MERCEDES-BENZ GREENER MANUFACTURING

Introduction

Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include, for example, the passenger safety cell with 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 car makers. Daimler’s Mercedes-Benz cars are leaders 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 each and every unique car configuration before they hit the road, Daimler’s engineers have developed a robust testing system. But, optimizing the speed of their testing system for so many possible feature combinations is complex and time-consuming without a powerful algorithmic approach. As one of the world’s biggest manufacturers of premium cars, safety and efficiency are paramount on Daimler’s production lines.

In this competition, Daimler is challenging Kagglers to tackle the curse of dimensionality and reduce the time that cars spend on the test bench. Competitors will work with a dataset representing different permutations of Mercedes-Benz car features to predict the time it takes to pass testing. Winning algorithms will contribute to speedier testing, resulting in lower carbon dioxide emissions without reducing Daimler’s standards.

Project overview

1. In an automobile industry,there is a testing phase for every vehicle that comes out from production manufacturing. Safety and reliable testing is a crucial part in the automobile manufacturing process.
2. The Mercedes -Benz automobile industry every day manufactures a huge rate of producing vehicles and finally sends to the testing phase at the final stage in production. Every possible vehicle combination must undergo a test bench to ensure the vehicle is robust enough to keep passengers safe and withstand in daily use. More tests result in more time spent on the test stand, increasing costs for Mercedes and generating carbon dioxide, a polluting greenhouse gas.

Objective of the project

The main objective of this project is to optimize/reduce the testing time process of every production vehicle that comes under the test bench. By this optimization it certainly decreases the Carbon dioxide emission associated with the testing procedure.

Machine Learning Problem Formulation:

The above problem can be solve using Classical Machine Learning techniques to predict the time(target variable) that car will spend on the test bench based on the vehicle configuration.(Independent Features)

1. The type of problem is a supervised learning problem and the model can learn from the labelled data.
2. It is an example of a Machine Learning Regression task thus it should predict the result in a continuous target variable.(time duration of test bench).

Performance Metrics:

Since it is a regression problem a key performance metric are used as

1. R^2(coefficient of determination).

Data source:

Mercedes Benz posted the above problem as csv data format in kaggle platform.

Link :https://www.kaggle.com/c/mercedes-benz-greener-manufacturing

Data Overview:

The data provided in two csv file formats such as train.csv and test.csv.

1. There are about 386 total features and represented as x0,x1,x2….x386 and each features are anonymized meaning as not physically represented like configuration options such as

suspension setting, adaptive cruise control, all-wheel drive,together define a car model.

2. The total features is consist of two format a. Categorical variable (8 categorical features) b. Numerical binary variable.
3. The product ID columns (unique configuration of vehicle ID).

Loading the libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.random_projection import GaussianRandomProjection
from sklearn.preprocessing import OneHotEncoder
import pickle
from itertools import combinations
from sklearn.linear_model import LinearRegression
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectKBest, f_regression
from scipy.stats import pearsonr
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import prettytable
from xgboost import XGBRegressor
from mlxtend.regressor import StackingCVRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import r2_score
from xgboost import plot_importance
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import RepeatedKFold,KFold
from sklearn.metrics import r2_score
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
```

Loading the Data set

```
from google.colab import drive

drive.mount("/content/drive",force_remount=True)
```

```
train_data=pd.read_csv("/content/drive/My Drive/mercedes _traindata.csv")
train_data.head(5)
```

```
train_data.shape
```

Pre-processing of Train dataset

Description of the dataset

```
train_data.describe()
```

```
len(test_data.loc[0].values)
```

Checking wheather the dataset contain the Null values

```
print("Null value in the dataset:",len(train_data[train_data.isnull().any(axis=1)]))
```

1. In the above dataset there is no NULL values in all the features.
2. Thus every columns filled by the number and categorical values.

Checking whether the Duplicate ID is present in the Dataset

```
print("No. of Duplicates ID in the dataset:",(len(train_data["ID"])-len(train_data["ID"].drop
```

1. The above dataset has no duplicate ID
2. so there is unique categories of vechicle id is provided in the dataset

Finding how many numerical binary features and categorical features

```
data=train_data.drop(["ID","y"],axis=1)
```

```
#https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.select_dtypes.ht
num_binary_feat=data.select_dtypes(include="int").columns
cat_feat =data.select_dtypes(include="object").columns
print('Numerical feature size: {}'.format(len(num_binary_feat)))
print('Categorical feature size: {} '.format(len(cat_feat)))
```

Observations:

From here we can conclude that in the dataset we have 368 numerical binary features and 8 categorical features

Performing Exploratory Data Analysis on train data

Exploratory Data Analysis is statical analysis perform in the data which can know about the characteristic of data,distributions,visualization of data in many kind of plots. This gives the data insights before applying into the modelling.

Here we are exploring the Data and visualize into three parts.

part 1

1. Categorical variable (Analysis)
2. cateorical variable (visualization)

part2

3. Numerical binary varible (Analysis)
4. Numerical binary variable(visualization)

part 3

5. Target y output (Analysis)
6. Target y output(Visualization)

Part 1 (Analyzing and visualization of categorical variable)

Checking the unique categories items and top count of category values present in the Categorical variables

```
# category 1
print("Number of unique categorical values in X0:",len(np.unique(train_data["X0"])))
print("The unique set of categories are:",np.unique(train_data["X0"]))
print("The Top five counts of the categorical values:",train_data.X0.value_counts().head())
print("-"*150)
# category 2
print("Number of unique categorical values in X1:",len(np.unique(train_data["X1"])))
print("The unique set of categories are:",np.unique(train_data["X1"]))
print("The Top five counts of the categorical values:",train_data.X1.value_counts().head())
print("-"*150)
# category 3
print("Number of unique categorical values in X2:",len(np.unique(train_data["X2"])))
print("The unique set of categories are:",np.unique(train_data["X2"]))
print("The Top five counts of the categorical values:",train_data.X2.value_counts().head())
print("-"*150)
# category 4
print("Number of unique categorical values in X3:",len(np.unique(train_data["X3"])))
print("The unique set of categories are:",np.unique(train_data["X3"]))
print("The Top five counts of the categorical values:",train_data.X3.value_counts().head())
print("-"*150)
# category 5
print("Number of unique categorical values in X4:",len(np.unique(train_data["X4"])))
print("The unique set of categories are:",np.unique(train_data["X4"]))
print("The Top five counts of the categorical values:",train_data.X4.value_counts().head())
print("-"*150)
# category 6
print("Number of unique categorical values in X5:",len(np.unique(train_data["X5"])))
print("The unique set of categories are:",np.unique(train_data["X5"]))
print("The Top five counts of the categorical values:",train_data.X5.value_counts().head())
print("-"*150)
# category 7
print("Number of unique categorical values in X6:",len(np.unique(train_data["X6"])))
print("The unique set of categories are:",np.unique(train_data["X6"]))
print("The Top five counts of the categorical values:",train_data.X6.value_counts().head())
print("-"*150)
# category 8
print("Number of unique categorical values in X8:",len(np.unique(train_data["X8"])))
print("The unique set of categories are:",np.unique(train_data["X8"]))
print("The Top five counts of the categorical values:",train_data.X8.value_counts().head())
print("-"*150)
```

Bar plot visualization of the categorical variable

```
plt.figure(figsize=(20,25))
plt.subplot(4,2,1)
train_data.X0.value_counts().plot(kind="bar",title="X0")
plt.subplot(4,2,2)
train_data.X1.value_counts().plot(kind="bar",title="X1")
plt.subplot(4,2,3)
train_data.X2.value_counts().plot(kind="bar",title="X2")
```

```
plt.subplot(4,2,4)
train_data.X3.value_counts().plot(kind="bar",title="X3")
plt.subplot(4,2,5)
train_data.X4.value_counts().plot(kind="bar",title="X4")
plt.subplot(4,2,6)
train_data.X5.value_counts().plot(kind="bar",title="X5")
plt.subplot(4,2,7)
train_data.X6.value_counts().plot(kind="bar",title="X6")
plt.subplot(4,2,8)
train_data.X8.value_counts().plot(kind="bar",title="X8")
plt.savefig('b.png')
```

observation

1. The above plots shows the count of each category present in the each categorical features.
2. It is same like histogram plot we can about frequency of each category occurs in categorical features.
3. This plot gives more intutions that which category present high in range and which category present low in range.

Ploting scatterplot, Box plot and Violin plot on various categorical features

For X0 categorical feature

```
plt.figure(figsize=(10,5))
sns.scatterplot(x="X0", y="y",data=train_data);
plt.xlabel("X0", fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+"X0", fontsize=15)
plt.show()
plt.savefig('c.jpg')
```

Observations:

1. The above plot represent the categorical scatter plot which is able to represent like how the categorical values distributed with corresponding output variable.
2. The X-axis represent the categories and y axis represent output variable.
3. This plot helps to visualize the outlier of output variable with corresponding category value present in the categorical features.
4. In the above plot "y" category of which one datapoint present faraway from normally distributed of that category and we can consider this as an outlier of that category.

For X1 categorical features

```
plt.figure(figsize=(10,5))
sns.scatterplot(x='X1',y='y', data=train_data)
plt.xlabel("X1", fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+"X1", fontsize=15)
plt.show()
```

Observations:

1. The above plot represent the categorical scatter plot which is able to represent like how the categorical values distributed with corresponding output variable.
2. The X-axis represent the categories and y axis represent output variable.
3. This plot helps to visualize the outlier of output variable with corresponding category value present in the categorical features.
4. In the above plot "r" category of which one datapoint present faraway from normally distributed of that category and we can consider this as an outlier of that category.

for x2 categorical feature

```
plt.figure(figsize=(10,5))
sns.scatterplot(x="X2", y="y",data=train_data);
plt.xlabel("X2", fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+"X2", fontsize=15)
plt.show()
```

Observations:

1. The above plot represent the categorical scatter plot which is able to represent like how the categorical values distributed with corresponding output variable.
2. The X-axis represent the categories and y axis represent output variable.
3. This plot helps to visualize the outlier of output variable with corresponding category value present in the categorical features.
4. In the above plot "ai" category of which one datapoint present faraway from normally distributed of that category and we can consider this as an outlier of that category.

Box plot

A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a verticle line is also there which goes through the box at the median. Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.

for x3 categorical feature

```
plt.figure(figsize=(10,5))
sns.boxplot(x="X3", y="y",data=train_data);
plt.xlabel("X3", fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+"X3", fontsize=15)
plt.show()
```

Observation:

1. The above plot represent each categories are distributed uniformly in overlap with each other.
2. Most of the categories lies in the range of 85 to 120 values of output variable.
3. But in category "f" which one data point lies faraway from normally distributed.Here we can conclude that it is an outlier

for x4 categorical feature

```
plt.figure(figsize=(10,5))
sns.boxplot(x="X4", y="y",data=train_data);
plt.xlabel("X4", fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+"X4", fontsize=15)
plt.show()
```

Observation:

1. The above plot shows the "d" category contain most in the features.which it mostly distributed in the range of 90 to 110 range of values.
2. The category "b" and "c" are present with few in numbers and mostly at the point of 120 and 130 output value respectively.

for x5 categorical variable

```
plt.figure(figsize=(10,5))
sns.boxplot(x="X5", y="y",data=train_data);
plt.xlabel("X5", fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+"X5", fontsize=15)
plt.show()
```

Observation:

1. The above plot represent most of the categorical values 85 to 120 output values.
2. This features shows that most of the features occurs and uniformly distributed and we can see some of the features which present in few in number.

Violin plot

Violin plots are similar to box plots, except that they also show the probability density of the data at different values, usually smoothed by a kernel density estimator. Typically a violin plot will include all the data that is in a box plot: a marker for the median of the data; a box or marker indicating the interquartile range; and possibly all sample points, if the number of samples is not too high. A violin plot is more informative than a plain box plot. While a box plot only shows summary statistics such as mean/median and interquartile ranges, the violin plot shows the full distribution of the data. The difference is particularly useful when the data distribution is multimodal (more than one peak). In

for x6 categorical variable

```
plt.figure(figsize=(15,5))
sns.violinplot(x="X6", y="y",data=train_data);
plt.xlabel("X6", fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+"X6", fontsize=15)
plt.show()
```

Observation:

1. The plot which represent the most of the categories PDF curve lies under the range of 75 to 125 output y variables.
2. But we can see that category "i" which is highly skewed and this shows that this category have the outlier with respect to output variable y.

for x8 categorical features

```
plt.figure(figsize=(15,5))
sns.violinplot(x="X8", y="y",data=train_data);
plt.xlabel("X8", fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable with "+"X8", fontsize=15)
plt.show()
```

Observation:

1. The above plot show all the categorical values are present with uniformly and almost PDF curve lies in the range of 75 to 125 values of output y variables.
2. But we can see that category "t" which is highly skewed and this shows that this category have the outlier with respect to output variable y.

Part 2 (Analyzing and visualization of Numerical Binary data)

Analysis on Numerical binary features

```
constant_binary_number=0
variable_binary_number=0
constant_binary_features=[]
variable_binary_features=[]

for col in train_data.columns:
  if col not in ["ID","y","X0","X1","X2","X3","X4","X5","X6","X8"]:
      if int(1) in train_data[col].values:
        variable_binary_number+=1
        variable_binary_features.append(col)
      elif int(0) in train_data[col].values:
        constant_binary_number+=1
        constant_binary_features.append(col)
constant_binary_number
```

```
print("The number of columns which contain single binary value as zero only:",constant_binary
print("The features columns is:",np.array(constant_binary_features))
print("_"*150)
print("The number of columns which contain both the binary values:",variable_binary_number)
print("The features columns is:",np.array(variable_binary_features))
```

Visualizing the binary features using bar garph

```
one_count=[]
zero_count=[]
width=0.5
data=train_data.drop(["ID","y","X0","X1","X2","X3","X4","X5","X6","X8"],axis=1)
```

```
num=len(data.columns)
index=np.arange(num)
for col in data.columns:
      one_count.append((data[col]==1).sum())
      zero_count.append((data[col]==0).sum())
plt.figure(figsize=(5,70))
y1=plt.barh(index,zero_count,width,color="red")
y2=plt.barh(index,one_count,width,left=zero_count,color="blue")
plt.yticks(index,data.columns)
plt.legend((y1[0],y2[0]), ('Zero count', 'One Count'))
plt.grid()
plt.savefig('a.jpg')
```

Observation:

1. The above plot is the bar plot which represent frequency count of numerical binary values in the dataset of corresponding features.
2. By this plot we can able to find the majority and minority of values of occurance of numerical binary values in corresponding numerical binary features.
3. The X-axis represent the index of the datapoint and y-axis represent the all numerical binary features contain the dataset.

Part 3 (Analyzing and visualizing the output variable)

Analyzing the output variable

```
train_data.y.describe()
```

Visualize into various plot of Target output variable

Scatter plot

```
plt.figure(figsize=(15,10))
index=len(train_data)
plt.scatter(x=range(index), y=np.sort(train_data["y"].values))
plt.xlabel("index", fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title("Distribution of y variable", fontsize=15)
plt.show()
```

observations:

1. The above plot is the Scatter plot which x-axis contain the index of the datapoint and y-axis contain the output variable.

2. From this plot we can colnclude that most of the value of output varaibele lies between the range of 80 to 120 values.
3. we can also see some of the outlier points faraway from the group of values.

Histogram PDF function

```
nbin=int(np.sqrt(4209))# sqrt of number of datapoints
plt.figure(figsize=(15,10))
sns.distplot(train_data['y'], hist=True, kde=True,
             bins=nbin, color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
plt.ylabel("PDF")
plt.title("PDF distribution")
plt.show()
```

Observations:

1. The above plot is PDF distributions of output variable and the plot shows there is rise of two peak cuvrve. This can be conclude that the distribution of the output variable are following the Gaussian distribution.
2. The plot which provides both the histogram plot and pdf curve and which give more intuition on distributions
3. The plot show most of the values in 80 and 120 values and more values present with this value region.
4. The pdf curve is highly skewed at right this shows that few values lies more than the value of 250.

```
#https://stackoverflow.com/questions/39297523/plot-cdf-cumulative-histogram-using-seaborn-pyt
x = train_data["y"].values
kwargs = {'cumulative': True}
plt.xlabel("y")
plt.ylabel("CDF")
plt.title("CDF Distributions")
sns.distplot(x, hist_kws=kwargs, kde_kws=kwargs)
```

Observations:

1. The plot represent the Cdf of the output y variables.
2. This conclude that 95% of the output variable lies less than 120

3. And very few variables are lies beyond the range of 150.

Let do some small analysis from lookin on above visualizations:

```
low_range=0
inter_range=0
high_range=0
for i in train_data["y"].values:
  if i<80:
    low_range+=1
  elif i>80 and i<150:
    inter_range+=1
  else:
    high_range+=1
```

```
print("The number of datapoint with ouput values less than 80 value threshold:",low_range)
print("The number of datapoint with with inter range value of 80 to 150 value threshold: ",in
print("The number of datapoint with ouput values more than 150 value threshold:",high_range)
```

Conclusion of EDA:

1.From the above analysis of the data we can conlcude that the dataset has an outlier and some of the columns which are same values of in entire columns in numerical binary features.

2.By the above conclusion we are removing those outlier points and repeated values in the numerical binary features.

3.By the removal the of model will perform well by eliminating the features collinearity and outlier.

Removal of the same values contain in the numerical binary featues

```
removal_col=[]
for col in train_data.columns:
  if col not in ["ID","X0","X1","X2","X3","X4","X5","X6","X8"]:
    if train_data[col].var()==0:
      removal_col.append(col)

removal_col
```

```
train_data.drop(removal_col, axis=1,inplace=True)
train_data.shape
```

Removal of outlier points

```
#https://stackoverflow.com/questions/22354094/pythonic-way-of-detecting-outliers-in-one-dimen
def outlier(points, thresh = 3.5):
    points = points[:,None]
    median = np.median(points, axis=0)
    diff = np.sum((points - median)**2, axis=-1)
    diff = np.sqrt(diff)
    med_abs_deviation = np.median(diff)
    modified_z_score = 0.6745 * diff / med_abs_deviation # we are doing the modified z score
    return points[modified_z_score > thresh]
```

```
outliers= outlier(train_data["y"])#printing the outlier points from the output variable
outliers=outliers.flatten()
outliers
```

```
for i in outliers:
  train_data.drop(train_data[train_data["y"]==i].index,inplace=True)# Dropping the outliers
```

```
print("The shape of the data with removed outlier points:",train_data.shape)
```

Loading the Test data

```
test_data=pd.read_csv("/content/drive/My Drive/mercedes_testdata.csv")
test_data.head()
```

```
print("The shape of the Test Dataset:",test_data.shape)
```

```
test_data.drop(removal_col, axis=1,inplace=True)
test_data.shape
```

```
y_train=train_data["y"]
train_data.drop("y",axis=1,inplace=True)
```

```
print("The Final train Dataset shape is:",train_data.shape)
print("The Final test Dataset shape is:",test_data.shape)
```

Feature Engineering

Technique 1: Dimensionality reductions techniques using PCA

convertions of categorical variable into numerical features

1. Label encoding

```
label=LabelEncoder()
label_enc=label.fit(train_data["X0"].values)
test_data["X0"]=test_data["X0"].map(lambda x:"Nil" if x not in label.classes_ else x)
label.classes_ = np.append(label.classes_, 'Nil')
train_data["X0"] = label.transform(train_data["X0"])
test_data["X0"] = label.transform(test_data["X0"])

#---------------------------------------------------------------------------------------------
label=LabelEncoder()
label_enc=label.fit(train_data["X1"].values)
test_data.loc[:,("X1")]=test_data["X1"].map(lambda x:"Nil" if x not in label.classes_ else x)
label.classes_ = np.append(label.classes_, 'Nil')
train_data.loc[:,("X1")] = label.transform(train_data["X1"])
test_data.loc[:,("X1")] = label.transform(test_data["X1"])

#---------------------------------------------------------------------------------------------
label=LabelEncoder()
label_enc=label.fit(train_data["X2"].values)
test_data.loc[:,("X2")]=test_data["X2"].map(lambda x:"Nil" if x not in label.classes_ else x)
label.classes_ = np.append(label.classes_, 'Nil')
train_data.loc[:,("X2")] = label.transform(train_data["X2"])
test_data.loc[:,("X2")] = label.transform(test_data["X2"])

#---------------------------------------------------------------------------------------------
label=LabelEncoder()
label_enc=label.fit(train_data["X3"].values)
test_data.loc[:,("X3")]=test_data["X3"].map(lambda x:"Nil" if x not in label.classes_ else x)
label.classes_ = np.append(label.classes_, 'Nil')
train_data.loc[:,("X3")] = label.transform(train_data["X3"])
test_data.loc[:,("X3")] = label.transform(test_data["X3"])

#---------------------------------------------------------------------------------------------
label=LabelEncoder()
label_enc=label.fit(train_data["X4"].values)
test_data.loc[:,("X4")]=test_data["X4"].map(lambda x:"Nil" if x not in label.classes_ else x)
label.classes_ = np.append(label.classes_, 'Nil')
train_data.loc[:,("X4")] = label.transform(train_data["X4"])
test_data.loc[:,("X4")] = label.transform(test_data["X4"])

#---------------------------------------------------------------------------------------------
label=LabelEncoder()
label_enc=label.fit(train_data["X5"].values)
test_data.loc[:,("X5")]=test_data["X5"].map(lambda x:"Nil" if x not in label.classes_ else x)
label.classes_ = np.append(label.classes_, 'Nil')
train_data.loc[:,("X5")] = label.transform(train_data["X5"])
test_data.loc[:,("X5")] = label.transform(test_data["X5"])

#---------------------------------------------------------------------------------------------
label=LabelEncoder()
label_enc=label.fit(train_data["X6"].values)
```

```
test_data.loc[:,("X6")]=test_data["X6"].map(lambda x:"Nil" if x not in label.classes_ else x)
label.classes_ = np.append(label.classes_, 'Nil')
train_data.loc[:,("X6")] = label.transform(train_data["X6"])
test_data.loc[:,("X6")] = label.transform(test_data["X6"])

#-------------------------------------------------------------------------------------------
label=LabelEncoder()
label_enc=label.fit(train_data["X8"].values)
test_data.loc[:,("X8")]=test_data["X8"].map(lambda x:"Nil" if x not in label.classes_ else x)
label.classes_ = np.append(label.classes_, 'Nil')
train_data.loc[:,("X8")] = label.transform(train_data["X8"])
test_data.loc[:,("X8")] = label.transform(test_data["X8"])

#-------------------------------------------------------------------------------------------


train_data.to_csv("label_train.csv",index=False)
test_data.to_csv("label_test.csv",index=False)


X_train_label=pd.read_csv("/content/label_train.csv")
X_test_label=pd.read_csv("/content/label_test.csv")


print("The train label data:",X_train_label.shape)
print("The test label data:",X_test_label.shape)
```

2. frequency encoding

```
train_feq=train_data.groupby("X0").size()/len(train_data)
train_data["X0"]=[train_feq.loc[i] for i in train_data["X0"].values]
test_data["X0"]=[train_feq.loc[i] if i in train_feq.index else 0 for i in test_data["X0"].val
#-------------------------------------------------------------------------------------------

train_feq=train_data.groupby("X1").size()/len(train_data)
train_data["X1"]=[train_feq.loc[i] for i in train_data["X1"].values]
test_data["X1"]=[train_feq.loc[i] if i in train_feq.index else 0 for i in test_data["X1"].val

#-------------------------------------------------------------------------------------------

train_feq=train_data.groupby("X2").size()/len(train_data)
train_data["X2"]=[train_feq.loc[i] for i in train_data["X2"].values]
test_data["X2"]=[train_feq.loc[i] if i in train_feq.index else 0 for i in test_data["X2"].val

#-------------------------------------------------------------------------------------------

train_feq=train_data.groupby("X3").size()/len(train_data)
```

```
train_data["X3"]=[train_feq.loc[i] for i in train_data["X3"].values]
test_data["X3"]=[train_feq.loc[i] if i in train_feq.index else 0 for i in test_data["X3"].val

#---------------------------------------------------------------------------------------------

train_feq=train_data.groupby("X4").size()/len(train_data)
train_data["X4"]=[train_feq.loc[i] for i in train_data["X4"].values]
test_data["X4"]=[train_feq.loc[i] if i in train_feq.index else 0 for i in test_data["X4"].val

#---------------------------------------------------------------------------------------------

train_feq=train_data.groupby("X5").size()/len(train_data)
train_data["X5"]=[train_feq.loc[i] for i in train_data["X5"].values]
test_data["X5"]=[train_feq.loc[i] if i in train_feq.index else 0 for i in test_data["X5"].val

#---------------------------------------------------------------------------------------------

train_feq=train_data.groupby("X6").size()/len(train_data)
train_data["X6"]=[train_feq.loc[i] for i in train_data["X6"].values]
test_data["X6"]=[train_feq.loc[i] if i in train_feq.index else 0 for i in test_data["X6"].val

#---------------------------------------------------------------------------------------------

train_feq=train_data.groupby("X8").size()/len(train_data)
train_data["X8"]=[train_feq.loc[i] for i in train_data["X8"].values]
test_data["X8"]=[train_feq.loc[i] if i in train_feq.index else 0 for i in test_data["X8"].val

#---------------------------------------------------------------------------------------------


train_data.to_csv("freq_train.csv",index=False)
test_data.to_csv("freq_test.csv",index=False)


X_train_feq=pd.read_csv("/content/freq_train.csv")
X_test_feq=pd.read_csv("/content/freq_test.csv")


print("The train frequency data:",X_train_feq.shape)
print("The test frequency data:",X_test_feq.shape)
```

3. Mean encoding

```
y_mean=train_data.y.mean()
train_mean=train_data.groupby("X0")["y"].mean()
train_data["X0"]=[train_mean.loc[i] for i in train_data["X0"].values]
test_data["X0"]=[train_mean.loc[i] if i in train_mean.index else y_mean for i in test_data["X
```

```
#---------------------------------------------------------------------------------------------

train_mean=train_data.groupby("X1")["y"].mean()
train_data["X1"]=[train_mean.loc[i] for i in train_data["X1"].values]
test_data["X1"]=[train_mean.loc[i] if i in train_mean.index else y_mean for i in test_data["X

#---------------------------------------------------------------------------------------------

train_mean=train_data.groupby("X2")["y"].mean()
train_data["X2"]=[train_mean.loc[i] for i in train_data["X2"].values]
test_data["X2"]=[train_mean.loc[i] if i in train_mean.index else y_mean for i in test_data["X

#---------------------------------------------------------------------------------------------

train_mean=train_data.groupby("X3")["y"].mean()
train_data["X3"]=[train_mean.loc[i] for i in train_data["X3"].values]
test_data["X3"]=[train_mean.loc[i] if i in train_mean.index else y_mean for i in test_data["X

#---------------------------------------------------------------------------------------------

train_mean=train_data.groupby("X4")["y"].mean()
train_data["X4"]=[train_mean.loc[i] for i in train_data["X4"].values]
test_data["X4"]=[train_mean.loc[i] if i in train_mean.index else y_mean for i in test_data["X

#---------------------------------------------------------------------------------------------

train_mean=train_data.groupby("X5")["y"].mean()
train_data["X5"]=[train_mean.loc[i] for i in train_data["X5"].values]
test_data["X5"]=[train_mean.loc[i] if i in train_mean.index else y_mean for i in test_data["X

#---------------------------------------------------------------------------------------------

train_mean=train_data.groupby("X6")["y"].mean()
train_data["X6"]=[train_mean.loc[i] for i in train_data["X6"].values]
test_data["X6"]=[train_mean.loc[i] if i in train_mean.index else y_mean for i in test_data["X

#---------------------------------------------------------------------------------------------

train_mean=train_data.groupby("X8")["y"].mean()
train_data["X8"]=[train_mean.loc[i] for i in train_data["X8"].values]
test_data["X8"]=[train_mean.loc[i] if i in train_mean.index else y_mean for i in test_data["X

#---------------------------------------------------------------------------------------------


y_train=train_data["y"]
train_data.drop("y",axis=1,inplace=True)


train_data.to_csv("mean_train.csv",index=False)
test_data.to_csv("mean_test.csv",index=False)
```

```
X_train_mean=pd.read_csv("/content/mean_train.csv")
X_test_mean=pd.read_csv("/content/mean_test.csv")


print("The train mean categorical data:",X_train_mean.shape)
print("The test mean categorical data:",X_test_mean.shape)
```

Features Scaling

1. Label encoding

```
std = StandardScaler()
train_label_std=std.fit_transform(X_train_label.values)
test_label_std=std.transform(X_test_label.values)
```

2. Frequency encoding

```
std = StandardScaler()
train_feq_std=std.fit_transform(X_train_feq.values)
test_feq_std=std.transform(X_test_feq.values)
```

3.Mean encoding

```
std = StandardScaler()
train_mean_std=std.fit_transform(X_train_mean.values)
test_mean_std=std.transform(X_test_mean.values)
```

Selecting the principal component features which has highly variance explained

1. Label encoding

```
pca=PCA(n_components=365)
pca_data = pca.fit_transform(train_label_std)
percentage_var_explained = pca.explained_variance_ / np.sum(pca.explained_variance_); # getti
```

```
cum_var_explained = np.cumsum(percentage_var_explained)
for i,j in enumerate(cum_var_explained):
  if np.round(j,2)==0.95:
    feature_select=i

plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.hlines(y =0.95, xmin = 0, xmax = 350, color ='r')
plt.show()
print("Thus {0} principle component explain about 95 percentage of the variance in feature on
```

2. Frequency encoding

```
pca=PCA(n_components=365)
pca_data = pca.fit_transform(train_feq_std)
percentage_var_explained = pca.explained_variance_ / np.sum(pca.explained_variance_); # getti
cum_var_explained = np.cumsum(percentage_var_explained)
for i,j in enumerate(cum_var_explained):
  if np.round(j,2)==0.95:
    feature_select=i
plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.hlines(y =0.95, xmin = 0, xmax = 350, color ='r')
plt.show()
print("Thus {0} principle component explain about 95 percentage of the variance in feature on
```

3. Mean encoding

```
pca=PCA(n_components=365)
pca_data = pca.fit_transform(train_mean_std)
percentage_var_explained = pca.explained_variance_ / np.sum(pca.explained_variance_);# gettin
cum_var_explained = np.cumsum(percentage_var_explained)
for i,j in enumerate(cum_var_explained):
  if np.round(j,2)==0.95:
    feature_select=i


plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.hlines(y =0.95, xmin = 0, xmax = 350, color ='r')
plt.show()
print("Thus {0} principle component explain about 95 percentage of the variance in feature on
```

label encoder

```
pca=PCA(n_components=150)
pca_train_label=pca.fit_transform(train_label_std)
pca_test_label=pca.transform(test_label_std)

print("The pca reductions of the train label encoding:",pca_train_label.shape)
print("The pca reductions of the test label encoding:",pca_test_label.shape)
```

frequency encoder

```
pca=PCA(n_components=151)
pca_train_feq=pca.fit_transform(train_feq_std)
pca_test_feq=pca.transform(test_feq_std)

print("The pca reductions of the train frequency encoding:",pca_train_feq.shape)
print("The pca reductions of the test frequency encoding:",pca_test_feq.shape)
```

mean encoder

```
pca=PCA(n_components=149)
pca_train_mean=pca.fit_transform(train_mean_std)
pca_test_mean=pca.transform(test_mean_std)

print("The pca reductions of the train mean encoding:",pca_train_mean.shape)
print("The pca reductions of the test mean encoding:",pca_test_mean.shape)
```

Model Implementation on Technique 1

Using Base line model to select which categorical features encoding performs better

Label encoding

1.Lasso Regression with Hyperameter tuning with Grid Search CV

```
hyper_value=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
param={"alpha":hyper_value}
model1=Lasso()
grid_cv1 = GridSearchCV(model1, param, cv=12, scoring='r2',return_train_score=True)
grid_cv1.fit(pca_train_label,y_train)
```

```
#https://stackoverflow.com/questions/54608088/what-is-gridsearch-cv-results-could-any-explain
result_df=pd.DataFrame(grid_cv1.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]
```

Plotting of Train score and Test score

```
fold= list(range(1,10))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```

```
print("The best R2 score:",grid_cv1.best_score_)
```

Training of best paramter

```
final_model=grid_cv1.best_estimator_
final_model.fit(pca_train_label,y_train)
predicted=final_model.predict(pca_test_label)
```

```
# submission result csv
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("label_lasso_submission.csv",index=False)
```

Frequency encoding

1.Lasso Regression with Hyperameter tuning with Grid Search CV

```
hyper_value=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
param={"alpha":hyper_value}
model1=Lasso()
grid_cv = GridSearchCV(model1, param, cv=12, scoring='r2',return_train_score=True)
grid_cv.fit(pca_train_feq,y_train)
```

```
result_df=pd.DataFrame(grid_cv.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]
```

ploting the train and test score

```
fold= list(range(1,10))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```

```
print("The best R2 score:",grid_cv.best_score_)
```

Training with the best Parameter

```
final_model=grid_cv.best_estimator_
final_model.fit(pca_train_feq,y_train)
predicted=final_model.predict(pca_test_feq)
```

```
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("feq_lasso_submission.csv",index=False)
```

Mean Encoding

1.Lasso Regression with Hyperameter tuning with Grid Search CV

```
hyper_value=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
param={"alpha":hyper_value}
model1=Lasso()
grid_cv = GridSearchCV(model1, param, cv=12, scoring='r2',return_train_score=True)
grid_cv.fit(pca_train_mean,y_train)
```

```
result_df=pd.DataFrame(grid_cv.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]
```

Ploting the Train and Test Score

```
fold= list(range(1,10))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```

```
print("The best R2 score:",grid_cv.best_score_)
```

Training the best parameter

```
final_model=grid_cv.best_estimator_
final_model.fit(pca_train_mean,y_train)
predicted=final_model.predict(pca_test_mean)
```

```
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("mean_lasso_submission.csv",index=False)
```

```
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["S.no.","Model","R2score","Private_score","Public_score"]
x.add_row(["1","lasso+label_encoding","0.5861","0.51592","0.51392"])
x.add_row(["2","lasso+frequency_encoding","0.5957","0.51980","0.51248"])
x.add_row(["3","lasso+mean_encoding","0.6094","0.52773","0.52489"])

print(x)
```

From the Technique 1 we can conclude that mean encoding categorical features perform well when compare to above encoding.

So here i select the mean encoding categorical features in upcoming Feature engineering.

Technique 2: Analysis on features and removing the features based low variance and highly correlations between them and adding two way and three way combination interactive features in them

Mean Encoding

```
y_mean=train_data.y.mean()
```

```
train_meanX0=train_data.groupby("X0")["y"].mean()
train_data["X0"]=[train_meanX0.loc[i] for i in train_data["X0"].values]
test_data["X0"]=[train_meanX0.loc[i] if i in train_meanX0.index else y_mean for i in test_dat

#---------------------------------------------------------------------------------------------

train_meanX1=train_data.groupby("X1")["y"].mean()
train_data["X1"]=[train_meanX1.loc[i] for i in train_data["X1"].values]
test_data["X1"]=[train_meanX1.loc[i] if i in train_meanX1.index else y_mean for i in test_dat

#---------------------------------------------------------------------------------------------

train_meanX2=train_data.groupby("X2")["y"].mean()
train_data["X2"]=[train_meanX2.loc[i] for i in train_data["X2"].values]
test_data["X2"]=[train_meanX2.loc[i] if i in train_meanX2.index else y_mean for i in test_dat

#---------------------------------------------------------------------------------------------

train_meanX3=train_data.groupby("X3")["y"].mean()
train_data["X3"]=[train_meanX3.loc[i] for i in train_data["X3"].values]
test_data["X3"]=[train_meanX3.loc[i] if i in train_meanX3.index else y_mean for i in test_dat

#---------------------------------------------------------------------------------------------

train_meanX4=train_data.groupby("X4")["y"].mean()
train_data["X4"]=[train_meanX4.loc[i] for i in train_data["X4"].values]
test_data["X4"]=[train_meanX4.loc[i] if i in train_meanX4.index else y_mean for i in test_dat

#---------------------------------------------------------------------------------------------

train_meanX5=train_data.groupby("X5")["y"].mean()
train_data["X5"]=[train_meanX5.loc[i] for i in train_data["X5"].values]
test_data["X5"]=[train_meanX5.loc[i] if i in train_meanX5.index else y_mean for i in test_dat

#---------------------------------------------------------------------------------------------

train_meanX6=train_data.groupby("X6")["y"].mean()
train_data["X6"]=[train_meanX6.loc[i] for i in train_data["X6"].values]
test_data["X6"]=[train_meanX6.loc[i] if i in train_meanX6.index else y_mean for i in test_dat

#---------------------------------------------------------------------------------------------

train_meanX8=train_data.groupby("X8")["y"].mean()
train_data["X8"]=[train_meanX8.loc[i] for i in train_data["X8"].values]
test_data["X8"]=[train_meanX8.loc[i] if i in train_meanX8.index else y_mean for i in test_dat

#---------------------------------------------------------------------------------------------


train_dataX0=pickle.dump(train_meanX0,open("meanX0","wb"))
train_dataX1=pickle.dump(train_meanX1,open("meanX1","wb"))
train_dataX2=pickle.dump(train_meanX2,open("meanX2","wb"))
```

```
train_dataX3=pickle.dump(train_meanX3,open("meanX3","wb"))
train_dataX4=pickle.dump(train_meanX4,open("meanX4","wb"))
train_dataX5=pickle.dump(train_meanX5,open("meanX5","wb"))
train_dataX6=pickle.dump(train_meanX6,open("meanX6","wb"))
train_dataX8=pickle.dump(train_meanX8,open("meanX8","wb"))
```

```
y_train=train_data["y"]
train_data.drop("y",axis=1,inplace=True)
```

Removing the Features which have low variance in categorical features and numerical features which gives less informations.

Numerical binary features and categorical features

```
removal_feat=[]
```

```
#https://github.com/level14taken/mercedes-benz-greener-manufacturing/blob/master/SelfCaseStud
threshold_value=0.01
var_dic={}
for col in train_data.columns:
    if train_data[col].var()<=threshold_value:#setting the variance threshold value to get le
       var_dic[col]=train_data[col].var()
```

```
print("The no. of less variance numerical binary features:" ,len(var_dic))
```

```
less_var_col=list(var_dic.keys())
train_data=train_data.drop(less_var_col,axis=1)# removing the features which has less varianc
print("The dataset shape after removing the less variance features:",train_data.shape)
```

```
removal_feat=less_var_col
```

checking the duplicate numerical binary columns and and highly correlated numerical binary columns and finally removing them.

```
threshold=1
feat_corr = set() # Set of all the names of deleted columns
corr_mat = train_data.corr()
```

```
for i in range(len(corr_mat.columns)):
    for j in range(i):
        if (corr_mat.iloc[i, j] == threshold) and (corr_mat.columns[j] not in feat_corr):
            colname = corr_mat.columns[i] # getting the name of column
            feat_corr.add(colname)
            if colname in train_data.columns:
                del train_data[colname] # deleting the column from the dataset
```

```
print("No. of Duplicate Featureas:",len(feat_corr))
```

```
print('The size of the features after removing the duplicate features:',train_data.shape)
```

```
col_corr=list(feat_corr)
removal_feat.extend(feat_corr)
```

```
test_data.drop(removal_feat,axis=1,inplace=True)
```

```
print("The final Train dataset:",train_data.shape)
print("The final test dataset:",test_data.shape)
```

Adding two way and three way interaction features

Let pick some of the top interactive features which are hightly correlated with the target y variable using pearson correlation coefficient technique

```
pear_cor_dic={}
for col in train_data.columns:
  corr=pearsonr(train_data[col],y_train)
  if corr[0]>=0.25:
     pear_cor_dic["Top Person correlation coeffienct value with respect to output variable",c
```

```
pear_cor_dic
```

Observations:

The above are important features which highly correlated with the target variable. Let pick some top high correlated and add them with two way interaction and three wahy interaction features to the dataset

Two way interction features

```
train_data["X136_add_X261"]=train_data.apply(lambda row: row.X136 + row.X261, axis=1)
train_data["X118_add_X275"]=train_data.apply(lambda row: row.X118 + row.X275, axis=1)
train_data["X314_add_X315"]=train_data.apply(lambda row: row.X314 + row.X315,axis=1)
train_data["X0_add_X1"]=train_data.apply(lambda row: row.X0 + row.X1,axis=1)
train_data["X263_add_X264"]=train_data.apply(lambda row: row.X263 + row.X264,axis=1)
print("The correlation between X136_add_X261 and target variable is:",pearsonr(train_data["X1
print("The correlation between X118_add_X275 and target variable is:",pearsonr(train_data["X1
print("The correlation between X314_add_X315 and target variable is:",pearsonr(train_data["X3
#print("The correlation between X0_add_X1 and target variable is:",pearsonr(train_data["X0_ad
print("The correlation between X263_add_X264 and target variable is:",pearsonr(train_data["X2
```

```
test_data["X136_add_X261"]=test_data.apply(lambda row: row.X136 + row.X261, axis=1)
test_data["X118_add_X275"]=test_data.apply(lambda row: row.X118 + row.X275, axis=1)
test_data["X314_add_X315"]=test_data.apply(lambda row: row.X314 + row.X315,axis=1)
test_data["X0_add_X1"]=test_data.apply(lambda row: row.X0 + row.X1,axis=1)
test_data["X263_add_X264"]=test_data.apply(lambda row: row.X263 + row.X264,axis=1)
```

Three way interaction features

```
train_data["X136_add_X275_add_X261"]=train_data.apply(lambda row: row.X136 + row.X275 + row.X
train_data["X118_add_X314_add_X315"]=train_data.apply(lambda row: row.X118 + row.X314 + row.X
train_data["X261_add_X263_add_X264"]=train_data.apply(lambda row: row.X261 + row.X263 + row.X
train_data["X0_add_X1_add_X2"]=train_data.apply(lambda row: row.X0 + row.X1 + row.X2, axis=1)
print("The correlation between X136_add_X275_add_X261 and target variable is:",pearsonr(train_
print("The correlation between X118_add_X314_add_X315 and target variable is:",pearsonr(train_
print("The correlation between X261_add_X263_add_X264 and target variable is:",pearsonr(train_
print("The correlation between X0_add_X1_add_X2 and target variable is:",pearsonr(train_data[
```

```
test_data["X136_add_X275_add_X261"]=test_data.apply(lambda row: row.X136 + row.X275 + row.X26
test_data["X118_add_X314_add_X315"]=test_data.apply(lambda row: row.X118 + row.X314 + row.X31
test_data["X261_add_X263_add_X264"]=test_data.apply(lambda row: row.X261 + row.X263 + row.X26
test_data["X0_add_X1_add_X2"]=test_data.apply(lambda row: row.X0 + row.X1 + row.X2, axis=1)
```

Let remove some highly correlated features which keeping threshold as 0.95

```
threshold=0.95
feat_corr = set() # Set of all the names of deleted columns
corr_mat = train_data.corr()

for i in range(len(corr_mat.columns)):
    for j in range(i):
        if (corr_mat.iloc[i, j] > threshold) and (corr_mat.columns[j] not in feat_corr):
            colname = corr_mat.columns[i] # getting the name of column

            feat_corr.add(colname)
            if colname in train_data.columns:
                del train_data[colname] # deleting the column from the dataset


print("The high correlated features:",len(feat_corr))
```

```
removal_feat.extend(feat_corr)
```

```
removal_features=pickle.dump(removal_feat,open("rmoval_feat.pkl","wb"))
```

```
col_removal=list(feat_corr)
test_data.drop(col_removal,axis=1,inplace=True)
```

```
print("The final train data shape:",train_data.shape)
print("The final test data shape:",test_data.shape)
print( The final test data shape: ,test_data.shape)
```

```
train_data.to_csv("final_traindata.csv",index=False)
test_data.to_csv("final_testdata.csv",index=False)
```

```
train_data=pd.read_csv("/content/drive/My Drive/final_traindata.csv")
test_data=pd.read_csv("/content/drive/My Drive/final_testdata.csv")
```

```
train_data.head(3)
```

```
test_data.head(3)
```

Checking the feature importance through simple RandomforestRegressor model

```
model = RandomForestRegressor(n_estimators=100, max_depth=5,
                              min_samples_leaf=3, max_features=0.2,
                              n_jobs=-1, random_state=0)
model.fit(train_data,y_train)
feat_names = train_data.columns.values
importances = model.feature_importances_

indices = np.argsort(importances)[::-1][:50]
plt.subplots(figsize=(10,10))
plt.title("Top 50 featues of Feature importances by RandomForestRegressor")
plt.ylabel("Features")
plt.barh(range(len(indices)), importances[indices], color="green", align="center")
plt.yticks(range(len(indices)), feat_names[indices], rotation='horizontal')
plt.ylim([-1, len(indices)])
plt.savefig("b.jpg")
```

observations:

1.From the features important plot we can conclude that the two way and three way interactive features give more contributions in predicting the output variable.

2.Thus the ID features and categorical features are not giving much improtance in the predicting the values.

Checking the feature importance through simple XGboostRegressor model

```
def r2_score_xgb(preds, final):
 labels = dtrain.get_label()
 return r2_score(labels, preds)

xgb_params = {'n_trees': 500,'max_depth':5,'subsample': 0.98,'objective': 'reg:linear','eval_
train = xgb.DMatrix(train_data,y_train, feature_names=train_data.columns.values)
model = xgb.train(dict(xgb_params), train, num_boost_round=1200, feval=r2_score_xgb, maximize
fig, ax = plt.subplots(figsize=(10,10))
xgb.plot_importance(model, max_num_features=50, height=0.8, ax=ax,color = 'coral')
print("Top 50 features as Feature Importance by XGBoost")
plt.show()
```

Observations:

1. By applying the Xgboost model checking the features importance the ID features and categorical features gives more importance in predicting the output variable.

Model Implementation in Technique 2

1. Lasso Regression with Hyper parameter tuning

```
hyper_value=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
param={"alpha":hyper_value}
model1=Lasso()
grid_cv = GridSearchCV(model1, param, cv=12, scoring='r2',return_train_score=True)
grid_cv.fit(train_data,y_train)
```

```
pred_stack_label = stack.predict(test_data.values)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=pred_stack_label
submission_df.to_csv("tech2_stack_submission.csv",index=False)
```

```
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["S.no.","Model","R2score","Private_score","Public_score"]
x.add_row(["1","Tech2_lasso","0.64125","0.54564","0.55149"])
x.add_row(["3","Tech2_Random_forest","0.6390","0.55006","0.55913"])
x.add_row(["4","Tech2_Xgboost","0.6347","0.54450","0.54872"])
x.add_row(["5","Tech2_Extratree","0.6357","0.54792","0.55156"])
x.add_row(["6","Tech2_stack_model","","0.55096","0.55623"])
print(x)
```

Technique 3: Selection of Top Features in Technique 2 using SelectKbest technique

Tuning to select the number of features using baseline linear Regression Model

```
#https://analyticsweek.com/content/how-to-perform-feature-selection-for-regression-data/
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
# define the pipeline to evaluate
model = LinearRegression()
fs = SelectKBest(score_func=f_regression)
pipeline = Pipeline(steps=[('sel',fs), ('lr', model)])
# define the grid
grid = dict()
grid['sel__k'] = [i for i in range(train_data.shape[1]-50, train_data.shape[1]+1)]
# define the grid search
search = GridSearchCV(pipeline, grid, scoring='r2', n_jobs=-1, cv=cv)
# perform the search
results = search.fit(train_data, y_train)
# summarize best
print('Best R2: %.3f' % results.best_score_)
print('Best no. of features: %s' % results.best_params_)
# summarize all
means = results.cv_results_['mean_test_score']
params = results.cv_results_['params']
for mean, param in zip(means, params):
    print(">%.3f with: %r" % (mean, param))
```

```
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
        positive)
      /usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
```

```
result_df=pd.DataFrame(grid_cv.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]
```

Result:

1. Here the above process it use linear regression as the base model with repeated cv tuning to select the best no. of features using SelectKBest technique.
2. From the above results the best no. of features which gives as 146 features.

```
select_k_best=SelectKBest(score_func=f_regression, k=146)
best_train=select_k_best.fit_transform(train_data,y_train)
best_test=select_k_best.transform(test_data)
```

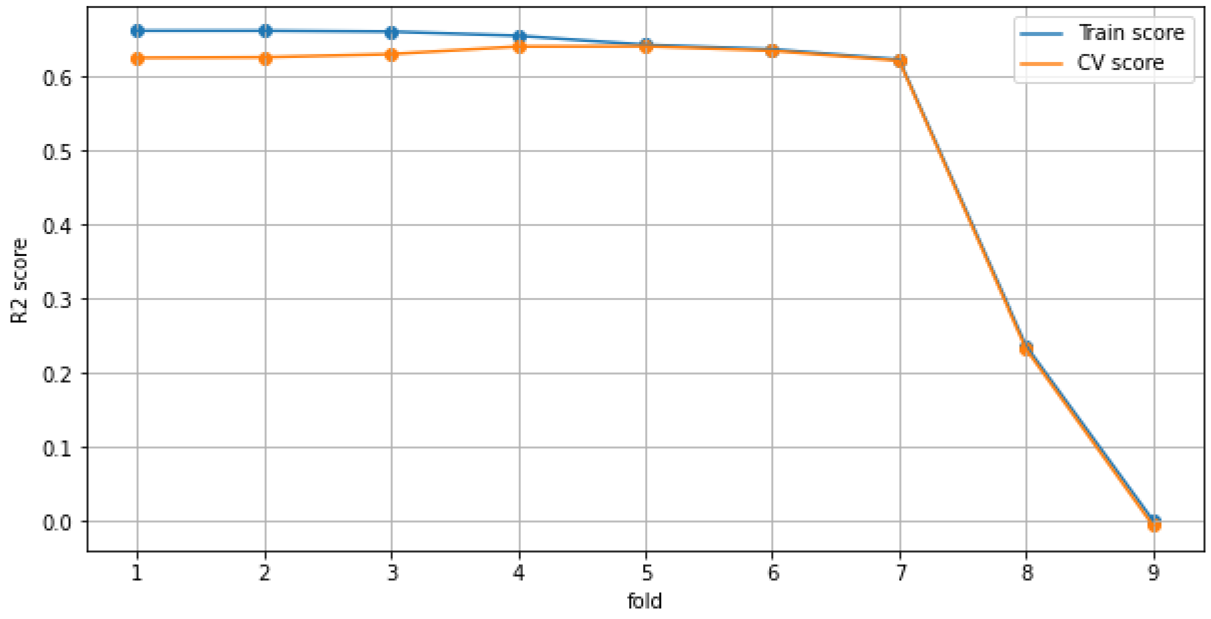
```
best_columns=pickle.dump(train_data.columns.values[select_k_best.get_support()],open("best_co
```

```
best_train=pd.DataFrame(data=best_train,columns=train_data.columns.values[select_k_best.get_s
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_coordinate_descent.py:476:
```

Plotting the Train Score and Test score

```
  positive)
```

```
fold= list(range(1,10))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```



```
print("The best R2 score:",grid_cv.best_score_)
final_model=grid_cv.best_estimator_
final_model.fit(train_data,y_train)
predicted=final_model.predict(test_data)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("tech2_lasso1_submission.csv",index=False)
```

```
The best R2 score: 0.6412596426192403
```

3.Random forest with hyperparameter tuning

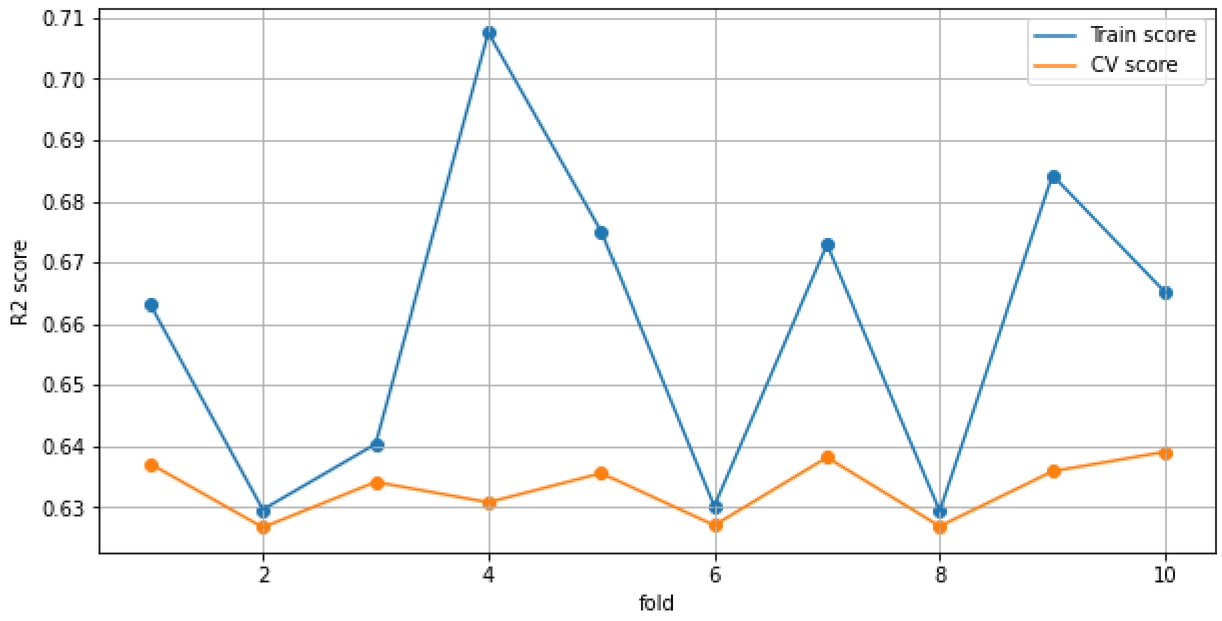
```
neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
```

```
parameters = {'n_estimators':[40,50,60,70,100],
             'max_depth':[3,5,6,7,8],
             'min_samples_split':[2,3,4,5,6,7,8,9,10],
             'max_features': [.95],
             'min_samples_leaf': [1, 2,3,4,5,6,7,8,9],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10]}
rand_clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_job
rand_clf.fit(train_data,y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   10.7s
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.7min finished
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100, n_jobs=-1,
                                                   oob_score=False...
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [3, 5, 6, 7, 8],
                                        'max_features': [0.95],
                                        'min_impurity_decrease': [1e-05, 0.0001,
                                                                  0.001, 0.01,
                                                                  0.1, 0, 1,
                                                                  10],
                                        'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                             7, 8, 9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
                                        'n_estimators': [40, 50, 60, 70, 100]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

```
result_df=pd.DataFrame(rand_clf.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]
```

```
fold= list(range(1,11))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```



```
print("The best score is :",rand_clf.best_score_)
rand_model=rand_clf.best_estimator_
rand_model.fit(train_data,y_train)
predicted=rand_model.predict(test_data)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("tech2_rf_submission.csv",index=False)
```

```
The best score is : 0.6390731589013059
```

4. XGBoost Regressor with Randomized Search CV

```
neigh=XGBRegressor(random_state=42,n_jobs=-1)
parameters = {'learning_rate':[0.001,0.01,0.05,0.1,1],
             'n_estimators':[50,70,80,100],
             'max_depth':[2,3,4,5],
             'colsample_bytree':[0.1,0.5,0.7,1],
             'subsample':[0.2,0.3,0.5,1],
             'gamma':[1e-2,1e-3,0,0.1,0.01,0.5,1],
             'reg_alpha':[1e-5,1e-3,1e-1,1,1e1]}
Xgb_clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs
Xgb_clf.fit(train_data,y_train)
```

```
best_train=pd.DataFrame(data=best_train,columns=train_data.columns.values[select_k_best.get_s
best_test=pd.DataFrame(data=best_test,columns=test_data.columns.values[select_k_best.get_supp


print("The best train dataset shape:",best_train.shape)
print("The best test dataset shape:",best_test.shape)
```

Making the top K best features dataset

```
names = train_data.columns.values[select_k_best.get_support()]
scores = select_k_best.scores_[select_k_best.get_support()]
names_scores = list(zip(names, scores))
ns_df = pd.DataFrame(data = names_scores, columns=['Feat_names', 'F_Scores'])
ns_df_sorted = ns_df.sort_values(['F_Scores', 'Feat_names'], ascending = [False, True])
ns_df_sorted.head()
```

```
model = RandomForestRegressor(n_estimators=100, max_depth=5, min_samples_leaf=3, max_features
model.fit(best_train,y_train)
feat_names = best_train.columns.values
importances = model.feature_importances_

indices = np.argsort(importances)[::-1][:50]
plt.subplots(figsize=(10,10))
plt.title("Top 50 featues of Feature importances by RandomForestRegressor")
plt.ylabel("Features")
plt.barh(range(len(indices)), importances[indices], color="green", align="center")
plt.yticks(range(len(indices)), feat_names[indices], rotation='horizontal')
plt.ylim([-1, len(indices)])
plt.show()
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   15.3s
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:   47.3s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.1min finished
[08:37:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now de
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=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=42, reg_alp...
                   param_distributions={'colsample_bytree': [0.1, 0.5, 0.7, 1],
                                        'gamma': [0.01, 0.001, 0, 0.1, 0.01,
                                                  0.5, 1],
                                        'learning_rate': [0.001, 0.01, 0.05,
                                                          0.1, 1],
                                        'max_depth': [2, 3, 4, 5],
                                        'n_estimators': [50, 70, 80, 100],
                                        'reg_alpha': [1e-05, 0.001, 0.1, 1,
                                                      10.0],
                                        'subsample': [0.2, 0.3, 0.5, 1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='r2', verbose=5)
```

```
result_df=pd.DataFrame(Xgb_clf.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]
```

```
fold= list(range(1,11))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```

Observations:

1.Thus by applying the Selectkbest features selection technique applied top 146 select features thus the interactive features gives more importantance in the predicting the output variable.

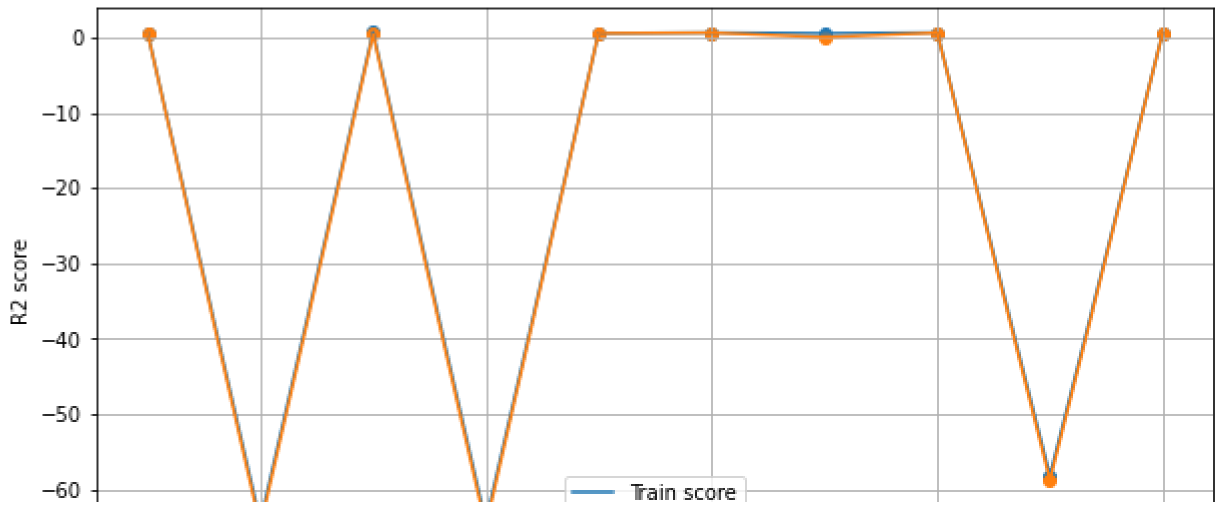
Model Implementation

1.Lasso Regression with hyper-parameter tuning

```
hyper_value=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
param={"alpha":hyper_value}
model1=Lasso(fit_intercept=True, normalize=False)
grid_cv = GridSearchCV(model1, param, cv=12, scoring='r2',return_train_score=True)
grid_cv.fit(best_train,y_train)
```

```
result_df=pd.DataFrame(grid_cv.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]
```

```
fold= list(range(1,10))
```



```
print("The best R2 score:",Xgb_clf.best_score_)
Xgb_model=Xgb_clf.best_estimator_
Xgb_model.fit(train_data,y_train)
predicted=Xgb_model.predict(test_data)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("tech2_XGB_submission.csv",index=False)
```

```
The best R2 score: 0.6347174855910269
[08:37:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now de
```

```
predicted = clf.predict(d_test)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("tech2_XGB1_submission.csv",index=False)
```

EXtra Tree Regressor with Randomized search cv

```
neigh=ExtraTreesRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[50,60,70,80,100],
             'max_depth':[3,4,5,6,7],
             'min_samples_split':[2, 5, 10],
             'max_features': [.95],
             'min_samples_leaf': [3,4,5,6,7,10],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]}
Extra_clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jo
Extra_clf.fit(train_data,y_train)
```

```
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```

```
print("The best R2 score:",grid_cv.best_score_)
```

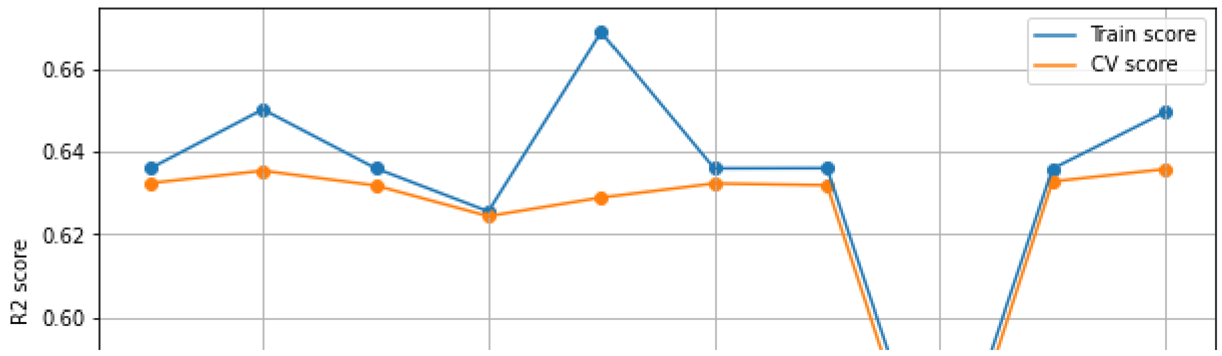
```
lasso_reg=grid_cv.best_estimator_
print("The best score:",grid_cv.best_score_)
lasso_reg.fit(best_train,y_train)
predicted=lasso_reg.predict(best_test)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("tech3_lasso_submission.csv",index=False)
```

2. Random Forest with Randomized search cv

