Mercedes-Benz Greener Manufacturing Project

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import LabelEncoder
```

In [4]:

```
# read in train and test data
data_train= pd.read_csv('mbtrain.csv')
data_test = pd.read_csv('mbtest.csv')

print(train.shape, test.shape)
```

```
(4209, 378) (4209, 377)
```

In [5]:

```
data_train.head()
```

Out[5]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0

5 rows × 378 columns

In [6]:

```
data_train.shape
```

Out[6]:

```
(4209, 378)
```

In [7]:

```
##chech missing value
data_train.isna().sum()
```

Out[7]:

```
ID      0
y       0
X0      0
X1      0
X2      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 378, dtype: int64
```

In [8]:

```
data_train.nunique()
```

Out[8]:

```
ID      4209
y       2545
X0        47
X1        27
X2        44
        ... 
X380       2
X382       2
X383       2
X384       2
X385       2
Length: 378, dtype: int64
```

In [9]:

```
data_train.describe()
```

Out[9]:

	ID	y	X10	X11	X12	X13	X1
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.00000
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.42813
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.49486
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.00000
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.00000
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.00000
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.00000
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.00000

8 rows × 370 columns

In [11]:

```
## try find variance of column
print(data_train.var)
```

```
<bound method DataFrame.var of         ID       y  X0 X1  X2 X3 X4  X5 X6
X8  ...  X375  X376  X377  X378  \
0        0  130.81   k  v  at  a  d   u  j  o  ...     0     0     1     0
1        6   88.53   k  t  av  e  d   y  l  o  ...     1     0     0     0
2        7   76.26  az  w   n  c  d   x  j  x  ...     0     0     0     0
3        9   80.62  az  t   n  f  d   x  l  e  ...     0     0     0     0
4       13   78.02  az  v   n  f  d   h  d  n  ...     0     0     0     0
...    ...     ...  .. ..  .. .. ..  .. .. ..  ...   ...   ...   ...   ...
4204  8405  107.39  ak  s  as  c  d  aa  d  q  ...     1     0     0     0
4205  8406  108.77   j  o   t  d  d  aa  h  h  ...     0     1     0     0
4206  8412  109.22  ak  v   r  a  d  aa  g  e  ...     0     0     1     0
4207  8415   87.48  al  r   e  f  d  aa  l  u  ...     0     0     0     0
4208  8417  110.85   z  r  ae  c  d  aa  g  w  ...     1     0     0     0

      X379  X380  X382  X383  X384  X385
0        0     0     0     0     0     0
1        0     0     0     0     0     0
2        0     0     1     0     0     0
3        0     0     0     0     0     0
4        0     0     0     0     0     0
...    ...   ...   ...   ...   ...   ...
4204     0     0     0     0     0     0
4205     0     0     0     0     0     0
4206     0     0     0     0     0     0
4207     0     0     0     0     0     0
4208     0     0     0     0     0     0

[4209 rows x 378 columns]>
```

In [12]:

```
cols=[c for c in data_train.columns if 'X' in c]
print('Number of features: {}'.format(len(cols)))
```

```
Number of features: 376
```

In [13]:

```
print('Feature types:')
data_train[cols].dtypes.value_counts()
```

```
Feature types:
```

Out[13]:

```
int64     368
object      8
dtype: int64
```

In [14]:

```
##Count the data in each of the columns
counts = [[], [], []]
for c in cols:
    typ = data_train[c].dtype
    uniq = len(np.unique(data_train[c]))
    if uniq == 1:
        counts[0].append(c)
    elif uniq == 2 and typ == np.int64:
        counts[1].append(c)
    else:
        counts[2].append(c)

print('Constant features: {} Binary features: {} Categorical features: {}\n'
      .format(*[len(c) for c in counts]))
print('Constant features:', counts[0])
print('Categorical features:', counts[2])
```

```
Constant features: 12 Binary features: 356 Categorical features: 8

Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289',
'X290', 'X293', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

In [16]:

```
# the prediction output
y_train = data_train['y'].values
y_train
```

Out[16]:

```
array([130.81,  88.53,  76.26, ..., 109.22,  87.48, 110.85])
```

In [17]:

```
##remove columns ID and Y from the data
remove_columns = list(set(data_train.columns) - set(['ID', 'y']))
y_train = data_train['y'].values
id_test = data_test['ID'].values

x_train = data_train[remove_columns]
x_test = data_test[remove_columns]
```

In [19]:

```
#Check for null and unique values for tain and test dataset
def check_missing_values(df):
    if df.isnull().any().any():
        print("There are missing values in the dataframe")
    else:
        print("There are no missing values in the dataframe")
```

In [20]:

```
##check null in train dataest
check_missing_values(x_train)
```

```
There are no missing values in the dataframe
```

In [21]:

```
## check null in test dataset
check_missing_values(x_test)
```

```
There are no missing values in the dataframe
```

In [22]:

```
## If for any column(s), the variance is equal to zero
## Apply label encoder
for column in remove_columns:
    val = len(np.unique(x_train[column]))
    if val == 1:
        x_train.drop(column, axis=1) # Column with only one
        # value is useless so we drop it
        x_test.drop(column, axis=1)
    if val > 2: # Column is categorical
        mapper = lambda x: sum([ord(digit) for digit in x])
        x_train[column] = x_train[column].apply(mapper)
        x_test[column] = x_test[column].apply(mapper)
x_train.head()
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:11: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  # This is added back by InteractiveShellApp.init_path()
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:12: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
```

Out[22]:

	X327	X297	X353	X382	X99	X292	X115	X257	X252	X89	...	X66	X203	X321	X19
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	1	0	0	0	0	1	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	1	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	1	0	...	0	0	0	

5 rows × 376 columns

In [23]:

```
print('Feature types:')
x_train[cols].dtypes.value_counts()
```

```
Feature types:
```

Out[23]:

```
int64    376
dtype: int64
```

In [24]:

```
## Perform dimensionality reduction (PCA)

from sklearn.decomposition import PCA
```

In [25]:

```
pca = PCA(n_components=12, random_state=420)
pca2_results_train = pca.fit_transform(x_train)
pca2_results_test = pca.transform(x_test)
```

In [26]:

```
## Training using xgboost
import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [27]:

```
x_train, x_valid, y_train, y_valid = train_test_split(
        pca2_results_train,
        y_train, test_size=0.2,
        random_state=4242)
```

In [28]:

```
d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)
d_test = xgb.DMatrix(pca2_results_test)
```

In [29]:

```
params = {}
params['objective'] = 'reg:linear'
params['eta'] = 0.02
params['max_depth'] = 4

def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)

watchlist = [(d_train, 'train'), (d_valid, 'valid')]
```

In [30]:

```
clf = xgb.train(params, d_train,
                1000, watchlist, early_stopping_rounds=50,
                feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[06:03:51] WARNING: /workspace/src/objective/regression_obj.cu:167: reg:li
near is now deprecated in favor of reg:squarederror.
[0]     train-rmse:99.14835     valid-rmse:98.26297     train-r2:-58.35295
valid-r2:-67.63754
Multiple eval metrics have been passed: 'valid-r2' will be used for early
stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[10]    train-rmse:81.27653     valid-rmse:80.36433     train-r2:-38.88428
valid-r2:-44.91014
[20]    train-rmse:66.71610     valid-rmse:65.77334     train-r2:-25.87403
valid-r2:-29.75260
[30]    train-rmse:54.86957     valid-rmse:53.88973     train-r2:-17.17752
valid-r2:-19.64401
[40]    train-rmse:45.24491     valid-rmse:44.21970     train-r2:-11.35979
valid-r2:-12.89996
[50]    train-rmse:37.44729     valid-rmse:36.37237     train-r2:-7.46666
valid-r2:-8.40428
[60]    train-rmse:31.14748     valid-rmse:30.01874     train-r2:-4.85757
valid-r2:-5.40570
[70]    train-rmse:26.08660     valid-rmse:24.90890     train-r2:-3.10872
valid-r2:-3.41053
[80]    train-rmse:22.04638     valid-rmse:20.83274     train-r2:-1.93458
valid-r2:-2.08514
[90]    train-rmse:18.84403     valid-rmse:17.60316     train-r2:-1.14397
valid-r2:-1.20274
[100]   train-rmse:16.33631     valid-rmse:15.08444     train-r2:-0.61131
valid-r2:-0.61749
[110]   train-rmse:14.40372     valid-rmse:13.14818     train-r2:-0.25262
valid-r2:-0.22889
[120]   train-rmse:12.92871     valid-rmse:11.68941     train-r2:-0.00921
valid-r2:0.02867
[130]   train-rmse:11.80812     valid-rmse:10.61535     train-r2:0.15815
valid-r2:0.19897
[140]   train-rmse:10.98603     valid-rmse:9.84998      train-r2:0.27129
valid-r2:0.31031
[150]   train-rmse:10.37399     valid-rmse:9.32204      train-r2:0.35023
valid-r2:0.38226
[160]   train-rmse:9.92029      valid-rmse:8.95919      train-r2:0.40582
valid-r2:0.42942
[170]   train-rmse:9.59071      valid-rmse:8.71397      train-r2:0.44464
valid-r2:0.46022
[180]   train-rmse:9.34334      valid-rmse:8.55560      train-r2:0.47292
valid-r2:0.47967
[190]   train-rmse:9.15814      valid-rmse:8.45152      train-r2:0.49361
valid-r2:0.49225
[200]   train-rmse:9.01373      valid-rmse:8.38985      train-r2:0.50945
valid-r2:0.49963
[210]   train-rmse:8.90228      valid-rmse:8.34352      train-r2:0.52151
valid-r2:0.50514
[220]   train-rmse:8.82529      valid-rmse:8.32079      train-r2:0.52975
valid-r2:0.50783
[230]   train-rmse:8.76744      valid-rmse:8.30674      train-r2:0.53589
valid-r2:0.50950
[240]   train-rmse:8.71781      valid-rmse:8.29981      train-r2:0.54113
valid-r2:0.51031
[250]   train-rmse:8.67893      valid-rmse:8.29033      train-r2:0.54522
valid-r2:0.51143
[260]   train-rmse:8.64604      valid-rmse:8.28552      train-r2:0.54866
valid-r2:0.51200
[270]   train-rmse:8.61700      valid-rmse:8.28521      train-r2:0.55168
```

```
valid-r2:0.51204
[280]	train-rmse:8.58766	valid-rmse:8.28510	train-r2:0.55473
valid-r2:0.51205
[290]	train-rmse:8.55980	valid-rmse:8.28708	train-r2:0.55762
valid-r2:0.51181
[300]	train-rmse:8.53510	valid-rmse:8.28718	train-r2:0.56017
valid-r2:0.51180
[310]	train-rmse:8.50959	valid-rmse:8.28848	train-r2:0.56279
valid-r2:0.51165
[320]	train-rmse:8.48800	valid-rmse:8.28924	train-r2:0.56501
valid-r2:0.51156
Stopping. Best iteration:
[272]	train-rmse:8.60985	valid-rmse:8.28187	train-r2:0.55243
valid-r2:0.51243
```

In [31]:

```
## Predict  data_test values using xgboost
p_test = clf.predict(d_test)

sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = p_test
sub.to_csv('xgb.csv', index=False)
```

In [32]:

```
p_test
```

Out[32]:

```
array([ 82.87171 ,  97.357605,  83.43425 , ...,  98.78521 , 107.399536,
        96.946365], dtype=float32)
```