# 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:

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

In [1]:
```
#Loading the library
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np
from sklearn.model_selection import train_test_split
```

```
from sklearn import preprocessing # Import Label Encoderf
from xgboost import XGBClassifier
from sklearn.metrics import mean_squared_error, r2_score
```

In [2]:
```
#loading the dataset
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
train_df.shape,test_df.shape
```

Out[2]: ((4209, 378), (4209, 377))

In [3]:
```
#checking first row
train_df.head(1)
```

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 rows × 378 columns

In [4]:
```
#checking first row of test data
test_df.head(1)
```

Out[4]:

| | 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 rows × 377 columns

---

**Droping unwanted columns like ID in both train_df and test_df**

In [5]:
```
#removing ID columns from both dataset train and test
train_df = train_df.drop(columns=['ID'])
```

In [6]:
```
train_df.head(1)
```

Out[6]:

| | y | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X8 | X10 | ... | X375 | X376 | X377 | X378 | X379 | X380 | X382 | X383 | X384 | X385 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 130.81 | k | v | at | a | d | u | j | o | 0 | ... | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

1 rows × 377 columns

In [7]:
```
test_df =test_df.drop(columns=['ID'])
```

In [8]:
```
test_df.head(0)
```

Out[8]:

| | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X8 | X10 | X11 | ... | X375 | X376 | X377 | X378 | X379 | X380 | X382 | X383 | X384 | X385 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

0 rows × 376 columns

**So, successfully dropped ID column**

---

## 1. Checking the null values and if available then will drop

In [9]:
```
#checking null value in both dataset
print(train_df.isnull().sum(axis=0).value_counts())
print(test_df.isnull().sum(axis=0).value_counts())
```

```
0    377
Name: count, dtype: int64
0    376
Name: count, dtype: int64
```

---

## Seperating out target variable from train_df dataset

In [10]:
```
Y_target= train_df['y']
Y_target.shape
```

Out[10]: (4209,)

In [11]:
```
#droping target variable from train_df
train_df = train_df.drop(columns=['y'])
train_df.head(0)
```

Out[11]:

| X0 | X1 | X2 | X3 | X4 | X5 | X6 | X8 | X10 | X11 | ... | X375 | X376 | X377 | X378 | X379 | X380 | X382 | X383 | X384 | X385 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

0 rows × 376 columns

In [12]:
```
#checking the row and column of the dataset
train_df.shape, test_df.shape, Y_target.shape
```

Out[12]: ((4209, 376), (4209, 376), (4209,))

there is not null values are available in both dataset

---

## 2. Now checking categorical variable (columns) in both train_df and test_df

In [13]:
```
#collect categorical columns names from train_df
categorical_train_df = train_df.select_dtypes(include=object).columns
```

In [14]:
```
#collect categorical columns names from test_df
categorical_test_df = test_df.select_dtypes(include=object).columns
```

In [15]:
```
#checking the no. of categorical columns in both dataset
categorical_train_df, categorical_test_df
```

Out[15]:
```
(Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='object'),
 Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='object'))
```

Based on above observation both have same categorical columns

So, now will do the label_encoder.

#laoding label_encoder l_encoder = preprocessing.LabelEncoder() l_encoder #appling label_encoder in categorical column of both train_df, test_df for col in categorical_train_df: train_df[col] = l_encoder.fit_transform(train_df[col]) test_df[col] = l_encoder.transform(test_df[col]) #Note: while using Label_encoder got this error:- ValueError: y contains previously unseen labels: 'av' #for Solving this error we are using OrdinalEncoder.

In [16]:
```
from sklearn.preprocessing import OrdinalEncoder
```

In [17]:
```
# Apply OrdinalEncoder with handle_unknown='use_encoded_value'
ordinal_encoder = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
ordinal_encoder
```

Out[17]:

OrdinalEncoder

```
OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
```

In [18]:
```
#appling label_encoder in categorical column of both train_df, test_df
for col in categorical_train_df:
    train_df[col] = ordinal_encoder.fit_transform(train_df[[col]])
    test_df[col] = ordinal_encoder.transform(test_df[[col]])
```

In [19]:
```
train_df.head(), test_df.head()
```

Out[19]:
```
(     X0    X1    X2   X3   X4    X5    X6    X8  X10  X11  ...  X375  X376  \
 0  32.0  23.0  17.0  0.0  3.0  24.0   9.0  14.0    0    0  ...     0     0
 1  32.0  21.0  19.0  4.0  3.0  28.0  11.0  14.0    0    0  ...     1     0
 2  20.0  24.0  34.0  2.0  3.0  27.0   9.0  23.0    0    0  ...     0     0
 3  20.0  21.0  34.0  5.0  3.0  27.0  11.0   4.0    0    0  ...     0     0
 4  20.0  23.0  34.0  5.0  3.0  12.0   3.0  13.0    0    0  ...     0     0

    X377  X378  X379  X380  X382  X383  X384  X385
 0     1     0     0     0     0     0     0     0
 1     0     0     0     0     0     0     0     0
 2     0     0     0     0     1     0     0     0
 3     0     0     0     0     0     0     0     0
 4     0     0     0     0     0     0     0     0

 [5 rows x 376 columns],
      X0    X1    X2   X3   X4    X5    X6    X8  X10  X11  ...  X375  X376  \
 0  20.0  23.0  34.0  5.0  3.0  -1.0   0.0  22.0    0    0  ...     0     0
 1  40.0   3.0   7.0  0.0  3.0  -1.0   6.0  24.0    0    0  ...     0     0
 2  20.0  23.0  16.0  5.0  3.0  -1.0   9.0   9.0    0    0  ...     0     0
 3  20.0  13.0  34.0  5.0  3.0  -1.0  11.0  13.0    0    0  ...     0     0
 4  43.0  20.0  16.0  2.0  3.0  28.0   8.0  12.0    0    0  ...     1     0

    X377  X378  X379  X380  X382  X383  X384  X385
 0     0     1     0     0     0     0     0     0
 1     1     0     0     0     0     0     0     0
 2     0     1     0     0     0     0     0     0
 3     0     1     0     0     0     0     0     0
 4     0     0     0     0     0     0     0     0

 [5 rows x 376 columns])
```

In [20]:
```
#checking the label_encoder applied or not
train_df.head(0)
```

Out[20]:

| X0 | X1 | X2 | X3 | X4 | X5 | X6 | X8 | X10 | X11 | … | X375 | X376 | X377 | X378 | X379 | X380 | X382 | X383 | X384 | X385 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

0 rows × 376 columns

In [21]:
```
test_df.head(0)
```

Out[21]:

| X0 | X1 | X2 | X3 | X4 | X5 | X6 | X8 | X10 | X11 | ... | X375 | X376 | X377 | X378 | X379 | X380 | X382 | X383 | X384 | X385 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

0 rows × 376 columns

Successfully encoded categorical column in to numeric

## 3. Checking the variance of each column, if columns has variance 0 then will drop those columns

In [22]:
```
#checking columns with variance with 0 in both train_df and test_df
(train_df.var() ==0).value_counts(), (test_df.var() ==0).value_counts()
```

Out[22]:
```
(False    364
 True      12
 Name: count, dtype: int64,
 False    371
 True       5
 Name: count, dtype: int64)
```

Above observation, In train_df has 12 columns with zero variance and test_df has 5 columns with 0 Variance

So, we are droping those columns in both train and test dataset to make no. of columns in both dataset.

In [23]:
```
#colecting zero variance columns
Zeror_variance_columns_train = train_df.var()[train_df.var() == 0].index
Zeror_variance_columns_test = test_df.var()[test_df.var() == 0].index
```

In [24]:
```
#checking size of train and test dataset before droping 0 variance column
train_df.shape , test_df.shape
```

Out[24]:
```
((4209, 376), (4209, 376))
```

In [25]:
```
#droping column with 0 variance
train_df = train_df.drop(columns=Zeror_variance_columns_train)
test_df = test_df.drop(columns=Zeror_variance_columns_test)
```

```
#checking size of train and test dataset after droping 0 variance column
train_df.shape , test_df.shape
```

Out[25]: ((4209, 364), (4209, 371))

In [26]:
```
#checking the different column in test dataset
diff_column_in_train_not_in_test =  list(set(train_df.columns) -set(test_df.columns))
#checking the different column in train dataset
diff_column_in_test_not_in_train =  list(set(test_df.columns) - set(train_df.columns))
diff_column_in_test_not_in_train, diff_column_in_train_not_in_test
```

Out[26]:
```
(['X268',
  'X233',
  'X293',
  'X290',
  'X93',
  'X297',
  'X289',
  'X330',
  'X235',
  'X347',
  'X107',
  'X11'],
 ['X258', 'X295', 'X296', 'X369', 'X257'])
```

In [27]:
```
train_df = train_df.drop(columns=diff_column_in_train_not_in_test)
test_df = test_df.drop(columns=diff_column_in_test_not_in_train)
train_df.shape, test_df.shape
```

Out[27]: ((4209, 359), (4209, 359))

#this align funtion are used to remove different column in both dataset and make all columns same in numbers in both dset train_df, test_df = train_df.align(test_df, axis=1, join='inner')

---

## 4. Perform dimensionality reduction.

Before perform dimensionality reduction, need to use scalling in input features.

In [28]:
```
#Loading the StandardScaler
scaled = preprocessing.StandardScaler()
scaled
```

Out[28]:
```
StandardScaler()
```

In [29]:
```
#perform scaling
train_df_scaled = scaled.fit_transform(train_df)
test_df_scaled = scaled.transform(test_df)
```

In [30]:
```
#after appling the standardScaling it return array. So, again need to assign column name
train_df_scaled.shape, test_df_scaled.shape
```

Out[30]:
```
((4209, 359), (4209, 359))
```

In [31]:
```
#loading the library for PCA and appling the pca in scaled dataset
from sklearn.decomposition import PCA
pca = PCA(n_components=0.95)
#apply PCA
train_df_pca = pca.fit_transform(train_df_scaled)
test_df_pca = pca.transform(test_df_scaled)
```

In [32]:
```
#checking how many componet were keep
train_df_pca.shape, test_df_pca.shape
```

Out[32]:
```
((4209, 146), (4209, 146))
```

In [33]:
```
#spliting the train_df_pca and Y_train in to x_train, x_test, y_train, y_testy_train using train_df_PCA data.
x_train, x_test, y_train, y_test = train_test_split(train_df_pca, Y_target, test_size=0.2, random_state=42)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[33]:
```
((3367, 146), (842, 146), (3367,), (842,))
```

## Perform XGBoost algorithm

In [34]:
```
#Loading xgboost algorithm for regression.
from xgboost import XGBRegressor
model_xgb = XGBRegressor(objective="reg:squarederror",learning_rate=0.1)
model_xgb
```

Out[34]:
```
XGBRegressor(colsample_bynode=None, colsample_bytree=None,
             enable_categorical=False, gamma=None, gpu_id=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.1, max_delta_step=None, max_depth=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, reg_alpha=None, reg_lambda=None,
             scale_pos_weight=None, subsample=None, tree_method=None,
             validate_parameters=None, verbosity=None)
```

Predicting the train_df_pca data which we split above into x_train, x_test, y_train, y_test

In [35]:
```
#fitting the dataset in to modelXGB
model_xgb.fit(x_train, y_train)
model_xgb
```

Out[35]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.1, max_delta_step=0,
             max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=4,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

In [36]:

```
#predicting the model using x_test then will compare the predicted value to y_test.
predict_y_test = model_xgb.predict(x_test)
```

In [37]:

```
#checking the mean square ERROR
print("MSE:", mean_squared_error(y_test, predict_y_test))
print("R² Score:", r2_score(y_test, predict_y_test))
```

```
MSE: 85.52299901642205
R² Score: 0.45054267793632186
```

## 5. Now Predict your test_df values using XGBoost.

In [38]:

```
#predicting the model with test_df dataset
predict_x_test = model_xgb.predict(test_df_pca)
```

In [39]:

```
#converting predict_x_test in to dataframe for save in .csv file
predict_x_test_df_using_XGBOOST = pd.DataFrame(predict_x_test)
```

In [40]:

```
#saving predict_x_test_result with .csv file.
predict_x_test_df_using_XGBOOST.to_csv("predict_test_df_using_XGBOOST.csv")
```

**********Thanks, Done by Deepak Kumar
**********

---

**********The End
**********

In [ ]: