```
# importing important libraries.

import pandas as pd
import numpy as np

# calling data sets.

mbtrain_df = pd.read_csv(r"C:\Users\Shams\Downloads\train.csv")
mbtest_df = pd.read_csv(r"C:\Users\Shams\Downloads\test.csv")

mbtrain_df.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]
```

```
print(f' {mbtrain_df.shape[0]} train observations \n
{mbtrain_df.shape[1]} train columns \n {mbtest_df.shape[0]} test
observations \n {mbtest_df.shape[1]} test columns')
```

```
 4209 train observations
 378 train columns
 4209 test observations
 377 test columns
```

```
mbtrain_df.columns[mbtrain_df.isnull().any()].tolist()
```

```
[]
```

```
# We can also count the null with:

mbtrain_df.isnull().sum().sum()
```

```
0
```

```
#  Variance of each column. (standard deviation ==0 will also do : mbtrain_df.std(axis=0))
mbtrain_df.var(axis=0)
```

```
C:\Users\Shams\AppData\Local\Temp\ipykernel_174660\837775923.py:2: FutureWarning: The default value of numeric_only in DataFrame.var is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  mbtrain_df.var(axis=0)
```

```
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
Length: 370, dtype: float64
```

```
type(mbtrain_df.var(axis=0))
```

```
C:\Users\Shams\AppData\Local\Temp\ipykernel_174660\3952969281.py:1: FutureWarning: The default value of numeric_only in DataFrame.var is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  type(mbtrain_df.var(axis=0))
```

```
pandas.core.series.Series
```

```
to_drop = [ind for ind,val in enumerate(mbtrain_df.var(axis=0)) if val ==0]
```

```
C:\Users\Shams\AppData\Local\Temp\ipykernel_174660\2804627180.py:1: FutureWarning: The default value of numeric_only in DataFrame.var is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_only=None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
  to_drop = [ind for ind,val in enumerate(mbtrain_df.var(axis=0)) if val ==0]
```

```
mbtrain_df.iloc[:,to_drop]
```

```
     X1  X85  X99  X225  X227  X260  X281  X282  X285  X289  X322
X339
0     v    1    0     0     0     0     0     0     1     0     0
0
1     t    1    0     0     0     0     0     0     1     0     0
0
2     w    1    0     0     0     0     0     0     0     0     0
0
3     t    0    0     0     0     0     0     0     0     0     0
0
4     v    0    0     0     0     0     0     0     0     0     0
0
...  ..  ...  ...   ...   ...   ...   ...   ...   ...   ...   ...   ..
.
4204  s    1    0     1     0     0     0     0     1     0     0
0
4205  o    0    0     0     0     0     1     0     0     0     0
0
4206  v    1    0     0     0     0     0     0     1     0     0
0
4207  r    0    0     0     0     0     0     0     0     0     0
0
4208  r    0    0     0     0     0     0     0     0     0     0
0

[4209 rows x 12 columns]
```

```
mbtrain_df.iloc[:,to_drop].nunique()
```

```
X1      27
X85      2
X99      2
X225     2
X227     2
X260     2
X281     2
X282     2
X285     2
X289     1
X322     2
X339     2
dtype: int64
```

```
mbtrain_df.iloc[:,to_drop].sum()
```

```
X1      vtwtvbrlsbrrbrslraacasaarbrslaasbvebssblvvslha...
X85                                                  1718
X99                                                    36
X225                                                  408
X227                                                   13
X260                                                    1
```

```
X281                                                    11
X282                                                    17
X285                                                   866
X289                                                     0
X322                                                    92
X339                                                     1
dtype: object
```

```
## Checking for null in test
mbtest_df.columns[mbtest_df.isnull().any()].tolist()
