```
#Mercedes project
import numpy as np
import pandas as pd
import matplotlib.pyplot as pl
```

```
cd
```

```
/root
```

```
cd /content/sample_data
```

```
/content/sample_data
```

```
#Fetch train data
x_train = pd.read_csv('train.csv')
```

```
x_train.head(3)
```

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17
0	0	130.81	k	v	at	a	d	u	j	o	0	0	0	1	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	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	0	1

3 rows × 378 columns

```
#Fetch test data
x_test = pd.read_csv('test.csv')
```

```
x_test.head(3)
```

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17	X18	X
0	1	az	v	n	f	d	t	a	w	0	0	0	0	0	0	0	0	0	
1	2	t	b	ai	a	d	b	g	y	0	0	0	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	

3 rows × 377 columns

```
#Check the dimensions
x_train.shape
```

```
(4209, 378)
```

```
x_test.shape
```

```
(4209, 377)
```

```
#We can drop the ID column as currently it is not required
x_train.drop("ID",inplace=True, axis=1)
```

```
x_test.drop("ID", inplace=True, axis=1)
```

```
x_test.head(2)
```

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19
0	az	v	n	f	d	t	a	w	0	0	0	0	0	0	0	0	0	0
1	t	b	ai	a	d	b	g	y	0	0	0	0	0	0	0	0	0	1

2 rows × 376 columns

```
#Need to check for Zero variance column and if available they should be dropped.
zero_var = x_train.var()[x_train.var() == 0].index.values
zero_var
```

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

```
x_train.drop(zero_var, inplace=True, axis=1)
```

```
x_train.head(2)
```

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19
0	130.81	k	v	at	a	d	u	j	o	0	0	1	0	0	0	0	1	0
1	88.53	k	t	av	e	d	y	l	o	0	0	0	0	0	0	0	1	0

2 rows × 365 columns

```
x_test.drop(zero_var, inplace=True, axis=1)
```

```
x_test.head(2)
```

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20
0	az	v	n	f	d	t	a	w	0	0	0	0	0	0	0	0	0	0
1	t	b	ai	a	d	b	g	y	0	0	0	0	0	0	0	0	1	0

2 rows × 364 columns

```
#Check if any null values exist
```

```
#check if any null values exist.
np.sum(x_train.isnull().sum())
```

```
0
```

```
x_train.shape
```

```
(4209, 365)
```

```
x_train.shape
```

```
(4209, 365)
```

```
x_train.isnull().sum()
```

```
y       0
X0      0
X1      0
X2      0
X3      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 365, dtype: int64
```

```
#check if any columns which are not numeric
x_train.describe(include=['object'])
```

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	47	27	44	7	4	29	12	25
top	z	aa	as	c	d	w	g	j
freq	360	833	1659	1942	4205	231	1042	277

```
x_val = x_train.describe(include=['object']).columns.values
```

```
x_val
```

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

```
#label Encoder
from sklearn.preprocessing import LabelEncoder
```

```
y_val = x_test.describe(include=['object']).columns.values
y_val
```

```
#Apply label Encoder for the columns which are not numeric. DO it for both train & test da
 le = LabelEncoder()


for i , val in enumerate(list(x_val)):
 if(x_train[val].dtype in [object]):
    x_train[val]=le.transform(x_train[val])
    le.fit_transform(x_train['X0'])


np.unique(x_train['X5'].values)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28])
```

```
x_test.head()
```

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20
0	az	v	n	f	d	t	a	w	0	0	0	0	0	0	0	0	0	0
1	t	b	ai	a	d	b	g	y	0	0	0	0	0	0	0	0	1	0
2	az	v	as	f	d	a	j	j	0	0	0	1	0	0	0	0	0	0
3	az	l	n	f	d	z	l	n	0	0	0	0	0	0	0	0	0	0
4	w	s	as	c	d	y	i	m	0	0	0	1	0	0	0	0	0	0

