Mercedes Benz Project

Importing datasets into python environment

In [1]:
```
import pandas as pd
import numpy as np
df_train=pd.read_csv(r'C:\Users\kuldeep\my files\Machine learning\mercedes benz project\train.csv')
df_test=pd.read_csv(r'C:\Users\kuldeep\my files\Machine learning\mercedes benz project\test.csv')
df_train.head()
```

Out[1]:

	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 [2]:
```
df_test.head()
```

Out[2]:

	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

`As we can see the 'y' column is missing in test dataset, y column is our target column and rest are features`

Shape of datasets

In [3]:
```
print('Shape of Train dataset :',df_train.shape)
print('Shape of Test dataset :',df_test.shape)
```
```
Shape of Train dataset : (4209, 378)
Shape of Test dataset : (4209, 377)
```

Finding if there are null values in dataset

In [4]:
```
df_train.isnull().sum().any()
```
Out[4]:
```
False
```

In [5]:
```
df_test.isnull().sum().any()
```
Out[5]:
```
False
```

`There are no null values in train and test dataframe`

Checking if any column has standard deviation = 0

In [6]:
```
df_train.describe()
```

Out[6]:

	ID	y	X10	X11	X12	X13	X14	X15	X16	X17	...
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	...
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.000475	0.002613	0.007603	...
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.021796	0.051061	0.086872	...
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	...
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...

8 rows × 370 columns

`'X11' column have std=0, there might be more such columns`

Getting number of columns which has std=0

In [7]:
```
(df_train.describe().iloc[2:3,:]==0).T.sum()
```
Out[7]:
```
std    12
dtype: int64
```

`There are total 12 columns with std=0`

Getting names of columns for which std=0

In [8]:
```
df_std=(df_train.describe().iloc[2:3,:].T==0)
df_std.index[df_std['std']==True]
```
Out[8]:
```
Index(['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293',
       'X297', 'X330', 'X347'],
      dtype='object')
```

Dropping These columns from Train and Test Dataframes

Along with ID column which is irrelevent

In [9]:
```
df_train.drop(['ID','X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293','X297', 'X330', 'X347'
df_test.drop(['ID','X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293','X297', 'X330', 'X347']

# Getting resulting shapes

print('Shape of Train dataset :',df_train.shape)
print('Shape of Test dataset :',df_test.shape)
```
```
Shape of Train dataset : (4209, 365)
Shape of Test dataset : (4209, 364)
```

Getting object columns in Train and Test Dfs

In [10]:
```
df_train.dtypes[df_train.dtypes=='object'].index
```
Out[10]:
```
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='object')
```

In [11]:
```
df_test.dtypes[df_test.dtypes=='object'].index
```
Out[11]:
```
Index(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='object')
```

`Above listed are object columns in both datasets`

getting unique values for object columns

In [12]:
```
# listing object columns
obj_cols=list(df_train.dtypes[df_train.dtypes=='object'].index)
obj_cols
```
Out[12]:
```
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

In [13]:
```
for x in obj_cols:
    print('Unique vaues in df_train',[x],'column :','\n',df_train[x].unique(),'\n')
```
```
Unique vaues in df_train ['X0'] column :
 ['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']

Unique vaues in df_train ['X1'] column :
 ['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']

Unique vaues in df_train ['X2'] column :
 ['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']

Unique vaues in df_train ['X3'] column :
 ['a' 'e' 'c' 'f' 'd' 'b' 'g']

Unique vaues in df_train ['X4'] column :
 ['d' 'b' 'c' 'a']

Unique vaues in df_train ['X5'] column :
 ['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']

Unique vaues in df_train ['X6'] column :
 ['j' 'l' 'd' 'h' 'i' 'a' 'g' 'c' 'k' 'e' 'f' 'b']

Unique vaues in df_train ['X8'] column :
 ['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']
```

`Above mentioned are column-wise unique values`

Label encoding object columns in Train and Test both datasets

In [14]:
```
from sklearn.preprocessing import LabelEncoder
l_encode=LabelEncoder()
for x in obj_cols:
    df_train[x]=l_encode.fit_transform(df_train[x])
    df_test[x]=l_encode.fit_transform(df_test[x])
```

Checking unique values after label encoding

In [15]:
```
for x in obj_cols:
    print('Unique vaues in df_train',[x],'column :','\n',df_train[x].unique(),'\n')
```
```
Unique vaues in df_train ['X0'] column :
 [32 20 40  9 36 43 31 29 39 35 19 27 44 45  7  8 10 46 37 15 12 42  5  0
 26  6 25 13 24  1 22 14 30 38 21 18 23 41  4 16 34 33 17 11  3 28  2]

