



# DBSCAN e *K-means*

Uma comparação

Alunos: Vinícius Calixto e Wellington Cesar



# Índice

## Introdução

- O que é agrupamento
- Base de dados usadas

## Sobre o projeto

- Pré-processamento
- Mineração de dados
- Pós processamento

## Desenvolvimento

- *K-means*
- DBSCAN

## Resultados

- Comparativo de agrupamentos
- Coeficiente de silhueta
- Complexidade de tempo
- Considerações finais



# O que é agrupamento de dados?

Aprendizagem não supervisionada

Baseadas em distância entre objetos

Exemplos: mineração de texto, segmentação de indivíduos, extração de conhecimento da web



# Base de dados usadas

- Iris

<https://www.kaggle.com/datasets/uciml/iris>

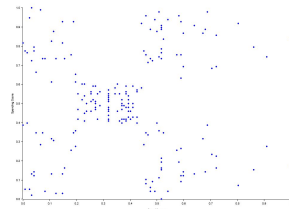
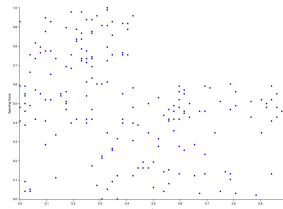
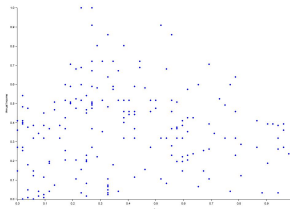
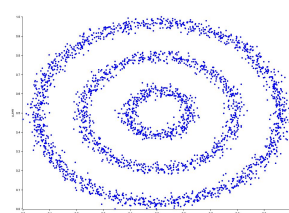
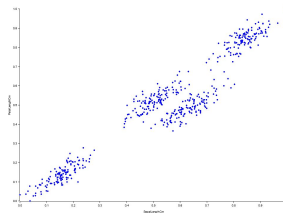
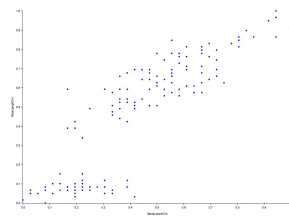
- Auto Geradas(Kaggle e Python)

Clustering gmm: <https://www.kaggle.com/datasets/anikit8467/dataset-for-dbscan>

Python created: <https://oralytics.com/2021/10/18/dbscan-clustering-in-python/>

- Mall Costumers 3D(Kaggle)

<https://www.kaggle.com/datasets/vichoudhary7/customer-segmentation-tutorial-in-python>






Introdução

# Sobre o projeto

# Sobre o projeto

- Projeto codificado na linguagem Rust 
  - Eficiente
  - Desenvolvida para evitar problemas de memória
  - Permite controle de baixo nível
- Versionado no GitHub
  - <https://github.com/VCalixtoR/dbscan/>
- Etapas
  - Pré processamento
  - Mineração de dados - Kmeans & DBSCAN
  - Pós processamento

README.md

## Agrupamento de dados - DBSCAN e Kmeans

Este repositório contém o trabalho final da disciplina de mineração de dados. O objetivo é a implementação e análise de resultados relacionados a mineração de dados utilizando algumas bases de dados.

No contexto foram abordadas duas técnicas de agrupamento (clustering), DBSCAN e Kmeans, estes foram implementados sem uso de bibliotecas terceiras na linguagem Rust, por ser uma linguagem bastante eficiente se comparado a outras linguagens usadas para data mining.

### Início rápido

Para execução do projeto no sistema operacional windows ou linux/Fedora 30-36, executar o respectivo binário compilado na pasta bin **pelo terminal**, necessário a descompactação.

- A execução irá realizar todas as etapas, pré-processamento, data mining e pós processamento salvando resultados na pasta postprocessing presente na pasta do executável.
- Ex: Descompactar windows-release.zip ou fedora-release.zip seguido pela execução do arquivo executável nomeado dbscan.
- **Atenção:** Repare que antes de processar resultados da postprocessing não estarão presentes e após serão criadas.

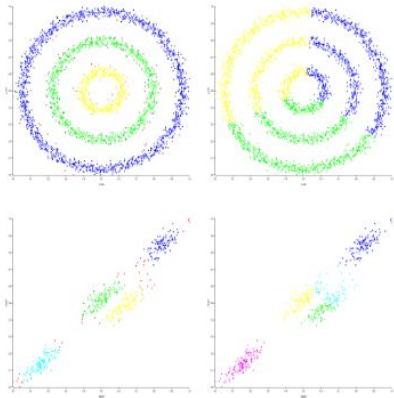
Para manipulações ao **código fonte** para compilação e execução utilizando ferramentas do rust:

- Instale Rust em sua máquina
- Clone este repositório
- Na pasta do clone execute no terminal o comando `$cargo run` para compilação e execução do projeto

### Dependências

- rust 1.70.0
- csv = "1.2.2"
- plotters = "0.3.3"

### Alguns resultados





# ***Pré-processamento***

- Etapas
  - Acesso e carregamento da base para a memória principal
  - Conversão da base em um TAD de índices para eficiência de acesso
  - Escolha de parâmetros do TAD
  - Normalização

TAD - Tipo abstrato de dados



# *Mineração de dados*

- Etapas
  - Clusterização utilizando K-means
  - Clusterização utilizando DBSCAN





## ***Pós processamento***

- Etapas
  - Cálculo do coeficiente de silhueta para K-means e DBSCAN
  - Geração de gráficos 2D com parâmetros parametrizáveis com pontos coloridos (Pontos core, border e outlier)
  - Geração do gráfico da base de dados não classificada para comparação
- Testes de velocidade fora do escopo do programa no terminal fedora



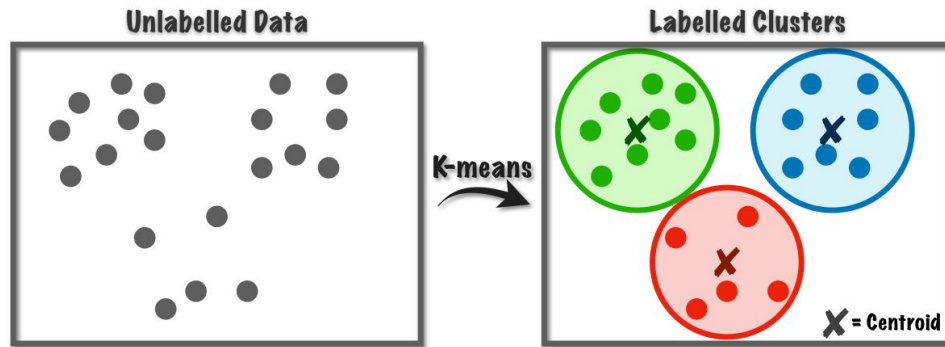
Primeiro algoritmo

# *K-means*



# *K-means*

- Descrito por várias pessoas entre as décadas de 1950 e 1960. Termo cunhado em 1967.
- Algoritmo de agrupamento particional iterativo baseado em centróides





# *K-means*

- Dependente do hiperparâmetro composto por um conjunto de pontos denominados **centróides**
- Agrupamento iterativo com base na proximidade dos pontos ao centróide
- Iterativamente muda os centróides baseado na média dos elementos próximos selecionados

---

**Algorithm 1** *k*-means algorithm

---

- 1: Specify the number *k* of clusters to assign.
  - 2: Randomly initialize *k* centroids.
  - 3: **repeat**
  - 4:   **expectation:** Assign each point to its closest centroid.
  - 5:   **maximization:** Compute the new centroid (mean) of each cluster.
  - 6: **until** The centroid positions do not change.
-

# Código Fonte: K-means



```
4 // do k-means clusterization with initial centroid points hyperparam
5 pub fn kmeans_clusterization(centroid_points: &mut PointVector, database_points: &PointVector) -> ClusterGroup {
6
7     let mut cluster_group: ClusterGroup = ClusterGroup::new(centroid_points.len());
8     let mut min_centroid_distance: f32;
9     let mut min_centroid_index: usize;
10    let mut tmp_float: f32;
11    let mut iteration: u32 = 0;
12
13    // while centroid changes
14    let mut cluster_group_has_changed: bool = true;
15    while cluster_group_has_changed {
16        iteration += 1;
17        // temporary cluster group
18        let mut tmp_cluster_group = ClusterGroup::new(centroid_points.len());
19        // foreach point
20        for point_index in 0..database_points.len() {
21            min_centroid_distance = f32::MAX;
22            min_centroid_index = 0;
23
24            // calculate the distance to the centroids and group it to the nearest centroid
25            for centroid_index in 0..centroid_points.len() {
26                tmp_float = euclidean_distance(&centroid_points[centroid_index], &database_points[point_index]);
27                if min_centroid_distance > tmp_float {
28                    min_centroid_distance = tmp_float;
29                    min_centroid_index = centroid_index;
30                }
31            }
32            tmp_cluster_group.add_index_to_cluster(PointType::Core, point_index, min_centroid_index);
33        }
34        cluster_group_has_changed = !cluster_group.is_ordinated_equals_to(&tmp_cluster_group);
35
36        if cluster_group_has_changed {
37            cluster_group = tmp_cluster_group; // move pointer to cluster_group and drops tmp_cluster_group
38            calc_centroid_by_mean(database_points, centroid_points, &cluster_group);
39        }
40    }
41    return cluster_group;
42 }
```

```
// calculate new centroids positions based on average of its points
fn calc_centroid_by_mean(database_points: &PointVector, centroid_points: &mut PointVector, cluster_group: &ClusterGroup) -> () {

    let mut cluster_sum: Point;

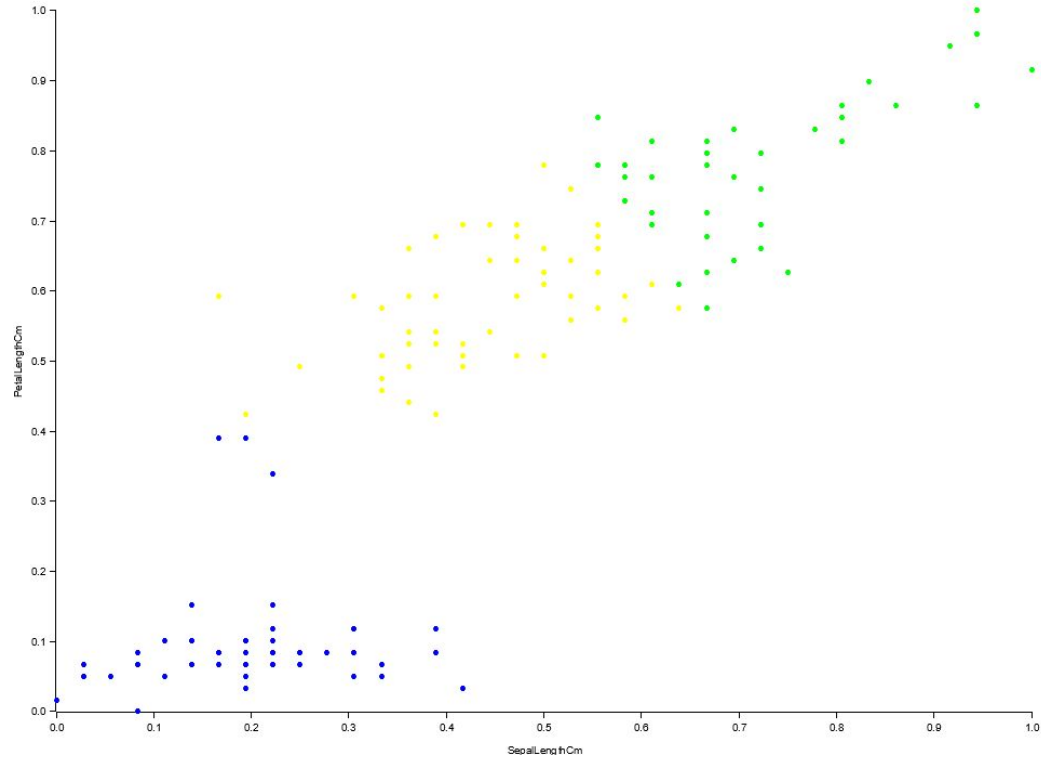
    for cluster_pos in 0..cluster_group.clusters.len() {
        // sum all values from a cluster
        cluster_sum = vec![0.0; database_points[0].len() as usize];
        for core_pos in 0..cluster_group.clusters[cluster_pos].core_indexes.len() {
            let index = cluster_group.clusters[cluster_pos].core_indexes[core_pos];
            for value_pos in 0..database_points[index].len() {
                cluster_sum[value_pos] += database_points[index][value_pos];
            }
        }
        // assign average value to centroid
        for value_pos in 0..cluster_sum.len() {
            centroid_points[cluster_pos][value_pos] = cluster_sum[value_pos] /
                (cluster_group.clusters[cluster_pos].core_indexes.len() as f32);
        }
    }
}
```

# Íris - SepalLengthCm X PetalLengthCm



## Parâmetros:

Número de Centróides: 3

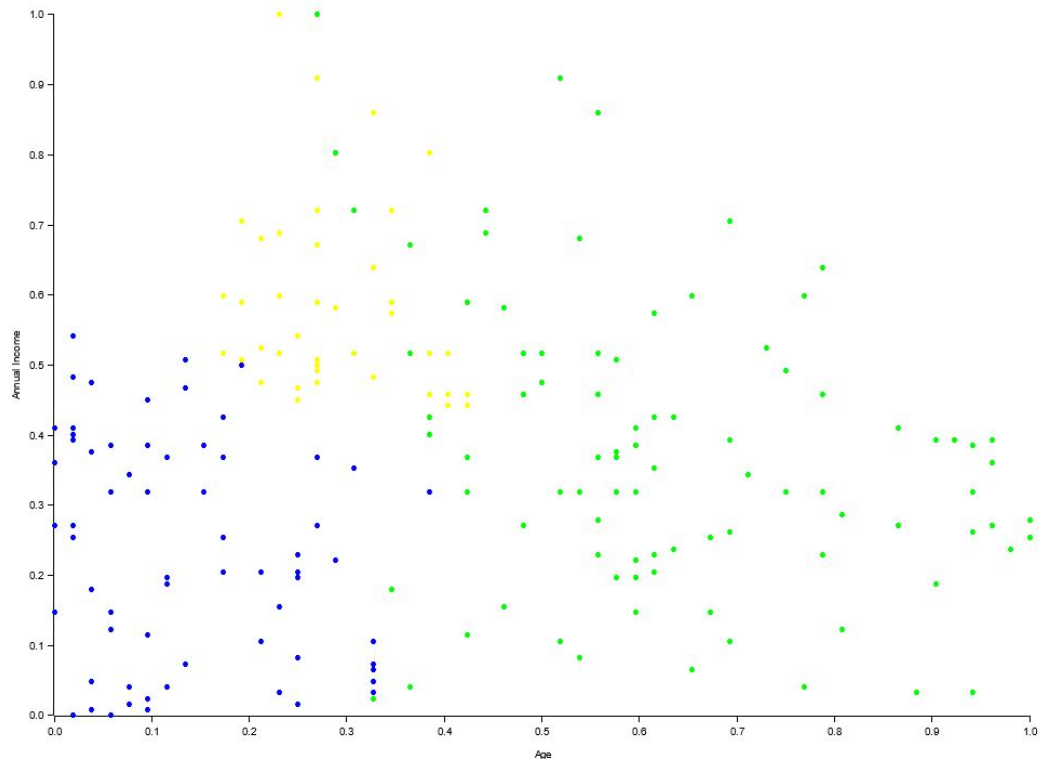


# Mall Costumers 3D - Age X AnualIncome



## Parâmetros:

Número de Centróides: 3



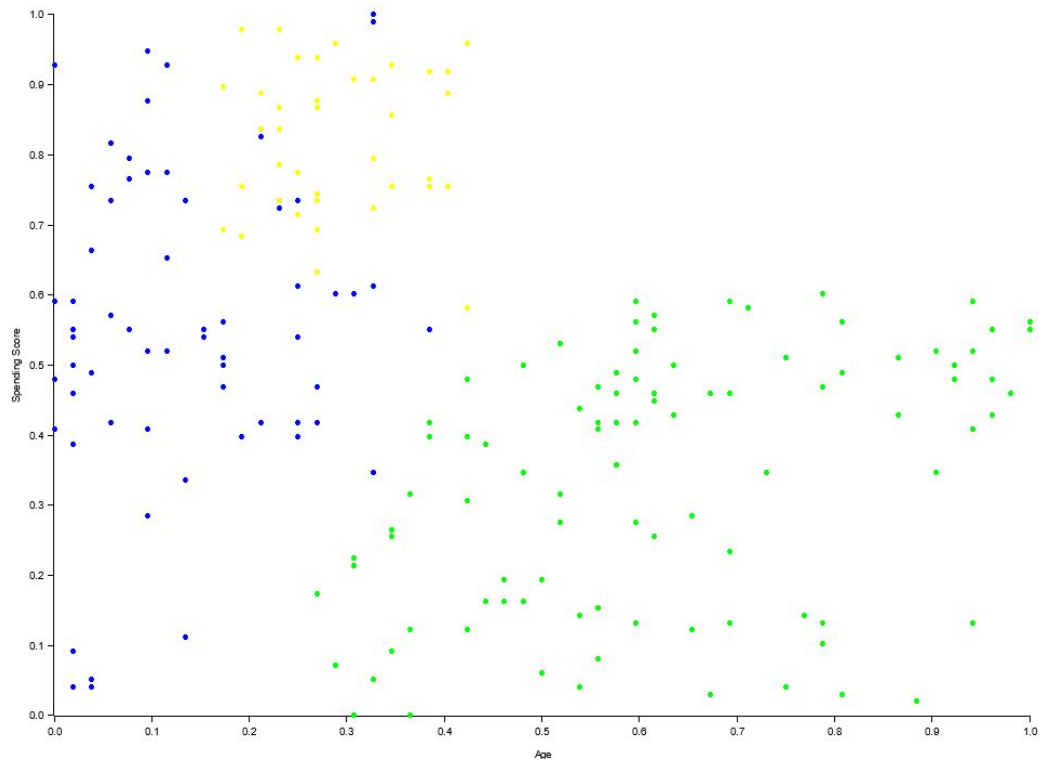


# Mall Costumers 3D - Age X SpendingScore



## Parâmetros:

Número de Centróides: 3

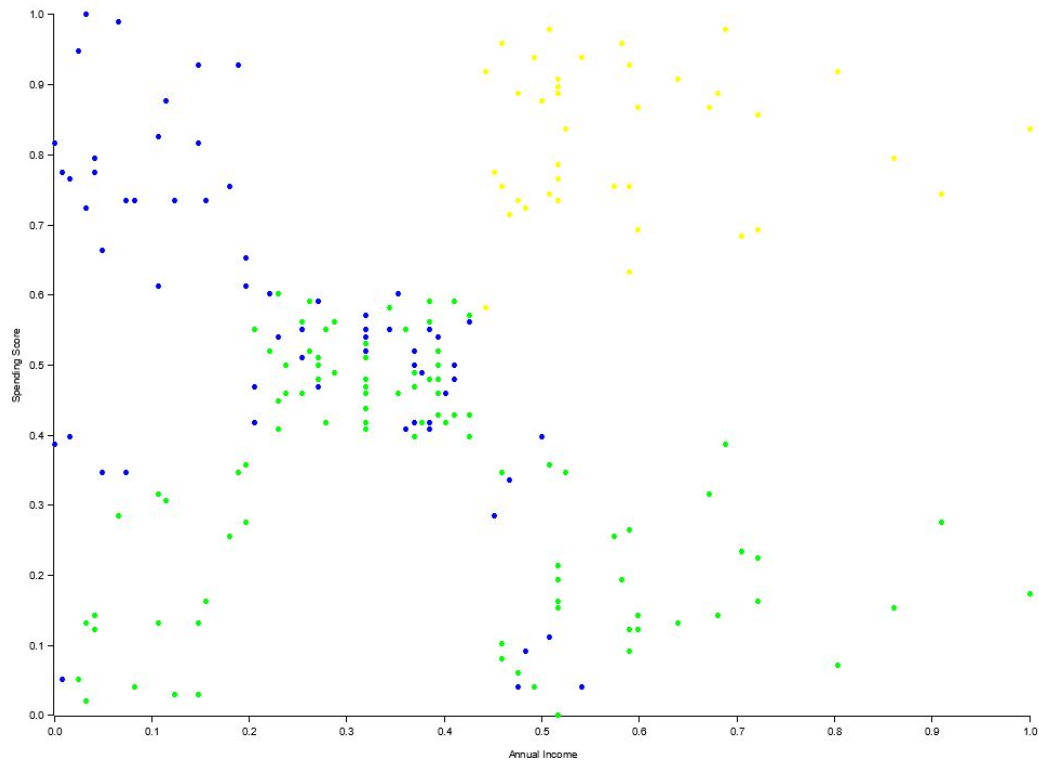


# Mall Costumers 3D - AnnualIncome X SpendingScore



## Parâmetros:

Número de Centróides: 3

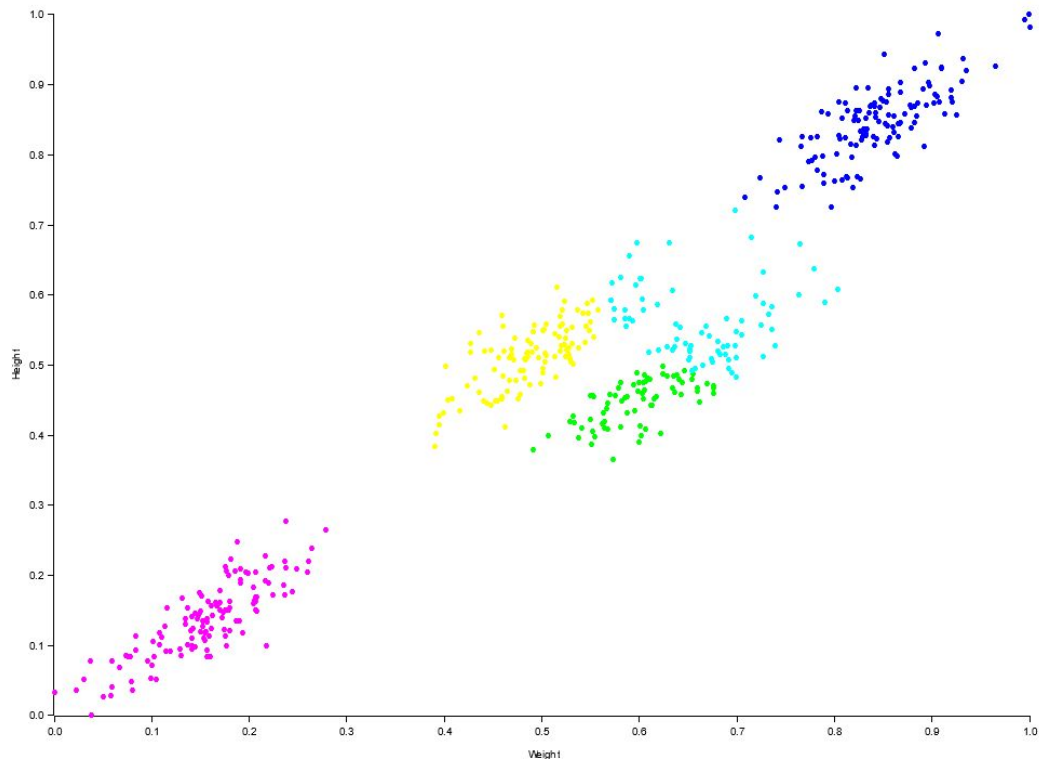


# Clustering\_gmm - Weight X Height



## Parâmetros:

Número de Centróides: 5

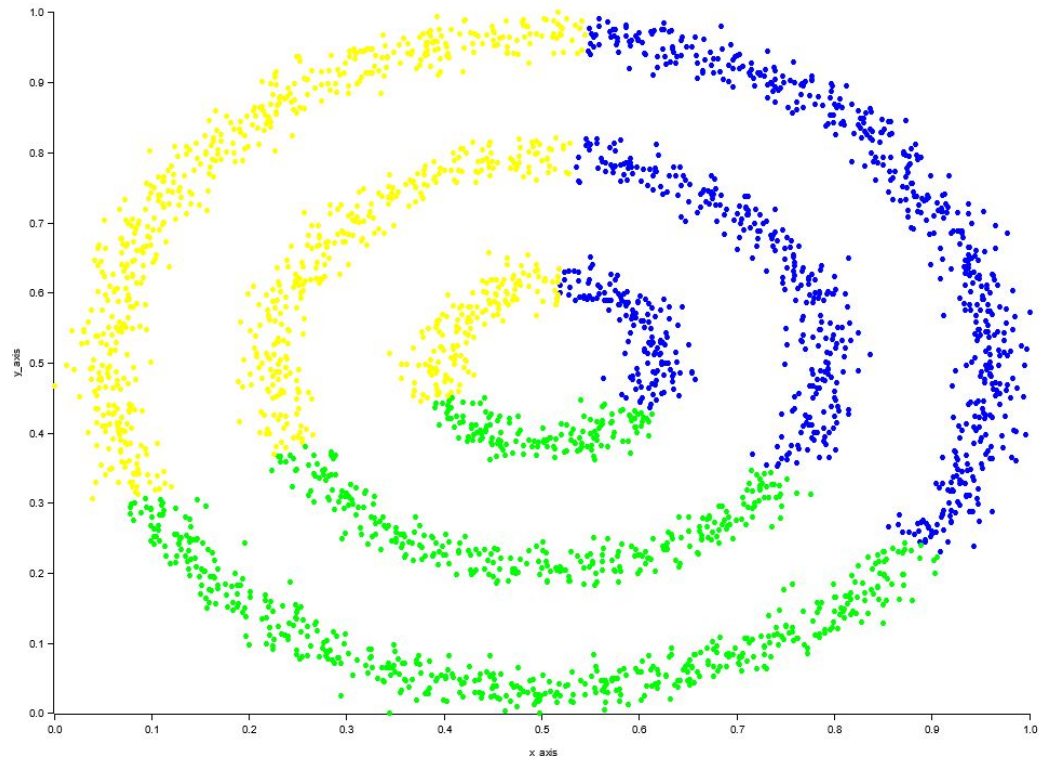


# Circular Generated - x\_axis X y\_axis



## Parâmetros:

Número de Centróides: 3





Segundo algoritmo

# *DBSCAN*



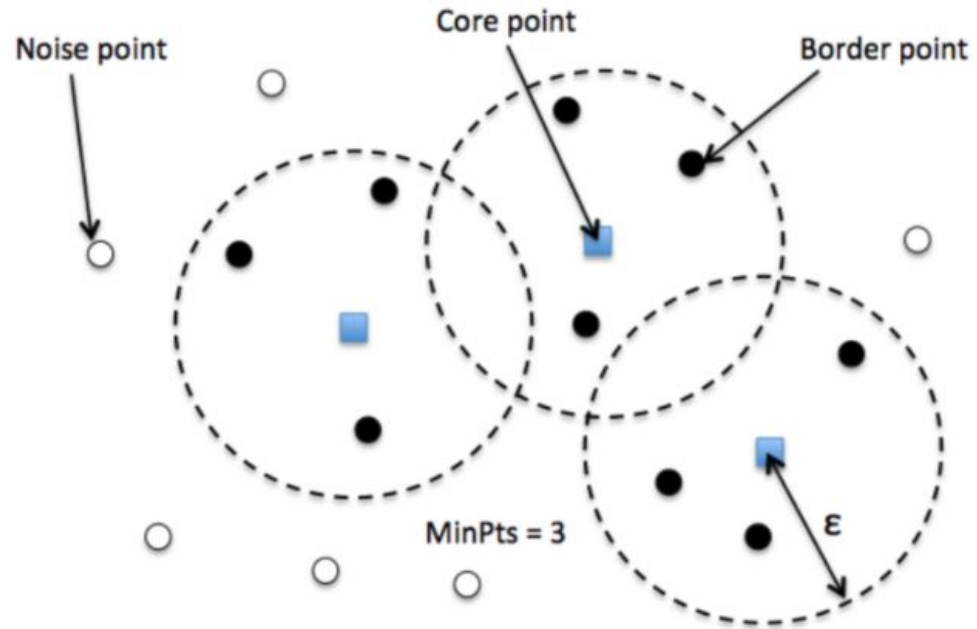
# DBSCAN

- Descrito pela primeira vez em um artigo de 1996
- Algoritmo de agrupamento particional baseado em densidade



# DBSCAN

- Dependente dos hiperparâmetros  $\epsilon$  e **minPts**
- Pontos com **minPts** pontos ou mais em sua  $\epsilon$ -vizinhança são considerados pertencentes a uma região de alta densidade

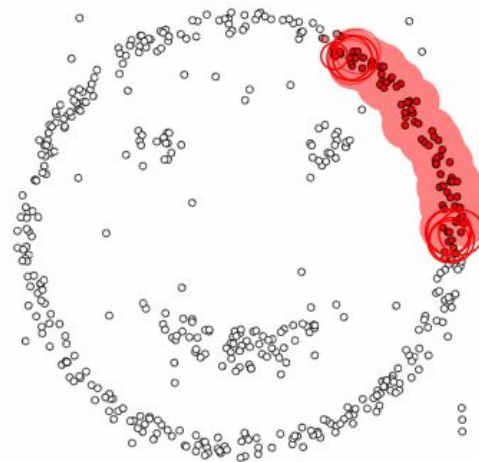




# DBSCAN

- Dependente dos hiperparâmetros  $\epsilon$  e **minPts**
- Pontos com **minPts** pontos ou mais em sua  $\epsilon$ -vizinhança são considerados pertencentes a uma região de alta densidade

epsilon = 1.00  
minPoints = 4



Restart




Pause





```
let mut current_cluster_id = 0;
for i in 0..dataset.0.len() {
    match dataset.0[i].cluster_label {
        ClusterLabel::Undefined => {
            let shoud_increment_id = expand_cluster(dataset, i, current_cluster_id, eps, min);
            if shoud_increment_id {
                current_cluster_id += 1;
            }
        }
        _ => {}
    }
}
```



```
let mut expansion = neighbourhood(dataset, index, eps);
if expansion.len() < min {
    dataset.0[index].cluster_label = ClusterLabel::Noise;
    return false;
}

dataset.0[index].cluster_label = ClusterLabel::ClusterId(cluster_id);
.
.
.
```

```
while expansion.len() != 0 {  
    let current = expansion.pop_front().unwrap();  
    match dataset.0[current].cluster_label {  
        ClusterLabel::Undefined => {  
            dataset.0[current].cluster_label = ClusterLabel::ClusterId(cluster_id);  
            let mut current_expansion = neighbourhood(dataset, current, eps);  
            if current_expansion.len() >= min {  
                expansion.append(&mut current_expansion);  
            }  
        }  
        ClusterLabel::Noise => {  
            dataset.0[current].cluster_label = ClusterLabel::ClusterId(cluster_id);  
        }  
        ClusterLabel::ClusterId(_) => {}  
    }  
}  
true
```