```
neigh=RandomForestRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[100,150,200,300,350,500],
             'max_depth':[2,3,5,7,10],
     Fitting 10 folds for each of 10 candidates, totalling 100 fits
     [Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
     [Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   11.3s
     [Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:   48.0s
     [Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.2min finished
     RandomizedSearchCV(cv=10, error_score=nan,
                        estimator=ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0,
                                                      criterion='mse',
                                                      max_depth=None,
                                                      max_features='auto',
                                                      max_leaf_nodes=None,
                                                      max_samples=None,
                                                      min_impurity_decrease=0.0,
                                                      min_impurity_split=None,
                                                      min_samples_leaf=1,
                                                      min_samples_split=2,
                                                      min_weight_fraction_leaf=0.0,
                                                      n_estimators=100, n_jobs=-1,
                                                      oob_score=False,...
                        iid='deprecated', n_iter=10, n_jobs=-1,
                        param_distributions={'max_depth': [3, 4, 5, 6, 7],
                                             'max_features': [0.95],
                                             'min_impurity_decrease': [1e-05, 0.0001,
                                                                       0.001, 0.01,
                                                                       0.1, 0, 1, 10,
                                                                       100],
                                             'min_samples_leaf': [3, 4, 5, 6, 7, 10],
                                             'min_samples_split': [2, 5, 10],
                                             'n_estimators': [50, 60, 70, 80, 100]},
```

```
result_df=pd.DataFrame(Extra_clf.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]


fold= list(range(1,11))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```



```
print("The best R2 score:",Extra_clf.best_score_)
Extratree_model=Extra_clf.best_estimator_
Extratree_model.fit(train_data,y_train)
predicted=Extratree_model.predict(test_data)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("tech2_Extratree_submission.csv",index=False)
```

Stacking all the above model

```
ridge= Ridge(random_state=42,fit_intercept= False,alpha=0.1)
stack = StackingCVRegressor(regressors=(rand_model,Extratree_model,Xgb_model),
                            meta_regressor=ridge,
                            use_features_in_secondary=False,refit=True,cv=5)

cv_score=cross_val_score(stack,train_data,y_train,scoring='r2',cv= 10,verbose=5,n_jobs=-1)
print('Mean Score:',cv_score.mean())
print('Standard Deviation:',cv_score.std())
stack.fit(train_data.values,y_train.values)
```

```
             'min_samples_split':[2,3,4,5,6,7,8,9,10],
             'max_features': [.95],
             'min_samples_leaf': [1,2,3,4,5,6,7,8,9],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10]}
rand_clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_job
rand_clf.fit(best_train,y_train)
```

```
result_df=pd.DataFrame(rand_clf.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]
```

```
fold= list(range(1,11))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```

```
print("The best R2 score:",rand_clf.best_score_)
rand_reg=rand_clf.best_estimator_
rand_reg.fit(best_train,y_train)
predicted=rand_reg.predict(best_test)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("tech3_rf1_submission.csv",index=False)
```

4. Xgboost Regressor With Randomized search cv

```
neigh=XGBRegressor(random_state=42,n_jobs=-1)
parameters = {'learning_rate':[0.001,0.01,0.05,0.1,1],
             'n_estimators':[100,150,200,500],
             'max_depth':[2,3,5,10],
             'colsample_bytree':[0.1,0.5,0.7,1],
             'subsample':[0.2,0.3,0.5,1],
             'gamma':[1e-2,1e-3,0,0.1,0.01,0.5,1],
             'reg_alpha':[1e-5,1e-3,1e-1,1,1e1]}
Xgb_clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jobs
Xgb_clf.fit(best_train,y_train)
```

```
result_df=pd.DataFrame(Xgb_clf.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]


fold= list(range(1,11))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```

```
print("The best R2 score:",Xgb_clf.best_score_)
Xgb_reg=Xgb_clf.best_estimator_
Xgb_reg.fit(best_train,y_train)
predicted=Xgb_reg.predict(best_test)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("tech3_xgb_submission.csv",index=False)
```

Extra-Tree REgression with hyper-paramter

```
neigh=ExtraTreesRegressor(random_state=42, n_jobs=-1)
parameters = {'n_estimators':[150,200,300,500],
             'max_depth':[2,3,4,5,10],
             'min_samples_split':[2, 5, 10],
             'max_features': [.95],
             'min_samples_leaf': [3,4,5,6,7,10],
             'min_impurity_decrease':[1e-5,1e-4,1e-3,1e-2,1e-1,0,1,10,100]}
Extra_clf=RandomizedSearchCV(neigh,parameters,cv=10,scoring='r2',return_train_score=True,n_jo
Extra_clf.fit(best_train,y_train)
```

```
result_df=pd.DataFrame(Extra_clf.cv_results_)
train_score=result_df["mean_train_score"]
cv_score=result_df["mean_test_score"]


fold= list(range(1,11))
plt.figure(figsize=(10,5))
plt.plot(fold,train_score,label='Train score')
plt.plot(fold,cv_score,label='CV score')
plt.scatter(fold,train_score)
plt.scatter(fold, cv_score)
plt.xlabel("fold")
plt.ylabel("R2 score")
plt.grid()
plt.legend()
plt.show()
```

```
print("The best R2 score:",Extra_clf.best_score_)
Extra_reg=Extra_clf.best_estimator_
Extra_reg.fit(best_train,y_train)
predicted=Extra_reg.predict(best_test)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=predicted
submission_df.to_csv("tech3_Extra_submission.csv",index=False)
```

Stacking of all model

```
ridge = Ridge(random_state=42, fit_intercept=False, alpha=0.1)
stack = StackingCVRegressor(regressors=(rand_reg,Xgb_reg,Extra_reg),
                           meta_regressor=ridge,
                           use_features_in_secondary = False, refit=True, cv=5)

cv_score=cross_val_score(stack,best_train,y_train.values,scoring='r2',cv= 5,verbose=5,n_jobs=
print('Mean Score:',cv_score.mean())
print('Standard Deviation:',cv_score.std())
stack.fit(best_train.values,y_train.values)
```

```
pred_stack_label = stack.predict(best_test.values)
submission_df=pd.DataFrame()
submission_df["ID"]=test_data["ID"]
submission_df["y"]=pred_stack_label
submission_df.to_csv("tech3_stack_submission.csv",index=False)
```

```
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["S.no.","Model","R2score","Private_score","Public_score"]
x.add_row(["1","Tech3_lasso","0.6424","0.54360","0.54846"])
x.add_row(["2","Tech3_Random_forest","0.6381","0.55072","0.55870"])
x.add_row(["3","Tech3_Xgboost","0.6439","0.54902","0.55458"])
x.add_row(["4","Tech3_Extratree","0.6450","0.54950","0.55411"])
x.add_row(["5","Tech3_stack_model","","0.55170","0.55704"])

print(x)
```

Step by Step Procedure to aproach the Problem

1. Loading the Train data provided by the kaggle to work with the above problem.
2. Perform Statistical analysis in the train data.

3. Applying some pre-processing technique to check the Null values and duplicate values present inside the train dataset.
4. Performing Exploratory Data Analysis on train data of categorical Variables, numerical binary variables and output variables.
5. Preparation of Test data from the Kaggle
6. Performing Feature Engineering in both train data and test data.Thus feature Engineering split into three techniques.

i) Technique 1-Dimentionality Reduction with PCA.

a) Label Encoding

b) Frequency Encoding

c) Mean Encoding

ii) Technique 2- Performing Analysis on Features and adding some Interative features.

iii) Technique 3- Performing Top Features Selection in Technique2 by using SelectKbest Technique.

7. Model Implementation

i.) Lasso Regression

ii.) Random Forest Regressor

iii.) ExtraRegressor

iv.) Xgboost Regressor

v.) Stacking model

8. Result and conclusion are performed in the below table.

```
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["S.no.","Model","R2score","Private_score","Public_score"]
x.add_row(["1","lasso+label_encoding","0.5861","0.51592","0.51392"])
x.add_row(["2","lasso+frequency_encoding","0.5957","0.51980","0.51248"])
x.add_row(["3","lasso+mean_encoding","0.6094","0.52773","0.52489"])
x.add_row(["4","Tech2_lasso","0.64125","0.54564","0.55149"])
x.add_row(["5","Tech2_Random_forest","0.6390","0.55006","0.55913"])
x.add_row(["6","Tech2_Xgboost","0.6347","0.54450","0.54872"])
x.add_row(["7","Tech2_Extratree","0.6357","0.54792","0.55156"])
x.add_row(["8","Tech2_stack_model","","0.55096","0.55623"])
x.add_row(["9","Tech3_lasso","0.6424","0.54360","0.54846"])
x.add_row(["10","Tech3_Random_forest","0.6381","0.55072","0.55870"])
x.add_row(["11","Tech3_Xgboost","0.6439","0.54902","0.55458"])
x.add_row(["12","Tech3_Extratree","0.6450","0.54950","0.55411"])
```

```
x.add_row(["12","Tech3_ExtraTree","0.6496","0.54950","0.55411"])
x.add_row(["13","Tech3_stack_model","","0.55170","0.55704"])

print(x)
```

From the above Result we can observe that the stacking model perform well when compared to other model.

we can clearly observe that both technique 2 and Technique 3 the the stacking model perform good and can see the more raise in private score.

By the conclusion we declare that the Technique 3 features engineering stacking model as the best model.

Saving the best model in pickle file to perform with pipeline design of the problem

```
filename=open("finalized_model.pkl","wb")
saved_model = pickle.dump(stack,filename)

# Load the pickled model

stack_saved = pickle.load(open("/content/drive/My Drive/finalized_model.pkl","rb"))

# Use the loaded pickled model to make predictions
stack_saved.predict(best_test.values)
```