Unique vaues in df_train ['X1'] column :
 [23 21 24  3 19 13 20  1  4  0  6  9 26 11 16 22 17 15 10 25  5  7 14 12
  8 18  2]

Unique vaues in df_train ['X2'] column :
 [17 19 34 25 16 14 38  7  8 33  0 31  3 39 26 24  5 21  2 13 27 29 20 42
 22 12  9 28 41 18 40 11 43  6 36 10 30 37  4 32  1 23 35 15]

Unique vaues in df_train ['X3'] column :
 [0 4 2 5 3 1 6]

Unique vaues in df_train ['X4'] column :
 [3 1 2 0]

Unique vaues in df_train ['X5'] column :
 [24 28 27 12 11 10 14 13  9  8  5  6  1  2  3  4  7 16 15 18 17 20 21 23
 22 25 26 19  0]

Unique vaues in df_train ['X6'] column :
 [ 9 11  3  7  8  0  6  2 10  4  5  1]

Unique vaues in df_train ['X8'] column :
 [14 23  4 13 18  0  7 15 12 10  3  8 21  9  1 16 22  6 24 11  5 20 17 19
  2]
```

Scaling the df_train

In [184]:
```
# First splitting target(y) from df_train
y=df_train['y']
x=df_train.drop(['y'],axis=1)
print(y.head())
print(x.head())
```
```
0    130.81
1     88.53
2     76.26
3     80.62
4     78.02
Name: y, dtype: float64
   X0  X1  X2  X3  X4  X5  X6  X8  X10  X12  ...  X375  X376  X377  X378  \
0  32  23  17   0   3  24   9  14    0    0  ...     0     0     1     0
1  32  21  19   4   3  28  11  14    0    0  ...     1     0     0     0
2  20  24  34   2   3  27   9  23    0    0  ...     0     0     0     0
3  20  21  34   5   3  27  11   4    0    0  ...     0     0     0     0
4  20  23  34   5   3  12   3  13    0    0  ...     0     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

