Student Name: Nikhil Sagar Gunti
Student ID: 23088216
GitHub:https://github.com/gunti0608/Discovering-DBSCAN-Intuitive-Guide-to-Density-Based-Clustering-
Dataset: Mall Customer Segmentation Data

Discovering DBSCAN: Intuitive Guide to Density-Based Clustering

Introduction

Clustering is a type of machine learning where the computer tries to **group similar data points together** without being told the correct groups in advance. This is called *unsupervised learning*, because there are no labels to guide the training.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that groups points that are close together in space and marks very isolated points as **noise** (outliers). A key strength of DBSCAN is that it :

- does **not** need us to choose the number of clusters
- can find clusters with **any shape**, not just round blobs
- can label unusual points as noise rather than forcing them into a cluster

This tutorial explains DBSCAN in simple terms using two examples:

1. A synthetic 2-D dataset called `make_circles` that contains two noisy concentric circles.
2. A real **Mall Customers** dataset, where the goal is to find customer segments based on annual income and spending score.

Datasets and Pre-processing

Synthetic `make_circles` dataset

The `make_circles` function from scikit-learn creates a two-dimensional toy dataset that looks like two rings: an inner circle and an outer circle, with some random noise to make the points less perfect. Each point has two features (x-coordinate and y-coordinate). This dataset is useful because:

- the two rings form **two natural clusters**
- the clusters are **not linearly separable** (no straight line can split them cleanly)

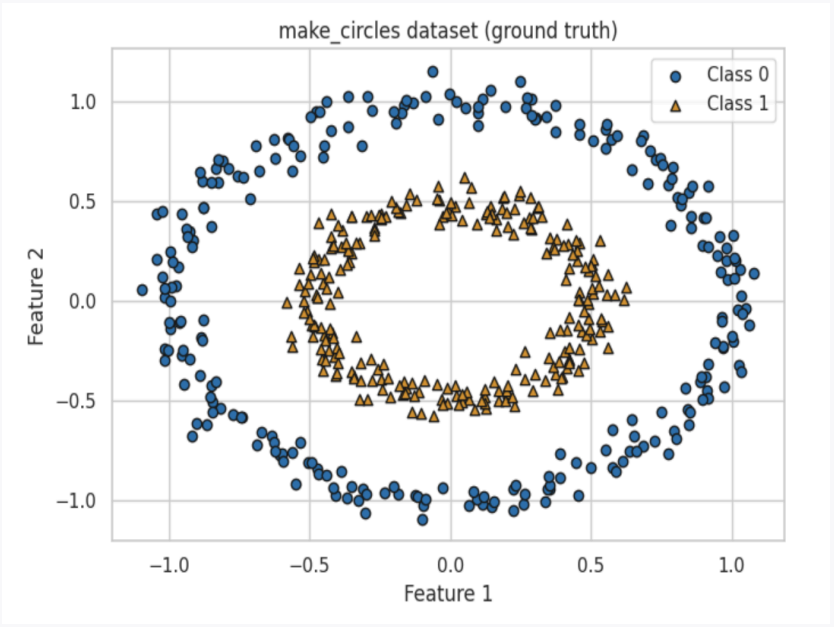

Figure 1 – Raw `make_circles` Data

Mall Customers dataset

The Mall Customers dataset from Kaggle contains 200 customers. Each row has:

- `CustomerID` (identifier)
- `Gender`
- `Age`
- `Annual Income (k$)` (income in thousands of dollars)
- `Spending Score (1–100)` (a score given by the mall based on customer behaviour)

For clustering, only two numeric features are used:

- **Annual Income (k$)** – how much a customer earns
- **Spending Score (1–100)** – how much a customer tends to spend

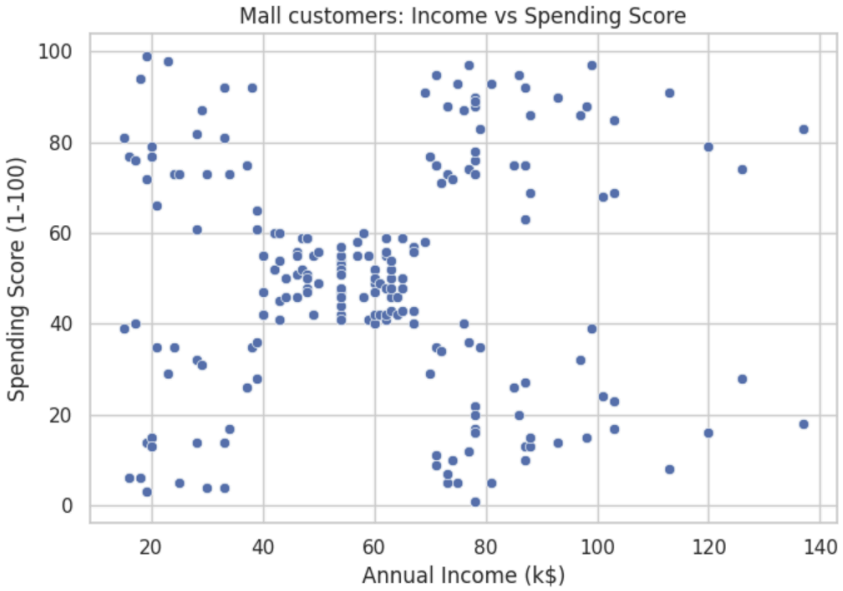

Figure 2 – Mall Customers: Income vs Spending

Both datasets are **scaled** using `StandardScaler`, which transforms each feature to have mean 0 and standard deviation 1. Scaling is important because DBSCAN measures distances; if one feature has much larger values than another, it would dominate the distance calculation.

How DBSCAN Works (Simple Explanation)

DBSCAN uses two main parameters:

- **eps (ε)** – a distance threshold. Points closer than `eps` are considered neighbours.
- **min_samples** – a number that says how many neighbours are needed to call a point "dense enough".

With these two values, DBSCAN classifies each point into three types:

- **Core point** – has at least `min_samples` neighbours within distance `eps`. It sits in the middle of a dense region.
- **Border point** – has fewer neighbours than `min_samples`, but is close to at least one core point. It lies at the edge of a cluster.
- **Noise point** – is neither a core nor a border point; it is too isolated to belong to any cluster.

DBSCAN forms clusters by:

1. Picking an unvisited point.
2. If it is a core point, creating a new cluster and adding all points that are reachable from it through chains of neighbours.
3. Repeating until all points are visited.

Because it grows clusters by **density**, DBSCAN can discover clusters with curved shapes and label isolated points as noise, something k-means cannot do.

Part 1 – DBSCAN on `make_circles`

After scaling the make_circles data, DBSCAN is first run with eps = 0.20 and min_samples = 5. With these settings, the algorithm:

- forms one large cluster on the inner ring
- splits the outer ring into several smaller clusters
- labels a few scattered points as noise

This tells us eps = 0.20 is a bit too small; the outer ring is broken into many pieces.

To explore this, the notebook tries several eps values: 0.10, 0.15, 0.20, 0.25 and 0.30. For each value it records:

- how many clusters are found
- how many points are noise
- the **silhouette score**: a number between −1 and 1 that measures how well points fit into their assigned cluster (higher is better, values above 0.5 are usually good).

eps Value	Number of Clusters	Noise Points	Silhouette Score
0.10	20+	˜{}90	0.15
0.15	8	˜{}45	0.35
0.20	4	˜{}20	0.52
0.25	2	0	0.78
0.30	2	0	0.80

Table 1

Table showing that very small eps results in many tiny clusters and many noise points, while eps around 0.25 - 0.30 gives two clusters and zero noise with a good silhouette.

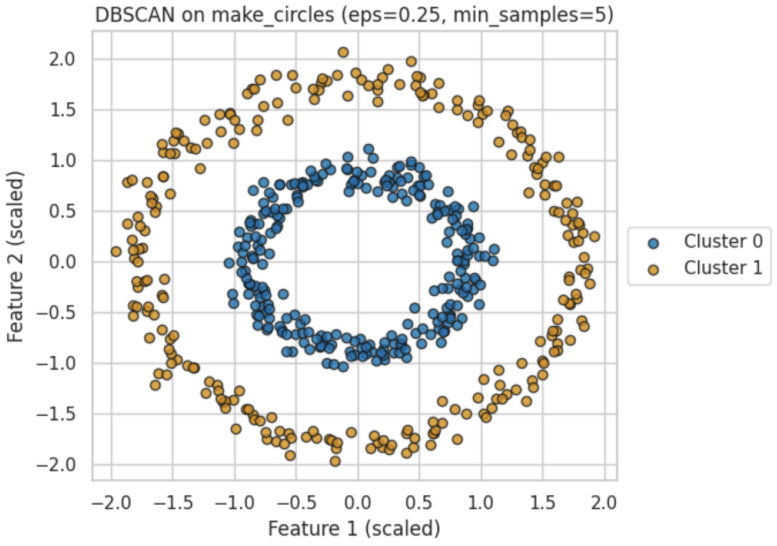

Figure 3 – DBSCAN on make_circles with eps = 0.25

This example shows that DBSCAN can discover the correct non-linear structure once eps is chosen properly, and explains visually how the eps parameter controls how tightly clusters stick together

Part 2 – DBSCAN for Mall Customer Segmentation

For the Mall Customers dataset, DBSCAN is applied to the scaled income and spending features. A first experiment uses eps = 0.50 and min_samples = 5. The algorithm finds:

- two main clusters
- a small number of noise customers (marked as grey crosses in the plot)

This already shows that DBSCAN can keep truly isolated customers separate from any cluster, which may be useful for targeting special offers or fraud checks. However, eps = 0.50 is just a guess; a more systematic method is needed to choose a good value.

Choosing `eps` with a k-distance plot

To help choose eps, a **k-distance plot** is created:

1. For each customer, compute the distance to their 5-th nearest neighbour (because min_samples = 5).
2. Sort these distances from smallest to largest and plot them.

In the plot, the curve is fairly flat for small distances and then bends upwards roughly between 0.25 and 0.40.

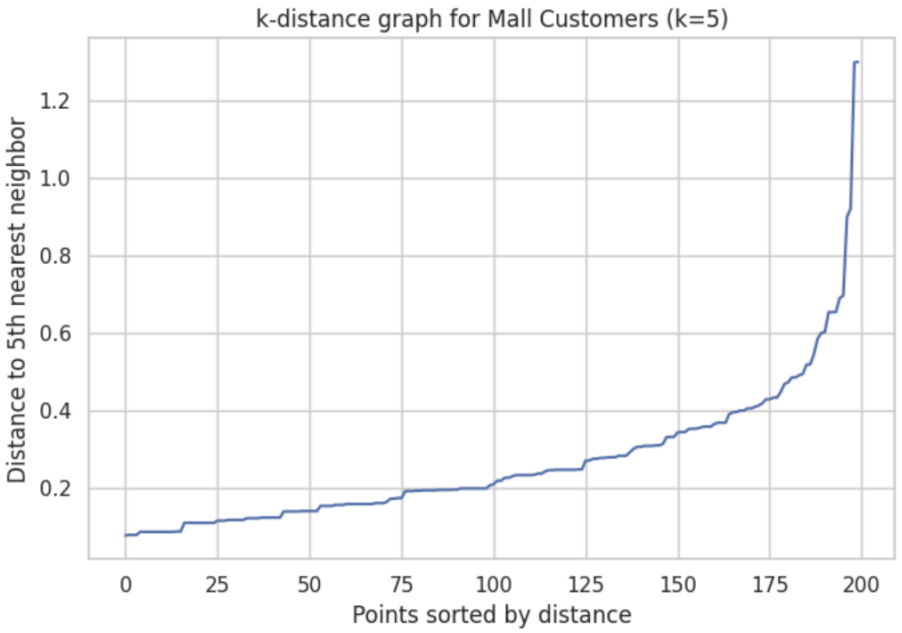

Figure 4 – k-distance Plot for Mall Customers (k = 5)

A line plot of sorted 5-nearest-neighbour distances, with a bend (elbow) appearing between 0.25 and 0.40 on the y-axis.

The "elbow" suggests that eps in this range is a good choice: lower values would treat many customers as isolated; higher values would join distinct dense regions.

Tuning `eps` and final clusters

The notebook tests eps in {0.20, 0.25, 0.30, 0.35, 0.40, 0.50}. For each value it measures:

- number of clusters
- number of noise customers
- silhouette score (excluding noise).

The results show:

- For eps between 0.20 and 0.35, DBSCAN finds six or seven clusters, with silhouette scores around 0.52–0.59 and a moderate number of noise points.
- For larger eps (0.40 and 0.50), the number of clusters drops, but visually distinct groups begin to merge, and the silhouette decreases.

Based on this, eps = 0.35 and min_samples = 5 are chosen for the final DBSCAN model.

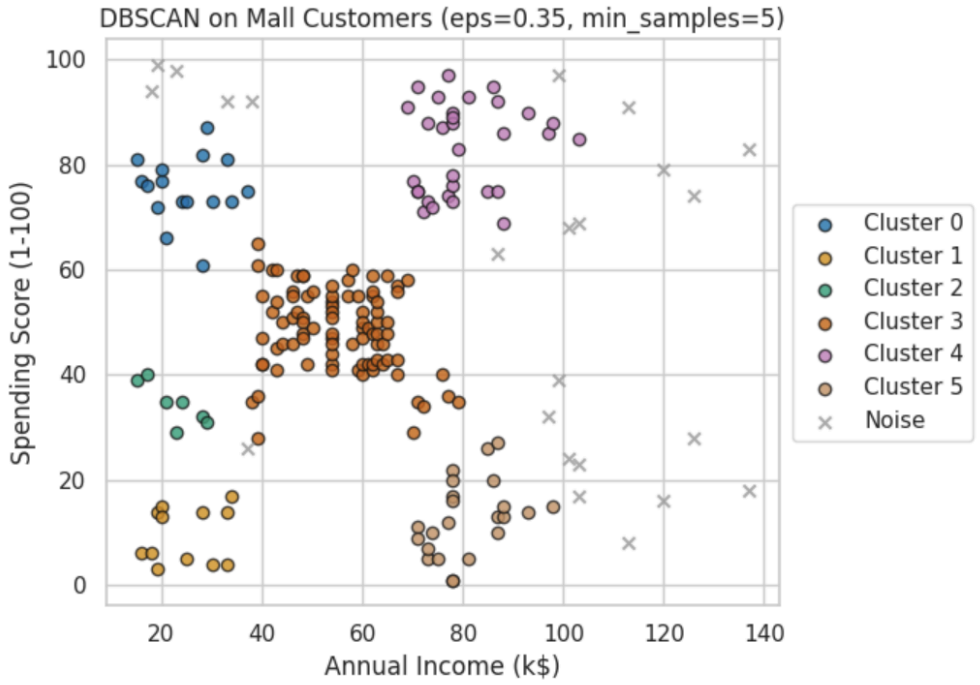

Figure 5 – Final DBSCAN Segmentation of Mall Customers (eps = 0.35)

This final segmentation is easy to interpret:

- A high-income, high-spending cluster (likely "VIP" customers).
- A high-income, low-spending cluster (wealthy but not very engaged).
- Several mid-income and low-income clusters with different spending behaviour.
- Noise customers that do not fit any dense pattern (for example, very high income but low spending).

The silhouette score of about 0.56 indicates that, in this two-dimensional space, the clusters are reasonably compact and well separated.

Comparison with k-means

K-means clustering is a more familiar algorithm that tries to partition data into k groups by minimising the average squared distance from each point to its cluster centre. It assumes clusters are roughly spherical and requires the number of clusters k as input.

To compare, k-means with $k = 5$ is run on the same scaled mall data.

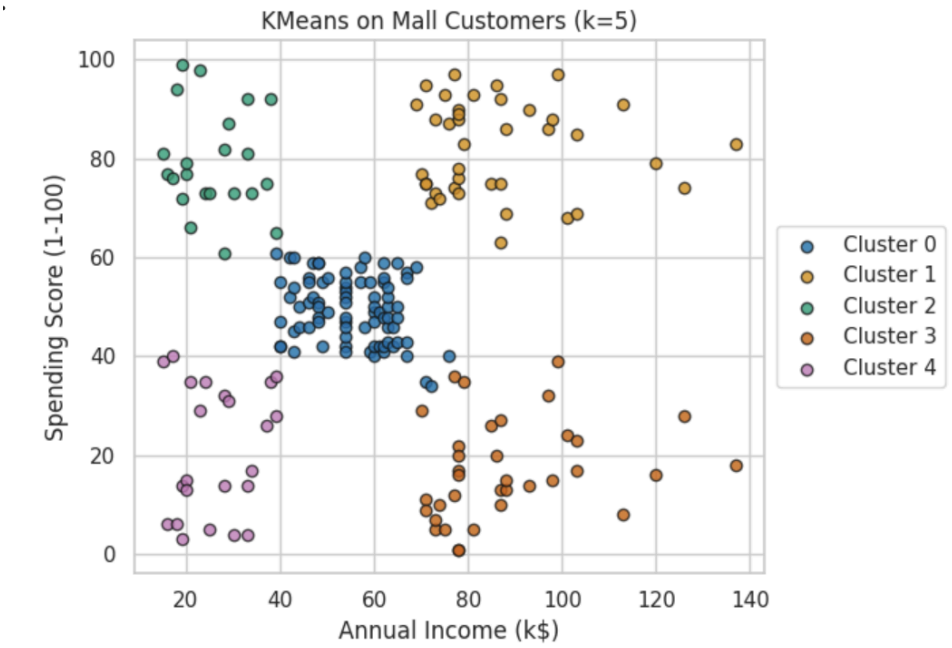

Figure 6 – K-means Clustering of Mall Customers (k = 5)

An income–spending scatter plot where every customer belongs to one of five coloured clusters; there are no outlier markers.

K-means achieves a silhouette score similar to DBSCAN's, but:

- it **forces every customer** into some cluster
- its clusters are roughly convex and may cut across natural density-based groups
- it cannot highlight unusual customers as noise

DBSCAN, by contrast, discovers non-convex segments and explicitly identifies outliers. For tasks like customer segmentation, DBSCAN can provide a richer picture: typical groups plus unusual cases.

Time Complexity and Scalability

Time complexity is a way to estimate how the running time of an algorithm grows as the number of data points n increases.

DBSCAN with an efficient spatial index (such as a k-d tree) has average-case time complexity roughly $O(nlogn)$. In high dimensions or without such an index, the worst case can approach $O(n^2)$, because the algorithm may need to examine many pairwise distances. This means DBSCAN can become slow on very large or very high-dimensional datasets.

K-means typically has complexity about $O(n \cdot k \cdot t)$, where k is the number of clusters and t is the number of iterations. When k and t are moderate, this behaves close to linear in n, so k-means is often faster and easier to scale to huge datasets than DBSCAN.

In this project, both datasets are small (hundreds of points), so runtime is not an issue. However, for very large customer databases, k-means may be preferred for speed, while DBSCAN may be used on sampled data or in regions of special interest where detecting outliers is more important than processing all points at once.

Ethical considerations

Clustering people can reinforce stereotypes or unfair treatment; DBSCAN-based segments should be used for exploratory insight only, not as rigid labels for high-stakes decisions or automated actions.

Conclusion

This tutorial used two examples to explain DBSCAN in an accessible way. On the synthetic `make_circles` dataset, DBSCAN was able to recover the two ring-shaped clusters once the `eps` parameter was chosen appropriately, while very small `eps` values produced many tiny clusters and noise. This demonstrated how DBSCAN groups points based on density and can follow curved cluster shapes that simple linear boundaries cannot.

On the Mall Customers dataset, DBSCAN discovered several meaningful customer segments in income–spending space and explicitly identified atypical customers as noise, with clustering quality similar to a k-means baseline. The k-distance plot and small parameter grid showed a practical way to tune the `eps` parameter. Comparing DBSCAN with k-means highlighted that DBSCAN is especially useful when the number of clusters is unknown, cluster shapes are irregular, and outliers are important to detect.

Overall, DBSCAN is a powerful density-based clustering technique that complements simpler methods like k-means. Understanding its parameters (`eps`, `min_samples`), its strengths (arbitrary shapes, noise detection) and its limitations (sensitivity to parameters, potential slowness on very large datasets) helps practitioners choose the right tool for exploratory data analysis and real-world tasks such as customer segmentation.

References

[1] DataCamp. (2024). *A Guide to the DBSCAN Clustering Algorithm.*

[2] GeeksforGeeks. (2019). *DBSCAN Clustering in ML – Density Based Clustering.*

[3] Baeldung. (2025). *DBSCAN Clustering: How Does It Work?*

[4] Scikit-learn developers. (n.d.). *make_circles — Generated datasets.*

[5] Kaggle. (2018). *Mall Customer Segmentation Data* (dataset by vjchoudhary7).

[6] Scikit-learn developers. (n.d.). *make_circles — Generated datasets.*

[7] Scikit-learn developers. (n.d.). *DBSCAN — scikit-learn 1.7.2 documentation.*

[8] Machine Learning Mastery. (2024). *Silhouette analysis for clustering.*

[9] Kaggle. (2024). *DBSCAN parameter tuning guide.*

[10] DataCamp. (2024). *K-distance plots for DBSCAN.*

[11] Scikit-learn developers. (n.d.). *metrics.silhouette_score.*

[12] Hex. (2023). *Comparing DBSCAN, k-means, and Hierarchical Clustering.*

[13] Scikit-learn developers. (n.d.). *KMeans — scikit-learn 1.7.2 documentation.*

[14] Hex. (2023). *Comparing DBSCAN, k-means, and Hierarchical Clustering.*

[15] UCI Machine Learning Repository. (2024). *Comparative studies of DBSCAN and k-means clustering.*