# PG DS – Machine Learning :

Project 1: **Mercedes-Benz Greener Manufacturing**

**Name : Baba Siddharth**

* Description:

Reduce the time a Mercedes-Benz spends on the test bench.

Problem Statement Scenario:  
Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include the passenger safety cell with a crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium carmakers. Mercedes-Benz is the leader in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams.

To ensure the safety and reliability of every unique car configuration before they hit the road, the company’s engineers have developed a robust testing system. As one of the world’s biggest manufacturers of premium cars, safety and efficiency are paramount on Mercedes-Benz’s production lines. However, optimizing the speed of their testing system for many possible feature combinations is complex and time-consuming without a powerful algorithmic approach.

You are required to reduce the time that cars spend on the test bench. Others will work with a dataset representing different permutations of features in a Mercedes-Benz car to predict the time it takes to pass testing. Optimal algorithms will contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Mercedes-Benz’s standards.

Following actions should be performed:

* If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
* Check for null and unique values for test and train sets.
* Apply label encoder.
* Perform dimensionality reduction.
* Predict your test\_df values using XGBoost.
* Code and Explanation:

#importing

import numpy as np

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.preprocessing import LabelEncoder

#Dataset

train\_set = pd.read\_csv("train.csv")

test\_set=pd.read\_csv("test.csv")

#Check for missing values

train\_set.isnull().any().any()

Graphical user interface, text, application, chat or text message

Description automatically generated

test\_set.isnull().any().any()

Graphical user interface, text, application, chat or text message

Description automatically generated

# First, lets check the original variance for all the features in the train\_set and store it to new object:

train\_var=pd.DataFrame(train\_set.var(axis=0),columns=['Variance'])

train\_var

#Second, lets define a function to remove the features of train\_set with zero variance:

def features\_zero\_var(df):

df\_original\_var=pd.DataFrame(df.var(axis=0),columns=['Variance'])

return((df\_original\_var[df\_original\_var.Variance==0]))

# Call the function to return the train\_set features having zero variance:

features\_zero\_var(train\_set)

Table, calendar

Description automatically generated

# Then we will remove the features with 0 variance from train\_set and store the data in the new object:

train\_set\_modified= train\_set.drop(columns=train\_var[train\_var.Variance==0].index)

The modified train set contains 366 features, which means the 12 features with zero variance from the original dataset is removed.

Similarly remove the zero variance features from the test dataset

# First, lets check the original variance of all the features in the test\_set and store it to new object:

test\_set\_var=pd.DataFrame(test\_set.var(axis=0),columns=['Variance'])

test\_set\_var

# Call the function to return the test\_set features having zero variance:

features\_zero\_var(test\_set)

Table

Description automatically generated

Above listed features have zero variance. However, since test\_set is not considered for training and only used for testing, we can remove the same features of train\_set in the test\_set as well. This will ensure the same size and shape of the train and test dataset.

#In test\_set, we will remove the same features of train\_set having 0 variance:

test\_set\_modified= test\_set.drop(columns=['X11', 'X93', 'X107','X233', 'X235', 'X268', 'X289', 'X290', 'X293','X297','X330','X347'])

#Null Check for the modified data:

test\_set\_modified.isnull().any().any() Graphical user interface, text, application

Description automatically generated

train\_set\_modified.isnull().any().any()

No null values found.

# Check the unique values in the train dataset:

train\_set\_modified\_UV=pd.DataFrame(train\_set\_modified.nunique(),columns=['Unique\_Values'])

train\_set\_modified\_UV

Graphical user interface, table

Description automatically generated

1.All the values in ID's are unique. In the provided dataset, the ID represents the unique car configuration. Hence, for training we will ignore this feature as it would make no difference in our prediction.

2.'y' feature is the target feature.

3.Features X0,X1,X2,X3,X4,X5,X6,X8 are the categorical features which must be converted to numerical values/one hot encoded values.

4.All the features after X8 are having binary values.

# Print the train dataset features with unique values = 2 and >2:

print('Train Features with unique values greater than 2 are as follows:\n',train\_set\_modified\_UV[train\_set\_modified\_UV.Unique\_Values>2].unstack())

print('Test Features with unique values equal to 2 are as follows:\n',train\_set\_modified\_UV[train\_set\_modified\_UV.Unique\_Values==2].unstack())

Table

Description automatically generated

Table

Description automatically generated

Out of 366 train dataset features, 10 features are having greater than 2 unique values and remaining features are having only 2 unique values (0 and 1).

#Similarly check the unique values in the test dataset:

test\_set\_modified\_UV=pd.DataFrame(test\_set\_modified.nunique(),columns=['Unique\_Values'])

test\_set\_modified\_UV

# Print the test dataset features with unique values = 2 and >2:

print('Test Features with unique values greater than 2 are as follows:\n',test\_set\_modified\_UV[test\_set\_modified\_UV.Unique\_Values>2].unstack())

print('Test Features with unique values equal to 2 are as follows:\n',test\_set\_modified\_UV[test\_set\_modified\_UV.Unique\_Values==2].unstack())

Table

Description automatically generated

Table

Description automatically generated with medium confidence

Out of 365 test dataset features, 9 features are having greater than 2 unique values and remaining features are having only 2 unique values (0 and 1).

#Apply label encoder:

Before applying label encoder, we need to separate the ID and 'y' features from the train dataset

Drop the columns 'ID' and 'y' and store the data into new object train\_X\_check and verify the shape

train\_X\_check = train\_set\_modified.drop(columns = ['ID','y'])

train\_X\_check.shape

We then define the function to apply the label encoder for the categories features of train\_X\_check.

def label\_encoder(df,x):

features\_cat=df.select\_dtypes(include='object').columns

le=LabelEncoder() # instantiate the label encoder

for i in features\_cat:

x[i]=le.fit\_transform(x[i]) # Fit,transform and replace with label encoded data for the existing data in the object datype columns of train\_X\_check

# Call the function to apply label encoder for the train\_X\_check data

label\_encoder(train\_set\_modified,train\_X\_check)

It is observed from the data in train\_X\_check.head() that label encoder is applied to categorial features of train\_X\_check data, the encoded labels are not binary (0 and 1) since the features has 4 or more different categories. Example: X5 feature has 29 unique categories. So, when label encoder is applied, the categories will be replaced with 28 unique values starting from 0 to 28. This may impact the accuracy level.

In order to fix this issue, the following procedure is followed here

-Identify the top 10 most frequent categories from each feature.

-Perform one hot encoding only for the top 10 most frequent categories.

-All Top 10 most frequent categories will be considered as '1' and all the remaining categories will be considered as '0' in each feature.

-By performing above 3 steps we ensure that only binary values (0 and 1) are present in all the features.

# Identify the top 10 most frequent categories of features X0,X1,X2,X3,X4,X5,X6,X8

#X0

train\_set\_modified.X0.value\_counts().sort\_values(ascending=False).head(10)

# create a list for the top 10 most frequent categories of feature X0:

top\_10\_X0 = [x for x in train\_set\_modified.X0.value\_counts().sort\_values(ascending=False).head(10).index]

top\_10\_X0

# Define a function to perform one hot encoding for the top 10 most frequent categories of features:

def one\_hot\_top10(df,feature,top10\_categories):

for category in top10\_categories:

df[feature+'\_'+category]=np.where(train\_set\_modified[feature]==category,1,0)

# Call the function to perform one hot encoding on feature X0

one\_hot\_top10(train\_set\_modified,'X0',top\_10\_X0)

train\_set\_modified.head(3)

# verify for one feature if it contains only binary value

train\_set\_modified.X0\_z.unique() #Yes (0,1)

It is evident from the output that 10 new features are created with only binary values, for the top 10 most frequent categories of feature X0.

Hence the total columns/features are increased from 366 to 376.

Similarly perform one hot encoding for the top 10 frequent categories of remaining categorical features.

# X1

top\_10\_X1 = [x for x in train\_set\_modified.X1.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(train\_set\_modified,'X1',top\_10\_X1)

train\_set\_modified.head(2)

# X2

top\_10\_X2 = [x for x in train\_set\_modified.X2.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(train\_set\_modified,'X2',top\_10\_X2)

train\_set\_modified.head(2)

# X3

top\_10\_X3 = [x for x in train\_set\_modified.X3.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(train\_set\_modified,'X3',top\_10\_X3)

train\_set\_modified.head(2)

#In case of X3, only 7 columns were added since there are only 7 unique categories in in this feature.

