In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
%matplotlib inline
```

In [ ]:
```
dF=pd.read_csv('Raisin_Dataset.csv')
dF
```

Out[ ]:

	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
...	...	...	...	...	...	...	...	...
895	83248	430.077308	247.838695	0.817263	85839	0.668793	1129.072	Besni
896	87350	440.735698	259.293149	0.808629	90899	0.636476	1214.252	Besni
897	99657	431.706981	298.837323	0.721684	106264	0.741099	1292.828	Besni
898	93523	476.344094	254.176054	0.845739	97653	0.658798	1258.548	Besni
899	85609	512.081774	215.271976	0.907345	89197	0.632020	1272.862	Besni

900 rows × 8 columns

Data Set Information:

Images of Kecimen and Besni raisin varieties grown in Turkey were obtained with CVS. A total of 900 raisin grains were used, including 450 pieces from both varieties. These images were subjected to various stages of pre-processing and 7 morphological features were extracted. These features have been classified using three different artificial intelligence techniques.

Attribute Information:

1.) Area: Gives the number of pixels within the boundaries of the raisin.

2.) Perimeter: It measures the environment by calculating the distance between the boundaries of the raisin and the pixels around it.

3.) MajorAxisLength: Gives the length of the main axis, which is the longest line that can be drawn on the raisin.

4.) MinorAxisLength: Gives the length of the small axis, which is the shortest line that can be drawn on the raisin.

5.) Eccentricity: It gives a measure of the eccentricity of the ellipse, which has the same moments as raisins.

6.) ConvexArea: Gives the number of pixels of the smallest convex shell of the region formed by the raisin.

7.) Extent: Gives the ratio of the region formed by the raisin to the total pixels in the bounding box.

8.) Class: Kecimen and Besni raisin.

In [ ]:
```
#Exploring the Details of the dataset
dF.info()
```

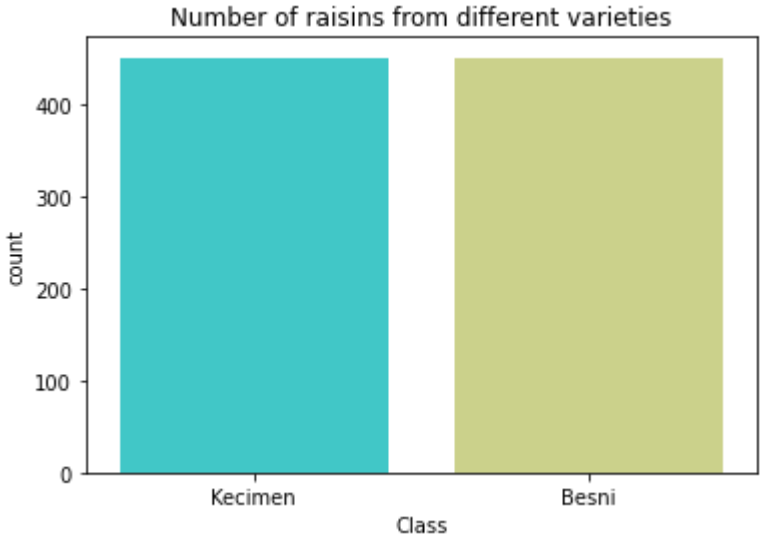
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 900 entries, 0 to 899
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Area             900 non-null    int64  
 1   MajorAxisLength  900 non-null    float64
 2   MinorAxisLength  900 non-null    float64
 3   Eccentricity     900 non-null    float64
 4   ConvexArea       900 non-null    int64  
 5   Extent           900 non-null    float64
 6   Perimeter        900 non-null    float64
 7   Class            900 non-null    object 
dtypes: float64(5), int64(2), object(1)
memory usage: 56.4+ KB
```

