Here's an outline of the steps you can take to complete the Mercedes-Benz Greener Manufacturing course-end project:

Load the dataset into a Pandas DataFrame

The dataset is available in CSV format and can be loaded using the read_csv() method of Pandas. Remove columns with zero variance

Use the var() method to calculate the variance for each column, and remove any columns with zero variance using the drop() method. Check for null and unique values in the train and test sets

Use the isnull() method to identify any null values in the DataFrame, and the nunique() method to identify any columns with a single unique value. If any null or single-value columns are found, you can remove them using the drop() method. Apply label encoding

Use the LabelEncoder() method from the scikit-learn library to encode categorical variables in the dataset. Perform dimensionality reduction

Use techniques such as PCA or t-SNE to reduce the number of dimensions in the dataset while retaining as much variance as possible. Train an XGBoost model

Use the XGBoost library to train a model on the preprocessed dataset. You can split the dataset into training and validation sets, and use techniques such as cross-validation and hyperparameter tuning to optimize the model. Make predictions on the test set

Use the trained model to make predictions on the test set, and save the predictions to a CSV file.

In [1]:

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
train=pd.read_csv('MERCDIZ TRAIN.csv')
```

In [3]:

```
train.head()
```

Out[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X3
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 [4]:

```
test=pd.read_csv('MERCDIZ TEST.csv')
```

In [5]:

```
test.head()
```

Out[5]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0

5 rows × 377 columns

In [6]:

```
print(f'the size of training data is {train.shape}')
print(f'the size of testing data is {test.shape}')
```

```
the size of training data is (4209, 378)
the size of testing data is (4209, 377)
```

In [7]:

```
# lets collect Y values in an array from Train data:
y_train=train['y'].values
y_train
```

Out[7]:

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

In [8]:

```
# Understand the data types
cols = [c for c in train.columns if 'X' in c]
print('Number of features:',format(len(cols)))
print('Feature types:')
train[cols].dtypes.value_counts()
```

```
Number of features: 376
Feature types:
```

Out[8]:

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

In [9]:

```
train.dtypes.unique()
```

Out[9]:

```
array([dtype('int64'), dtype('float64'), dtype('O')], dtype=object)
```

In [10]:

```
train.dtypes.count()
```

Out[10]:

```
378
```

In [11]:

```
train.select_dtypes(int).shape
```

Out[11]:

```
(4209, 369)
```

In [12]:

```
train.select_dtypes(object).shape
```

Out[12]:

```
(4209, 8)
```

In [13]:

```
train.select_dtypes(float).shape
```

Out[13]:

```
(4209, 1)
```

In [14]:

```
#checking catagorical columns in my DF :
cat_col=train.select_dtypes(include=['object']).columns
print(list(cat_col))
#train.select_dtypes(object).shape
```

```
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

In [15]:

```
#checking numeric columns in my DF :
numeric_col=train.select_dtypes(include=['int']).columns
numeric_col.value_counts()
#train.select_dtypes(int).shape
```

Out[15]:

```
ID      1
X258    1
X267    1
X266    1
X265    1
       ..
X133    1
X132    1
X131    1
X130    1
X385    1
Length: 369, dtype: int64
```

In [16]:

```
#now we will split data so that we can do prediction here we will not consider target s
# Splitting the data
usable_columns = list(set(train.columns) - set(['ID', 'y']))
y_train = train['y'].values
id_test = test['ID'].values
x_train = train[usable_columns]
x_test = test[usable_columns]
```

In [17]:

```
#look for missing values in my train and test data:
```

In [18]:

```
x_train.isna().sum()
```

Out[18]:

```
X240    0
X380    0
X20     0
X11     0
X66     0
       ..
X242    0
X229    0
X249    0
X165    0
X214    0
Length: 376, dtype: int64
```

In [19]:

```
x_test.isna().sum()
```

Out[19]:

```
X240    0
X380    0
X20     0
X11     0
X66     0
       ..
X242    0
X229    0
X249    0
X165    0
X214    0
Length: 376, dtype: int64
```

Label Encoding the categorical values

In [20]:

```
from sklearn.preprocessing import LabelEncoder
```

In [21]:

```
le=LabelEncoder()
```

In [22]:

```python
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()
```

Out[22]:

	X240	X380	X20	X11	X66	X236	X205	X224	X296	X34	...	X30	X253	X284	X3	X3(
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	97	
1	0	0	0	0	0	0	1	0	0	0	...	0	0	0	101	
2	0	0	0	0	0	0	1	1	0	0	...	0	0	0	99	
3	0	0	0	0	0	0	1	0	0	0	...	0	0	0	102	
4	0	0	0	0	0	0	1	0	0	0	...	0	0	0	102	

5 rows × 376 columns

In [23]:

```python
#make sure our data is changed in numeric type:
```

In [24]:

```python
train.shape
```

Out[24]:

```
(4209, 378)
```

In [30]:

```
train = train.apply(le.fit_transform)
```

In [31]:

```
train
```

Out[31]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380
0	0	2466	32	23	17	0	3	24	9	14	...	0	0	1	0	0	0
1	1	366	32	21	19	4	3	28	11	14	...	1	0	0	0	0	0
2	2	69	20	24	34	2	3	27	9	23	...	0	0	0	0	0	0
3	3	133	20	21	34	5	3	27	11	4	...	0	0	0	0	0	0
4	4	106	20	23	34	5	3	12	3	13	...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	4204	1657	8	20	16	2	3	0	3	16	...	1	0	0	0	0	0
4205	4205	1766	31	16	40	3	3	0	7	7	...	0	1	0	0	0	0
4206	4206	1801	8	23	38	0	3	0	6	4	...	0	0	1	0	0	0
4207	4207	280	9	19	25	5	3	0	11	20	...	0	0	0	0	0	0
4208	4208	1921	46	19	3	2	3	0	6	22	...	1	0	0	0	0	0

4209 rows × 378 columns

In [33]:

```
train.select_dtypes(int).shape
```

Out[33]:

```
(4209, 378)
```

Perform Dimensionality reduction

Dimensionality reduction is a common technique used in machine learning to reduce the number of features or variables in a dataset, while retaining as much of the original information as possible. This can help to reduce the computational cost of training a model, improve its generalization performance, and aid in data visualization. Here are two commonly used methods for dimensionality reduction: Principal Component Analysis (PCA) PCA is a linear transformation technique that can be used to identify the most important features or patterns in a dataset. It works by finding the directions of maximum variance in the data and projecting the data onto these directions. Here's an example of how to perform PCA using scikit-learn in Python:

In [35]:

```
from sklearn.decomposition import PCA
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)

'''PCA(n_components = n_comp,random_state = 420): This line initializes a new PCA objec
pca.fit_transform(x_train): This line fits the PCA model to the training data x_train a
pca.transform(x_test): This line transforms the test data x_test to the same 12-dimensi
and assigns the results to pca2_results_test.'''
```

Out[35]:

```
'PCA(n_components = n_comp,random_state = 420): This line initializes a new
PCA object with the number of components set to n_comp and the random seed s
et to 420.\npca.fit_transform(x_train): This line fits the PCA model to the
training data x_train and transforms it to a new 12-dimensional space, and a
ssigns the results to pca2_results_train.\npca.transform(x_test): This line
transforms the test data x_test to the same 12-dimensional space as the trai
ning data using the PCA model that was already fitted on the training data,
\nand assigns the results to pca2_results_test.'
```

In [38]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 12,random_state = 420)
pca_results_train = pca.fit_transform(x_train)
pca_results_test = pca.transform(x_test)
```

training using XGBoost

In [40]:

```
pip install xgboost

```

```
Collecting xgboost
  Downloading xgboost-1.7.4-py3-none-win_amd64.whl (89.1 MB)
     ---------------------------------------- 89.1/89.1 MB 1.3 MB/s eta 0:0
0:00
Requirement already satisfied: numpy in d:\mypy\lib\site-packages (from xgbo
ost) (1.21.5)
Requirement already satisfied: scipy in d:\mypy\lib\site-packages (from xgbo
ost) (1.9.1)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.4
Note: you may need to restart the kernel to use updated packages.
```

In [42]:

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

In [43]:

```
x_train,x_val,y_train,y_val = train_test_split(pca_results_train, y_train, test_size=0.

```

In [44]:

```
d_train = xgb.DMatrix(x_train,label = y_train)
d_val = xgb.DMatrix(x_val,label = y_val)

# dtest = xgb.DMatrix(x_test)

d_test = xgb.DMatrix(pca2_results_test)
```

In [45]:

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

```
[00:28:36] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscal
ing-group-i-0fc7796c793e6356f-1/xgboost/xgboost-ci-windows/src/objective/reg
ression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederro
r.
[0]     train-rmse:99.14834     train-r2:-58.35295      valid-rmse:98.26297
valid-r2:-67.63754
[10]    train-rmse:81.27653     train-r2:-38.88428      valid-rmse:80.36433
valid-r2:-44.91014
[20]    train-rmse:66.71610     train-r2:-25.87403      valid-rmse:65.77334
valid-r2:-29.75260
[30]    train-rmse:54.86913     train-r2:-17.17722      valid-rmse:53.89147
valid-r2:-19.64534
[40]    train-rmse:45.24710     train-r2:-11.36098      valid-rmse:44.22334
valid-r2:-12.90225
[50]    train-rmse:37.44856     train-r2:-7.46723       valid-rmse:36.37638
valid-r2:-8.40634
[60]    train-rmse:31.14586     train-r2:-4.85696       valid-rmse:30.02276
valid-r2:-5.40742
[70]    train-rmse:26.08420     train-r2:-3.10796       valid-rmse:24.91520
valid-r2:-3.41276
[80]    train-rmse:22.04315     train-r2:-1.93372       valid-rmse:20.83302
valid-r2:-2.08522
[90]    train-rmse:18.84672     train-r2:-1.14458       valid-rmse:17.59925
valid-r2:-1.20176
[100]   train-rmse:16.33388     train-r2:-0.61083       valid-rmse:15.08776
valid-r2:-0.61820
[110]   train-rmse:14.40312     train-r2:-0.25252       valid-rmse:13.15610
valid-r2:-0.23037
[120]   train-rmse:12.93267     train-r2:-0.00983       valid-rmse:11.69731
valid-r2:0.02736
[130]   train-rmse:11.81057     train-r2:0.15780        valid-rmse:10.61908
valid-r2:0.19840
[140]   train-rmse:10.97712     train-r2:0.27248        valid-rmse:9.85265
valid-r2:0.30994
[150]   train-rmse:10.36657     train-r2:0.35115        valid-rmse:9.32499
valid-r2:0.38187
[160]   train-rmse:9.91740      train-r2:0.40616        valid-rmse:8.96438
valid-r2:0.42875
[170]   train-rmse:9.58573      train-r2:0.44522        valid-rmse:8.72085
valid-r2:0.45937
[180]   train-rmse:9.33421      train-r2:0.47395        valid-rmse:8.55878
valid-r2:0.47928
[190]   train-rmse:9.14666      train-r2:0.49488        valid-rmse:8.45476
valid-r2:0.49186
[200]   train-rmse:9.00443      train-r2:0.51046        valid-rmse:8.39202
valid-r2:0.49937
```

```
[210]	train-rmse:8.90139	train-r2:0.52160	valid-rmse:8.35400
valid-r2:0.50390
[220]	train-rmse:8.82413	train-r2:0.52987	valid-rmse:8.32627
valid-r2:0.50719
[230]	train-rmse:8.77090	train-r2:0.53553	valid-rmse:8.31418
valid-r2:0.50862
[240]	train-rmse:8.72336	train-r2:0.54055	valid-rmse:8.30887
valid-r2:0.50924
[250]	train-rmse:8.68129	train-r2:0.54497	valid-rmse:8.30371
valid-r2:0.50985
[260]	train-rmse:8.64782	train-r2:0.54847	valid-rmse:8.29920
valid-r2:0.51038
[270]	train-rmse:8.61485	train-r2:0.55191	valid-rmse:8.29987
valid-r2:0.51031
[280]	train-rmse:8.59091	train-r2:0.55439	valid-rmse:8.29829
valid-r2:0.51049
[290]	train-rmse:8.56735	train-r2:0.55684	valid-rmse:8.29783
valid-r2:0.51055
[300]	train-rmse:8.54437	train-r2:0.55921	valid-rmse:8.29630
valid-r2:0.51073
[310]	train-rmse:8.51856	train-r2:0.56187	valid-rmse:8.29570
valid-r2:0.51080
[320]	train-rmse:8.49720	train-r2:0.56406	valid-rmse:8.29657
valid-r2:0.51070
[330]	train-rmse:8.46977	train-r2:0.56687	valid-rmse:8.29420
valid-r2:0.51097
[340]	train-rmse:8.44582	train-r2:0.56932	valid-rmse:8.28979
valid-r2:0.51150
[350]	train-rmse:8.41366	train-r2:0.57259	valid-rmse:8.28888
valid-r2:0.51160
[360]	train-rmse:8.38763	train-r2:0.57523	valid-rmse:8.28962
valid-r2:0.51152
[370]	train-rmse:8.36438	train-r2:0.57759	valid-rmse:8.28879
valid-r2:0.51161
[380]	train-rmse:8.33907	train-r2:0.58014	valid-rmse:8.28530
valid-r2:0.51202
[390]	train-rmse:8.31292	train-r2:0.58277	valid-rmse:8.28232
valid-r2:0.51238
[400]	train-rmse:8.29008	train-r2:0.58506	valid-rmse:8.28049
valid-r2:0.51259
[410]	train-rmse:8.25961	train-r2:0.58810	valid-rmse:8.27819
valid-r2:0.51286
[420]	train-rmse:8.23707	train-r2:0.59035	valid-rmse:8.27717
valid-r2:0.51298
[430]	train-rmse:8.21144	train-r2:0.59289	valid-rmse:8.27726
valid-r2:0.51297
[440]	train-rmse:8.18882	train-r2:0.59513	valid-rmse:8.27403
valid-r2:0.51335
[450]	train-rmse:8.16525	train-r2:0.59746	valid-rmse:8.27367
valid-r2:0.51339
[460]	train-rmse:8.13739	train-r2:0.60020	valid-rmse:8.27194
valid-r2:0.51360
[470]	train-rmse:8.11066	train-r2:0.60282	valid-rmse:8.27239
valid-r2:0.51354
[480]	train-rmse:8.08653	train-r2:0.60518	valid-rmse:8.27091
valid-r2:0.51372
[490]	train-rmse:8.06536	train-r2:0.60725	valid-rmse:8.26943
valid-r2:0.51389
[500]	train-rmse:8.04295	train-r2:0.60943	valid-rmse:8.27073
valid-r2:0.51374
[510]	train-rmse:8.02304	train-r2:0.61136	valid-rmse:8.27099
```

```
valid-r2:0.51371
[520]	train-rmse:8.00139	train-r2:0.61345	valid-rmse:8.26800
valid-r2:0.51406
[530]	train-rmse:7.98241	train-r2:0.61528	valid-rmse:8.27042
valid-r2:0.51377
[540]	train-rmse:7.96465	train-r2:0.61699	valid-rmse:8.27174
valid-r2:0.51362
[550]	train-rmse:7.93525	train-r2:0.61982	valid-rmse:8.27126
valid-r2:0.51368
[560]	train-rmse:7.91605	train-r2:0.62165	valid-rmse:8.27148
valid-r2:0.51365
[568]	train-rmse:7.90075	train-r2:0.62311	valid-rmse:8.27261
valid-r2:0.51352
```

In [46]:

```
#Prediction using XGBoost
```

In [47]:

```
p_test = clf.predict(d_test)

```

In [48]:

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

Out[48]:

	ID	y
0	1	82.930893
1	2	96.948074
2	3	82.922882
3	4	76.936714
4	5	112.465836

In [ ]:

```

```