Mercedes Benz Greener Manufacturing

Introduction

The first luxury car maker Benz Patented 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.

Overview

In an automobile industry, there is a testing department in which every vehicle that comes out from production manufacturing. Safety and reliable testing is a crucial part in the automobile manufacturing process. The Mercedes -Benz automobile industry every day manufactures a huge rate in producing vehicles and send to the testing department which is a 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 to the company and generating carbon dioxide, a polluting greenhouse gas.

Aim

The main objective of this project is to optimize/reduce the testing time in 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.

Preliminary tasks

Let us now import the required libraries and datasets.

In [144…

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\ruben\anaconda3\lib\site-packages (1.5.
1)
Requirement already satisfied: scipy in c:\users\ruben\anaconda3\lib\site-packages (from x
gboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\ruben\anaconda3\lib\site-packages (from x
gboost) (1.20.3)
```

In [145…

```
#Load the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
%matplotlib inline
```

In [146…

```python
#Load the data
train = pd.read_csv("C:/Users/ruben/OneDrive/Documents/Python_DS/Projects/Mercedes Benz G
test = pd.read_csv("C:/Users/ruben/OneDrive/Documents/Python_DS/Projects/Mercedes Benz Gr
```

In [147…

```python
train.head(10) #Top 10 rows of train set
```

Out[147…

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	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
3	9	80.62	az	t	n	f	d	x	l	e	...	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
5	18	92.93	t	b	e	c	d	g	h	s	...	0	0	1	0	0	0	0
6	24	128.76	al	r	e	f	d	f	h	s	...	0	0	0	0	0	0	0
7	25	91.91	o	l	as	f	d	f	j	a	...	0	0	0	0	0	0	0
8	27	108.67	w	s	as	e	d	f	i	h	...	1	0	0	0	0	0	0
9	30	126.99	j	b	aq	c	d	f	a	e	...	0	0	1	0	0	0	0

10 rows × 378 columns

In [148…

```python
train.shape
```

Out[148…

```
(4209, 378)
```

In [149…

```python
test.head(10) #Top 10 rows of test set
```

Out[149…

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0
5	8	y	aa	ai	e	d	x	g	s	0	...	1	0	0	0	0	0	0
6	10	x	b	ae	d	d	x	d	y	0	...	0	0	0	0	0	1	0
7	11	f	s	ae	c	d	h	d	a	0	...	0	0	1	0	0	0	0
8	12	ap	l	s	c	d	h	j	n	0	...	0	0	0	0	0	0	0
9	14	o	v	as	f	d	g	f	v	0	...	0	0	0	0	0	0	0

10 rows × 377 columns

Data Preparation

Let us prepare the data which is fit for modelling purposes. We need to ensure the data is indexed correctly, and that there is no missing values. Let us now check the statistical info of the dataset for more info

In [150…

```
train.describe()
```

Out[150…

	ID	y	X10	X11	X12	X13	X14	
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.0
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.0
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.0
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.0
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.0
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.0
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.0
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.0

8 rows × 370 columns

In [151…

```
train.info() #Structure of the dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 378 entries, ID to X385
dtypes: float64(1), int64(369), object(8)
memory usage: 12.1+ MB
```

In [152…

```
#Finding the number of null values
print("Number of NaN values in train dataset is:",len(train[train.isna().any(axis=1)]))
print("Number of NaN values in test dataset is:",len(test[test.isna().any(axis=1)]))
```

```
Number of NaN values in train dataset is: 0
Number of NaN values in test dataset is: 0
```

In [153…

```
#Finding if there are any duplicates w.r.t ID column
print("Number of duplicated values in train dataset is:",train['ID'].duplicated().sum())
print("Number of duplicated values in test dataset is:",test['ID'].duplicated().sum())
```

```
Number of duplicated values in train dataset is: 0
Number of duplicated values in test dataset is: 0
```

In [154…

```
len(train.select_dtypes(include="int").columns) #number of Numerical columns
```

Out[154…

```
369
```

In [155…

```
len(train.select_dtypes(include="object").columns) #number of categorical columns
```

Out[155…

```
8
```

In [156…

```
train = train.drop('ID',axis =1)
```

In [157…

```
test = test.drop('ID',axis =1)
```

Loading [MathJax]/extensions/Safe.js

There are no missing data or duplicated data, hence we can proceeed further with the analysis on the 369 numerical columns and 8 columns.

Exploratory Data Analysis (EDA)

The first step is to split the dataset into the feature and target dataframes. proceed with some visualizations to give some proper insight into the dataset.This provides a statistical characteristic and behavior of distribution of the data and also provide an insights in data.

In [158…

```
#Feature and target selection of variables
X_train = train.drop('y',axis=1)
y_train = train['y']
```

In [159…

```
X_train.head()
```

Out[159…

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X382	X:
0	k	v	at	a	d	u	j	o	0	0	...	0	0	1	0	0	0	0	
1	k	t	av	e	d	y	l	o	0	0	...	1	0	0	0	0	0	0	
2	az	w	n	c	d	x	j	x	0	0	...	0	0	0	0	0	0	1	
3	az	t	n	f	d	x	l	e	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	

5 rows × 376 columns

In [160…

```
y_train.head()
```

Out[160…

```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
```

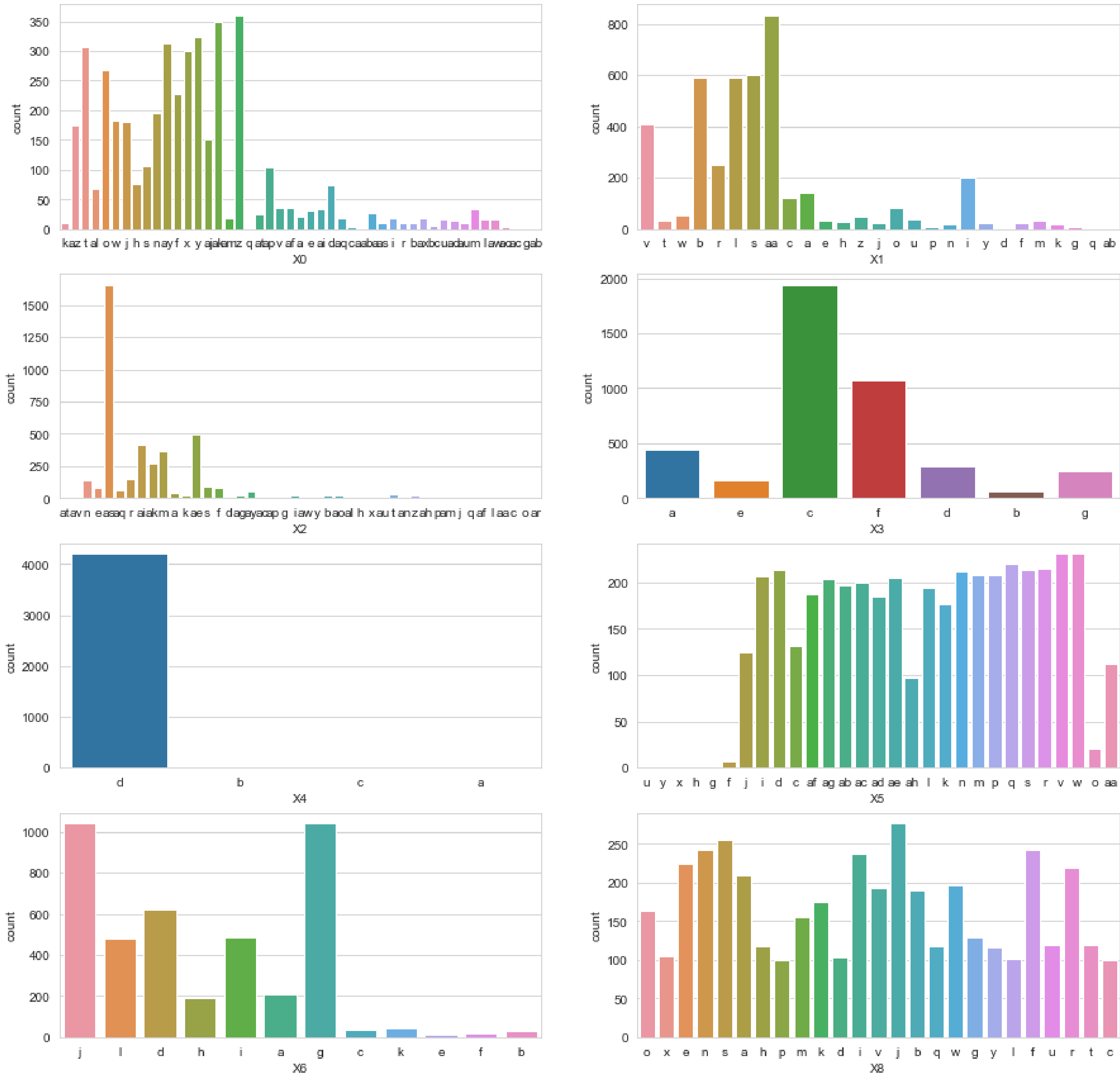
train.select_dtypes(include="object").columns #Selecting which is the categorical columns

Analyzing and visualization of Categorical Variables

Bar plots for all the categorical variables

In [161…

