Logestic regression

📘 Logistic Regression – Conceptual Overview

All values in this example are assumed for understanding.

🔷 1. Input Dataset

x_1	x_2	y
3	2	0
1	4	1
6	8	0
4	7	1

- x_1, x_2: Input features
- *y*: Target class (binary: 0 or 1)

🔷 2. Linear Model + Sigmoid Conversion

We apply a linear equation on the input features:

$z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$

Then, apply the *sigmoid function*:

$\hat{y} = \frac{1}{1+e^{-z}}$

This gives predicted probabilities for each data point.

> Example (assumed values):

x_1	x_2	z (linear output)	Sigmoid Output ($\hat{y}$)	Predicted y
3	2	—	0.3	0
1	4	—	0.7	1
6	8	—	0.5	1 (≥0.5)
4	7	—	0.4	0 (<0.5)

🔹 3. Thresholding

To convert the probability into a class label:

- If $(\hat{y} \geq 0.5)$ → predict **1**
- If $(\hat{y} < 0.5)$ → predict **0**

🔹 4. Sigmoid Curve (S-shaped Graph)

- **X-axis**: Linear value `z` (can depend on x_1, x_2, etc.)
- **Y-axis**: Output of sigmoid (ŷ)
- The curve passes through (0, 0.5)
- Approaches 1 as `z → +∞`, and 0 as `z → −∞`

- You marked 0.3, 0.5, 0.6, 0.7 → correct interpretation
- At **z = 0**, sigmoid output = 0.5 (decision boundary)

> On your graph:

🔹 5. Final Concept Flow

Below is the end-to-end flow of how logistic regression works conceptually:

1. **Input Features**
   Data consists of multiple features (e.g., x_1, x_2) and a binary label `y`.

2. **Linear Combination**
   For each data point, compute a weighted sum:
   $z = w_1 x_1 + w_2 x_2 + b$

3. **Apply Sigmoid Activation**
   Convert the linear output `z` into a probability score:
   $\hat{y} = \frac{1}{1+e^{-z}}$

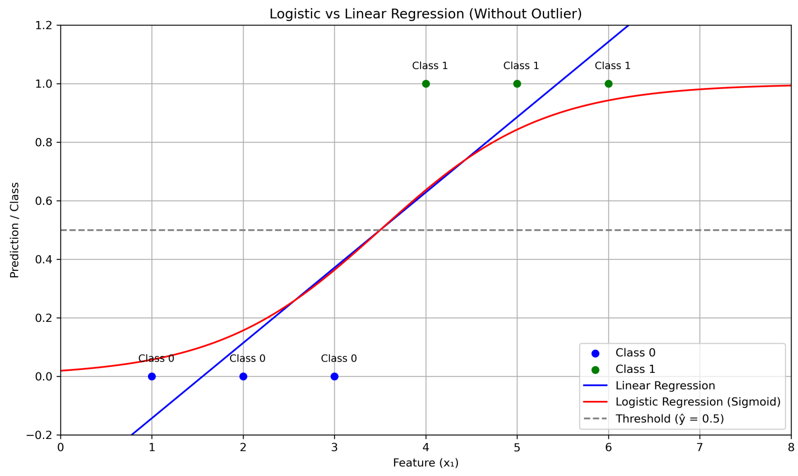
4. **Classification via Threshold**

   - If $(\hat{y} \geq 0.5)$: Predict class **1**
   - If $(\hat{y} < 0.5)$: Predict class **0**
5. **Model Training (Behind the Scenes)**

   - During training, the model adjusts weights `w₁, w₂, b` to minimize a **log-loss** function.
   - This is typically done using **gradient descent** optimization.
6. **Decision Boundary**
   The sigmoid function centers around (z = 0). So, the decision boundary is where the linear equation equals zero (i.e., where $(\hat{y} = 0.5)$).

Logistic Regression is a simple yet powerful method for binary classification:

- It models probability using the sigmoid function.
- Converts inputs via a linear equation.
- Applies a threshold to make final predictions.
- Trained using log-loss to optimize performance.

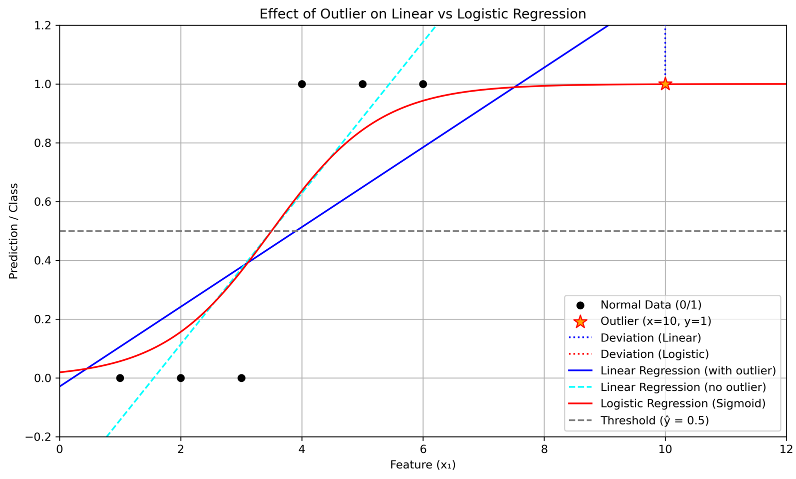


```
In [ ]:
```

Visual Element	**Description**
🔵 Blue Dots	Class 0 points (y = 0)
🟢 Green Dots	Class 1 points (y = 1)
🔵 Blue Line	Linear regression prediction
🔴 Red Curve	Logistic regression sigmoid curve
🟣 Dashed Line	Threshold at ŷ = 0.5 (decision line)

---

comparing



plt.show()

Element	Description
⭐ **Outlier**	Plotted at (10, 1) with orange star marker
🔵 **Dotted Blue Line**	Shows how far the outlier is from the blue line
🔴 **Dotted Red Line**	Shows difference from sigmoid output (bounded)
🔹 **Cyan Dashed Line**	Ideal linear regression (ignores outlier)
🔵 **Blue Line**	Skewed linear regression due to outlier
🔴 **Red Curve**	Logistic regression handles it well (sigmoid)

Aspect	Logistic Regression ✅	Linear Regression ❌
Prediction Range	Always between 0 and 1	Can go below 0 or above 1
Effect of Outlier (x = 10)	Minimal, curve still bounded	Pulls line upward significantly
Decision Threshold	Clear at $\hat{y} = 0.5$	Arbitrary, can be distorted
Interpretation	Probabilistic	Continuous value (not class)

Confusion Matrix

	Actual +ve	Actual -ve
Predicted +ve	✅ TP = 80	❌ FN = 10
Predicted -ve	❌ FP = 20	✅ TN = 890

Classification Metric Formulas

🔹 Precision

$$\text{Precision} = \frac{TP}{TP+FP}$$

🔹 Recall

$$\text{Recall} = \frac{TP}{TP+FN}$$

🔹 F1 Score

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

🔹 Accuracy (optional)

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Where:

- (TP): True Positives
- (FP): False Positives
- (FN): False Negatives
- (TN): True Negatives

Understanding Precision, Recall, and F1-Score

With a Simple Example: Predicting Women from a Crowd

---

1. Dataset (20 People)

You built a model to predict who is a *woman* in a group.
Out of 20 people:

- 10 are *actual women*
- 10 are *actual men*

Here's the predicted outcome:

Person ID	Actual Gender	Predicted Gender	Type
1	Woman	Woman	✅ True Positive (TP)
2	Woman	Woman	✅ TP
3	Woman	Woman	✅ TP
4	Woman	Woman	✅ TP
5	Woman	Woman	✅ TP
6	Woman	Woman	✅ TP
7	Woman	Man	❌ False Negative (FN)
8	Woman	Man	❌ FN
9	Woman	Man	❌ FN
10	Woman	Man	❌ FN
11	Man	Woman	❌ False Positive (FP)
12	Man	Woman	❌ FP
13	Man	Woman	❌ FP
14	Man	Woman	❌ FP
15	Man	Woman	❌ FP
16	Man	Woman	❌ FP

Person ID	Actual Gender	Predicted Gender	Type
17	Man	Man	✅ True Negative (TN)
18	Man	Man	✅ TN
19	Man	Man	✅ TN
20	Man	Man	✅ TN

---

2. Confusion Matrix

	Actual: Woman	Actual: Man
Predicted: Woman	6 (✅ TP)	6 (❌ FP)
Predicted: Man	4 (❌ FN)	4 (✅ TN)

📐 3. Metric Calculations

✅ Basic Counts:

- *True Positives (TP)* = 6
- *False Negatives (FN)* = 4
- *False Positives (FP)* = 6
- *True Negatives (TN)* = 4

🔹 *Recall*

"Out of all actual women, how many did we correctly predict?"

$$Recall = \frac{TP}{TP+FN} = \frac{6}{6+4} = \frac{6}{10} = 0.60 = 60\%$$

🔹 *Precision*

"Out of all predicted women, how many were actually women?"

$$Precision = \frac{TP}{TP+FP} = \frac{6}{6+6} = \frac{6}{12} = 0.50 = 50\%$$

🔹 *F1 Score*

"Balance between precision and recall"

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} = 2 \times \frac{0.5 \times 0.6}{0.5+0.6} = \frac{0.6}{1.1} \approx 0.545 = 54.5\%$$

🧘 4. Final Conclusion

- ✅ *Recall = 60%* → The model correctly found 6 out of 10 actual women.
- ❌ *Precision = 50%* → Only half of the people predicted as women were correct.
- ⚖️ *F1 Score = 54.5%* → Shows *average performance*, since both FP and FN are high.

🧠 5. When to Use What?

Metric	What it Tells You	Use When...
Recall	How many actual women were caught	Missing positives is worse
Precision	How many predicted women were actually women	False alarms are more harmful
F1 Score	Did the model balance both well?	You care about both recall and precision

---

✅ Key Takeaway:

> A good model should have *high recall* if missing women is risky,
> *high precision* if wrongly tagging men as women is risky,
> and *high F1 Score* if *both errors are equally bad*.

Metric	Weak	Acceptable	Strong	Excellent
Precision	< 0.60	0.60–0.75	0.75–0.90	> 0.90
Recall	< 0.60	0.60–0.75	0.75–0.90	> 0.90
F1 Score	< 0.60	0.60–0.75	0.75–0.90	> 0.90

📘 K-Means Clustering – Theory, Steps, Diagrams, and Formulas

🔍 What is K-Means?

K-Means is an **unsupervised learning algorithm** used to group similar data points into `k` clusters based on distance.

🔢 Step-by-Step Process (With Diagram)

✅ Step 1: Initialization

- Choose the number of clusters `k` (e.g., 3)
- Randomly place `k` cluster centers (centroids)

✅ Step 2: Assignment Step

- Assign each data point to the **nearest centroid** using **Euclidean distance**:

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

✅ Step 3: Update Step

- Recalculate the **centroid** of each cluster (mean of all points assigned to it)

$$\text{New centroid} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

✅ Step 4: Repeat Until Convergence

- Repeat assignment and update steps until:
  - Centroids don't move much
  - Or cluster assignments don't change

📊 Summary Table

Step	Description
1	Choose `k` clusters
2	Place `k` random centroids
3	Assign each point to nearest
4	Recalculate centroids
5	Repeat until stable

📌 Limitations

- You must choose `k` beforehand
- Sensitive to outliers
- Different runs may give different results

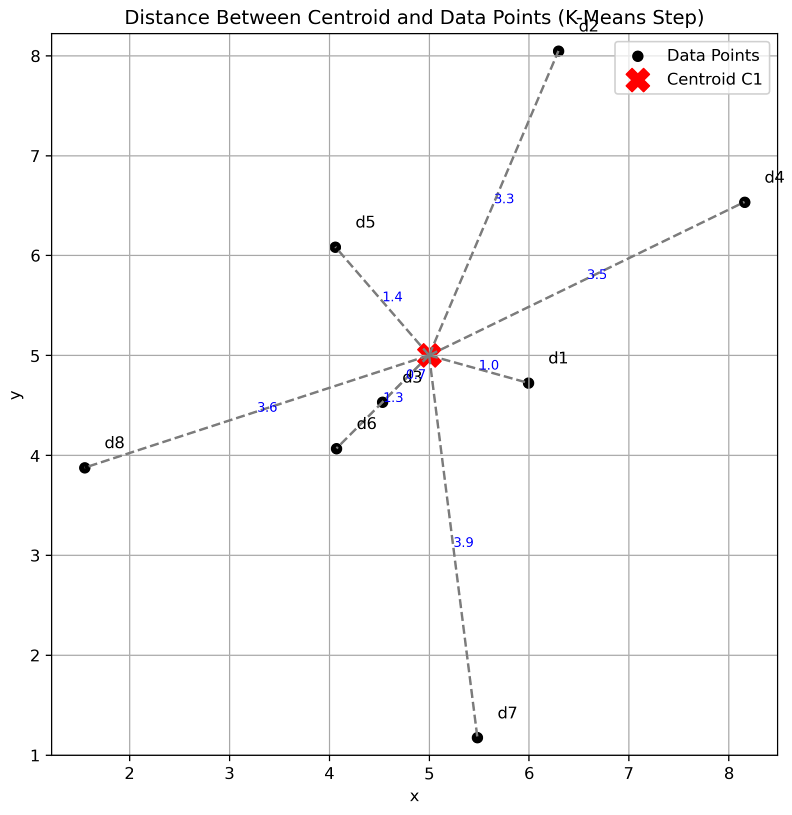
📷 Diagram Overview

K-Means Diagram

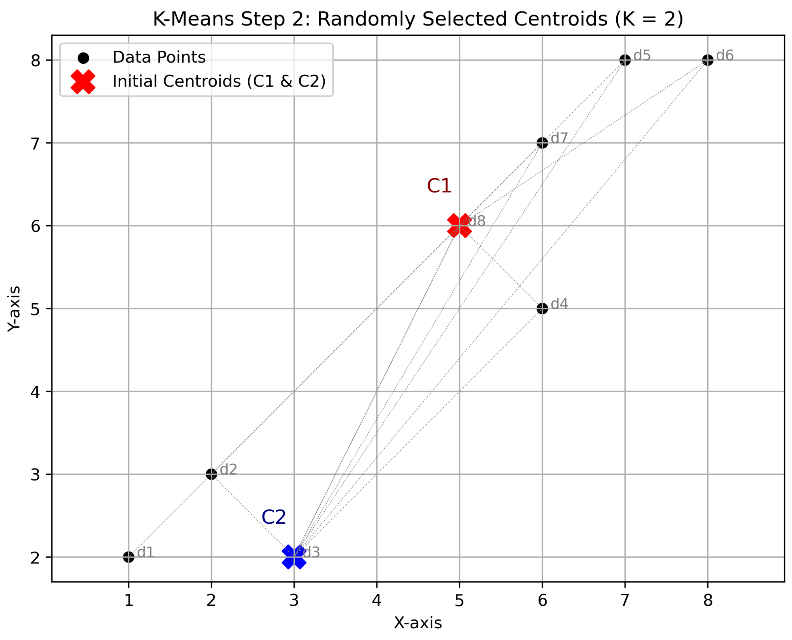
Step :1 find k value

Step :2 randomly select 2 cetroid

how to calculate distance two data points

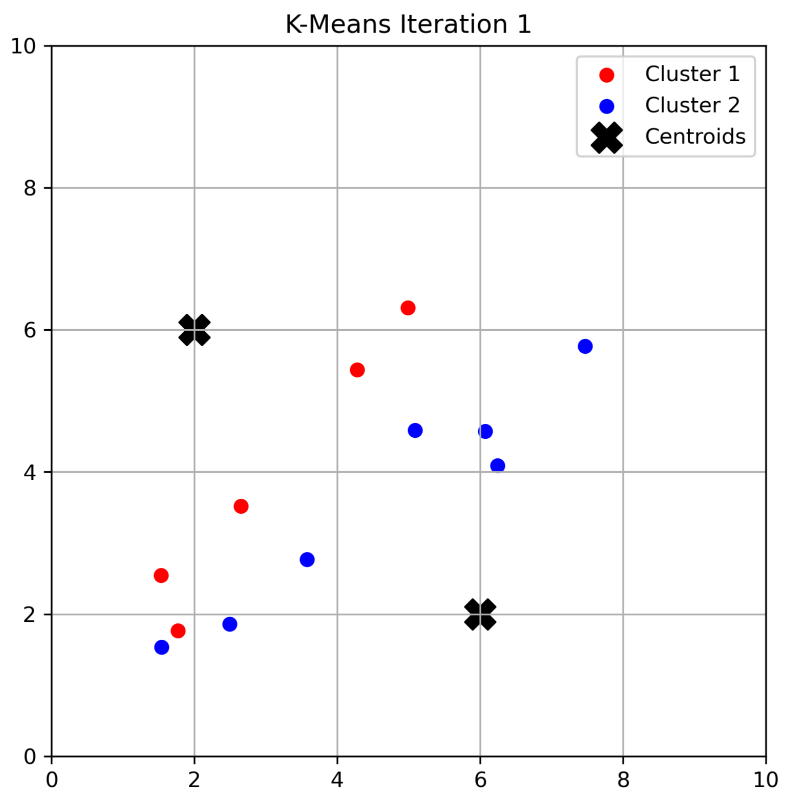


how to calculate distance between all the points



Step 4: lets consider k=2 and choosen random datapoints

Iteration 1



🔷 Simple Centroid Formula

$$\mu_i = \left(\frac{x_1 + x_2 + \cdots + x_n}{n}, \frac{y_1 + y_2 + \cdots + y_n}{n} \right)$$

🧮 Example

Points in Cluster 1:

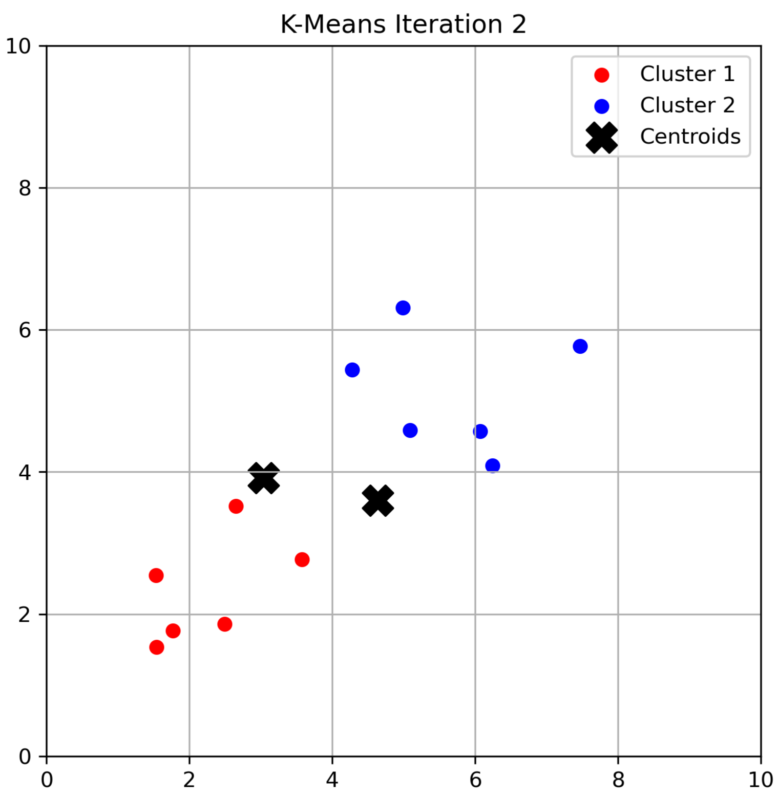
- (2, 3), (1, 2), (3, 4)

Then:

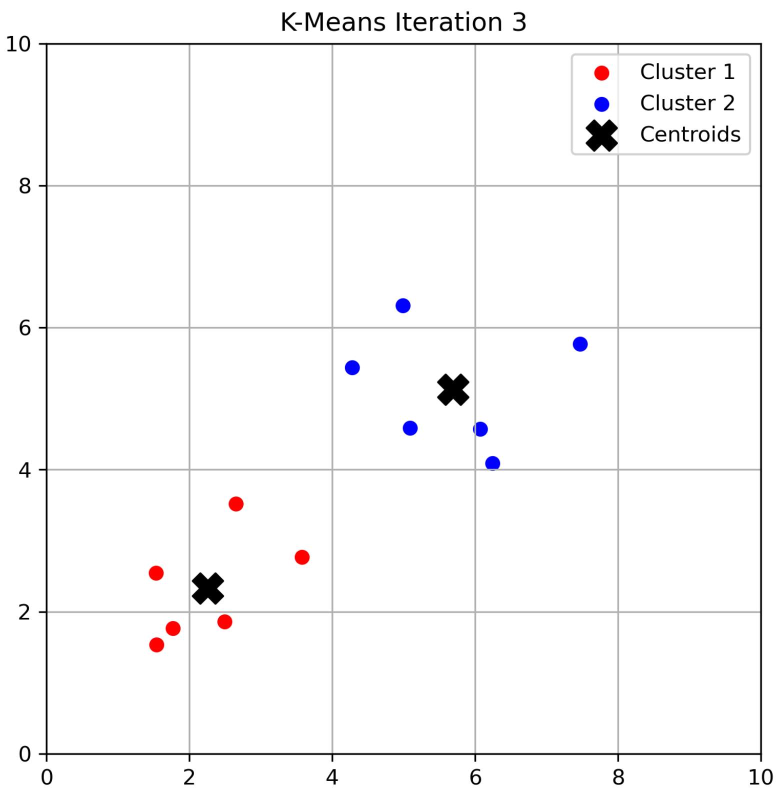
$$\mu_1 = \left(\frac{2+1+3}{3}, \frac{3+2+4}{3} \right) = (2.0,\ 3.0)$$

