ercedes-benz-greener-manufacturing

September 8, 2023

0.0.1 Importing Libraries and Datasets

[1]:
```
# Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
# Reading the dataset
df1 = pd.read_csv('train.csv')
df2 = pd.read_csv('test.csv')
```

[3]:
```
df1.head()
```

[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	…	X375	X376	X377	X378	X379	\
0	0	130.81	k	v	at	a	d	u	j	o	…	0	0	1	0	0	
1	6	88.53	k	t	av	e	d	y	l	o	…	1	0	0	0	0	
2	7	76.26	az	w	n	c	d	x	j	x	…	0	0	0	0	0	
3	9	80.62	az	t	n	f	d	x	l	e	…	0	0	0	0	0	
4	13	78.02	az	v	n	f	d	h	d	n	…	0	0	0	0	0	

	X380	X382	X383	X384	X385
0	0	0	0	0	0
1	0	0	0	0	0
2	0	1	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

[5 rows x 378 columns]

[4]:
```
df2.head()
```

[4]:

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

```
   X382  X383  X384  X385
0     0     0     0     0
1     0     0     0     0
2     0     0     0     0
3     0     0     0     0
4     0     0     0     0

[5 rows x 377 columns]
```

0.0.2 Data Exploration

[5]:
```
# Describe the dataset
df1.describe()
```

[5]:
```
                ID            y           X10     X11          X12  \
count  4209.000000  4209.000000  4209.000000  4209.0  4209.000000
mean   4205.960798   100.669318     0.013305     0.0     0.075077
std    2437.608688    12.679381     0.114590     0.0     0.263547
min       0.000000    72.110000     0.000000     0.0     0.000000
25%    2095.000000    90.820000     0.000000     0.0     0.000000
50%    4220.000000    99.150000     0.000000     0.0     0.000000
75%    6314.000000   109.010000     0.000000     0.0     0.000000
max    8417.000000   265.320000     1.000000     0.0     1.000000

                X13          X14          X15          X16          X17  …  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000  …
mean      0.057971     0.428130     0.000475     0.002613     0.007603  …
std       0.233716     0.494867     0.021796     0.051061     0.086872  …
min       0.000000     0.000000     0.000000     0.000000     0.000000  …
25%       0.000000     0.000000     0.000000     0.000000     0.000000  …
50%       0.000000     0.000000     0.000000     0.000000     0.000000  …
75%       0.000000     1.000000     0.000000     0.000000     0.000000  …
max       1.000000     1.000000     1.000000     1.000000     1.000000  …

               X375         X376         X377         X378         X379  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean      0.318841     0.057258     0.314802     0.020670     0.009503
std       0.466082     0.232363     0.464492     0.142294     0.097033
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       1.000000     0.000000     1.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

               X380         X382         X383         X384         X385
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
```

```
mean      0.008078     0.007603     0.001663     0.000475     0.001426
std       0.089524     0.086872     0.040752     0.021796     0.037734
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     0.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

[8 rows x 370 columns]
```

[6]:
```
df2.describe()
```

[6]:
```
                ID          X10          X11          X12          X13  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean   4211.039202     0.019007     0.000238     0.074364     0.061060
std    2423.078926     0.136565     0.015414     0.262394     0.239468
min       1.000000     0.000000     0.000000     0.000000     0.000000
25%    2115.000000     0.000000     0.000000     0.000000     0.000000
50%    4202.000000     0.000000     0.000000     0.000000     0.000000
75%    6310.000000     0.000000     0.000000     0.000000     0.000000
max    8416.000000     1.000000     1.000000     1.000000     1.000000

               X14          X15          X16          X17          X18  …  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000  …
mean      0.427893     0.000713     0.002613     0.008791     0.010216  …
std       0.494832     0.026691     0.051061     0.093357     0.100570  …
min       0.000000     0.000000     0.000000     0.000000     0.000000  …
25%       0.000000     0.000000     0.000000     0.000000     0.000000  …
50%       0.000000     0.000000     0.000000     0.000000     0.000000  …
75%       1.000000     0.000000     0.000000     0.000000     0.000000  …
max       1.000000     1.000000     1.000000     1.000000     1.000000  …

              X375         X376         X377         X378         X379  \
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean      0.325968     0.049656     0.311951     0.019244     0.011879
std       0.468791     0.217258     0.463345     0.137399     0.108356
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       1.000000     0.000000     1.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

              X380         X382         X383         X384         X385
count  4209.000000  4209.000000  4209.000000  4209.000000  4209.000000
mean      0.008078     0.008791     0.000475     0.000713     0.001663
std       0.089524     0.093357     0.021796     0.026691     0.040752
min       0.000000     0.000000     0.000000     0.000000     0.000000
```

```
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.000000     0.000000     0.000000     0.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

[8 rows x 369 columns]
```

[7]:
```
df1.shape
```

[7]:
```
(4209, 378)
```

[8]:
```
df2.shape
```

[8]:
```
(4209, 377)
```

As stated in the question, if for any column(s), the variance is equal to zero, then you need to remove those variable(s).

[9]:
```
variance = pow(df1.drop(columns={'ID','y'}).std(),2).to_dict()

variance_is_zero_count = 0
for key, value in variance.items():
    if(value==0):
        print('Name = ',key)
        variance_is_zero_count = variance_is_zero_count+1
print('No of columns which has zero variance = ', variance_is_zero_count)
```

```
Name =  X11
Name =  X93
Name =  X107
Name =  X233
Name =  X235
Name =  X268
Name =  X289
Name =  X290
Name =  X293
Name =  X297
Name =  X330
Name =  X347
No of columns which has zero variance =  12
```

```
C:\Users\lrnem\AppData\Local\Temp\ipykernel_4380\1335638582.py:1: FutureWarning:
The default value of numeric_only in DataFrame.std is deprecated. In a future
version, it will default to False. In addition, specifying 'numeric_only=None'
is deprecated. Select only valid columns or specify the value of numeric_only to
silence this warning.
  variance = pow(df1.drop(columns={'ID','y'}).std(),2).to_dict()
```

[10]:
```
df1 = df1.drop(columns={'X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289',␣
↪'X290', 'X293', 'X297', 'X330', 'X347'})
```

[11]:
```
df1.shape
```

[11]:
```
(4209, 366)
```

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

[12]:
```
check1 = df1.isnull().sum()
```

[13]:
```
check1.any()
```

[13]:
```
False
```

[14]:
```
check2 = df2.isnull().sum()
```

[15]:
```
check2.any()
```

[15]:
```
False
```

0.0.3 Label encoding

[16]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
```

[17]:
```
x1 = df1.drop(columns={'y', 'ID'}, axis=1)
y1 = df1['y']
```

[18]:
```
x1.shape
```

[18]:
```
(4209, 364)
```

[19]:
```
y1.shape
```

[19]:
```
(4209,)
```

[20]:
```
x1.describe(include=object)
```

[20]:

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	47	27	44	7	4	29	12	25
top	z	aa	as	c	d	w	g	j
freq	360	833	1659	1942	4205	231	1042	277

[21]:
```
x1['X0'] = label_encoder.fit_transform(x1.X0)
x1['X1'] = label_encoder.fit_transform(x1.X1)
```

```
x1['X2'] = label_encoder.fit_transform(x1.X2)
x1['X3'] = label_encoder.fit_transform(x1.X3)
x1['X4'] = label_encoder.fit_transform(x1.X4)
x1['X5'] = label_encoder.fit_transform(x1.X5)
x1['X6'] = label_encoder.fit_transform(x1.X6)
x1['X8'] = label_encoder.fit_transform(x1.X8)
```

0.0.4 Dimensionality reduction

[22]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.90)
```

[23]:
```
pca.fit(x1, y1)
```

[23]:
```
PCA(n_components=0.9)
```

[24]:
```
x1_transformed = pca.fit_transform(x1)
x1_transformed.shape
```

[24]:
```
(4209, 5)
```

Predict your test_df values using XGBoost

[25]:
```
from math import sqrt
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
import xgboost as xgb
```

[26]:
```
x_train, x_test, y_train, y_test = train_test_split(x1_transformed, y1,␣
↪test_size=.1, random_state=3)
```

[27]:
```
x_train.shape, x_test.shape
```

[27]:
```
((3788, 5), (421, 5))
```

[28]:
```
y_train.shape, y_test.shape
```

[28]:
```
((3788,), (421,))
```

[29]:
```
dtrain_reg = xgb.DMatrix(x_train, y_train, enable_categorical=True)
dtest_reg = xgb.DMatrix(x_test, y_test, enable_categorical=True)
```

[30]:
```
params = {"objective": "reg:squarederror", "tree_method": "gpu_hist"}
```

[31]:
```
params = {"objective": "reg:squarederror", "tree_method": "gpu_hist"}

n = 300
```

```
xgb1 = xgb.train(params=params, dtrain=dtrain_reg, num_boost_round=n,)
```

[32]:
```
from sklearn.metrics import mean_squared_error

pred = xgb1.predict(dtest_reg)
```

[33]:
```
rmse = mean_squared_error(y_test, pred, squared=False)

print(f"Base model RMSE val : {rmse:.3f}")
```

```
Base model RMSE val : 10.211
```

[34]:
```
params = {"objective": "reg:squarederror", "tree_method": "gpu_hist"}
n = 1000
results = xgb.cv(params, dtrain_reg, num_boost_round=n, nfold=5,␣
 ↪early_stopping_rounds=20)
```

[35]:
```
results.head()
```

[35]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	71.148586	0.139979	71.156159	0.752683
1	50.469313	0.122809	50.496843	0.822578
2	36.170590	0.123311	36.214890	0.906990
3	26.413531	0.136587	26.525787	0.959477
4	19.851765	0.161268	20.093561	1.039389

[36]:
```
best_rmse = results['test-rmse-mean'].min()
```

[37]:
```
best_rmse
```

[37]:
```
10.195234054808457
```

Using k-fold cross validation, RMSE comes as 10.195.

0.0.5 Testing dataset

[38]:
```
df2 = df2.
 ↪drop(columns={'X11','X93','X107','X233','X235','X268','X289','X290','X293','X297','X330','X
```

[39]:
```
df2.shape
```

[39]:
```
(4209, 365)
```

[40]:
```
check3 = df2.isnull().sum()
```

[41]:
```
check3.any()
```

[41]:
```
False
```

[42]:
```
x2 = df2.drop(columns={'ID'})
print(x2.shape)
```

```
(4209, 364)
```

[43]:
```
x2.describe(include='object')
```

[43]:

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	49	27	45	7	4	32	12	25
top	ak	aa	as	c	d	v	g	e
freq	432	826	1658	1900	4203	246	1073	274

[44]:
```
x2['X0'] = label_encoder.fit_transform(x2.X0)
x2['X1'] = label_encoder.fit_transform(x2.X1)
x2['X2'] = label_encoder.fit_transform(x2.X2)
x2['X3'] = label_encoder.fit_transform(x2.X3)
x2['X4'] = label_encoder.fit_transform(x2.X4)
x2['X5'] = label_encoder.fit_transform(x2.X5)
x2['X6'] = label_encoder.fit_transform(x2.X6)
x2['X8'] = label_encoder.fit_transform(x2.X8)
```

[45]:
```
pca.fit(x2)
```

[45]:
```
PCA(n_components=0.9)
```

[46]:
```
x2_transformed = pca.fit_transform(x2)
x2_transformed.shape
```

[46]:
```
(4209, 5)
```

[47]:
```
dtest = xgb.DMatrix(x2_transformed)
```

[48]:
```
test_pred = xgb1.predict(dtest)
test_pred
```

[48]:
```
array([ 73.144936,  95.4681  ,  95.36935 , …,  91.54465 , 108.6302  ,
        96.124054], dtype=float32)
```

[ ]:
```

```