In [1]:
```
# Importing all the required packages

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
label=LabelEncoder

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Reading the train and test files

train_data= pd.read_csv(r'/content/train.csv')
test_data= pd.read_csv(r'/content/test.csv')
```

In [3]:
```
# Checking the first five rows of train data

train_data.head()
```

Out[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0

5 rows × 378 columns

In [4]:
```
# Checking the first five rows of test data

test_data.head()
```

Out[4]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X
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 [5]:
```
train_data.shape
```

Out[5]:
```
(4209, 378)
```

In [6]:
```
test_data.shape
```

Out[6]:
```
(4209, 377)
```

In [7]:
```
train_data.describe()
```

Out[7]:

	ID	y	X10	X11	X12	X13	X14	
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.00
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.00
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.02
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.00
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.00
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.00
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.00
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.00

8 rows × 370 columns

In [8]:
```
test_data.describe()
```

Out[8]:

	ID	X10	X11	X12	X13	X14	X15	
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4
mean	4211.039202	0.019007	0.000238	0.074364	0.061060	0.427893	0.000713	
std	2423.078926	0.136565	0.015414	0.262394	0.239468	0.494832	0.026691	
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2115.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	4202.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	6310.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	
max	8416.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 369 columns

In [9]:
```
# Understand the data types of train and test datasets we have

cols = [c for c in train_data.columns if 'X' in c]
```

```
print('Number of features: {}'.format(len(cols)))

print('Feature types:')
train_data[cols].dtypes.value_counts()
```

Out[9]:
```
Number of features: 376
Feature types:
int64     368
object      8
dtype: int64
```

In [10]:
```
cols = [c for c in test_data.columns if 'X' in c]
print('Number of features: {}'.format(len(cols)))

print('Feature types:')
test_data[cols].dtypes.value_counts()
```

Out[10]:
```
Number of features: 376
Feature types:
int64     368
object      8
dtype: int64
```

In [11]:
```
# Counting the number & type of datatypes availabe in the dataframe

counts = [[], [], []]
for c in cols:
    typ = train_data[c].dtype
    uniq = len(np.unique(train_data[c]))
    if uniq == 1:
        counts[0].append(c)
    elif uniq == 2 and typ == np.int64:
        counts[1].append(c)
    else:
        counts[2].append(c)

print('Constant features: {} Binary features: {} Categorical features: {}\n'
      .format(*[len(c) for c in counts]))
print('Constant features:', counts[0])
print('Categorical features:', counts[2])
```

```
Constant features: 12 Binary features: 356 Categorical features: 8

Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X2
93', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

In [12]:
```
# Checking for variables having zero variance

variance_with_zero = train_data.var()[train_data.var()==0].index.values
variance_with_zero
```

Out[12]:
```
array(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290',
       'X293', 'X297', 'X330', 'X347'], dtype=object)
```

In [13]:
```
# Viewing the variables having zero variance

train_data[['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290',
            'X293', 'X297', 'X330', 'X347']].head(10)
```

Out[13]:

	X11	X93	X107	X233	X235	X268	X289	X290	X293	X297	X330	X347
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0

In [14]:
```
# Dropping zero variance variables

train_data = train_data.drop(variance_with_zero, axis=1)
```

In [15]:
```
train_data.shape
```

Out[15]:
```
(4209, 366)
```

In [16]:
```
# Removing columns ID and Y from the datasets as they are not used for learning

usable_columns = list(set(train_data.columns) - set(['ID', 'y']))
y_train = train_data['y'].values
id_test = test_data['ID'].values

x_train = train_data[usable_columns]
x_test = test_data[usable_columns]
```

In [17]:
```
x_train.shape
```

Out[17]:
```
(4209, 364)
```

In [18]:
```
x_test.shape
```

Out[18]:
```
(4209, 364)
```

In [19]:
```
# Checking for null and unique values for test and train sets

def check_missing_values(df):
    if df.isnull().any().any():
        print("There are missing values in the dataframe")
    else:
        print("There are no missing values in the dataframe")
check_missing_values(x_train)
check_missing_values(x_test)
```

```
There are no missing values in the dataframe
There are no missing values in the dataframe
```

In [20]:
```
x_train.agg(['nunique','count','size', 'dtypes'])
```

Out[20]:

	X172	X220	X270	X59	X170	X204	X361	X159	X282	X165	...	X83	X186	X49	X
nunique	2	2	2	2	2	2	2	2	2	2	...	2	2	2	
count	4209	4209	4209	4209	4209	4209	4209	4209	4209	4209	...	4209	4209	4209	
size	4209	4209	4209	4209	4209	4209	4209	4209	4209	4209	...	4209	4209	4209	
dtypes	int64	int64	int64	int64	int64	int64	int64	int64	int64	int64	...	int64	int64	int64	i

4 rows × 364 columns

In [21]:
```
x_test.agg(['nunique','count','size', 'dtypes'])
```

Out[21]:

	X172	X220	X270	X59	X170	X204	X361	X159	X282	X165	...	X83	X186	X49	X
nunique	2	2	2	2	2	2	2	2	2	2	...	2	2	2	
count	4209	4209	4209	4209	4209	4209	4209	4209	4209	4209	...	4209	4209	4209	
size	4209	4209	4209	4209	4209	4209	4209	4209	4209	4209	...	4209	4209	4209	
dtypes	int64	int64	int64	int64	int64	int64	int64	int64	int64	int64	...	int64	int64	int64	i

4 rows × 364 columns

In [22]:
```
x_train.dtypes.unique() # here we have int64 ad object datatype values
```

Out[22]:
```
array([dtype('int64'), dtype('O')], dtype=object)
```

In [23]:
```
x_test.dtypes.unique()
```

Out[23]:
```
array([dtype('int64'), dtype('O')], dtype=object)
```

In [24]:
```
# Applying label encoder to train data

for column in usable_columns:
    cardinality = len(np.unique(x_train[column]))
    if cardinality > 2: # Column is categorical
        mapper = lambda x: sum([ord(digit) for digit in x])
        x_train[column] = x_train[column].apply(mapper)
        x_test[column] = x_test[column].apply(mapper)
```

In [25]:
```
x_train.head()
```

Out[25]:

	X172	X220	X270	X59	X170	X204	X361	X159	X282	X165	...	X83	X186	X49	X251	X15
0	0	1	0	0	1	1	1	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	0	1	0	0	1	...	0	0	0	0	
2	0	1	0	0	0	0	1	0	0	0	...	0	0	0	0	
3	0	1	0	0	0	0	1	0	0	0	...	0	0	0	0	
4	0	1	0	0	0	0	1	0	0	0	...	0	0	0	0	

5 rows × 364 columns

In [26]:
```
x_test.head()
```

Out[26]:

	X172	X220	X270	X59	X170	X204	X361	X159	X282	X165	...	X83	X186	X49	X251	X15
0	0	1	0	0	0	1	1	0	0	0	...	0	0	0	0	
1	0	0	0	0	0	1	1	0	0	0	...	0	1	1	0	
2	0	0	0	0	0	1	1	1	0	0	...	0	0	0	1	
3	0	1	0	0	0	0	1	0	0	0	...	0	0	0	0	
4	0	1	0	0	0	0	1	0	0	0	...	0	0	0	1	

5 rows × 364 columns

In [27]:
```
x_test.dtypes.unique() # now we have only int64 datatype values
```

Out[27]:
```
array([dtype('int64')], dtype=object)
```

In [28]:
```
x_train.dtypes.unique()
```

Out[28]:
```
array([dtype('int64')], dtype=object)
```

In [29]:
```
# Perform dimensionality reduction (PCA) on train and test datasets

from sklearn.decomposition import PCA
```

In [30]:
```
n_comp = 12
pca = PCA(n_components=n_comp, random_state=420)
pca2_results_train = pca.fit_transform(x_train)
pca2_results_test = pca.transform(x_test)
print(pca2_results_train.shape)
print(pca2_results_test.shape)
```

```
(4209, 12)
(4209, 12)
```

In [31]:
```
# Training the model using xgboost
```

```
import xgboost as xgb
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
```

In [32]:
```
x_train, x_valid, y_train, y_valid = train_test_split(pca2_results_train, y_train,
                                                      test_size=0.2, random_state=42)
```

In [33]:
```
# Creating the Xgboost specific DMatrix data format from the numpy array

d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)
d_test = xgb.DMatrix(pca2_results_test)
```

In [34]:
```
# Setting the parameters for Xgboost to work

params = {}
params['objective'] = 'reg:linear'
params['eta'] = 0.02
params['max_depth'] = 4
```

In [35]:
```
def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)
```

In [36]:
```
watchlist = [(d_train, 'train'), (d_valid, 'valid')]
```

In [37]:
```
clf = xgb.train(params, d_train,
                1000, watchlist, early_stopping_rounds=50,
                feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[14:17:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
deprecated in favor of reg:squarederror.
[0]     train-rmse:98.997       valid-rmse:98.8888      train-r2:-59.4973       valid
-r2:-61.8269
Multiple eval metrics have been passed: 'valid-r2' will be used for early stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[10]    train-rmse:81.1441      valid-rmse:81.0785      train-r2:-39.6449       valid
-r2:-41.234
[20]    train-rmse:66.5975      valid-rmse:66.5561      train-r2:-26.3784       valid
-r2:-27.4595
[30]    train-rmse:54.7579      valid-rmse:54.7334      train-r2:-17.5091       valid
-r2:-18.2467
[40]    train-rmse:45.1401      valid-rmse:45.1379      train-r2:-11.5782       valid
-r2:-12.0898
[50]    train-rmse:37.3467      valid-rmse:37.355       train-r2:-7.60987       valid
-r2:-7.96497
[60]    train-rmse:31.0496      valid-rmse:31.0843      train-r2:-4.9512        valid
-r2:-5.20775
[70]    train-rmse:25.984       valid-rmse:26.031       train-r2:-3.16777       valid
-r2:-3.35345
[80]    train-rmse:21.9387      valid-rmse:21.9981      train-r2:-1.97108       valid
-r2:-2.10901
[90]    train-rmse:18.7353      valid-rmse:18.8164      train-r2:-1.16678       valid
-r2:-1.27469
[100]   train-rmse:16.2245      valid-rmse:16.3355      train-r2:-0.62493       valid
-r2:-0.714411
[110]   train-rmse:14.2849      valid-rmse:14.4272      train-r2:-0.259639      valid
-r2:-0.33726
[120]   train-rmse:12.8138      valid-rmse:12.9847      train-r2:-0.013559      valid
-r2:-0.083212
[130]   train-rmse:11.7009      valid-rmse:11.9025      train-r2:0.154862       valid
-r2:0.089819
[140]   train-rmse:10.8647      valid-rmse:11.1157      train-r2:0.271332       valid
-r2:0.206168
[150]   train-rmse:10.2583      valid-rmse:10.5449      train-r2:0.350406       valid
-r2:0.285605
[160]   train-rmse:9.80741      valid-rmse:10.1432      train-r2:0.406252       valid
-r2:0.338995
[170]   train-rmse:9.47502      valid-rmse:9.87157      train-r2:0.445818       valid
-r2:0.373929
[180]   train-rmse:9.23021      valid-rmse:9.6806       train-r2:0.474085       valid
-r2:0.397918
[190]   train-rmse:9.05007      valid-rmse:9.5422       train-r2:0.494413       valid
-r2:0.415011
[200]   train-rmse:8.9152       valid-rmse:9.44924      train-r2:0.50937        valid
-r2:0.426354
[210]   train-rmse:8.81472      valid-rmse:9.38247      train-r2:0.520366       valid
-r2:0.434431
[220]   train-rmse:8.72949      valid-rmse:9.33751      train-r2:0.529597       valid
-r2:0.439839
[230]   train-rmse:8.67189      valid-rmse:9.30556      train-r2:0.535784       valid
-r2:0.443665
[240]   train-rmse:8.62195      valid-rmse:9.28001      train-r2:0.541115       valid
-r2:0.446717
[250]   train-rmse:8.57733      valid-rmse:9.26159      train-r2:0.545853       valid
-r2:0.448911
[260]   train-rmse:8.53354      valid-rmse:9.25157      train-r2:0.550478       valid
-r2:0.450102
[270]   train-rmse:8.50239      valid-rmse:9.24001      train-r2:0.553754       valid
```

```
-r2:0.451476
[280]   train-rmse:8.46002      valid-rmse:9.23107      train-r2:0.55819        valid
-r2:0.452536
[290]   train-rmse:8.42982      valid-rmse:9.22206      train-r2:0.561339       valid
-r2:0.453605
[300]   train-rmse:8.39257      valid-rmse:9.2194       train-r2:0.565207       valid
-r2:0.45392
[310]   train-rmse:8.36474      valid-rmse:9.21595      train-r2:0.568086       valid
-r2:0.454328
[320]   train-rmse:8.33314      valid-rmse:9.21501      train-r2:0.571344       valid
-r2:0.45444
[330]   train-rmse:8.30499      valid-rmse:9.2118       train-r2:0.574233       valid
-r2:0.45482
[340]   train-rmse:8.27547      valid-rmse:9.20951      train-r2:0.577255       valid
-r2:0.455091
[350]   train-rmse:8.24726      valid-rmse:9.20602      train-r2:0.580133       valid
-r2:0.455504
[360]   train-rmse:8.21246      valid-rmse:9.20264      train-r2:0.583669       valid
-r2:0.455904
[370]   train-rmse:8.18836      valid-rmse:9.19909      train-r2:0.586108       valid
-r2:0.456324
[380]   train-rmse:8.16162      valid-rmse:9.20041      train-r2:0.588807       valid
-r2:0.456167
[390]   train-rmse:8.13265      valid-rmse:9.20335      train-r2:0.591722       valid
-r2:0.45582
[400]   train-rmse:8.10459      valid-rmse:9.20685      train-r2:0.594533       valid
-r2:0.455405
[410]   train-rmse:8.0821       valid-rmse:9.2075       train-r2:0.596781       valid
-r2:0.455329
[420]   train-rmse:8.06055      valid-rmse:9.2109       train-r2:0.598929       valid
-r2:0.454926
Stopping. Best iteration:
[376]   train-rmse:8.1717       valid-rmse:9.19791      train-r2:0.587791       valid
-r2:0.456462
```

In [38]:

```
# Predicting test df values using xgboost

p_test = clf.predict(d_test)

sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = p_test
sub.to_csv('xgb.csv', index=False)

sub.head()
```

Out[38]:

	ID	y
0	1	79.287155
1	2	96.074661
2	3	81.250526
3	4	77.294121
4	5	109.705254