Merc_Project

May 9, 2021

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

[2]:
```
train_data = pd.read_csv("trainmerc.csv")
```

[3]:
```
train_data
```

[3]:
```
         ID       y  X0 X1  X2 X3 X4  X5 X6 X8  …  X375  X376  X377  X378  \
0         0  130.81   k  v  at  a  d   u  j  o  …     0     0     1     0
1         6   88.53   k  t  av  e  d   y  l  o  …     1     0     0     0
2         7   76.26  az  w   n  c  d   x  j  x  …     0     0     0     0
3         9   80.62  az  t   n  f  d   x  l  e  …     0     0     0     0
4        13   78.02  az  v   n  f  d   h  d  n  …     0     0     0     0
…    …      …  .. ..  .. .. ..  .. .. ..  …   …   …   …   …
4204  8405  107.39  ak  s  as  c  d  aa  d  q  …     1     0     0     0
4205  8406  108.77   j  o   t  d  d  aa  h  h  …     0     1     0     0
4206  8412  109.22  ak  v   r  a  d  aa  g  e  …     0     0     1     0
4207  8415   87.48  al  r   e  f  d  aa  l  u  …     0     0     0     0
4208  8417  110.85   z  r  ae  c  d  aa  g  w  …     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        0     0     1     0     0     0
3        0     0     0     0     0     0
4        0     0     0     0     0     0
…    …   …   …   …   …   …
4204     0     0     0     0     0     0
4205     0     0     0     0     0     0
4206     0     0     0     0     0     0
4207     0     0     0     0     0     0
4208     0     0     0     0     0     0

[4209 rows x 378 columns]
```

[4]:
```
train_data.describe()
```

[4]:
```
                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]
```

[5]:
```
train_data.isna().any()
```

[5]:
```
ID      False
y       False
X0      False
```

```
X1      False
X2      False
        …
X380    False
X382    False
X383    False
X384    False
X385    False
Length: 378, dtype: bool
```

[6]:
```
train_data.dtypes
```

[6]:
```
ID        int64
y       float64
X0       object
X1       object
X2       object
         …
X380      int64
X382      int64
X383      int64
X384      int64
X385      int64
Length: 378, dtype: object
```

[7]:
```
isNull=train_data.isna().sum().any()
if isNull:
    print('There are null values in train.')
else:
    print('There is no null values in train.')
```

```
There is no null values in train.
```

[8]:
```
isDuplicate=train_data.duplicated().sum().any()
if isDuplicate:
    print('There are duplicate values in train.')
else:
    print('There is no duplicate values in train.')
```

```
There is no duplicate values in train.
```

[9]:
```
from sklearn.preprocessing import LabelEncoder
```

[10]:
```
le = LabelEncoder()
```

[11]:
```
train_data.columns
```

[11]:
```
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)
```

[12]:
```
 #indentifying object type data type for categorical column
count=0
data_continuous=train_data
for i in train_data.columns:
#print("The string type features are :", i)
    if data_continuous[i].dtype == 'object':
        data_continuous = data_continuous.drop([i], axis = 1)
        count=count+1
        print("The string type features are :"+ i)
        print("Count :",count)#will print the number of categorical columns
        print(i)
```

```
The string type features are :X0
Count : 1
X0
The string type features are :X1
Count : 2
X1
The string type features are :X2
Count : 3
X2
The string type features are :X3
Count : 4
X3
The string type features are :X4
Count : 5
X4
The string type features are :X5
Count : 6
X5
The string type features are :X6
Count : 7
X6
The string type features are :X8
Count : 8
X8
```

[13]:
```
data_continuous.count()
```

[13]:
```
ID      4209
y       4209
```

```
X10     4209
X11     4209
X12     4209
        …
X380    4209
X382    4209
X383    4209
X384    4209
X385    4209
Length: 370, dtype: int64
```

[14]:
```
data_categorical=train_data[['X0','X1','X2','X3','X4','X5','X6','X8']]
data_categorical
```

[14]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	k	v	at	a	d	u	j	o
1	k	t	av	e	d	y	l	o
2	az	w	n	c	d	x	j	x
3	az	t	n	f	d	x	l	e
4	az	v	n	f	d	h	d	n
…	..	..	..	..	..	..	..	..
4204	ak	s	as	c	d	aa	d	q
4205	j	o	t	d	d	aa	h	h
4206	ak	v	r	a	d	aa	g	e
4207	al	r	e	f	d	aa	l	u
4208	z	r	ae	c	d	aa	g	w

[4209 rows x 8 columns]

[15]:
```
le=LabelEncoder()
for col_i in data_categorical:
    data_categorical[col_i]=le.fit_transform(data_categorical[col_i])
    data_categorical
```

```
<ipython-input-15-7d9110948e9b>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_categorical[col_i]=le.fit_transform(data_categorical[col_i])
```

[16]:
```
data_categorical
```

[16]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	32	23	17	0	3	24	9	14
1	32	21	19	4	3	28	11	14

```
2     20  24  34   2   3  27   9  23
3     20  21  34   5   3  27  11   4
4     20  23  34   5   3  12   3  13
…   ..  ..  ..  ..  ..  ..  ..  ..
4204   8  20  16   2   3   0   3  16
4205  31  16  40   3   3   0   7   7
4206   8  23  38   0   3   0   6   4
4207   9  19  25   5   3   0  11  20
4208  46  19   3   2   3   0   6  22

[4209 rows x 8 columns]
```

[17]:
```
ZeroVariance=data_continuous.var()[data_continuous.var()==0].count()
print('Number of columns having zero variance is ',ZeroVariance)
```

```
Number of columns having zero variance is  12
```

[18]:
```
 NonZeroVariance=data_continuous.var()[data_continuous.var()!=0].count()
print('Number of columns having non zero variance is ',NonZeroVariance)
```

```
Number of columns having non zero variance is  358
```

[19]:
```
NonZeroVarDataSet=train_data[train_data.var()[train_data.var()!=0].index.values]
print(NonZeroVarDataSet)
```

```
        ID       y  X10  X12  X13  X14  X15  X16  X17  X18  …  X375  X376  \
0        0  130.81    0    0    1    0    0    0    0    1  …     0     0
1        6   88.53    0    0    0    0    0    0    0    1  …     1     0
2        7   76.26    0    0    0    0    0    0    1    0  …     0     0
3        9   80.62    0    0    0    0    0    0    0    0  …     0     0
4       13   78.02    0    0    0    0    0    0    0    0  …     0     0
…     …      …  …  …  …  …  …  …  …  …  …  …    …    …
4204  8405  107.39    0    0    0    1    0    0    0    0  …     1     0
4205  8406  108.77    0    0    0    0    0    0    0    0  …     0     1
4206  8412  109.22    0    1    1    0    0    0    0    0  …     0     0
4207  8415   87.48    0    0    0    1    0    0    0    0  …     0     0
4208  8417  110.85    0    0    0    0    0    0    0    0  …     1     0

      X377  X378  X379  X380  X382  X383  X384  X385
0        1     0     0     0     0     0     0     0
1        0     0     0     0     0     0     0     0
2        0     0     0     0     1     0     0     0
3        0     0     0     0     0     0     0     0
4        0     0     0     0     0     0     0     0
…    …   …   …   …   …   …   …   …
4204     0     0     0     0     0     0     0     0
4205     0     0     0     0     0     0     0     0
4206     1     0     0     0     0     0     0     0
```

```
4207     0     0     0     0     0     0     0     0
4208     0     0     0     0     0     0     0     0

[4209 rows x 358 columns]
```

[20]:
```
Train_data_New = pd.concat([NonZeroVarDataSet,data_categorical],axis=1)
Train_data_New
```

[20]:
```
         ID       y  X10  X12  X13  X14  X15  X16  X17  X18  …  X384  X385  \
0         0  130.81    0    0    1    0    0    0    0    1  …     0     0
1         6   88.53    0    0    0    0    0    0    0    1  …     0     0
2         7   76.26    0    0    0    0    0    0    1    0  …     0     0
3         9   80.62    0    0    0    0    0    0    0    0  …     0     0
4        13   78.02    0    0    0    0    0    0    0    0  …     0     0
…     …      …   …  …  …  …  …  …  …  …  …  …  …  …  …
4204  8405  107.39    0    0    0    1    0    0    0    0  …     0     0
4205  8406  108.77    0    0    0    0    0    0    0    0  …     0     0
4206  8412  109.22    0    1    1    0    0    0    0    0  …     0     0
4207  8415   87.48    0    0    0    1    0    0    0    0  …     0     0
4208  8417  110.85    0    0    0    0    0    0    0    0  …     0     0

      X0  X1  X2  X3  X4  X5  X6  X8
0     32  23  17   0   3  24   9  14
1     32  21  19   4   3  28  11  14
2     20  24  34   2   3  27   9  23
3     20  21  34   5   3  27  11   4
4     20  23  34   5   3  12   3  13
…   ..  ..  ..  ..  ..  ..  ..  ..
4204   8  20  16   2   3   0   3  16
4205  31  16  40   3   3   0   7   7
4206   8  23  38   0   3   0   6   4
4207   9  19  25   5   3   0  11  20
4208  46  19   3   2   3   0   6  22

[4209 rows x 366 columns]
```

[21]:
```
features = Train_data_New.drop(['y','ID'], axis = 1)
target = Train_data_New[['y']]
print(target)
print(features)
```

```
          y
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
…      …
```

```
4204  107.39
4205  108.77
4206  109.22
4207   87.48
4208  110.85

[4209 rows x 1 columns]
      X10  X12  X13  X14  X15  X16  X17  X18  X19  X20  …  X384  X385  X0  \
0       0    0    1    0    0    0    0    1    0    0  …     0     0  32
1       0    0    0    0    0    0    0    1    0    0  …     0     0  32
2       0    0    0    0    0    0    1    0    0    0  …     0     0  20
3       0    0    0    0    0    0    0    0    0    0  …     0     0  20
4       0    0    0    0    0    0    0    0    0    0  …     0     0  20
…     …  …  …  …  …  …  …  …  …  …  …  …  …  …  ..
4204    0    0    0    1    0    0    0    0    0    0  …     0     0   8
4205    0    0    0    0    0    0    0    0    0    0  …     0     0  31
4206    0    1    1    0    0    0    0    0    0    0  …     0     0   8
4207    0    0    0    1    0    0    0    0    0    0  …     0     0   9
4208    0    0    0    0    0    0    0    0    0    1  …     0     0  46

      X1  X2  X3  X4  X5  X6  X8
0     23  17   0   3  24   9  14
1     21  19   4   3  28  11  14
2     24  34   2   3  27   9  23
3     21  34   5   3  27  11   4
4     23  34   5   3  12   3  13
…   ..  ..  ..  ..  ..  ..  ..
4204  20  16   2   3   0   3  16
4205  16  40   3   3   0   7   7
4206  23  38   0   3   0   6   4
4207  19  25   5   3   0  11  20
4208  19   3   2   3   0   6  22

[4209 rows x 364 columns]
```

[22]:
```
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

[23]:
```
pca = PCA(n_components=8)
```

[24]:
```
pca.fit(features)
```

[24]:
```
PCA(n_components=8)
```

[25]:
```
pca.explained_variance_ratio_
```

[25]:
```
array([0.38334782, 0.21388033, 0.13261866, 0.11826642, 0.09206008,
       0.01590604, 0.0074454 , 0.00433701])
```

[26]:
```
X_train, X_test, y_train, y_test =␣
↪train_test_split(features,target,random_state=10)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(3156, 364)
(3156, 1)
(1053, 364)
(1053, 1)
```

[27]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
X_train_transformed = pca.transform(X_train)
X_test_transformed = pca.transform(X_test)
print(X_train_transformed.shape)
print(X_test_transformed.shape)
```

```
(3156, 8)
(1053, 8)
```

[28]:
```
  #XGBOOST Model
from xgboost import XGBRegressor
my_xgb_reg = XGBRegressor(booster = 'gbtree')
```

[30]:
```
my_xgb_reg.fit(X_train_transformed,y_train)
```

[30]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[31]:
```
my_xgb_pred=my_xgb_reg.predict(X_test_transformed)
print("Prediction Using XGBoost :",my_xgb_pred)
```

```
Prediction Using XGBoost : [ 99.859665 110.37215   92.39008  … 101.80522
100.11856  114.48965 ]
```

[32]:
```
print(mean_squared_error(y_test,my_xgb_pred))
print(np.sqrt(mean_squared_error(y_test,my_xgb_pred)))
```

```
104.9784507981446
10.245899218621302
```

[51]:
```
##########################################################################Treating␣
↪Test Data ############
```

[33]:
```
test_data = pd.read_csv("testmerc.csv")
```

[34]:
```
test_data.describe()
```

[34]:

