```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from xgboost import XGBRegressor
from xgboost import XGBRFRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt

import warnings
warnings.filterwarnings('ignore')

# import train dataset
mdz_train_df = pd.read_csv('train.csv')
mdz_train_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]
```

```
mdz_train_df.shape
```

```
(4209, 378)
```

```
# import test dataset
mdz_test_df = pd.read_csv('test.csv')
mdz_test_df.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]
```

```
mdz_test_df.shape
```

```
(4209, 377)
```

In both datasets, we dont need the ID column. Lets remove it.

```
mdz_train_df.drop(['ID'], axis=1, inplace=True)
mdz_test_df.drop(['ID'], axis=1, inplace=True)
```

Task 1. If for any column(s), the variance is equal to zero, then you need to remove those variable(s)

Now lets remove the columns having variance equal to zero in train dataset.

```
var_zero_cols = mdz_train_df.var()[mdz_train_df.var() ==
0].index.values.tolist()
var_zero_cols
```

```
['X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X289',
 'X290',
 'X293',
 'X297',
```

```
 'X330',
 'X347']

# Remove above columns from train and test
mdz_train_df.drop(var_zero_cols, axis=1, inplace=True)
mdz_test_df.drop(var_zero_cols, axis=1, inplace=True)

print(mdz_train_df.shape)
print(mdz_test_df.shape)
```

```
(4209, 365)
(4209, 364)
```

Task 2. Check for null and unique values for test and train sets

```
mdz_train_df.isnull().sum().sum()
```

```
0
```

```
mdz_test_df.isnull().sum().sum()
```

```
0
```

There is no NaN values in both train and test dataset.

```
mdz_train_df.nunique().sum()
```

```
3452
```

```
mdz_test_df.nunique().sum()
```

```
908
```

There are 3452 unique values in train and 908 unique values in test datasets.

Task 3. Apply label encoder

```
object_cols = mdz_train_df.describe(include=[object]).columns.values
object_cols
```

```
array(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype=object)
```

Above columns are of object datatype.

```
mdz_train_df[object_cols].head()
```

```
   X0 X1  X2 X3 X4 X5 X6 X8
0   k  v  at  a  d  u  j  o
1   k  t  av  e  d  y  l  o
2  az  w   n  c  d  x  j  x
3  az  t   n  f  d  x  l  e
4  az  v   n  f  d  h  d  n
```

```
le = LabelEncoder()
```

```
for col in object_cols:
    le.fit(mdz_train_df[col].append(mdz_test_df[col]).values)
    mdz_train_df[col] = le.transform(mdz_train_df[col])
    mdz_test_df[col] = le.transform(mdz_test_df[col])
```

Now we can proceed with the dimensionality reduction and ML model.

```
x = mdz_train_df.drop(['y'], axis=1)
y = mdz_train_df.y

# train test split

x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=0)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2946, 364)
(1263, 364)
(2946,)
(1263,)
```

Task 4. Perform dimensionality reduction

```
pca = PCA(0.98)

pca.fit(x)
pca.n_components_
```

```
12
```

So we need 12 components generated from all 364 columns to get 98% variance and project all the data to lower dimension.

```
pca.explained_variance_ratio_
```

```
array([0.40868988, 0.21758508, 0.13120081, 0.10783522, 0.08165248,
       0.0140934 , 0.00660951, 0.00384659, 0.00260289, 0.00214378,
       0.00209857, 0.00180388])
```

Above are the explained varance ratio/percentage (high to low).

```
# Convert the data into lower dimension (ld)

x_train_ld = pca.transform(x_train)
x_test_ld = pca.transform(x_test)
x_test_set_ld = pca.transform(mdz_test_df)

print(x_train_ld.shape)
```

```
print(x_test_ld.shape)
print(x_test_set_ld.shape)
```

```
(2946, 12)
(1263, 12)
(4209, 12)
```

Task 5. Predict your test_df values using xgboost

We can build xgboost (XGBRegeessor, XGBRFRegressor) models with different configuration of learning rates.

```
xgbRegressor_1 = XGBRegressor(objective='reg:squarederror',
learning_rate = 0.1)
xgbRegressor_1.fit(x_train_ld, y_train)
```

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=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=6,
             min_child_weight=1, missing=nan,
monotone_constraints=None,
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
subsample=1,
             tree_method=None, validate_parameters=False,
verbosity=None)
```

```
xgbRegressor_1_y_pred = xgbRegressor_1.predict(x_test_ld)
print('RMSE', sqrt(mean_squared_error(y_test, xgbRegressor_1_y_pred)))
```

```
RMSE 9.996668252109297
```

```
xgbRegressor_2 = XGBRegressor(objective='reg:squarederror',
learning_rate = 1)
xgbRegressor_2.fit(x_train_ld, y_train)
```

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-
1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=1, max_delta_step=0, max_depth=6,
min_child_weight=1,
             missing=nan, monotone_constraints=None, n_estimators=100,
n_jobs=0,
             num_parallel_tree=1, random_state=0, reg_alpha=0,
reg_lambda=1,
             scale_pos_weight=1, subsample=1, tree_method=None,
             validate_parameters=False, verbosity=None)
```

```
xgbRegressor_2_y_pred = xgbRegressor_2.predict(x_test_ld)
print('RMSE', sqrt(mean_squared_error(y_test, xgbRegressor_2_y_pred)))
```

```
RMSE 12.141280477327937
```

Here we can choose xgbRegressor_1 model. It gave less RMSE.

```
xgbRFRegressor_1 = XGBRFRegressor(objective='reg:squarederror',
learning_rate = 0.1)
xgbRFRegressor_1.fit(x_train_ld, y_train)
```

```
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=6, min_child_weight=1,
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,
               tree_method=None, validate_parameters=False,
verbosity=None)
```

```
xgbRFRegressor_1_y_pred = xgbRFRegressor_1.predict(x_test_ld)
print('RMSE', sqrt(mean_squared_error(y_test,
xgbRFRegressor_1_y_pred)))
```

```
RMSE 91.53699087619204
```

```
xgbRFRegressor_2 = XGBRFRegressor(objective='reg:squarederror',
learning_rate = 1)
xgbRFRegressor_2.fit(x_train_ld, y_train)
```

```
XGBRFRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
               colsample_bytree=1, gamma=0, gpu_id=-1,
importance_type='gain',
               interaction_constraints=None, max_delta_step=0,
max_depth=6,
               min_child_weight=1, 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, tree_method=None,
validate_parameters=False,
               verbosity=None)
```

```
xgbRFRegressor_2_y_pred = xgbRFRegressor_2.predict(x_test_ld)
print('RMSE', sqrt(mean_squared_error(y_test,
xgbRFRegressor_2_y_pred)))
```

```
RMSE 10.313216428415938
```

XGBRegressor model with learning rate 0.1 gave the least RMSE. So we can choose this model for prediction.

```
# Predict for test dataset
xgbRegressor_1.predict(x_test_set_ld)
```

```
array([ 76.920044,  93.76541 ,  76.93008 , ..., 100.39338 ,
107.170105,
        93.93597 ], dtype=float32)
```