AI Project
Iris Flower

Kmeans Algorithm

Course: (23164402-4) Artificial Intelligence

Instructor Name: I.Manar Aljohani

Due Date: 13 April ,2021

Team Members

Student name	Student ID
Shatha Muslih Alharb	**438007605**
Afrah Bakheet Allhyani	**438009788**
Bayader Saad AlHarbi	**438009056**
Atheer Waleed Al Hazmi	**438006113**
Safia Mrzoog AlFahmi	**438007175**

Content Table

1.Abtract

We all know the importance of artificial intelligence, especially with the development of science, this area is almost becoming an integral part of our day.

In our project, we will use the kmeans algorithm to classify the iris flower. This could benefit for farmers or those interested of environment and botany, we will explain in a simple way the kmeans algorithm and implementation technique, in addition to a detailed explanation of the source code and at the end we will show the results.

2. Introduction about kmeans

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms, its aim is simple grouping similar data points together and discovering basic patterns. Briefly, how it works the K-means algorithm identifies k number of centroids (number of cluster), and then allocates every data point to the nearest cluster.

3. Iris flower dataset

The Iris flower dataset is also called the Fisher's Iris dataset.
It was created by the scientist Sir Ronald Aylmer Fisher. he was born in 1890 in London, England, and was well-known as a statistician and geneticist. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

Predicted attribute:
Class of iris plant.

Attribute information:

1. Sepal length in cm.
2. Sepal width in cm.
3. Petal length in cm.
4. Petal width in cm.
5. Also have 3 Classes as shown in Figure 1.

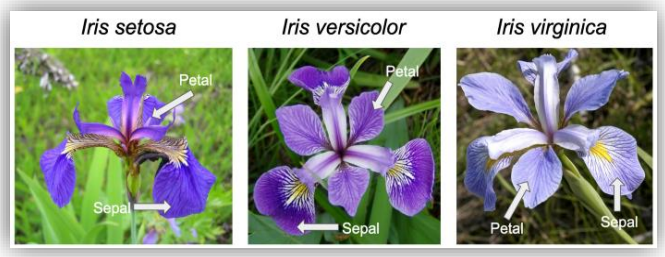

Figure1 Iris classes

4. How we used iris dataset

We will take the SepalLength and PetalLength values and will be spilt into training and testing data in order to classify the flower type.

5. Implementation technique

To implement the project, we downloaded the iris dataset from its website and then downloaded the required libraries.

We do import for the downloaded libraries and then read the data from an csv file and specify the columns required from it, which are the SepalLength and PetalLength columns, after that the data is spilt into training and testing data with specifying the number of cluster to be three according to the flower classifications, then printing the results with the evaluation.

6. About source code

```
#AI Project iris flower- Kmeans algorithm
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import scale
from scipy.spatial.distance import cdist
from sklearn.metrics import accuracy_score
from sklearn.metrics import silhouette_score
from sklearn import metrics
```

– Import required libraries after downloaded it.

```
#read dataset file
data = pd.read_csv("/iris.csv")
```

- From pandas library we use read_csv function to read the file that contain the dataset and store it into data variable.

```
#splitting the data and select it
dataSplit = 130
selectedData = data[["SepalLength","PetalLength","Species"]]
```

- dataSpilt variable to store the number by which the data is splitting, selectedData variable to determined which columns we needed from dataset .

```
#to determine the training data
trainingData = selectedData[:dataSplit]
X_train = np.array(trainingData[["SepalLength","PetalLength"]].va
lues)
y_train_flower_name = np.array(trainingData[["Species"]].values)
```

- trainingData variable to store the data from 0 to the number was stored into dataSpilt variable, X_train variable to store trainingData as np array and take SpalLength and PetalLength as values to it, y_train_flower_name also store trainingData as np array but take values of Species .

```
#to determine the testing  data
testingtData = selectedData[dataSplit:]
X_test = np.array(testingtData[["SepalLength","PetalLength"]].val
ues)
y_test_flower_name = np.array(testingtData[["Species"]].values)
```

- The same way for testingData, testingData variable to store the data from the number was stored into dataSpilt variable to the end of dataset, X_test variable to store testingData as np array and take SpalLength and PetalLength as values to it, y_test_flower_name also store testingData as np array but take values of Species .

```
#create model and train it
k = 3
model = KMeans(n_clusters=k)
model = model.fit((X_train))
```

- k variable to determined number of cluster, then we created model object from KMeans class and pass k as number of cluster, now we need to train this model by using fit function and pass the X_train as parameter.

```
#the centroid the algorithm generated it
centroidsArray = model.cluster_centers_
print("Centoids:")
print(centroidsArray)
```

- From K Means we use cluster_centers_ attribute to return the coordinates or center points as array and store it into centroidArray variable.

```
# We can look at the clusters each data point was assigned to
print("Training Model Classes Result:")
y_result = model.labels_
print(y_result)
```

- Now we need to see the labels of training data in model so we used labels_ attributes in KMeans class to return the labels as array and store it into y_result variable .

```
#we create dictionary for flower classes
i=0
flower_to_class = {}
for flower in y_train_flower_name[:,0]:
    flower_to_class[flower] = y_result[i]
    i = i+1

#print the dictionary
print("Mapped Flower Name to Class Number:")
print(flower_to_class)
```

- Here we created dictionary to know the class which associated with each flower , where the key is flower name and the values is class for it .

```
#test the model
print("Testing Classes With Predict (y_pred):")
y_pred = model.predict(X_test)
print(y_pred)
```

- For test the model we used predict function in KMeans class and pass x_test as parameter, then store it into y_pred variable.

```
# What true predict ?
y_test = [0] * len(y_test_flower_name)
i = 0
for flower in y_test_flower_name[:,0]:
    for key in flower_to_class:
        if(y_test_flower_name[i] == key):
            y_test[i] = flower_to_class[key]
            i = i + 1
            break

#print true predict
print("True Testing Classes (y_test):")
print(np.array(y_test))
```

- Now we need to ensure that the model predicts the data correctly or not? We use the previously created dictionary to compare classes and then print the correct classes in y_test variable.

```
#validate the model
print("Accuracy:")
score = accuracy_score(y_test, y_pred, normalize=True, sample_wei
ght=None) * 100
print(score)
```

- To validation of model we used score function, this function take y_test and y_pred as parameter then return the accuracy.Larger accuracy is better for model.

```
# And we'll visualize it:
plt.figure(figsize=(8, 6))
#for training
scatter_x = X_train[:,0]
scatter_y = X_train[:,1]
group = y_train_flower_name
colors = {1: 'red', 2: 'blue', 3: 'green'}

i=1
fig, ax = plt.subplots()
for flowerName in np.unique(group):
    filteredData = trainingData.loc[trainingData['Species'] == fl
owerName]
    scatter_x = np.array(filteredData.iloc[:,0])
    scatter_y = np.array(filteredData.iloc[:,1])
    ax.scatter(scatter_x, scatter_y, c =colors[i], label = flower
Name, s = 100)
```

```
    i = i +1
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
#for centroid
plt.scatter(centroidsArray[:,0] ,centroidsArray[:,1], color='yell
ow')
plt.show()

#for testing
scatter_x = X_test[:,0]
scatter_y = X_test[:,1]
group = y_test_flower_name
colors = {1: 'red', 2: 'blue', 3: 'green'}
i=1
fig, ax = plt.subplots()
for flowerName in np.unique(group):
    filteredData = testingtData.loc[testingtData['Species'] == fl
owerName]
    scatter_x = np.array(filteredData.iloc[:,0])
 scatter_y = np.array(filteredData.iloc[:,1])
 ax.scatter(scatter_x, scatter_y, c =colors[i], label =
flowerName, s = 100)
    i = i +1

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
#for centroid
plt.scatter(centroidsArray[:,0] ,centroidsArray[:,1], color='yell
ow')
plt.show()
```

- Finally, we visualize the data by matplotlib library. scatter_x to store the first columns , scatter_y for second columns , then we store y_test_flower_name in group variable , also created dictionary to assign color for each flower .
The for loop for group but we using unique function to take unique flower name from y_test_flower_name so the counter is 3 ,then with each iteration we created graph for each flower and assign color for it . Also we assign another color for centroid . The same thing with testing data ..

7. Result

7.1 When training data larger than testing

130 data as training and 20 as testing we have output as shown in Figure 2 and Figure 3.

```
Centoids:
[[5.00666667 1.49111111]
 [5.91428571 4.43265306]
 [6.85555556 5.69722222]]
Training Model Classes Result:
[2 1 2 1 1 2 0 2 1 2 0 1 2 1 1 0 0 2 2 1 0 1 0 2 1 0 1 0 2 1 2 2 0 1 1 0 0
 1 0 1 1 0 2 2 0 0 1 1 0 1 1 1 1 0 0 1 1 2 1 0 0 0 0 2 1 0 1 1 0 1 2 1 2 2
 0 2 2 0 2 0 2 2 1 0 0 0 2 1 1 0 0 2 0 1 1 1 1 0 1 0 0 0 2 0 0 1 2 0 1 1 2
 0 1 1 1 2 2 0 2 2 1 1 2 0 0 0 2 1 2 1]
Mapped Flower Name to Class Number:
{'Iris-virginica': 2, 'Iris-versicolor': 1, 'Iris-setosa': 0}
Testing Classes With Predict (y_pred):
[2 0 0 1 1 0 0 1 1 2 1 0 0 2 1 1 1 1 2]
True Testing Classes (y_test):
[2 0 0 1 1 0 0 1 1 2 1 0 0 2 1 1 2 1 2 2]
Accuracy:
90.0
```

Figure 2 Output1

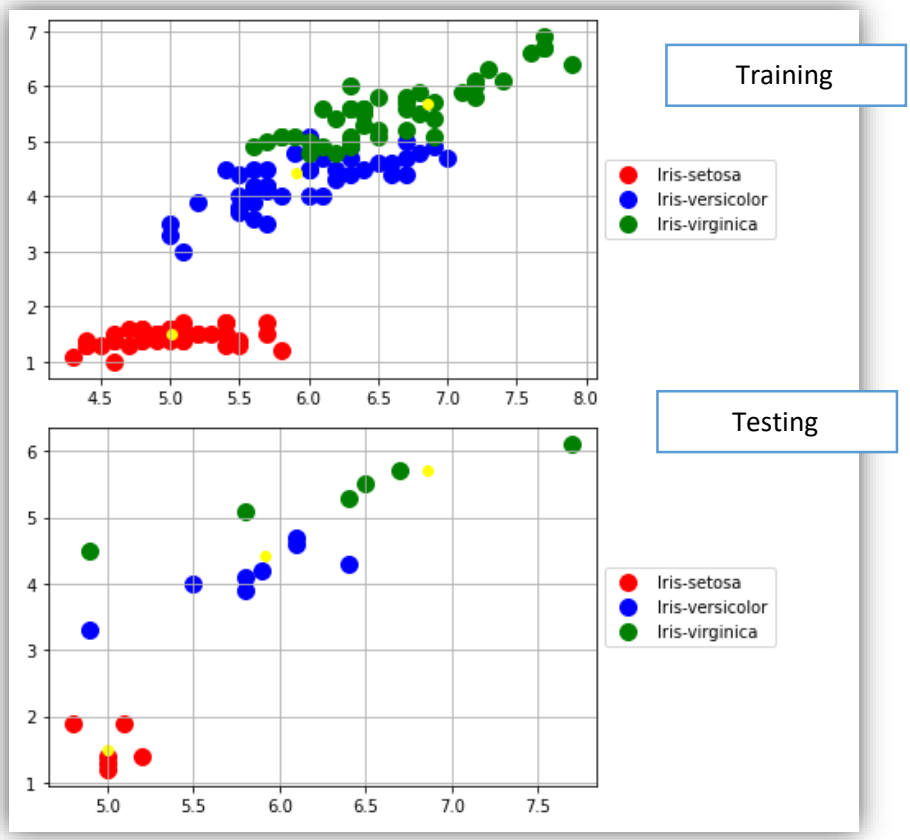

Figure 3 Visualiz1

7.2 When training data is less than testing

40 data as training and 110 as testing we have output as shown in Figure 4 and Figure 5.

```
Centoids:
[[5.175      1.46666667]
 [5.9625     4.3625    ]
 [6.86666667 5.85833333]]
Training Model Classes Result:
[2 1 2 1 1 2 0 2 1 2 0 1 2 1 1 0 0 2 2 1 0 1 0 2 1 0 1 0 2 1 2 2 0 1 1 0 0
 1 0 1]
Mapped Flower Name to Class Number:
{'Iris-virginica': 1, 'Iris-versicolor': 1, 'Iris-setosa': 0}
Testing Classes With Predict (y_pred):
[1 0 2 2 0 0 1 1 0 1 1 1 1 0 0 1 1 2 1 0 0 0 0 2 1 0 1 1 0 1 2 1 2 2 0 2 2
 0 2 0 2 2 1 0 0 0 2 1 1 0 0 2 0 1 1 1 1 0 1 0 0 0 2 0 0 1 2 0 1 1 2 0 1 1
 1 2 2 0 1 2 1 1 1 0 0 0 2 1 2 1 2 0 0 1 1 0 0 1 1 2 1 0 0 2 1 1 1 1 1 2]
True Testing Classes (y_test):
[1 0 1 1 0 0 1 1 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 0 1 1 1 0 1 1
 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1]
Accuracy:
75.45454545454545
```

Figure 4 Output2

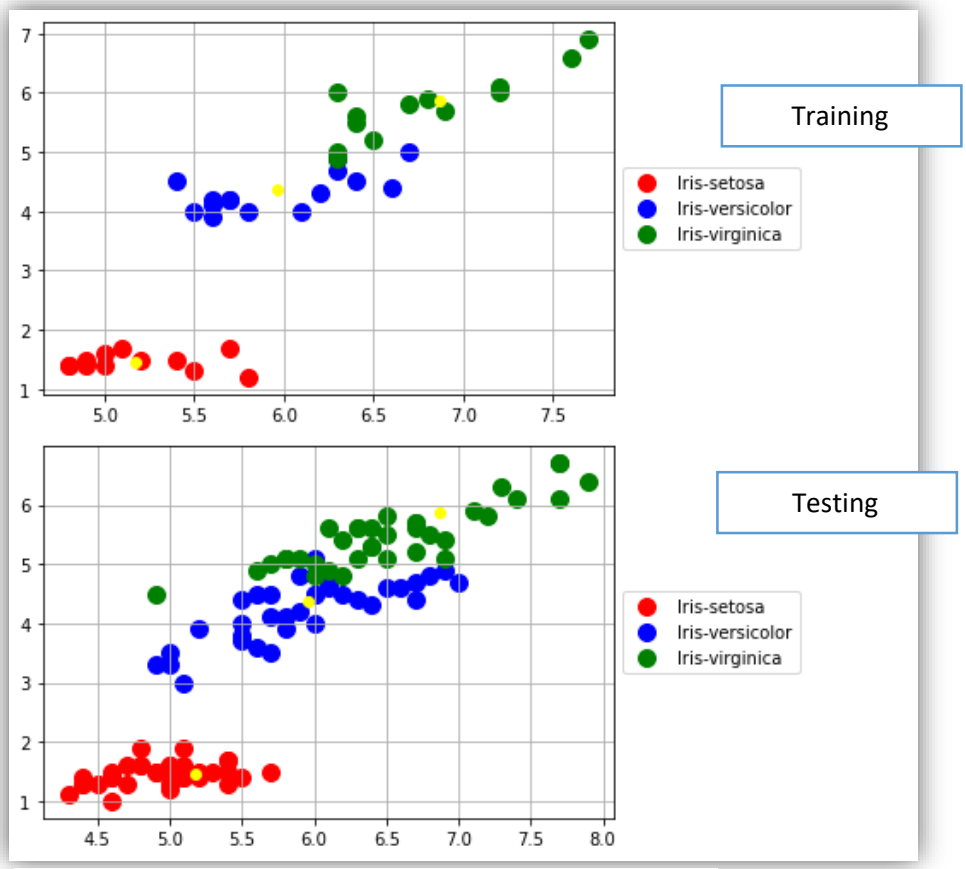

Figure 5 Visualize2

7.3 When training and testing data is equal

75 data as training and 75 as testing we have output as shown in Figure 6 and Figure 7.

```
Centoids:
[[5.8483871  4.31612903]
 [5.092      1.476     ]
 [6.82105263 5.82105263]]
Training Model Classes Result:
[2 0 2 0 0 2 1 2 0 2 1 0 2 0 0 1 1 2 2 0 1 0 1 2 0 1 0 1 2 0 2 2 1 0 0 1 1
 0 1 0 0 1 2 2 1 1 0 0 1 0 0 0 0 1 1 0 0 2 0 1 1 1 1 2 0 1 0 0 1 0 2 0 2 2
 1]
Mapped Flower Name to Class Number:
{'Iris-virginica': 2, 'Iris-versicolor': 0, 'Iris-setosa': 1}
Testing Classes With Predict (y_pred):
[2 2 1 2 1 2 2 0 1 1 1 2 0 0 1 1 2 1 2 0 0 0 1 0 1 1 1 2 1 1 0 2 1 0 0 2 1
 0 0 0 2 2 1 2 2 0 0 2 1 0 1 2 0 2 0 2 1 1 0 0 1 1 0 0 2 0 1 1 2 0 0 0 0 0
 2]
True Testing Classes (y_test):
[2 2 1 2 1 2 2 2 1 1 1 0 0 0 1 1 2 1 2 2 0 2 1 2 1 1 1 2 1 1 0 2 1 0 0 2 1
 0 2 2 2 2 1 0 2 0 0 0 1 0 1 2 0 2 0 2 1 1 0 0 1 1 0 0 2 0 1 1 2 0 0 2 0 2
 2]
Accuracy:
85.33333333333334
```

Figure 6 Output3

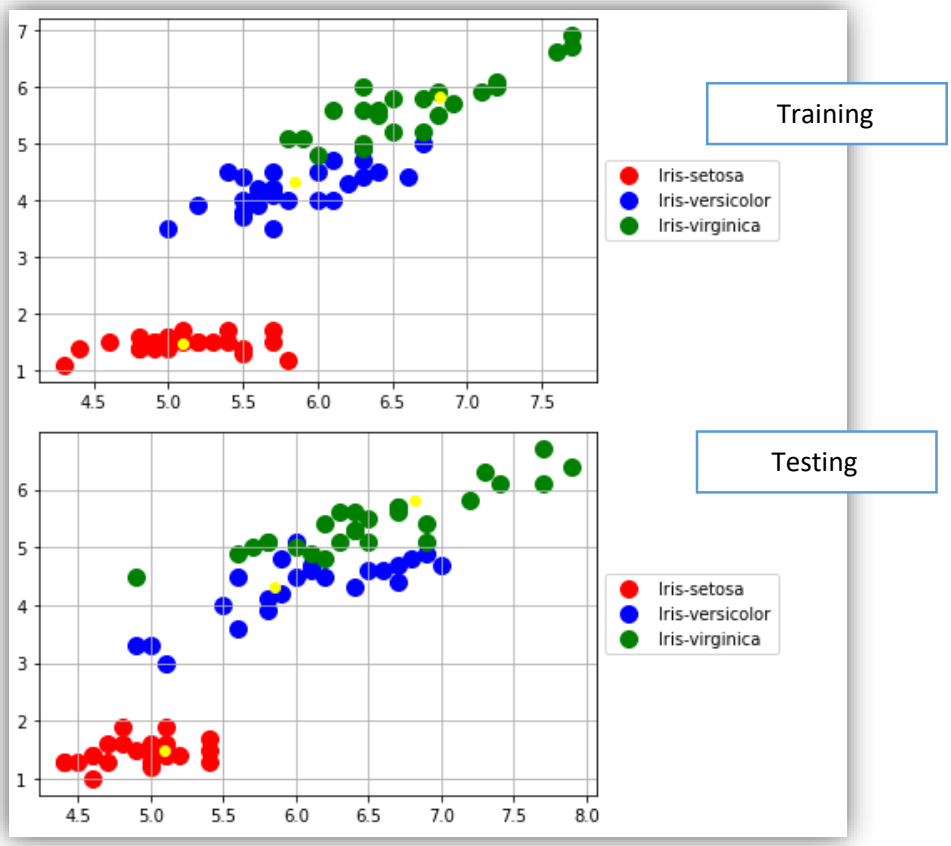

Figure 7 Visualize3

7.4 Conclusion of Results

Training	Testing	Correct classes	Incorrect classes	Accuracy
130	20	18	2	90.0
40	110	83	27	75.45
75	75	64	11	85.33

- NOTE: The accuracy increased when increase the training data.

8. Source Code

```
#AI Project iris flower- Kmeans algorithm
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import scale
from scipy.spatial.distance import cdist
from sklearn.metrics import accuracy_score
from sklearn.metrics import silhouette_score
from sklearn import metrics

#read dataset file
data = pd.read_csv("/iris.csv")

#To make sure the data is read correctly,
#you can try printing the first five columns
#print(data.head(5))

#splitting the data and select it
dataSplit = 75
selectedData = data[["SepalLength","PetalLength","Species"]]
```

```
#to determine the training data
trainingData = selectedData[:dataSplit]
X_train = np.array(trainingData[["SepalLength","PetalLength"]].va
lues)
y_train_flower_name = np.array(trainingData[["Species"]].values)


#to determine the testing  data
testingtData = selectedData[dataSplit:]
X_test = np.array(testingtData[["SepalLength","PetalLength"]].val
ues)
y_test_flower_name = np.array(testingtData[["Species"]].values)

#create model and train it
k = 3
model = KMeans(n_clusters=k)
model = model.fit((X_train))


#the centroid the algorithm generated it
centroidsArray = model.cluster_centers_
print("Centoids:")
print(centroidsArray)

# We can look at the clusters each data point was assigned to
print("Training Model Classes Result:")
y_result = model.labels_
print(y_result)

#we create dictionary for flower classes
i=0
flower_to_class = {}
for flower in y_train_flower_name[:,0]:
    flower_to_class[flower] = y_result[i]
    i = i+1

#print the dictionary
print("Mapped Flower Name to Class Number:")
print(flower_to_class)

#test the model
print("Testing Classes With Predict (y_pred):")
y_pred = model.predict(X_test)
print(y_pred)
```

```
# What true predict ?
y_test = [0] * len(y_test_flower_name)
i = 0
for flower in y_test_flower_name[:,0]:
    for key in flower_to_class:
        if(y_test_flower_name[i] == key):
            y_test[i] = flower_to_class[key]
            i = i + 1
            break

#print true predict
print("True Testing Classes (y_test):")
print(np.array(y_test))

#validate the model
print("Accuracy:")
score = accuracy_score(y_test, y_pred, normalize=True, sample_wei
ght=None) * 100
print(score)

# And we'll visualize it:
plt.figure(figsize=(8, 6))
#for training
scatter_x = X_train[:,0]
scatter_y = X_train[:,1]
group = y_train_flower_name
colors = {1: 'red', 2: 'blue', 3: 'green'}

i=1
fig, ax = plt.subplots()
for flowerName in np.unique(group):
    filteredData = trainingData.loc[trainingData['Species'] == fl
owerName]
    scatter_x = np.array(filteredData.iloc[:,0])
    scatter_y = np.array(filteredData.iloc[:,1])
    ax.scatter(scatter_x, scatter_y, c =colors[i], label = flower
Name, s = 100)
    i = i +1
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
#for centroid
plt.scatter(centroidsArray[:,0] ,centroidsArray[:,1], color='yell
ow')
plt.show()
```

```
#for testing
scatter_x = X_test[:,0]
scatter_y = X_test[:,1]
group = y_test_flower_name
colors = {1: 'red', 2: 'blue', 3: 'green'}

i=1
fig, ax = plt.subplots()
for flowerName in np.unique(group):
    filteredData = testingtData.loc[testingtData['Species'] == fl
owerName]
    scatter_x = np.array(filteredData.iloc[:,0])
    scatter_y = np.array(filteredData.iloc[:,1])
    ax.scatter(scatter_x, scatter_y, c =colors[i], label = flower
Name, s = 100)
    i = i +1

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)
#for centroid
plt.scatter(centroidsArray[:,0] ,centroidsArray[:,1], color='yell
ow')
plt.show()
```

9. References

[1]https://archive.ics.uci.edu/ml/datasets/iris
[2] https://towardsdatascience.com/the-iris-dataset-a-little-bit-of-history-and-biology-fb4812f5a7b5

[3]https://scikit-learn.org/stable/modules/clustering.html#k-means

[4]https://saskeli.github.io/data-analysis-with-python-summer-2019/clustering.html

[5]https://www.youtube.com/watch?v=qg_M37WGKG8&list=RDCMUC2nvtxeY_rJLnlJKEg-J82g&start_radio=1&t=1955s

[6] https://www.youtube.com/watch?v=qs7vES46Rq8&t=528s

[7] https://www.youtube.com/watch?v=pKBUmK4XT2I

[8] https://www.youtube.com/watch?v=IGNiMBkSfus