Project -1 : Mercedes-Benz Greener Manufacturing

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.

Import required python modules

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
```

Load datasets

In [2]:
```
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
print(df_train.shape)
print(df_test.shape)
```

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

In [3]:
```
df_train.head()
```

Out[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0	0

5 rows × 378 columns

In [4]:
```

df_test.head()
```

Out[4]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X3
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0	0	
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0	0	
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0	0	
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0	0	
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0	0	

5 rows × 377 columns

Check Missing Value/Null in training data and test data

In [5]:
```
df_train.isnull().sum()
```

Out[5]:
```
ID       0
y        0
X0       0
X1       0
X2       0
        ..
X380     0
X382     0
X383     0
X384     0
X385     0
Length: 378, dtype: int64
```

In [6]:
```
df_test.isnull().sum()
```

Out[6]:
```
ID      0
X0      0
X1      0
X2      0
X3      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 377, dtype: int64
```

In [7]:
```
# descriptive analysis
df_train.describe()
```

Out[7]:

	ID	y	X10	X11	X12	X13	X14	X1
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.00000
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.00047
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.02179
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.00000
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.00000
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.00000
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.00000
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.00000

8 rows × 370 columns

In [8]:
```
# we will create a new target column (same as training) in testing dataset
# and then append testing dataset after training dataset
df_test['y'] = np.nan
df_test.shape
df_test.isnull().sum()
```

Out[8]:
```
ID         0
X0         0
X1         0
X2         0
X3         0
        ...
X382       0
X383       0
X384       0
X385       0
y       4209
Length: 378, dtype: int64
```

In [9]:
```
# append testing dataset after training dataset
df_appended = df_train.append(df_test)
df_appended.shape
df_appended.isnull().sum()
df_appended.head()
```

Out[9]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0	0

5 rows × 378 columns

In [10]:
```
# NULL value checking. Replace NULL/Nan with mean value
df_appended.fillna(df_appended.mean(),inplace = True)
df_appended.isnull().sum()
```

Out[10]:
```
ID       0
y        0
X0       0
X1       0
X2       0
        ..
X380     0
X382     0
X383     0
X384     0
X385     0
Length: 378, dtype: int64
```

It's good that no null values found in test and train datasets

Check for unique values for test and traing sets

In [11]:
```
# check unique values
#['X0','X1','X2','X3','X4','X5','X6','X8']
column_values = df_appended[['X0','X1','X2','X3','X4','X5','X6','X8']].values
unique_values =  np.unique(column_values)
print("Unique Values :",unique_values)
```

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

If for any columns, the variance is equal to zero, then you need to remove those variables.

In [12]:
```
# check variance :
# variance is the expectation of the squared deviation of a random
# variable from its mean. Informally, it measures how far a set of (random)
# numbers are spread out from their average value.
df_appended.var()
```

Out[12]:
```
ID       5.905928e+06
y        8.037380e+01
X10      1.589673e-02
X11      1.187931e-04
X12      6.914585e-02
             ...
X380     8.013627e-03
X382     8.130501e-03
X383     1.068121e-03
X384     5.936830e-04
X385     1.542108e-03
Length: 370, dtype: float64
```

Apply label encoder.

In [13]:
```
# Apply Label Encoder on below category columns :
# ['X0','X1','X2','X3','X4','X5','X6','X8']
le = LabelEncoder()
df_appended['X0'] = le.fit_transform(df_appended['X0'])
df_appended['X1'] = le.fit_transform(df_appended['X1'])
df_appended['X2'] = le.fit_transform(df_appended['X2'])
df_appended['X3'] = le.fit_transform(df_appended['X3'])
df_appended['X4'] = le.fit_transform(df_appended['X4'])
df_appended['X5'] = le.fit_transform(df_appended['X5'])
df_appended['X6'] = le.fit_transform(df_appended['X6'])
df_appended['X8'] = le.fit_transform(df_appended['X8'])
df_appended
```

Out[13]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X3
0	0	130.810000	37	23	20	0	3	27	9	14	...	0	0	1	0	0	0	
1	6	88.530000	37	21	22	4	3	31	11	14	...	1	0	0	0	0	0	
2	7	76.260000	24	24	38	2	3	30	9	23	...	0	0	0	0	0	0	
3	9	80.620000	24	21	38	5	3	30	11	4	...	0	0	0	0	0	0	
4	13	78.020000	24	23	38	5	3	14	3	13	...	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	8410	100.669318	9	9	19	5	3	1	9	4	...	0	0	0	0	0	0	
4205	8411	100.669318	46	1	9	3	3	1	9	24	...	0	1	0	0	0	0	
4206	8413	100.669318	51	23	19	5	3	1	3	22	...	0	0	0	0	0	0	
4207	8414	100.669318	10	23	19	0	3	1	2	16	...	0	0	1	0	0	0	
4208	8416	100.669318	46	1	9	2	3	1	6	17	...	1	0	0	0	0	0	

8418 rows × 378 columns

In [14]:
```python
# remove unnecessary column ID and target column 'y'
PCA_df1 = df_appended
PCA_df1.isnull().sum()
PCA_df1 = PCA_df1.drop(['ID','y'],axis = 1)
PCA_df1
```

Out[14]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X382	X383
0	37	23	20	0	3	27	9	14	0	0	...	0	0	1	0	0	0	0	0
1	37	21	22	4	3	31	11	14	0	0	...	1	0	0	0	0	0	0	0
2	24	24	38	2	3	30	9	23	0	0	...	0	0	0	0	0	0	1	0
3	24	21	38	5	3	30	11	4	0	0	...	0	0	0	0	0	0	0	0
4	24	23	38	5	3	14	3	13	0	0	...	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
4204	9	9	19	5	3	1	9	4	0	0	...	0	0	0	0	0	0	0	0
4205	46	1	9	3	3	1	9	24	0	0	...	0	1	0	0	0	0	0	0
4206	51	23	19	5	3	1	3	22	0	0	...	0	0	0	0	0	0	0	0
4207	10	23	19	0	3	1	2	16	0	0	...	0	0	1	0	0	0	0	0
4208	46	1	9	2	3	1	6	17	0	0	...	1	0	0	0	0	0	0	0

8418 rows × 376 columns

In [15]:
```python
# split the data with 80:20 ratio
X = PCA_df1.loc[:, PCA_df1.columns]
Y = df_appended['y']
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(6734, 376)
(1684, 376)
(6734,)
(1684,)
```

Perform dimensionality reduction

In [16]:
```python
# Perform dimensionality reduction - we are using PCA
# n_components : Number of components to keep.
# if n_components is not set all components are kept.
from sklearn.decomposition import PCA

sklearn_pca = PCA(n_components=0.95)
sklearn_pca.fit(X_train)

X_train_transformed = sklearn_pca.transform(X_train)
X_test_transformed =sklearn_pca.transform(X_test)
print(X_train.shape)
print(X_train_transformed.shape)
print(X_test.shape)
print(X_test_transformed.shape)
```

```
(6734, 376)
(6734, 6)
(1684, 376)
(1684, 6)
```

Predict your test_df values using XGBoost

In [19]:
```python
# Predict your test_df values using XGBoost
# XGBOOST will give the lowest RMSE

from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
import math

xgbreg = XGBRegressor()

xgbreg.fit(X_train_transformed,Y_train)
Y_predict_XGBoost = xgbreg.predict(X_test_transformed)
Y_predict_XGBoost
RMSE_XGBoost = math.sqrt(mean_squared_error(Y_predict_XGBoost,Y_test))
print("Predicted test_df values using XGBoost :")
print(Y_predict_XGBoost)
print('\n')
print('RMSE of XGBoost Regression : ',RMSE_XGBoost)
```

```
Predicted test_df values using XGBoost :
[101.94168  102.145424  95.77367  ... 101.34008   97.34563   94.999245]


RMSE of XGBoost Regression :  8.03571006298545
```

We will predict test_df values from some other Models aslo

Model Building - Regression

1. We will try Ridge Regression

2. We will try Lasso Regression

3. We will try ElasticNet regression

In [26]:
```
# Ridge Regression :
# Ridge Regression (L2) is used when there is a problem of multicollinearity.
# By adding a degree of bias to the regression estimates, ridge regression reduces th

from sklearn.metrics import mean_squared_error
from sklearn import metrics
from sklearn.linear_model import Ridge
import math

ridgeReg = Ridge(alpha=0.001, normalize=True)
ridgeReg.fit(X_train_transformed,Y_train)
mse_ridge1 = metrics.mean_squared_error(Y_train, ridgeReg.predict(X_train_transformed
sqrt_mse_ridge1 = math.sqrt(mse_ridge1)
print('Square root of MSE Ridge 1 : ',sqrt_mse_ridge1)

mse_ridge2 = metrics.mean_squared_error(Y_test, ridgeReg.predict(X_test_transformed))
sqrt_mse_ridge2 = math.sqrt(mse_ridge2)
print('Square root of MSE Ridge 2 : ',sqrt_mse_ridge2)
Y_predict_ridge = ridgeReg.predict(X_test_transformed)

print('R2 Value/Coefficient of Determination: ',ridgeReg.score(X_test_transformed , Y

RMSE_ridge = math.sqrt(mean_squared_error(Y_predict_ridge,Y_test))
print('RMSE of Ridge Regression : ',RMSE_ridge)
```

```
Square root of MSE Ridge 1 :  8.98009800449775
Square root of MSE Ridge 2 :  8.401881121922498
R2 Value/Coefficient of Determination:  0.01658826026368343
RMSE of Ridge Regression :  8.401881121922498
```

In [25]:
```
# Lasso Regression :
# Lasso Regression (L1) is similar to ridge, but it also performs feature selection.

from sklearn.linear_model import Lasso

lassoreg = Lasso(alpha=0.001, normalize=True)
lassoreg.fit(X_train_transformed,Y_train)

mse_lassoreg1 = metrics.mean_squared_error(Y_train, lassoreg.predict(X_train_transfor
sqrt_mse_lassoreg1 = math.sqrt(mse_lassoreg1)

print('Square root of MSE Lassoreg 1 : ',sqrt_mse_lassoreg1)

mse_lassoreg2 = metrics.mean_squared_error(Y_test, lassoreg.predict(X_test_transforme
sqrt_mse_lassoreg2 = math.sqrt(mse_lassoreg2)
Y_predict_lasso = lassoreg.predict(X_test_transformed)

print('Square root of MSE Lassoreg 2 : ',sqrt_mse_lassoreg2)

print('R2 Value/Coefficient of Determination: ',lassoreg.score(X_test_transformed , Y

RMSE_lasso = math.sqrt(mean_squared_error(Y_predict_lasso,Y_test))
print('RMSE of Lasso Regression : ',RMSE_lasso)
```

```
Square root of MSE Lassoreg 1 :  8.982184631817486
Square root of MSE Lassoreg 2 :  8.398928318540477
R2 Value/Coefficient of Determination:  0.017279370076204503
RMSE of Lasso Regression :  8.398928318540477
```

In [24]:

```
# ElasticNet Regression :
# ElasticNet Regression combines the strength of lasso and ridge regression
# If you are not sure whether to use lasso or ridge, use ElasticNet

from sklearn.linear_model import ElasticNet

elasticnetreg = ElasticNet(alpha=0.001, normalize=True)
elasticnetreg.fit(X_train_transformed,Y_train)

mse_elasticnetreg1 = metrics.mean_squared_error(Y_train, elasticnetreg.predict(X_trai
sqrt_mse_elasticnetreg1 = math.sqrt(mse_elasticnetreg1)

print('Square root of MSE Elasticnetreg 1 : ',sqrt_mse_elasticnetreg1)

mse_elasticnetreg2 = metrics.mean_squared_error(Y_test, elasticnetreg.predict(X_test_
sqrt_mse_elasticnetreg2 = math.sqrt(mse_elasticnetreg2)
Y_predict_elasticnet = elasticnetreg.predict(X_test_transformed)
print('Square root of MSE Elasticnetreg 2 : ',sqrt_mse_elasticnetreg2)

print('R2 Value/Coefficient of Determination: ',elasticnetreg.score(X_test_transforme

RMSE_elasticnet = math.sqrt(mean_squared_error(Y_predict_elasticnet,Y_test))
print('RMSE of ElasticNet Regression : ',RMSE_elasticnet)
```

```
Square root of MSE Elasticnetreg 1 :  9.04175831727264
Square root of MSE Elasticnetreg 2 :  8.45150514394438
R2 Value/Coefficient of Determination:  0.004937307027778393
RMSE of ElasticNet Regression :  8.45150514394438
```

Summary :

RMSE of Ridge Regression : 8.401881121922498

RMSE of Lasso Regression : 8.398928318540479

RMSE of ElasticNet Regression : 8.45150514394438

RMSE of XGBoost Regression : 7.718938076728523

The aboove output shows XGBoost Regression gives slightly better result than the other regression model.

NOTE : Lower values of RMSE indicate better fit.

In [ ]: