DBSCAN

ESEN
École Supérieure d'Économie Numérique

Préparer par :

Maissa Othman

Chaima Sliti

Oumayma Rouahi

Plan :

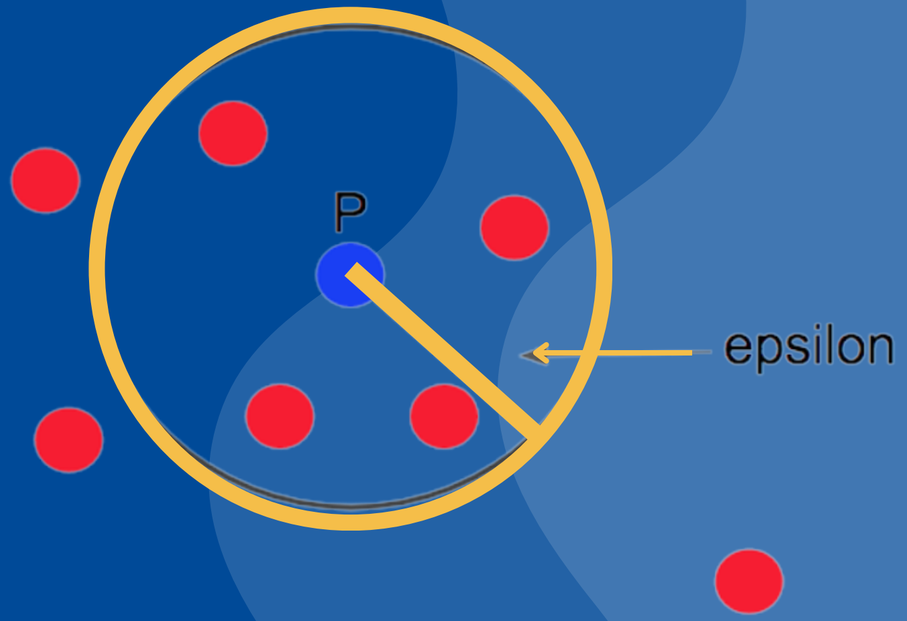
Définition

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

est un algorithme de regroupement qui identifie des clusters de points de données similaires en fonction de leur densité dans l'espace des données.

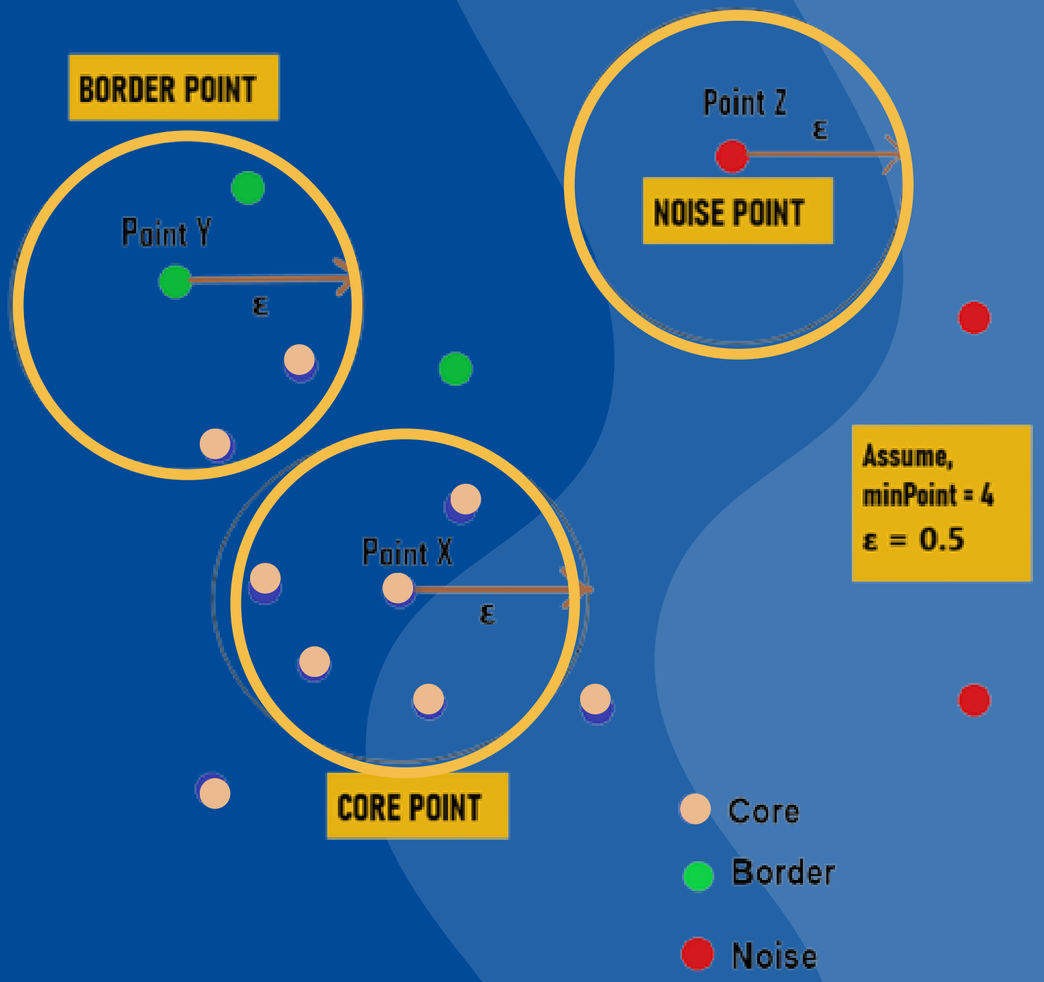
C'est quoi cette densité ?

La densité **= nombre de points dans un rayon déterminé (appelé epsilon)**
et qui est l'un des paramètres de cet algorithme

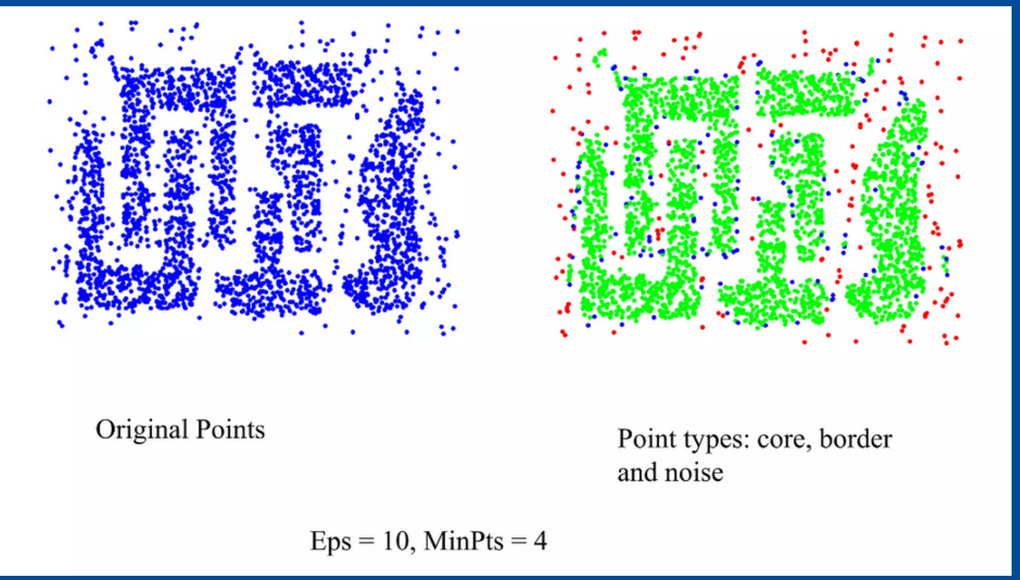


On a aussi comme 2éme paramètre :

minPts : Le nombre minimum de points requis pour former une région dense



Principe :



Le principe de base est de prendre un point de données et d'identifier tous les autres points de données dans son voisinage direct, c'est-à-dire dans le rayon epsilon autour de ce point. Si le nombre de points dans ce voisinage direct est supérieur ou égal à minPts, alors un cluster est formé. Si le nombre de points dans le voisinage direct est inférieur à minPts, alors ce point est considéré comme du bruit (noise).

L'algorithme continue à explorer l'espace des données de cette manière, en identifiant les points qui appartiennent à des clusters et ceux qui sont du bruit, jusqu'à ce que tous les points de données soient couverts.

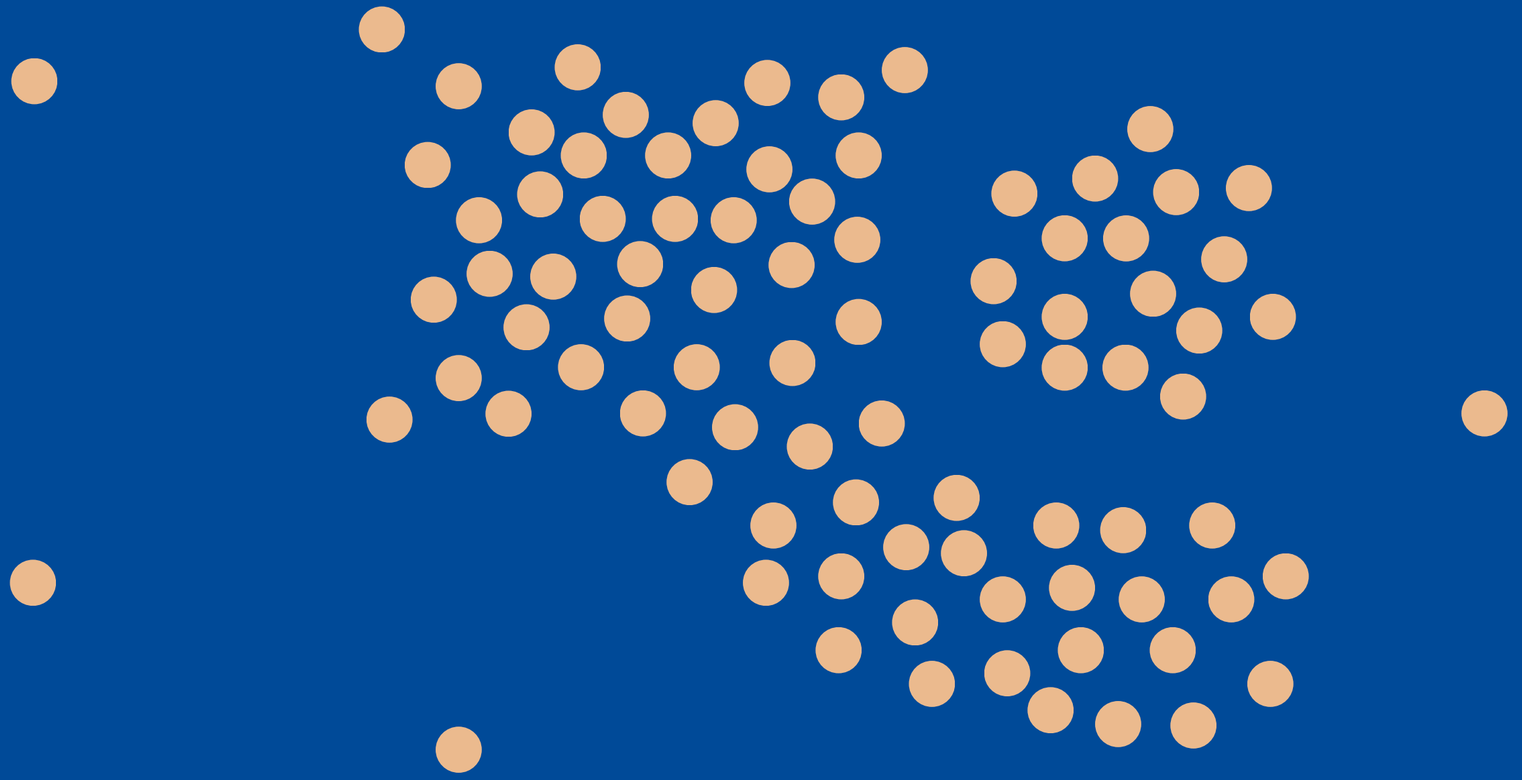
Vous avez donc besoin de définir deux informations avant d'utiliser le DBSCAN :

- Quelle distance ε pour déterminer pour chaque observation le ε-voisinage ?

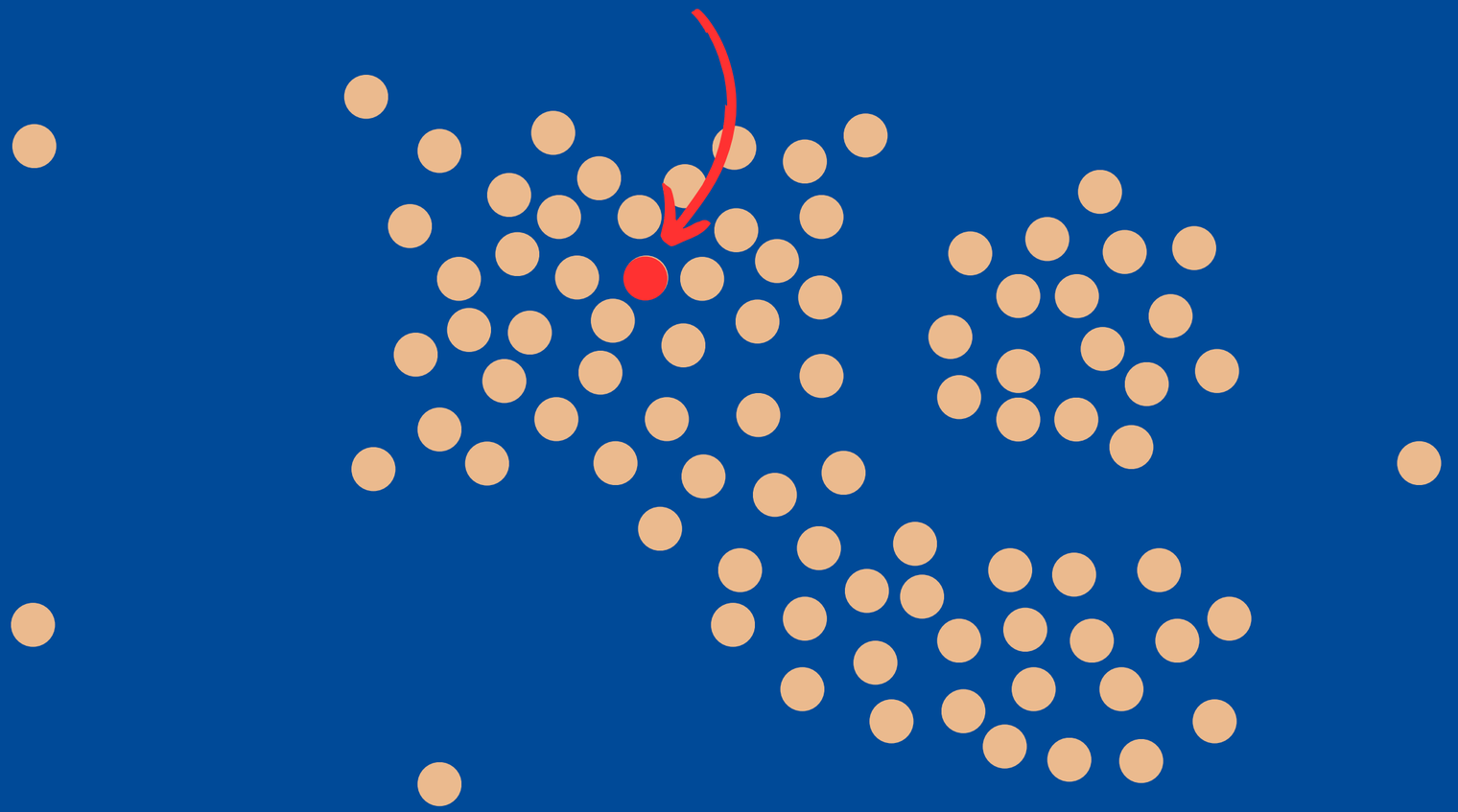
- Quel est le nombre minimal de voisins nécessaire pour considérer qu'une observation est une observation cœur ?

EXEMPLE

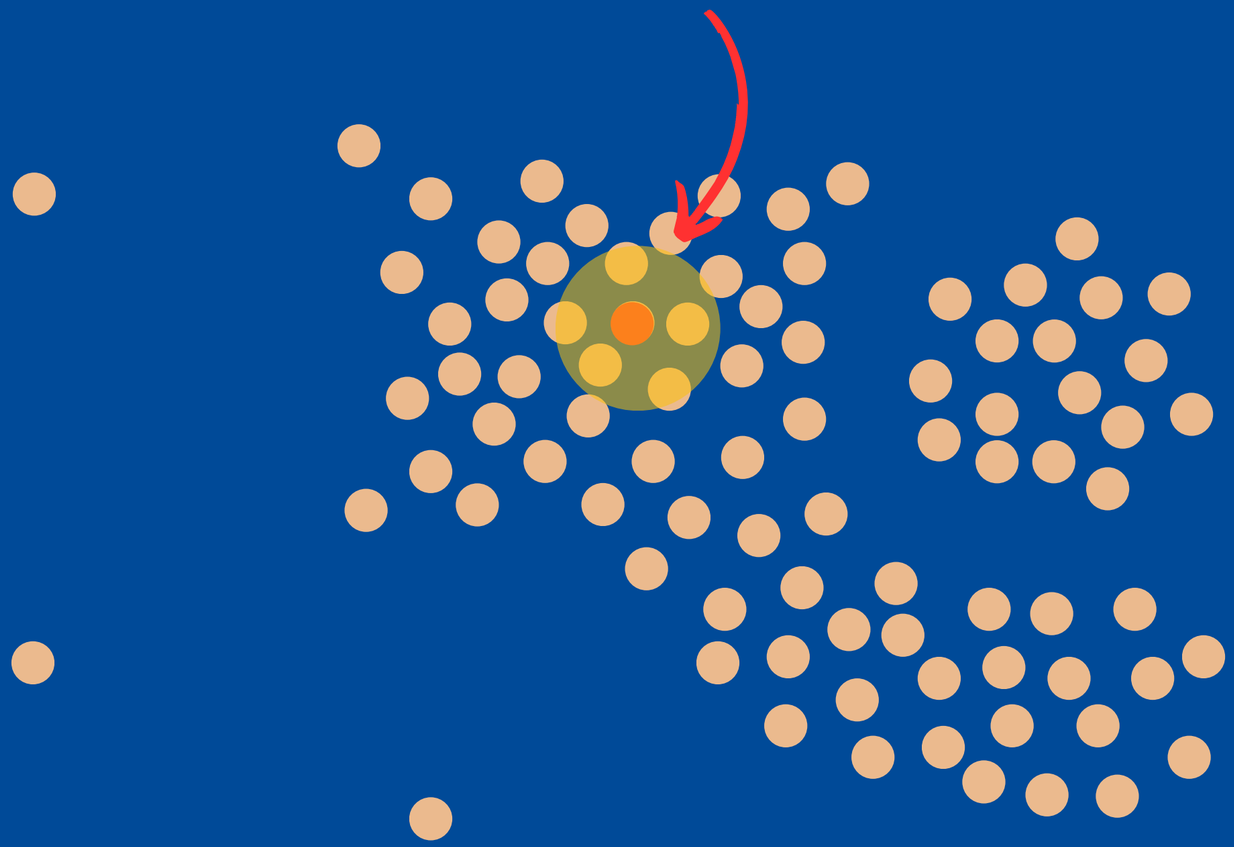
1) La première chose à faire est de compter le nombre de points proches de chaque point.



par exemple, on peut commencer par **cet point** :



et on fait **un cercle** autour de cet point

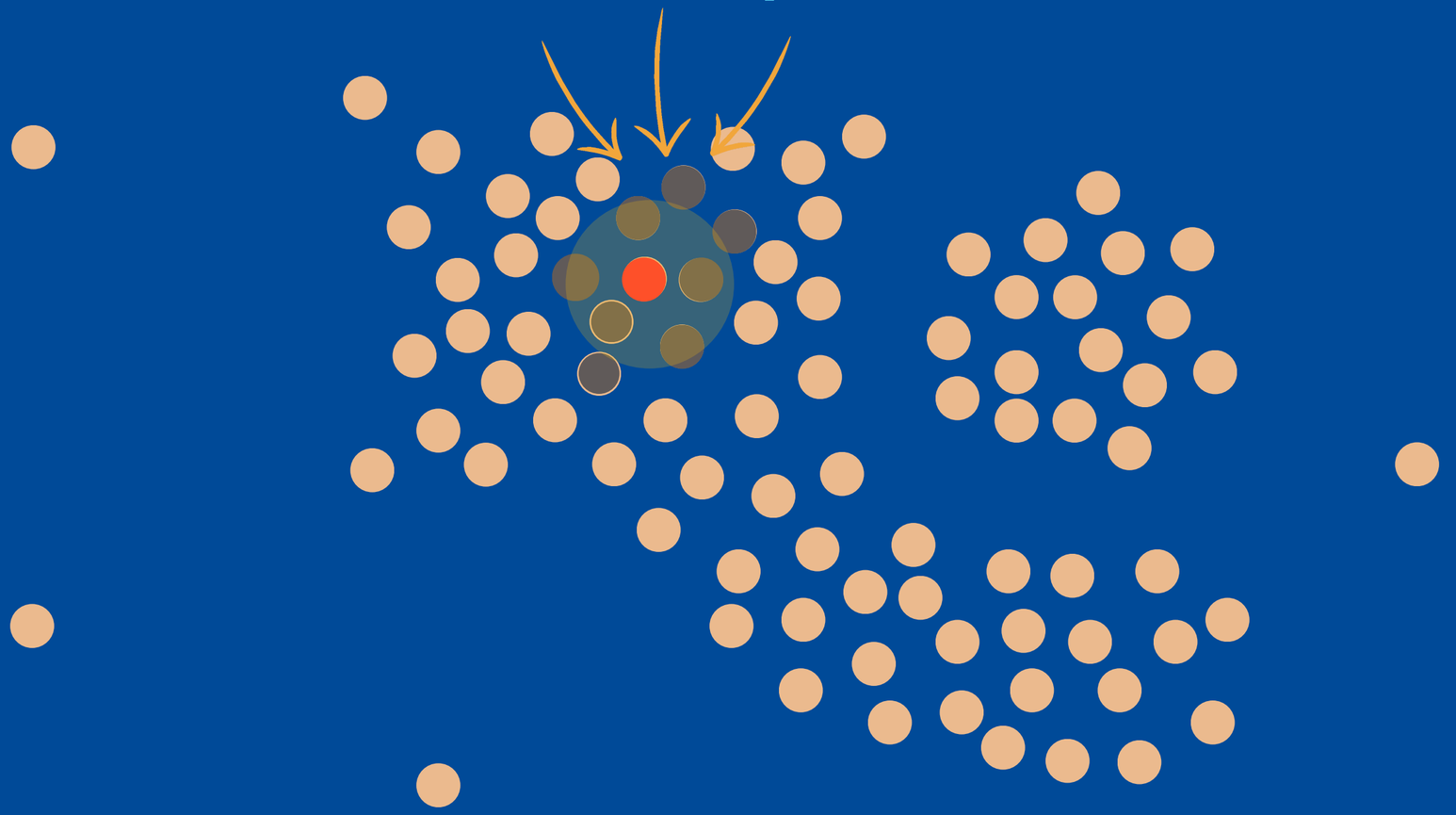


!!

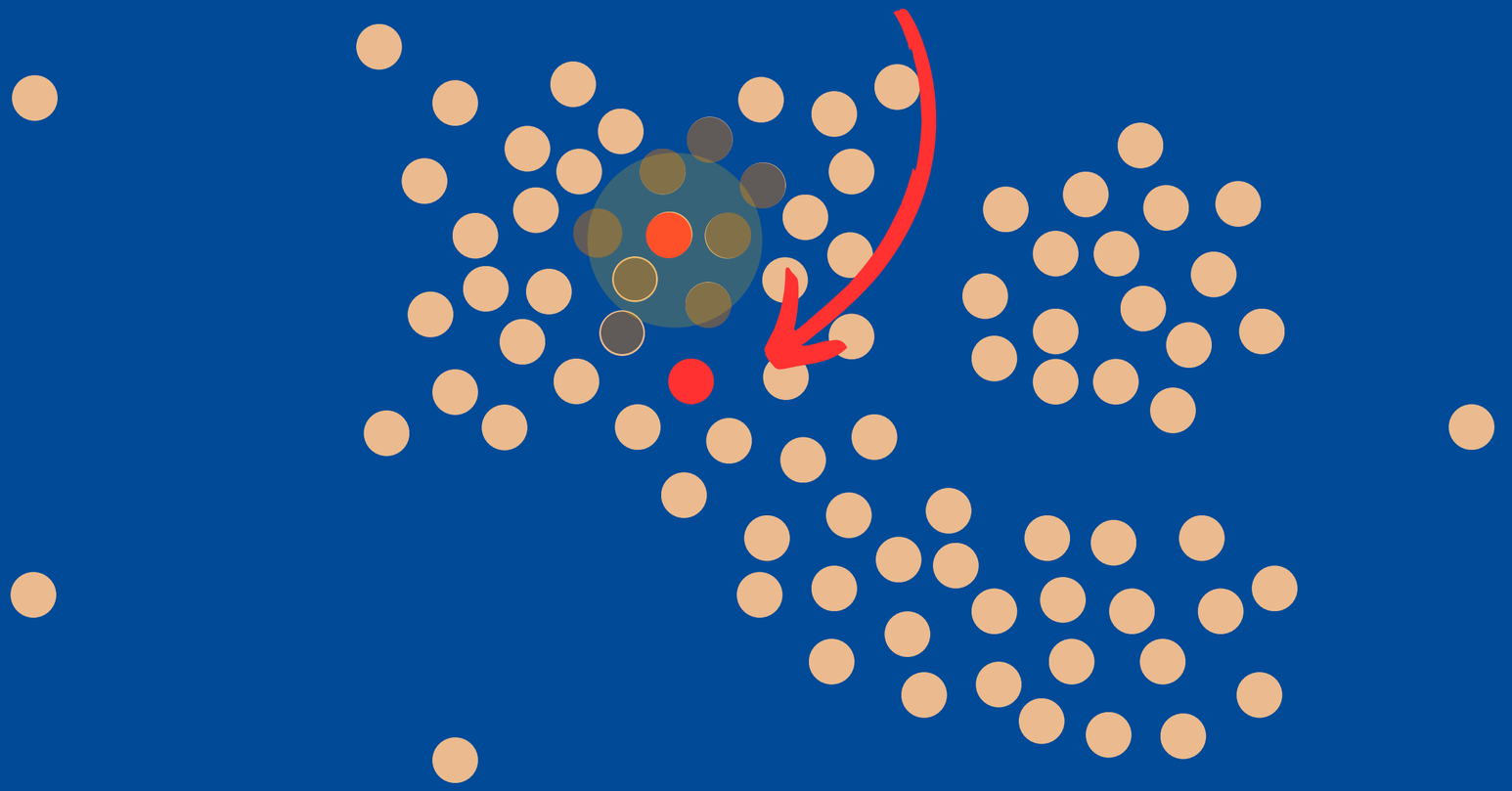
- on NOTE que **le rayon** du cercle jaune **est défini par l'utilisateur**, donc lors de l'utilisation de DBSCAN, vous devrez peut-être jouer avec ce paramètre.

