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 seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
mb_train=pd.read_csv('train.csv')
mb_test=pd.read_csv('test.csv')
```

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

Out[3]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	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
3	9	80.62	az	t	n	f	d	x	l	e	...	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

5 rows × 378 columns

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

Out[4]: (4209, 378)

In [5]:
```
mb_test.head()
```

Out[5]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X:
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]:
```
mb_test.shape
```

Out[6]: (4209, 377)

In [7]:
```
mb_train.drop('ID',axis=1,inplace=True)
```

In [8]:
```
mb_train.head(2)
```

Out[8]:

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382
0	130.81	k	v	at	a	d	u	j	o	0	...	0	0	1	0	0	0	(
1	88.53	k	t	av	e	d	y	l	o	0	...	1	0	0	0	0	0	(

2 rows × 377 columns

In [9]:
```
mb_test.drop('ID',axis=1,inplace=True)
mb_test.head()
```

Out[9]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X382
0	az	v	n	f	d	t	a	w	0	0	...	0	0	0	1	0	0	0
1	t	b	ai	a	d	b	g	y	0	0	...	0	0	1	0	0	0	0
2	az	v	as	f	d	a	j	j	0	0	...	0	0	0	1	0	0	0
3	az	l	n	f	d	z	l	n	0	0	...	0	0	0	1	0	0	0
4	w	s	as	c	d	y	i	m	0	0	...	1	0	0	0	0	0	0

5 rows × 376 columns

We can clearly see the sparsity in training and testing data.It means that there are more binary features as compared to categorical features.

In [10]:
```
dtypes_df=mb_train.dtypes.reset_index()
dtypes_df.columns=['feature type','dtypes']
dtypes_df.groupby('dtypes').agg('count').reset_index()
```

Out[10]:

	dtypes	feature type
0	int64	368
1	float64	1
2	object	8

In [11]:
```
dtypes_test=mb_test.dtypes.reset_index()
dtypes_test.columns=['feature type','dtypes']
dtypes_test.groupby('dtypes').agg('count').reset_index()
```

Out[11]:

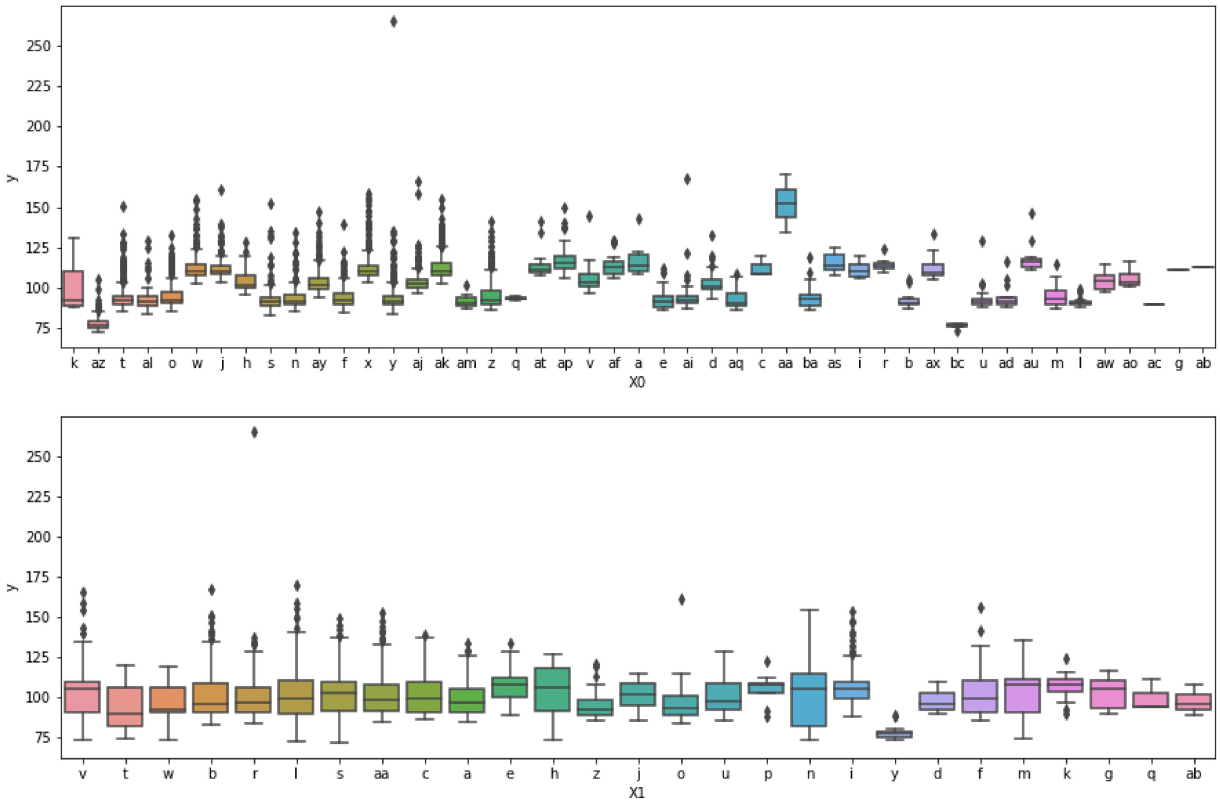
	dtypes	feature type
0	int64	368
1	object	8

After analyzing the data type we can say,there are 368 binary features, 8 feature which have data type object which is *categorical feature* and one remaining feature is our target variable which is **y**.

Analyzing categorical features-

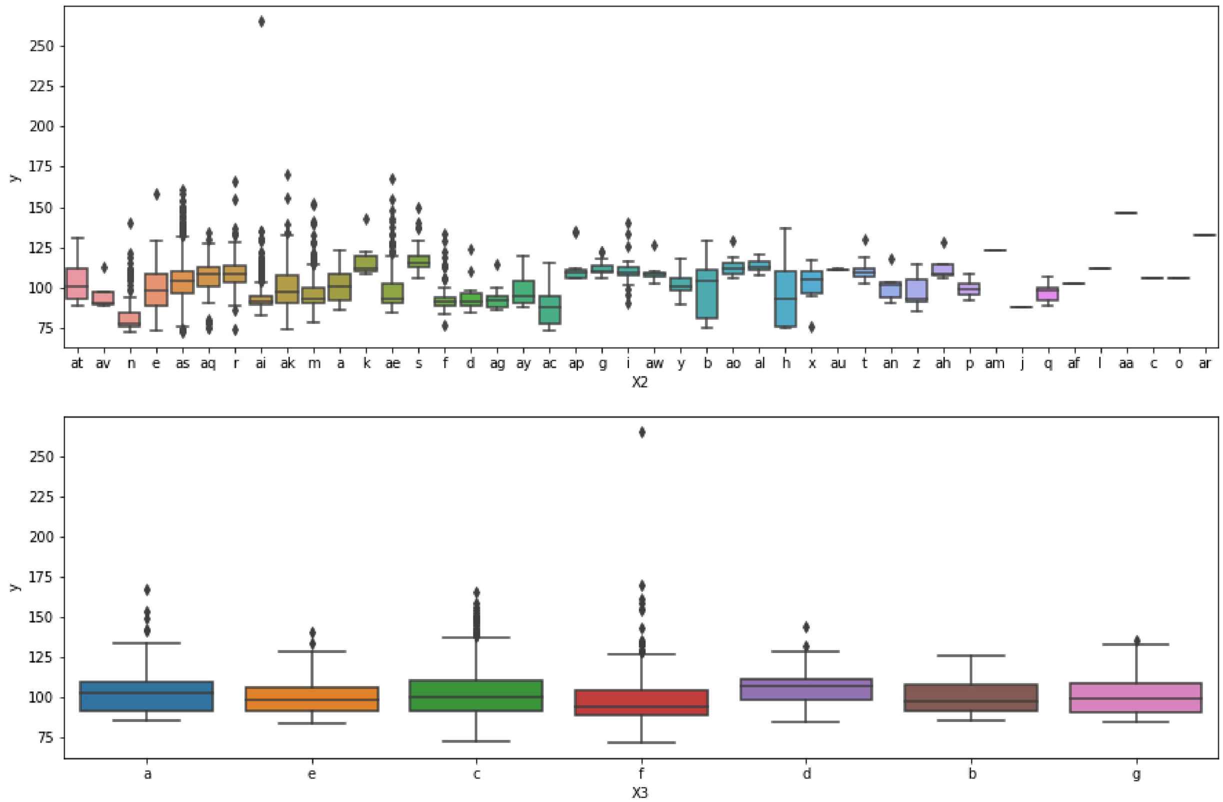
In [12]:
```
fig,ax=plt.subplots(2,1,figsize=(15,10))
sns.boxplot(x=mb_train['X0'],y=mb_train['y'],ax=ax[0])
sns.boxplot(x=mb_train['X1'],y=mb_train['y'],ax=ax[1])
```

Out[12]: <AxesSubplot:xlabel='X1', ylabel='y'>



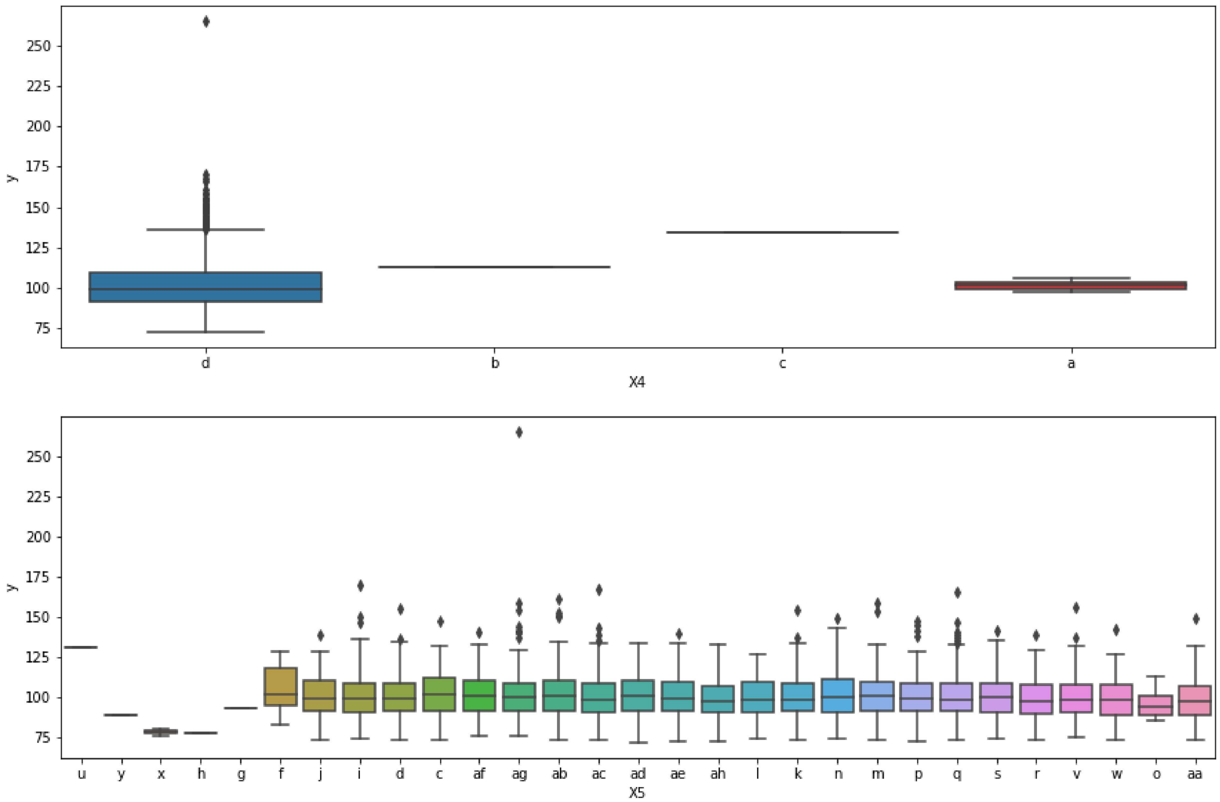
In [13]:
```
fig,ax=plt.subplots(2,1,figsize=(15,10))
sns.boxplot(x=mb_train['X2'],y=mb_train['y'],ax=ax[0])
sns.boxplot(x=mb_train['X3'],y=mb_train['y'],ax=ax[1])
```

Out[13]: <AxesSubplot:xlabel='X3', ylabel='y'>



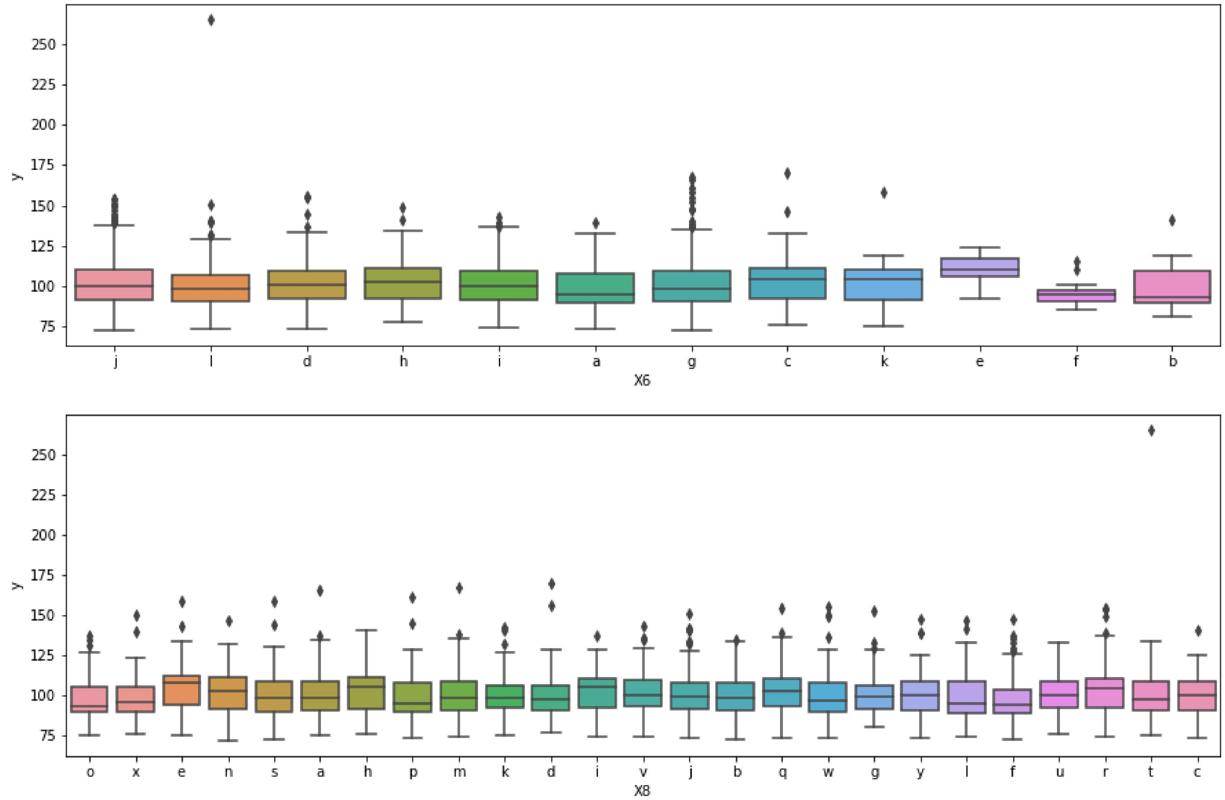
In [14]:
```
fig,ax=plt.subplots(2,1,figsize=(15,10))
sns.boxplot(x=mb_train['X4'],y=mb_train['y'],ax=ax[0])
sns.boxplot(x=mb_train['X5'],y=mb_train['y'],ax=ax[1])
```

Out[14]: <AxesSubplot:xlabel='X5', ylabel='y'>



In [15]:
```
fig,ax=plt.subplots(2,1,figsize=(15,10))
sns.boxplot(x=mb_train['X6'],y=mb_train['y'],ax=ax[0])
sns.boxplot(x=mb_train['X8'],y=mb_train['y'],ax=ax[1])
```

Out[15]: <AxesSubplot:xlabel='X8', ylabel='y'>



We obsereved that X4 has low variance so we can remove that feature from our dataset

In [16]:
```
mb_train1=mb_train.drop('X4',axis=1,inplace=True)
mb_test1=mb_test.drop('X4',axis=1,inplace=True)
```

In [17]:
```
mb_train.head()
```

Out[17]:

	y	X0	X1	X2	X3	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X38
0	130.81	k	v	at	a	u	j	o	0	0	...	0	0	1	0	0	0	
1	88.53	k	t	av	e	y	l	o	0	0	...	1	0	0	0	0	0	
2	76.26	az	w	n	c	x	j	x	0	0	...	0	0	0	0	0	0	
3	80.62	az	t	n	f	x	l	e	0	0	...	0	0	0	0	0	0	
4	78.02	az	v	n	f	h	d	n	0	0	...	0	0	0	0	0	0	

5 rows × 376 columns

Analysis of Binary Features-

In [18]:
```
mb_train_num=mb_train.select_dtypes(include='int64')
```

In [19]:
```
mb_train_num
```

Out[19]:

	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	...	X375	X376	X377	X378	X379	X
0	0	0	0	1	0	0	0	0	1	0	...	0	0	1	0	0	
1	0	0	0	0	0	0	0	0	1	0	...	1	0	0	0	0	
2	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	0	0	0	0	1	0	0	0	0	0	...	1	0	0	0	0	
4205	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	
4206	0	0	1	1	0	0	0	0	0	0	...	0	0	1	0	0	
4207	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	
4208	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	

4209 rows × 368 columns

In [20]:
```
# removing features with 0 variance
temp = []
for i in mb_train_num.columns:
    if mb_train_num[i].var()==0:
        temp.append(i)

print(len(temp))
print(temp)
```

```
12
['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297',
'X330', 'X347']
```

From this, we observed that there are 12 features that have constant value across all data points. So, as per my assumption, these features will not contribute to the modeling.

In [21]:
```
mb_train.drop(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X29
```

In [22]:
```
mb_train
```

Out[22]:

	y	X0	X1	X2	X3	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380
0	130.81	k	v	at	a	u	j	o	0	0	...	0	0	1	0	0	0
1	88.53	k	t	av	e	y	l	o	0	0	...	1	0	0	0	0	0
2	76.26	az	w	n	c	x	j	x	0	0	...	0	0	0	0	0	0
3	80.62	az	t	n	f	x	l	e	0	0	...	0	0	0	0	0	0
4	78.02	az	v	n	f	h	d	n	0	0	...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	107.39	ak	s	as	c	aa	d	q	0	0	...	1	0	0	0	0	0
4205	108.77	j	o	t	d	aa	h	h	0	0	...	0	1	0	0	0	0
4206	109.22	ak	v	r	a	aa	g	e	0	1	...	0	0	1	0	0	0
4207	87.48	al	r	e	f	aa	l	u	0	0	...	0	0	0	0	0	0
4208	110.85	z	r	ae	c	aa	g	w	0	0	...	1	0	0	0	0	0

4209 rows × 364 columns

In [23]:
```
mb_test.drop(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X29
```

In [24]:
```
mb_test
```

Out[24]:

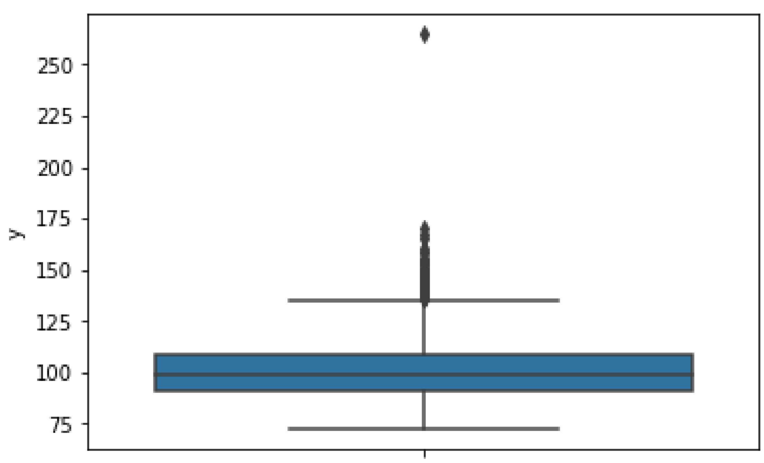
	X0	X1	X2	X3	X5	X6	X8	X10	X12	X13	...	X375	X376	X377	X378	X379	X380	X3
0	az	v	n	f	t	a	w	0	0	0	...	0	0	0	1	0	0	
1	t	b	ai	a	b	g	y	0	0	0	...	0	0	1	0	0	0	
2	az	v	as	f	a	j	j	0	0	0	...	0	0	0	1	0	0	
3	az	l	n	f	z	l	n	0	0	0	...	0	0	0	1	0	0	
4	w	s	as	c	y	i	m	0	0	0	...	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	aj	h	as	f	aa	j	e	0	0	0	...	0	0	0	0	0	0	
4205	t	aa	ai	d	aa	j	y	0	0	0	...	0	1	0	0	0	0	
4206	y	v	as	f	aa	d	w	0	0	0	...	0	0	0	0	0	0	
4207	ak	v	as	a	aa	c	q	0	0	1	...	0	0	1	0	0	0	
4208	t	aa	ai	c	aa	g	r	0	0	0	...	1	0	0	0	0	0	

4209 rows × 363 columns

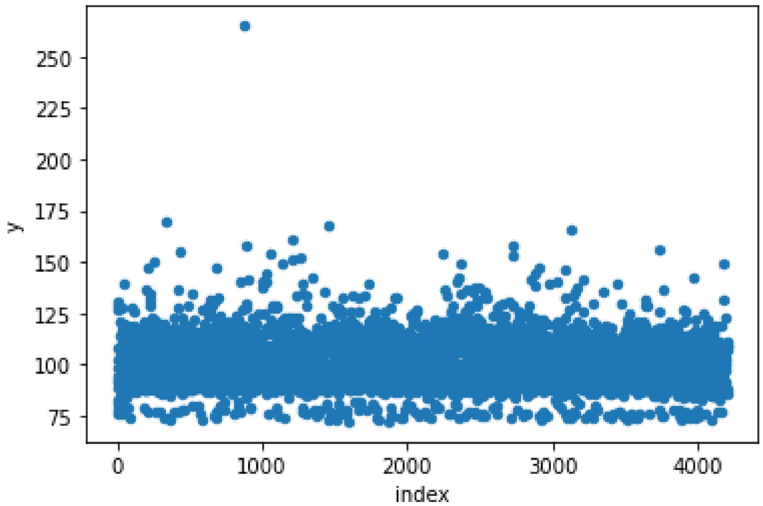
Analysis of Target variable(y):-

In [25]:
```
sns.boxplot(y='y',data=mb_train)
```

Out[25]: <AxesSubplot:ylabel='y'>



In [26]:
```
mb_train.reset_index().plot(kind='scatter', x='index', y='y')
plt.show()
```



Here, we can see there are many duplicates values and thershold of target variable lies between 150 and above it can be considered as outliers.

Check for duplicates features

In [27]:
```
mb_train1 = mb_train.drop_duplicates(keep=False)
```

In [28]:
```
mb_train1
```

Out[28]:

	y	X0	X1	X2	X3	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380
0	130.81	k	v	at	a	u	j	o	0	0	...	0	0	1	0	0	0
1	88.53	k	t	av	e	y	l	o	0	0	...	1	0	0	0	0	0
2	76.26	az	w	n	c	x	j	x	0	0	...	0	0	0	0	0	0
3	80.62	az	t	n	f	x	l	e	0	0	...	0	0	0	0	0	0
4	78.02	az	v	n	f	h	d	n	0	0	...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	107.39	ak	s	as	c	aa	d	q	0	0	...	1	0	0	0	0	0
4205	108.77	j	o	t	d	aa	h	h	0	0	...	0	1	0	0	0	0
4206	109.22	ak	v	r	a	aa	g	e	0	1	...	0	0	1	0	0	0
4207	87.48	al	r	e	f	aa	l	u	0	0	...	0	0	0	0	0	0
4208	110.85	z	r	ae	c	aa	g	w	0	0	...	1	0	0	0	0	0

4207 rows × 364 columns

In [29]:
```
mb_test1=mb_test.drop_duplicates(keep=False)
mb_test1
```

Out[29]:

	X0	X1	X2	X3	X5	X6	X8	X10	X12	X13	...	X375	X376	X377	X378	X379	X380	X3
0	az	v	n	f	t	a	w	0	0	0	...	0	0	0	1	0	0	
1	t	b	ai	a	b	g	y	0	0	0	...	0	0	1	0	0	0	
2	az	v	as	f	a	j	j	0	0	0	...	0	0	0	1	0	0	
3	az	l	n	f	z	l	n	0	0	0	...	0	0	0	1	0	0	
4	w	s	as	c	y	i	m	0	0	0	...	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	aj	h	as	f	aa	j	e	0	0	0	...	0	0	0	0	0	0	
4205	t	aa	ai	d	aa	j	y	0	0	0	...	0	1	0	0	0	0	
4206	y	v	as	f	aa	d	w	0	0	0	...	0	0	0	0	0	0	
4207	ak	v	as	a	aa	c	q	0	0	1	...	0	0	1	0	0	0	
4208	t	aa	ai	c	aa	g	r	0	0	0	...	1	0	0	0	0	0	

3691 rows × 363 columns

Check for null and unique values

In [30]:
```
mb_train1.isnull().sum().any()
```

Out[30]: False

In [31]:
```
mb_test1.isnull().sum().any()
```

Out[31]: False

Apply Label Encoder

In [32]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [33]:
```
mb_train_feature=mb_train1.drop(columns={'y'})
mb_train_target=mb_train1.y
print(mb_train_feature.shape)
print(mb_train_target.shape)
```

```
(4207, 363)
(4207,)
```

In [34]:
```
mb_train_feature.describe(include='object')
```

Out[34]:

	X0	X1	X2	X3	X5	X6	X8
count	4207	4207	4207	4207	4207	4207	4207
unique	47	27	44	7	29	12	25
top	z	aa	as	c	w	g	j
freq	360	833	1659	1942	231	1042	277

In [35]:
```
mb_train_feature['X0']=le.fit_transform(mb_train_feature.X0)
mb_train_feature['X1']=le.fit_transform(mb_train_feature.X1)
mb_train_feature['X2']=le.fit_transform(mb_train_feature.X2)
mb_train_feature['X3']=le.fit_transform(mb_train_feature.X3)
mb_train_feature['X5']=le.fit_transform(mb_train_feature.X5)
mb_train_feature['X6']=le.fit_transform(mb_train_feature.X6)
mb_train_feature['X8']=le.fit_transform(mb_train_feature.X8)
```

In [36]:
```
mb_train_feature
```

Out[36]:

	X0	X1	X2	X3	X5	X6	X8	X10	X12	X13	...	X375	X376	X377	X378	X379	X380	X3
0	32	23	17	0	24	9	14	0	0	1	...	0	0	1	0	0	0	
1	32	21	19	4	28	11	14	0	0	0	...	1	0	0	0	0	0	
2	20	24	34	2	27	9	23	0	0	0	...	0	0	0	0	0	0	
3	20	21	34	5	27	11	4	0	0	0	...	0	0	0	0	0	0	
4	20	23	34	5	12	3	13	0	0	0	...	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	8	20	16	2	0	3	16	0	0	0	...	1	0	0	0	0	0	
4205	31	16	40	3	0	7	7	0	0	0	...	0	1	0	0	0	0	
4206	8	23	38	0	0	6	4	0	1	1	...	0	0	1	0	0	0	
4207	9	19	25	5	0	11	20	0	0	0	...	0	0	0	0	0	0	
4208	46	19	3	2	0	6	22	0	0	0	...	1	0	0	0	0	0	

4207 rows × 363 columns

In [46]:
```
mb_test
```

Out[46]:

	X0	X1	X2	X3	X5	X6	X8	X10	X12	X13	...	X375	X376	X377	X378	X379	X380	X3
0	az	v	n	f	t	a	w	0	0	0	...	0	0	0	1	0	0	
1	t	b	ai	a	b	g	y	0	0	0	...	0	0	1	0	0	0	
2	az	v	as	f	a	j	j	0	0	0	...	0	0	0	1	0	0	
3	az	l	n	f	z	l	n	0	0	0	...	0	0	0	1	0	0	
4	w	s	as	c	y	i	m	0	0	0	...	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	aj	h	as	f	aa	j	e	0	0	0	...	0	0	0	0	0	0	
4205	t	aa	ai	d	aa	j	y	0	0	0	...	0	1	0	0	0	0	
4206	y	v	as	f	aa	d	w	0	0	0	...	0	0	0	0	0	0	
4207	ak	v	as	a	aa	c	q	0	0	1	...	0	0	1	0	0	0	
4208	t	aa	ai	c	aa	g	r	0	0	0	...	1	0	0	0	0	0	

4209 rows × 363 columns

In [47]:
```
mb_test.describe(include='object')
```

Out[47]:

	X0	X1	X2	X3	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209
unique	49	27	45	7	32	12	25
top	ak	aa	as	c	v	g	e
freq	432	826	1658	1900	246	1073	274

In [48]:
```
mb_test['X0']=le.fit_transform(mb_test.X0)
mb_test['X1']=le.fit_transform(mb_test.X1)
mb_test['X2']=le.fit_transform(mb_test.X2)
mb_test['X3']=le.fit_transform(mb_test.X3)
mb_test['X5']=le.fit_transform(mb_test.X5)
mb_test['X6']=le.fit_transform(mb_test.X6)
mb_test['X8']=le.fit_transform(mb_test.X8)
```

In [49]:
```
mb_test
```

Out[49]:

	X0	X1	X2	X3	X5	X6	X8	X10	X12	X13	...	X375	X376	X377	X378	X379	X380	X3
0	21	23	34	5	26	0	22	0	0	0	...	0	0	0	1	0	0	
1	42	3	8	0	9	6	24	0	0	0	...	0	0	1	0	0	0	
2	21	23	17	5	0	9	9	0	0	0	...	0	0	0	1	0	0	
3	21	13	34	5	31	11	13	0	0	0	...	0	0	0	1	0	0	
4	45	20	17	2	30	8	12	0	0	0	...	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	6	9	17	5	1	9	4	0	0	0	...	0	0	0	0	0	0	
4205	42	1	8	3	1	9	24	0	0	0	...	0	1	0	0	0	0	
4206	47	23	17	5	1	3	22	0	0	0	...	0	0	0	0	0	0	
4207	7	23	17	0	1	2	16	0	0	1	...	0	0	1	0	0	0	
4208	42	1	8	2	1	6	17	0	0	0	...	1	0	0	0	0	0	

4209 rows × 363 columns

Perform Dimensionality Reduction

In [58]:
```
from sklearn.decomposition import PCA
pca=PCA(n_components=0.95)
```

In [59]:
```
pca.fit(mb_train_feature,mb_train_target)
```

Out[59]:
```
PCA(n_components=0.95)
```

In [60]:
```
mb_train_feature1=pca.fit_transform(mb_train_feature)
print(mb_train_feature1.shape)
```

```
(4207, 6)
```

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

Out[61]:
```
PCA(n_components=0.95)
```

In [62]:
```
mb_test_trans=pca.fit_transform(mb_test)
mb_test_trans.shape
```

Out[62]:
```
(4209, 6)
```

Predict values using XGBoost

In [63]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\admin\anaconda3\lib\site-pac
kages (1.6.2)
Requirement already satisfied: scipy in c:\users\admin\anaconda3\lib\site-packa
ges (from xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\admin\anaconda3\lib\site-packa
ges (from xgboost) (1.20.3)
```

In [64]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_squared_error
from math import sqrt
```

In [65]:
```
train_x,test_x,train_y,test_y=train_test_split(mb_train_feature1,mb_train_target,
print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
print(test_y.shape)
```

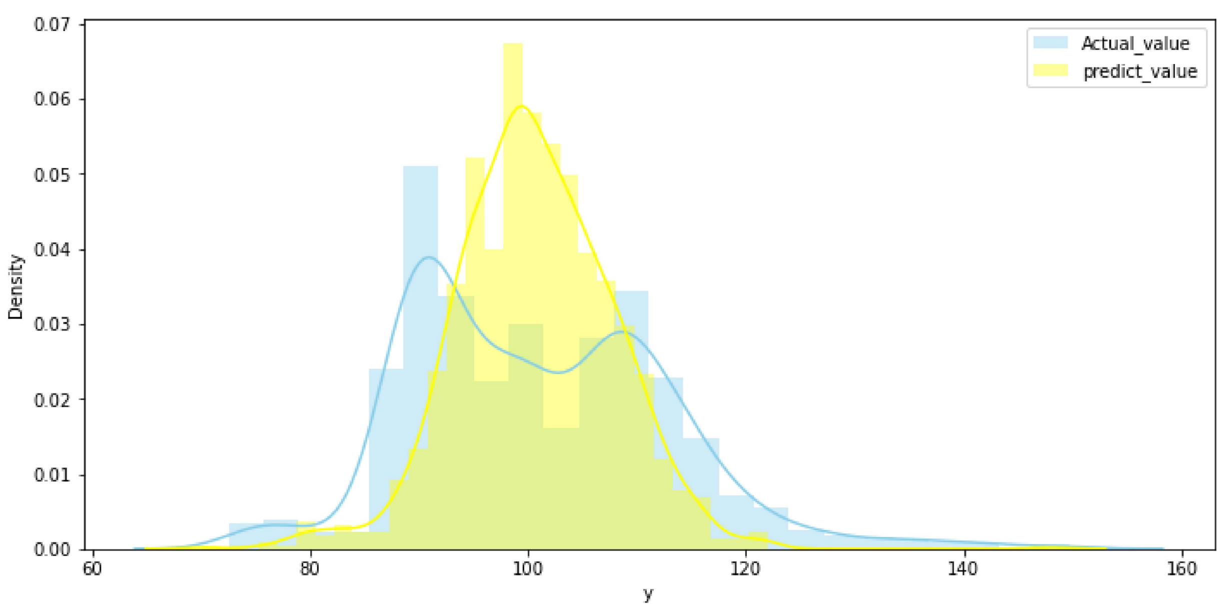
```
(2944, 6)
(2944,)
(1263, 6)
(1263,)
```

In [66]:
```
xgb_reg=xgb.XGBRegressor(objective='reg:linear',colsample_bytree=0.3,learning_rat
model=xgb_reg.fit(train_x,train_y)
print('RMSE=',sqrt(mean_squared_error(model.predict(test_x),test_y)))
```

```
[02:29:05] WARNING: C:/Users/administrator/workspace/xgboost-win64_release_1.6.
0/src/objective/regression_obj.cu:203: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE= 11.909503384884726
```

In [67]:
```
pred_test_y=model.predict(test_x)
plt.figure(figsize=(10,5))
sns.distplot(test_y[test_y<150],color="skyblue",label="Actual_value")
sns.distplot(pred_test_y[pred_test_y<150],color="yellow",label="predict_value")
plt.legend()
plt.tight_layout()
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Futur
eWarning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Futur
eWarning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```



In [68]:
```
test_pred=model.predict(mb_test_trans)
test_pred
```

Out[68]:
```
array([ 83.55246,  95.07384,  99.91767, ...,  92.87924, 120.2429 ,
        98.79591], dtype=float32)
```

In [ ]: