Muhammad Asif Khan

S19004470

Final Assignment: Analytical Design Tasks

Step 1 (Exploratory Data Analysis)

Introduction

In this task, I will use clustering algorithms to classify and predict a species of flower based on the characteristics of the flower.

I will use the Iris flower data set or Fisher's Iris data set which is multivariate data set named after a British biologist and statistician Ronald Fisher. I have used the Learning Repository link of the University of California to download the dataset.

https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

The data set consists of information on 150 Iris flowers, 50 each from one of three Iris species: Setosa, Versicolour, and Virginica. Each flower is characterized by four attributes as shown below in Fig1.

a) sepal length in centimetres
b) sepal width in centimetres
c) petal length in centimetres
d) petal width in centimetres

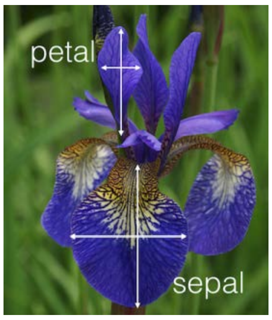


Fig 1

Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other [3].

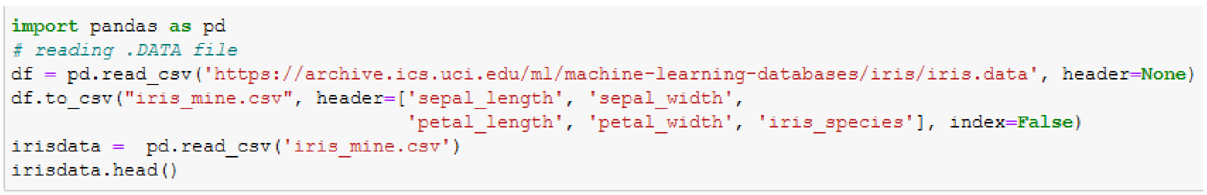
I have used the following steps to perform exploratory analysis on the given set of data. For each step, I have provided the Python code and the corresponding output.

Steps

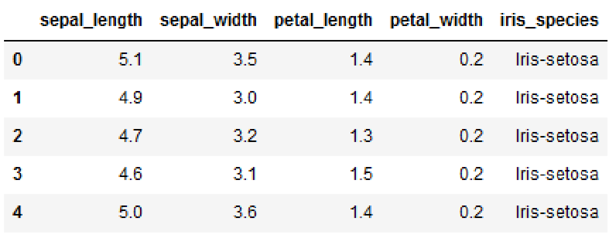
1.1 Load the Iris dataset from the University of California at Irvine Machine Learning Repository by using the given link.

https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

1.2 Convert the data into a csv file with appropriate header information and save it on my computer. Finally, load the dataset by using numpy.read_csv() method as shown below [1].



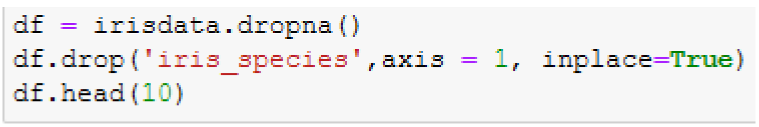
```
import pandas as pd
# reading .DATA file
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None)
df.to_csv("iris_mine.csv", header=['sepal_length', 'sepal_width',
                                   'petal_length', 'petal_width', 'iris_species'], index=False)
irisdata =  pd.read_csv('iris_mine.csv')
irisdata.head()
```



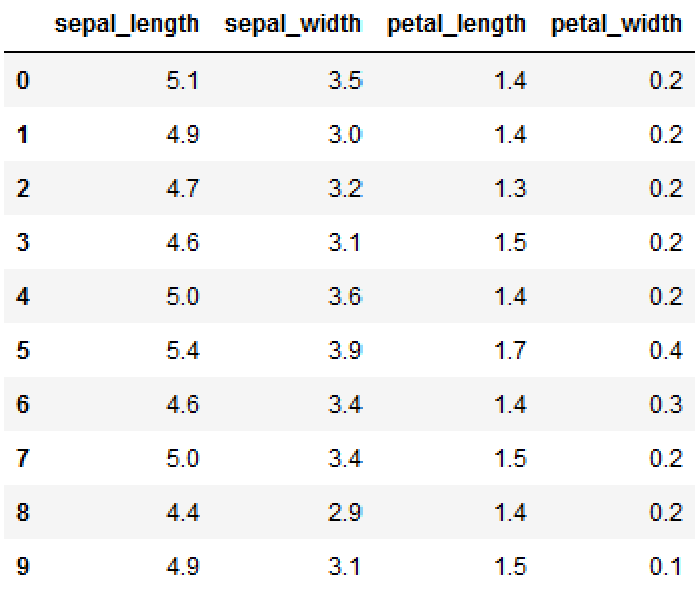
	sepal_length	sepal_width	petal_length	petal_width	iris_species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Fig 2

1.3 Clean the data frame by checking for any NAN (Not A Number) or NA (Not applicable) values and remove them. Also remove all the other attributes and keep only the required attributes i.e., sepal width, sepal length, petal length, and petal width in an appropriate format. The first 10 rows are shown below [1,2].



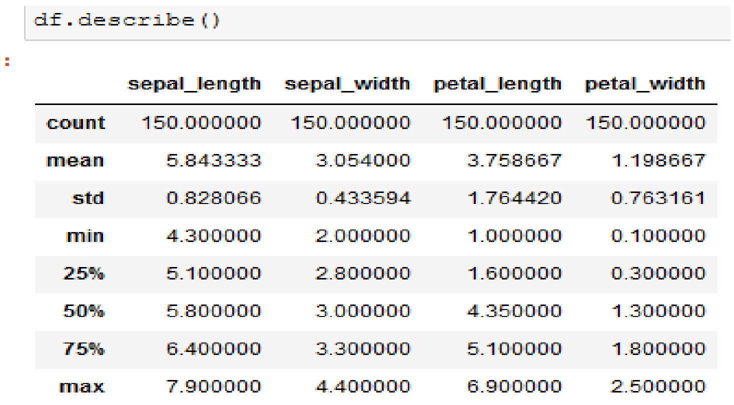
```
df = irisdata.dropna()
df.drop('iris_species',axis = 1, inplace=True)
df.head(10)
```



	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1

Fig 3

1.4 Use pandas describe() method to extract all the meaningful statistics. These include a summary statistic of the data frame including number of data counts, its mean, standard deviation, minimum, maximum for each attribute as shown below.

