Following actions should be performed:

- If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
- Check for null and unique values for test and train sets.
- Apply label encoder.
- Perform dimensionality reduction.
- Predict your test_df values using XGBoost.

In [45]:

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

In [46]:

```
df_train=pd.read_csv("train.csv")
df_test=pd.read_csv("test.csv")
```

In [47]:

```
df_train.head()
```

Out[47]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X3
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	

5 rows × 378 columns

In [48]:

```
df_train.shape
```

Out[48]:

```
(4209, 378)
```

In [49]:

```
df_train.describe()
```

Out[49]:

	ID	y	X10	X11	X12	X13	X14
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000
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

8 rows × 370 columns

In [50]:

```
df_train.columns
```

Out[50]:

```
Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X38
4',
       'X385'],
      dtype='object', length=378)
```

In [51]:

```
df_train.info
```

Out[51]:

```
<bound method DataFrame.info of         ID       y  X0 X1  X2 X3 X4  X5 X6 X
8  ...  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]>
```

If for any column(s), the variance is equal to zero, then you need to remove those variable(s).

- Seperating x_train and y_train dataset

In [52]:

```
xtrain=df_train.drop('y',axis=1)
ytrain=df_train['y']
```

In [53]:

```
xtrain.shape
```

Out[53]:

```
(4209, 377)
```

In [54]:

```
ytrain.shape
```

Out[54]:

```
(4209,)
```

In [55]:

```
xtrain.var()
```

Out[55]:

```
ID       5.941936e+06
X10      1.313092e-02
X11      0.000000e+00
X12      6.945713e-02
X13      5.462335e-02
             ...
X380     8.014579e-03
X382     7.546747e-03
X383     1.660732e-03
X384     4.750593e-04
X385     1.423823e-03
Length: 369, dtype: float64
```

In [56]:

```
df_test.head()
```

Out[56]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0

5 rows × 377 columns

In [57]:

```
df_test.shape
```

Out[57]:

```
(4209, 377)
```

In [58]:

```
df_test.describe()
```

Out[58]:

	ID	X10	X11	X12	X13	X14	
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000
mean	4211.039202	0.019007	0.000238	0.074364	0.061060	0.427893	0.000
std	2423.078926	0.136565	0.015414	0.262394	0.239468	0.494832	0.026
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	2115.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
50%	4202.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
75%	6310.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000
max	8416.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000

8 rows × 369 columns

In [59]:

```
df_test.info
```

Out[59]:

```
<bound method DataFrame.info of         ID  X0  X1  X2 X3 X4  X5 X6 X8  X10
...  X375  X376  X377  X378  \
0        1  az   v   n  f  d   t  a  w    0  ...     0     0     0     1
1        2   t   b  ai  a  d   b  g  y    0  ...     0     0     1     0
2        3  az   v  as  f  d   a  j  j    0  ...     0     0     0     1
3        4  az   l   n  f  d   z  l  n    0  ...     0     0     0     1
4        5   w   s  as  c  d   y  i  m    0  ...     1     0     0     0
...    ...  ..  ..  .. .. ..  .. .. ..  ...  ...   ...   ...   ...   ...
4204  8410  aj   h  as  f  d  aa  j  e    0  ...     0     0     0     0
4205  8411   t  aa  ai  d  d  aa  j  y    0  ...     0     1     0     0
4206  8413   y   v  as  f  d  aa  d  w    0  ...     0     0     0     0
4207  8414  ak   v  as  a  d  aa  c  q    0  ...     0     0     1     0
4208  8416   t  aa  ai  c  d  aa  g  r    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        0     0     0     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 377 columns]>
```

In [60]:

```
df_test.columns
```

Out[60]:

```
Index(['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X38
4',
       'X385'],
      dtype='object', length=377)
```

In [61]:

```
for i in xtrain:
    if (xtrain[i].dtype=='O'):
            continue
    elif(xtrain.var()[i]==0):
        xtrain.drop(i,axis=1,inplace=True)
        df_test.drop(i,axis=1,inplace=True)
```

In [63]:

```
xtrain.head()
```

Out[63]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382
0	0	k	v	at	a	d	u	j	o	0	...	0	0	1	0	0	0	0
1	6	k	t	av	e	d	y	l	o	0	...	1	0	0	0	0	0	0
2	7	az	w	n	c	d	x	j	x	0	...	0	0	0	0	0	0	1
3	9	az	t	n	f	d	x	l	e	0	...	0	0	0	0	0	0	0
4	13	az	v	n	f	d	h	d	n	0	...	0	0	0	0	0	0	0

5 rows × 365 columns

In [65]:

```
xtrain.shape
```

Out[65]:

```
(4209, 365)
```

Check for null and unique values for test and train sets.

In [68]:

```
xtrain.isnull().sum()
```

Out[68]:

```
ID      0
X0      0
X1      0
X2      0
X3      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 365, dtype: int64
```

In [74]:

```
for i in xtrain:
    if(xtrain[i].isnull().sum!=0):
        print(i)
```

```
ID
X0
X1
X2
X3
X4
X5
X6
X8
X10
X12
X13
X14
X15
X16
X17
X18
X19
X20
```

In [73]:

```
for i in df_test:
    if(df_test[i].isnull().sum()!=0):
        print(i)
```

In [75]:

```
ytrain.isnull().sum()
```

Out[75]:

```
0
```

In [76]:

```
for i in xtrain:
    if (xtrain[i].dtype=='O'):
        print(i)
        print(xtrain[i].unique())
```

```
X0
['k' 'az' 't' 'al' 'o' 'w' 'j' 'h' 's' 'n' 'ay' 'f' 'x' 'y' 'aj' 'ak' 'am'
 'z' 'q' 'at' 'ap' 'v' 'af' 'a' 'e' 'ai' 'd' 'aq' 'c' 'aa' 'ba' 'as' 'i'
 'r' 'b' 'ax' 'bc' 'u' 'ad' 'au' 'm' 'l' 'aw' 'ao' 'ac' 'g' 'ab']
X1
['v' 't' 'w' 'b' 'r' 'l' 's' 'aa' 'c' 'a' 'e' 'h' 'z' 'j' 'o' 'u' 'p' 'n'
 'i' 'y' 'd' 'f' 'm' 'k' 'g' 'q' 'ab']
X2
['at' 'av' 'n' 'e' 'as' 'aq' 'r' 'ai' 'ak' 'm' 'a' 'k' 'ae' 's' 'f' 'd'
 'ag' 'ay' 'ac' 'ap' 'g' 'i' 'aw' 'y' 'b' 'ao' 'al' 'h' 'x' 'au' 't' 'an'
 'z' 'ah' 'p' 'am' 'j' 'q' 'af' 'l' 'aa' 'c' 'o' 'ar']
X3
['a' 'e' 'c' 'f' 'd' 'b' 'g']
X4
['d' 'b' 'c' 'a']
X5
['u' 'y' 'x' 'h' 'g' 'f' 'j' 'i' 'd' 'c' 'af' 'ag' 'ab' 'ac' 'ad' 'ae'
 'ah' 'l' 'k' 'n' 'm' 'p' 'q' 's' 'r' 'v' 'w' 'o' 'aa']
X6
['j' 'l' 'd' 'h' 'i' 'a' 'g' 'c' 'k' 'e' 'f' 'b']
X8
['o' 'x' 'e' 'n' 's' 'a' 'h' 'p' 'm' 'k' 'd' 'i' 'v' 'j' 'b' 'q' 'w' 'g'
 'y' 'l' 'f' 'u' 'r' 't' 'c']
```

In [77]:

```
for i in df_test:
    if (df_test[i].dtype=='O'):
        print(i)
        print(df_test[i].unique())
```

```
X0
['az' 't' 'w' 'y' 'x' 'f' 'ap' 'o' 'ay' 'al' 'h' 'z' 'aj' 'd' 'v' 'ak'
 'ba' 'n' 'j' 's' 'af' 'ax' 'at' 'aq' 'av' 'm' 'k' 'a' 'e' 'ai' 'i' 'ag'
 'b' 'am' 'aw' 'as' 'r' 'ao' 'u' 'l' 'c' 'ad' 'au' 'bc' 'g' 'an' 'ae' 'p'
 'bb']
X1
['v' 'b' 'l' 's' 'aa' 'r' 'a' 'i' 'p' 'c' 'o' 'm' 'z' 'e' 'h' 'w' 'g' 'k'
 'y' 't' 'u' 'd' 'j' 'q' 'n' 'f' 'ab']
X2
['n' 'ai' 'as' 'ae' 's' 'b' 'e' 'ak' 'm' 'a' 'aq' 'ag' 'r' 'k' 'aj' 'ay'
 'ao' 'an' 'ac' 'af' 'ax' 'h' 'i' 'f' 'ap' 'p' 'au' 't' 'z' 'y' 'aw' 'd'
 'at' 'g' 'am' 'j' 'x' 'ab' 'w' 'q' 'ah' 'ad' 'al' 'av' 'u']
X3
['f' 'a' 'c' 'e' 'd' 'g' 'b']
X4
['d' 'b' 'a' 'c']
X5
['t' 'b' 'a' 'z' 'y' 'x' 'h' 'g' 'f' 'j' 'i' 'd' 'c' 'af' 'ag' 'ab' 'ac'
 'ad' 'ae' 'ah' 'l' 'k' 'n' 'm' 'p' 'q' 's' 'r' 'v' 'w' 'o' 'aa']
X6
['a' 'g' 'j' 'l' 'i' 'd' 'f' 'h' 'c' 'k' 'e' 'b']
X8
['w' 'y' 'j' 'n' 'm' 's' 'a' 'v' 'r' 'o' 't' 'h' 'c' 'k' 'p' 'u' 'd' 'g'
 'b' 'q' 'e' 'l' 'f' 'i' 'x']
```

Apply Label Encoder.

In [78]:

```
from sklearn import preprocessing
```

In [80]:

```
label_encoder = preprocessing.LabelEncoder()
```

In [81]:

```
for i in xtrain:
    if (xtrain[i].dtype!='int64'):
        xtrain[i]= label_encoder.fit_transform(xtrain[i])
        print(i)
        print(xtrain[i].unique())
```

```
X0
[32 20 40  9 36 43 31 29 39 35 19 27 44 45  7  8 10 46 37 15 12 42  5  0
 26  6 25 13 24  1 22 14 30 38 21 18 23 41  4 16 34 33 17 11  3 28  2]
X1
[23 21 24  3 19 13 20  1  4  0  6  9 26 11 16 22 17 15 10 25  5  7 14 12
  8 18  2]
X2
[17 19 34 25 16 14 38  7  8 33  0 31  3 39 26 24  5 21  2 13 27 29 20 42
 22 12  9 28 41 18 40 11 43  6 36 10 30 37  4 32  1 23 35 15]
X3
[0 4 2 5 3 1 6]
X4
[3 1 2 0]
X5
[24 28 27 12 11 10 14 13  9  8  5  6  1  2  3  4  7 16 15 18 17 20 21 23
 22 25 26 19  0]
X6
[ 9 11  3  7  8  0  6  2 10  4  5  1]
X8
[14 23  4 13 18  0  7 15 12 10  3  8 21  9  1 16 22  6 24 11  5 20 17 19
  2]
```

In [82]:

```
for i in df_test:
    if (df_test[i].dtype!='int64'):
        df_test[i]= label_encoder.fit_transform(df_test[i])
        print(i)
        print(df_test[i].unique())
```

```
X0
[21 42 45 47 46 29 12 38 20  8 31 48  6 27 44  7 23 37 33 41  3 19 15 13
 17 36 34  0 28  5 32  4 22  9 18 14 40 11 43 35 26  1 16 25 30 10  2 39
 24]
X1
[23  3 13 20  1 19  0 10 17  4 16 14 26  6  9 24  8 12 25 21 22  5 11 18
 15  7  2]
X2
[34  8 17  4 38 24 26 10 33  0 16  6 37 32  9 23 14 13  2  5 22 29 30 27
 15 35 19 39 44 43 21 25 18 28 12 31 42  1 41 36  7  3 11 20 40]
X3
[5 0 2 4 3 6 1]
X4
[3 1 0 2]
X5
[26  9  0 31 30 29 14 13 12 16 15 11 10  6  7  2  3  4  5  8 18 17 20 19
 22 23 25 24 27 28 21  1]
X6
[ 0  6  9 11  8  3  5  7  2 10  4  1]
X8
[22 24  9 13 12 18  0 21 17 14 19  7  2 10 15 20  3  6  1 16  4 11  5  8
 23]
```

Perform dimensionality reduction.

In [83]:

```
std_scalar=preprocessing.StandardScaler()
```

In [84]:

```
xtrain = std_scalar.fit_transform(xtrain)
```

In [85]:

```
df_test = std_scalar.fit_transform(df_test)
```

In [86]:

```
from sklearn.decomposition import PCA
```

In [87]:

```
pca=PCA()
```

In [88]:

```
xtrain_pca=pca.fit_transform(xtrain)
```

In [89]:

```
df_test_pca=pca.fit_transform(df_test)
```

In [90]:

```
pca.explained_variance_ratio_
```

Out[90]:

```
array([7.12486653e-02, 5.62415084e-02, 4.79979004e-02, 3.48818222e-02,
       3.26672655e-02, 3.17006619e-02, 2.82351002e-02, 2.14035131e-02,
       1.91298040e-02, 1.74214677e-02, 1.66286781e-02, 1.64393781e-02,
       1.49384640e-02, 1.38777649e-02, 1.35286203e-02, 1.27571090e-02,
       1.22387389e-02, 1.17690111e-02, 1.10581757e-02, 1.06911106e-02,
       1.02639043e-02, 9.37624352e-03, 9.29049134e-03, 9.05267345e-03,
       8.59958787e-03, 8.54160869e-03, 7.82044339e-03, 7.48747967e-03,
       7.34403625e-03, 7.23314434e-03, 7.03553099e-03, 6.87577048e-03,
       6.75483979e-03, 6.50479088e-03, 6.37950579e-03, 6.19292941e-03,
       6.13882865e-03, 6.03783025e-03, 5.99492525e-03, 5.89027248e-03,
       5.55576539e-03, 5.48572168e-03, 5.28121707e-03, 5.25910912e-03,
       5.12864286e-03, 5.02484447e-03, 4.94931135e-03, 4.76961699e-03,
       4.71278343e-03, 4.63198213e-03, 4.55830740e-03, 4.47537732e-03,
       4.35658769e-03, 4.28679476e-03, 4.22548636e-03, 4.04582798e-03,
       4.00033185e-03, 3.93454157e-03, 3.90301437e-03, 3.81967831e-03,
       3.80986083e-03, 3.72237302e-03, 3.65473153e-03, 3.64320550e-03,
       3.61870358e-03, 3.55727206e-03, 3.46128792e-03, 3.41047056e-03,
       3.37771387e-03, 3.36256219e-03, 3.31946796e-03, 3.30330864e-03,
       3.27652584e-03, 3.22861002e-03, 3.19479418e-03, 3.16074032e-03,
       3.10798244e-03, 3.10645053e-03, 3.07178635e-03, 3.05166619e-03,
       2.99650329e-03, 2.97911342e-03, 2.95233531e-03, 2.92968079e-03,
       2.88970794e-03, 2.88161868e-03, 2.85350724e-03, 2.83014050e-03,
       2.79960822e-03, 2.79022195e-03, 2.74931322e-03, 2.73849941e-03,
       2.69588542e-03, 2.68941004e-03, 2.64962483e-03, 2.62783267e-03,
       2.60106610e-03, 2.59075141e-03, 2.56129618e-03, 2.52025181e-03,
       2.48444007e-03, 2.46570355e-03, 2.45669962e-03, 2.43955390e-03,
       2.42671765e-03, 2.39406272e-03, 2.37753636e-03, 2.35029790e-03,
       2.32810902e-03, 2.26482158e-03, 2.24156399e-03, 2.23458252e-03,
       2.21686329e-03, 2.19543397e-03, 2.16864220e-03, 2.13939602e-03,
       2.10540805e-03, 2.09147504e-03, 2.08213708e-03, 2.05483315e-03,
       2.03818073e-03, 2.00300911e-03, 1.94897936e-03, 1.93704004e-03,
       1.93230270e-03, 1.92060136e-03, 1.88244702e-03, 1.83813991e-03,
       1.80528747e-03, 1.78224784e-03, 1.74610028e-03, 1.73211634e-03,
       1.71216167e-03, 1.67986342e-03, 1.64771357e-03, 1.61453168e-03,
       1.59628090e-03, 1.54925339e-03, 1.49466381e-03, 1.47998598e-03,
       1.47003925e-03, 1.43038582e-03, 1.41599576e-03, 1.38297063e-03,
       1.37771035e-03, 1.36436365e-03, 1.33269348e-03, 1.30417942e-03,
       1.27480706e-03, 1.26589598e-03, 1.24773234e-03, 1.23094905e-03,
       1.19234860e-03, 1.17073219e-03, 1.15206168e-03, 1.12350066e-03,
       1.10748818e-03, 1.10150579e-03, 1.06583699e-03, 1.04346695e-03,
       1.02463714e-03, 9.95758731e-04, 9.71998658e-04, 9.57689300e-04,
       9.30738127e-04, 9.16635380e-04, 8.99017230e-04, 8.76356461e-04,
       8.56126178e-04, 8.30582393e-04, 8.21694292e-04, 8.02493814e-04,
       7.76511877e-04, 7.74551191e-04, 7.43885041e-04, 7.28663565e-04,
       7.24732922e-04, 6.79961784e-04, 6.73459240e-04, 6.61756500e-04,
       6.39414138e-04, 6.25266526e-04, 6.03204345e-04, 5.79144335e-04,
       5.71735111e-04, 5.45853876e-04, 5.35603334e-04, 5.24151043e-04,
       5.14994605e-04, 4.97477026e-04, 4.89448595e-04, 4.76733679e-04,
       4.65189068e-04, 4.50214685e-04, 4.37565516e-04, 4.31458477e-04,
       4.23318421e-04, 4.08333193e-04, 3.97759383e-04, 3.80890485e-04,
       3.73848521e-04, 3.57616625e-04, 3.49990343e-04, 3.42519877e-04,
       3.24772837e-04, 3.18177219e-04, 3.13961066e-04, 3.09515606e-04,
       3.01351052e-04, 2.98079849e-04, 2.85629703e-04, 2.76391851e-04,
       2.64331204e-04, 2.47022641e-04, 2.40246893e-04, 2.25020442e-04,
       2.10275717e-04, 2.05191206e-04, 1.91552886e-04, 1.86367134e-04,
       1.84770368e-04, 1.77355620e-04, 1.74325950e-04, 1.55890284e-04,
       1.49256838e-04, 1.39360019e-04, 1.34030870e-04, 1.29998583e-04,
       1.23571153e-04, 1.15848469e-04, 1.10343527e-04, 1.04443878e-04,
       9.97539777e-05, 9.82964906e-05, 8.96761687e-05, 8.83401122e-05,
       8.03503684e-05, 7.90402106e-05, 6.56112731e-05, 6.45180163e-05,
       6.10411770e-05, 5.17238011e-05, 4.94839204e-05, 4.49320985e-05,
       4.46999173e-05, 4.26923912e-05, 3.78614608e-05, 3.72160577e-05,
       2.90012446e-05, 2.84127650e-05, 2.66169878e-05, 2.09023191e-05,
       1.87169099e-05, 1.71968838e-05, 1.54956321e-05, 1.48022947e-05,
       1.33989721e-05, 1.12515686e-05, 9.88750490e-06, 8.65025002e-06,
       8.21766320e-06, 5.41349347e-06, 5.27376104e-06, 2.83947057e-06,
       2.80859842e-06, 2.69316844e-06, 1.26563907e-06, 1.04541178e-06,
       5.39685714e-07, 1.40513216e-32, 1.74825768e-33, 1.53623391e-33,
       1.25875050e-33, 1.15623236e-33, 1.01020744e-33, 9.48390741e-34,
       6.95898634e-34, 6.90933879e-34, 6.29630793e-34, 5.07826731e-34,
       4.38349541e-34, 3.29509409e-34, 2.79272818e-34, 2.62963307e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 2.53937676e-34,
       2.53937676e-34, 2.53937676e-34, 2.53937676e-34, 1.58662115e-34,
       9.89689520e-35, 6.53655308e-35, 4.74423424e-35, 2.76815570e-35,
       8.76515394e-36])
```

In [92]:

```
pca=PCA(n_components=200)

xtrain_pca=pca.fit_transform(xtrain)

df_test_pca=pca.fit_transform(df_test)

pca.explained_variance_ratio_
```

Out[92]:

```
array([0.07124867, 0.05624151, 0.0479979 , 0.03488182, 0.03266727,
       0.03170066, 0.0282351 , 0.02140351, 0.0191298 , 0.01742147,
       0.01662868, 0.01643938, 0.01493846, 0.01387776, 0.01352862,
       0.01275711, 0.01223874, 0.01176901, 0.01105818, 0.01069111,
       0.0102639 , 0.00937624, 0.00929049, 0.00905267, 0.00859959,
       0.00854161, 0.00782044, 0.00748748, 0.00734404, 0.00723314,
       0.00703553, 0.00687577, 0.00675484, 0.00650479, 0.00637951,
       0.00619293, 0.00613883, 0.00603783, 0.00599493, 0.00589027,
       0.00555577, 0.00548572, 0.00528122, 0.00525911, 0.00512864,
       0.00502484, 0.00494931, 0.00476962, 0.00471278, 0.00463198,
       0.00455831, 0.00447538, 0.00435659, 0.00428679, 0.00422549,
       0.00404583, 0.00400033, 0.00393454, 0.00390301, 0.00381968,
       0.00380986, 0.00372237, 0.00365473, 0.00364321, 0.0036187 ,
       0.00355727, 0.00346129, 0.00341047, 0.00337771, 0.00336256,
       0.00331947, 0.00330331, 0.00327653, 0.00322861, 0.00319479,
       0.00316074, 0.00310798, 0.00310645, 0.00307179, 0.00305167,
       0.0029965 , 0.00297911, 0.00295234, 0.00292968, 0.00288971,
       0.00288162, 0.00285351, 0.00283014, 0.00279961, 0.00279022,
       0.00274931, 0.0027385 , 0.00269589, 0.00268941, 0.00264962,
       0.00262783, 0.00260107, 0.00259075, 0.0025613 , 0.00252025,
       0.00248444, 0.0024657 , 0.0024567 , 0.00243955, 0.00242672,
       0.00239406, 0.00237754, 0.0023503 , 0.00232811, 0.00226482,
       0.00224156, 0.00223458, 0.00221686, 0.00219543, 0.00216864,
       0.0021394 , 0.00210541, 0.00209147, 0.00208214, 0.00205483,
       0.00203818, 0.00200301, 0.00194898, 0.00193704, 0.0019323 ,
       0.0019206 , 0.00188245, 0.00183814, 0.00180529, 0.00178225,
       0.0017461 , 0.00173212, 0.00171216, 0.00167986, 0.00164771,
       0.00161453, 0.00159628, 0.00154925, 0.00149466, 0.00147999,
       0.00147004, 0.00143038, 0.001416  , 0.00138297, 0.00137771,
       0.00136436, 0.00133269, 0.00130418, 0.00127481, 0.00126589,
       0.00124772, 0.00123095, 0.00119234, 0.00117073, 0.00115206,
       0.0011235 , 0.00110748, 0.0011015 , 0.00106583, 0.00104346,
       0.00102463, 0.00099575, 0.00097199, 0.00095768, 0.00093071,
       0.00091662, 0.00089901, 0.00087633, 0.00085608, 0.00083057,
       0.00082165, 0.00080243, 0.00077647, 0.00077451, 0.00074386,
       0.0007283 , 0.00072465, 0.00067978, 0.00067339, 0.00066167,
       0.00063934, 0.00062479, 0.00060306, 0.00057847, 0.0005714 ,
       0.00054553, 0.0005354 , 0.00052402, 0.00051364, 0.00049689,
       0.00048843, 0.00047489, 0.00046366, 0.00044856, 0.00043099,
       0.00042752, 0.00041881, 0.00040684, 0.00039639, 0.00037901])
```

Predict your test_df values using XGBoost.

In [93]:

```
import xgboost as xgb
```

In [94]:

```
xgb_model=xgb.XGBRegressor(objective ='reg:linear', n_estimators = 10)
```

In [95]:

```
xgb_model.fit(xtrain_pca,ytrain)
```

```
[21:40:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in f
avor of reg:squarederror.
```

Out[95]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=Fals
e,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=10, n_jobs=4,
             num_parallel_tree=1, objective='reg:linear', predictor='auto',
             random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             subsample=1, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

In [96]:

```
xgb_model.predict(df_test_pca)
```

Out[96]:

```
array([ 78.506714,  90.84197 ,  80.794586, ...,  98.53473 , 103.381874,
        90.35204 ], dtype=float32)
```

In [98]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [100]:

```
ytrain_pred_xgb2=xgb_model.predict((xtrain_pca))
np.sqrt(mean_squared_error(ytrain ,ytrain_pred_xgb2))
```

Out[100]:

```
7.7529378889435625
```

In [102]:

```
r2_score(ytrain ,ytrain_pred_xgb2)
```

Out[102]:

```
0.6260274887750187
```

In [103]:

```
pd.DataFrame({"Actual Train Values" : ytrain , "Model Predictions" :  ytrain_pred_xgb2})
```

Out[103]:

	Actual Train Values	Model Predictions
0	130.81	111.508102
1	88.53	89.199455
2	76.26	77.150223
3	80.62	81.635017
4	78.02	78.241142
...	...	...
4204	107.39	102.831642
4205	108.77	104.847939
4206	109.22	107.898170
4207	87.48	89.820564
4208	110.85	93.163147

4209 rows × 2 columns

In [104]:

```
#####################################################
```

In [ ]:

```

```