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

In [2]:
```python
train_data = pd.read_csv(r'C:\Users\abha mohan\Desktop\train.csv')
test_data = pd.read_csv(r'C:\Users\abha mohan\Desktop\test.csv')
```

In [3]:
```python
train_data.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]:
```python
train_data.shape
```

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

In [5]:
```python
train_data.dtypes
```

Out[5]:
```
ID        int64
y       float64
X0       object
X1       object
X2       object
         ...
X380      int64
X382      int64
X383      int64
X384      int64
X385      int64
Length: 378, dtype: object
```

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

In [6]:
```python
train_data.var()
```

Out[6]:
```
ID      5.941936e+06
y       1.607667e+02
X10     1.313092e-02
X11     0.000000e+00
X12     6.945713e-02
            ...
X380    8.014579e-03
X382    7.546747e-03
X383    1.660732e-03
X384    4.750593e-04
X385    1.423823e-03
Length: 370, dtype: float64
```

In [7]:
```python
for i in train_data.columns:
    data_type = train_data[i].dtype
    if data_type == 'object':
        print(i)
```

```
X0
X1
X2
X3
X4
X5
X6
X8
```

In [8]:
```python
variance = pow(train_data.drop(columns={'ID','y'}).std(),2).to_dict()

null_cnt = 0
for key, value in variance.items():
    if(value==0):
        print(key)
        null_cnt = null_cnt+1
print('No of columns which has zero variance = ',null_cnt)
```

```
X11
X93
X107
X233
X235
X268
X289
X290
X293
X297
X330
X347
No of columns which has zero variance =  12
```

In [9]:
```python
train_data = train_data.drop(columns={'X11','X93','X107','X233','X235','X268','X289','X290','X293','X297','X330','X347'})
train_data.shape
```

Out[9]:
```
(4209, 366)
```

check for null values

In [10]:
```python
train_data.isnull().sum().sum()
```

Out[10]:
```
0
```

In [11]:
```python
test_data.isnull().sum().sum()
```

Out[11]:
```
0
```

unique values in train and test data

In [12]:
```python
train_data['X0'].unique()
```

Out[12]:
```
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 [13]:
```python
train_data['X1'].unique()
```

Out[13]:
```
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 [14]:
```python
train_data['X2'].unique()
```

Out[14]:
```
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 [15]:
```python
train_data['X3'].unique()
```

Out[15]:
```
array(['a', 'e', 'c', 'f', 'd', 'b', 'g'], dtype=object)
```

In [16]:
```python
train_data['X4'].unique()
```

Out[16]:
```
array(['d', 'b', 'c', 'a'], dtype=object)
```

In [17]:
```python
train_data['X5'].unique()
```

Out[17]:
```
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 [18]:
```python
train_data['X6'].unique()
```

Out[18]:
```
array(['j', 'l', 'd', 'h', 'i', 'a', 'g', 'c', 'k', 'e', 'f', 'b'],
      dtype=object)
```

In [19]:
```python
train_data['X8'].unique()
```

Out[19]:
```
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)
```

In [20]:
```python
test_data['X0'].unique()
```

Out[20]:
```
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 [21]:
```python
test_data['X1'].unique()
```

Out[21]:
```
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 [22]:
```python
test_data['X2'].unique()
```

Out[22]:
```
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 [23]:
```python
test_data['X3'].unique()
```

Out[23]:
```
array(['f', 'a', 'c', 'e', 'd', 'g', 'b'], dtype=object)
```

In [24]:
```python
test_data['X4'].unique()
```

Out[24]:
```
array(['d', 'b', 'a', 'c'], dtype=object)
```

In [25]:
```python
test_data['X5'].unique()
```

Out[25]:
```
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 [26]:
```python
test_data['X6'].unique()
```

Out[26]:
```
array(['a', 'g', 'j', 'l', 'i', 'd', 'f', 'h', 'c', 'k', 'e', 'b'],
      dtype=object)
```

In [27]:
```python
test_data['X8'].unique()
```

Out[27]:
```
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)
```

Apply Label encoder

In [28]:
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [29]:
```python
train_data['X0'] = le.fit_transform(train_data.X0)
train_data['X1'] = le.fit_transform(train_data.X1)
train_data['X2'] = le.fit_transform(train_data.X2)
train_data['X3'] = le.fit_transform(train_data.X3)
train_data['X4'] = le.fit_transform(train_data.X4)
train_data['X5'] = le.fit_transform(train_data.X5)
train_data['X6'] = le.fit_transform(train_data.X6)
train_data['X8'] = le.fit_transform(train_data.X8)
```

In [30]:
```python
features = train_data.drop(['y','ID'],axis=1)
target = train_data[['y']]

features.shape,target.shape
```

Out[30]:
```
((4209, 364), (4209, 1))
```

In [31]:
```python
from sklearn.model_selection import train_test_split
```

In [32]:
```python
X_train, X_test, y_train, y_test = train_test_split(features,target,train_size=0.8)
```

In [33]:
```python
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(3367, 364)
(842, 364)
(3367, 1)
(842, 1)
```

In [34]:
```python
train_data.head()
```

Out[34]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	32	23	17	0	3	24	9	14	...	0	0	1	0	0	0	0	0	0	0
1	6	88.53	32	21	19	4	3	28	11	14	...	1	0	0	0	0	0	0	0	0	0
2	7	76.26	20	24	34	2	3	27	9	23	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	20	21	34	5	3	27	11	4	...	0	0	0	0	0	0	0	0	0	0
4	13	78.02	20	23	34	5	3	12	3	13	...	0	0	0	0	0	0	0	0	0	0

5 rows × 366 columns

In [35]:
```python
train_data.dtypes
```

Out[35]:
```
ID        int64
y       float64
X0        int32
X1        int32
X2        int32
         ...
X380      int64
X382      int64
X383      int64
X384      int64
X385      int64
Length: 366, dtype: object
```

Perform Dimensionality reduction

In [36]:
```python
from sklearn.decomposition import PCA
pca = PCA(n_components=.9)
```

In [37]:
```python
pca.fit(X_train)
```

Out[37]:
```
PCA(n_components=0.9)
```

In [38]:
```python
pca.explained_variance_ratio_
```

Out[38]:
```
array([0.38049891, 0.21558917, 0.13119024, 0.11911883, 0.09364883])
```

In [39]:
```python
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [40]:
```python
X_train_pca.shape, X_test_pca.shape
```

Out[40]:
```
((3367, 5), (842, 5))
```

In [41]:
```python
X_train_pca
```

Out[41]:
```
array([[ -8.18887269, -13.84530082,  12.33491469,  -1.65654752,
          8.51630149],
       [  8.10739277,  -2.86338488, -14.23373304,  -5.04023912,
        -10.49350464],
       [-15.75807568,   1.24447381,   0.21806228,  -6.37402159,
        -11.81708529],
       ...,
       [ 12.33694493, -13.02816897,  -4.83217329,  14.28191569,
         11.40132304],
       [-16.122173  ,   7.15897352,  -1.06019843,  -5.78130533,
          7.73132272],
       [ -5.5375811 ,  -0.11408029,   5.11700565,   1.43491012,
          0.70881494]])
```

Predict your test_df values using XGBoost

In [42]:
```python
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 [43]:
```python
X_train,X_test,Y_train,Y_test = train_test_split(features,target,test_size=.2)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(3367, 364)
(3367, 1)
(842, 364)
(842, 1)
```

In [44]:
```python
xgb_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3, learning_rate = 0.4, max_depth = 10, alpha = 6,
                          n_estimators = 20)
model = xgb_reg.fit(X_train,y_train)
print('RMSE = ',sqrt(mean_squared_error(model.predict(X_test),y_test)))
```

```
[11:24:34] WARNING: c:\ci\xgboost-split_1619728435298\work\src\objective\regression_obj.cu:170: reg:linear is now deprecated in favor of reg:squarederror.
RMSE =  13.701496319881393
```

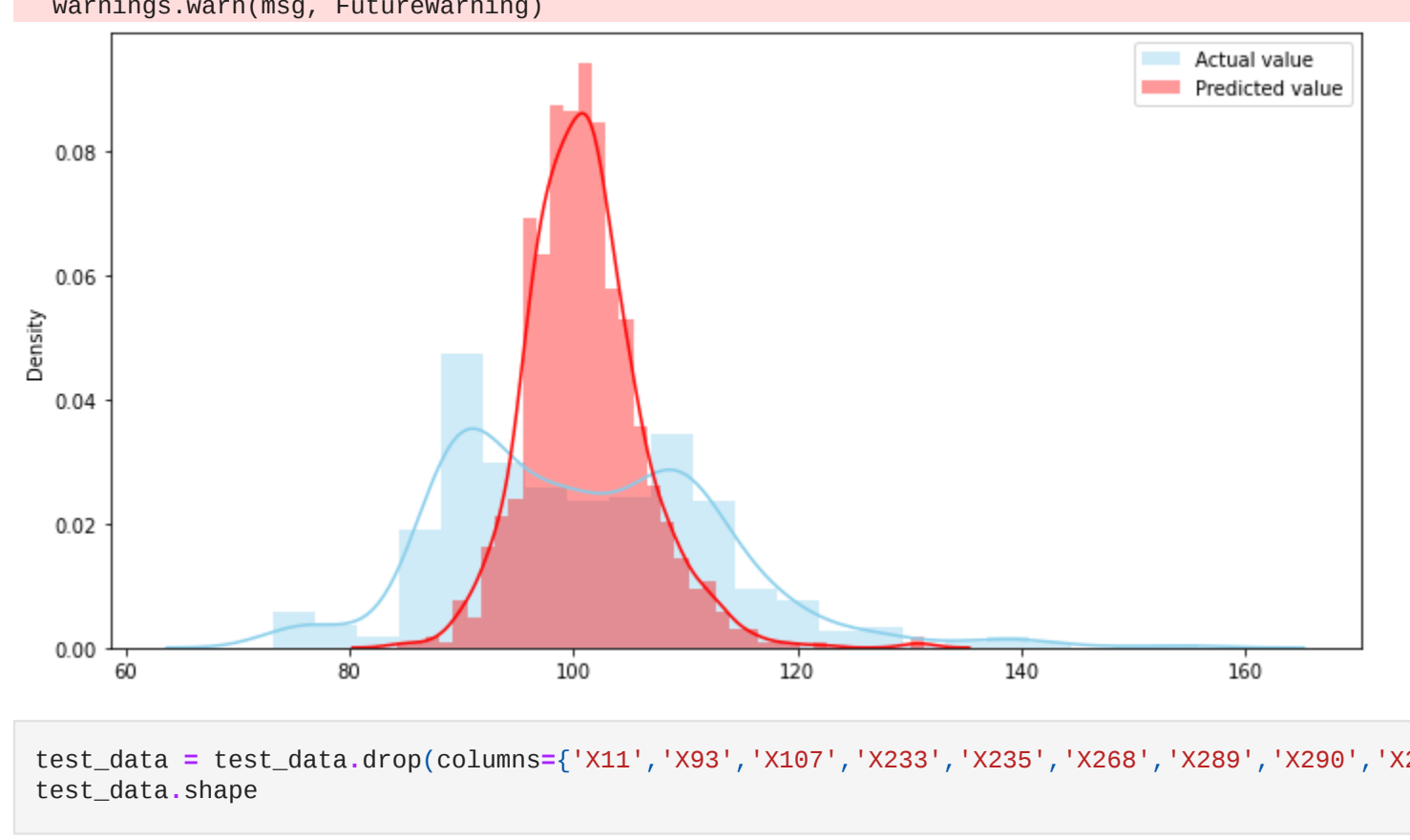
In [45]:
```python
pred_y_test= model.predict(X_test)

plt.figure(figsize=(10,5))

sns.distplot(y_test[y_test<160], color="skyblue", label="Actual value")
sns.distplot(pred_y_test[pred_y_test<160] , color="red", label="Predicted value")
plt.legend()

plt.tight_layout()
```

```
C:\Users\abha mohan\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please ad
apt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\abha mohan\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please ad
apt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [46]:
```python
test_data = test_data.drop(columns={'X11','X93','X107','X233','X235','X268','X289','X290','X293','X297','X330','X347'})
test_data.shape
```

Out[46]:
```
(4209, 365)
```

In [47]:
```python
test_data.describe()
```

Out[47]:

	ID	X10	X12	X13	X14	X15	X16	X17	X18	X19	...	X375	X376	X377	X378	X379	X380	
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	...	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	42
mean	4211.039202	0.019007	0.074364	0.061060	0.427893	0.000713	0.002613	0.008791	0.010216	0.111665	...	0.325968	0.049656	0.311951	0.019244	0.011879	0.008078	
std	2423.078926	0.136565	0.262394	0.239468	0.494832	0.026691	0.051061	0.093357	0.100570	0.314992	...	0.468791	0.217258	0.463345	0.137399	0.108356	0.089524	
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2115.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	4202.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	6310.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	
max	8416.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 357 columns

In [48]:
```python
test_data['X0'] = le.fit_transform(test_data.X0)
test_data['X1'] = le.fit_transform(test_data.X1)
test_data['X2'] = le.fit_transform(test_data.X2)
test_data['X3'] = le.fit_transform(test_data.X3)
test_data['X4'] = le.fit_transform(test_data.X4)
test_data['X5'] = le.fit_transform(test_data.X5)
test_data['X6'] = le.fit_transform(test_data.X6)
test_data['X8'] = le.fit_transform(test_data.X8)
```

In [49]:
```python
pca.fit(X_test)
```

Out[49]:
```
PCA(n_components=0.9)
```

In [50]:
```python
pca.explained_variance_ratio_
```

Out[50]:
```
array([0.38860808, 0.21291585, 0.12624563, 0.12215104, 0.09295198])
```

In [51]:
```python
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [52]:
```python
X_train_pca.shape, X_test_pca.shape
```

Out[52]:
```
((3367, 5), (842, 5))
```

In [53]:
```python
X_test_pca
```

Out[53]:
```
array([[  6.65929348,  -3.32435835,  11.69821332, -14.31424233,
         10.57710837],
       [ 22.26179173,  -7.00489447,  -8.88034695,   4.20110289,
          0.61073553],
       [ 20.16244474,  -6.50240615,   3.97729938,  -4.25994744,
          8.92187057],
       ...,
       [ 23.45244295,  -6.38855011,   9.95279477,   3.65382259,
          6.60490032],
       [-12.25690948,  18.73442844,  -2.45916615,  -5.40963712,
          1.49221072],
       [ -4.66580345, -10.04106416, -12.43678641,  -9.81862946,
          5.66339449]])
```

In [ ]: