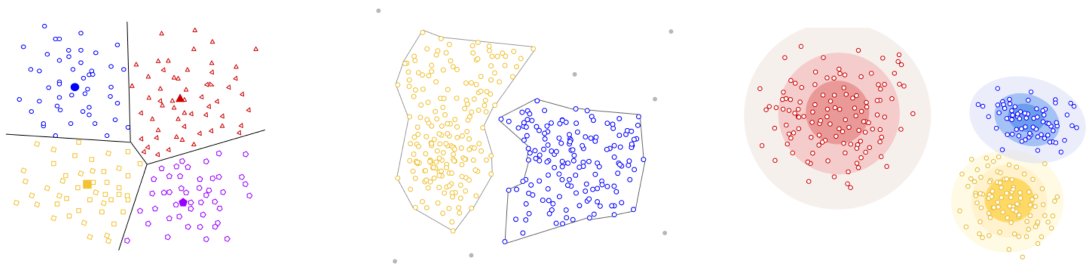


Midterm Review

▼ **What are the steps of PCA? Give your reason(s) for each step if possible.**

- Standardize the data: ensure all features contribute equally.
- Compute covariance matrix: Capture how features vary together.
- Compute the eigenvectors & eigenvalues from the covariance matrix or perform SVD: Identify directions of most variance.
- Sort eigenvalues in descending order and choose the k eigenvectors that correspond to the k largest eigenvalues where k is the number of dimensions of the new feature subspace $(k \leq d)$.
- Construct projection matrix $\mathbf{B}$ from the selected k eigenvectors.
- Transform the original dataset $\mathbf{X}$ via $\mathbf{B}$ to obtain a k-dimensional feature subspace $\mathbf{Y}$.

▼ **How to choose the the number of dimensions to reduce when building PCA?**

We can use the information provided by eigenvalues with some methods:

- Scree Test**:** pick k when slope starts leveling off.
- Elbow method**:** look for an "elbow" where the curve flattens out significantly.
- % of Cumulative Variance**:** choose k that explains a desired threshold of variance (e.g., 90%).

▼ **What are the limitations of PCA?**

- Linearity assumption: potentially oversimplifying structures.
- Outlier sensitivity: outliers can significantly influence the directions of greatest variance, potentially distorting the results of PCA.
- Low interpretability of principal components.
- Non-trivial solution when data entries are missing.
- Orthogonality assumption: since the principal components are by design orthogonal to each other. Depending on the situation, far "better" basis vectors may exist to summarize the data that are not orthogonal.
- PCA overfits to noise if $d > n$ (i.e. it is an inconsistent estimator of the subspace of maximal variance). → Sparse PCA

▼ **Why do we have to use t-distribution in t-SNE?**

Student's t-distribution is used to address a specific issue: **crowding problem** in the low-dimensional embedding. It allows for some distances to be less faithfully preserved in the embedding as the t-distribution has fatter tails, which can help preserve local structure while still preserving the global structure of the data as much as possible.

Additional advantages:

- We don't have to perform binary search for variance, instead we use the same distribution for all points (1 degree of freedom)
- t-distribution is more computationally convenient as it does not involve an exponential.

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l}\left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

▼ **Reconstruct PCA/Formulate PCA.**

We consider an independent and identically distributed (i.i.d.) dataset $X = x_1, x_2, \ldots, x_n$ with mean 0 and $x_n \in \mathbb{R}^D$. In general, if a dataset does not have a non-zero mean, we can standardize it before other steps.

Our aim is to maximize the variance of projected data $z_n = b^T x_n$, i.e:

$$\begin{aligned} \text{maximize } V &= \frac{1}{N}\sum_{n=1}^{N} z_n^2 \\ V &= \frac{1}{N}\sum_{n=1}^{N}\left(b^T x_n\right)^2 = \frac{1}{N}\sum_{n=1}^{N} b^T x_n x_n^T b \\ &= b^T\left(\frac{1}{N}\sum_{n=1}^{N} x_n x_n^T\right) b = b^T S b \end{aligned}$$

There's a problem with is objective function: V does not have an upper bound, i.e. increasing the magnitude of b will increase V. Therefore, we need to impose constraint $b^T b = 1$.

Now the problem becomes

$$\begin{gathered} \text{maximize } V = b^T S b \\ \text{s.t } b^T b = 1 \end{gathered}$$

The Lagrangian function for the problem is:

$$\mathcal{L}(b, \lambda) = b^T S b + \lambda\left(1 - b^T b\right)$$

Take the derivative of $\mathcal{L}$ w.r.t b, λ:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b} &= 2b^T S + 2\lambda b^T = 0 \iff & S \cdot b = \lambda b \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= 1 - b^T b \iff & b^T b = 1 \end{aligned}$$

We can see that b and λ must be eigenvectors and eigenvalues of S. Now we can construct projection matrix B by choose k eigenvectors coresponding to k largest eigenvalues, i.e.:

$$B = \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \cdots & b_k \\ | & | & & | \end{bmatrix}$$

where (λ_i, b_i) are eigenpairs and $\lambda_1 > \lambda_2 > \ldots > \lambda_k$.

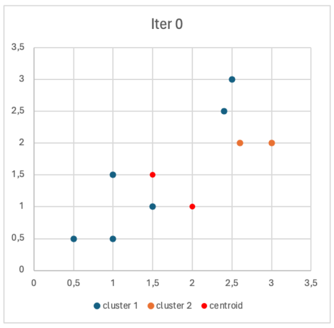
▼ **List the steps of the k-means algorithm.**

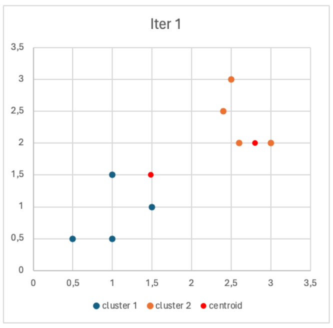
1. Specify number of clusters k.

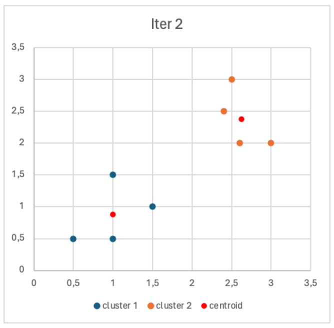
2. Initialize centroids by first shuffling the dataset and then randomly selecting k data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.
    - Compute the sum of the squared distance between data points and all centroids.
    - Assign each data point to the closest cluster (centroid).
    - Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

▼ **k-means** // Given the following pairs of points x,y with coordinates: (0.5; 0.5), (1; 0.5), (1; 1.5), (1.5; 1), (2.6; 2), (3; 2), (2.4; 2.5), (2.5; 3)

▼ **Apply the k-means algorithm with the number of centroids equal to 2.**







▼ **Initialize the centroid coordinates randomly, run the k-means algorithm for 3 iterations and return the coordinates of the 2 centroids.**

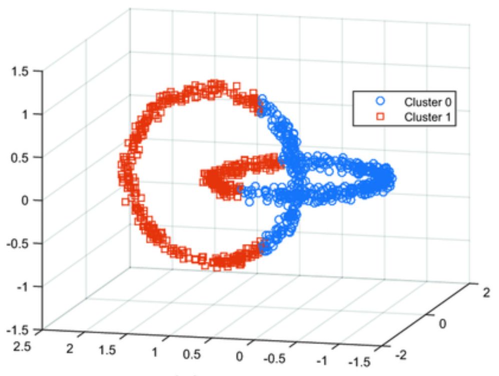
x	y	dis-c1	dis-c2	assign	dis-c1	dis-c2	assign	dis-c1	dis-c2	assign
0,5	0,5	1,414	1,581	1	1,402	2,746	1	0,625	2,834	1
1	0,5	1,118	1,118	1	1,111	2,343	1	0,375	2,481	1
1	1,5	0,500	1,118	1	0,483	1,868	1	0,625	1,846	1
1,5	1	0,500	0,500	1	0,500	1,640	1	0,515	1,777	1
2,4	2,5	1,345	1,552	1	1,357	0,640	2	2,145	0,257	2
2,5	3	1,803	2,062	1	1,812	1,044	2	2,601	0,637	2
2,6	2	1,208	1,166	2	1,223	0,200	2	1,956	0,376	2
3	2	1,581	1,414	2	1,597	0,200	2	2,295	0,530	2
centroid		**x**	**y**		**x**	**y**		**x**	**y**	
1		1,5	1,5		1,483	1,500		1,000	0,875	
2		2	1		2,800	2,000		2,625	2,375	

▼ **k-means** // Draw 3 different cases with data sets where clustering does not work/is not good using k-means. State the reason. Suggest a suitable algorithm for that data set and explain why it is suitable.

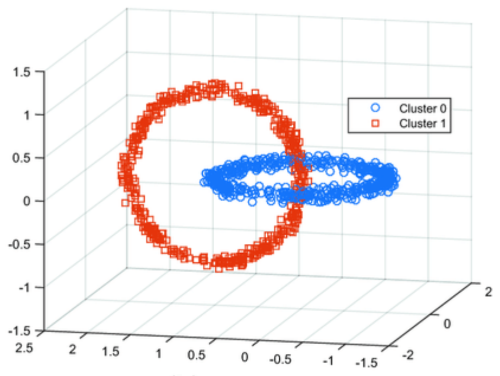
1. **Arbitrary shaped clusters → DBSCAN**

    Reason: K-means assumes that clusters are spherical, meaning they have similar variance in all directions. When clusters are elongated or irregularly shaped, k-means may fail to accurately identify the clusters because it tries to fit spherical clusters to non-spherical data.

    Solution: DBSCAN is suitable for datasets with non-spherical clusters because it does not assume any specific shape of clusters. It identifies clusters based on density and can handle irregularly shaped clusters effectively.



(a) k-means

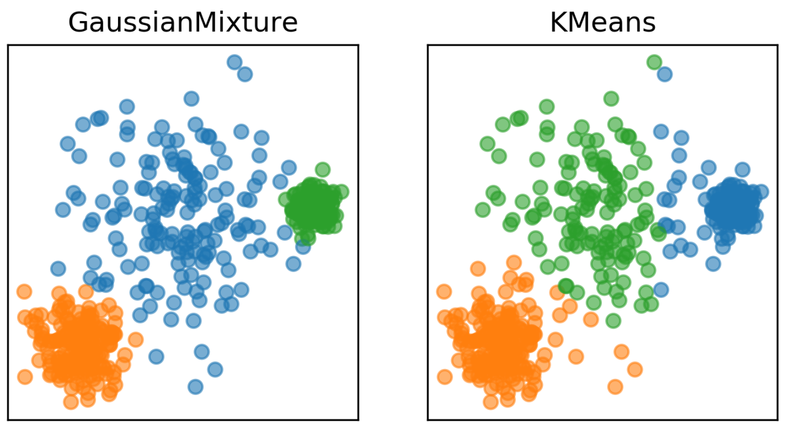


(b) DBSCAN

2. **Clusters with uneven density → GMM**

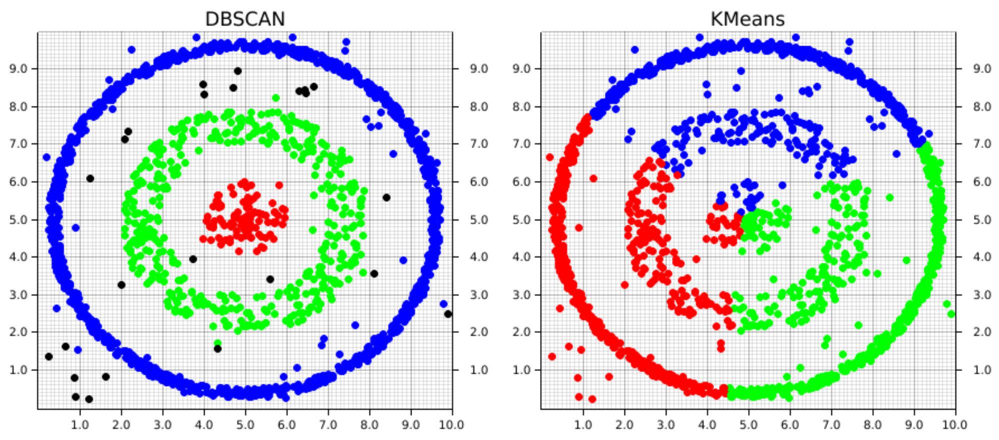
   Reason: K-means minimizes the distance between points and their centroid. The spread-out cluster will have a larger average distance, causing the centroid to drift towards the denser cluster, leading to uneven and poorly separated clusters.

   Solution: GMM assumes data points come from a mixture of Gaussian distributions (bell curves). It can model clusters with varying densities by fitting multiple Gaussians to the data. Denser clusters will have narrower Gaussians, while looser clusters will have broader ones.



3. **Dataset with Outliers → DBSCAN**

   Reason: K-means is sensitive to outliers. These faraway points can significantly distort the centroids, leading to poorly defined clusters that may even include outliers themselves.
   Solution: DBSCAN is robust to outliers. It ignores points that don't have enough neighbors within a certain radius, effectively excluding them from the clusters. This allows it to identify the underlying structure of the data without being influenced by outliers.



▼ **GMM algorithm.**

The Gaussian Mixture Model (GMM) algorithm is a probabilistic approach used for clustering and density estimation.

▼ **Loss function.**

Suppose that we are given a training set $\left\{x^{(1)}, \ldots, x^{(n)}\right\}$ as usual. Since we are in the unsupervised learning setting, these points do not come with any labels.

We wish to model the data by specifying a joint distribution $p\left(x^{(i)}, z^{(i)}\right)=p\left(x^{(i)} \mid z^{(i)}\right) p\left(z^{(i)}\right)$. Here,

- $z^{(i)} \sim \operatorname{Multinomial}(\phi)$
  - $\phi_j \geq 0, \sum_{j=1}^k \phi_j=1$
  - ϕ_j: $p\left(z^{(i)}=j\right)$
- $x^{(i)} \mid z^{(i)}=j \sim \mathcal{N}\left(\mu_j, \Sigma_j\right)$

We let k denote the number of values that the $z^{(i)}$'s can take on. Thus, our model posits that each $x^{(i)}$ was generated by randomly choosing $z^{(i)}$ from $\{1, \ldots, k\}$, and then $x^{(i)}$ was drawn from one of k Gaussians depending on $z^{(i)}$. This is called the mixture of Gaussians model. Also, note that the $z^{(i)}$'s are **latent** random variables, meaning that they're hidden/unobserved. This is what will make our estimation problem difficult.

The parameters of our model are thus ϕ, μ and Σ. To estimate them, we can write down the likelihood of our data:

$$\ell(\phi, \mu, \Sigma)=-\sum_i \log \left[\sum_k \phi_k \mathcal{N}\left(x_i \mid \mu_k, \Sigma_k\right)\right]$$

- $\sum_i$: Summation over all data points x_i.
- ϕ_k: Probability of the k-th Gaussian component.
- $N(x_i|\mu_k, \Sigma_k)$: PDF of the x_i under the k-th Gaussian component.

However, we'll find that it is not possible to find the maximum likelihood estimates of the parameters in closed form. Therefore, while a direct solution isn't feasible, alternative methods like **numerical optimization** and the **EM algorithm** are used to effectively estimate the parameters.

▼ Why is it not possible?

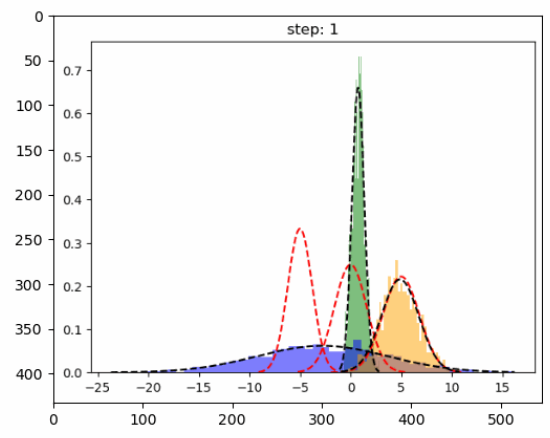
- **Nested Sums:** The formula involves intertwined summations, making it hard to solve algebraically.
- **Non-linearity:** Products within the logarithm term prevent straightforward solutions when taking derivatives.
- **Matrix Complexities:** Deriving with respect to the matrix (Σ) requires advanced matrix calculus.

▼ **The EM algorithm.**

The EM algorithm is an iterative algorithm that has two main steps.

- In the E-step, it tries to "guess" the values of the $z^{(i)}$'s. In the M-step, it updates the parameters of our model based on our guesses.
- Since in the M-step we are pretending that the guesses in the first part were correct, the maximization becomes easy.

▼ **List the steps of the GMM algorithm.** Reference.



Step 01: Initialize mean, covariance, and weight parameters

1. Mean (μ): initialize randomly.
2. Covariance (Σ): initialize randomly.
3. Weight (mixing coefficients) (ϕ): equal for all clusters.

   Weight is fraction per class refers to the likelihood that a particular data point belongs to each class. In the beginning, this will be equal for all clusters. *Assume that we fit a GMM with three components. In this case weight parameter might be set to 1/3 for each component, resulting in a probability distribution of (1/3, 1/3, 1/3).*

Repeat step 2 and 3 until convergence.

Step 02: Expectation Step (E-step)

In the E-step, we calculate the **posterior probability** of our parameters the $z^{(i)}$'s, given the $x^{(i)}$ and using the current setting of our parameters. i.e., using Bayes rule, we obtain:

$$\begin{aligned} w_j^{(i)} &= p\left(z^{(i)} = j \mid x^{(i)}; \phi, \mu, \Sigma\right) && (1)\\ &= \frac{p\left(x^{(i)} \mid z^{(i)} = j; \mu, \Sigma\right) p\left(z^{(i)} = j; \phi\right)}{\sum_{k=1}^{K} p\left(x^{(i)} \mid z^{(i)} = k; \mu, \Sigma\right) p\left(z^{(i)} = k; \phi\right)} && (2)\\ &= \frac{\phi_j N\left(x_i \mid \mu_j, \Sigma_j\right)}{\Sigma_{k=1}^{K} \phi_k N\left(x_i \mid \mu_k, \Sigma_k\right)} && (3) \end{aligned}$$

Where:

- $w_j^{(i)}$: **Posterior probability** of data point $x^{(i)}$ belonging to the jth distribution.

- $p(x^{(i)} \mid z^{(i)} = j; \mu, \Sigma)$: **Likelihood** of observing $x^{(i)}$ given it belongs to distribution j.
- $p(z^{(i)} = j; \phi)$: **Prior probability** of data point $x^{(i)}$ belonging to component j (represented by the mixing proportion ϕ_j).

Here, $p\left(x^{(i)} \mid z^{(i)} = j; \mu, \Sigma\right)$ is given by evaluating the density of a Gaussian with mean μ_j and covariance Σ_j at $x^{(i)}$; $p\left(z^{(i)} = j; \phi\right)$ is given by ϕ_j, and so on. The values $w_j^{(i)}$ calculated in the E-step represent our "soft" guesses for the values of $z^{(i)}$.

$$w_j^{(i)} = \frac{\phi_j N\left(x_i \mid \mu_j, \Sigma_j\right)}{\Sigma_{k=1}^{K} \phi_k N\left(x_i \mid \mu_k, \Sigma_k\right)}$$

The E-step outputs a set of **responsibilities (w)**, where each data point has a weight associated with every Gaussian component in the mixture model. These weights reflect the **updated belief** about the latent variable assignments based on the current parameter estimates.

Step 03: Maximization Step (M-step)

In this step, the algorithm uses the **responsibilities (weights)**, denoted by $wj^{(i)}$, calculated in the E-step, to update the estimates of the model's parameters.

The M-step updates the model parameters (mixing proportions: ϕ, means: μ, and covariances: Σ) based on the following equations:

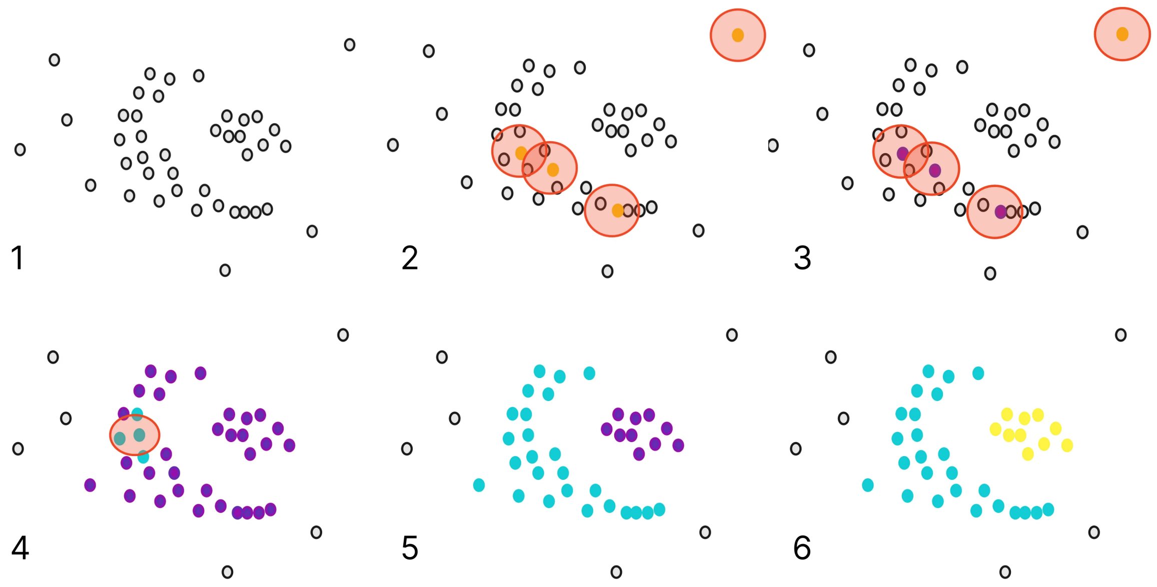
$$\phi_j = \frac{\sum_{i=1}^{m} w_j^{(i)}}{m}$$
$$\mu_j = \frac{\sum_{i=1}^{m} w^{(i)} j x_i}{\sum i = 1^m w_j^{(i)}}$$
$$\Sigma_j = \frac{\sum_{i=1}^{m} w_j^{(i)} \left(x_i - \mu_j\right)^2}{\sum_{i=1}^{m} w_j^{(i)}}$$

This updated estimate is used in the next E-step to compute new responsibilities for the data points.

▼ **Aspect Comparison: K-Means vs DBSCAN.**

Feature	k-means	DBSCAN
Type	Centroid-based	Density-based
Cluster shape	Assumes clusters are spherical	Can handle clusters of arbitrary shape
Number of clusters	Must be specified in advance	Automatically determines clusters
Noise handling	May assign noises to normal clusters	Can identify outliers/anomalies as noise
Performance	Efficient for large datasets with a small number of clusters	Less efficient for high-dimensional data or clusters with varying densities
Initialization	Random initialization of centroids	Not applicable; density-based
Parameter tuning	Requires tuning of the number of clusters (k)	Requires tuning of epsilon (ε) and minimum points (MinPts)
Scalability	Scales well with large datasets and low dimensions	Less scalable with high-dimensional data or varying densities
Robustness	Sensitive to initial centroid placement	Robust to outliers and noise
Applications	Commonly used in applications where cluster shapes are relatively uniform	Suitable for applications with irregular cluster shapes or varying cluster densities

▼ **DBSCAN //** State how DBSCAN works in your understanding (you can use math to explain or not). Reference: Medium.



- Step 1

  Initial dataset: we aim to groups points that are densely packed into clusters.

  In this data, we would expect 2 cluster with outliers should be assigned as noises.

- Step 2

  First, we **draw circle** with radius r (hyperparameter) around each point and count numnber of neighbors n (points within the circle).

- Step 3

  Next, we consider points that have $n \geq \text{MinEps}$ **core points.**

  In this example, we set $\text{MinEps} = 3$.

- Step 4

  Then, we randomly select a core point and assign it as the first point in our first cluster. Other points that are close to this point are also assigned to the same cluster *(i.e., within the circle of the selected point).*

- Step 5

  Then, we extend it to the other points that are close.

  We stop when we cannot assign more core points to the first cluster.

- Step 6

  Now, we still have points that are not assigned. We randomly **select another point from them and start over**.

  Points which are not assigned to any clusters are considered **noises**.

▼ **DBSCAN //** Apply DBSCAN algorithm with radius (ϵ) = 1.5 and min points = 3. Identify clusters and noise points.