Assessment
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.

Problem Statement

To predict the target variable 'y' lie time in seconds that the car needs to pass the testing.

The performance metric used:

R2 (Coefficient of determination): It is the proportion of the variance in the dependent variable that is predictable from the independent variables.

$$R\text{-}Squared = 1 - \frac{SS_{regression}}{SS_{total}}$$

Problem Approach

1. EDA Understanding the data.

File train.csv contains 4209 data points and 378 features, and test.csv contains 4209 data points and 377 features.

```
[3]: train_data.head()
```

[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

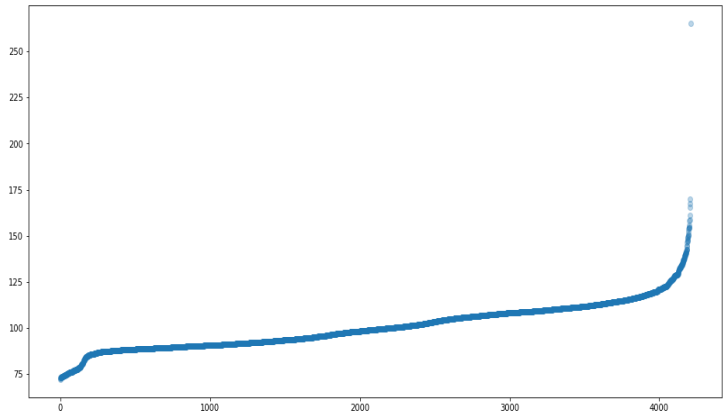
5 rows × 378 columns

Train data has 8 categorical features, 396 binary features, and target variable ‘y’ as float.

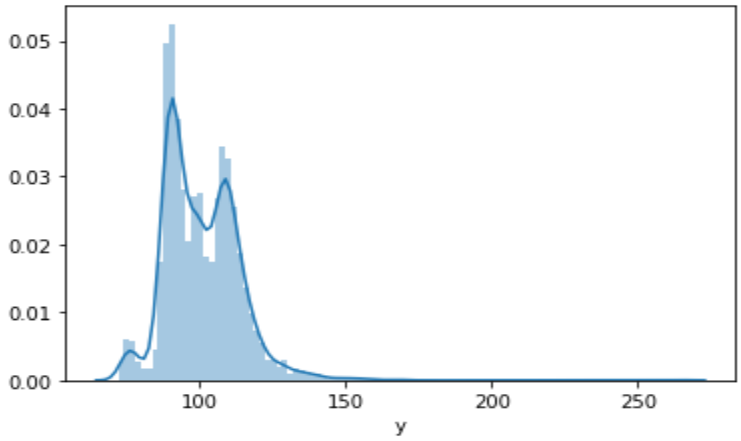
```
[6]: train_data.info()

<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
```

There are no null values in the dataset.

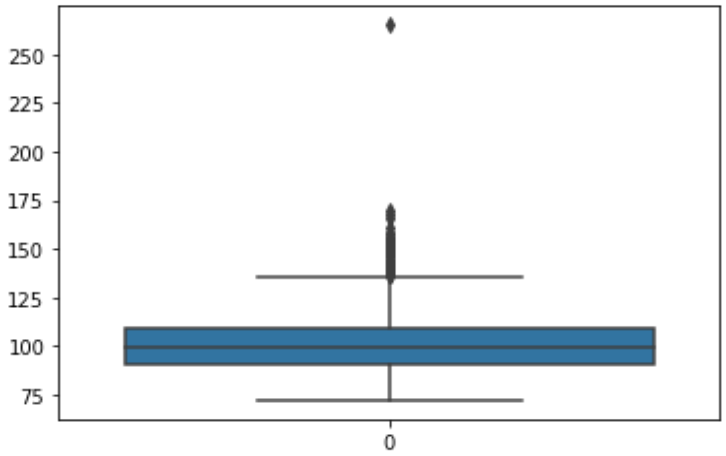
2. Target and Feature Analysis



Index vs y



Histogram of target variable y



Boxplot of target variable y

As there is outliers in the target variable. We can set a threshold to it and remove data points outside the threshold. In this assessment the threshold value is set to 180.

