Project Mercedes-Benz Greener Manufacturing

August 19, 2022

Project Mercedes-Benz Greener Manufacturing

[1]:
```python
#Importing the necessary library
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```python
#Getting the data and storying it in a DataFrame
merc_data_df=pd.read_csv('train.csv')
merc_data_df
```

[2]:
```
        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]
```

[3]:
```python
merc_data_df.shape
```

[3]:
```
(4209, 378)
```

[4]:
```python
merc_data_df['y'].value_counts()
```

[4]:
```
90.76     7
89.06     7
89.38     7
91.88     7
93.62     6
         ..
93.26     1
93.24     1
105.94    1
94.17     1
79.00     1
Name: y, Length: 2545, dtype: int64
```

[5]:
```python
#checking for the Variance of each and every column
merc_var=pd.DataFrame(merc_data_df.var())
merc_var.columns=['Variance']
print(merc_var)
print(merc_var.loc[merc_var['Variance']==0]) #This gives me the columns with␣
 ↪Variance value equal to 0
#method2_to_get_list_of_columns_with_variance_zero
#var_zero_list=merc_var[merc_var==0].dropna() #Gives Nan value to all the␣
 ↪columns with a variance other the zero but in our case since we need only␣
 ↪the ones with 0 variance, so we have removed all the Nan columns using dropna
#print(var_zero_list.loc)
```

```
          Variance
ID    5.941936e+06
y     1.607667e+02
X10   1.313092e-02
X11   0.000000e+00
X12   6.945713e-02
…              …
X380  8.014579e-03
X382  7.546747e-03
X383  1.660732e-03
X384  4.750593e-04
X385  1.423823e-03

[370 rows x 1 columns]
     Variance
X11       0.0
```

```
X93         0.0
X107        0.0
X233        0.0
X235        0.0
X268        0.0
X289        0.0
X290        0.0
X293        0.0
X297        0.0
X330        0.0
X347        0.0
```

[6]:
```
merc_data_df['X11']
```

[6]:
```
0       0
1       0
2       0
3       0
4       0
       ..
4204    0
4205    0
4206    0
4207    0
4208    0
Name: X11, Length: 4209, dtype: int64
```

#Here i have checked a Variable with Zero Variance and as you can see it is a single deterministic value. This might be a issue when we are running certain algorithms

[7]:
```
#Dropping the Columns with Zero Variance
merc_data_df.
 ↪drop(['X11','X93','X107','X233','X235','X268','X289','X290','X293','X297','X330','X347'],ax
merc_data_df.head()
```

[7]:
```
   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 366 columns]
```

Removing the columns with Zero Variance cause they have a effect on PCA algorithm. Since this dataset is having large no of features it is better to perform PCA and bring down no of varibles. Since the column with Zero Variance is a blocker for PCA we are removing those columns from our Dataset

[8]:
```
merc_data_df.describe()
```

[8]:
```
                ID            y          X10          X12          X13  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean   4205.960798   100.669318     0.013305     0.075077     0.057971
std    2437.608688    12.679381     0.114590     0.263547     0.233716
min       0.000000    72.110000     0.000000     0.000000     0.000000
25%    2095.000000    90.820000     0.000000     0.000000     0.000000
50%    4220.000000    99.150000     0.000000     0.000000     0.000000
75%    6314.000000   109.010000     0.000000     0.000000     0.000000
max    8417.000000   265.320000     1.000000     1.000000     1.000000

               X14          X15          X16          X17          X18  …  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000  …
mean      0.428130     0.000475     0.002613     0.007603     0.007840  …
std       0.494867     0.021796     0.051061     0.086872     0.088208  …
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      0.318841     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 358 columns]
```

From this also we can infer if a column has zero variance or not just by looking into the row of Standard Deviation

[9]:
```python
#Importing the Test set Data
merc_test_df=pd.read_csv('test.csv')
merc_test_df.head()
```

[9]:
```
   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]
```

[10]:
```python
#checking for the Variance of each and every column
merc_test_var=pd.DataFrame(merc_test_df.var())
merc_test_var.columns=['Variance']
print(merc_test_var)
print(merc_test_var.loc[merc_test_var['Variance']==0]) #This gives me the␣
↪columns with Variance value equal to 0
```

```
          Variance
ID    5.871311e+06
X10   1.865006e-02
X11   2.375861e-04
X12   6.885074e-02
X13   5.734498e-02
…              …
X380  8.014579e-03
X382  8.715481e-03
X383  4.750593e-04
X384  7.124196e-04
X385  1.660732e-03
```

```
[369 rows x 1 columns]
      Variance
X257       0.0
X258       0.0
X295       0.0
X296       0.0
X369       0.0
```

[11]:
```
#Dropping the Columns with Zero Variance
merc_test_df.drop(['X257','X258','X295','X296','X369'],axis=1,inplace=True)
merc_test_df.head()
```

[11]:
```
   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 372 columns]
```

Since we are going to do PCA to Test Data As well i might throw an error if we have columns with zero variance,So i am removing it from the Test Dataset

[12]:
```
#Checking for null values in both the Train Dataset
merc_train_is_null=pd.DataFrame(merc_data_df.isnull().sum())
merc_train_is_null.columns=['Total_Null_values']
print(merc_train_is_null)
print(merc_train_is_null.loc[merc_train_is_null['Total_Null_values']!=0])
```

```
      Total_Null_values
ID                    0
y                     0
X0                    0
X1                    0
X2                    0
…                   …
X380                  0
X382                  0
X383                  0
X384                  0
```

```
X385                   0

[366 rows x 1 columns]
Empty DataFrame
Columns: [Total_Null_values]
Index: []
```

From the last print Statement we can infer that no column has null value, here i basically checked if any column had a null value. Usually the first line alon is sufficient to infer but since here it is not displaying all the the columns, so i had to check it via code.

[13]:
```
#Checking for null values in both the Test Dataset
merc_test_is_null=pd.DataFrame(merc_test_df.isnull().sum())
merc_test_is_null.columns=['Total_Null_values']
print(merc_test_is_null)
print(merc_test_is_null.loc[merc_test_is_null['Total_Null_values']!=0])
```

```
      Total_Null_values
ID                     0
X0                     0
X1                     0
X2                     0
X3                     0
…                    …
X380                   0
X382                   0
X383                   0
X384                   0
X385                   0

[372 rows x 1 columns]
Empty DataFrame
Columns: [Total_Null_values]
Index: []
```

From the last print Statement we can infer that no column has null value, here i basically checked if any column had a null value.

[14]:
```
#Checking uniques Values of Train Dataset
print(merc_data_df['X0'].value_counts())
print(merc_data_df.nunique())
```

```
z      360
ak     349
y      324
ay     313
t      306
x      300
o      269
```

```
f      227
n      195
w      182
j      181
az     175
aj     151
s      106
ap     103
h       75
d       73
al      67
v       36
af      35
ai      34
m       34
e       32
ba      27
at      25
a       21
ax      19
i       18
aq      18
am      18
u       17
aw      16
l       16
ad      14
b       11
au      11
k       11
r       10
as      10
bc       6
ao       4
c        3
aa       2
q        2
ab       1
ac       1
g        1
Name: X0, dtype: int64
ID       4209
y        2545
X0         47
X1         27
X2         44
         …
X380        2
```

```
X382       2
X383       2
X384       2
X385       2
Length: 366, dtype: int64
```

From this i can infer how many unique values are available in each column of Dataset

