Practical #6: Cluster Analysis with Python

Objectives:

Consolidate the understanding of the basic concepts and methods of k-means, hierarchical and density-based clustering using Python.

Submission:

- *What to submit*: The saved document (**.html**) containing **PART 2 and PART 3** of the practical.
- *Deadline of the submission*: 11:59PM (Adelaide Time), Tuesday of Week 9.
- *Submission link*: "**Submission Link of Prac #6"** in **Week 8 section** on Learnonline course site.
- *Marks*: Prac#6 (part of the ongoing assessment of the course) is worth 2% of the total marks of the course.

Instructions:

PART1: *K*-means Clustering

The *k*-means clustering algorithm represents each cluster by its corresponding cluster centroid. It would iteratively partition the input data into k disjoint clusters by forming k clusters (assigning each instance to its nearest centroid) and recomputing the centroid of each cluster.

Here, we perform *k*-means clustering using Python on a toy example of **movie ratings** dataset, including:

ratings =
[['john',5,5,2,1], ['mary',4,5,3,2], ['bob',4,4,4,3], ['lisa',2,2,4,5], ['lee',1,2,3,4], ['harry',2,1,5,5]]

titles =
['user', 'Jaws', 'Star Wars', 'Exorcist', 'Omen']

```
import pandas as pd

ratings = [['john',5,5,2,1],['mary',4,5,3,2],['bob',4,4,4,3],['lisa',2,2,4,5],['lee',1,2,3,4],['harry',2,1,5,5]]
titles = ['user','Jaws','Star Wars','Exorcist','Omen']
movies = pd.DataFrame(ratings,columns=titles)
movies
```

	user	Jaws	Star Wars	Exorcist	Omen
0	john	5	5	2	1
1	mary	4	5	3	2
2	bob	4	4	4	3
3	lisa	2	2	4	5
4	lee	1	2	3	4
5	harry	2	1	5	5

In this example dataset, the first 3 users (e.g., john, mary, and bob) liked (e.g., high ratings) action movies (Jaws and Star Wars), while the last 3 users enjoyed horror movies (Exorcist and Omen). Our goal is to apply *k*-means clustering on the users to identify groups of users with similar movie preferences.

The example below shows how to apply *k*-means clustering (**with k=2**; i.e., n_clusters=2) to the movie ratings data. We must **remove** (using **drop** function) the "user" column first before applying the clustering algorithm (using **KMeans** function in the cluster of **sklearn** library package). The cluster assignment for each user is displayed as a dataframe object.

```
from sklearn import cluster

data = movies.drop('user',axis=1)
k_means = cluster.KMeans(n_clusters=2, max_iter=50, random_state=1)
k_means.fit(data)
labels = k_means.labels_
pd.DataFrame(labels, index=movies.user, columns=['Cluster ID'])
```

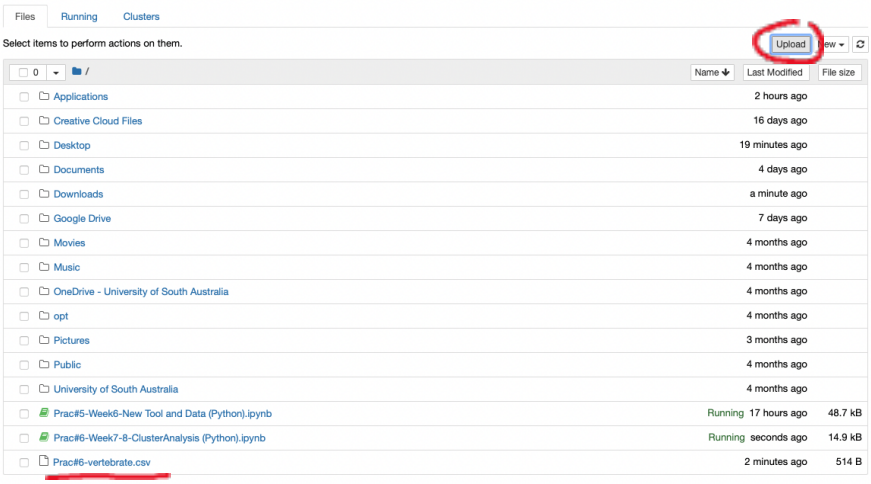
user	Cluster ID
john	1
mary	1
bob	1
lisa	0
lee	0
harry	0

The *k*-means clustering algorithm assigns the first three users to one cluster and the last three users to the second cluster. The results are consistent with our expectation. We can also display the centroid for each of the two clusters.

PART2: Hierarchical Clustering

Here are some examples of applying hierarchical clustering to the **vertebrate** dataset by using two hierarchical clustering algorithms provided by the Python **scipy** library package: single link (MIN) and complete link (MAX).

Note: you need to download the dataset (**Prac6-data-vertebrate.csv**) from Week 8 section on Learnonline course site and upload it to the Jupyter Notebook home folder as shown below.

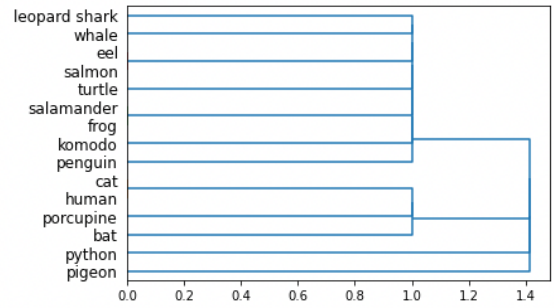


```
import pandas as pd

data = pd.read_csv('Prac#6-vertebrate.csv',header='infer')
data
```

	Name	Warm-blooded	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hibernates	Class
0	human	1	1	0	0	1	0	mammals
1	python	0	0	0	0	0	1	reptiles
2	salmon	0	0	1	0	0	0	fishes
3	whale	1	1	1	0	0	0	mammals
4	frog	0	0	1	0	1	1	amphibians
5	komodo	0	0	0	0	1	0	reptiles
6	bat	1	1	0	1	1	1	mammals
7	pigeon	1	0	0	1	1	0	birds
8	cat	1	1	0	0	1	0	mammals
9	leopard shark	0	1	1	0	0	0	fishes
10	turtle	0	0	1	0	1	0	reptiles
11	penguin	1	0	1	0	1	0	birds
12	porcupine	1	1	0	0	1	1	mammals
13	eel	0	0	1	0	0	0	fishes
14	salamander	0	0	1	0	1	1	amphibian

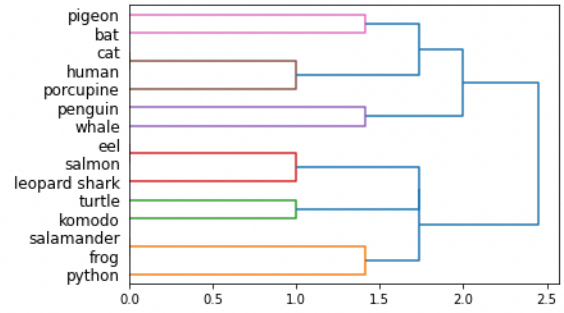
Single Link (MIN)

```
from scipy.cluster import hierarchy
import matplotlib.pyplot as plt
%matplotlib inline

names = data['Name']
Y = data['Class']
X = data.drop(['Name','Class'],axis=1)

Z = hierarchy.linkage(X.to_numpy(), 'single')
dn = hierarchy.dendrogram(Z,labels=names.tolist(),orientation='right')
```



Complete Link (MAX)

```
Z = hierarchy.linkage(X.to_numpy(), 'complete') # change the condition to 'complete'
dn = hierarchy.dendrogram(Z,labels=names.tolist(),orientation='right')
```



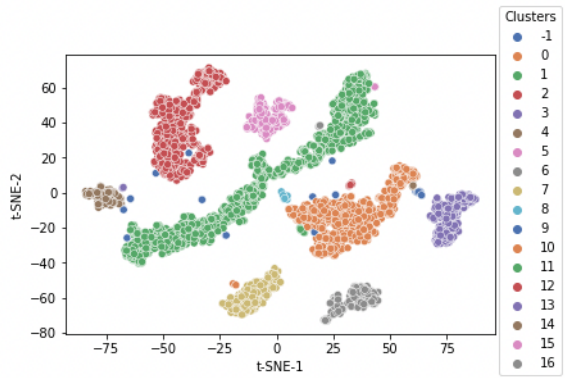
PART3: Density-Based Clustering

Density-based clustering identifies the individual clusters as high-density regions that are separated by regions of low density. **DBSCAN** is one of the most popular density-based clustering algorithms, where data points are classified into 3 types (core points, border points, and noise points) based on the density of their local neighbourhood (defined according to 2 parameters: radius of neighbourhood size and the minimum number of points in the neighborhood).

Here, we will use a gene expression dataset with two dimensions (t-SNE-1 and t-SNE-2) originally from the GitHub link https://reneshbedre.github.io/assets/posts/tsne/tsne_scores.csv. The example code shown below will load and plot the distribution of the data.

Of note, we apply the DBSCAN clustering algorithm to the dataset by setting the neighborhood radius (eps) to 3 and the minimum number of points (min_samples) to 4, and plot up to 16 clusters finally as shown below.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN

df = pd.read_csv("https://reneshbedre.github.io/assets/posts/tsne/tsne_scores.csv")

clusters = DBSCAN(eps=3, min_samples=4).fit(df)

p = sns.scatterplot(data=df, x="t-SNE-1", y="t-SNE-2", hue=clusters.labels_, legend="full", palette="deep")
sns.move_legend(p, "upper right", bbox_to_anchor=(1.17, 1.2), title='Clusters')
plt.show()
```



Save the .html document in PART 2 and 3 as your submission by following the instruction given on page 1 of the practical document.