HW 4

Mohammad Hosein Ashoori - 97149068

In [ ]:

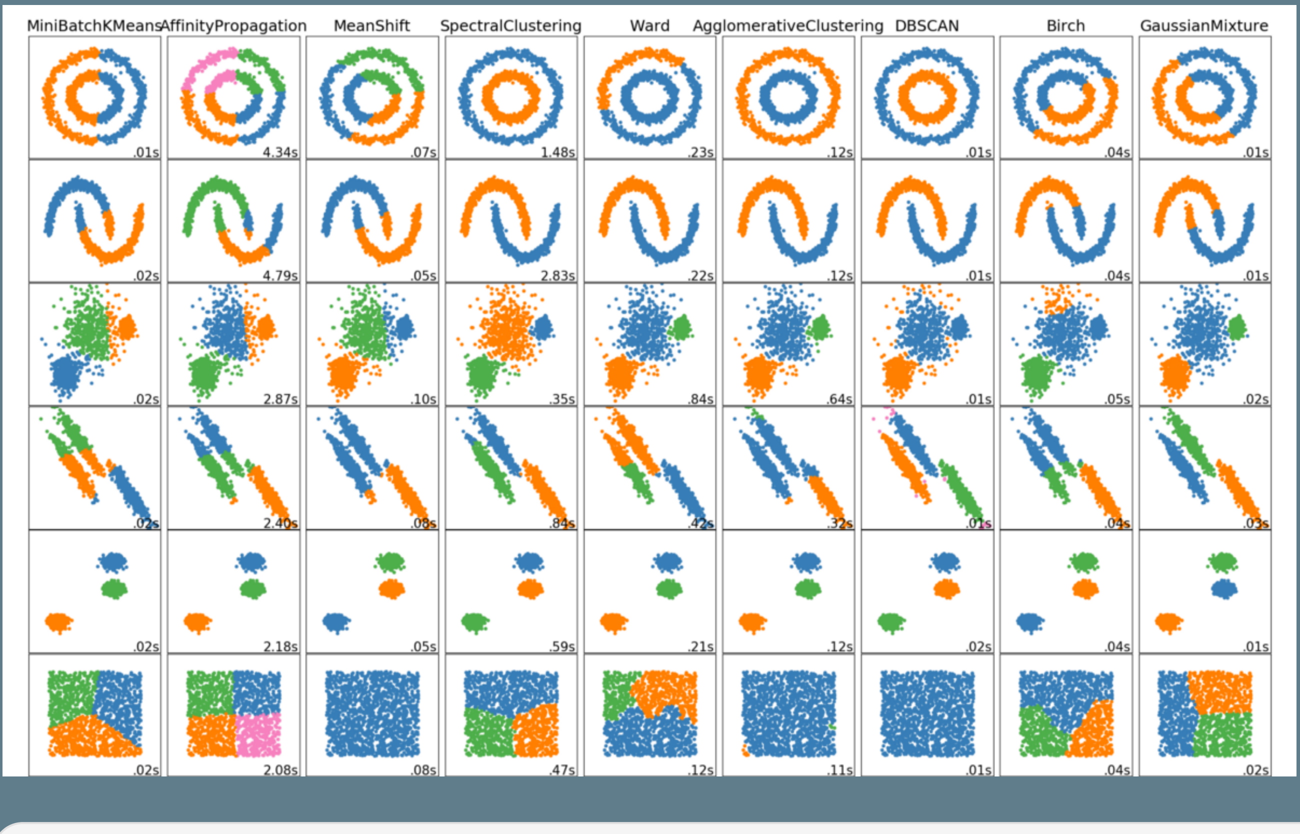
```python
import pandas as pd
from sklearn import preprocessing, decomposition
import scipy.stats as stats
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
% matplotlib inline
matplotlib.style.use('fivethirtyeight')
```

```
UsageError: Line magic function `%` not found.
```

In [ ]:

```python
import random
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
```

Difference Between Clustring Methods



In [ ]:

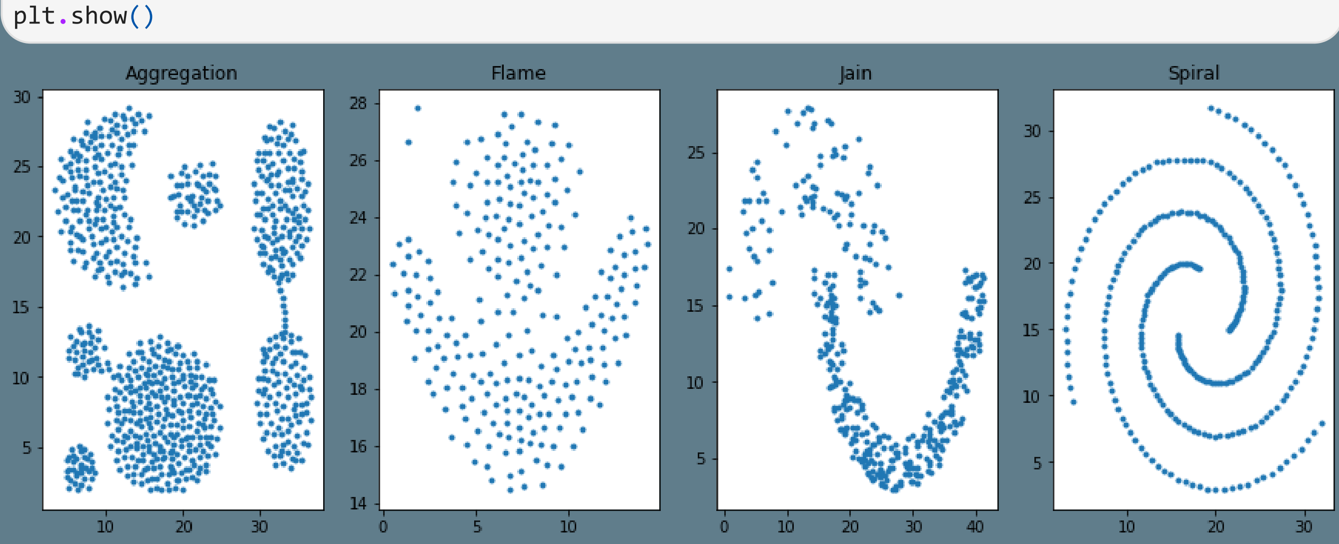
```python
AggregationFile = np.genfromtxt("files/Aggregation.txt", delimiter="\t",dtype=np.float64)
FlameFile = np.genfromtxt("files/flame.txt", delimiter="\t",dtype=np.float64)
JainFile = np.genfromtxt("files/jain.txt", delimiter="\t",dtype=np.float64)
SpiralFile = np.genfromtxt("files/spiral.txt", delimiter="\t",dtype=np.float64)
```

1 - Plot Raw Data

In [ ]:

```python
fig, axs = plt.subplots(nrows=1, ncols=4, figsize=(15, 5))
axs[0].scatter(AggregationFile[:, 0], AggregationFile[:, 1], marker='.')
axs[1].scatter(FlameFile[:, 0], FlameFile[:, 1], marker='.')
```

```
axs[2].scatter(JainFile[:, 0], JainFile[:, 1], marker='.')
axs[3].scatter(SpiralFile[:, 0], SpiralFile[:, 1], marker='.')

for ax,name in zip(axs,["Aggregation","Flame","Jain","Spiral"]):
    ax.set_title(name)
plt.scatter
plt.show()
```



2 - KMeans

In [ ]:

```
KMAggregation = KMeans(init = "k-means++", n_clusters = len(set(AggregationFile[:,2])), n_ini
KMFlame = KMeans(init = "k-means++", n_clusters = len(set(FlameFile[:,2])), n_init = 12)
KMJain = KMeans(init = "k-means++", n_clusters = len(set(JainFile[:,2])), n_init = 12)
KMSpiral = KMeans(init = "k-means++", n_clusters = len(set(SpiralFile[:,2])), n_init = 12)
```

In [ ]:

```
KMAggregation.fit(AggregationFile[:,:2])
KMJain.fit(JainFile[:,:2])
KMFlame.fit(FlameFile[:,:2])
KMSpiral.fit(SpiralFile[:,:2])
```

Out[ ]:

```
KMeans(n_clusters=3, n_init=12)
```

In [ ]:

```
KMAggregationFile_labels = KMAggregation.labels_
KMJainFile_labels = KMJain.labels_
KMFlameFile_labels = KMFlame.labels_
KMSpiralFile_labels = KMSpiral.labels_

KMAggregationFile_cluster_centers = KMAggregation.cluster_centers_
KMJainFile_cluster_centers = KMJain.cluster_centers_
KMFlameFile_cluster_centers = KMFlame.cluster_centers_
KMSpiralFile_cluster_centers = KMSpiral.cluster_centers_

KMAggregationFile_cluster_centers
```

Out[ ]:

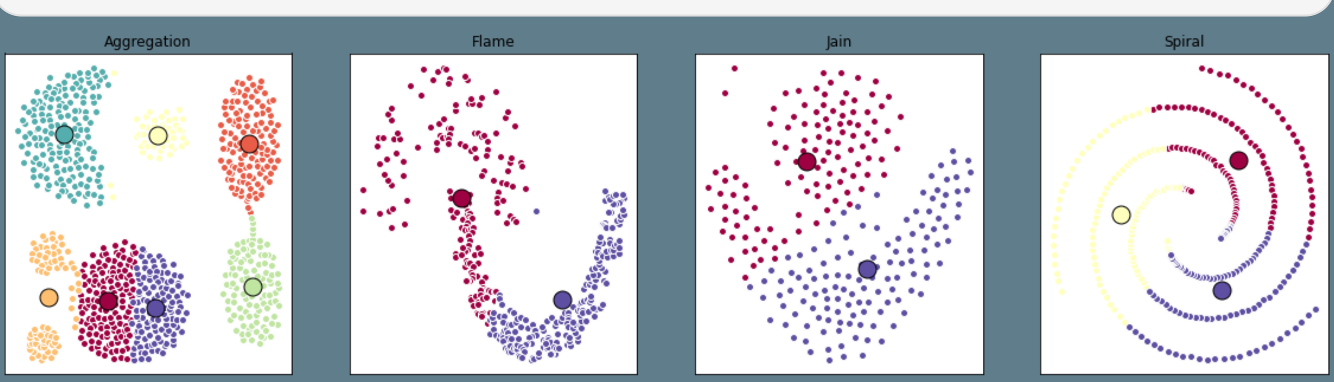
```
array([[14.89751773,  7.41843972],
       [32.69453125, 22.13789062],
       [ 7.36858974,  7.75705128],
       [21.16041667, 22.89895833],
       [33.14278846,  8.79375   ],
       [ 9.25928144, 22.98113772],
       [20.89836066,  6.81229508]])
```

In [ ]:

```
def pltmaker(X, KMeanTrained:KMeans = None ,parent_axs=None):

    if KMeanTrained == None :
        k_means_cluster_centers = X[:,:2]
        k_means_labels = X[:,2]
    else:
        k_means_cluster_centers = KMeanTrained.cluster_centers_
        k_means_labels = KMeanTrained.labels_
    # Colors uses a color map, which will produce an array of colors based on
    # the number of labels there are. We use set(k_means_labels) to get the
    # unique labels.
    colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k_means_labels))))


    if parent_axs :
        ax=parent_axs
    else :
        # Initialize the plot with the specified dimensions.
        fig = plt.figure(figsize=(20, 15))
        # Create a plot
        ax = fig.add_subplot(1, 1, 1)

    # For loop that plots the data points and centroids.
    # k will range from 0-3, which will match the possible clusters that each
    # data point is in.
    for k, col in zip(range(len(k_means_labels)), colors):

        # Create a list of all data points, where the data points that are
        # in the cluster (ex. cluster 0) are labeled as true, else they are
        # labeled as false.
        my_members = (k_means_labels == k)

        # Define the centroid, or cluster center.
        cluster_center = k_means_cluster_centers[k]

        # Plots the datapoints with color col.
        ax.plot(X[my_members, 0], X[my_members, 1], 'w', markerfacecolor=col, marker='.' , ma
        # Plots the centroids with specified color, but with a darker outline
        if KMeanTrained != None :
            ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,  markered

    # Title of the plot
    ax.set_title('KMeans')

    # Remove x-axis ticks
    ax.set_xticks(())

    # Remove y-axis ticks
    ax.set_yticks(())

    # Show the plot
    if parent_axs:
        return ax
    plt.show()
```

In [ ]:

```
fig, axs = plt.subplots(nrows=1, ncols=4, figsize=(20, 5))
axs[0] = pltmaker(AggregationFile,KMAggregation,axs[0])
axs[1] = pltmaker(JainFile,KMJain,axs[1])
axs[2] = pltmaker(FlameFile,KMFlame,axs[2])
axs[3] = pltmaker(SpiralFile,KMSpiral,axs[3])
for ax,name in zip(axs,["Aggregation","Flame","Jain","Spiral"]):
    ax.set_title(name)
```



3 - Agglomerative Clustering

In [ ]:

```
from scipy import ndimage
from scipy.cluster import hierarchy
from scipy.spatial import distance_matrix
from sklearn import manifold, datasets
from sklearn.cluster import AgglomerativeClustering
```

In [ ]:

```
AgglomAggregation = AgglomerativeClustering(n_clusters = len(set(AggregationFile[:,2])), link
AgglomFlame = AgglomerativeClustering(n_clusters = len(set(FlameFile[:,2])), linkage = 'avera
AgglomJain = AgglomerativeClustering(n_clusters = len(set(JainFile[:,2])), linkage = 'average
AgglomSpiral = AgglomerativeClustering(n_clusters = len(set(SpiralFile[:,2])), linkage = 'ave



AgglomAggregation.fit(AggregationFile[:,:2],AggregationFile[:,2])
AgglomFlame.fit(FlameFile[:,:2],FlameFile[:,2])
AgglomJain.fit(JainFile[:,:2],JainFile[:,2])
AgglomSpiral.fit(SpiralFile[:,:2],SpiralFile[:,2])
```

Out[ ]:

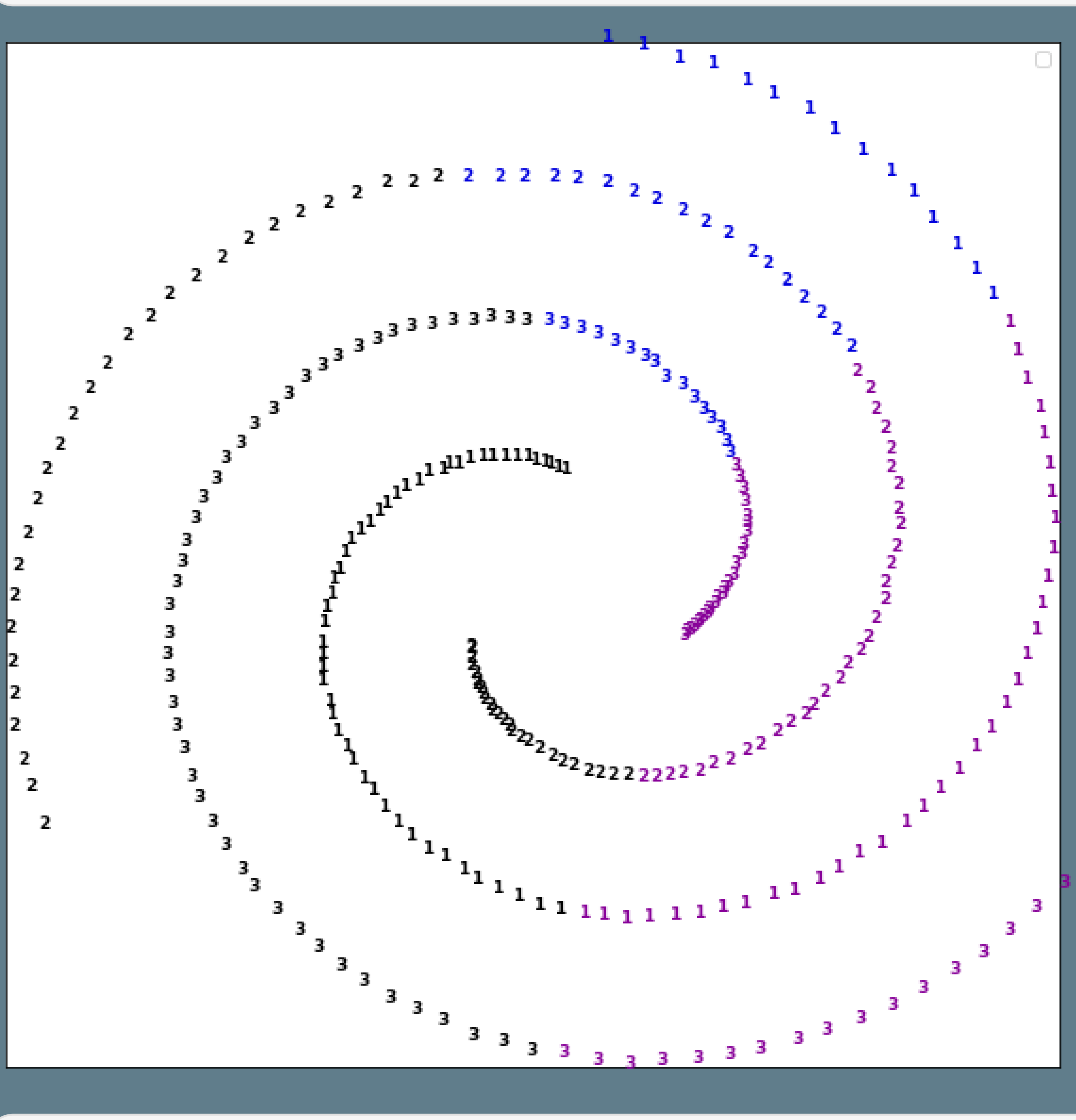
```
AgglomerativeClustering(linkage='average', n_clusters=3)
```

In [ ]:

```
def agglomCustring(data:np.array, agglom:AgglomerativeClustering,parent_axs=None):
    # Create a figure of size 6 inches by 4 inches.
    if parent_axs :
        ax=parent_axs
    else :
        # Initialize the plot with the specified dimensions.
        fig = plt.figure(figsize=(10, 10))
        # Create a plot
        ax = plt
    X1 = data[:,:2]
    y1 = data[:,2]
    # These two lines of code are used to scale the data points down,
    # Or else the data points will be scattered very far apart.

    # Create a minimum and maximum range of X1.
    x_min, x_max = np.min(X1, axis=0), np.max(X1, axis=0)

    # Get the average distance for X1.
    X1 = (X1 - x_min) / (x_max - x_min)

    # This loop displays all of the datapoints.
    for i in range(X1.shape[0]):
        # Replace the data points with their respective cluster value
        # (ex. 0) and is color coded with a colormap (plt.cm.spectral)
        # ax.plot(X1[i:, 0], X1[i:, 1], 'w',
        #  markerfacecolor=plt.cm.nipy_spectral(y1[i] / 10),
        #  marker='.' , markersize=15)
        ax.text(X1[i, 0], X1[i, 1], str(int(y1[i])),
                color=plt.cm.nipy_spectral(agglom.labels_[i] /10 ),
                fontdict={'weight': 'bold', 'size': 9},
                )

    if parent_axs :
        # Remove the x ticks, y ticks, x and y axis
        ax.set_xticks([])
        ax.set_yticks([])
    else :
        plt.xticks([])
        plt.yticks([])
    #ax.axis('off')

    ax.legend(['First line', 'Second line'])
        # Show the plot
    if parent_axs:
        return ax
    # Display the plot of the original data before clustering
    # Display the plot
    ax.show()
```

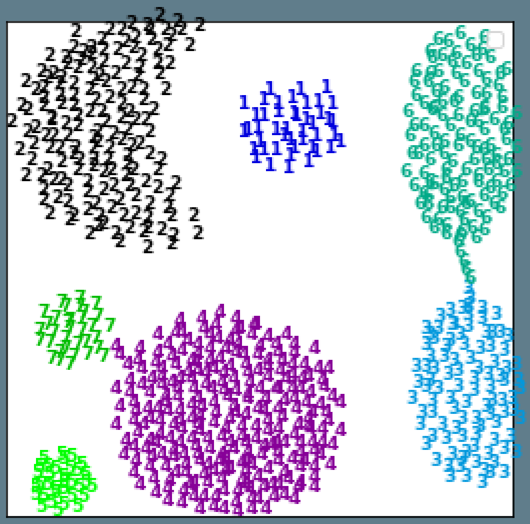
In [ ]:

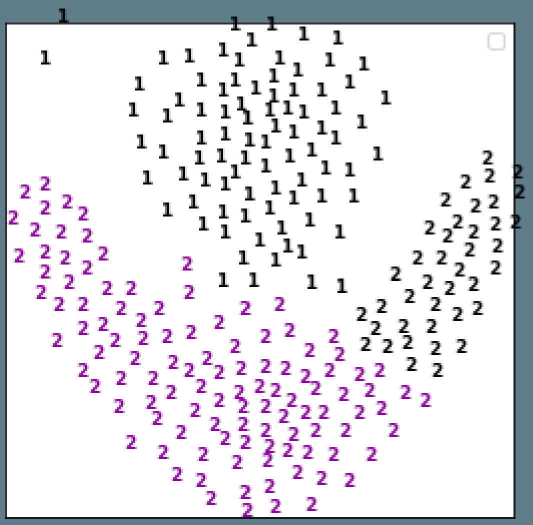
```
# agglomCustring(data=AggregationFile,agglom=AgglomAggregation,)
# agglomCustring(data=FlameFile,agglom=AgglomFlame,)
# agglomCustring(data=JainFile,agglom=AgglomJain,)
agglomCustring(data=SpiralFile,agglom=AgglomSpiral)
```

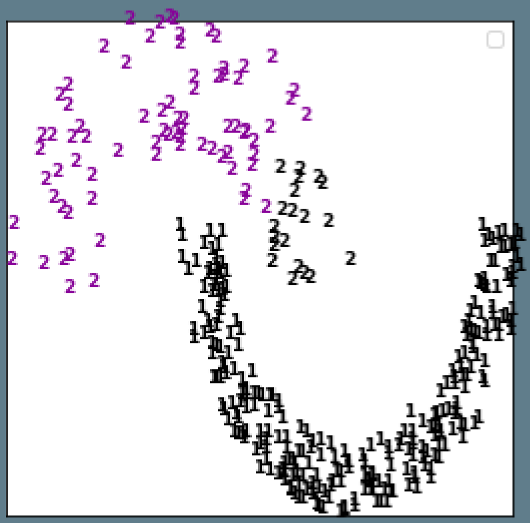


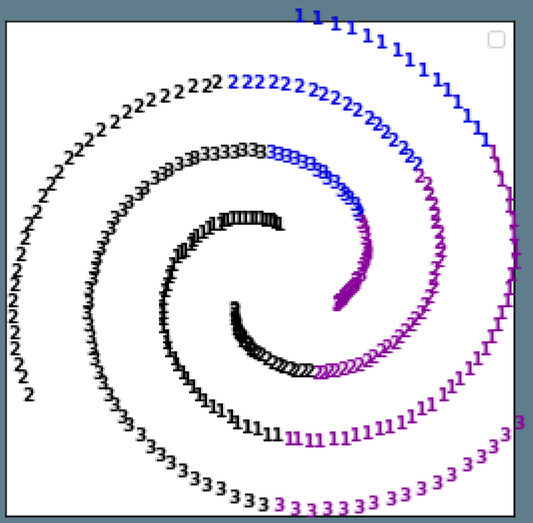
In [ ]:

```
fig, ((ax1,ax2),(ax3,ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))
ax1 = agglomCustring(data=AggregationFile,agglom=AgglomAggregation,parent_axs=ax1)
ax2 = agglomCustring(data=FlameFile,agglom=AgglomFlame,parent_axs=ax2)
ax3 = agglomCustring(data=JainFile,agglom=AgglomJain,parent_axs=ax3)
ax4 = agglomCustring(data=SpiralFile,agglom=AgglomSpiral,parent_axs=ax4)
```









4 - DBSCAN

In [ ]:

```
# import numpy as np
# from sklearn.cluster import DBSCAN
# from sklearn.datasets.samples_generator import make_blobs
# from sklearn.preprocessing import StandardScaler
# import matplotlib.pyplot as plt
# %matplotlib inline
```

In [ ]:

```
from sklearn.cluster import DBSCAN
```

In [ ]:

```
epsilon = 1.48
minimumSamples = 2
DBAggregation = DBSCAN(eps=1.5, min_samples=8).fit(AggregationFile[:,:2])
DBFlame = DBSCAN(eps=1.48, min_samples=12).fit(FlameFile[:,:2])
DBJain = DBSCAN(eps=2.5, min_samples=15).fit(JainFile[:,:2])
DBSpiral = DBSCAN(eps=1.65, min_samples=3).fit(SpiralFile[:,:2])
DBAggregation_labels = DBAggregation.labels_
DBFlame_labels = DBFlame.labels_
DBJain_labels = DBJain.labels_
DBSpiral_labels = DBSpiral.labels_
# DBJain_labels
```

In [ ]:

```
# Firts, create an array of booleans using the labels from db.

DBAggregation-core_samples_mask = np.zeros_like(DBAggregation.labels_, dtype=bool)
DBAggregation-core_samples_mask[DBAggregation.core_sample_indices_] = True

DBFlame-core_samples_mask = np.zeros_like(DBFlame.labels_, dtype=bool)
```

```
DBFlame-core_samples_mask[DBFlame.core_sample_indices_] = True

DBJain-core_samples_mask = np.zeros_like(DBJain.labels_, dtype=bool)
DBJain-core_samples_mask[DBJain.core_sample_indices_] = True

DBSpiral-core_samples_mask = np.zeros_like(DBSpiral.labels_, dtype=bool)
DBSpiral-core_samples_mask[DBSpiral.core_sample_indices_] = True

# core_samples_mask
```

In [ ]:

```
# Number of clusters in labels, ignoring noise if present.
DBAggregation-n_clusters_ = len(set(DBAggregation_labels)) - (1 if -1 in DBAggregation_labels
DBFlame-n_clusters_ = len(set(DBFlame_labels)) - (1 if -1 in DBFlame_labels else 0)
DBJain-n_clusters_ = len(set(DBJain_labels)) - (1 if -1 in DBJain_labels else 0)
DBSpiral-n_clusters_ = len(set(DBSpiral_labels)) - (1 if -1 in DBSpiral_labels else 0)
DBSpiral-n_clusters_
```

Out[ ]:

```
3
```

In [ ]:

```
# Remove repetition in labels by turning it into a set.
DBAggregation_unique_labels = set(DBAggregation_labels)
DBFlame_unique_labels = set(DBFlame_labels)
DBJain_unique_labels = set(DBJain_labels)
DBSpiral_unique_labels = set(DBSpiral_labels)
DBSpiral_unique_labels
```

Out[ ]:

```
{0, 1, 2}
```

In [ ]:

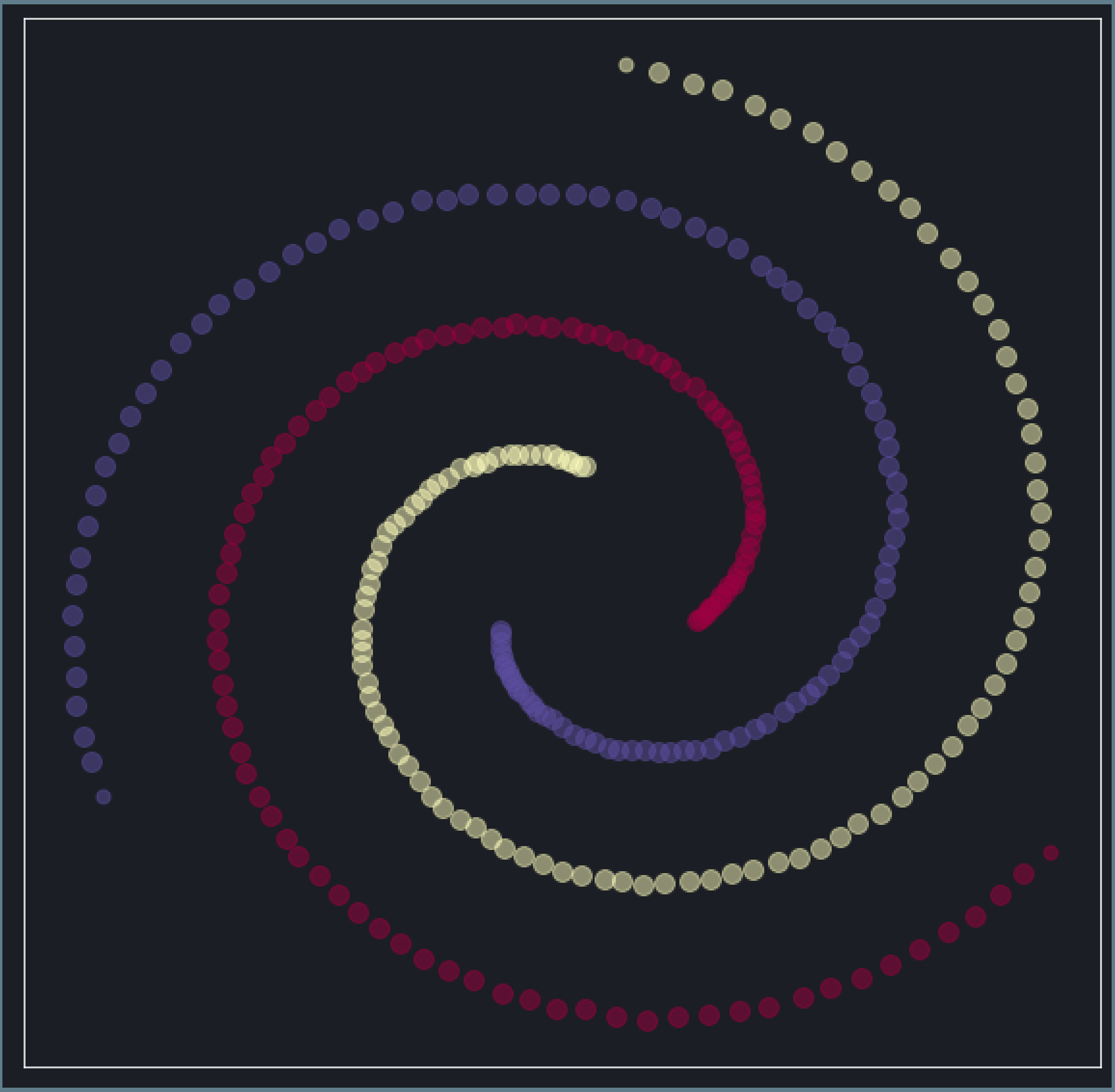
```
plt.style.use("dark_background")
for param in ['text.color', 'axes.labelcolor', 'xtick.color', 'ytick.color']:
    plt.rcParams[param] = '0.9'  # very light grey
for param in ['figure.facecolor', 'axes.facecolor', 'savefig.facecolor']:
    plt.rcParams[param] = '#1b1e24'  # bluish dark grey
colors = [
    '#AB46D2',  # teal/cyan
    '#55D8C1',  # yellow
    '#FF6FB5',  # pink
    '#FCF69C',  # matrix green
    '#36AE7C',
    '#F7E9D7',
    '#F9D923',
    '#FFFF00',
    '#FF00FF',
    '#FF0000',
    '#FFFFFF'
]
```

In [ ]:

```
def dbPlotMaker(X, unique_labels, lables, core_samples_mask,parent_axs=None):
    """
    X : Raw data
    """
    if parent_axs :
        ax=parent_axs
    else :
        # Initialize the plot with the specified dimensions.
        fig = plt.figure(figsize=(10, 10))
        # Create a plot
        ax = plt

    # Plot the points with colors
    colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))

    for k, col in zip(unique_labels, colors):
        # Create colors for the clusters.

        if k == -1:
            # Black used for noise.
            col = 'k'

        class_member_mask = (lables == k)

        # Plot the datapoints that are clustered
        xy = X[class_member_mask & core_samples_mask]
        ax.scatter(xy[:, 0], xy[:, 1],s=100, c=[col], marker=u'o', alpha=0.5)

        # Plot the outliers
        xy = X[class_member_mask & ~core_samples_mask]
        # ax.plot(xy[:, 0], xy[:, 1],s=50, c=[col], marker=u'o', alpha=0.5)
        ax.scatter(xy[:, 0], xy[:, 1],s=50, c=[col], marker=u'o', alpha=0.5)
    if parent_axs :
        # Remove the x ticks, y ticks, x and y axis
        ax.set_xticks([])
        ax.set_yticks([])
    else :
        plt.xticks([])
        plt.yticks([])

    if parent_axs:
        return ax
    ax.show()
```

In [ ]:

```
dbPlotMaker(SpiralFile, DBSpiral_unique_labels, DBSpiral_labels, DBSpiral.core_samples_mask )
```



In [ ]:

```
fig, ((ax1,ax2),(ax3,ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(20, 20))
ax1 = dbPlotMaker(X=AggregationFile, unique_labels=DBAggregation_unique_labels, lables=DBAggr
ax2 = dbPlotMaker(FlameFile, DBFlame_unique_labels, DBFlame_labels, DBFlame.core_samples_mask,
ax3 = dbPlotMaker(JainFile, DBJain_unique_labels, DBJain_labels, DBJain.core_samples_mask, par
ax4 = dbPlotMaker(SpiralFile, DBSpiral_unique_labels, DBSpiral_labels, DBSpiral.core_samples_m
```