[15]:
```python
#Checking uniques Values of Test Dataset
print(merc_test_df['X0'].value_counts())
print(merc_test_df.nunique())
```

```
ak    432
y     348
z     335
x     302
ay    299
t     293
o     246
f     213
w     198
j     171
n     167
aj    162
az    161
s     116
ap    108
al     88
h      64
d      61
e      48
v      40
ai     38
m      34
af     34
am     28
i      25
at     21
u      20
ba     19
a      18
b      13
k      12
ad     12
aq     11
aw     11
r      10
ax      8
```

```
bc       6
l        6
as       6
c        6
au       5
ao       5
g        3
an       1
av       1
ae       1
bb       1
ag       1
p        1
Name: X0, dtype: int64
ID       4209
X0         49
X1         27
X2         45
X3          7
         …
X380        2
X382        2
X383        2
X384        2
X385        2
Length: 372, dtype: int64
```

#Setting Up Data for Model Building

[16]:
```
#splitting the train dataset into input and output for the model
x_train=merc_data_df.drop('y',axis=1)
print(x_train.head())
y_train=merc_data_df['y']
print(y_train.head())
```

```
   ID  X0 X1  X2 X3 X4 X5 X6 X8  X10  …  X375  X376  X377  X378  X379  X380  \
0   0   k  v  at  a  d  u  j  o    0  …     0     0     1     0     0     0
1   6   k  t  av  e  d  y  l  o    0  …     1     0     0     0     0     0
2   7  az  w   n  c  d  x  j  x    0  …     0     0     0     0     0     0
3   9  az  t   n  f  d  x  l  e    0  …     0     0     0     0     0     0
4  13  az  v   n  f  d  h  d  n    0  …     0     0     0     0     0     0

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

```
[5 rows x 365 columns]
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

[17]:
```
#Assigning Test Data to a variable
x_test=merc_test_df
print(x_test.head())
```

```
   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 372 columns]
```

[18]:
```
x_test.head()
```

[18]:
```
   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 372 columns]
```

Since both the Test and Train Data have certain columns with Object Datatype, it is better to convert *them into numerical datatype for model training*

[19]:
```
#Label_Encoding on Train_Data
from sklearn.preprocessing import LabelEncoder
lab_enc=LabelEncoder()
train_columns=x_train.columns   #To get the column Names into a list so that i␣
↪can use it to loop and fit on every column
print(train_columns)
for i in train_columns:
    lab_enc.fit(x_train[i])     #Train on the data
    x_train[i]=lab_enc.transform(x_train[i]) #transforming the data
print(x_train.head())
```

```
Index(['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=365)
   ID  X0  X1  X2  X3  X4  X5  X6  X8  X10  …  X375  X376  X377  X378  X379  \
0   0  32  23  17   0   3  24   9  14    0  …     0     0     1     0     0
1   1  32  21  19   4   3  28  11  14    0  …     1     0     0     0     0
2   2  20  24  34   2   3  27   9  23    0  …     0     0     0     0     0
3   3  20  21  34   5   3  27  11   4    0  …     0     0     0     0     0
4   4  20  23  34   5   3  12   3  13    0  …     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 365 columns]
```

[20]:
```
#Label_Encoding on Test_data
from sklearn.preprocessing import LabelEncoder
lab_enc_test=LabelEncoder()
test_columns=x_test.columns    #To get the column Names into a list so that i␣
↪can use it to loop and fit on every colum
print(test_columns)
for j in test_columns:
    lab_enc_test.fit(x_test[j])  #Train on the data
    x_test[j]=lab_enc_test.transform(x_test[j]) #transforming the data
print(x_test.head())
```

```
Index(['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       …
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
```

```
       'X385'],
      dtype='object', length=372)
   ID  X0  X1  X2  X3  X4  X5  X6  X8  X10  …  X375  X376  X377  X378  X379  \
0   0  21  23  34   5   3  26   0  22    0  …     0     0     0     1     0
1   1  42   3   8   0   3   9   6  24    0  …     0     0     1     0     0
2   2  21  23  17   5   3   0   9   9    0  …     0     0     0     1     0
3   3  21  13  34   5   3  31  11  13    0  …     0     0     0     1     0
4   4  45  20  17   2   3  30   8  12    0  …     1     0     0     0     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     0     0     0     0     0
4     0     0     0     0     0

[5 rows x 372 columns]
```

#The major takeaways from the Label Encoding is that actually we must fit on the train set and then transform both train and test set. But here due improper dataset if i try to train on train dataset and transform both datasets, it is throwing an error because if you observe carefully, we can infer that Train Data has less no of features when compared to Test Set, so, if we try to transform the test set on the basis of fitted train set. Then it will throw errors cause some features are missing in test set.SO, we can also infer that it is better to do dimensionality reduction and bring down no of features similar in both Test and Train Datasets.

#Dimensionality Reduction

[21]:
```
#Dimensionality Reduction using Principal Component Analysis for Train Set
from sklearn.decomposition import PCA
pca_merc_train=PCA(n_components=10)  #here n_components is hyperparameter
pca_merc_train.fit(x_train)   #training on the x_train Data
print(pca_merc_train.explained_variance_ratio_)  #Inorder to check the Total␣
 ↪Variation our algo is accounting to
x_train_trans=pca_merc_train.transform(x_train)  #Transforming our training␣
 ↪data, that is bringing down the no of features
```

```
[9.99659608e-01 1.38083753e-04 7.69770342e-05 4.40107808e-05
 3.31849481e-05 2.66009200e-05 5.73061934e-06 2.67845282e-06
 1.56203752e-06 1.05802559e-06]
```

[22]:
```
#Dimensionality Reduction using Principal Component Analysis for Test Set
from sklearn.decomposition import PCA
pca_merc_test=PCA(n_components=10)  #here n_components is hyperparameter
pca_merc_test.fit(x_test)   #training on the x_test Data
print(pca_merc_test.explained_variance_ratio_)  #Inorder to check the Total␣
 ↪Variation our algo is accounting to
```

```
x_test_trans=pca_merc_test.transform(x_test)  #Transforming our testing data,␣
 ↪that is bringing down the no of features
print(x_test_trans)
```

```
[9.99637654e-01 1.67216554e-04 6.79028030e-05 4.33597184e-05
 3.31749814e-05 2.93170202e-05 5.51112705e-06 2.77341533e-06
 1.56251928e-06 1.04907502e-06]
[[ 2.10391944e+03  1.48901578e+01  1.43106871e+01 … -1.71520611e+00
   2.68015002e+00 -1.39321798e+00]
 [ 2.10300998e+03 -1.48068922e+01 -8.10199579e+00 …  4.15276621e+00
   1.92372374e+00  4.46863094e-01]
 [ 2.10203711e+03  1.23521650e+01 -2.18928463e+00 … -8.20388024e-01
   7.81621434e-01 -1.89336891e-01]
 …
 [-2.10191831e+03 -1.37536955e+01  3.21752326e+00 … -2.88530584e+00
  -1.06187275e+00 -2.33170640e+00]
 [-2.10292321e+03  2.46245283e+01 -5.02830129e+00 …  3.48120149e+00
  -2.02639839e+00  3.81031514e-02]
 [-2.10390525e+03 -1.56684906e+01 -7.94246831e+00 …  7.06982115e-02
   1.06468235e+00  2.08030214e+00]]
```

#Main takeaways from Principal COmponent Analysis is that, actually fitting should only be performed on train data and using this we need to transform our train and test data, but here since the train and test data are from different datasets and also they have unequal number of features, it will throw an size mismatch error when we try to transform of test data.So, that is the reason why i fitted on the test data as well.Following this, since n_components is a hyperparameter it is difficult to guess the exact number to use. Here we transform our data which had around 350-380 features to a dataset which has only 10-30 features.

