Mercedes-Benz Greener Manufacturing

importing the library "pandas" and "numpay"

Project 1 - 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 wit
h the crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premi
um carmakers. 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 o
f their dreams.

    To ensure the safety and reliability of every unique car configuration before they hit the road, Daimler’s engineers have developed a robust testing system. As one of t
he world’s biggest manufacturers of premium cars, safety and efficiency are paramount on Daimler’s production lines. However, optimizing the speed of their testing system f
or 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-Ben
z 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 Daiml
er’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]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

importng the train data structure

In [2]:
```
train=pd.read_csv("train.csv")
Test = pd.read_csv("test.csv")
train
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	8405	107.39	ak	s	as	c	d	aa	d	q	...	1	0	0	0	0	0	0	0	0	0
4205	8406	108.77	j	o	t	d	d	aa	h	h	...	0	1	0	0	0	0	0	0	0	0
4206	8412	109.22	ak	v	r	a	d	aa	g	e	...	0	0	1	0	0	0	0	0	0	0
4207	8415	87.48	al	r	e	f	d	aa	l	u	...	0	0	0	0	0	0	0	0	0	0
4208	8417	110.85	z	r	ae	c	d	aa	g	w	...	1	0	0	0	0	0	0	0	0	0

4209 rows × 378 columns

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

In [3]:
```
train.var()
#  RangeIndex: 4209
#  Columns: 378
#  dtypes: float64(1) int64(369), object(8)
```

Out[3]:
```
ID      5.941936e+06
y       1.607667e+02
X10     1.313092e-02
X11     0.000000e+00
X12     6.945713e-02
            ...
X380    8.014579e-03
X382    7.546747e-03
X383    1.660732e-03
X384    4.750593e-04
X385    1.423823e-03
Length: 370, dtype: float64
```

In [4]:
```
Test.var()
```

Out[4]:
```
ID      5.871311e+06
X10     1.865006e-02
X11     2.375861e-04
X12     6.885074e-02
X13     5.734498e-02
            ...
X380    8.014579e-03
X382    8.715481e-03
X383    4.750593e-04
X384    7.124196e-04
X385    1.660732e-03
Length: 369, dtype: float64
```

In [5]:
```
n = pd.DataFrame((train.var()==0), columns=["a"])
dr = n[n.a==True].T
dr
```

Out[5]:

	X11	X93	X107	X233	X235	X268	X289	X290	X293	X297	X330	X347
a	True	True	True	True	True	True	True	True	True	True	True	True

In [6]:
```
list(dr.columns)
```

Out[6]:
```
['X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X289',
 'X290',
 'X293',
 'X297',
 'X330',
 'X347']
```

In [7]:
```
train.drop(columns=list(dr.columns),inplace=True)
Test.drop(columns=list(dr.columns),inplace=True)
```

In [8]:
```
train
```

Out[8]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	8405	107.39	ak	s	as	c	d	aa	d	q	...	1	0	0	0	0	0	0	0	0	0
4205	8406	108.77	j	o	t	d	d	aa	h	h	...	0	1	0	0	0	0	0	0	0	0
4206	8412	109.22	ak	v	r	a	d	aa	g	e	...	0	0	1	0	0	0	0	0	0	0
4207	8415	87.48	al	r	e	f	d	aa	l	u	...	0	0	0	0	0	0	0	0	0	0
4208	8417	110.85	z	r	ae	c	d	aa	g	w	...	1	0	0	0	0	0	0	0	0	0

4209 rows × 366 columns

2. Check for null and unique values for test and train sets

---------- For Train Data ----------

1 . find null values

In [9]:
```
#null values
train.isna().sum()
```

Out[9]:
```
ID      0
y       0
X0      0
X1      0
X2      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 366, dtype: int64
```

In [10]:
```
# pd.DataFrame(train.isna().sum()>0)
n1 = pd.DataFrame((train.isna().sum()>0), columns=["a"])
dr1 = n1[n1.a==True]
dr1
```

Out[10]:

a

hence there is no null values in train DataFrame

2. Unique Values of Train sets

In [11]:
```
train_columns = train.columns
train_columns
```

Out[11]:
```
Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=366)
```

In [12]:
```
columns=[]               # define list
counts=[]
unique =[]
colm =train.columns
for i in colm:
    columns.append(i)
    a = train[i].nunique()
    counts.append(a)
    b=train[i].unique()
    unique.append(b)
    print("The no. of unique values of {} column is = {} \n  & unique values of {} column is = \n {} \n".format(i,train[i].nunique(),i,train[i].unique()))
```

```
The no. of unique values of ID column is = 4209
  & unique values of ID column is =
 [   0    6    7 ... 8412 8415 8417]

The no. of unique values of y column is = 2545
  & unique values of y column is =
 [130.81  88.53  76.26 ...  85.71 108.77  87.48]

The no. of unique values of X0 column is = 47
  & unique values of X0 column is =
 ['k' 'az' 't' 'al' 'o' 'w' 'j' 'h' 's' 'n' 'ay' 'f' 'x' 'y' 'aj' 'ak' 'am'
 'z' 'q' 'at' 'ap' 'v' 'af' 'a' 'e' 'ai' 'd' 'aq' 'c' 'aa' 'ba' 'as' 'i'
 'r' 'b' 'ax' 'bc' 'u' 'ad' 'au' 'm' 'l' 'aw' 'ao' 'ac' 'g' 'ab']

The no. of unique values of X1 column is = 27
  & unique values of X1 column is =
 ['v' 't' 'w' 'b' 'r' 'l' 's' 'aa' 'c' 'a' 'e' 'h' 'z' 'j' 'o' 'u' 'p' 'n'
 'i' 'y' 'd' 'f' 'm' 'k' 'g' 'q' 'ab']
```

In [13]:
```
# Making DataFrame accoeding to unique value

train_unique_info =pd.DataFrame({"columns":columns,
              "counts":counts,
               "unique":unique})
```

In [14]:
```
train_unique_info.head()
```

Out[14]:

	columns	counts	unique
0	ID	4209	[0, 6, 7, 9, 13, 18, 24, 25, 27, 30, 31, 32, 3...
1	y	2545	[130.81, 88.53, 76.26, 80.62, 78.02, 92.93, 12...
2	X0	47	[k, az, t, al, o, w, j, h, s, n, ay, f, x, y, ...
3	X1	27	[v, t, w, b, r, l, s, aa, c, a, e, h, z, j, o,...
4	X2	44	[at, av, n, e, as, aq, r, ai, ak, m, a, k, ae,...

---------- For Test Data ----------

1 . find null values

In [15]:
```
Test.isnull().sum()
```

Out[15]:
```
ID      0
X0      0
X1      0
X2      0
X3      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 365, dtype: int64
```

hence there is no null values in test DataFrame

2. Unique Values of Train sets

In [16]:
```
test_columns = Test.columns
test_columns
```

Out[16]:
```
Index(['ID', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'X10',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=365)
```

In [20]:
```
columns_test=[]               # define list
counts_test=[]
unique_test =[]
colm =Test.columns
for i in colm:
    columns_test.append(i)
    a = Test[i].nunique()
    counts_test.append(a)
    b=Test[i].unique()
    unique_test.append(b)
    print("The no. of unique values of {} column is = {} \n  & unique values of {} column is = \n {} \n".format(i,Test[i].nunique(),i,Test[i].unique()))
```

```
The no. of unique values of ID column is = 4209
  & unique values of ID column is =
 [   1    2    3 ... 8413 8414 8416]

The no. of unique values of X0 column is = 49
  & unique values of X0 column is =
 ['az' 't' 'w' 'y' 'x' 'f' 'ap' 'o' 'ay' 'al' 'h' 'z' 'aj' 'd' 'v' 'ak'
 'ba' 'n' 'j' 's' 'af' 'ax' 'at' 'aq' 'av' 'm' 'k' 'a' 'e' 'ai' 'i' 'ag'
 'b' 'am' 'aw' 'as' 'r' 'ao' 'u' 'l' 'c' 'ad' 'au' 'bc' 'g' 'an' 'ae' 'p'
 'bb']

The no. of unique values of X1 column is = 27
  & unique values of X1 column is =
 ['v' 'b' 'l' 's' 'aa' 'r' 'a' 'i' 'p' 'c' 'o' 'm' 'z' 'e' 'h' 'w' 'g' 'k'
 'y' 't' 'u' 'd' 'j' 'q' 'n' 'f' 'ab']

The no. of unique values of X2 column is = 45
  & unique values of X2 column is =
 ['n' 'ai' 'as' 'ae' 's' 'b' 'e' 'ak' 'm' 'a' 'aq' 'ag' 'r' 'k' 'aj' 'ay'
```

In [21]:
```
# making DataFrame accoeding to unique value
test_unique_info =pd.DataFrame({"columns":columns_test,
                                "counts":counts_test,
                                "unique":unique_test})
```

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

Out[22]:

	columns	counts	unique
0	ID	4209	[1, 2, 3, 4, 5, 8, 10, 11, 12, 14, 15, 16, 17,...
1	X0	49	[az, t, w, y, x, f, ap, o, ay, al, h, z, aj, d...
2	X1	27	[v, b, l, s, aa, r, a, i, p, c, o, m, z, e, h,...
3	X2	45	[n, ai, as, ae, s, b, e, ak, m, a, aq, ag, r, ...
4	X3	7	[f, a, c, e, d, g, b]

3. Apply label encoder.

In [23]:
```
train.info(verbose=True)
# X0      object
# X1      object
# X2      object
# X3      object
# X4      object
# X5      object
# X6      object
# X8      object
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Data columns (total 366 columns):
ID      int64
y       float64
X0      object
X1      object
X2      object
X3      object
X4      object
X5      object
X6      object
X8      object
X10     int64
X12     int64
X13     int64
X14     int64
X15     int64
X16     int64
```

In [24]:
```
features = train.iloc[:,2:].values
label = train.iloc[:,1].values
test = Test.iloc[:,1:].values
features
```

Out[24]:
```
array([['k', 'v', 'at', ..., 0, 0, 0],
       ['k', 't', 'av', ..., 0, 0, 0],
       ['az', 'w', 'n', ..., 0, 0, 0],
       ...,
       ['ak', 'v', 'r', ..., 0, 0, 0],
       ['al', 'r', 'e', ..., 0, 0, 0],
       ['z', 'r', 'ae', ..., 0, 0, 0]], dtype=object)
```

In [25]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
LE = LabelEncoder()
features[:,0] = LE.fit_transform(features[:,0])
features[:,1] = LE.fit_transform(features[:,1])
features[:,2] = LE.fit_transform(features[:,2])
features[:,3] = LE.fit_transform(features[:,3])
features[:,4] = LE.fit_transform(features[:,4])
features[:,5] = LE.fit_transform(features[:,5])
features[:,6] = LE.fit_transform(features[:,6])
features[:,7] = LE.fit_transform(features[:,7])
features
test[:,0] = LE.fit_transform(test[:,0])
test[:,1] = LE.fit_transform(test[:,1])
test[:,2] = LE.fit_transform(test[:,2])
test[:,3] = LE.fit_transform(test[:,3])
test[:,4] = LE.fit_transform(test[:,4])
test[:,5] = LE.fit_transform(test[:,5])
test[:,6] = LE.fit_transform(test[:,6])
test[:,7] = LE.fit_transform(test[:,7])
test
```

Out[25]:
```
array([[21, 23, 34, ..., 0, 0, 0],
       [42, 3, 8, ..., 0, 0, 0],
       [21, 23, 17, ..., 0, 0, 0],
       ...,
       [47, 23, 17, ..., 0, 0, 0],
       [7, 23, 17, ..., 0, 0, 0],
       [42, 1, 8, ..., 0, 0, 0]], dtype=object)
```

In [26]:
```
test.shape
```

Out[26]:
```
(4209, 364)
```

In [27]:
```
features.shape
```

Out[27]:
```
(4209, 364)
```

4. Perform dimensionality reduction.

In [28]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
features = sc.fit_transform(features)
```

In [29]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
pca.fit(features,label)
```

Out[29]:
```
PCA(copy=True, iterated_power='auto', n_components=3, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [30]:
```
pca.explained_variance_ratio_
```

Out[30]:
```
array([0.06892669, 0.05688412, 0.04537457])
```

In [31]:
```
features1 = pca.transform(features)
```

5 Predict your test_df values using xgboost

In [32]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test  = train_test_split(features1 ,
                                                  label ,
                                                  test_size = 0.2,
                                                  random_state = 1)
```

In [33]:
```
from xgboost import XGBRFRegressor
xgb = XGBRFRegressor()
xgb.fit(features1,label)
```

```
[16:37:37] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[33]:
```
XGBRFRegressor(base_score=0.5, colsample_bylevel=1, colsample_bynode=0.8,
               colsample_bytree=1, gamma=0, learning_rate=1, max_delta_step=0,
               max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
               n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
               reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
               silent=None, subsample=0.8, verbosity=1)
```

In [34]:
```
xgb.score(features1,label)
```

Out[34]:
```
0.23688570776320028
```

In [35]:
```
xgb.score(x_test,y_test)
```

Out[35]:
```
0.24085395877279092
```

Test

In [36]:
```
t=sc.transform(test)
test1=pca.transform(t)

xgb.predict(test1)
```

Out[36]:
```
array([ 77.485634, 103.923965,  77.485634, ..., 102.710686, 102.710686,
        99.05701 ], dtype=float32)
```

hence variance is not greater than 70% or not greater than 50%

Quality of model is not good

5 .Predict your test_df values using xgboost

In [37]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features,
                                                 label,
                                                 test_size=0.2,
                                                 random_state=1)
```

In [38]:
```
from xgboost import XGBRFRegressor
model = XGBRFRegressor()
model.fit(X_train,y_train)
```

```
[16:37:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[38]:
```
XGBRFRegressor(base_score=0.5, colsample_bylevel=1, colsample_bynode=0.8,
               colsample_bytree=1, gamma=0, learning_rate=1, max_delta_step=0,
               max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
               n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
               reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
               silent=None, subsample=0.8, verbosity=1)
```

In [39]:
```
#Check the Quality of the model
print(model.score(X_train,y_train))
print(model.score(X_test,y_test))
```

```
0.5579728624404683
0.5835283609616132
```

In [40]:
```
model.predict(test)
```

Out[40]:
```
array([108.26955 ,  94.1049  , 108.26955 , ...,  94.1049  , 112.576195,
        94.1049  ], dtype=float32)
```