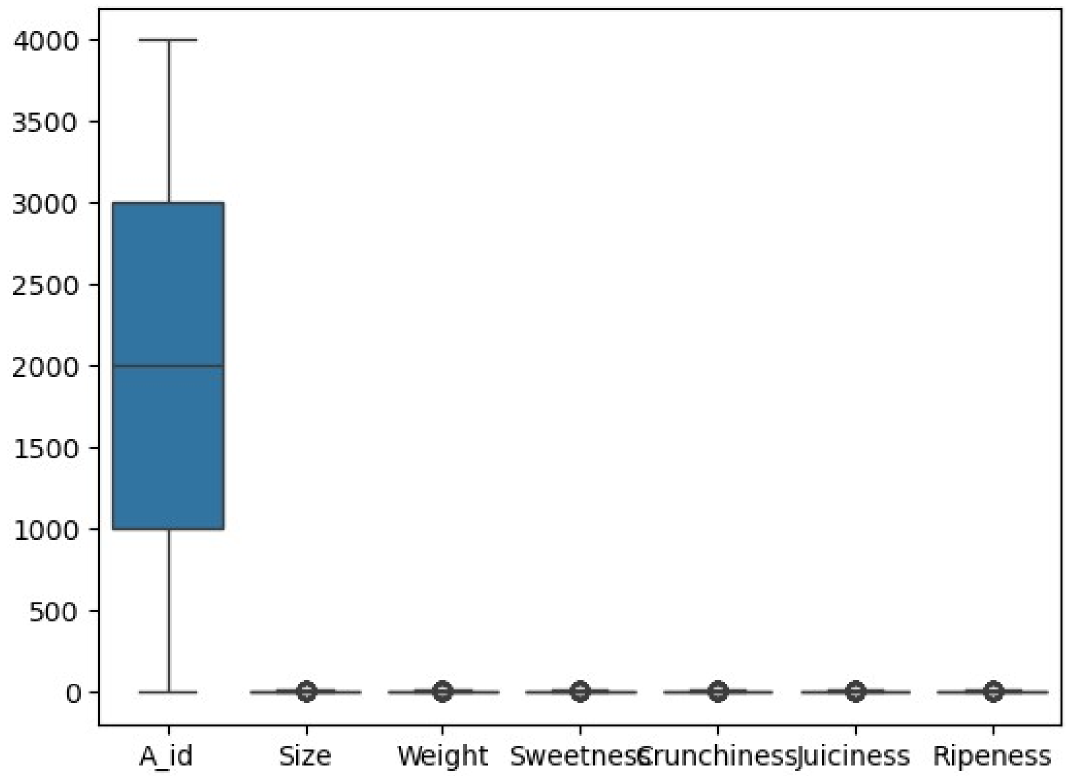
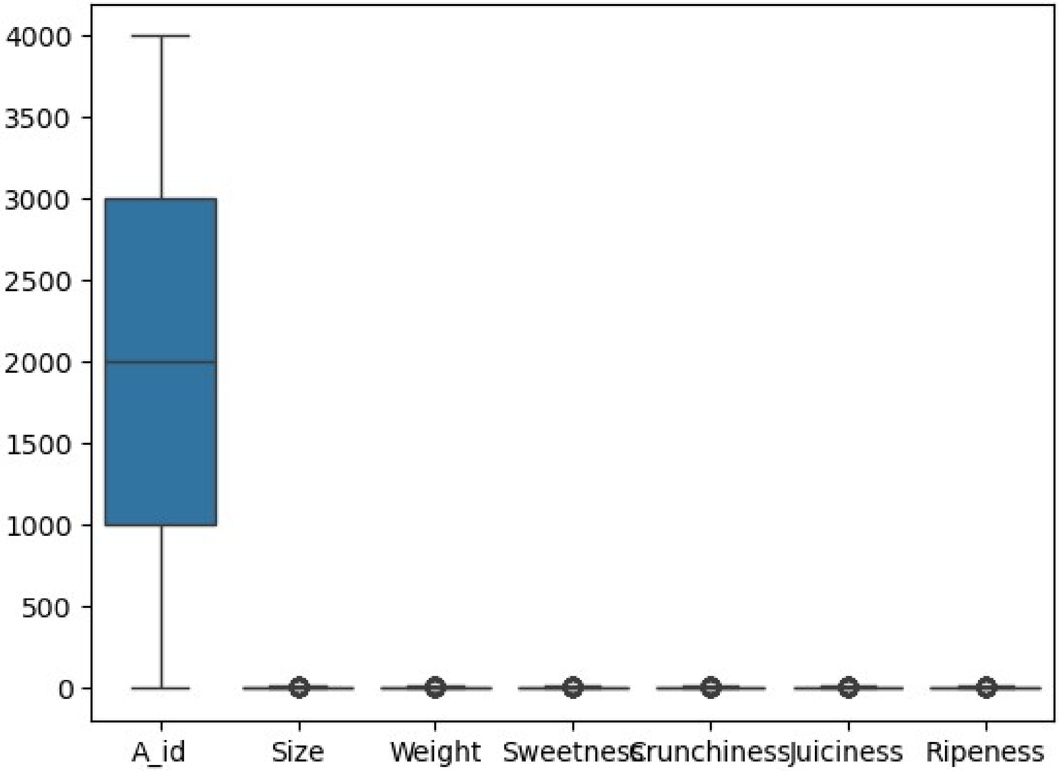
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data=pd.read_csv(r"C:\Users\mruna\Downloads\apple_quality
prediction.csv")

data.head()

   A_id      Size    Weight  Sweetness  Crunchiness  Juiciness
Ripeness  \
0   0.0 -3.970049 -2.512336   5.346330    -1.012009   1.844900
0.329840
1   1.0 -1.195217 -2.839257   3.664059     1.588232   0.853286
0.867530
2   2.0 -0.292024 -1.351282  -1.738429    -0.342616   2.838636 -
0.038033
3   3.0 -0.657196 -2.271627   1.324874    -0.097875   3.637970 -
3.413761
4   4.0  1.364217 -1.296612  -0.384658    -0.553006   3.030874 -
1.303849

        Acidity Quality
0  -0.491590483    good
1  -0.722809367    good
2   2.621636473     bad
3   0.790723217    good
4   0.501984036    good

sns.boxplot(data=data)
plt.show()
```



```
def OT_IQR(data,col):
    Q3=data[col].quantile(.75)
    Q1=data[col].quantile(.25)
    IQR=Q3-Q1
    UW=Q3+1.5*IQR
    LW=Q1-1.5*IQR
    upper_outlier=data[col]>UW
    lower_outlier=data[col]<LW
    data.loc[upper_outlier,col]=data[col].median()
    data.loc[lower_outlier,col]=data[col].median()
    return data

for i in data.select_dtypes('int','float'):
    OT_IQR(data,i)

sns.boxplot(data=data)
plt.show()
```



```
data.isnull().sum()
```

```
A_id           1
Size           1
Weight         1
Sweetness      1
Crunchiness    1
Juiciness      1
Ripeness       1
Acidity        0
Quality        1
dtype: int64
```

```
data['Quality'].mode()[0]
```

```
'good'
```

```
data['Quality'].fillna('good',inplace=True)
```

```
C:\Users\mruna\AppData\Local\Temp\ipykernel_16908\2325883758.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.
```

```
For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data['Quality'].fillna('good',inplace=True)
```

```python
data.isnull().sum()
```

```
A_id           1
Size           1
Weight         1
Sweetness      1
Crunchiness    1
Juiciness      1
Ripeness       1
Acidity        0
Quality        0
dtype: int64
```

```python
data['Size'].fillna(data['Size'].mean(),inplace=True)
```

```
C:\Users\mruna\AppData\Local\Temp\ipykernel_16908\2044219956.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data['Size'].fillna(data['Size'].mean(),inplace=True)
```

```python
data.isnull().sum()
```

```
A_id           1
Size           0
Weight         1
Sweetness      1
Crunchiness    1
Juiciness      1
Ripeness       1
Acidity        0
Quality        0
dtype: int64
```

```python
data['Weight'].fillna(data['Weight'].mean(),inplace=True)
```

```
C:\Users\mruna\AppData\Local\Temp\ipykernel_16908\534792383.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data['Weight'].fillna(data['Weight'].mean(),inplace=True)
```

```python
data['Sweetness'].fillna(data['Sweetness'].mean(),inplace=True)
```

```
C:\Users\mruna\AppData\Local\Temp\ipykernel_16908\3623260067.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data['Sweetness'].fillna(data['Sweetness'].mean(),inplace=True)
```

```python
data['Crunchiness'].fillna(data['Crunchiness'].mean(),inplace=True)
```

```
C:\Users\mruna\AppData\Local\Temp\ipykernel_16908\3067257997.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data['Crunchiness'].fillna(data['Crunchiness'].mean(),inplace=True)
```

```
data['Juiciness'].fillna(data['Juiciness'].mean(),inplace=True)
```

```
C:\Users\mruna\AppData\Local\Temp\ipykernel_16908\3160588133.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data['Juiciness'].fillna(data['Juiciness'].mean(),inplace=True)
```

```
data['Ripeness'].fillna(data['Ripeness'].mean(),inplace=True)
```

```
C:\Users\mruna\AppData\Local\Temp\ipykernel_16908\1313261182.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data['Ripeness'].fillna(data['Ripeness'].mean(),inplace=True)
```

```
data.isnull().sum()
```

```
A_id           1
Size           0
Weight         0
Sweetness      0
Crunchiness    0
Juiciness      0
Ripeness       0
Acidity        0
Quality        0
dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
LB=LabelEncoder()
data['Quality']=LB.fit_transform(data['Quality'])
```

```python
data.head()
```

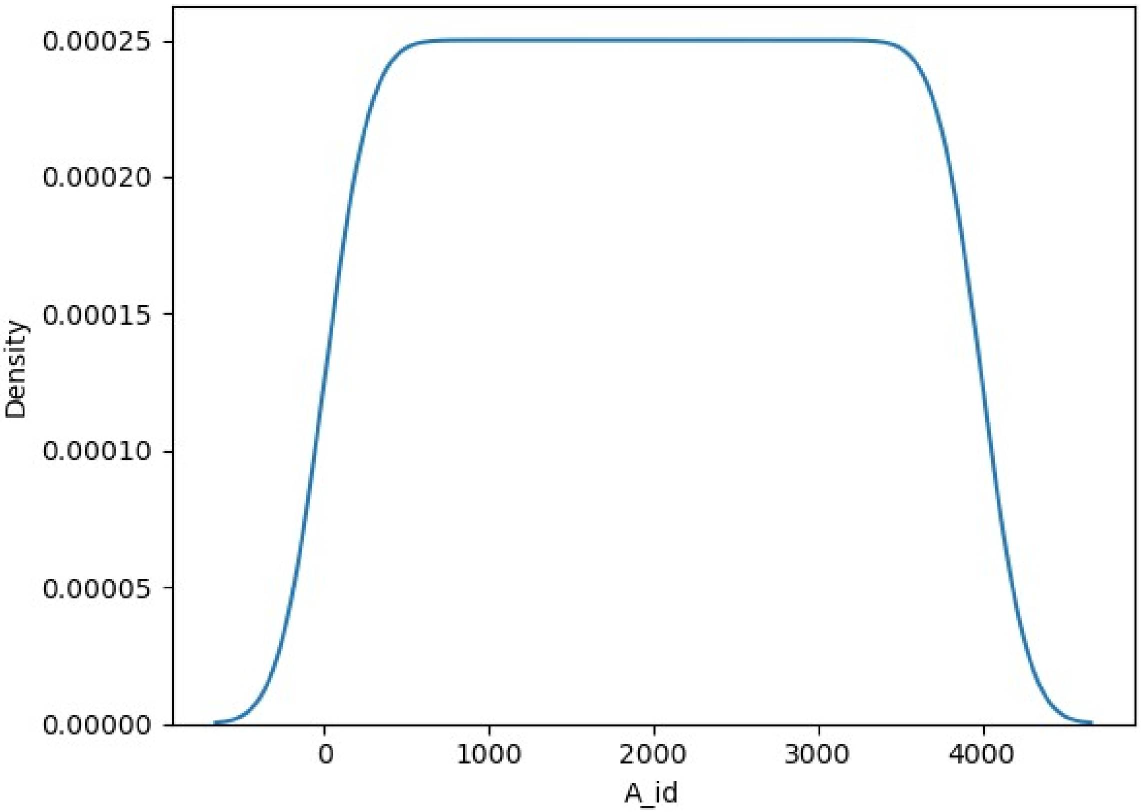
```
   A_id      Size    Weight  Sweetness  Crunchiness  Juiciness
Ripeness  \
0   0.0 -3.970049 -2.512336   5.346330    -1.012009   1.844900
0.329840
1   1.0 -1.195217 -2.839257   3.664059     1.588232   0.853286
0.867530
2   2.0 -0.292024 -1.351282  -1.738429    -0.342616   2.838636 -
0.038033
3   3.0 -0.657196 -2.271627   1.324874    -0.097875   3.637970 -
3.413761
4   4.0  1.364217 -1.296612  -0.384658    -0.553006   3.030874 -
1.303849

        Acidity  Quality
0  -0.491590483        1
1  -0.722809367        1
2   2.621636473        0
3   0.790723217        1
4   0.501984036        1
```

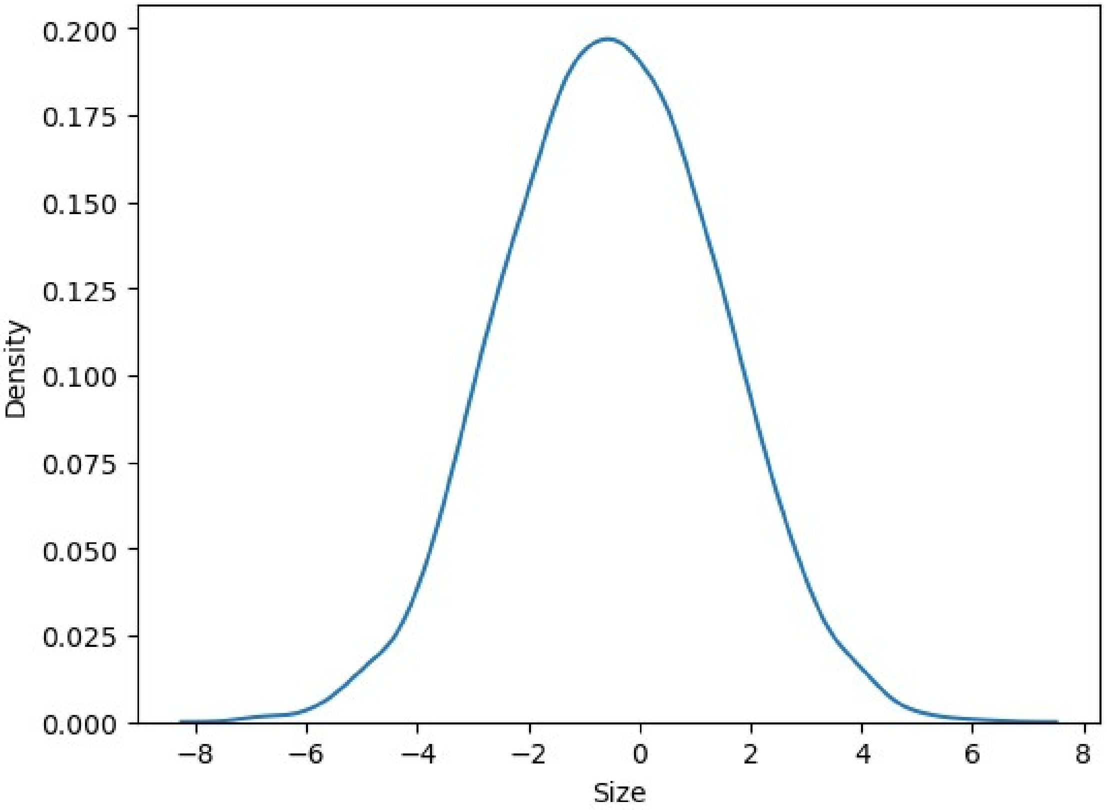
```python
data.skew(numeric_only=True)
```

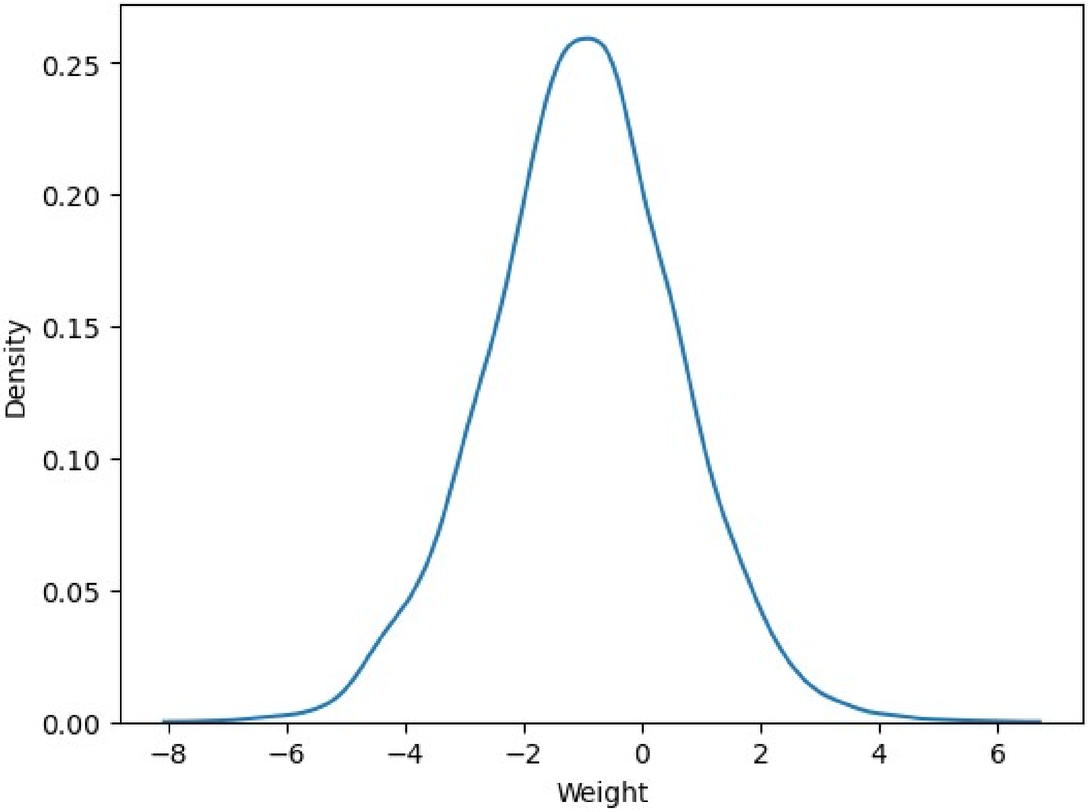
```
A_id           0.000000
Size          -0.002437
Weight         0.003102
Sweetness      0.083860
Crunchiness    0.000230
Juiciness     -0.113435
Ripeness      -0.008765
Quality       -0.004501
dtype: float64
```

```python
numerical_data=data.select_dtypes(['int','float'])
```

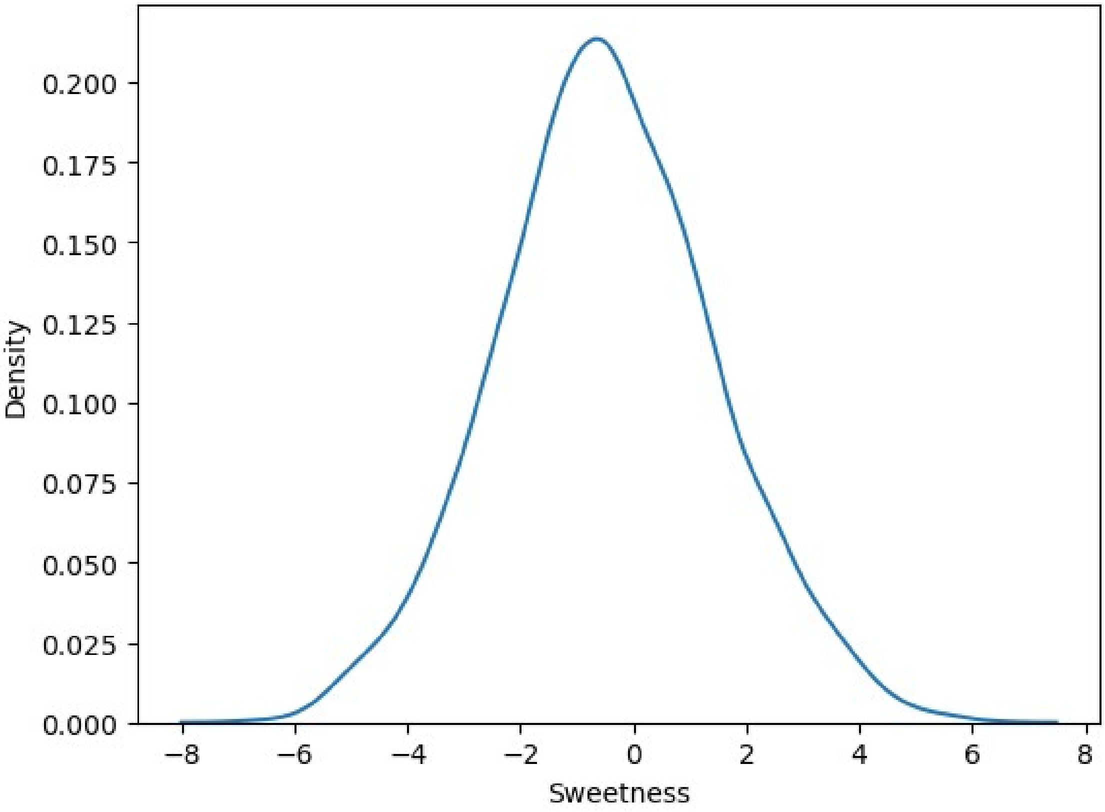
```python
for i in numerical_data:
    sns.kdeplot(data=data,x=i)
    plt.show()
```


Density
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
1000
2000
3000
4000
A_id

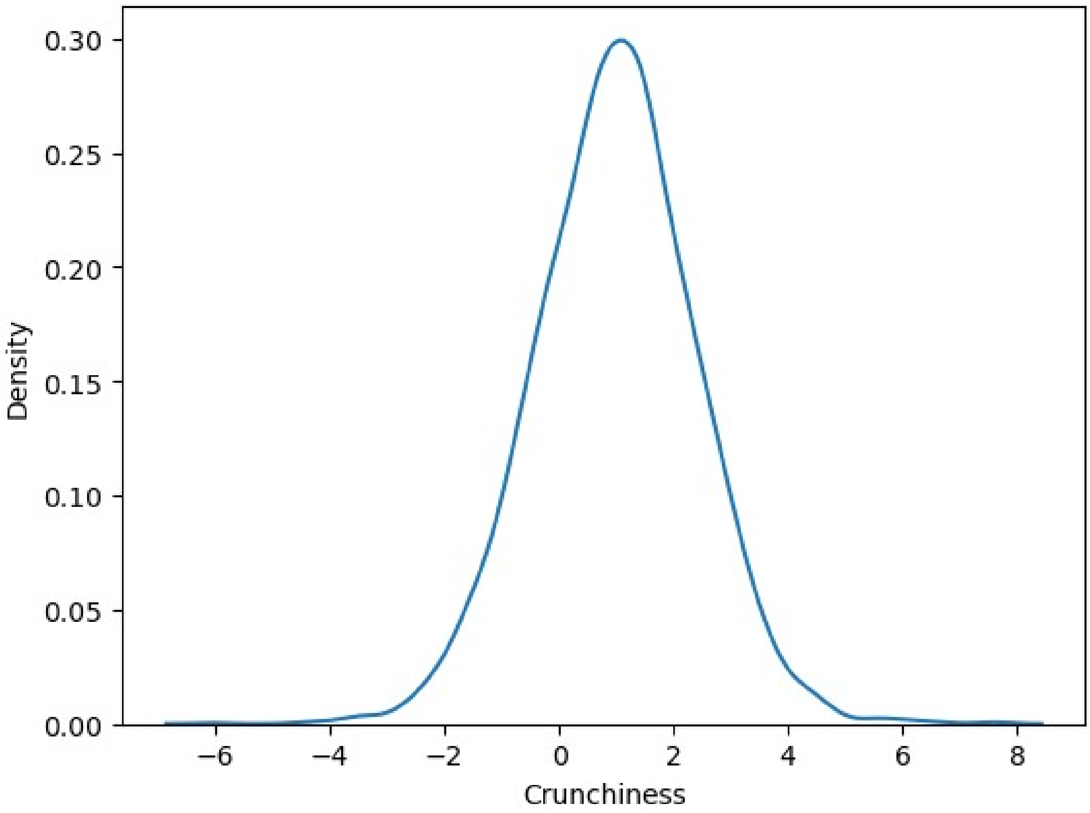

Density
