Advanced Machine Learning HW1

Keitaro Ogawa, ko2593

September 2025

1 EM algorithm for spherical Gaussians and Lloyd's algorithm

1.1 (a)

In Lloyd's algorithm, we use a hard cluster assignment instead of the soft responsibilities $\tilde{\pi}_{ik}^{(t)}$ from EM. The hard assignment indicator is given by

$$w_{ik}^{(t)} \;=\; \mathbf{1}\Big\{k \;=\; \arg\max_{1\le l\le K} \mathcal{N}\big(x_i;\, \mu_l^{(t)}, I_d\big)\Big\}.$$

Since for spherical Gaussians with covariance I_d we have

$$\mathcal{N}(x;\mu, I_d) \;\propto\; \exp\!\big(-\tfrac{1}{2}\|x-\mu\|^2\big)\,,$$

maximizing the Gaussian density is equivalent to minimizing the squared Euclidean distance. Thus, an equivalent form is

$$w_{ik}^{(t)} \;=\; \mathbf{1}\Big\{k \;=\; \arg\min_{1\le l\le K} \|x_i-\mu_l^{(t)}\|^2\Big\}.$$

1.2 (b)

The Lloyd's update for the cluster centers uses these hard assignments:

$$\mu_k^{(t+1)} \;=\; \frac{\sum_{i=1}^n w_{ik}^{(t)}\, x_i}{\sum_{i=1}^n w_{ik}^{(t)}}.$$

In comparison, the EM update for spherical Gaussian mixtures replaces the hard assignments $w_{ik}^{(t)}$ with the soft responsibilities $\tilde{\pi}_{ik}^{(t)}$:

$$\mu_k^{(t+1)}\Big|_{\mathrm{EM}} \;=\; \frac{\sum_{i=1}^n \tilde{\pi}_{ik}^{(t)}\, x_i}{\sum_{i=1}^n \tilde{\pi}_{ik}^{(t)}}.$$

Hence, Lloyd's update can be seen as the EM update under the approximation that each point is assigned fully to a single cluster, rather than distributed across clusters.

2 Optimizing over K in Lloyd's algorithm

Recall the K-means objective for a partition $\mathcal{C} = \{C_i\}_{i=1}^K$ with centers $\{\mu_i\}_{i=1}^K$:

$$\mathcal{L}(\mathcal{C}) \;=\; \sum_{i=1}^{K} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2.$$

If K is allowed to vary (with no penalty) then the minimum possible value of the objective is

$$\min_{K, \mathcal{C}, \{\mu_i\}} \mathcal{L}(\mathcal{C}) \;=\; 0,$$

because we can take $K = n$, choose each cluster to be a singleton $C_i = \{x_i\}$ and set $\mu_i = x_i$. Then every term $\|x_i - \mu_i\|^2$ equals zero.

Why this is wrong

- This solution is a trivial overfit: it perfectly fits the training points but has no generalization power.
- The objective alone encourages increasing model complexity (larger K) rather than balancing fit and complexity.
- Allowing arbitrary growth of K yields meaningless clusters and makes the problem ill-posed.

Common remedies

1. Constrain K (choose a maximum) or pick K by external criteria.
2. Use a penalized objective, e.g.

   $$\mathcal{L}_\lambda(\mathcal{C}) \;=\; \mathcal{L}(\mathcal{C}) + \lambda K,$$

   which trades off fit vs. complexity.
3. Apply model-selection methods (BIC/AIC/MDL), or validation measures such as silhouette score, gap statistic, or cross-validation to choose a meaningful K.

3 Using L1 norm in K-means type objective

Suppose we change the clustering objective to use the L_1 norm:

$$\mathcal{L}(\{C_i\}) \;=\; \sum_{i=1}^{K} \sum_{X_j \in C_i} \|X_j - \mu_i\|_1,$$

instead of the standard L_2 objective

$$\mathcal{L}(\{C_i\}) \;=\; \sum_{i=1}^{K} \sum_{X_j \in C_i} \|X_j - \mu_i\|_2^2.$$

In the L_2 case, the minimizer of $\sum_{X_j \in C_i} \|X_j - \mu_i\|_2^2$ is the *mean* of the points in C_i:

$$\mu_i \;=\; \frac{1}{|C_i|} \sum_{X_j \in C_i} X_j.$$

In contrast, for the L_1 case, the minimizer of $\sum_{X_j \in C_i} \|X_j - \mu_i\|_1$ is the *median* of the points (coordinate-wise in $\mathbb{R}^d$). That is,

$$\mu_i \;=\; \text{median}\{X_j : X_j \in C_i\},$$

where the median is taken independently in each coordinate.

Thus, when using the L_1 objective, it would no longer be correct to assign μ_i as the mean of the cluster points. Instead, μ_i should be set to the coordinate-wise median of the points in C_i.

4 Bad Local Minima

Lloyd's algorithm (K-means) minimizes a nonconvex objective, so it can get stuck in poor local minima that depend on the initialization. If centers u1 and u2 are not chosen near the true clusters, the assignment step may give both centers roughly half the points, even if the points clearly form two separate clusters. As a result, the centers can get stuck between the true clusters, converging to positions that do not represent the actual data structure.

What we might converge to. We typically converge to a local minimum in which:

- One centroid lies near the center of one true cloud, the other centroid splits the other cloud (or vice versa), producing a high within-cluster sum of squares compared to the global optimum.
- Or an *empty* cluster occurs (no points assigned to a centroid), which must be handled explicitly; otherwise the update step is undefined.

Intuition: Because K-means only performs local (greedy) updates—reassign points to nearest centroid and then move centroids to cluster means, it cannot escape an assignment configuration that is locally optimal but globally bad.

5 Projection based clustering

In projection-based clustering, PCA relies on the covariance matrix of the data. This already assumes that Euclidean distance is the "right" way of measuring spread, since variance and covariance are both defined with respect to squared Euclidean deviations from the mean. The statement that the top K eigenvectors capture the directions of cluster means is therefore implicitly tied to Euclidean geometry.

If it turned out that Euclidean distance was not the most appropriate notion of similarity for the data, I would first ask whether my chosen distance can be turned into an equivalent Euclidean form by a change of variables. For example, if the right metric is a Mahalanobis distance

$$d_M(x, y) = (x - y)^\top M(x - y),$$

with $M \succ 0$, then I could apply the linear transform $M^{1/2}$ to the data. In that transformed space, the Mahalanobis distance simply becomes Euclidean, and then PCA or K-means could still be used.

If the distance is more nonlinear (for instance, one that reflects manifold structure or some domain-specific similarity), then I would prefer to replace PCA altogether with an alternative method. Kernel PCA or classical multidimensional scaling (MDS) both work with pairwise similarities and are designed to preserve non-Euclidean structure. Spectral clustering is another natural choice, since it builds an embedding directly from a similarity graph defined by the chosen distance.

In short, PCA is appropriate only if Euclidean variance captures the signal I care about. If a different distance makes more sense, then either (a) transform the data so that the distance becomes Euclidean, or (b) use an embedding/clustering method such as kernel PCA, MDS, or spectral clustering that respects the distance directly.

6 Projection based clustering (cont)

Let $V = [v_1, \ldots, v_K] \in \mathbb{R}^{d \times K}$ be the matrix whose columns are the top K principal directions, with $\|v_i\| = 1$ and $V^\top V = I_K$. Consider the two transformations

$$\text{(i)} \quad \widetilde{X}_i \leftarrow V^\top X_i \in \mathbb{R}^K, \qquad \text{(ii)} \quad \widehat{X}_i \leftarrow VV^\top X_i \in \mathbb{R}^d.$$

Are the learned clusters different? No — running Lloyd's (K-means) on $\{\widetilde{X}_i\}$ produces the same cluster assignments as running Lloyd's on $\{\widehat{X}_i\}$. The reason is that Euclidean inter-point distances used by K-means are identical in the two representations. For any two data points $x, y \in \mathbb{R}^d$ set $u = x - y$. Using $V^\top V = I_K$ we have

$$\|VV^\top u\|_2^2 = u^\top VV^\top VV^\top u = u^\top VV^\top u = (V^\top u)^\top (V^\top u) = \|V^\top u\|_2^2.$$

Thus

$$\|\widehat{X}_i - \widehat{X}_j\|_2 = \|VV^\top(X_i - X_j)\|_2 = \|V^\top(X_i - X_j)\|_2 = \|\widetilde{X}_i - \widetilde{X}_j\|_2.$$

Because K-means uses only pairwise Euclidean distances (and centroid means computed by averaging), the entire K-means objective and the argmin partitions are equivalent under the two transforms. (Equivalently, centroids in K-space map to the same reconstructed centroids in d-space via multiplication by V.)

Is there a runtime difference? Yes, there is a practical difference:

- **Preprocessing cost:** computing $\widetilde{X}_i = V^\top X_i$ for all n points costs $O(ndK)$. Computing $\widehat{X}_i = VV^\top X_i$ can be implemented as $\widehat{X}_i = V(V^\top X_i)$, so it also costs $O(ndK)$ in total. Thus the one-time transform cost is the same (up to constants).
- **Lloyd's iterations (distance computations):** during K-means iterations, the dominant cost is computing distances from points to centroids. If you run K-means on $\widetilde{X}_i \in \mathbb{R}^K$ each distance is $O(K)$. If you run K-means naively on $\widehat{X}_i \in \mathbb{R}^d$ each distance is $O(d)$. Since typically $K \ll d$, K-means on the compressed $\widetilde{X}_i$ is much faster per iteration.
- **Memory:** storing $\widetilde{X}$ requires nK numbers, while $\widehat{X}$ requires nd. So $\widetilde{X}$ is much lighter when $K \ll d$.

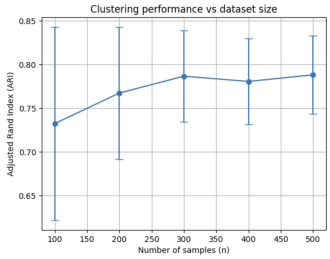
A practical note: if one implements K-means on $\widehat{X}$ but realizes $\widehat{X} = V(V^\top X)$, one can avoid the d-dimensional operations by storing the K-dimensional coordinates $V^\top X$ and performing all K-means computations in the K-space (then mapping centroids back to d-space only when needed). This shows that the theoretical equivalence is also exploitable for speed and memory: always perform clustering in the low-dimensional coordinate system whenever possible.

Exceptions / caveats

- If V columns are not orthonormal (or if the transform is approximate) the equal-norm identity above does not hold and clusterings may differ.
- If your clustering algorithm uses additional operations that depend on the ambient coordinates (e.g., explicit regularizers in $\mathbb{R}^d$ or constraints defined in the original coordinate system), then equivalence can break.

Short takeaway Projecting to K-coordinates, $\widetilde{X}_i = V^\top X_i$, and reconstructing to $\mathbb{R}^d$, $\widehat{X}_i = VV^\top X_i$, are equivalent for standard K-means in terms of cluster assignments (because distances are preserved). However, running Lloyd's on the K-dimensional coordinates is computationally and memory-wise preferable when $K \ll d$.

7 Comparing LLoyd's algorithm and Projection-based Clustering



(a) Clustering Performance vs. Data Size

(b) Clustering after PCA vs. Data Size

(c) Clustering Performance w/wo PCA

The results of the experiment show that the performance of Lloyd's algorithm is comparable whether it is applied directly to the original nine-dimensional data or to the data projected onto its top three principal components. Both approaches achieve a steadily increasing Adjusted Rand Index (ARI) as the number of samples grows, reflecting the fact that more data allows KMeans to find more accurate cluster centroids. By the time the sample size reaches around 400 to 500, the ARI values of the two methods are nearly indistinguishable, indicating that PCA does not result in any meaningful loss of clustering accuracy when sufficient data is available.

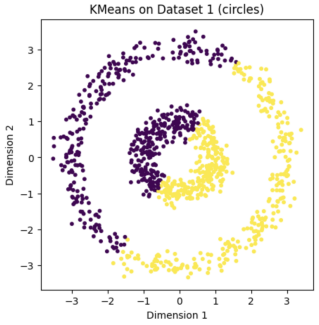
One notable difference, however, lies in the variability of performance. When clustering is performed directly on the original data, the standard deviations of the ARI scores are larger, particularly for smaller sample sizes. This suggests that the algorithm's performance in higher dimensions is more sensitive to randomness in sampling and initialization. In contrast, the PCA-based approach yields smaller error bars, showing more stability across repeated runs. This makes sense because PCA compresses the data into the directions of largest variance, effectively filtering out noise and redundant dimensions.

Overall, clustering in the projected three-dimensional space provides results that are just as accurate as clustering in the full nine-dimensional space, but

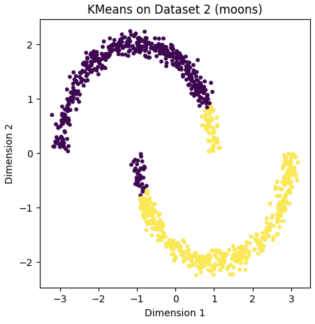
with the added benefit of greater consistency. PCA therefore serves as a useful preprocessing step in this setting, improving stability without sacrificing clustering performance.

8 Comparing K-means (Lloyd's), Meanshift and Spectral Clustering on synthetic data

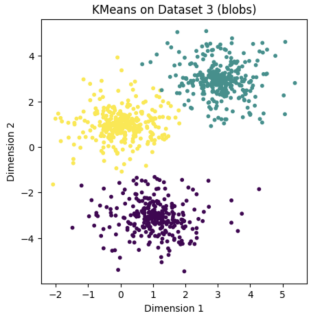
8.1 (a)



(a)

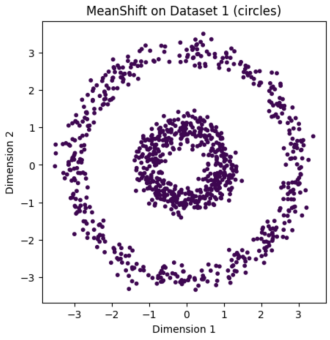


(b)

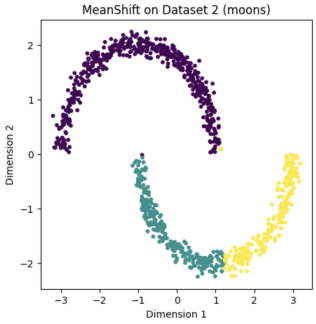


(c)

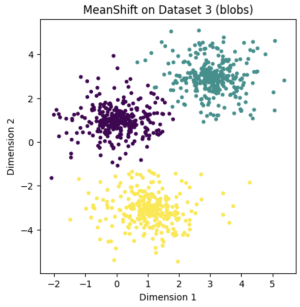
8.2 (b)


MeanShift on Dataset 1 (circles)
Dimension 1
Dimension 2

(a)

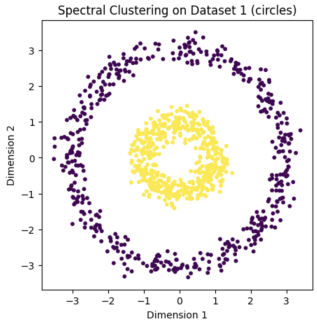

MeanShift on Dataset 2 (moons)
Dimension 1
Dimension 2

(b)

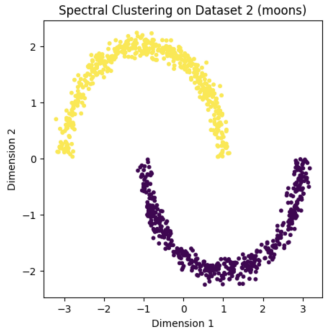

MeanShift on Dataset 3 (blobs)
Dimension 1
Dimension 2

(c)

8.3 (c)



(a)



(b)

(c)

I applied three clustering algorithms: KMeans, MeanShift, and Spectral Clustering to three synthetic datasets: concentric circles, two interleaving moons, and Gaussian blobs.

KMeans partitions the data into clusters by minimizing the variance within each cluster and assuming spherical decision boundaries. As shown in Figures for Q8 (a), KMeans performs poorly on the circles and moons datasets. This is expected because the algorithm relies on Euclidean distance and linear boundaries, which cannot capture the non-convex cluster shapes. In contrast, on the blobs dataset, KMeans performs well, as the clusters are spherical and well-separated, which matches its assumptions.

MeanShift is a density-based clustering algorithm that identifies clusters by locating regions of high point density. On the circles dataset, MeanShift fails to separate the two rings, since both rings have similar density distributions and the algorithm cannot distinguish between them. On the moons dataset, MeanShift is able to find more structure than KMeans but still struggles, leading to spurious cluster splits. On the blobs dataset, MeanShift performs effectively, correctly detecting the three Gaussian clusters without requiring the number of clusters to be specified in advance.

Spectral Clustering uses the eigenvalues of a similarity matrix to perform dimensionality reduction before applying clustering. This method excels at identifying clusters with complex, non-convex boundaries. Spectral Clustering successfully separates the concentric circles and the two moons, which are problematic for KMeans and MeanShift. On the blobs dataset, Spectral Clustering performs comparably to KMeans and MeanShift, correctly identifying the Gaussian clusters.

- **KMeans:** Works best for spherical, convex clusters (e.g., blobs) but fails for non-convex shapes (circles, moons).
- **MeanShift:** Adaptive to density but struggles when densities overlap (e.g., circles) and can over-segment in complex shapes.
- **Spectral Clustering:** Performs well on non-linear cluster structures (circles, moons) and is robust to irregular shapes.

9 Kernel Density Estimator

Let the kernel $K : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ satisfy $\int_{\mathbb{R}^d} K(u)\,du = 1$. Consider the kernel density estimator

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \qquad h > 0.$$

We show $\displaystyle\int_{\mathbb{R}^d} \hat{f}(x)\,dx = 1$.

Proof. Integrate $\hat{f}$ over $\mathbb{R}^d$ and use linearity of the integral:

$$\int_{\mathbb{R}^d} \hat{f}(x)\,dx = \frac{1}{nh^d} \sum_{i=1}^{n} \int_{\mathbb{R}^d} K\left(\frac{x - x_i}{h}\right) dx.$$

For each fixed i perform the change of variables

$$u = \frac{x - x_i}{h} \quad \Longrightarrow \quad x = x_i + hu,$$

so $dx = h^d\,du$ (Jacobian $= h^d$ for this linear map). Thus

$$\int_{\mathbb{R}^d} K\left(\frac{x - x_i}{h}\right) dx = \int_{\mathbb{R}^d} K(u)\,h^d\,du = h^d \int_{\mathbb{R}^d} K(u)\,du = h^d \cdot 1 = h^d.$$

Substituting back gives

$$\int_{\mathbb{R}^d} \hat{f}(x)\,dx = \frac{1}{nh^d} \sum_{i=1}^{n} h^d = \frac{1}{n} \sum_{i=1}^{n} 1 = 1.$$

Finally, since $K(u) \geq 0$ for all u, each summand of $\hat{f}(x)$ is nonnegative and hence $\hat{f}(x) \geq 0$ for all x. Combining nonnegativity and unit integral shows $\hat{f}$ is a valid probability density. □

Remark (intuition): scaling by $1/h^d$ exactly cancels the volume change of the linear map $u \mapsto x_i + hu$, which is why the kernel's integral property $\int K = 1$ suffices to ensure $\hat{f}$ integrates to one.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
from sklearn.decomposition import PCA
```

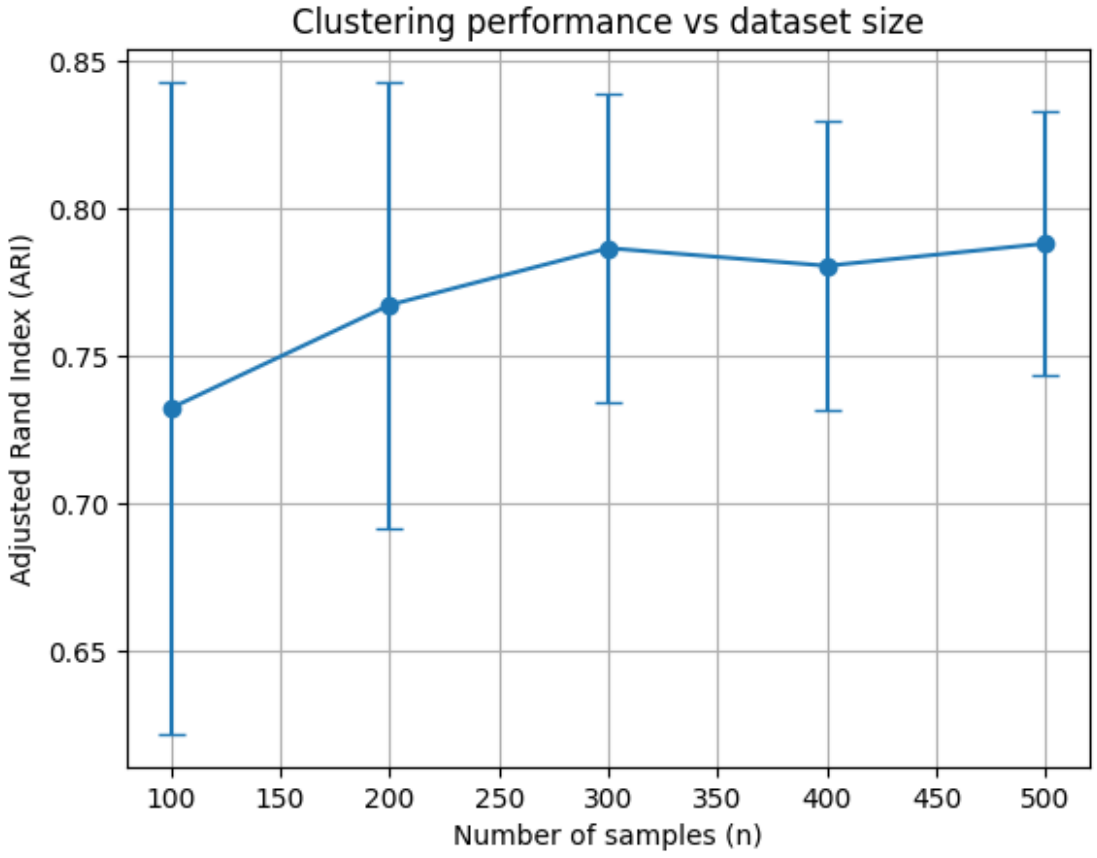
7.

```
# DATA GENERATOR
dim = 9
def data_generator (seed,n ,dim ) :
    mu1 = [1 , 1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 ]
    mu2 = [0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 1 ]
    mu3 = [0 , 0 , 0 , 1 , 1 , 1 , 0 , 0 , 0 ]

    sigma1 = np.diag ( [1 , 1 , 1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 ,
0.1 ] )
    sigma2 = np.diag ( [0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 0.1 , 1 , 1 ,
1 ] )
    sigma3 = np.diag ( [ 0.1 , 0.1 , 0.1 , 1 , 1 , 1 , 0.1 , 0.1 , 0.1
] )
    np.random.seed( seed )
    rand_int = np.random.choice ( [1 , 2 , 3 ] , size = n )
    unique_values , counts = np.unique ( rand_int , return_counts =
True )
    datapoints = np.zeros (( 0 , dim ) )
    labels = np.array ( [ ] )
    for i , ( uv , mu , sigma ) in enumerate ( zip ( unique_values , [
mu1 , mu2 , mu3 ] , [sigma1 , sigma2 , sigma3 ] ) ) :
        datapoints = np.vstack (( datapoints ,
np.random.multivariate_normal ( mu ,sigma , size = counts [ i ] ) ) )
        labels = np.hstack (( labels , uv * np.ones ( counts [ i ] ) )
)
    shuff = np.random.permutation ( len ( labels ) )
    return datapoints [ shuff ] , labels [ shuff ]
```

a)

```
n_values = [100, 200, 300, 400, 500]
n_runs = 100

mean_ari = []
std_ari = []

for n in n_values:
    ari_scores = []
```

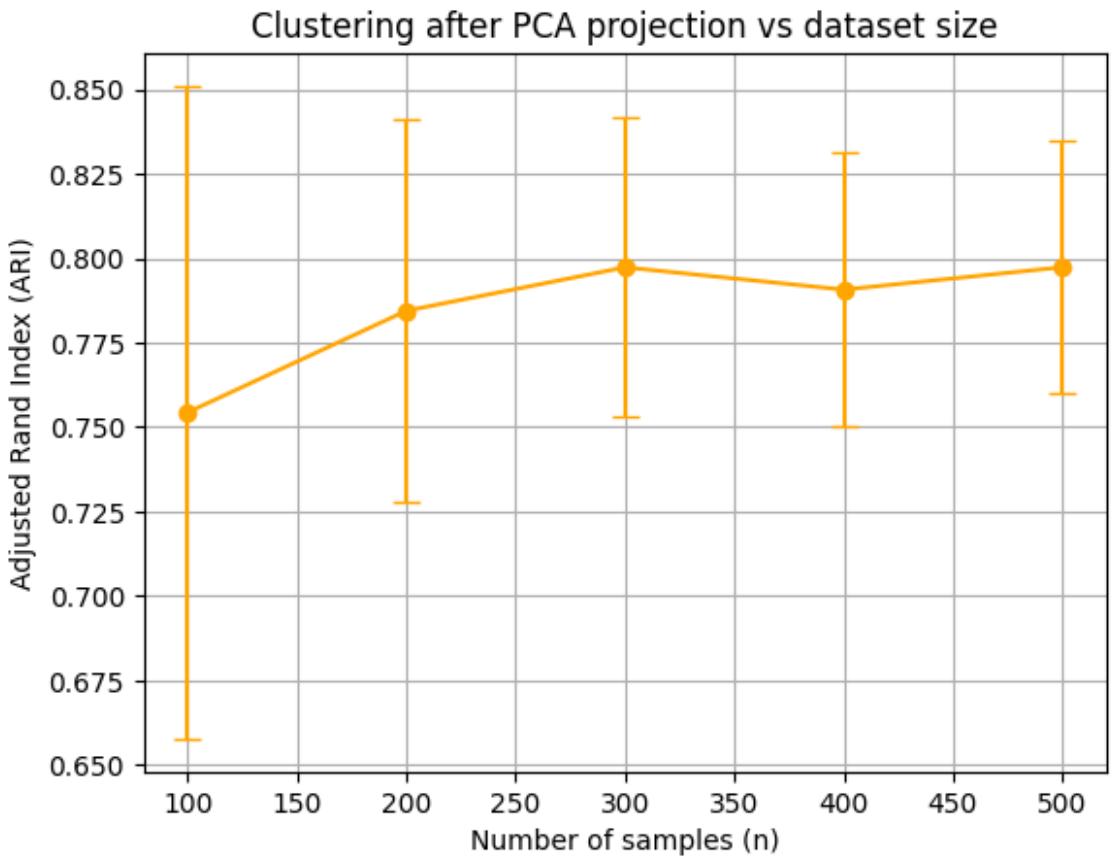
```
    for seed in range(1, n_runs + 1):
        X, y_true = data_generator(seed, n, dim=9)

        kmeans = KMeans(n_clusters=3, n_init=10, random_state=seed)
        y_pred = kmeans.fit_predict(X)

        ari = adjusted_rand_score(y_true, y_pred)
        ari_scores.append(ari)

    mean_ari.append(np.mean(ari_scores))
    std_ari.append(np.std(ari_scores))

plt.errorbar(n_values, mean_ari, yerr=std_ari, fmt='-o', capsize=5)
plt.xlabel("Number of samples (n)")
plt.ylabel("Adjusted Rand Index (ARI)")
plt.title("Clustering performance vs dataset size")
plt.grid(True)
plt.show()
```



b)

```
mean_ari_pca = []
std_ari_pca = []
```

```
for n in n_values:
    ari_scores = []
    for seed in range(1, n_runs + 1):
        X, y_true = data_generator(seed, n, dim=9)

        # --- Project to 3 principal components ---
        X_proj = PCA(n_components=3,
random_state=seed).fit_transform(X)

        # --- Lloyd's algorithm on projected data ---
        kmeans = KMeans(n_clusters=3, n_init=10, random_state=seed)
        y_pred = kmeans.fit_predict(X_proj)

        ari = adjusted_rand_score(y_true, y_pred)
        ari_scores.append(ari)

    mean_ari_pca.append(np.mean(ari_scores))
    std_ari_pca.append(np.std(ari_scores))

plt.errorbar(n_values, mean_ari_pca, yerr=std_ari_pca, fmt='-o',
capsize=5, color="orange")
plt.xlabel("Number of samples (n)")
plt.ylabel("Adjusted Rand Index (ARI)")
plt.title("Clustering after PCA projection vs dataset size")
plt.grid(True)
plt.show()
```



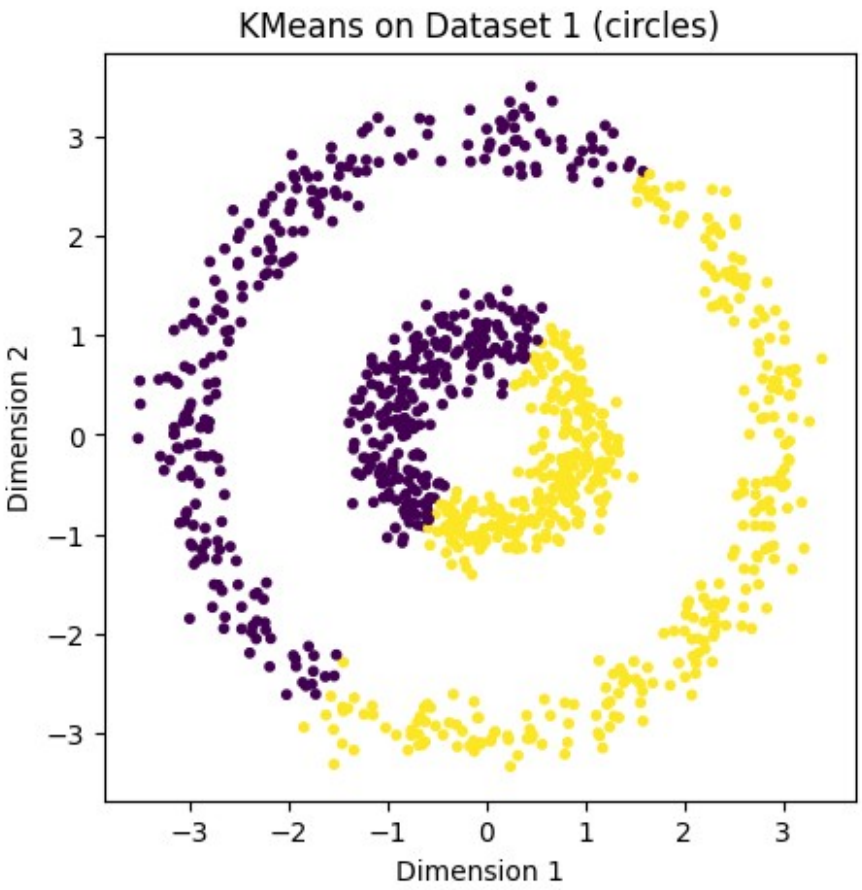
c)

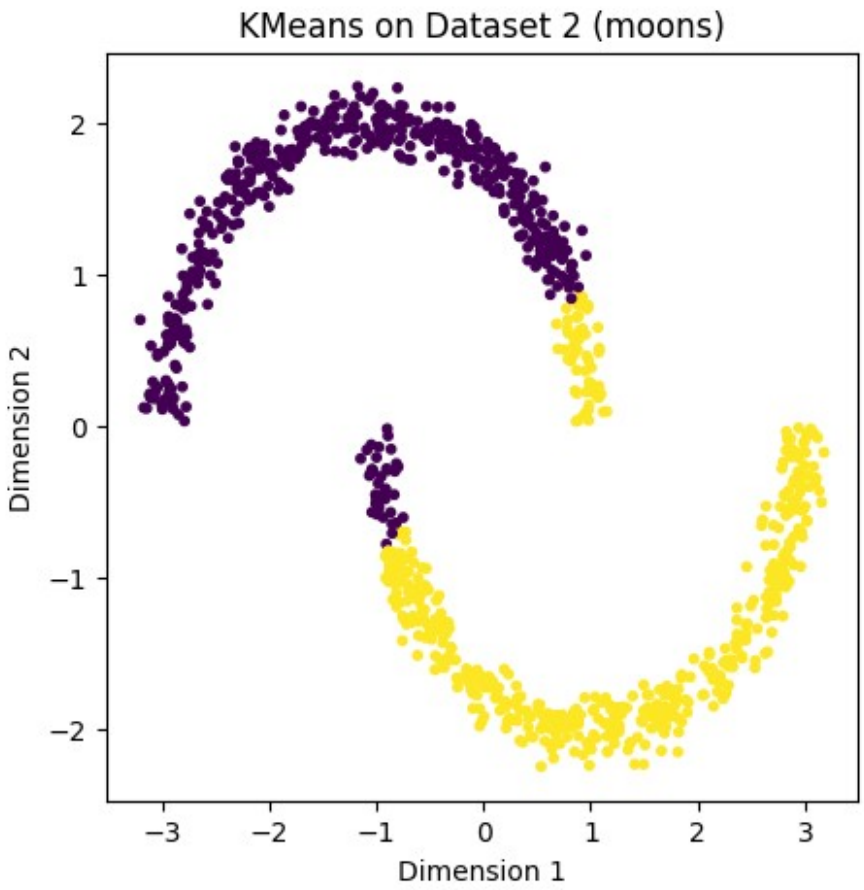
```
plt.errorbar(n_values, mean_ari, yerr=std_ari, fmt='-o', capsize=5,
label="Original data")
plt.errorbar(n_values, mean_ari_pca, yerr=std_ari_pca, fmt='-s',
capsize=5, label="PCA (3D projection)", color="orange")

plt.xlabel("Number of samples (n)")
plt.ylabel("Adjusted Rand Index (ARI)")
plt.title("Clustering performance with/without PCA projection")
plt.legend()
plt.grid(True)
plt.show()
```

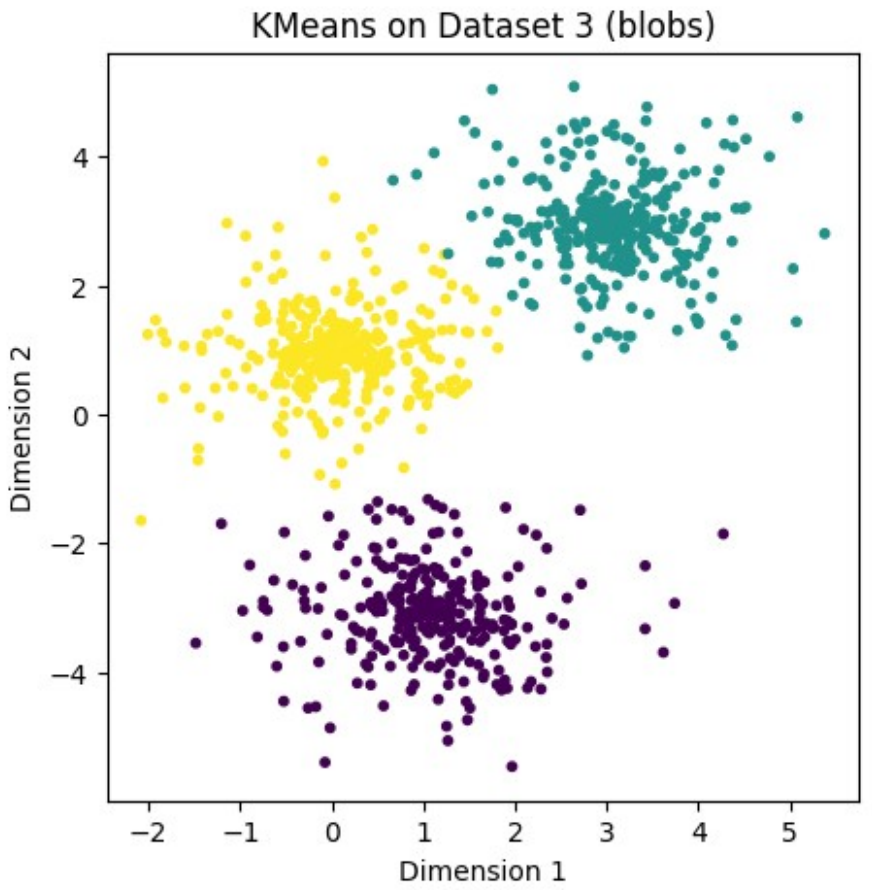
8.

```
from sklearn.cluster import MeanShift, KMeans , estimate_bandwidth,
SpectralClustering

import random
import math
random.seed(0)
# # Data set 1

X1 = [ ]
for i in range (1000):
    theta = random.uniform (0, 2 * math.pi )
    radius = random.gauss (0 , 0.2 ) + random.choice ( [1 , 3 ] )
    X1.append ( [ radius * math . cos ( theta ) , radius *
math.sin( theta ) ] )
X1 = np.array ( X1 )

# # Data Set 2
X2 = [ ]
for i in range (1000):
    theta = random.uniform (0, 2 * math . pi )
```

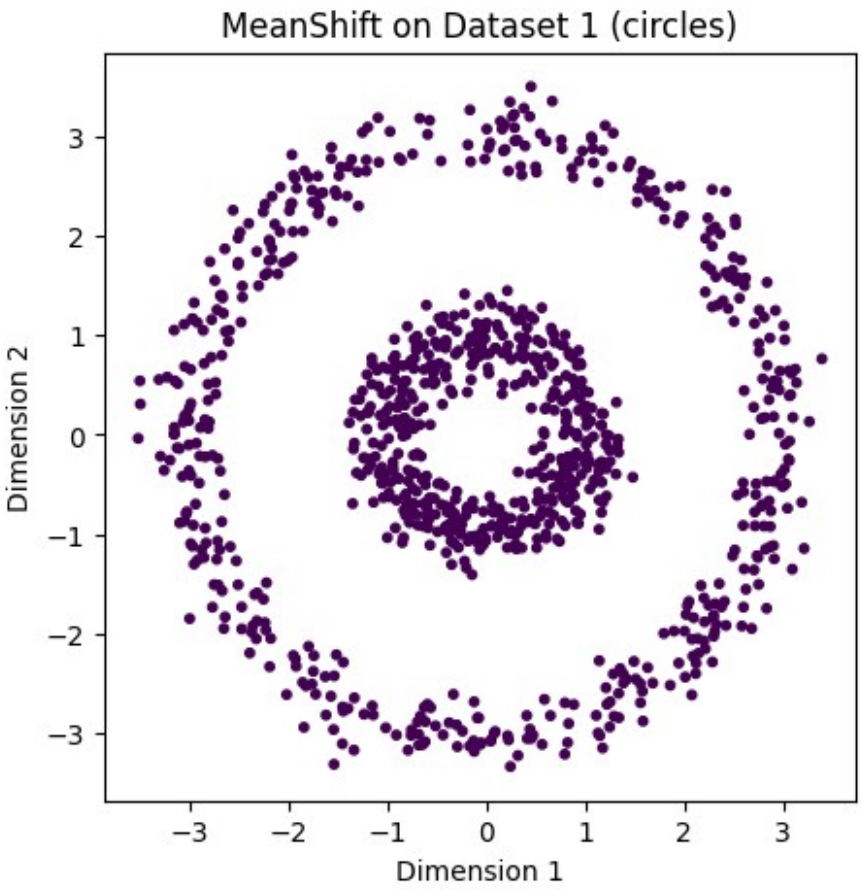
```
    radius = random.gauss (0 , 0.1) + 2

    if theta < math.pi :
        X2.append ( [ radius * math.cos ( theta ) -1 , radius *
math.sin ( theta ) ] )
    else :
        X2.append ( [ radius * math.cos ( theta ) +1 , radius *
math.sin ( theta ) ] )
X2 = np.array (X2)

# # Data Set 3
X3 = [ ]
for i in range ( 1000 ) :
    radius = random.gauss (0 , 1 )
    theta = random.uniform (0 , 2 * math . pi )
    center = random.choice ( [ [0 , 1 ] ,[3 , 3 ] ,[1 , - 3 ] ] )
    X3.append ( [ radius * math.cos ( theta ) + center [ 0 ] , radius
* math.sin ( theta ) + center[ 1 ] ] )
X3 = np.array ( X3 )

datasets = [(X1, 2, "Dataset 1 (circles)"),
            (X2, 2, "Dataset 2 (moons)"),
            (X3, 3, "Dataset 3 (blobs)")]

def plot_clusters(X, labels, title):
    plt.figure(figsize=(5,5))
    plt.scatter(X[:,0], X[:,1], c=labels, cmap="viridis", s=10)
    plt.xlabel("Dimension 1")
    plt.ylabel("Dimension 2")
    plt.title(title)
    plt.show()

#Kmeans
for X, k, title in datasets:
    km = KMeans(n_clusters=k, n_init=10, random_state=0)
    labels = km.fit_predict(X)
    plot_clusters(X, labels, f"KMeans on {title}")
```

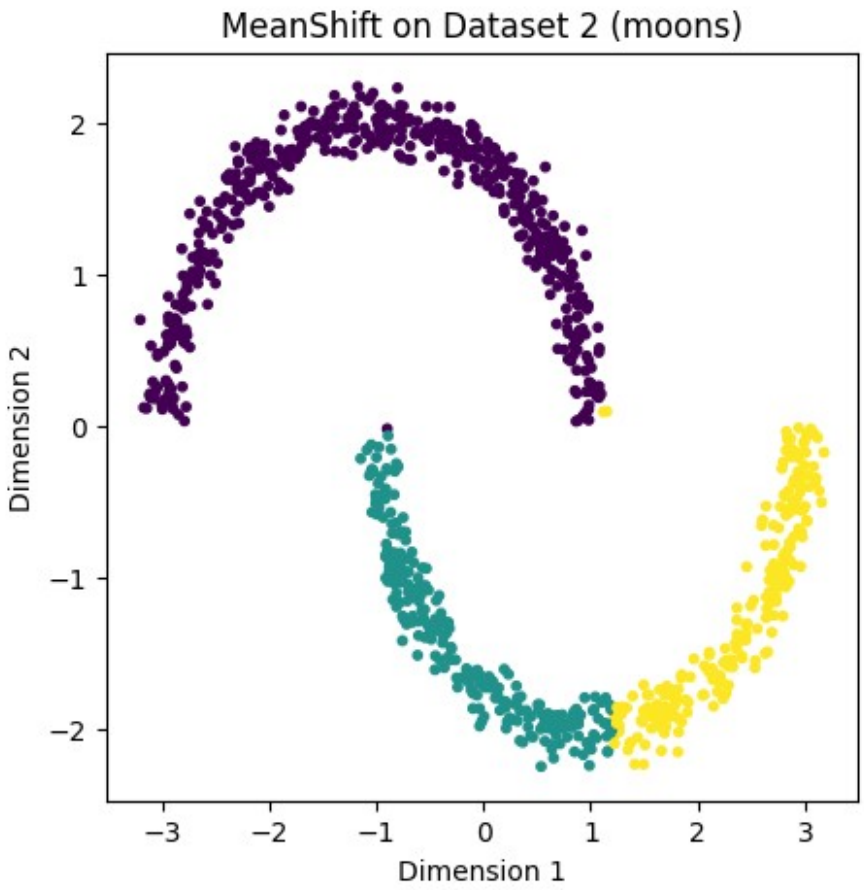

KMeans on Dataset 1 (circles)
Dimension 2
Dimension 1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

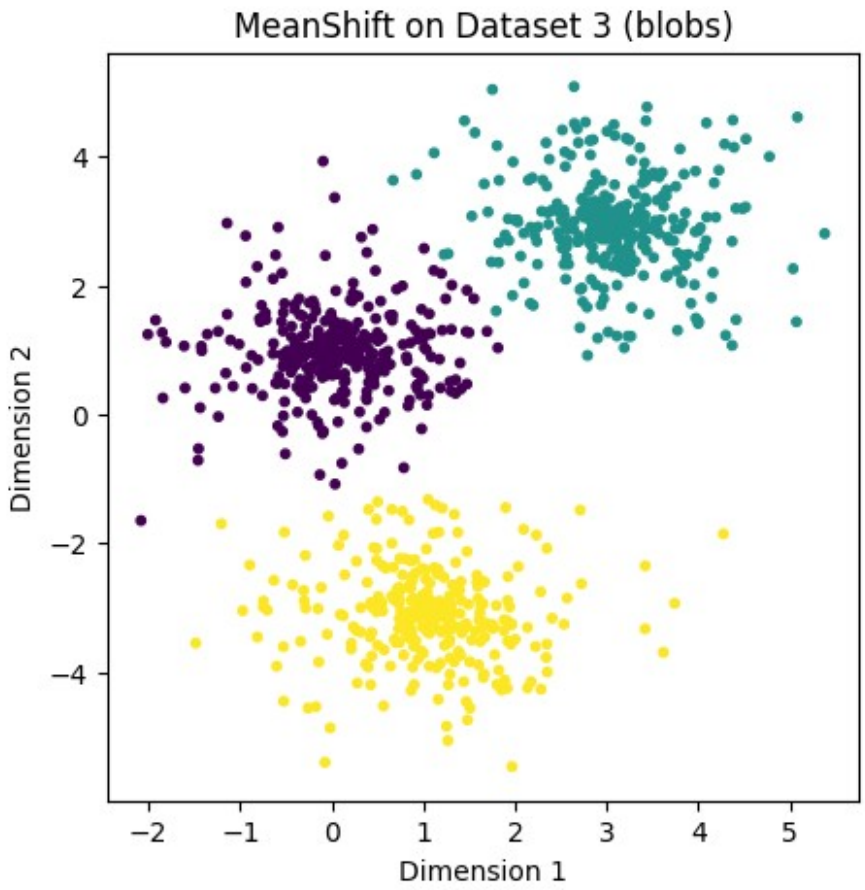

KMeans on Dataset 2 (moons)
Dimension 2
Dimension 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3




```
#Meanshift
for X, k, title in datasets:
    bandwidth = estimate_bandwidth(X, quantile=0.2)
    ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
    labels = ms.fit_predict(X)
    plot_clusters(X, labels, f"MeanShift on {title}")
```

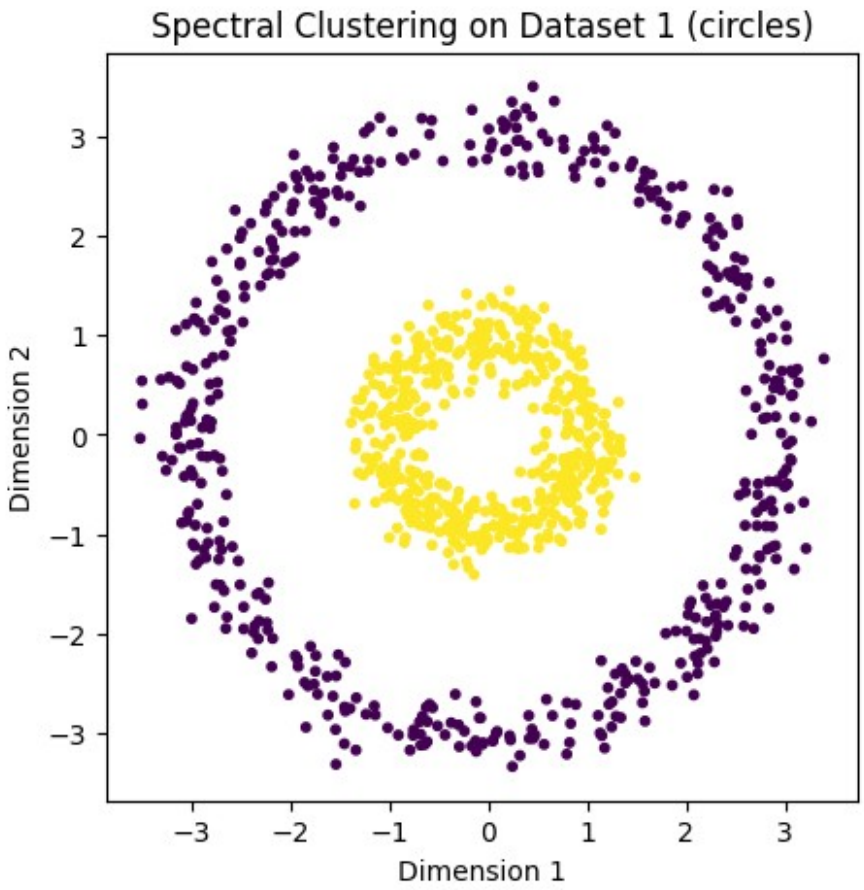

MeanShift on Dataset 1 (circles)
Dimension 2
Dimension 1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


MeanShift on Dataset 2 (moons)
Dimension 2
Dimension 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

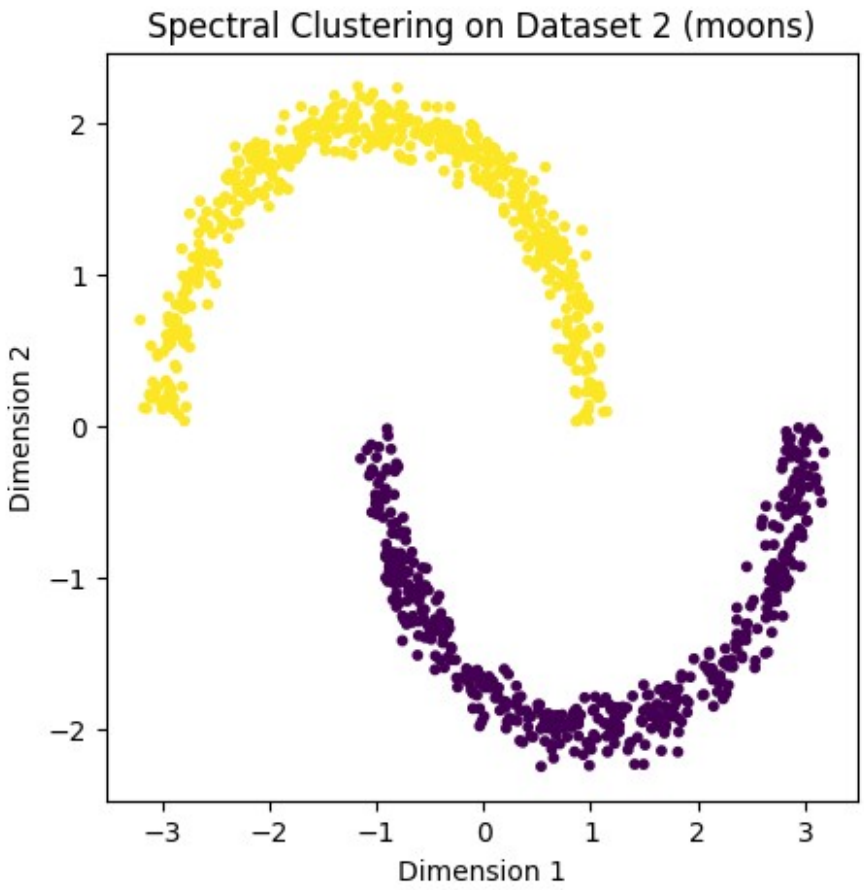


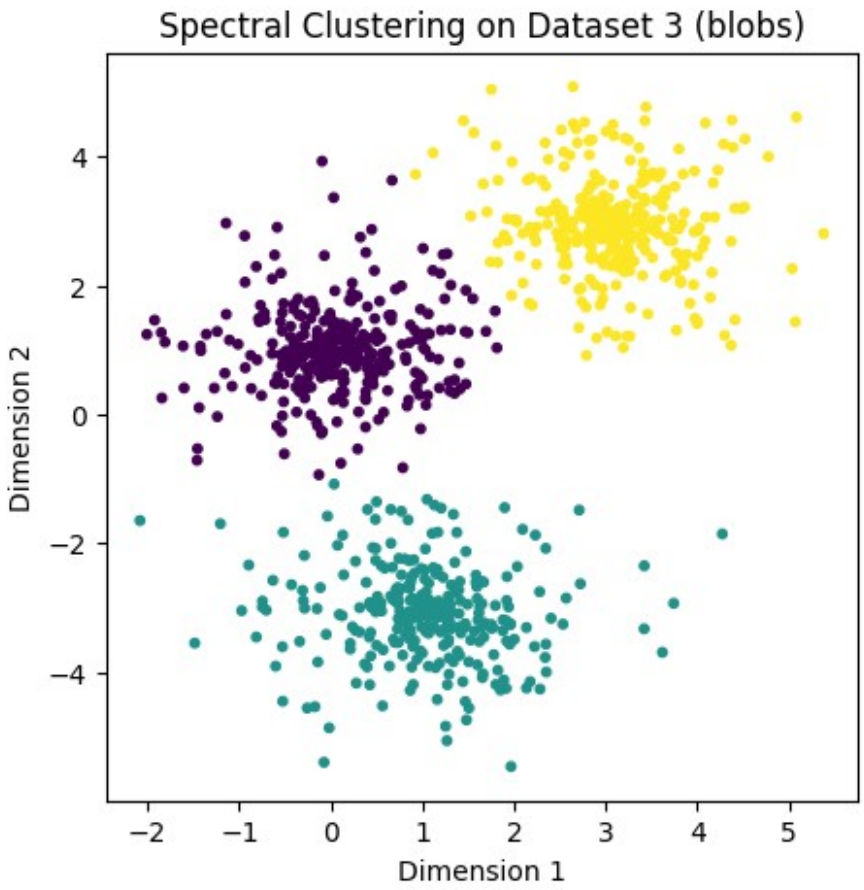

```
#Spectral
for X, k, title in datasets:
    sc = SpectralClustering(n_clusters=k,
affinity="nearest_neighbors", assign_labels="kmeans", random_state=0)
    labels = sc.fit_predict(X)
    plot_clusters(X, labels, f"Spectral Clustering on {title}")
```

```
/Users/kei/anaconda3/lib/python3.11/site-packages/sklearn/manifold/
_spectral_embedding.py:273: UserWarning: Graph is not fully connected,
spectral embedding may not work as expected.
  warnings.warn(
```




```
/Users/kei/anaconda3/lib/python3.11/site-packages/sklearn/manifold/
_spectral_embedding.py:273: UserWarning: Graph is not fully connected,
spectral embedding may not work as expected.
  warnings.warn(
```


Spectral Clustering on Dataset 2 (moons)
Dimension 2
Dimension 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


Spectral Clustering on Dataset 3 (blobs)
Dimension 2
Dimension 1
4
2
0
−2
−4
−2
−1
0
1
2
3
4
5