Mercedes-Benz Greener Manufacturing

In [1]:
```
#loading libaries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# imprting data
df_train = pd.read_csv('train_benz.csv')
df_test= pd.read_csv('test_benz.csv')
```

In [3]:
```
#Basic data check
print(df_train.shape)
print(df_test.shape)
```

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

In [4]:
```
df_train.head()
```

Out[4]:

	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 [5]:
```
df_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]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 378 entries, ID to X385
dtypes: float64(1), int64(369), object(8)
memory usage: 12.1+ MB
```

In [7]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 377 entries, ID to X385
dtypes: int64(369), object(8)
memory usage: 12.1+ MB
```

In [ ]:

In [ ]:

checking missing and Duplicate values in train and test dataset

In [8]:
```
#for i in df_train.columns:
#    if df_train[i].isnull().sum() >0:
#        print(i,'has missing values')
#    else:
#       print("There are no missing values in column", i)
```

In [9]:
```
df_train.isnull().sum().any()
```

Out[9]: False

In [10]:
```
if df_train.isnull().sum().any()==True:
    print('There are missing values in the training dataset')
else:
    print('There are no missing values in the training dataset')
```

```
There are no missing values in the training dataset
```

In [11]:
```
df_test.isnull().sum().any()
```

Out[11]:
```
False
```

In [12]:
```
#checking duplicate values
if df_test.isnull().sum().any()==True:
    print('There are missing values in the test dataset')
else:
    print('There are no missing values in the test dataset')
```

```
There are no missing values in the test dataset
```

In [13]:
```
df_train.duplicated().sum().any()
```

Out[13]:
```
False
```

In [14]:
```
if df_train.duplicated().sum().any()==True:
    print('There are duplicate values in the training dataset')
else:
    print('There are no duplicate values in the training dataset')
```

```
There are no duplicate values in the training dataset
```

In [15]:
```
df_test.duplicated().sum().any()
```

Out[15]:
```
False
```

In [16]:
```
if df_test.duplicated().sum().any()==True:
    print('There are duplicate values in the test dataset')
else:
    print('There are no duplicate values in the test dataset')
```

```
There are no duplicate values in the test dataset
```

checking categorical columns

In [17]:
```
df_train.describe(include='object')
```

Out[17]:

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	47	27	44	7	4	29	12	25
top	z	aa	as	c	d	w	g	j
freq	360	833	1659	1942	4205	231	1042	277

In [18]:
```
df_test.describe(include='object')
```

Out[18]:

	X0	X1	X2	X3	X4	X5	X6	X8
count	4209	4209	4209	4209	4209	4209	4209	4209
unique	49	27	45	7	4	32	12	25
top	ak	aa	as	c	d	v	g	e
freq	432	826	1658	1900	4203	246	1073	274

Observation 1:

```
*  Both training and test data sets have same number of records - 4209 ent
ries with 377 columns/features and one target  variable y
*  We have three types of data in the training dataset: float64(1), int64
(369), object(8)
* There are no missing values in both training and test dataset
* There are no duplicate values in both training and test dataset
* 8 categorical features in training nd test datasets are X0,X1 ,X2 ,X3 ,X
4 ,X5 ,X6 ,X8
```

In [ ]:
```

```

EDA

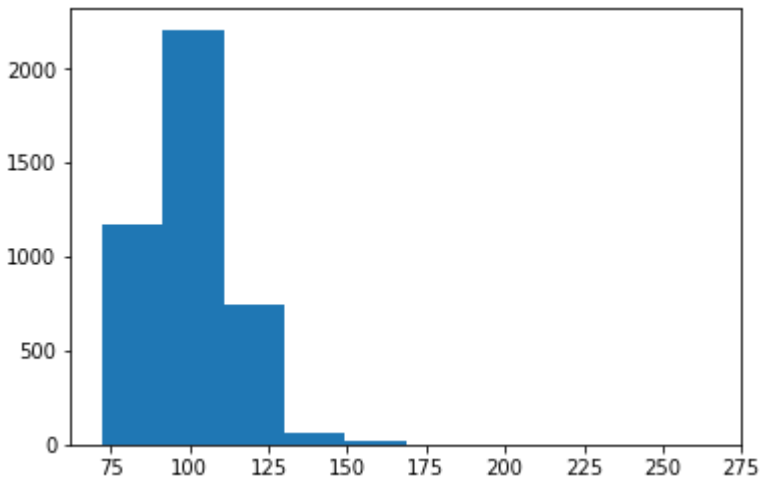
In [19]:
```
df_train.describe()
```

Out[19]:

	ID	y	X10	X11	X12	X13	X14
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000

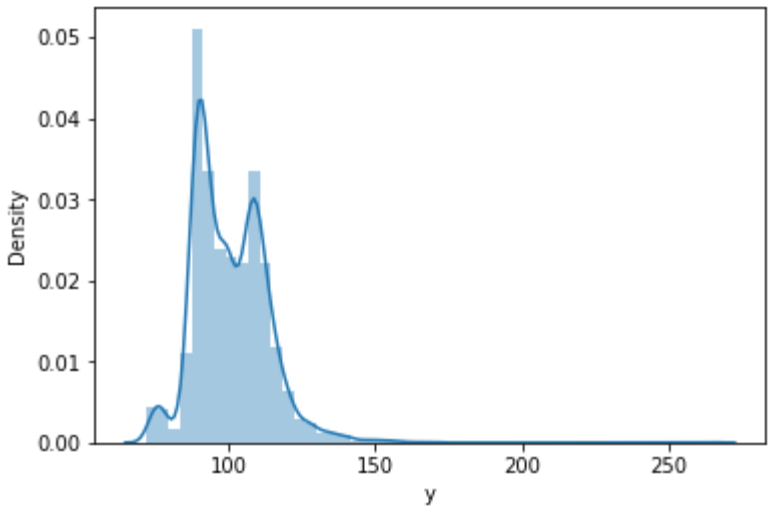
8 rows × 370 columns

In [20]:
```
plt.hist(df_train['y'])
plt.show()
```



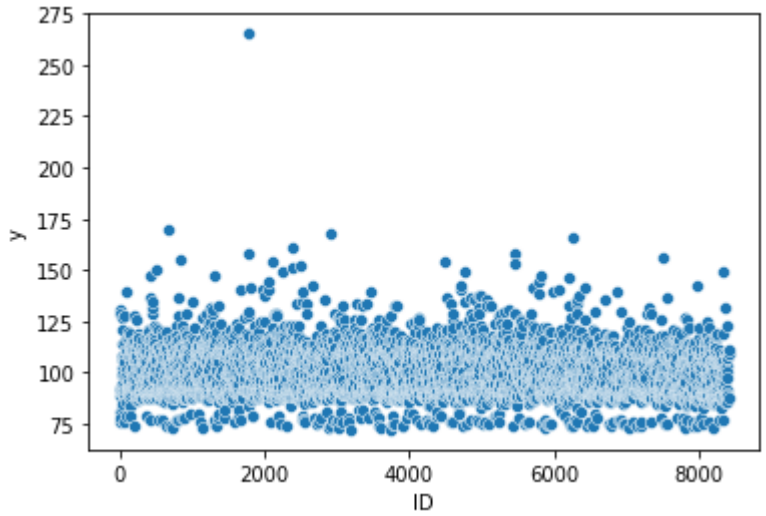
In [21]:
```
sns.distplot(df_train['y'])
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23678f89630>
```



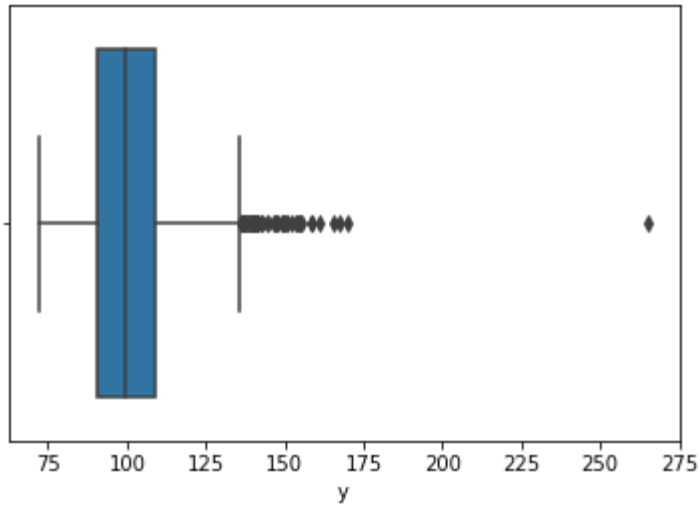
In [22]:
```
sns.scatterplot(data=df_train, x='ID', y="y")
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x236783247b8>
```



In [23]:
```
sns.boxplot(df_train['y'])
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2367841c828>
```



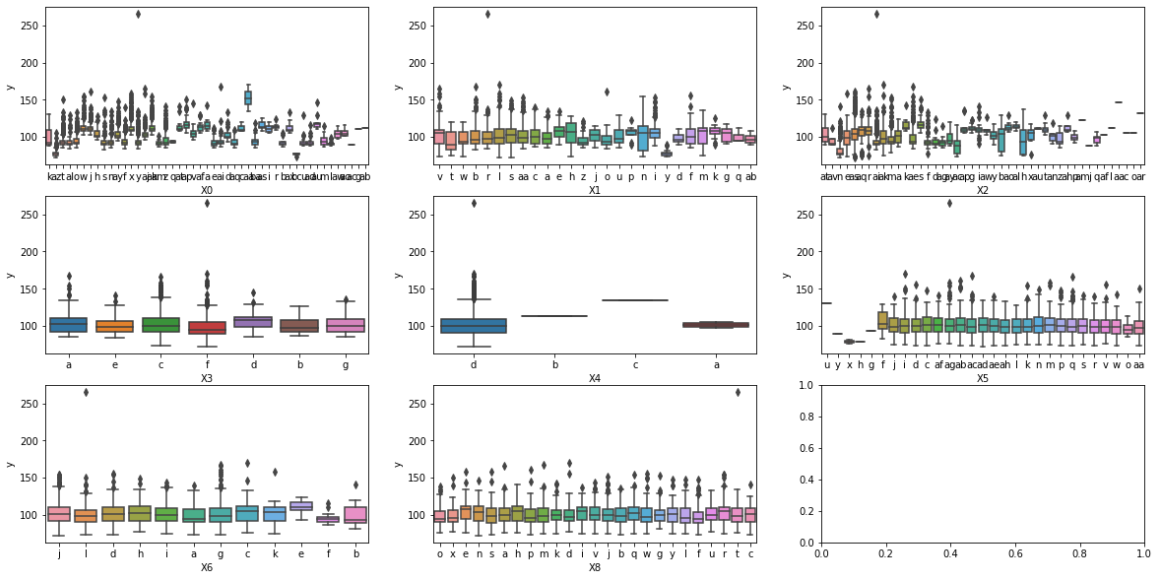
Observation 2:

* We see that most of the data points belong to the range 75 to 150 seconds
* we see one data pointwihch is in the range of 250 seconds. Presumbly this appears to be an outlier
* using boxplot we do see that there are some ouliers in the coly y

In [24]:
```
#Plotting box plot for categorical variables
```

In [25]:
```
fig, axes = plt.subplots(3, 3, figsize=(20, 10))
sns.boxplot(ax=axes[0, 0], data=df_train, x='X0', y='y')
sns.boxplot(ax=axes[0, 1], data=df_train, x='X1', y='y')
sns.boxplot(ax=axes[0, 2], data=df_train, x='X2', y='y')
sns.boxplot(ax=axes[1, 0], data=df_train, x='X3', y='y')
sns.boxplot(ax=axes[1, 1], data=df_train, x='X4', y='y')
sns.boxplot(ax=axes[1, 2], data=df_train, x='X5', y='y')
sns.boxplot(ax=axes[2, 0], data=df_train, x='X6', y='y')
sns.boxplot(ax=axes[2, 1], data=df_train, x='X8', y='y')
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x236785bdc18>
```



In [26]:
```
df_train['y'].describe()
```

Out[26]:
```
count    4209.000000
mean      100.669318
std        12.679381
min        72.110000
25%        90.820000
50%        99.150000
75%       109.010000
max       265.320000
Name: y, dtype: float64
```

In [27]:
```
df_train['y'].quantile(.75)
```

Out[27]:
```
109.01
```

In [28]:
```
# Treating Outliers with IQR method
Q1=df_train['y'].quantile(.25)
Q3=df_train['y'].quantile(0.75)
IQR=Q3-Q1
lower_bound = Q1-1.5*IQR
upper_bound = Q3+1.5*IQR
print(IQR)
print(lower_bound)
print(upper_bound)
```