```
#For all categorical feature
plt.figure(figsize = (15,15),)
sns.set_style('whitegrid')
for i in range(0,8):
    plt.subplot(4,2,i+1)
    sns.countplot(x=X_train.iloc[:,i], data=X_train)
plt.show()
```



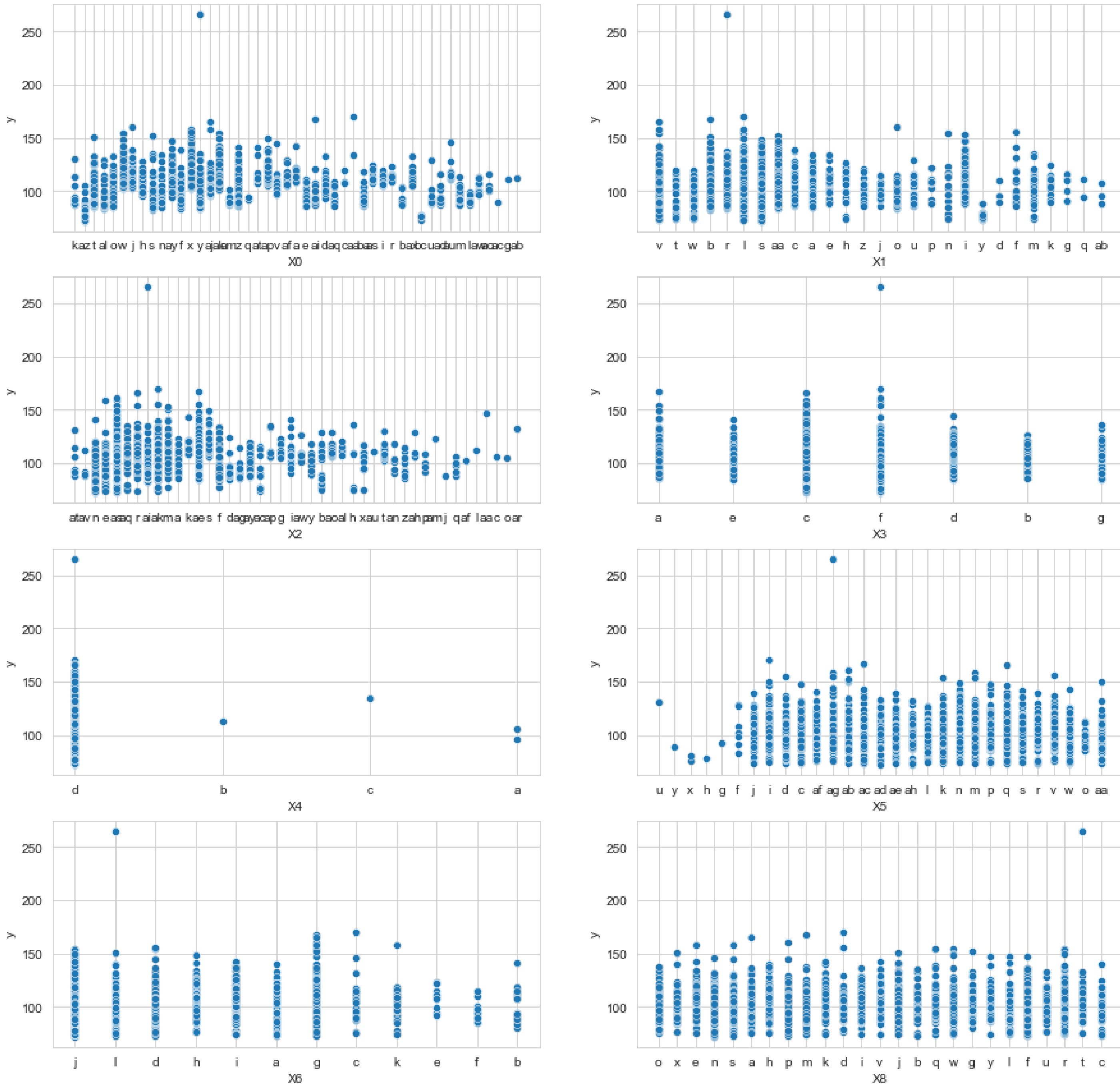
From the above plot it can observe that X4 features have less variance in it.

Variation of different variables with time

In [162…

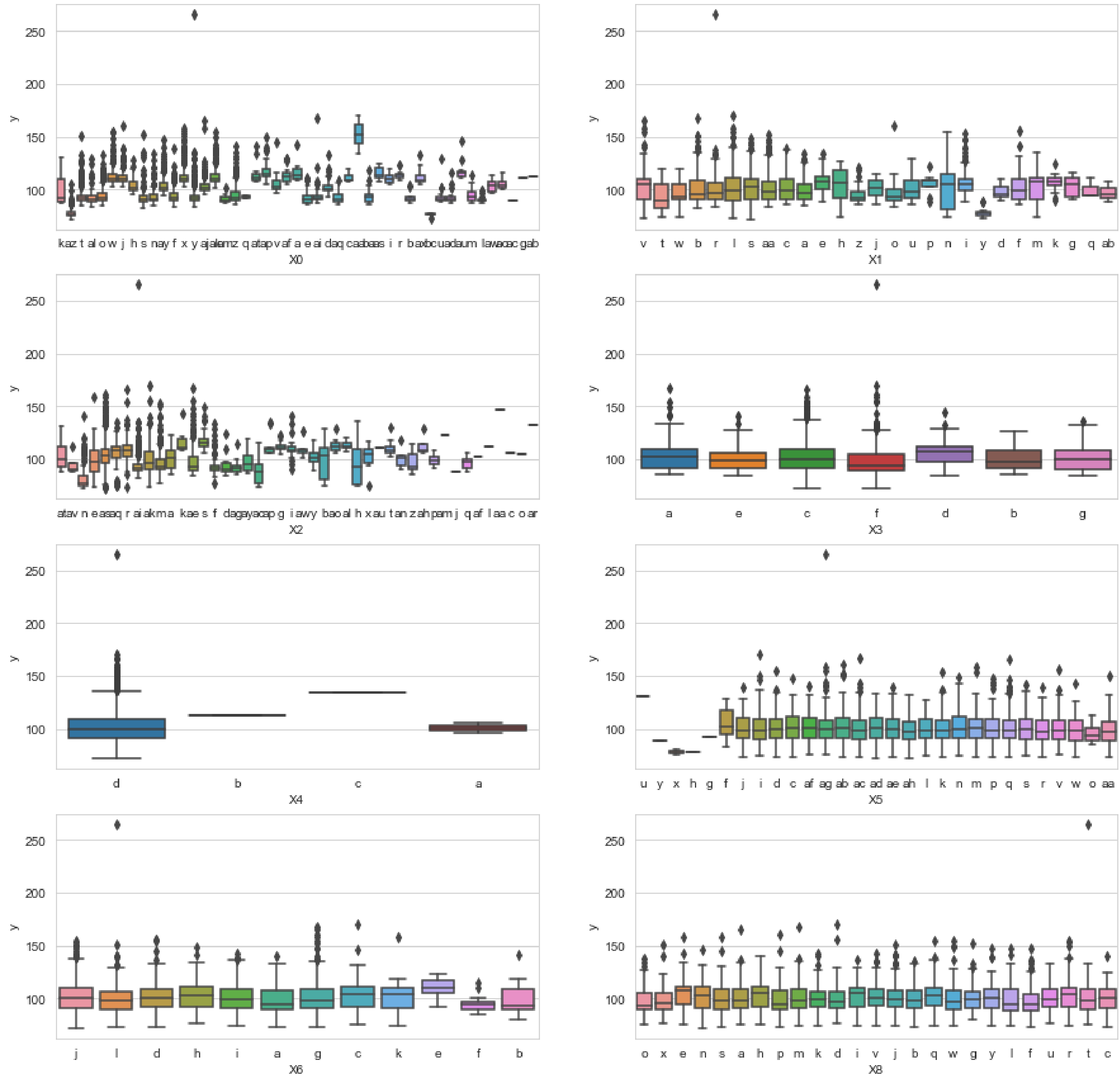
```
plt.figure(figsize = (15,15),)
sns.set_style('whitegrid')
for i in range(0,8):
    plt.subplot(4,2,i+1)
    sns.scatterplot(x=X_train.iloc[:,i], y= y_train, data=X_train)
plt.show()
```

Loading [MathJax]/extensions/Safe.js



In [163…

```
plt.figure(figsize = (15,15),)
sns.set_style('whitegrid')
for i in range(0,8):
    plt.subplot(4,2,i+1)
    sns.boxplot(x=X_train.iloc[:,i], y= y_train, data=X_train)
plt.show()
```



The observations made from the following plots:

- In the x0 plot, “y” category has a data point present faraway from normally distributed of that category and can be considered as outlier.
- In the x1 plot, “r” category has a data point present faraway from normally distributed of that category and can be considered as an outlier.
- In the x2 plot, “ai” category has a data point present faraway from normally distributed of that category and can be considered as an outlier.
- In the x3 plot, most of the categories lies in the range of 85 to 120 values of output variable. But in category “f” has a data point present faraway from normally distributed of that category and can be considered as an outlier.
- In the x4 plot, “d” category distributed in the range of 90 to 110 range of values. The category “b” and “c” are present with just few in numbers and mostly at the point of 120 and 130 output value.
- The x5 plot, represent most of the categorical values distributed at the range of 85 to 120 output values. This features shows that most of the features occurs and uniformly distributed

and it can observe some of the features which are present in few numbers.

- The x6 plot, which represent the most of the category's PDF curve lies under the range of 75 to 125 output y variable. We can observe that category "i" is highly skewed and shows that this category have an outlier with respect to output variable y.
- The x8 plot show all the categorical values are present in uniformly and almost PDF curve lies in the range of 75 to 125 values of output y variable. We can observe that category "t" is highly skewed and shows that this category have an outlier with respect to output variable y.

Data Preprocessing

If for any column(s), the variance is equal to zero, then you need to remove those variable(s). In this step,we apply label encoding.

In [164…

```
X_test = test.copy()
usable_columns = list(set(X_train.columns))
```

In [165…

```
test.head()
```

Out[165…

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X382	X:
0	az	v	n	f	d	t	a	w	0	0	...	0	0	0	1	0	0	0	
1	t	b	ai	a	d	b	g	y	0	0	...	0	0	1	0	0	0	0	
2	az	v	as	f	d	a	j	j	0	0	...	0	0	0	1	0	0	0	
3	az	l	n	f	d	z	l	n	0	0	...	0	0	0	1	0	0	0	
4	w	s	as	c	d	y	i	m	0	0	...	1	0	0	0	0	0	0	

5 rows × 376 columns

In [166…

```
#If for any column(s), the variance is equal to zero, then you need to remove those varia
for col in usable_columns:
    cardinality = len(np.unique(X_train[col]))
    if cardinality == 1:
        X_train.drop(col, axis=1) # Column with only one, value is useless so we drop it
        X_test.drop(col, axis=1)
    if cardinality > 2: # Column is categorical
        mapper = lambda x: sum([ord(digit) for digit in x])
        X_train[col] = X_train[col].apply(mapper)
        X_test[col] = X_test[col].apply(mapper)
X_train.head()
```

Out[166…

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X3
0	107	118	213	97	100	117	106	111	0	0	...	0	0	1	0	0	0	
1	107	116	215	101	100	121	108	111	0	0	...	1	0	0	0	0	0	
2	219	119	110	99	100	120	106	120	0	0	...	0	0	0	0	0	0	
3	219	116	110	102	100	120	108	101	0	0	...	0	0	0	0	0	0	
4	219	118	110	102	100	104	100	110	0	0	...	0	0	0	0	0	0	

5 rows × 376 columns