[5 rows x 364 columns]
```

Importing min max scaler

In [186]:
```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
```

Scaling x data

In [188]:
```
x=scale.fit_transform(x)
```

Visualising for confirmation

In [190]:
```
pd.DataFrame(x)
```

Out[190]:

	0	1	2	3	4	5	6	7	8	9	...	354	355	356	357	358	359	360	361	362	3
0	0.695652	0.884615	0.395349	0.000000	1.0	0.857143	0.818182	0.583333	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.695652	0.807692	0.441860	0.666667	1.0	1.000000	1.000000	0.583333	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.434783	0.923077	0.790698	0.333333	1.0	0.964286	0.818182	0.958333	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
3	0.434783	0.807692	0.790698	0.833333	1.0	0.964286	1.000000	0.166667	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.434783	0.884615	0.790698	0.833333	1.0	0.428571	0.272727	0.541667	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4204	0.173913	0.769231	0.372093	0.333333	1.0	0.000000	0.272727	0.666667	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4205	0.673913	0.615385	0.930233	0.500000	1.0	0.000000	0.636364	0.291667	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4206	0.173913	0.884615	0.883721	0.833333	1.0	0.000000	0.545455	0.166667	0.0	1.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
4207	0.195652	0.730769	0.581395	0.833333	1.0	0.000000	1.000000	0.833333	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4208	1.000000	0.730769	0.069767	0.333333	1.0	0.000000	0.545455	0.916667	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

4209 rows × 364 columns

In [191]:
```
pd.DataFrame(x).describe()
```

Out[191]:

	0	1	2	3	4	5	6	7	8	9	...
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	...
mean	0.646973	0.427445	0.402476	0.486616	0.999287	0.476437	0.618847	0.483805	0.013305	0.075077	...
std	0.298660	0.328115	0.253186	0.289985	0.024633	0.294673	0.265179	0.293245	0.114590	0.263547	...
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.413043	0.115385	0.186047	0.333333	1.000000	0.178571	0.545455	0.208333	0.000000	0.000000	...
50%	0.760870	0.500000	0.372093	0.333333	1.000000	0.535714	0.636364	0.458333	0.000000	0.000000	...
75%	0.934783	0.769231	0.581395	0.833333	1.000000	0.750000	0.818182	0.750000	0.000000	0.000000	...
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...

8 rows × 364 columns

Dimensionality reduction using PCA

In [192]:
```
from sklearn.decomposition import PCA   # Importing PCA
```

In [196]:
```
pca=PCA(n_components=0.95)         # Dimensionality reduction to keep 95% variance
pca.fit(x)                         # Fitting train dataset(x) into PCA
pca.explained_variance_ratio_      # Getting columns needed to keep 95% variance
```
Out[196]:
```
array([0.12971609, 0.08752198, 0.08518613, 0.06747936, 0.05637602,
       0.06695891, 0.03763481, 0.03240439, 0.0277885 , 0.02474944,
       0.02309335, 0.019499  , 0.01695621, 0.01628515, 0.01522578,
       0.01464508, 0.01375481, 0.01211626, 0.01030365, 0.01012051,
       0.00959583, 0.00888674, 0.00876302, 0.00834512, 0.00790975,
       0.00755243, 0.0072127 , 0.00647693, 0.00638372, 0.0057495 ,
       0.00527787, 0.00506331, 0.00475259, 0.00460551, 0.00437066,
       0.00423079, 0.00420284, 0.00403129, 0.00400921, 0.0038454 ,
       0.00359432, 0.00343719, 0.00342564, 0.00335641, 0.0033054 ,
       0.00314255, 0.00309007, 0.00297706, 0.00289158, 0.0028196 ,
       0.00270358, 0.00260119, 0.00257916, 0.00244181, 0.00236859,
       0.00234041, 0.0022658 , 0.0021305 , 0.0020916 , 0.00200343,
       0.00197906, 0.00194319, 0.00183772, 0.00178412, 0.0017257 ,
       0.00170344, 0.00165346, 0.00160384, 0.00159121, 0.00152272,
       0.00147502, 0.00142071, 0.0013887 , 0.00137817, 0.00134106,
       0.00130345])
```

In [197]:
```
pca.explained_variance_ratio_.shape
```
Out[197]:
```
(76,)
```

In [198]:
```
np.sum(pca.explained_variance_ratio_)
```
Out[198]:
```
0.9503070409802702
```

`76 columns carry 95% variance and hence dimensionality can be reduced from '365' to '76'`

Getting transformed train(x) dataset

In [199]:
```
x_trans=pca.transform(x)
x_trans.shape
```
Out[199]:
```
(4209, 76)
```

Performing train test split

In [200]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_trans,y,test_size=0.25,random_state=5)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```
```
(3156, 76)
(1053, 76)
(3156,)
(1053,)
```

Model building using XGBoost

In [201]:
```
from xgboost import XGBRegressor
xgb_model=XGBRegressor()
```

In [202]:
```
xgb_model.fit(x_train,y_train)
```
Out[202]:
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, ...)
```

In [223]:
```
from sklearn.metrics import r2_score,mean_squared_error
```

In [222]:
```
train_pred=xgb_model.predict(x_train)
test_pred=xgb_model.predict(x_test)
print('R2 score for x_train=',r2_score(y_train,train_pred))
print('R2 score for x_test=',r2_score(y_test,test_pred))
```
```
R2 score for x_train= 0.6445477412483394
R2 score for x_test= 0.47406706787565533
```

`A very good accuracy for train data while low accuracy for test data can be seen while using XGBoost regressor model with default settings. This indicates our model is overfitting`

Using GridSearchCV to find best parameters

In [210]:
```
params = {'booster': ['dart','gbtree',], 'learning_rate' : [0.025,0.05] , 'max_depth' : [3,5,6],  'subsample' :
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(xgb_model, params, cv = 2, n_jobs = -1,verbose=2)
grid_search.fit(x_train, y_train)
```
```
Fitting 2 folds for each of 72 candidates, totalling 144 fits
```
Out[210]:
```
GridSearchCV(cv=2,
             estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                    callbacks=None, colsample_bylevel=1,
                                    colsample_bynode=1, colsample_bytree=1,
                                    early_stopping_rounds=None,
                                    enable_categorical=False, eval_metric=None,
                                    gamma=0, gpu_id=-1, grow_policy='depthwise',
                                    importance_type=None,
                                    interaction_constraints='',
                                    learning_rate=0.025, max_bin=256,
                                    max_cat_to_on...
                                    max_depth=3, max_leaves=0,
                                    min_child_weight=5, missing=nan,
                                    monotone_constraints='()', n_estimators=300,
                                    n_jobs=0, num_parallel_tree=1,
                                    predictor='auto', random_state=0,
                                    reg_alpha=0, reg_lambda=1, ...),
             n_jobs=-1,
             param_grid={'booster': ['dart', 'gbtree'],
                         'learning_rate': [0.025, 0.05], 'max_depth': [3, 5, 6],
                         'n_estimators': [100, 200, 300],
                         'subsample': [0.6, 0.8]},
             verbose=2)
```

