**CLUSTER MOD – 4**

**1. What is Clustering?**

* Clustering is an **unsupervised learning** technique.
* It groups **unlabelled data** into clusters based on similarity.
* Similar objects → same cluster,
* Dissimilar objects → different clusters.

**Why unsupervised?**

* No output labels are present.
* We only try to find hidden patterns inside data.

**Important Concept**

* **Similarity measure** is the most important part in clustering.
* Common similarity measure → **Distance (Euclidean distance).**

**✅ 2. Difference: Classification vs Clustering**

| **Classification** | **Clustering** |
| --- | --- |
| Supervised learning | Unsupervised learning |
| Uses labelled data | Uses unlabelled data |
| Predict predefined classes | Find hidden groups in data |

**✅ 3. Types of Clustering**

**A. Hard (Exclusive) Clustering**

* A data point belongs to **only one** cluster.
* Example: **K-Means, K-Medoids**

**B. Soft (Fuzzy) Clustering**

* A data point can belong to **multiple clusters with probability**.
* Example: **Fuzzy C-Means**

**✅ 4. Major Clustering Methods**

**1. Partitioning / Centroid-based**

* Divides data into **k non-overlapping clusters**.
* Example: **K-Means, K-Medoids**

**2. Hierarchical Clustering**

* Creates a **tree-like structure (dendrogram)**.
* Two types:
  + Agglomerative (bottom-up)
  + Divisive (top-down)

**3. Density-Based Clustering**

* Forms clusters based on **dense regions**.
* Can discover **arbitrary shapes**.
* Good for **noise and outliers**.
* Example: **DBSCAN, OPTICS**

**4. Distribution Model-Based**

* Assumes data follows a probability distribution (often **Gaussian**).
* Example: **GMM (Gaussian Mixture Model)**

**5. Fuzzy / Overlapping Clustering**

* Points have **membership scores** for each cluster.
* Example: **Fuzzy C-Means**

**✅ 5. K-MEANS CLUSTERING (Very Important for Exam)**

**Definition**

K-means is an **unsupervised**, **partition-based** clustering algorithm that groups data into **K clusters**.

**Steps in K-Means (Algorithm)**

1. **Choose K** (number of clusters).
2. **Randomly select K initial centroids**.
3. Assign each data point to the **nearest centroid**.
4. Recalculate **new centroids** for each cluster.
5. Reassign points to the **closest centroid** again.
6. Repeat steps 4–5 until:
   * Centroids no longer change
   * OR cluster assignments don’t change
   * OR SSE does not decrease much
7. Output final clusters.

**Sum of Squared Error (SSE)**

Used to check **quality of clusters**:

Where

* = cluster j
* = centroid
* = Euclidean distance

**Lower SSE → Better clustering**

**✅ 6. Strengths of K-Means**

* Simple & easy to implement
* Fast and efficient for large datasets
* Works well when clusters are **spherical and separated**

**❌ 7. Limitations of K-Means**

* User must choose **K** in advance.
* Sensitive to **outliers**.
* Random initialization → different results each run.
* Not good for **non-spherical clusters** (like moon-shape, curved shapes).
* Performs poorly when clusters vary in **size or density**.



**K-MEANS NUMERICAL EXPLAINED (From Your Image)**

We have **6 data points**:

| **Point** | **x** | **y** |
| --- | --- | --- |
| P1 | 1 | 2 |
| P2 | 1 | 4 |
| P3 | 5 | 8 |
| P4 | 8 | 8 |
| P5 | 7 | 5 |
| P6 | 9 | 6 |

We want to make **k = 2 clusters**.

**⭐ Step 1: Choose K**

k = 2 clusters.

**⭐ Step 2: Choose Initial Centroids Randomly**

Selected from data:

* **Centroid C1 = P1 = (1, 2)**
* **Centroid C2 = P4 = (8, 8)**

**⭐ Step 3: Assign Each Point to Nearest Centroid**

Use Euclidean Distance:

Now calculate:

**Distances from C1 (1,2) and C2 (8,8)**

| **Point** | **Dist to C1** | **Dist to C2** | **Nearest** |
| --- | --- | --- | --- |
| P1(1,2) | 0 | 9.22 | C1 |
| P2(1,4) | 2 | 7.81 | C1 |
| P3(5,8) | 7.21 | 3 | C2 |
| P4(8,8) | 9.22 | 0 | C2 |
| P5(7,5) | 7.21 | 3.16 | C2 |
| P6(9,6) | 9.22 | 2.83 | C2 |

**Cluster Assignment (Iteration 1)**

* **Cluster C1** → {P1, P2}
* **Cluster C2** → {P3, P4, P5, P6}

**⭐ Step 4: Recalculate New Centroids**

**New Centroid C1**

Mean of P1(1,2) & P2(1,4):

**New Centroid C2**

Mean of P3(5,8), P4(8,8), P5(7,5), P6(9,6):

**⭐ Step 5: Reassign Using New Centroids (Iteration 2)**

| **Point** | **Dist to C1 (1,3)** | **Dist to C2 (7.25,6.75)** | **Assigned** |
| --- | --- | --- | --- |
| P1 | 1 | 7.81 | C1 |
| P2 | 1 | 6.57 | C1 |
| P3 | 5.83 | 2.5 | C2 |
| P4 | 7.62 | 1.25 | C2 |
| P5 | 6.4 | 1.03 | C2 |
| P6 | 8.06 | 1.8 | C2 |

Assignments **do not change**.

**⭐ Step 6: Convergence**

Centroids remain same:

* **C1 = (1,3)**
* **C2 = (7.25, 6.75)**

Thus algorithm stops.

**🎉 Final Answer (Clusters)**

**Cluster 1**

📌 C1 = (1, 3)  
Members → **P1, P2**

**Cluster 2**

📌 C2 = (7.25, 6.75)  
Members → **P3, P4, P5, P6**

This is exactly what K-Means does.



**💡 K-Means ka rule:**

**“Jab tak centroid change hota rahe → algorithm chalta rahe.”**

**Thus process:**

**1️⃣ Assign points to nearest centroid**

**2️⃣ Recalculate centroid (mean)**

**3️⃣ Agar centroid change hua →**

**👉 Go back to Step 1**

**4️⃣ Agar centroid same ho jaye →**

**👉 STOP (algorithm converged)**

**✅ DIFFERENCE BETWEEN K-MEANS AND K-NN (Exam Important)**

| **Feature** | **K-Means** | **K-NN** |
| --- | --- | --- |
| Type of Algorithm | **Unsupervised** | **Supervised** |
| Purpose | Clustering (grouping) | Classification / Regression |
| Input Data | **Unlabelled** | **Labelled** |
| Output | Cluster number (C1, C2…) | Class label (A, B, C…) or value |
| What is K? | Number of clusters | Number of nearest neighbors |
| Model Type | Centroid-based | Instance-based (lazy learner) |
| Training | Finds centroids iteratively | No training (only stores data) |
| Computation | Many iterations | Distance calculated per prediction |
| Use Case | Customer segmentation, grouping | Spam detection, classification |

**🔷 NUMERICAL-0 (1,2,3,4,5 – given solution wale screenshots)**

Distance matrix:

|  | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- |
| **1** | 0 | 9 | 7 | 5 | 9 |
| **2** | 9 | 0 | 3 | 6 | 7 |
| **3** | 7 | 3 | 0 | 9 | 8 |
| **4** | 5 | 6 | 9 | 0 | 2 |
| **5** | 9 | 7 | 8 | 2 | 0 |

Goal: **Agglomerative hierarchical clustering** using

1️⃣ **Complete linkage**  
2️⃣ **Average linkage**

**1️⃣ COMPLETE LINKAGE (Max distance)**

Distance between two clusters = **maximum** distance between any pair of points (one from each cluster).

Initially: clusters = {1},{2},{3},{4},{5}

**🔹 Step 1 – Smallest distance**

Check non-zero distances:

* d(4,5) = **2** ✅ (smallest)
* d(2,3) = 3
* others bigger

So **merge 4 and 5** at height **2**

Cluster: **{4,5}**

Now naya distance of {4,5} to 1,2,3 – **max** rule:

* D({4,5},1) = max(d41, d51) = max(5,9) = **9**
* D({4,5},2) = max(d42, d52) = max(6,7) = **7**
* D({4,5},3) = max(d43, d53) = max(9,8) = **9**

Updated matrix:

|  | **1** | **2** | **3** | **{4,5}** |
| --- | --- | --- | --- | --- |
| 1 | 0 | 9 | 7 | 9 |
| 2 | 9 | 0 | 3 | 7 |
| 3 | 7 | 3 | 0 | 9 |
| {4,5} | 9 | 7 | 9 | 0 |

**🔹 Step 2 – Next smallest distance**

From matrix, smallest non-zero = **3** between **2 and 3**

→ Merge **{2,3}** at height **3**

Now clusters: {1}, {2,3}, {4,5}

Distances (complete):

* D({2,3},1) = max(d21,d31) = max(9,7) = **9**
* D({2,3},{4,5}) = max(6,7,9,8) = **9**

New matrix:

|  | **1** | **{2,3}** | **{4,5}** |
| --- | --- | --- | --- |
| 1 | 0 | 9 | 9 |
| {2,3} | 9 | 0 | 9 |
| {4,5} | 9 | 9 | 0 |

**🔹 Step 3 – Remaining merges**

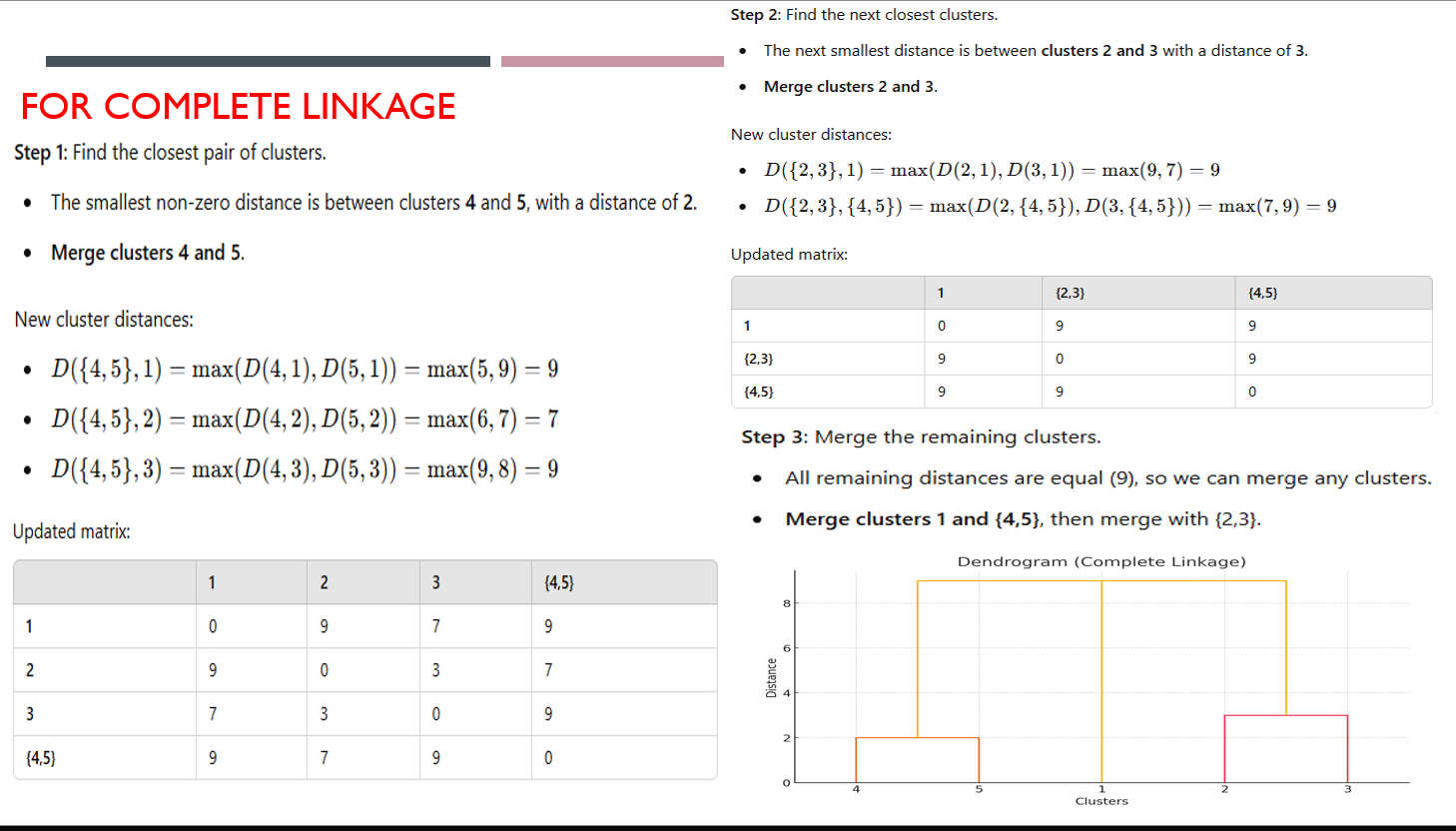
Sab non-zero distances = **9**, koi bhi pair merge kar sakte:

Common choice:

* Merge **1** and **{4,5}** at height **9**
* Then merge **{1,4,5}** with **{2,3}** at height **9**

👉 **Dendrogram (Complete linkage)**

* 4 & 5 join at **2**
* 2 & 3 join at **3**
* All clusters join together at **9**



**2️⃣ AVERAGE LINKAGE**

Distance between two clusters = **average** of all pair distances (one point from each cluster).

Start again with singleton clusters.

**🔹 Step 1 – Merge 4 & 5**

Smallest distance again: d(4,5)= **2**  
→ Merge {4,5} at height 2.

Distances of {4,5}:

* D({4,5},1) = (d41+d51)/2 = (5+9)/2 = **7**
* D({4,5},2) = (6+7)/2 = **6.5**
* D({4,5},3) = (9+8)/2 = **8.5**

Updated matrix:

|  | **1** | **2** | **3** | **{4,5}** |
| --- | --- | --- | --- | --- |
| 1 | 0 | 9 | 7 | 7 |
| 2 | 9 | 0 | 3 | 6.5 |
| 3 | 7 | 3 | 0 | 8.5 |
| {4,5} | 7 | 6.5 | 8.5 | 0 |

**🔹 Step 2 – Merge 2 & 3**

Smallest non-zero = **3** between 2 and 3  
→ Merge {2,3} at height 3.

Distances:

* D({2,3},1) = (9+7)/2 = **8**
* D({2,3},{4,5}) = average(6,7,9,8)  
  = (6+7+9+8)/4 = **7.5**

Updated matrix:

|  | **1** | **{2,3}** | **{4,5}** |
| --- | --- | --- | --- |
| 1 | 0 | 8 | 7 |
| {2,3} | 8 | 0 | 7.5 |
| {4,5} | 7 | 7.5 | 0 |

**🔹 Step 3 – Merge 1 & {4,5}**

Smallest distance now = **7** between 1 and {4,5}

→ Merge {1,4,5} at height 7.

Finally, distance between {1,4,5} and {2,3}:

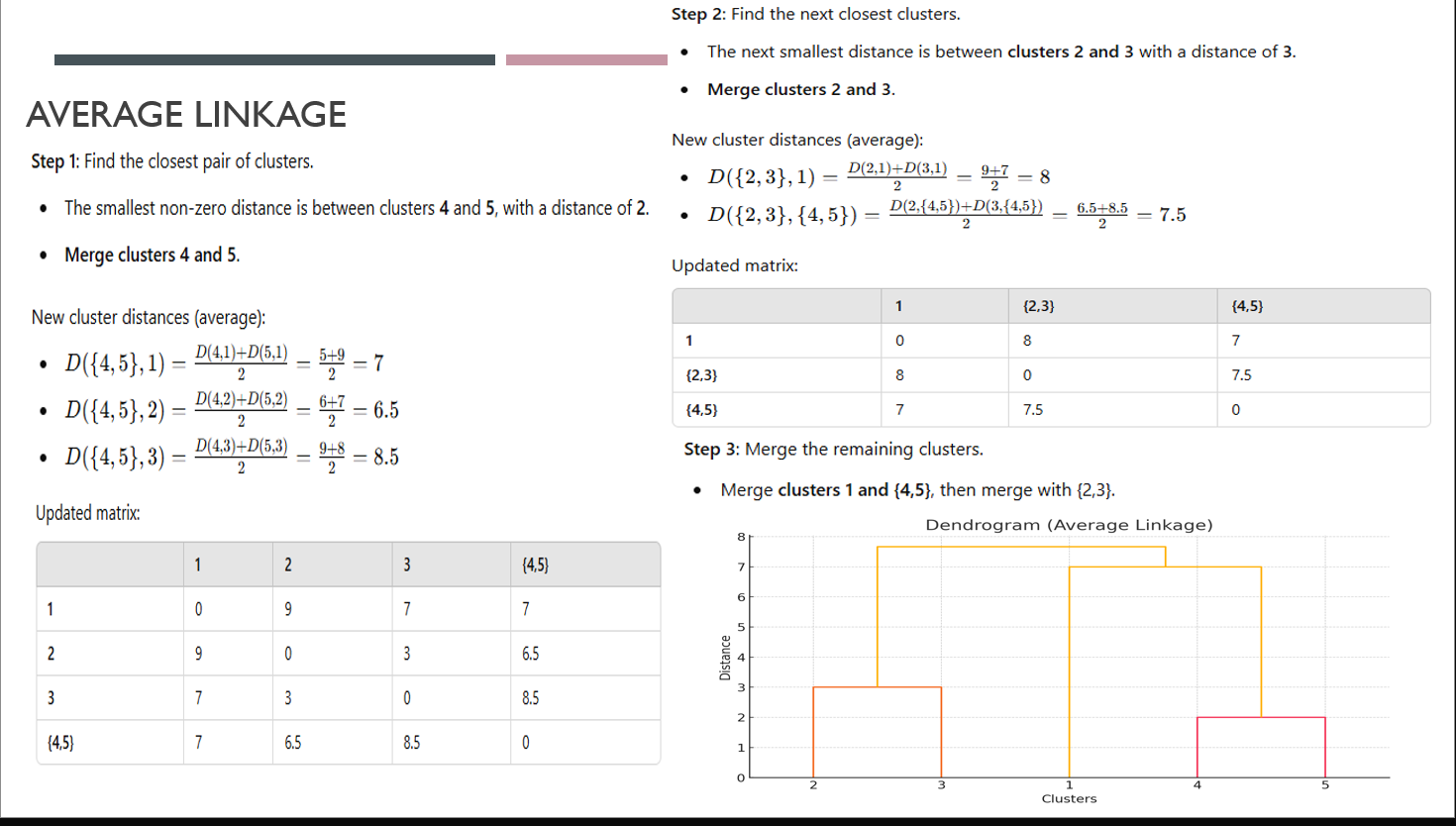
Pairs: (1,2)=9, (1,3)=7, (4,2)=6, (4,3)=9, (5,2)=7, (5,3)=8  
Average = (9+7+6+9+7+8)/6 = **46/6 ≈ 7.67**

So last merge at ≈ **7.7**.

👉 **Dendrogram (Average linkage)**

* 4 & 5 join at **2**
* 2 & 3 join at **3**
* 1 joins {4,5} at **7**
* {1,4,5} joins {2,3} at **≈7.7**

That’s your first numerical fully solved.



**🔷 NUMERICAL-1 (Complete Linkage – A,B,C,D,E)**

Distance matrix:

|  | **A** | **B** | **C** | **D** | **E** |
| --- | --- | --- | --- | --- | --- |
| A | 0 | 4 | 6 | 9 | 10 |
| B | 4 | 0 | 5 | 7 | 8 |
| C | 6 | 5 | 0 | 3 | 4 |
| D | 9 | 7 | 3 | 0 | 2 |
| E | 10 | 8 | 4 | 2 | 0 |

**Step 1 – Find smallest distance**

Non-zero distances: 4,6,9,10,5,7,8,3,4,2  
**Smallest = 2** between **D and E**

→ Merge **{D,E}** at height **2**

Distances from {D,E} (complete = max):

* To A: max(AD,AE) = max(9,10) = **10**
* To B: max(BD,BE) = max(7,8) = **8**
* To C: max(CD,CE) = max(3,4) = **4**

Matrix:

|  | **A** | **B** | **C** | **{DE}** |
| --- | --- | --- | --- | --- |
| A | 0 | 4 | 6 | 10 |
| B | 4 | 0 | 5 | 8 |
| C | 6 | 5 | 0 | 4 |
| {DE} | 10 | 8 | 4 | 0 |

**Step 2 – Next smallest distance**

Non-zero distances: 4(A,B),6,10,4(C,{DE}),5,8,…  
Smallest value = **4**, tie between **A–B** and **C–{DE}**.  
Usually hum ek tie ko randomly choose kar sakte. Let’s:

→ Merge **A and B** at height **4**, cluster {A,B}.

Distances from {A,B}:

* To C: max(AC,BC)=max(6,5)=**6**
* To {DE}: max(AD,AE,BD,BE)=max(9,10,7,8)=**10**

Matrix:

|  | **{AB}** | **C** | **{DE}** |
| --- | --- | --- | --- |
| {AB} | 0 | 6 | 10 |
| C | 6 | 0 | 4 |
| {DE} | 10 | 4 | 0 |

**Step 3 – Next merge**

Smallest non-zero distance now = **4** between **C and {DE}**

→ Merge **{C,DE}** at height **4**

Clusters: {AB}, {CDE}

Distance between them (complete):

max of all pairs (A/B with C/D/E):

* A–C=6, A–D=9, A–E=10
* B–C=5, B–D=7, B–E=8

Maximum = **10**

→ Final merge {AB} and {CDE} at **10**

👉 **Dendrogram:**

* D & E merge at **2**
* A & B merge at **4**
* C merges with {D,E} at **4**
* {AB} and {CDE} merge at **10**

**🔷 NUMERICAL-2 (Average Linkage – A,B,C,D)**

Distance matrix:

|  | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| A | 0 | 2 | 6 | 10 |
| B | 2 | 0 | 5 | 9 |
| C | 6 | 5 | 0 | 4 |
| D | 10 | 9 | 4 | 0 |

**Step 1 – Smallest distance**

Smallest non-zero = **2** between **A and B**

→ Merge **{A,B}** at height **2**

Average distances from {A,B}:

* To C: (AC+BC)/2 = (6+5)/2 = **5.5**
* To D: (AD+BD)/2 = (10+9)/2 = **9.5**

Matrix:

|  | **{AB}** | **C** | **D** |
| --- | --- | --- | --- |
| {AB} | 0 | 5.5 | 9.5 |
| C | 5.5 | 0 | 4 |
| D | 9.5 | 4 | 0 |

**Step 2 – Next merge**

Smallest non-zero = **4** between **C and D**

→ Merge **{C,D}** at height **4**

Now only two clusters: {AB} and {CD}

Distance (average) between {AB} and {CD}:

Pairs: A–C=6, A–D=10, B–C=5, B–D=9  
Average = (6+10+5+9) / 4 = **30/4 = 7.5**

→ Final merge at **7.5**

👉 **Dendrogram:**

* A & B at **2**
* C & D at **4**
* {AB} & {CD} at **7.5**

**✅ What to write in exam (short pattern)**

For each such question, likho:

1. **Initial distance matrix**
2. **Step-wise merges** (with distance value, and which linkage you used – max / avg)
3. **Updated matrices** after each merge
4. **Heights** at which merges happen → uske basis pe **dendrogram draw** karo.

**1-PAGE SHEET: CLUSTERING NUMERICALS (Formula + Steps)**

*(All algorithms: K-Means, Hierarchical, Linkage, Elbow Method)*

**✅ 1. K-MEANS CLUSTERING — NUMERICAL STEPS**

**Formula**

**Euclidean Distance**

**Centroid (mean):**

**SSE (Sum of Squared Errors):**

**✅ Steps for any K-Means Numerical**

1. **Choose K (given)**
2. **Select K initial centroids** (random or given)
3. **Compute distances** of each point to each centroid  
   using Euclidean distance
4. **Assign** each point to nearest centroid → form clusters
5. **Recalculate new centroid** = mean of points in each cluster
6. **Recompute distances** to new centroids
7. **Reassign points** to nearest new centroid
8. **Repeat steps 5–7 until:**
   * Centroids stop changing OR
   * Cluster assignment stable OR
   * SSE stops decreasing

**Convergence ⇒ Final clusters**

**✅ 2. ELBOW METHOD (To find K)**

**Formula: WSS (Within Sum of Squares)**

**Steps**

1. Run K-means for K = 1, 2, 3, … n
2. Calculate WSS for each K
3. Plot: K vs. WSS
4. Identify **“bend”** (elbow point)
5. That **K** = optimal number of clusters

**🔵 3. HIERARCHICAL CLUSTERING (AGGLOMERATIVE)**

**Bottom-up** method: Start with each point as a separate cluster.

**Steps**

1. Start with **n** clusters (each point = 1 cluster)
2. Compute distance matrix
3. **Find smallest distance** → merge those 2 clusters
4. **Update** distance matrix using chosen linkage method
5. Repeat steps 3–4 until only **1 cluster** remains
6. Draw **dendrogram**
   * y-axis = distance at which merge happens

**🟣 4. HIERARCHICAL CLUSTERING (DIVISIVE)**

**Top-down** method.

**Steps**

1. Start with **one cluster containing all points**
2. Select cluster with **largest internal distance**
3. **Split** into two subclusters
4. Compute distances and repeat splitting
5. Continue until each point is a single cluster
6. Draw dendrogram (top → down)

**6. HOW TO DRAW A DENDROGRAM (Exam Method)**

1. Write points on x-axis
2. For each merge, draw a **horizontal line** at merge height
3. Connect two clusters with a **vertical line**
4. Height = distance at which merged
5. Continue until all clusters merge at the top

**🟩 7. HOW TO SOLVE ANY LINKAGE NUMERICAL**

**Steps**

1. Note the **distance matrix**
2. Find **smallest non-zero** distance → merge
3. Compute new cluster distance using:

| **Linkage** | **Formula** |
| --- | --- |
| Single | min(d) |
| Complete | max(d) |
| Average | average(d) |
| Centroid | centroid distance |
| Cluster centroid = mean of points |  |
| → Merge clusters that **minimally increase** SSE → Produces best compact clusters |  |

1. 4 Update matrix
2. Repeat until all clusters merge
3. Draw dendrogram

**⭐ 1. Density-Based Clustering Algorithms (DBSCAN)**

**✔ What is Density-Based Clustering?**

* It is an **unsupervised learning method**.
* It groups data points based on **density** (how closely points are packed).
* Useful when clusters are **irregular, arbitrary-shaped** (not circular like K-means).
* It can **detect outliers/noise**, which K-means cannot do.

**✔ How does DBSCAN define a cluster?**

A cluster = **dense region** separated by **low-density region**.

In density-based logic:

* If many points are close → cluster
* If points are isolated → noise/outlier

**✔ Two important parameters:**

**1. Epsilon (ε)**

* Radius around a point.
* Maximum distance to consider another point a “neighbor”.

**2. minPts**

* Minimum number of points required inside ε-radius
* To form a dense region (i.e., a cluster)

**⭐ Categories of points in DBSCAN**

**1. Core Point**

A point is *core* if:

(Include itself in counting)

**2. Border Point**

* Not a core point
* But lies inside ε-radius of a core point

**3. Noise**

* Neither core nor border
* Does not belong to any cluster

**⭐ 2. HOW DBSCAN WORKS? (Slide Explanation)**

**✔ Core point**

If a point has **minPts or more** points in neighborhood → Core

**✔ Border point**

Fewer than minPts  
BUT  
inside ε of a **core** → Border

**✔ Noise point**

Not a core  
Not a border  
→ Noise / Outlier

**⭐ 3. DBSCAN Algorithm Steps (Slide Explanation)**

**Step 1: Select an unvisited point**

* Choose any unvisited point & mark as visited

**Step 2: Check density**

* Find all points within ε
* If neighbors ≥ minPts → point is Core → new cluster starts
* Else → temporary Noise (can later become Border)

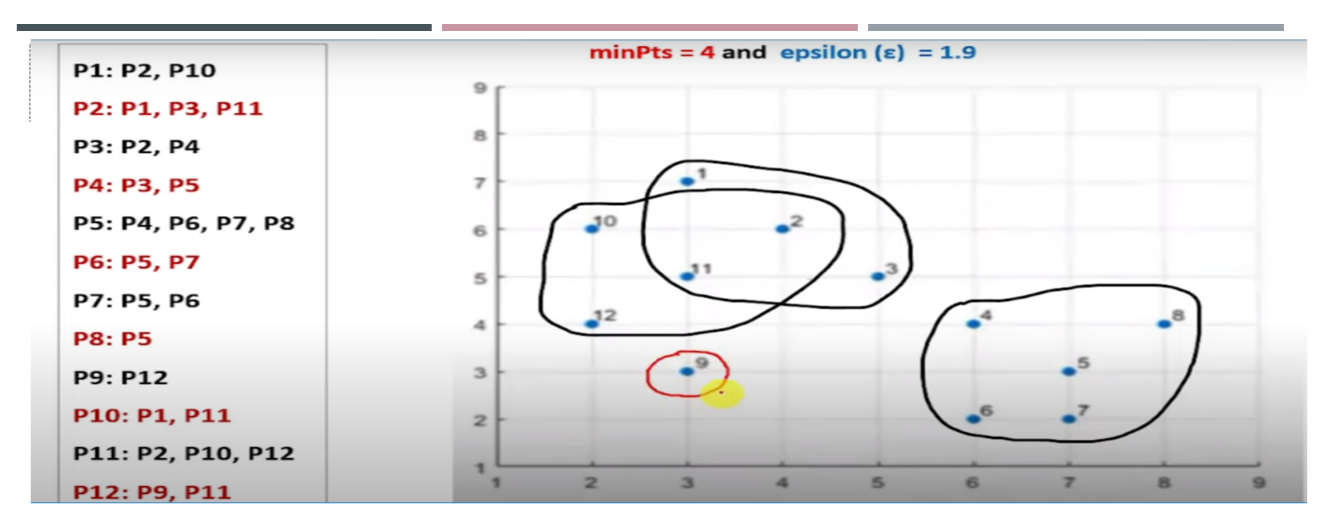
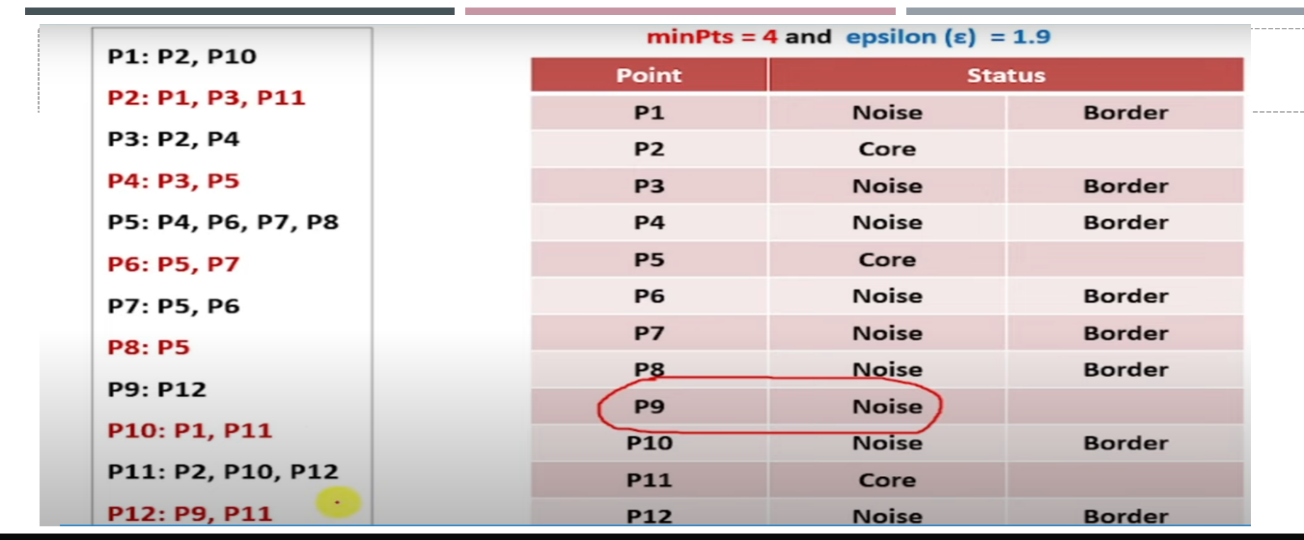
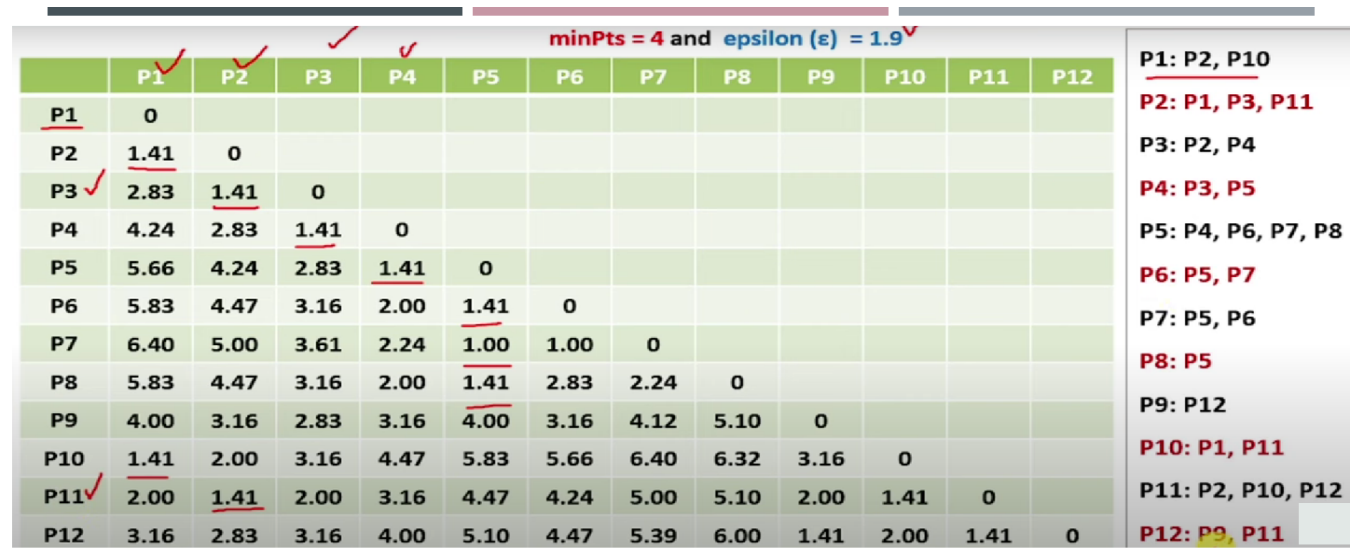
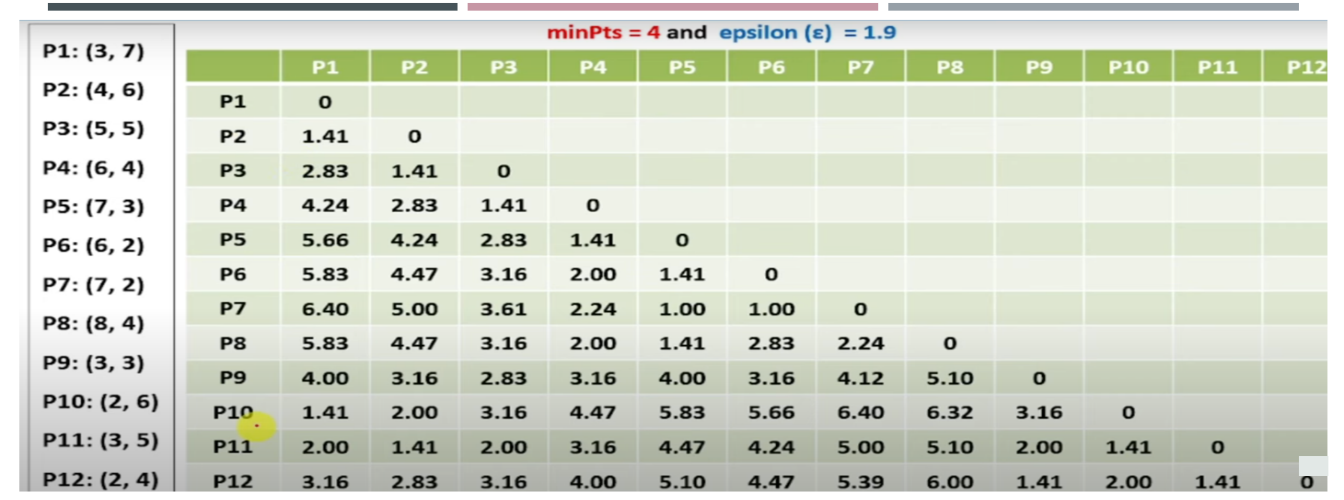
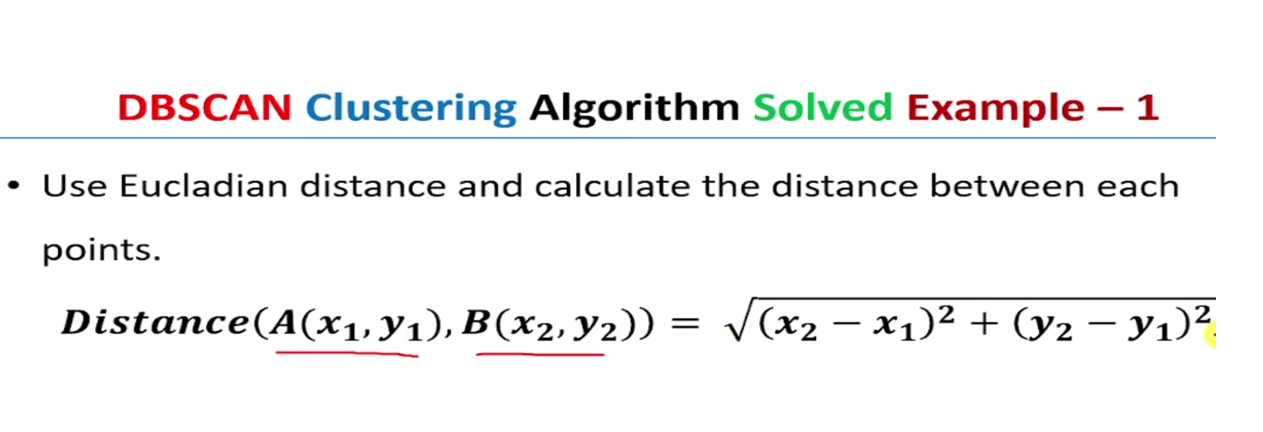
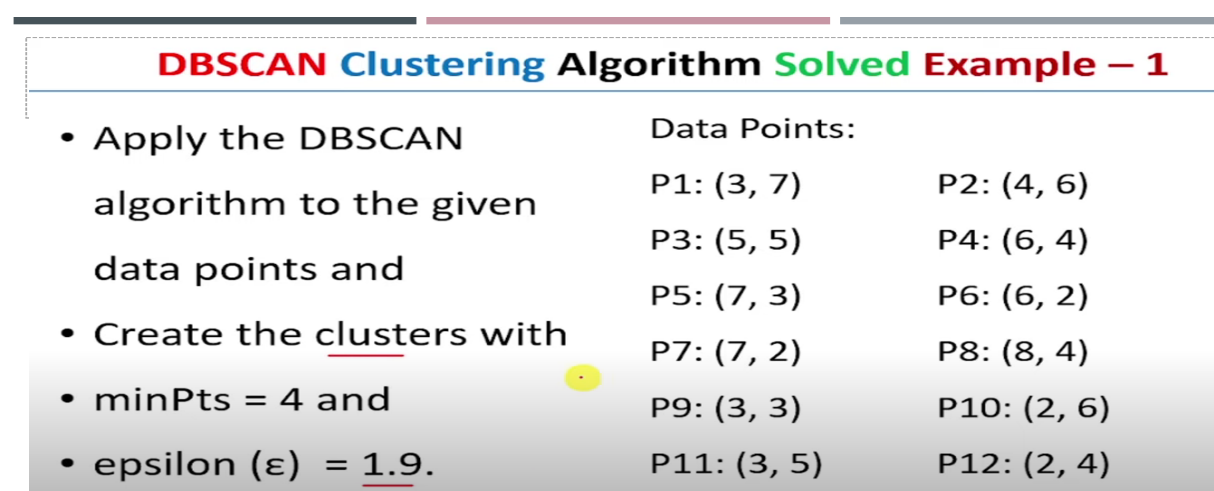
**Step 3: Expand cluster**

For every point in cluster:

* If it is Core → add all its neighbors
* Repeat until no new points can be added

**Step 4: Repeat**

* Continue with next unvisited point
* Stop when all points processed



**⭐ CLUSTER QUALITY (Slide Explanation)**

Clustering me hum groups banate hain, but **kya clustering sahi bani?**  
Isko check karne ke liye **Cluster Quality** measure karte hain.

**📌 Why needed?**

* Clustering me labels nahi hote → “unsupervised”.
* But hume dekhna hota hai ki algorithm ne sahi group banaye ya nahi.

**📌 2 Important Measures:**

1. **Rand Index** → External measure (needs ground truth)
2. **Silhouette Coefficient** → Internal measure (does not need ground truth)

**⭐ RAND INDEX (R INDEX)**

(External Cluster Evaluation)

Rand Index compare karta hai:

👉 **True labels**  
vs  
👉 **Predicted (algorithm) clusters**

**Rand Index kya check karta hai?**

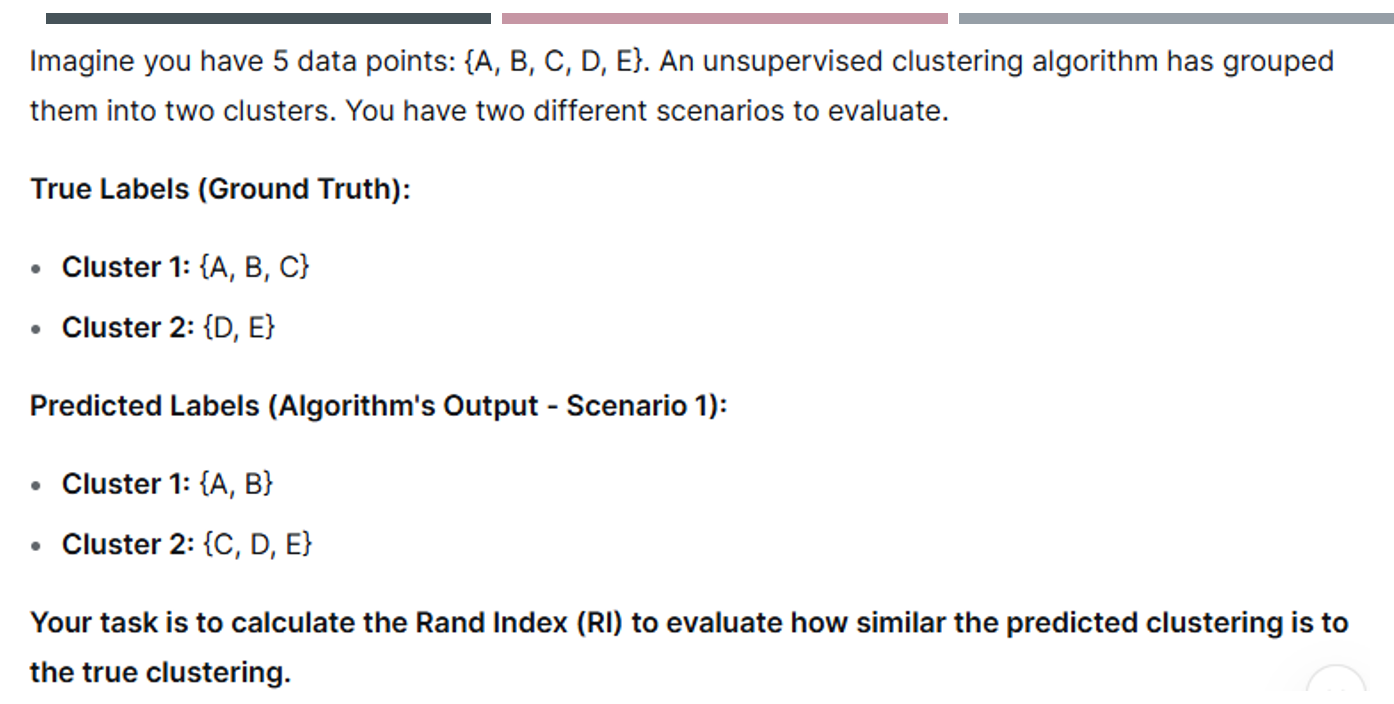
Har possible **pair of points** dekhta hai:

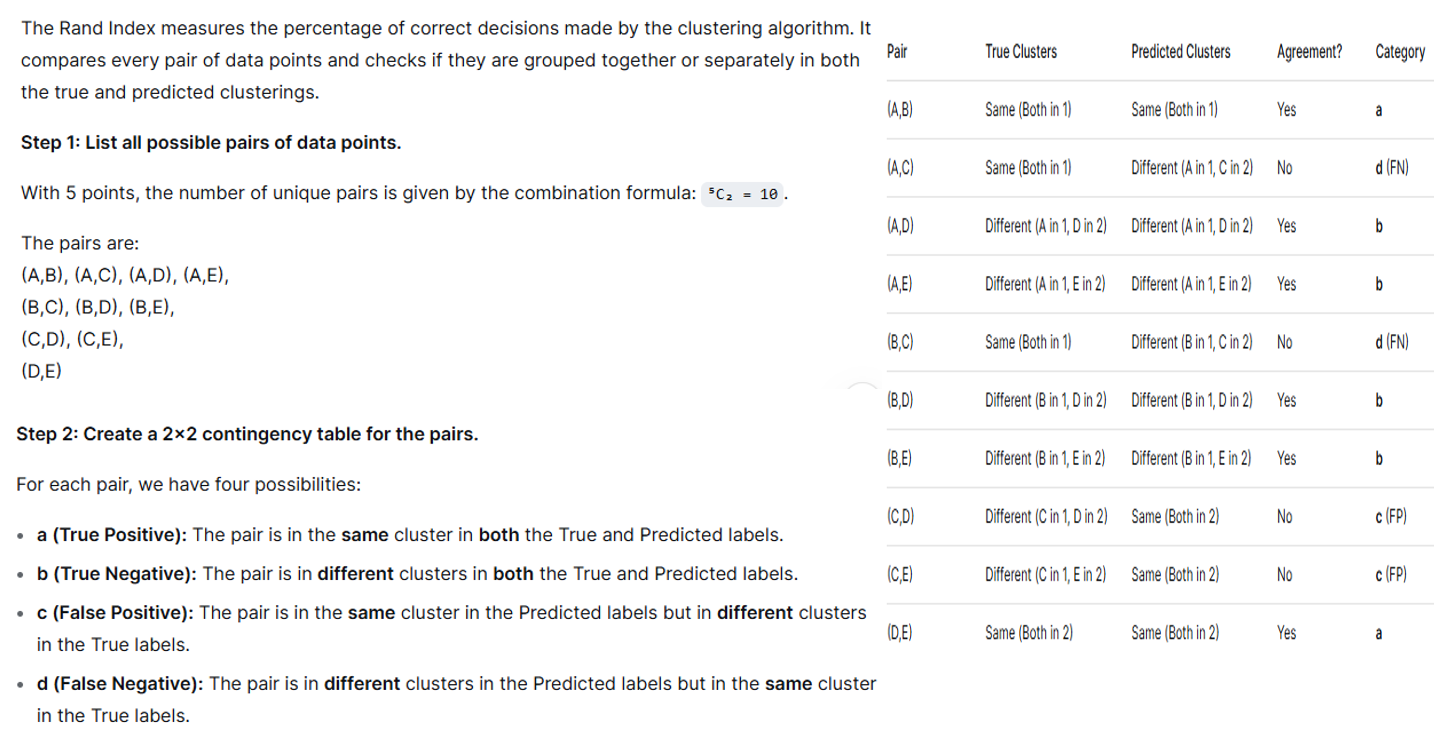
| **TRUE** | **PREDICTED** | **Meaning** |
| --- | --- | --- |
| Same | Same | ✔ Correct (TP) |
| Diff | Diff | ✔ Correct (TN) |
| Same | Diff | ❌ Wrong (FN) |
| Diff | Same | ❌ Wrong (FP) |

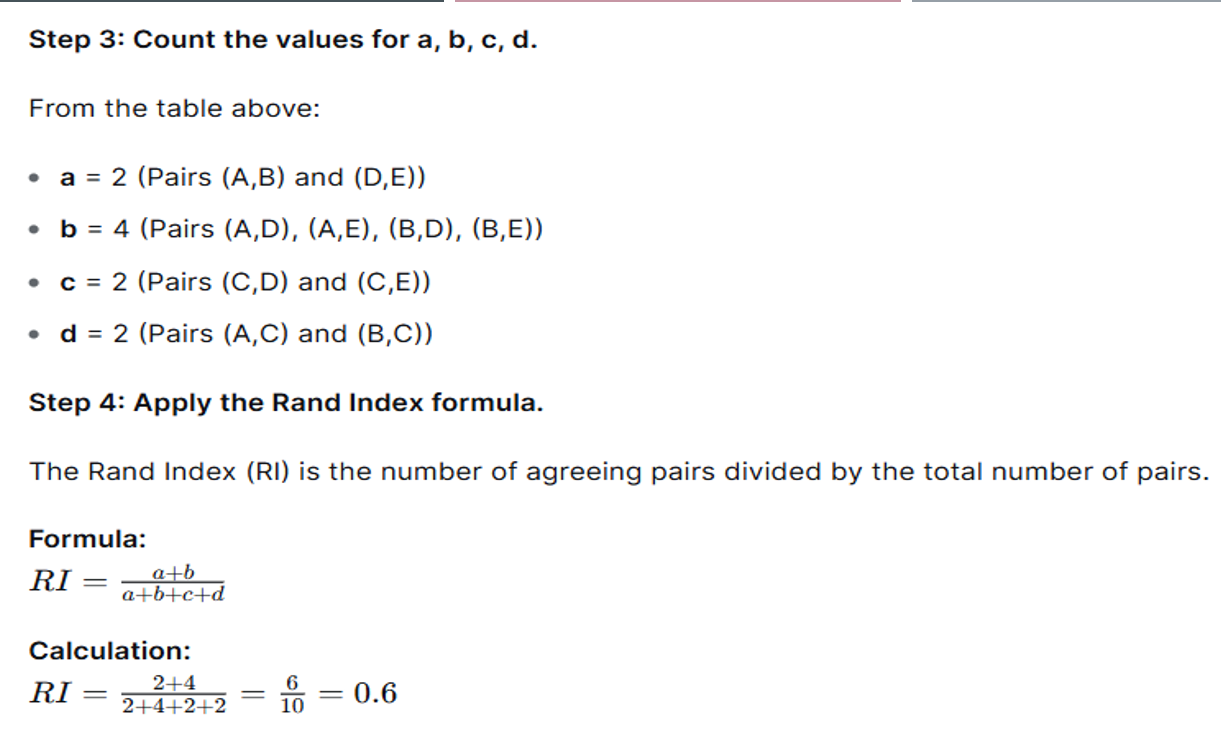
**Rand Index Formula:**

Range: **0 to 1**

* 1 = perfect match
* 0 = totally wrong







**STEP-2: Categorise each pair (TP, TN, FP, FN)**

**Legend:**

* **a** = TP (same in true & predicted)
* **b** = TN (diff in true & predicted)
* **c** = FP (same in predicted but diff in true)
* **d** = FN (diff in predicted but same in true)

**⭐ LIMITATIONS OF RAND INDEX**

Exam important:

* Needs **ground truth labels** → always available nahi hota.
* Sensitive to **imbalanced clusters**.
* Only pairwise check karta → **cluster structure** ka idea nahi deta.
* Non-convex shapes ko consider nahi karta.
* Interpretation thoda tricky.

**⭐ SILHOUETTE COEFFICIENT**

(Internal Cluster Evaluation – no labels needed)

Silhouette check karta hai:

**1️⃣ a(i): Intra-cluster distance**

Point i apne cluster me kitna close hai?

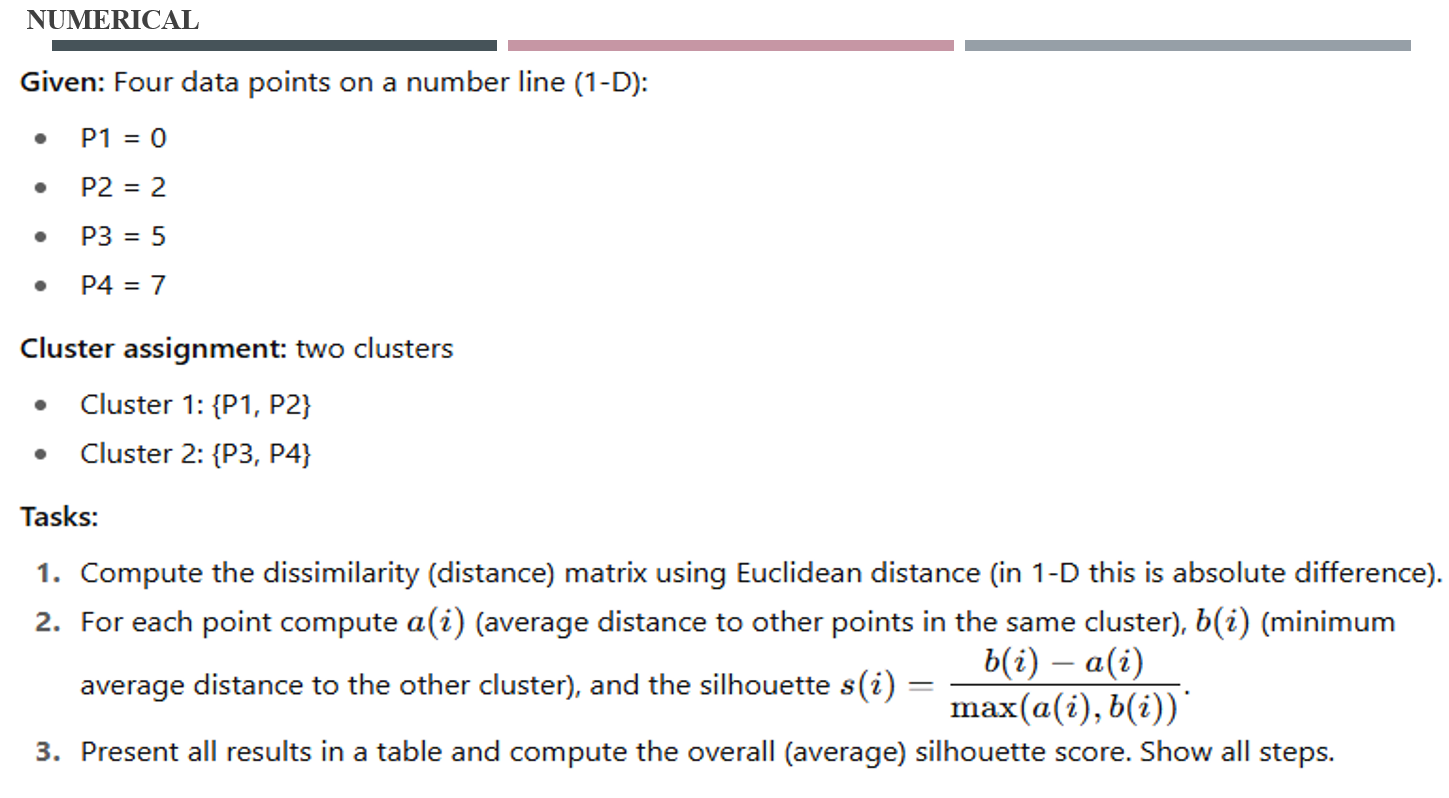
**2️⃣ b(i): Inter-cluster distance**

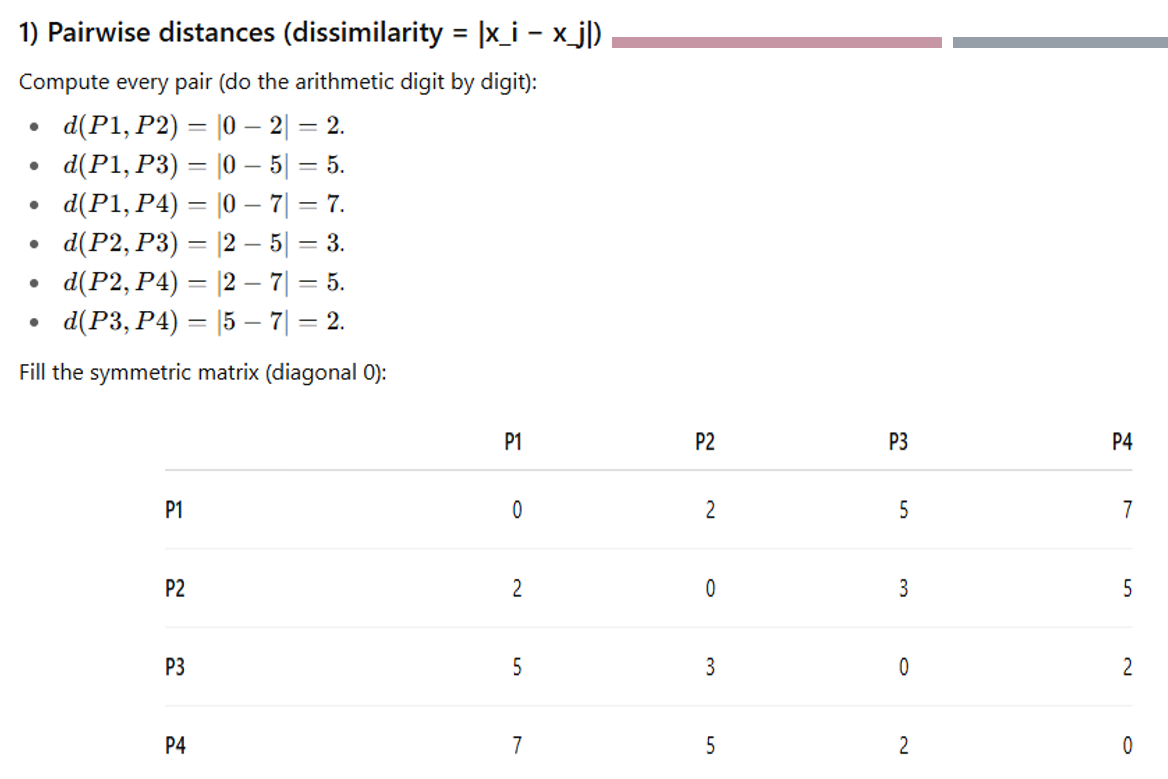
Dusre clusters se kitna door hai?

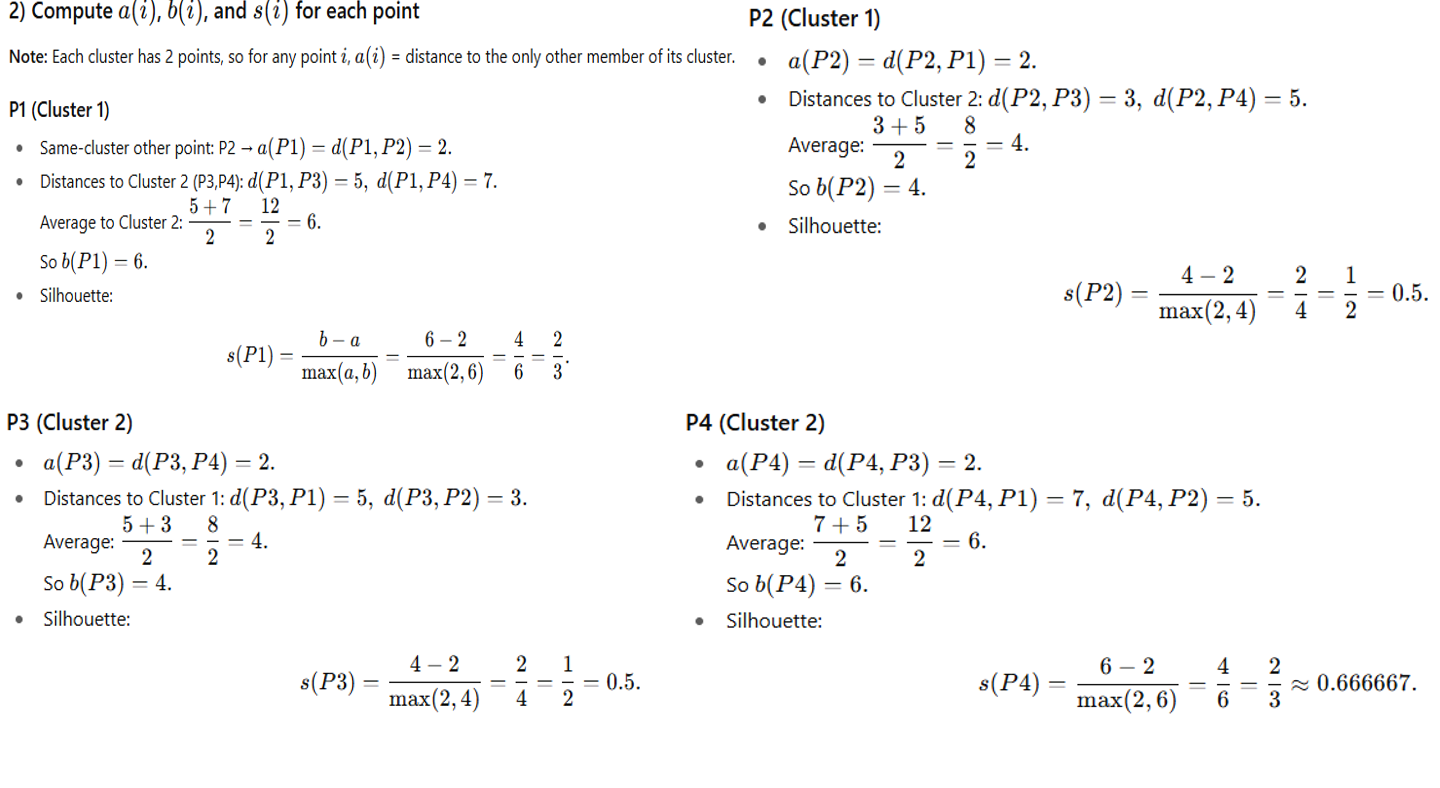
**Formula:**

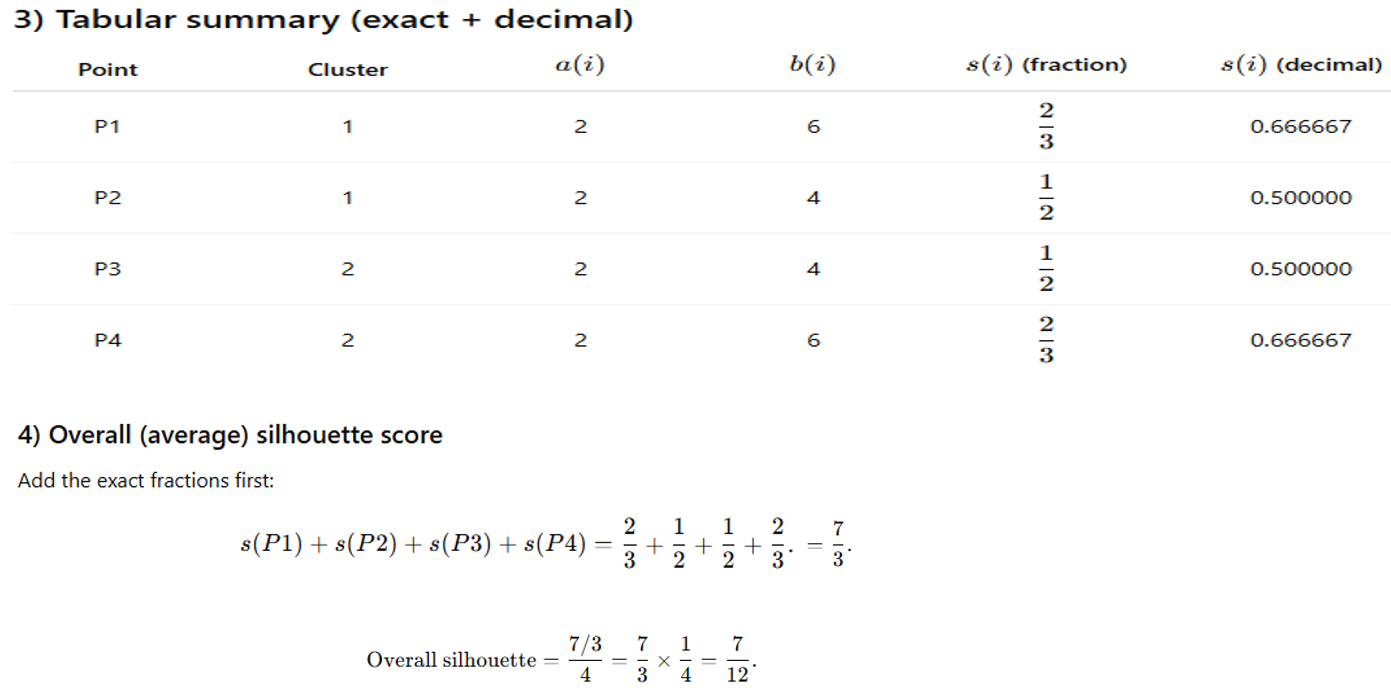
**Range:**

* +1 → Perfectly assigned cluster
* 0 → Overlapping clusters
* Negative → Wrongly assigned









**⭐ COMPARISON (IMPORTANT SLIDE)**

| **Feature** | **Rand Index** | **Silhouette** |
| --- | --- | --- |
| Needs Labels? | Yes | No |
| Type | External | Internal |
| Checks | Pairwise correctness | Cohesion + Separation |
| Range | 0 to 1 | -1 to 1 |
| Use Case | Supervised evaluation | Unsupervised evaluation |
| Weakness | Needs ground truth | Slow for large data |

**⭐ EXAM QUICK REVISION NOTES**

**Rand Index:**

* Compares predicted vs true clusters
* TP, TN, FP, FN
* RI = (TP+TN)/Total pairs
* Range: 0–1

**Silhouette:**

* No labels needed
* a(i) = intra distance
* b(i) = nearest cluster distance
* s(i) = (b−a)/max(a,b)
* Range −1 to +1

**1. Why Dimensionality Reduction? (Very Important)**

When a dataset has **too many features**, these problems occur:

**🔹 Problems:**

1. **Slow computation**
2. **Overfitting** (model memorizes noise)
3. **Hard to visualize** (only 2D or 3D possible)
4. **Curse of dimensionality**
   * Irrelevant features
   * Redundant features

**🔹 Solution:**

➡ **Dimensionality Reduction = Reduce number of features while keeping important info.**

It **converts high-dimensional data → lower dimensional** (like 3D → 2D).

**⭐ 2. How Dimensionality Reduction Works?**

Example from slides:

You have 3D data: X, Y, Z.

Most variation is in **X-Y direction**, Z adds **no useful information**.

So PCA removes Z → gives 2D data.

**📌 Key idea:**

* Keep features that contain **maximum information (variance)**
* Remove features that add **noise or redundancy**
* Make data easier to visualize & process

**⭐ 3. Dimensionality Reduction Techniques**

**✔ TWO MAIN TYPES:**

**🔷 A. Feature Selection (choose important original features)**

* Only select the best features
* Do NOT create new features

**Types:**

1. **Filter Methods**
   * Variance Threshold
   * Chi-square
   * Correlation
   * F-score
2. **Wrapper Methods**
   * Recursive Feature Elimination (RFE)
   * Forward Selection
   * Backward Elimination
3. **Embedded Methods**
   * Lasso Regression
   * Decision Trees / Random Forests

**🔷 B. Feature Extraction (create NEW features)**

Transforms old features into new ones.

**Types:**

**🔹 Linear methods**

* PCA
* LDA
* SVD

**🔹 Non-Linear methods**

* Kernel PCA
* t-SNE
* Isomap
* MDS

**⭐ 4. Principal Component Analysis (PCA)**

Most important for exams.

**PCA USE:**

* Reduce dimensions
* Remove noise
* Improve speed
* Visualize data
* Find most important directions of data

**⭐ 5. How PCA Works? (Exam Steps)**

Memorize these **4 steps**, examiner loves them ❤️

**🟩 STEP 1: Standardize the Data**

Make each feature have:

* Mean = 0
* Std deviation = 1

Formula:

**🟩 STEP 2: Compute Covariance Matrix**

Shows relation between features:

If covariance is:

* **High positive** → features increase together
* **High negative** → one increases, one decreases
* **Zero** → uncorrelated

**🟩 STEP 3: Compute Eigenvalues & Eigenvectors**

Solve:

This gives:

* Eigenvalues (importance)
* Eigenvectors (directions)

The **biggest eigenvalue = 1st principal component**  
Next = 2nd PC, etc.

**🟩 STEP 4: Select Top k Principal Components**

Usually choose components that capture **95% variance**.

**🟩 STEP 5: Project data onto New Components**

New data =

Where **V = eigenvectors matrix**

This gives reduced-dimension data.

**7. Advantages & Disadvantages**

**✔ Advantages:**

* Removes noise
* Faster computation
* Reduces storage
* Avoids overfitting
* Better generalization
* Handles multicollinearity
* Easy visualization

**❌ Disadvantages:**

* May lose important information
* PCA extra computation cost
* Hard to interpret new features
* Not good for nonlinear data (use t-SNE instead)
* Risk of removing useful features

**⭐ 8. NEED OF DIMENSIONALITY REDUCTION (6 points)**

1. Simplifies data
2. Reduces overfitting
3. Increases speed
4. Enables visualization
5. Removes noise & redundancy
6. Improves model accuracy
7. Saves storage

**⭐ 9. Applications of Dimensionality Reduction**

**📌 Real-world uses:**

1. Image Compression
2. Visualizing high-dimensional data
3. Genetics (10000+ genes -> 10 PCs)
4. NLP topic modelling
5. Finance (stock price analysis)
6. Recommendation systems
7. Speech signal cleaning
8. Medical imaging
9. IoT sensor compression
10. Fraud/anomaly detection

**⭐ 10. Complete Meaning of PCA in One Line (Exam Gold)**

➡ **PCA converts correlated features into uncorrelated components (principal components) that capture maximum variance.**