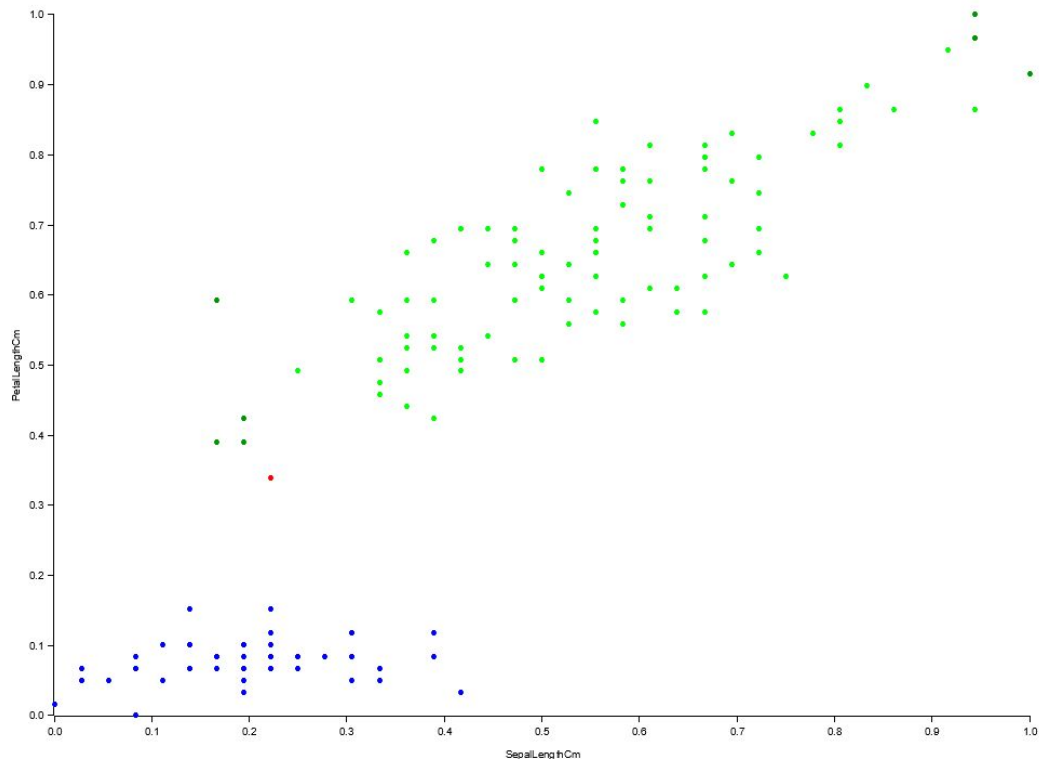
# Íris - SepalLengthCm X PetalLengthCm



## Parâmetros:

raio  $\epsilon$  (epsilon): 0.15

pontos mínimos: 7

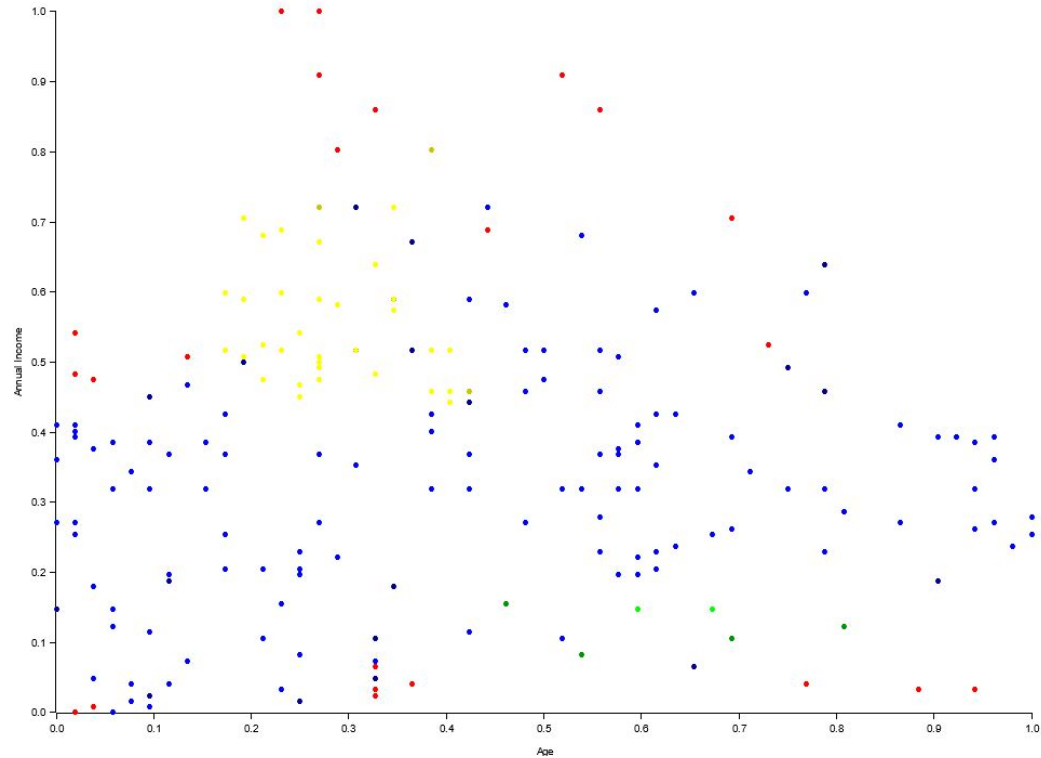


# Mall Customers 3D - Age X AnnualIncome



## Parâmetros:

raio  $\epsilon$  (epsilon): 0.151  
pontos mínimos: 4



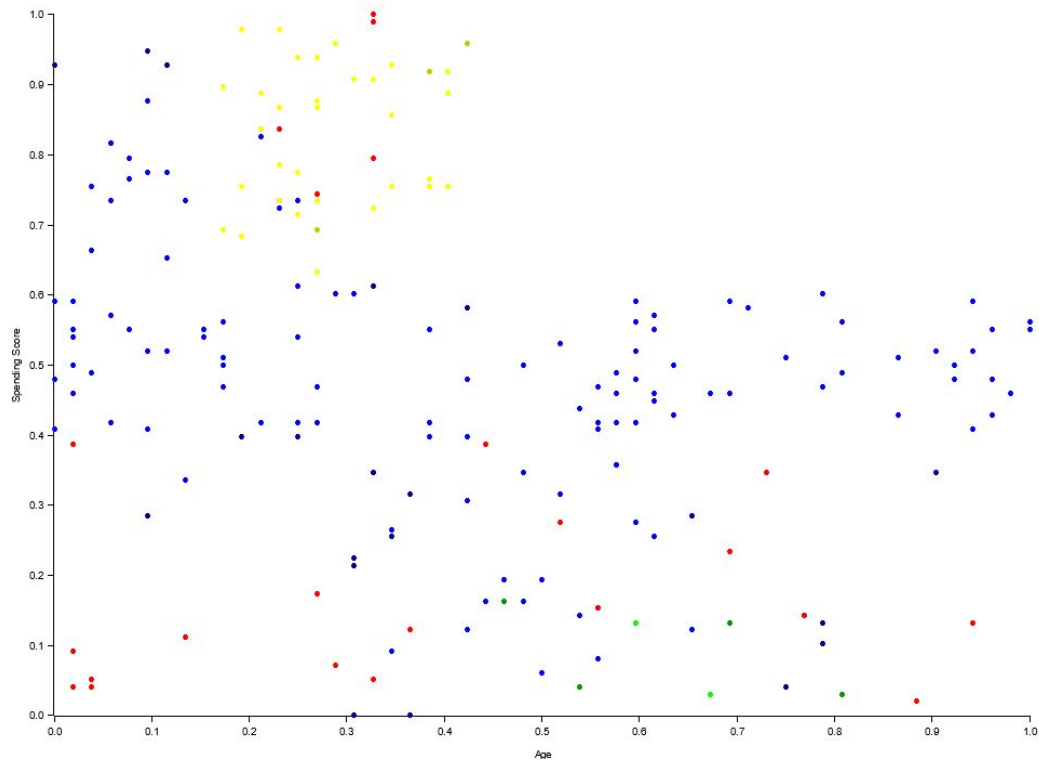
# Mall Customers 3D - Age X SpendingScore



## Parâmetros:

raio  $\epsilon$  (epsilon): 0.151

pontos mínimos: 4



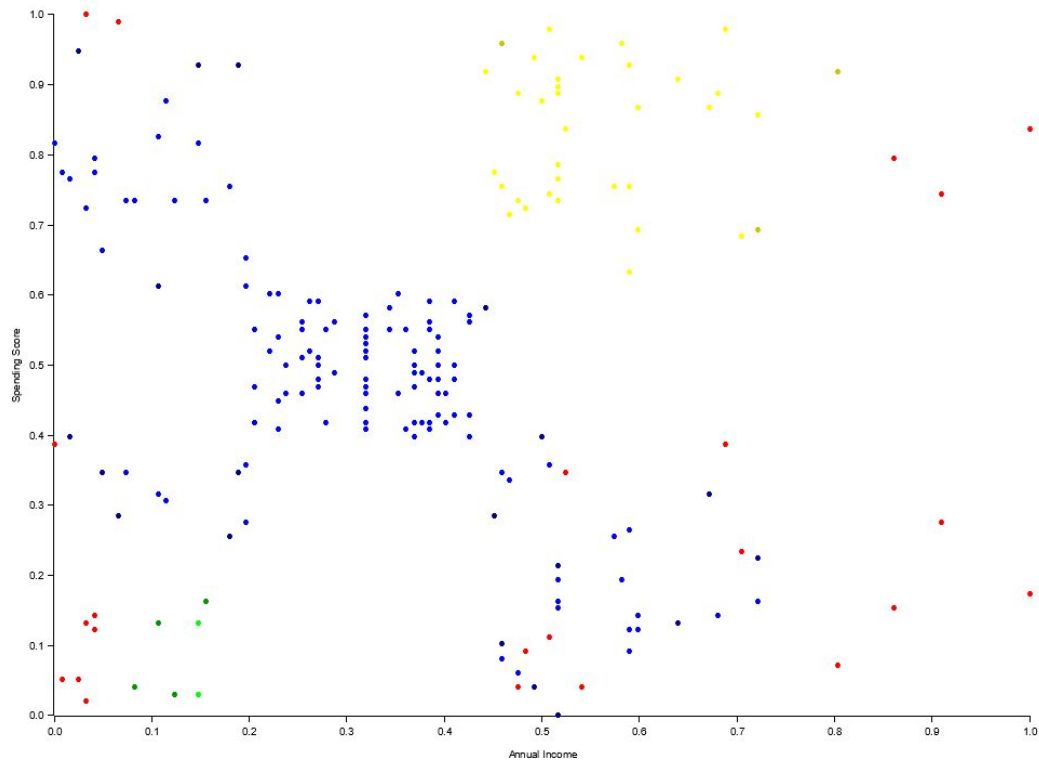
# Mall Customers 3D - AnnualIncome X SpendingScore



