KMEANS-CRICKET_DATA

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [42]:
```
df=pd.read_csv("total_data_na.csv")
```

In [43]:
```
df.head()
```

Out[43]:

	PLAYER	Mat.x	Inns.x	NO	Runs.x	HS	Avg.x	BF	SR.x	X100	...	Ov	Runs.y	Wkts	BBI
0	Aaron Finch	10	9	1	134	46	16.75	100	134.00	0	...	0.0	0	0	0
1	AB de Villiers	12	11	2	480	90	53.33	275	174.54	0	...	0.0	0	0	0
2	Abhishek Sharma	3	3	2	63	46	63	33	190.90	0	...	0.0	0	0	0
3	Ajinkya Rahane	15	14	1	370	65	28.46	313	118.21	0	...	0.0	0	0	0
4	Alex Hales	6	6	0	148	45	24.66	118	125.42	0	...	0.0	0	0	0

5 rows × 25 columns

In [44]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 143 entries, 0 to 142
Data columns (total 25 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   PLAYER  143 non-null    object 
 1   Mat.x   143 non-null    int64  
 2   Inns.x  143 non-null    int64  
 3   NO      143 non-null    int64  
 4   Runs.x  143 non-null    int64  
 5   HS      143 non-null    int64  
 6   Avg.x   143 non-null    object 
 7   BF      143 non-null    int64  
 8   SR.x    143 non-null    float64
 9   X100    143 non-null    int64  
 10  X50     143 non-null    int64  
 11  X4s     143 non-null    int64  
 12  X6s     143 non-null    int64  
 13  Mat.y   143 non-null    int64  
 14  Inns.y  143 non-null    int64  
 15  Ov      143 non-null    float64
 16  Runs.y  143 non-null    int64  
 17  Wkts    143 non-null    int64  
 18  BBI     143 non-null    int64  
 19  Avg.y   143 non-null    object 
 20  Econ    143 non-null    float64
 21  SR.y    143 non-null    object 
 22  X4w     143 non-null    int64  
 23  X5w     143 non-null    int64  
 24  y       143 non-null    int64  
dtypes: float64(3), int64(18), object(4)
memory usage: 28.1+ KB
```

In [45]:
```
df.describe()
```

Out[45]:

	Mat.x	Inns.x	NO	Runs.x	HS	BF	SR.x	X10
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.00000
mean	7.286713	6.013986	1.251748	132.349650	33.153846	95.027972	93.120280	0.03496
std	6.077692	5.499022	1.629259	175.482243	31.969684	120.286919	67.202818	0.21923
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
50%	7.000000	5.000000	1.000000	52.000000	27.000000	41.000000	117.020000	0.00000
75%	13.000000	11.000000	2.000000	202.000000	53.500000	152.500000	140.595000	0.00000
max	17.000000	17.000000	9.000000	735.000000	128.000000	516.000000	300.000000	2.00000

8 rows × 21 columns

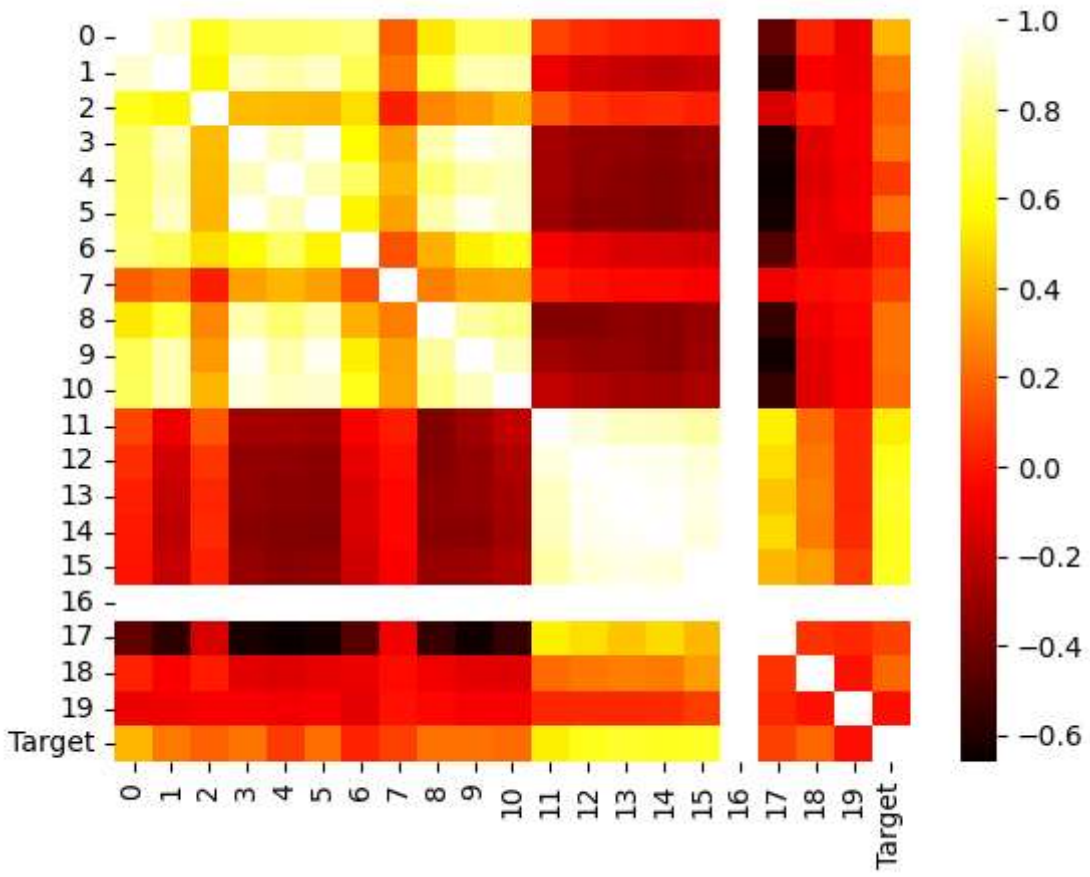
In [46]:
```
df.isna().sum()
```

Out[46]:
```
PLAYER    0
Mat.x     0
Inns.x    0
NO        0
Runs.x    0
HS        0
Avg.x     0
BF        0
SR.x      0
X100      0
X50       0
X4s       0
X6s       0
Mat.y     0
Inns.y    0
Ov        0
Runs.y    0
Wkts      0
BBI       0
Avg.y     0
Econ      0
SR.y      0
X4w       0
X5w       0
y         0
dtype: int64
```

CORELATION

In [86]:
```
cor=df.corr()
sns.heatmap(cor,cmap="hot",)
```

Out[86]:
```
<AxesSubplot:>
```



DROP UNWANTED COLUMNS

In [47]:
```
df.drop(["PLAYER","y"],axis=1,inplace=True)
```

In [48]:
```
cat=df.select_dtypes("object").columns
```

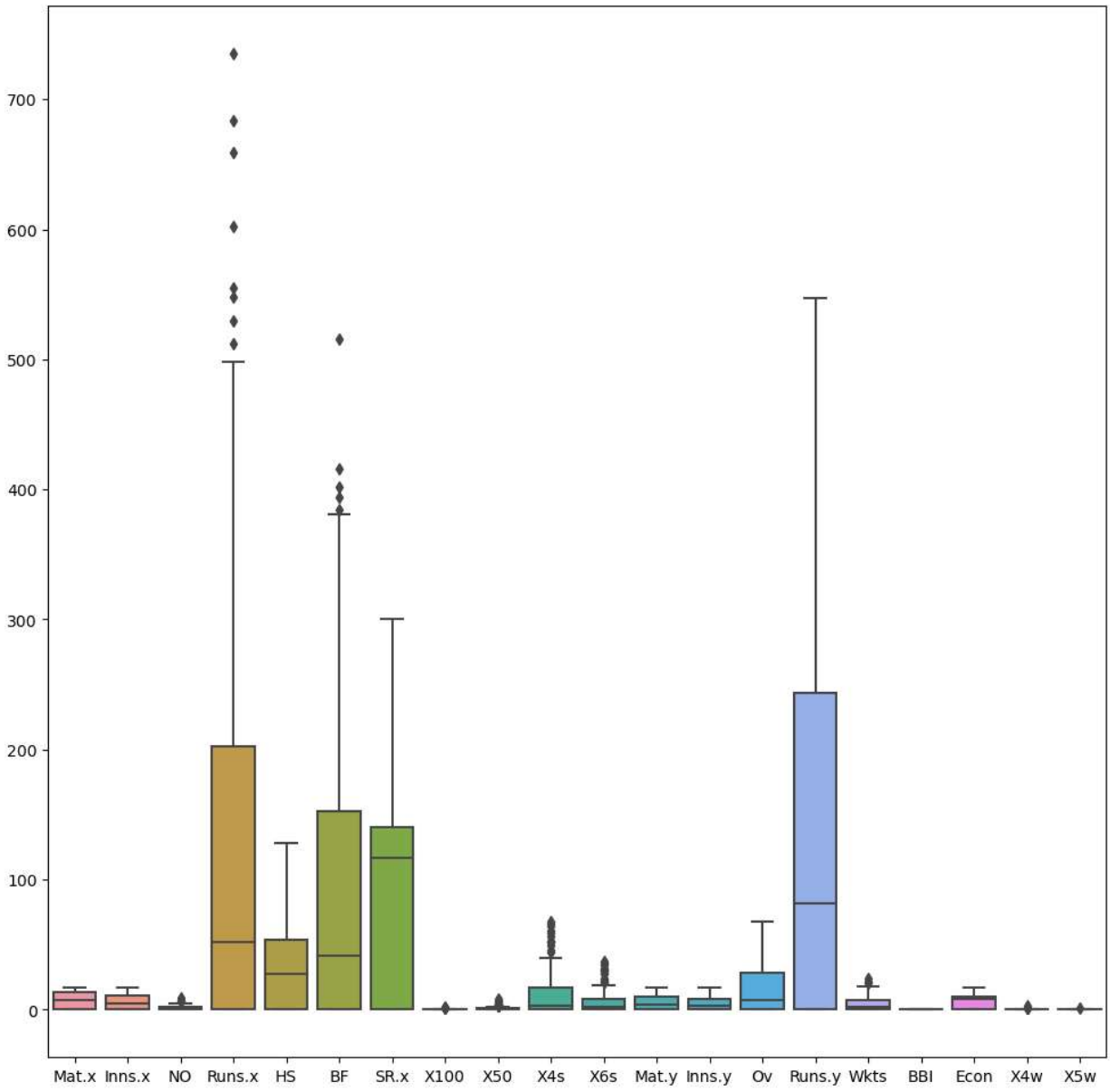
In [49]:
```
df.drop(cat,axis=1,inplace=True)
```

In [50]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 143 entries, 0 to 142
Data columns (total 20 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Mat.x   143 non-null    int64  
 1   Inns.x  143 non-null    int64  
 2   NO      143 non-null    int64  
 3   Runs.x  143 non-null    int64  
 4   HS      143 non-null    int64  
 5   BF      143 non-null    int64  
 6   SR.x    143 non-null    float64
 7   X100    143 non-null    int64  
 8   X50     143 non-null    int64  
 9   X4s     143 non-null    int64  
 10  X6s     143 non-null    int64  
 11  Mat.y   143 non-null    int64  
 12  Inns.y  143 non-null    int64  
 13  Ov      143 non-null    float64
 14  Runs.y  143 non-null    int64  
 15  Wkts    143 non-null    int64  
 16  BBI     143 non-null    int64  
 17  Econ    143 non-null    float64
 18  X4w     143 non-null    int64  
 19  X5w     143 non-null    int64  
dtypes: float64(3), int64(17)
memory usage: 22.5 KB
```

DETECT OUTLIERS

In [51]:
```
plt.figure(figsize=(12,12))
sns.boxplot(data=df)
plt.show()
```



OUTLIERS HANDALING

In [52]:
```
def whiskers(col):
    q1=np.quantile(col,0.25)
    q3=np.quantile(col,0.75)
    iqr=q3-q1
    uw=q3+1.5*iqr
    lw=q1-1.5*iqr
    return uw,lw
```

In [53]:
```
print(whiskers(df["Runs.x"]))
run_out=df[df["Runs.x"]>505.0].index
df.loc[run_out,"Runs.x"]=505.0
```

```
(505.0, -303.0)
```

In [54]:
```
whiskers(df["BF"])
```

Out[54]:
```
(381.25, -228.75)
```

In [55]:
```
bf_out=df[df["BF"]>381.25].index
df.loc[bf_out,"BF"]=381
```

In [56]:
```
whiskers(df["X4s"])
```

Out[56]:
```
(42.5, -25.5)
```

In [57]:
```
x4s_out=df[df["X4s"]>42.5].index
df.loc[x4s_out,"X4s"]=42.5
```

In [58]:
```
whiskers(df["X6s"])
```

Out[58]:
```
(20.0, -12.0)
```

In [59]:
```
x6s_out=df[df["X6s"]>20.0].index
df.loc[x6s_out,"X6s"]=20.0
```

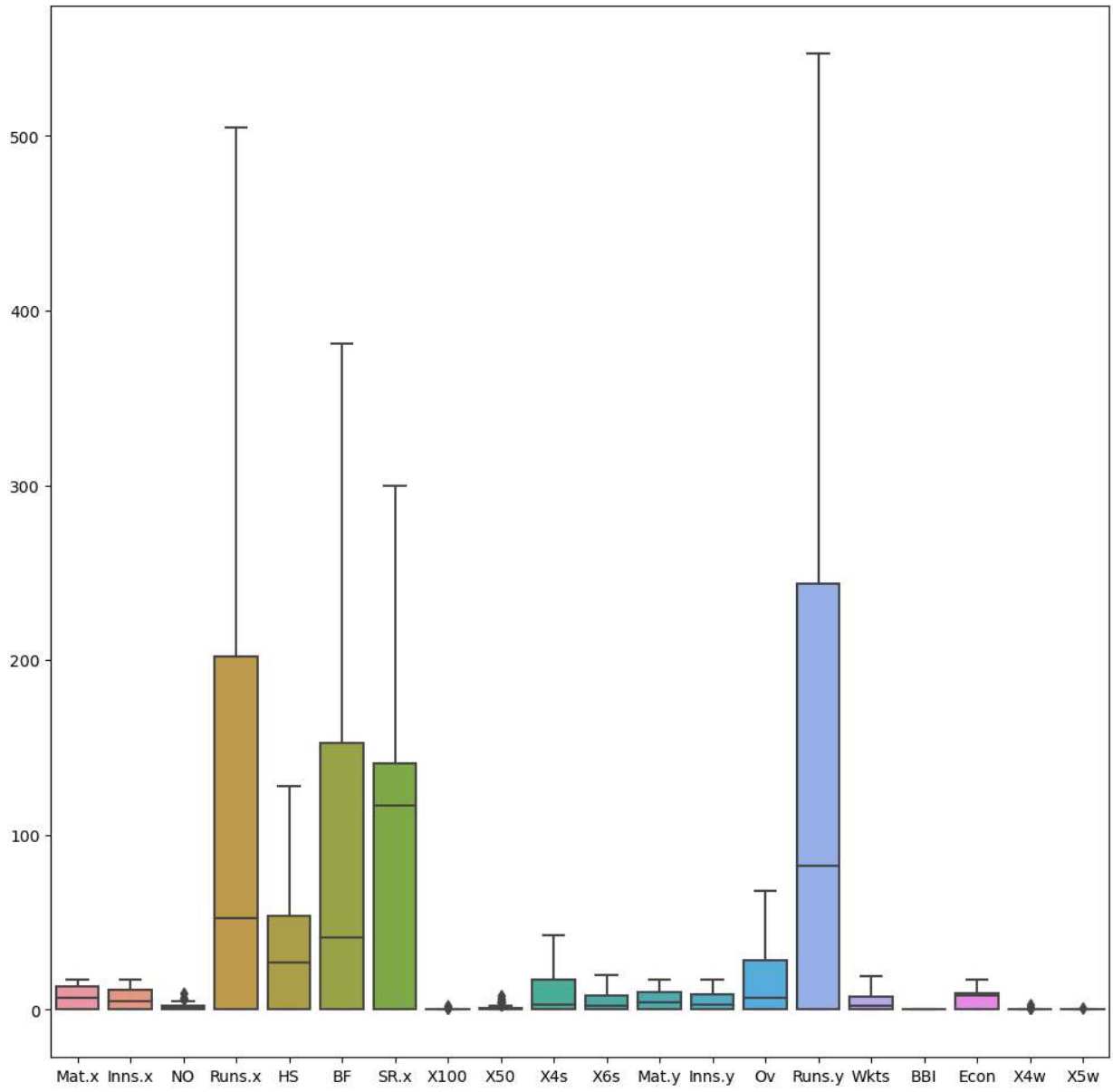
In [60]:
```
whiskers(df["Wkts"])
```

Out[60]:
```
(18.75, -11.25)
```

In [61]:
```
wkts_out=df[df["Wkts"]>18.75].index
df.loc[wkts_out,"Wkts"]=18.75
```

OUTLIERS REMOVED

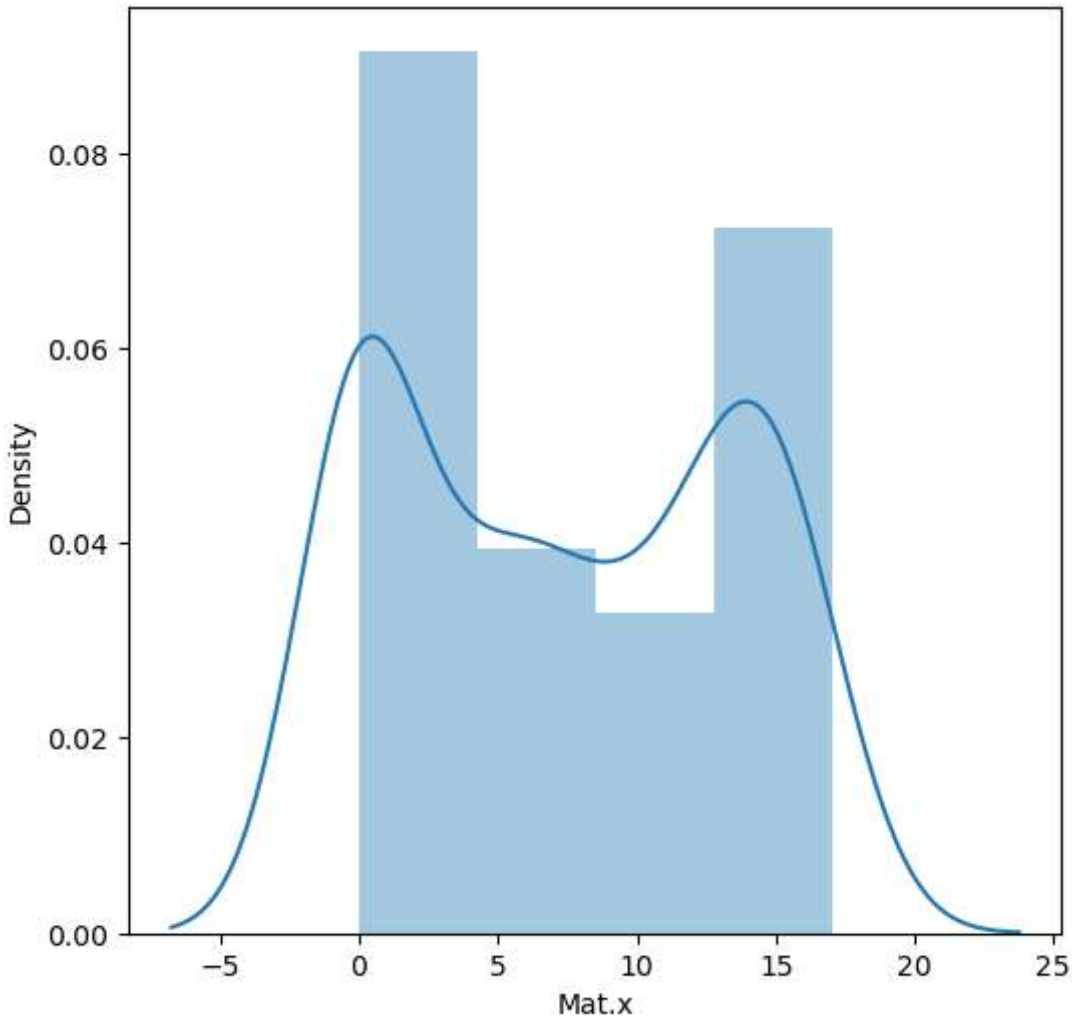
In [62]:
```
plt.figure(figsize=(12,12))
sns.boxplot(data=df)
plt.show()
```



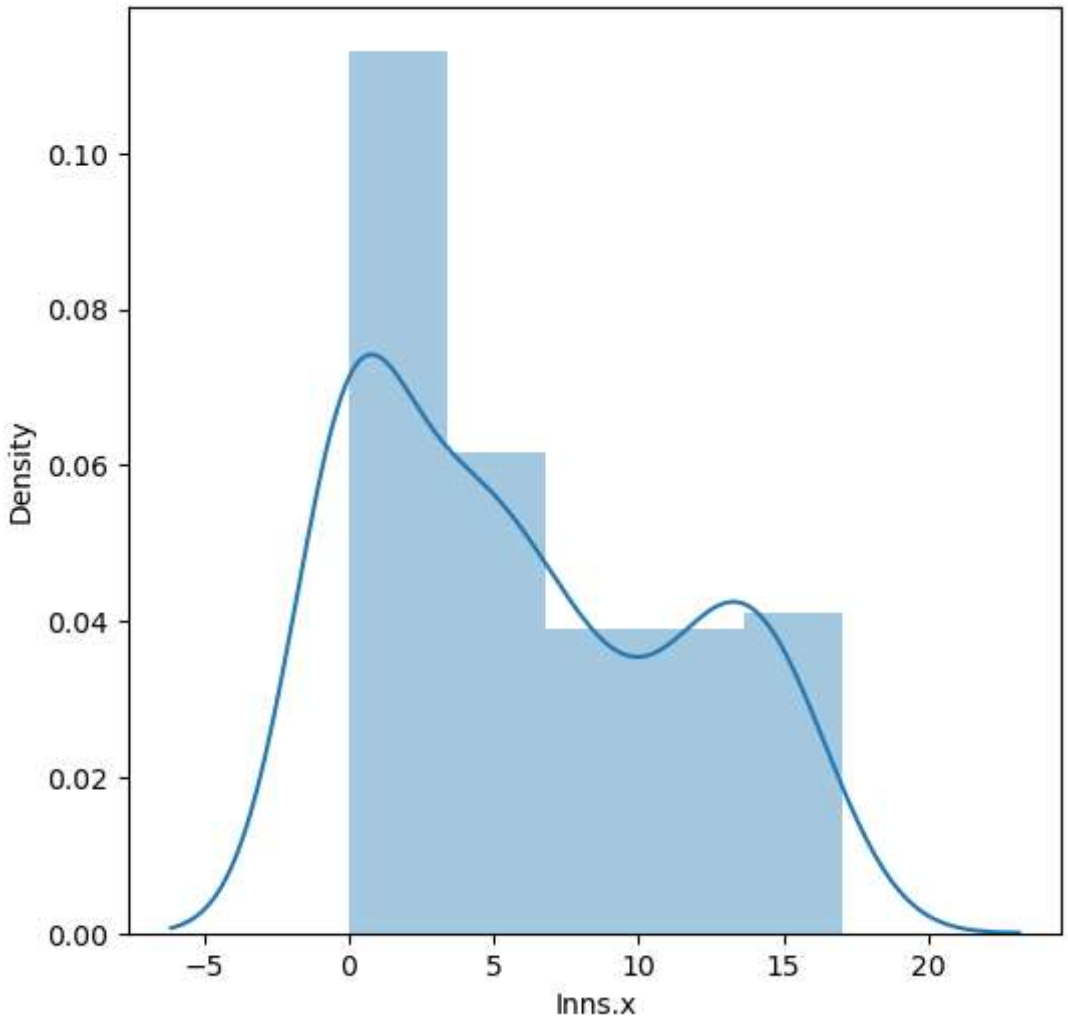
In [ ]:

In [67]:
```
col=df.columns
from scipy.stats import skew
for i in col:
    print(i)
    print(skew(df[i]))
    plt.figure(figsize=(6,6))
    sns.distplot(df[i])
    plt.show()
```

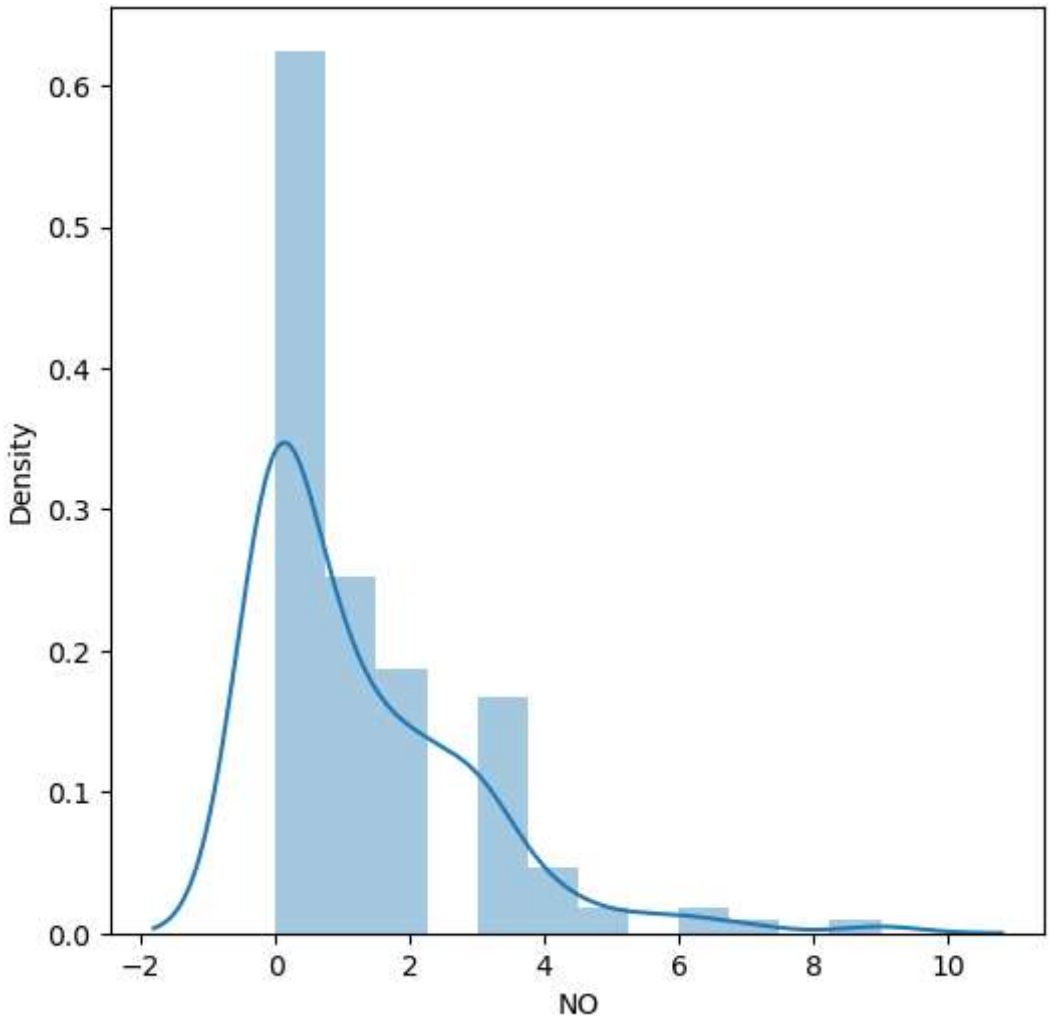
```
Mat.x
0.07580070648211645
```



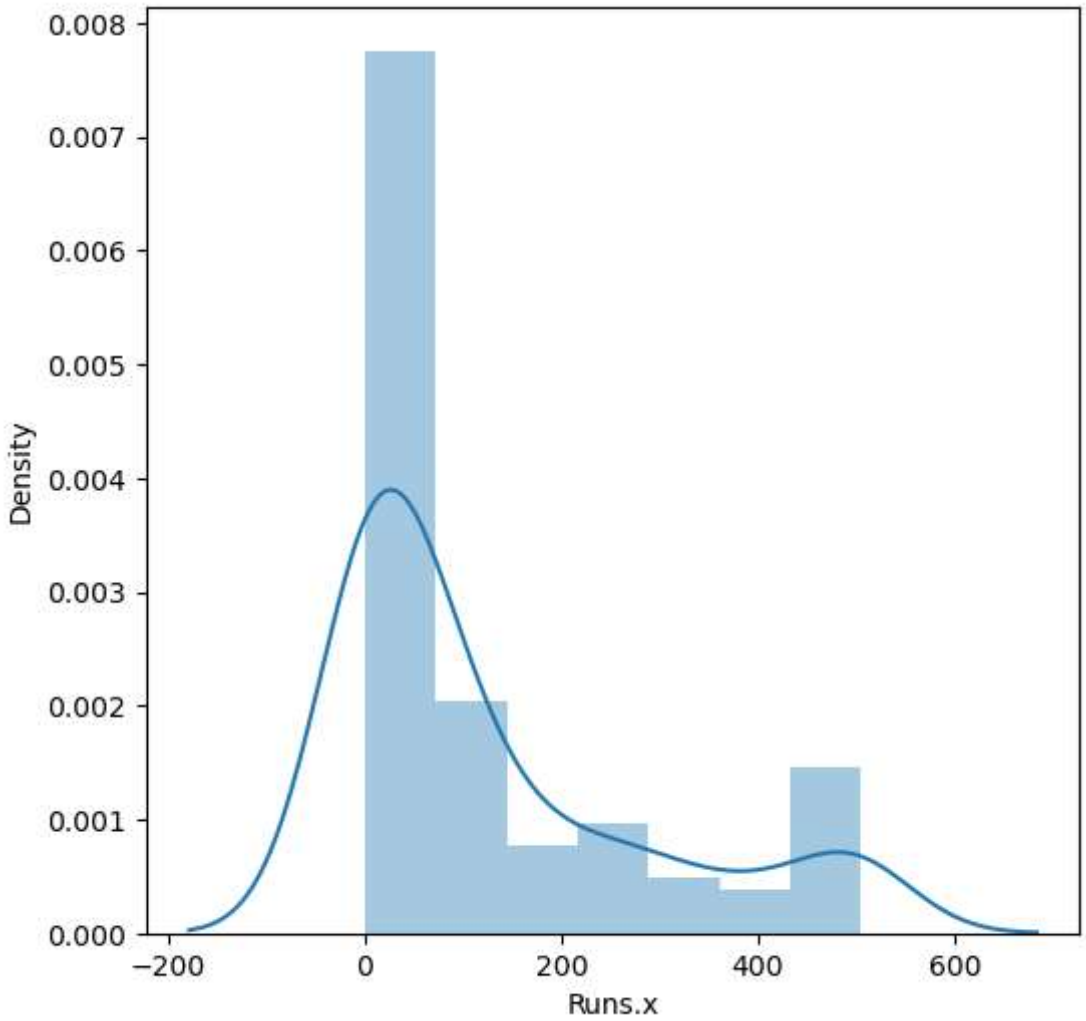

```
Inns.x
0.42770332009084794
```



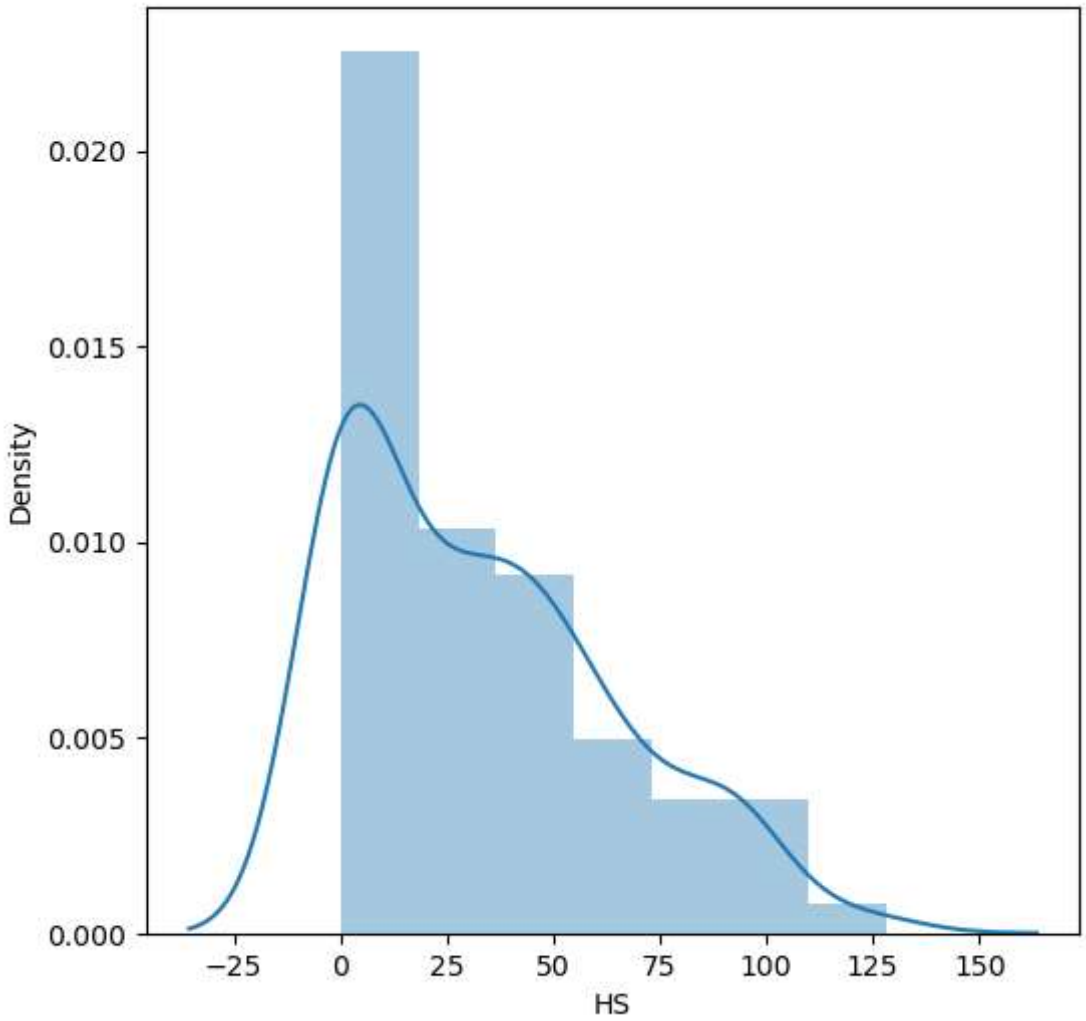
NO
1.717447440547279



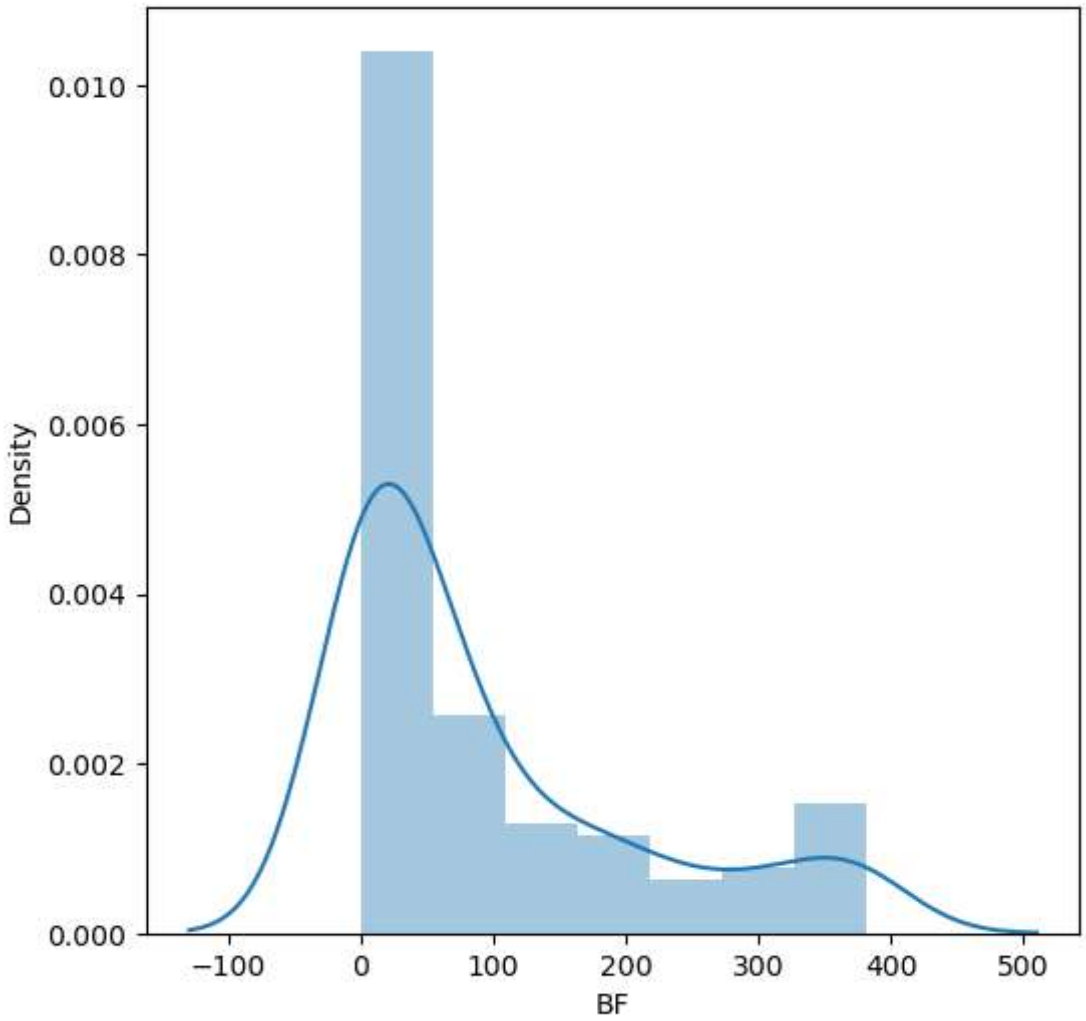

```
Runs.x
1.2658218844431892
```



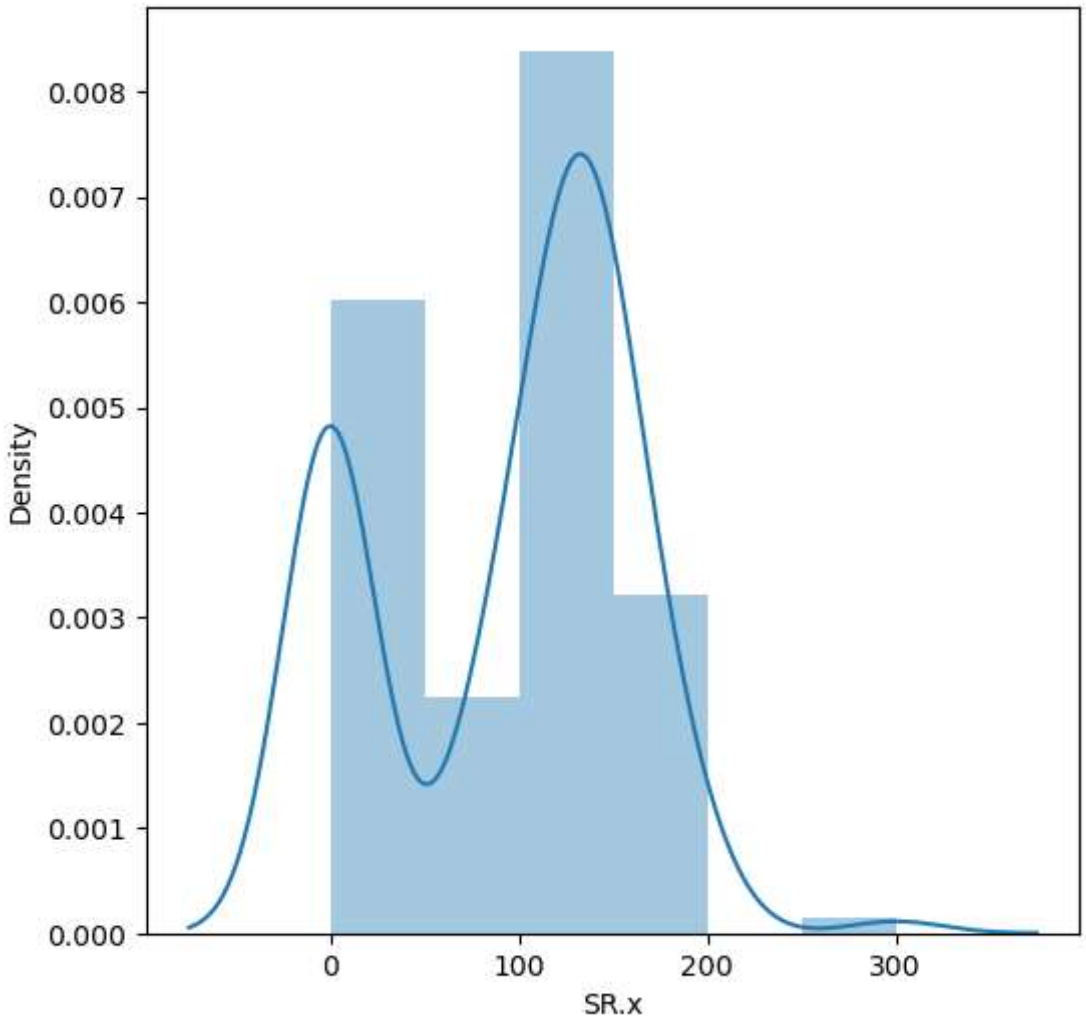
HS
0.7391454774064703



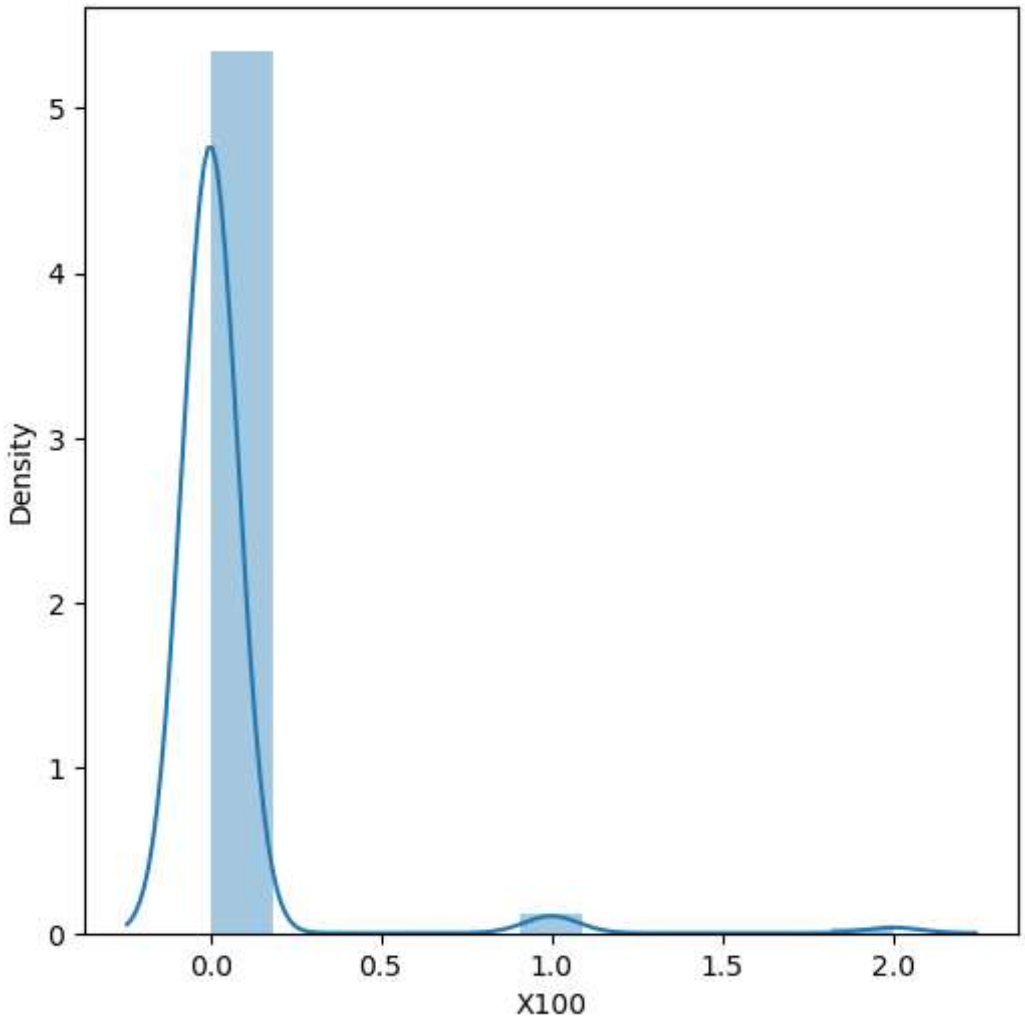
BF
1.2845050117588985



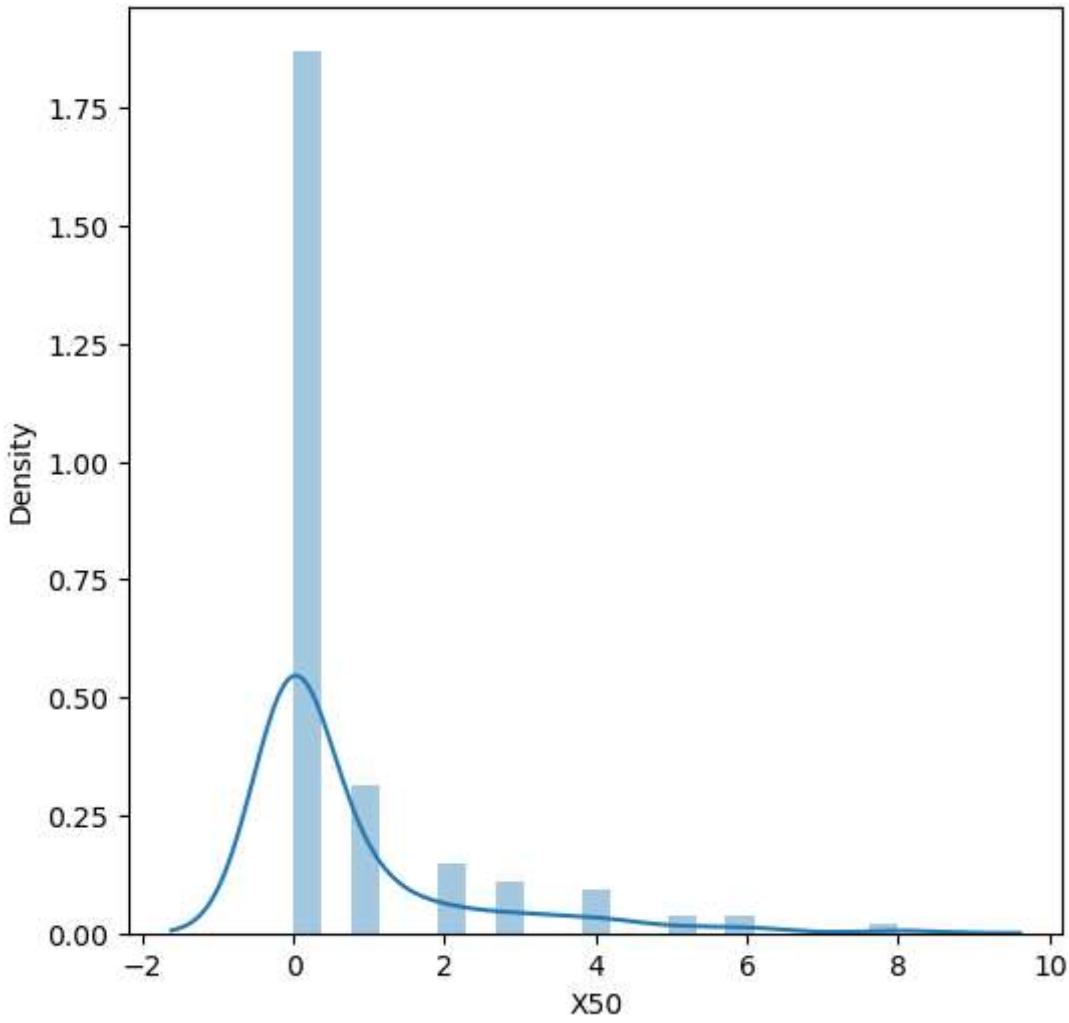

```
SR.x
-0.25773348793914563
```



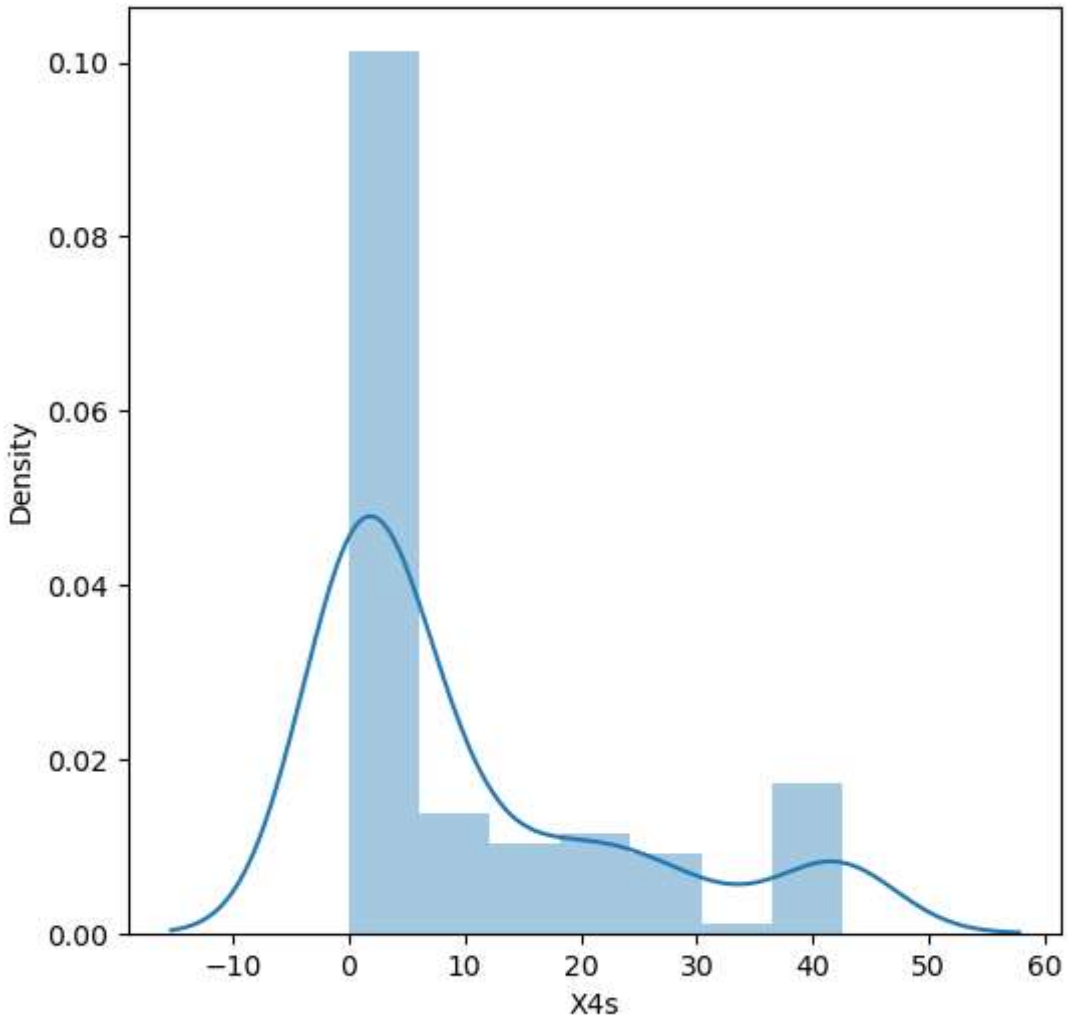

```
X100
6.892936271264464
```



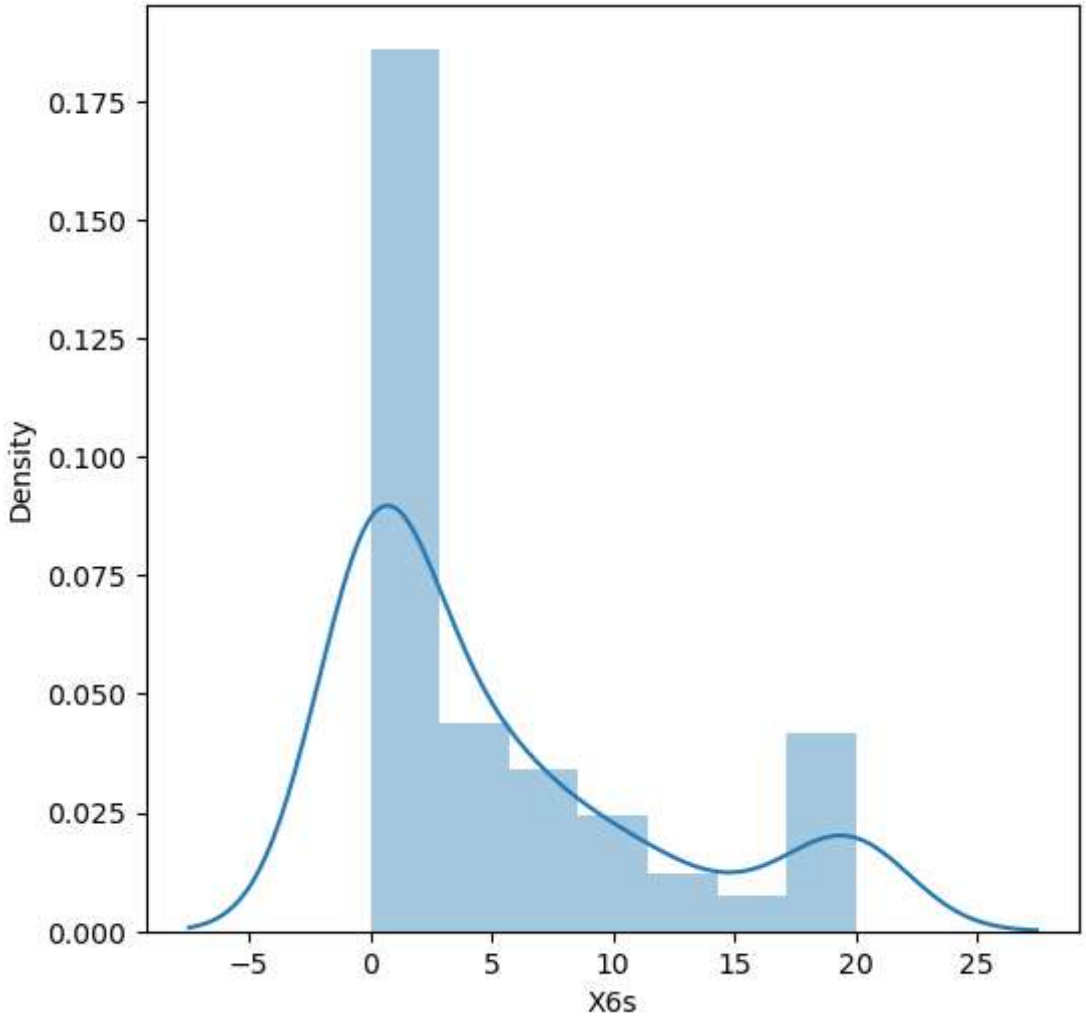

```
X50
2.5052439125468493
```



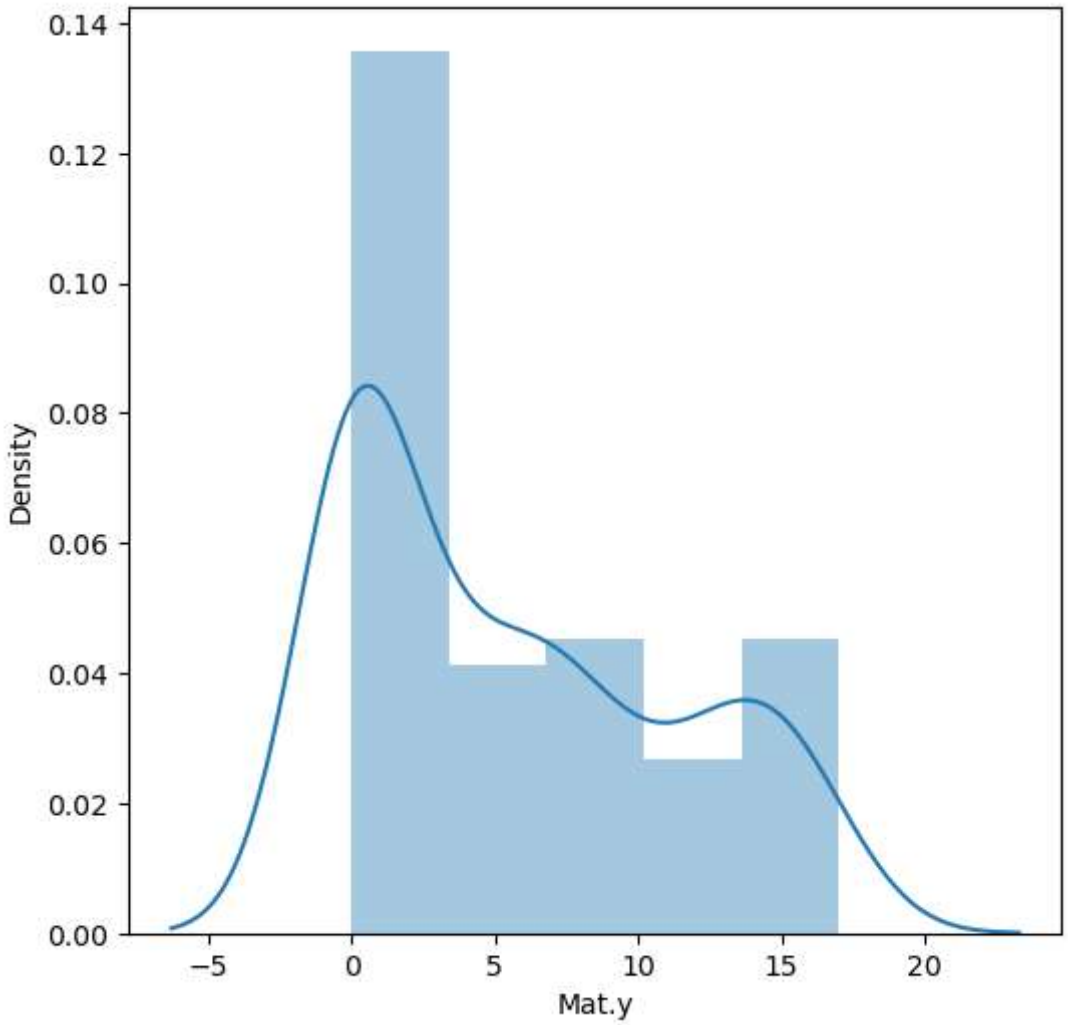
X4s
1.302338071917717



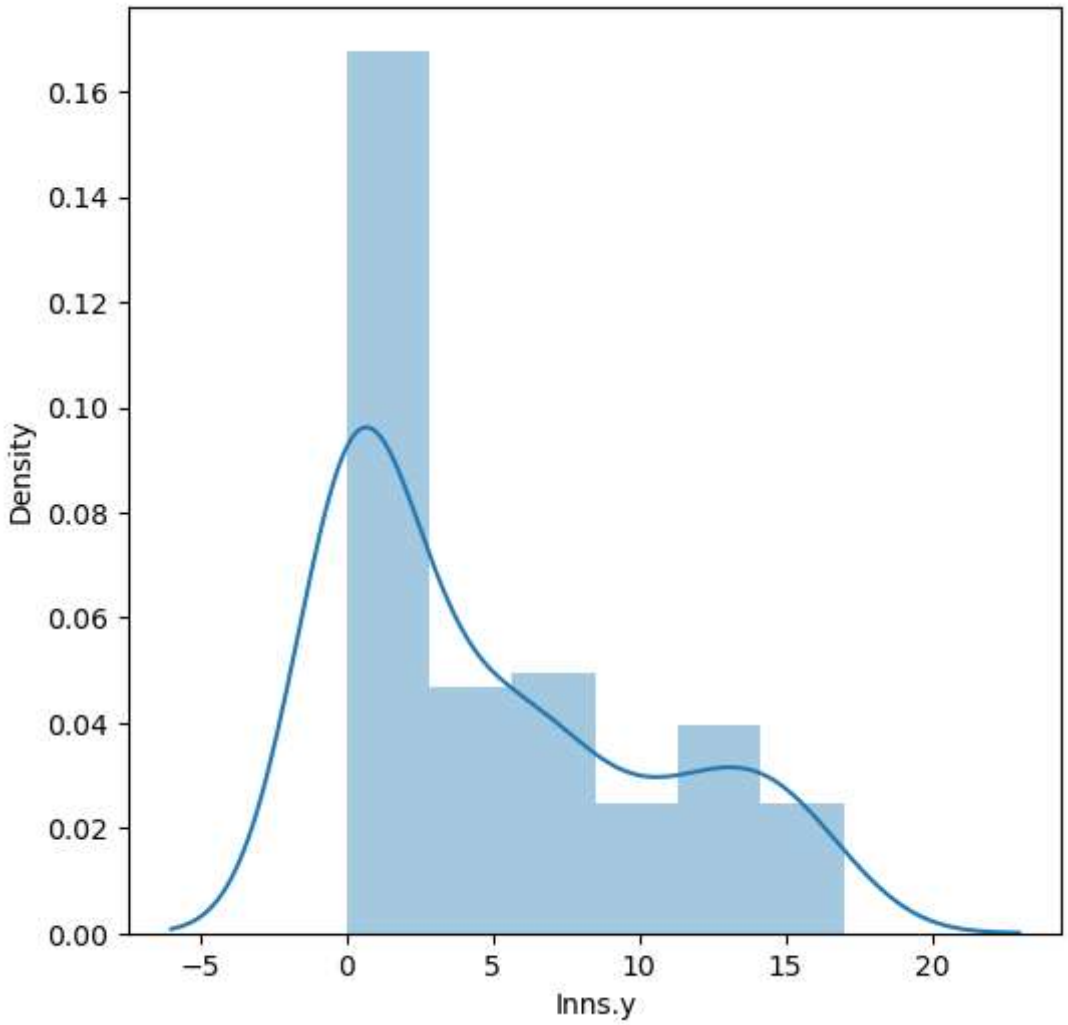
X6s
1.1649074729912177



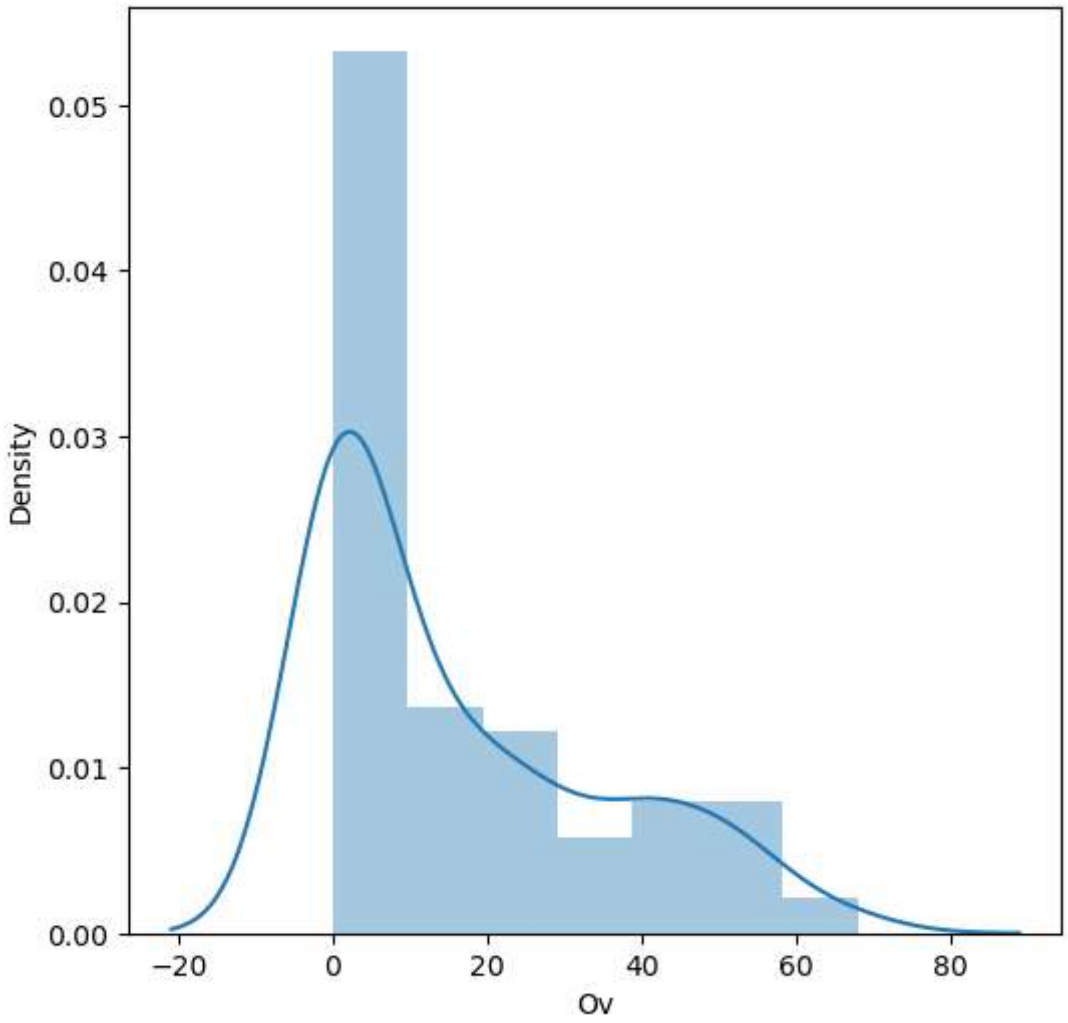

```
Mat.y
0.5657374572184163
```



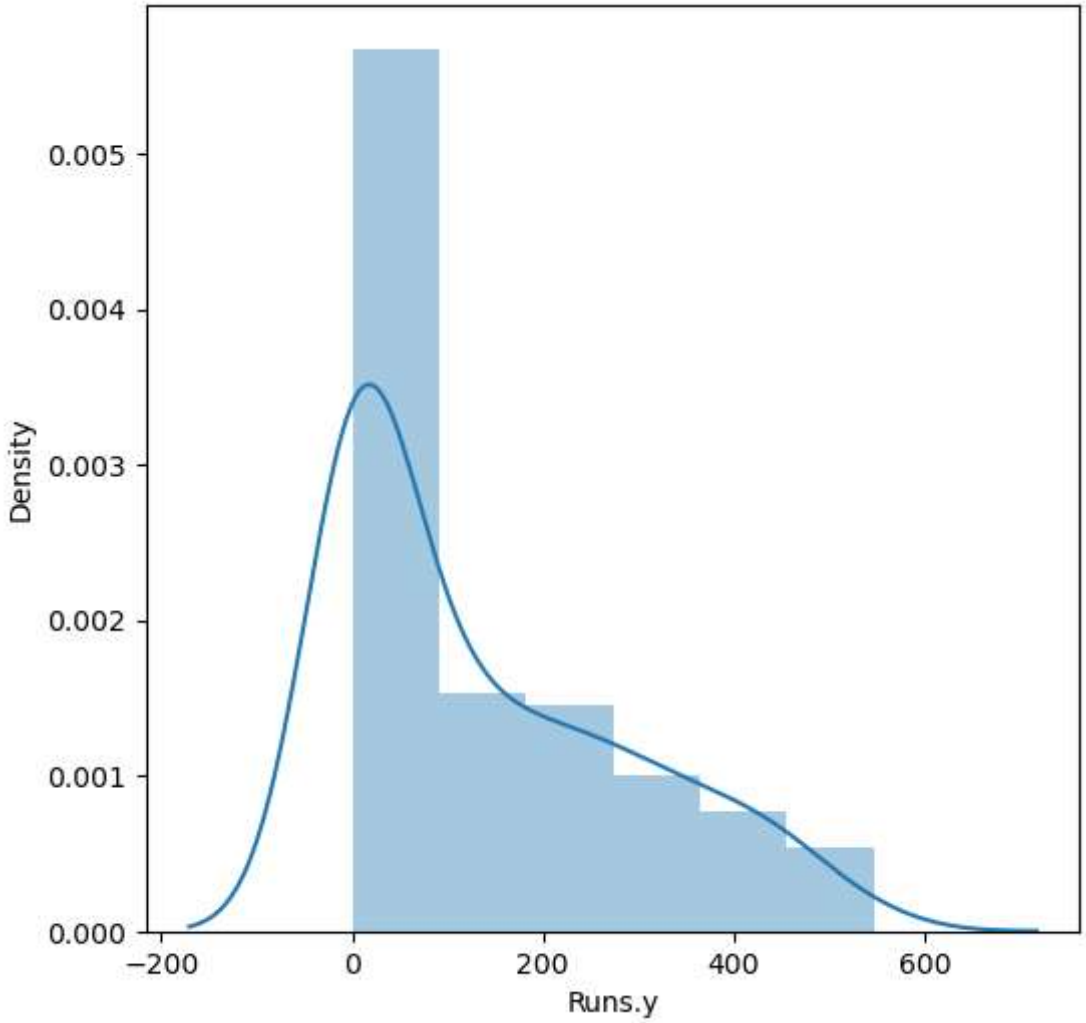

```
Inns.y
0.7642229601139375
```



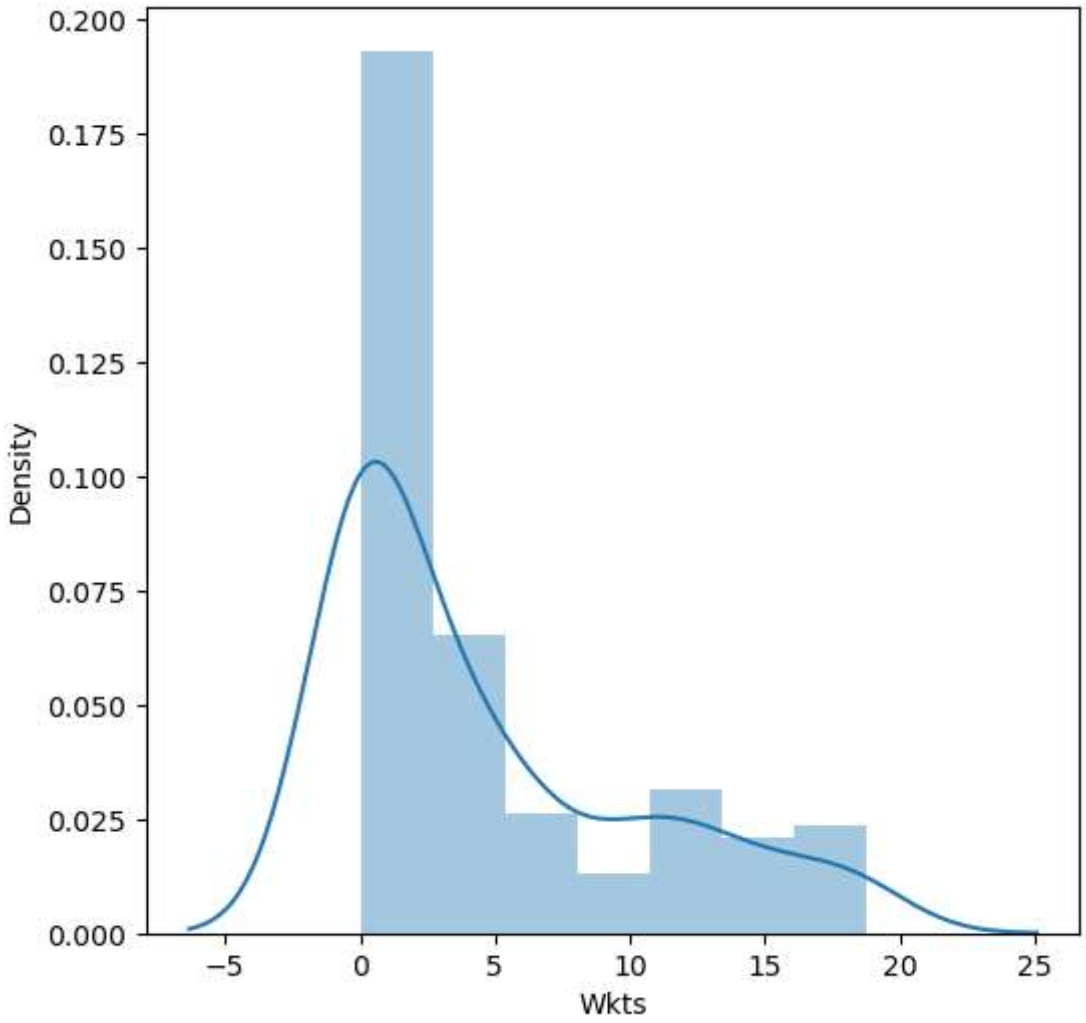
Ov
0.9778656243554076



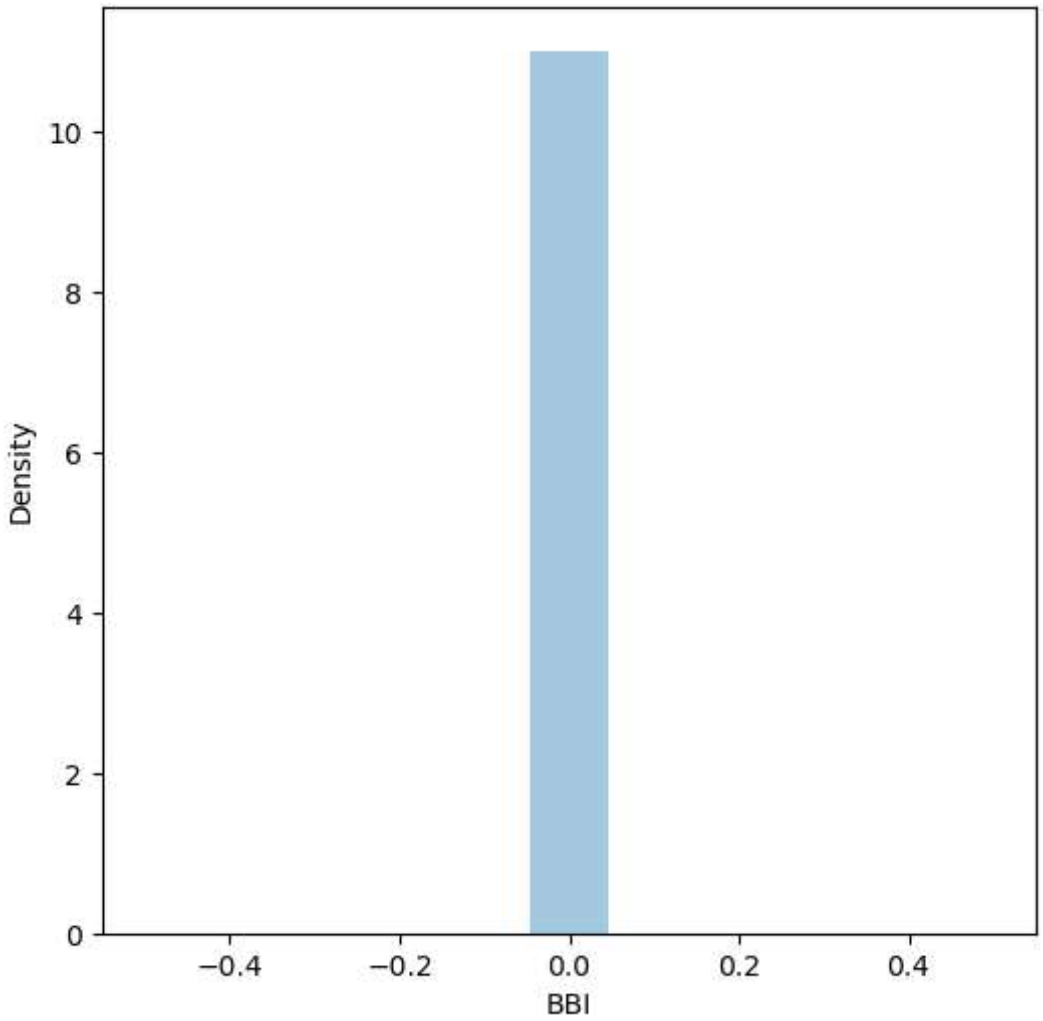

```
Runs.y
0.8851968871419482
```



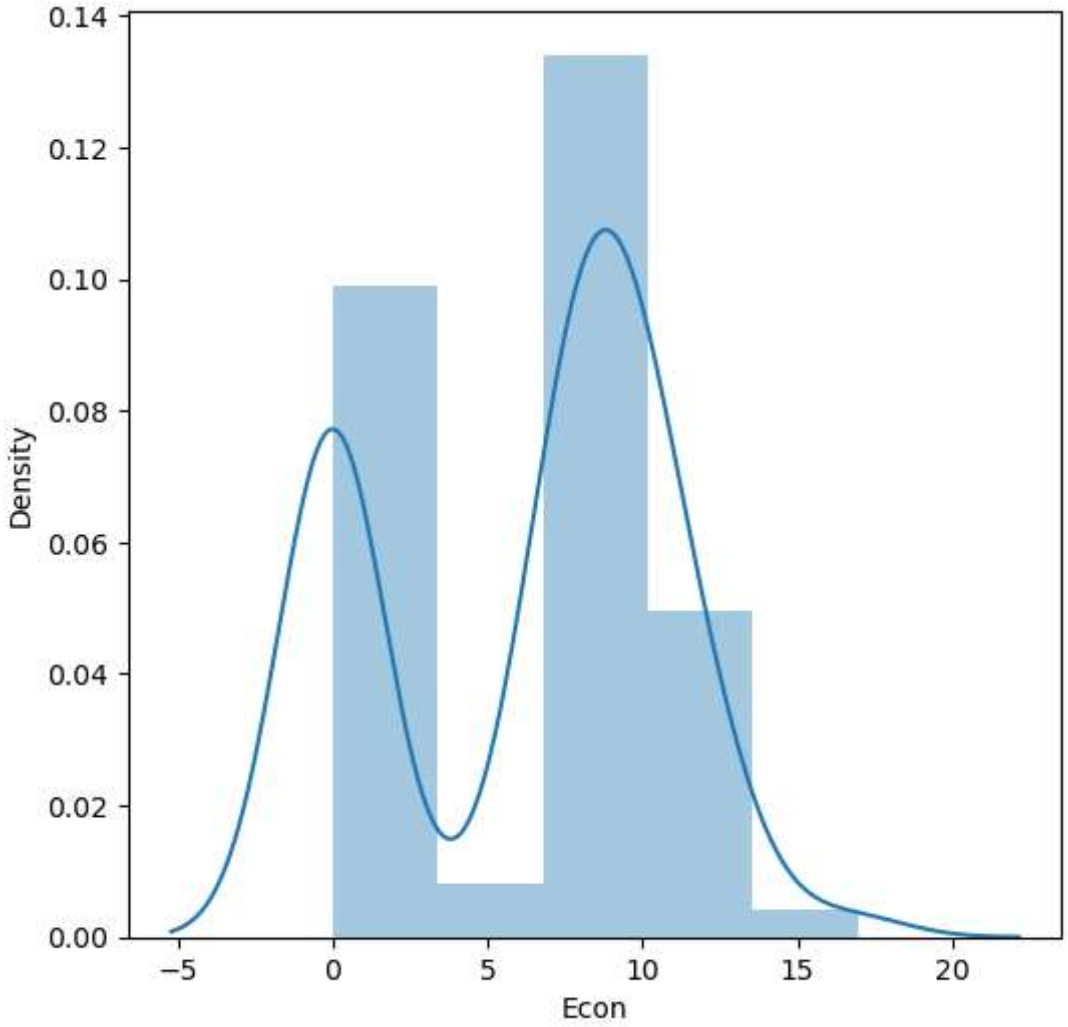

```
Wkts
1.1032415779836604
```


0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Density
−5
0
5
10
15
20
25
Wkts

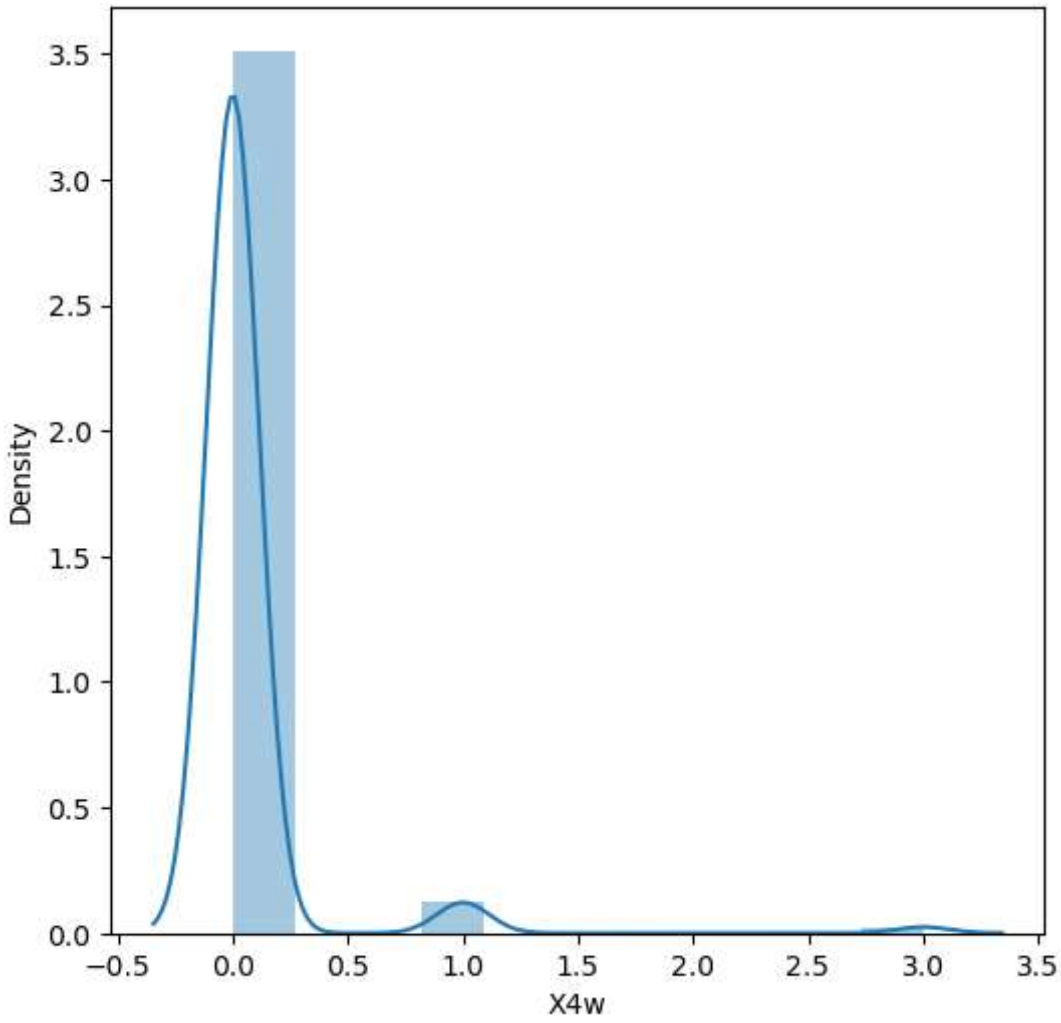
```
BBI
nan
```



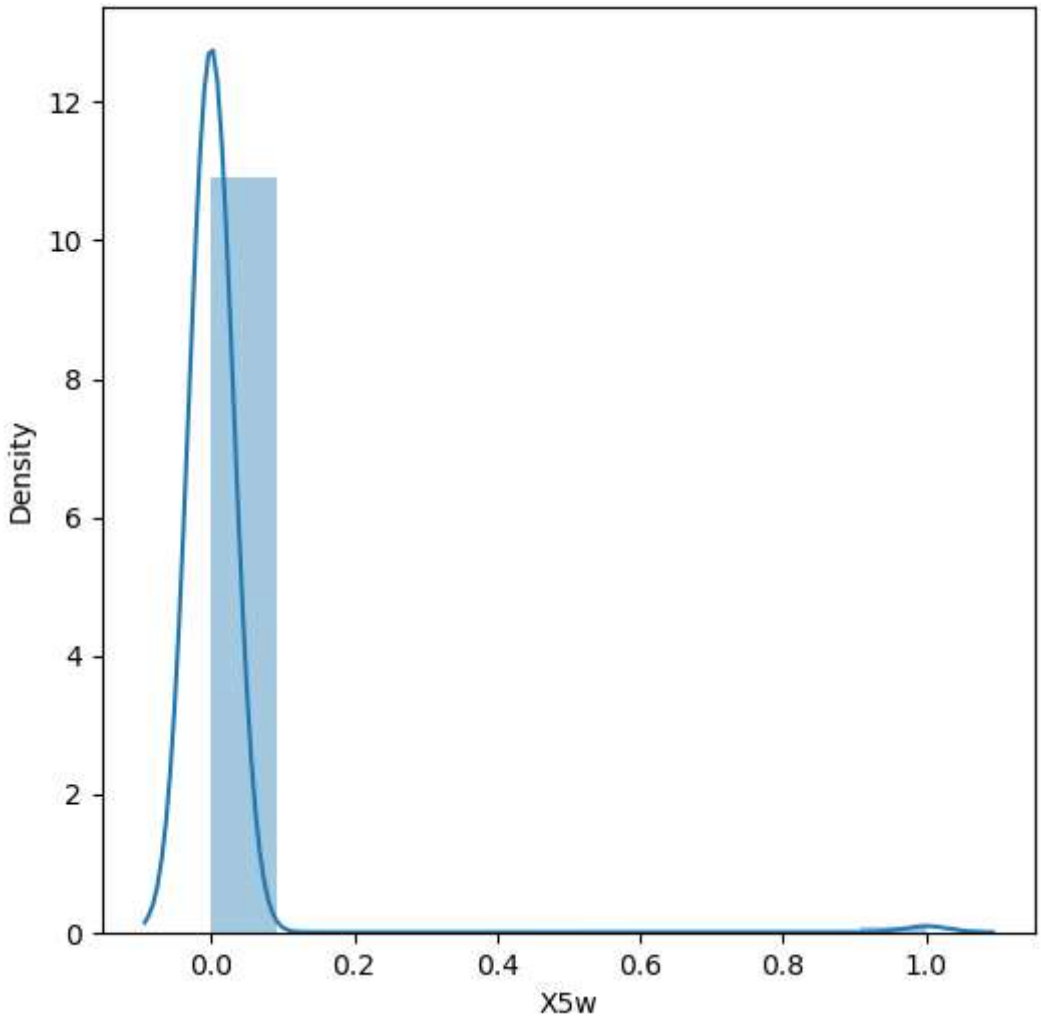
Econ
-0.27618384376075306



X4w
7.118755653768533



X5w
11.83245715198333



SCALING

In [68]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
df=ss.fit_transform(df)
```

CLUSTERING

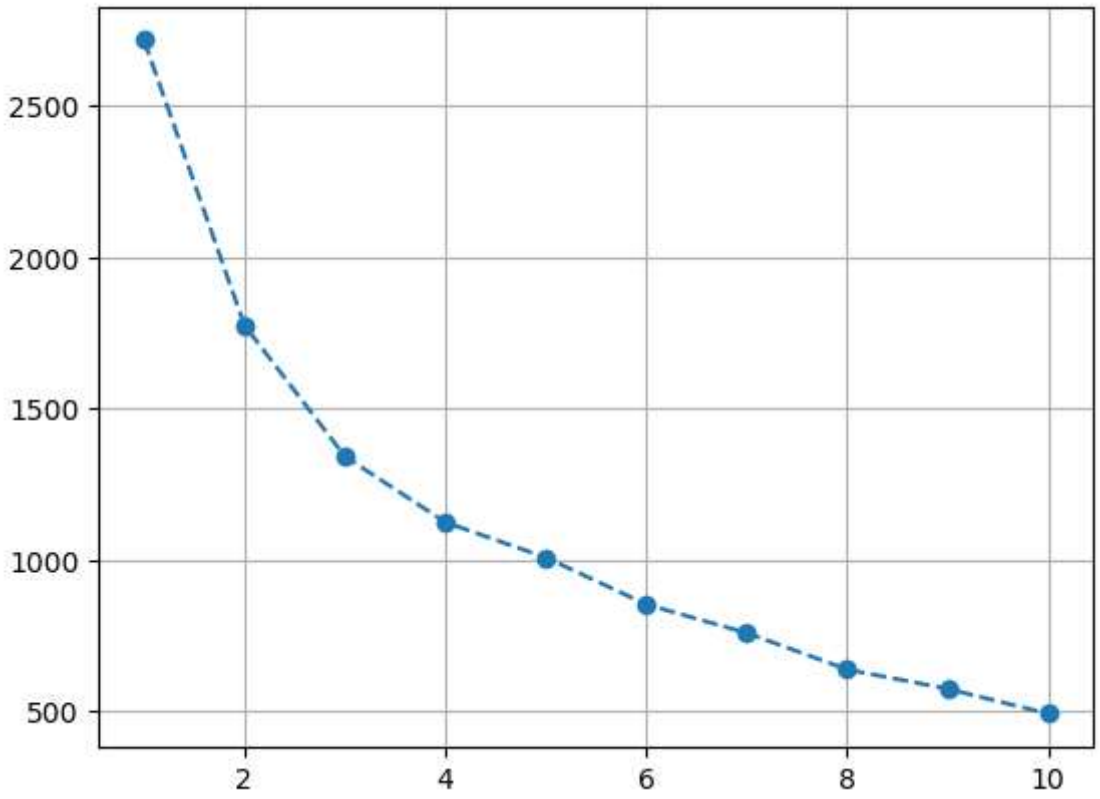
In [69]:
```
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,11):
    kmeans=KMeans(n_clusters=i,random_state=1)
    kmeans.fit(df)
    wcss.append(kmeans.inertia_)
```

In [70]:
```
wcss
```

Out[70]:
```
[2717.0,
 1775.5324792773022,
 1342.2530769196699,
 1124.9736882820698,
 1008.2678164565501,
 853.4369058543106,
 759.1478620641292,
 638.518930591215,
 574.6685040956326,
 492.5392714434837]
```

FIND N_CLUSTER USING ELBOW METHOD

In [87]:
```
plt.plot(range(1,11),wcss,"o--")
plt.grid()
plt.show()
```



In [102…
```
kmeans=KMeans(n_clusters=4,random_state=1)
ylabel=kmeans.fit_predict(df)
```

In [103…
```
df=pd.DataFrame(df)
```

In [104…
```
df["Target"]=ylabel
df.head()
```

Out[104]:

	0	1	2	3	4	5	6	7	8	
0	0.448003	0.544917	-0.155060	0.044627	0.403235	0.055462	0.610442	-0.160046	-0.489542	-0.3
1	0.778232	0.909896	0.460873	2.207278	1.784377	1.567378	1.215811	-0.160046	3.669141	2.(
2	-0.707798	-0.550021	0.460873	-0.399154	0.403235	-0.523386	1.460109	-0.160046	-0.489542	-0.5
3	1.273575	1.457365	-0.155060	1.519730	0.999637	1.895680	0.374656	-0.160046	0.203572	2.(
4	-0.212455	-0.002552	-0.770993	0.132133	0.371846	0.210973	0.482320	-0.160046	-0.489542	0.1

5 rows × 21 columns

In [105…]:
```
kmeans.cluster_centers_
```

Out[105]:
```
array([[ 2.94569159e-02,  1.12034267e-01,  1.45744474e-01,
        -9.85513276e-02,  2.78407046e-01, -8.39757512e-02,
         5.96884375e-01, -1.60046100e-01, -2.79995928e-01,
        -1.03344310e-01, -4.42486937e-02, -4.10173104e-01,
        -5.64475856e-01, -6.07048573e-01, -6.20246604e-01,
        -6.13474879e-01,  0.00000000e+00, -4.91511796e-01,
        -1.81724340e-01, -8.39181358e-02,  1.11627907e+00],
       [-1.04879568e+00, -9.82442776e-01, -6.63874464e-01,
        -7.66570451e-01, -9.56752306e-01, -7.70926870e-01,
        -1.09258590e+00, -1.60046100e-01, -4.89541995e-01,
        -7.36073734e-01, -7.72009158e-01, -1.23679503e-01,
        -4.98975377e-02, -3.84693687e-02,  1.66651277e-02,
        -2.07267061e-02,  0.00000000e+00,  8.27968855e-01,
        -4.04929236e-02,  1.76957808e-01,  9.34782609e-01],
       [ 1.19432032e+00,  1.49386343e+00,  5.10147730e-01,
         1.87700457e+00,  1.53200468e+00,  1.89464334e+00,
         7.29675895e-01,  7.55417592e-01,  1.81159620e+00,
         1.82168900e+00,  1.71731709e+00, -7.84107502e-01,
        -8.13485051e-01, -7.82943882e-01, -8.19142356e-01,
        -7.31454563e-01,  0.00000000e+00, -1.18259178e+00,
        -1.81724340e-01, -8.39181358e-02,  2.00000000e+00],
       [ 5.90342964e-01,  1.04424431e-01,  3.97155851e-01,
        -2.56040219e-01, -2.15897033e-01, -2.85947939e-01,
         2.19000892e-01, -1.60046100e-01, -3.70039596e-01,
        -2.49621653e-01, -1.90269383e-01,  1.48032373e+00,
         1.61740948e+00,  1.63607851e+00,  1.59939886e+00,
         1.57307629e+00,  0.00000000e+00,  4.34939119e-01,
         4.90342401e-01, -8.39181358e-02, -4.44089210e-16]])
```

CLASIFICATION

In [106…]:
```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

In [107…]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=3)
```

In [108…]:
```
def algorithm(algo):
    algo.fit(xtrain,ytrain)
    ypred=algo.predict(xtest)
```

```
    train=algo.score(xtrain,ytrain)
    test=algo.score(xtest,ytest)
    print(f"TrainingAccuracy: {train}\n Testin Accuracy: {test}\n\n")
    print(classification_report(ytest,ypred))
    return algo
from sklearn.metrics import classification_report
```

In [109…
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB,MultinomialNB
```

In [110…
```
dt=algorithm(DecisionTreeClassifier())
```

```
TrainingAccuracy: 1.0
 Testin Accuracy: 0.9767441860465116


              precision    recall  f1-score   support

           0       0.93      1.00      0.97        14
           1       1.00      0.93      0.96        14
           2       1.00      1.00      1.00         9
           3       1.00      1.00      1.00         6

    accuracy                           0.98        43
   macro avg       0.98      0.98      0.98        43
weighted avg       0.98      0.98      0.98        43
```

In [111…
```
knn=algorithm(KNeighborsClassifier())
```

```
TrainingAccuracy: 0.97
 Testin Accuracy: 0.9069767441860465


              precision    recall  f1-score   support

           0       0.87      0.93      0.90        14
           1       0.92      0.79      0.85        14
           2       1.00      1.00      1.00         9
           3       0.86      1.00      0.92         6

    accuracy                           0.91        43
   macro avg       0.91      0.93      0.92        43
weighted avg       0.91      0.91      0.91        43
```

In [112…
```
lr=algorithm(LogisticRegression())
```

```
TrainingAccuracy: 1.0
 Testin Accuracy: 0.8837209302325582
```

	precision	recall	f1-score	support
0	0.92	0.79	0.85	14
1	0.92	0.86	0.89	14
2	1.00	1.00	1.00	9
3	0.67	1.00	0.80	6
accuracy			0.88	43
macro avg	0.88	0.91	0.88	43
weighted avg	0.90	0.88	0.89	43

In [113…
```
svc=algorithm(SVC())
```

```
TrainingAccuracy: 0.98
 Testin Accuracy: 0.9069767441860465
```

	precision	recall	f1-score	support
0	0.93	0.93	0.93	14
1	0.92	0.79	0.85	14
2	1.00	1.00	1.00	9
3	0.75	1.00	0.86	6
accuracy			0.91	43
macro avg	0.90	0.93	0.91	43
weighted avg	0.91	0.91	0.91	43

In [114…
```
gb=algorithm(GaussianNB())
```

```
TrainingAccuracy: 0.95
 Testin Accuracy: 0.9069767441860465
```

	precision	recall	f1-score	support
0	0.82	1.00	0.90	14
1	1.00	0.71	0.83	14
2	1.00	1.00	1.00	9
3	0.86	1.00	0.92	6
accuracy			0.91	43
macro avg	0.92	0.93	0.91	43
weighted avg	0.92	0.91	0.90	43

In [ ]:
```
```

In [ ]:
```
```