Mercedes-Benz Greener Manufacturing project

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]:

```
# Importing necessary libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
```

In [2]:

```
# Importing the dataset

train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
```

In [3]:

```
# Number of rows and columns in train dataset
print(train_df.shape)

# Number of rows and columns in test dataset
print(test_df.shape)
```

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

In [4]:

```
train_df.head()
```

Out[4]:

	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 [5]:

```
test_df.head()
```

Out[5]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X3
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	

5 rows × 377 columns

In [6]:

```
# Description of data in train dataset

train_df.describe()
```

Out[6]:

	ID	y	X10	X11	X12	X13	X14
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 [7]:

```
# Description of data in test dataset

test_df.describe()
```

Out[7]:

	ID	X10	X11	X12	X13	X14	
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.(
mean	4211.039202	0.019007	0.000238	0.074364	0.061060	0.427893	0.(
std	2423.078926	0.136565	0.015414	0.262394	0.239468	0.494832	0.(
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(
25%	2115.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(
50%	4202.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(
75%	6310.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.(
max	8416.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.(

8 rows × 369 columns

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

In [8]:

```
# Checking the variance in train dataset

train_df.var()
```

Out[8]:

```
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 [9]:

```
# To find out if variance is equal to zero for any columns

(train_df.var() == 0)
```

Out[9]:

```
ID       False
y        False
X10      False
X11       True
X12      False
         ...
X380     False
X382     False
X383     False
X384     False
X385     False
Length: 370, dtype: bool
```

In [10]:

```
# Finding out the columns with zero variance

variance_with_zero = train_df.var()[train_df.var()==0].index.values
variance_with_zero
```

Out[10]:

```
array(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290',
       'X293', 'X297', 'X330', 'X347'], dtype=object)
```

In [11]:

```
# Dropping columns with zero variance variables in train dataset

train_df = train_df.drop(variance_with_zero, axis=1)
```

In [12]:

```
train_df.shape
```

Out[12]:

```
(4209, 366)
```

In [13]:

```
# Dropping columns with zero variance variables in test dataset

test_df = test_df.drop(variance_with_zero, axis=1)
test_df.shape
```

Out[13]:

```
(4209, 365)
```

In [14]:

```
# Dropping ID column from both datasets as its not needed

train_df = train_df.drop(['ID'], axis=1)
test_df = test_df.drop(['ID'], axis=1)
```

Check for null and unique values for test and train sets

In [15]:

```
# Null values in train dataset

train_df.isnull().sum().values
```

Out[15]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [16]:

```
train_df.isnull().any()
```

Out[16]:

```
y       False
X0      False
X1      False
X2      False
X3      False
        ...
X380    False
X382    False
X383    False
X384    False
X385    False
Length: 365, dtype: bool
```

In [17]:

```
# Null values in test dataset

test_df.isnull().sum().values
```

Out[17]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [18]:

```
test_df.isnull().any()
```

Out[18]:

```
X0      False
X1      False
X2      False
X3      False
X4      False
        ...
X380    False
X382    False
X383    False
X384    False
X385    False
Length: 364, dtype: bool
```

Finding unique values in the dataset

In [19]:

```
train_df.nunique()
```

Out[19]:

```
y        2545
X0         47
X1         27
X2         44
X3          7
         ... 
X380        2
X382        2
X383        2
X384        2
X385        2
Length: 365, dtype: int64
```

In [20]:

```
test_df.nunique()
```

Out[20]:

```
X0      49
X1      27
X2      45
X3       7
X4       4
        ..
X380     2
X382     2
X383     2
X384     2
X385     2
Length: 364, dtype: int64
```

Apply label encoder

In [118]:

```
# to find out the columns having object datatype

object_dt = train_df.select_dtypes(include=[object])
object_dt
```

Out[118]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	k	v	at	a	d	u	j	o
1	k	t	av	e	d	y	l	o
2	az	w	n	c	d	x	j	x
3	az	t	n	f	d	x	l	e
4	az	v	n	f	d	h	d	n
...	...	...	...	...	...	...	...	...
4204	ak	s	as	c	d	aa	d	q
4205	j	o	t	d	d	aa	h	h
4206	ak	v	r	a	d	aa	g	e
4207	al	r	e	f	d	aa	l	u
4208	z	r	ae	c	d	aa	g	w

4209 rows × 8 columns

In [119]:

```
object_dt_columns = object_dt.columns
object_dt_columns
```

Out[119]:

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

In [120]:

```
# Applying Label encoding to train dataset

label_encoder = preprocessing.LabelEncoder()

# Encoding and transforming object data to interger

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

In [121]:

```
# Applying Label encoding to test dataset

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

Perform dimensionality reduction

In [122]:

```
# Importing necessary libraries

import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import Normalizer
```

In [124]:

```
# Normalizing and scaling train dataset

train_scaler=Normalizer().fit(train_df.drop('y',1))
norm_train_df = train_scaler.transform(train_df.drop('y',1))
norm_train_df.shape
```

Out[124]:

```
(4209, 364)
```

In [125]:

```
# Normalizing and scaling test dataset

test_scaler=Normalizer().fit(test_df)
norm_test_df = test_scaler.transform(test_df)
norm_test_df.shape
```

Out[125]:

```
(4209, 364)
```

In [126]:

```
pca =PCA()
pca.fit(norm_train_df)
```

Out[126]:

```
PCA()
```

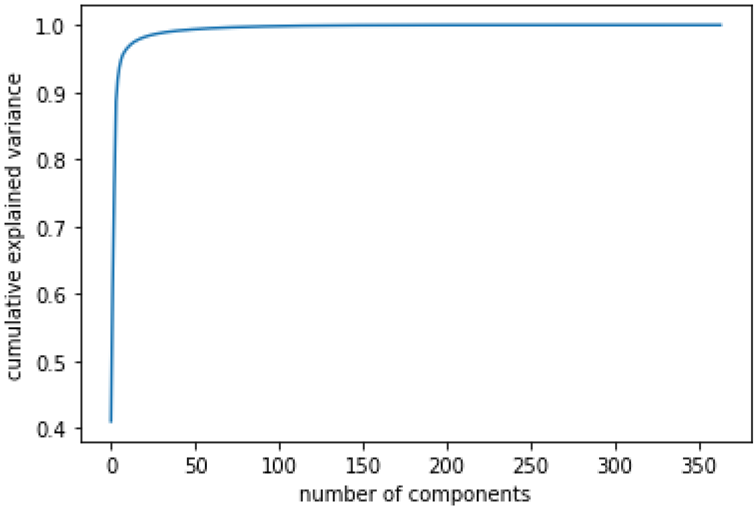
In [127]:

```
# Finding optimal number of components by depicting a graph

f =np.cumsum(pca.explained_variance_ratio_)
plt.plot(f)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
```

Out[127]:

```
Text(0, 0.5, 'cumulative explained variance')
```



From the above graph, the optimum variance is between [0.95 - 0.99]

In [128]:

```
# PCA with 97%

pca = PCA(n_components=0.97, whiten=True)
norm_features = pca.fit_transform(norm_train_df)
```

In [129]:

```
print(f'The midpoint method retains {norm_features.shape[1]} features')
```

```
The midpoint method retains 13 features
```

Predict your test_df values using XGBoost.

In [130]:

```
# Importing necessary libraries

import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
```

In [131]:

```
# Splitting into training and validations sets

X_train, X_val, y_train, y_val = train_test_split(train_df.iloc[:,1:], train_df['y'].va
lues, test_size=0.25, random_state=4321)
```

In [132]:

```
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

Out[132]:

```
((3156, 364), (1053, 364), (3156,), (1053,))
```

In [133]:

```
train_xgb_reg = xgb.XGBRegressor( objective = 'reg:squarederror', colsample_bytree = 0.
1,  learning_rate = 0.2, max_depth = 7, alpha = 10)
```

In [134]:

```
# fit and validate the training model

train_xgb_reg.fit(X_train,y_train)
train_valid = train_xgb_reg.predict(X_val)
```

In [135]:

```
training_rmse = np.sqrt(mean_squared_error(y_val, train_valid))
print(f'Training RMSE: {training_rmse}')
```

```
Training RMSE: 9.855368735242184
```

In [136]:

```
train_xgb_reg.score(X_train,y_train)
```

Out[136]:

```
0.7772092553569827
```

In [137]:

```
testing_preds = train_xgb_reg.predict(test_df)
testing_rmse = np.sqrt(mean_squared_error(train_df['y'], testing_preds))
print(f'Testing RMSE: {testing_rmse}')
```

```
Testing RMSE: 15.938119151947369
```

Output

We can see that the RMSE (15.93) for the testing is not good, this suggests that the model did not do well on the testing set.