```
18.190000000000012
63.534999999999975
136.29500000000002
```

In [29]:
```
#considering only the values that is above lowerbound and below upper b
ound
df_train1 = df_train[(df_train['y']>lower_bound)]
```

In [30]:
```
df_train1.shape
```

Out[30]: (4209, 378)

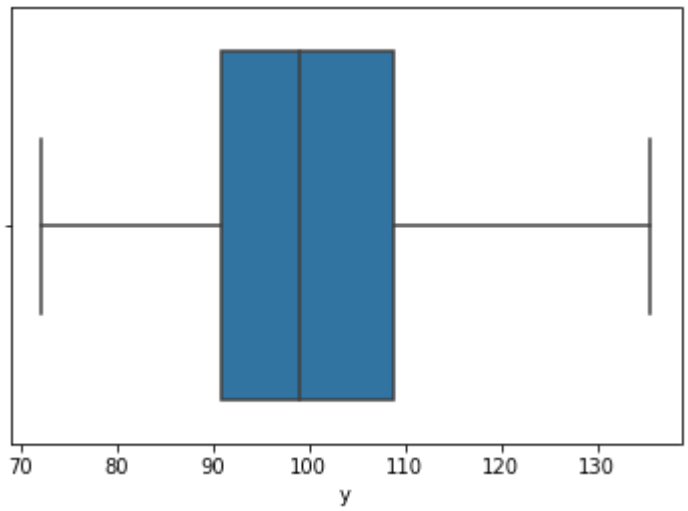
In [31]:
```
df_train1 = df_train1[(df_train1['y']<upper_bound)]
```

In [32]:
```
df_train1.shape
```

Out[32]: (4159, 378)

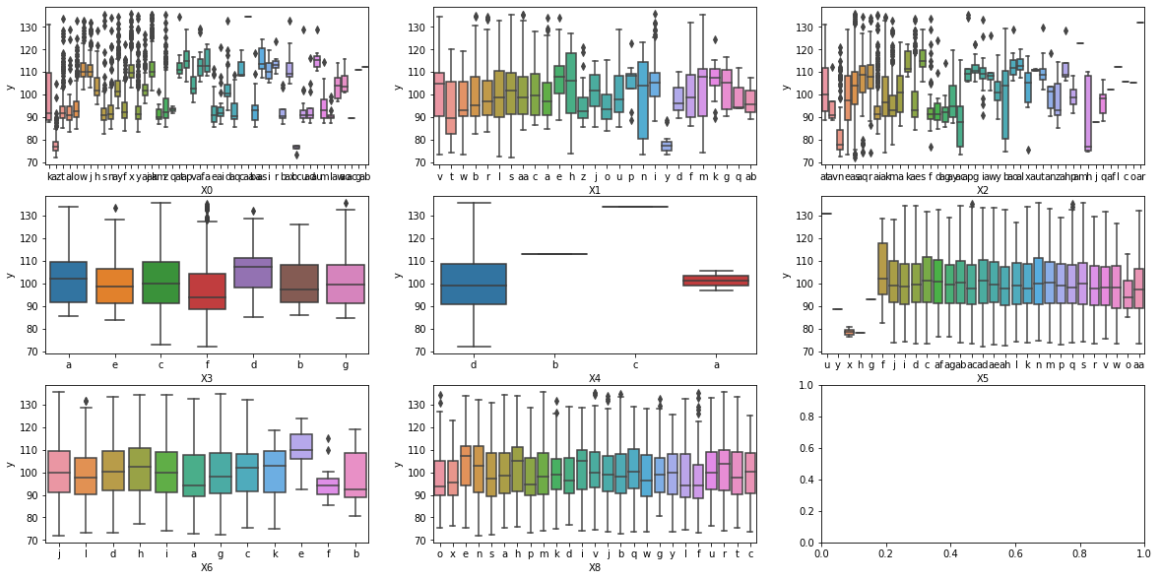
In [33]:
```
sns.boxplot(df_train1['y'])
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x2367d1b7400>



In [34]:
```
fig, axes = plt.subplots(3, 3, figsize=(20, 10))
sns.boxplot(ax=axes[0, 0], data=df_train1, x='X0', y='y')
sns.boxplot(ax=axes[0, 1], data=df_train1, x='X1', y='y')
sns.boxplot(ax=axes[0, 2], data=df_train1, x='X2', y='y')
sns.boxplot(ax=axes[1, 0], data=df_train1, x='X3', y='y')
sns.boxplot(ax=axes[1, 1], data=df_train1, x='X4', y='y')
sns.boxplot(ax=axes[1, 2], data=df_train1, x='X5', y='y')
sns.boxplot(ax=axes[2, 0], data=df_train1, x='X6', y='y')
sns.boxplot(ax=axes[2, 1], data=df_train1, x='X8', y='y')
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x2367cca45f8>



Observation 3:

* We can see after the removal of outliers(extreme 'y' values) dataset is looking much cleaner
* X4 appears to be low variance categorical feature

In [ ]:

Checking the other features, to check If for any column(s), the variance is equal to zero, then we need to remove those variable(s).

In [35]:
```
binary_df_train = df_train1.drop(['ID','y','X0','X1','X2','X3','X4','X5
','X6','X8'],axis=1)
```

In [36]:
```
print(binary_df_train.var())
```

```
X10     0.013287
X11     0.000000
X12     0.068998
X13     0.054602
X14     0.244838
X15     0.000481
X16     0.002639
X17     0.007637
X18     0.007637
X19     0.090425
X20     0.122626
X21     0.002639
X22     0.079285
X23     0.020486
X24     0.001920
X26     0.005025
X27     0.216528
X28     0.031638
X29     0.041636
X30     0.004549
X31     0.178619
X32     0.011176
X33     0.000240
X34     0.005263
X35     0.178619
X36     0.004549
X37     0.178619
X38     0.032312
X39     0.000240
X40     0.000721
          ...
X355    0.235517
X356    0.147232
X357    0.001201
X358    0.244628
X359    0.030964
X360    0.069407
X361    0.033209
X362    0.249637
X363    0.186003
X364    0.002878
X365    0.002878
X366    0.001201
X367    0.048819
X368    0.058411
X369    0.000481
X370    0.006689
X371    0.014222
X372    0.000481
X373    0.019101
X374    0.176025
X375    0.217054
X376    0.053751
X377    0.215552
X378    0.020486
X379    0.009292
```

```
X380    0.008110
X382    0.007637
X383    0.001201
X384    0.000481
X385    0.001441
Length: 368, dtype: float64
```

In [37]:
```
for i in binary_df_train.columns:
    if df_train1[i].var() <=0:
        print(i,'has 0 variance')
```

```
X11 has 0 variance
X93 has 0 variance
X107 has 0 variance
X233 has 0 variance
X235 has 0 variance
X268 has 0 variance
X289 has 0 variance
X290 has 0 variance
X293 has 0 variance
X297 has 0 variance
X330 has 0 variance
X339 has 0 variance
X347 has 0 variance
```

In [38]:
```
df_train2 = df_train1.drop(['X11','X93','X107','X233','X235','X268','X2
89','X290','X293','X297','X330','X339','X347'], axis=1)
df_train2.shape
```

Out[38]: (4159, 365)

In [39]:
```
df_train2.head()
```

Out[39]:

	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 × 365 columns

Observation 4:

- These columns (X11,X93,X107,X233,X235,X268,X289,X290,X293,X297,X330,X339,X347) are features with Zero variance
- removed these features from the dataset

In [40]:
```
# dropping X4 from the training dataset based on observation 3
df_train3 = df_train2.drop(['X4'],axis=1)
df_train3.shape
```

Out[40]: (4159, 364)

Applying label encoder for the cetegorical features

In [41]:
```
df_train3.head()
```

Out[41]:

	ID	y	X0	X1	X2	X3	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X
0	0	130.81	k	v	at	a	u	j	o	0	...	0	0	1	0	0	0	
1	6	88.53	k	t	av	e	y	l	o	0	...	1	0	0	0	0	0	
2	7	76.26	az	w	n	c	x	j	x	0	...	0	0	0	0	0	0	
3	9	80.62	az	t	n	f	x	l	e	0	...	0	0	0	0	0	0	
4	13	78.02	az	v	n	f	h	d	n	0	...	0	0	0	0	0	0	

5 rows × 364 columns

In [42]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
cat_cols = ['X0', 'X1', 'X2', 'X3','X5','X6','X8']
df_train3[cat_cols] = df_train3[cat_cols].apply(LabelEncoder().fit_tran
sform)
df_train3.head()
```

Out[42]:

	ID	y	X0	X1	X2	X3	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X
0	0	130.81	32	23	16	0	24	9	14	0	...	0	0	1	0	0	0	
1	6	88.53	32	21	18	4	28	11	14	0	...	1	0	0	0	0	0	
2	7	76.26	20	24	33	2	27	9	23	0	...	0	0	0	0	0	0	
3	9	80.62	20	21	33	5	27	11	4	0	...	0	0	0	0	0	0	
4	13	78.02	20	23	33	5	12	3	13	0	...	0	0	0	0	0	0	

5 rows × 364 columns

In [ ]:
```

```

Perform dimensionality reduction

In [43]:
```
df_train_feature = df_train3.drop(['ID','y'], axis=1)
df_train_target = df_train3['y']
print(df_train_feature.shape)
print(df_train_target.shape)
```

```
(4159, 362)
(4159,)
```

In [44]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=.95)
```

In [45]:
```
pca.fit(df_train_feature, df_train_target)
```

Out[45]:
```
PCA(n_components=0.95)
```

In [46]:
```
df_train_feature_trans = pca.fit_transform(df_train_feature)
print(df_train_feature_trans.shape)
```

```
(4159, 6)
```

Predict your test_df values using XGBoost.

In [87]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error, accuracy_scor
e
from math import sqrt
```

In [88]:
```
X = df_train_feature_trans
y = df_train_target
```

In [89]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,r
andom_state=1)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(3327, 6)
(3327,)
(832, 6)
(832,)
```

In [90]:
```
from xgboost import XGBRegressor
xgb_model = XGBRegressor()
```

In [91]:
```
xgb_model.fit(X_train, y_train)
```

Out[91]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=
-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=
6,
             min_child_weight=1, missing=nan, monotone_constraints=
'()',
             n_estimators=100, n_jobs=8, num_parallel_tree=1, random_
state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample
=1,
             tree_method='exact', validate_parameters=1, verbosity=No
ne)
```

In [92]:
```
xgb_pred_test = xgb_model.predict(X_test)
```

In [93]:
```
print('mean squared error= ',mean_squared_error(y_pred=xgb_pred_test, y
_true=y_test))
print('RMSE = ',sqrt(mean_squared_error(y_pred=xgb_pred_test, y_true=y_
test)))
```

```
mean squared error=  83.3043940006912
RMSE =  9.127124081587322
```

In [70]:
```
# XGBoost's hyperparameters tuning manually
```

In [94]:
```
xgb_reg_model = XGBRegressor(objective ='reg:linear', colsample_bytree
= 0.3, learning_rate = 0.4, max_depth = 10, alpha = 6,
                             n_estimators = 20)
```

In [95]:
```
xgb_reg_model.fit(X_train, y_train)
```

```
[19:48:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_re
lease_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now de
precated in favor of reg:squarederror.
```

Out[95]:
```
XGBRegressor(alpha=6, base_score=0.5, booster='gbtree', colsample_byl
evel=1,
             colsample_bynode=1, colsample_bytree=0.3, gamma=0, gpu_i
d=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.4, max_delta_step=0, max_depth=10,
             min_child_weight=1, missing=nan, monotone_constraints=
'()',
             n_estimators=20, n_jobs=8, num_parallel_tree=1,
             objective='reg:linear', random_state=0, reg_alpha=6, reg
_lambda=1,
             scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

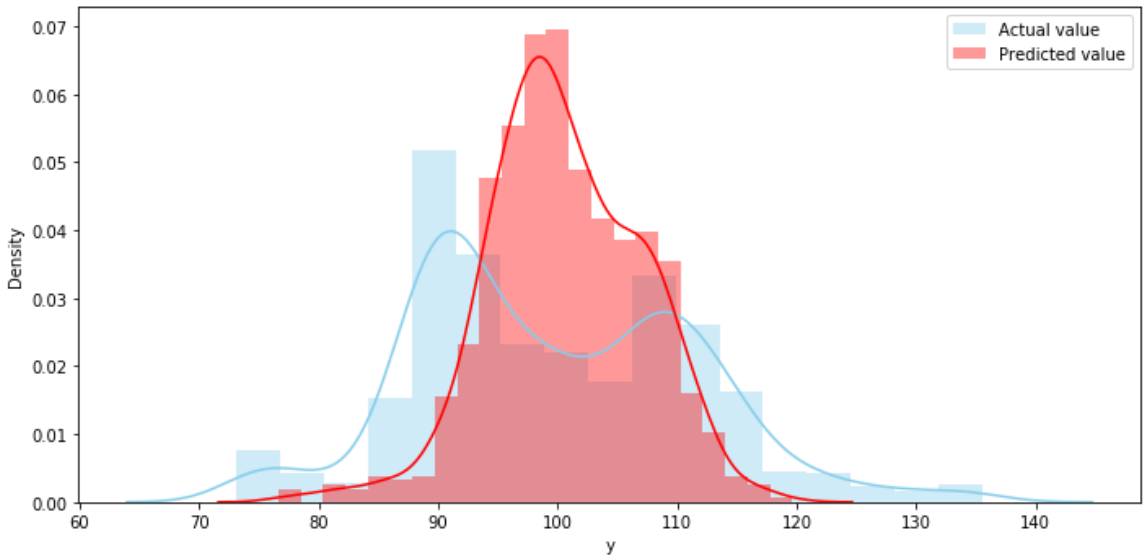
In [96]:
```
xgb_pred_test_y = xgb_reg.predict(X_test)
```

In [99]:
```
print('mean squared error= ',mean_squared_error(y_pred=xgb_pred_test_y,
y_true=y_test))
print('RMSE = ',sqrt(mean_squared_error(y_pred=xgb_pred_test_y, y_true=
y_test)))
```

```
mean squared error=  111.11380628279547
RMSE =  10.541053376337464
```

In [100]:
```
plt.figure(figsize=(10,5))
sns.distplot(y_test[y_test<160], color="skyblue", label="Actual valu
e")
sns.distplot(xgb_pred_test_y[xgb_pred_test_y<160] , color="red", label
="Predicted value")
plt.legend()

plt.tight_layout()
```



In [101]:
```
# k-fold Cross Validation using XGBoost
import xgboost as xgb
dmatrix_train = xgb.DMatrix(data=X,label=y)

params = {'objective':'reg:linear', 'colsample_bytree': 0.3, 'learning
_rate': 0.3, 'max_depth': 5, 'alpha': 10}

model_cv = xgb.cv(dtrain=dmatrix_train, params=params, nfold=3, num_bo
ost_round=50, early_stopping_rounds=10,
                        metrics="rmse", as_pandas=True, seed=7)
model_cv.tail(4)
```

```
[19:49:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_re
lease_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now de
precated in favor of reg:squarederror.
[19:49:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_re
lease_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now de
precated in favor of reg:squarederror.
[19:49:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_re
lease_1.4.0/src/objective/regression_obj.cu:171: reg:linear is now de
precated in favor of reg:squarederror.
```

Out[101]:

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
31	8.102104	0.118755	9.753032	0.048490
32	8.052445	0.102472	9.746553	0.046135
33	8.010449	0.097193	9.745268	0.043489
34	7.972205	0.089377	9.734202	0.049432

Obesrvation after modelling using XGBoost:

* using k-fold cross validation, RMSE comes as 9.7

In [ ]:

Prediction using the test data

In [79]:
```
df_test.describe()
```

Out[79]:

	ID	X10	X11	X12	X13	X14	
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.00
mean	4211.039202	0.019007	0.000238	0.074364	0.061060	0.427893	0.00
std	2423.078926	0.136565	0.015414	0.262394	0.239468	0.494832	0.02
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	2115.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
50%	4202.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
75%	6310.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.00
max	8416.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00

8 rows × 369 columns

In [81]:
```
#dropping columns where variance = zero from test dataset
df_test1 = df_test.drop(['X11','X93','X107','X233','X235','X268','X289
','X290','X293','X297','X330','X339','X347'], axis=1)
df_test1.shape
```

Out[81]: (4209, 364)

In [82]:
```
df_test1.head()
```

Out[82]:

	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 × 364 columns

In [83]:
```
df_test2 = df_test1.drop(['X4'],axis=1)
df_test2.shape
```

Out[83]: (4209, 363)

In [84]:
```
# Applying label encoder for the cetegorical features
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
cat_cols = ['X0', 'X1', 'X2', 'X3','X5','X6','X8']
df_test2[cat_cols] = df_test2[cat_cols].apply(LabelEncoder().fit_transform)
df_test2.head()
```

Out[84]:

	ID	X0	X1	X2	X3	X5	X6	X8	X10	X12	...	X375	X376	X377	X378	X379	X380	X38
0	1	21	23	34	5	26	0	22	0	0	...	0	0	0	1	0	0	
1	2	42	3	8	0	9	6	24	0	0	...	0	0	1	0	0	0	
2	3	21	23	17	5	0	9	9	0	0	...	0	0	0	1	0	0	
3	4	21	13	34	5	31	11	13	0	0	...	0	0	0	1	0	0	
4	5	45	20	17	2	30	8	12	0	0	...	1	0	0	0	0	0	

5 rows × 363 columns

In [85]:
```
#Perform dimensionality reduction
df_test_feature = df_test2.drop(['ID'], axis=1)
print(df_train_feature.shape)


from sklearn.decomposition import PCA
pca = PCA(n_components=.95)

pca.fit(df_test_feature)
```

```
(4159, 362)
```

Out[85]:
```
PCA(n_components=0.95)
```

In [86]:
```
df_test_feature_trans = pca.fit_transform(df_test_feature)
print(df_test_feature_trans.shape)
```

```
(4209, 6)
```

In [103]:
```
test_pred = xgb_reg_model.predict(df_test_feature_trans)
test_pred
```

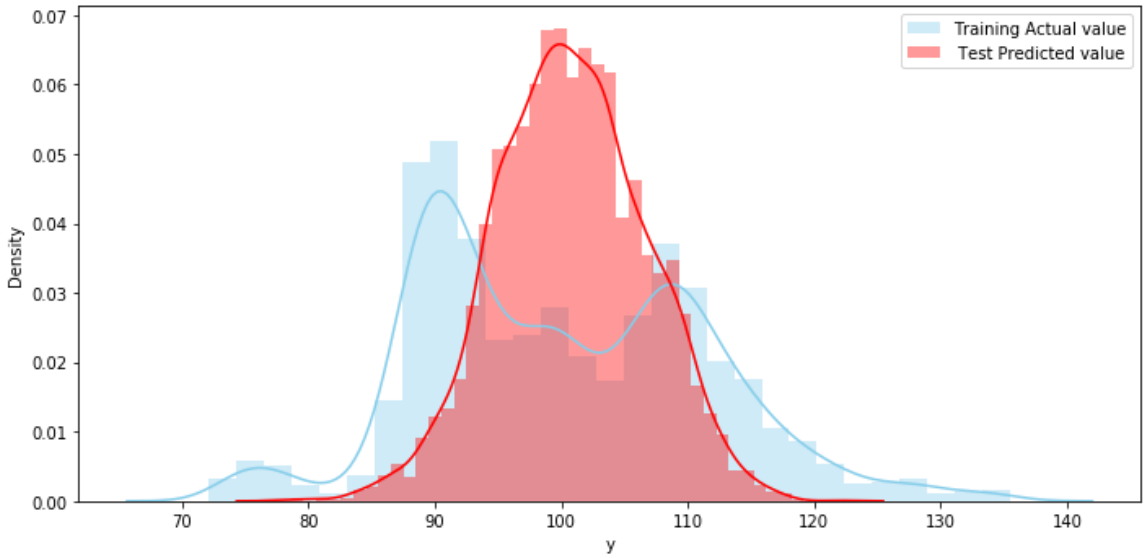
Out[103]:
```
array([ 83.72235 ,  94.573235,  98.27545 , ...,  97.5608  , 112.67989
,
       104.503815], dtype=float32)
```

In [ ]:
```

```

In [110]:
```
plt.figure(figsize=(10,5))
sns.distplot(df_train_target[df_train_target<160], color="skyblue", la
bel="Training Actual value")
sns.distplot(test_pred[test_pred<160] , color="red", label=" Test Pred
icted value")
plt.legend()

plt.tight_layout()
```



In [ ]: