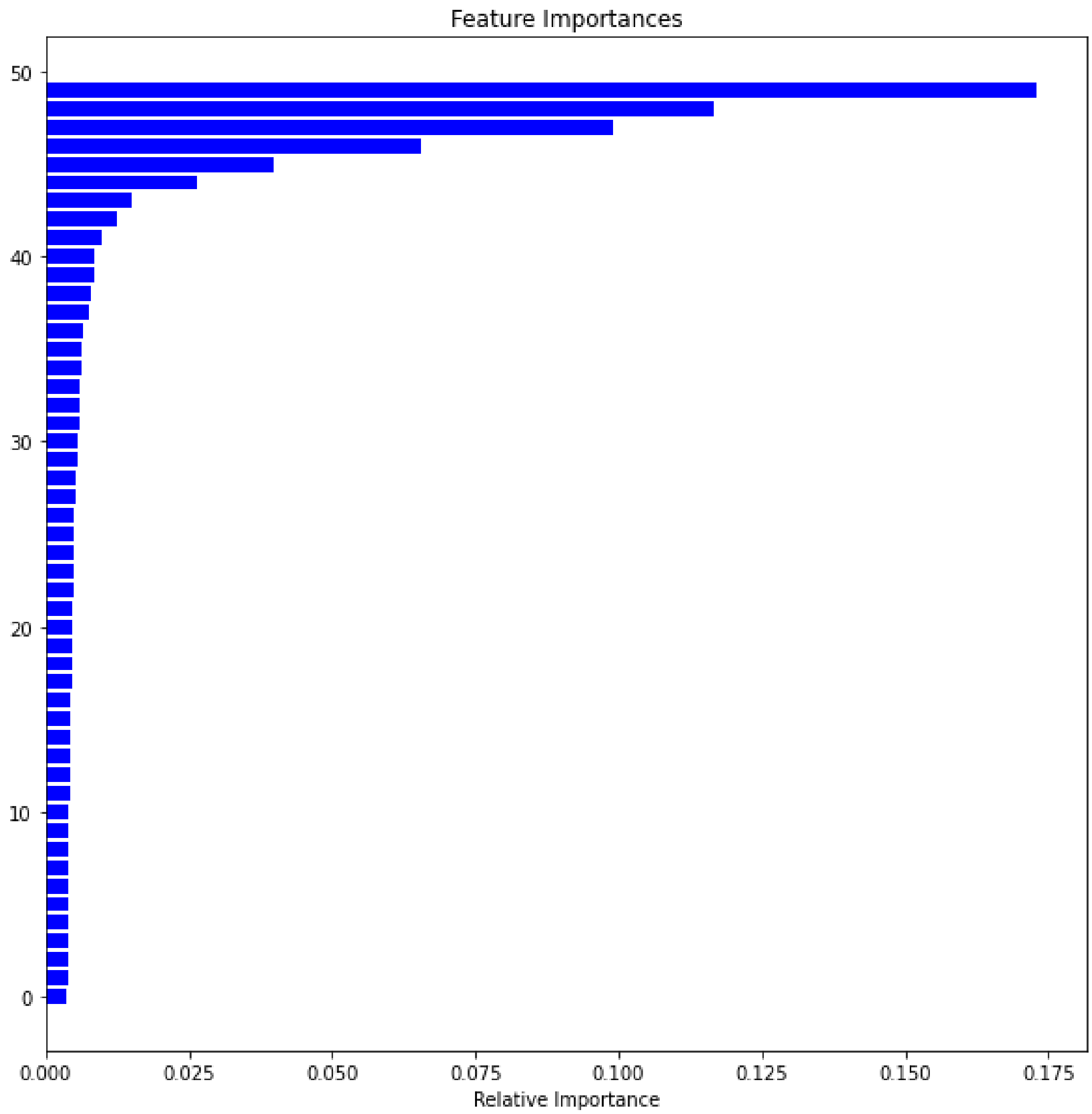
[87]:
```
r2_score(y_tr_test, y_pred_rfr)
```

[87]:
```
0.5274245940408445
```

[88]:
```
#0.5274245940408445 - 100 estimators
#0.5295699851806822 - 200 estimators
```

Importance of Features for Random Forest Regressor in the PCA applied X Train Data

[89]:
```
# features = X_tr_train.columns   --- Which columns are picked in PCA, order of␣
 ↪columns, we are not aware - So not plotting Features Names
importances = model_rfr.feature_importances_
indices = np.argsort(importances)[-50:]  # top 50 features
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
# plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



[ ]:

#2 Apply XGBoost Techniques

[90]:
```python
import xgboost as xgb
```

[91]:
```python
from sklearn.model_selection import RandomizedSearchCV
# model_xgbr = xgb.XGBRFRegressor()
```

Even with RandomizedSearchCV best estimator or params - XGBRFRegression is giving negative r2 score

[92]:
```
# params = {
#     'learning_rate':[0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
#     'max_depth':[3, 4, 5, 6, 8, 10, 12, 15, 20, 25, 30, 35],
#     'min_child_weight':[1, 3, 5, 7, 9, 11, 13, 15],
#     'gamma':[0.0, 0.1, 0.2, 0.3, 0.4],
#     'colsample_bytree': [0.3, 0.4, 0.5, 0.7, 0.8],
#     'colsample_bylevel':[0.3, 0.4, 0.5, 0.7, 0.8],
#     'colsample_bynode':[0.3, 0.4, 0.5, 0.7, 0.8],
#     'n_estimators': [100, 150, 200, 250, 300],
#     'subsample':[0.3, 0.4, 0.5, 0.7, 0.8],
#     'base_score': [0.1, 0.2, 0.3, 0.5, 0.7, 0.9],
#     'booster': ['gblinear', 'gbtree', 'dart'],
#     'num_parallel_tree': [100, 150, 200, 250, 300, 350],
#     'random_state': [0, 5, 10, 15, 20],
#     'scale_pos_weight': [0.1, 0.3, 0.5, 0.7, 0.9]
# }
```

[93]:
```
# model_rsv = RandomizedSearchCV(model_xgbr, param_distributions=params,␣
 ↪n_iter=5, scoring='r2', n_jobs=1, cv=5, verbose=3)
#model_rsv.fit(X_tr_train_pca,y_tr_train)
# model_rsv.best_estimator_
# model_rsv.best_params_
```

[94]:
```
# model_rsv = RandomizedSearchCV(model_xgbreg, param_distributions=params,␣
 ↪n_iter=5, scoring='r2', n_jobs=1, cv=5, verbose=3)
# model_rsv.fit(X_tr_train_pca,y_tr_train)
```

2.1 Apply XGBRFRegressor without RandomizedSearchCV suggested params

[95]:
```
model_xgbr = xgb.XGBRFRegressor(booster='gblinear', learning_rate=1,␣
 ↪n_estimators=550)
```

[96]:
```
model_xgbr.fit(X_tr_train_pca,y_tr_train)
```

[96]:
```
XGBRFRegressor(base_score=0.5, booster='gblinear', colsample_bylevel=None,
               colsample_bynode=0.8, colsample_bytree=None, gamma=None,
               gpu_id=-1, importance_type='gain', interaction_constraints=None,
               learning_rate=1, max_delta_step=None, max_depth=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               n_estimators=550, n_jobs=0, num_parallel_tree=None,
               objective='reg:squarederror', random_state=0, reg_alpha=0,
               reg_lambda=1e-05, scale_pos_weight=1, subsample=0.8,
               tree_method=None, validate_parameters=False, verbosity=None)
```

[97]:
```
y_pred_xgbr = model_xgbr.predict(X_tr_test_pca)
```

[98]:
```
r2_score(y_tr_test, y_pred_xgbr)
```

[98]: 0.5801253507635217

r2 scores for XGBRFRegressor with different Params combinations

[99]:
```
#booster='gblinear', learning_rate=1, n_estimators=550 - 0.5801333198113591 ␣
↪--- Best Params

####Model is performing worse (getting -ve r2 score) if learning rate is less␣
↪than 1####
```

Importance of Features for XGBRFRegressor in the PCA applied X Train Data

[100]:
```
# AttributeError: Feature importance is not defined for Booster type gblinear



# # features = X_tr_train.columns   --- Which columns are picked in PCA, order␣
↪of columns, we are not aware - So not plotting Features Names
# importances = model_xgbr.feature_importances_
# indices = np.argsort(importances)[-50:]  # top 50 features
# plt.title('Feature Importances')
# plt.barh(range(len(indices)), importances[indices], color='b', align='center')
# # plt.yticks(range(len(indices)), [features[i] for i in indices])
# plt.xlabel('Relative Importance')
# plt.show()
```

[ ]:
```

```

2.2 Apply XGBRegressor without RandomizedSearchCV suggested params

[101]:
```
model_xgbreg = xgb.XGBRegressor(booster='gblinear', learning_rate=0.01,␣
↪random_state=2, n_estimators=900)
```

[102]:
```
model_xgbreg.fit(X_tr_train_pca,y_tr_train)
```

[102]:
```
XGBRegressor(base_score=0.5, booster='gblinear', colsample_bylevel=None,
             colsample_bynode=None, colsample_bytree=None, gamma=None,
             gpu_id=-1, importance_type='gain', interaction_constraints=None,
             learning_rate=0.01, max_delta_step=None, max_depth=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=900, n_jobs=0, num_parallel_tree=None,
             objective='reg:squarederror', random_state=2, reg_alpha=0,
             reg_lambda=0, scale_pos_weight=1, subsample=None, tree_method=None,
             validate_parameters=False, verbosity=None)
```

[103]:
```
y_pred_xgbreg = model_xgbreg.predict(X_tr_test_pca)
```

[104]:
```
r2_score(y_tr_test, y_pred_xgbreg)
```

[104]: 0.580113426895613

[ ]:

r2 scores for XGBRegressor with different Params combinations

[105]:
```
#booster='gbtree', learning_rate=0.01, random_state=2, n_estimators=650  - 0.
↪55611427609201  --- Best Params for gbtree

####Descreasing from 800 for gbtree


# booster='gblinear', learning_rate=0.01, random_state=2, n_estimators=900  - 0.
↪5801134272878716   ---- Best Params for gblinear

#####Even for larger increase r2 is increasing sligthlt after n_estimator=1000
```

[ ]:

2.3 Apply XGBModel

[106]:
```
import xgboost as xgb
```

[107]:
```
d_tr_train = xgb.DMatrix(data=X_tr_train_pca, label=y_tr_train)
```

[108]:
```
d_tr_test = xgb.DMatrix(data=X_tr_test_pca, label=y_tr_test)
```

[109]:
```
xgbparam = {'eta':0.1, 'objective':'reg:squarederror' }
```

[110]:
```
model_xgbmod = xgb.train(xgbparam, d_tr_train, num_boost_round=80)
```

[111]:
```
y_pred_xgbmod = model_xgbmod.predict(d_tr_test)
```

[112]:
```
r2_score(y_tr_test, y_pred_xgbmod)
```

[112]: 0.5498822937983738

r2 scores for XGBModel with different Params combinations

[113]:
```
# 'eta':0.1, num_boost_round=41   --  0.5357897395464993

# 'eta':0.1, num_boost_round=50   --  0.5487014627426428

# 'eta':0.1, num_boost_round=80   --  0.5498822937983738    --- Best r2 score

# 'eta':0.05, num_boost_round=200  - 0.5452391643606216
```

Verifying with Cross Validation Techniques

Applying PCA for Complete Train Data without Split

```
[114]: pca.fit(features_train_scaled)
       pca.explained_variance_ratio_.size
```

```
[114]: 145
```

Complete Train Data Transform

```
[115]: features_train_pca = pca.transform(features_train_scaled)
       features_train_pca.shape
```

```
[115]: (3924, 145)
```

Complete Train Data Transform

```
[116]: features_test_pca = pca.transform(features_test_scaled)
       features_test_pca.shape
```

```
[116]: (4209, 145)
```

```
[117]: from sklearn.model_selection import cross_val_score
```

Cross Validation for Random Forest Regressor

```
[118]: score_rfr = cross_val_score(model_rfr, features_train_pca, target, cv=5)
       print(score_rfr)
       print("Avergae Score is", score_rfr.mean())
```

```
[0.52677099 0.53974165 0.50936533 0.49776623 0.55030036]
Avergae Score is 0.5247889139891366
```

Cross Validation for Lasso Regression

```
[119]: score_lasso = cross_val_score(model_lasso, features_train_pca, target, cv=10)
       print(score_lasso)
       print("Avergae Score is", score_lasso.mean())
```

```
[0.65216063 0.60686744 0.58726428 0.59925705 0.60993274 0.47009923
 0.45956763 0.62917372 0.5466997  0.47382519]
Avergae Score is 0.563484760205291
```

Cross Validation for XGBRFRegressor

```
[120]: score_xgbrf = cross_val_score(model_xgbr, features_train_pca, target, cv=10)
       print(score_xgbrf)
       print("Avergae Score is", score_xgbrf.mean())
```

```
[0.63289467 0.60235251 0.58036717 0.59732555 0.62567197 0.48609073
 0.54069294 0.61889123 0.6292286  0.6186795 ]
Avergae Score is 0.5932194872615244
```

Cross Validation for XGBRegressor

[121]:
```
score_xgbreg = cross_val_score(model_xgbreg, features_train_pca, target, cv=10)
print(score_xgbreg)
print("Avergae Score is", score_xgbreg.mean())
```

```
[-0.29939695  0.49298066  0.40307513  0.4364044   0.1873739  -0.29531965
  0.30564547  0.3590198  -0.39473727  0.1214718 ]
Avergae Score is 0.1316517298540869
```

Cross Validation for XGBModel

[122]:
```
from sklearn.model_selection import KFold
X = pd.DataFrame(features_train_pca)
Y = pd.DataFrame(target).reset_index(drop=True)
r2_score_arr = []
kf = KFold(n_splits=5, random_state=None)
kf.get_n_splits(features_train_pca, target)
for train_index,test_index in kf.split(features_train_pca, target):
#     print("Train Index:", train_index, "Validation", test_index)
    X1_train, X1_test = X.iloc[train_index], X.iloc[test_index]
    y1_train, y1_test = Y.iloc[train_index], Y.iloc[test_index]

    d_train = xgb.DMatrix(data=X1_train, label=y1_train['y'].values)
    d_test = xgb.DMatrix(data=X1_test, label=y1_test['y'].values)

    xgbparam = {'eta':0.1, 'objective':'reg:squarederror' }
    model_kfxgb = xgb.train(xgbparam, d_train, num_boost_round=80)
    y_pred_kfold = model_kfxgb.predict(d_test)
    r2score = r2_score(y1_test, y_pred_kfold)
    r2_score_arr.append(r2score)
np.array(r2_score_arr).mean()
```

[122]: 0.5604749318243738

Step 6. Evaluate Model

[123]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

Linear Regression Algorithm Metrics Evaluation - With Test Train Split

[124]:
```
print("R2score is", r2_score(y_tr_test, y_pred_lr))
print("RMSE is", np.sqrt(mean_squared_error(y_tr_test, y_pred_lr)))
```

```
R2score is 0.5801537981789575
RMSE is 7.360733591629869
```

Lasso Regression Algorithm Metrics Evaluation - With Test Train Split vs KFold CV

[125]:
```python
print("R2score is", r2_score(y_tr_test, y_pred_lasso))
print("RMSE is", np.sqrt(mean_squared_error(y_tr_test, y_pred_lasso)))
```

```
R2score is 0.5815549312956898
RMSE is 7.34844101101138
```

[126]:
```python
print("R2 score with KFold CV is", score_lasso.mean())
```

```
R2 score with KFold CV is 0.563484760205291
```

K Nearest Neighbors Regressor Algorithm Metrics Evaluation - With Train Test Split

[127]:
```python
print("R2score is", r2_score(y_tr_test, y_pred_knn))
print("RMSE is", np.sqrt(mean_squared_error(y_tr_test, y_pred_knn)))
```

```
R2score is 0.45545247129565414
RMSE is 8.382892191079792
```

Decision Tree Regressor Algorithm Metrics Evaluation - With Train Test Split

[128]:
```python
print("R2score is", r2_score(y_tr_test, y_pred_dtr))
print("RMSE is", np.sqrt(mean_squared_error(y_tr_test, y_pred_dtr)))
```

```
R2score is -0.020689970825636506
RMSE is 11.476855374689661
```

Random Forest Ensemble Alogorithm Metrics Evaluation - With Train Test Split vs KFold CV

[129]:
```python
print("R2 score with Train Test Split is", r2_score(y_tr_test, y_pred_rfr))
print("RMSE is", np.sqrt(mean_squared_error(y_tr_test, y_pred_rfr)))
```

```
R2 score with Train Test Split is 0.5274245940408445
RMSE is 7.809289864510982
```

[130]:
```python
print("R2 score with KFold CV is", score_rfr.mean())
```

```
R2 score with KFold CV is 0.5247889139891366
```

XGBRFRegressor Metrics Evaluation - With Train Test Split vs KFold CV

[131]:
```python
print("R2 score with Train Test Split is",r2_score(y_tr_test, y_pred_xgbr))
print("RMSE is", np.sqrt(mean_squared_error(y_tr_test, y_pred_xgbr)))
```

```
R2 score with Train Test Split is 0.5801253507635217
RMSE is 7.360982957110185
```

[132]:
```python
print("R2 score with KFold CV is", score_xgbrf.mean())
```

```
R2 score with KFold CV is 0.5932194872615244
```

XGBRegressor Metrics Evaluation - With Train Test Split vs KFold CV

[133]:
```
print("R2 score with Train Test Split is", r2_score(y_tr_test, y_pred_xgbreg))
print("RMSE is", np.sqrt(mean_squared_error(y_tr_test, y_pred_xgbreg)))
```

```
R2 score with Train Test Split is 0.580113426895613
RMSE is 7.361087477311012
```

[134]:
```
print("R2 score with KFold CV is",score_xgbreg.mean())
```

```
R2 score with KFold CV is 0.1316517298540869
```

XGBoost Metrics Evaluation - With Train Test Split vs KFold CV

[135]:
```
print("R2 score with Train Test Split is", r2_score(y_tr_test, y_pred_xgbmod))
print("RMSE is", np.sqrt(mean_squared_error(y_tr_test, y_pred_xgbmod)))
```

```
R2 score with Train Test Split is 0.5498822937983738
RMSE is 7.621475075766128
```

[136]:
```
print("R2 score with KFold CV is", np.array(r2_score_arr).mean())
```

```
R2 score with KFold CV is 0.5604749318243738
```

With KFold Cross Validation XGBoost and XGBRFRegressor is performing better

Prediction with XGBoost

[137]:
```
d_train_matx = xgb.DMatrix(data=features_train_pca, label=target)
d_test_matx = xgb.DMatrix(data=features_test_pca)
```

[138]:
```
xgbparam = {'eta':0.1, 'objective':'reg:squarederror'}
```

[139]:
```
model_xgb_compl = xgb.train(xgbparam, d_train_matx, num_boost_round=80)
```

[140]:
```
y_pred_test_vals = model_xgb_compl.predict(d_test_matx)
```

[141]:
```
y_pred_test_vals
```

[141]:
```
array([ 84.9248 , 106.31748,  84.7337 , …,  89.16706, 109.81949,
        93.89433], dtype=float32)
```

Prediction with XGBRFRegressor

[142]:
```
model_xgbrf_compl = xgb.XGBRFRegressor(booster='gblinear', learning_rate=1,␣
 ↪n_estimators=550)
```

[143]:
```
model_xgbrf_compl.fit(features_train_pca, target)
```

[143]:
```
XGBRFRegressor(base_score=0.5, booster='gblinear', colsample_bylevel=None,
               colsample_bynode=0.8, colsample_bytree=None, gamma=None,
               gpu_id=-1, importance_type='gain', interaction_constraints=None,
```

```
                       learning_rate=1, max_delta_step=None, max_depth=None,
                       min_child_weight=None, missing=nan, monotone_constraints=None,
                       n_estimators=550, n_jobs=0, num_parallel_tree=None,
                       objective='reg:squarederror', random_state=0, reg_alpha=0,
                       reg_lambda=1e-05, scale_pos_weight=1, subsample=0.8,
                       tree_method=None, validate_parameters=False, verbosity=None)
```

[144]:
```
y_pred_test_vals_xgbrf = model_xgbrf_compl.predict(features_test_pca)
```

[145]:
```
y_pred_test_vals_xgbrf
```

[145]:
```
array([ 99.50013 , 116.646935, 101.892395, …,  92.42944 , 111.85654 ,
        94.21398 ], dtype=float32)
```

[ ]:
```

```