In [ ]:
```
#Returning All the keys from the dataset
dF.keys()
```

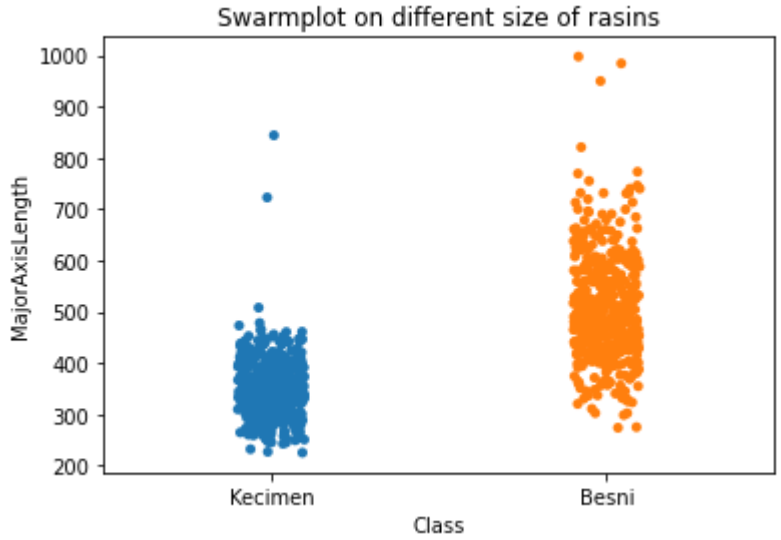
Out[ ]:
```
Index(['Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity',
       'ConvexArea', 'Extent', 'Perimeter', 'Class'],
      dtype='object')
```

In [ ]:
```
#Showing the number of rasisns from different varieties on a plot
sns.countplot(x='Class',data=dF, palette='rainbow')
plt.title('Number of raisins from different varieties')
```

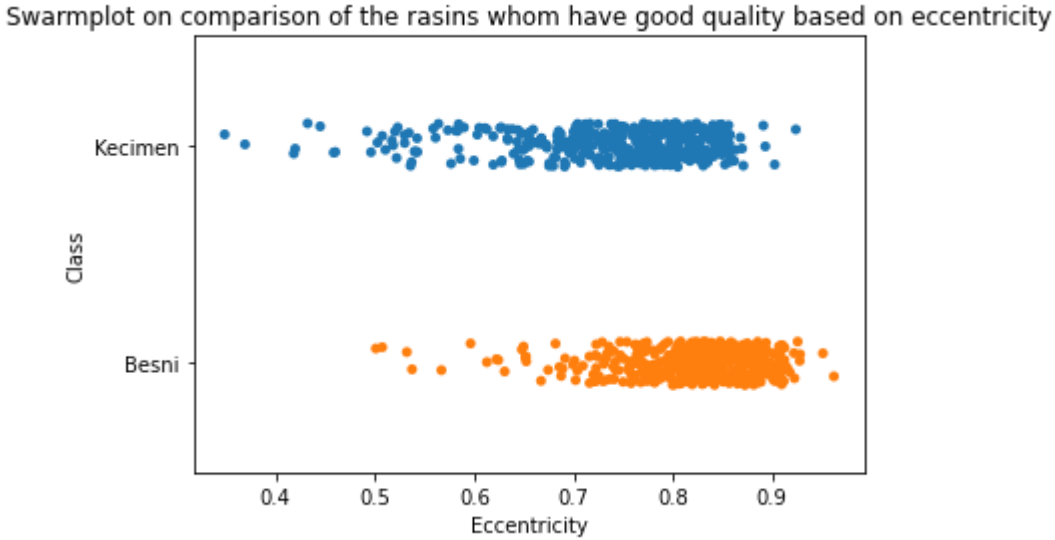
Out[ ]:
```
Text(0.5, 1.0, 'Number of raisins from different varieties')
```



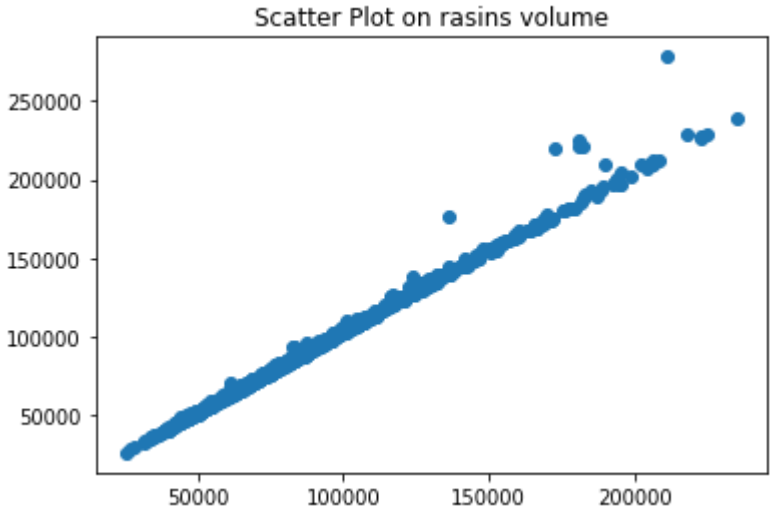
In [ ]:
```
#Plotting the Data for largest rasins
sns.stripplot(x='Class', y='MajorAxisLength', data=dF)
plt.title("Swarmplot on different size of rasins");
```



In [ ]:
```
#Plotting Data to compare the rasin whom have good quality
sns.stripplot(x='Eccentricity', y='Class', data=dF)
plt.title("Swarmplot on comparison of the rasins whom have good quality based on eccentricity");
```



In [ ]:
```
#Plotting Data to compare the area and convex area of rasins
plt.scatter(dF['Area'],dF['ConvexArea'])
plt.title("Scatter Plot on rasins volume");
```



Outcomes from the plotting.

There was actually not that much idea generated from watching the plotting. There was no pattern to identify something.

In [ ]:
```
#Counting the number of target variable
dF.groupby('Class').size()
```

Out[ ]:
```
Class
Besni      450
Kecimen    450
dtype: int64
```

In [ ]:
```
# corr() to calculate the correlation between variables
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix,precision_recall_curve, auc, roc_curve
from sklearn.tree import DecisionTreeClassifier, export_graphviz

correlation_matrix = dF.corr().round(2)
# changing the figure size
plt.figure(figsize = (14, 12))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

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


Area
MajorAxisLength
MinorAxisLength
Eccentricity
ConvexArea
Extent
Perimeter
1 0.93 0.91 0.34 1 -0.01 0.96
0.93 1 0.73 0.58 0.95 -0.2 0.98
0.91 0.73 1 -0.03 0.9 0.15 0.83
0.34 0.58 -0.03 1 0.35 -0.36 0.45
1 0.95 0.9 0.35 1 -0.05 0.98
-0.01 -0.2 0.15 -0.36 -0.05 1 -0.17
0.96 0.98 0.83 0.45 0.98 -0.17 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

In [ ]:
```
#creating the updated feature matrix using the important columns
up_df=dF[['Area','MajorAxisLength','MinorAxisLength','ConvexArea','Perimeter']]
up_df
```

Out[ ]:

	Area	MajorAxisLength	MinorAxisLength	ConvexArea	Perimeter
0	87524	442.246011	253.291155	90546	1184.040
1	75166	406.690687	243.032436	78789	1121.786
2	90856	442.267048	266.328318	93717	1208.575
3	45928	286.540559	208.760042	47336	844.162
4	79408	352.190770	290.827533	81463	1073.251
...	...	...	...	...	...
895	83248	430.077308	247.838695	85839	1129.072
896	87350	440.735698	259.293149	90899	1214.252
897	99657	431.706981	298.837323	106264	1292.828
898	93523	476.344094	254.176054	97653	1258.548
899	85609	512.081774	215.271976	89197	1272.862

900 rows × 5 columns

In [ ]:
```
#seperating the target variable
y=dF['Class']
y
```

Out[ ]:
```
0       Kecimen
1       Kecimen
2       Kecimen
3       Kecimen
4       Kecimen
         ...
895       Besni
896       Besni
897       Besni
898       Besni
899       Besni
Name: Class, Length: 900, dtype: object
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(up_df, y, test_size = 0.3,random_state = 16)
print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
print("y_train shape: ", y_train.shape)
print("y_test shape: ", y_test.shape)
```

```
X_train shape:  (630, 5)
X_test shape:  (270, 5)
y_train shape:  (630,)
y_test shape:  (270,)
```

Training: 630 out of 900 data has been chosen for training the model.

Testing: 270 out of 900 data has been chosen for testing the model.

This is a supervised machine learning model.

Because this is a labeled dataset and I know what type of data I am working with. Total 2 types of algorithms have been used to test the accuracy of this model. They are listed below:

1.Support Vector Machine algorithm

2.Decision Tree algorithm

Checking the Accuracy of the model based on Different ML Algorithms

In [ ]:
```
# importing the necessary package to use the svm algorithm
from sklearn import svm
model_svm = svm.SVC()
model_svm.fit(X_train, y_train)
y_prediction_svm = model_svm.predict(X_test)
score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)
print('The accuracy of the model based on Support Vector Machine Algorithm is {}%'.format(score_svm*100))
# perc.append(score_svm)
```

```
The accuracy of the model based on Support Vector Machine Algorithm is 83.7%
```

In [ ]:
```
# importing the necessary package to use the decision tree algorithm
from sklearn.tree import DecisionTreeClassifier
model_dt = DecisionTreeClassifier(random_state=4)
```

```
model_dt.fit(X_train, y_train)
y_prediction_dt = model_dt.predict(X_test)
score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
print('The accuracy of the model based on Decision Tree Algorithm is {}%'.format(score_dt*100))
# perc.append(score_dt)
```

```
The accuracy of the model based on Decision Tree Algorithm is 82.22%
```

In [ ]:

```
# Example of getting neighbors for an instance
from math import sqrt

# calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
        distance = 0.0
        for i in range(len(row1)-1):
                distance += (row1[i] - row2[i])**2
        return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
        distances = list()
        for train_row in train:
                dist = euclidean_distance(test_row, train_row)
                distances.append((train_row, dist))
        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(num_neighbors):
                neighbors.append(distances[i][0])
        return neighbors

# Test distance function
dataset = dF.values
neighbors = get_neighbors(dataset, dataset[0], 3)
for neighbor in neighbors:
        print(neighbor)
```

```
[87524 442.2460114 253.291155 0.819738392 90546 0.758650579 1184.04
 'Kecimen']
[87350 440.7356978 259.2931487 0.808628995 90899 0.636476246 1214.252
 'Besni']
[87252 433.4845202 261.0666438 0.798306516 90160 0.73579464 1166.517
 'Besni']
```

Model Accuracy:

This model has an accuracy of 83.7% based on Support Vector Machine algorithm.

In [ ]:
```
# Example of separating data by class value

# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
        separated = dict()
        for i in range(len(dataset)):
                vector = dataset[i]
                class_value = vector[-1]
                if (class_value not in separated):
                        separated[class_value] = list()
                separated[class_value].append(vector)
        return separated

separated = separate_by_class(dF.values)
for label in separated:
        print(label)
        for row in separated[label]:
                print(row)
```