## Parâmetros:

raio  $\epsilon$  (epsilon): 0.151

pontos mínimos: 4



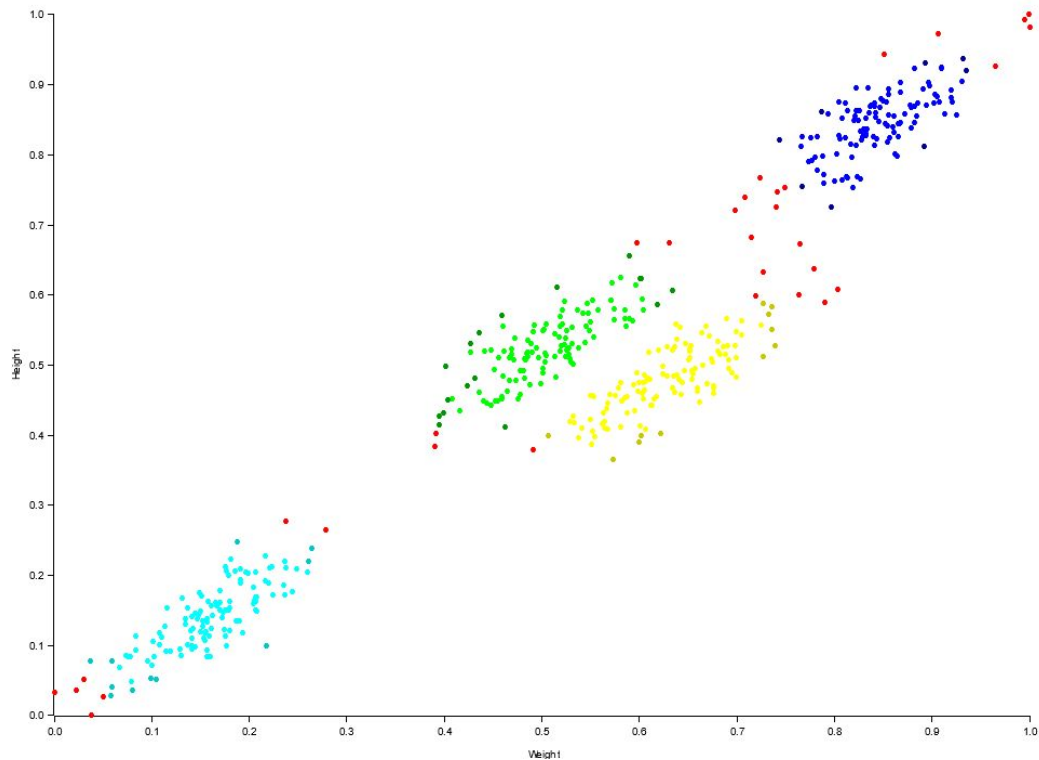
# Clustering\_gmm - Weight X Height



## Parâmetros:

raio  $\epsilon$  (epsilon): 0.035

pontos mínimos: 7





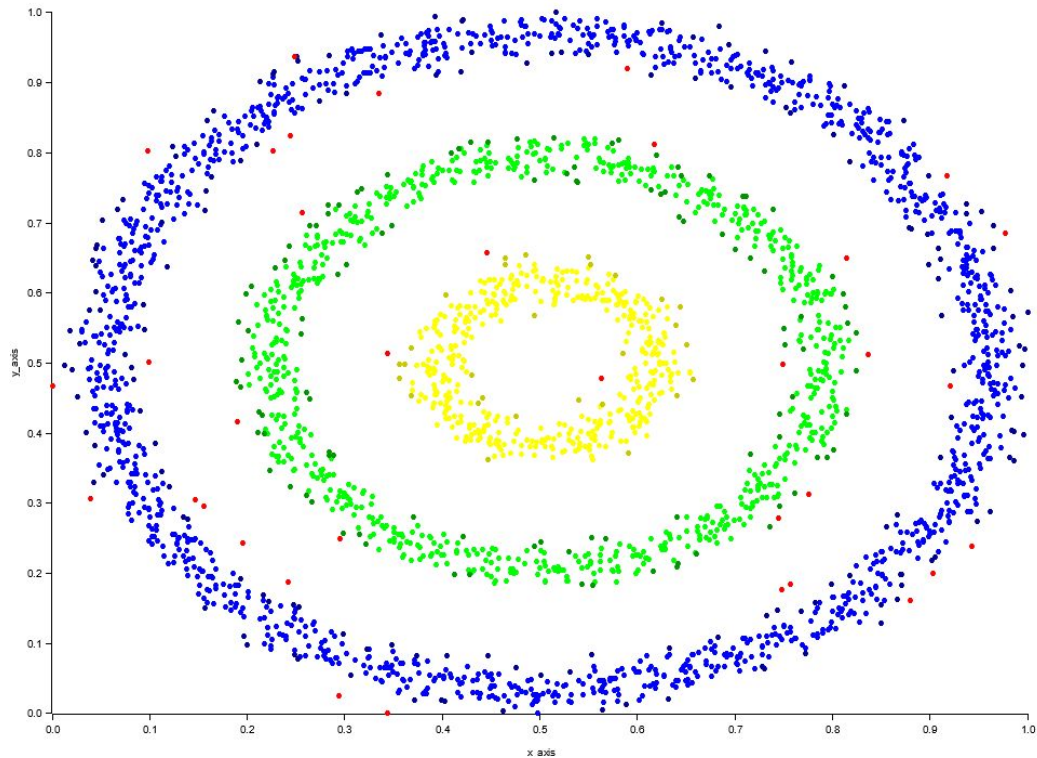
# Circular Generated - x\_axis X y\_axis



## Parâmetros:

raio  $\epsilon$  (epsilon): 0.025

pontos mínimos: 8



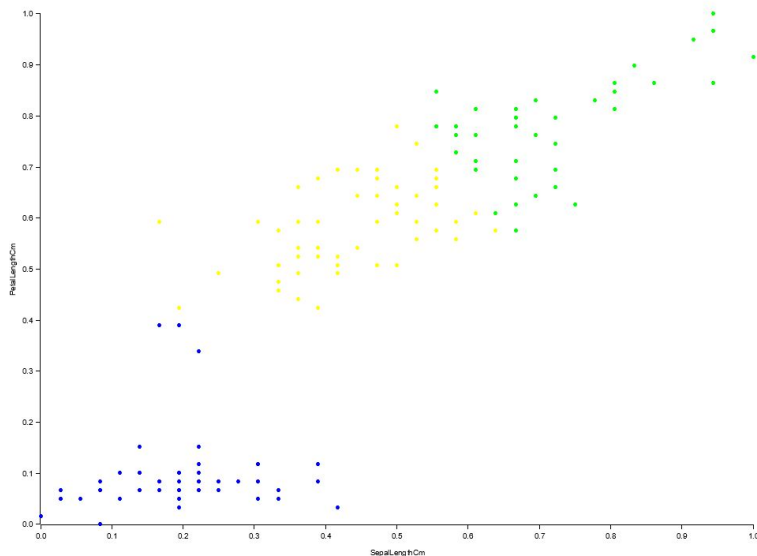


## Resultados

# Íris - SepalLengthCm X PetalLengthCm

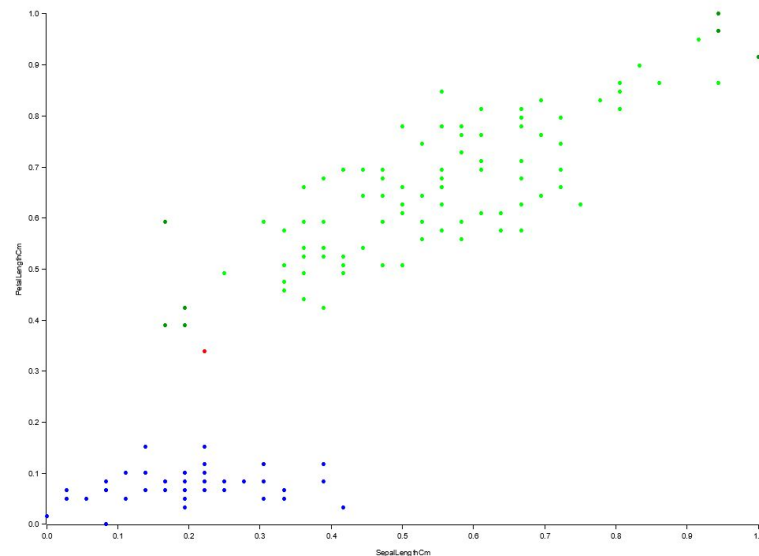


## K-means



```
Kmeans: Silhouette Coefficients:  
For C0 = 0.73  
For C1 = 0.41  
For C2 = 0.49  
Global = 0.55
```

## DBSCAN



```
DBSCAN: Silhouette Coefficients:  
For C0 = 0.83  
For C1 = 0.58  
Global = 0.66
```

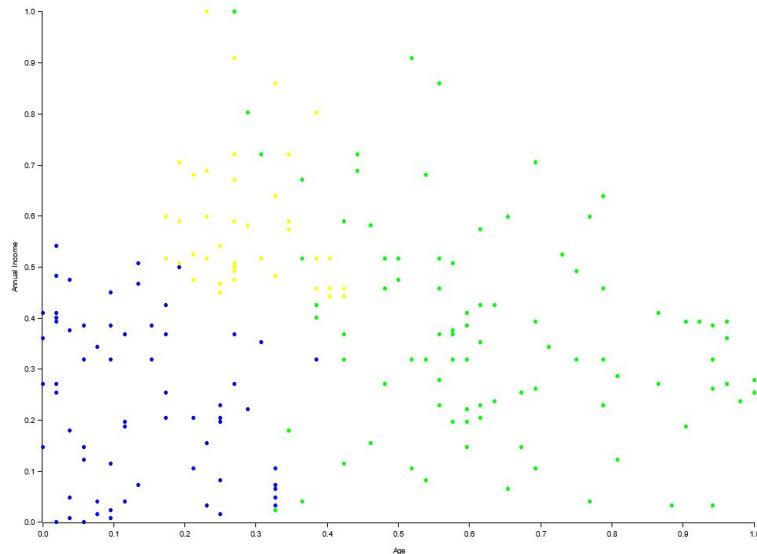
K-means took 5 milliseconds

DBSCAN took 25 milliseconds

# Mall Customers 3D - Age X AnnualIncome



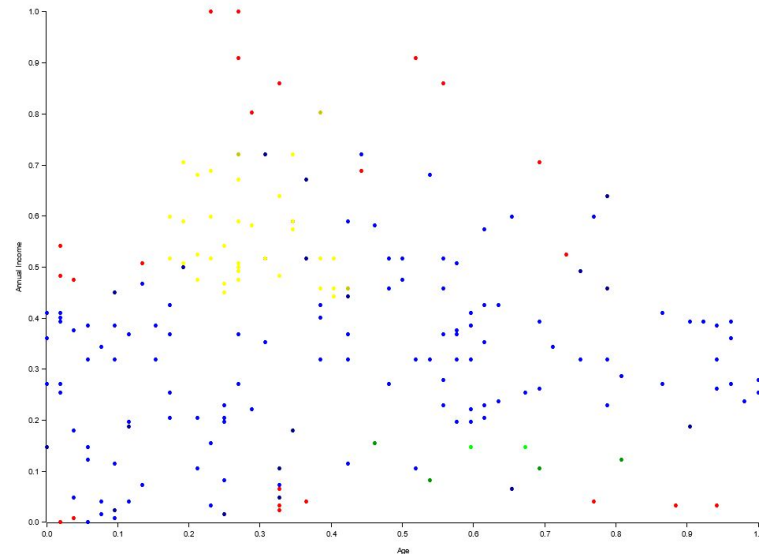
## K-means



```
Kmeans: Silhouette Coefficients:  
For C0 = 0.30  
For C1 = 0.32  
For C2 = 0.57  
Global = 0.36
```

K-means took 4 milliseconds

## DBSCAN



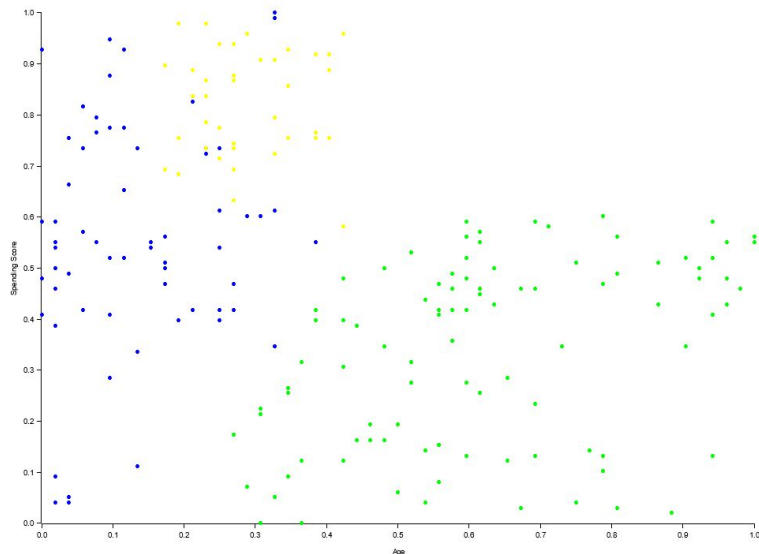
```
DBSCAN: Silhouette Coefficients:  
For C0 = -0.03  
For C1 = 0.69  
For C2 = 0.66  
Global = 0.14
```

DBSCAN took 79 milliseconds

# Mall Customers 3D - Age X SpendingScore



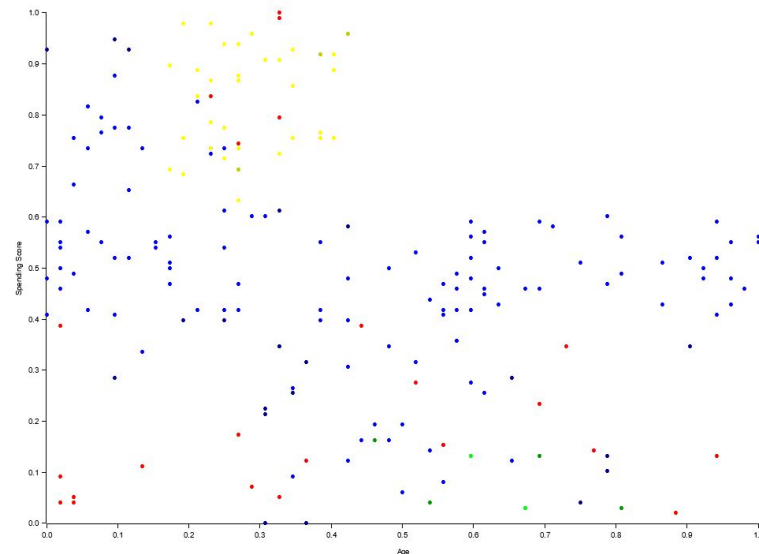
## K-means



```
Kmeans: Silhouette Coefficients:  
For C0 = 0.30  
For C1 = 0.32  
For C2 = 0.57  
Global = 0.36
```

K-means took 4 milliseconds

## DBSCAN



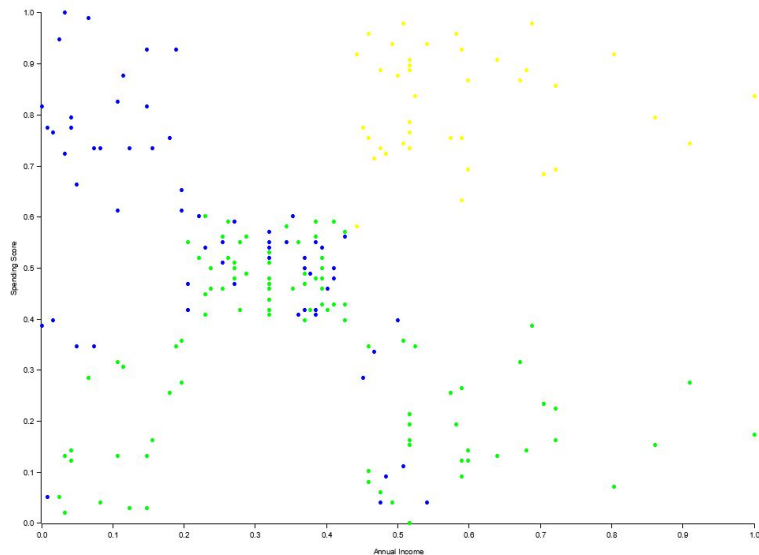
```
DBSCAN: Silhouette Coefficients:  
For C0 = -0.03  
For C1 = 0.69  
For C2 = 0.66  
Global = 0.14
```

DBSCAN took 79 milliseconds

# Mall Customers 3D - AnnualIncome X SpendingScore



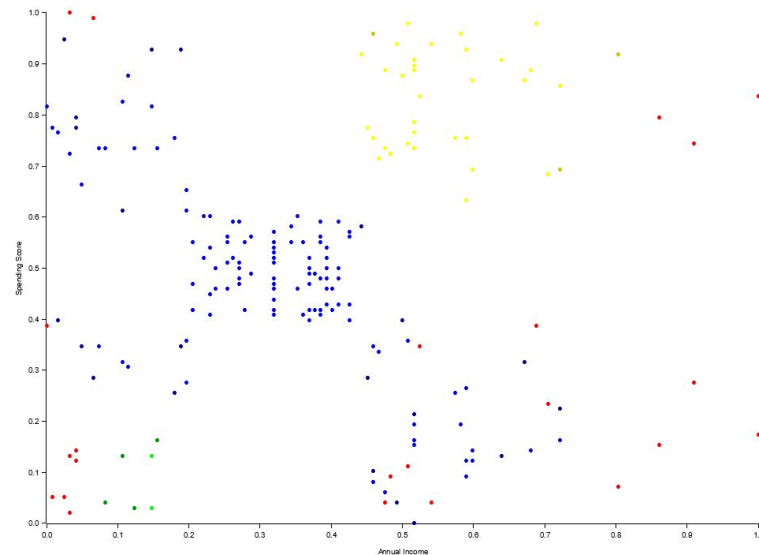
## K-means



```
Kmeans: Silhouette Coefficients:  
For C0 = 0.30  
For C1 = 0.32  
For C2 = 0.57  
Global = 0.36
```

K-means took 4 milliseconds

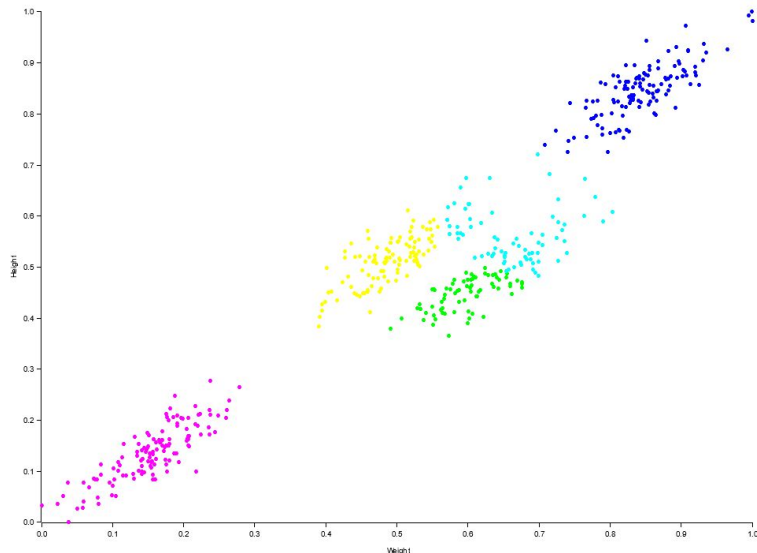
## DBSCAN



```
DBSCAN: Silhouette Coefficients:  
For C0 = -0.03  
For C1 = 0.69  
For C2 = 0.66  
Global = 0.14
```

DBSCAN took 79 milliseconds

## K-means



Kmeans: Silhouette Coefficients:

For C0 = 0.72

For C1 = 0.47

For C2 = 0.47

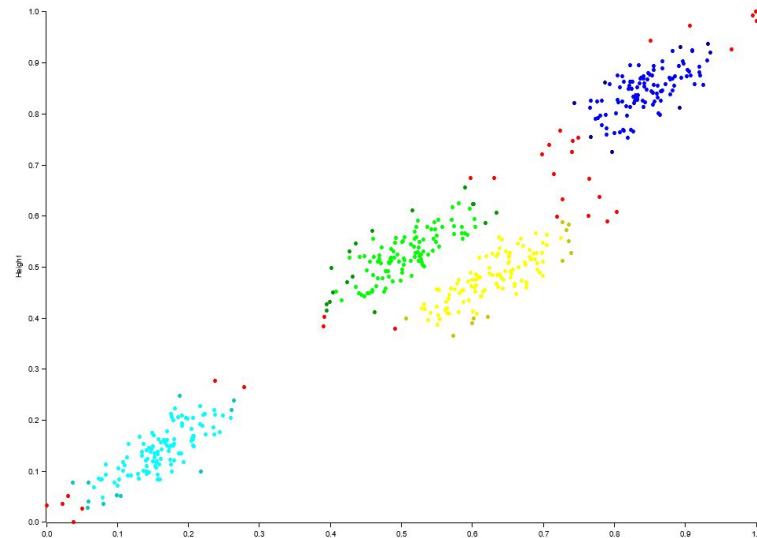
For C3 = 0.28

For C4 = 0.81

Global = 0.59

K-means took 10 milliseconds

## DBSCAN



DBSCAN: Silhouette Coefficients:

For C0 = 0.82

For C1 = 0.44

For C2 = 0.44

For C3 = 0.84

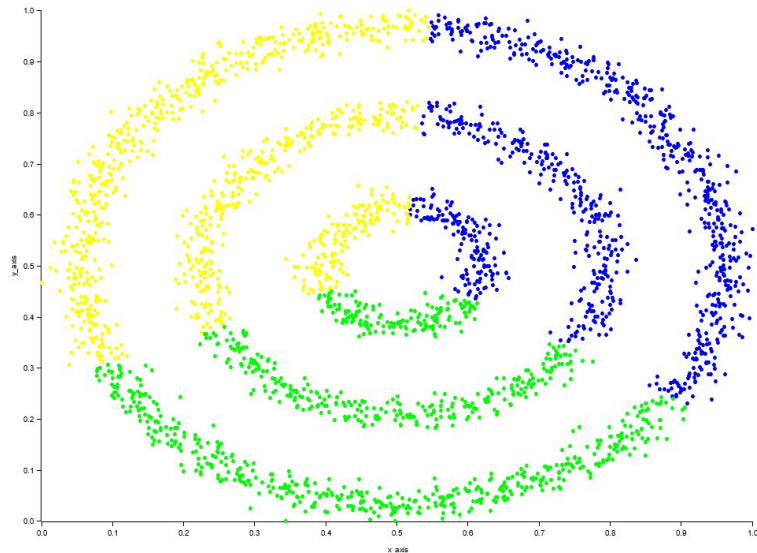
Global = 0.63

DBSCAN took 79 milliseconds

# Circular Generated - x\_axis X y\_axis



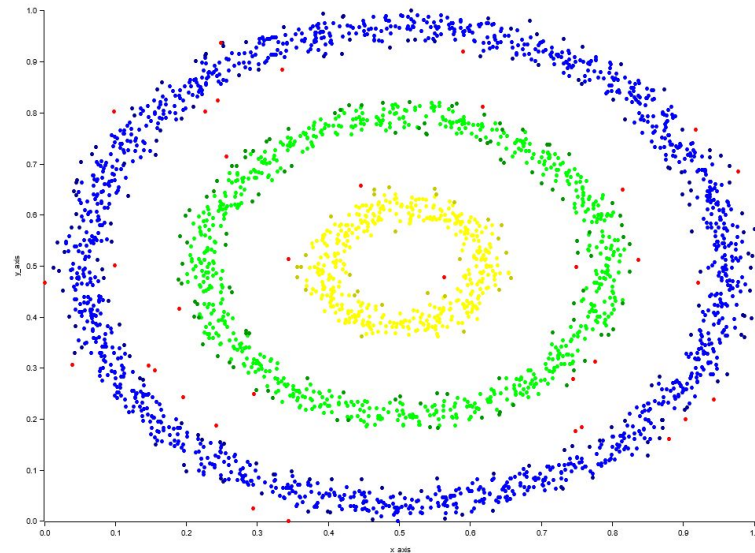
## K-means



```
Kmeans: Silhouette Coefficients:  
For C0 = 0.37  
For C1 = 0.37  
For C2 = 0.37  
Global = 0.37
```

K-means took 19 milliseconds

## DBSCAN



```
DBSCAN: Silhouette Coefficients:  
For C0 = -0.20  
For C1 = -0.19  
For C2 = 0.50  
Global = -0.08
```


DBSCAN took 1391 milliseconds





## Complexidade temporal

K-means took 19 milliseconds  
DBSCAN took 1391 milliseconds



```
(0..dataset.0.len()).filter(|index2| {  
    let dist = euclidean_distance(dataset, index, *index2);  
    dist <= eps && dist > 0.0  
}).collect()
```



# Conclusões

- Média de coeficiente de silhueta:  
*K-means*: 0,4675  
**DBSCAN**: 0,3375
- Média de tempo de execução:  
*K-means*: 9,5 ms  
**DBSCAN**: 393,5 ms
- DBSCAN realiza um agrupamento melhor considerando análise visual empírica
- O agrupamento do K-means pode ser melhorado ao considerar o uso de mais centróides seguido pelo agrupamento de cluster próximos
- A complexidade temporal do DBSCAN pode ser melhorada com armazenamento correto
- O agrupamento não-esférico do DBSCAN nem sempre tem o melhor coeficiente de silhueta
- Outras métricas para estudar a qualidade dos algoritmos podem ser usadas

**Alguma pergunta?**



# Obrigado!