Getting Best parameters from Grid Search

In [211]:
```
grid_search.best_params_
```
Out[211]:
```
{'booster': 'gbtree',
 'learning_rate': 0.025,
 'max_depth': 3,
 'n_estimators': 300,
 'subsample': 0.6}
```

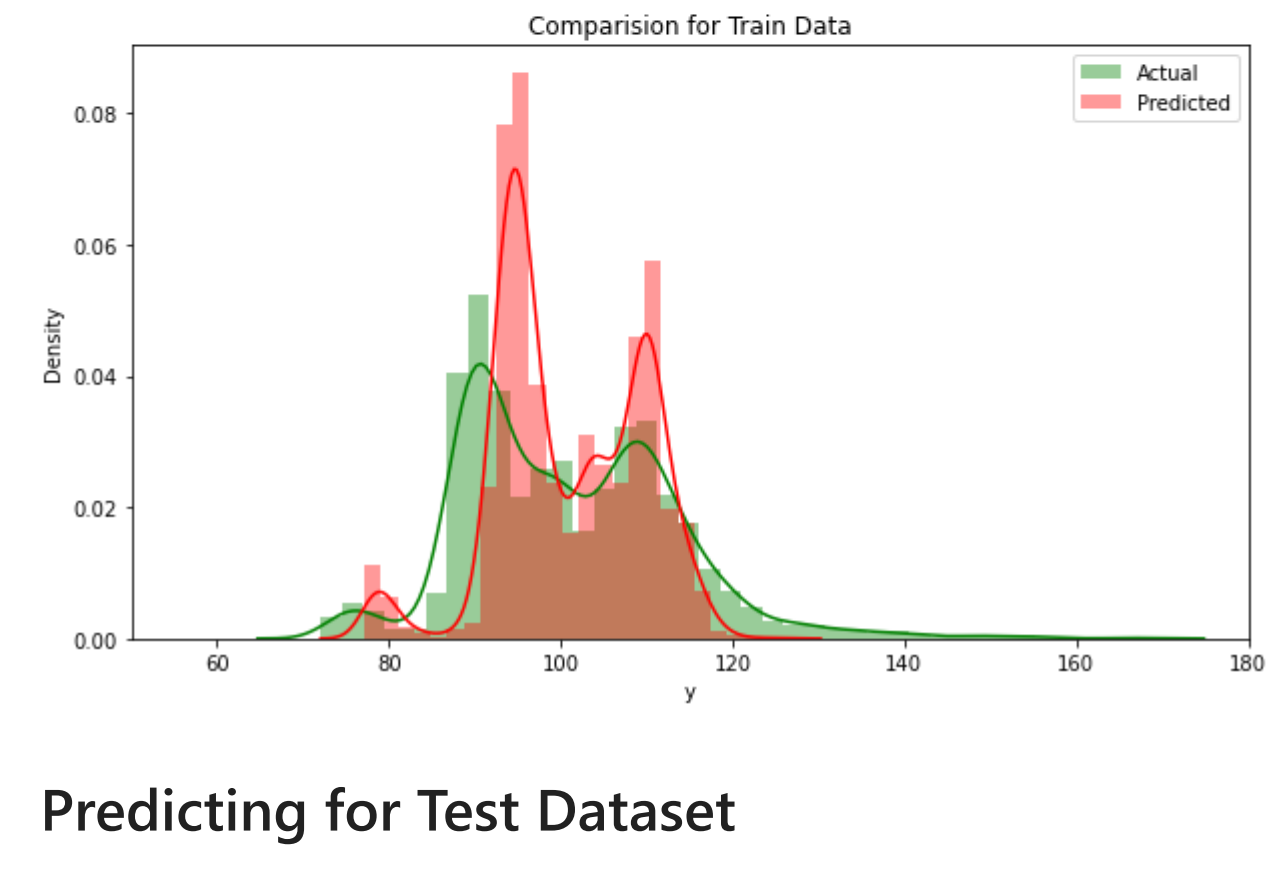
Using best parametesr and checking accuracy

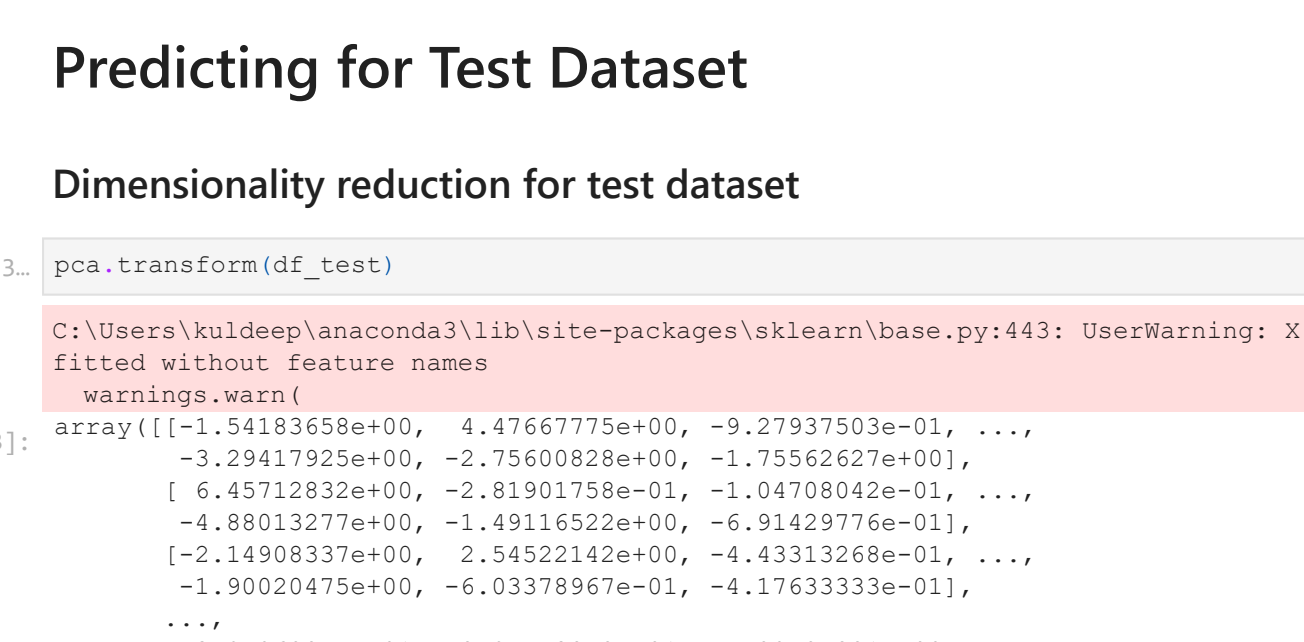
In [225]:
```
xgb_model=XGBRegressor(booster='gbtree',learning_rate= 0.025, max_depth= 3, min_child_weight= 5, subsample= 0.6
xgb_model.fit(x_train,y_train)
train_pred=xgb_model.predict(x_train)
test_pred=xgb_model.predict(x_test)
print('R2 score for x_trai n=',r2_score(y_train,train_pred))
print('RMSE for x_train =',np.sqrt(mean_squared_error(y_train,train_pred)))
print('R2 score for x_test =',r2_score(y_test,test_pred))
print('RMSE for x_test =',np.sqrt(mean_squared_error(y_test,test_pred)))
```
```
R2 score for x_trai n= 0.6445477412483394
RMSE for x_train = 7.37305555648733
R2 score for x_test = 0.47406706787565533
RMSE for x_test = 9.831868775025752
```

Visually comparing actual and predicted data using distplot

In [206]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [243]:
```
plt.figure(figsize=(10,12))
plt.subplot(211)
sns.distplot(y_test,color='g',label='Actual',)
sns.distplot(test_pred,color='r',label='Predicted')
plt.title('Comparision for Test Data')
plt.xlim(50,180)
plt.legend()
plt.subplot(212)
sns.distplot(y_train,color='g',label='Actual',)
sns.distplot(train_pred,color='r',label='Predicted')
plt.title('Comparision for Train Data')
plt.xlim(50,180)
plt.legend()
```
```
C:\Users\kuldeep\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a depr
ecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figu
re-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\kuldeep\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a depr
ecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figu
re-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\kuldeep\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a depr
ecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figu
re-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\kuldeep\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a depr
ecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figu
re-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```
Out[243]:
```
<matplotlib.legend.Legend at 0x1e00c030d90>
```





Predicting for Test Dataset

Dimensionality reduction for test dataset

In [253]:
```
pca.transform(df_test)
```
```
C:\Users\kuldeep\anaconda3\lib\site-packages\sklearn\base.py:443: UserWarning: X has feature names, but PCA was
fitted without feature names
  warnings.warn(
```
Out[253]:
```
array([[-1.54183658e+00,  4.47667775e+00, -9.27937503e-01, ...,
        -3.29417925e+00, -2.75600828e+00, -1.75562627e+00],
       [ 6.45712832e+00, -2.81901758e-01, -1.04708042e-01, ...,
        -4.88013277e+00, -1.49116522e+00, -6.91429776e-01],
       [-2.14908337e+00,  2.54522142e+00, -4.43313268e-01, ...,
        -1.90020475e+00, -6.03378967e-01, -4.17633333e-01],
       ...,
       [-3.97060877e-01, -8.94443872e-01, -4.00407881e+00, ...,
        -4.13580284e+00, -3.44746488e-01, -4.01920754e-01],
       [-3.90180322e+00,  1.43317578e+00,  2.19355380e+00, ...,
        -4.83084356e-01, -1.18877955e+00, -1.24119881e+00],
       [ 4.60209198e+00,  2.68275333e-04, -2.95056022e+00, ...,
        -4.22149654e+00, -4.61268640e-01, -2.81262972e-01]])
```

In [257]:
```
pca.explained_variance_ratio_.sum()
```
Out[257]:
```
0.9503070409802702
```

In [255]:
```
pca.explained_variance_ratio_.shape
```
Out[255]:
```
(76,)
```

In [256]:
```
test_trans=pca.transform(df_test)
test_trans.shape
```
```
C:\Users\kuldeep\anaconda3\lib\site-packages\sklearn\base.py:443: UserWarning: X has feature names, but PCA was
fitted without feature names
  warnings.warn(
```
Out[256]:
```
(4209, 76)
```

`Dimensions are reduced`

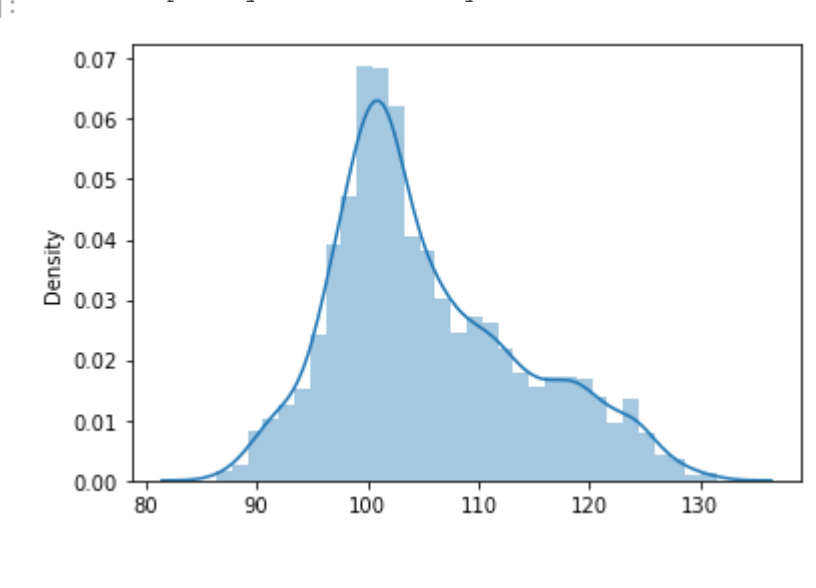
Predicting values for test dataset using XGBoost model with tuned hyper-parameters

In [258]:
```
test_predictions= xgb_model.predict(test_trans)
```

In [260]:
```
test_predictions.shape
```
Out[260]:
```
(4209,)
```

Visluaising predictions using distplot

In [261]:
```
sns.distplot(test_predictions)
```
```
C:\Users\kuldeep\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a depr
ecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figu
re-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```
Out[261]:
```
<AxesSubplot:ylabel='Density'>
```



Complete!

In [ ]: