Problem 1:

1. Cluster 1 has 5 points assigned to it and cluster 2 has 4
2. 2

Problem 2:

- 12

M11-HW1

November 26, 2023

1 Problem 1

1.1 Problem Description

In this problem you will use DBSCAN to cluster two melt pool images from a powder bed fusion metal 3D printer. Often times during printing there can be spatter around the main melt pool, which is undesirable. If we can successfully train a model to identify images with large amounts of spatter, we can automatically monitor the printing process.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

- Bitmap visualization of melt pool images 1 and 2
- Visualization of final DBSCAN clustering result for both melt pool images
- Discussion of tuning, final number of clusters, and the sensitivity of the model parameters for the two images.

Imports and Utility Functions:

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.cluster import DBSCAN

def points_to_bitmap(x):
    bitmap = np.zeros((64, 64), dtype=int)
    cols, rows = x[:, 0], x[:, 1]
    bitmap[rows, cols] = 1
    return bitmap

def plot_bitmap(bitmap):
    _, ax = plt.subplots(figsize=(3,3), dpi = 200)
    colors = ListedColormap(['black', 'white'])
    ax.imshow(bitmap, cmap = colors, origin = 'lower')
    ax.axis('off')

def plot_points(x, labels):
```

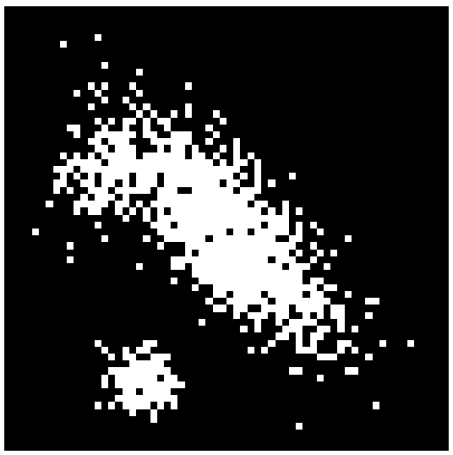
```
    fig = plt.figure(figsize = (5,4), dpi = 150)
    for i in range(min(labels),max(labels)+1):
        plt.scatter(x[labels == i, 0], x[labels == i, 1], alpha = 0.5, marker =␣
↪'s')
    plt.gca().set_aspect('equal')
    plt.tight_layout()
    plt.show()
```

1.2 Melt Pool Image #1

Load the first meltpool scan from the `m11-hw1-data1.txt` file using `np.loadtxt()`, and pass the `dtype = int` argument to ensure all values are loaded with their integer coordinates. You can convert these points to a binary bitmap using the provided `points_to_bitmap()` function, and then visualize the image using the provided `plot_bitmap()` function.

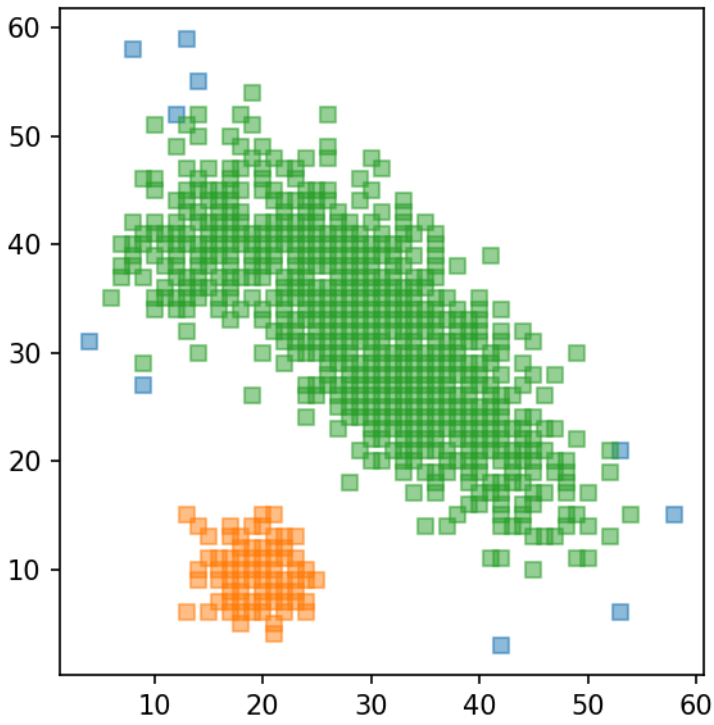
Note: you will use the integer coordinates for clustering, the bitmap is just for visualizing the data.

[ ]:
```
x1 = np.loadtxt("data/m11-hw1-data1.txt", dtype=int)
bitmap1 = points_to_bitmap(x1)
plot_bitmap(bitmap1)
```



Using the `sklearn.cluster.DBSCAN()` function, cluster the melt pool until you get well defined clusters. You will have to modify the `eps` and `min_samples` parameters to get satisfactory results. You can visualize the clustering with the provided `plot_points(x, labels)` function, where `x` is the integer coordinates of all the points, and labels are the labels assigned by `DBSCAN`. Plot the results of your final clustering using the `plot_points()` function

[ ]:
```
model1 = DBSCAN(eps=5, min_samples=20).fit(x1)
plot_points(x1, model1.labels_)
```



1.3 Melt Pool Image #2

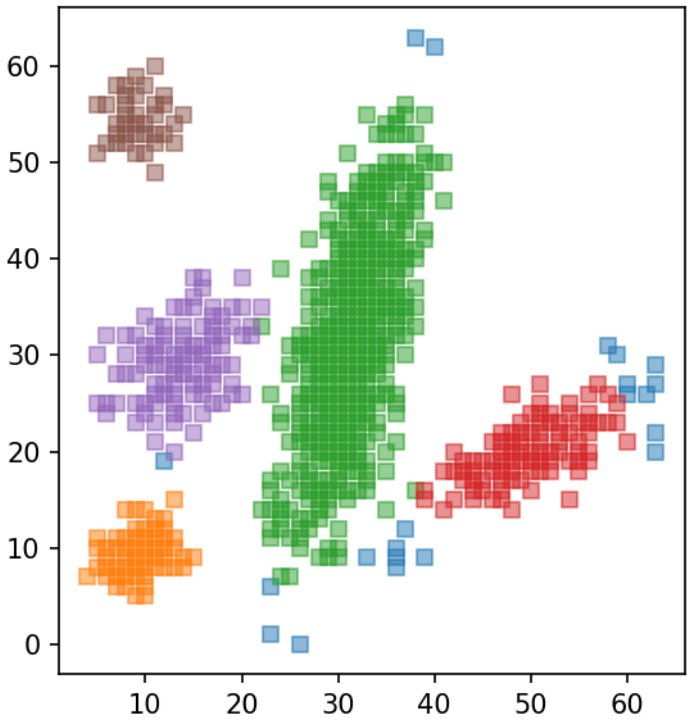
Now load the second melt pool scan from the `m11-hw1-data2.txt` file using `np.loadtxt()`, and the `dtype = int` argument to ensure all values are loaded with their integer coordinates. Again, convert the points to a binary bitmap, and visualize the bitmap using the provided functions.

[ ]:
```
x2 = np.loadtxt("data/m11-hw1-data2.txt", dtype=int)
bitmap2 = points_to_bitmap(x2)
plot_bitmap(bitmap2)
```

Using the `sklearn.cluster.DBSCAN()` function, cluster the meltpool until you get well defined clusters. You will have to modify the `eps` and `min_samples` parameters to get satisfactory results. You can visualize the clustering with the provided `plot_points(x, labels)` function, where `x` is the integer coordinates of all the points, and labels are the labels assigned by `DBSCAN`. Plot the results of your final clustering using the `plot_points()` function.

Note: this melt pool is significantly noisier than the last and therefore requires more sensitive tuning of the two DBSCAN parameters.

[ ]:
```
model2 = DBSCAN(eps=4, min_samples=15).fit(x2)
plot_points(x2, model2.labels_)
```



1.4 Discussion

Discuss how you tuned the `eps` and `min_samples` parameters for the two models. How many clusters did you end up finding in each image? Why does a wider range of `eps` and `min_samples` values successfully cluster melt pool image #1 compared to melt pool image #2?

The eps parameter was tuned according to the minimum distance within visible clusters. For example, on the first dataset, the smallest cluster had a width of about 10-20 with a large density of points so the eps was made to be about half the width and then adjusted from there. The min_samples was tuned by visually inspecting the number of samples that appeared to be in each

distinct cluster and then lowballing this estimate for the final min_samples value. For the first image this ended up finding 2 clusters and for the second image this ended up finding 5 clusters. The wider parameters for image 1 work better because there are less clusters and they are more spread out than in the second image.

M11-L1-P1

November 26, 2023

0.1 M10-L1 Problem 1

In this problem you will implement the K-Means algorithm from scratch, and use it to cluster two datasets: a "blob" shaped dataset with three classes, and a "moon" shaped dataset with two classes.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs, make_moons

## DO NOT MODIFY
def plotter(x, y, labels = None, centers = None):
    fig = plt.figure(dpi = 200)
    for i in range(len(np.unique(y))):
        if labels is not None:
            plt.scatter(x[labels == i, 0], x[labels == i, 1], alpha = 0.5)
        else:
            plt.scatter(x[y == i, 0], x[y == i, 1], alpha = 0.5)
    if labels is not None:
        if (labels != y).any():
            plt.scatter(x[labels != y, 0], x[labels != y, 1], s = 100, c =␣
↪'None', edgecolors = 'black', label = 'Misclassified Points')
    if centers is not None:
        plt.scatter(centers[:,0], centers[:,1], c = 'red', label = 'Cluster␣
↪Centers')
    plt.xlabel('$x_0$')
    plt.ylabel('$x_1$')
    if labels is not None or centers is not None:
        plt.legend()
    plt.show()
```

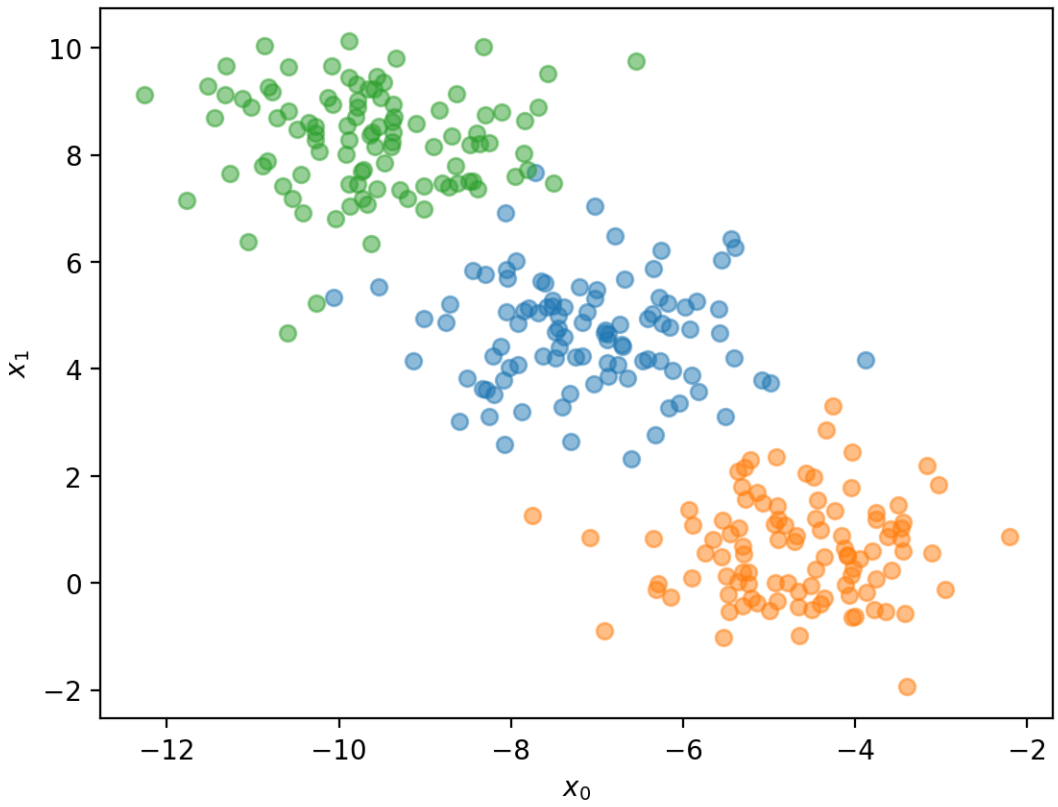
We will use `sklearn.datasets.make_blobs()` to generate the dataset. The `random_state = 12` argument is used to ensure all students have the same data.

[ ]:
```
## DO NOT MODIFY
x, y = make_blobs(n_samples = 300, n_features = 2, random_state = 12)
```

Visualize the data using the `plotter(x,y)` function. You do not need to pass the `labels` or

`centers` arguments

[ ]:
```
plotter(x,y)
```



Now we will begin to create our own K-Means function.

First you will write a function `find_cluster(point, centers)` which returns the index of the cluster center closest to the given point. - `point` is a one dimensional numpy array containing the x_0 and x_1 coordinates of a single data point - `centers` is a 3×2 numpy array containing the coordinates of the three cluster centers at any given iteration - **return** the index of the closest cluster center

[ ]:
```
def find_cluster(point, centers):
    dist = np.sum(np.power(np.subtract(centers, point),2), axis=1)
    return np.where(dist == min(dist))[0][0]
```

Next, write a function `assign_labels(x, centers)` which will loop through all the points in `x` and use the `find_cluster()` function we just wrote to assign the label of the closest cluster center. Your function should return the labels - `x` is a 300×2 numpy array containing the coordinates of all the points in the dataset - `centers` is a 3×2 numpy array containing the coordinates of the three cluster centers at any given iteration - **return** a one dimensional numpy array of length 300 containing the corresponding label for each point in `x`

[ ]:
```
def assign_labels(x, centers):
    labels = np.zeros(x.shape[0])
    for i,point in enumerate(x):
        labels[i] = find_cluster(point, centers)
    return labels
```

Next, write a function `update_centers(x, labels)` which will compute the new cluster centers using the centroid of each cluster, provided all the points in `x` and their corresponding labels - `x` is a 300×2 numpy array containing the coordinates of all the points in the dataset - `labels` is a one dimensional numpy array of length 300 containing the corresponding label for each point in `x` - **return** a 3×2 numpy array containing the coordinates of the three cluster centers

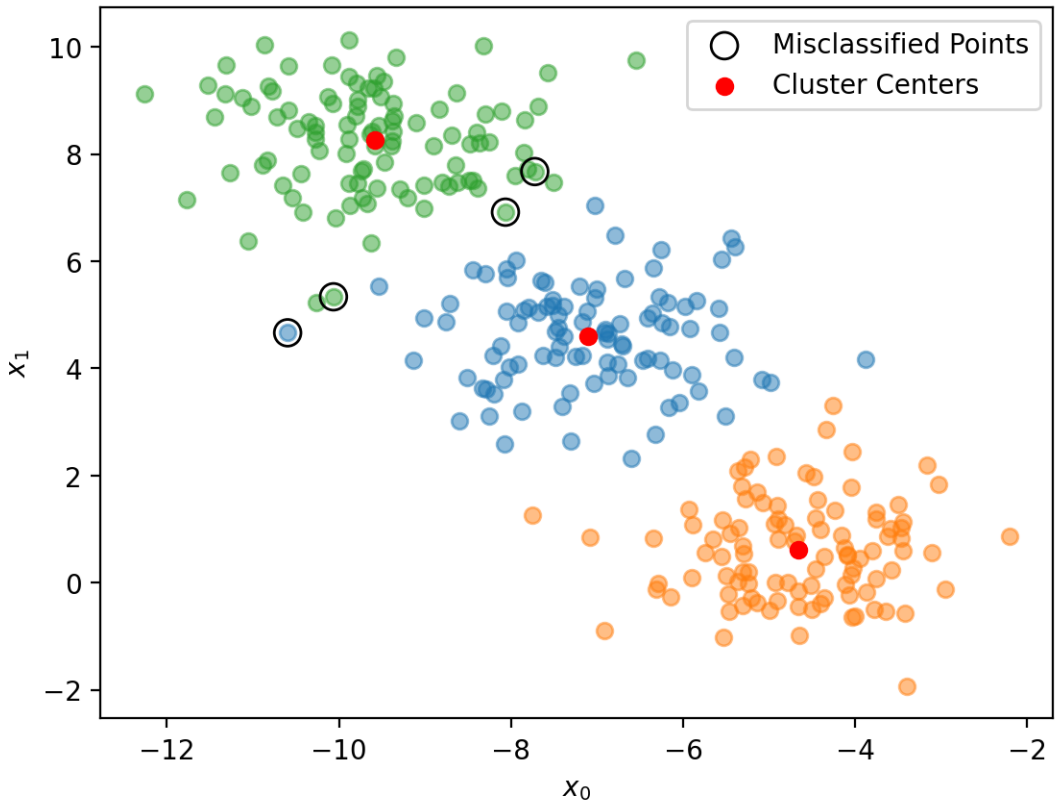
[ ]:
```
def update_centers(x, labels):
    centers = []
    for label in np.unique(labels):
        label_idx = np.where(labels == label)[0]
        centers.append(np.average(x[label_idx], axis=0))
    return np.array(centers)
```

Finally write a function `myKMeans(x, init_centers)` which will run the KMeans algorithm, provided all the points in `x` and the coordinates of the initial cluster centers in `init_centers`. Run the algorithm until there is no change in cluster membership in subsequent iterations. Your function should return both the `labels`, the labels of each point in x, and `centers`, the final coordinates of each of the cluster centers. - `x` is a 300×2 numpy array containing the coordinates of all the points in the dataset - `init_centers` is a 3×2 numpy array containing the coordinates of the three cluster centers provided to you - **return** `labels` and `centers` as defined above

[ ]:
```
def myKMeans(x, init_centers):
    centers = init_centers
    for i in range(100):
        labels = assign_labels(x, centers)
        new_centers = update_centers(x, labels)
        dist = 0
        for center, new_center in zip(centers, new_centers):
            dist += np.sum(np.power(np.subtract(center, new_center),2))
        centers = new_centers
        if (dist < 0.001): return labels, centers
```

Now use your `myKMeans()` function to cluster the provided data points `x` and set the initial cluster centers as `init_centers = np.array([[-5,5],[0,0],[-10,10]])`. Then use the provided plotting function, `plotter(x,y,labels,centers)` to visualize your model's clustering.

[ ]:
```
init_centers = np.array([[-5,5],[0,0],[-10,10]])
labels, centers = myKMeans(x, init_centers)

plotter(x, y, labels, centers)
```



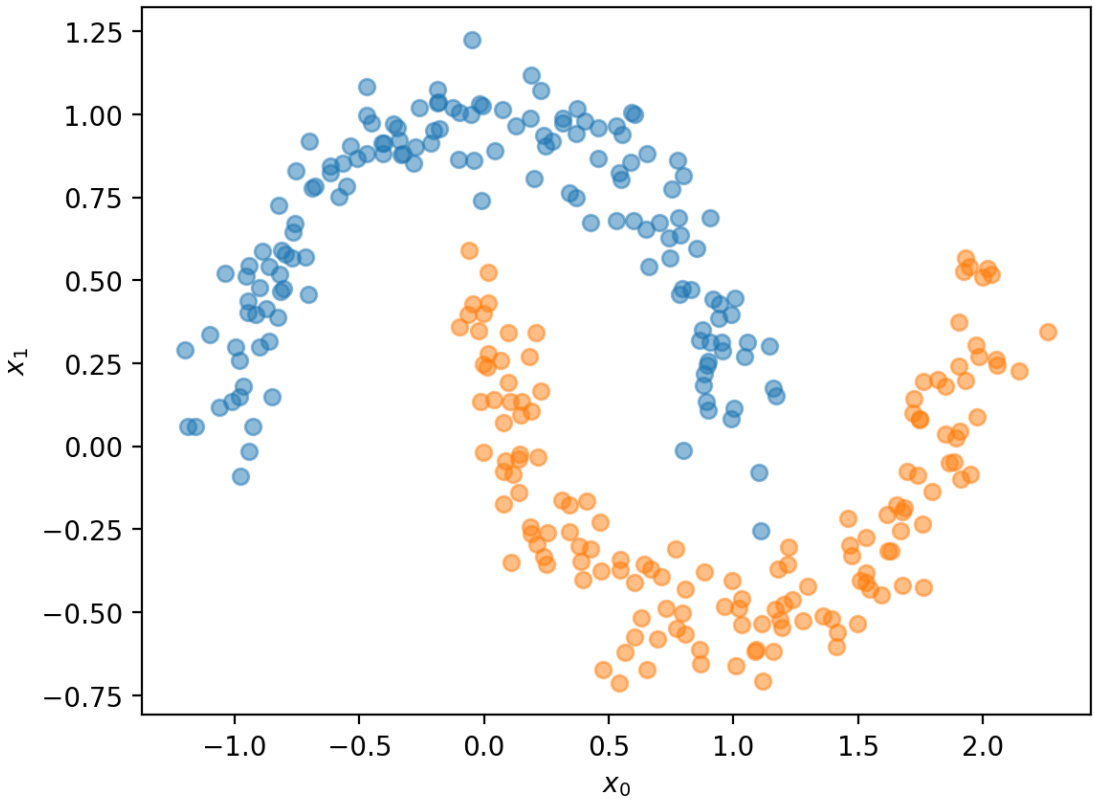
0.2 Moon Dataset

Now we will try using our `myKMeans()` function on a more challenging dataset, as generated below.

[ ]:
```
## DO NOT MODIFY
x,y = make_moons(n_samples = 300, noise = 0.1, random_state = 0)
```

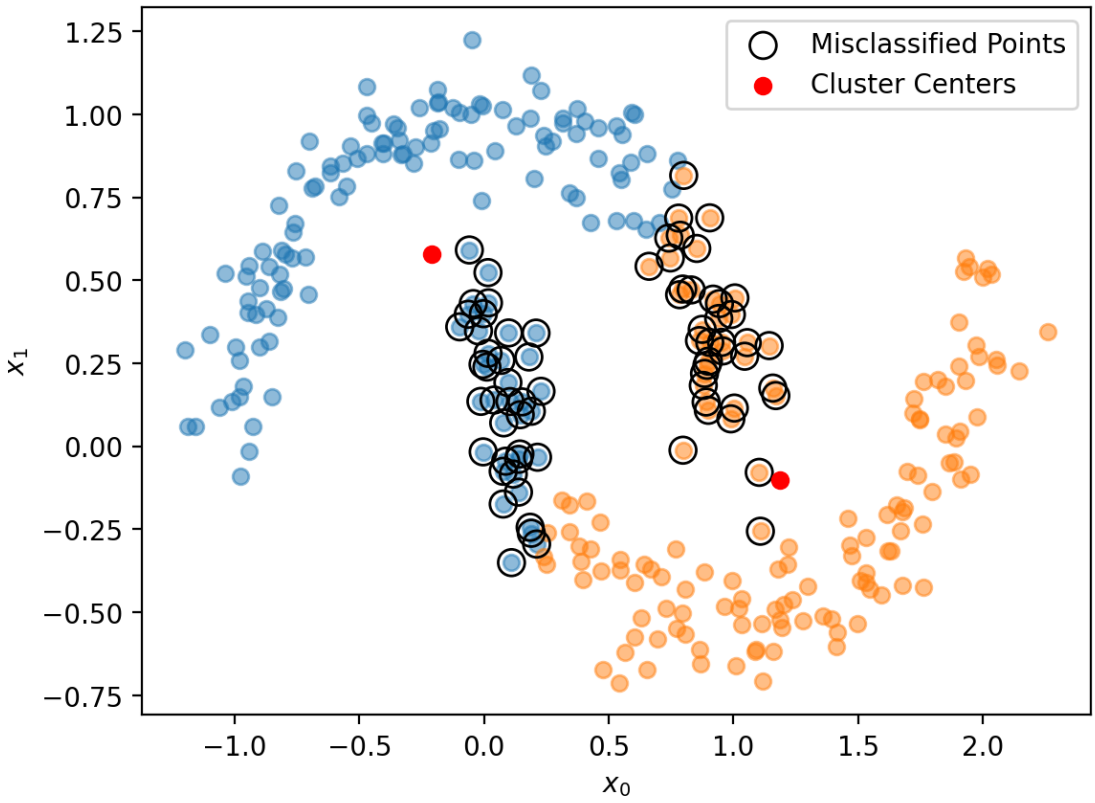
Visualize the data using the `plotter(x,y)` function.

[ ]:
```
plotter(x, y)
```



Using your `myKMeans()` function and `init_centers = np.array([[0,1],[1,-0.5]])` cluster the data, and visualize the results using `plotter(x,y,labels,centers)`.

[ ]:
```
init_centers = np.array([[0,1],[1,-0.5]])
labels, centers = myKMeans(x, init_centers)

plotter(x, y, labels, centers)
```


Misclassified Points
Cluster Centers
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
x_1
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_0

M11-L1-P2

November 26, 2023

0.1 M10-L1 Problem 2: Solution

In this problem you will use the `sklearn` implementation of the K-Means algorithm to cluster the same two datasets from problem 1.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs, make_moons
from sklearn.cluster import KMeans

## DO NOT MODIFY
def plotter(x, y, labels = None, centers = None):
    fig = plt.figure(dpi = 200)
    for i in range(len(np.unique(y))):
        if labels is not None:
            plt.scatter(x[labels == i, 0], x[labels == i, 1], alpha = 0.5)
        else:
            plt.scatter(x[y == i, 0], x[y == i, 1], alpha = 0.5)
    if labels is not None:
        if (labels != y).any():
            plt.scatter(x[labels != y, 0], x[labels != y, 1], s = 100, c =␣
 ↪'None', edgecolors = 'black', label = 'Misclassified Points')
    if centers is not None:
        plt.scatter(centers[:,0], centers[:,1], c = 'red', label = 'Cluster␣
 ↪Centers')
    plt.xlabel('$x_0$')
    plt.ylabel('$x_1$')
    if labels is not None or centers is not None:
        plt.legend()
    plt.show()
```

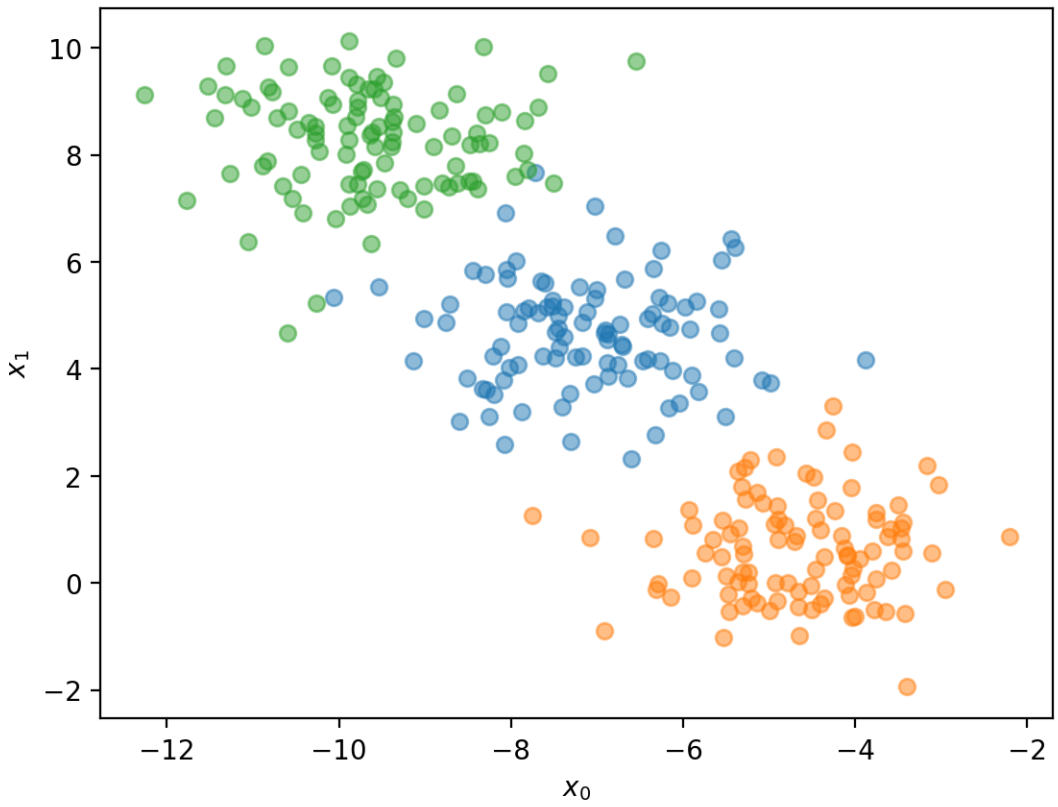
We will use `sklearn.datasets.make_blobs()` to generate the dataset. The `random_state = 12` argument is used to ensure all students have the same data.

[ ]:
```
## DO NOT MODIFY
x, y = make_blobs(n_samples = 300, n_features = 2, random_state = 12)
```

Visualize the data using the `plotter(x,y)` function. You do not need to pass the `labels` or

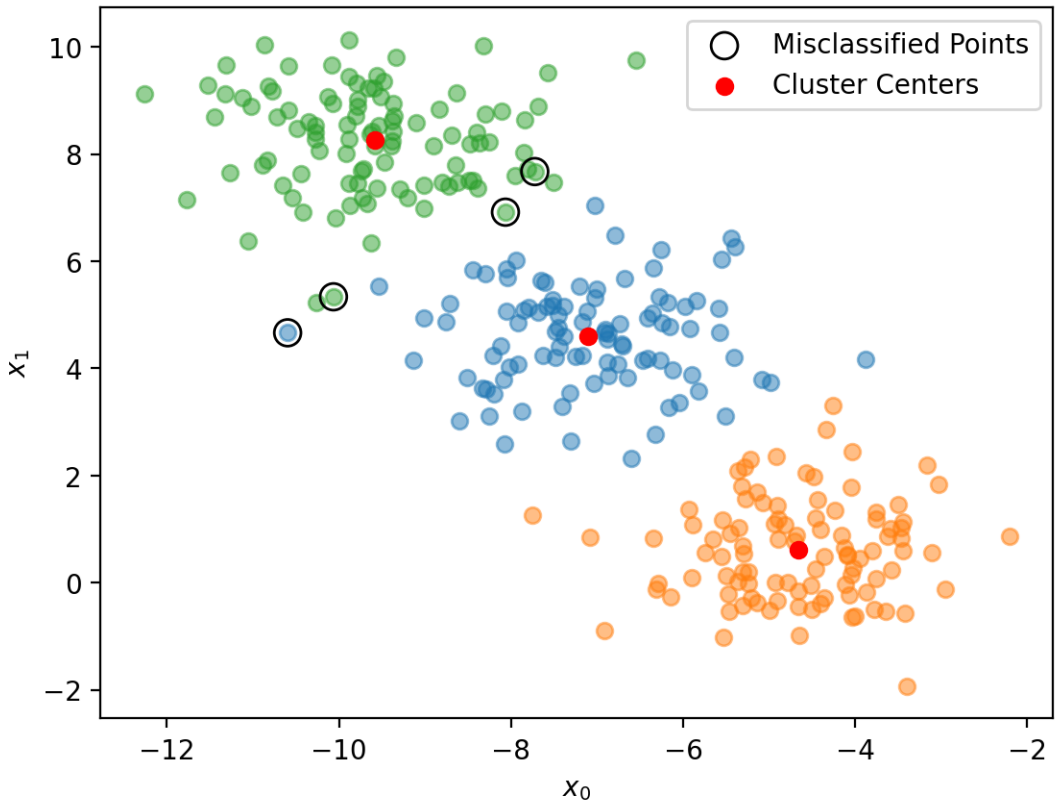
`centers` arguments

[ ]:
```
plotter(x,y)
```



Now you will use `sklearn.cluster.KMeans()` to cluster the provided data points `x`. For the `KMeans()` function to perform identically to our implementation, we need to provide the same initial clusters with the `init` argument. The cluster centers should be initialized as `np.array([[-5,5],[0,0],[-10,10]])`, and you can additionally pass in the `n_init = 1` argument to silence a runtime warning that comes from passing explicit initial cluster centers. Then plot the results using the provided `plotter(x,y,labels,centers)` function.

[ ]:
```
model = KMeans(n_clusters=3, init=np.array([[-5,5],[0,0],[-10,10]]), n_init=1).
  ↪fit(x)
centers = model.cluster_centers_
labels = model.labels_

plotter(x, y, labels, centers)
```



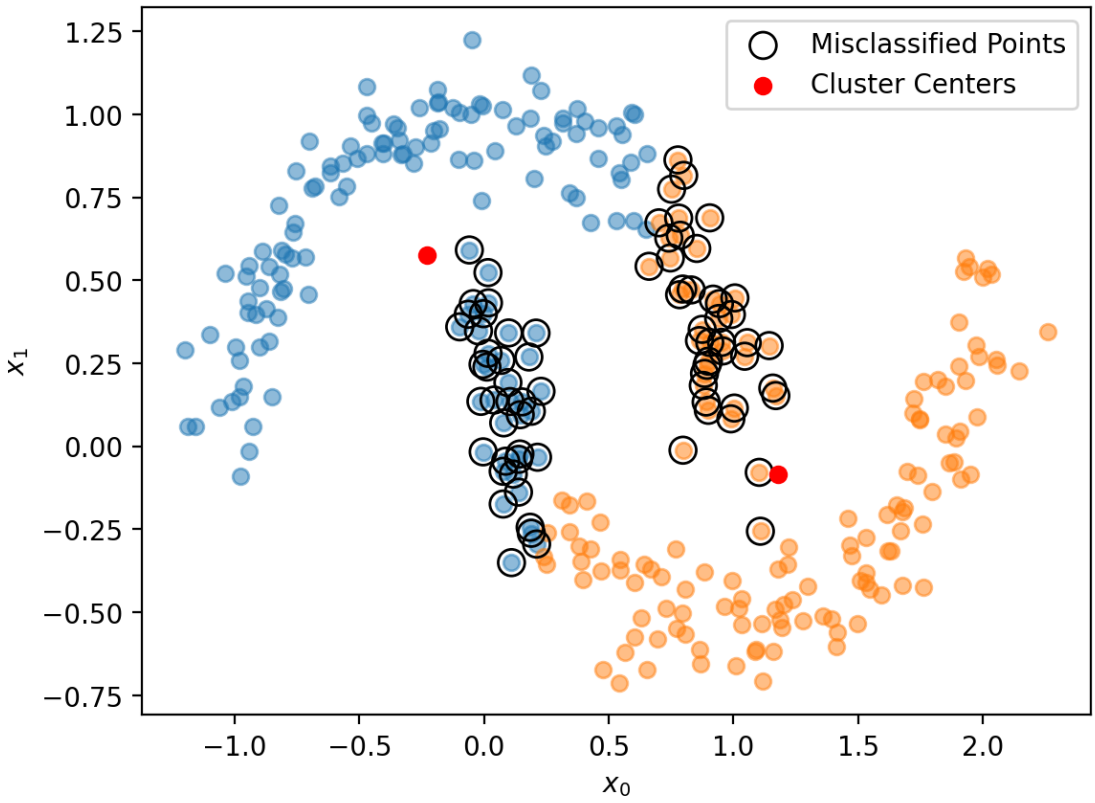
0.2 Moon Dataset

Now we will try using the `sklearn.cluster.KMeans()` function on the moons dataset from problem 1.

[ ]:
```
## DO NOT MODIFY
x,y = make_moons(n_samples = 300, noise = 0.1, random_state = 0)
```

Using the same initial cluster centers from problem 1, namely, `np.array([[0,1],[1,-0.5]])`, cluster the moons datasets and plot the results using the provided `plotter(x,y,labels,centers)` function.

[ ]:
```
model = KMeans(n_clusters=2, init=np.array([[0,1],[1,-0.5]]), n_init=1).fit(x)
centers = model.cluster_centers_
labels = model.labels_

plotter(x, y, labels, centers)
```



0.3 Discussion

How do the results of your hand coded implementation of the K-Means algorithm compare to the `sklearn` implementation? If there is any discrepancy between the results, provide your reasoning why.

My hand coded implementation exactl matches the sklearn algorithm. This is because using the same initial conditions wil result in the same outcome from an optimization standpoint. Since both are following the same general algorithm and all the input parameters are the same, it isn't a surprise that the results are the same.

M11-L1-P3

November 26, 2023

0.1 M11-L1 Problem 3

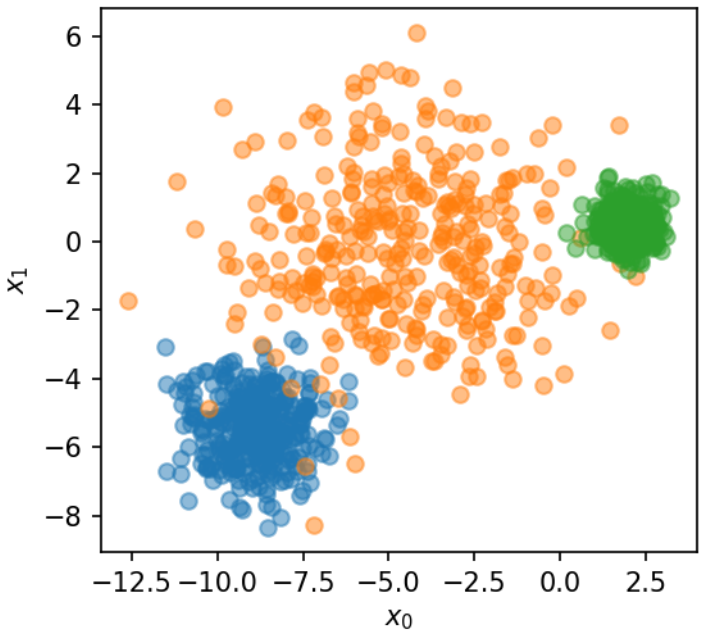
In this problem you will use the `sklearn` implementation of hierarchical clustering with three different linkage criteria `('single', 'complete', 'average')` to clusters two datasets: a "blob" shaped dataset with three classes, and a concentric circle dataset with two classes.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs, make_circles
from sklearn.cluster import AgglomerativeClustering

## DO NOT MODIFY
def plotter(x, labels = None, ax = None, title = None):
    if ax is None:
        _, ax = plt.subplots(dpi = 150, figsize = (4,4))
        flag = True
    else:
        flag = False
    for i in range(len(np.unique(labels))):
        ax.scatter(x[labels == i, 0], x[labels == i, 1], alpha = 0.5)
    ax.set_xlabel('$x_0$')
    ax.set_ylabel('$x_1$')
    ax.set_aspect('equal')
    if title is not None:
        ax.set_title(title)
    if flag:
        plt.show()
    else:
        return ax
```

First we will consider the "blob" dataset, generated below. Visualize the data using the provided `plotter(x, labels)` function.

[ ]:
```
## DO NOT MODIFY
x, labels = make_blobs(n_samples = 1000, cluster_std=[1.0, 2.5, 0.5],␣
↪random_state = 170)
```

[ ]:
```
plotter(x, labels)
```

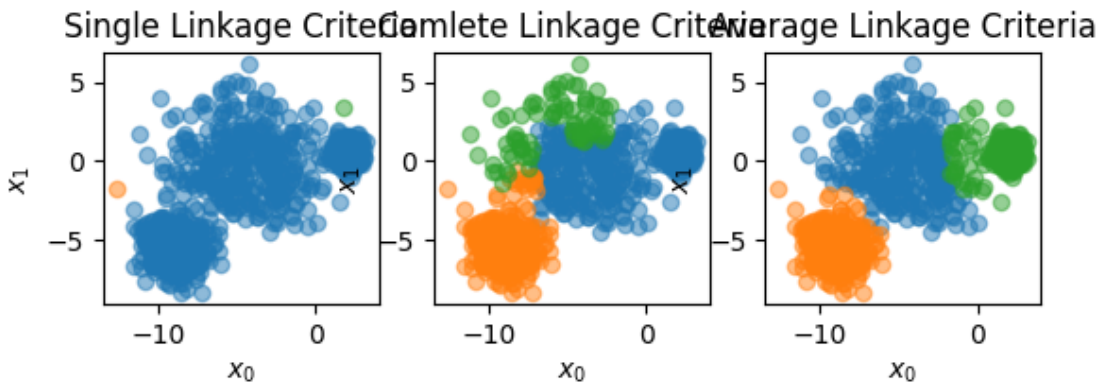


Using the `AgglomerativeClustering()` function, generate 3 side-by-side plots using `plt.subplots()` and the provided `plotter(x, labels, ax, title)` function to visualize the results of the following three linkage criteria `['single', 'complete', 'average']`.

Note: the `plt.subplots()` function will return `fig, ax`, where `ax` is an array of all the subplot axes in the figure. Each individual subplot can be accessed with `ax[i]` which you can then pass to the `plotter()` function's `ax` argument.

[ ]:
```
fig, ax = plt.subplots(1,3)
model1 = AgglomerativeClustering(n_clusters=3,linkage='single').fit(x)
plotter(x, model1.labels_, ax[0], "Single Linkage Criteria")

model2 = AgglomerativeClustering(n_clusters=3,linkage='complete').fit(x)
plotter(x, model2.labels_, ax[1], "Comlete Linkage Criteria")

model3 = AgglomerativeClustering(n_clusters=3,linkage='average').fit(x)
plotter(x, model3.labels_, ax[2], "Average Linkage Criteria")
```

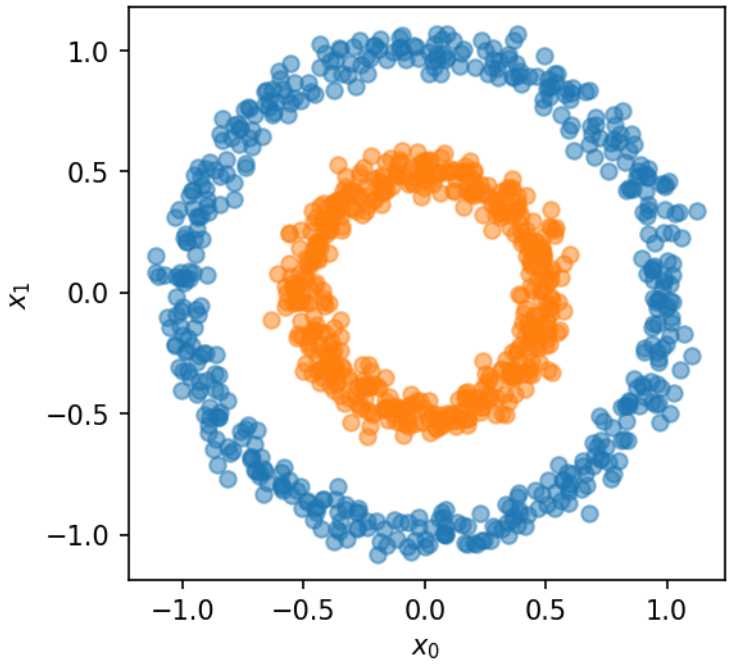
[ ]:
```
<Axes: title={'center': 'Average Linkage Criteria'}, xlabel='$x_0$',
ylabel='$x_1$'>
```



Now we will work on the concentric circle dataset, generated below. Visualize the data using the provided `plotter(x, labels)` function.

[ ]:
```
## DO NOT MODIFY
x, labels = make_circles(1000, factor = 0.5, noise = 0.05, random_state = 0)
```

[ ]:
```
plotter(x, labels)
```

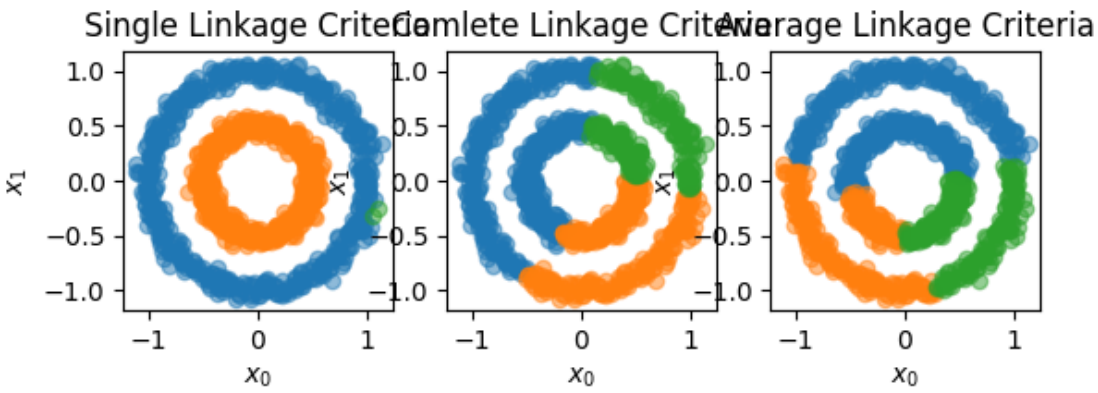


Again, use the `AgglomerativeClustering()` function to generate 3 side-by-side plots using `plt.subplots()` and the provided `plotter(x, labels, ax, title)` function to visualize the results of the following three linkage criteria `['single', 'complete', 'average']` for the con-

centric circle dataset.

[ ]:
```
fig, ax = plt.subplots(1,3)
model1 = AgglomerativeClustering(n_clusters=3,linkage='single').fit(x)
plotter(x, model1.labels_, ax[0], "Single Linkage Criteria")

model2 = AgglomerativeClustering(n_clusters=3,linkage='complete').fit(x)
plotter(x, model2.labels_, ax[1], "Comlete Linkage Criteria")

model3 = AgglomerativeClustering(n_clusters=3,linkage='average').fit(x)
plotter(x, model3.labels_, ax[2], "Average Linkage Criteria")
```

[ ]:
```
<Axes: title={'center': 'Average Linkage Criteria'}, xlabel='$x_0$',
ylabel='$x_1$'>
```



1 Discussion

Discuss the performance of the three different linkage criteria on the "blob" dataset, and then on the concentric circle dataset. Why do some linkage criteria perform better on one dataset, but worse on others?

The average linkage criteria performed the best on the blob dataset while the single linkage criteria performed best on the concectric dataset. Some linkage criteria perform better on different types of data because of the types of distances used and how they apply to the features of the data. For example, the blob data sets center around an average point and thus lend well to the average linkage criteria as demonstrated above.

M11-L2-P1

November 26, 2023

0.1 M11-L2 Problem 1

In this problem you will implement the elbow method using three different sklearn clustering algorithms: (`KMeans`, `SpectralClustering`, `GaussianMixture`). You will use the algorithms to find the number of natural clusters for two different datasets, one "blob" shaped dataset, and one concetric circle dataset.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200

from sklearn.datasets import make_blobs, make_circles
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.mixture import GaussianMixture

def plot_loss(loss, ax = None, title = None):
    if ax is None:
        ax = plt.gca()
    ax.plot(np.arange(2, len(loss)+2), loss, 'k-o')
    ax.set_xlabel('Number of Clusters')
    ax.set_ylabel('Loss')
    if title:
        ax.set_title(title)
    return ax

def plot_pred(x, labels, ax = None, title = None):
    if ax is None:
        ax = plt.gca()
    n_clust = len(np.unique(labels))
    for i in range(n_clust):
        ax.scatter(x[labels == i,0], x[labels == i,1], alpha = 0.5)
    ax.set_title(title)
    return ax

def compute_loss(x, labels):
    # Initialize loss
    loss = 0
    # Number of clusters
```

```
    n_clust = len(np.unique(labels))
    # Loop through the clusters
    for i in range(n_clust):
        # Compute the center of a given label
        center = np.mean(x[labels == i, :], axis = 0)
        # Compute the sum of squared distances between each point and its␣
↪corresponding cluster center
        loss += np.sum(np.linalg.norm(x[labels == i, :] - center, axis = 1)**2)
    return loss
```

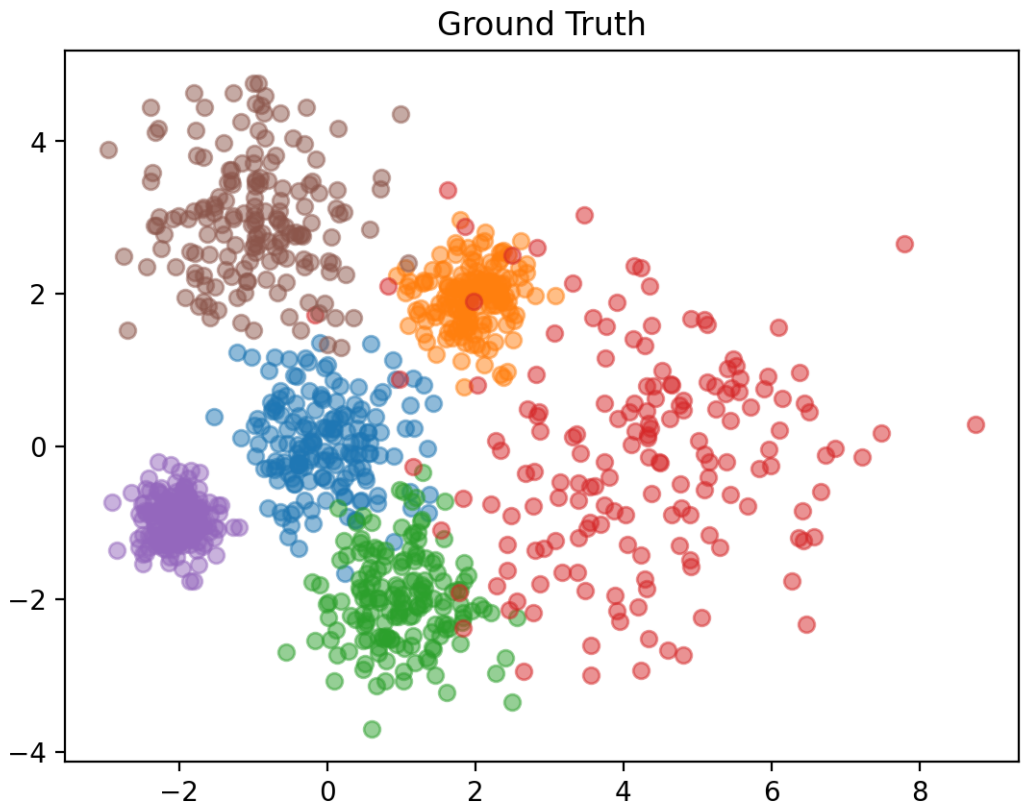
0.2 Blob dataset

Visualize the "blob" dataset generated below, using a unique color for each cluster of points, where y contains the label of each corresponding point in x.

[ ]:
```
## DO NOT MODIFY
x, y = make_blobs(n_samples = 1000, n_features = 2, centers =␣
↪[[0,0],[2,2],[1,-2],[4,0],[-2,-1],[-1,3]], cluster_std = [0.6,0.4,0.6,1.5,0.
↪3,0.8], random_state = 0)
```

[ ]:
```
fig, ax = plt.subplots()
plot_pred(x, y, ax, "Ground Truth")
```

[ ]:
```
<Axes: title={'center': 'Ground Truth'}>
```

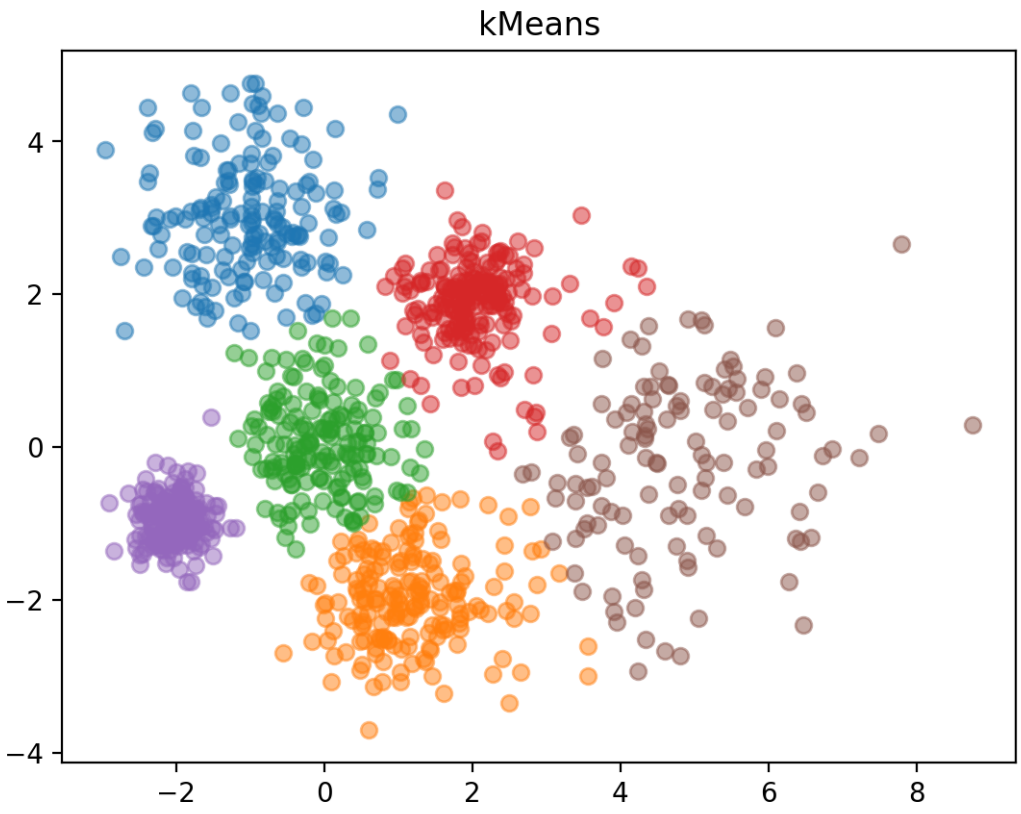


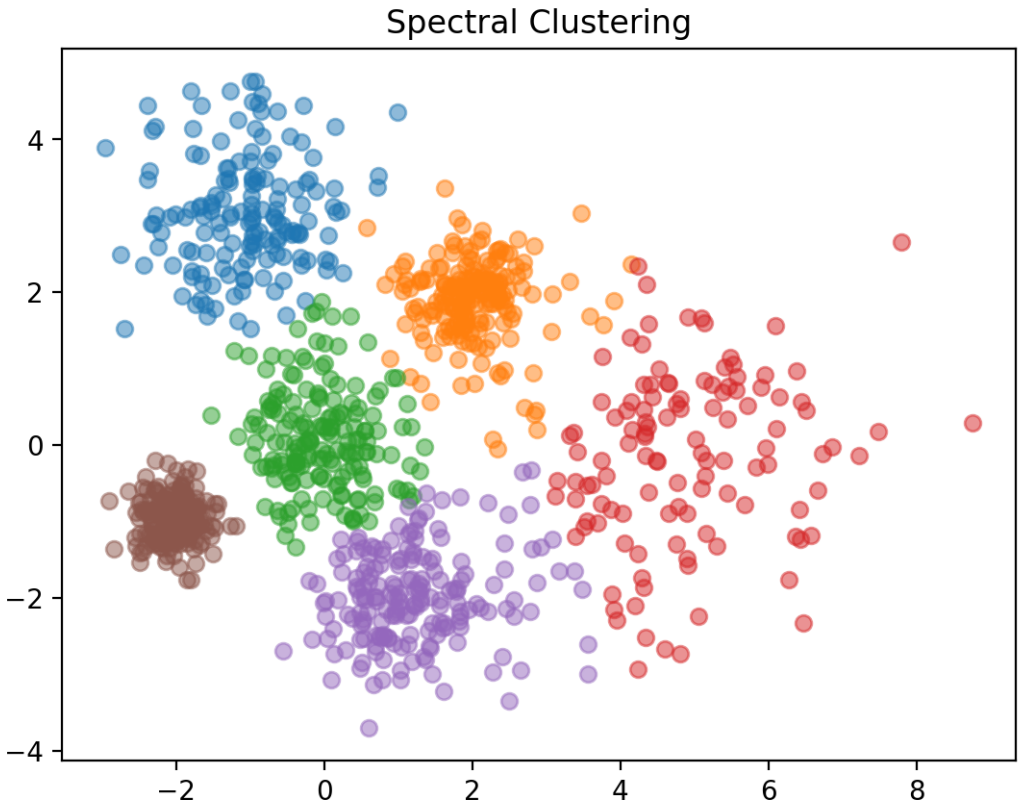
Use the `sklearn` KMeans, Spectral Clustering, and Gaussian Mixture Model functions to cluster the "blob" data with 6 clusters, and modify the parameters until you get satisfactory results. Plot the results of your three models side-by-side using `plt.subplots` and the provided `plot_pred(x, labels, ax, title)` function.

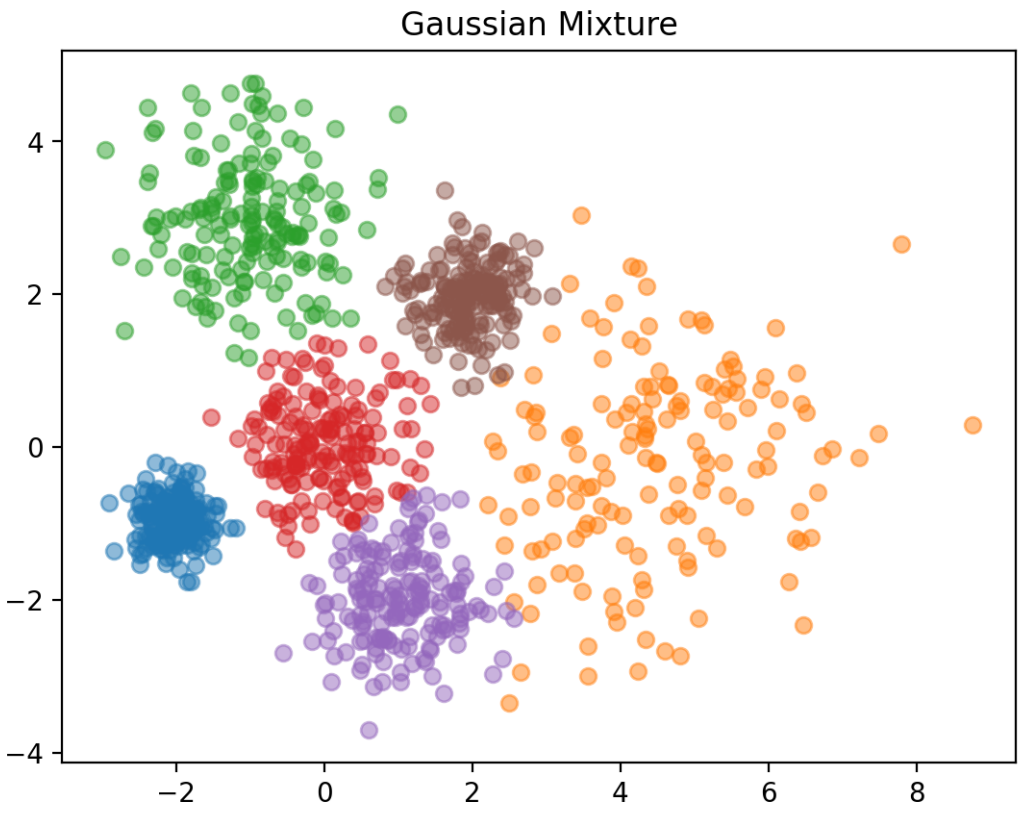
[ ]:
```
fig, ax = plt.subplots()
model1 = KMeans(n_clusters=6).fit(x)
plot_pred(x, model1.labels_, ax, "kMeans")

fig, ax = plt.subplots()
model2 = SpectralClustering(n_clusters=6).fit(x)
plot_pred(x, model2.labels_, ax, "Spectral Clustering")

fig, ax = plt.subplots()
model3 = GaussianMixture(n_components=6).fit(x)
plot_pred(x, model3.predict(x), ax, "Gaussian Mixture")
```

```
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-packages/threadpoolctl.py:1019:
RuntimeWarning: libc not found. The ctypes module in Python 3.8 is maybe too old
for this OS.
  warnings.warn(
```

[ ]:
```
<Axes: title={'center': 'Gaussian Mixture'}>
```


kMeans
4
2
0
−2
−4
−2
0
2
4
6
8


Spectral Clustering
4
2
0
−2
−4
−2
0
2
4
6
8



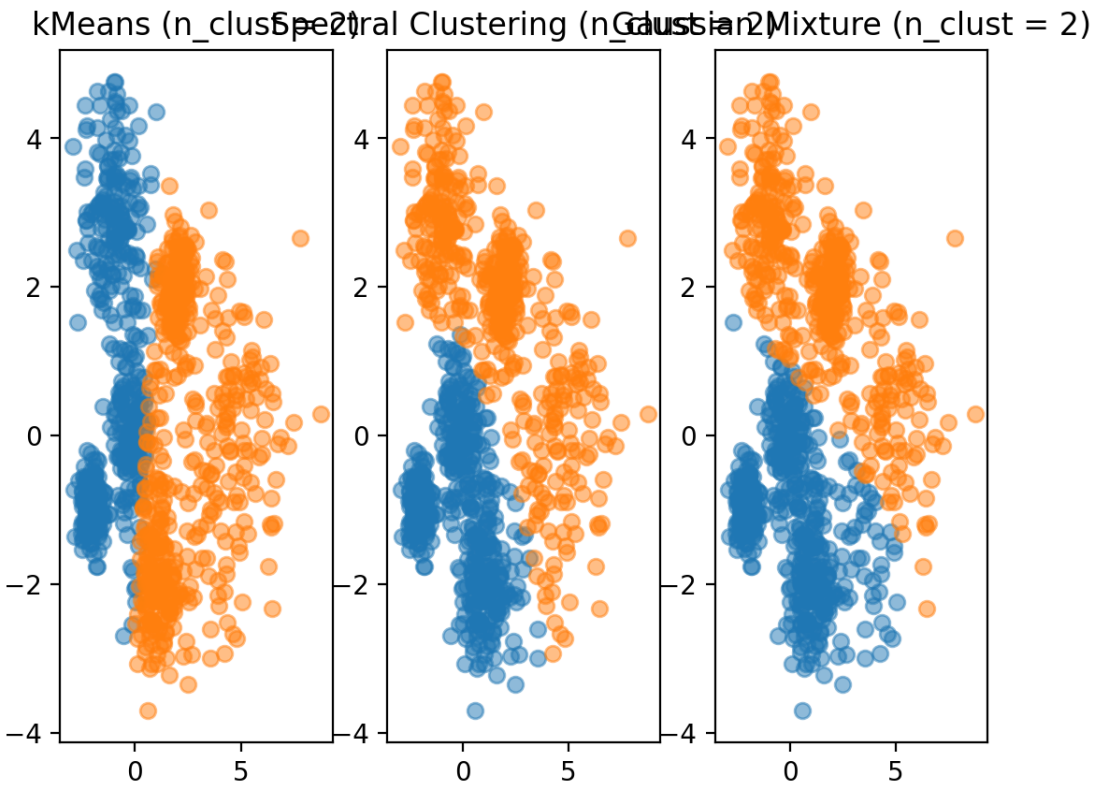
Using the parameters you found for the three models above, run each of the clustering algorithms for `n_clust = [2,3,4,5,6,7,8,9]` and compute the sum of squared distances loss for each case using the provided `compute_loss(x, labels)` function, where labels is the cluster assigned to each point by the algorithm. Plot loss versus number of cluster for each your three models in side-by-side subplots using the provided `plot_pred(x, labels, ax, title)` function.

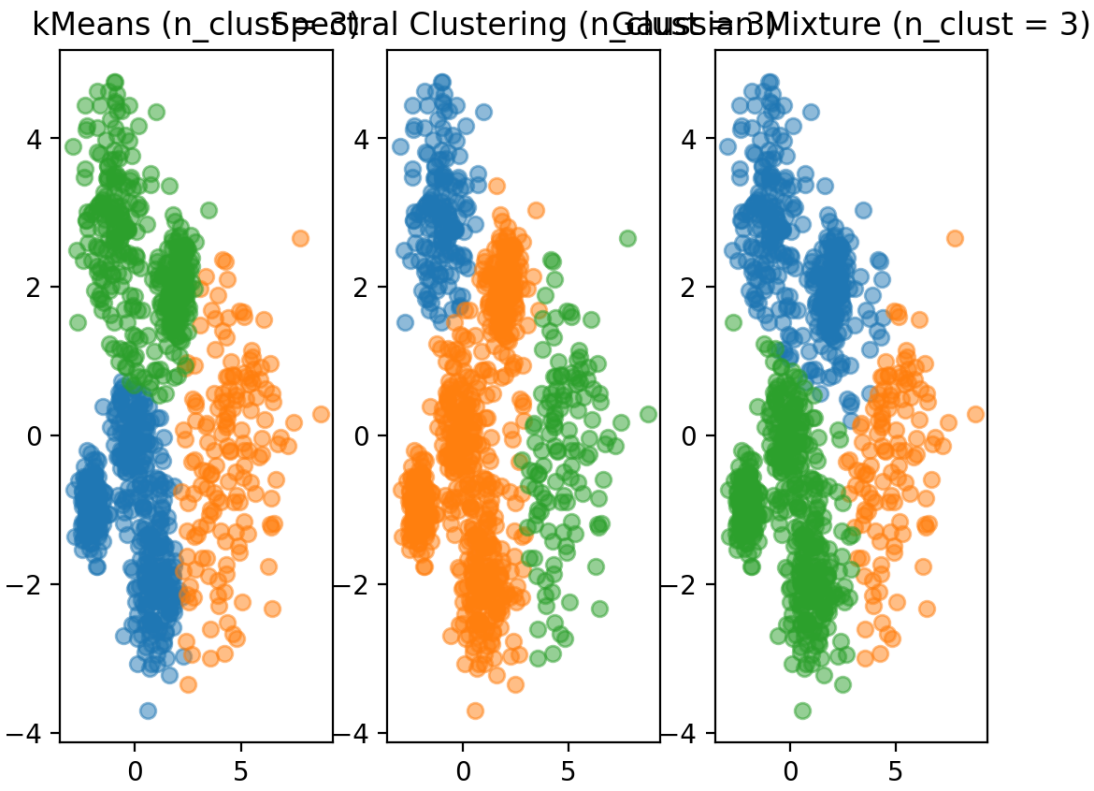
[ ]:
```
model1_sum_square_dist = []
model2_sum_square_dist = []
model3_sum_square_dist = []

for n_clust in [2, 3, 4, 5, 6, 7, 8, 9]:
    #Train models
    model1 = KMeans(n_clusters=n_clust).fit(x)
    model2 = SpectralClustering(n_clusters=n_clust).fit(x)
    model3 = GaussianMixture(n_components=n_clust).fit(x)

    model1_sum_square_dist.append(compute_loss(x, model1.labels_))
    model2_sum_square_dist.append(compute_loss(x, model2.labels_))
    model3_sum_square_dist.append(compute_loss(x, model3.predict(x)))

    fig, ax = plt.subplots(1,3)
```

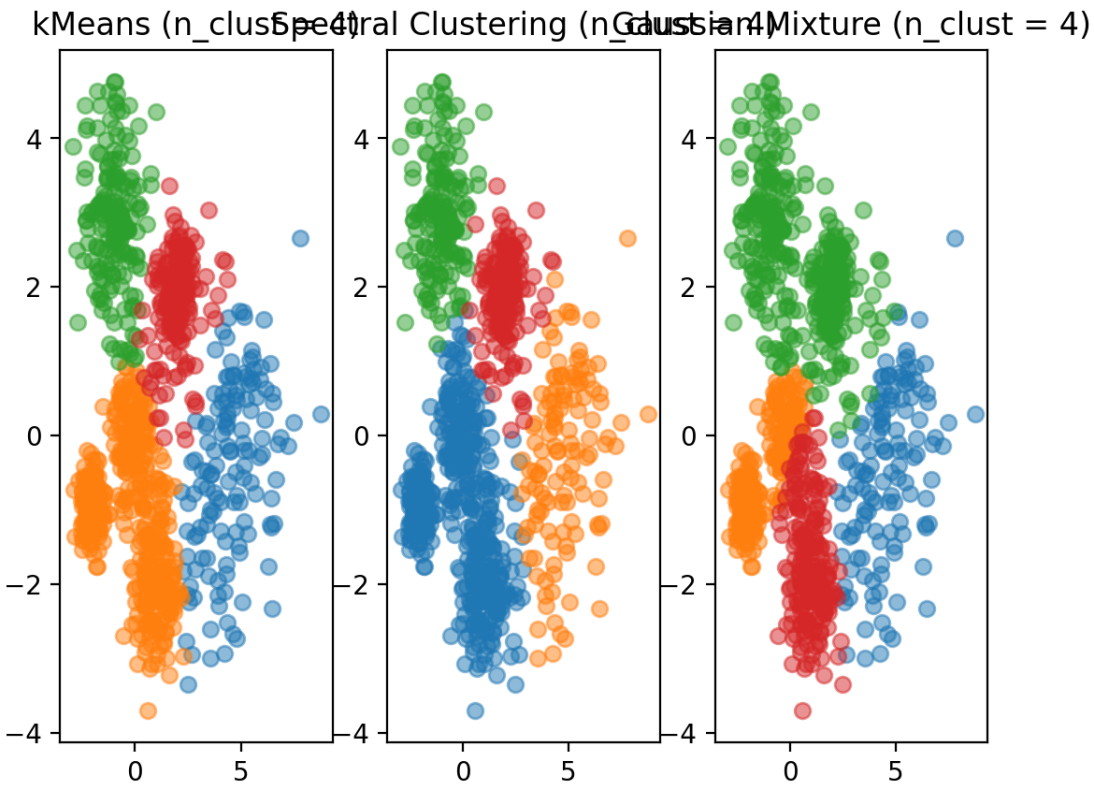
```
    plot_pred(x, model1.labels_, ax[0], f"kMeans (n_clust = {n_clust})")
    plot_pred(x, model2.labels_, ax[1], f"Spectral Clustering (n_clust =␣
↪{n_clust})")
    plot_pred(x, model3.predict(x), ax[2], f"Gaussian Mixture (n_clust =␣
↪{n_clust})")

fig, ax = plt.subplots(1,3)
plot_loss(model1_sum_square_dist, ax[0], "kMeans Sum Square Dist")
plot_loss(model2_sum_square_dist, ax[1], "Spectral Clustering Sum Square Dist")
plot_loss(model3_sum_square_dist, ax[2], "Gaussian Mixture Sum Square Dist")
```

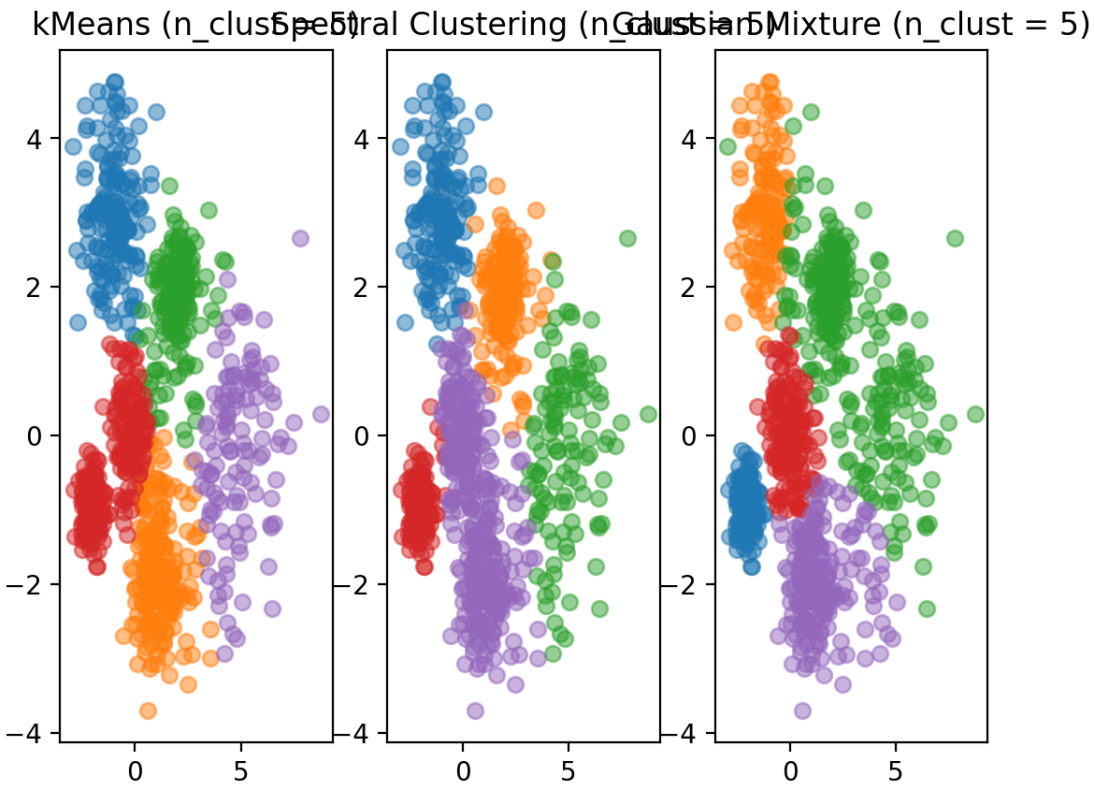
```
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

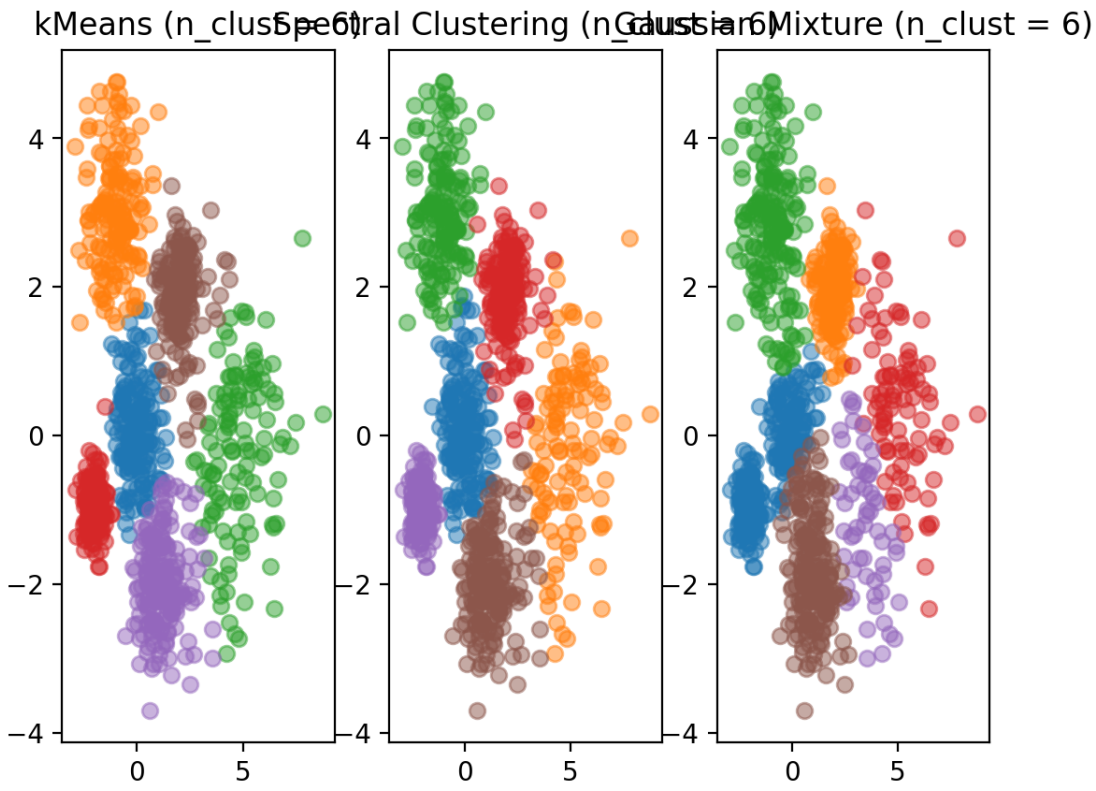
[ ]:
```
<Axes: title={'center': 'Gaussian Mixture Sum Square Dist'}, xlabel='Number of
Clusters', ylabel='Loss'>
```

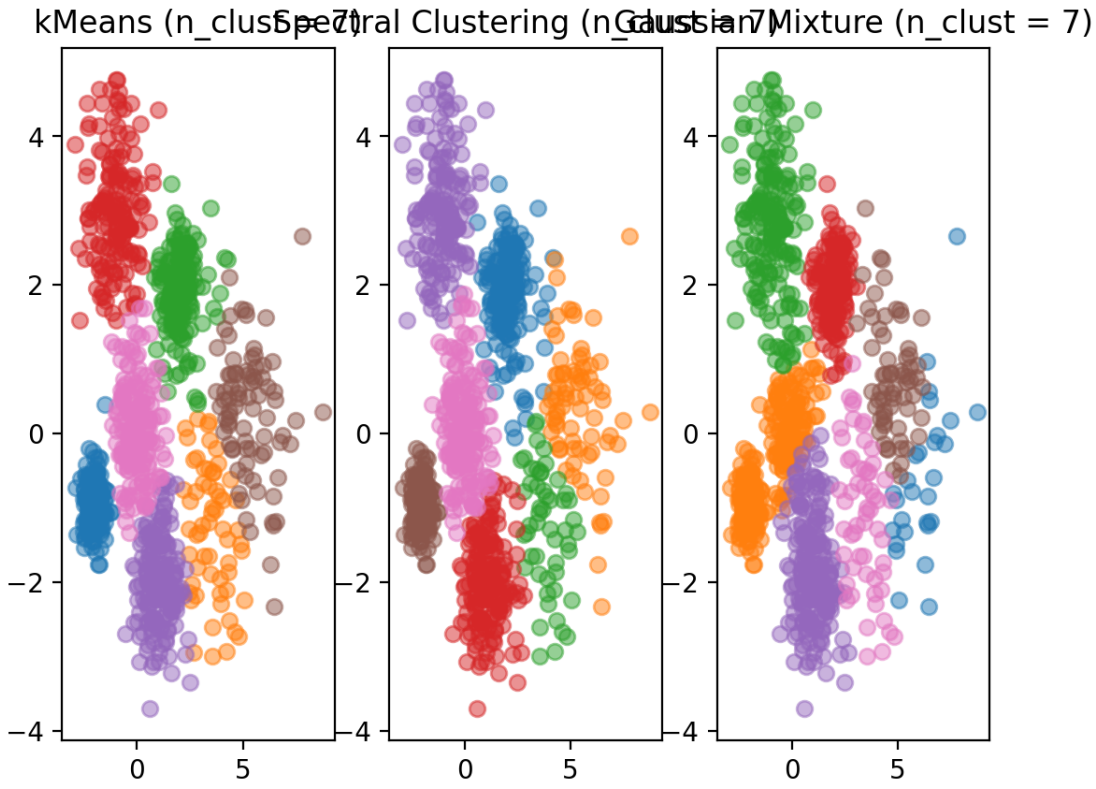

kMeans (n_clust = 2)
Spectral Clustering (n_clust = 2)
Gaussian Mixture (n_clust = 2)
4
2
0
−2
−4
0
5

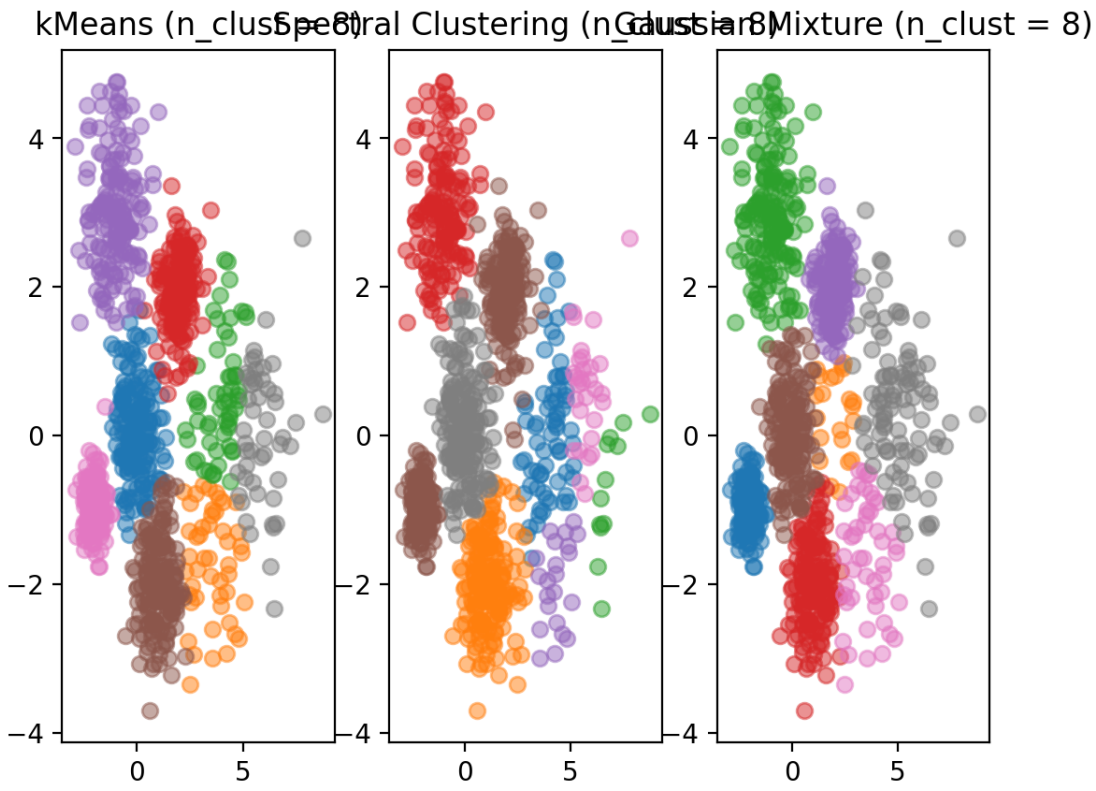

kMeans (n_clust = 3)
Spectral Clustering (n_clust = 3)
Gaussian Mixture (n_clust = 3)
4
2
0
−2
−4
0
5

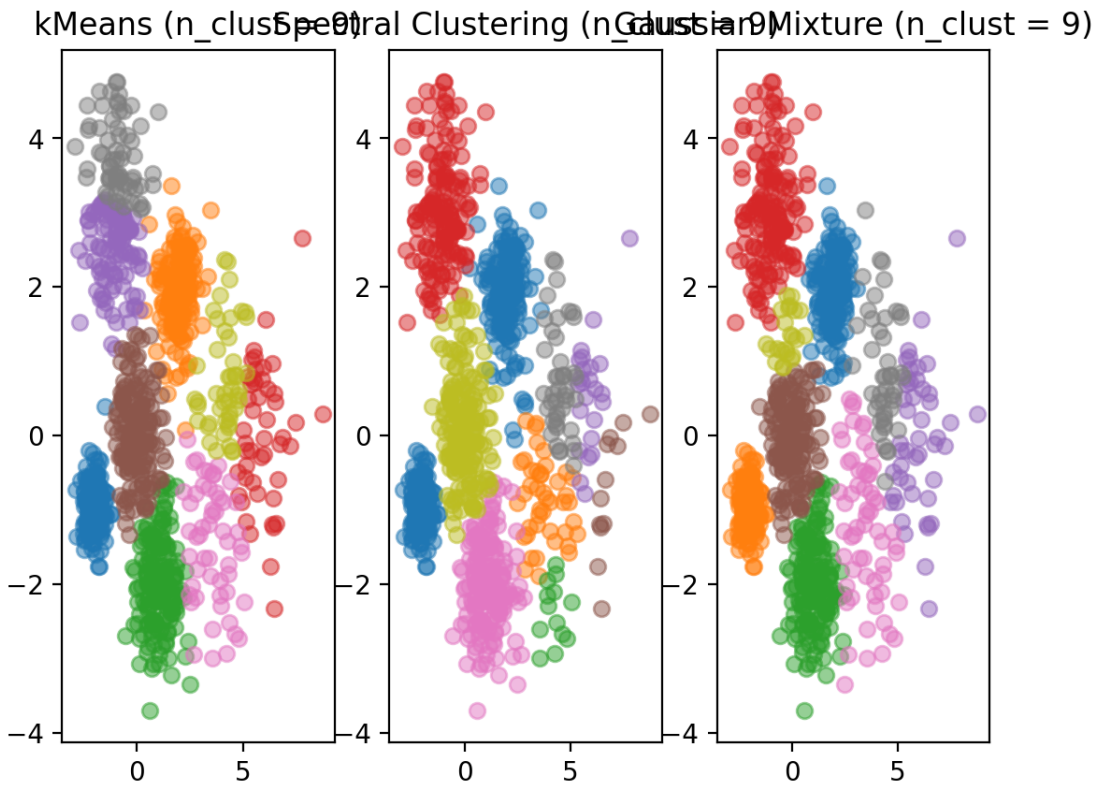

kMeans (n_clust = 4)
Spectral Clustering (n_clust = 4)
Gaussian Mixture (n_clust = 4)
4
2
0
−2
−4
0
5

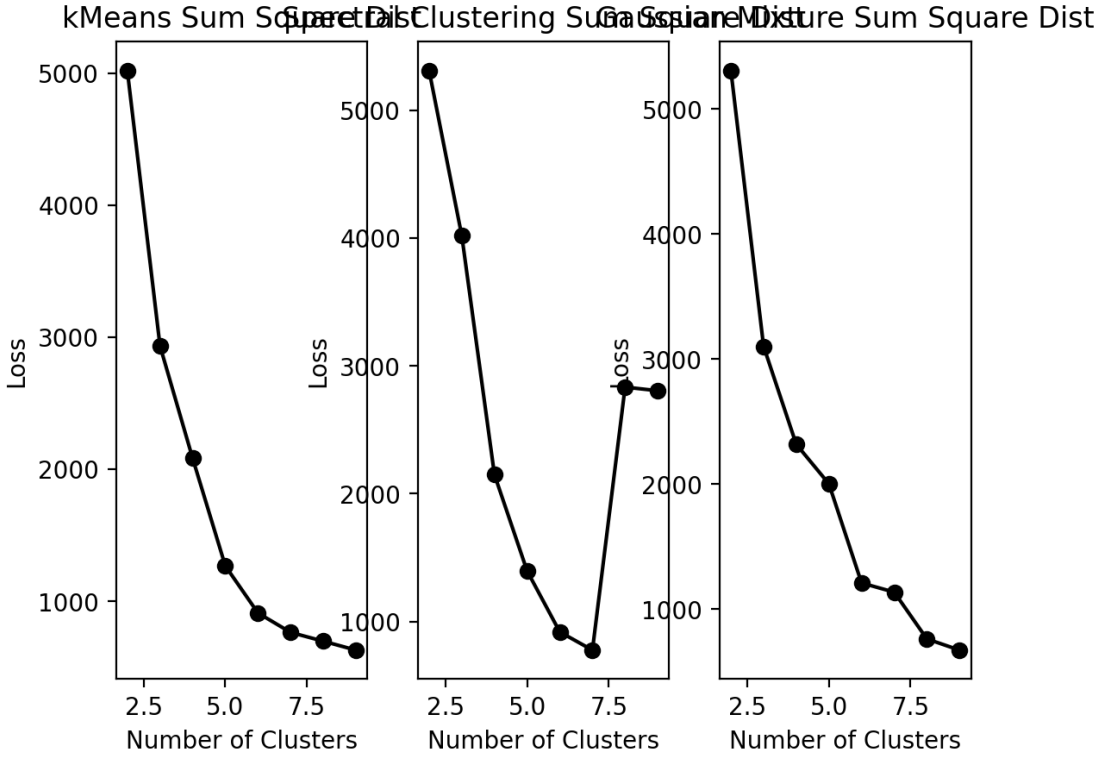

kMeans (n_clust = 5)
Spectral Clustering (n_clust = 5)
Gaussian Mixture (n_clust = 5)
4
2
0
−2
−4
0
5


kMeans (n_clust = 6)
Spectral Clustering (n_clust = 6)
Gaussian Mixture (n_clust = 6)
4
2
0
−2
−4
0
5


kMeans (n_clust = 7)
Spectral Clustering (n_clust = 7)
Gaussian Mixture (n_clust = 7)
4
2
0
−2
−4
0
5


kMeans (n_clust = 8)
Spectral Clustering (n_clust = 8)
Gaussian Mixture (n_clust = 8)
4
2
0
−2
−4
0
5


kMeans (n_clust = 9)
Spectral Clustering (n_clust = 9)
Gaussian Mixture (n_clust = 9)


kMeans Sum Square Dist
Spectral Clustering Sum Square Dist
Gaussian Mixture Sum Square Dist
Loss
Number of Clusters

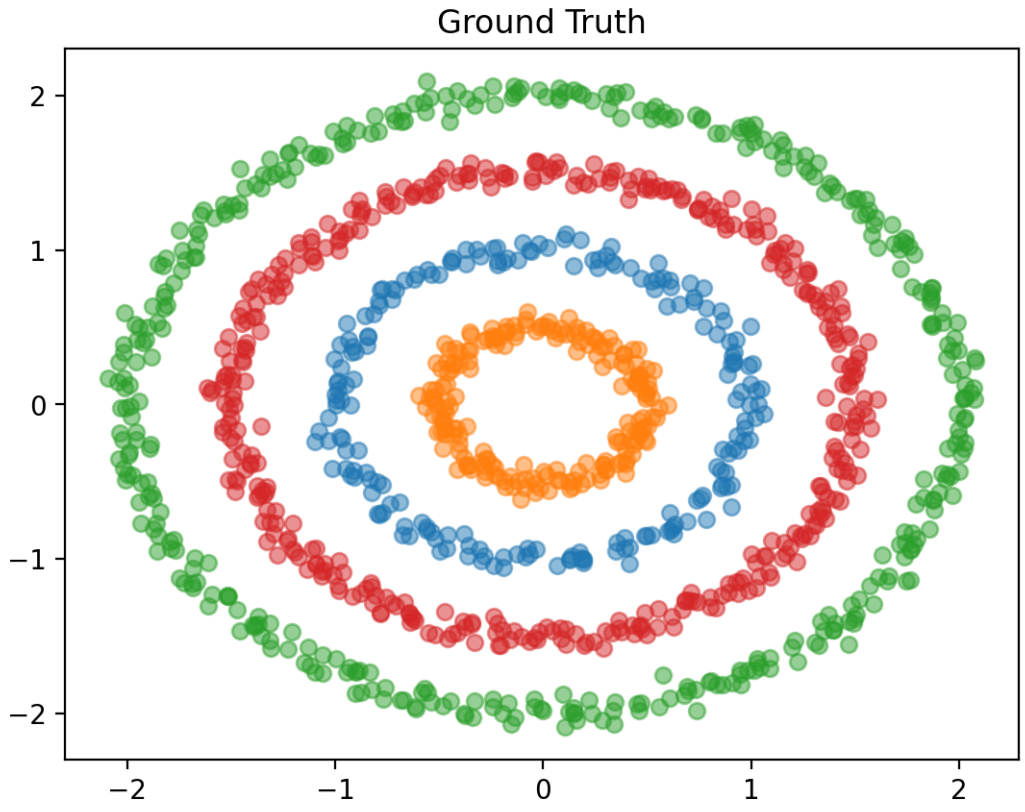
0.3 Concentric circles dataset

Visualize the "blob" dataset generated below, using a unique color for each cluster of points, where y contains the label of each corresponding point in x.

[ ]:
```
## DO NOT MODIFY
x1, y1 = make_circles(n_samples = 400, noise = 0.05, factor = 0.5, random_state␣
↪= 0)
x2, y2 = make_circles(n_samples = 800, noise = 0.025, factor = 0.75,␣
↪random_state = 1)

x = np.vstack([x1, x2*2])
y = np.hstack([y1, y2+2])
```

[ ]:
```
fig, ax = plt.subplots()
plot_pred(x, y, ax, "Ground Truth")
```

[ ]:
```
<Axes: title={'center': 'Ground Truth'}>
```

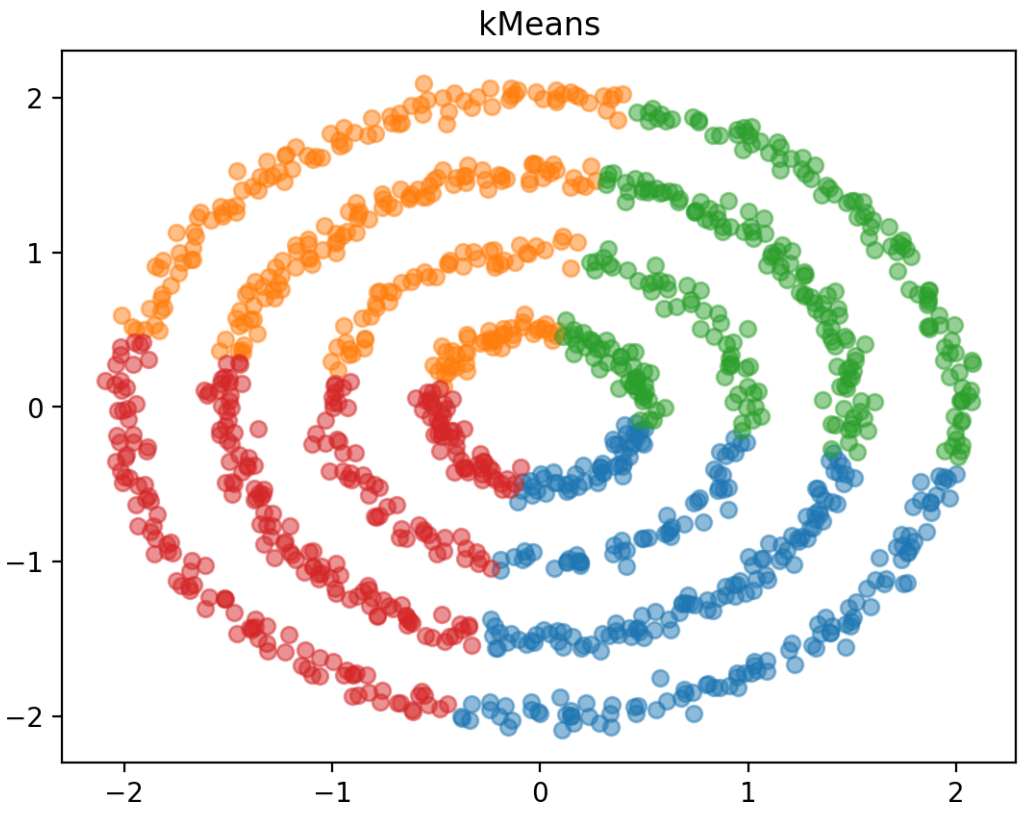


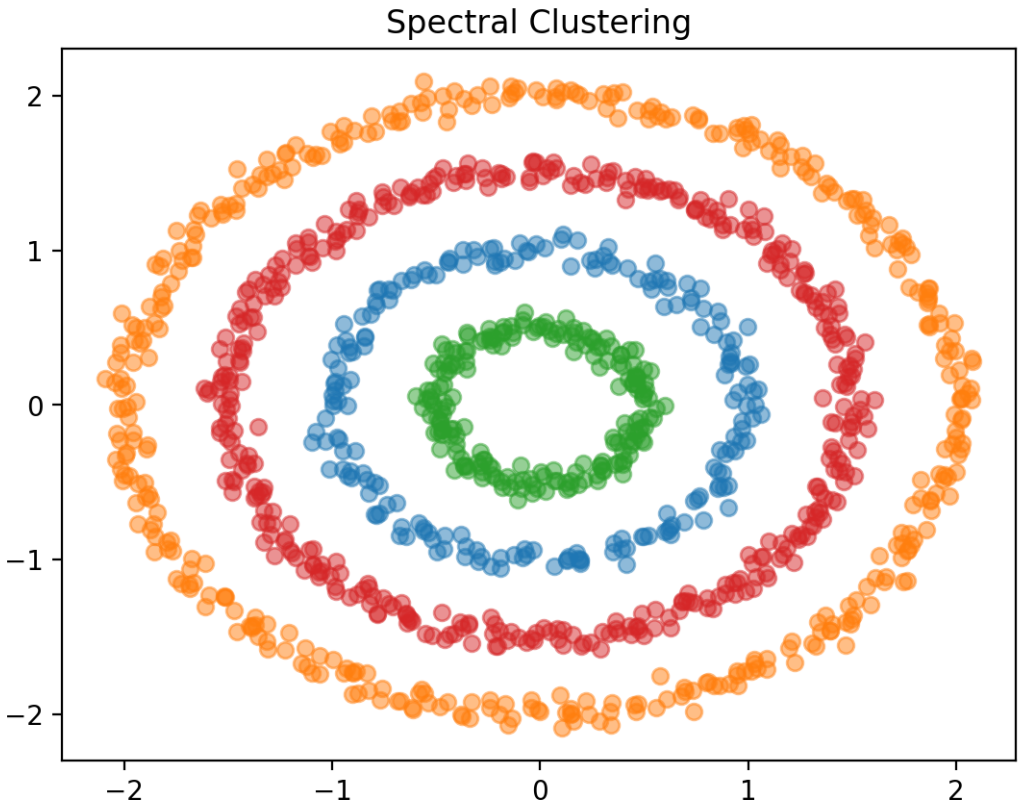
Use the `sklearn` KMeans, Spectral Clustering, and Gaussian Mixture Model functions to cluster the concentric circle data with 4 clusters, and attempt to modify the parameters until you get satisfactory results. Note: you should get good clustering results with Spectral Clustering, but the KMeans and GMM models will struggle to cluster this dataset well. Plot the results of your three models side-by-side using `plt.subplots` and the provided `plot_pred(x, labels, ax, title)` function.

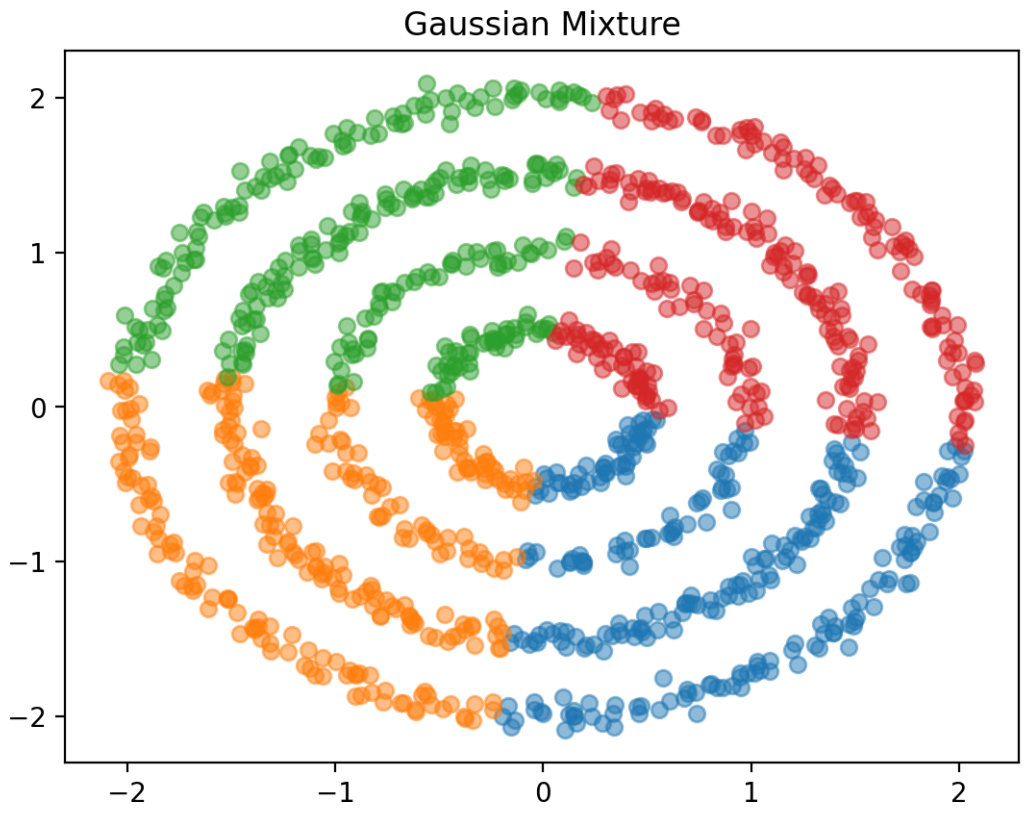
[ ]:
```
fig, ax = plt.subplots()
model1 = KMeans(n_clusters=4).fit(x)
plot_pred(x, model1.labels_, ax, "kMeans")

fig, ax = plt.subplots()
model2 = SpectralClustering(n_clusters=4, affinity='nearest_neighbors',␣
 ↪n_neighbors=15).fit(x)
plot_pred(x, model2.labels_, ax, "Spectral Clustering")

fig, ax = plt.subplots()
model3 = GaussianMixture(n_components=4, covariance_type='spherical').fit(x)
plot_pred(x, model3.predict(x), ax, "Gaussian Mixture")
```

```
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-
packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
```

[ ]:
```
<Axes: title={'center': 'Gaussian Mixture'}>
```


kMeans
2
1
0
−1
−2
−2
−1
0
1
2


Spectral Clustering
2
1
0
−1
−2
−2
−1
0
1
2



Using the parameters you found for the three models above, run each of the clustering algorithms for `n_clust = [2,3,4,5,6,7,8,9]` and compute the sum of squared distances loss for each case using the provided `compute_loss(x, labels)` function, where labels is the cluster assigned to each point by the algorithm. Plot loss versus number of cluster for each your three models in side-by-side subplots using the provided `plot_pred(x, labels, ax, title)` function.

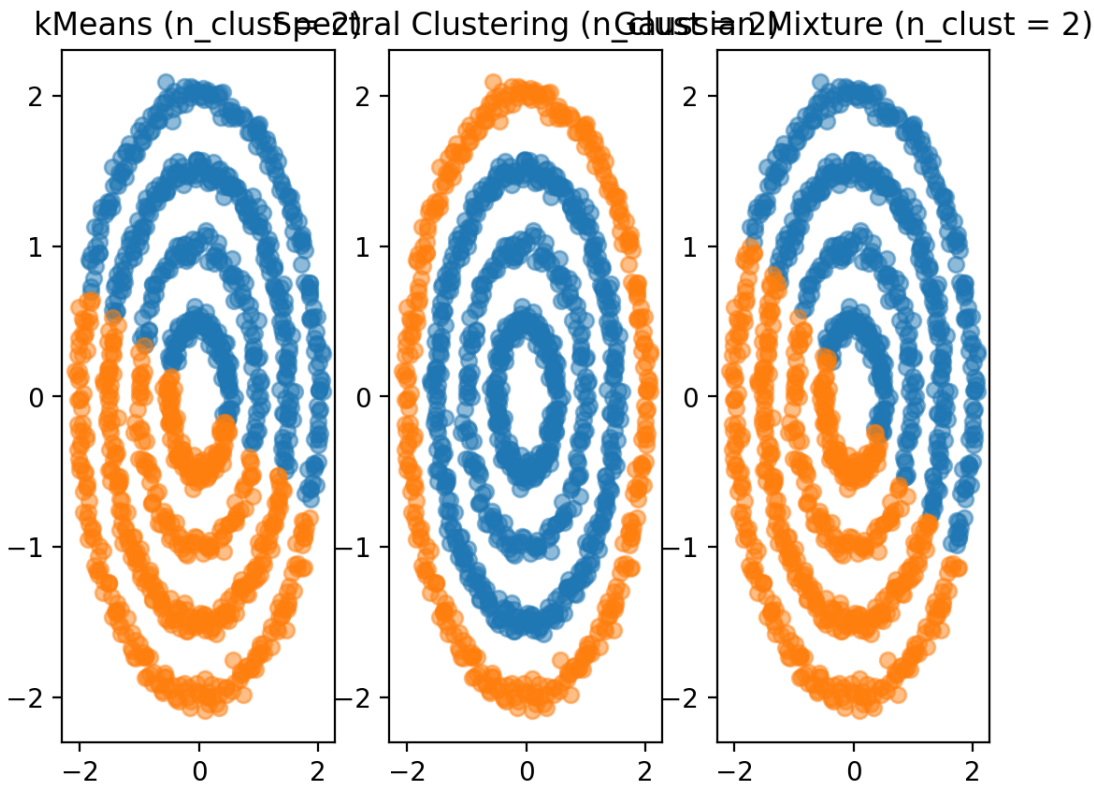
[ ]:
```
model1_sum_square_dist = []
model2_sum_square_dist = []
model3_sum_square_dist = []

for n_clust in [2, 3, 4, 5, 6, 7, 8, 9]:
    #Train models
    model1 = KMeans(n_clusters=n_clust).fit(x)
    model2 = SpectralClustering(n_clusters=n_clust,␣
 ↪affinity='nearest_neighbors', n_neighbors=10).fit(x)
    model3 = GaussianMixture(n_components=n_clust, covariance_type='spherical').
 ↪fit(x)

    model1_sum_square_dist.append(compute_loss(x, model1.labels_))
    model2_sum_square_dist.append(compute_loss(x, model2.labels_))
    model3_sum_square_dist.append(compute_loss(x, model3.predict(x)))
```

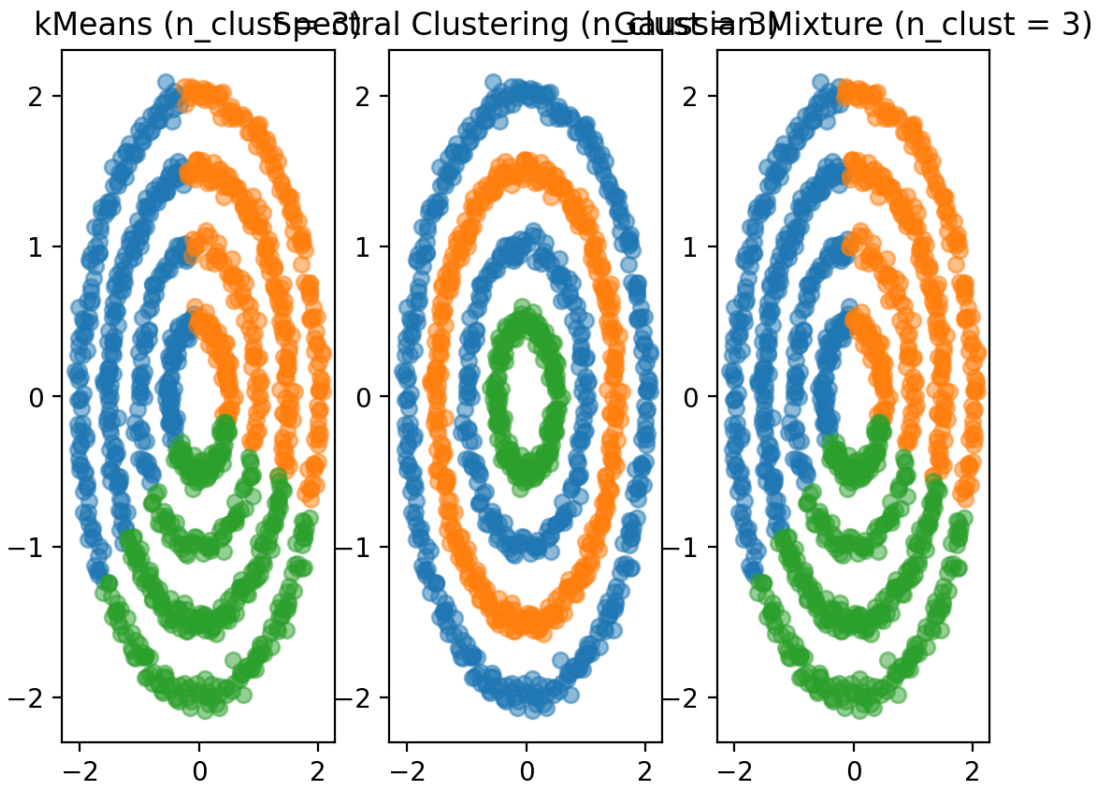
```
    fig, ax = plt.subplots(1,3)
    plot_pred(x, model1.labels_, ax[0], f"kMeans (n_clust = {n_clust})")
    plot_pred(x, model2.labels_, ax[1], f"Spectral Clustering (n_clust =␣
↪{n_clust})")
    plot_pred(x, model3.predict(x), ax[2], f"Gaussian Mixture (n_clust =␣
↪{n_clust})")

fig, ax = plt.subplots(1,3)
plot_loss(model1_sum_square_dist, ax[0], "kMeans Sum Square Dist")
plot_loss(model2_sum_square_dist, ax[1], "Spectral Clustering Sum Square Dist")
plot_loss(model3_sum_square_dist, ax[2], "Gaussian Mixture Sum Square Dist")
```

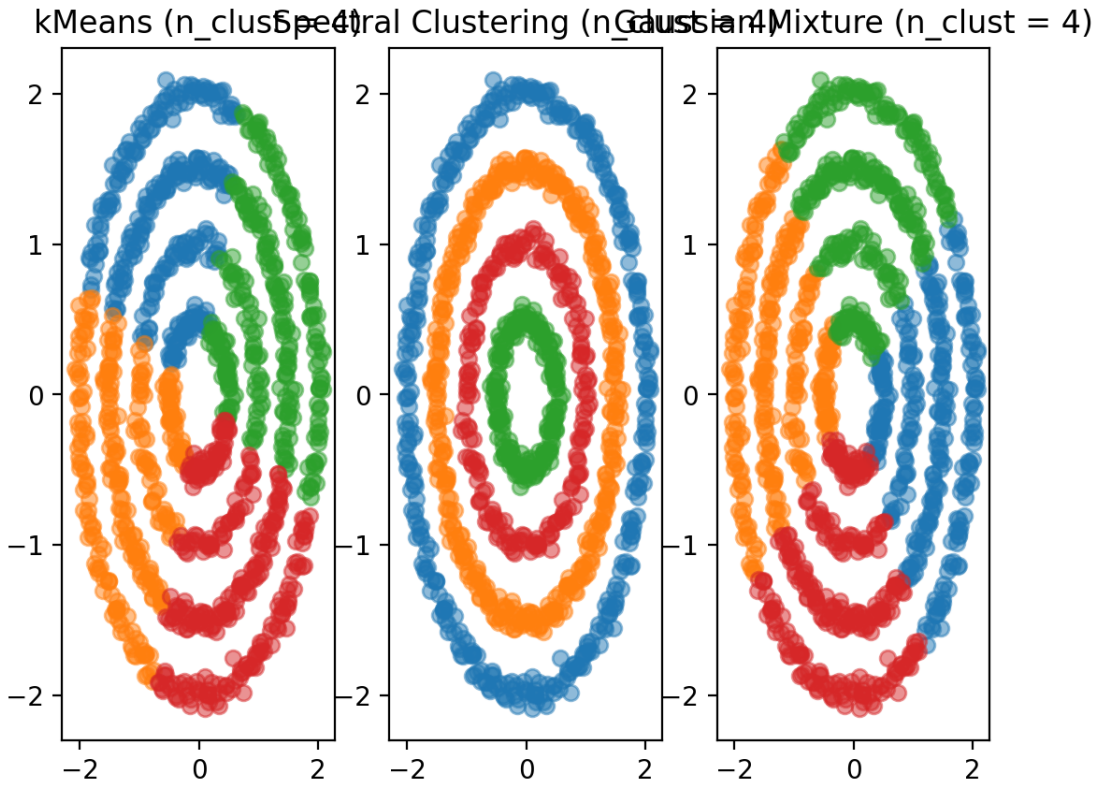
```
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-
packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-
packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-
packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-
packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
```

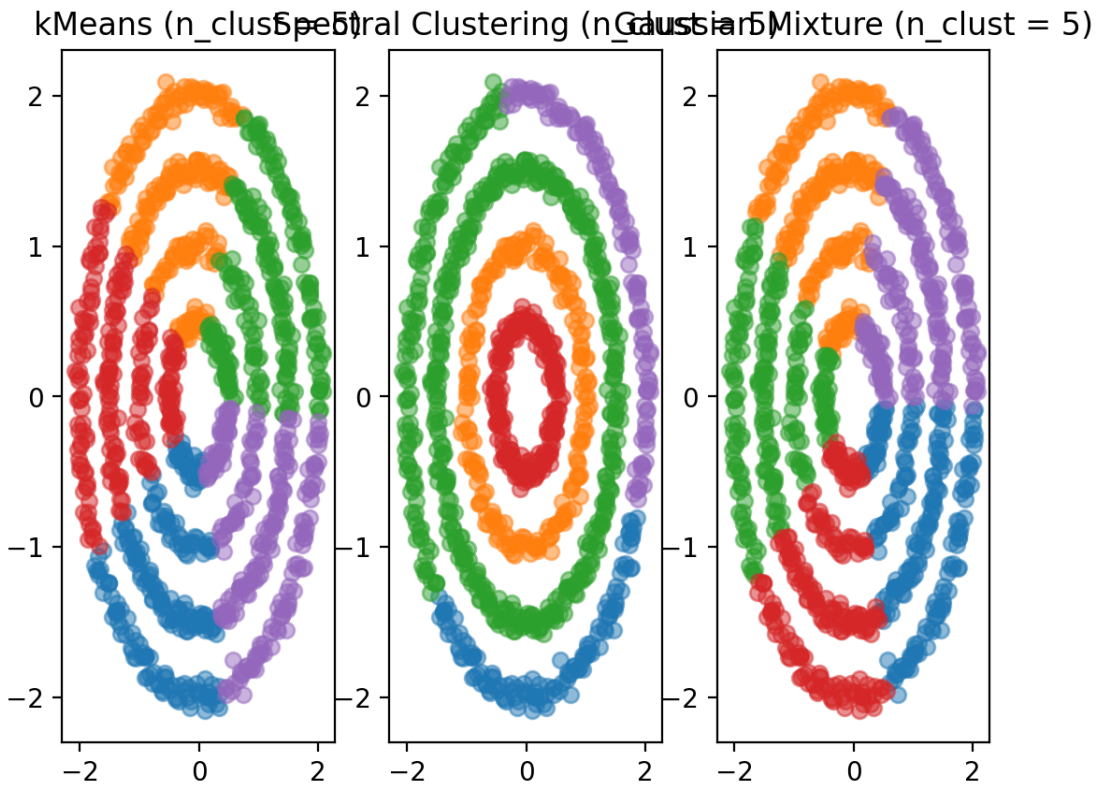
```
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-
packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-
packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-
packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
/opt/miniconda3/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1412:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/opt/miniconda3/lib/python3.8/site-
packages/sklearn/manifold/_spectral_embedding.py:273: UserWarning: Graph is not
fully connected, spectral embedding may not work as expected.
  warnings.warn(
```

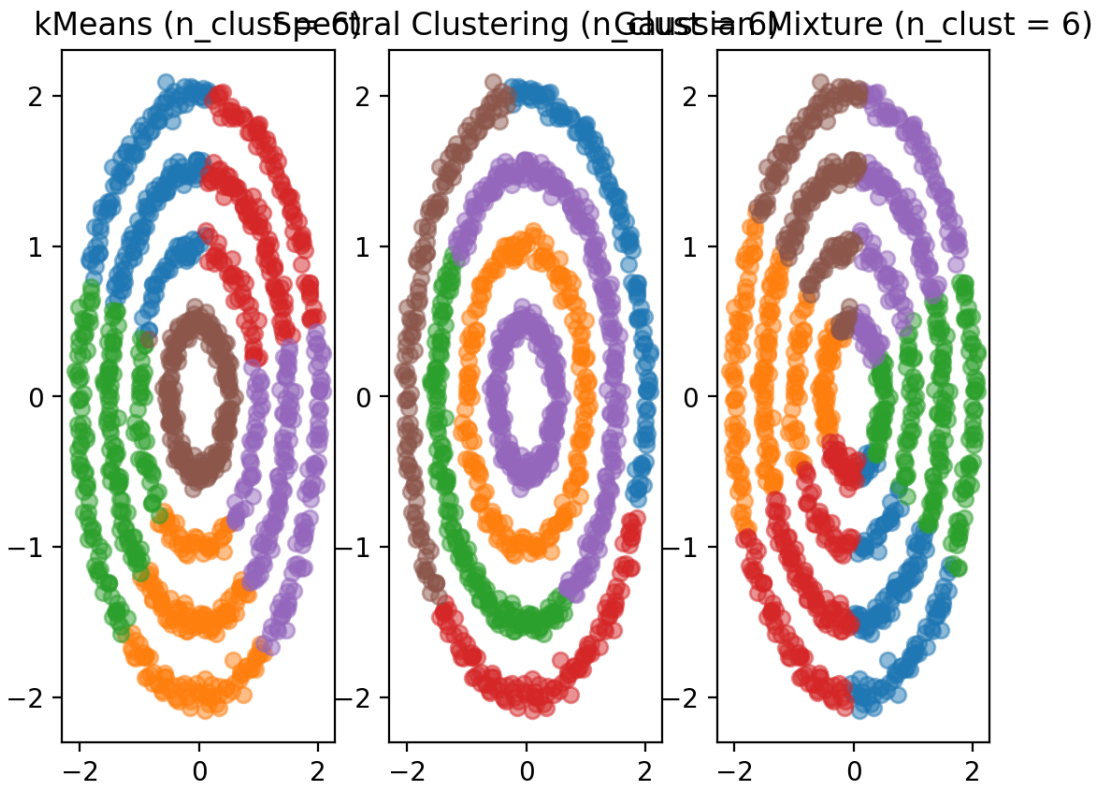
```
[ ]: <Axes: title={'center': 'Gaussian Mixture Sum Square Dist'}, xlabel='Number of
     Clusters', ylabel='Loss'>
```

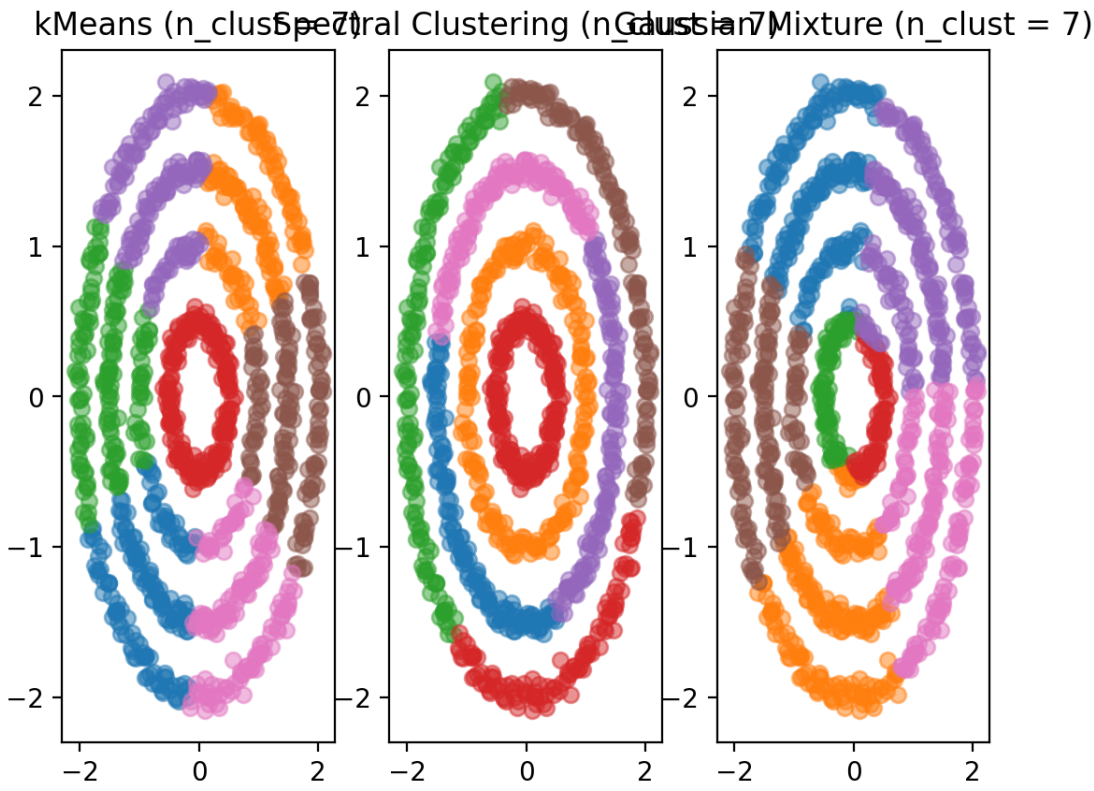

kMeans (n_clust = 2)
Spectral Clustering (n_clust = 2)
Gaussian Mixture (n_clust = 2)
2
1
0
−1
−2
−2
0
2

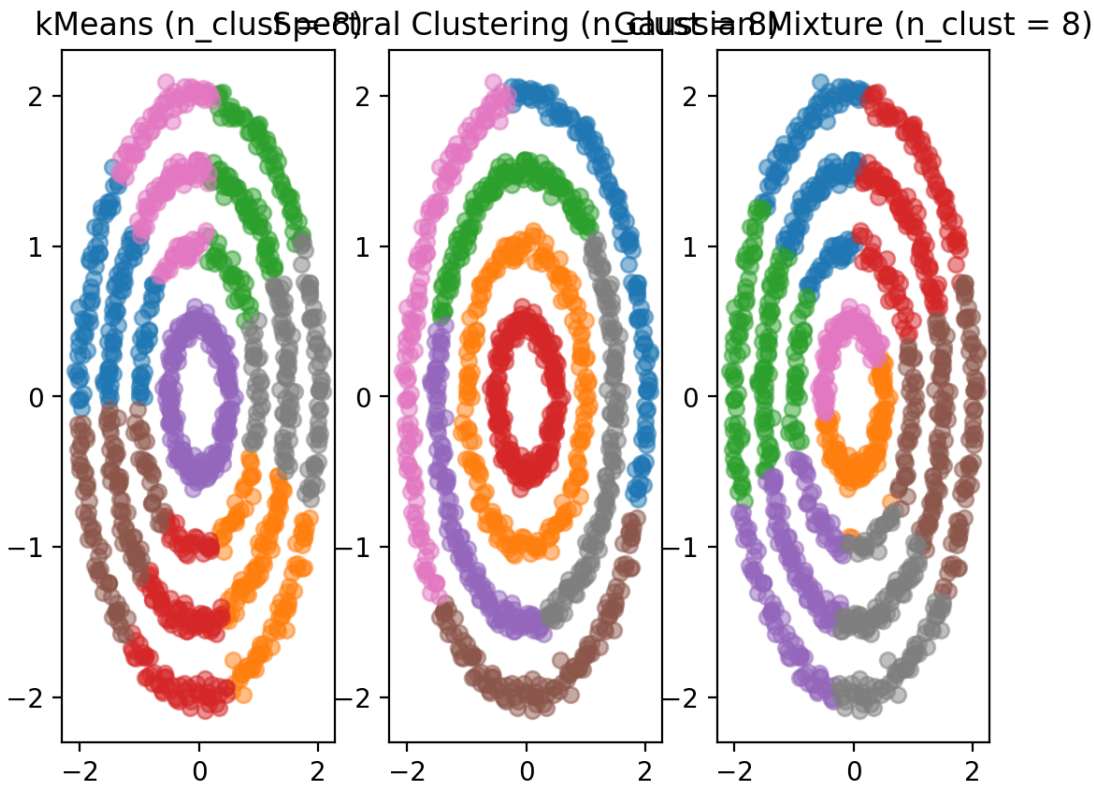

kMeans (n_clust = 3)
Spectral Clustering (n_clust = 3)
Gaussian Mixture (n_clust = 3)
2
1
0
−1
−2
−2
0
2

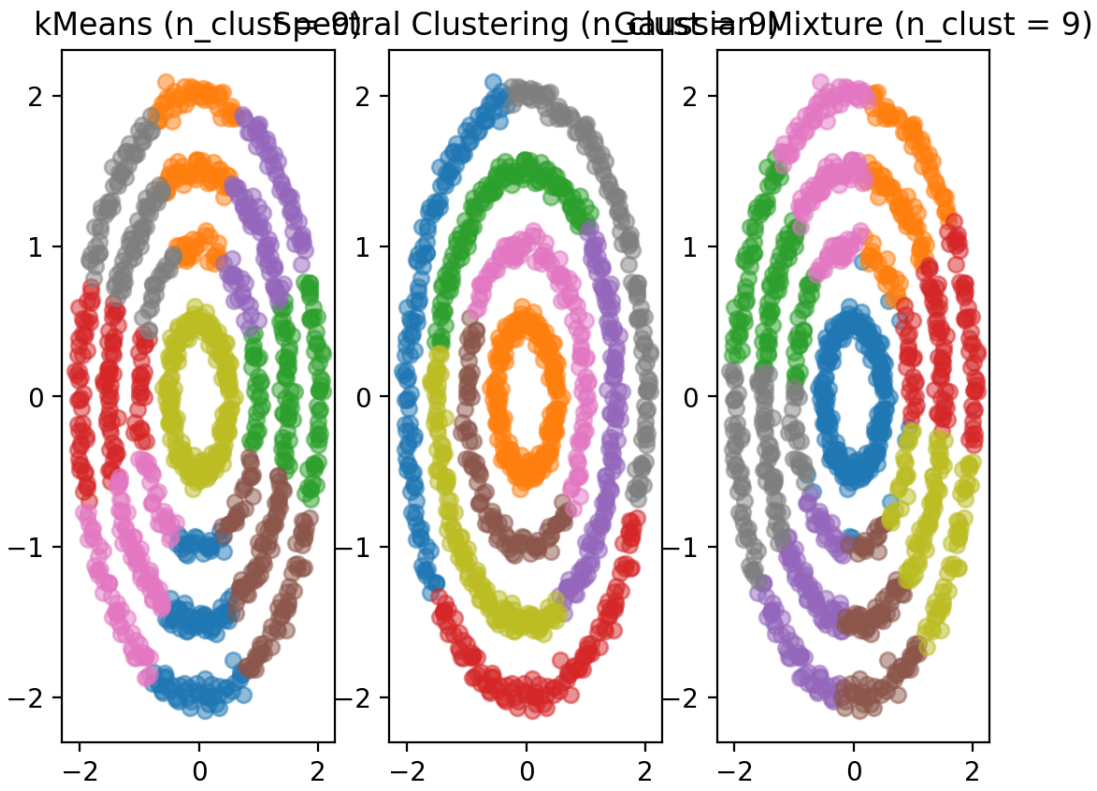

kMeans (n_clust = 4)
Spectral Clustering (n_clust = 4)
Gaussian Mixture (n_clust = 4)
2
1
0
−1
−2
−2
0
2

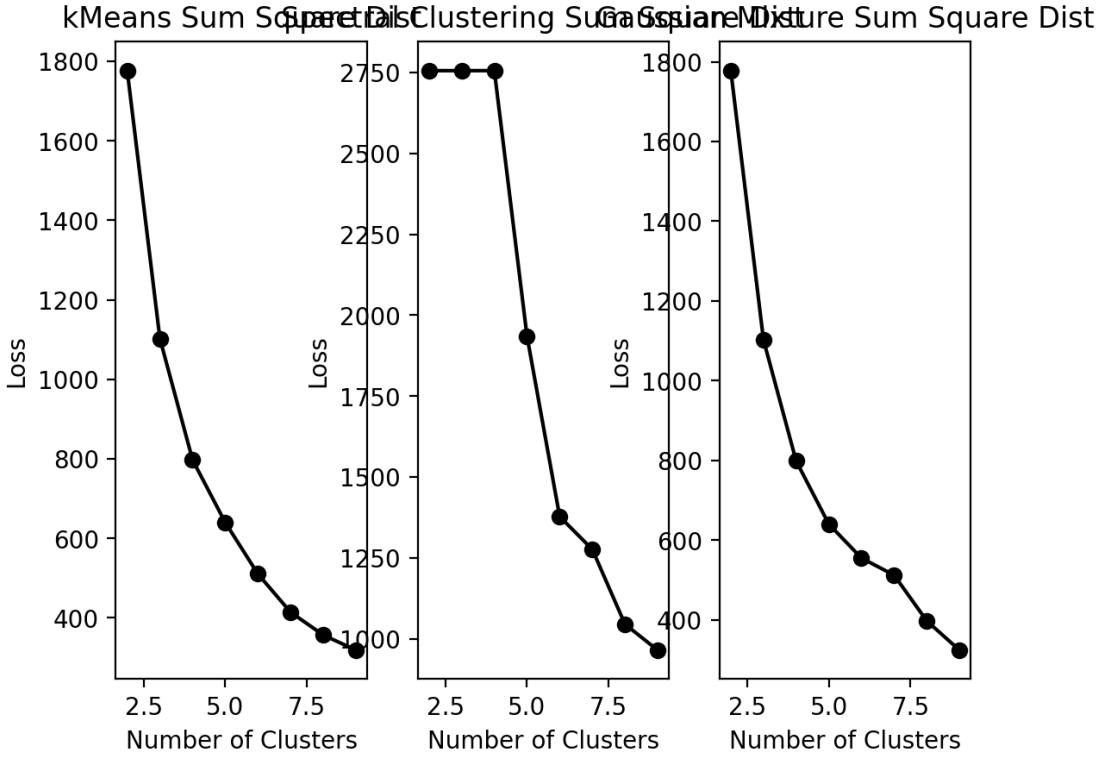

kMeans (n_clust = 5)
Spectral Clustering (n_clust = 5)
Gaussian Mixture (n_clust = 5)
2
1
0
−1
−2
−2
0
2


kMeans (n_clust = 6)
Spectral Clustering (n_clust = 6)
Gaussian Mixture (n_clust = 6)
2
1
0
−1
−2
−2
0
2


kMeans (n_clust = 7)
Spectral Clustering (n_clust = 7)
Gaussian Mixture (n_clust = 7)
2
1
0
−1
−2
−2
0
2


kMeans (n_clust = 8)
Spectral Clustering (n_clust = 8)
Gaussian Mixture (n_clust = 8)
2
1
0
−1
−2
−2
0
2


kMeans (n_clust = 9)
Spectral Clustering (n_clust = 9)
Gaussian Mixture (n_clust = 9)


kMeans Sum Square Dist
Spectral Clustering Sum Square Dist
Gaussian Mixture Sum Square Dist
Loss
Number of Clusters

0.4 Discussion

1. Discuss the performance of the clustering algorithms on the "blob" dataset. Using the elbow method, were you able to identify the number of natural clusters in the dataset for each of the methods? Does the elbow method work better for some algorithms versus others?

The elbow method was only able to be used on the spectral clustering model. For the kMeans and the Gaussian mixture models, the loss kept decreasing as the number of clusters increased because it was just over fitting the data. This is only able to be viewed when looking at the loss of each model.

2. Discuss the performance of the clustering algorithms on the concentric circles dataset. Using the elbow method, were you able to identify the number of natural clusters in the dataset for each of the methods?

The elbow method was not useful at all for determing the number of clusters in the concentric dataset. In terms of performance n the data, the spectral clustering algorithm performed the best, being the only one to accurately fit to the ring shape at the best number of clsuters (4). The other methods weren't able to separate these rings out from one another andn instead blended them together.

3. Does the sum of squared distances work well as a loss function for each of the three clustering algorithms we implemented? Does the sum of squared distance fail on certain types of clusters?

The sum of squares method works well when it comes to the blob datasets but not when it comes to the concentric datasets. In terms of the clustering algorithms, it doesn't seem to have a great advantage for any of the algorithms used.