Mercedes-Benz Greener Manufacturing

Reduce the time a Mercedes-Benz spends on the test bench.

Problem Statement Scenario: Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include the passenger safety cell with a crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium carmakers. Mercedes-Benz is the leader in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams.

To ensure the safety and reliability of every unique car configuration before they hit the road, the company's engineers have developed a robust testing system. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Mercedes-Benz's production lines. However, optimizing the speed of their testing system for many possible feature combinations is complex and time-consuming without a powerful algorithmic approach.

You are required to reduce the time that cars spend on the test bench. Others will work with a dataset representing different permutations of features in a Mercedes-Benz car to predict the time it takes to pass testing. Optimal algorithms will contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Mercedes-Benz's standards.

Following actions should be performed:

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

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

3.Apply label encoder.

4.Perform dimensionality reduction.

5.Predict your test_df values using XGBoost.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
```

Import Train data and understand the data

In [2]:
```
#import Train data
train_file=r'C:\Users\sinun\OneDrive\Documents\Simplilearn\machime learning\project\mercedez\dataset\train\train.csv'
train_df =pd.read_csv(train_file)
```

In [3]:
```
train_df.head()
```

Out[3]:

	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

5 rows × 378 columns

In [4]:
```
train_df.shape
```

Out[4]:
```
(4209, 378)
```

In [5]:
```
train_df.describe
```

Out[5]:
```
<bound method NDFrame.describe 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 [6]:
```
train_df.columns
```

Out[6]:
```
Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       ...
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=378)
```

In [7]:
```
train_df.dtypes
```

Out[7]:
```
ID        int64
y       float64
X0       object
X1       object
X2       object
         ...
X380      int64
X382      int64
X383      int64
X384      int64
X385      int64
Length: 378, dtype: object
```

In [8]:
```
train_df.dtypes.value_counts()
```

Out[8]:
```
int64      369
object       8
float64      1
dtype: int64
```

Import TEST data and understand the data

In [9]:
```
#import Test DAta
test_file=r'C:\Users\sinun\OneDrive\Documents\Simplilearn\machime learning\project\mercedez\dataset\test\test.csv'
test_df =pd.read_csv(test_file)
```

In [10]:
```
test_df.shape
```

Out[10]:
```
(4209, 377)
```

In [11]:
```
test_df.head()
```

Out[11]:

	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 [12]:
```
test_df.dtypes.value_counts()
```

Out[12]:
```
int64     369
object      8
dtype: int64
```

create Y train data

In [13]:
```
Y_train=train_df["y"]
Y_train
```

Out[13]:
```
0       130.81
1        88.53
2        76.26
3        80.62
4        78.02
         ...
4204    107.39
4205    108.77
4206    109.22
4207     87.48
4208    110.85
Name: y, Length: 4209, dtype: float64
```

Checking for Missing Values in train data

In [14]:
```
train_df.isnull().sum().sum()
```

Out[14]:
```
0
```

Checking for Missing Values in test data

In [15]:
```
test_df.isnull().sum().sum()
```

Out[15]:
```
0
```

* No missing values in the train and test dataset

REMOVE the coloumns with Variance = 0 from train data

In [16]:
```
train_df_var = pd.DataFrame(train_df.var(axis=0),columns=['Variance'])
train_df_var.head()
```

Out[16]:

	Variance

	Variance
ID	5.941936e+06
y	1.607667e+02
X10	1.313092e-02
X11	0.000000e+00
X12	6.945713e-02

In [17]:
```
#Dropping the columns with variance is == 0
train_df_var0 = train_df.drop(columns=train_df_var[train_df_var.Variance==0].index)
train_df_var0.head()
```

Out[17]:

	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

5 rows × 366 columns

In [18]:
```
train_df_var[train_df_var.Variance==0] # 12 columns with variance ==0 which are dropped
```

Out[18]:

	Variance
X11	0.0
X93	0.0
X107	0.0
X233	0.0

	Variance
X235	0.0
X268	0.0
X289	0.0
X290	0.0
X293	0.0
X297	0.0
X330	0.0
X347	0.0

Remove the same coloumns of TEST DATA which have 'variance=0 in TRAIN DATA'

In [19]:
```
# In test dataset, remove the same features of train dataset having 0 variance.
test_df_var0= test_df.drop(columns=['X11', 'X93', 'X107','X233', 'X235', 'X268', 'X289', 'X290', 'X293','X297','X330','X347'])
```

Seperate NUMERICAL & CATEGORICAL COLOUMNS OF TRAIN DATA

In [20]:
```
# CATEGORICAL coloumns in Train data
train_df_cat = train_df_var0.select_dtypes(exclude=["float64","int64"])
train_df_cat.head()
```

Out[20]:

	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

In [21]:
```
# Numerical coloumns in Train Data
train_df_num = train_df_var0.select_dtypes(include=["float64","int64"])
train_df_num.head()
```

Out[21]:

	ID	y	X10	X12	X13	X14	X15	X16	X17	X18	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	0	0	1	0	0	0	0	1	...	0	0	1	0	0	0	0	0	0	0
1	6	88.53	0	0	0	0	0	0	0	1	...	1	0	0	0	0	0	0	0	0	0
2	7	76.26	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	13	78.02	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 358 columns

Seperate NUMERICAL & CATEGORICAL COLOUMNS OF TEST DATA

In [22]:
```
# CATEGORICAL coloumns in Test data
test_df_cat = test_df_var0.select_dtypes(exclude=["float64","int64"])
test_df_cat.head()
```

Out[22]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	az	v	n	f	d	t	a	w
1	t	b	ai	a	d	b	g	y
2	az	v	as	f	d	a	j	j
3	az	l	n	f	d	z	l	n
4	w	s	as	c	d	y	i	m

In [23]:
```
# Numerical coloumns in Test Data
test_df_num = test_df_var0.select_dtypes(include=["float64","int64"])
test_df_num.head()
```

Out[23]:

	ID	X10	X12	X13	X14	X15	X16	X17	X18	X19	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
1	2	0	0	0	0	0	0	0	0	1	...	0	0	1	0	0	0	0	0	0	0
2	3	0	0	0	1	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
4	5	0	0	0	1	0	0	0	0	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 357 columns

Check for unique value in categorical columns in TRAIN DATA

In [24]:
```
train_df_cat ['X0'].unique()
```

Out[24]:
```
array(['k', 'az', 't', 'al', 'o', 'w', 'j', 'h', 's', 'n', 'ay', 'f', 'x',
       'y', 'aj', 'ak', 'am', 'z', 'q', 'at', 'ap', 'v', 'af', 'a', 'e',
       'ai', 'd', 'aq', 'c', 'aa', 'ba', 'as', 'i', 'r', 'b', 'ax', 'bc',
       'u', 'ad', 'au', 'm', 'l', 'aw', 'ao', 'ac', 'g', 'ab'],
      dtype=object)
```

In [25]:
```
j=len(train_df_cat ['X0'].unique())
j
```

Out[25]:
```
47
```

In [26]:
```
train_df_cat ['X0'].unique()
```

Out[26]:
```
array(['k', 'az', 't', 'al', 'o', 'w', 'j', 'h', 's', 'n', 'ay', 'f', 'x',
       'y', 'aj', 'ak', 'am', 'z', 'q', 'at', 'ap', 'v', 'af', 'a', 'e',
       'ai', 'd', 'aq', 'c', 'aa', 'ba', 'as', 'i', 'r', 'b', 'ax', 'bc',
       'u', 'ad', 'au', 'm', 'l', 'aw', 'ao', 'ac', 'g', 'ab'],
      dtype=object)
```

In [27]:
```
train_df_cat ['X1'].unique()
```

Out[27]:
```
array(['v', 't', 'w', 'b', 'r', 'l', 's', 'aa', 'c', 'a', 'e', 'h', 'z',
       'j', 'o', 'u', 'p', 'n', 'i', 'y', 'd', 'f', 'm', 'k', 'g', 'q',
       'ab'], dtype=object)
```

In [28]:
```
train_df_cat ['X2'].unique()
```

Out[28]:
```
array(['at', 'av', 'n', 'e', 'as', 'aq', 'r', 'ai', 'ak', 'm', 'a', 'k',
       'ae', 's', 'f', 'd', 'ag', 'ay', 'ac', 'ap', 'g', 'i', 'aw', 'y',
       'b', 'ao', 'al', 'h', 'x', 'au', 't', 'an', 'z', 'ah', 'p', 'am',
       'j', 'q', 'af', 'l', 'aa', 'c', 'o', 'ar'], dtype=object)
```

In [29]:
```
train_df_cat ['X3'].unique()
```

Out[29]:
```
array(['a', 'e', 'c', 'f', 'd', 'b', 'g'], dtype=object)
```

In [30]:
```
train_df_cat ['X4'].unique()
```

Out[30]:
```
array(['d', 'b', 'c', 'a'], dtype=object)
```

In [31]:
```
train_df_cat ['X5'].unique()
```

Out[31]:
```
array(['u', 'y', 'x', 'h', 'g', 'f', 'j', 'i', 'd', 'c', 'af', 'ag', 'ab',
       'ac', 'ad', 'ae', 'ah', 'l', 'k', 'n', 'm', 'p', 'q', 's', 'r',
       'v', 'w', 'o', 'aa'], dtype=object)
```

In [32]:
```
train_df_cat ['X6'].unique()
```

Out[32]:
```
array(['j', 'l', 'd', 'h', 'i', 'a', 'g', 'c', 'k', 'e', 'f', 'b'],
      dtype=object)
```

In [33]:
```
train_df_cat ['X8'].unique()
```

Out[33]:
```
array(['o', 'x', 'e', 'n', 's', 'a', 'h', 'p', 'm', 'k', 'd', 'i', 'v',
       'j', 'b', 'q', 'w', 'g', 'y', 'l', 'f', 'u', 'r', 't', 'c'],
      dtype=object)
```

Check for unique value in catagorical columns in TEST DATA

In [34]:
```
test_df_cat ['X0'].unique()
```

Out[34]:
```
array(['az', 't', 'w', 'y', 'x', 'f', 'ap', 'o', 'ay', 'al', 'h', 'z',
       'aj', 'd', 'v', 'ak', 'ba', 'n', 'j', 's', 'af', 'ax', 'at', 'aq',
       'av', 'm', 'k', 'a', 'e', 'ai', 'i', 'ag', 'b', 'am', 'aw', 'as',
       'r', 'ao', 'u', 'l', 'c', 'ad', 'au', 'bc', 'g', 'an', 'ae', 'p',
       'bb'], dtype=object)
```

In [35]:
```
test_df_cat ['X1'].unique()
```

Out[35]:
```
array(['v', 'b', 'l', 's', 'aa', 'r', 'a', 'i', 'p', 'c', 'o', 'm', 'z',
       'e', 'h', 'w', 'g', 'k', 'y', 't', 'u', 'd', 'j', 'q', 'n', 'f',
       'ab'], dtype=object)
```

In [36]:
```
test_df_cat ['X2'].unique()
```

Out[36]:
```
array(['n', 'ai', 'as', 'ae', 's', 'b', 'e', 'ak', 'm', 'a', 'aq', 'ag',
       'r', 'k', 'aj', 'ay', 'ao', 'an', 'ac', 'af', 'ax', 'h', 'i', 'f',
       'ap', 'p', 'au', 't', 'z', 'y', 'aw', 'd', 'at', 'g', 'am', 'j',
       'x', 'ab', 'w', 'q', 'ah', 'ad', 'al', 'av', 'u'], dtype=object)
```

In [37]:
```
test_df_cat ['X3'].unique()
```

Out[37]:
```
array(['f', 'a', 'c', 'e', 'd', 'g', 'b'], dtype=object)
```

In [38]:
```
test_df_cat ['X4'].unique()
```

Out[38]:
```
array(['d', 'b', 'a', 'c'], dtype=object)
```

In [39]:
```
test_df_cat ['X5'].unique()
```

Out[39]:
```
array(['t', 'b', 'a', 'z', 'y', 'x', 'h', 'g', 'f', 'j', 'i', 'd', 'c',
       'af', 'ag', 'ab', 'ac', 'ad', 'ae', 'ah', 'l', 'k', 'n', 'm', 'p',
       'q', 's', 'r', 'v', 'w', 'o', 'aa'], dtype=object)
```

In [40]:
```
test_df_cat ['X6'].unique()
```

Out[40]:
```
array(['a', 'g', 'j', 'l', 'i', 'd', 'f', 'h', 'c', 'k', 'e', 'b'],
      dtype=object)
```

In [41]:
```
test_df_cat ['X8'].unique()
```

Out[41]:
```
array(['w', 'y', 'j', 'n', 'm', 's', 'a', 'v', 'r', 'o', 't', 'h', 'c',
       'k', 'p', 'u', 'd', 'g', 'b', 'q', 'e', 'l', 'f', 'i', 'x'],
      dtype=object)
```

Check Whether the train and test have same unique values

In [42]:
```
set(test_df_cat.X0.unique()).issubset(set(train_df_cat.X0.unique()))
```

Out[42]: False

In [43]:
```
length_unique_train_X0=len(train_df_cat ['X0'].unique())
length_unique_train_X0
```

Out[43]: 47

In [44]:
```
length_unique_test_X0=len(test_df_cat ['X0'].unique())
length_unique_test_X0
```

Out[44]: 49

** Categorical column X0 of Train and Test Data have some differnt unique values which is not present in the other

In [45]:
```
set(test_df_cat.X1.unique()).issubset(set(train_df_cat.X1.unique()))
```

Out[45]: True

In [46]:
```
set(test_df_cat.X2.unique()).issubset(set(train_df_cat.X2.unique()))
```

Out[46]: False

In [47]:
```
set(test_df_cat.X3.unique()).issubset(set(train_df_cat.X3.unique()))
```

Out[47]: True

In [48]:
```
set(test_df_cat.X4.unique()).issubset(set(train_df_cat.X4.unique()))
```

Out[48]: True

In [49]:
```
set(test_df_cat.X5.unique()).issubset(set(train_df_cat.X5.unique()))
```

Out[49]: False

In [50]:
```
set(test_df_cat.X6.unique()).issubset(set(train_df_cat.X6.unique()))
```

Out[50]: True

In [51]:
```
set(test_df_cat.X8.unique()).issubset(set(train_df_cat.X8.unique()))
```

Out[51]: True

** Out of the 8 categorical columns of Test data, X0,X2 and X5 have some unique values which are not present in that of Train Data

APPLY LABEL ENCODER ON CATEGORICAL VALUES OF TRAIN & TEST DATA

In [52]:
```
#Lable encoder for catagorical data
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
usable_columns = list((train_df_cat.columns))
usable_columns
```

Out[52]: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']

In [53]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [54]:
```
for column in usable_columns:
    le = LabelEncoder()
    le.fit(train_df_cat[column])# Fitting on categorical data of Train Data
    #Test data have some values which are not seen in Train data, So label them as 'unknown'
    test_df_cat[column] = test_df_cat[column ].map(lambda s: '<unknown>' if s not in le.classes_ else s)
    le.classes_ = np.append(le.classes_, '<unknown>')
    train_df_cat[column] = le.transform(train_df_cat[column])# Apply label encoding on Train data
    test_df_cat[column] = le.transform(test_df_cat[column])# Apply label encoding on Test data
```

In [55]:
```
train_df_cat.head()
```

Out[55]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	32	23	17	0	3	24	9	14
1	32	21	19	4	3	28	11	14
2	20	24	34	2	3	27	9	23
3	20	21	34	5	3	27	11	4
4	20	23	34	5	3	12	3	13

In [56]:
```
test_df_cat.head()
```

Out[56]:

	X0	X1	X2	X3	X4	X5	X6	X8
0	20	23	34	5	3	29	0	22
1	40	3	7	0	3	29	6	24
2	20	23	16	5	3	29	9	9
3	20	13	34	5	3	29	11	13
4	43	20	16	2	3	28	8	12

Apply Dimensionality Reduction using PCA

In [57]:
```
# Applying PCA in Numerical Columns of Train and Test Data
X_train_df_num=train_df_num.drop(["ID","y"], axis=1) # Dropping ID and y columns in Train data
X_test_df_num=test_df_num.drop(["ID"], axis=1)# Dropping ID column in Train data
```

In [58]:
```
X_train_df_num.shape
```

Out[58]: (4209, 356)

In [59]:
```
X_test_df_num.shape
```

Out[59]: (4209, 356)

In [60]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=.9)
```

In [61]:
```
pca.fit(X_train_df_num)
```

Out[61]: PCA(n_components=0.9)

In [62]:
```
pca.explained_variance_ratio_.shape
```

Out[62]: (47,)

In [63]:
```
X_train_num_transformed = pca.transform(X_train_df_num)
X_test_num_transformed = pca.transform(X_test_df_num)
```

In [64]:
```
X_train_num_transformed.shape, X_test_num_transformed.shape
```

Out[64]: ((4209, 47), (4209, 47))

*** Out of the 356 numerical coloumns we are considering only 47 columns by applying PCA**

In [65]:

```
X_train_num_transformed
```

Out[65]:
```
array([[ 0.74758898,  2.21751705,  1.08596716, ...,  0.60629391,
        -0.31115318,  1.21939435],
       [-0.21453499,  1.12918616, -0.79412482, ..., -0.17348811,
        -0.02058919,  0.22223161],
       [-0.88975207,  2.96636978,  0.36727639, ...,  0.55830278,
        -0.43364057,  0.40360302],
       ...,
       [-1.08450421,  1.06354261,  1.36791009, ...,  0.20211471,
        -0.07649919,  0.03771631],
       [ 0.45278287, -0.60281218, -2.96835473, ..., -0.74720796,
        -0.39446344,  0.05521501],
       [ 0.81793706,  0.24270176, -1.33868599, ..., -0.13832731,
        -0.00602778,  0.48716583]])
```

In [66]:
```
X_train_num_transformed_df=pd.DataFrame(X_train_num_transformed)
```

In [67]:
```
# Train data is concatenated by label encoded categorical Data and PCA reduced Numerical colomns
X_train = pd.concat([train_df_cat,X_train_num_transformed_df], axis=1)
X_train
```

Out[67]:

	X0	X1	X2	X3	X4	X5	X6	X8	0	1	...	37	38	39	40	41	42	43	44	
0	32	23	17	0	3	24	9	14	0.747589	2.217517	...	-0.422005	0.134376	-0.297600	0.295136	-0.683612	0.260144	-0.551908	0.606294	-
1	32	21	19	4	3	28	11	14	-0.214535	1.129186	...	-0.234432	0.233675	0.147839	0.099123	0.142777	0.127248	-0.039948	-0.173488	-
2	20	24	34	2	3	27	9	23	-0.889752	2.966370	...	0.069117	-0.038065	0.215614	0.037337	0.424439	-0.504067	-0.021656	0.558303	-
3	20	21	34	5	3	27	11	4	-0.508901	2.463393	...	-0.079615	0.182731	-0.097329	0.342282	-0.590581	-0.476471	0.063032	-0.553598	
4	20	23	34	5	3	12	3	13	-0.486936	2.256592	...	0.063160	0.062220	0.203850	0.104985	-0.491399	-0.345375	-0.182477	-0.064865	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	8	20	16	2	3	0	3	16	-2.221273	0.334446	...	-0.119792	-0.426938	0.360326	-0.771698	0.050686	-0.504319	-0.041802	-0.507679	-
4205	31	16	40	3	3	0	7	7	0.944791	0.265777	...	1.089183	-0.243256	-0.075054	-0.344975	0.594630	-0.438612	0.987487	0.512236	-
4206	8	23	38	0	3	0	6	4	-1.084504	1.063543	...	0.119667	0.039851	0.101657	0.011870	-0.098111	-0.071329	0.042960	0.202115	-
4207	9	19	25	5	3	0	11	20	0.452783	-0.602812	...	0.462973	-0.009476	-0.197084	0.148773	0.635525	0.225981	-0.364973	-0.747208	-

	X0	X1	X2	X3	X4	X5	X6	X8	0	1	...	37	38	39	40	41	42	43	44
4208	46	19	3	2	3	0	6	22	0.817937	0.242702	...	0.404155	0.634337	0.141889	-0.261971	0.289901	0.146002	-0.181430	-0.138327

4209 rows × 55 columns

In [68]:
```
X_test_num_transformed_df=pd.DataFrame(X_test_num_transformed)
```

In [69]:
```
## Test data is concatenated by label encoded categorical Columns and PCA reduced Numerical columns
X_test = pd.concat([test_df_cat,X_test_num_transformed_df], axis=1)
X_test
```

Out[69]:

	X0	X1	X2	X3	X4	X5	X6	X8	0	1	...	37	38	39	40	41	42	43	44
0	20	23	34	5	3	29	0	22	-0.395807	2.405792	...	0.052551	0.041514	-0.106927	-0.055423	-0.276480	-0.212131	0.110340	-0.022521
1	40	3	7	0	3	29	6	24	3.726913	0.554368	...	0.186851	-0.080701	0.405209	0.095309	-0.598350	-0.040674	-0.345509	0.076059
2	20	23	16	5	3	29	9	9	-1.189752	0.926565	...	-0.041110	0.228033	-0.177122	0.009419	0.399592	0.260493	-0.088188	0.193560
3	20	13	34	5	3	29	11	13	-0.393356	2.338412	...	0.068993	0.054043	-0.099210	-0.007818	-0.376715	-0.126426	0.089467	-0.111999
4	43	20	16	2	3	28	8	12	-2.858966	-0.462167	...	-0.115218	0.226761	0.047958	0.212427	0.248885	-0.209748	-0.072348	0.129055
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	7	9	16	5	3	0	9	4	-2.852526	-0.770581	...	-0.037057	-0.036438	-0.055740	-0.040530	0.066450	0.043214	-0.015203	0.144410
4205	40	1	7	3	3	0	9	24	2.080839	1.040710	...	-0.418662	0.233465	-0.189211	-0.023116	0.049528	0.072256	-0.183880	0.064283
4206	45	23	16	5	3	0	3	22	-1.225581	-2.038197	...	0.375754	-0.174411	0.252313	-0.225309	0.122990	-0.400723	0.134293	-0.497919
4207	8	23	16	0	3	0	2	16	-2.241787	-0.370710	...	-0.264900	0.179415	0.098985	0.276181	-0.071203	-0.008795	-0.145908	0.352675
4208	40	1	7	2	3	0	6	17	1.806556	0.869159	...	-0.195440	-0.104244	0.159011	-0.136257	0.169369	-0.058328	-0.201269	0.154034

4209 rows × 55 columns

Apply XGBoost for finding the Y values in test Data

In [70]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\sinun\anaconda3\lib\site-packages (1.5.1)
Requirement already satisfied: numpy in c:\users\sinun\anaconda3\lib\site-packages (from xgboost) (1.20.1)
Requirement already satisfied: scipy in c:\users\sinun\anaconda3\lib\site-packages (from xgboost) (1.6.2)
Note: you may need to restart the kernel to use updated packages.
```

In [71]:
```
from xgboost import XGBClassifier, XGBRFClassifier
```

In [72]:
```
my_xgb_clf = XGBRFClassifier(booster = 'gbtree')
```

In [73]:
```
my_xgb_clf.fit(X_train,Y_train)
```

```
[09:48:30] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, t
he default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval
_metric if you'd like to restore the old behavior.
```

Out[73]:
```
XGBRFClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                colsample_bytree=1, enable_categorical=False, gamma=0,
                gpu_id=-1, importance_type=None, interaction_constraints='',
                max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
                monotone_constraints='()', n_estimators=100, n_jobs=8,
                num_parallel_tree=100, objective='multi:softprob',
                predictor='auto', random_state=0, reg_alpha=0,
                scale_pos_weight=None, tree_method='exact',
                validate_parameters=1, verbosity=None)
```

Predict your test_df values using XGBoost.

In [74]:
```
y_xgb_preds = my_xgb_clf.predict(X_test)
```

In [75]:
```
y_xgb_preds
```

Out[75]:
```
array([89.06, 91.88, 89.19, ..., 90.38, 90.41, 91.88])
```

In [76]:
```
y_predicted_test=pd.DataFrame(y_xgb_preds)
```

In [77]:
```
y_predicted_test
```

Out[77]:

	0
0	89.06
1	91.88
2	89.19
3	89.06
4	107.90
...	...
4204	97.96
4205	91.88
4206	90.38
4207	90.41
4208	91.88

4209 rows × 1 columns