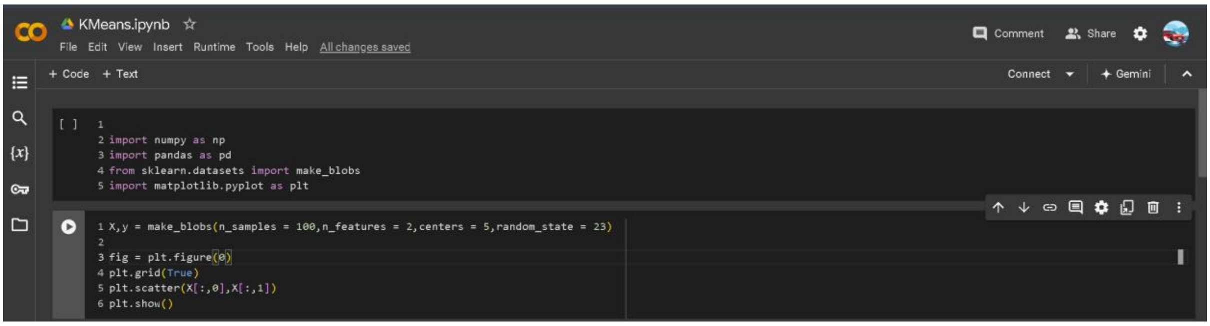
NAME: MARK LOPES

ROLL NO: 9913

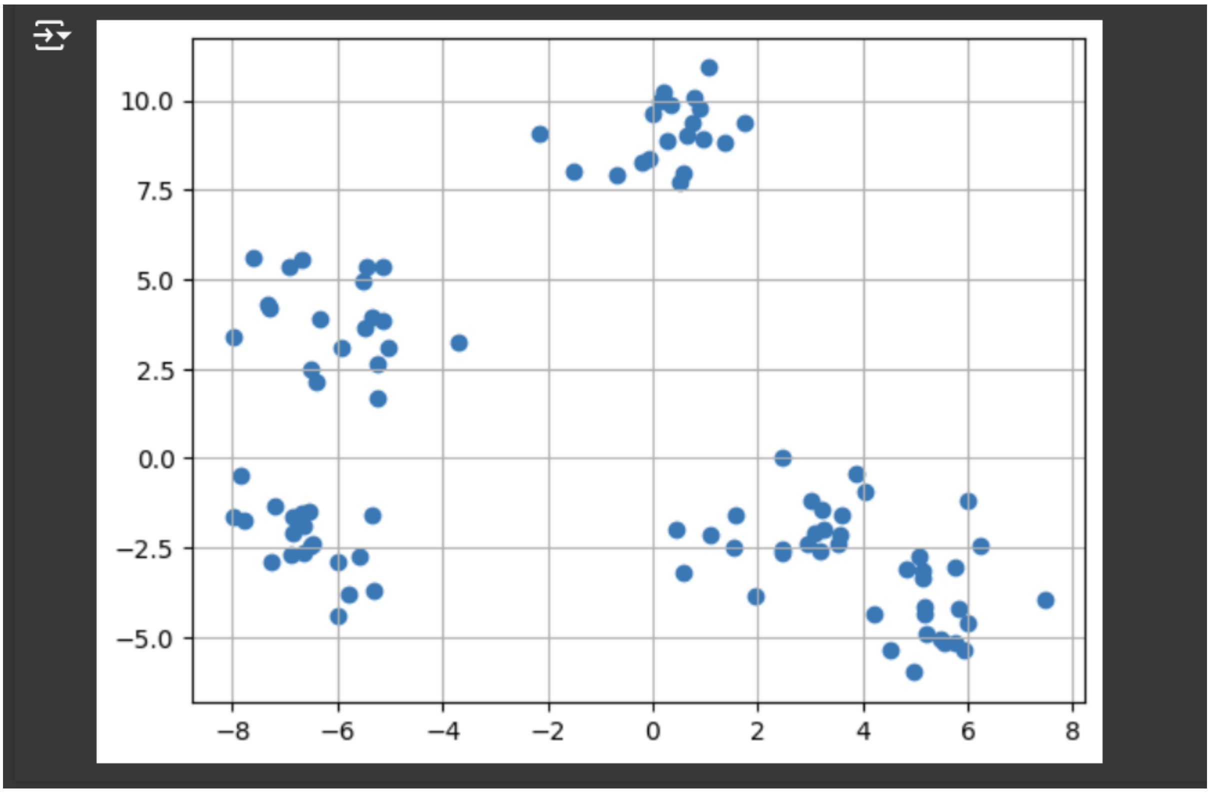
BRANCH: COMPUTER A

EXP 7: K MEANS CLUSTERING

K MEANS (WITHOUT USING INBUILT PYTHON FUNCTIONS)



Output:



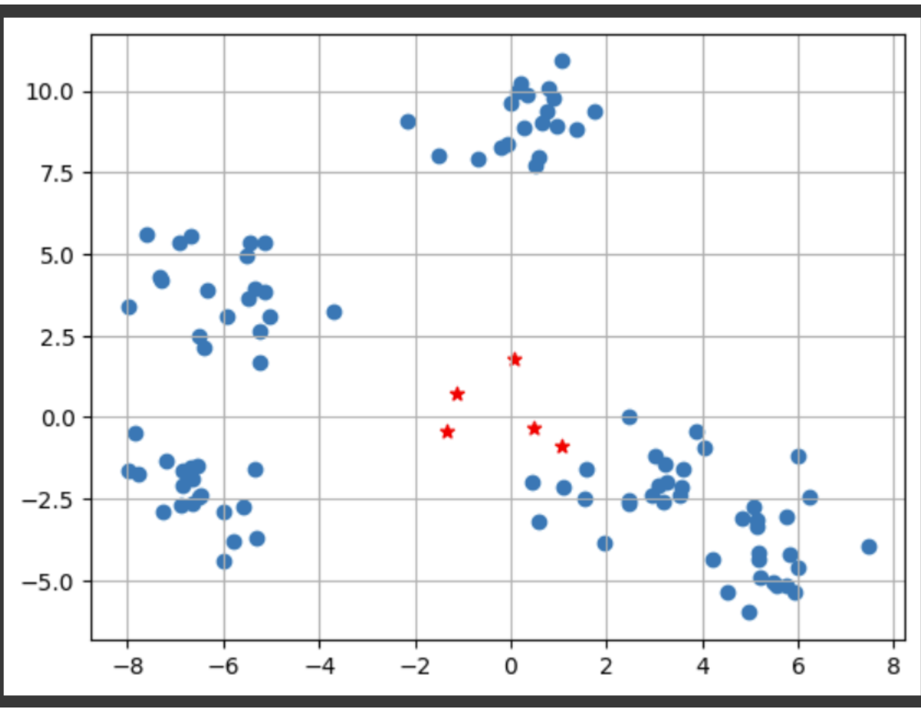
```
k = 5

clusters = {}
np.random.seed(23)

for idx in range(k):
    center = 2*(2*np.random.random((X.shape[1],))-1)
    points = []
    cluster = {
        'center' : center,
        'points' : []
    }

    clusters[idx] = Untitled1cluster

clusters
```

Output:

```
{0: {'center': array([0.06919154, 1.78785042]), 'points': []},
 1: {'center': array([ 1.06183904, -0.87041662]), 'points': []},
 2: {'center': array([-1.11581855,  0.74488834]), 'points': []},
 3: {'center': array([-1.33144319, -0.43023013]), 'points': []},
 4: {'center': array([ 0.47220939, -0.35227962]), 'points': []}}
```

```
plt.scatter(X[:,0],X[:,1])
plt.grid(True)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '*',c = 'red')
plt.show()
```

Output:



```python
def distance(p1,p2):
    return np.sqrt(np.sum((p1-p2)**2))
```

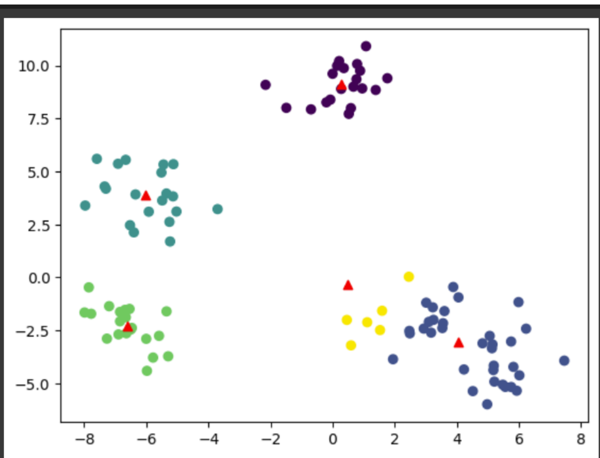
```python
#Implementing E step
def assign_clusters(X, clusters):
    for idx in range(X.shape[0]):
        dist = []

        curr_x = X[idx]

        for i in range(k):
            dis = distance(curr_x,clusters[i]['center'])
            dist.append(dis)
        curr_cluster = np.argmin(dist)
        clusters[curr_cluster]['points'].append(curr_x)
    return clusters

#Implementing the M-Step
def update_clusters(X, clusters):
    for i in range(k):
        points = np.array(clusters[i]['points'])
        if points.shape[0] > 0:
            new_center = points.mean(axis =0)
            clusters[i]['center'] = new_center

            clusters[i]['points'] = []
    return clusters

```

```python
def pred_cluster(X, clusters):
    pred = []
    for i in range(X.shape[0]):
        dist = []
        for j in range(k):
            dist.append(distance(X[i],clusters[j]['center']))
        pred.append(np.argmin(dist))
    return pred

```

```python
clusters = assign_clusters(X,clusters)
clusters = update_clusters(X,clusters)
pred = pred_cluster(X,clusters)


```

```python
plt.scatter(X[:,0],X[:,1],c = pred)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.show()
```

Output:



K- MEDIODS (WITHOUT USING INBUILT PYTHON FUNCTIONS)

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import pairwise_distances
```

```python
def compute_total_cost(X, medoid, cluster):
    total_cost = np.sum([np.linalg.norm(X[point] - X[medoid]) for point in cluster])
    return total_cost
```

```python
def assign_points_to_medoids(X, medoids):
    clusters = {}
    for idx, point in enumerate(X):
        distances = [np.linalg.norm(point - X[medoid]) for medoid in medoids]
        nearest_medoid = medoids[np.argmin(distances)]
        if nearest_medoid not in clusters:
            clusters[nearest_medoid] = []
        clusters[nearest_medoid].append(idx)
    return clusters

```

```python
def update_medoids(X, clusters):
    new_medoids = []
    for medoid, cluster in clusters.items():
        cluster_costs = [compute_total_cost(X, point, cluster) for point in cluster]
        new_medoid = cluster[np.argmin(cluster_costs)]
        new_medoids.append(new_medoid)
    return new_medoids
```

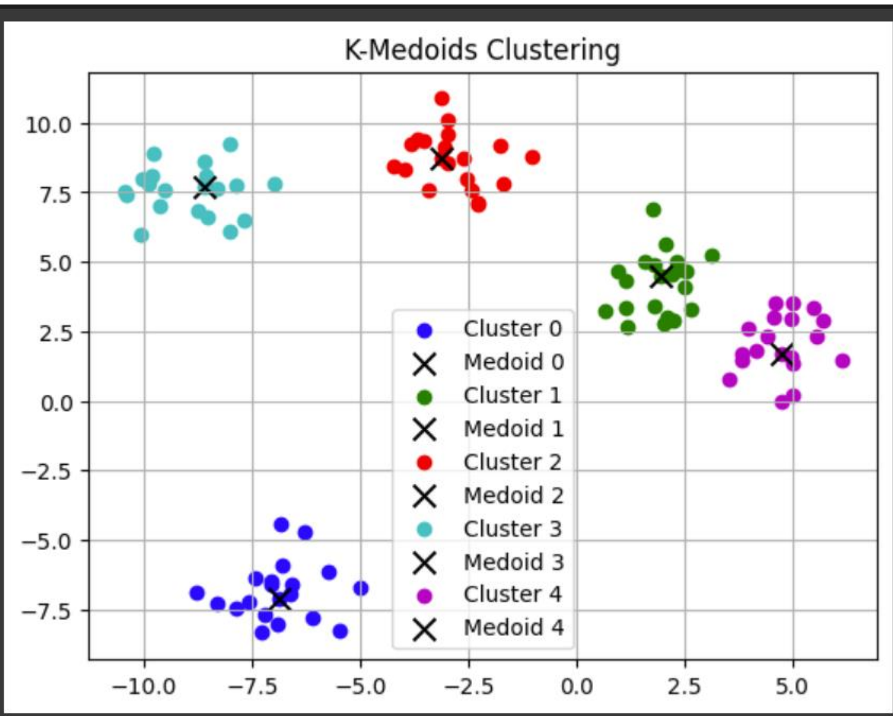
```python
def k_medoids(X, k, max_iter=100):
    # Step 1: Initialize k random medoids
    medoids = np.random.choice(range(len(X)), size=k, replace=False)

    for iteration in range(max_iter):
        # Step 2: Assign each point to the nearest medoid
        clusters = assign_points_to_medoids(X, medoids)

        # Step 3: Update the medoids
        new_medoids = update_medoids(X, clusters)

        # If the medoids don't change, break the loop
        if np.array_equal(medoids, new_medoids):
            break
        else:
            medoids = new_medoids

    return medoids, clusters
```

```python
from sklearn.datasets import make_blobs
X, _ = make_blobs(n_samples=100, n_features=2, centers=5, random_state=42)
```

```python
k = 5

# Run K-Medoids algorithm
medoids, clusters = k_medoids(X, k)
```

```python
colors = ['b', 'g', 'r', 'c', 'm']
for medoid, cluster in clusters.items():
    cluster_points = X[cluster]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], c=colors[medoids.index(medoid)], label=f'Cluster {medoids.index(medoid)}')
    plt.scatter(X[medoid][0], X[medoid][1], c='k', marker='x', s=100, label=f'Medoid {medoids.index(medoid)}')

plt.title('K-Medoids Clustering')
plt.grid(True)
plt.legend()
plt.show()
```

Output:



KMEANS (USING INBUILT FUNCTIONS)

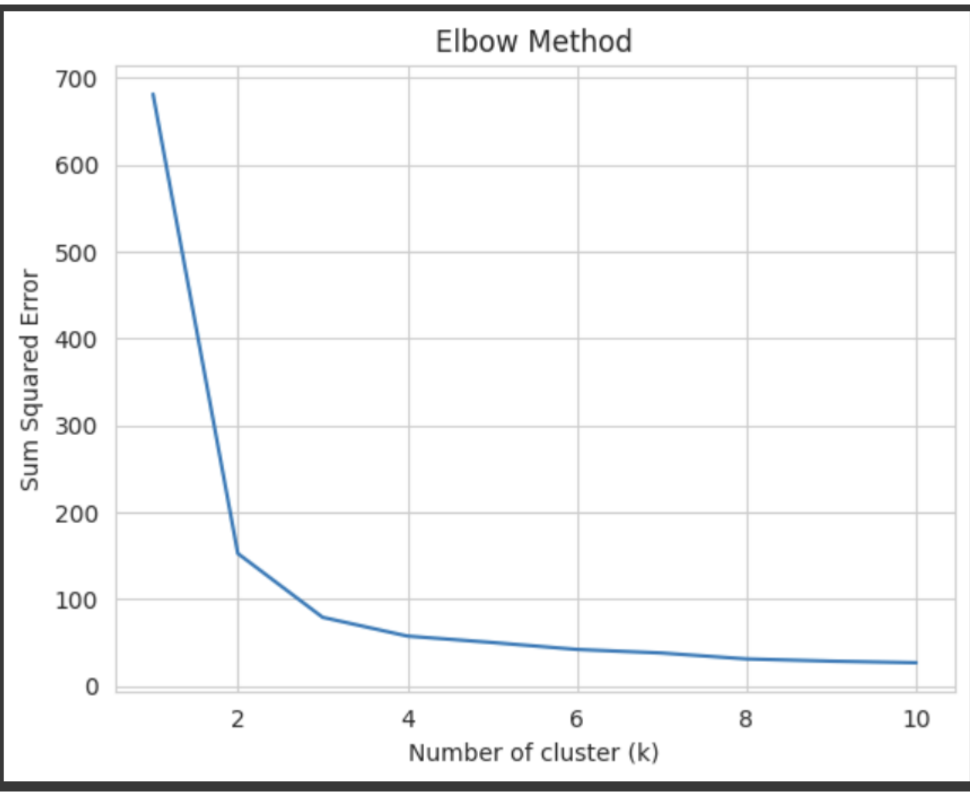
```
[1]
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
```

```
[2]
X, y = load_iris(return_X_y=True)
```

```
#Find optimum number of cluster
sse = [] #SUM OF SQUARED ERROR
for k in range(1,11):
    km = KMeans(n_clusters=k, random_state=2)
    km.fit(X)
    sse.append(km.inertia_)
```

```
[4]
sns.set_style("whitegrid")
g=sns.lineplot(x=range(1,11), y=sse)

g.set(xlabel ="Number of cluster (k)",
      ylabel = "Sum Squared Error",
      title ='Elbow Method')

plt.show()
```

Output:




```
[5]
kmeans = KMeans(n_clusters = 3, random_state = 2)
kmeans.fit(X)
```

```
KMeans
KMeans(n_clusters=3, random_state=2)
```

```
kmeans.cluster_centers_
```

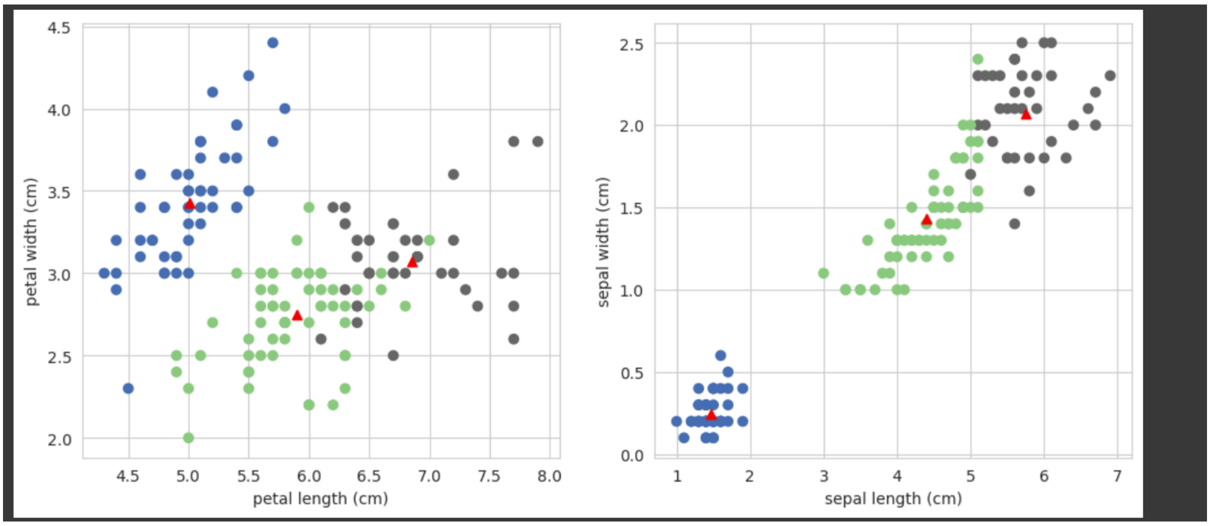
```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

```
[7]
pred = kmeans.fit_predict(X)
pred
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int32)
```

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.scatter(X[:,0],X[:,1],c = pred, cmap=cm.Accent)
plt.grid(True)
for center in kmeans.cluster_centers_:
    center = center[:2]
    plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.xlabel("petal length (cm)")
plt.ylabel("petal width (cm)")

plt.subplot(1,2,2)
plt.scatter(X[:,2],X[:,3],c = pred, cmap=cm.Accent)
plt.grid(True)
for center in kmeans.cluster_centers_:
    center = center[2:4]
    plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.xlabel("sepal length (cm)")
plt.ylabel("sepal width (cm)")
plt.show()
```

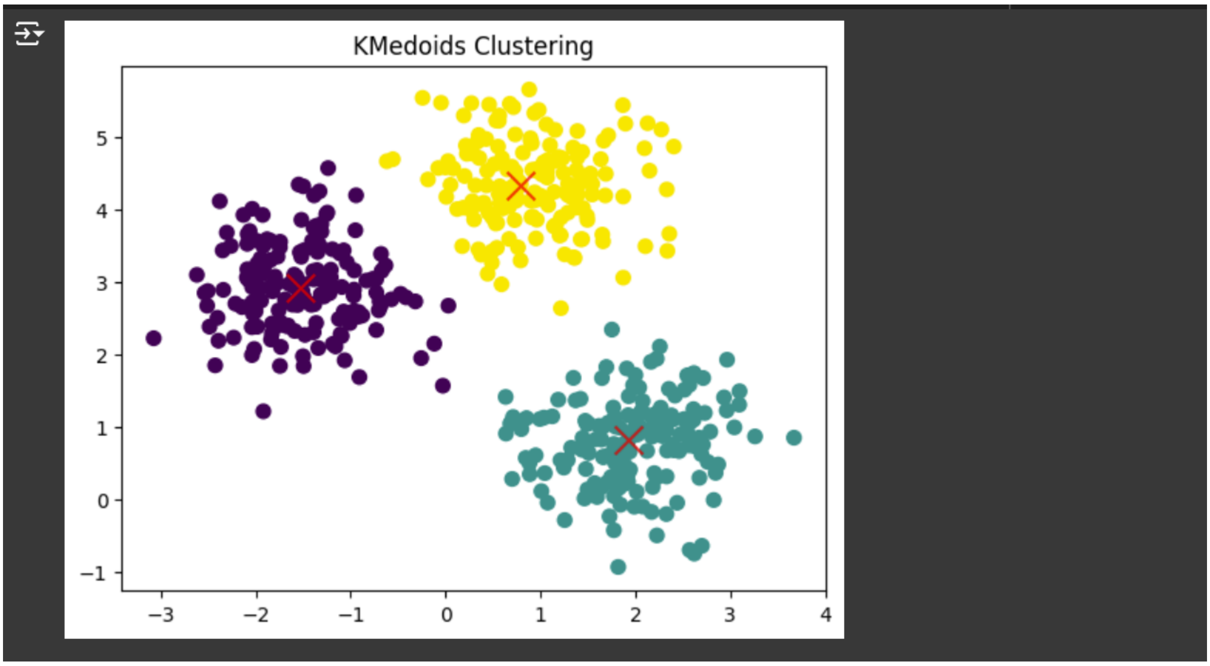
Output:



KMEDIOS (USING INBUILT FUNCTIONS)

```python
# Using lib
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn_extra.cluster import KMedoids

# Create sample data
n_samples = 500
n_clusters = 3
X, _ = make_blobs(n_samples=n_samples, centers=n_clusters, cluster_std=0.60, random_state=0)

# Fit KMedoids clustering model
kmedoids = KMedoids(n_clusters=n_clusters, random_state=0).fit(X)
labels = kmedoids.labels_

# Plot the data points with different colors for each cluster
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')

# Plot the medoid points (cluster centers)
medoids = kmedoids.cluster_centers_
plt.scatter(medoids[:, 0], medoids[:, 1], c='red', s=200, alpha=0.75, marker='x')

plt.title("KMedoids Clustering")
plt.show()

```

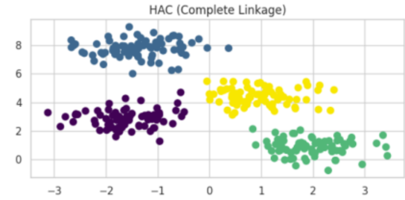
Output:

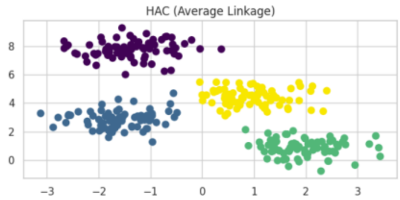


K MEANS (HIERARCHIAL -AGGLOMERATIVE SIMPLE ,COMPLETE ,AVERAGE LINKAGE)

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans, AgglomerativeClustering

# Set seaborn style for better aesthetics
sns.set(style="whitegrid")

# Generate synthetic data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Apply K-Means
k = 4  # Number of clusters
kmeans = KMeans(n_clusters=k, random_state=0)
kmeans_labels = kmeans.fit_predict(X)
kmeans_centers = kmeans.cluster_centers_

# Plot K-Means clusters
plt.figure(figsize=(18, 6))
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels, s=50, cmap='viridis')
plt.scatter(kmeans_centers[:, 0], kmeans_centers[:, 1],
            c='red', s=200, alpha=0.75, marker='X', label='Centroids')
plt.title("K-Means Clustering")
plt.legend()

# Hierarchical Agglomerative Clustering with different linkages
linkages = ['single', 'complete', 'average']
titles = ['Single Linkage', 'Complete Linkage', 'Average Linkage']

for i, linkage in enumerate(linkages, 2):
    hac = AgglomerativeClustering(n_clusters=k, linkage=linkage)
    hac_labels = hac.fit_predict(X)

    plt.subplot(1, 2, 2) if i == 2 else plt.subplot(2, 3, i)
    plt.scatter(X[:, 0], X[:, 1], c=hac_labels, s=50, cmap='viridis')
    plt.title(f"HAC ({titles[i-2]})")

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

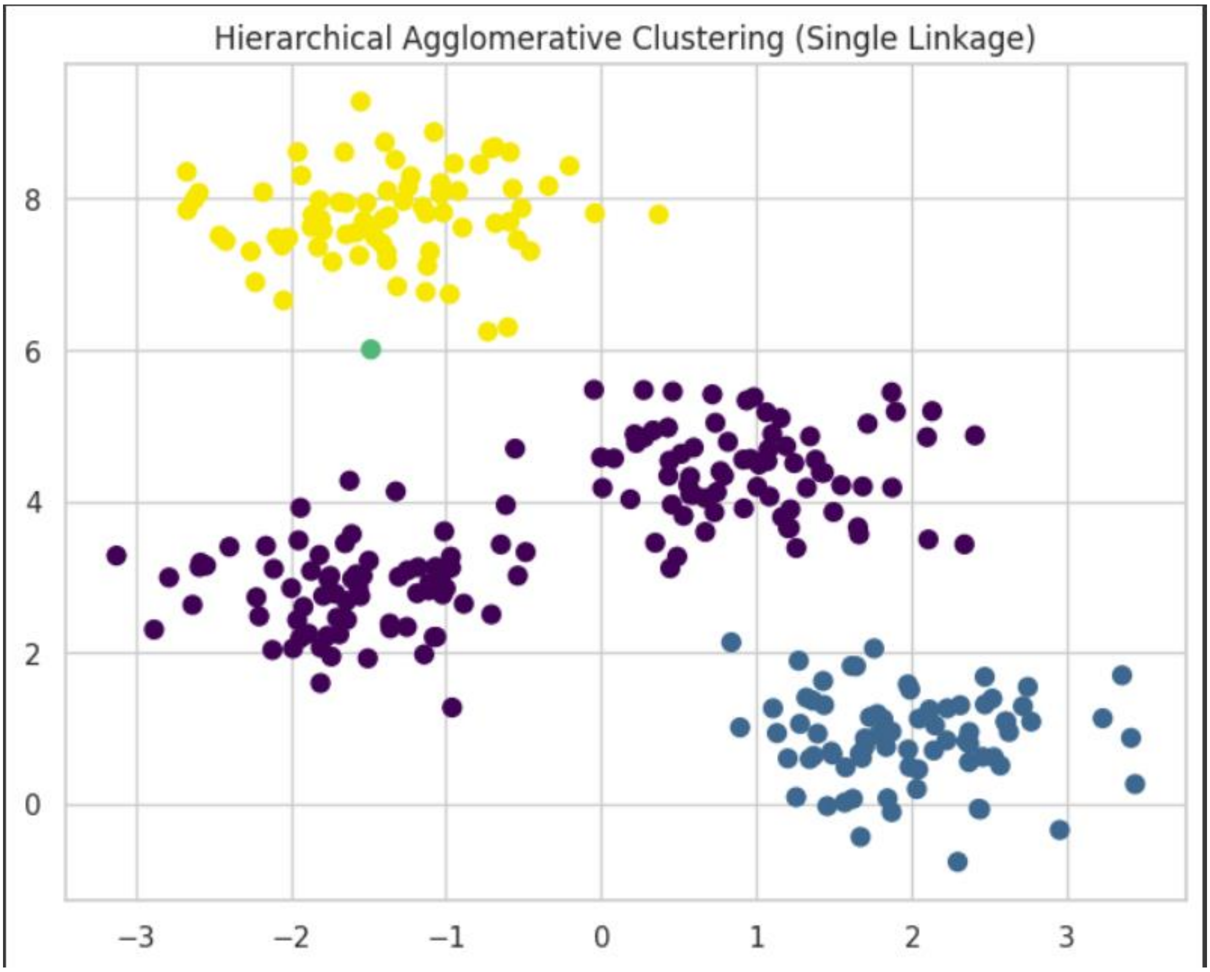
Output:






```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

# Set seaborn style for better aesthetics
sns.set(style="whitegrid")

# Generate synthetic data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Apply Hierarchical Agglomerative Clustering with Single Linkage
hac_single = AgglomerativeClustering(n_clusters=4, linkage='single')
hac_labels_single = hac_single.fit_predict(X)

# Plot the results
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=hac_labels_single, s=50, cmap='viridis')
plt.title("Hierarchical Agglomerative Clustering (Single Linkage)")
plt.show()

```

Output:


Hierarchical Agglomerative Clustering (Single Linkage)
8
6
4
2
0
−3
−2
−1
0
1
2
3