

# Advanced Algorithms

# Clustering

Imaging that you have designing T-shirt for consumers. You are looking at your consumers' weight and height, which would allow you to decide how many sizes to offer and which size fits a particular individual. Instead of classifying your consumers arbitrarily, you can use data to perform the task more precisely.

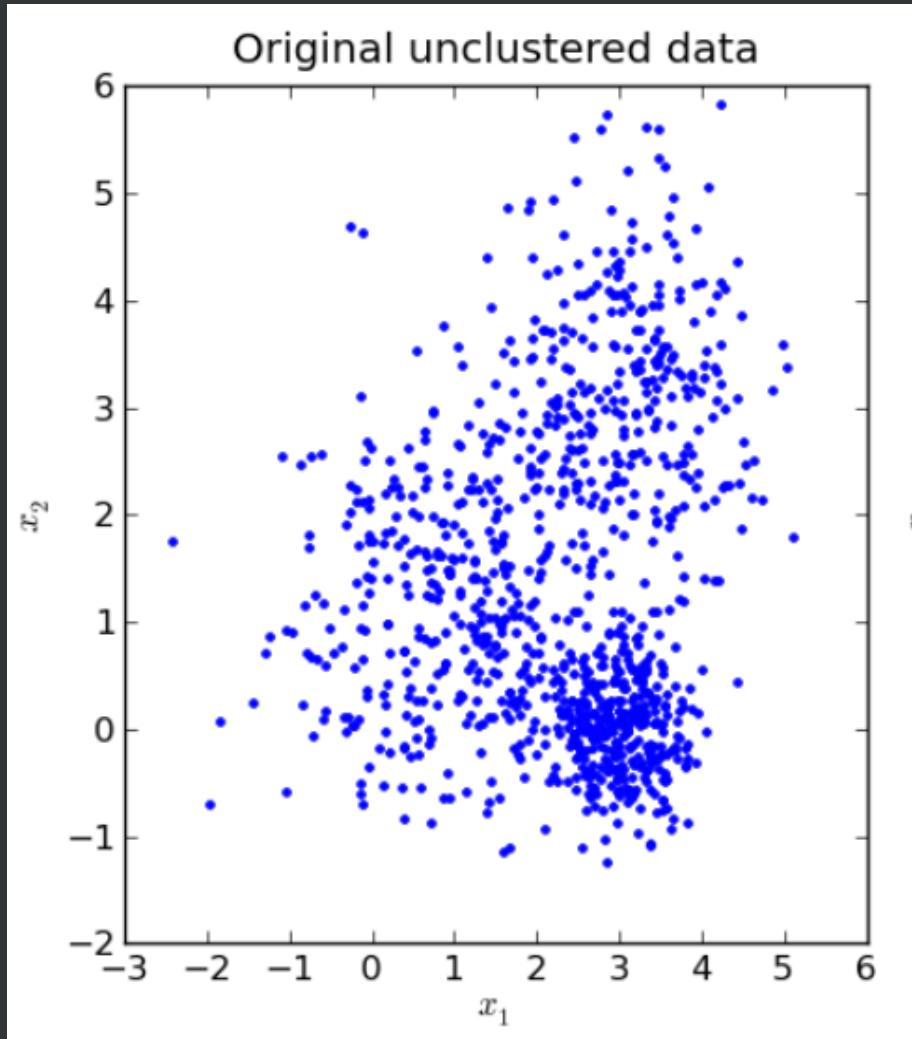
## SIZE SELECTION. 尺码选择表

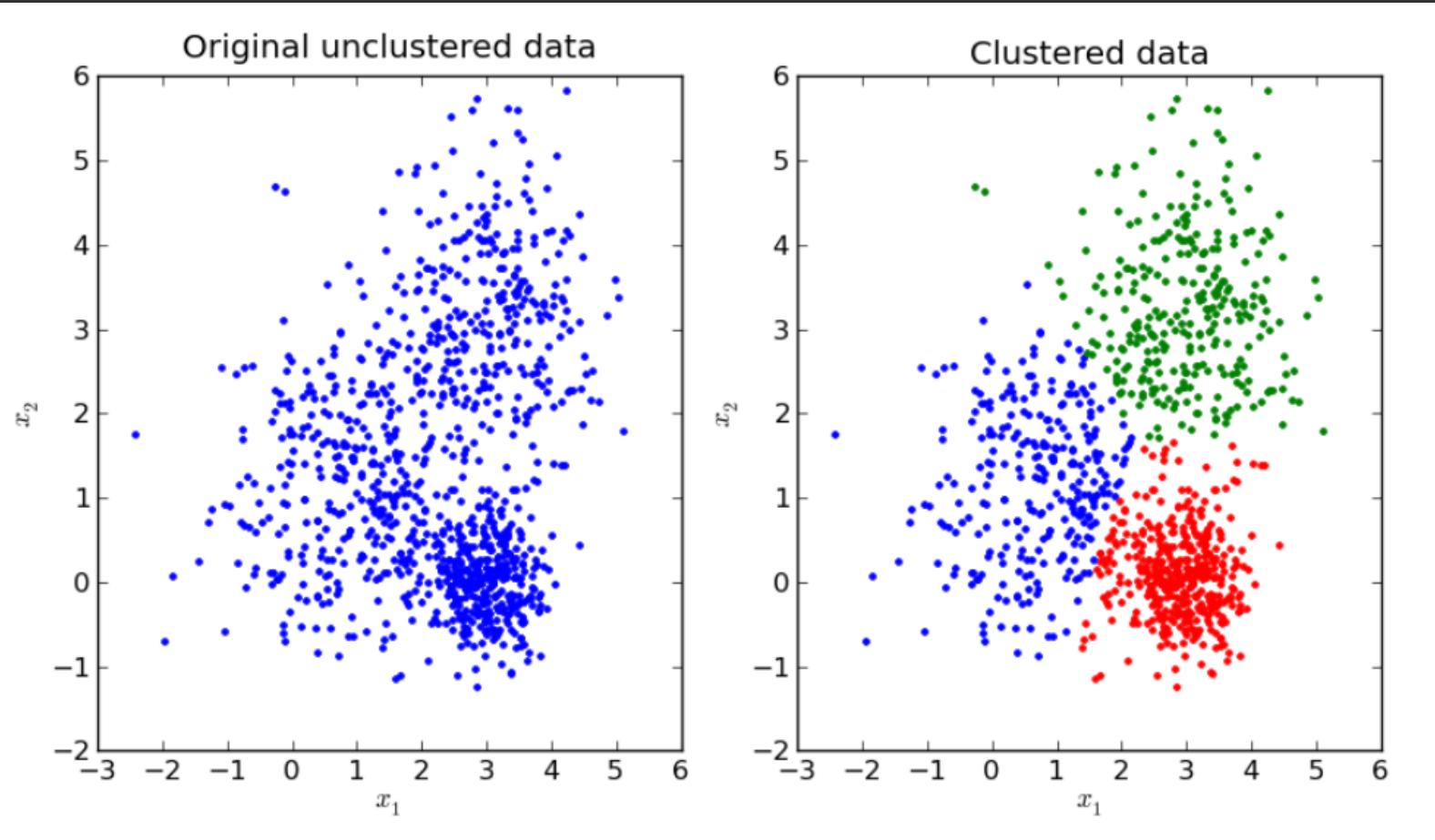
身高 (cm)	体重 (斤)	95	105	115	125	135	145	155	165	175	185	195	205
165													
170	S												
175		M					L						
180								XL					
185									XXL				
190										XXXL			
195											XXXXL		

Why offer 7 sizes? Are these sizes optimal?

Image credit: taobao.com

Here is consumer data. How would you classify them into groups?





## The $K$ -means Algorithm

The  $K$ -means algorithm is an EM (expectation-maximization) algorithm commonly used for classifying objects.

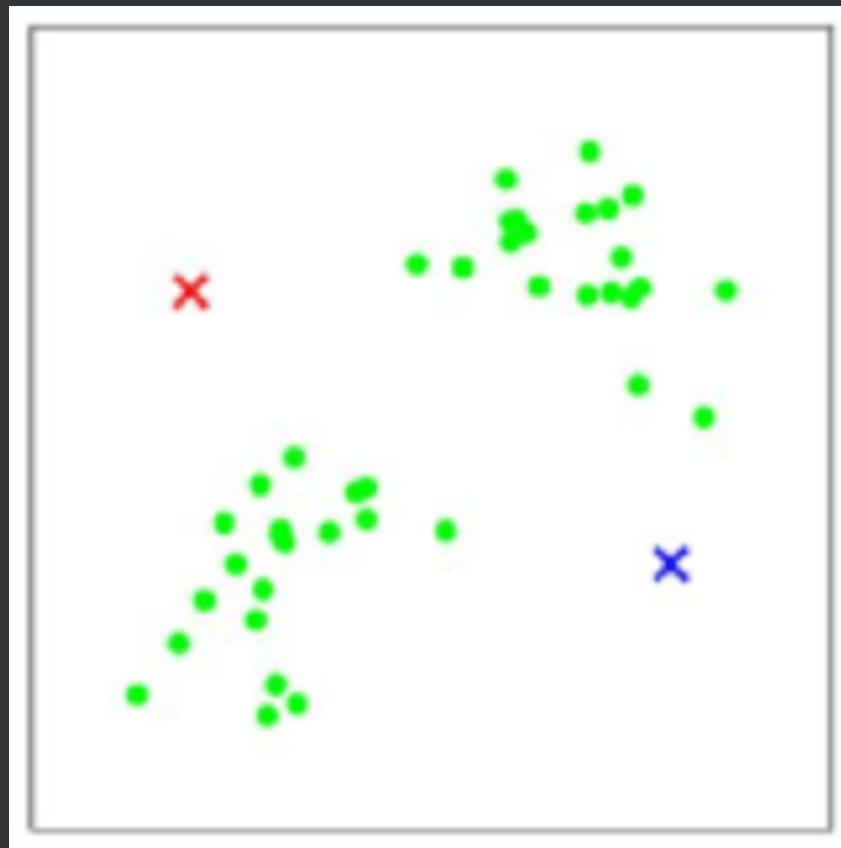
**Input:** A number of observations  $(X_1, X_2, \dots, X_n)$  and  $k$ , the number of groups to be classified

**Output:**  $k$  mutually exclusive and collectively exhaustive groups containing all observations

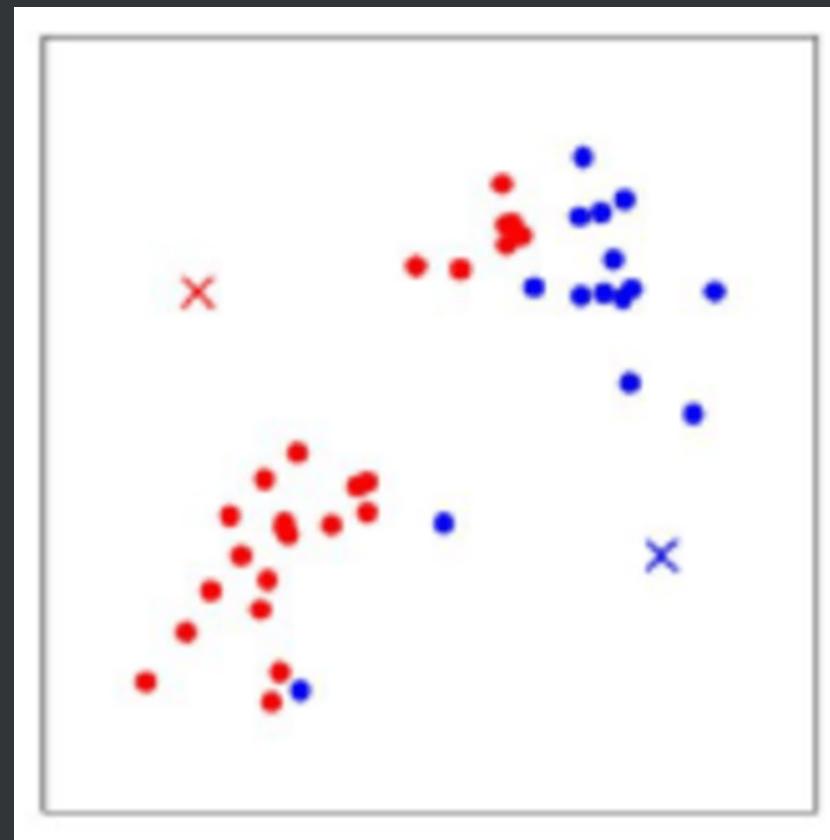
Classify the following observations into  $k = 2$  groups:



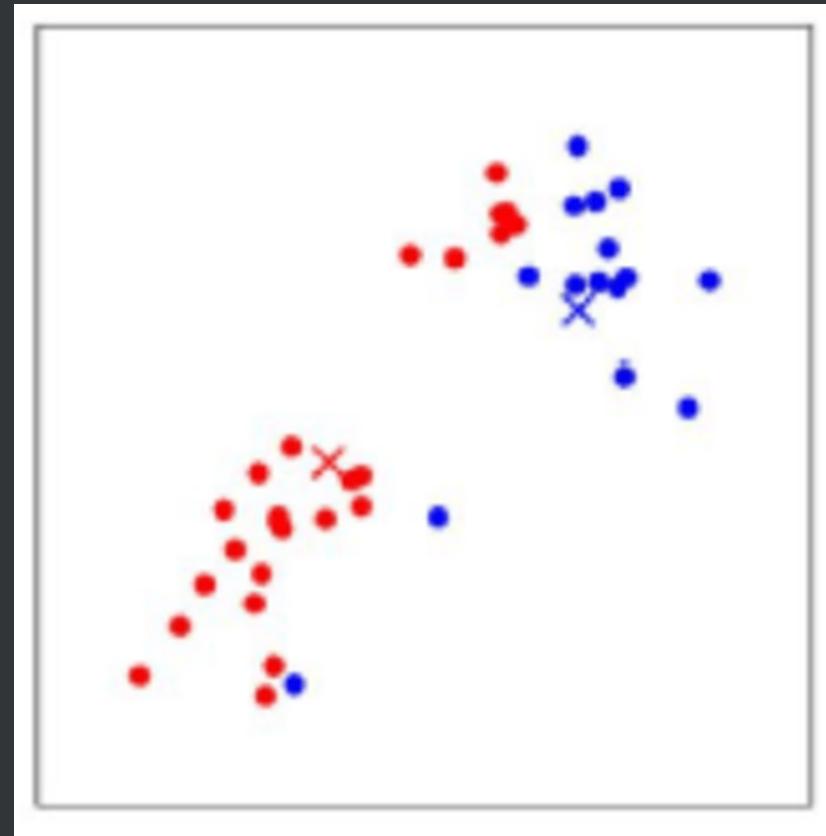
Step 1: Random choose  $k = 2$  “centers” for your clusters.



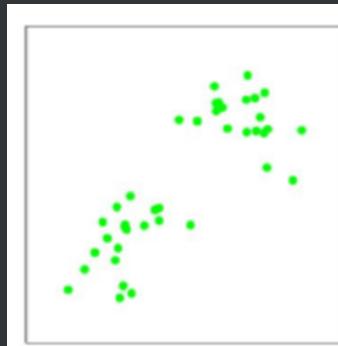
Step 2: Assign each observation to the nearest center.



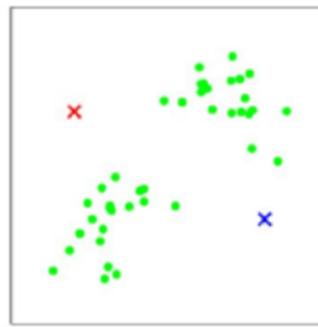
Step 3: Update the location the centers, which is given by the average location of all points in the corresponding cluster.



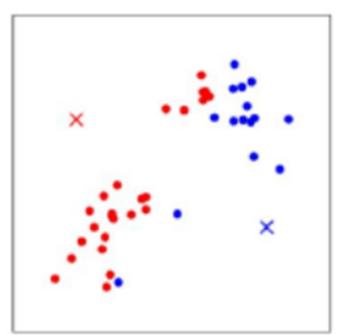
Repeat the above process again and again until the centers no longer change.



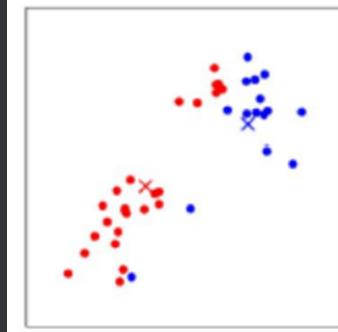
(a)



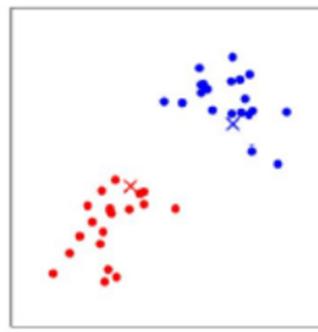
(b)



(c)



(d)



(e)



(f)

## The $K$ -means algorithm:

1. Select cluster centers  $c_1, \dots, c_k \in \mathbb{R}^d$  arbitrarily.
2. Assign every  $x \in \mathcal{X}$  to the cluster  $\mathcal{C}_i$  whose cluster center  $c_i$  is closest to it, i.e.,  $\|x - c_i\| \leq \|x - c_j\|$  for all  $j \neq i$ .
3. Set  $c_i = \frac{1}{|\mathcal{C}_i|} \sum_{x \in \mathcal{C}_i} x$ .
4. If clusters or centers have changed, goto 2. Otherwise, terminate.

[https://www.youtube.com/embed/R2e3Ls9H\\_fc?enablejsapi=1](https://www.youtube.com/embed/R2e3Ls9H_fc?enablejsapi=1)

## Performing $K$ -means with Python:

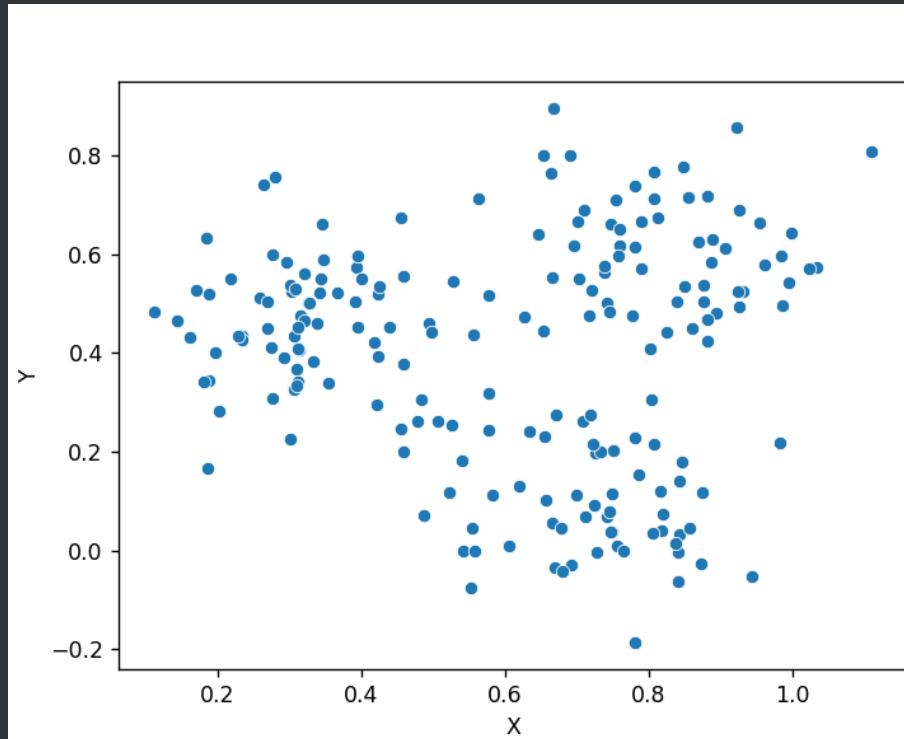
```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 from sklearn.cluster import KMeans
5 from sklearn.metrics import silhouette_score
6 data = pd.read_csv('https://ximarketing.github.io/data/clustering.csv')
7 print(data)
```

	X	Y
0	0.627123	0.473972
1	0.562636	0.714202
2	0.295934	0.583077
3	0.815801	0.121524
4	0.345354	0.662115

The input is a CSV file containing the X and Y coordinates of some individuals.

```
1 sns.scatterplot(data = data, x = 'X', y = 'Y')  
2 plt.show()
```

Let's visualize these locations!

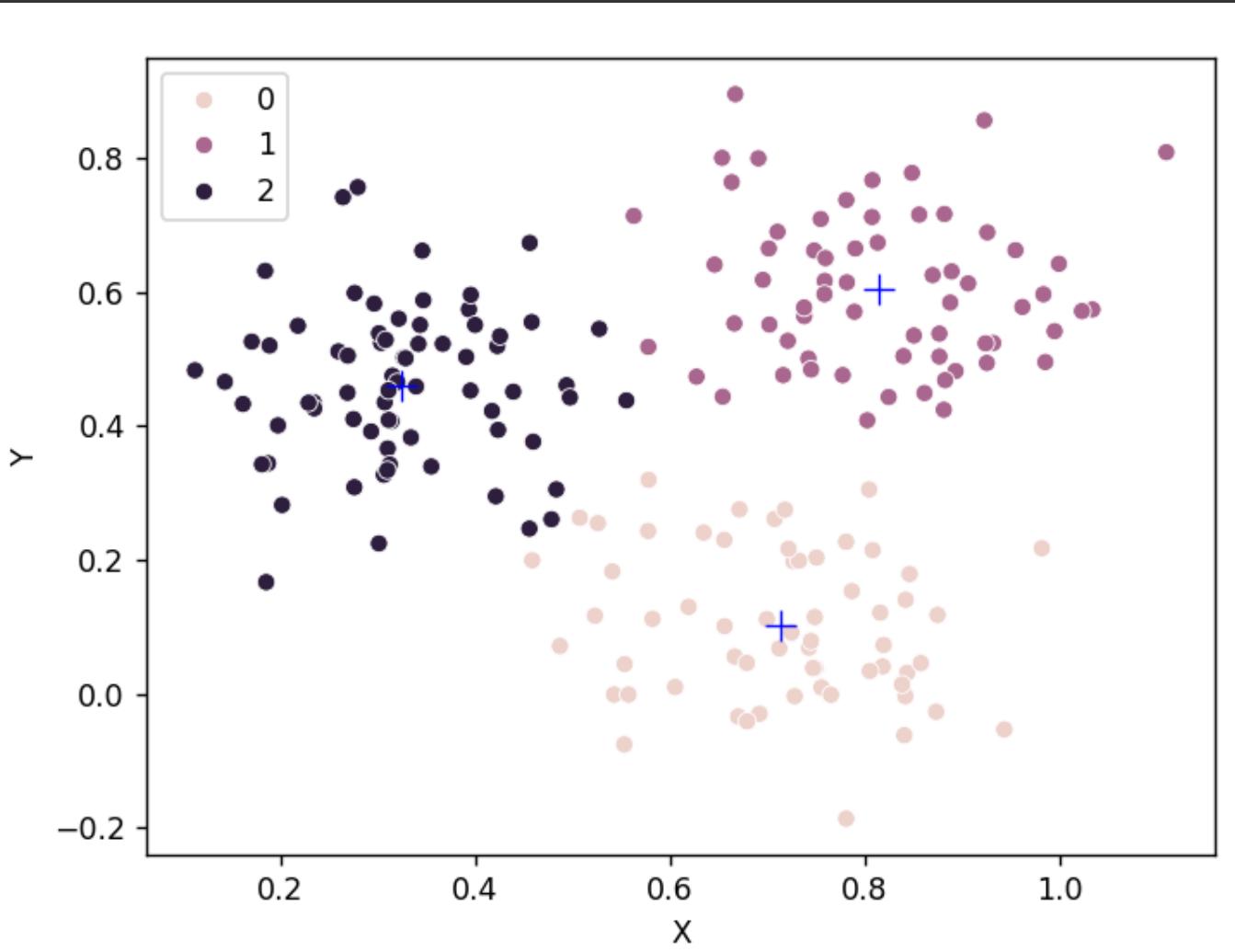


How many groups do they belong to?

```
1 kmeans = KMeans(n_clusters = 3, random_state = 0, n_init='auto')
2 kmeans.fit(data)
3 sns.scatterplot(x = data['X'], y = data['Y'], hue = kmeans.labels_)
4 centers = kmeans.cluster_centers_
5 sns.scatterplot(x = centers[:, 0], y = centers[:, 1],
6                  color = 'blue', s=100, marker='+')
7 plt.show()
```

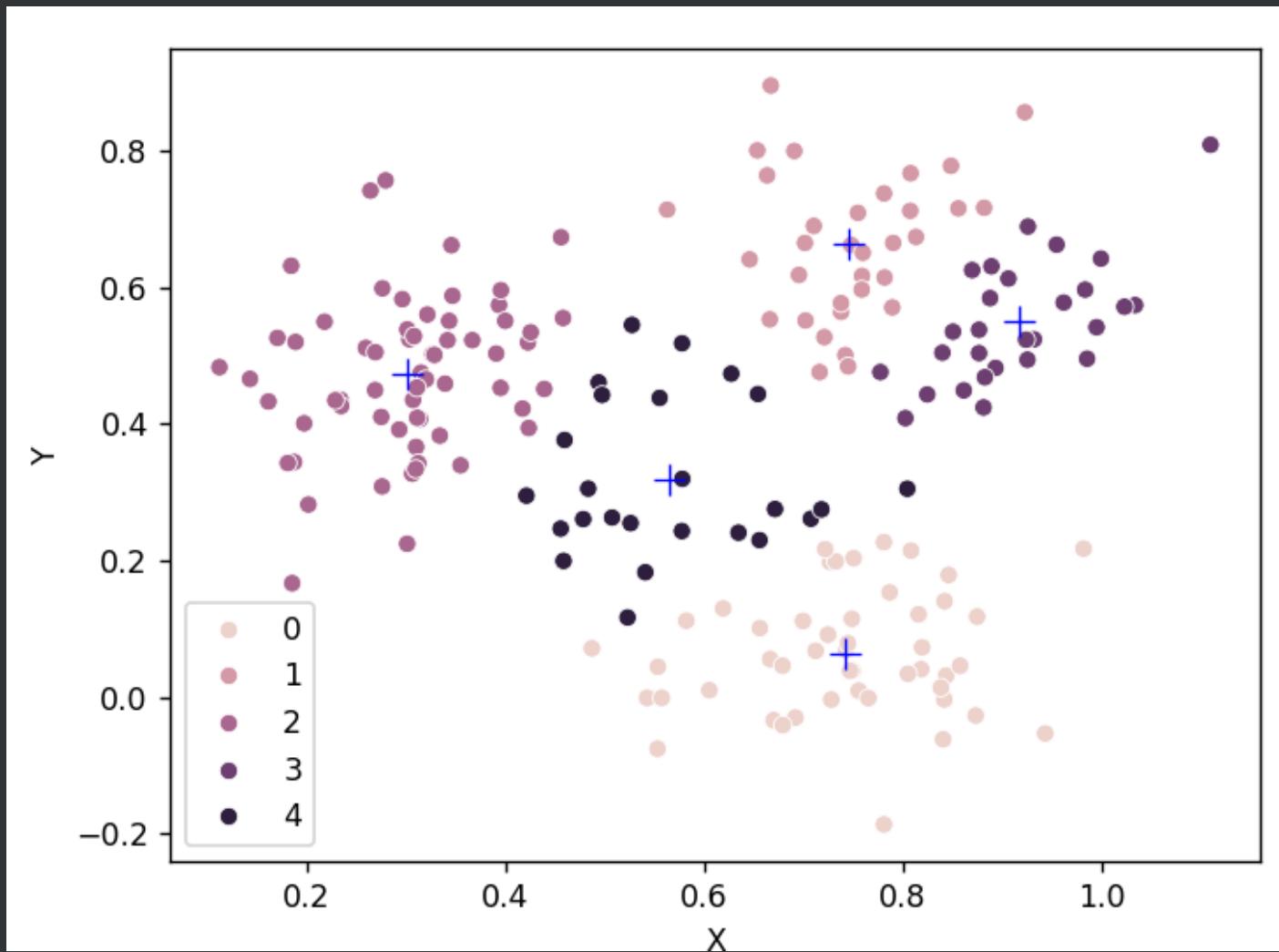
Now, we use the KMeans function to classify the observations. If you don't understand the code, ask GPT:

The screenshot shows a GPT-4 conversation interface. At the top, there is a user profile icon for 'Xi Li' and a timestamp '15/04/2024, 12:46:31'. Below this, a question is asked: 'What is the meaning of random\_state in Python's KMeans?'. In the bottom left corner, there is a green icon of a laptop with a brain, labeled 'GPT4 (Tokens used 801)'. The main response area contains the following text: "'random\_state' is a parameter used in Python's KMeans implementation, specifically in the `sklearn.cluster.KMeans` function. It controls the random number generation used during the initialization of the centroids in the K-means algorithm."



Does it look good?

Let's try five groups instead...



## Exercise

Use ChatGPT to generate a desktop APP which demonstrates the K-means algorithm

Question:  
What is the optimal number of clusters?

We use a measure called “Silhouette Score.”

It is done in the following way:

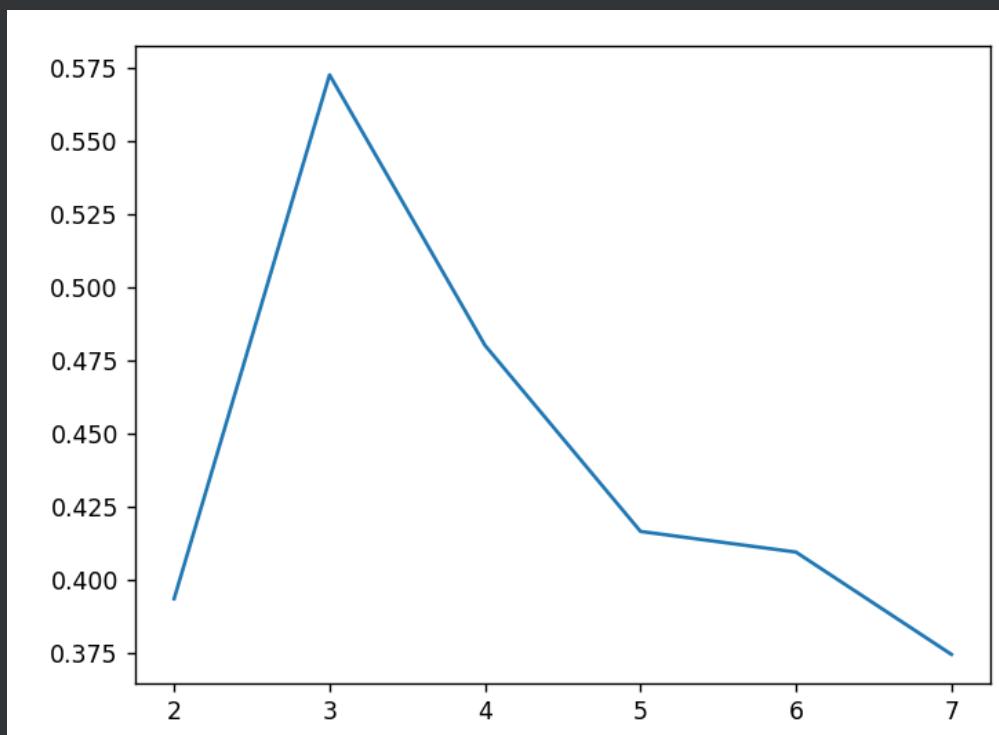
1. For each data point, calculate the average distance between the data point and all other points in the same cluster. This is called the intra-cluster distance.
2. For each data point, calculate the average distance between the data point and all other points in the nearest cluster. This is called the nearest-cluster distance.
3. Calculate the silhouette score for each data point as  $(\text{nearest-cluster distance} - \text{intra-cluster distance}) / \text{nearest-cluster distance}$ .
4. The overall silhouette score is the average of the silhouette scores for all data points.

We use a measure called “Silhouette Score.”

The idea is simple: Each point should be as close as possible to points in the same cluster, and as far as possible from points in other clusters.

It should be value between 0 and 1. The greater the score is, the better the algorithm performs.

```
1 K = range(2, 8)
2 fits = []
3 score = []
4 for k in K:
5     model = KMeans(n_clusters=k, random_state=0, n_init='auto').fit(data)
6     fits.append(model)
7     score.append(silhouette_score(data, model.labels_, metric='euclidean'))
8 sns.lineplot(x = K, y = score)
9 plt.show()
```



3 is the optimal number!

# The complete code is here:

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 from sklearn.cluster import KMeans
5 from sklearn.metrics import silhouette_score
6 data = pd.read_csv('https://ximarketing.github.io/data/clustering.csv')
7 print(data)
8 sns.scatterplot(data = data, x = 'X', y = 'Y')
9 plt.show()
10 kmeans = KMeans(n_clusters = 5, random_state = 0, n_init='auto')
11 kmeans.fit(data)
12 sns.scatterplot(x = data['X'], y = data['Y'], hue = kmeans.labels_)
13 centers = kmeans.cluster_centers_
14 sns.scatterplot(x = centers[:, 0], y = centers[:, 1],
15                  color = 'blue', s=100, marker='+')
16 plt.show()
17 K = range(2, 8)
18 fits = []
19 score = []
20 for k in K:
21     model = KMeans(n_clusters=k, random_state=0, n_init='auto').fit(data)
22     fits.append(model)
23     score.append(silhouette_score(data, model.labels_, metric='euclidean'))
24 sns.lineplot(x = K, y = score)
25 plt.show()
```

The  $k$ -means algorithm can also be used for image compression. How could this be done?

# How would you compress this image?



# Image Compression

An image has millions of pixels. Each pixel is represented by its RGB color, e.g., (127, 176, 96).

For each pixel, you need to save a lot of information! There are  $256 \times 256 \times 256 = 16,777,216$  possible color combinations for each pixel. This requires a lot of storage.

# Image Compression

The idea is that we do not need so many colors! Some colors are very similar and we can combine them into a single color. For instance, consider two pixels,  $(45, 80, 97)$  and  $(46, 79, 98)$ . Their colors are very much similar and we can combine them into a single color.

# Image Compression

If we combine similar colors and come up with only 20 possible colors for the images, the size of the image will be much smaller! Previous, each pixel is represented by a vector  $(r_i, g_i, b_i)$ , where  $0 \leq r_i, b_i, g_i \leq 255$ . Now, each pixel is represented by simply one number  $0 \leq x_i \leq 19$ , where  $x_i = 0$  denotes the first color, and so on.

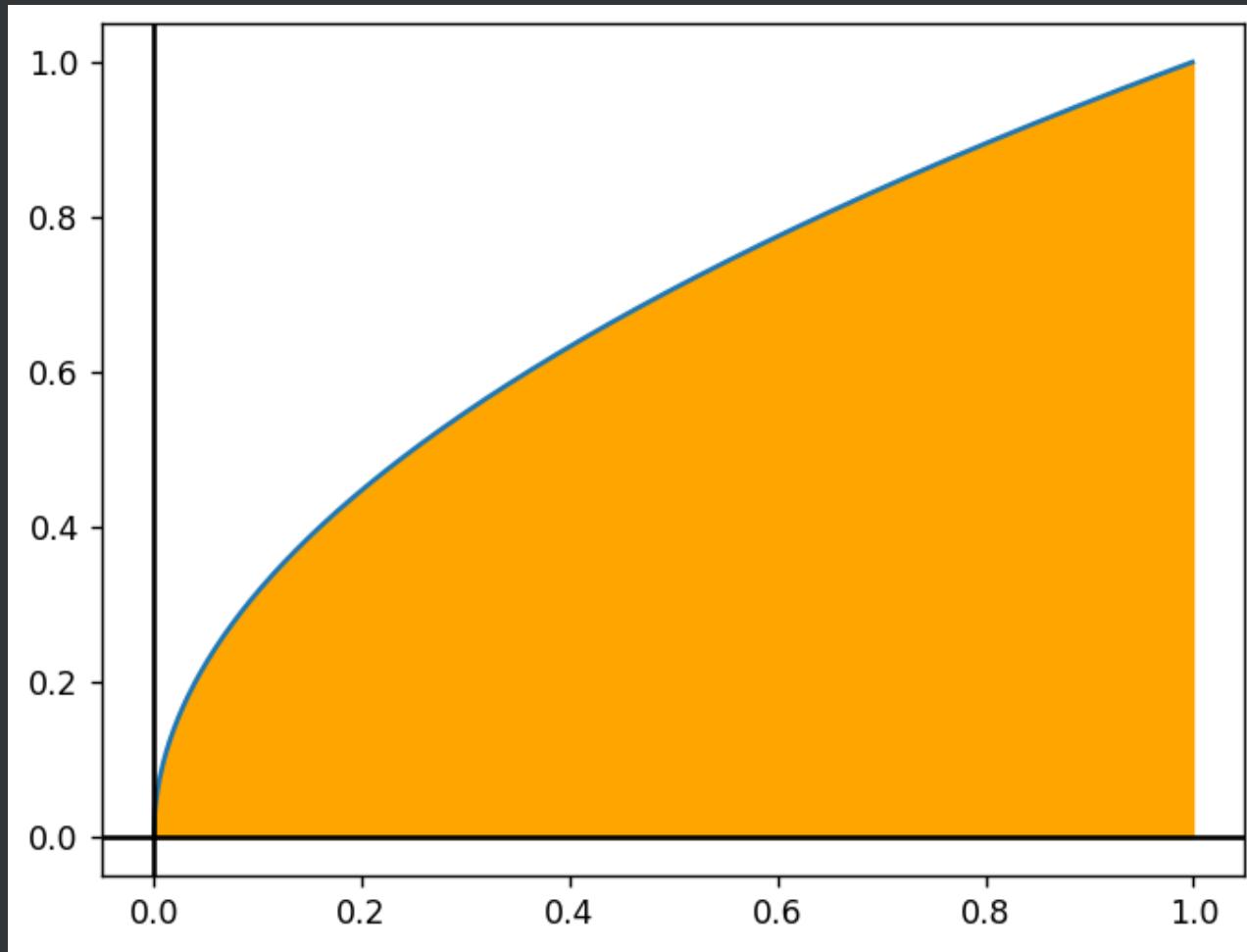
# The algorithm

- Each pixel is represented by a point in the 3D space,  $(r_i, g_i, b_i)$ .
- Cluster the points into  $k$  clusters, call them cluster  $j = 1, 2, \dots, k$ .
- Calculate the centroid (i.e., center) of each cluster.
- Replace the color of each pixel with the color of the corresponding centroid, and when saving the image, only save the corresponding cluster number  $j$ .

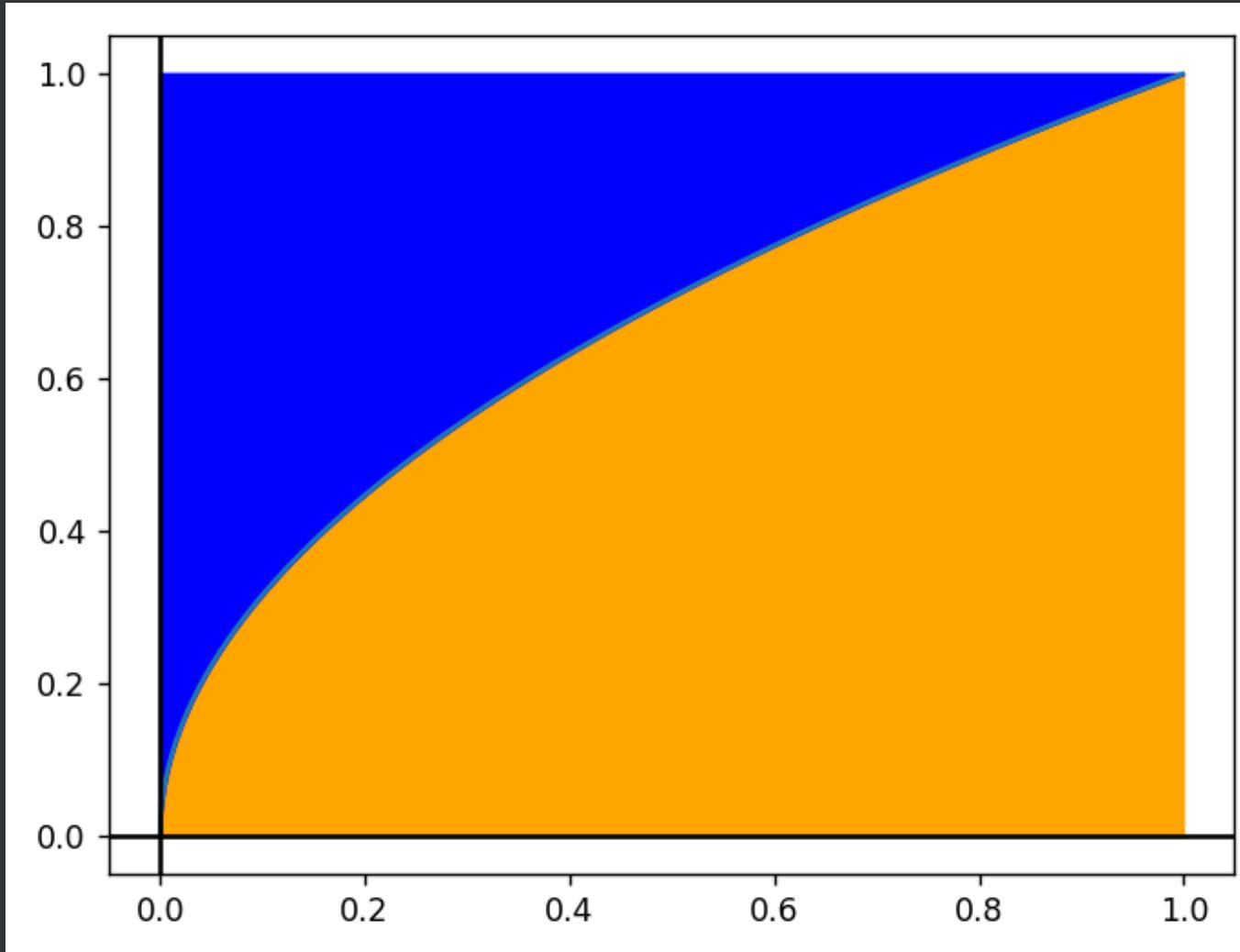
Calculate the following integral:

$$\int_0^1 \sqrt{x} dx$$

# Monte Carlo Algorithm

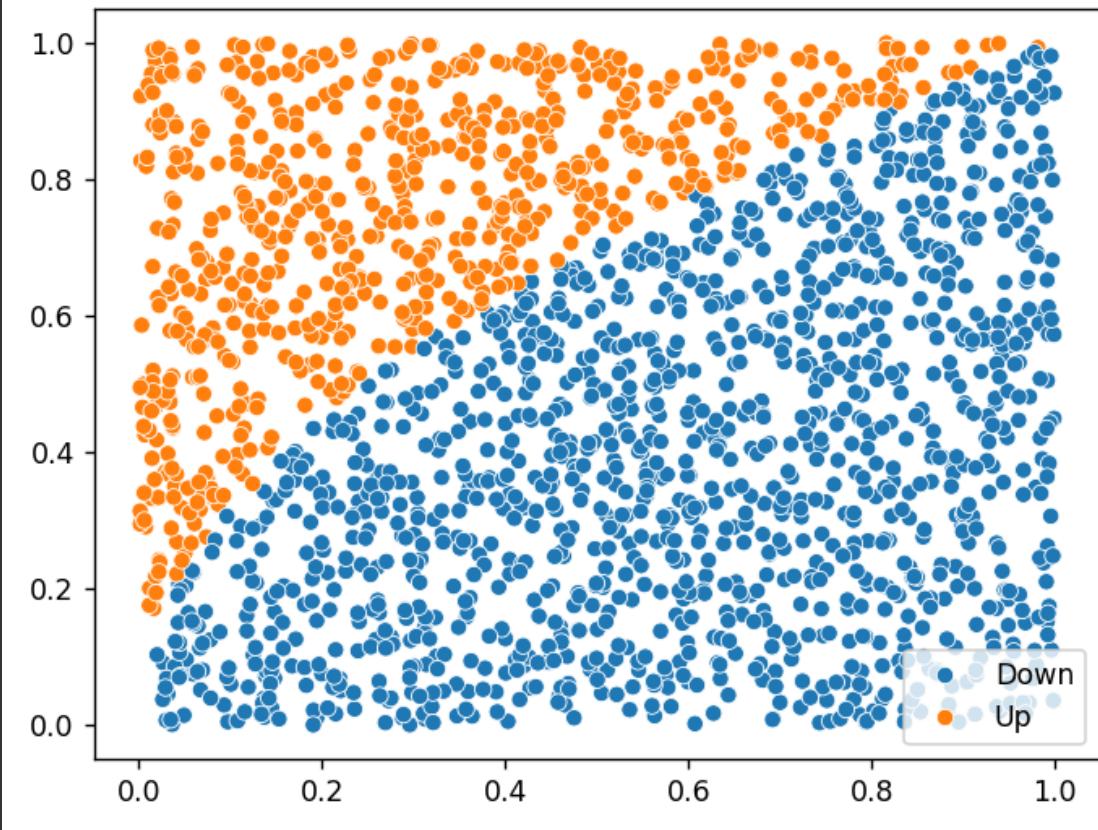


Integral is the area below the line  $f(x) = \sqrt{x}$



The area size of blue + orange = 1

We want to calculate the fraction of the orange area



The idea is as follows. We random draw a large number of points in the area  $[0, 1] \times [0, 1]$ , and count how many points are above  $\sqrt{x}$ . The size can be calculated as follows:

$$\text{size of orange area} = \frac{\text{number of orange points}}{\text{total number of points}} \times 1$$

Algorithm:

1. Randomly choose  $K$  points that are uniformly distributed over  $[0, 1] \times [0, 1]$ . Denote these points by  $(x_1, x_2), \dots, (x_K, y_K)$ .
2. Calculate how many points satisfy  $y_i < \sqrt{x_i}$ , denote by count\_below.
3. The size of the area is count\_below / K.

The complete code is here:

```
1 import random
2 import math
3 K = 10000
4 count_below = 0
5 for i in range(0, K):
6     x = random.uniform(0, 1)
7     y = random.uniform(0, 1)
8     if y <= math.sqrt(x):
9         count_below = count_below + 1
10 print (count_below/K)
```

Exercise: Calculate the value of  $\pi$ .

Exercise: Calculate the value of  $\pi$ .

In 1733, the French mathematician Buffon first came up with a probabilistic method to calculate  $\pi$  numerically. He threw needles to parallel lines, and found that the probability that a needle crosses a line is a function of  $\pi$ . Based on that, he calculated the value of  $\pi$ .

In Chinese, this is known as “布丰投针”.

<https://www.youtube.com/embed/kazgQXaeOHk?enablejsapi=1>

## Exercise

Use ChatGPT to generate a desktop APP which demonstrates how to calculate  $\pi$  numerically.

# Dynamic Programming

## Question

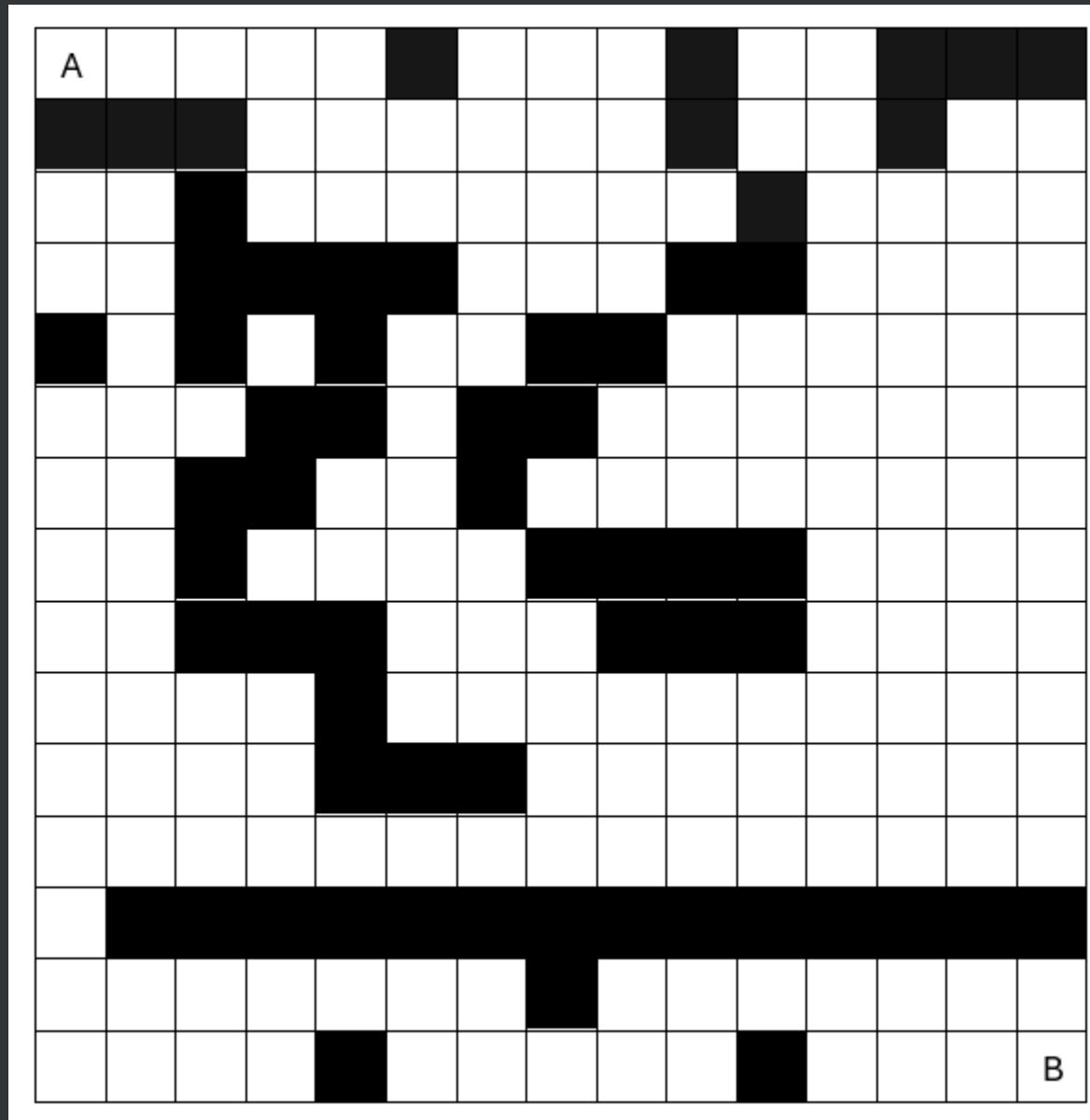
You are climbing a staircase. It takes 10 steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

## A More General Question

You are climbing a staircase. It takes  $n$  steps to reach the top.  
Each time you can either climb 1 or 2 steps. In how many  
distinct ways can you climb to the top?

<https://www.youtube.com/embed/tOYZcy2IzJA?enablejsapi=1>

Exercise: Answer the question for  $n = 10$ .



## Exercise: Write the code yourself

```
1 input = [[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1],  
2 [1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0],  
3 [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],  
4 [0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],  
5 [1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],  
6 [0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],  
7 [0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
8 [0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0],  
9 [0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],  
10 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
11 [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
12 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
13 [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
14 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
15 [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]]
```

Here, 1 means a stone and 0 means a feasible block

# Reference Solution

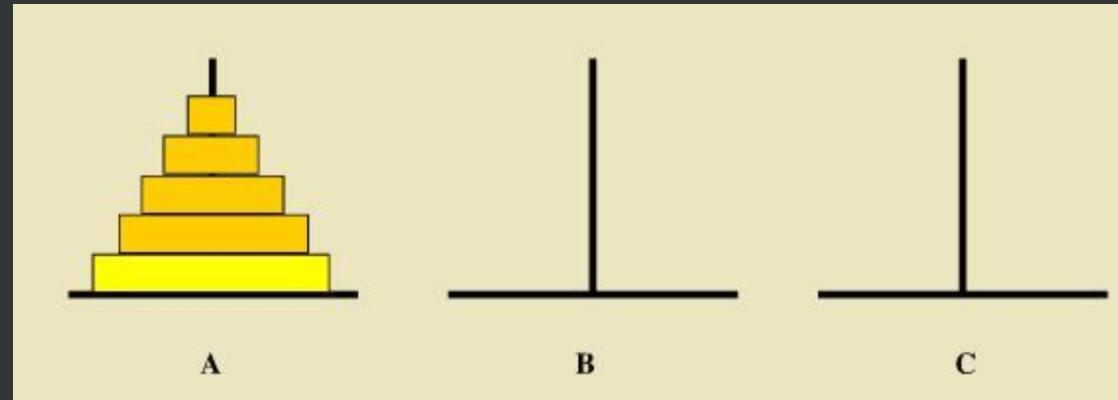
```
1 distance = [[-1] * 15 for _ in range(15)]
2 distance[0][0] = 0
3
4 for round in range(100):
5     for i in range(15):
6         for j in range(15):
7             if distance[i][j] == round:
8                 if ((i>=1) and (distance[i-1][j]==-1) and input[i-1][j]==0):
9                     distance[i-1][j] = round + 1
10                if ((j>=1) and (distance[i][j-1]==-1) and input[i][j-1]==0):
11                    distance[i][j-1] = round + 1
12                if ((i<=13) and (distance[i+1][j]==-1) and input[i+1][j]==0):
13                    distance[i+1][j] = round + 1
14                if ((j<=13) and (distance[i][j+1]==-1) and input[i][j+1]==0):
15                    distance[i][j+1] = round + 1
16
17 print(distance[14][14])
```

The answer is 46.

## Exercise

Use ChatGPT to generate an interactive APP which demonstrates the algorithm as a game

## Tower of Hanoi (汉诺塔)

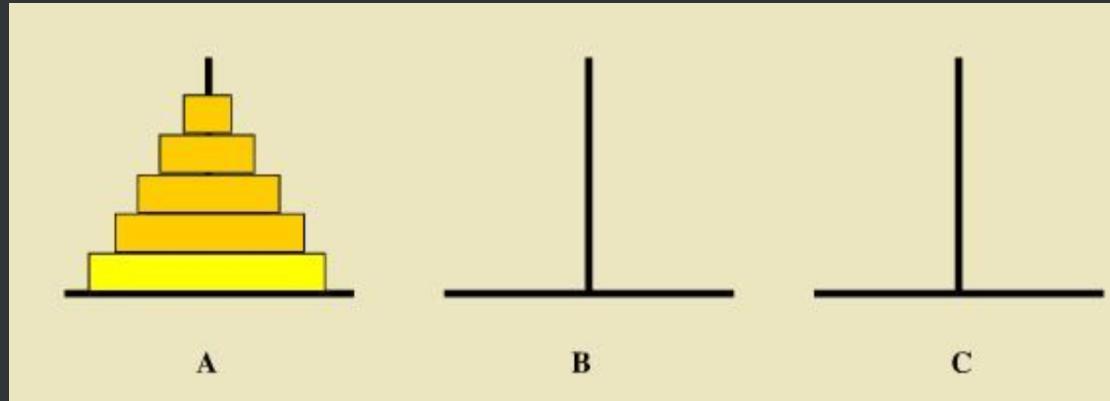


## Tower of Hanoi

You want to move  $n$  disks from rod A to rod C. Here are the rules:

- Only one disk can be moved at each round.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No disk may be placed on top of a disk that is smaller than it.

How to solve the problem?



# Divide-and-Conquer

Exercise: Write a program which solves the Hanoi problem given any  $n$ . The output is a few steps such as  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $B \rightarrow A$ , etc.

```
1 def tower_of_hanoi(n, source, target, auxiliary):
2     if n == 1:
3         print(f"Move disk 1 from rod {source} to rod
{target}")
4         return
5     tower_of_hanoi(n-1, source, auxiliary, target)
6     print(f"Move disk {n} from rod {source} to rod {target}")
7     tower_of_hanoi(n-1, auxiliary, target, source)
8
9 n = 3 # Number of disks
10 source_rod = 'A'
11 target_rod = 'C'
12 auxiliary_rod = 'B'
13
14 print(f"Solving Tower of Hanoi problem for {n} disks:")
15 tower_of_hanoi(n, source_rod, target_rod, auxiliary_rod)
```

Exercise: Can you propose another algorithm which uses the idea of divide-and-conquer?