Mercedes-Benz Greener Manufacturing Project

Rohit Gaurav Mishra

Project Description

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.

Problem Statements:

1. If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
2. Check for null and unique values for test and train sets.
3. Apply label encoder.
4. Perform dimensionality reduction.
5. Predict your test_df values using XGBoost.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
%matplotlib inline
```

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

In [3]:
```
print('Size of training set: {} rows and {} columns'
      .format(*df_train.shape))
```

```
Size of training set: 4209 rows and 378 columns
```

In [4]:
```
print('Size of test set: {} rows and {} columns'
      .format(*df_test.shape))
```

```
Size of test set: 4209 rows and 377 columns
```

In [5]:
```
df_train.head()
```

Out[5]:

	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 [6]:
```
df_test.head()
```

Out[6]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0	0	0	0
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0	0	0	0
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

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

Out[7]:

	ID	y	X10	X11	X12	X13	X14	X15	X16	X17	...	X375	X[illegible]
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	...	4209.000000	4209.000[illegible]
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.000475	0.002613	0.007603	...	0.318841	0.057[illegible]
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.021796	0.051061	0.086872	...	0.466082	0.232[illegible]
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000[illegible]
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000[illegible]
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000[illegible]
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	...	1.000000	0.000[illegible]
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.000[illegible]

8 rows × 370 columns

Dropping `ID` Column from `test` and `train`

In [8]:
```
# Dropping ID column from datasets
df_train.drop('ID',inplace=True,axis=1)
df_test.drop('ID',inplace=True,axis=1)
```

`ID` column is dropped

In [9]:
```
df_train.columns
```

Out[9]:
```
Index(['y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=377)
```

In [10]:
```
df_test.columns
```

Out[10]:
```
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10', 'X11',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=376)
```

1. Column with Zero Variance

In [11]:
```
#Identifying the column with Zero variance.
zero_var_cols = df_train.var()[df_train.var()==0].index.values
```

In [12]:
```
zero_var_cols
```

Out[12]:
```
array(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290',
       'X293', 'X297', 'X330', 'X347'], dtype=object)
```

From above result we can see there are 12 columns which have Zero variance. We can drop these columns as they do not have any impact as such

In [13]:
```
df_train.drop(zero_var_cols,inplace=True,axis=1)
df_test.drop(zero_var_cols,inplace=True,axis=1)
```

In [14]:
```
df_train.columns
```

Out[14]:
```
Index(['y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=365)
```

2. Check for missing values

In [15]:
```
missing_df = df_train.isnull().sum(axis=0).reset_index()
missing_df.columns = ['column_name', 'missing_count']
missing_df = missing_df.loc[missing_df['missing_count']>0]
missing_df = missing_df.sort_values(by='missing_count')
missing_df
```

Out[15]:

column_name	missing_count

From above results we can say that we do not have any missing values in train dataframe.

Check for Datatypes of columns

In [16]:
```
datatype = df_train.dtypes.reset_index()
datatype.columns = ["Count", "Column Type"]
datatype.groupby("Column Type").aggregate('count').reset_index()
```

Out[16]:

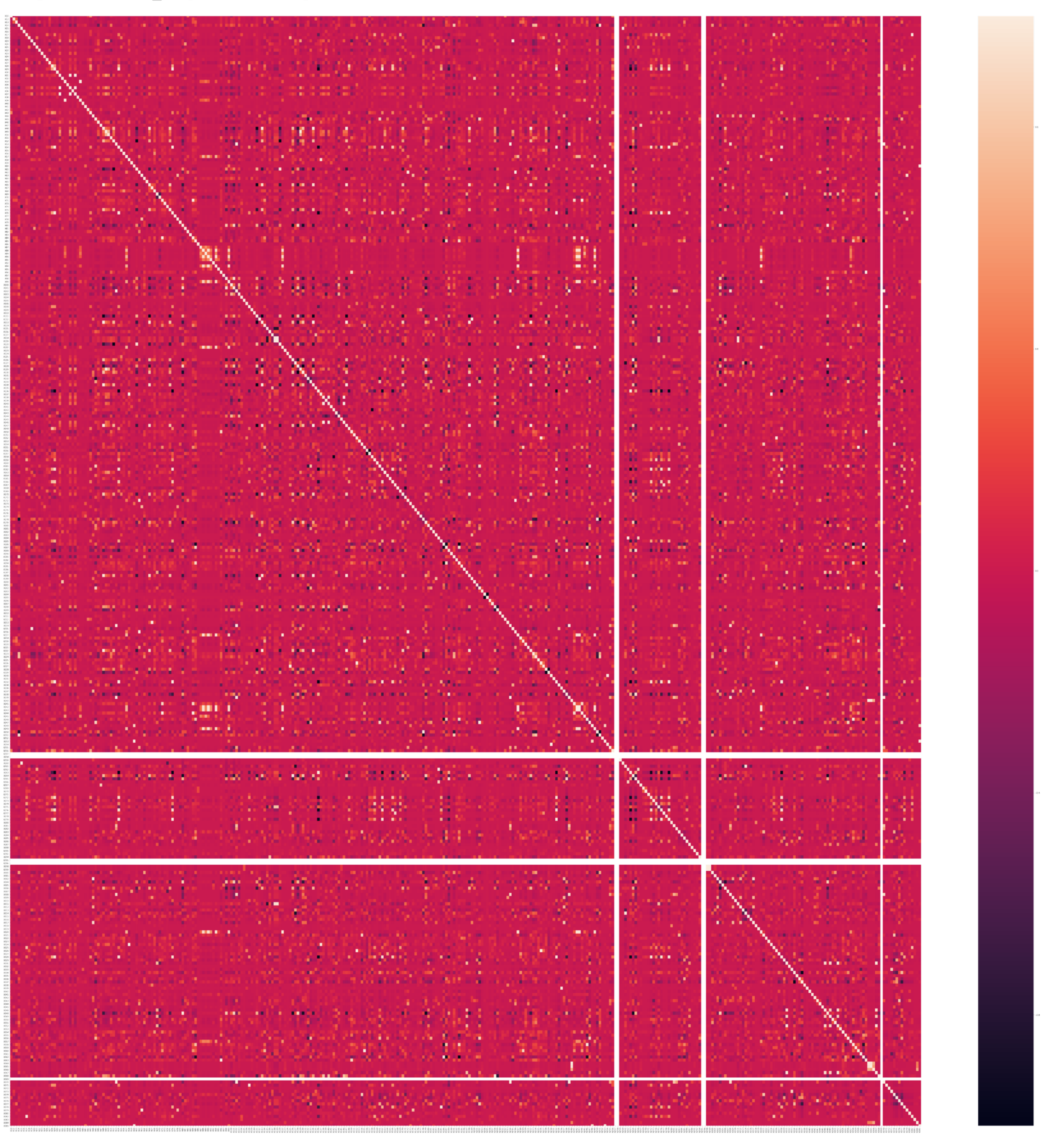
	Column Type	Count
0	int64	356
1	float64	1
2	object	8

There are 8 Categorical columns, 1 float column and 369 integer values

Heat Map Visualization

In [17]:
```
plt.figure(figsize=(100,100))
sns.heatmap(df_test.corr())
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10a857110>
```



3. Apply label encoder

In [18]:
```
#Check for all columns with dtypes object. We'll covert them to numeric values.
label_columns = df_train.describe(include=['object']).columns.values
label_columns
```

Out[18]:
```
array(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype=object)
```

In [19]:
```
#We'll perform label encoding to converting above 8 columns to numeric values.
lab_enc = LabelEncoder()
```

In [20]:
```
for col in label_columns:
    lab_enc.fit(df_train[col].append(df_test[col]).values)
    df_train[col]=lab_enc.transform(df_train[col])
    df_test[col]=lab_enc.transform(df_test[col])
```

In [21]:
```
df_train.head()
```

Out[21]:

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	130.81	37	23	20	0	3	27	9	14	0	...	0	0	1	0	0	0	0	0	0	0
1	88.53	37	21	22	4	3	31	11	14	0	...	1	0	0	0	0	0	0	0	0	0
2	76.26	24	24	38	2	3	30	9	23	0	...	0	0	0	0	0	0	1	0	0	0
3	80.62	24	21	38	5	3	30	11	4	0	...	0	0	0	0	0	0	0	0	0	0
4	78.02	24	23	38	5	3	14	3	13	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 365 columns

In [22]:
```
df_test.head()
```

Out[22]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	24	23	38	5	3	26	0	22	0	0	...	0	0	0	1	0	0	0	0	0	0
1	46	3	9	0	3	9	6	24	0	0	...	0	0	1	0	0	0	0	0	0	0
2	24	23	19	5	3	0	9	9	0	0	...	0	0	0	1	0	0	0	0	0	0
3	24	13	38	5	3	32	11	13	0	0	...	0	0	0	1	0	0	0	0	0	0
4	49	20	19	2	3	31	8	12	0	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 364 columns

From above result we can see all 8 columns of object type are converted to numeric values

4. Perform dimensionality reduction

In [23]:
```
# Perform dimensionality reduction using PCA
pca = PCA(0.98,svd_solver='full')
```

In [24]:
```
X = df_train.drop('y',axis=1)
y = df_train['y']
```

In [25]:
```
#splliting the data into test train split.
X_train,X_val,y_train,y_val=train_test_split(X,y,test_size=0.2,random_state=42)
```

In [26]:
```
pca.fit(X)
```

Out[26]:
```
PCA(copy=True, iterated_power='auto', n_components=0.98, random_state=None,
    svd_solver='full', tol=0.0, whiten=False)
```

In [27]:
```
pca.n_components_
```

Out[27]:
```
12
```

From above result we can infer that `98%` of variance in data is captured by just `12 features`. As compared to 365 features this is huge reduction in components

In [28]:
```
pca.explained_variance_ratio_
```

Out[28]:
```
array([0.40868988, 0.21758508, 0.13120081, 0.10783522, 0.08165248,
       0.0140934 , 0.00660951, 0.00384659, 0.00260289, 0.00214378,
       0.00209857, 0.00180388])
```

In [29]:
```
pca_X_train = pd.DataFrame(pca.transform(X_train))
pca_X_val = pd.DataFrame(pca.transform(X_val))
pca_test = pd.DataFrame(pca.transform(df_test))
```

5. Predict your test_df values using XGBoost

In [30]:
```
model = xgb.XGBRegressor(objective='reg:squarederror',learning_rate=0.1)
```

In [31]:
```
model.fit(pca_X_train,y_train)
```

```
/Users/rgm/Python/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

Out[31]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:squarederror',
             random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             seed=None, silent=None, subsample=1, verbosity=1)
```

In [32]:
```
pred_y_val = model.predict(pca_X_val)
```

In [33]:
```
mse_score = mean_squared_error(y_val,pred_y_val)
```

In [34]:
```
print(mse_score)
```

```
84.22703928013917
```

Trying another alogorithm i.e. `Random Forest Classifier`

In [35]:
```
model_RF = RandomForestRegressor(max_depth=2,random_state=0,n_estimators=100)
model_RF.fit(pca_X_train,y_train)
```

Out[35]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=100,
                      n_jobs=None, oob_score=False, random_state=0, verbose=0,
                      warm_start=False)
```

In [36]:
```
pred_y_val_RF = model_RF.predict(pca_X_val)
```

In [37]:
```
mse_score = mean_squared_error(y_val,pred_y_val_RF)
print(mse_score)
```

```
119.54165930411521
```

In [38]:
```
model.predict(pca_test)
```

Out[38]:
```
array([ 76.27864 ,  96.719086,  83.004944, ...,  99.94769 , 109.16827 ,
        95.07033 ], dtype=float32)
```

From above result we can conclude that mse score for `xgboost` is `84` which is less than mse score for `Random Forest Classifier` i.e. `119`. So `xgBoost` gives us better results

END