```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

## Import train dataset

data_train=pd.read_csv('train.csv')

data_train.head()
```

```
   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]
```

```
data_train.shape
```

```
(4209, 378)
```

```
##chech missing value

data_train.isna().sum()
```

```
ID      0
y       0
X0      0
X1      0
X2      0
       ..
X380    0
X382    0
X383    0
X384    0
```

```
X385    0
Length: 378, dtype: int64
```

```
data_train.nunique()
```

```
ID      4209
y       2545
X0        47
X1        27
X2        44
        ...
X380       2
X382       2
X383       2
X384       2
X385       2
Length: 378, dtype: int64
```

```
data_train.describe()
```

```
                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]
```

```
## try find variance of column

print(data_train.var)
```

```
<bound method NDFrame._add_numeric_operations.<locals>.var of
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]>
```

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

```
Number of features: 376
```

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

```
Feature types:

int64     368
object      8
dtype: int64
```

```
##Count the data in each of the columns

counts = [[], [], []]
for c in cols:
```

```
    typ = data_train[c].dtype
    uniq = len(np.unique(data_train[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', 'X293', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

```
# the prediction output
y_train = data_train['y'].values
```

```
## Import Test dataset
```

```
data_test=pd.read_csv('test.csv')
data_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 377 columns]
```

```
##remove columns ID and Y from the data

remove_columns = list(set(data_train.columns) - set(['ID', 'y']))
y_train = data_train['y'].values
id_test = data_test['ID'].values

x_train = data_train[remove_columns]
x_test = data_test[remove_columns]

#Check for null and unique values for tain and test dataset

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 null in train dataest

check_missing_values(x_train)
```

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

```
## check null in test dataset

check_missing_values(x_test)
```

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

```
## If for any column(s), the variance is equal to zero
## Apply label encoder

for column in remove_columns:
    val = len(np.unique(x_train[column]))
    if val == 1:
        x_train.drop(column, axis=1) # Column with only one
        # value is useless so we drop it
        x_test.drop(column, axis=1)
    if val > 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)
x_train.head()
```

```
<ipython-input-28-34faa544c303>:9: 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
  x_train[column] = x_train[column].apply(mapper)
<ipython-input-28-34faa544c303>:10: 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
  x_test[column] = x_test[column].apply(mapper)
```

	X379	X378	X115	X131	X163	X34	X48	X136	X140	X28	...	X95	X262	\
0	0	0	0	1	0	0	0	1	0	0	...	0	1	
1	0	0	0	0	0	0	0	1	0	0	...	0	0	
2	0	0	0	0	1	0	0	0	0	1	...	0	0	
3	0	0	0	0	0	0	0	0	0	1	...	0	0	
4	0	0	0	0	0	0	0	0	0	1	...	0	0	

	X317	X283	X166	X65	X278	X134	X355	X344
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0
4	0	0	1	0	0	0	0	0

```
[5 rows x 376 columns]
```

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

```
Feature types:

int64    376
dtype: int64
```

```
## Perform dimensionality reduction (PCA)

from sklearn.decomposition import PCA

pca = PCA(n_components=12, random_state=420)
pca2_results_train = pca.fit_transform(x_train)
pca2_results_test = pca.transform(x_test)

## Training using xgboost

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

```
x_train, x_valid, y_train, y_valid = train_test_split(
        pca2_results_train,
        y_train, test_size=0.2,
        random_state=4242)

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)

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

def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)

watchlist = [(d_train, 'train'), (d_valid, 'valid')]

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

```
[17:48:11] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.4.0/src/objective/regression_obj.cu:171: reg:linear is
now deprecated in favor of reg:squarederror.
[0]   train-rmse:99.14835   train-r2:-58.35295    valid-rmse:98.26297
      valid-r2:-67.63754
[10]  train-rmse:81.27651   train-r2:-38.88428    valid-rmse:80.36433
      valid-r2:-44.91014
[20]  train-rmse:66.71610   train-r2:-25.87403    valid-rmse:65.77334
      valid-r2:-29.75260
[30]  train-rmse:54.86915   train-r2:-17.17724    valid-rmse:53.89119
      valid-r2:-19.64513
[40]  train-rmse:45.24564   train-r2:-11.36018    valid-rmse:44.22232
      valid-r2:-12.90160
[50]  train-rmse:37.44741   train-r2:-7.46672     valid-rmse:36.37773
      valid-r2:-8.40705
[60]  train-rmse:31.15105   train-r2:-4.85891     valid-rmse:30.01782
      valid-r2:-5.40531
[70]  train-rmse:26.08691   train-r2:-3.10882     valid-rmse:24.90733
      valid-r2:-3.40998
[80]  train-rmse:22.04894   train-r2:-1.93526     valid-rmse:20.82346
      valid-r2:-2.08239
[90]  train-rmse:18.85430   train-r2:-1.14631     valid-rmse:17.59753
      valid-r2:-1.20133
[100] train-rmse:16.34233   train-r2:-0.61250     valid-rmse:15.07638
      valid-r2:-0.61576
[110] train-rmse:14.41168   train-r2:-0.25401     valid-rmse:13.14548
      valid-r2:-0.22839
```

```
[120] train-rmse:12.94262   train-r2:-0.01138     valid-rmse:11.68336
      valid-r2:0.02967
[130] train-rmse:11.82560   train-r2:0.15566 valid-rmse:10.61321
      valid-r2:0.19929
[140] train-rmse:10.99171   train-r2:0.27054 valid-rmse:9.84869
      valid-r2:0.31049
[150] train-rmse:10.38934   train-r2:0.34830 valid-rmse:9.31929
      valid-r2:0.38263
[160] train-rmse:9.94055    train-r2:0.40339 valid-rmse:8.95843
      valid-r2:0.42951
[170] train-rmse:9.60575    train-r2:0.44290 valid-rmse:8.71915
      valid-r2:0.45958
[180] train-rmse:9.36104    train-r2:0.47092 valid-rmse:8.55947
      valid-r2:0.47919
[190] train-rmse:9.17712    train-r2:0.49151 valid-rmse:8.45814
      valid-r2:0.49145
[200] train-rmse:9.02794    train-r2:0.50791 valid-rmse:8.39234
      valid-r2:0.49933
[210] train-rmse:8.92379    train-r2:0.51919 valid-rmse:8.35805
      valid-r2:0.50342
[220] train-rmse:8.83687    train-r2:0.52851 valid-rmse:8.33338
      valid-r2:0.50635
[230] train-rmse:8.77426    train-r2:0.53517 valid-rmse:8.32326
      valid-r2:0.50754
[240] train-rmse:8.72745    train-r2:0.54012 valid-rmse:8.31429
      valid-r2:0.50860
[250] train-rmse:8.68730    train-r2:0.54434 valid-rmse:8.31208
      valid-r2:0.50886
[260] train-rmse:8.64810    train-r2:0.54844 valid-rmse:8.31297
      valid-r2:0.50876
[270] train-rmse:8.60765    train-r2:0.55266 valid-rmse:8.31136
      valid-r2:0.50895
[280] train-rmse:8.57949    train-r2:0.55558 valid-rmse:8.31203
      valid-r2:0.50887
[290] train-rmse:8.55134    train-r2:0.55849 valid-rmse:8.30992
      valid-r2:0.50912
[300] train-rmse:8.52779    train-r2:0.56092 valid-rmse:8.30644
      valid-r2:0.50953
[310] train-rmse:8.50280    train-r2:0.56349 valid-rmse:8.30659
      valid-r2:0.50951
[320] train-rmse:8.47743    train-r2:0.56609 valid-rmse:8.30403
      valid-r2:0.50982
[330] train-rmse:8.44532    train-r2:0.56937 valid-rmse:8.30214
      valid-r2:0.51004
[340] train-rmse:8.42141    train-r2:0.57181 valid-rmse:8.30014
      valid-r2:0.51027
[350] train-rmse:8.39842    train-r2:0.57414 valid-rmse:8.29835
      valid-r2:0.51048
[360] train-rmse:8.37635    train-r2:0.57638 valid-rmse:8.29697
      valid-r2:0.51065
```

```
[370] train-rmse:8.35410    train-r2:0.57862 valid-rmse:8.29262
      valid-r2:0.51116
[380] train-rmse:8.33095    train-r2:0.58095 valid-rmse:8.29179
      valid-r2:0.51126
[390] train-rmse:8.30154    train-r2:0.58391 valid-rmse:8.29098
      valid-r2:0.51136
[400] train-rmse:8.27336    train-r2:0.58673 valid-rmse:8.29057
      valid-r2:0.51140
[410] train-rmse:8.25648    train-r2:0.58841 valid-rmse:8.29034
      valid-r2:0.51143
[420] train-rmse:8.22607    train-r2:0.59144 valid-rmse:8.29332
      valid-r2:0.51108
[430] train-rmse:8.19721    train-r2:0.59430 valid-rmse:8.29163
      valid-r2:0.51128
[440] train-rmse:8.17110    train-r2:0.59688 valid-rmse:8.29193
      valid-r2:0.51124
[444] train-rmse:8.16357    train-r2:0.59762 valid-rmse:8.29272
      valid-r2:0.51115
```

```
## Predict  data_test 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)
```