**Mercedes Benz Greener Manufacturing\_Final**

**Step 1: Importing all the required packages**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

label=LabelEncoder

import warnings

warnings.filterwarnings( 'ignore')

**Step 2: Reading the train and test files**

train\_data= pd.read\_csv(r'/content/train.csv')

test\_data= pd.read\_csv(r'/content/test.csv')

**Step 3: Checking the first five rows of train and test data**

train\_data.head()

test\_data.head()

**Step 4: Checking the shape of train and test data**

train\_data.shape

(4209, 378)

test\_data.shape

(4209, 377)

**Step 5: Describing train and test data**

train\_data.describe()

test\_data.describe()

**Step 6: Understand the data types of train and test datasets we have**

cols = [c for c in train\_data.columns if 'X' in c]

print('Number of features: {}'.format(len(cols)))

print('Feature types:')

train\_data[cols].dtypes.value\_counts()

Number of features: 376

Feature types:

int64 368

object 8

dtype: int64

cols = [c for c in test\_data.columns if 'X' in c]

print('Number of features: {}'.format(len(cols)))

print('Feature types:')

test\_data[cols].dtypes.value\_counts()

Number of features: 376

Feature types:

int64 368

object 8

dtype: int64

**Step 7: Counting the number & type of datatypes availabe in the dataframe**

counts = [[], [], []]

for c in cols:

typ = train\_data[c].dtype

uniq = len(np.unique(train\_data[c]))

if uniq == 1:

counts[0].append(c)

elif uniq == 2 and typ == np.int64:

counts[1].append(c)

else:

counts[2].append(c)

print('Constant features: {} Binary features: {} Categorical features: {}\n'

.format(\*[len(c) for c in counts]))

print('Constant features:', counts[0])

print('Categorical features:', counts[2])

Constant features: 12 Binary features: 356 Categorical features: 8

Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293

Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']

**Step 8: Checking for variables having zero variance**

variance\_with\_zero = train\_data.var()[train\_data.var()==0].index.values

variance\_with\_zero

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

**Step 9: Viewing the variables having zero variance**

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

train\_data.head()

**Step 10: Dropping zero variance variables**

train\_data = train\_data.drop(variance\_with\_zero, axis=1)

train\_data.shape

(4209, 366)

**Step 11: Removing columns ID and Y from the datasets as they are not used for learning**

usable\_columns = list(set(train\_data.columns) - set(['ID', 'y']))

y\_train = train\_data[ 'y'].values

id\_test = test\_data[ 'ID'].values

x\_train = train\_data[usable\_columns]

x\_test = test\_data[usable\_columns]

x\_train.shape

(4209, 364)

x\_test.shape

(4209, 364)

**Step 12: Checking for null and unique values for test and train sets**

def check\_missing\_values(df):

if df.isnull().any().any():

print("There are missing values in the dataframe")

else:

print("There are no missing values in the dataframe")

check\_missing\_values(x\_train)

check\_missing\_values(x\_test)

There are no missing values in the dataframe

There are no missing values in the dataframe

x\_train.agg([ 'nunique','count','size', 'dtypes'])

x\_test.agg([ 'nunique','count','size', 'dtypes'])

x\_train.dtypes.unique() # here we have int64 ad object datatype values

array([dtype('int64'), dtype('O')], dtype=object)

x\_test.dtypes.unique()

array([dtype('int64'), dtype('O')], dtype=object)

**Step 13: Applying label encoder to train data**

for column in usable\_columns:

cardinality = len(np.unique(x\_train[column]))

if cardinality > 2: # Column is categorical

mapper = lambda x: sum([ord(digit) for digit in x])

x\_train[column] = x\_train[column].apply(mapper)

x\_test[column] = x\_test[column].apply(mapper)

x\_train.head()

x\_test.head()

x\_test.dtypes.unique() # now we have only int64 datatype values

array([dtype('int64')], dtype=object)

x\_train.dtypes.unique()

array([dtype('int64')], dtype=object)

**Step 14: Perform dimensionality reduction (PCA) on train and test datasets**

from sklearn.decomposition import PCA

n\_comp = 12

pca = PCA(n\_components=n\_comp, random\_state=420)

pca2\_results\_train = pca.fit\_transform(x\_train)

pca2\_results\_test = pca.transform(x\_test)

print(pca2\_results\_train.shape)

print(pca2\_results\_test.shape)

(4209, 12)

(4209, 12)

**Step 15: Training the model using xgboost**

import xgboost as xgb

from sklearn.metrics import r2\_score, mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

x\_train, x\_valid, y\_train, y\_valid = train\_test\_split(pca2\_results\_train, y\_train, test\_size=0.2, random\_state=42)

**Step 16: Creating the Xgboost specific DMatrix data format from the numpy array**

d\_train = xgb.DMatrix(x\_train, label=y\_train)

d\_valid = xgb.DMatrix(x\_valid, label=y\_valid)

d\_test = xgb.DMatrix(pca2\_results\_test)

**Step 17: Setting the parameters for Xgboost to work**

params = {}

params['objective'] = 'reg:linear'

params['eta'] = 0.02

params['max\_depth'] = 4

def xgb\_r2\_score(preds, dtrain):

labels = dtrain.get\_label()

return 'r2', r2\_score(labels, preds)

watchlist = [(d\_train, 'train'), (d\_valid, 'valid')]

clf = xgb.train(params, d\_train,

1000, watchlist, early\_stopping\_rounds=50,

feval=xgb\_r2\_score, maximize=True, verbose\_eval=10)

reg:linear is now d

[0] train-rmse:98.997 valid-rmse:98.8888 train-r2:-59.4973 valid-r

Multiple eval metrics have been passed: 'valid-r2' will be used for early stopping.

Will train until valid-r2 hasn't improved in 50 rounds.

Stopping. Best iteration:

[376] train-rmse:8.1717 valid-rmse:9.19791 train-r2:0.587791 valid-r

**Step 18: Predicting test df values using xgboost and saving the Predictions CSV file.**

p\_test = clf.predict(d\_test)

sub = pd.DataFrame()

sub['ID'] = id\_test

sub['y'] = p\_test

sub.to\_csv('xgb.csv', index=False)

sub.head()

ID y

0 1 79.287155

1 2 96.074661

2 3 81.250526

3 4 77.294121

4 5 109.705254