```
import pandas as pd
import numpy as np



df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')


print(df_test.shape)
print(df_train.shape)
```

```
(4209, 377)
(4209, 378)
```

```
df_train.describe()
```

```
ID           y         X10     X11         X12         X13         X14         X15         X16         X17  ...
```

```
train_data=np.var(df_train, axis=0)
train_data
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3721: FutureWarning: Dropping of nuisance columns in DataFrame
  return var(axis=axis, dtype=dtype, out=out, ddof=ddof, **kwargs)
ID      5.940524e+06
y       1.607285e+02
X10     1.312780e-02
X11     0.000000e+00
X12     6.944063e-02
            ...
X380    8.012675e-03
X382    7.544954e-03
X383    1.660337e-03
X384    4.749465e-04
X385    1.423485e-03
Length: 370, dtype: float64
```

```
test_data=np.var(df_test, axis=0)
test_data
```

```
ID      5.869917e+06
X10     1.864563e-02
X11     2.375297e-04
X12     6.883438e-02
X13     5.733136e-02
            ...
X380    8.012675e-03
X382    8.713410e-03
X383    4.749465e-04
X384    7.122504e-04
X385    1.660337e-03
Length: 369, dtype: float64
```

```
train_name=[]
```

```
for i in train_data.iteritems():
    if(i[1]==0):
        train_name.append(i[0])
print(train_name)
```

```
['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297', 'X330', 'X347']
```

```
test_name=[]
for i in test_data.iteritems():
    if(i[1]==0):
        test_name.append(i[0])
print(test_name)
```

```
['X257', 'X258', 'X295', 'X296', 'X369']
```

```
df_train.drop(train_name,axis=1, inplace=True)
df_train.drop(test_name,axis=1, inplace=True)
```

```
df_test.drop(train_name,axis=1, inplace=True)
df_test.drop(test_name,axis=1, inplace=True)
```

```
print(df_train.shape)
print(df_test.shape)
```

```
(4209, 361)
(4209, 360)
```

```
#Check for null and unique values for dataset.
for i,j in zip(df_train.columns,df_train.isnull().sum()):
    if(j!=0):
        print(i)
```

```
for i,j in zip(df_test.columns,df_test.isnull().sum()):
    if(j!=0):
        print(i)
```

```
train_desc=df_train.describe(include='O')
df_train.describe(include='O')
```

	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

```
train_desc.columns
```

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

```
test_desc=df_test.describe(include='O')
df_train.describe(include='O')
```

	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

```
test_desc.columns
```

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

```
for i in ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']:
    print('df_train')
    print(i, df_train[i].unique())
    print('df_test')
    print(i, df_test[i].unique())
```

```
df_train
X0 ['k' 'az' 't' 'al' 'o' 'w' 'j' 'h' 's' 'n' 'ay' 'f' 'x' 'y' 'aj' 'ak' 'am'
 'z' 'q' 'at' 'ap' 'v' 'af' 'a' 'e' 'ai' 'd' 'aq' 'c' 'aa' 'ba' 'as' 'i'
 'r' 'b' 'ax' 'bc' 'u' 'ad' 'au' 'm' 'l' 'aw' 'ao' 'ac' 'g' 'ab']
df_test
X0 ['az' 't' 'w' 'y' 'x' 'f' 'ap' 'o' 'ay' 'al' 'h' 'z' 'aj' 'd' 'v' 'ak'
 'ba' 'n' 'j' 's' 'af' 'ax' 'at' 'aq' 'av' 'm' 'k' 'a' 'e' 'ai' 'i' 'ag'
 'b' 'am' 'aw' 'as' 'r' 'ao' 'u' 'l' 'c' 'ad' 'au' 'bc' 'g' 'an' 'ae' 'p'
 'bb']
df_train
X1 ['v' 't' 'w' 'b' 'r' 'l' 's' 'aa' 'c' 'a' 'e' 'h' 'z' 'j' 'o' 'u' 'p' 'n'
 'i' 'y' 'd' 'f' 'm' 'k' 'g' 'q' 'ab']
df_test
X1 ['v' 'b' 'l' 's' 'aa' 'r' 'a' 'i' 'p' 'c' 'o' 'm' 'z' 'e' 'h' 'w' 'g' 'k'
 'y' 't' 'u' 'd' 'j' 'q' 'n' 'f' 'ab']
df_train
X2 ['at' 'av' 'n' 'e' 'as' 'aq' 'r' 'ai' 'ak' 'm' 'a' 'k' 'ae' 's' 'f' 'd'
 'ag' 'ay' 'ac' 'ap' 'g' 'i' 'aw' 'y' 'b' 'ao' 'al' 'h' 'x' 'au' 't' 'an'
 'z' 'ah' 'p' 'am' 'j' 'q' 'af' 'l' 'aa' 'c' 'o' 'ar']
df_test
X2 ['n' 'ai' 'as' 'ae' 's' 'b' 'e' 'ak' 'm' 'a' 'aq' 'ag' 'r' 'k' 'aj' 'ay'
 'ao' 'an' 'ac' 'af' 'ax' 'h' 'i' 'f' 'ap' 'p' 'au' 't' 'z' 'y' 'aw' 'd'
 'at' 'g' 'am' 'j' 'x' 'ab' 'w' 'q' 'ah' 'ad' 'al' 'av' 'u']
df_train
X3 ['a' 'e' 'c' 'f' 'd' 'b' 'g']
df_test
X3 ['f' 'a' 'c' 'e' 'd' 'g' 'b']
df_train
X4 ['d' 'b' 'c' 'a']
df_test
X4 ['d' 'b' 'a' 'c']
df_train
X5 ['u' 'y' 'x' 'h' 'g' 'f' 'j' 'i' 'd' 'c' 'af' 'ag' 'ab' 'ac' 'ad' 'ae'
```

```
 'ah' 'l' 'k' 'n' 'm' 'p' 'q' 's' 'r' 'v' 'w' 'o' 'aa']
df_test
X5 ['t' 'b' 'a' 'z' 'y' 'x' 'h' 'g' 'f' 'j' 'i' 'd' 'c' 'af' 'ag' 'ab' 'ac'
 'ad' 'ae' 'ah' 'l' 'k' 'n' 'm' 'p' 'q' 's' 'r' 'v' 'w' 'o' 'aa']
df_train
X6 ['j' 'l' 'd' 'h' 'i' 'a' 'g' 'c' 'k' 'e' 'f' 'b']
df_test
X6 ['a' 'g' 'j' 'l' 'i' 'd' 'f' 'h' 'c' 'k' 'e' 'b']
df_train
X8 ['o' 'x' 'e' 'n' 's' 'a' 'h' 'p' 'm' 'k' 'd' 'i' 'v' 'j' 'b' 'q' 'w' 'g'
 'y' 'l' 'f' 'u' 'r' 't' 'c']
df_test
X8 ['w' 'y' 'j' 'n' 'm' 's' 'a' 'v' 'r' 'o' 't' 'h' 'c' 'k' 'p' 'u' 'd' 'g'
 'b' 'q' 'e' 'l' 'f' 'i' 'x']
```

```
#Apply label encoder
cols = df_train.columns
num_cols = df_train._get_numeric_data().columns
cat_cols = list(set(cols) - set(num_cols))
cat_cols
```

```
['X5', 'X3', 'X2', 'X0', 'X4', 'X1', 'X6', 'X8']
```

```
from sklearn.preprocessing import LabelEncoder
import numpy as np


class LabelEncoderExt(object):
    def __init__(self): #differs from LabelEncoder by handling new classes and providing a value for it
        self.label_encoder = LabelEncoder()
    def fit(self, data_list): #fit the encoder for all the unique values and introduce 'unknown' value
        self.label_encoder = self.label_encoder.fit(list(data_list) + ['Unknown'])
        self.classes_ = self.label_encoder.classes_
        return self
    def transform(self, data_list): #transform the data_list to id list where the new values get assigned
        new_data_list = list(data_list)
        for unique_item in np.unique(data_list):
```

```
            if unique_item not in self.label_encoder.classes_:
                new_data_list = ['Unknown' if x==unique_item else x for x in new_data_list]

        return self.label_encoder.transform(new_data_list)

for c in cat_cols:
    label_encoder = LabelEncoderExt()
    label_encoder.fit(df_train[c])
    df_train[c] =label_encoder.transform(df_train[c])
    df_test[c]=label_encoder.transform(df_test[c])


#Performing Dimensionality Reduction using PCA
y_train = df_train['y']
del df_train['y']


del df_train['ID']


test_data_ID=df_test['ID']
del df_test['ID']


from sklearn.decomposition import PCA
pca = PCA(n_components=8)
pca.fit(df_train)
x_pca = pca.transform(df_train)
x_pca.shape
```

```
(4209, 8)
```

```
pca.explained_variance_ratio_
```

```
array([0.38335038, 0.21388171, 0.13261954, 0.1182672 , 0.0920607 ,
       0.01590615, 0.00744544, 0.00433704])
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=5)
pca.fit(df_train)
x_train_pca = pca.transform(df_train)
x_train_pca.shape
```

```
(4209, 5)
```

```
x_test_pca = pca.transform(df_test)
x_test_pca.shape
```

```
(4209, 5)
```

```
#Predicting test_dataframe values using XGBoost.
import xgboost as xg
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE

x_train_f, x_valid_f, y_train_f, y_valid_f = train_test_split(x_train_pca, y_train, test_size=0.3,random_state=123)



d_train = xg.DMatrix(x_train_f, label=y_train_f)
d_valid = xg.DMatrix(x_valid_f, label=y_valid_f)
d_test = xg.DMatrix(x_test_pca)


#parameters for XGB
param = {'objective': 'reg:linear', 'eta': 0.03, 'max_depth': 4}
xgb_r = xg.train(params=param,dtrain = d_train, num_boost_round = 10)
y_train_pred = xgb_r.predict(d_train)
y_valid_pred = xgb_r.predict(d_valid)
y_test_pred = xgb_r.predict(d_test)
```

```
[14:26:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
# Calculating R^2 score
R_square_train = r2_score(y_train_pred, y_train_f)
R_square_valid = r2_score(y_valid_pred, y_valid_f)
print('R^2 train:', R_square_train)
print('R^2 valid set:', R_square_valid)
```

```
R^2 train: -2447.975955655754
R^2 valid set: -2439.5525358364634
```

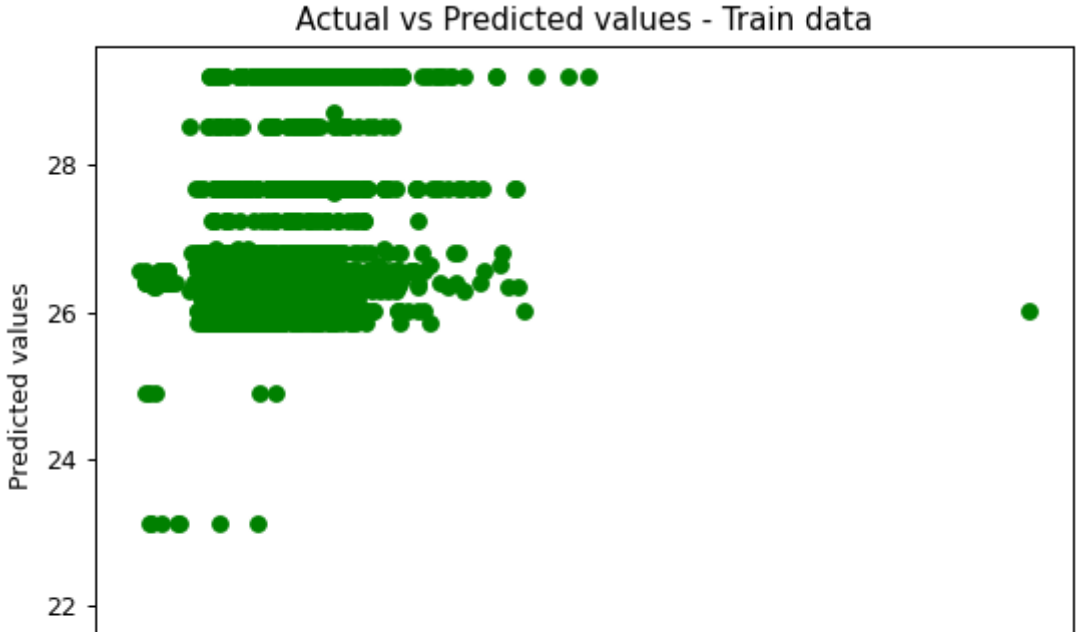
```
### ----- RMSE Computation
rmse_train = np.sqrt(MSE(y_train_f, y_train_pred))
print("RMSE of train data: % f" %(rmse_train))
```

```
RMSE of train data:  74.899321
```

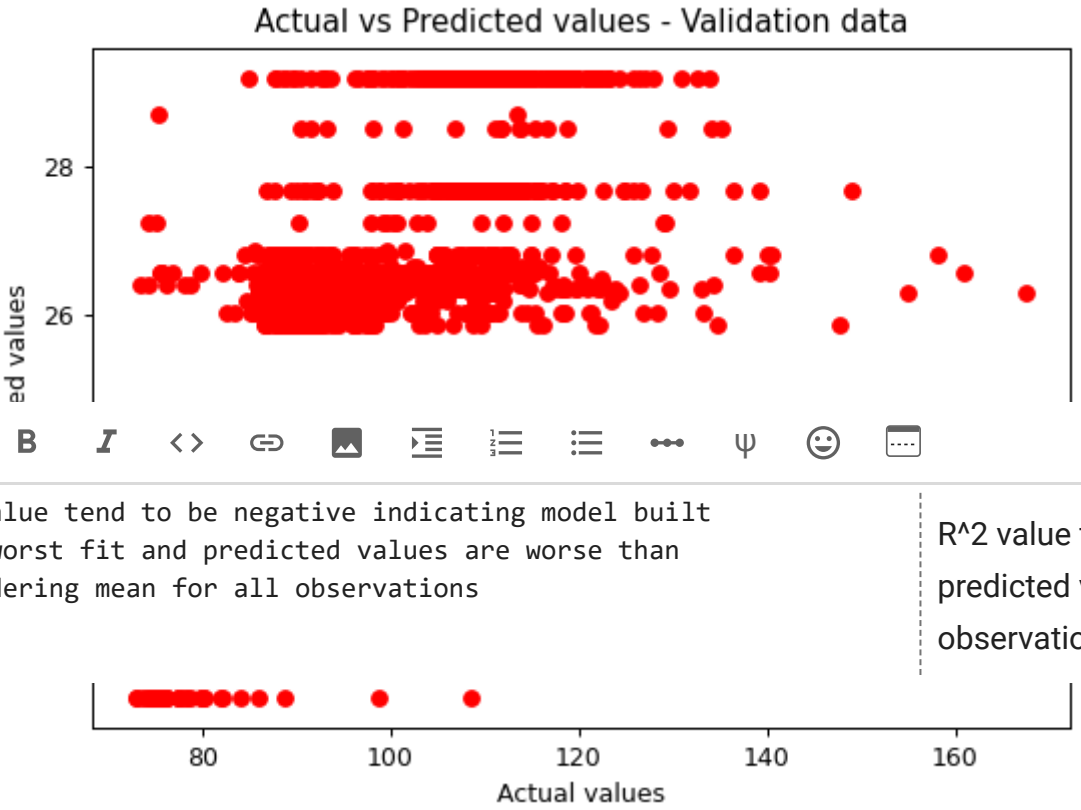
```
rmse_valid = np.sqrt(MSE(y_valid_f, y_valid_pred))
print("RMSE of valid set data : % f" %(rmse_valid))
```

```
RMSE of valid set data :  74.824379
```

```
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
# Actual vs Predicted on Train data
figure(figsize=(7, 5), dpi=90)
plt.scatter(y_train_f,y_train_pred, color = 'green')
plt.title("Actual vs Predicted values - Train data")
plt.ylabel("Predicted values")
plt.xlabel("Actual values")
plt.show()
```




```
# Actual vs Predicted on Validation data
figure(figsize=(7, 5), dpi=90)
plt.scatter(y_valid_f,y_valid_pred, color='red')
plt.title("Actual vs Predicted values - Validation data")
plt.ylabel("Predicted values")
plt.xlabel("Actual values")
plt.show()
```




```
R^2 value tend to be negative indicating model built
is a worst fit and predicted values are worse than
considering mean for all observations
```

R^2 value tend to be negative indicating model built is a worst fit and predicted values are worse than considering mean for all observations

Colab paid products - Cancel contracts here

✓ 0s completed at 8:02 PM