Mercedes-Benz Greener Manufacturing Challenge

By Ifalore Simeon

- `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.

In [1]:
```
# Importing dependencies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import MinMaxScaler

# Ignoring warnings
import warnings
warnings.filterwarnings("ignore", message=r"Passing", category=FutureWarning)
```

In [2]:
```
train_df = pd.read_csv('Benz_train.csv')
test_df = pd.read_csv('Benz_test.csv')
```

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

Out[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	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	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	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	0	0

5 rows × 378 columns

In [4]:
```
test_df.shape, test_df.columns
```

Out[4]:
```
((4209, 377),
 Index(['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
        ...
        'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
        'X385'],
       dtype='object', length=377))
```

In [5]:
```
train_df.shape, train_df.columns
```

Out[5]:
```
((4209, 378),
 Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
        ...
        'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
        'X385'],
       dtype='object', length=378))
```

In [6]:
```
train_df.describe()
```

Out[6]:

	ID	y	X10	X11	X12	X13	X14	X15	X16	X17	...	
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	...	4209
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.000475	0.002613	0.007603	...	0
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.021796	0.051061	0.086872	...	0
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	...	1
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1

8 rows × 370 columns

In [7]:
```
test_df.describe()
```

Out[7]:

	ID	X10	X11	X12	X13	X14	X15	X16	X17	X18	...
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	...
mean	4211.039202	0.019007	0.000238	0.074364	0.061060	0.427893	0.000713	0.002613	0.008791	0.010216	...
std	2423.078926	0.136565	0.015414	0.262394	0.239468	0.494832	0.026691	0.051061	0.093357	0.100570	...
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	2115.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	4202.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	6310.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	...
max	8416.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...

8 rows × 369 columns

In [8]:
```
# Checking for any null values in train and test data set
print(train_df.isnull().values.any())
print(test_df.isnull().values.any())
```

```
False
False
```

If for any column(s), the variance is equal to zero, then you need to remove those variable(s).

In [9]:
```
# Find out the variance is equal to zero for any columns
(train_df.var(numeric_only=True) == 0)
```

Out[9]:
```
ID      False
y       False
X10     False
X11      True
X12     False
        ...  
X380    False
X382    False
X383    False
X384    False
X385    False
Length: 370, dtype: bool
```

In [10]:
```
(train_df.var(numeric_only = True) == 0).values.sum()
```

Out[10]:
```
12
```

In [11]:
```
variance_with_zero = train_df.var(numeric_only = True)[train_df.var(numeric_only = True)==0].index.values
variance_with_zero
```

Out[11]:
```
array(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290',
       'X293', 'X297', 'X330', 'X347'], dtype=object)
```

In [12]:
```
# Drop zero variance variables for train and test
train_df = train_df.drop(variance_with_zero, axis=1)
train_df.shape
```

Out[12]:
```
(4209, 366)
```

In [13]:
```
# Drop zero variance variables for train and test
test_df = test_df.drop(variance_with_zero, axis=1)
test_df.shape
```
Out[13]:
```
(4209, 365)
```

In [14]:
```
test_df.shape
```
Out[14]:
```
(4209, 365)
```

In [15]:
```
 # Since ID column is irrelevant for our prediction hence we drop this column
train_df = train_df.drop(['ID'], axis=1)
train_df.shape
```
Out[15]:
```
(4209, 365)
```

In [16]:
```

#  drop ID for test_df also

test_df = test_df.drop(['ID'], axis = 1)
test_df.shape
```
Out[16]:
```
(4209, 364)
```

In [17]:
```
train_df.shape
```
Out[17]:
```
(4209, 365)
```

Checking for null and unique values for test and train sets.

In [18]:
```
train_df.isnull().any().value_counts()
```
Out[18]:
```
False    365
dtype: int64
```

In [19]:
```
test_df.isnull().any().value_counts()
```
Out[19]:
```
False    364
dtype: int64
```

In [20]:
```
# Find unique records
train_df.nunique()
```
Out[20]:
```
y       2545
X0        47
X1        27
X2        44
X3         7
        ...
X380       2
X382       2
X383       2
X384       2
X385       2
Length: 365, dtype: int64
```

In [21]:
```
# Looking at the first 20 features columns we can see that "X0" to "X8" are object data types
train_df.iloc[:, 1:21]
```
Out[21]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22
0	k	v	at	a	d	u	j	o	0	0	1	0	0	0	0	1	0	0	1	0
1	k	t	av	e	d	y	l	o	0	0	0	0	0	0	0	1	0	0	0	0
2	az	w	n	c	d	x	j	x	0	0	0	0	0	0	1	0	0	0	0	0
3	az	t	n	f	d	x	l	e	0	0	0	0	0	0	0	0	0	0	0	0
4	az	v	n	f	d	h	d	n	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	ak	s	as	c	d	aa	d	q	0	0	0	1	0	0	0	0	0	0	0	0
4205	j	o	t	d	d	aa	h	h	0	0	0	0	0	0	0	0	0	0	0	0
4206	ak	v	r	a	d	aa	g	e	0	1	1	0	0	0	0	0	0	0	0	0
4207	al	r	e	f	d	aa	l	u	0	0	0	1	0	0	0	0	0	0	0	0
4208	z	r	ae	c	d	aa	g	w	0	0	0	0	0	0	0	0	0	1	0	0

4209 rows × 20 columns

In [22]:
```
# Looking at the first 20 features columns we can see that "X0" to "X8" are object data types
test_df.iloc[:, 1:21]
```

Out[22]:

	X1	X2	X3	X4	X5	X6	X8	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23
0	v	n	f	d	t	a	w	0	0	0	0	0	0	0	0	0	0	0	0	0
1	b	ai	a	d	b	g	y	0	0	0	0	0	0	0	0	1	0	0	0	0
2	v	as	f	d	a	j	j	0	0	0	1	0	0	0	0	0	0	0	0	0
3	l	n	f	d	z	l	n	0	0	0	0	0	0	0	0	0	0	0	0	0
4	s	as	c	d	y	i	m	0	0	0	1	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	h	as	f	d	aa	j	e	0	0	0	1	0	0	0	0	0	0	0	0	0
4205	aa	ai	d	d	aa	j	y	0	0	0	0	0	0	0	0	1	0	0	0	0
4206	v	as	f	d	aa	d	w	0	0	0	1	0	0	0	0	0	0	0	0	0
4207	v	as	a	d	aa	c	q	0	0	1	1	0	0	0	0	0	0	0	0	0
4208	aa	ai	c	d	aa	g	r	0	0	0	0	0	0	0	0	1	0	0	0	0

4209 rows × 20 columns

Filter out the columns having object datatype

In [23]:
```
object_datatypes = train_df.select_dtypes(include=[object])
object_datatypes
```

Out[23]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	k	v	at	a	d	u	j	o
1	k	t	av	e	d	y	l	o
2	az	w	n	c	d	x	j	x
3	az	t	n	f	d	x	l	e
4	az	v	n	f	d	h	d	n
...	...	...	...	...	...	...	...	...
4204	ak	s	as	c	d	aa	d	q
4205	j	o	t	d	d	aa	h	h
4206	ak	v	r	a	d	aa	g	e
4207	al	r	e	f	d	aa	l	u
4208	z	r	ae	c	d	aa	g	w

4209 rows × 8 columns

In [24]:
```
object_datatype_columns = object_datatypes.columns
object_datatype_columns
```

Out[24]:
```
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='object')
```

In [25]:
```
from sklearn import preprocessing # Import Label Encoder
# Initialize Label Encoder object
label_encoder = preprocessing.LabelEncoder()
```

In [26]:
```
# Encode and transform object data to interger
train_df['X0'] = label_encoder.fit_transform(train_df['X0'])
train_df['X1'] = label_encoder.fit_transform(train_df['X1'])
train_df['X2'] = label_encoder.fit_transform(train_df['X2'])
train_df['X3'] = label_encoder.fit_transform(train_df['X3'])
train_df['X4'] = label_encoder.fit_transform(train_df['X4'])
train_df['X5'] = label_encoder.fit_transform(train_df['X5'])
train_df['X6'] = label_encoder.fit_transform(train_df['X6'])
train_df['X8'] = label_encoder.fit_transform(train_df['X8'])
```

In [27]:
```
train_df.head()
```

Out[27]:

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	130.81	32	23	17	0	3	24	9	14	0	...	0	0	1	0	0	0	0	0	0	0
1	88.53	32	21	19	4	3	28	11	14	0	...	1	0	0	0	0	0	0	0	0	0
2	76.26	20	24	34	2	3	27	9	23	0	...	0	0	0	0	0	0	1	0	0	0
3	80.62	20	21	34	5	3	27	11	4	0	...	0	0	0	0	0	0	0	0	0	0
4	78.02	20	23	34	5	3	12	3	13	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 365 columns

In [28]:
```
test_df.head()
```

Out[28]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	az	v	n	f	d	t	a	w	0	0	...	0	0	0	1	0	0	0	0	0	0
1	t	b	ai	a	d	b	g	y	0	0	...	0	0	1	0	0	0	0	0	0	0
2	az	v	as	f	d	a	j	j	0	0	...	0	0	0	1	0	0	0	0	0	0
3	az	l	n	f	d	z	l	n	0	0	...	0	0	0	1	0	0	0	0	0	0
4	w	s	as	c	d	y	i	m	0	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 364 columns

In [29]:
```
test_object_datatypes = test_df.select_dtypes(include=[object])
test_object_datatypes
```

Out[29]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	az	v	n	f	d	t	a	w
1	t	b	ai	a	d	b	g	y
2	az	v	as	f	d	a	j	j
3	az	l	n	f	d	z	l	n
4	w	s	as	c	d	y	i	m
...	...	...	...	...	...	...	...	...
4204	aj	h	as	f	d	aa	j	e
4205	t	aa	ai	d	d	aa	j	y
4206	y	v	as	f	d	aa	d	w
4207	ak	v	as	a	d	aa	c	q
4208	t	aa	ai	c	d	aa	g	r

4209 rows × 8 columns

In [30]:
```
# Encode and transform object data to interger
test_df['X0'] = label_encoder.fit_transform(test_df['X0'])
test_df['X1'] = label_encoder.fit_transform(test_df['X1'])
test_df['X2'] = label_encoder.fit_transform(test_df['X2'])
test_df['X3'] = label_encoder.fit_transform(test_df['X3'])
test_df['X4'] = label_encoder.fit_transform(test_df['X4'])
test_df['X5'] = label_encoder.fit_transform(test_df['X5'])
test_df['X6'] = label_encoder.fit_transform(test_df['X6'])
test_df['X8'] = label_encoder.fit_transform(test_df['X8'])
```

In [31]:
```
test_df.head()
```

Out[31]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	21	23	34	5	3	26	0	22	0	0	...	0	0	0	1	0	0	0	0	0	0
1	42	3	8	0	3	9	6	24	0	0	...	0	0	1	0	0	0	0	0	0	0
2	21	23	17	5	3	0	9	9	0	0	...	0	0	0	1	0	0	0	0	0	0
3	21	13	34	5	3	31	11	13	0	0	...	0	0	0	1	0	0	0	0	0	0
4	45	20	17	2	3	30	8	12	0	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 364 columns

In [32]:
```
train_df.shape, test_df.shape
```

Out[32]:
```
((4209, 365), (4209, 364))
```

Performing dimensionality reduction (PCA)

In [33]:
```
# seperating y from the train dataFrame before scaling the features
X_train = train_df.iloc[:,1:]
y_train = train_df.iloc[:,0]

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

In [34]:
```
X_train_scaled.shape
```

Out[34]:
```
(4209, 364)
```

In [35]:
```
y_train
```

Out[35]:
```
0       130.81
1        88.53
2        76.26
3        80.62
4        78.02
         ...  
4204    107.39
4205    108.77
4206    109.22
4207     87.48
4208    110.85
Name: y, Length: 4209, dtype: float64
```

In [36]:
```
# applying the PCA model
from sklearn.decomposition import PCA
```

In [37]:
```
# PCA with 95% of x_train
sklearn_pca = PCA(n_components=0.95, random_state = 420)
```

In [38]:
```
pca_x_train_transformed = sklearn_pca.fit_transform(X_train_scaled)
```

In [39]:
```
print(pca_x_train_transformed.shape)
```

```
(4209, 76)
```

Dimensionality reduction for Test DataFrame.

In [40]:
```
# Scaling the data

scaling = MinMaxScaler()
test_scaled = scaling.fit_transform(test_df)
```

In [41]:
```
test_scaled.shape
```

Out[41]:
```
(4209, 364)
```

In [42]:
```
# Performing Dimensionality reduction with PCA with 95% for test_df

pca_x_test_transformed = sklearn_pca.fit_transform(test_scaled)
```

In [43]:
```

print(pca_x_test_transformed.shape)
```

```
(4209, 76)
```

In [44]:
```
pca_x_test_transformed.shape, pca_x_train_transformed.shape
```

Out[44]:
```
((4209, 76), (4209, 76))
```

Performing XGBOOST

In [45]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error as MSE

# fitting the model

model = xgb.XGBRegressor(objective="reg:linear",learning_rate=0.1)
model.fit(pca_x_train_transformed, y_train)
y_pred = model.predict(pca_x_test_transformed)
y_pred
```

```
[11:15:17] WARNING: c:\ci\xgboost-split_1638290375667\work\src\objective\regression_obj.cu:188: reg:linear is now d
eprecated in favor of reg:squarederror.
```

Out[45]:
```
array([ 79.616974,  91.12512 ,  82.31594 , ...,  98.8274  , 111.627235,
        93.72134 ], dtype=float32)
```

In [46]:

```
y_pred.shape, pca_x_test_transformed.shape
```

Out[46]:

```
((4209,), (4209, 76))
```

In [47]:

```
rmse = np.sqrt(MSE(y_train, y_pred))
print("RMSE : %f" %(rmse))
```

```
RMSE : 14.892768
```