# X4

top\_10\_X4 = [x for x in train\_set\_modified.X4.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(train\_set\_modified,'X4',top\_10\_X4)

train\_set\_modified.head(2)

#In case of X4, only 4 columns were added since there are only 4 unique categories in in this feature.

# X5

top\_10\_X5 = [x for x in train\_set\_modified.X5.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(train\_set\_modified,'X5',top\_10\_X5)

train\_set\_modified.head(2)

# X6

top\_10\_X6 = [x for x in train\_set\_modified.X6.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(train\_set\_modified,'X6',top\_10\_X6)

train\_set\_modified.head(2)

# X8

top\_10\_X8 = [x for x in train\_set\_modified.X8.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(train\_set\_modified,'X8',top\_10\_X8)

train\_set\_modified.head(2)

label encoder/one hot encoding is successfully applied to the train dataset.

# Store the data set in train\_original\_modified to new object train\_original\_modified\_OHE

# drop the columns which are not required after performing one hot encoding

train\_set\_modified\_OHE = train\_set\_modified.drop(columns=['X0','X1','X2','X3','X4','X5','X6','X8'])

train\_set\_modified\_OHE.shape

The shape of the train dataset is reduced to 429 from 437 since 8 features were dropped after performing one hot encoding

Apply the label encoder/one hot encoding for the test dataset similar to the train dataset:

# create a list for the top 10 most frequent catergories of features X0, X1, X2,X3,X4,X5,X6 and X8

# Call the function to perform one hot encoding on features X0,X1,X2,X3,X4,X5,X6 and X8

# verify the train dataset after applying one hot encoding for the features X0,X1,X2,X3,X4,X5,X6 and X8

# X0

top\_10\_test\_X0 = [x for x in test\_set\_modified.X0.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(test\_set\_modified,'X0',top\_10\_test\_X0)

test\_set\_modified.head(2)

# X1

top\_10\_test\_X1 = [x for x in test\_set\_modified.X1.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(test\_set\_modified,'X1',top\_10\_test\_X1)

test\_set\_modified.head(2)

# X2

top\_10\_test\_X2 = [x for x in test\_set\_modified.X2.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(test\_set\_modified,'X2',top\_10\_test\_X2)

test\_set\_modified.head(2)

# X3

top\_10\_test\_X3 = [x for x in test\_set\_modified.X3.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(test\_set\_modified,'X3',top\_10\_test\_X3)

test\_set\_modified.head(2)

# X4

top\_10\_test\_X4 = [x for x in test\_set\_modified.X4.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(test\_set\_modified,'X4',top\_10\_test\_X4)

test\_set\_modified.head(2)

# X5

top\_10\_test\_X5 = [x for x in test\_set\_modified.X5.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(test\_set\_modified,'X5',top\_10\_test\_X5)

test\_set\_modified.head(2)

# X6

top\_10\_test\_X6 = [x for x in test\_set\_modified.X6.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(test\_set\_modified,'X6',top\_10\_test\_X6)

test\_set\_modified.head(2)

# X8

top\_10\_test\_X8 = [x for x in test\_set\_modified.X8.value\_counts().sort\_values(ascending=False).head(10).index]

one\_hot\_top10(test\_set\_modified,'X8',top\_10\_test\_X8)

test\_set\_modified.head(2)

label encoder/one hot encoding is successfully applied to the test dataset.

# Store the data set in test\_original\_modified to new object test\_original\_modified\_OHE

# drop the columns which are not required after performing one hot encoding

test\_set\_modified\_OHE = test\_set\_modified.drop(columns=['X0','X1','X2','X3','X4','X5','X6','X8'])

# Check the shape of the modified one hot encoded train dataset

test\_set\_modified\_OHE.shape

The shape of the train dataset is reduced to 428 from 436 since 8 features were dropped after performing one hot encoding.

#Perform dimensionality reduction.

# Before performing dimensional reduction (PCA), we will separate the following:

-features 'ID' and 'y' from train\_original\_modified\_OHE dataset and store it in a new object

-features 'ID' from test\_original\_modified\_OHE.shape and store it in a new object

# store the 'ID' values into the new object train\_ID and test\_ID and verify the shape

# Train dataset

train\_ID = train\_set\_modified\_OHE.ID

print('The shape of the ID feature in train dataset is:',train\_ID.shape)

# Test dataset

test\_ID = test\_set\_modified\_OHE.ID

print('\nThe shape of the ID feature in test dataset is:',test\_ID.shape)

(4209,) for both

# store the remaining values into the new object train\_X and test\_X and verify the shape

# Train dataset

train\_X = train\_set\_modified\_OHE.drop(columns = ['ID','y'])

print('The shape of the final train dataset is:',train\_X.shape)

# Test dataset

test\_X = test\_set\_modified\_OHE.drop(columns = ['ID'])

print('\nThe shape of the final test dataset is:',test\_X.shape)

(4209, 427) for both

train\_y=train\_set\_modified\_OHE.y

print('The shape of the target feature of train dataset is:',train\_y.shape)

(4209,)

Successfully separated the 'ID' and target feature 'y' from train dataset. Successfully separated the 'ID' feature from test dataset. Also, observed that shape of the train and test data are same.

# Before performing PCA, the data needs to centered and scaled

# After centering, the average value for each train and test features will be 0

# After scaling, the standard deviation for each feature will be 1

Since all the features are having 0 and 1, there is no need to standardize.

pca=PCA()

# fit the train dataset

pca\_fit\_train\_X = pca.fit(train\_X)

# transform the train dataset

pca\_fit\_transform\_train\_X = pca\_fit\_train\_X.transform(train\_X)

# transform the test dataset

pca\_transform\_test\_X = pca.transform(test\_X)

pca\_train\_X\_variation = np.round(pca\_fit\_train\_X.explained\_variance\_ratio\_.cumsum()\*100,decimals=1)

pca\_train\_X\_variation

Table

Description automatically generatedTable

Description automatically generated

By looking into the array of elements, It is observed that among 427 Principal components, ~90% of variation of the data in train\_X dataset is explained by only first 72 principal components .

lets validate by visualizing scree plot for the first 72 pricipal components

# create a PCA object again (instantiate) by considering first 72 principal components

pca\_72=PCA(n\_components=72)

# fit the train dataset

pca\_fit\_train\_X\_72 = pca\_72.fit(train\_X)

# transform the train dataset

pca\_fit\_transform\_train\_X\_72 = pca\_fit\_train\_X\_72.transform(train\_X)

# transform the test dataset

pca\_transform\_test\_X\_72 = pca\_72.transform(test\_X)

# check the variation for the first 72 principal components

pca\_train\_X\_variation\_72 = np.round(pca\_fit\_train\_X\_72.explained\_variance\_ratio\_.cumsum()\*100,decimals=1)

pca\_train\_X\_variation\_72

Text, letter

Description automatically generated

# assign labels for each PC's as PC1,2,etc., for visulaization in scree plot

labels = ['PC' + str(x) for x in range(1, len(pca\_train\_X\_variation\_72)+1)]

import matplotlib.pyplot as plt

# generate the scree plot

plt.figure(figsize=(25,5))

plt.bar(x=range(1, len(pca\_train\_X\_variation\_72)+1), height=pca\_train\_X\_variation\_72,tick\_label=labels)

plt.xticks(rotation=90, color='indigo', size=15)

plt.yticks(rotation=0, color='indigo', size=15)

plt.title('Scree Plot',color='tab:orange', fontsize=25)

plt.xlabel('Principal Components', {'color': 'tab:orange', 'fontsize':15})

plt.ylabel('Cumulative percentage of explained variance ', {'color': 'tab:orange', 'fontsize':15})

Chart, bar chart

Description automatically generated

We will draw the 2D PCA plot by considering only PC1 and PC2

PCA plot is to visualize how the data is spread across the origin with new coordinates, based on the loading scores and scaling.

# Put the new coordinates created by pca\_fit\_transform\_train\_X\_72 into matrix

# Rows are the observations (X) and columns are the Principal components (Y)

pca\_fit\_transform\_train\_X\_72\_df = pd.DataFrame(pca\_fit\_transform\_train\_X\_72,columns=labels )

# verify the first 2 rows of data with new coordinates

pca\_fit\_transform\_train\_X\_72\_df.head(2)

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

# Draw the 2D PCA plot for PC1 and PC2

# Removing the cumsum() from the earlier explained ratio calculation

pca\_train\_X\_variation\_72\_Nocumsum = np.round(pca\_fit\_train\_X\_72.explained\_variance\_ratio\_\*100,decimals=1)

plt.title('PCA Plot',color='tab:orange', fontsize=20)

plt.scatter(pca\_fit\_transform\_train\_X\_72\_df.PC1, pca\_fit\_transform\_train\_X\_72\_df.PC2)

plt.xticks(rotation=90, color='indigo', size=15)

plt.yticks(rotation=0, color='indigo', size=15)

plt.xlabel('PC1 - {0}%'.format(pca\_train\_X\_variation\_72\_Nocumsum[0]), {'color': 'tab:orange', 'fontsize':15});

plt.ylabel('PC2 - {0}%'.format(pca\_train\_X\_variation\_72\_Nocumsum[1]), {'color': 'tab:orange', 'fontsize':15});

## The principal components are zero-indexed, So, PC1=[0], PC2=[1]

Chart, scatter chart

Description automatically generated

Above PCA plot shows how the data is spread along X-axis(PC1) and Y-axis (PC2). 11.9% variance of the data is explained by PC1 and 8.2 % of data is explained by PC2 Similarly we visualize how the data is spread among other principal components as well.

# Print the loading scores

Loading scores explain the proportion of each observation with respect to each principal components

# Lets check only for the PC1

loading\_scores = pd.Series(pca\_72.components\_[0])

# Sort the loading scores based on absolute value

sorted\_loading\_scores=loading\_scores.abs().sort\_values(ascending=False)

# Display only the top 10 loading scores

sorted\_loading\_scores[0:10]

Table

Description automatically generated

# Print the minimum and maximum loading scores of PC1

print(sorted\_loading\_scores.min())

print(sorted\_loading\_scores.max())

Text

Description automatically generated

It can be concluded from the above loading scores that, almost all the observations of the train datasets play a role in separating the Principal components PC1 Example: The 175th observation has a 1 unit long vector consisting of the following: 0.191403 PC1 +.......+ Xn PCn 0.191403 is the proportion of 175th observation for PC1 This unit vector is called singular vector or eigen vector for PC1 similarly the loading scores will be calculated for PC2 as pca72.components[1], etc

# From PCA, the final train and test datasets are as follows

#Train data

pca\_fit\_transform\_train\_X\_72.shape

#Test data

pca\_transform\_test\_X\_72.shape #both (4209, 72)

#train label

train\_y.shape #(4209,)

# train ID

train\_ID.shape

test\_ID.shape #(4209,)

#Predict test\_data using XGBoost.

# Before predicting the test values, lets check the target variable train\_y for any outliers

plt.boxplot(train\_y);

Chart, box and whisker chart

Description automatically generated

# Print the 50th percentile value which is the median

print(train\_y.quantile(0.50)) #99.15

# Print the 95th percentile value

print(train\_y.quantile(0.95)) #120.80600000000001

# Replace the outlier with median values

train\_y = np.where(train\_y > 120.80600000000001, 99.15, train\_y)

# Verify again with box\_plot after replacing the outliers with median values

plt.boxplot(train\_y);

Chart, box and whisker chart

Description automatically generated

# Check the shape again

train\_y.shape #(4209,)

It is evident from the box plot that outliers are replaced with median values in the target variable train\_y Also, there is no change in the shape of the target variable. Hence its good to go with further steps

# import the required libraries

import xgboost as xgb

from sklearn.model\_selection import cross\_val\_score,cross\_val\_predict

# Since objective is to predict continuous variable we use XGBregressor

from xgboost import XGBRegressor

# Evaluation metrics for regression

### Mean Absolute Error, Mean Squared Error and R2

# We will use R2 in this case

# R2 is also known as Coefficient of Determination

# It gives the percentage variation in 'y' (test time) explained by 'X'variables

# or,it gives the percentage of data points that fall within the regression line

# R2= (1-SSR/SST)

# SSR- Sum of square residual; SST- Sum of squares total

# R2 value should be between 0 to 1

# -R2 valve indicates the worst model

# print the XGBoost parameters

print(XGBRegressor())

Text

Description automatically generated

xgb\_reg = xgb.XGBRegressor()

#To find best XGBoost Parameters

params={ 'learning\_rate' : [0.01,0.05,0.1,1] ,

'max\_depth' : [2,3,5,10],

'min\_child\_weight': [ 0, 1, 3],

'n\_estimators' : [100,150,200,500],

'gamma' : [1e-2,1e-3,0,0.1,0.01,0.5,1],

'colsample\_bytree': [0.1,0.5,0.7,1],

'subsample' : [0.2,0.3,0.5,1],

'reg\_lambda' : [0,1,10],

'reg\_alpha' : [1e-5,1e-3,1e-1,1,1e1]

# Optimize the Hyperparameter using RandomizedSearchCV

from sklearn.model\_selection import RandomizedSearchCV

# Using Random search of parameters with 10 fold cross validation

# Improve the predictions using cross validation to optimize the parameters

Random\_Search=RandomizedSearchCV (xgb\_reg,params,cv=10, scoring='r2', return\_train\_score=True, n\_jobs=-1,verbose=1)

# cv=10 - Number of folds in a `(Stratified)KFold`

Random\_Search.fit(pca\_fit\_transform\_train\_X\_72,train\_y)

Text

Description automatically generated with low confidence

Text

Description automatically generated

#Print the best parameters

Random\_Search.best\_params\_

Text

Description automatically generated

# Instantiate the XGBoost classifier with the best estimators and parameters

xgb\_reg=XGBRegressor(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,

colsample\_bynode=1, colsample\_bytree=0.5, gamma=0.01, gpu\_id=-1,

importance\_type='gain', interaction\_constraints='',

learning\_rate=0.1, max\_delta\_step=0, max\_depth=2,

min\_child\_weight=3, missing=None, monotone\_constraints='()',

n\_estimators=500, n\_jobs=2, num\_parallel\_tree=1, random\_state=0,

reg\_alpha=0.1, reg\_lambda=10, scale\_pos\_weight=1, subsample=0.5,

tree\_method='exact', validate\_parameters=1, verbosity=None)

# Check the r2 score of the model using Number of folds in a `(Stratified)KFold` cv=10

r2\_Score = cross\_val\_score(xgb\_reg,pca\_fit\_transform\_train\_X\_72,train\_y,scoring='r2',cv=10)

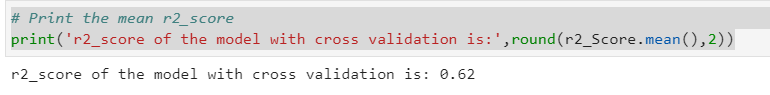
r2\_Score

A picture containing text, tool

Description automatically generated

# Print the mean r2\_score

print('r2\_score of the model with cross validation is:',round(r2\_Score.mean(),2))



Since r2\_score with cross validation is: 0.62 or 62 % which is between 50 to 100%. Hence, its good to proceed with the prediction of time the car takes to pass testing using test data. This means the model explains 62% variability of the target variable (y) around its mean.

# Fit the training data

xgb\_reg.fit(pca\_fit\_transform\_train\_X\_72,train\_y)

Text

Description automatically generated

# predict the time taken by car to pass testing using test dataset

X\_test\_pred = xgb\_reg.predict(pca\_transform\_test\_X\_72)

X\_test\_pred



# print the predicted value (time) in the form of table

df\_test\_pred = pd.DataFrame({'ID': test\_ID, 'y': X\_test\_pred})

# Print the first 10 predicted values

df\_test\_pred.head(10)

Table

Description automatically generated with medium confidence

Conclusion:

For the given dataset, XGBoost Regressor algorithm with cross validation results in the R2 score of 0.62.