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−8
−6
−4
−2
0
2
4
6
8
Size

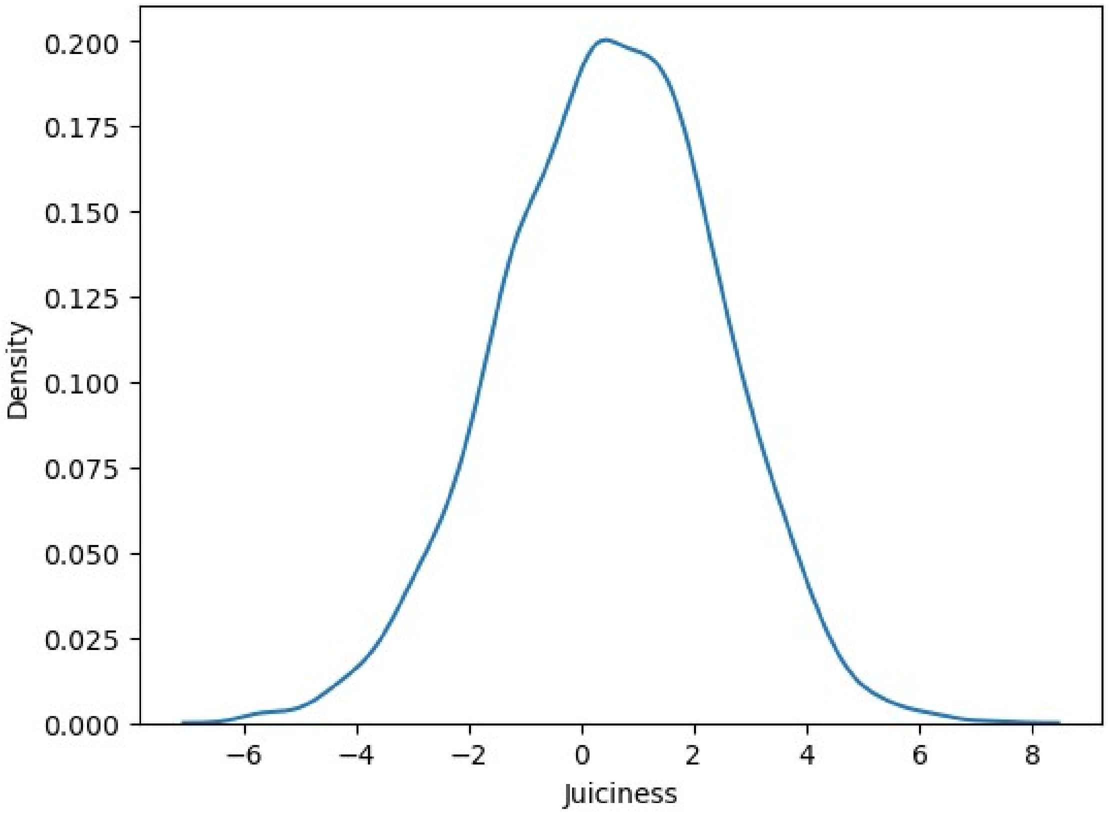


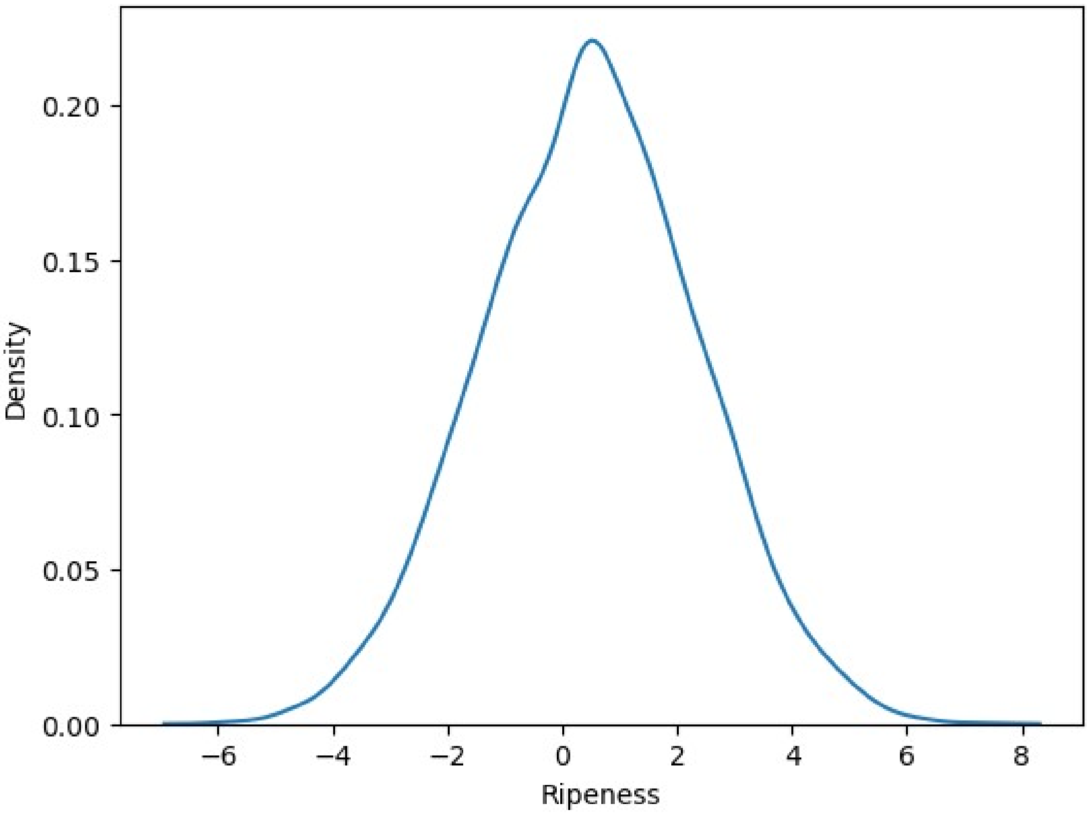
0.25
0.20
0.15
0.10
0.05
0.00
Density
−8
−6
−4
−2
0
2
4
6
Weight

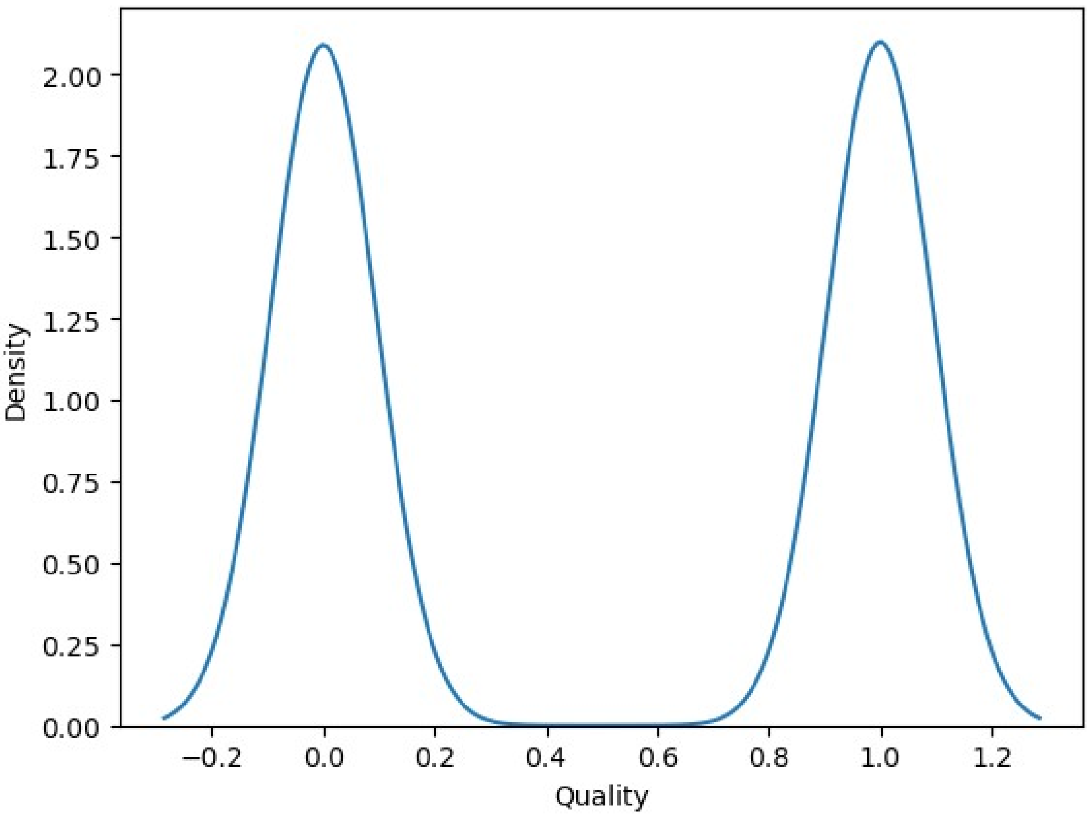


0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Density
−8
−6
−4
−2
0
2
4
6
8
Sweetness


0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
−6
−4
−2
0
2
4
6
8
Crunchiness


0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Density
−6
−4
−2
0
2
4
6
8
Juiciness


Density
0.20
0.15
0.10
0.05
0.00
−6
−4
−2
0
2
4
6
8
Ripeness


Density
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Quality

```
X=data.drop('Quality',axis=1)
Y=data.Quality

data['Acidity'] = pd.to_numeric(data['Acidity'], errors='coerce')
data = data.dropna(subset=['Acidity'])

print(data)
```

```
        A_id      Size    Weight  Sweetness  Crunchiness  Juiciness
Ripeness  \
0         0.0 -3.970049 -2.512336   5.346330    -1.012009   1.844900
0.329840
1         1.0 -1.195217 -2.839257   3.664059     1.588232   0.853286
0.867530
2         2.0 -0.292024 -1.351282  -1.738429    -0.342616   2.838636 -
0.038033
3         3.0 -0.657196 -2.271627   1.324874    -0.097875   3.637970 -
3.413761
4         4.0  1.364217 -1.296612  -0.384658    -0.553006   3.030874 -
1.303849
...       ...       ...       ...        ...          ...        ...
...
3995   3995.0  0.059386 -1.067408  -3.714549     0.473052   1.697986
2.244055
3996   3996.0 -0.293118  1.949253  -0.204020    -0.640196   0.024523 -
1.087900
3997   3997.0 -2.634515 -2.138247  -2.440461     0.657223   2.199709
4.763859
3998   3998.0 -4.008004 -1.779337   2.366397    -0.200329   2.161435
0.214488
3999   3999.0  0.278540 -1.715505   0.121217    -1.154075   1.266677 -
0.776571

       Acidity  Quality
0    -0.491590        1
1    -0.722809        1
2     2.621636        0
3     0.790723        1
4     0.501984        1
...        ...      ...
3995  0.137784        0
3996  1.854235        1
3997 -1.334611        0
3998 -2.229720        1
3999  1.599796        1

[4000 rows x 9 columns]
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(X,Y,train_size=.80,rand
om_state=0)
```

```
from sklearn.linear_model import LogisticRegression
```

```
LR=LogisticRegression(max_iter=15000)
```

```
LR.fit(x_train,y_train)
```

```
LogisticRegression(max_iter=15000)
```

```
model=LR.predict(x_test)
```

```
from sklearn.metrics import accuracy_score,f1_score
```

```
accuracy_score(y_test,model)
```

```
0.7525
```

```
f1_score(y_test,model)
```

```
0.7468030690537084
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
KNNC=KNeighborsClassifier(n_neighbors=3)
```

```
KNNC.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
model_pred2=KNNC.predict(x_test)
```

```
accuracy_score(y_test,model_pred2)
```

```
0.5675
```

```
model_pred2
```

```
array([1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1,
0,
       0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
0,
       0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1,
1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0,
1,
        0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
0,
        1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1,
        1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0,
1,
        0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0,
        1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
1,
        0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
0,
        1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1,
        0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0,
        0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
1,
        1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
0,
        0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1,
1,
        1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
1,
        1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0,
1,
        0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
0,
        0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
1,
        0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1,
1,
        1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
1,
        0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1,
0,
        1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
1,
        0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1,
0,
        0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
0,
        1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,
0,
        1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0,
        0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1,
1,
        1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0,
0,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
1,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
0,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
0,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0,
       0, 1, 1, 0, 0, 1, 0, 1])
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
DTC=DecisionTreeClassifier(criterion='entropy',max_depth=1501,random_s
tate=0)
```

```
DTC.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=1501,
random_state=0)
```

```
model_pred3=DTC.predict(x_test)
```

```
from sklearn .metrics import accuracy_score
```

```
accuracy_score(y_test,model_pred3)
```

```
0.80625
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
RFC=RandomForestClassifier()
```

```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
```

```
GSCV=GridSearchCV(estimator=RandomForestClassifier(),
                    param_grid={'n_estimators':[100,120,150],
                    'criterion':['gini','entropy'],
                    'max_depth':[4,8,12],
                    'min_samples_split':[2,3,4,5],
                    'min_samples_leaf':[1,2]},scoring='accuracy')
```

```
GCV=GridSearchCV(estimator=KNeighborsClassifier(),param_grid={'n_neigh
bors':range(3,31,2)},scoring='accuracy')
```

```
GCV.fit(x_train,y_train)
```

```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': range(3, 31, 2)},
scoring='accuracy')
```

```
GCV.best_params_
```

```
{'n_neighbors': 3}
```

```
GCV.best_score_
```

```
0.550625
```

```
RSCV=RandomizedSearchCV(estimator=KNeighborsClassifier(),param_distrib
utions={'n_neighbors':range(3,91,2)},scoring='accuracy',random_state=0
)
```

```
RSCV.fit(x_train,y_train)
```

```
RandomizedSearchCV(estimator=KNeighborsClassifier(),
                   param_distributions={'n_neighbors': range(3, 91,
2)},
                   random_state=0, scoring='accuracy')
```

```
RSCV.best_params_
```

```
{'n_neighbors': 65}
```

```
from sklearn.metrics import
accuracy_score,f1_score,classification_report,confusion_matrix
```

```
f1_score(y_test,model_pred2)
```

```
0.5471204188481675
```