Binary Feature Analysis

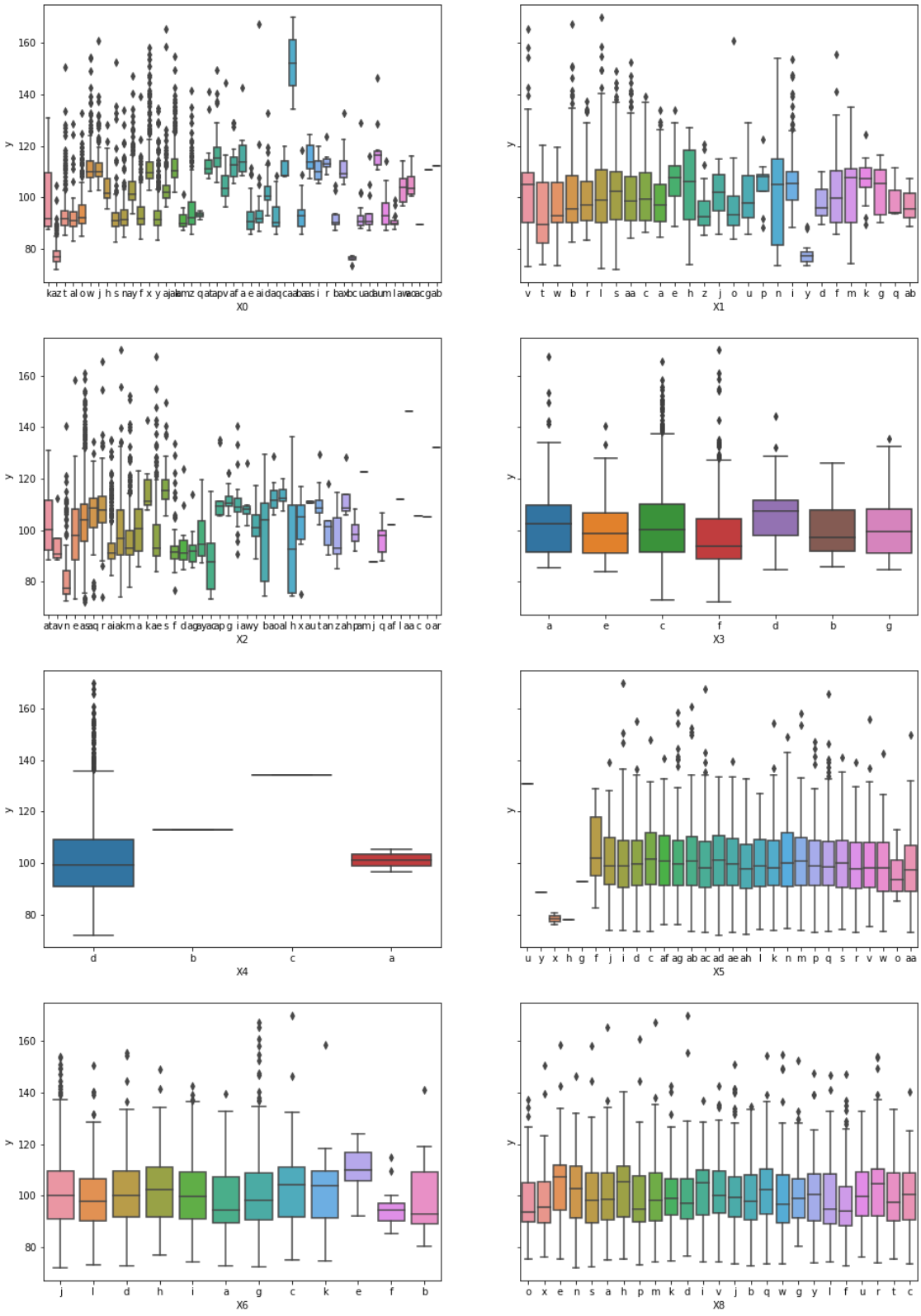
Variance of below 13 features is 0, So we can remove these features.

Checking column(s) Variance, if the variance is equal to zero, then need to remove those variable(s).

[9]:
```
column_var = train_data.var()
column_var_zero = column_var[column_var == 0]
print(column_var_zero)
```

```
X11     0.0
X93     0.0
X107    0.0
X233    0.0
X235    0.0
X268    0.0
X289    0.0
X290    0.0
X293    0.0
X297    0.0
X330    0.0
X347    0.0
dtype: float64
```

Categorical Feature Analysis



Variance for X4 (X4 : 0.005462503197188894) is low, so we can remove this also.

