```
#Load Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
#Load Data
df = pd.read_csv('/content/Raisin_Dataset.csv')
df.head(10)
```

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
0	87524	442.246011	253.291155	0.819738	90546	0.758651	1184.040	Kecimen
1	75166	406.690687	243.032436	0.801805	78789	0.684130	1121.786	Kecimen
2	90856	442.267048	266.328318	0.798354	93717	0.637613	1208.575	Kecimen
3	45928	286.540559	208.760042	0.684989	47336	0.699599	844.162	Kecimen
4	79408	352.190770	290.827533	0.564011	81463	0.792772	1073.251	Kecimen
5	49242	318.125407	200.122120	0.777351	51368	0.658456	881.836	Kecimen
6	42492	310.146072	176.131449	0.823099	43904	0.665894	823.796	Kecimen
7	60952	332.455472	235.429835	0.706058	62329	0.743598	933.366	Kecimen
8	42256	323.189607	172.575926	0.845499	44743	0.698031	849.728	Kecimen
9	64380	366.964842	227.771615	0.784056	66125	0.664376	981.544	Kecimen

```
df.columns = df.columns.str.lower()
df.head(3)
```

	area	majoraxislength	minoraxislength	eccentricity	convexarea	extent	perimete
0	87524	442.246011	253.291155	0.819738	90546	0.758651	1184.04
1	75166	406.690687	243.032436	0.801805	78789	0.684130	1121.78
2	90856	442.267048	266.328318	0.798354	93717	0.637613	1208.57

```
#Identify number of Classes (i.e. Species)
df['class'].unique()
```

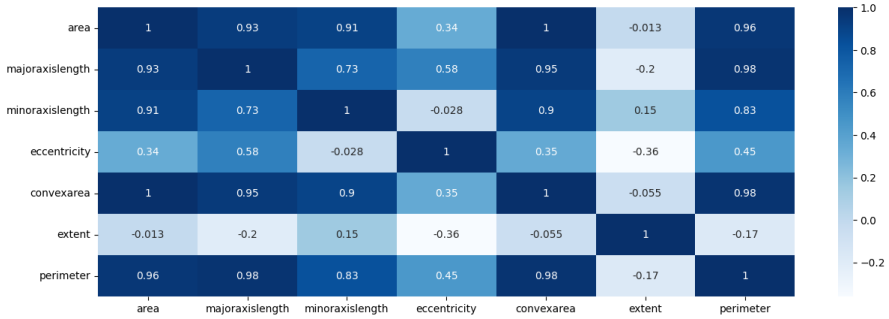
```
array(['Kecimen', 'Besni'], dtype=object)
```

```
df.describe()
```

	area	majoraxislength	minoraxislength	eccentricity	convexarea	
count	900.000000	900.000000	900.000000	900.000000	900.000000	900
mean	87804.127778	430.929950	254.488133	0.781542	91186.090000	0
std	39002.111390	116.035121	49.988902	0.090318	40769.290132	0
min	25387.000000	225.629541	143.710872	0.348730	26139.000000	0
25%	59348.000000	345.442898	219.111126	0.741766	61513.250000	0
50%	78902.000000	407.803951	247.848409	0.798846	81651.000000	0
75%	105028.250000	494.187014	279.888575	0.842571	108375.750000	0
max	235047.000000	997.291941	492.275279	0.962124	278217.000000	0

```
#Visualization of Correlations
fig = plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),annot=True,cmap="Blues")
```

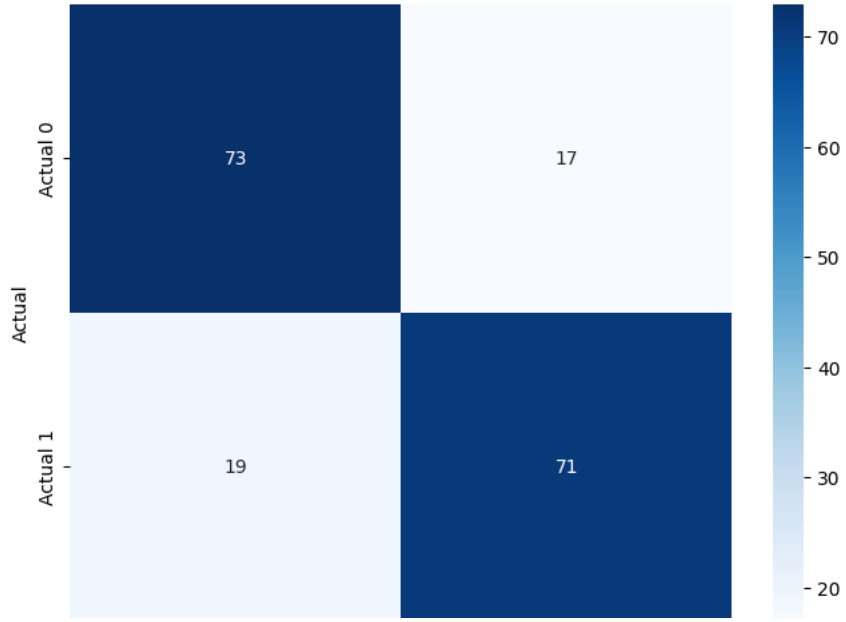
```
<ipython-input-30-9d4b9b7785a3>:3: FutureWarning: The default value of numeric_only in
  sns.heatmap(df.corr(),annot=True,cmap="Blues")
<Axes: >
```




```
#Create x and y variables
X = df.drop('class',axis=1).to_numpy()
y = df['class'].to_numpy()

#Create Train and Test datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,test_size = 0.20,random_state=100)

#Scale the data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train2 = sc.fit_transform(X_train)
x_test2 = sc.transform(X_test)


#Script for Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

for name,method in [('DT', DecisionTreeClassifier(random_state=100))]:
    method.fit(x_train2,y_train)
    predict = method.predict(x_test2)
    target_names=['Kecimen','Besni']
    print('\nEstimator: {}'.format(name))
    print(confusion_matrix(y_test,predict))
    print(classification_report(y_test,predict,target_names=target_names))
```

```
Estimator: DT
[[73 17]
 [19 71]]
              precision    recall  f1-score   support

     Kecimen       0.79      0.81      0.80        90
       Besni       0.81      0.79      0.80        90

    accuracy                           0.80       180
   macro avg       0.80      0.80      0.80       180
weighted avg       0.80      0.80      0.80       180
```

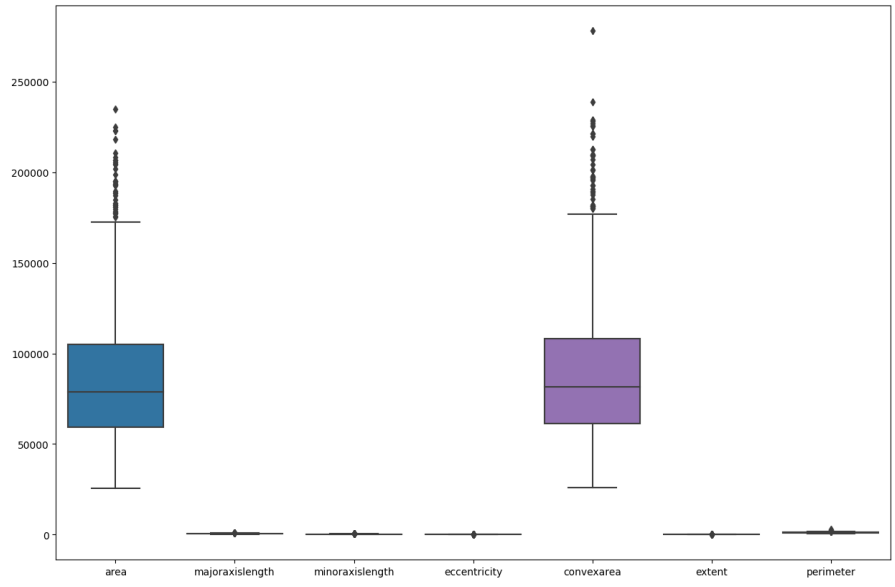
```
# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, predict)

# Visualize the confusion matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



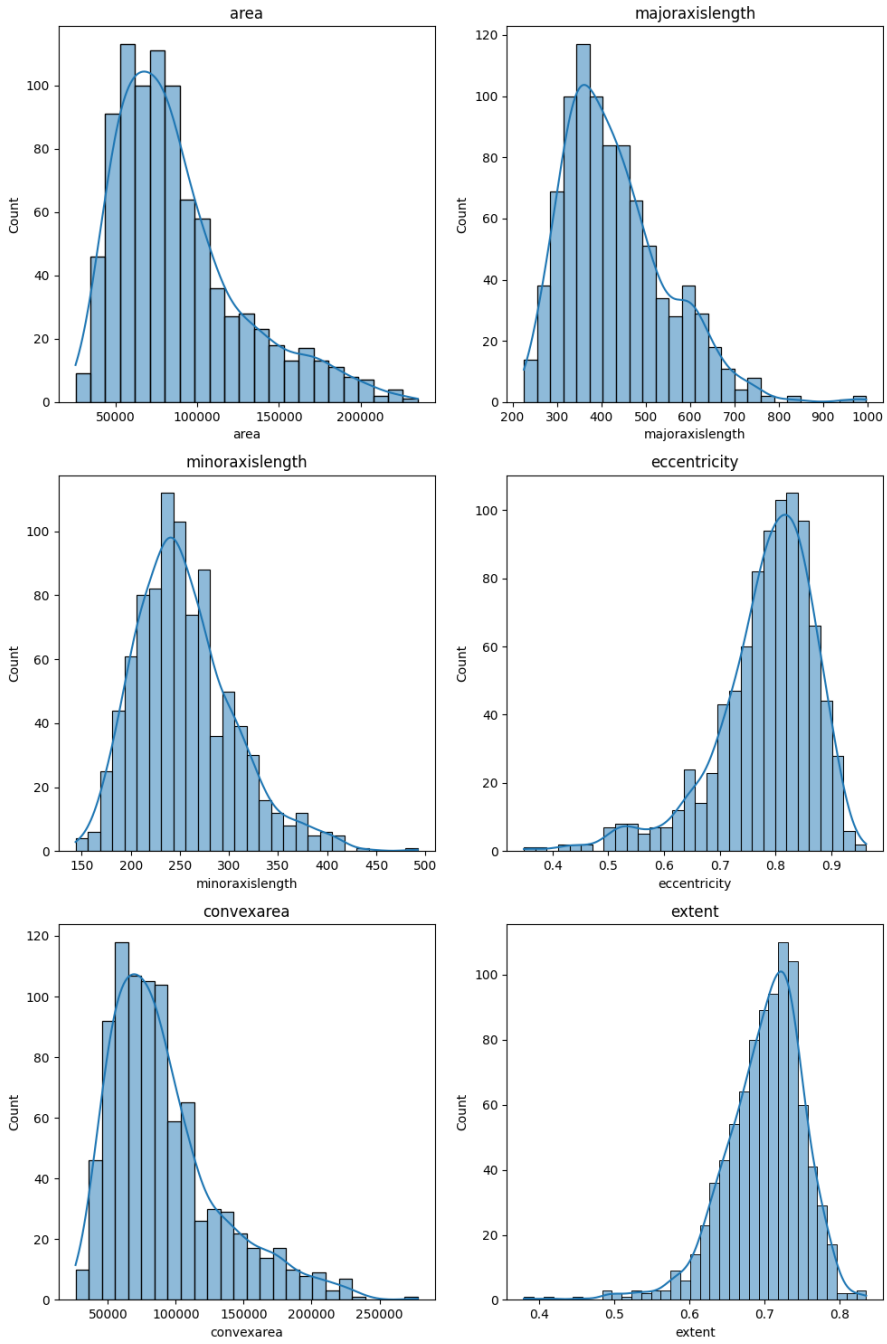
Bonus : Just for learning purpose

```
#Boxplot Visualization
plt.figure(figsize=(15,10))
sns.boxplot(data=df)
```

```
<Axes: >
```



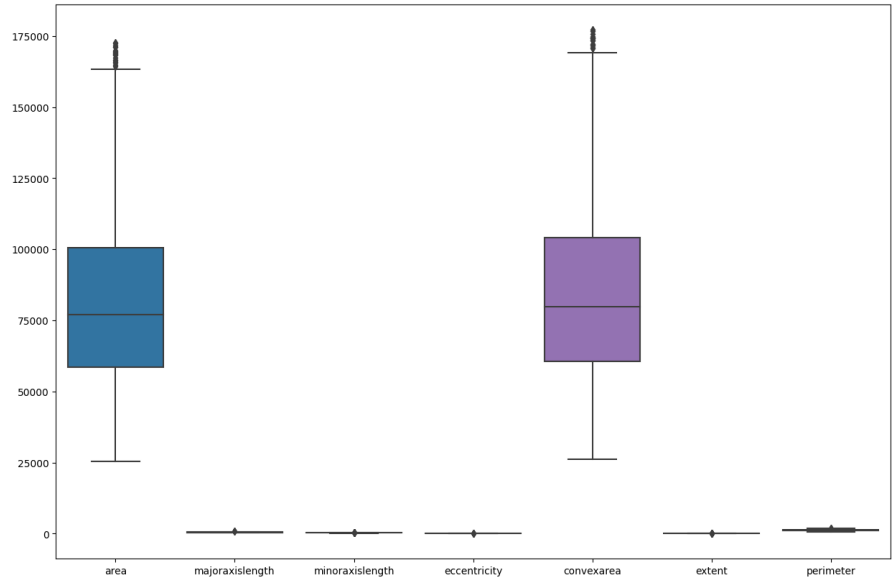
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame

num_cols = len(df.columns)
num_rows = (num_cols + 1) // 2

fig, axes = plt.subplots(num_rows, 2, figsize=(10, num_rows*5))

for idx, col in enumerate(df.columns):
    row_idx = idx // 2
    col_idx = idx % 2
    ax = axes[row_idx, col_idx]

    lmgraphhist = sns.histplot(x=df[col], kde=True, ax=ax)
    lmgraphhist.set_title(col)

for i in range(num_rows * 2 - num_cols):
    fig.delaxes(axes.flatten()[num_cols + i])

plt.tight_layout()
plt.show()
```



Cleaned Dataset

```
def remove_outliers(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    return column[(column >= lower_bound) & (column <= upper_bound)]

# Apply remove_outliers function to all columns except 'class'
for column in df.columns:
    if column != 'class':
        df[column] = remove_outliers(df[column])


#Boxplot Visualization
plt.figure(figsize=(15,10))
sns.boxplot(data=df)
```

```
<Axes: >
```

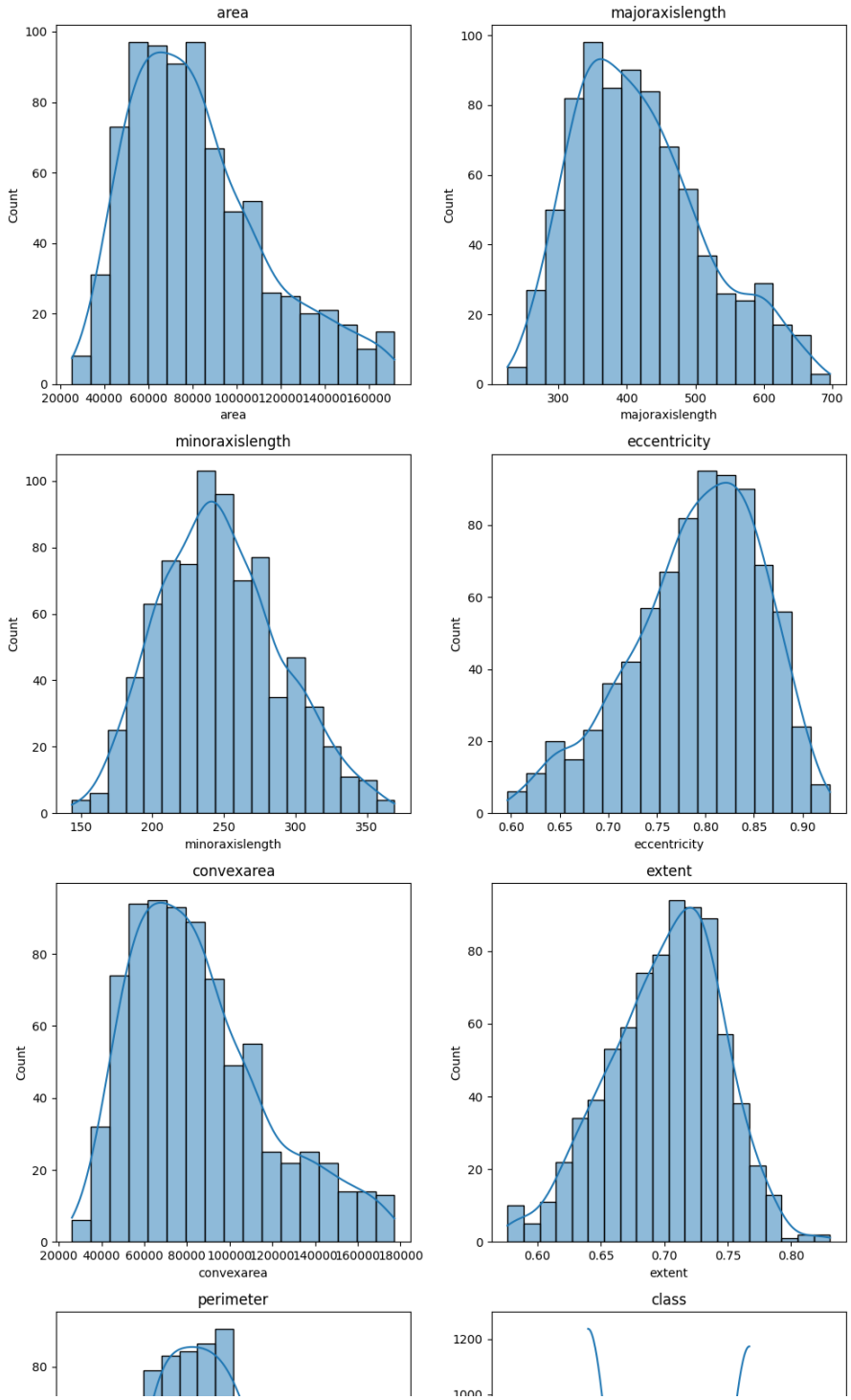



```
df=df.dropna()
```

```
df.describe()
```

	area	majoraxislength	minoraxislength	eccentricity	convexarea	
count	795.000000	795.000000	795.000000	795.000000	795.000000	795

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame

num_cols = len(df.columns)
num_rows = (num_cols + 1) // 2

fig, axes = plt.subplots(num_rows, 2, figsize=(10, num_rows*5))

for idx, col in enumerate(df.columns):
    row_idx = idx // 2
    col_idx = idx % 2
    ax = axes[row_idx, col_idx]

    lmgraphhist = sns.histplot(x=df[col], kde=True, ax=ax)
    lmgraphhist.set_title(col)

for i in range(num_rows * 2 - num_cols):
    fig.delaxes(axes.flatten()[num_cols + i])

plt.tight_layout()
plt.show()
```




```
#Create x and y variables
X = df.drop('class',axis=1).to_numpy()
y = df['class'].to_numpy()

#Create Train and Test datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,test_size = 0.20,random_state=100)

#Scale the data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train2 = sc.fit_transform(X_train)
x_test2 = sc.transform(X_test)

#Script for Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
for name,method in [('DT', DecisionTreeClassifier(random_state=100))]:
    method.fit(x_train2,y_train)
    predict = method.predict(x_test2)
    target_names=['Kecimen','Besni']
    print('\nEstimator: {}'.format(name))
    print(confusion_matrix(y_test,predict))
    print(classification_report(y_test,predict,target_names=target_names))
```

```
Estimator: DT
[[55 22]
 [14 68]]
              precision    recall  f1-score   support

      Kecimen       0.80      0.71      0.75        77
        Besni       0.76      0.83      0.79        82

     accuracy                           0.77       159
    macro avg       0.78      0.77      0.77       159
 weighted avg       0.78      0.77      0.77       159
```

```
# Create x and y variables
X = df.drop('class', axis=1).to_numpy()
y = df['class'].to_numpy()

# Create Train and Test datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.20, random_state=100)

# Scale the data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train2 = sc.fit_transform(X_train)
x_test2 = sc.transform(X_test)

# Script for Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

for name, method in [('RF', RandomForestClassifier(random_state=100))]:
    method.fit(x_train2, y_train)
    predict = method.predict(x_test2)
    target_names = ['Kecimen', 'Besni']
    print('\nEstimator: {}'.format(name))
    print(confusion_matrix(y_test, predict))
    print(classification_report(y_test, predict, target_names=target_names))
```

```
Estimator: RF
[[58 19]
 [13 69]]
              precision    recall  f1-score   support

      Kecimen       0.82      0.75      0.78        77
        Besni       0.78      0.84      0.81        82

     accuracy                           0.80       159
    macro avg       0.80      0.80      0.80       159
 weighted avg       0.80      0.80      0.80       159
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Create x and y variables
X = df.drop('class', axis=1).to_numpy()
y = df['class'].to_numpy()

# Create Train and Test datasets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.20, random_state=100)

# Scale the data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train2 = sc.fit_transform(X_train)
x_test2 = sc.transform(X_test)
```

```
x_test2 = sc.transform(x_test)

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=100)

# Instantiate GridSearchCV
grid_search = GridSearchCV(rf_classifier, param_grid, cv=5, scoring='accuracy')

# Fit the model
grid_search.fit(x_train2, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Evaluate the model
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(x_test2)

# Print confusion matrix and classification report
from sklearn.metrics import confusion_matrix, classification_report
target_names = ['Kecimen', 'Besni']
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}
[[59 18]
 [ 8 74]]
              precision    recall  f1-score   support

     Kecimen       0.88      0.77      0.82        77
       Besni       0.80      0.90      0.85        82

    accuracy                           0.84       159
   macro avg       0.84      0.83      0.84       159
weighted avg       0.84      0.84      0.84       159
```