```

```
[]
```

```
mbtest_df.isnull().sum().sum()
```

```
0
```

```
mbtest_df.var(axis=0)
```

```
C:\Users\Shams\AppData\Local\Temp\ipykernel_174660\2073915530.py:1:
FutureWarning: The default value of numeric_only in DataFrame.var is
deprecated. In a future version, it will default to False. In
addition, specifying 'numeric_only=None' is deprecated. Select only
valid columns or specify the value of numeric_only to silence this
warning.
  mbtest_df.var(axis=0)
```

```
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
Length: 369, dtype: float64
```

```
to_drop = [ind for ind,val in enumerate(mbtest_df.var(axis=0)) if val
==0]
```

```
C:\Users\Shams\AppData\Local\Temp\ipykernel_174660\937008718.py:1:
FutureWarning: The default value of numeric_only in DataFrame.var is
deprecated. In a future version, it will default to False. In
addition, specifying 'numeric_only=None' is deprecated. Select only
valid columns or specify the value of numeric_only to silence this
warning.
  to_drop = [ind for ind,val in enumerate(mbtest_df.var(axis=0)) if
val ==0]
```

```
to_drop
```

```
[242, 243, 280, 281, 353]
```

```
mbtest_df.iloc[:,to_drop].nunique()
```

```
X249    2
X250    2
X287    2
X288    2
X361    2
dtype: int64
```

```
mbtest_df.iloc[:,to_drop].sum()
```

```
X249      40
X250    2340
X287      65
X288       1
X361    4051
dtype: int64
```

```
mbtrain_df.shape
```

```
(4209, 378)
```

```
mbtrain_df.drop(['X249', 'X287', 'X288', 'X99', 'X227', 'X260',
'X281', 'X282', 'X289', 'X322', 'X339'], axis=1, inplace=True)
mbtest_df.drop(['X249', 'X287', 'X288', 'X99', 'X227', 'X260', 'X281',
'X282', 'X289', 'X322', 'X339'], axis=1, inplace=True)
```

```
print(f' After dropping zero-variance proedictors \n
{mbtrain_df.shape[0]} train observations \n {mbtrain_df.shape[1]}
train columns \n {mbtest_df.shape[0]} test observations \n
{mbtest_df.shape[1]} test columns')
```

```
 After dropping zero-variance proedictors
 4209 train observations
 367 train columns
 4209 test observations
 366 test columns
```

```
# Apply label encoder.

# For testing set
train2encode = mbtrain_df.select_dtypes(include=['object']).columns
# For testing set
test2encode = mbtest_df.select_dtypes(include=['object']).columns
```

```
print(f'Number of unique values in columns to be encoded: \n\nTraining
set\n{mbtrain_df[train2encode].nunique()}\n\nTesting set\
n{mbtest_df[test2encode].nunique()}' )
```

```
Number of unique values in columns to be encoded:

Training set
X0    47
X1    27
X2    44
X3     7
X4     4
X5    29
X6    12
X8    25
dtype: int64

Testing set
X0    49
X1    27
X2    45
X3     7
X4     4
X5    32
X6    12
X8    25
dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
labelenc = LabelEncoder()

# For training set
for encoded in enumerate(train2encode):

mbtrain_df[encoded[1]]=labelenc.fit_transform(mbtrain_df[encoded[1]].v
alues)

mbtrain_df.head()
```

```
   ID       y  X0  X1  X2  X3  X4  X5  X6  X8  ...  X375  X376  X377
X378  \
0   0  130.81  32  23  17   0   3  24   9  14  ...     0     0     1
0
1   6   88.53  32  21  19   4   3  28  11  14  ...     1     0     0
0
2   7   76.26  20  24  34   2   3  27   9  23  ...     0     0     0
0
3   9   80.62  20  21  34   5   3  27  11   4  ...     0     0     0
0
4  13   78.02  20  23  34   5   3  12   3  13  ...     0     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

[5 rows x 367 columns]
```

```
# For testing set
for encoded in enumerate(test2encode):

mbtest_df[encoded[1]]=labelenc.fit_transform(mbtest_df[encoded[1]].val
ues)

mbtest_df.head()
```

```
   ID  X0  X1  X2  X3  X4  X5  X6  X8  X10  ...  X375  X376  X377
X378  X379  \
0   1  21  23  34   5   3  26   0  22    0  ...     0     0     0
1     0
1   2  42   3   8   0   3   9   6  24    0  ...     0     0     1
0     0
2   3  21  23  17   5   3   0   9   9    0  ...     0     0     0
1     0
3   4  21  13  34   5   3  31  11  13    0  ...     0     0     0
1     0
4   5  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 366 columns]
```

```
# Dropping ID
mbtrain_df.drop('ID',axis=1,inplace=True)
mbtest_df.drop('ID',axis=1,inplace=True)

# duplicates rows in training set?
print(mbtrain_df.duplicated().any())
```

```
True
```

```
# duplicates rows in testing set?
print(mbtest_df.duplicated().any())
```

```
True
```

```
# For training set
mbtrain_df.loc[:,mbtrain_df.columns.str.contains("^X")].describe().T
```

```
       count       mean        std  min   25%   50%   75%   max
X0    4209.0  29.760751  13.738338  0.0  19.0  35.0  43.0  46.0
X1    4209.0  11.113566   8.531001  0.0   3.0  13.0  20.0  26.0
X2    4209.0  17.306486  10.899914  0.0   8.0  16.0  25.0  43.0
X3    4209.0   2.919696   1.739912  0.0   2.0   2.0   5.0   6.0
X4    4209.0   2.997862   0.073900  0.0   3.0   3.0   3.0   3.0
...      ...        ...        ...  ...   ...   ...   ...   ...
X380  4209.0   0.008078   0.089524  0.0   0.0   0.0   0.0   1.0
X382  4209.0   0.007603   0.086872  0.0   0.0   0.0   0.0   1.0
X383  4209.0   0.001663   0.040752  0.0   0.0   0.0   0.0   1.0
X384  4209.0   0.000475   0.021796  0.0   0.0   0.0   0.0   1.0
X385  4209.0   0.001426   0.037734  0.0   0.0   0.0   0.0   1.0

[365 rows x 8 columns]
```

```
# For testing set
mbtest_df.loc[:,mbtest_df.columns.str.contains("^X")].describe().T
```

```
       count       mean        std  min   25%   50%   75%   max
X0    4209.0  30.515324  15.221177  0.0  20.0  36.0  45.0  48.0
X1    4209.0  11.075315   8.544520  0.0   3.0  13.0  20.0  26.0
X2    4209.0  17.780708  10.227319  0.0  10.0  17.0  23.0  44.0
X3    4209.0   2.933476   1.776977  0.0   2.0   2.0   5.0   6.0
X4    4209.0   2.997149   0.078553  0.0   3.0   3.0   3.0   3.0
...      ...        ...        ...  ...   ...   ...   ...   ...
X380  4209.0   0.008078   0.089524  0.0   0.0   0.0   0.0   1.0
X382  4209.0   0.008791   0.093357  0.0   0.0   0.0   0.0   1.0
X383  4209.0   0.000475   0.021796  0.0   0.0   0.0   0.0   1.0
X384  4209.0   0.000713   0.026691  0.0   0.0   0.0   0.0   1.0
X385  4209.0   0.001663   0.040752  0.0   0.0   0.0   0.0   1.0

[365 rows x 8 columns]
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import Normalizer

mbtrain_df.shape
```

```
(4209, 366)
```

```
mbtrain_df.head()
```

```
        y  X0  X1  X2  X3  X4  X5  X6  X8  X10  ...  X375  X376  X377
X378  \
0  130.81  32  23  17   0   3  24   9  14    0  ...     0     0     1
0
1   88.53  32  21  19   4   3  28  11  14    0  ...     1     0     0
0
2   76.26  20  24  34   2   3  27   9  23    0  ...     0     0     0
```

```
0
3   80.62  20  21  34   5   3  27  11   4    0  ...     0     0     0
0
4   78.02  20  23  34   5   3  12   3  13    0  ...     0     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

[5 rows x 366 columns]
```

```
# Normalizing and scaling training set (without the target)
train_scaler=Normalizer().fit(mbtrain_df.drop('y',1))
norm_train_df = train_scaler.transform(mbtrain_df.drop('y',1))
norm_train_df.shape
```

```
C:\Users\Shams\AppData\Local\Temp\ipykernel_174660\4031491797.py:2:
FutureWarning: In a future version of pandas all arguments of
DataFrame.drop except for the argument 'labels' will be keyword-only.
  train_scaler=Normalizer().fit(mbtrain_df.drop('y',1))
C:\Users\Shams\AppData\Local\Temp\ipykernel_174660\4031491797.py:3:
FutureWarning: In a future version of pandas all arguments of
DataFrame.drop except for the argument 'labels' will be keyword-only.
  norm_train_df = train_scaler.transform(mbtrain_df.drop('y',1))
```

```
(4209, 365)
```

```
# Normalizing and scaling testing set
test_scaler=Normalizer().fit(mbtest_df)
norm_test_df = test_scaler.transform(mbtest_df)
norm_test_df.shape
```

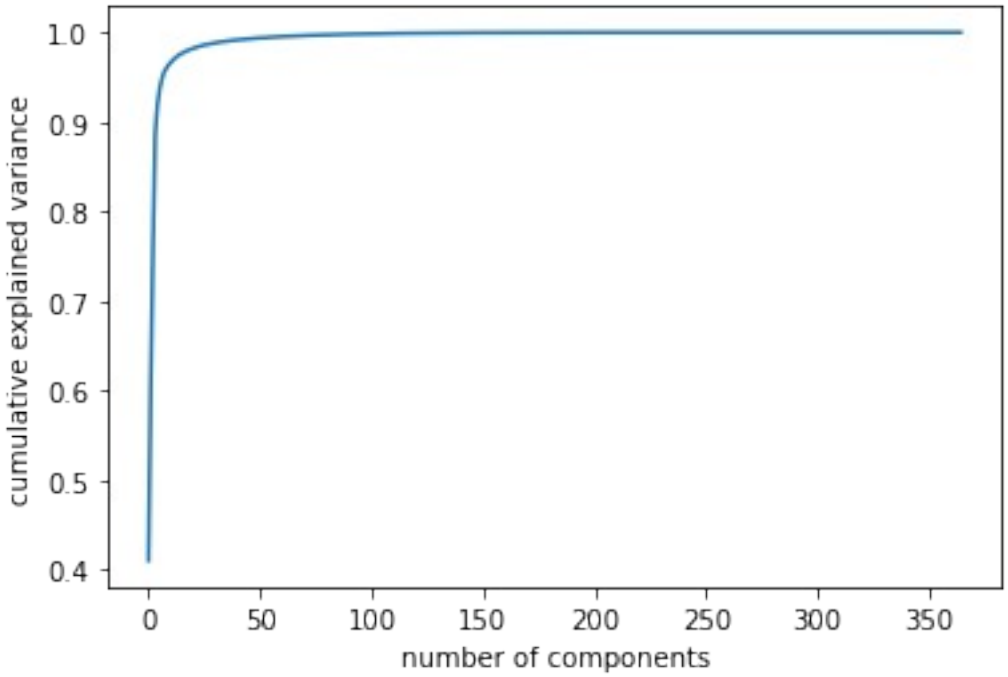
```
(4209, 365)
```

```
pca =PCA()
pca.fit(norm_train_df)
```