3. Label Encoding

Label Encoding

[19]:
```
from sklearn.preprocessing import LabelEncoder
```

[20]:
```
for col in cat_cols:
    lEnc = LabelEncoder()
    lEnc.fit(list(train_data[col].values) + list(test_data[col].values))
    train_data[col] = lEnc.transform(train_data[col])
    test_data[col] = lEnc.transform(test_data[col])
```

[23]:
```
train_data.head()
```

[23]:

	ID	y	X0	X1	X2	X3	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	37	23	20	0	27	9	14	0	...	0	0	1	0	0	0	0	0	0	0
1	6	88.53	37	21	22	4	31	11	14	0	...	1	0	0	0	0	0	0	0	0	0
2	7	76.26	24	24	38	2	30	9	23	0	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	24	21	38	5	30	11	4	0	...	0	0	0	0	0	0	0	0	0	0
4	13	78.02	24	23	38	5	14	3	13	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 365 columns

4. Dimensionality Reduction

Dimensionality reduction

[28]:
```
from sklearn.decomposition import PCA
```

[29]:
```
n_comp = 15
pca = PCA(n_components=n_comp, random_state=101)
pca_train_data = pca.fit_transform(X)
pca_test_data = pca.transform(test_data)
```

[30]:
```
pca_train_data.shape
```

[30]:
```
(4208, 15)
```

5. Model Training

- XGBoostRegressor + PCA components

[31]:
```
from xgboost import XGBRegressor
from sklearn.model_selection import KFold
```

[32]:
```
xgb = XGBRegressor(n_estimators=600,
                   max_depth=6,
                   learning_rate=0.01,
                   random_state=101,
                   min_child_weight=1)

kfold = KFold(n_splits=5, shuffle=True, random_state=101)
```

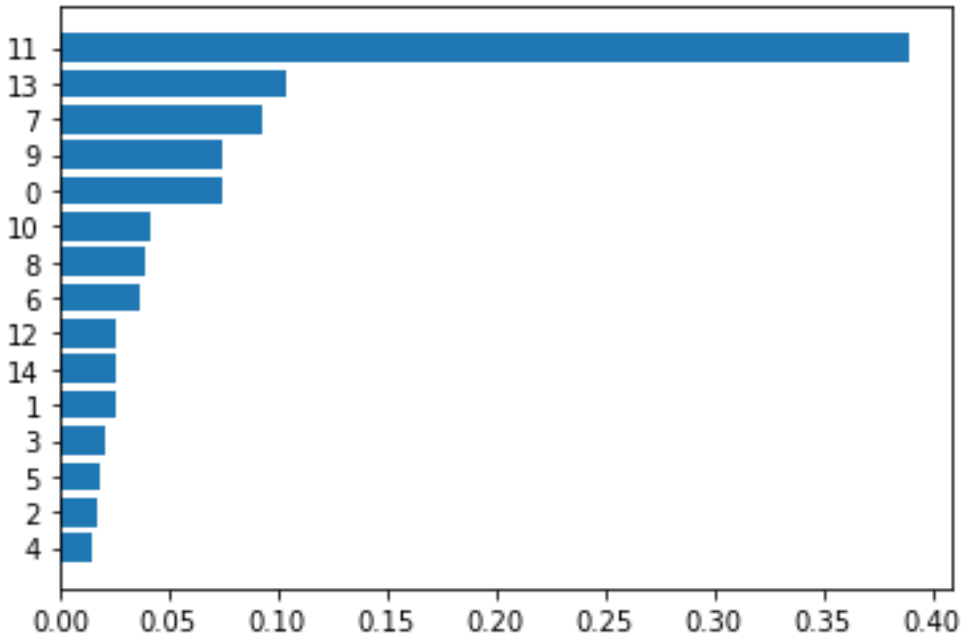
[33]:
```
from sklearn.metrics import r2_score
```

[34]:
```
sps = kfold.split(pca_train_data, y=Y)

for train, test in sps:
    xgb.fit(pca_train_data[train], Y[train])
    print(r2_score(Y[test], xgb.predict(pca_train_data[test])))
```

```
0.50929524123537
0.4461743552525812
0.5495015357196538
0.522390594368133
0.47026981844020244
```

average r2_score : 0.499526309003188

Important Feature in PCA components



- XGBoostRegressor + (Train Features + PCA components)

Train data + PCA

[36]:
```
for i in range(n_comp):
    X['pca_'+str(i)] = pca_train_data[:,i]
    test_data['pca_'+str(i)] = pca_test_data[:,i]

X.shape
```

[36]:
```
(4208, 378)
```

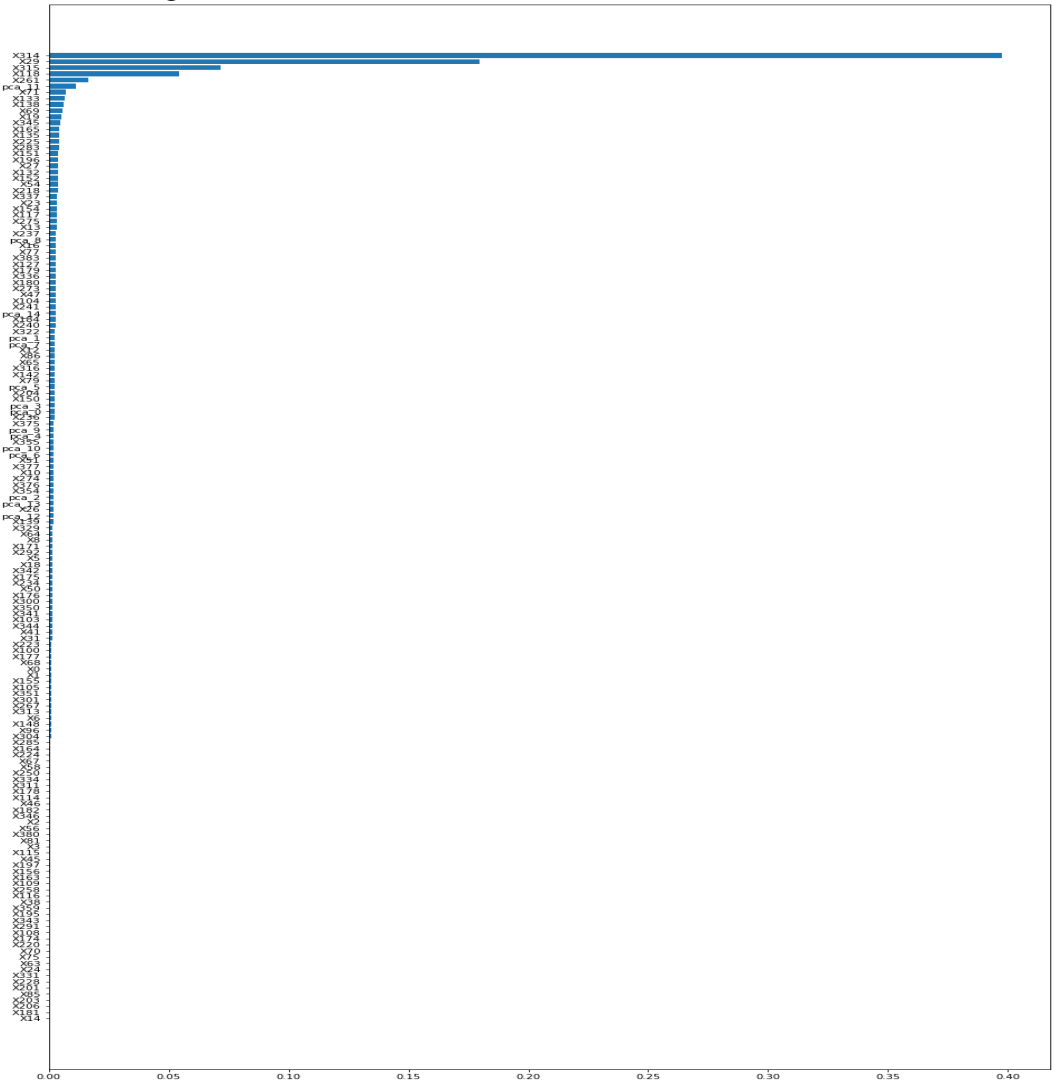
[37]:
```
xgb2 = XGBRegressor(n_estimators=600,
            max_depth=6,
            learning_rate=0.01,
            random_state=101,
            min_child_weight=1)

ksplits = kfold.split(X, y=Y)

for train, test in ksplits:
    xgb2.fit(X.iloc[train], Y[train])
    print(r2_score(Y[test], xgb2.predict(X.iloc[test])))
```

```
0.6246807263271787
0.5302870687249952
0.6185234796830945
0.5905418727337417
0.5339260779488582
```

---

average r2_score : 0.579591845083574

Feature Importance with above model



Conclusion

XGBoost Regressor with Train Features and PCA components the Average R2 score is 0.579591845083574

The above model is used for predicting test data as well.

1.1.5 Predicting Test time for Test data.

[40]:
```
test_pred = xgb2.predict(test_data)
```

[41]:
```
test_pred
```

[41]:
```
array([ 85.579346, 102.61905 ,  86.329765, …,  92.420616, 111.9709  ,
        90.913025], dtype=float32)
```

[ ]: