Mercedes-Benz Greener Manufacturing

In [172…
```
# Step1: Import the required libraries
```

In [105…
```
# linear algebra
# data processing, CSV file I/O (e.g. pd.read_csv)
# for dimensionality reduction
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
```

In [106…
```
# Step2: Read the data from train.csv
```

In [107…
```
df_train = pd.read_csv('train.csv')
```

In [108…
```
df_train.head(100)
```

Out[108…

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	207	115.43	x	b	m	c	d	j	j	n	...	0	0	1	0	0	0	0	0	0	0
96	208	91.94	n	l	as	f	d	j	d	n	...	0	0	0	0	0	0	0	0	0	0
97	209	108.37	x	aa	as	d	d	j	i	s	...	0	1	0	0	0	0	0	0	0	0
98	212	127.66	x	b	m	c	d	j	j	n	...	0	0	1	0	0	0	0	0	0	0
99	213	93.62	f	s	m	c	d	j	j	a	...	0	0	1	0	0	0	0	0	0	0

100 rows × 378 columns

In [109…
```
# let us understand the data
```

In [110…
```
print('Size of training set: {} rows and {} columns  '
      .format(*df_train.shape))
```

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

In [111…
```
# Step3: Collect the Y values into an array
# seperate the y from the data as we will use this to learn as
# the prediction output
```

In [112…
```
y_train = df_train['y'].values
```

In [113…
```
y_train
```

Out[113…
```
array([130.81,  88.53,  76.26, ..., 109.22,  87.48, 110.85])
```

In [114…
```
y_test.shape
```

Out[114…
```
(4209,)
```

In [115…
```
# Step4: Understand the data types we have

# iterate through all the columns which has X in the name of the column
```

In [116…
```
cols = [c for c in df_train.columns if 'X' in c]
print('Number of features: {}'.format(len(cols)))
```

```
Number of features: 376
```

In [117…
```
print('Feature types:')
df_train[cols].dtypes.value_counts()
```

Out[117…
```
Feature types:
int64     368
object      8
dtype: int64
```

In [118…
```
# Step5: Count the data in each of the columns
```

In [119…
```
counts = [[], [], []]
for c in cols:
    typ = df_train[c].dtype
    uniq = len(np.unique(df_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 [120…
```
# Step6: Read the test.csv data
```

In [121…
```
df_test = pd.read_csv('test.csv')
```

In [122…
```
df_test.head()
```

Out[122…

	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 [123…
```
print('Size of training set: {} rows and {} columns'
      .format(*df_test.shape))
```

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

In [124…
```
# remove columns ID and Y from the data as they are not used for learning
```

In [125…
```
usable_columns = list(set(df_train.columns) - set(['ID','y']))
y_train = df_train['y'].values
id_test = df_test['ID'].values

x_train = df_train[usable_columns]
x_test = df_test[usable_columns]
```

In [126…
```
x_train
```

Out[126…

	X361	X282	X153	X166	X135	X144	X68	X309	X356	X168	...	X137	X237	X44	X384	X235	X301	X245	X224	X165	X307
0	1	0	0	0	0	1	1	0	0	0	...	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0
2	1	0	0	0	0	1	1	0	0	0	...	1	0	0	0	0	0	0	1	0	0
3	1	0	0	1	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	1	0	0	1	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	1	0	0	0	0	1	0	0	0	0	...	1	0	0	0	0	0	0	1	0	0
4205	1	0	0	0	0	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	0
4206	1	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4207	1	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	1	0	0	0	0
4208	1	0	0	0	0	1	0	0	0	1	...	1	0	0	0	0	0	0	0	0	0

4209 rows × 376 columns

In [127…
```
x_test
```

Out[127…

	X361	X282	X153	X166	X135	X144	X68	X309	X356	X168	...	X137	X237	X44	X384	X235	X301	X245	X224	X165	X307
0	1	0	0	1	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	1	0	...	1	0	0	0	0	0	0	0	0	0
2	1	0	0	1	0	1	0	0	0	0	...	1	0	0	0	0	0	0	1	0	0
3	1	0	0	1	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	1	0	0	0	0	...	1	0	0	0	0	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	1	0	0	0	0	1	0	0	0	0	...	1	0	0	0	0	0	0	1	0	0
4205	1	0	0	0	0	1	1	0	0	0	...	1	0	0	0	0	0	0	0	0	0
4206	1	0	0	0	0	1	0	0	0	1	...	0	0	0	0	0	1	0	1	0	0
4207	1	0	0	0	0	1	0	0	0	0	...	1	0	0	0	0	0	0	1	0	0
4208	1	0	0	0	0	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	0

4209 rows × 376 columns

In [128…
```
# Step7: Check for null and unique values for test and train sets
```

In [129…
```
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")
check_missing_values(x_train)
check_missing_values(x_test)
```

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

In [130…
```
# Step8: If for any column(s), the variance is equal to zero,
# then you need to remove those variable(s).
# Apply label encoder
```

In [131…
```
for column in usable_columns:
    cardinality = len(np.unique(x_train[column]))
    if cardinality == 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 cardinality > 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()
```

```
C:\Users\Chandu\AppData\Local\Temp/ipykernel_4056/1478267456.py:9: SettingWithCopyWarning:
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-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x_train[column] = x_train[column].apply(mapper)
C:\Users\Chandu\AppData\Local\Temp/ipykernel_4056/1478267456.py:10: SettingWithCopyWarning:
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-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  x_test[column] = x_test[column].apply(mapper)
```

Out[131…

	X361	X282	X153	X166	X135	X144	X68	X309	X356	X168	...	X137	X237	X44	X384	X235	X301	X245	X224	X165	X307
0	1	0	0	0	0	1	1	0	0	0	...	1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0
2	1	0	0	0	0	1	1	0	0	0	...	1	0	0	0	0	0	0	1	0	0
3	1	0	0	1	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	1	0	0	1	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 376 columns

In [132…
```
# Step9: Make sure the data is now changed into numericals
```

In [133…
```
print('Feature types:')
x_train[cols].dtypes.value_counts()
```

Out[133…
```
Feature types:
int64    376
dtype: int64
```

In [134…
```
# Step10: Perform dimensionality reduction
# Linear dimensionality reduction using Singular Value Decomposition of
# the data to project it to a lower dimensional space.
```

In [135…
```
n_comp = 12
pca = PCA(n_components=n_comp, random_state=420)
pca2_results_train = pca.fit_transform(x_train)
pca2_results_test = pca.transform(x_test)
```

In [136…
```
pca2_results_train
```

Out[136…
```
array([[-49.08156207,  -4.90948084, -17.25085325, ...,   1.65805712,
          0.93303531,   1.67848456],
       [-48.94680383,  -7.22674339, -13.7631947 , ...,  -0.21429864,
          0.10920573,   0.44974261],
       [ 92.62761708,  31.9940341 , -26.17503456, ...,  -0.62195837,
          2.92587012,  -0.52754773],
       ...,
       [ 89.47970814,  20.44554421,  48.11999819, ...,  -1.27196642,
         -0.28718933,   2.00808361],
       [ 96.97110845,  31.50977186,  49.20059282, ...,   0.14365692,
         -0.9799505 ,   0.99156917],
       [-17.21024322, -14.22166025,  55.38091289, ...,  -0.2890469 ,
         -0.31643971,   0.6915163 ]])
```

In [137…
```
pca2_results_test
```

Out[137…
```
array([[ 9.22615149e+01,  3.29260839e+01, -3.01130736e+01, ...,
        -4.11416321e-01,  3.62102327e+00, -1.20766837e+00],
       [-3.48622379e+01,  6.87132606e+00, -3.74760829e+01, ...,
         6.09269213e-01, -6.95819541e-01, -4.24894953e-01],
       [ 4.36560426e+01, -5.05939489e+01, -6.10591086e+01, ...,
        -3.20457621e-01,  2.60146970e+00, -1.53755054e+00],
       ...,
       [-2.52437784e+01, -2.63794193e+01,  5.40742341e+01, ...,
         6.03526894e-01,  2.61297514e-02,  3.67043352e-02],
       [ 4.53823778e+01, -6.38062446e+01,  3.58666036e+01, ...,
        -9.15186916e-01, -6.72295847e-01,  5.15271534e-01],
       [-4.23807477e+01, -2.52862351e+01,  6.10815522e+01, ...,
        -2.98847672e-01, -9.77121161e-01,  5.35340845e-02]])
```

In [138…
```
# Step11: Training using xgboost
```

In [ ]:
```
import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

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

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

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')]

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

In [97]:
```
# Step12: Predict your test_df values using xgboost
```

In [ ]:
```
p_test = clf.predict(d_test)

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

sub.head()
```