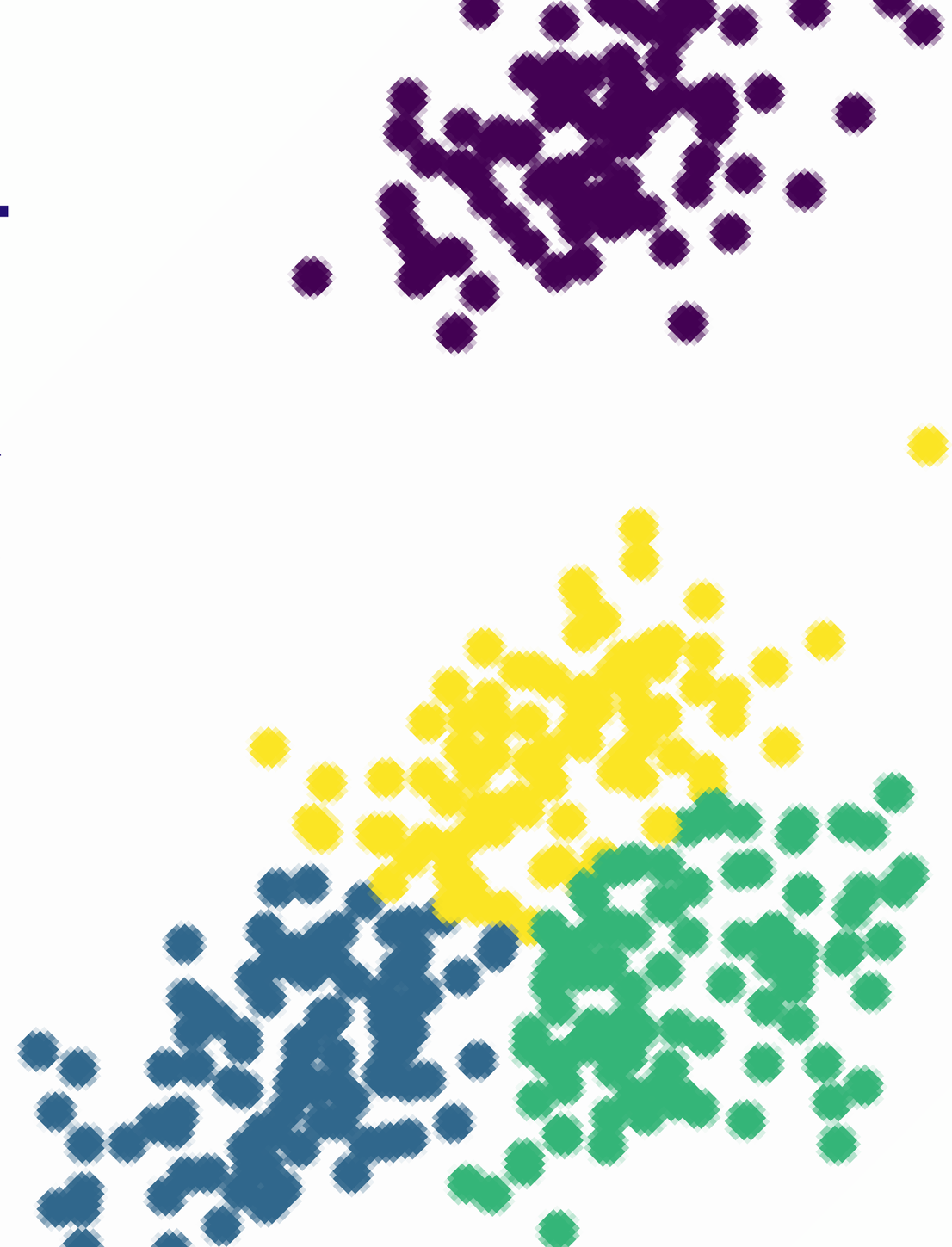
UNIVERSITY OF JEDDAH

جامعة جدة
University of Jeddah

MACHINE LEARNING PROJECT

KMEAN & DBSCAN

Algorithm Implementation and Visualization



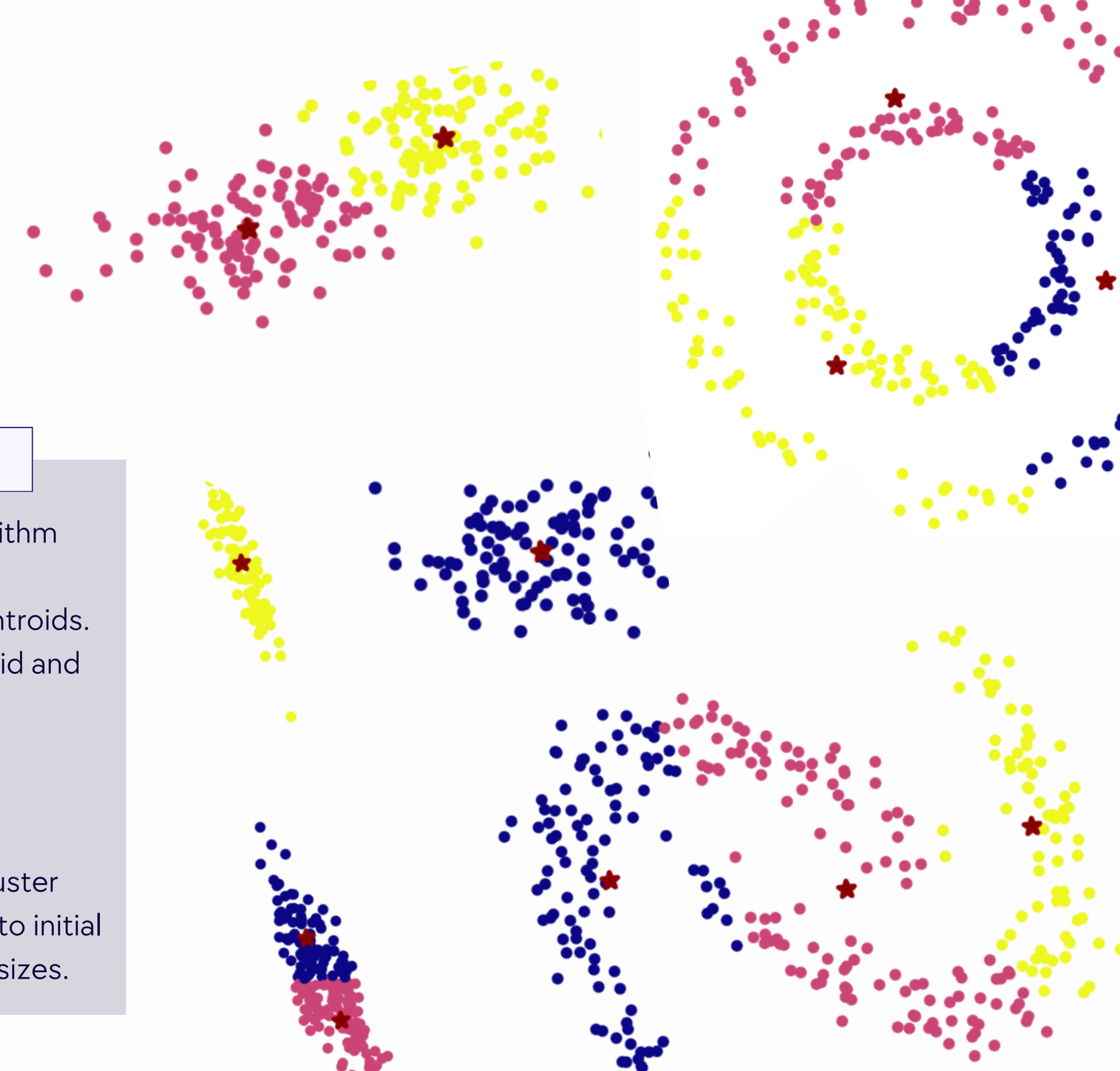
Introduction

In this unsupervised machine learning project, we implement two popular clustering algorithms, K-means and DBSCAN, from scratch. Clustering is a fundamental technique in unsupervised learning that groups data points based on similarity. This project offers an opportunity to understand unsupervised learning through the implementation and visualization of K-means and DBSCAN algorithms.

K-MEAN ALGORITHM

Create K-mean Algorithm from Scratch

k-means algorithm is an unsupervised clustering algorithm that groups data points into k clusters based on their similarity. It starts by randomly initializing k cluster centroids. Then, it assigns each data point to the nearest centroid and updates the centroids by calculating the mean of the assigned data points. This process is repeated until convergence, where the centroids no longer move significantly. Finally, the algorithm outputs the cluster assignments. K-means aims to minimize the within-cluster variance and has some limitations, such as sensitivity to initial positions and assumptions about cluster shapes and sizes.