✅ This is the new centroid for Cluster 1.

🔷 Iteration 2



🔹 Iteration 3



📍 K-Means Centroid Formula

$$\mu_i = \left(\frac{x_1 + x_2 + \cdots + x_n}{n}, \ \frac{y_1 + y_2 + \cdots + y_n}{n} \right)$$

🧮 Iteration Breakdown

Iteration 1

- C1 (init): (2.1, 1.9)
- C2 (init): (8.1, 8.3)

After Assignment

- Cluster 1: 6 points → New C1 = (2.03, 2.03)
- Cluster 2: 6 points → New C2 = (7.93, 7.88)

Iteration 2

- Reassign points → same clusters
- Centroids unchanged → ✅ Converged

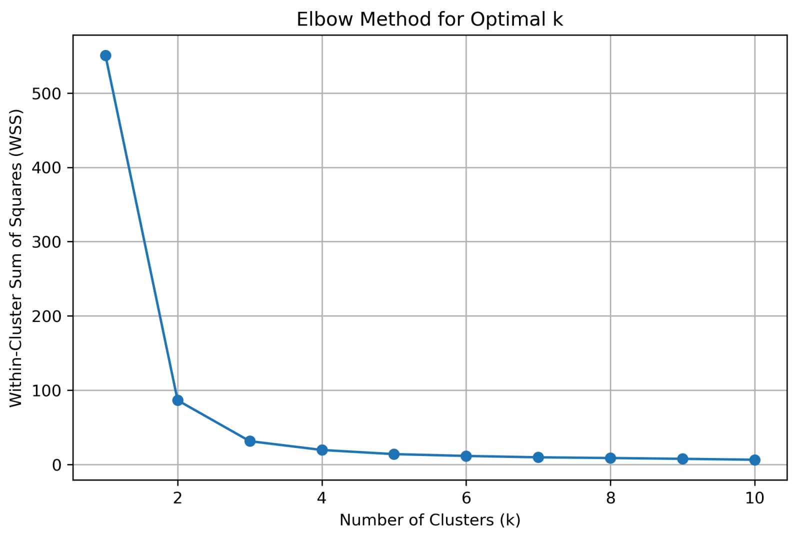
✅ Final Centroids

- C1 = (2.03, 2.03)
- C2 = (7.93, 7.88)

```
        ┌────────────────────────────┐
        │  Start with raw data (X)   │
        └────────────┬───────────────┘
                     │
                     ▼
┌──────────────────────────────────────────┐
│ Choose K (number of clusters)            │
└────────────┬─────────────────────────────┘
             │
             ▼
┌──────────────────────────────────────────┐
│ Randomly initialize K centroids          │
└────────────┬─────────────────────────────┘
             │
             ▼
┌──────────────────────────────────────────┐
│ Assign each point to nearest centroid    │
└────────────┬─────────────────────────────┘
             │
             ▼
┌──────────────────────────────────────────┐
│ Update centroids:                        │
│ Take mean of all points in each cluster│
└────────────┬─────────────────────────────┘
             │
             ▼
┌──────────────────────────────────────────┐
│ Have centroids changed?                  │
└───────┬────────────────┬─────────────────┘
        │Yes             │No
        ▼                ▼
```

if yes go back to assign each point to the centroid if no stop the process(iteration)

elbow point



```
In [ ]:
```