```
Kecimen
[87524 442.2460114 253.291155 0.819738392 90546 0.758650579 1184.04
 'Kecimen']
[75166 406.690687 243.0324363 0.801805234 78789 0.68412957 1121.786
 'Kecimen']
[90856 442.2670483 266.3283177 0.798353619 93717 0.637612812 1208.575
 'Kecimen']
[45928 286.5405586 208.7600423 0.684989217 47336 0.699599385 844.162
 'Kecimen']
[79408 352.1907699 290.8275329 0.56401133 81463 0.792771926 1073.251
 'Kecimen']
[49242 318.125407 200.12212 0.777351277 51368 0.658456354 881.836
 'Kecimen']
[42492 310.1460715 176.1314494 0.823098681 43904 0.665893562 823.796
 'Kecimen']
[60952 332.4554716 235.429835 0.706057518 62329 0.74359819 933.366
 'Kecimen']
[42256 323.1896072 172.5759261 0.845498789 44743 0.698030924 849.728
 'Kecimen']
[64380 366.9648423 227.7716147 0.784055626 66125 0.664375716 981.544
 'Kecimen']
[80437 449.4545811 232.3255064 0.856042518 84460 0.674235757 1176.305
 'Kecimen']
[43725 301.3222176 186.9506295 0.784258452 45021 0.697068248 818.873
 'Kecimen']
[43441 276.6108288 201.8131355 0.683882337 45133 0.690855598 803.748
 'Kecimen']
[76792 338.8575454 291.3592017 0.510583813 78842 0.772322237 1042.77
 'Kecimen']
[74167 387.7989307 247.8581228 0.769089738 76807 0.680181585 1084.729
 'Kecimen']
[33565 261.5543311 167.7084908 0.767374275 35794 0.68155052 751.413
 'Kecimen']
[64670 403.0839752 206.4846437 0.858829168 66419 0.75677257 1028.445
 'Kecimen']
[64762 354.2939396 235.7524629 0.746473726 66713 0.694998015 981.509
 'Kecimen']
[43295 304.2844667 182.8110368 0.799406959 44714 0.713838189 814.68
 'Kecimen']
[70699 418.6985723 216.5960537 0.855799392 72363 0.728075054 1061.321
 'Kecimen']
[69726 354.1769124 252.529208 0.701160962 71849 0.734398534 1035.501
 'Kecimen']
[57346 330.4784385 222.4437485 0.739555027 59365 0.723608833 928.272
```

```
 'Kecimen']
[82028 397.1149759 268.3337727 0.737169367 84427 0.686375085 1106.355
 'Kecimen']
[61251 301.5077895 273.6599414 0.419753707 64732 0.643595671 971.769
 'Kecimen']
[96277 447.1345225 275.2161542 0.788128405 97865 0.704057157 1181.921
 'Kecimen']
[75620 368.2242844 263.4592554 0.698627251 77493 0.726277372 1059.186
 'Kecimen']
[73167 340.055218 276.0151772 0.58410581 74545 0.778736856 1010.474
 'Kecimen']
[60847 336.9238696 231.4656959 0.726660229 62492 0.69858783 964.603
 'Kecimen']
[81021 347.7500583 297.6406265 0.517134931 82552 0.757559607 1063.868
 'Kecimen']
[59902 358.5919148 222.9020273 0.783331958 63250 0.744124224 982.788
 'Kecimen']
[88745 429.770355 265.6902361 0.786009488 90715 0.752063524 1162.877
 'Kecimen']
[41809 307.5327392 175.085568 0.822113695 43838 0.697444367 828.697
 'Kecimen']
[75329 364.2307798 265.8668635 0.683510499 77541 0.723079729 1075.792
 'Kecimen']
[61600 350.1827545 225.8427713 0.764243075 63397 0.746829611 972.472
 'Kecimen']
[46427 253.8420284 235.9068241 0.369212459 48275 0.684219059 844.312
 'Kecimen']
[40861 249.7402266 213.5732718 0.51832833 43096 0.743089401 784.912
 'Kecimen']
[55827 305.298843 234.6612245 0.639696077 57724 0.703287982 926.095
 'Kecimen']
[54182 366.0666742 192.013274 0.851391425 56450 0.611417674 968.729
 'Kecimen']
[77468 405.9365937 245.9897977 0.795479241 79220 0.721545397 1100.676
 'Kecimen']
[49882 287.2643272 222.1858727 0.633851884 50880 0.766377827 843.764
 'Kecimen']
[95245 397.094114 307.2739224 0.633422315 97988 0.75304396 1201.39
 'Kecimen']
[71464 364.1030896 253.7969272 0.717025543 73265 0.715484271 1036.94
 'Kecimen']
[77055 375.2501319 262.8124219 0.713783938 79255 0.732267077 1095.283
 'Kecimen']
[92384 368.4062138 320.7145792 0.492086934 93772 0.749006827 1135.662
```