Après on remarque que le cercle recouvre, au moins partiellement,

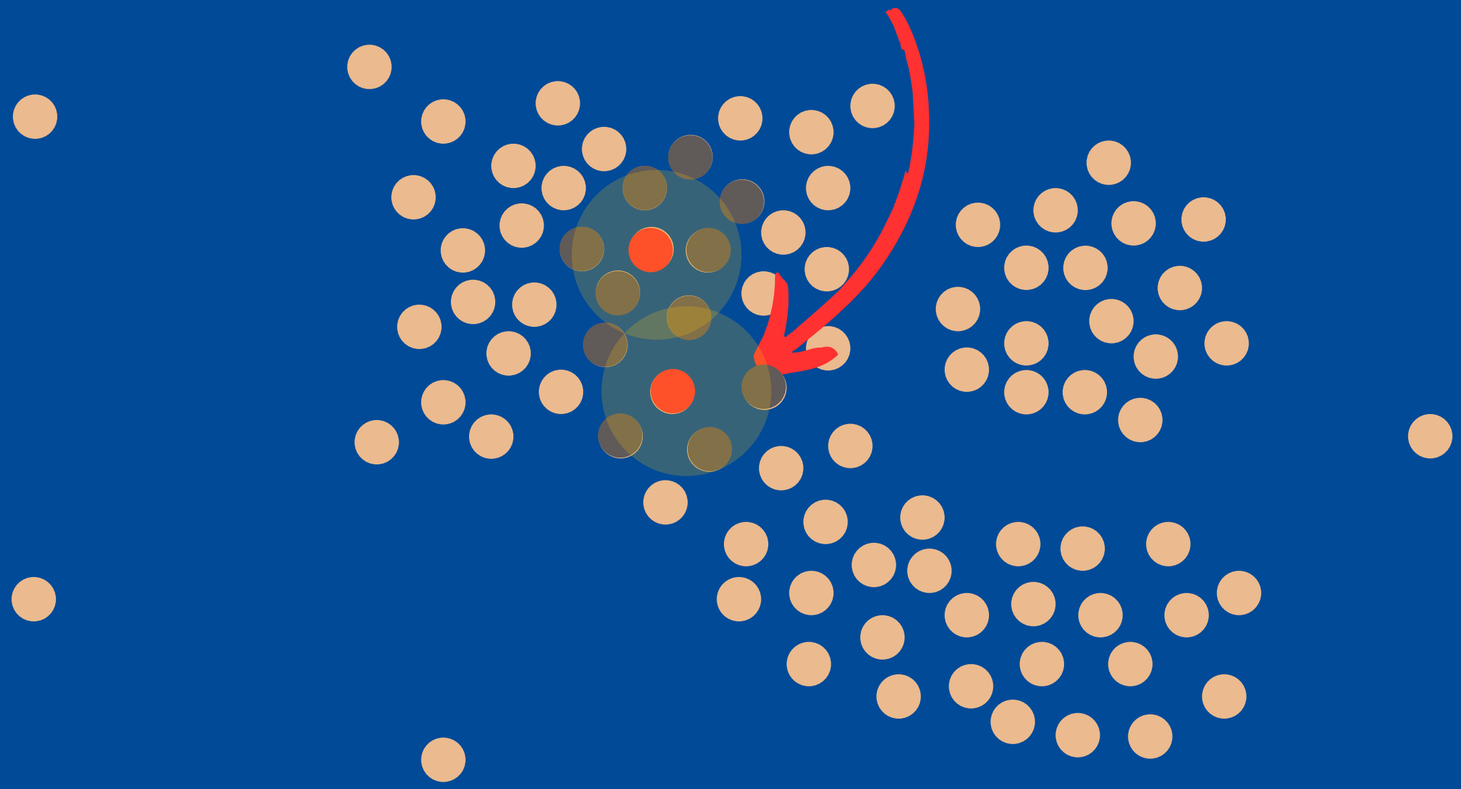
8 autres points



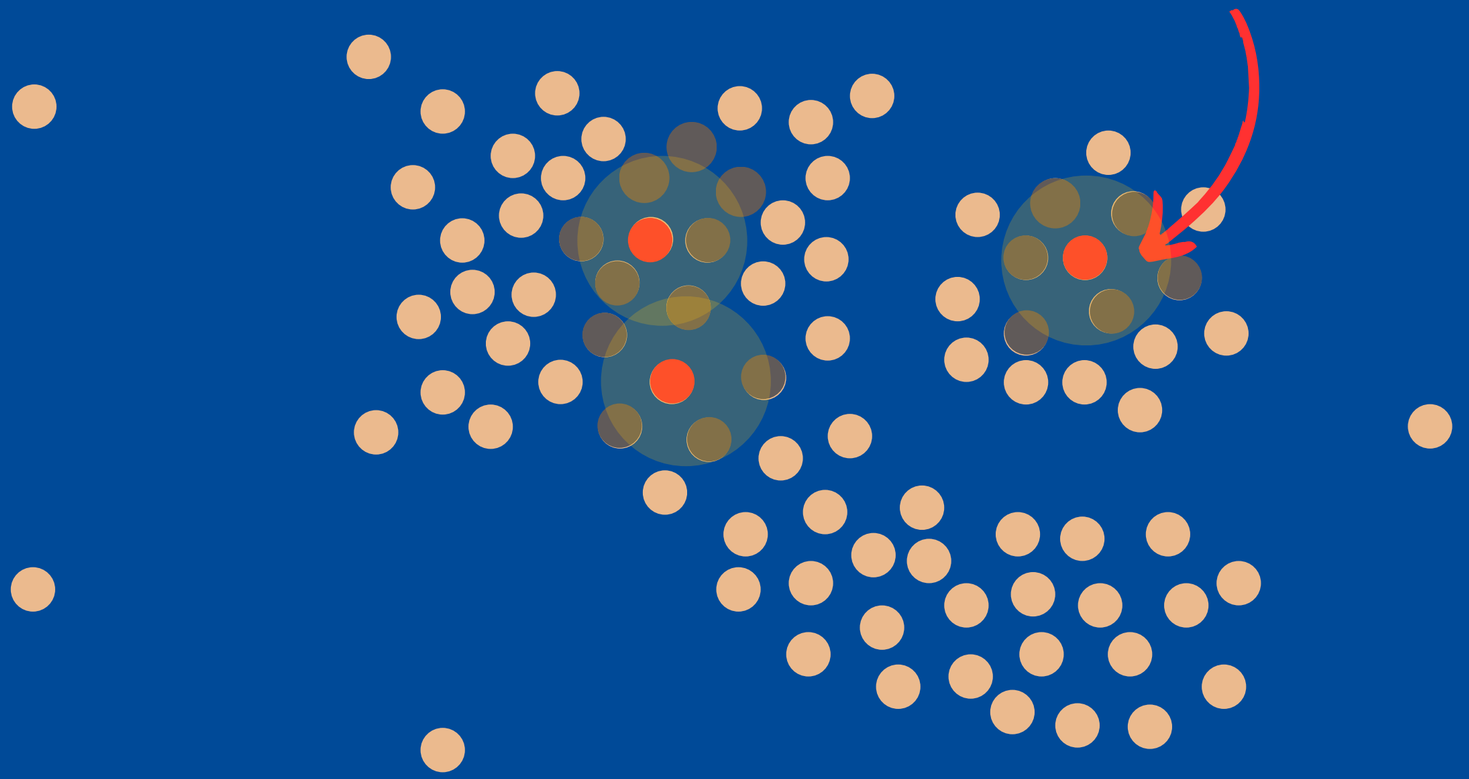
Après on prend un autre point et on recommence ……



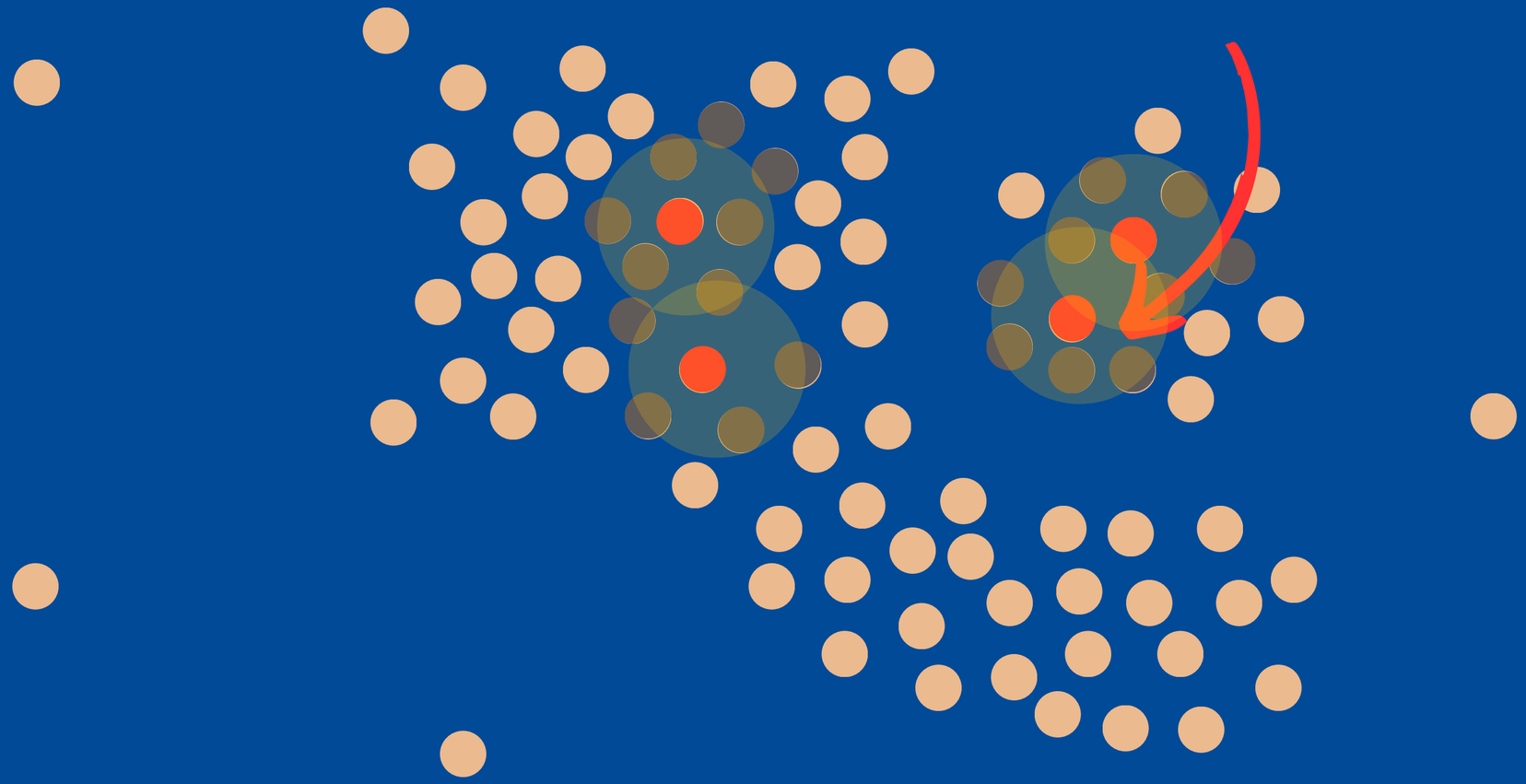
….. est proche de 5 autres points car le cercle jaune recouvre, au moins partiellement, **5 autres points**.



Puis on reprend le même démarche et on prend un autre point …
il est proche de 6 autres points…

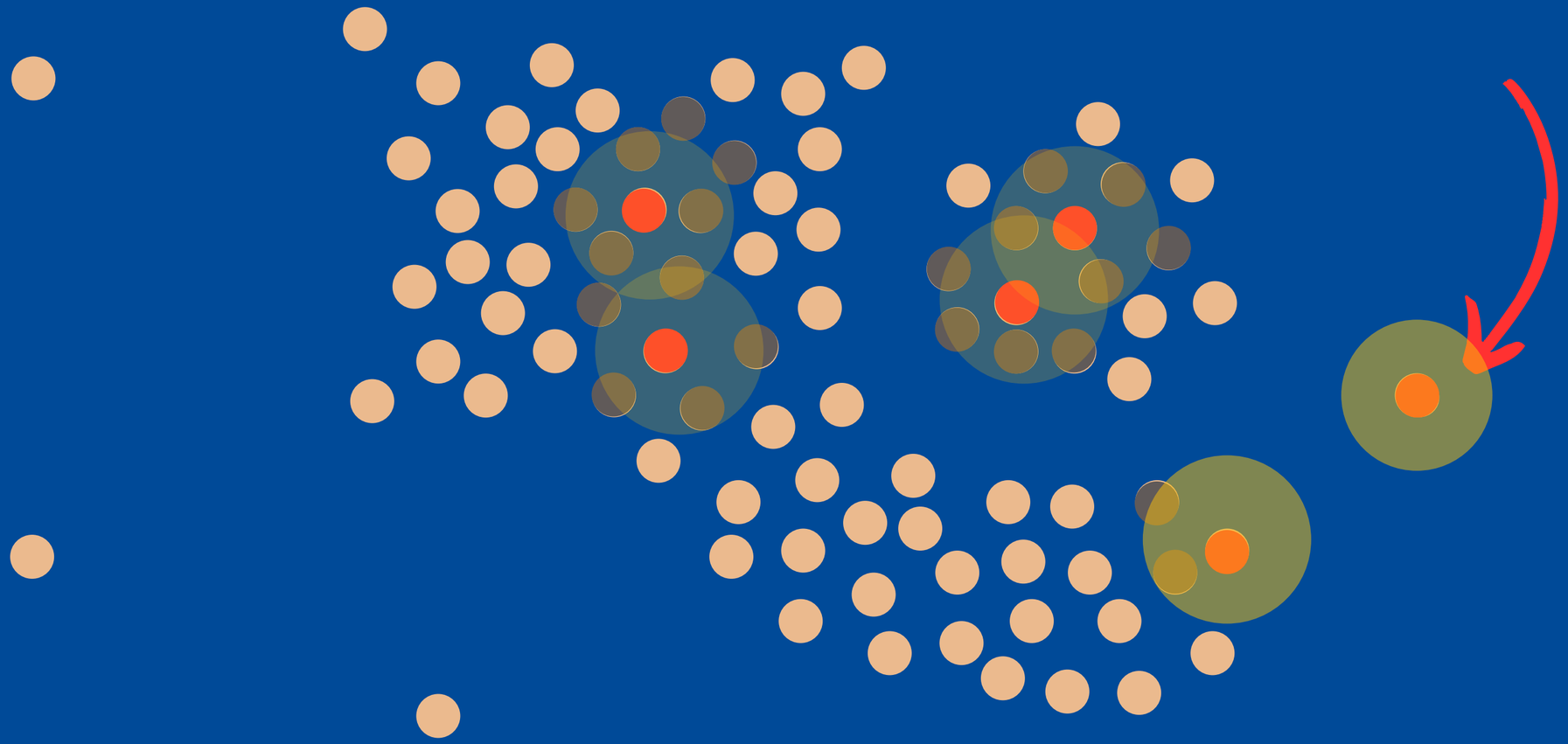


..... et ce point rouge est proche de **7 autres points**.

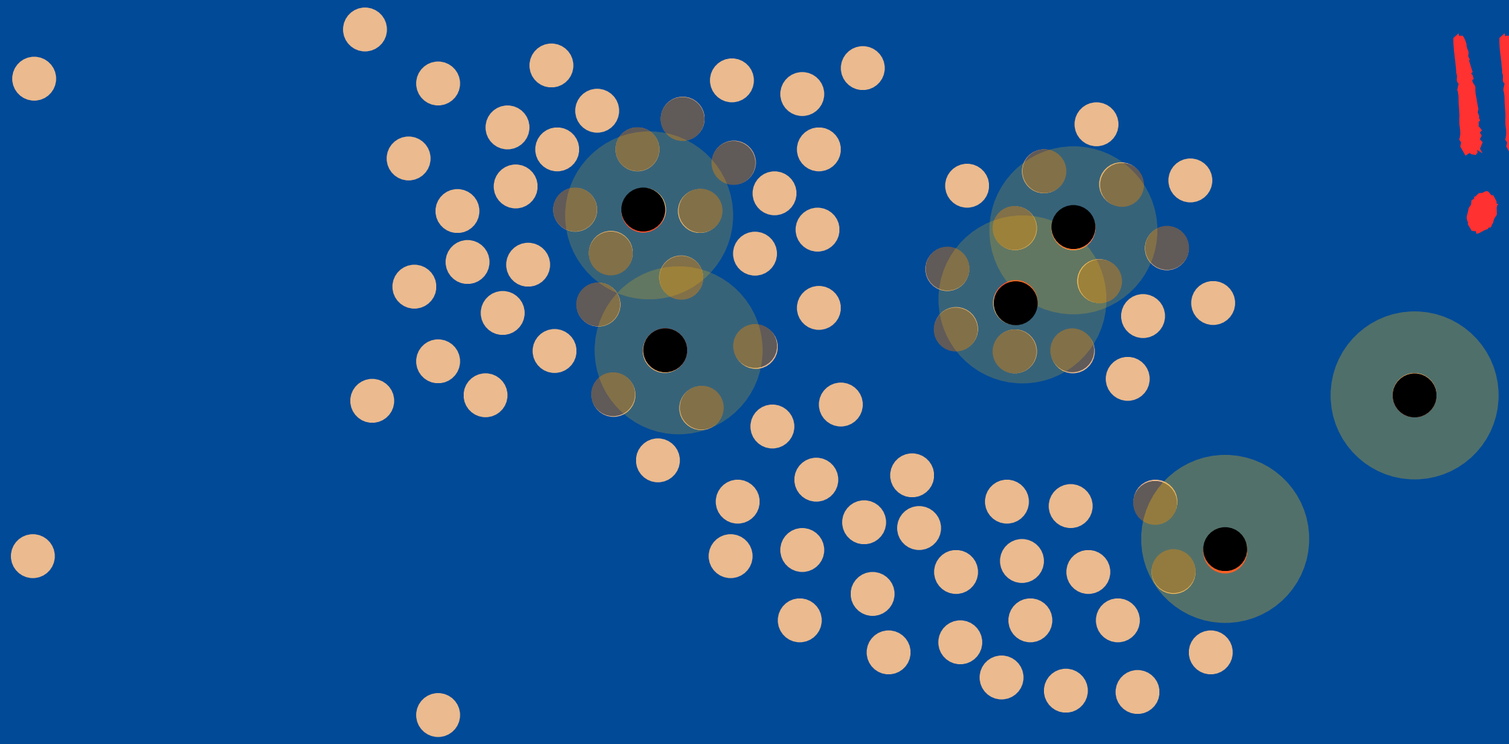


ce point rouge n'est proche que de **2 autres points**...

ce point rouge n'est proche d'**aucun autre point** car le cercle jaune ne recouvre rien d'autre...

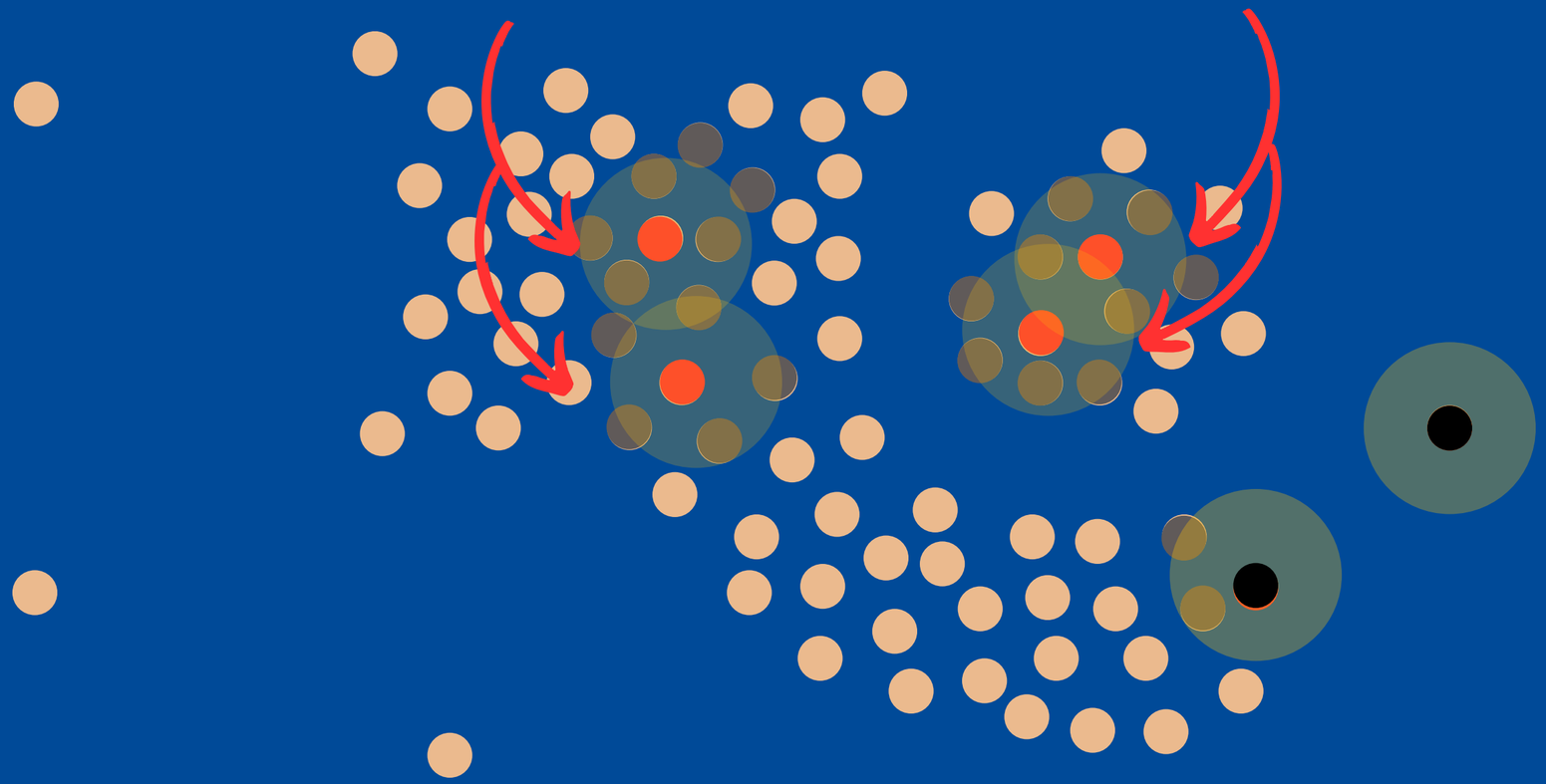


Maintenant pour cet exemple, nous définirons les **Core points** comme étant proche d'**au moins 4 autres points**

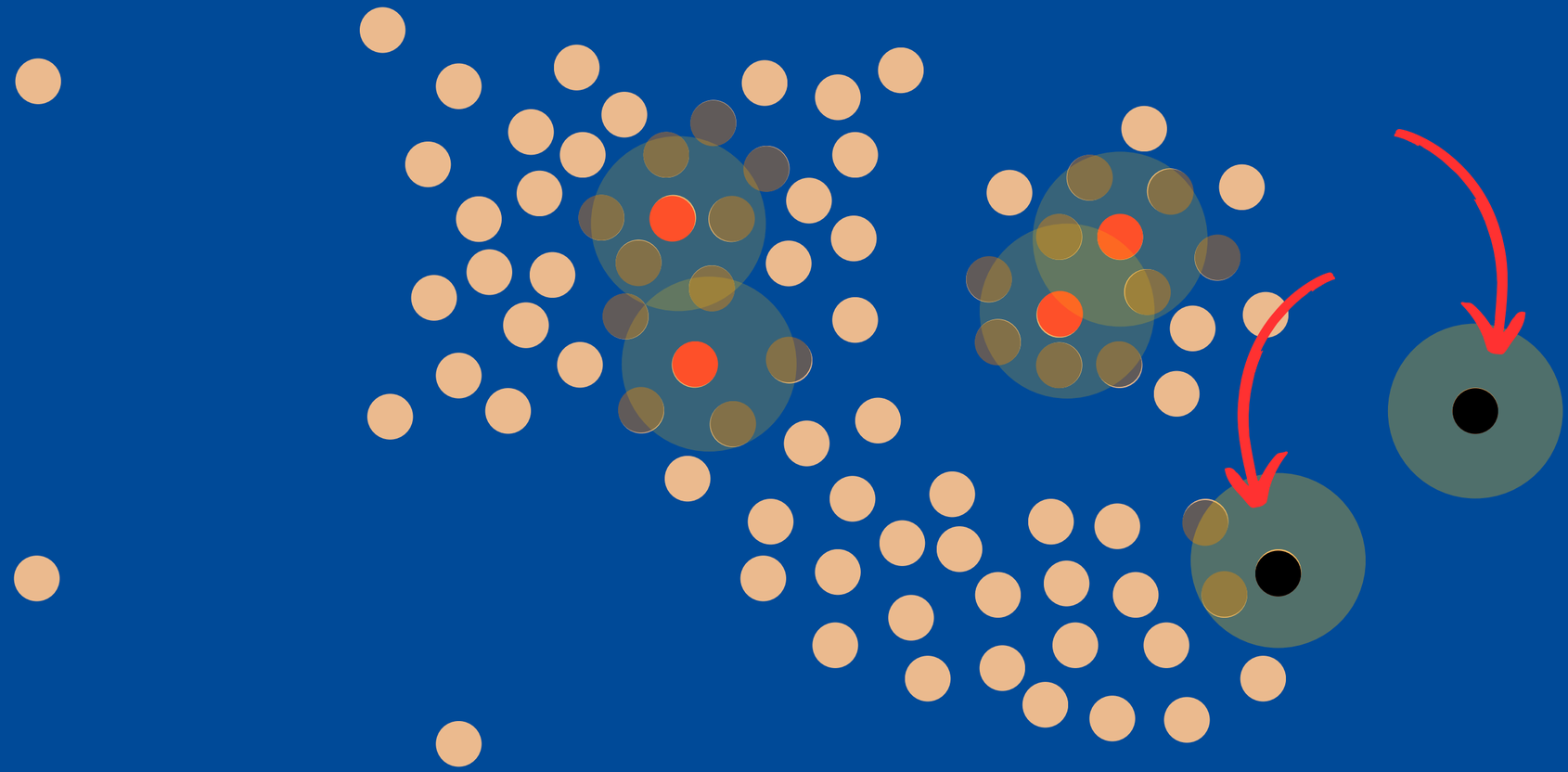


!! on NOTE que **le nombre de points minimum est défini par l'utilisateur**, donc lors de l'utilisation de DBSCAN, vous devrez peut-être jouer avec ce paramètre.

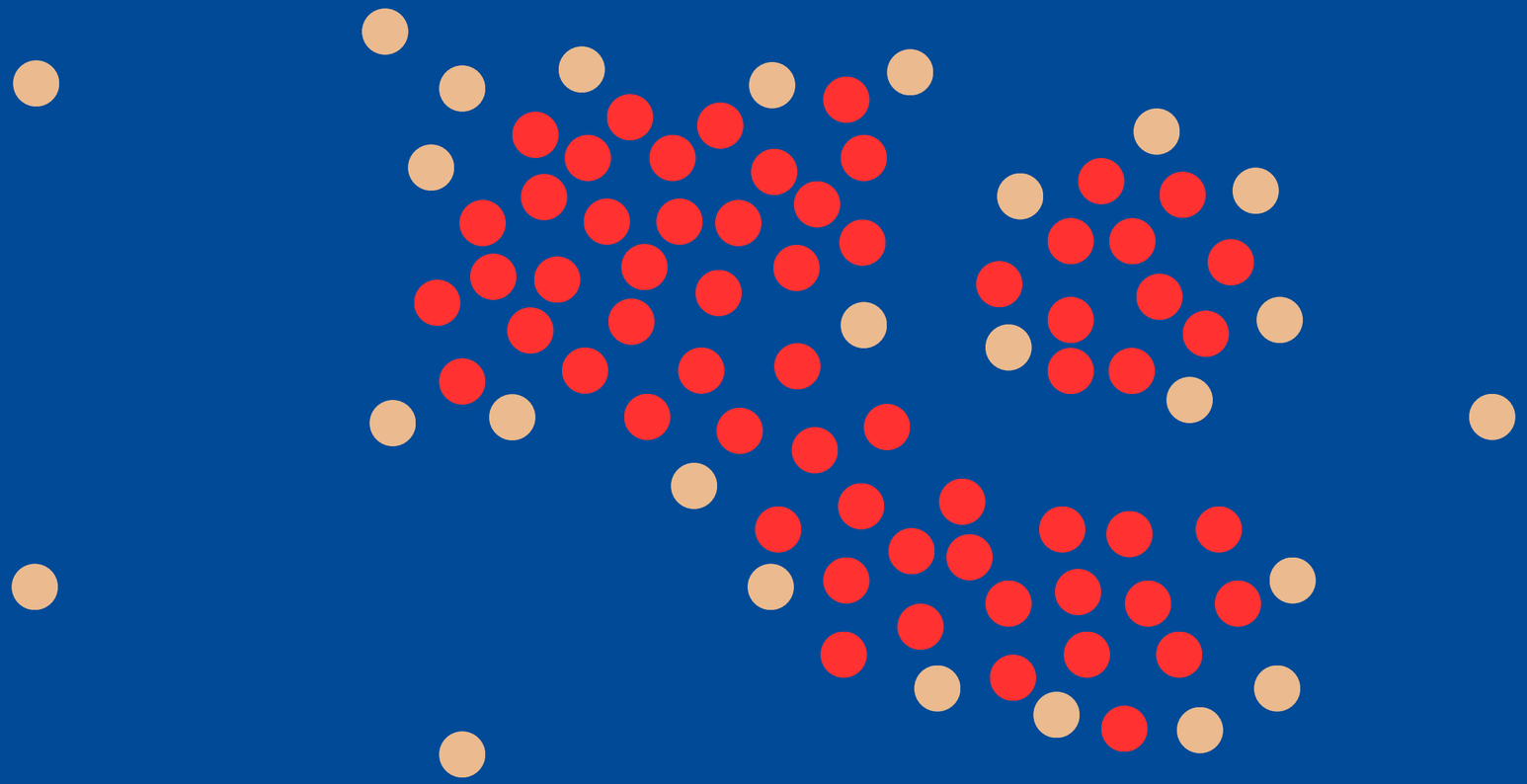
Donc ces 4 points font partie des **Core Points**, car leurs cercles jaunes chevauchent au moins 4 autres points ...



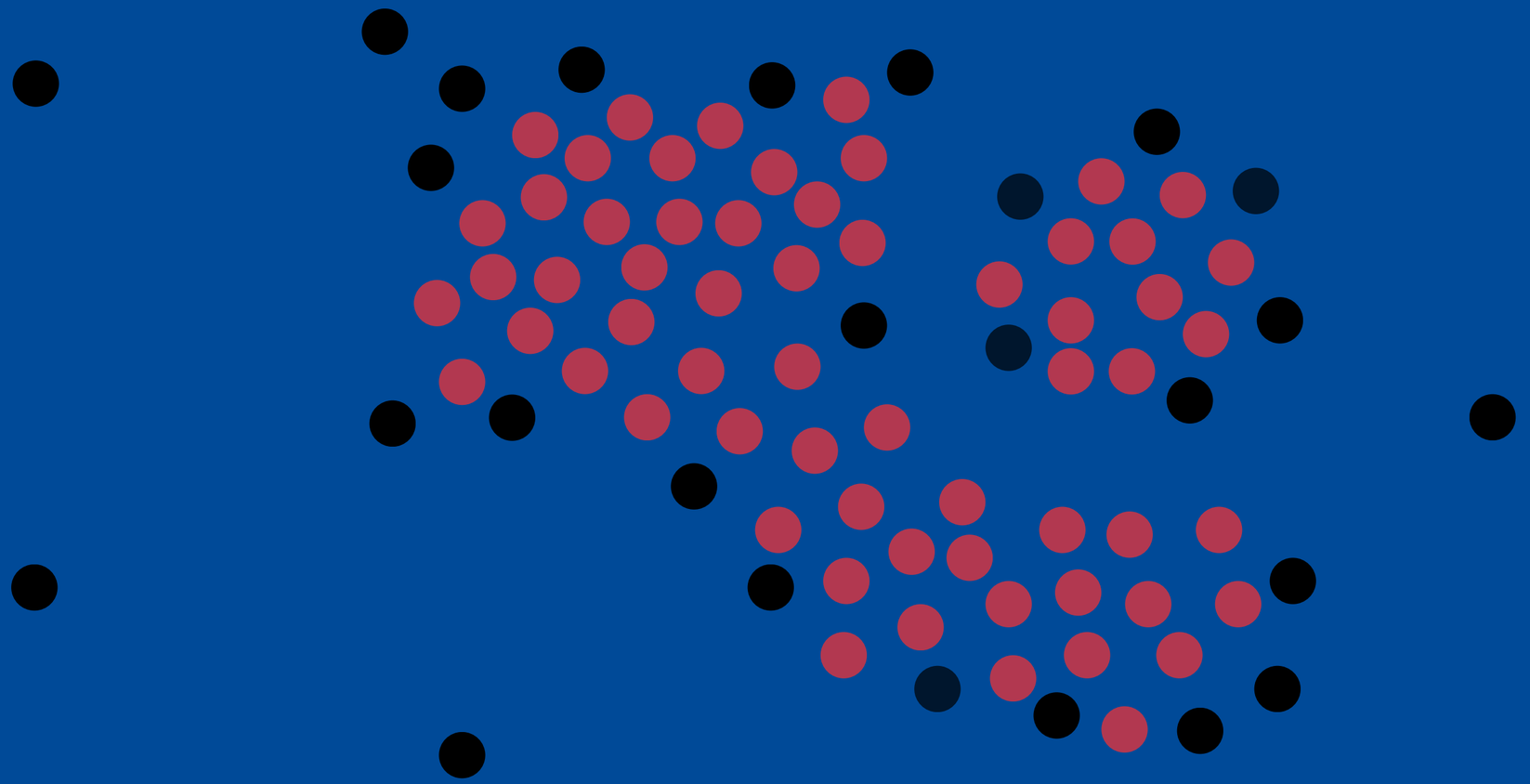
mais aucun de ces points **n'est Core point** car leurs cercles jaunes ne chevauchent pas 4 autres points ou plus.



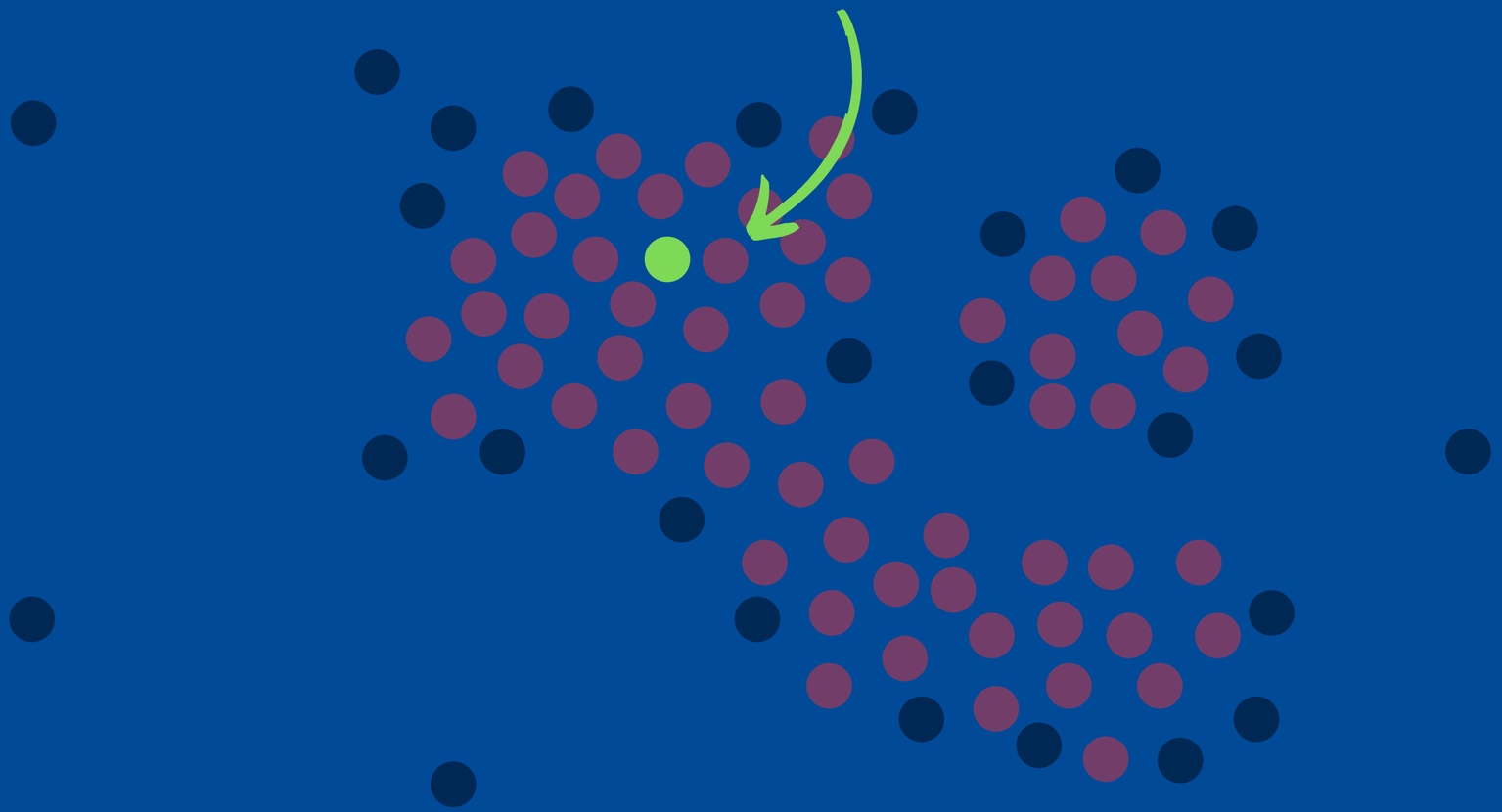
en définitive, on peut tous ces points rouges **Core Points** car ils sont tous proches de 4 ou plus d'autres points...



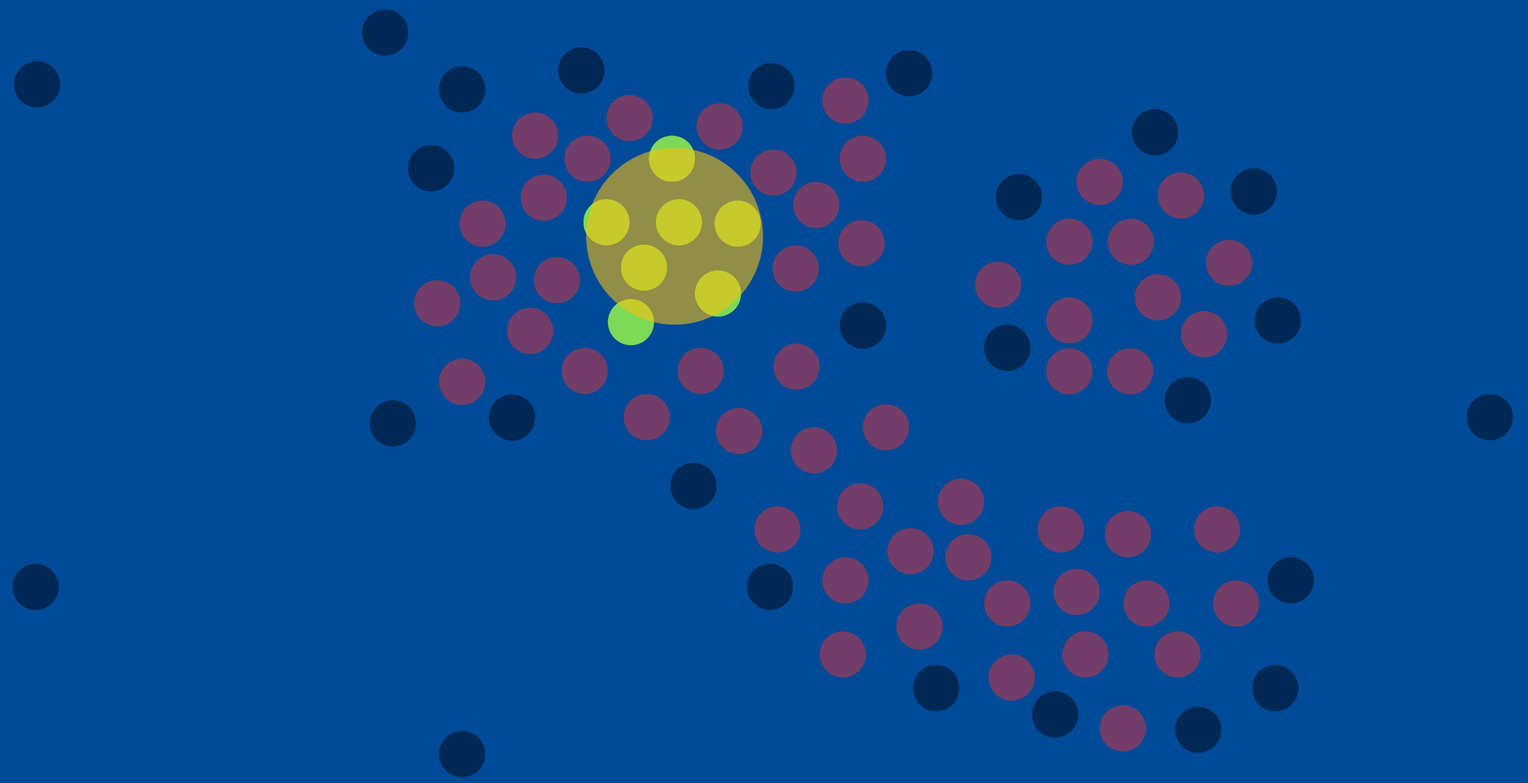
… et les autres sont **Non-Core.**



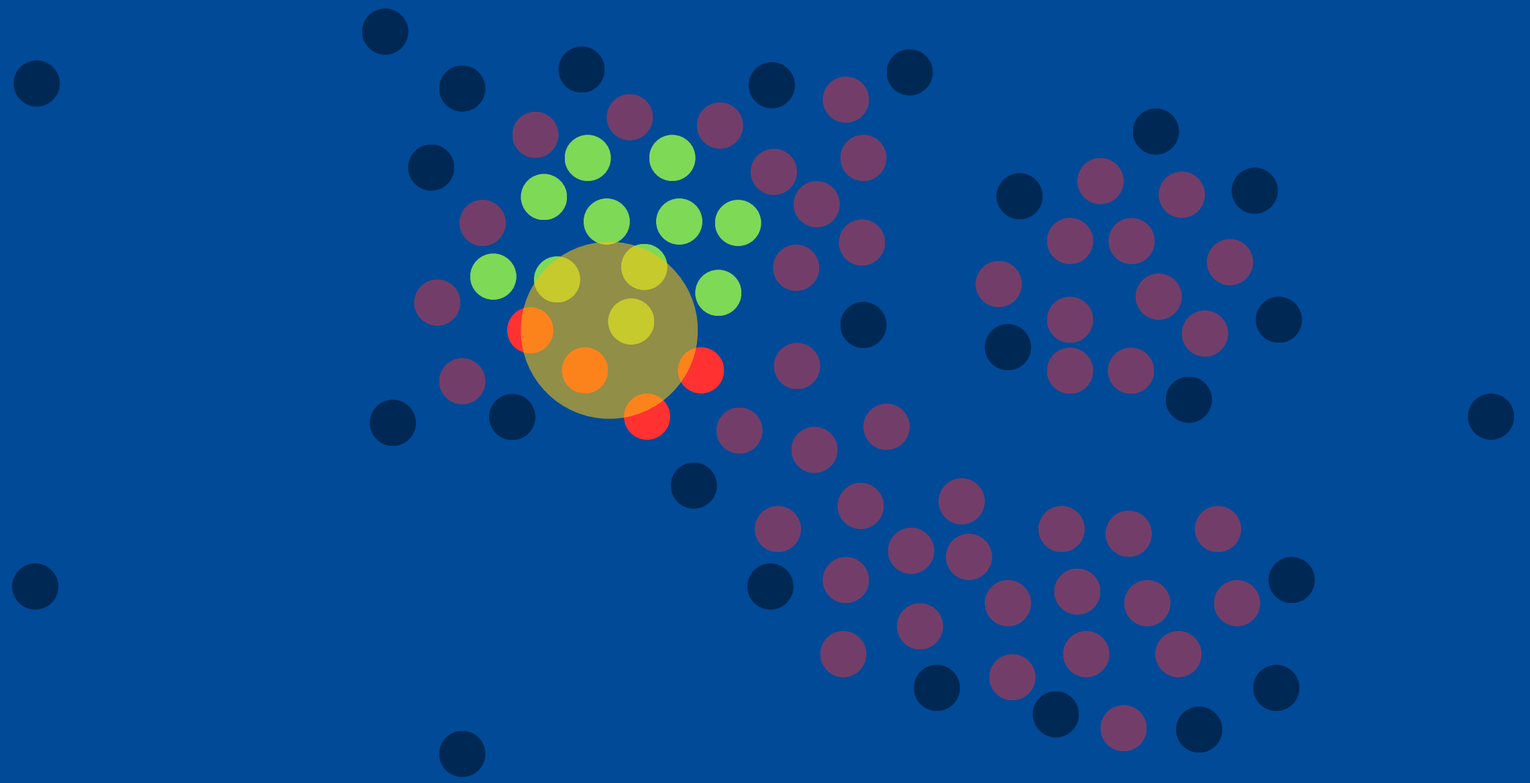
2) on va construire les clusters … donc on choisit aléatoirement un Core point et on l'ajoute au **premier Cluster**.



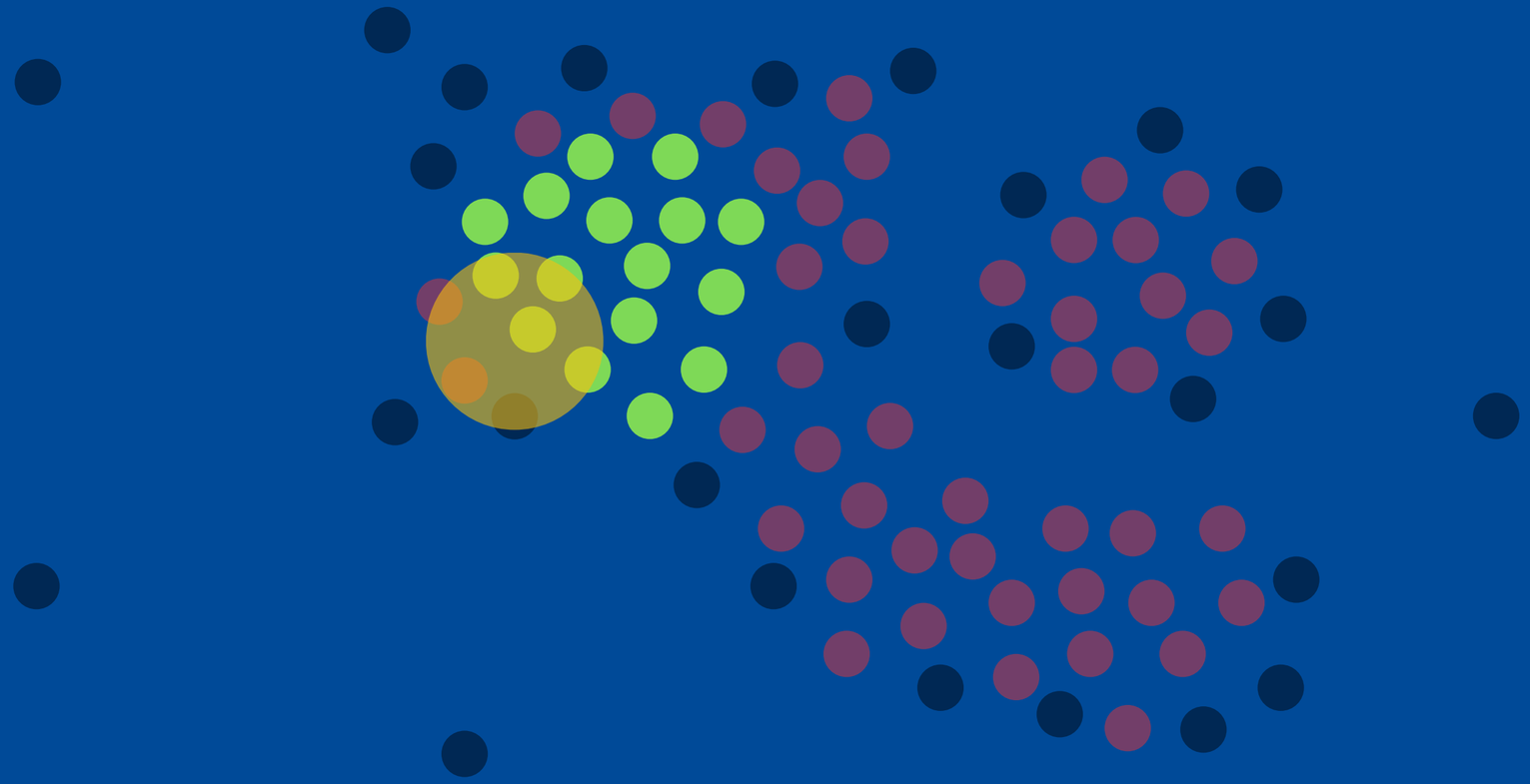
Après les Cores Points qui sont proches du premier cluster, c'est-à-dire qu'ils chevauchent le cercle seront ajouter au premier cluster.



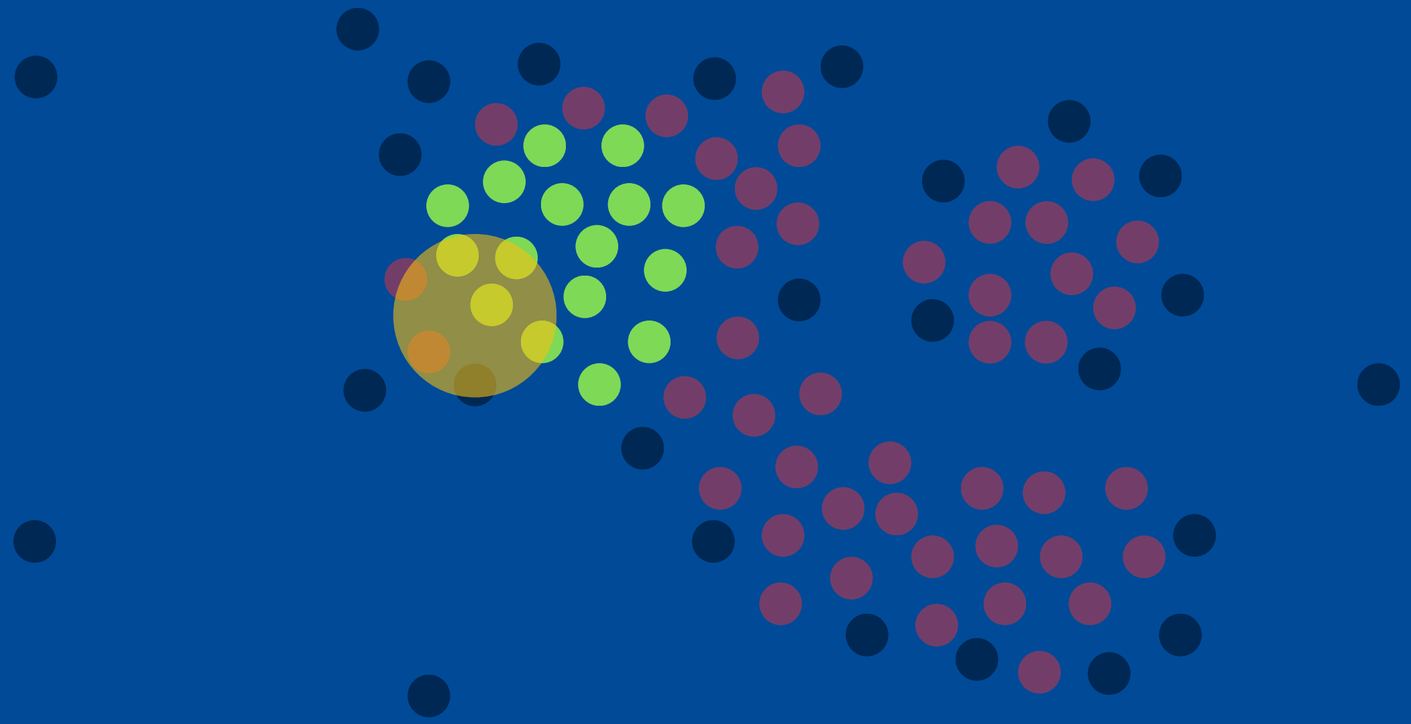
Ensuite, les Core points **proches** du premier cluster en croissance le rejoignent et l'étendent à d'autres Core points proches.



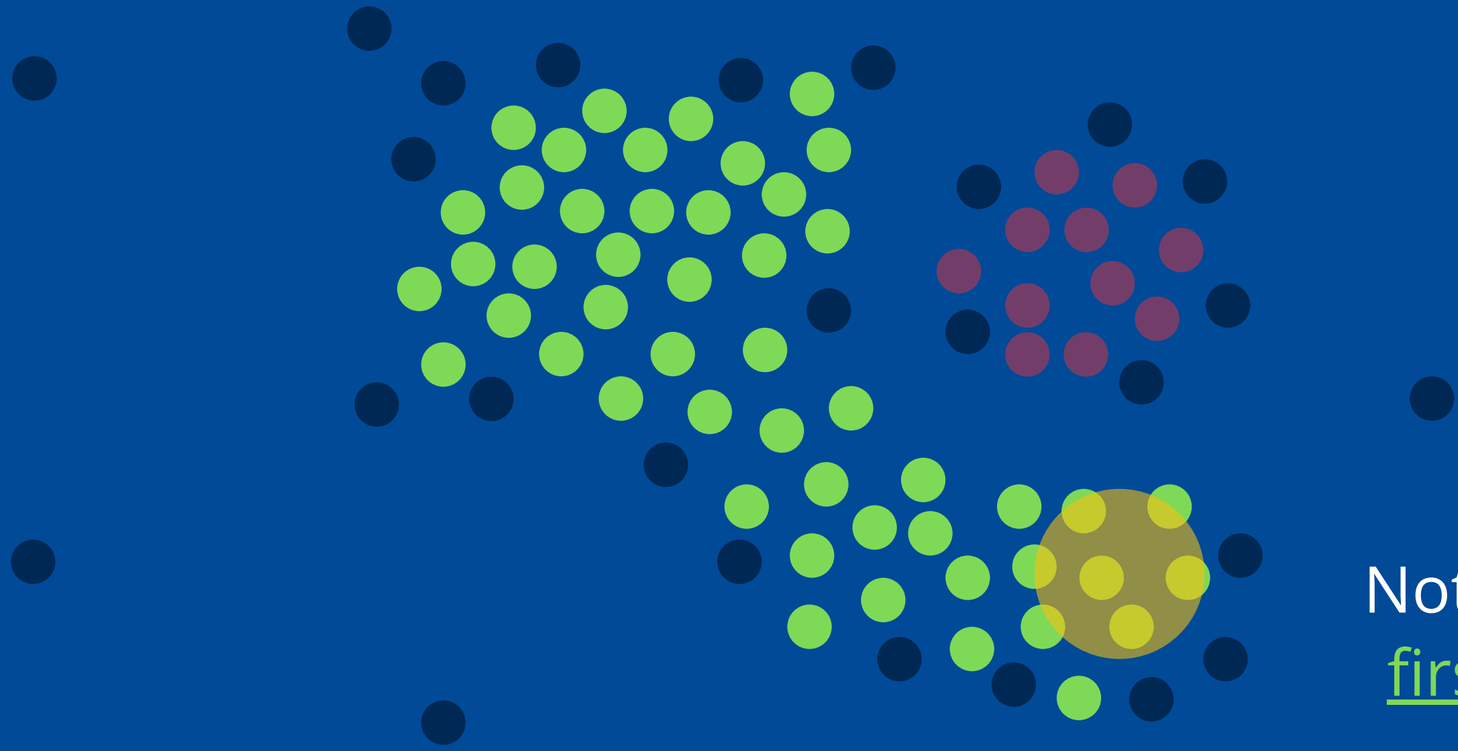
Ici on remarqu'on a **2 Core Points** **et 1 Non-Core Point** qui sont proches du premier cluster donc à ce point **on ajoute uniquement les Core Points.**



Ici on remarqu'on a **2 Core Points** **et 1 Non-Core Point** qui sont proches du premier cluster donc à ce point **on ajoute uniquement les Core Points** mais on va les ajouter ultérieurement.

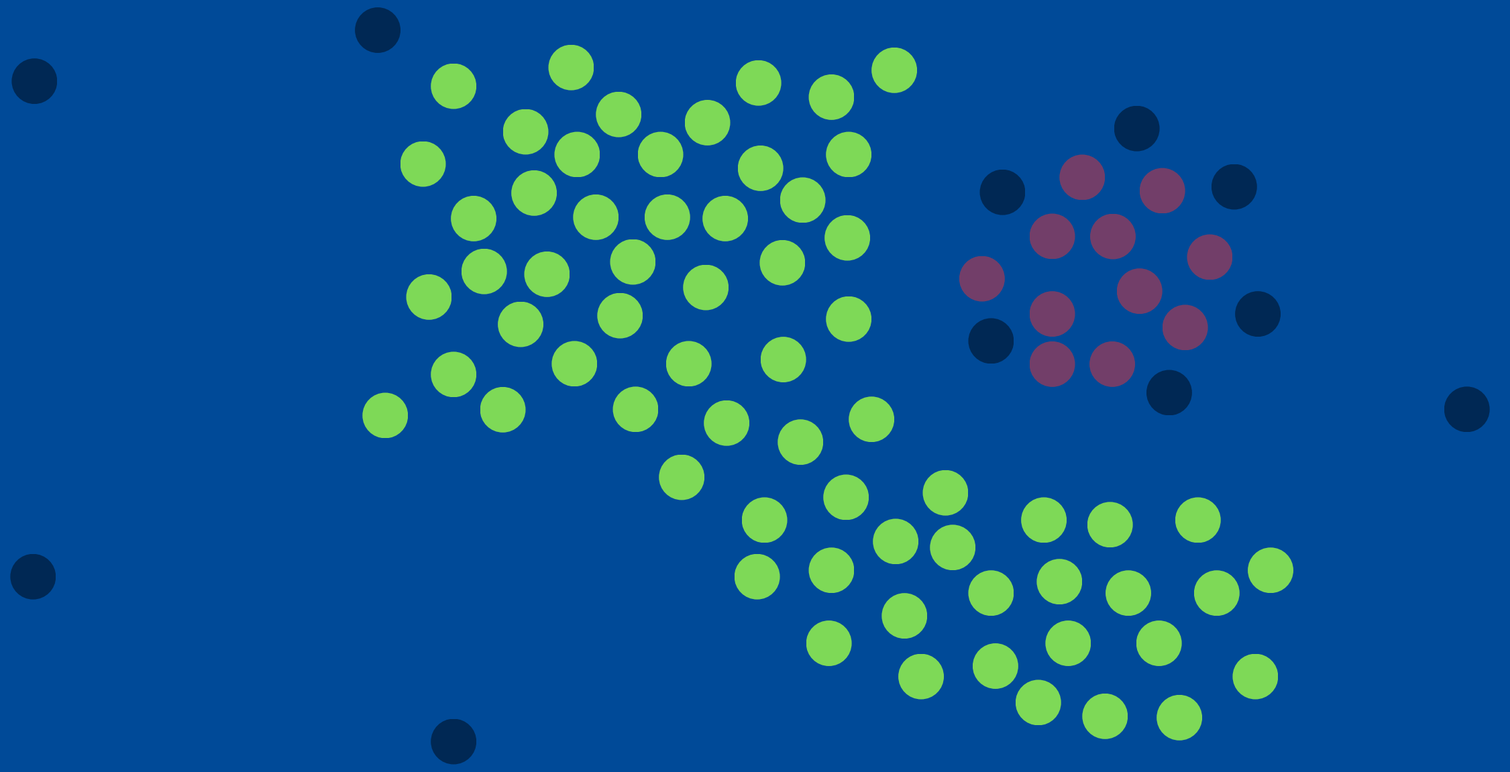


En fin de compte, tous les Core points proches du premier cluster en croissance y sont ajoutés, puis utilisés pour l'étendre davantage.

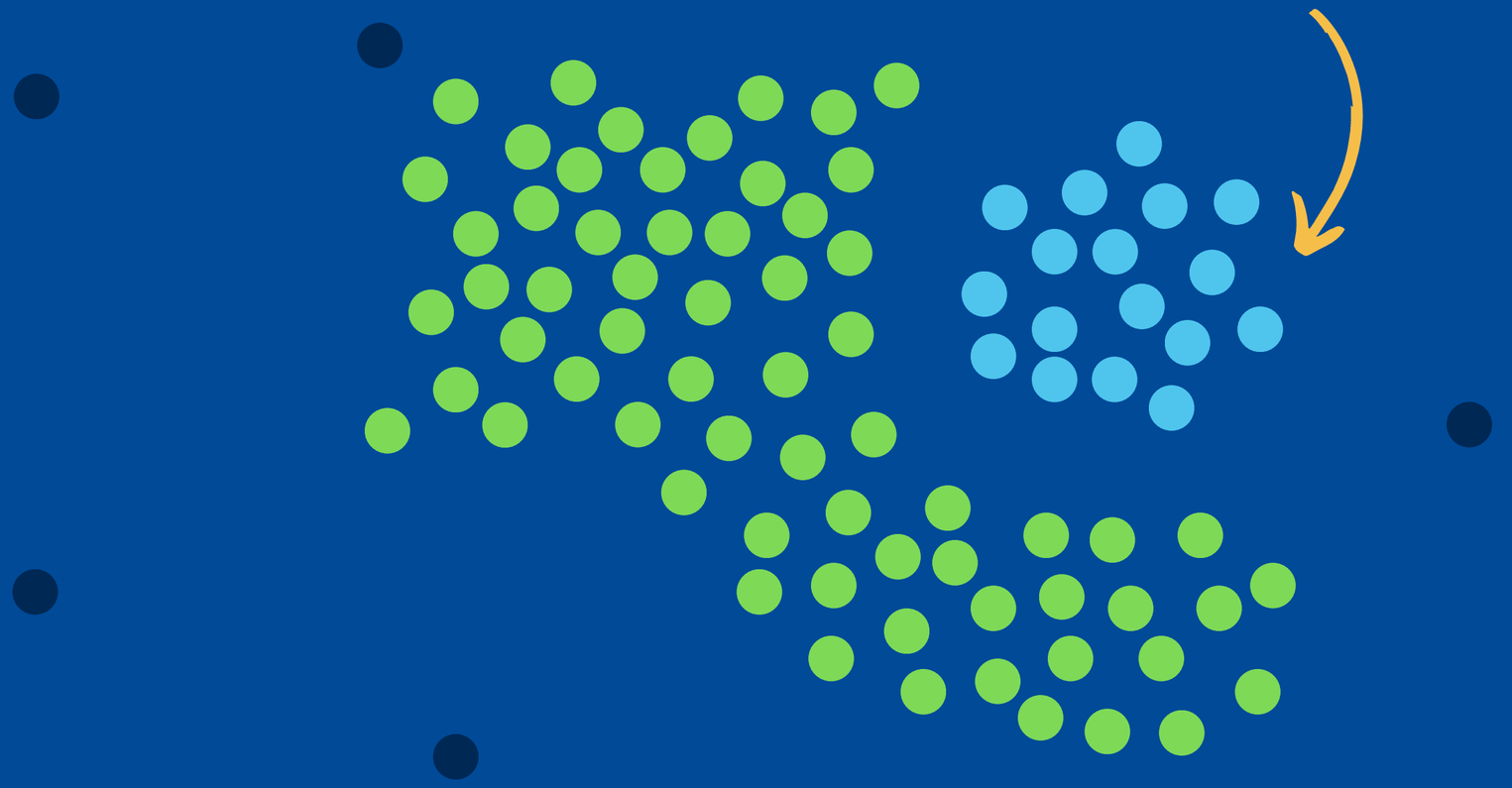


Note: A ce point tout les points du first cluster sont des Core Points

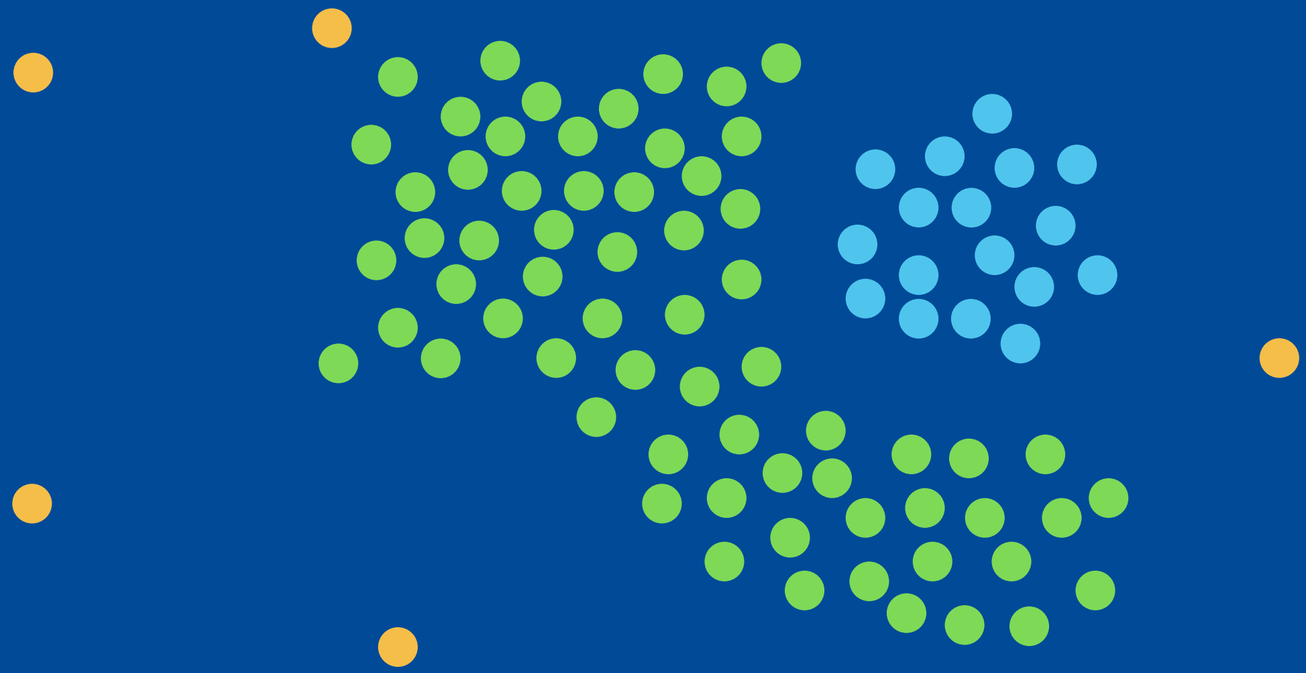
Et maintenant **on ajoute** tout **les Non-Core points** qui sont proches des Core points du premier Cluster au **premier Cluster**. Et comme ça on obtient notre premier Cluster.



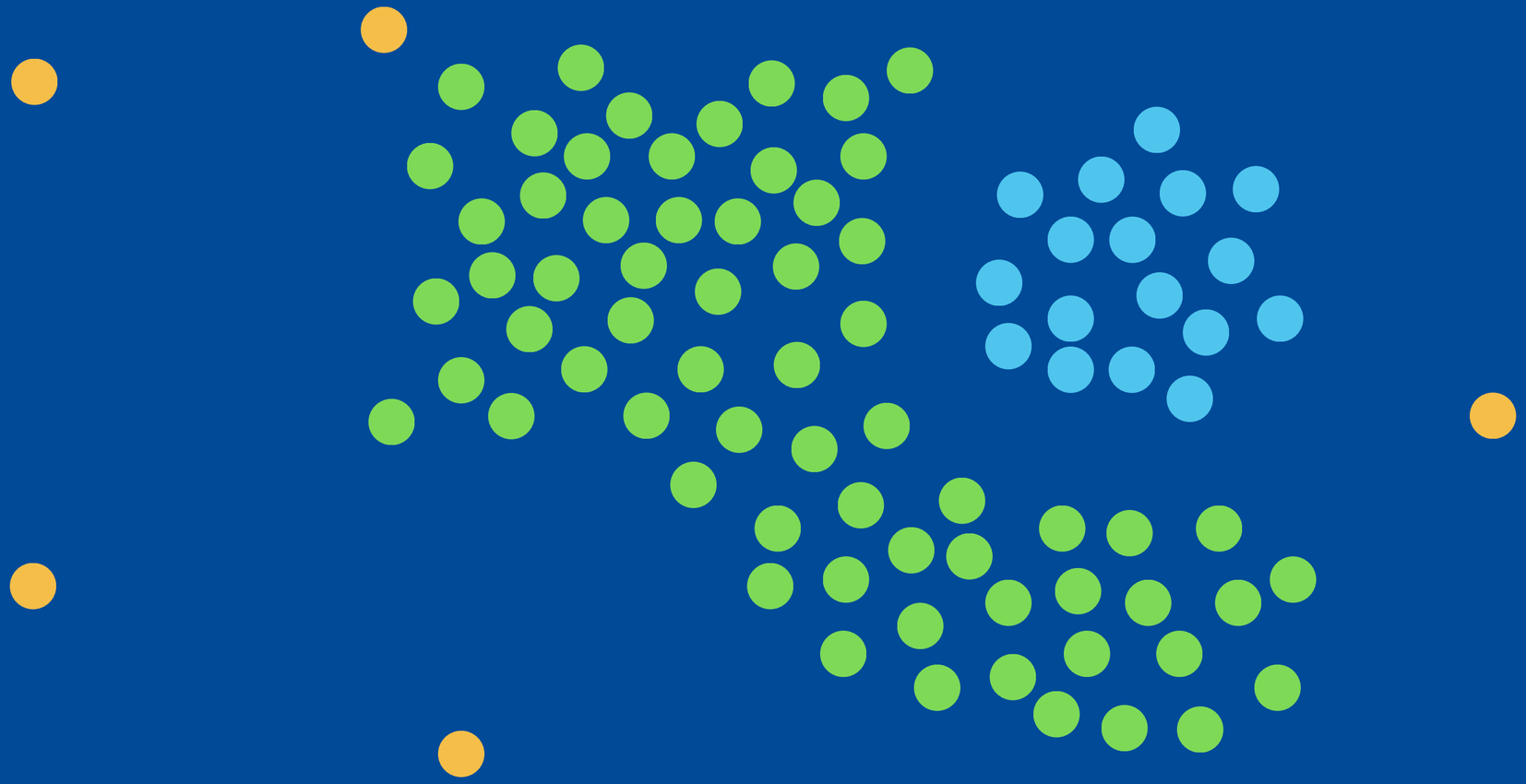
Puisque aucune de ces Core points n'est proche du premier Cluster , ils forment un **nouveau second Cluster** car ils sont proches les uns des autres.



Enfin, comme tous **les Core points** ont été attribués à un cluster, nous avons fini de créer de nouveaux clusters et tous les points **Non-Core** restants qui ne sont pas proches des Core points dans l'un ou l'autre des clusters ne sont pas ajoutés aux clusters et appelés **Noise points (valeurs aberrantes ou outliers)**.



et c'est ainsi que fonctionne l'algorithme DBSCAN !



Comment choisir epsilon ε ?

Dans cet algorithme deux points sont clés :

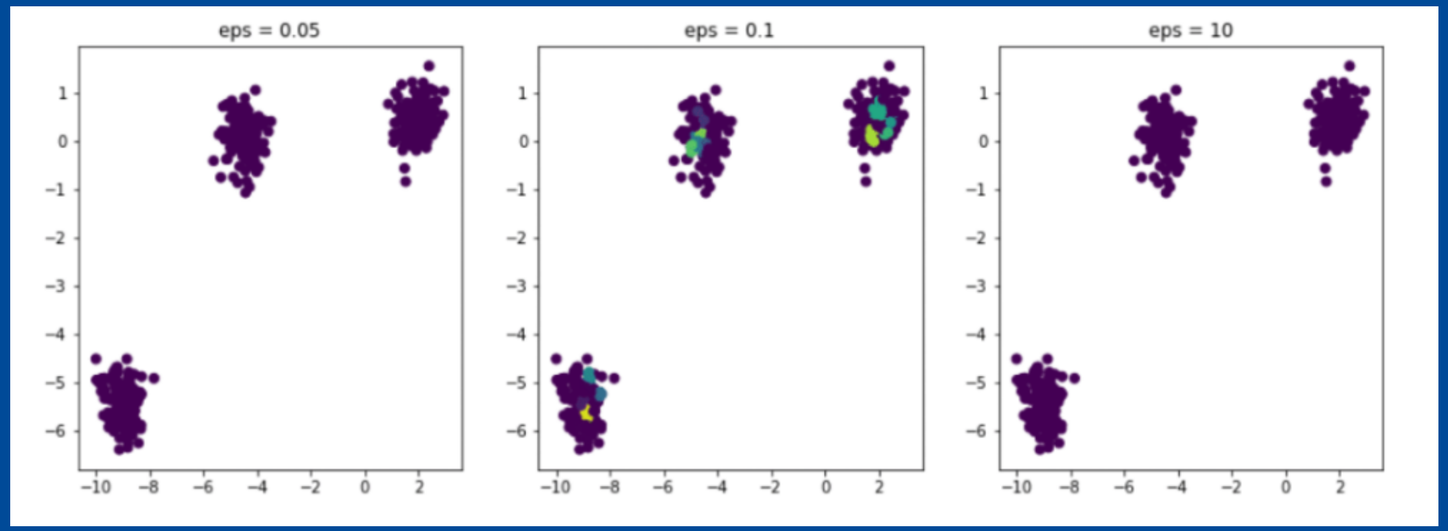
Quelle est la métrique utilisée pour évaluer la distance entre une observation et ses voisins ?
quel est le ε idéal ?

Dans le DBSCAN on utilise généralement la distance euclidienne, soient p = (p1,….,pn) et q = (q1,….,qn) :

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}.$$

À chaque observation, pour compter le nombre de voisins à au plus une distance ε , on calcule la distance euclidienne entre le voisin et l'observation et vérifie si c'est inférieur à ε.

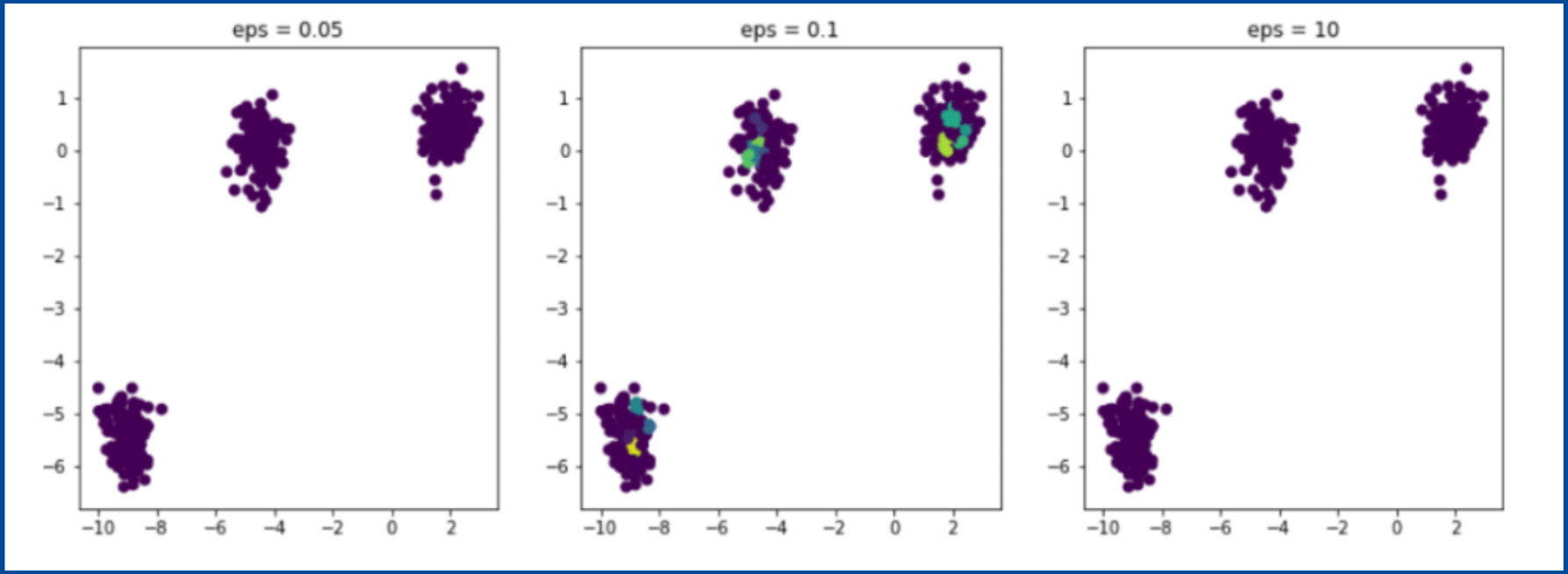
Reste maintenant à savoir comment choisir le bon epsilon. Supposons que dans notre exemple nous choisissons de tester l'algorithme avec des valeurs différentes de ε . Voici le résultat :



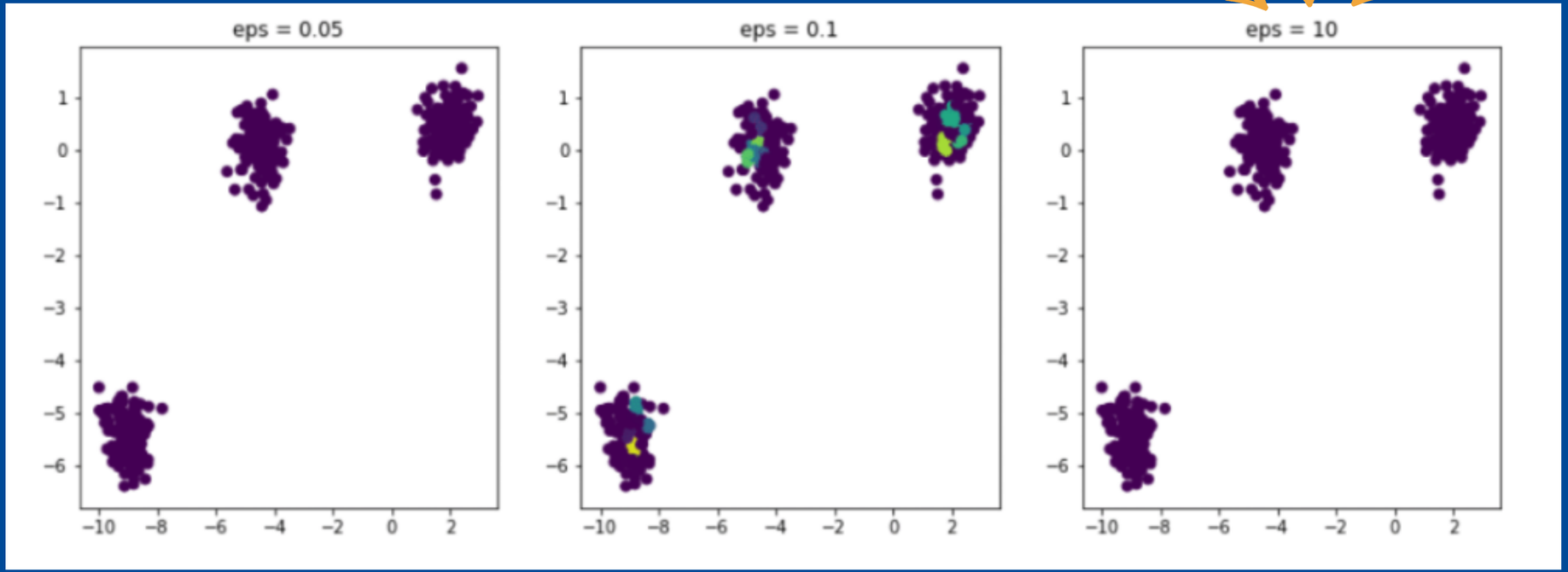
Dans les trois exemples le nombre de voisins minimal est toujours fixé à 5.

- Si ε est trop petit --> le ε-voisinage est trop faible et toutes **les observations du jeu de données** sont considérées comme **des anomalies**.

  C'est le cas de la figure de gauche eps = 0.05.



Au contraire, si epsilon est trop grand chaque observation contient dans son ε-voisinage toutes les autres observations du jeu de données. Par conséquent nous n'obtenons qu'**un unique cluster.** Il est donc très important de bien calibrer le ε pour obtenir un partitionnement de qualité.



Une méthode simple pour optimiser le ε consiste à regarder pour chaque observation à quelle distance se situe son voisin le plus proche. Ensuite il suffit de fixer un ε tel qu'une part « suffisamment grande » des observations aient une distance à son plus proche voisin inférieure à ε. Par « suffisamment grande » on entend 90-95% des observations qui doivent avoir au moins un voisin dans leur ε-voisinage.

Les avantages:

- Peut identifier des clusters de forme arbitraire et de taille variable, contrairement aux algorithmes de clustering basés sur la partition comme K-Means.

- Peut traiter efficacement les données à haute densité et peut ignorer les zones de faible densité ou de bruit.

- N'a pas besoin de spécifier à l'avance le nombre de clusters ou leur forme, ce qui rend l'algorithme plus facile à utiliser et plus généralisable.

Limites DBSCAN :

- La sélection des paramètres (epsilon et minPts) est cruciale pour obtenir des résultats de clustering optimaux. Des valeurs mal choisies peuvent entraîner des résultats de clustering incohérents ou incorrects.

- Il peut être sensible aux données bruyantes, car les points isolés peuvent être considérés comme des clusters de bruit.

- L'algorithme peut avoir des difficultés à identifier des clusters de densités différentes, car il traite tous les clusters avec la même sensibilité à la densité.

Exemple d'implementation :

Voici un exemple d'implémentation de DBSCAN en Python

https://colab.research.google.com/drive/1543fHN4GOLGgzZZiX_Fp1F3_-w8VlHhP#scrollTo=4FaK48vg2OOs

Conclusion :

Comme vous avez pu le voir, cet algorithme est plus performant que les k moyennes car il peut trouver des clusters non linéairement séparables, et il ne nécessite pas de spécifier un nombre minimum de clusters à trouver. L'inconvénient de cet algorithme est qu'il nécessite deux paramètres à devoir choisir, l'epsilon et le nombre minimum de points, ce qui peut s'avérer assez difficile.

Merci !