	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]
```

[65]:
```
 isNull=test_data.isnull().sum().any()
if isNull:
    print('There are null values in test.')
else:
    print('There is no null values in test.')
```

```
There is no null values in test.
```

[36]:
```
 isDuplicate=test_data.duplicated().sum().any()
if isDuplicate:
    print('There are duplicate values in test.')
else:
    print('There is no duplicate values in test.')
```

```
There is no duplicate values in test.
```

[37]:
```
test_data.columns
```

[37]:
```
Index(['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=377)
```

[38]:
```
#indentifying object type data type for categorical column
count=0
data_cont=test_data
for i in test_data.columns:
    #print("The string type features are :", i)
        if data_cont[i].dtype == 'object':
            data_cont = data_cont.drop([i], axis = 1)
            count=count+1
            print("The string type features are :"+ i)
            print("Count :",count)#will print the number of categorical columns
            print(i)
```

```
The string type features are :X0
Count : 1
X0
The string type features are :X1
Count : 2
X1
The string type features are :X2
```

```
Count : 3
X2
The string type features are :X3
Count : 4
X3
The string type features are :X4
Count : 5
X4
The string type features are :X5
Count : 6
X5
The string type features are :X6
Count : 7
X6
The string type features are :X8
Count : 8
X8
```

[39]:
```
data_cate=test_data[['X0','X1','X2','X3','X4','X5','X6','X8']]
data_cate
```

[39]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	az	v	n	f	d	t	a	w
1	t	b	ai	a	d	b	g	y
2	az	v	as	f	d	a	j	j
3	az	l	n	f	d	z	l	n
4	w	s	as	c	d	y	i	m
…	..	..	..	..	..	..	..	..
4204	aj	h	as	f	d	aa	j	e
4205	t	aa	ai	d	d	aa	j	y
4206	y	v	as	f	d	aa	d	w
4207	ak	v	as	a	d	aa	c	q
4208	t	aa	ai	c	d	aa	g	r

[4209 rows x 8 columns]

[40]:
```
le=LabelEncoder()
for col_i in data_cate:
    data_cate[col_i]=le.fit_transform(data_cate[col_i])
    data_cate
```

```
<ipython-input-40-7f664727b0ac>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_cate[col_i]=le.fit_transform(data_cate[col_i])
```

[41]:
```
ZeroVariance1=data_cont.var()[data_cont.var()==0].count()
print('Number of columns having zero variance is ',ZeroVariance1)
```

```
Number of columns having zero variance is  5
```

[42]:
```
NonZeroVariance1=data_cont.var()[data_cont.var()!=0].count()
print('Number of columns having non zero variance is ',NonZeroVariance1)
```

```
Number of columns having non zero variance is  364
```

[43]:
```
NonZeroVarDataSet_test=test_data[test_data.var()[test_data.var()!=0].index.
 ↪values]
print(NonZeroVarDataSet_test)
```

```
        ID  X10  X11  X12  X13  X14  X15  X16  X17  X18  …  X375  X376  \
0        1    0    0    0    0    0    0    0    0    0  …     0     0
1        2    0    0    0    0    0    0    0    0    0  …     0     0
2        3    0    0    0    0    1    0    0    0    0  …     0     0
3        4    0    0    0    0    0    0    0    0    0  …     0     0
4        5    0    0    0    0    1    0    0    0    0  …     1     0
…      …   …   …   …   …   …   …   …   …   …   …   …   …   …   …   …
4204  8410    0    0    0    0    1    0    0    0    0  …     0     0
4205  8411    0    0    0    0    0    0    0    0    0  …     0     1
4206  8413    0    0    0    0    1    0    0    0    0  …     0     0
4207  8414    0    0    0    1    1    0    0    0    0  …     0     0
4208  8416    0    0    0    0    0    0    0    0    0  …     1     0

      X377  X378  X379  X380  X382  X383  X384  X385
0        0     1     0     0     0     0     0     0
1        1     0     0     0     0     0     0     0
2        0     1     0     0     0     0     0     0
3        0     1     0     0     0     0     0     0
4        0     0     0     0     0     0     0     0
…      …   …   …   …   …   …   …   …   …
4204     0     0     0     0     0     0     0     0
4205     0     0     0     0     0     0     0     0
4206     0     0     0     0     0     0     0     0
4207     1     0     0     0     0     0     0     0
4208     0     0     0     0     0     0     0     0

[4209 rows x 364 columns]
```

[44]:
```
Test_data_New = pd.concat([NonZeroVarDataSet_test,data_cate],axis=1)
Test_data_New
```

[44]:
```
        ID  X10  X11  X12  X13  X14  X15  X16  X17  X18  …  X384  X385  X0  \
0        1    0    0    0    0    0    0    0    0    0  …     0     0  21
```

```
1         2    0    0    0    0    0    0    0    0    0  …      0      0  42
2         3    0    0    0    0    1    0    0    0    0  …      0      0  21
3         4    0    0    0    0    0    0    0    0    0  …      0      0  21
4         5    0    0    0    0    1    0    0    0    0  …      0      0  45
…     …   …   …   …   …   …   …   …   …   …   …   …   …   ..
4204  8410    0    0    0    0    1    0    0    0    0  …      0      0   6
4205  8411    0    0    0    0    0    0    0    0    0  …      0      0  42
4206  8413    0    0    0    0    1    0    0    0    0  …      0      0  47
4207  8414    0    0    0    1    1    0    0    0    0  …      0      0   7
4208  8416    0    0    0    0    0    0    0    0    0  …      0      0  42

      X1  X2  X3  X4  X5  X6  X8
0     23  34   5   3  26   0  22
1      3   8   0   3   9   6  24
2     23  17   5   3   0   9   9
3     13  34   5   3  31  11  13
4     20  17   2   3  30   8  12
…   ..  ..  ..  ..  ..  ..  ..
4204   9  17   5   3   1   9   4
4205   1   8   3   3   1   9  24
4206  23  17   5   3   1   3  22
4207  23  17   0   3   1   2  16
4208   1   8   2   3   1   6  17

[4209 rows x 372 columns]
```

[45]:
```
test_fetaure= Test_data_New.drop(['ID'], axis = 1)
test_fetaure
```

[45]:
```
      X10  X11  X12  X13  X14  X15  X16  X17  X18  X19  …  X384  X385  X0  \
0       0    0    0    0    0    0    0    0    0    0  …     0     0  21
1       0    0    0    0    0    0    0    0    0    1  …     0     0  42
2       0    0    0    0    1    0    0    0    0    0  …     0     0  21
3       0    0    0    0    0    0    0    0    0    0  …     0     0  21
4       0    0    0    0    1    0    0    0    0    0  …     0     0  45
…   …  …  …  …  …  …  …  …  …  …  …  …  …  …  …  ..
4204    0    0    0    0    1    0    0    0    0    0  …     0     0   6
4205    0    0    0    0    0    0    0    0    0    1  …     0     0  42
4206    0    0    0    0    1    0    0    0    0    0  …     0     0  47
4207    0    0    0    1    1    0    0    0    0    0  …     0     0   7
4208    0    0    0    0    0    0    0    0    0    1  …     0     0  42

      X1  X2  X3  X4  X5  X6  X8
0     23  34   5   3  26   0  22
1      3   8   0   3   9   6  24
2     23  17   5   3   0   9   9
3     13  34   5   3  31  11  13
```

```
4     20  17   2   3  30   8  12
…   ..  ..  ..  ..  ..  ..  ..
4204   9  17   5   3   1   9   4
4205   1   8   3   3   1   9  24
4206  23  17   5   3   1   3  22
4207  23  17   0   3   1   2  16
4208   1   8   2   3   1   6  17

[4209 rows x 371 columns]
```

[46]:
```
from sklearn.decomposition import PCA
pca2 = PCA(n_components=8)
```

[47]:
```
pca2.fit(test_fetaure)
```

[47]:
```
PCA(n_components=8)
```

[48]:
```
pca2.explained_variance_ratio_
```

[48]:
```
array([0.43515102, 0.17670897, 0.13646292, 0.10977912, 0.08622208,
       0.01433962, 0.00722966, 0.00406843])
```

[49]:
```
X_test_transformed = pca2.transform(test_fetaure)
```

[50]:
```
my_xgb_pred_test=my_xgb_reg.predict(X_test_transformed)
print("Prediction Using XGBoost :",my_xgb_pred_test)
```

```
Prediction Using XGBoost : [ 75.49396  89.62093  86.85658 …  97.15548
116.27424  89.2783 ]
```