```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from math import sqrt
import xgboost as xgb
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
%matplotlib inline


# read train and test data
df_train=pd.read_csv('train.csv')
df_test=pd.read_csv('test.csv')


df_train.head()
```

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	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	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	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	0	0

5 rows × 378 columns

```
df_test.head()
```

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X3
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0	0	0	
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0	0	0	
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0	0	0	
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0	0	0	
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0	0	0	

5 rows × 377 columns

Explore Data

```
#print first five records
df_train.head()
```

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	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	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	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	0	0

5 rows × 378 columns

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 378 entries, ID to X385
dtypes: float64(1), int64(369), object(8)
memory usage: 12.1+ MB
```

```
df_train.describe()
```

	ID	y	X10	X11	X12	X13	X14	X15
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.000000
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.000475
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.021796
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.000000
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.000000

8 rows × 370 columns

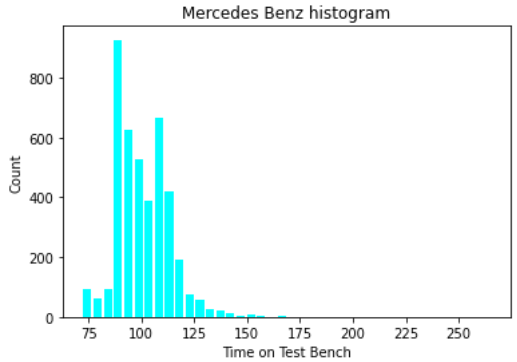
```
df_train['X0'].head()
```

```
0     k
1     k
2    az
3    az
4    az
Name: X0, dtype: object
```

```
df_train['y'].head()
```

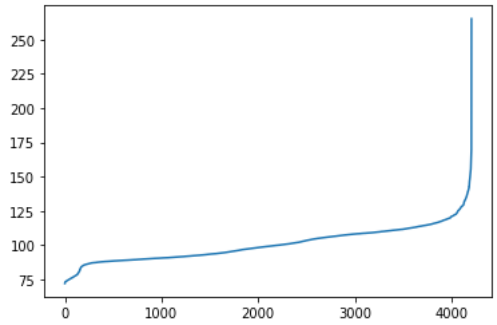
```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

```
#Plot Histograms of Y
Y = df_train['y'].sort_values()
range = (0, 100)
bins = 40
plt.hist(Y, bins, color = 'cyan',histtype = 'bar', rwidth = 0.8)
plt.xlabel('Time on Test Bench')
plt.ylabel('Count')
plt.title('Mercedes Benz histogram')
plt.show()
```




```
#plot Histogram of Y
x= np.arange(0,4209,1)
plt.plot(x,Y)
```

```
[<matplotlib.lines.Line2D at 0x7f27b6d3f1c0>]
```



```
# Find null values
df_train.isnull().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
```

Columns with zero variance

```
#Check columns with zero variance
df_train.var()==0
```

```
<ipython-input-15-49e6a90c249c>:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Se
  df_train.var()==0
ID      False
y       False
X10     False
X11      True
X12     False
        ...
X380    False
X382    False
X383    False
X384    False
X385    False
Length: 370, dtype: bool
```

```
# If any feature has no variance max=min, it is considered an unhelpful feature.
unhelpful_features = []
for feature in df_train:
    if max(df_train[feature]) == min(df_train[feature]):
    #if df_train[feature].var()==0:
        print(feature)
        unhelpful_features.append(feature)
```

```
X11
X93
X107
X233
X235
X268
X289
X290
X293
X297
X330
X347
```

```
# for zero variance columns length of unique values is 1
len(np.unique(df_train['X93']))
```

```
1
```

```
df_train['y'].head()
```

```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

```
# Find the categorical features, which will need to be converted into dummy features.
dummies = []
for column in df_train:
    if max(df_train[column]) != 1:
        print(column)
        dummies.append(column)
print(dummies)
df_train['y'].head()
```

```
ID
y
X0
X1
X2
X3
X4
X5
X6
X8
X11
X93
X107
X233
X235
X268
X289
X290
X293
X297
X330
X347
['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297', 'X330', 'X347']
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

```
# remove ID , y to get the list of feature columns
usable_columns = list(set(dummies) - set(['ID', 'y']))
print(usable_columns)
```

```
['X3', 'X6', 'X233', 'X293', 'X1', 'X8', 'X297', 'X330', 'X5', 'X268', 'X107', 'X93', 'X2', 'X4', 'X290', 'X347', 'X235', 'X289', 'X11', 'X0']
```

```
# label encode categorical data in train_f dataset
label_encoder = LabelEncoder()
df_train['X0'] = label_encoder.fit_transform(df_train['X0'])
df_train['X1'] = label_encoder.fit_transform(df_train['X1'])
df_train['X2'] = label_encoder.fit_transform(df_train['X2'])
df_train['X3'] = label_encoder.fit_transform(df_train['X3'])
df_train['X4'] = label_encoder.fit_transform(df_train['X4'])
df_train['X5'] = label_encoder.fit_transform(df_train['X5'])
df_train['X6'] = label_encoder.fit_transform(df_train['X6'])
df_train['X8'] = label_encoder.fit_transform(df_train['X8'])
```

```
# label encode categorical data in test dataset
df_test['X0'] = label_encoder.fit_transform(df_test['X0'])
df_test['X1'] = label_encoder.fit_transform(df_test['X1'])
df_test['X2'] = label_encoder.fit_transform(df_test['X2'])
df_test['X3'] = label_encoder.fit_transform(df_test['X3'])
df_test['X4'] = label_encoder.fit_transform(df_test['X4'])
df_test['X5'] = label_encoder.fit_transform(df_test['X5'])
df_test['X6'] = label_encoder.fit_transform(df_test['X6'])
df_test['X8'] = label_encoder.fit_transform(df_test['X8'])
```

```
#print first five records of feature columns
df_train[usable_columns].head()
df_train['y'].head()
```

```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

```
# find the correlation of feature columns
df_train[usable_columns].corr()
```

	X3	X6	X233	X293	X1	X8	X297	X330	X5	X268	X107	X93	
X3	1.000000	-0.048468	NaN	NaN	0.205657	-0.001249	NaN	NaN	-0.008161	NaN	NaN	NaN	-0.093
X6	-0.048468	1.000000	NaN	NaN	-0.079119	0.018565	NaN	NaN	-0.019917	NaN	NaN	NaN	0.065
X233	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
X293	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
X1	0.205657	-0.079119	NaN	NaN	1.000000	-0.000306	NaN	NaN	0.046417	NaN	NaN	NaN	0.088
X8	-0.001249	0.018565	NaN	NaN	-0.000306	1.000000	NaN	NaN	0.012746	NaN	NaN	NaN	-0.069
X297	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
X330	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
X5	-0.008161	-0.019917	NaN	NaN	0.046417	0.012746	NaN	NaN	1.000000	NaN	NaN	NaN	-0.017
X268	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

```
df_train['X3'].value_counts()
```

```
2    1942
5    1076
0     440
3     290
6     241
4     163
1      57
Name: X3, dtype: int64
```

X235	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

```
df_train['X4'].value_counts()
```

```
3    4205
0       2
1       1
2       1
Name: X4, dtype: int64
```

```
df_train['X6'].value_counts()
```

```
6     1042
9     1039
3      625
8      488
11     478
0      206
7      190
10      43
2       38
1       28
5       20
4       12
Name: X6, dtype: int64
```

```
#df_train=df_train[df_train['y']<150]
df_train['y'].head()
```

```
df_train.max()
```

```
ID      8417.00
y        265.32
X0        46.00
X1        26.00
X2        43.00
          ...
X380       1.00
X382       1.00
X383       1.00
X384       1.00
X385       1.00
Length: 378, dtype: float64
```

```
#split data into train and test(valid)
X = df_train[usable_columns]
y = df_train['y']
X_train, X_valid, y_train, y_valid = train_test_split(X,y , test_size=0.2,random_state=4242)
```

```
#print shape of train data
print(X_train.shape)
print(y_train.shape)
```

```
(3367, 20)
(3367,)
```

```
#print shape of test data
print(X_valid.shape)
print(y_valid.shape)
```

```
(842, 20)
(842,)
```

```
#print first five records of Y
df_train['y'].head()
```

```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

```
#stanadarise the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_valid = sc.fit_transform(X_valid)
```

```
# find the pricipal components
n_comp = 2
pca = PCA(n_components=n_comp, random_state=42)
pca2_results_train = pca.fit_transform(X_train)
pca2_results_test = pca.transform(X_valid)
```

```
#for efficient use of the xgboost model , convert dataset to the DMatrix format

d_train = xgb.DMatrix(pca2_results_train, label=y_train)
d_valid = xgb.DMatrix(pca2_results_test)


# Initialise a set of parameters
param = {'eta': 0.02, 'max_depth' : 4, 'objective' :'reg:linear','eval_metric': 'rmse'} ##### , 'num_class' : 0  }


# Train the model
xgb_model  = xgb.train(param, d_train , 20 )
```

```
[07:56:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Predict values for test data and calculte accuracy for train.csv

```
# predict the values for test from training data
y_xgb_pred_valid = xgb_model.predict(d_valid)


# Calculate accuracy , Evaluate the model.
print("The test accuracy for xgb model on Mercedes Benz is :")
print(r2_score(y_valid, y_xgb_pred_valid))
```

```
The test accuracy for xgb model on Mercedes Benz is :
-31.17274202187935
```

```
# Calculate and print RMSE
rmse = sqrt(mean_squared_error(y_valid, y_xgb_pred_valid))
print(rmse)
```

```
67.27489680734955
```

Predict values for training data and calculte accuracy for train.csv

```
# predict the values for training from training data
y_xgb_pred_train = xgb_model.predict(d_train)


# Calculate accuracy , Evaluate the model for training data
print("The test accuracy for xgb model on Mercedes Benz is :")
print(r2_score(y_train, y_xgb_pred_train))
```

```
The test accuracy for xgb model on Mercedes Benz is :
-27.085769388715686
```

```
# Calculate and print RMSE
rmse = sqrt(mean_squared_error(y_train, y_xgb_pred_train))
print(rmse)
```

```
68.20362396381216
```

Predict Values for test data(test.csv) using Training model

```
# Test data
X_test=df_test[usable_columns]
```

```
# find the pricipal components for test data
n_comp = 2
pca = PCA(n_components=n_comp, random_state=42)
pca2_X_test = pca.fit_transform(X_test)
pca2_results_test = pca.transform(X_valid)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but PCA was fitted with feature names
  warnings.warn(
```

```
# Creating DMatrices for Xgboost training with test data
d_test = xgb.DMatrix(pca2_X_test)
```

```
# predict the values for test data
X_test_pred = xgb_model.predict(d_test)
```

```
print(X_test_pred)
```

```
[32.929497 34.25553  35.535168 ... 32.929497 35.535168 34.25553 ]
```

✓ 0s completed at 1:24 PM