[23]:
```
#Building XGBoost Model
import xgboost
xgb_regressor=xgboost.XGBRegressor()   #Since the variable to be predicted is␣
 ↪Continous we are going to using regression algo here
```

#Here since we dont know the optimal parameter values to be used, it is better to use Gridsearch and find the right parameters

[24]:
```
#checking the GridSearchCV Algorithm to find the best parameter values
from sklearn.model_selection import GridSearchCV
params={'n_estimators':[100, 200, 400, 800], 'max_depth':[1,2,3,6,10]}
grid_search_cv=GridSearchCV(xgb_regressor,params,cv=3,n_jobs=-1)
grid_search_cv.fit(x_train_trans,y_train)
print(grid_search_cv.best_params_)
```

```
{'max_depth': 3, 'n_estimators': 100}
```

[25]:
```
#Using Gridsearch Algo to find the best parameter value of learning rate and␣
 ↪min child weight
```

```
params_2={'learning_rate' :[0.1, 0.2, 0.3, 0.5], 'min_child_weight' : [1, 2, 3,␣
 ↪4, 5]}
grid_search_cv_2=GridSearchCV(xgb_regressor,params_2,cv=3,n_jobs=-1)
grid_search_cv_2.fit(x_train_trans,y_train)
print(grid_search_cv_2.best_params_)
```

```
{'learning_rate': 0.1, 'min_child_weight': 4}
```

[26]:
```
#Using Gridsearch Algo to find the best parameter value of subsample
params_3={'subsample' : [0.5, 0.6, 0.7, 0.8, 1.0]}
grid_search_cv_3=GridSearchCV(xgb_regressor,params_3,cv=3,n_jobs=-1)
grid_search_cv_3.fit(x_train_trans,y_train)
print(grid_search_cv_3.best_params_)
```

```
{'subsample': 1.0}
```

#From the above part of the code we can infer that we are tuning to find the best parameters to run the xgbregressor algorthim. Here first i tried tuning all the 5 parameters together but it requires a lot of time since here we were trying to replicate almost 100 regressor DT with different parameters which leads to a lot of combination of trees. So to reduce this i have splitted them into different sets and tunned to find the best parameters. 1)started with tuning n_estimators which gives the number of boosting rounds required or number of trees to built. 2)max_depth: maximum tree depth required for baselearners, it should be in a optimal window, if it is too high the tree becomes more complex and tends to overfit 3)Learning rate:This also determines how our model coverges, if too high it gets difficult to converge, if low it might take a lot of boosting rounds to converge. 4)min_child_weight: gives the number of child nodes required at the present node. 5)subsample: subsampling ratio of the training instances. It will occur once in every boosting iteration. Subsample ratio = 0.5 means that the algorithm would randomly sample half of the training data prior to growing trees.

[27]:
```
#Building the tree with optimal parameters for improved prediction
xgb_regressor=xgboost.
 ↪XGBRFRegressor(n_estimators=100,max_depth=3,learning_rate=0.
 ↪1,min_child_weight=5,subsample=1.0)
xgb_regressor.fit(x_train_trans,y_train)   #Training the model
```

[27]:
```
XGBRFRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
               colsample_bytree=1, gamma=0, gpu_id=-1, importance_type='gain',
               interaction_constraints=None, learning_rate=0.1,
               max_delta_step=0, max_depth=3, min_child_weight=5, missing=nan,
               monotone_constraints=None, n_estimators=100, n_jobs=0,
               num_parallel_tree=100, objective='reg:squarederror',
               random_state=0, reg_alpha=0, scale_pos_weight=1, subsample=1.0,
               tree_method=None, validate_parameters=False, verbosity=None)
```

[28]:
```
#Preditcing the Test_df Values
y_test=xgb_regressor.predict(x_test_trans)
print(y_test)
```

```
[ 9.334505  9.703091 10.05097  … 10.969909 11.347246 10.152216]
```

#Finally, to conclude this we have first checked and removed any columns with zero variance just to overcome the situation where PCA algorithm might throw an error. Then we did dimensionality reduction using PCA to bring down the number of features. Then we built a model to predict the test_df values using the test data. Here since the test data doesnot have the output variable it is not possible to get the mean squared error for this dataset