5 rows × 364 columns

```
np.unique(x_train['X0'].values)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46])
```

```
for i , val in enumerate(list(x_test.columns)):
  if(x_test[val].dtype in [object]):
    x_test[val]=le.fit_transform(x_test[val])
le.fit_transform(x_test['X0'])
```

```
array([21, 42, 21, ..., 47,  7, 42])
```

```
#PCA
from sklearn.decomposition import PCA


x_train.head()
```

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19
0	130.81	32	23	17	0	3	24	9	14	0	0	1	0	0	0	0	1	0
1	88.53	32	21	19	4	3	28	11	14	0	0	0	0	0	0	0	1	0
2	76.26	20	24	34	2	3	27	9	23	0	0	0	0	0	0	1	0	0
3	80.62	20	21	34	5	3	27	11	4	0	0	0	0	0	0	0	0	0
4	78.02	20	23	34	5	3	12	3	13	0	0	0	0	0	0	0	0	0

5 rows × 365 columns

```
#Reduce the data set and retain the required columns using PCA
pcaModel = PCA(n_components=12,svd_solver='full')
pcaModel
#We can remove column y from our training data set as the values are already availble for
train_df = x_train.drop('y', axis=1)
test_df=x_train['y']


pcaModel.fit(train_df)
```

```
PCA(copy=True, iterated_power='auto', n_components=12, random_state=None,
    svd_solver='full', tol=0.0, whiten=False)
```

```
pcaModel.n_components_
#pca.explained_variance_ratio_
```

```
12
```

```
#Below array comprises of 98% variance if we sum it up.
pcaModel.explained_variance_ratio_
```

```
array([0.38334782, 0.21388033, 0.13261866, 0.11826642, 0.09206008,
       0.01590604, 0.0074454 , 0.00433701, 0.00294021, 0.00241796,
       0.00236488, 0.00203229])
```

```
#Split to get training and validation data
from sklearn.model_selection import train_test_split
X_train, X_val, Y_train, Y_val = train_test_split(train_df, test_df, test_size=0.2, random

X_train
```

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X2
1006	34	23	16	2	3	6	10	23	0	0	1	1	0	0	0	0	0	
1840	35	20	16	3	3	4	6	18	0	0	0	1	0	0	0	0	0	
3792	27	4	33	2	3	25	8	4	0	0	0	0	0	0	0	0	0	
152	31	1	27	3	3	13	8	4	0	0	0	0	0	0	0	0	0	
1557	19	10	16	2	3	3	11	10	0	0	0	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
3444	31	10	16	2	3	22	11	17	0	0	0	1	0	0	0	0	0	
466	20	25	25	2	3	9	9	9	0	0	0	1	0	0	1	0	0	
3092	45	24	3	2	3	21	8	2	0	0	0	0	0	0	0	0	0	
3772	45	19	8	5	3	25	8	1	0	1	0	0	0	0	0	0	0	
860	22	1	7	2	3	5	9	17	0	0	0	0	0	0	0	0	1	

3367 rows × 364 columns

```
Y_train.head(2)
```

```
1006    88.96
1840    89.90
Name: y, dtype: float64
```

```
from sklearn.metrics import mean_squared_error
```

```
 pca_X_train = pd.DataFrame(pcaModel.transform(X_train))
```

```
pca_X_train.describe()
```

	0	1	2	3	4	5	33
count	3367.000000	3367.000000	3367.000000	3367.000000	3367.000000	3367.000000	
mean	-0.033240	-0.197852	0.005932	-0.071544	-0.064515	0.015421	
std	14.263326	10.599765	8.464366	7.933253	7.040619	2.896251	
min	-21.899802	-21.151050	-20.048229	-20.802751	-13.745017	-5.036026	
25%	-11.827716	-7.337372	-6.681494	-5.914770	-5.851388	-2.131484	
50%	-3.838910	-2.602931	-0.189175	-0.369078	-0.658366	-0.206804	
75%	11.588808	5.303114	6.146733	5.326945	5.928378	1.243352	
max	34.764631	30.136712	21.277634	21.810742	14.501954	7.530676	

```
#Transform pca_X_val (validation data) in the model
```

```
pca_X_val = pd.DataFrame(pcaModel.transform(X_val))
pca_X_val
```

	0	1	2	3	4	5	6	
0	17.841395	-15.245960	-6.299416	4.391821	-0.724654	-2.742346	-2.120028	2.94
1	-0.990921	-15.208289	1.040124	2.620286	9.256619	-1.371609	-1.857634	1.88
2	-2.894524	3.927628	-1.575664	-10.600740	3.257902	-3.705331	-0.045750	-0.29
3	13.849039	1.874404	6.003682	8.912134	-0.749994	-2.604863	1.145621	1.26
4	20.639205	-15.373246	6.846871	3.755760	4.645073	-1.805657	0.259409	0.66
...	...	...	...	...	...	...	...	
837	-11.291745	19.029773	-6.784557	-1.324391	2.551003	4.524591	1.093646	-0.04
838	-8.140382	17.085313	7.767719	-12.088768	11.392493	-1.464076	0.854631	1.08
839	-16.952539	7.863151	-8.961566	-0.393542	6.819587	0.493156	-0.075996	-0.18
840	1.756738	-14.990083	13.314169	4.650982	-1.532119	0.151467	2.673551	0.90
841	-10.289994	2.836025	10.100763	10.139382	-11.975228	-2.744282	-1.625649	-1.92

842 rows × 12 columns

```
#Transform pca_X_Test(testing data) in the model
pca_X_test = pd.DataFrame(pcaModel.transform(x_test))
pca_X_test
```

	0	1	2	3	4	5	6	
0	15.122597	12.426344	16.575688	0.381591	10.749272	6.778299	-1.966062	2.2
1	-16.418523	-6.087805	-5.818108	-0.643844	11.846740	0.972063	3.852835	1.9
2	11.310890	-2.240987	-5.683210	15.249597	-2.777294	-2.557530	-1.093927	0.4
3	12.883278	13.284800	14.409574	-11.039926	2.535217	-3.919070	-2.910159	2.0
4	-12.119759	3.021287	20.739083	0.602495	-1.087117	-1.422969	0.863914	-3.1
...	...	...	...	...	...	...	...	
4204	22.045696	-5.791779	-14.969442	0.445593	-6.315912	-2.175568	-2.797583	-1.5
4205	-16.799934	-5.849004	-13.362178	2.585832	12.125719	-1.671761	0.317114	1.3
4206	-13.467664	3.524155	-0.387634	20.586709	9.051954	3.609708	-2.345971	-1.4
4207	24.086925	-6.512201	-6.389509	12.212753	4.557659	4.376524	3.877389	-2.0
4208	-16.557942	-5.495652	-13.703891	2.256959	5.142868	1.176588	0.380221	1.2

4209 rows × 12 columns

```
#Building XGBOOST model for linear regression
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
```

```
model = xgb.XGBRegressor(objective='reg:linear', learning_rate=0.1)
```

```
pca_X_train.columns
```

```
Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 'y'], dtype='object')
```

```
pca_X_train.drop(columns=['y'], inplace= True, axis=1)
```

```
pca_X_train.columns
```

```
Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], dtype='object')
```

```
#Fit below values in the model
model.fit(pca_X_train.values, Y_train.values)
```

```
[15:42:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

```
pred_y_val = model.predict(pca_X_val.values)
```

```
#Calculate mean square value
mse_score = mean_squared_error(Y_val,pred_y_val)
```

```
from sklearn.metrics import r2_score
print('Proj R2 score is: ', r2_score(Y_val, pred_y_val))
```

```
Proj R2 score is:  0.4537282860932641
```