Datasets:

1st Dataset: Blobs dataset

2nd Dataset: Anisotropicly dataset

3rd Dataset: noisy moon dataset

4th Dataset: noisy circles dataset

Attributes:

n_samples = 300

random_state :

Student1 id: 2115590	Student2 id: 2111953	Student3 id: 2110006	Student4 id: 2110149	Student5 id: 2111489
2+1+1+5+5+9+0= 23	2+1+1+1+9+5+3= 22	2+1+1+6= 10	2+1+1+1+4+9 = 18	2+1+1+1+4+8+9= 26

23+22+10+18+26 = 99 random states

class KmeanClustering:

1.def initialize_random_centroids(self, X):
This method creates an array that contains the coordinates of the randomly initialized centroids.

```python
def initialize_random_centroids(self, X):
    centroids = np.zeros((self.K, self.num_features)) # creates a numpy array that will sort the centroid coordinates

    for k in range(self.K): # itirate over the num of clusters
        centroid = X[np.random.choice(range(self.num_examples))] # initilize the centroid from randomly selecting a datapoint in the dataset
        centroids[k] = centroid # put the centroids inside the centroid array

    return centroids # return array of the initilize centroids for each cluster
```

```python
def create_clusters(self, X, centroids):
    # Will contain a list of the points that are associated with that specific cluster
    clusters = [[] for _ in range(self.K)] # initilize an empty list for each cluster, each list contains a list of the indices of the datapoint that belongs to that cluster

    # Loop through each point and check which is the closest cluster
    for point_idx, point in enumerate(X): # calculate distance of the points from the centroid
        closest_centroid = np.argmin( # np.argmin gives the minimum value in the area finding the centroid
            np.sqrt(np.sum((point - centroids) ** 2, axis=1)) # euclidean distance calculate everypoint and each centroid
        )
        clusters[closest_centroid].append(point_idx) # assign each datapoint to the list of closest centroid

        # the clusters are built where each cluster is a list of indices of datapoints assigned to that cluster
    return clusters # return the list of the list clusters where each inner list contains the indexs of the datapoints that belong to that cluster
```

2.def create_clusters(self, X, centroids):
This method creates a list for each centroid that contains the datapoints close to it using the euclidean distance creating a cluster.

3.def calculate_new_centroids(self, clusters, X):
This method finds the centroids by creating a list for each centroid that contains the datapoints close to it using the euclidean distance creating a cluster.

```python
def calculate_new_centroids(self, clusters, X):
    centroids = np.zeros((self.K, self.num_features)) # create an array of zeros corresponding the number of clusters
    for idx, cluster in enumerate(clusters): # for each current cluster
        new_centroid = np.mean(X[cluster], axis=0) # compute the mean of the datapoints in the clusters in the columns
        centroids[idx] = new_centroid # update the centroids for each cluster

    return centroids # return an array of the updated centroids
```

class KmeanClustering:

```python
def reassign_cluster(self, clusters, X):
    y_pred = np.zeros(self.num_examples) # initilize an array with zeros for datapoint for initializeing the cluster

    for cluster_idx, cluster in enumerate(clusters): # for each cluster index in cluster i, iterate over every cluster
        for sample_idx in cluster: # for each datapoint in the cluster i
            y_pred[sample_idx] = cluster_idx # assign each datapoint to the predicted cluster i

    return y_pred # return an array an array containing the predicted cluster label for each datapoint
```

4.def reassign_cluster(self, clusters, X):
This method creates an array of zeros for datapoints and reassigns every datapoint to the predicted clusters.

5.def fit(self, X):
This method iteratively improves the assignment of the clusters and their centroids. It initializes, updates, reassigns centroids and their datapoints and provides label predictions by calling the methods in the class. Calculates the difference between the new and previous centroid, if there is no difference then the algorithm reached convergence.

```python
def fit(self, X):
    centroids = self.initialize_random_centroids(X) # initilize the centroid by the calling function

    for i in range(self.max_iterations): # update centroid and ressign until reaching max_iteration
        clusters = self.create_clusters(X, centroids) # assign each datapoint to the closest centroid

        previous_centroids = centroids # store the current centroid as the previous centroid
        centroids = self.calculate_new_centroids(clusters, X) # calculate the new centroid based on the new centroid

        diff = centroids - previous_centroids # calculate the diff between the new and previous centroid

        if not diff.any(): # if there is no diff then terminate
            print("Termination criterion satisfied")
            break

    # Get label predictions
    y_pred = self.reassign_cluster(clusters, X)

    return y_pred, centroids # return the pedicted label for each datapoint
```

def plot_clusters(X, y_pred, centroids)

```python
def plot_clusters(X, y_pred, centroids):

    plt.figure(figsize=(18, 6))

    # Plot dataset without labels
    plt.subplot(1, 3, 1)
    plt.scatter(X[:, 0], X[:, 1], c='cornflowerblue')
    plt.title('Clustering Dataset (no labels)')
    plt.xlabel('X')
    plt.ylabel('Y')

    # Plot dataset with predicted labels
    plt.subplot(1, 3, 2)
    plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='plasma')
    plt.title('Clustering Dataset (with predicted labels)')
    plt.xlabel('X')
    plt.ylabel('Y')

    # Plot k-means clustering with centroids in red
    plt.subplot(1, 3, 3)
    plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='plasma')
    plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='*', edgecolor='darkred', linewidth=3, s=100, label='Kmeans.Centroid')
    plt.legend()
    plt.title('K-means Clustering (with k = 3)')
    plt.xlabel('X')
    plt.ylabel('Y')

    plt.tight_layout()
    plt.show()
```

This method plots the dataset without label, with label, and with centroids.

Evaluation Metrices

Evaluation Metrices:

F-Measure:

- The F1 score is a balanced measure, representing the harmonic mean of precision and recall. It ranges from 0 to 1, with higher values indicating better performance (1 for perfection, 0 for poor performance). Valuable for balanced precision and recall considerations in classification evaluation.

$$F_i = \frac{2}{\frac{1}{prec_i} + \frac{1}{recall_i}} = \frac{2 \cdot prec_i \cdot recall_i}{prec_i + recall_i} = \frac{2\, n_{i j_i}}{n_i + m_{j_i}}$$

Evaluation Metrices:

Rand Index:

- The Rand Index evaluates the similarity between two clustering results, measuring the agreement between true and predicted pairwise relationships of data points. Ranging from 0 to 1, 0 signifies no agreement (random clustering), and 1 denotes perfect agreement. It's valuable for assessing clustering algorithms, especially when the ground truth is known.

$$RI(\mathcal{C},\mathcal{G}) = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{FN}+\mathrm{TN}},$$

Evaluation Metrices:

Normalized Mutual Information(NMI):

- Normalized Mutual Information (NMI) gauges the agreement between true and predicted labels, often used in clustering or classification evaluations. Ranging from 0 to 1, a higher NMI signifies increased agreement. It's commonly employed to assess how well clusters align with true classes in data.

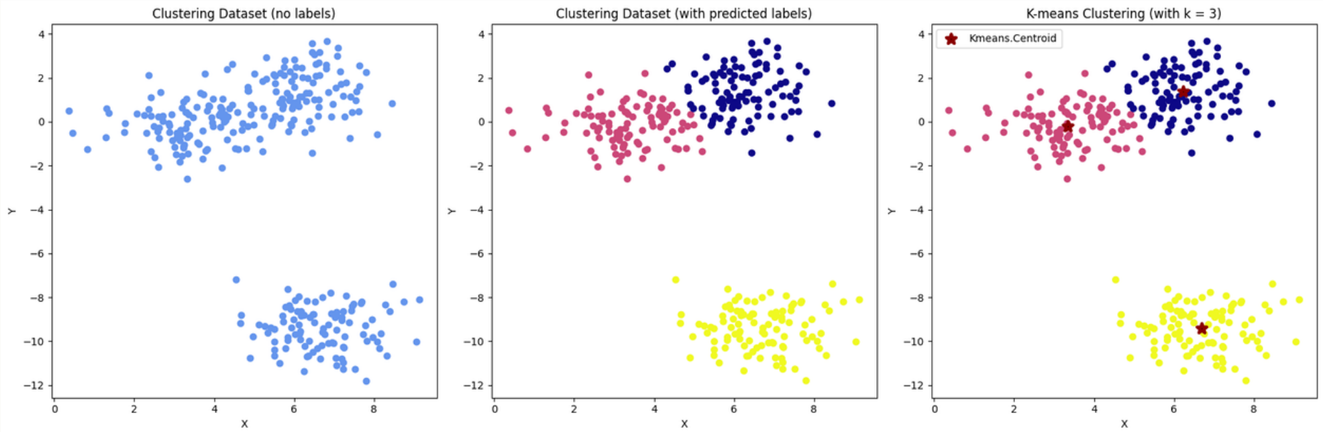
$$NMI(C,T) = \sqrt{\frac{I(C,T)}{H(C)} \cdot \frac{I(C,T)}{H(T)}} = \frac{I(C,T)}{\sqrt{H(C) \cdot H(T)}}$$

1st Dataset: Blobs dataset

```
n_samples = 300
random_state = 99
num_clusters = 3

X, y = datasets.make_blobs(n_samples=n_samples, random_state=random_state)

kmeans = KMeansClustering(X, num_clusters)
y_pred1, centroids = kmeans.fit(X)

plot_clusters(X, y_pred1, centroids)
```

Evaluation metrics:

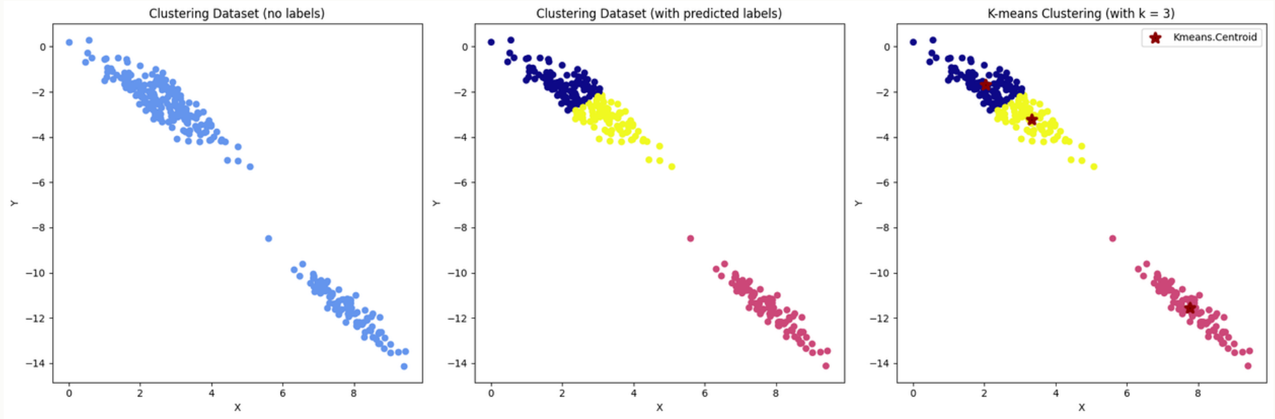
```
import pandas as pd
from sklearn import metrics

# Calculate evaluation metrics
f_measure_1k = metrics.f1_score(y1, y_pred1, average='weighted')
nmi_1k = metrics.normalized_mutual_info_score(y1, y_pred1)
rand_1k = metrics.rand_score(y1, y_pred1)
```



Evaluation Metrics DataSet 1	
Rand Index	0.7227
NMI	0.6489
F-measure	0.4669

2nd Dataset: Anisotropicly dataset

```
X, y = datasets.make_blobs(n_samples=n_samples,random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X = np.dot(X, transformation)

Kmeans = KMeansClustering(X, num_clusters)
y_pred2, centroids = Kmeans.fit(X)

plot_clusters(X, y_pred2, centroids)
```



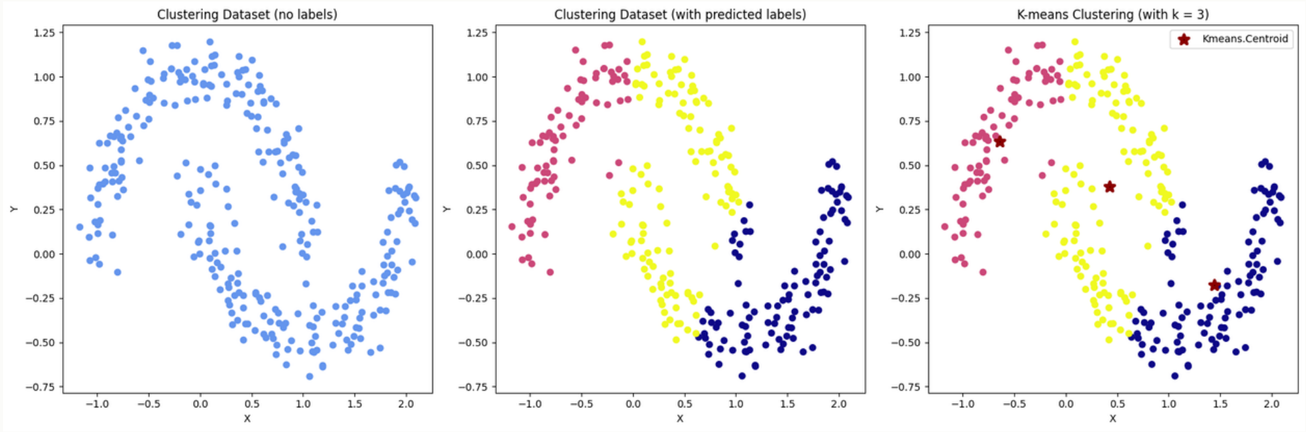
Evaluation metrics:

```
import pandas as pd
from sklearn import metrics

# Calculate evaluation metrics
f_measure_2k = metrics.f1_score(y2, y_pred2, average='weighted')
nmi_2k = metrics.normalized_mutual_info_score(y2, y_pred2)
rand_2k = metrics.rand_score(y2, y_pred2)
```

Evaluation Metrics DataSet 2	
Rand Index	0.7227
NMI	0.6489
F-measure	0.4669

3rd Dataset: noisy moon dataset

```
X, y = datasets.make_moons(n_samples=n_samples, noise=0.1,random_state=random_state)

Kmeans = KMeansClustering(X, num_clusters)
y_pred3, centroids = Kmeans.fit(X)
plot_clusters(X, y_pred3, centroids)
```



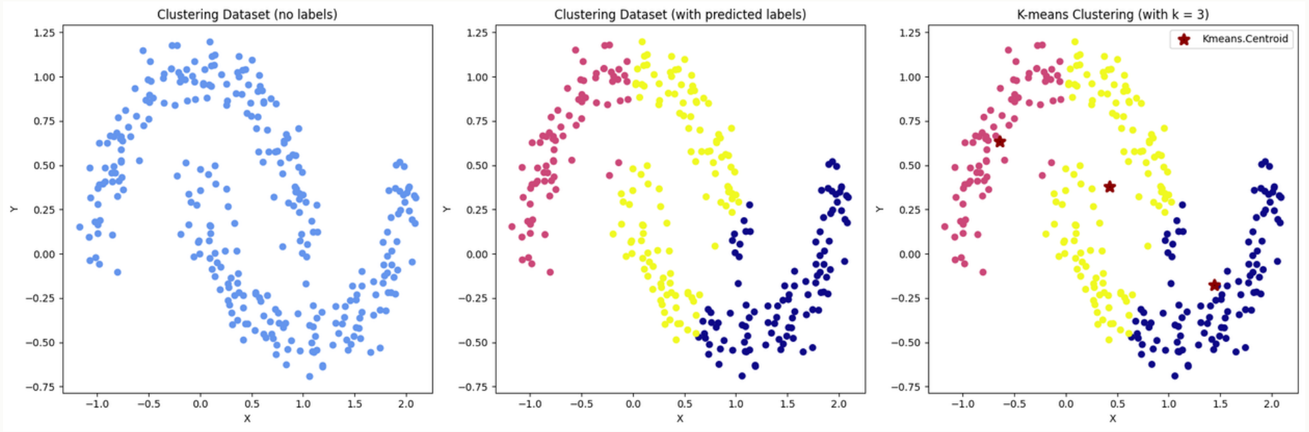
Evaluation metrics:

```
import pandas as pd
from sklearn import metrics

# Calculate evaluation metrics
f_measure_3k = metrics.f1_score(y3, y_pred3, average='weighted')
nmi_3k = metrics.normalized_mutual_info_score(y3, y_pred3)
rand_3k = metrics.rand_score(y3, y_pred3)
```

Evaluation Metrics DataSet 3	
Rand Index	0.6305
NMI	0.3214
F-measure	0.0411

3rd Dataset: noisy moon dataset

```
X, y = datasets.make_moons(n_samples=n_samples, noise=0.1,random_state=random_state)

Kmeans = KMeansClustering(X, num_clusters)
y_pred3, centroids = Kmeans.fit(X)
plot_clusters(X, y_pred3, centroids)
```



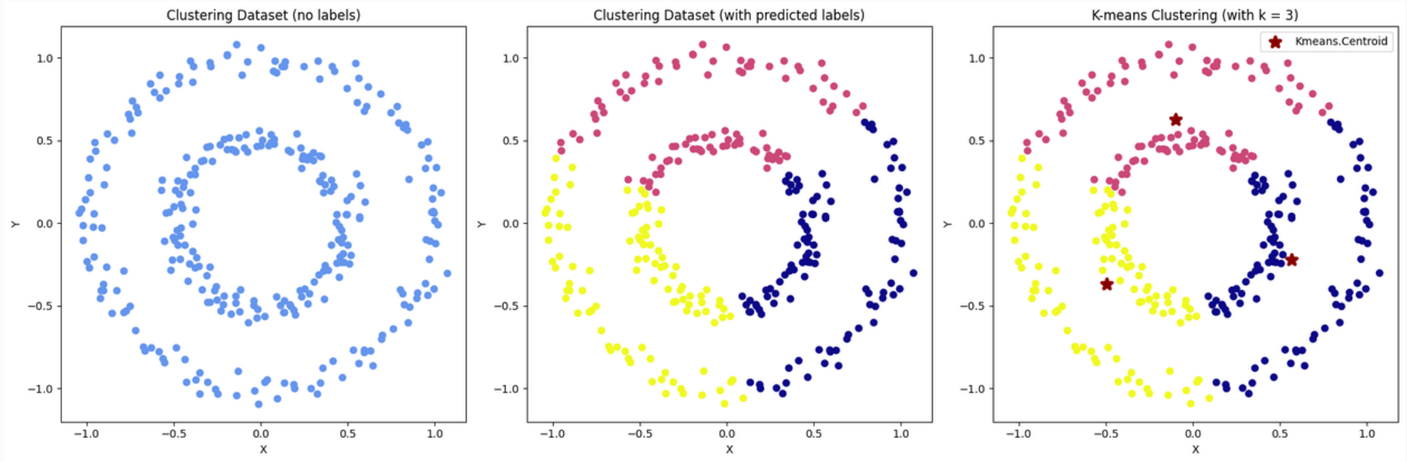
Evaluation metrics:

```
import pandas as pd
from sklearn import metrics

# Calculate evaluation metrics
f_measure_3k = metrics.f1_score(y3, y_pred3, average='weighted')
nmi_3k = metrics.normalized_mutual_info_score(y3, y_pred3)
rand_3k = metrics.rand_score(y3, y_pred3)
```

Evaluation Metrics DataSet 3	
Rand Index	0.6305
NMI	0.3214
F-measure	0.0411

4th Dataset: noisy circles dataset

```
X,y = datasets.make_circles(n_samples=n_samples, factor=.5, noise=.05, random_state=random_state)

Kmeans = KMeansClustering(X, num_clusters)
y_pred4, centroids = Kmeans.fit(X)

plot_clusters(X, y_pred4, centroids)
```



Evaluation metrics:

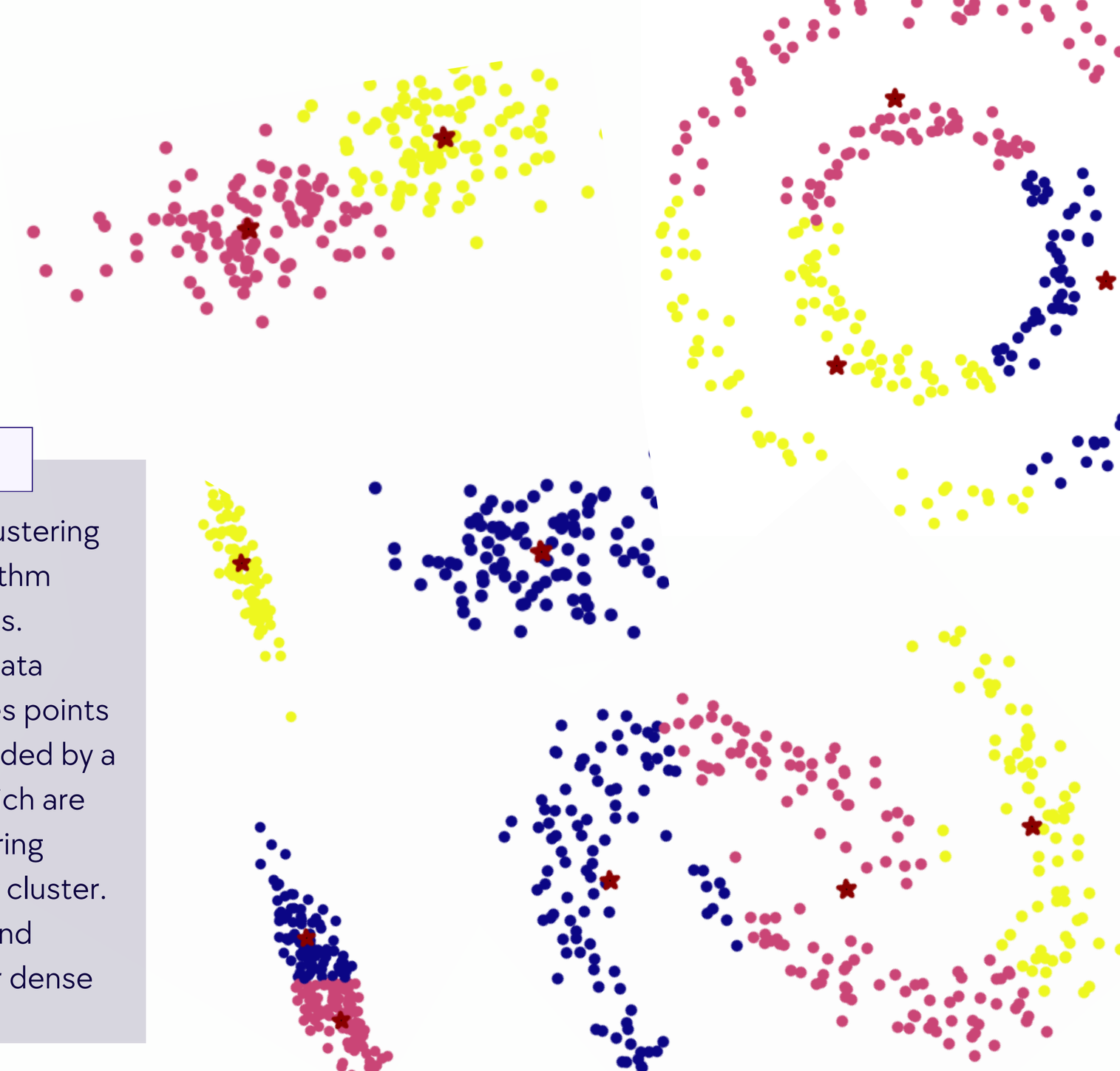
```
import pandas as pd
from sklearn import metrics

# Calculate evaluation metrics
f_measure_k4 = metrics.f1_score(y4, y_pred4, average='weighted')
nmi_k4 = metrics.normalized_mutual_info_score(y4, y_pred4)
rand_k4 = metrics.rand_score(y4, y_pred4)
```

Evaluation Metrics DataSet 4	
Rand Index	0.4986
F-measure	0.3909
NMI	0.0005

DBSCAN ALGORITHM

Create DBSCAN Algorithm from Scratch

DBSCAN, which stands for Density-Based Spatial Clustering of Applications with Noise, is a data clustering algorithm commonly used in machine learning and data analysis. The algorithm defines clusters as dense regions of data separated by areas of lower point density. It classifies points into three categories: core points, which are surrounded by a sufficient number of other points; border points, which are on the outskirts of a cluster and have fewer neighboring points; and noise points, which do not belong to any cluster. DBSCAN operates by iteratively exploring the data and forming clusters by connecting core points and their dense neighborhoods.



Datasets:

1st Dataset: Blobs dataset

2nd Dataset: Anisotropicly dataset

3rd Dataset: noisy moon dataset

4th Dataset: noisy circles dataset

Attributes:

n_samples = 300

random_state :

Student1 id: 2115590	Student2 id: 2111953	Student3 id: 2110006	Student4 id: 2110149	Student5 id: 2111489
2+1+1+5+5+9+0= 23	2+1+1+1+9+5+3= 22	2+1+1+6= 10	2+1+1+1+4+9 = 18	2+1+1+1+4+8+9= 26

23+22+10+18+26 = 99 random states

Finding best Epsilon & Minpoint

- **Generate The Datasets:**

```
n_samples = 300
random_state = 99

# Generate datasets

# Dataset1: Blobs dataset
X_blob, y_blob = datasets.make_blobs(n_samples=n_samples, random_state=random_state)

# Dataset2: Anisotropically distributed dataset
X_aniso, _ = datasets.make_blobs(n_samples=n_samples, random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X_aniso = np.dot(X_aniso, transformation)

# Dataset3: Noisy moons dataset
X_moons, y_moons = datasets.make_moons(n_samples=n_samples, noise=0.1, random_state=random_state)

# Dataset4: Noisy circles dataset
X_circles, y_circles = datasets.make_circles(n_samples=n_samples, factor=.5, noise=.05, random_state=random_state)
# Create a 2x2 grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
fig.suptitle('Elbow Method for Different Datasets')
```

This code generates four different datasets: blobs, anisotropically distributed points, noisy moons, and noisy circles.

Finding best Epsilon & Minpoint

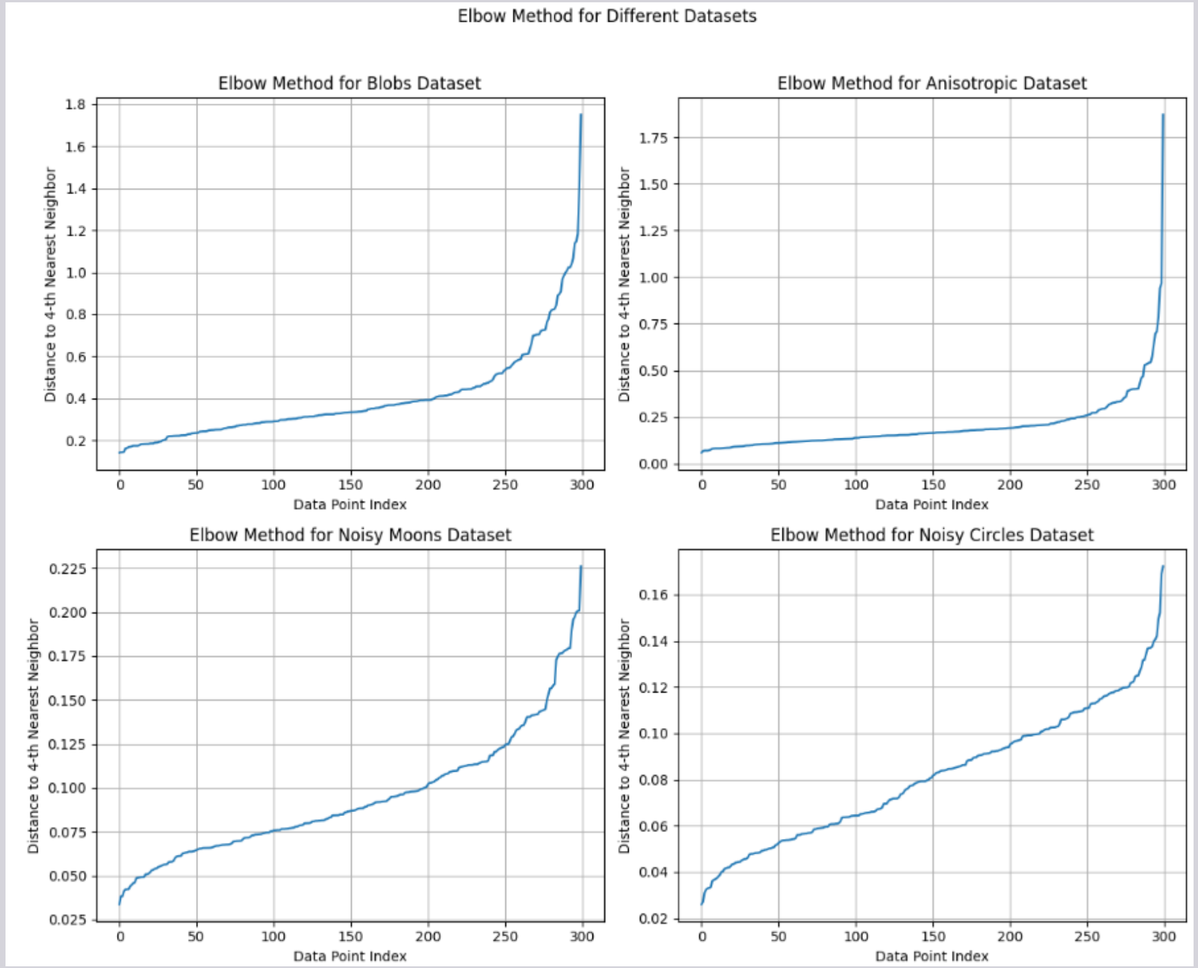
def plot_elbow(X, min_pts, dataset_name, ax):
This method generates an elbow plot to calculate the distance of the 4th neighbor.

```python
def plot_elbow(X, min_pts, dataset_name, ax):
    # Calculate the 4th nearest neighbor distances
    neigh = NearestNeighbors(n_neighbors=min_pts)
    neigh.fit(X)
    distances, _ = neigh.kneighbors(X)

    # Extract distances of the 4th neighbor
    fourth_neighbor_distances = distances[:, -1]

    # Sort the distances
    sorted_distances = np.sort(fourth_neighbor_distances)

    # Plot the sorted distances
    ax.plot(sorted_distances, label=dataset_name)
    ax.set_title(f'Elbow Method for {dataset_name} Dataset')
    ax.set_xlabel('Data Point Index')
    ax.set_ylabel(f'Distance to {min_pts}-th Nearest Neighbor')

    # Take the minimum value as the epsilon at the beginning of the increase
    epsilon_at_start_of_increase = np.min(sorted_distances)

    return epsilon_at_start_of_increase
```

This code applies the plot_elbow function to four datasets (blobs, anisotropic, noisy moons, and noisy circles) with a minimum points parameter of 4.

```python
# Apply the function for each dataset
epsilon_blob = plot_elbow(X_blob, min_pts=4, dataset_name='Blobs', ax=axes[0, 0])
epsilon_aniso = plot_elbow(X_aniso, min_pts=4, dataset_name='Anisotropic', ax=axes[0, 1])
epsilon_moons = plot_elbow(X_moons, min_pts=4, dataset_name='Noisy Moons', ax=axes[1, 0])
epsilon_circles = plot_elbow(X_circles, min_pts=4, dataset_name='Noisy Circles', ax=axes[1, 1])
```

Finding best Epsilon & Minpoint



The provided code generates four datasets and applies the elbow method for determining the optimal epsilon values in DBSCAN clustering.

Finding best Epsilon & Minpoint

MinPoint:

```
Blobs Dataset Shape: (300, 2)
Anisotropic Dataset Shape: (300, 2)
Noisy Moons Dataset Shape: (300, 2)
Noisy Circles Dataset Shape: (300, 2)
```

we will apply the formula (2* D) to find the MinPoint value.

The four datasets are 2D therefore,

2 * 2 = 4

the minPoint will be 4

class DBSCANClustering:

1. def _init_(self, eps, minpoint):
This method acts as the initializer for a class, likely designed to model a DBSCAN algorithm. It sets up the initial state of the class by assigning the values provided as arguments to the instance variables eps and minpoint.

```python
def __init__(self, eps, minpoint):
    # Constructor to initialize DBSCAN with epsilon (eps) and minimum points (minpoint)
    self.eps = eps
    self.minpoint = minpoint
```

2. def distance(point1, point2):
This method, named distance, calculates the Euclidean distance between two points.

```python
@staticmethod
def distance(point1, point2):
    # Static method to calculate the Euclidean distance between two points
    return np.linalg.norm(np.array(point1) - np.array(point2))
```

class DBSCANClustering:

3.def get_neighbors(self, point_index):
This method, get_neighbors, finds and returns the indices of data points that are neighbors to a specified point within a distance threshold (epsilon).

```python
def get_neighbors(self, point_index):
    # Method to get neighbors of a data point based on the distance threshold (epsilon)
    neighbors = []
    # Iterate over all data points to find neighbors within epsilon distance
    for i in range(len(self.data)):
        if self.distance(self.data[point_index], self.data[i]) <= self.eps:
            neighbors.append(i)
    return neighbors
```

4. def fit(self, data):
The fit method implements the DBSCAN clustering algorithm on a given dataset. It iterates through each data point, identifies neighbors, and forms clusters based on the specified conditions. The algorithm assigns cluster labels and marks points as noise if they don't meet the minimum point requirement.

```python
def fit(self, data):
    # Fit method to perform DBSCAN clustering on the given data
    self.data = data
    self.labels = [0] * len(data)  # Initialize cluster labels for each data point
    self.cluster_id = 0  # Initialize cluster ID

    # Iterate over each data point
    for i in range(len(data)):
        if self.labels[i] != 0:
            continue

        neighbors = self.get_neighbors(i)  # Get neighbors of the current data point

        # Check if the number of neighbors is greater than or equal to minpoint
        if len(neighbors) >= self.minpoint:
            self.cluster_id += 1  # Start a new cluster
            self.expand_cluster(i, neighbors)  # Expand the cluster with the current data point as the seed
        else:
            self.labels[i] = -1  # Mark the data point as noise (not part of any cluster)
```

class DBSCANClustering:

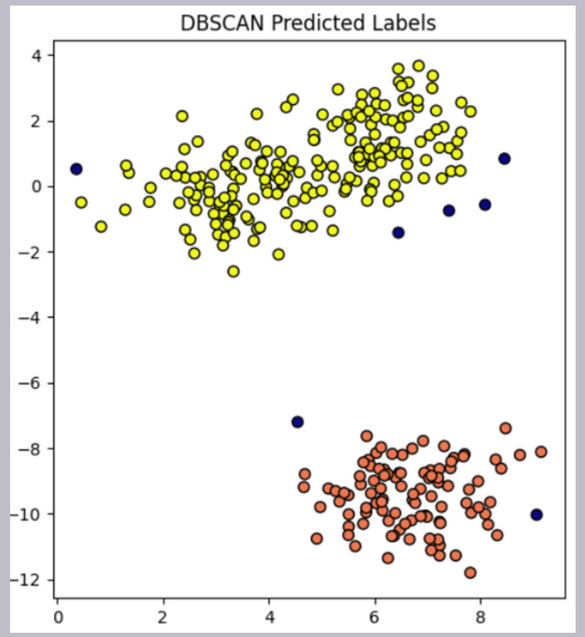
5. def expand_cluster(self, seed, neighbors):
The `expand_cluster` method expands a cluster from a seed data point, assigning the cluster ID to the seed point and iteratively incorporating neighbors. It handles noise points, assigns cluster IDs, and continues expanding the cluster based on specified conditions.

```python
def expand_cluster(self, seed, neighbors):
    # Method to expand a cluster starting from a seed data point
    self.labels[seed] = self.cluster_id  # Assign the cluster ID to the seed data point

    i = 0
    # Iterate over the neighbors of the seed data point
    while i < len(neighbors):
        current_point = neighbors[i]

        # Assign the cluster ID to noise points that are part of the current cluster
        if self.labels[current_point] == -1:
            self.labels[current_point] = self.cluster_id

        # Expand the cluster to the current neighbor if it's not assigned to any cluster
        elif self.labels[current_point] == 0:
            self.labels[current_point] = self.cluster_id
            current_point_neighbors = self.get_neighbors(current_point)

            # If the current neighbor has enough neighbors, add them to the list of neighbors
            if len(current_point_neighbors) >= self.minpoint:
                neighbors += current_point_neighbors

        i += 1
```

1st Dataset: Blobs dataset

```
n_samples = 300
random_state = 99
X, y = datasets.make_blobs(n_samples=n_samples, random_state=random_state)

# Initialize and fit the DBSCAN model
eps = 0.9
min_points = 4
dbscan_model = DBSCAN(eps=eps, minpoint=min_points)
dbscan_model.fit(X)

# Get the predicted labels from DBSCAN
predicted_labels = dbscan_model.labels

# Visualize the results
plt.figure(figsize=(12, 6))

# Plot DBSCAN predicted labels
plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=predicted_labels, cmap='plasma', edgecolors='k', s=40)
plt.title("DBSCAN Predicted Labels")

plt.show()
```

This code generates the Blobs dataset, applies DBSCAN clustering with specified parameters (epsilon = 0.9, min_points = 4), and visualizes the predicted labels in a scatter plot.



1st Dataset: Blobs dataset

Evaluation metrics:

```
# Calculate evaluation metrics
f1_DB1 = metrics.f1_score(y, predicted_labels, average='weighted')
nmi_DB1 = metrics.normalized_mutual_info_score(y, predicted_labels)
rand_index_DB1 = metrics.rand_score(y, predicted_labels)

# Create a styled table
results_styled = (
    pd.DataFrame({
        'Metric': ['F-measure', 'NMI', 'Rand Index'],
        'Score': [f1_DB1, nmi_DB1, rand_index_DB1]
    })
    .sort_values(by='Score', ascending=False)
    .set_index('Metric')
    .assign(Score=lambda x: x['Score'].apply(lambda y: f'{y:.4f}'))
    .style.set_caption('Evaluation Metrics DataSet 1')
    .set_table_styles([
        {'selector': 'table', 'props': [('margin-left', '20px'), ('width', '900px'),
                                        ('background-color', '#f2f2f2'), ('border', '1px solid #ddd')]},
        {'selector': 'thead', 'props': [('display', 'none')]},
        {'selector': 'th.col_heading', 'props': [('display', 'none')]},
        {'selector': 'td', 'props': [('font-size', '14px'), ('text-align', 'left'),
                                     ('font-family', 'Arial, sans-serif'), ('padding', '13px')]}
    ])
)

# Display the table
results_styled
```

This code calculates and displays F-measure, NMI, and Rand Index evaluation metrics for DBSCAN clustering on the first dataset.

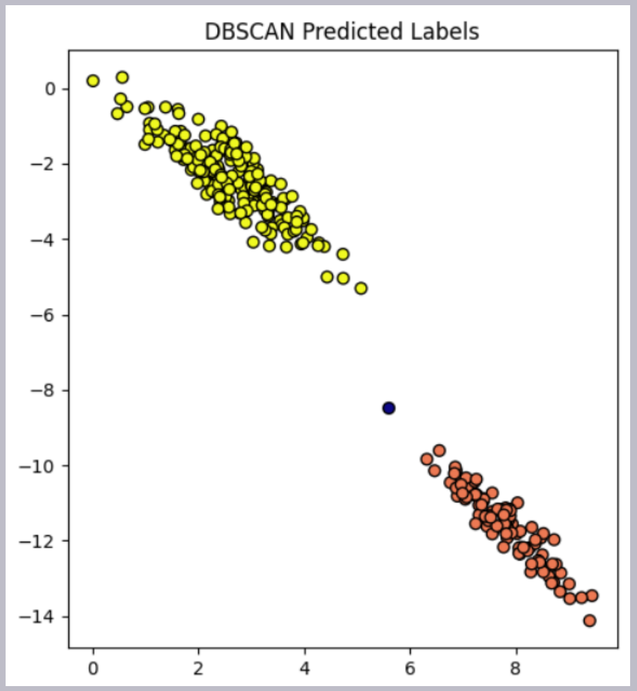
Evaluation Metrics DataSet 1

Rand Index	0.7726
NMI	0.6833
F-measure	0.5469

2nd Dataset: Anisotropicly dataset

```python
n_samples = 300
random_state = 99

X, y2 = datasets.make_blobs(n_samples=n_samples,random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X = np.dot(X, transformation)

# Initialize and fit the DBSCAN model
eps = 0.9
min_points = 4
dbscan_model = DBSCAN(eps=eps, minpoint=min_points)
dbscan_model.fit(X)

# Get the predicted labels from DBSCAN
predicted_labels2 = dbscan_model.labels

# Visualize the results
plt.figure(figsize=(12, 6))

# Plot DBSCAN predicted labels
plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=predicted_labels2, cmap='plasma', edgecolors='k', s=40)
plt.title("DBSCAN Predicted Labels")

plt.show()
```

This code generates the anisotropicly dataset, applies DBSCAN clustering with specified parameters (eps=0.9 and min_points=4), and visualizes the predicted labels in a scatter plot.



2nd Dataset: Anisotropicly dataset

Evaluation metrics:

```python
# Calculate evaluation metrics
f1_DB2 = metrics.f1_score(y2, predicted_labels2, average='weighted')
nmi_DB2 = metrics.normalized_mutual_info_score(y2, predicted_labels2)
rand_index_DB2 = metrics.rand_score(y2, predicted_labels2)

# Create a styled table
results_styled = (
    pd.DataFrame({
        'Metric': ['F-measure', 'NMI', 'Rand Index'],
        'Score': [f1_DB2, nmi_DB2, rand_index_DB2]
    })
    .sort_values(by='Score', ascending=False)
    .set_index('Metric')
    .assign(Score=lambda x: x['Score'].apply(lambda y: f'{y:.4f}'))
    .style.set_caption('Evaluation Metrics DataSet 2')
    .set_table_styles([
        {'selector': 'table', 'props': [('margin-left', '20px'), ('width', '900px'),
                                        ('background-color', '#f2f2f2'), ('border', '1px solid #ddd')]},
        {'selector': 'thead', 'props': [('display', 'none')]},
        {'selector': 'th.col_heading', 'props': [('display', 'none')]},
        {'selector': 'td', 'props': [('font-size', '14px'), ('text-align', 'left'),
                                     ('font-family', 'Arial, sans-serif'), ('padding', '13px')]}
    ])
)

# Display the table
results_styled
```

This code calculates and displays F-measure, NMI, and Rand Index evaluation metrics for DBSCAN clustering on the second dataset.

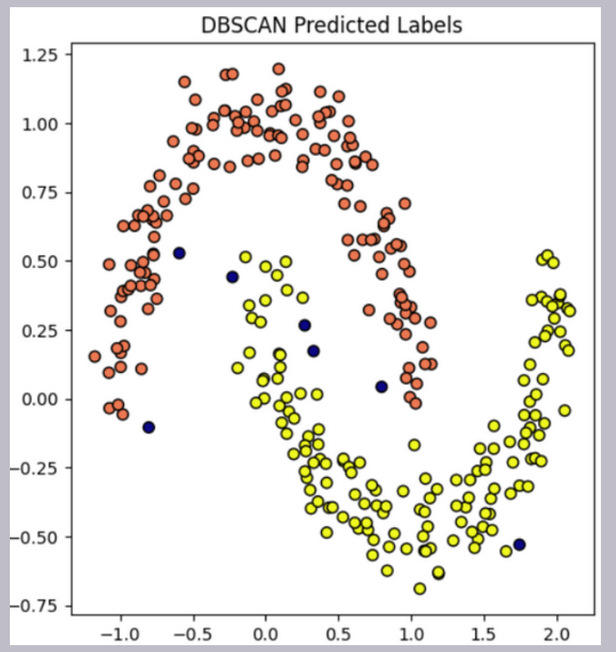
Evaluation Metrics DataSet 2

Rand Index	0.7748
NMI	0.7259
F-measure	0.5539

3rd Dataset: noisy moon dataset

```
n_samples = 300
random_state = 99

X, y3= datasets.make_moons(n_samples=n_samples, noise=0.1,random_state=random_state)

# Initialize and fit the DBSCAN model
eps = 0.149
min_points = 4
dbscan_model = DBSCAN(eps=eps, minpoint=min_points)
dbscan_model.fit(X)

# Get the predicted labels from DBSCAN
predicted_labels3 = dbscan_model.labels

# Visualize the results
plt.figure(figsize=(12, 6))

# Plot DBSCAN predicted labels
plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=predicted_labels3, cmap='plasma', edgecolors='k', s=40)
plt.title("DBSCAN Predicted Labels")

plt.show()
```

This code generates the noisy moons dataset, applies DBSCAN clustering with specified parameters (eps=0.149 and min_points=4), and visualizes the predicted labels in a scatter plot.



3rd Dataset: noisy moon dataset

Evaluation metrics:

```python
# Calculate evaluation metrics
f1_DB3 = metrics.f1_score(y3, predicted_labels3, average='weighted')
nmi_DB3 = metrics.normalized_mutual_info_score(y3, predicted_labels3)
rand_index_DB3 = metrics.rand_score(y3, predicted_labels3)

# Create a styled table
results_styled = (
    pd.DataFrame({
        'Metric': ['F-measure', 'NMI', 'Rand Index'],
        'Score': [f1_DB3, nmi_DB3, rand_index_DB3]
    })
    .sort_values(by='Score', ascending=False)
    .set_index('Metric')
    .assign(Score=lambda x: x['Score'].apply(lambda y: f'{y:.4f}'))
    .style.set_caption('Evaluation Metrics DataSet 3')
    .set_table_styles([
        {'selector': 'table', 'props': [('margin-left', '20px'), ('width', '900px'),
                                        ('background-color', '#f2f2f2'), ('border', '1px solid #ddd')]},
        {'selector': 'thead', 'props': [('display', 'none')]},
        {'selector': 'th.col_heading', 'props': [('display', 'none')]},
        {'selector': 'td', 'props': [('font-size', '14px'), ('text-align', 'left'),
                                     ('font-family', 'Arial, sans-serif'), ('padding', '13px')]}
    ])
)

# Display the table
results_styled
```

This code calculates and displays F-measure, NMI, and Rand Index evaluation metrics for DBSCAN clustering on the third dataset.

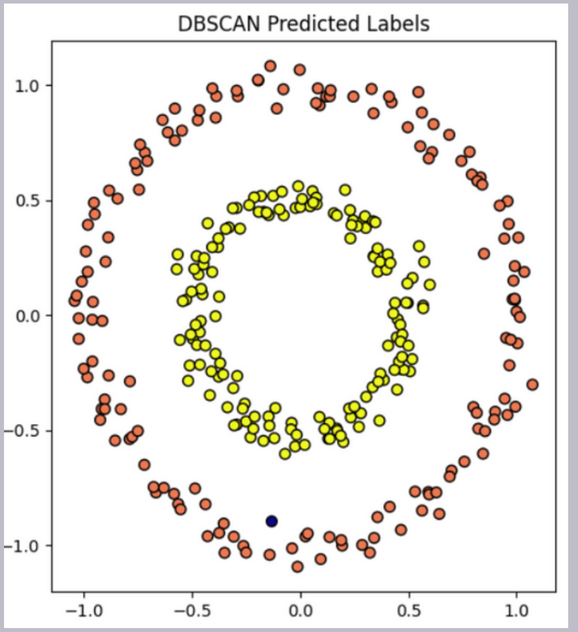
Evaluation Metrics DataSet 3

Rand Index	0.9704
NMI	0.8877
F-measure	0.0000

4th Dataset: noisy circles dataset

```
n_samples = 300
random_state = 99

# Dataset4: noisy circles dataset
X,y4 = datasets.make_circles(n_samples=n_samples, factor=.5, noise=.05, random_state=random_state)


# Initialize and fit the DBSCAN model
eps = 0.14
min_points = 4
dbscan_model = DBSCAN(eps=eps, minpoint=min_points)
dbscan_model.fit(X)

# Get the predicted labels from DBSCAN
predicted_labels4 = dbscan_model.labels

# Visualize the results
plt.figure(figsize=(12, 6))

# Plot DBSCAN predicted labels
plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=predicted_labels4, cmap='plasma', edgecolors='k', s=40)
plt.title("DBSCAN Predicted Labels")

plt.show()
```

This code generates the noisy circles dataset, applies DBSCAN clustering with specified parameters (eps=0.14 and min_points=4), and visualizes the predicted labels in a scatter plot.



4th Dataset: noisy circles dataset

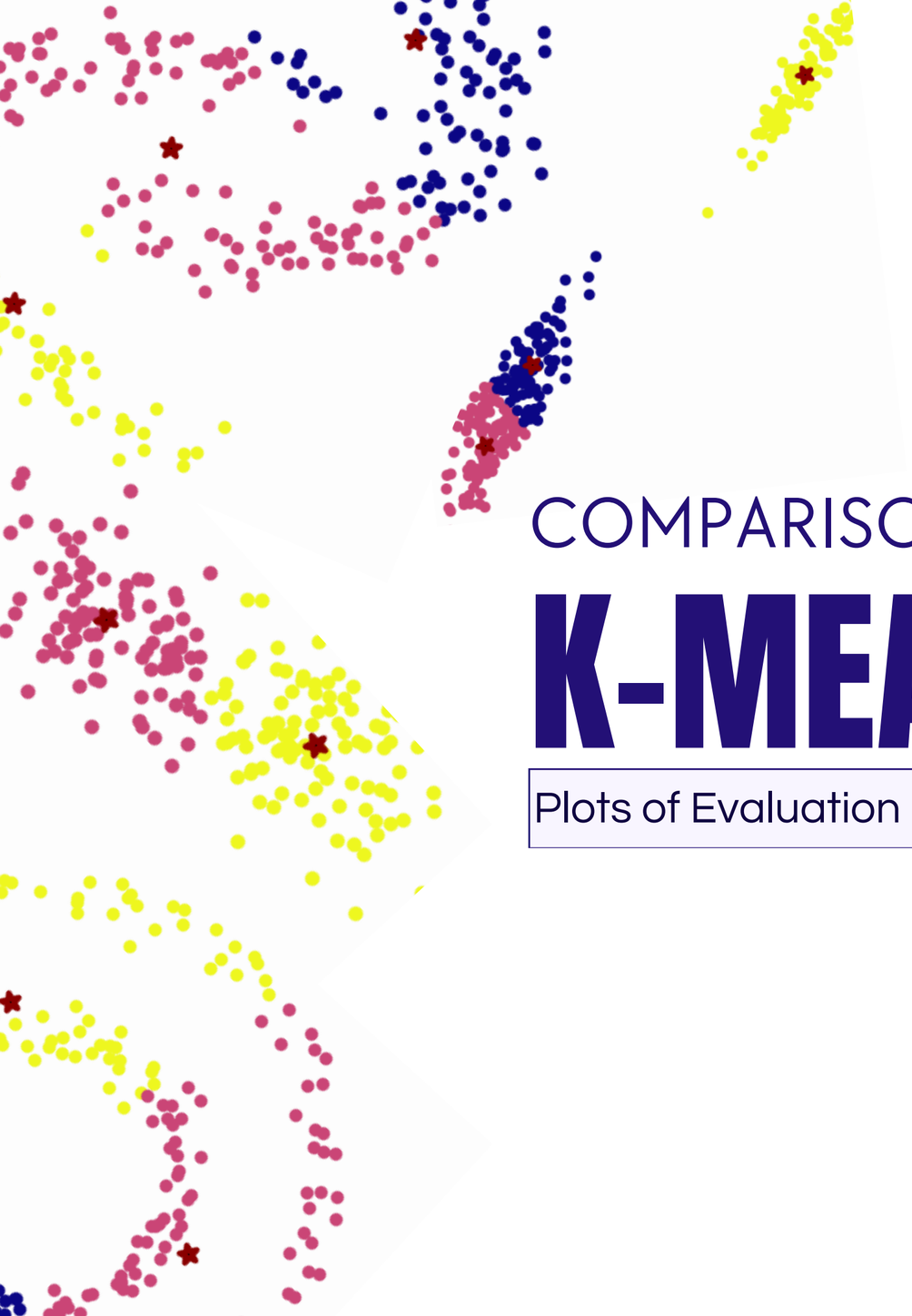
Evaluation metrics:

```
# Calculate evaluation metrics
f1_DB4 = metrics.f1_score(y4, predicted_labels4, average='weighted')
nmi_DB4 = metrics.normalized_mutual_info_score(y4, predicted_labels4)
rand_index_DB4 = metrics.rand_score(y4, predicted_labels4)

# Create a styled table
results_styled = (
    pd.DataFrame({
        'Metric': ['F-measure', 'NMI', 'Rand Index'],
        'Score': [f1_DB4, nmi_DB4, rand_index_DB4]
    })
    .sort_values(by='Score', ascending=False)
    .set_index('Metric')
    .assign(Score=lambda x: x['Score'].apply(lambda y: f'{y:.4f}'))
    .style.set_caption('Evaluation Metrics DataSet 4')
    .set_table_styles([
        {'selector': 'table', 'props': [('margin-left', '20px'), ('width', '900px'),
                                        ('background-color', '#f2f2f2'), ('border', '1px solid #ddd')]},
        {'selector': 'thead', 'props': [('display', 'none')]},
        {'selector': 'th.col_heading', 'props': [('display', 'none')]},
        {'selector': 'td', 'props': [('font-size', '14px'), ('text-align', 'left'),
                                     ('font-family', 'Arial, sans-serif'), ('padding', '13px')]}
    ])
)

# Display the table
results_styled
```

This code calculates and displays F-measure, NMI, and Rand Index evaluation metrics for DBSCAN clustering on the fourth dataset.

Evaluation Metrics DataSet 4

Rand Index	0.9967
NMI	0.9858
F-measure	0.0000



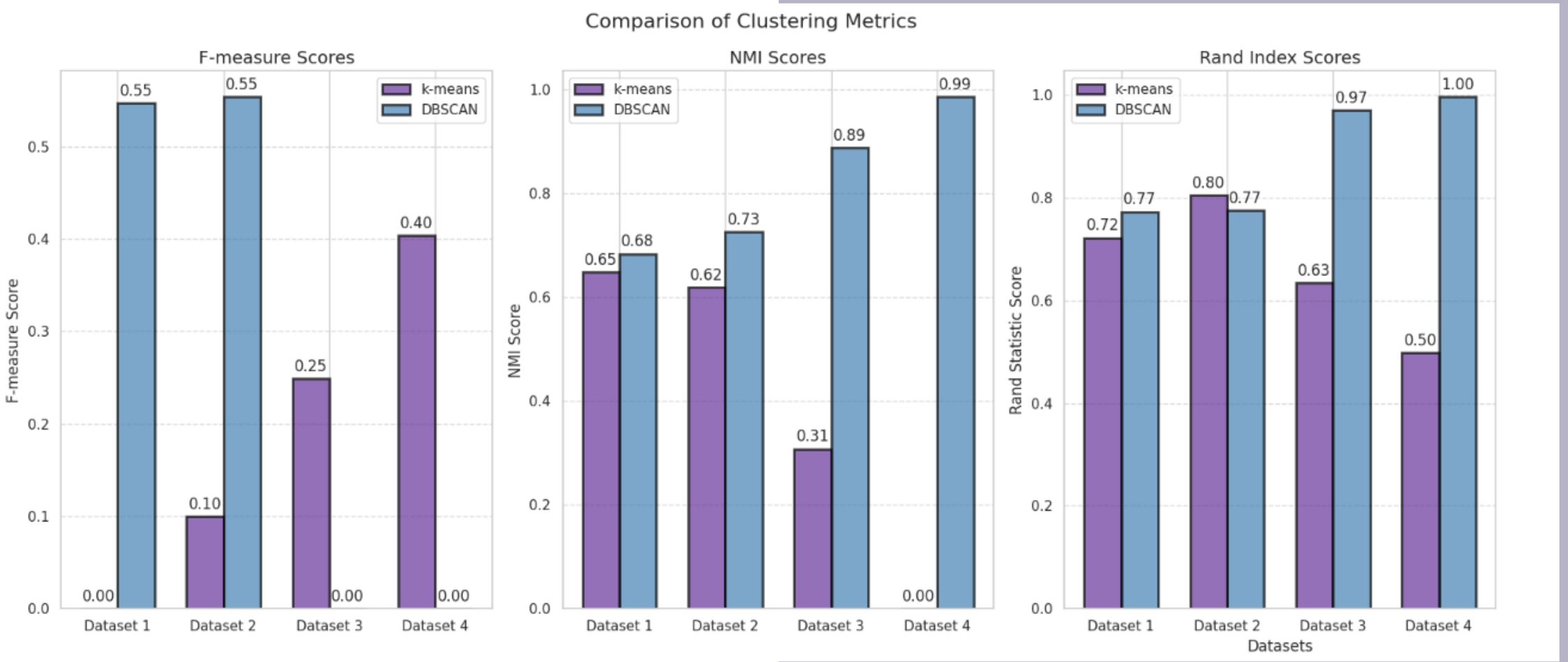
COMPARISON

K-MEAN VS DBSCAN

Plots of Evaluation Metrices

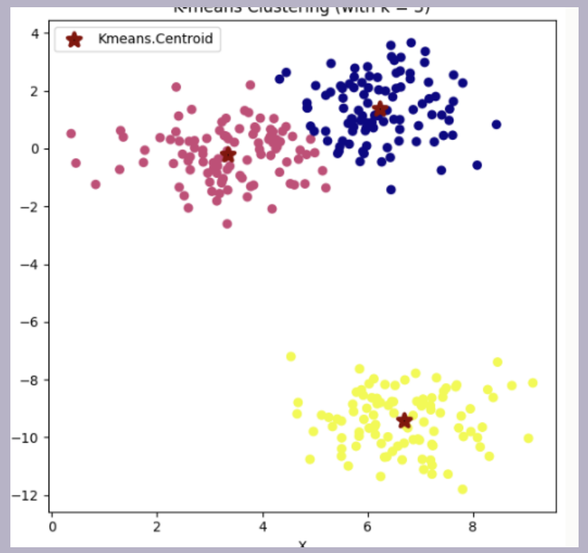
COMPARISON

K-MEAN VS DBSCAN



K-MEAN VS DBSCAN

Dataset 1

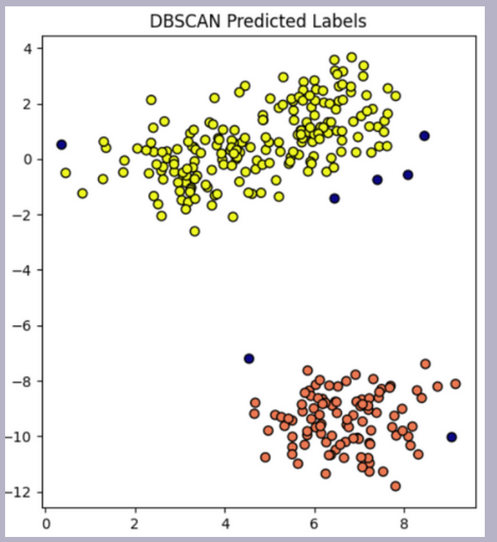


Evaluation Metrics DataSet 1

Rand Index	0.7221
NMI	0.6483
F-measure	0.0000

Evaluation Metrics DataSet 1

Rand Index	0.7726
NMI	0.6833
F-measure	0.5469



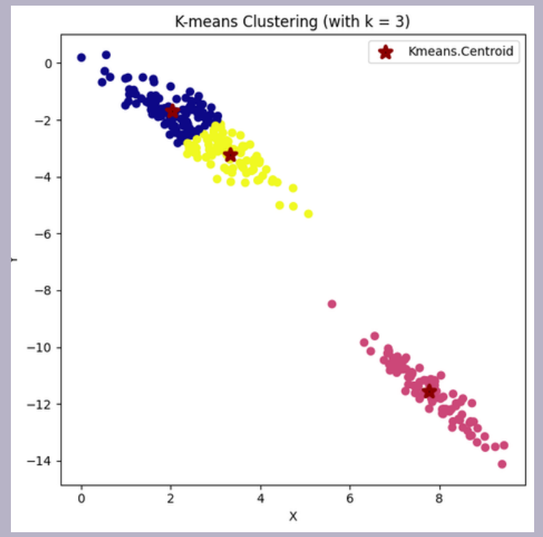
K-mean

DBSCAN

DBSCAN is more efficient for Dataset1

K-MEAN VS DBSCAN

Dataset 2



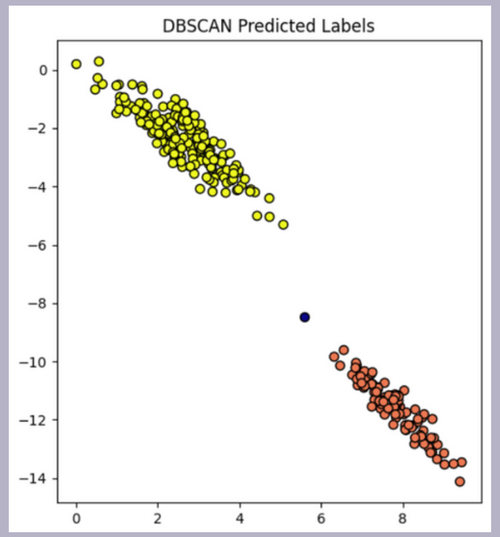
Evaluation Metrics DataSet 2

Rand Index	0.8043
NMI	0.6180
F-measure	0.1002

K-mean

Evaluation Metrics DataSet 2

Rand Index	0.7748
NMI	0.7259
F-measure	0.5539

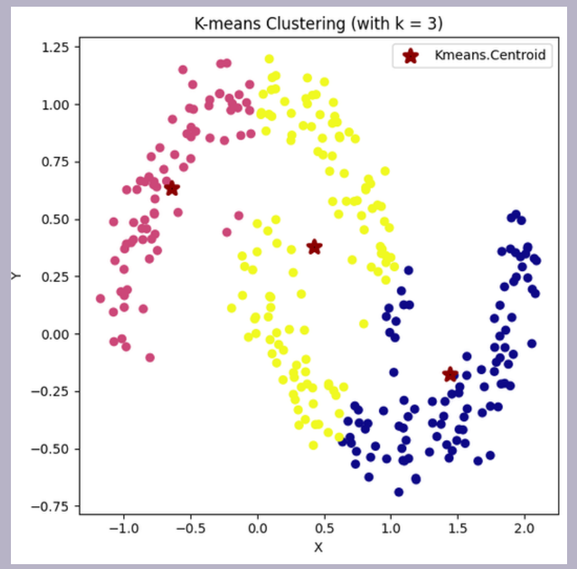


DBSCAN

DBSCAN is more efficient for Dataset2

K-MEAN VS DBSCAN

Dataset 3

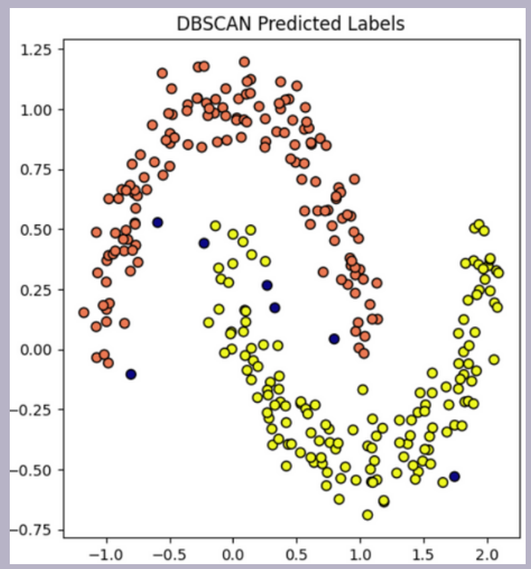


Evaluation Metrics DataSet 3

Rand Index	0.6340
NMI	0.3077
F-measure	0.2490

Evaluation Metrics DataSet 3

Rand Index	0.9704
NMI	0.8877
F-measure	0.0000

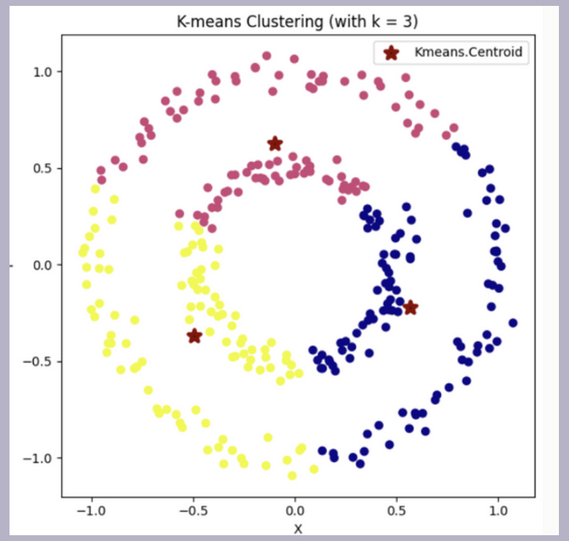


K-mean **DBSCAN**

DBSCAN is more efficient for Dataset3

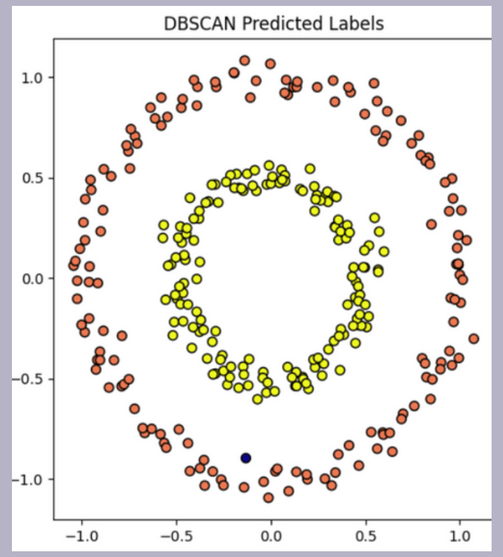
K-MEAN VS DBSCAN

Dataset 4



Evaluation Metrics DataSet 4	
Rand Index	0.4984
F-measure	0.4040
NMI	0.0000

Evaluation Metrics DataSet 4	
Rand Index	0.9967
NMI	0.9858
F-measure	0.0000



K-mean

DBSCAN

DBSCAN is more efficient for Dataset4

GROUP MEMBERS

- Joud Albaiti
- Fai Almeganni
- Danah Bawajeeh
- Raghad Alghamdi
- Nusaybah Altrabolsi