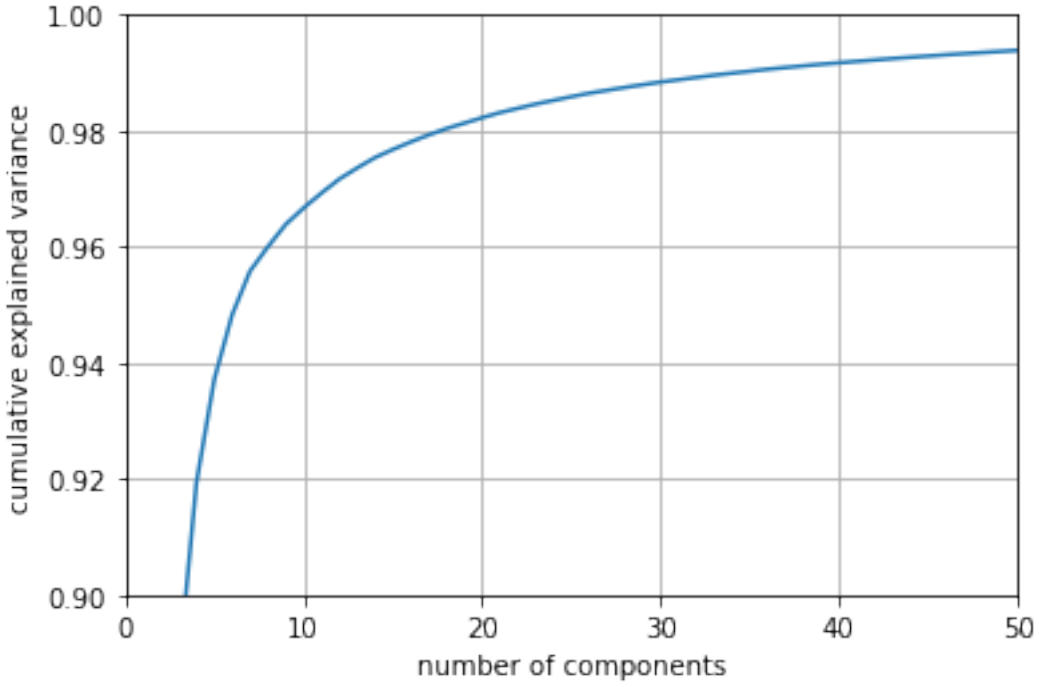
```
PCA()
```

```
f =np.cumsum(pca.explained_variance_ratio_)
plt.plot(f)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
```

```
Text(0, 0.5, 'cumulative explained variance')
```




```
plt.plot(f)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.grid(True)
plt.xlim(0,50)
plt.ylim(.9,1)
plt.show()
```



```
print(f'The mean explained_variance_ratio is
{np.mean(pca.explained_variance_ratio_)}')
```

```
The mean explained_variance_ratio is 0.0027397260273972603
```

```
print(f'The number of features whose mean are greater than the mean
explained_variance_ratio is {np.sum([pca.explained_variance_ratio_ >
np.mean(pca.explained_variance_ratio_)])}')
```

```
The number of features whose mean are greater than the mean
explained_variance_ratio is 11
```

```
pca = PCA(n_components=0.97, whiten=True)
norm_features = pca.fit_transform(norm_train_df)
```

```
print(f'The midpoint method retains {norm_features.shape[1]} candidate
features')
```

```
The midpoint method retains 13 candidate features
```

```
# Predict your test_df values using XGBoost.
# Import necessary modules
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
```

```
print(xgb.__version__)
```

```
1.7.1
```

```
# Splitting into training and validations  sets
X_train, X_val, y_train, y_val =
train_test_split(mbtrain_df.iloc[:,1:], mbtrain_df['y'].values,
test_size=0.25, random_state=4321)
```

```
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

```
((3156, 365), (1053, 365), (3156,), (1053,))
```

```
train_xgb_reg = xgb.XGBRegressor( objective = 'reg:squarederror',
colsample_bytree = 0.1,  learning_rate = 0.2, max_depth = 7, alpha =
10)
```

```
train_xgb_reg.fit(X_train,y_train)
train_valid = train_xgb_reg.predict(X_val)
```

```
training_rmse = np.sqrt(mean_squared_error(y_val, train_valid))
print(f'Training RMSE: {training_rmse}')
```

```
Training RMSE: 9.804432369870364
```

```
train_xgb_reg.score(X_train,y_train)
```

```
0.7896672187502718
```

```
testing_preds = train_xgb_reg.predict(mbtest_df)
```

```
testing_rmse = np.sqrt(mean_squared_error(mbtrain_df['y'],
testing_preds))
print(f'Testing RMSE: {testing_rmse}')
```

```
Testing RMSE: 15.670846627487933
```

Conclusion We can see that we got a larger RMSE for the testing. This suggest the model did not do well on the testing set. A better way would be to use the Cross Validation method of XGBoost to help identify the features that will yield a better training RMSE. We then use the model with a better RMSE to predict.