```
test.head()
```

Out[167…

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X382	X:
0	az	v	n	f	d	t	a	w	0	0	...	0	0	0	1	0	0	0	
1	t	b	ai	a	d	b	g	y	0	0	...	0	0	1	0	0	0	0	
2	az	v	as	f	d	a	j	j	0	0	...	0	0	0	1	0	0	0	
3	az	l	n	f	d	z	l	n	0	0	...	0	0	0	1	0	0	0	
4	w	s	as	c	d	y	i	m	0	0	...	1	0	0	0	0	0	0	

5 rows × 376 columns

Perform Principal Component Analysis (PCA)

Principal component analysis, or PCA, is a statistical procedure that allows you to summarize the information content in large data tables by means of a smaller set of “summary indices” that can be more easily visualized and analyzed. The goal is to extract the important information from the data and to express this information as a set of summary indices called **principal components** . Singular Value Decomposition or SVD is a computational method used to calculate principal components of a dataset. Linear dimensionality reduction using SVD of the data projects it to a lower dimensional space.

In [168…

```
#Dimensionality Reduction
n_comp = 12
pca = PCA(n_components=n_comp, random_state=420)
pca_train = pca.fit_transform(X_train)
pca_test = pca.transform(X_test)
```

In [169…

```
pca_train,pca_test
```

Out[169…

```
(array([[-49.08156207,  -4.90948084, -17.25085325, ...,   1.65808085,
           0.93316413,   1.67767261],
        [-48.94680383,  -7.22674339, -13.7631947 , ...,  -0.21429673,
           0.10928689,   0.44858868],
        [ 92.62761708,  31.9940341 , -26.17503456, ...,  -0.62195512,
           2.92579792,  -0.52629181],
        ...,
        [ 89.47970814,  20.44554421,  48.11999819, ...,  -1.27199613,
          -0.28730646,   2.00870035],
        [ 96.97110845,  31.50977186,  49.20059282, ...,   0.14362369,
          -0.98010171,   0.99232435],
        [-17.21024322, -14.22166025,  55.38091289, ...,  -0.28904254,
          -0.31644227,   0.6915868 ]]),
 array([[ 9.22615149e+01,  3.29260839e+01, -3.01130736e+01, ...,
         -4.11406384e-01,  3.62106392e+00, -1.20778172e+00],
        [-3.48622379e+01,  6.87132606e+00, -3.74760829e+01, ...,
          6.09253697e-01, -6.95870836e-01, -4.24945581e-01],
        [ 4.36560426e+01, -5.05939489e+01, -6.10591086e+01, ...,
         -3.20458181e-01,  2.60144802e+00, -1.53707632e+00],
        ...,
        [-2.52437784e+01, -2.63794193e+01,  5.40742341e+01, ...,
          6.03516083e-01,  2.60866858e-02,  3.68490704e-02],
        [ 4.53823778e+01, -6.38062446e+01,  3.58666036e+01, ...,
         -9.15187206e-01, -6.72291446e-01,  5.15293180e-01],
        [-4.23807477e+01, -2.52862351e+01,  6.10815522e+01, ...,
         -2.98836314e-01, -9.77070805e-01,  5.34362801e-02]]))
```

Loading [MathJax]/extensions/Safe.js

Now the data is ready for modelling purposes

Data Modelling

Since there is already a test set, usually it isnt necessary to use the train test split for the dataset. However to improve model accuracy, it is better to have another test set from the train set i.e. validation set. This is helpful as this ensures a more accurate calculation of model performance.

In [170…

```
#Splitting the train test into Train and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(pca_train, y_train, test_size=0.2,
```

Here we use xgboost regression which is boosting technique in ensemble to reduce bias while training the model.We begin by adding the datasets into the DMatrix or data matrix, which is an internal data structure that is used by XGBoost, which is optimized for both memory efficiency and training speed.

In [171…

```
#Creating D matrices
d_train = xgb.DMatrix(X_train, label=y_train)
d_valid = xgb.DMatrix(X_valid, label=y_valid)
#d_test = xgb.DMatrix(x_test)
d_test = xgb.DMatrix(pca_test)
```

In [172…

```
#Hyperparameters for the XGboost
params = {}
params['objective'] = 'reg:linear'
params['eta'] = 0.02
params['max_depth'] = 4
```

In [173…

```
#Defining the function to compute the R2 score
def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)
```

In [174…

```
#Creating the list of validation sets for which metrics will evaluated during training to
watchlist = [(d_train, 'train'), (d_valid, 'valid')]
```

In [175…

```
clf = xgb.train(params, d_train,
                1000, evals=watchlist, early_stopping_rounds=50,
                feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[23:29:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objec
tive/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
[0]     train-rmse:98.85892     train-r2:-60.19578      valid-rmse:99.44724     valid-r2:-
59.09219
[10]    train-rmse:81.02473     train-r2:-40.10786      valid-rmse:81.62129     valid-r2:-
39.47991
[20]    train-rmse:66.49194     train-r2:-26.68393      valid-rmse:67.08800     valid-r2:-
26.34780
[30]    train-rmse:54.66175     train-r2:-17.70926      valid-rmse:55.26186     valid-r2:-
17.55596
[40]    train-rmse:45.04908     train-r2:-11.70754      valid-rmse:45.67114     valid-r2:-
11.67407
[50]    train-rmse:37.25586     train-r2:-7.69118       valid-rmse:37.89442     valid-r2:-
7.72535
        se:30.96523     train-r2:-5.00397       valid-rmse:31.62873     valid-r2:-
```

Loading [MathJax]/extensions/Safe.js

```
5.07849
[70]	train-rmse:25.89201	train-r2:-3.19780	valid-rmse:26.61757	valid-r2:-
3.30496
[80]	train-rmse:21.84406	train-r2:-1.98783	valid-rmse:22.62199	valid-r2:-
2.10953
[90]	train-rmse:18.63692	train-r2:-1.17489	valid-rmse:19.46038	valid-r2:-
1.30110
[100]	train-rmse:16.11725	train-r2:-0.62657	valid-rmse:17.00610	valid-r2:-
0.75728
[110]	train-rmse:14.15940	train-r2:-0.25539	valid-rmse:15.12156	valid-r2:-
0.38939
[120]	train-rmse:12.66217	train-r2:-0.00394	valid-rmse:13.68786	valid-r2:-
0.13842
[130]	train-rmse:11.52109	train-r2:0.16886	valid-rmse:12.62682	valid-r2:
0.03123
[140]	train-rmse:10.68079	train-r2:0.28567	valid-rmse:11.84419	valid-r2:
0.14760
[150]	train-rmse:10.05614	train-r2:0.36678	valid-rmse:11.28621	valid-r2:
0.22602
[160]	train-rmse:9.59897	train-r2:0.42305	valid-rmse:10.88979	valid-r2:
0.27944
[170]	train-rmse:9.26316	train-r2:0.46271	valid-rmse:10.60380	valid-r2:
0.31679
[180]	train-rmse:9.01222	train-r2:0.49143	valid-rmse:10.39966	valid-r2:
0.34284
[190]	train-rmse:8.82391	train-r2:0.51246	valid-rmse:10.25291	valid-r2:
0.36126
[200]	train-rmse:8.67304	train-r2:0.52899	valid-rmse:10.14354	valid-r2:
0.37481
[210]	train-rmse:8.55986	train-r2:0.54120	valid-rmse:10.06293	valid-r2:
0.38471
[220]	train-rmse:8.47603	train-r2:0.55014	valid-rmse:10.00510	valid-r2:
0.39176
[230]	train-rmse:8.41103	train-r2:0.55702	valid-rmse:9.96356	valid-r2:
0.39680
[240]	train-rmse:8.36463	train-r2:0.56189	valid-rmse:9.93332	valid-r2:
0.40046
[250]	train-rmse:8.32923	train-r2:0.56559	valid-rmse:9.90829	valid-r2:
0.40347
[260]	train-rmse:8.29794	train-r2:0.56885	valid-rmse:9.89063	valid-r2:
0.40560
[270]	train-rmse:8.26746	train-r2:0.57201	valid-rmse:9.87484	valid-r2:
0.40749
[280]	train-rmse:8.23202	train-r2:0.57567	valid-rmse:9.86564	valid-r2:
0.40860
[290]	train-rmse:8.19694	train-r2:0.57928	valid-rmse:9.85346	valid-r2:
0.41006
[300]	train-rmse:8.17540	train-r2:0.58149	valid-rmse:9.84843	valid-r2:
0.41066
[310]	train-rmse:8.14618	train-r2:0.58447	valid-rmse:9.83957	valid-r2:
0.41172
[320]	train-rmse:8.12154	train-r2:0.58698	valid-rmse:9.83655	valid-r2:
0.41208
[330]	train-rmse:8.09927	train-r2:0.58925	valid-rmse:9.83342	valid-r2:
0.41245
[340]	train-rmse:8.07813	train-r2:0.59139	valid-rmse:9.83117	valid-r2:
0.41272
[350]	train-rmse:8.05792	train-r2:0.59343	valid-rmse:9.82747	valid-r2:
0.41317
[360]	train-rmse:8.03630	train-r2:0.59561	valid-rmse:9.82287	valid-r2:
0.41371
[370]	train-rmse:8.00681	train-r2:0.59857	valid-rmse:9.82217	valid-r2:
0.41380
[380]	train-rmse:7.98149	train-r2:0.60111	valid-rmse:9.81698	valid-r2:
0.41442
se:7.95326	train-r2:0.60392	valid-rmse:9.81526	valid-r2:
```

```
0.41462
[400]	train-rmse:7.92849	train-r2:0.60639	valid-rmse:9.81171	valid-r2:
0.41505
[410]	train-rmse:7.90173	train-r2:0.60904	valid-rmse:9.81202	valid-r2:
0.41501
[420]	train-rmse:7.87597	train-r2:0.61158	valid-rmse:9.80904	valid-r2:
0.41536
[430]	train-rmse:7.85423	train-r2:0.61372	valid-rmse:9.80795	valid-r2:
0.41549
[440]	train-rmse:7.82935	train-r2:0.61617	valid-rmse:9.80941	valid-r2:
0.41532
[450]	train-rmse:7.80591	train-r2:0.61846	valid-rmse:9.80192	valid-r2:
0.41621
[460]	train-rmse:7.78180	train-r2:0.62082	valid-rmse:9.80249	valid-r2:
0.41614
[470]	train-rmse:7.76132	train-r2:0.62281	valid-rmse:9.80010	valid-r2:
0.41643
[480]	train-rmse:7.73317	train-r2:0.62554	valid-rmse:9.80203	valid-r2:
0.41620
[490]	train-rmse:7.71659	train-r2:0.62714	valid-rmse:9.80219	valid-r2:
0.41618
[500]	train-rmse:7.69134	train-r2:0.62958	valid-rmse:9.80437	valid-r2:
0.41592
[510]	train-rmse:7.67415	train-r2:0.63123	valid-rmse:9.80081	valid-r2:
0.41635
[520]	train-rmse:7.64949	train-r2:0.63360	valid-rmse:9.80355	valid-r2:
0.41602
[530]	train-rmse:7.62429	train-r2:0.63601	valid-rmse:9.80267	valid-r2:
0.41612
[540]	train-rmse:7.60201	train-r2:0.63814	valid-rmse:9.80357	valid-r2:
0.41602
[550]	train-rmse:7.58272	train-r2:0.63997	valid-rmse:9.80376	valid-r2:
0.41599
[560]	train-rmse:7.56369	train-r2:0.64177	valid-rmse:9.80368	valid-r2:
0.41600
[565]	train-rmse:7.55255	train-r2:0.64283	valid-rmse:9.80344	valid-r2:
0.41603
```

At the end of the training, we have attained an optimal train R2 score of 0.642 and an optimal validation R2 score of 0.416. we can now proceed to predict the values with our test set.

In [176…

```python
#Predicting the time taken with respect to variables in the test set
predictions = clf.predict(d_test)
```

In [177…

```python
test['Estimated Time'] = predictions
```

In [178…

```python
test.head()
```

Out[178…

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X376	X377	X378	X379	X380	X382	X383	X:
0	az	v	n	f	d	t	a	w	0	0	...	0	0	1	0	0	0	0	
1	t	b	ai	a	d	b	g	y	0	0	...	0	1	0	0	0	0	0	
2	az	v	as	f	d	a	j	j	0	0	...	0	0	1	0	0	0	0	
3	az	l	n	f	d	z	l	n	0	0	...	0	0	1	0	0	0	0	
4	w	s	as	c	d	y	i	m	0	0	...	0	0	0	0	0	0	0	

5 rows × 377 columns

Results & Conclusion

We analysed the train set of each utility category variable of a vehicle with respect to its frequency and its variation with time and made some key observations. We concluded that there is very less variance of X4 feature variable with each categorical feature having some set of outliers.After implementing PCA reduction into 12 components, we created an xgboost model with an r2 score of 0.642. Finally the test set run through the model to estimate the minimum time taken for each vehicle configuration in the test set in order to enable faster testing, resulting in lower carbon dioxide emissions without reducing Mercedes-Benz's standards.