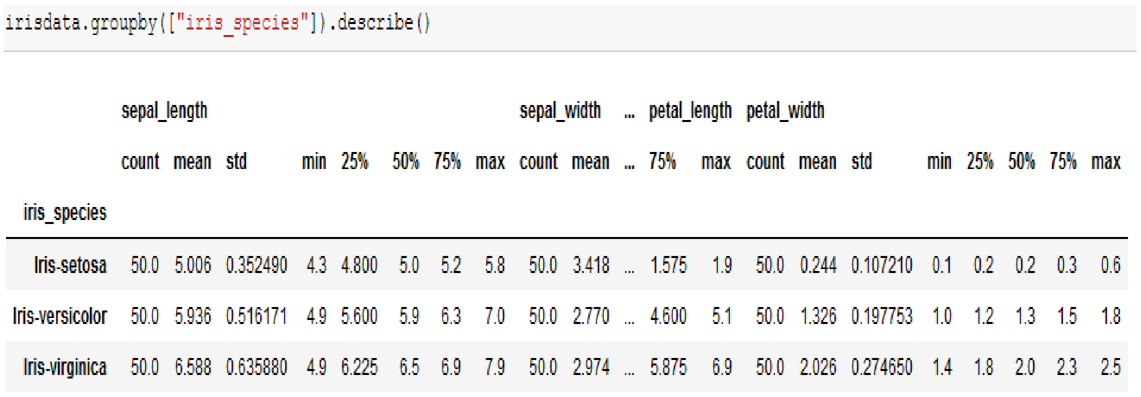


```
df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Fig 4

1.5 By using groupby(["iris_species"]).describe() method, get count, mean, standard deviation, minimum, maximum for each species as shown below [4].



```
irisdata.groupby(["iris_species"]).describe()
```

	sepal_length								sepal_width		...	petal_length		petal_width							
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%	75%	max
iris_species																					
Iris-setosa	50.0	5.006	0.352490	4.3	4.800	5.0	5.2	5.8	50.0	3.418	...	1.575	1.9	50.0	0.244	0.107210	0.1	0.2	0.2	0.3	0.6
Iris-versicolor	50.0	5.936	0.516171	4.9	5.600	5.9	6.3	7.0	50.0	2.770	...	4.600	5.1	50.0	1.326	0.197753	1.0	1.2	1.3	1.5	1.8
Iris-virginica	50.0	6.588	0.635880	4.9	6.225	6.5	6.9	7.9	50.0	2.974	...	5.875	6.9	50.0	2.026	0.274650	1.4	1.8	2.0	2.3	2.5

Fig 5

Observation

I noticed that Iris-setosa specie has a smaller mean lengths and widths of petals and sepals as compared to Iris-versicolor and Iris-virginica. It means that its flower will be the smallest.

Also, standard deviation (std) values of its attributes are smaller which means that it produces a flower of consistent size.

1.6 Show statistical significance by providing appropriate visualisation between various attributes in the given data frame by using a pairplot which shows the bivariate relation between each pair of features. I have used a ***Python*** useful library called ***seaborn*** to plot the pairplot as shown in Fig 6 below [1].

```
import seaborn as sns

sns.set(style="ticks", color_codes=True)
sns.pairplot(irisdata, hue="iris_species", height=3)
```

```
<seaborn.axisgrid.PairGrid at 0x2640c0cfd90>
```

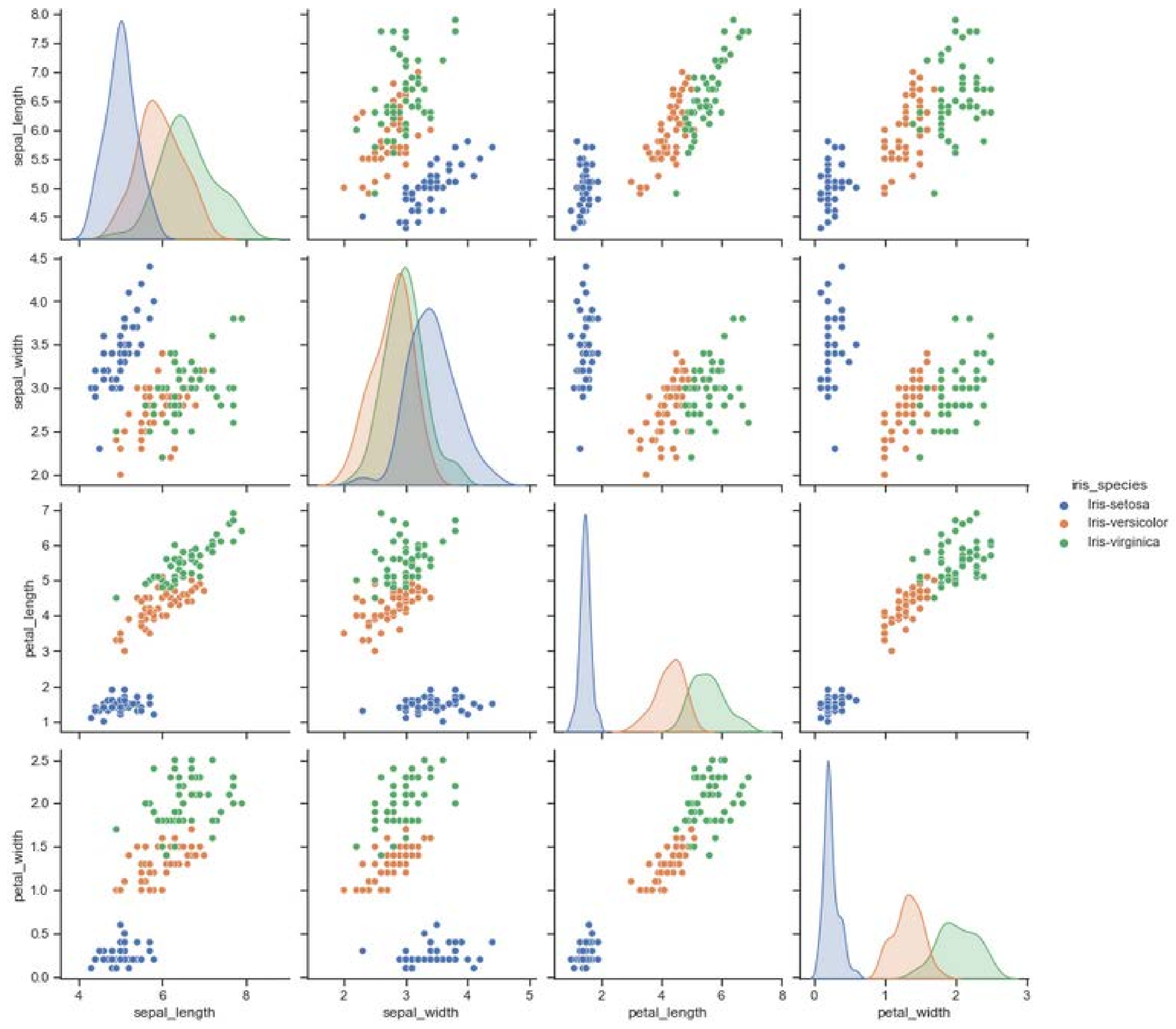


Fig 6

Observation

From the pairplot, I believed that the Iris-setosa specie is separated from the other two species across all feature combinations.

1.7 Use the Scatter diagram to show correlation between the petal widths attribute vs the petal length attribute by using the ***.plot*** extension from ***Pandas*** and ***matplotlib.pyplot*** as shown below [5].

```
df.plot.scatter(x='petal_length', y='petal_width',
                title= "Scatter plot between the petal lengths vs the petal widths ")
plot.show(block=True)
```

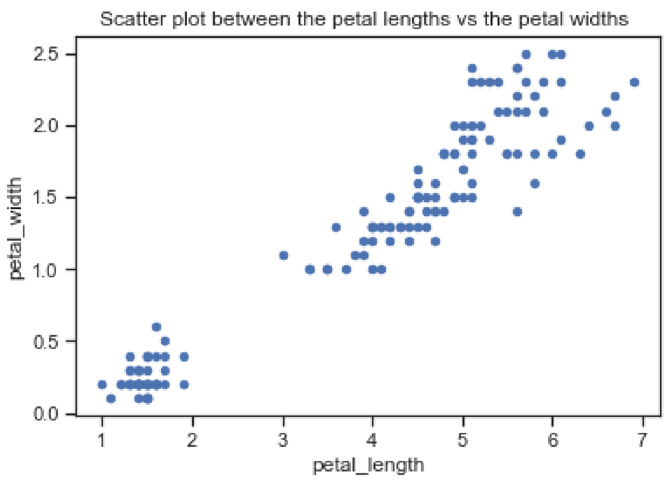


Fig 7

1.8 Show scatter plots of each species separately in the form of three clusters as shown below.

```
sns.FacetGrid(irisdata, hue="iris_species", height=8).map(plot.scatter, "petal_length", "petal_width").add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x264121a7160>
```

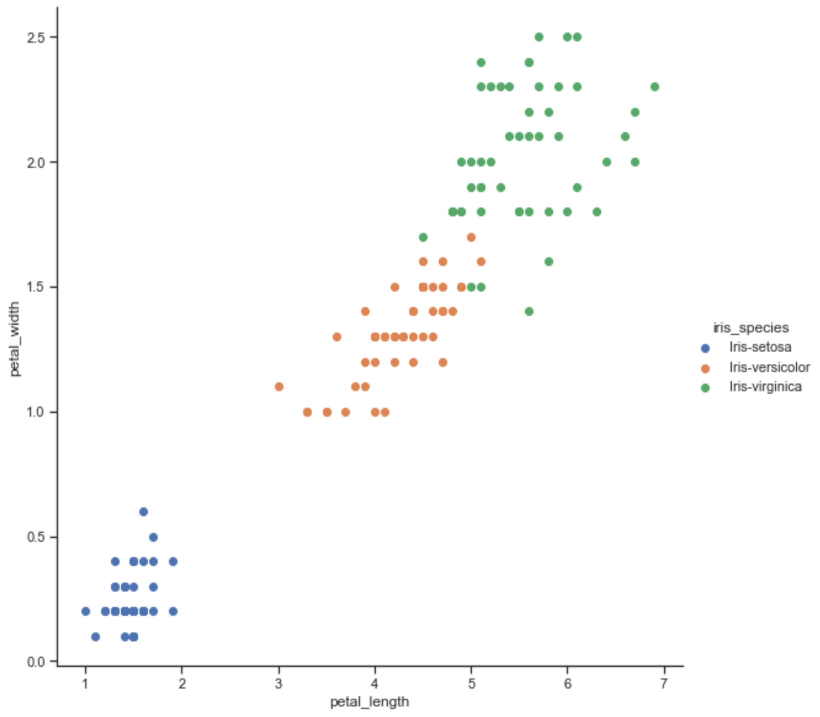


Fig 8

Observation: As expected, Iris-setosa is in a small cluster in the bottom left of the plot because it has smaller mean lengths and widths for both petals and sepals as compared to Iris-versicolor and Iris-virginica.

Step 2 (Plotting single linkage dendrogram to implement the Agglomerative Hierarchical Clustering Algorithm)

Introduction

In this stage, I will use the Agglomerative Hierarchical clustering algorithms, Euclidean distance, and single linkage (minimum distance between any two points) to cluster the given data points by using dendrogram. During this process, I will limit my dataset only to the first 6 rows of sepal widths and sepal length as per given condition.

Clustering Algorithms

Clustering is a technique of grouping similar data points together and the group of similar data points formed is known as a 'Cluster'. Hierarchical clustering is a type of unsupervised machine learning algorithm which is used to cluster unlabelled data points. Like K-means clustering, hierarchical clustering also groups together the data points with similar characteristics [3].

In the hierarchical clustering, data points are clustered using a bottom-up approach starting with individual data points [4].

Algorithm and Flowchart for the Hierarchical Clustering

Following are the steps involved in agglomerative clustering:

1. At the start, assign each data point as one cluster. Therefore, the number of clusters at the start will be K, while K is an integer representing the number of data points.
2. Compute distance matrix by calculating Euclidean distance between each cluster.
3. Form a cluster by joining the two closest data points resulting in K-1 clusters.
4. Form more clusters by joining the two closest clusters resulting in K-2 clusters.
5. Repeat the above three steps until one big cluster is formed.
6. Once single cluster is formed, dendrograms are used to divide into multiple clusters depending upon the problem [3,5].

Flowchart

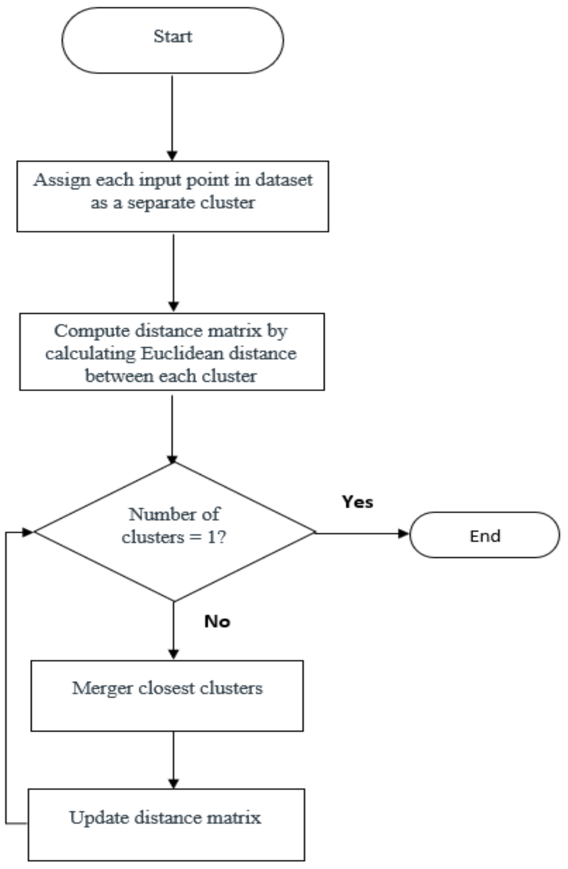
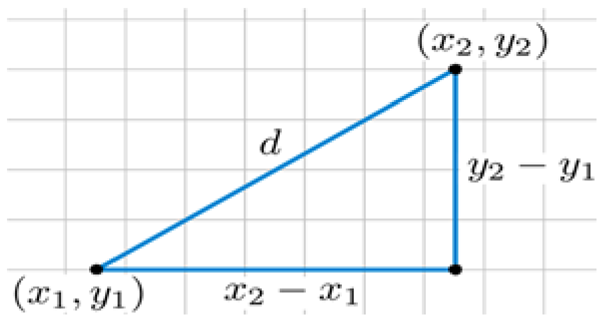

Start
Assign each input point in dataset as a separate cluster
Compute distance matrix by calculating Euclidean distance between each cluster
Number of clusters = 1?
Yes
End
No
Merger closest clusters
Update distance matrix

Fig 9

There are different ways to find distance between the clusters. I have decided using Euclidean distance which is calculated by using the following formula.

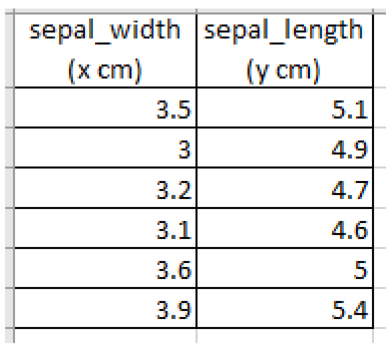


$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Fig 10

Steps:

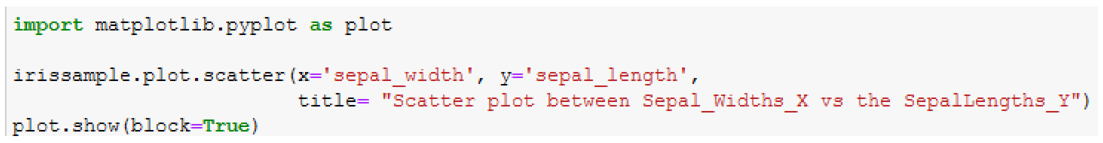
2.1 Produce a sample of 6 points from the given iris data set. By assuming sepal widths as ‘x’ and sepal length as ‘y’ coordinates as shown in Fig 11.



sepal_width (x cm)	sepal_length (y cm)
3.5	5.1
3	4.9
3.2	4.7
3.1	4.6
3.6	5
3.9	5.4

Fig 11

2.2 Draw a scatterplot() to draw each point on a grid and examine the location of each point as shown below [1].



```
import matplotlib.pyplot as plot

irissample.plot.scatter(x='sepal_width', y='sepal_length',
                        title= "Scatter plot between Sepal_Widths_X vs the SepalLengths_Y")
plot.show(block=True)
```

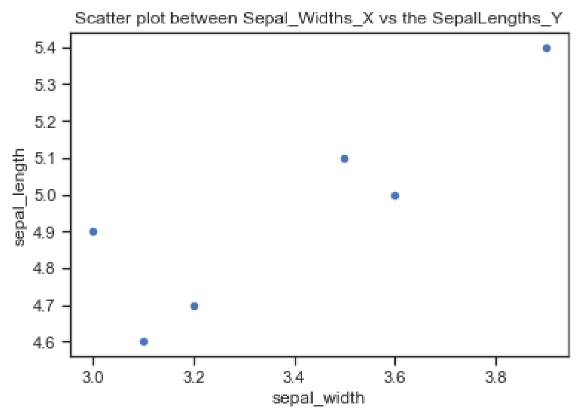


Fig 12

2.3 Follow the three steps as explained in explained the algorithm on Fig 9. Populate the Proximity distance matrix first by using the Euclidean distance formula as shown in Fig 12. Then draw the cluster graph and plot a dendrogram using single linkage to merge two closest clusters together as shown below.

	P_0 (3.5, 5.1)	P_1 (3, 4.9)	P_2 (3.2, 4.7)	P_3 (3.1, 4.6)	P_4 (3.6, 5)	P_5 (3.9, 5.4)
P_0 (3.5, 5.1)	0.00	0.54	0.50	0.64	0.14	0.50
P_1 (3, 4.9)	0.54	0.00	0.28	0.32	0.61	1.03
P_2 (3.2, 4.7)	0.50	0.28	0.00	0.14	0.5	0.99
P_3 (3.1, 4.6)	0.64	0.32	0.14	0.00	0.64	1.13
P_4 (3.6, 5)	0.14	0.61	0.50	0.64	0.00	0.50
P_5 (3.9, 5.4)	0.50	1.03	0.99	1.13	0.50	0.00

Proximity Distance Matrix (Fig 13)

2.4 By using the agglomerative clustering algorithm, lets assign each data point as one cluster. Therefore, the number of clusters at the start will be 6 and I used the scatter plot from Fig 12 to draw the appropriate location of each point as shown below in Fig 14.

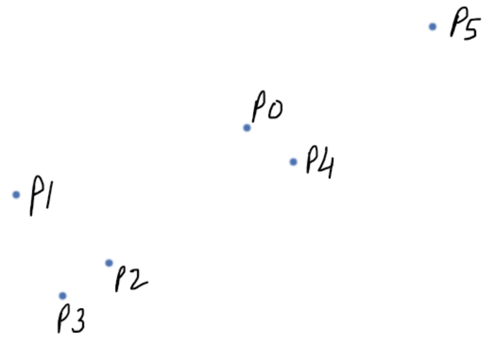
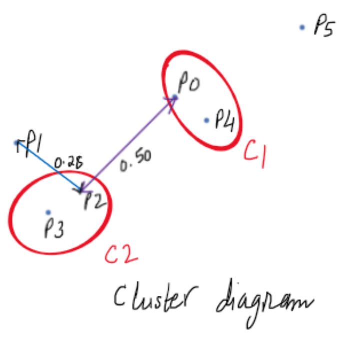


Fig 14

2.5 **EPOCH1:** Form a cluster by joining the two closest data points by using the Proximity distance matrix showing Euclidean distance form Fig 13. As a result, two clusters C1 and C2 will formed as shown below in Fig 15.



	C1	C2	P1	P5
C1	0.00	0.50	0.54	0.50
C2	0.50	0.00	0.28	0.99
P1	0.54	0.28	0.00	1.03
P5	0.50	0.99	1.03	0.00

Proximity distance matrix

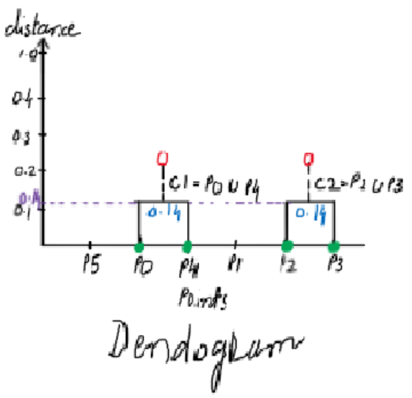
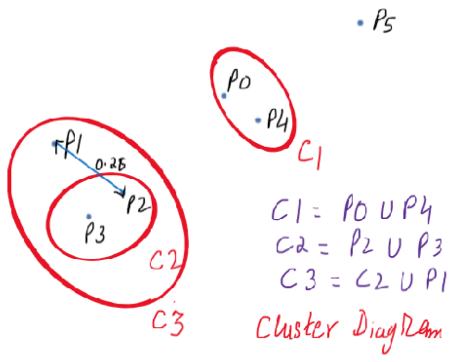


Fig 15

2.6 **EPOCH2:** Repeat the above step. First, compute distance matrix by calculating Euclidean distance between each cluster. Proximity of two clusters is based on the two closest points in the different clusters which is determined by only one pair of points or by one(single) link in the cluster diagram as shown in Fig 15.



	C1	C3	P5
C1	0.00	0.50	0.50
C3	0.50	0.00	0.99
P5	0.50	0.99	0.00

Proximity distance matrix

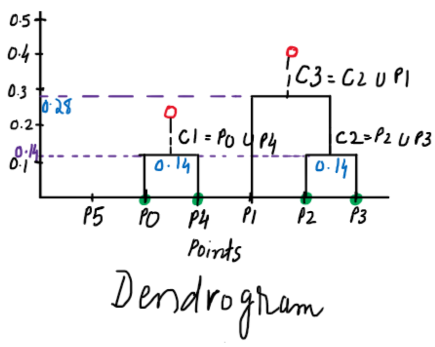
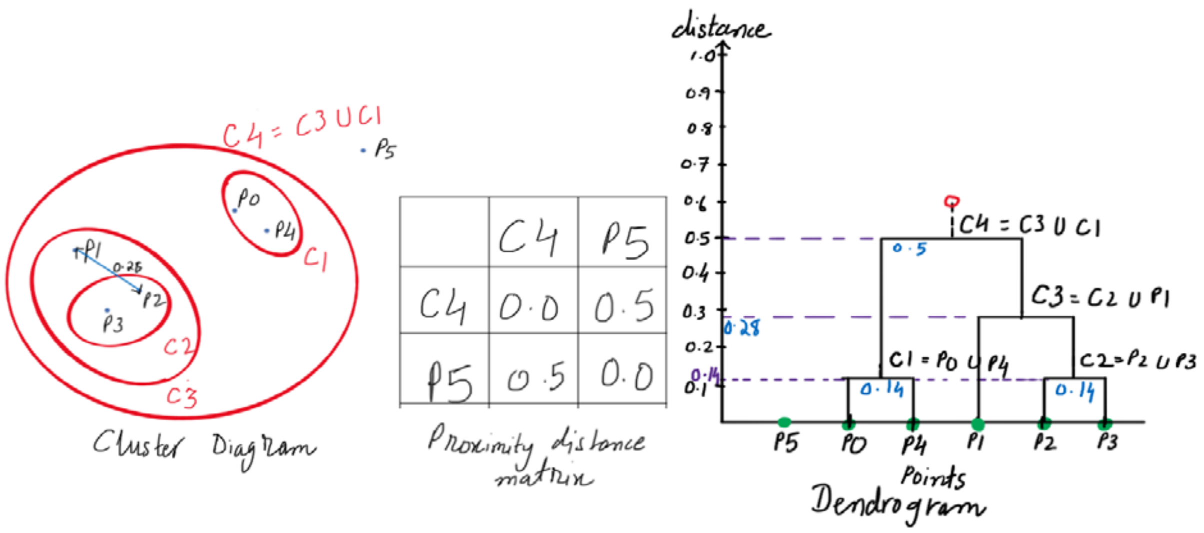


Fig 16

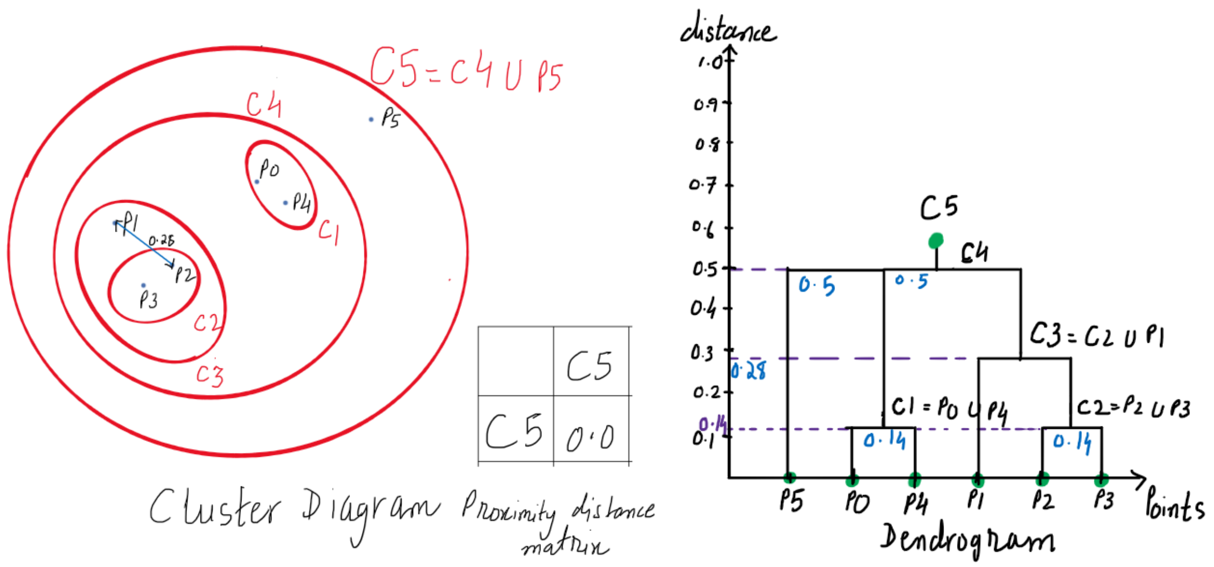
2.7 **EPOCH3**: Repeat the same three steps. Form more clusters by using a single link, upgrade the cluster diagram, distance matrix and Dendrogram as shown below.



	C4	P5
C4	0.0	0.5
P5	0.5	0.0

Fig 17

2.8 **EPOCH 4:** Repeat the above three steps and we will finally get a one big cluster. This is the last step.



	C5
C5	0.0

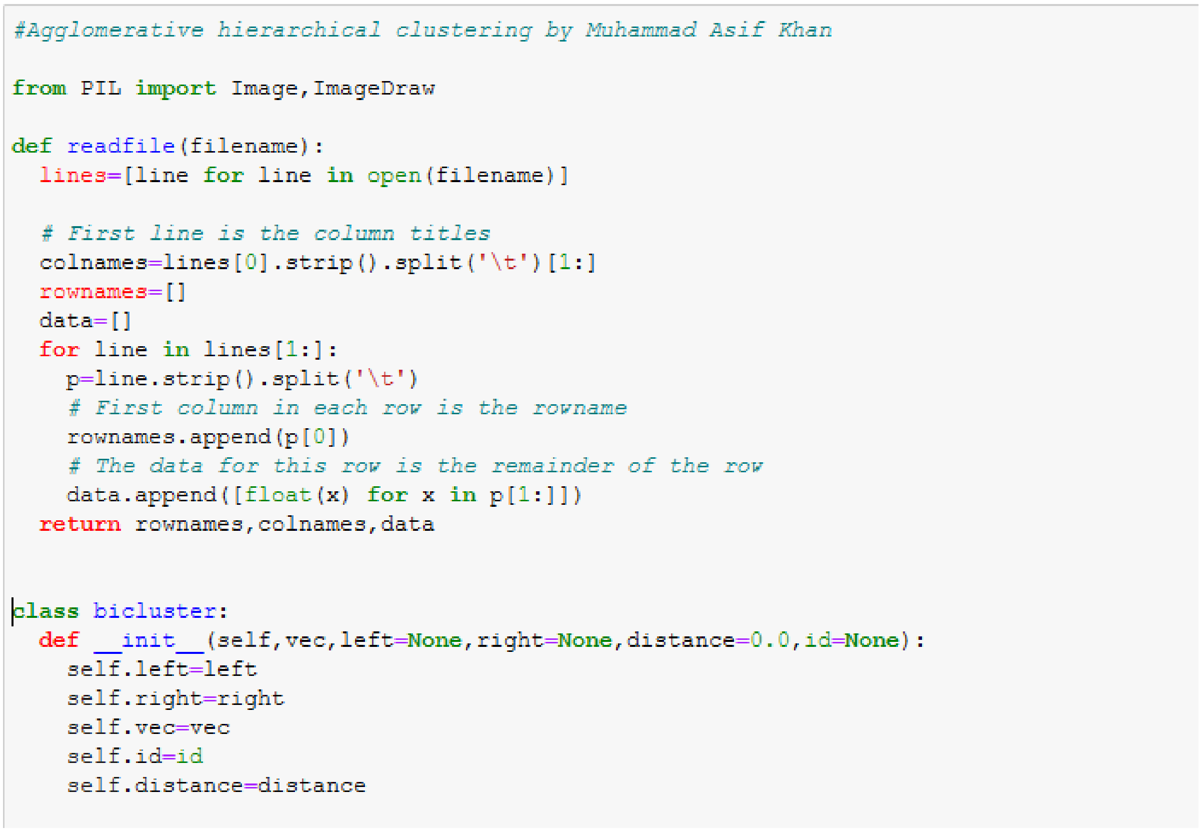
Fig 18

Muhammad Asif Khan

S19004470

Using Python code to implement the Agglomerative Hierarchical Clustering Algorithm

By using the following code in Python, you can implement the Agglomerative Hierarchical Clustering in pure without using any data analysis library modules.



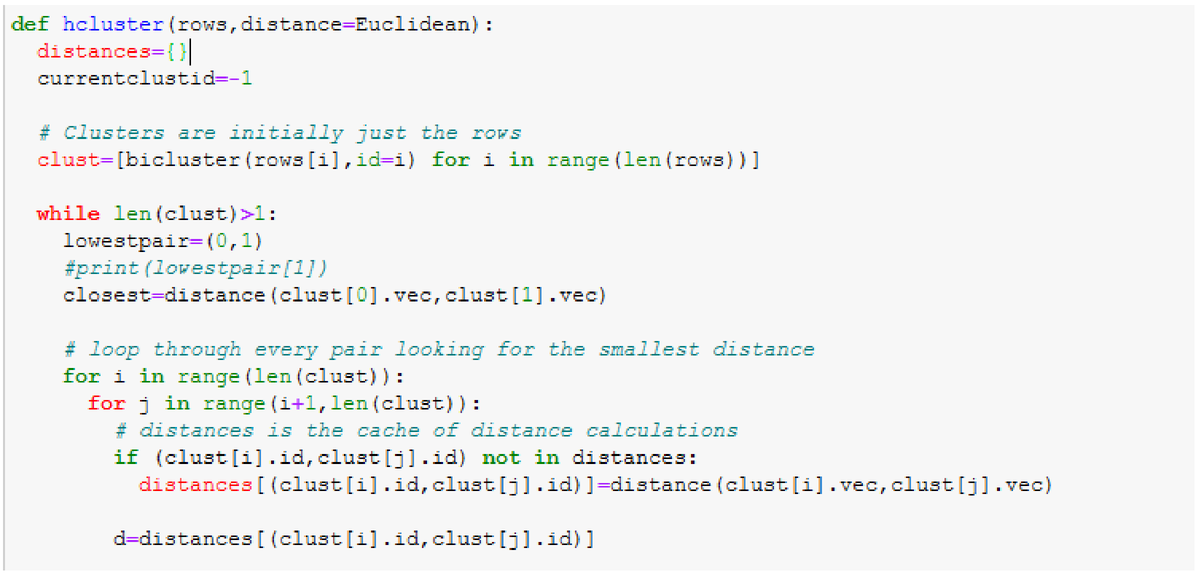
```
#Agglomerative hierarchical clustering by Muhammad Asif Khan

from PIL import Image,ImageDraw

def readfile(filename):
  lines=[line for line in open(filename)]

  # First line is the column titles
  colnames=lines[0].strip().split('\t')[1:]
  rownames=[]
  data=[]
  for line in lines[1:]:
    p=line.strip().split('\t')
    # First column in each row is the rowname
    rownames.append(p[0])
    # The data for this row is the remainder of the row
    data.append([float(x) for x in p[1:]])
  return rownames,colnames,data


class bicluster:
  def __init__(self,vec,left=None,right=None,distance=0.0,id=None):
    self.left=left
    self.right=right
    self.vec=vec
    self.id=id
    self.distance=distance
```



```
def hcluster(rows,distance=Euclidean):
  distances={}
  currentclustid=-1

  # Clusters are initially just the rows
  clust=[bicluster(rows[i],id=i) for i in range(len(rows))]

  while len(clust)>1:
    lowestpair=(0,1)
    #print(lowestpair[1])
    closest=distance(clust[0].vec,clust[1].vec)

    # loop through every pair looking for the smallest distance
    for i in range(len(clust)):
      for j in range(i+1,len(clust)):
        # distances is the cache of distance calculations
        if (clust[i].id,clust[j].id) not in distances:
          distances[(clust[i].id,clust[j].id)]=distance(clust[i].vec,clust[j].vec)

        d=distances[(clust[i].id,clust[j].id)]
        if d<closest:
          closest=d
          lowestpair=(i,j)

    # calculate the average of the two clusters
    mergevec=[
    (clust[lowestpair[0]].vec[i]+clust[lowestpair[1]].vec[i])/2.0
    for i in range(len(clust[0].vec))]

    # create the new cluster
    newcluster=bicluster(mergevec,left=clust[lowestpair[0]],
                         right=clust[lowestpair[1]],
                         distance=closest,id=currentclustid)

    # cluster ids that weren't in the original set are negative
    currentclustid-=1
    del clust[lowestpair[1]]
    del clust[lowestpair[0]]
    clust.append(newcluster)

  return clust[0]

def printclust(clust,labels=None,n=0):
  # indent to make a hierarchy layout
  for i in range(n): print (' '),
  if clust.id<0:
    # negative id means that this is branch
    print ('-')
  else:
    # positive id means that this is an endpoint
    if labels==None: print (clust.id)
    else: print ("this",labels[clust.id])

  # now print the right and left branches
  if clust.left!=None: printclust(clust.left,labels=labels,n=n+1)
  if clust.right!=None: printclust(clust.right,labels=labels,n=n+1)

Flowernames,species,sample = readfile('IrisSample.txt')
clust = hcluster(data)
printclust(clust,labels = Flowernames)
```

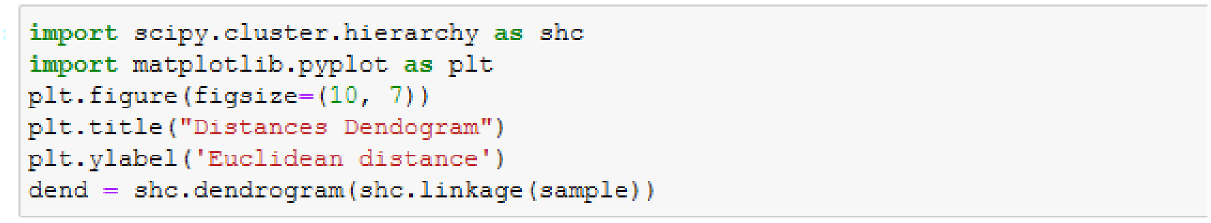
```
def getheight(clust):
  # Is this an endpoint? Then the height is just 1
  if clust.left==None and clust.right==None: return 1

  # Otherwise the height is the same of the heights of
  # each branch
  return getheight(clust.left)+getheight(clust.right)

def getdepth(clust):
  # The distance of an endpoint is 0.0
  if clust.left==None and clust.right==None: return 0

  # The distance of a branch is the greater of its two sides
  # plus its own distance
  return max(getdepth(clust.left),getdepth(clust.right))+clust.distance
```

I also used the following Python code to verify the correct implementation of Agglomerative Hierarchical Clustering Algorithm and check dendrogram for the given six points as shown below in Fig 19 [4,5].



```
import scipy.cluster.hierarchy as shc
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 7))
plt.title("Distances Dendogram")
plt.ylabel('Euclidean distance')
dend = shc.dendrogram(shc.linkage(sample))
```

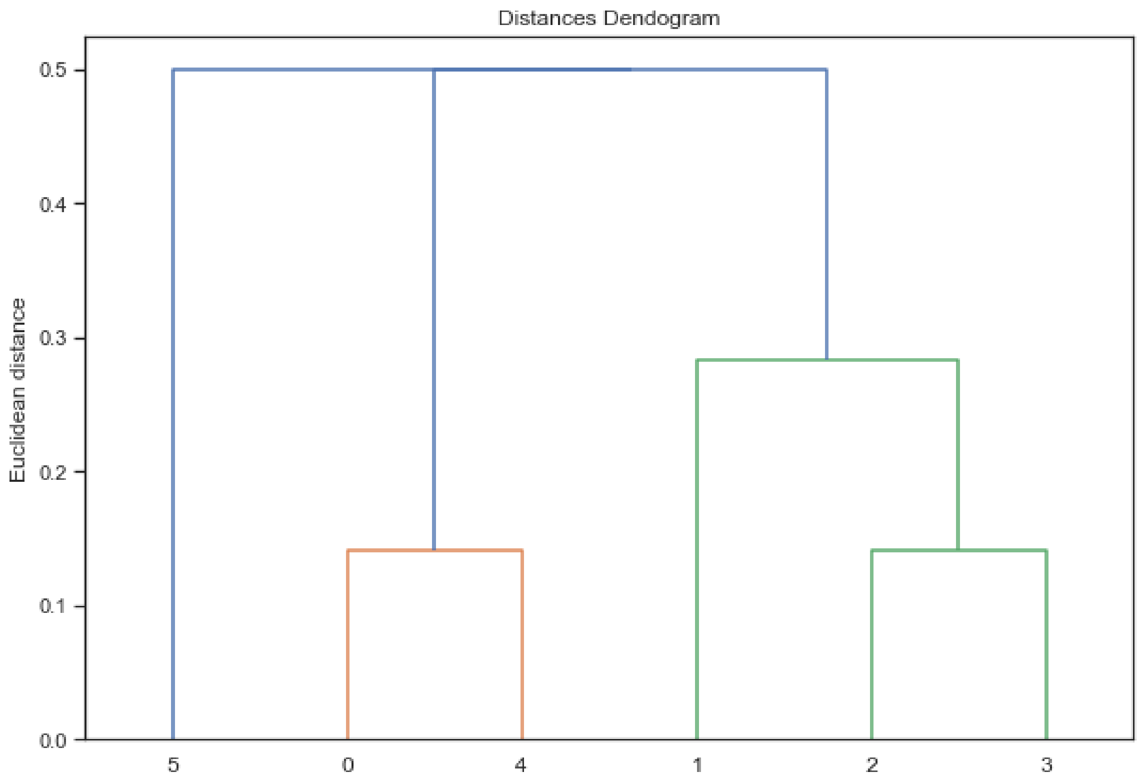


Fig 19

Conclusion:

By comparing results of hand-drawn implementation (Fig 18) and Python code (Fig 19), dendrograms for the given six points by using Agglomerative Hierarchical Clustering algorithm are the same.

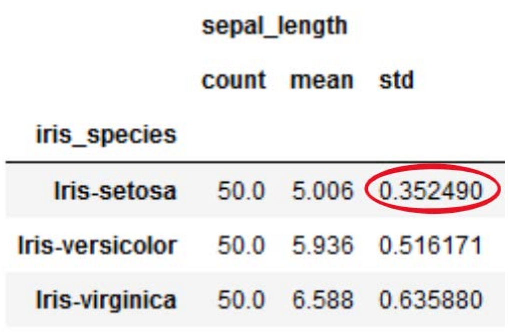
Step 3.

Answer the following question. How do you interpret your dendrogram? how many species does it show?

Interpretation

I think that a dendrogram shows the hierarchy relationship between data items in the given sample. In our case, the final height of the last cluster is 0.5. It shows that all six data points in the given sample will have an average shortest Euclidean distance of 0.5 cm from each other.

By using groupby(["iris_species"]).describe() method on the given data set, I got some key statistical characteristics of each iris species like mean, standard deviation and quartiles as shown below.



	sepal_length		
	count	mean	std
iris_species			
Iris-setosa	50.0	5.006	0.352490
Iris-versicolor	50.0	5.936	0.516171
Iris-virginica	50.0	6.588	0.635880

By looking at the standard deviation (std) values, I noticed that only Iris-setosa has a std less than 0.5.

As standard deviation is a measure of the amount of variation or dispersion of a set of values in the given sample, I believe that the final height (d = 0.5) of the last cluster in the dendrogram as shown in Fig 18 represents the standard deviation of the given data set if data values are spread out uniformly in the population.

Therefore, all the six points in the given sample represents Iris-setosa specie which could be verified by looking at the population data.

Furthermore, the final height in dendrogram is cumulative and could be used to group similar data points together in unsupervised machine learning.

References:

[1] Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython By Wes McKinne

[2] Ceder, N (2018) The Quick Python Book. Third Edition, Manning Publications Co, Shelter Island

[3] EMC Education Services, Data science and big data analytics: Discovering, analyzing, visualizing and presenting data. Nashville, TN, USA: John Wiley & Sons, 2015

[4] https://www.vshsolutions.com/blogs/clustering-iris-plant-data-using-hierarchical-clustering/

[5] https://www.datasciencelearner.com/how-to-do-hierarchical-clustering-in-python/