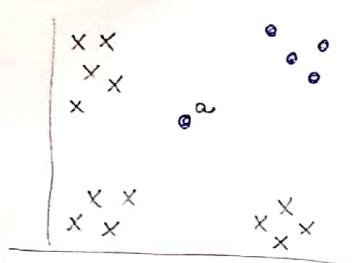


exercício 2) Assumindo que a inicialização se dá em pontos que não são da base de dados, pode ser um caso em que essa inicialização aleatória não está sendo muito "representativa".

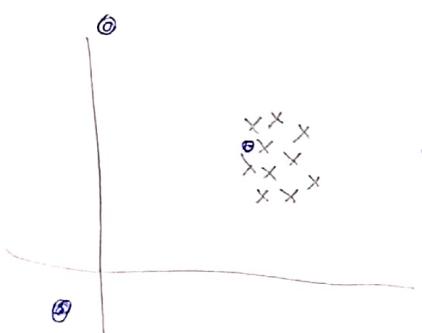
Suponha os dados distribuídos como a seguir



Os  $x$  são os dados e os  $\circ$  são as inicializações. Mesmo com todos as inicializações dentro de [min-coordenada, max-coordenada] em cada

atributo, estão em regiões distante e neste caso todos os exemplos tem "a" como centroide mais próximo.

Isso poderia acontecer também no cenário em que os centroides iniciais não respeitam os intervalos adequados. Algo como no desenho.



Com os  $\circ$  estão muito espalhados somente um deles é proximo a todos.

Esses dois casos seriam uma enorme coincidência.

Nesses casos o DBSCAN e o single-linkage não teriam esses problemas de inicialização, mas não teriam nenhum comportamento particular do exemplo.

Agora no cenário em que os centros de iniciação  
estão na base, então esse isso poderia ocorrer  
por exemplo se todos os pontos da base estão  
no mesmo lugar.

Daí os pontos seriam atribuídos a somente um  
grupo da partição.

Aqui, o DBSCAN nos retornaria o mesmo  
resultado do Kmeans.

O single linkage nos retornaria um dendograma colapsado em que todos os pon-  
tos se unem no mesmo patamar.  
Indicando também apenas 1 cluster.

Exercício 4) ②C) este é um exemplo em que os clusters tem elementos bem similares internamente. Neste caso temos a diagonal bem escura.

Nele temos ainda que o grupo "2" tá mais próximo do grupo "1" do que o "3". Neste caso, os exemplos ficam num tom de cinza mais escuro no grupo "2" na linha do grupo 1 do que os elementos do "3".

④B) Aqui tbm temos grupos bem definidos, com pouco ruído. Daí uma diagonal escura. Fora da diagonal não há clusters mais próximos. Neste caso o tom de cinza é "constante" na similaridade de elementos de cluster diferentes.

③A) Temos um exemplo parecido com o 2C, mas com clusters se misturando fazendo a matriz ficar ruídosinha. Ainda sim os tons de cinza diferentes fora da diagonal mostram que há cluster mais próximos.

①D) Um exemplo de agrupamento onde a presença de clusters é questionável. A matriz fica muito ruídosinha

Ex 1.1)

*menor distância*

	1	2	3	4	5	6	7
1	0.00	0.21	1.26	1.18	2.36	2.09	0.44
2	0.21	0.00	1.44	1.39	2.56	2.28	0.42
3	1.26	1.44	0.00	0.64	1.23	0.89	1.26
4	1.18	1.39	0.64	0.00	1.23	1.05	1.40
5	2.36	2.56	1.23	1.23	0.00	0.39	2.46
6	2.09	2.28	0.89	1.05	0.39	0.00	2.14
7	0.44	0.42	1.26	1.40	2.46	2.14	0.00

Junto  
(1 2)



AGRUPO pelo min

	(1 2)	3	4	5	6	7
(1 2)	0.00	1.26	1.18	2.36	2.09	0.42
3	1.26	0.00	0.64	1.23	0.89	1.26
4	1.18	0.64	0.00	1.23	1.05	1.40
5	2.36	1.23	1.23	0.00	0.39	2.46
6	2.09	0.89	1.05	0.39	0.00	2.14
7	0.42	1.26	1.40	2.46	2.14	0.00

Junto

(5 6)

→

	(1 2)	3	4	(5 6)	7
(1 2)	0.00	1.26	1.18	2.09	0.42
3	1.26	0.00	0.64	0.89	1.26
4	1.18	0.64	0.00	1.05	1.40
(5 6)	2.09	0.89	1.05	0.00	2.14
7	0.42	1.26	1.40	2.14	0.00

Junto

→

	(1 2 7)	3	4	(5 6)
(1 2 7)	0.00	1.26	1.18	2.09
3	1.26	0.00	0.64	0.89
4	1.18	0.64	0.00	1.05

Junto  
3 e 4

→

	(1 2 7)	(3 4)	(5 6)
(1 2 7)	0.00	1.18	2.09
(3 4)	1.18	0.00	0.89
(5 6)	2.09	0.89	0.00

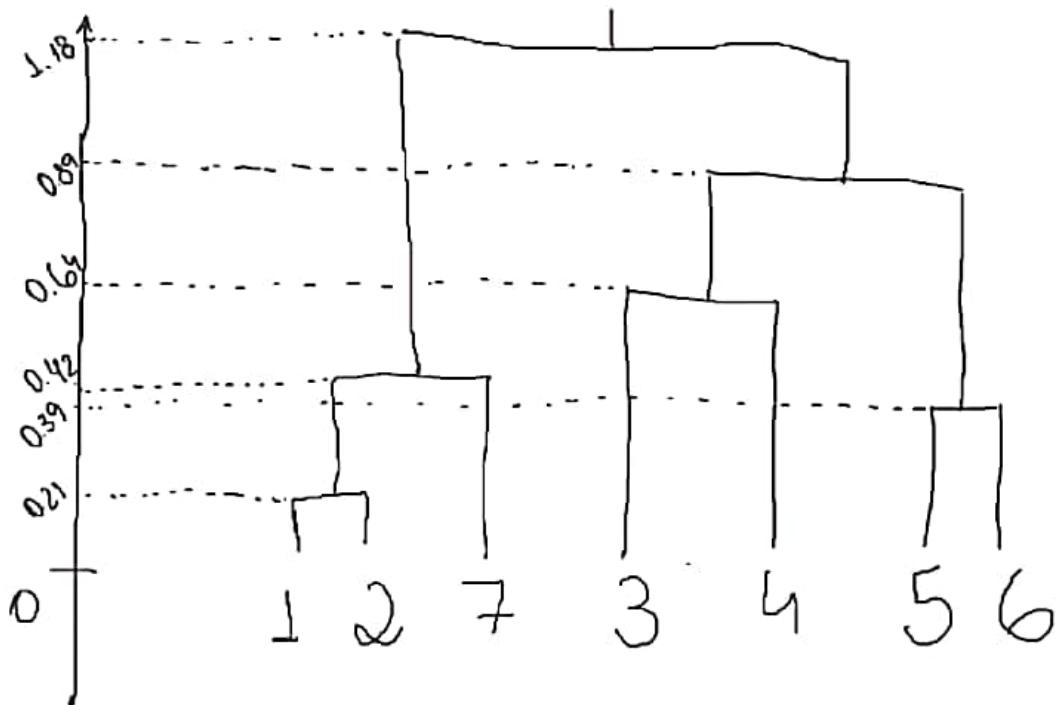
Junto

3 4  
5 6

	(1 2 7)	(3 4 5 6)
(1 2 7)	0.00	1.18
(3 4 5 6)	1.18	0.00

→  
todo mundo

(1 2 3 4 5 6 7)



Ex 1.2)

menor distância

	1	2	3	4	5	6	7
1	0.00	0.21	1.26	1.18	2.36	2.09	0.44
2	0.21	0.00	1.44	1.39	2.56	2.28	0.42
3	1.26	1.44	0.00	0.64	1.23	0.89	1.26
4	1.18	1.39	0.64	0.00	1.23	1.05	1.40
5	2.36	2.56	1.23	1.23	0.00	0.39	2.46
6	2.09	2.28	0.89	1.05	0.39	0.00	2.14
7	0.44	0.42	1.26	1.40	2.46	2.14	0.00

Junto  
(12)



agrupa pelo MAX

	(12)	3	4	5	6	7
(12)	0.00	1.44	1.39	2.56	2.28	0.44
3	1.44	0.00	0.64	1.23	1.26	
4	1.39	0.64	0.00	1.23	1.40	
5	2.56	1.23	1.23	0.00	0.39	2.46
6	2.28	0.89	1.05	0.39	0.00	2.14
7	0.44	1.26	1.40	2.46	2.14	0.00

(5,6)

	(12)	3	4	(5,6)	7
(12)	0.00	1.44	1.39	2.56	0.44
3	1.44	0.00	0.64	1.23	1.26
4	1.39	0.64	0.00	1.23	1.40
(5,6)	2.56	1.23	1.23	0.00	2.46
7	0.44	1.26	1.40	2.46	0.00

(127)

	(127)	3	4	(5,6)
(127)	0.00	1.44	1.40	2.56
3	1.44	0.00	0.64	1.23
4	1.40	0.64	0.00	1.23
(5,6)	2.56	1.23	1.23	0.00

(3,6)

127

	(127)	(34)	(5,6)
(127)	0.00	1.44	2.56
(34)	1.44	0.00	1.23

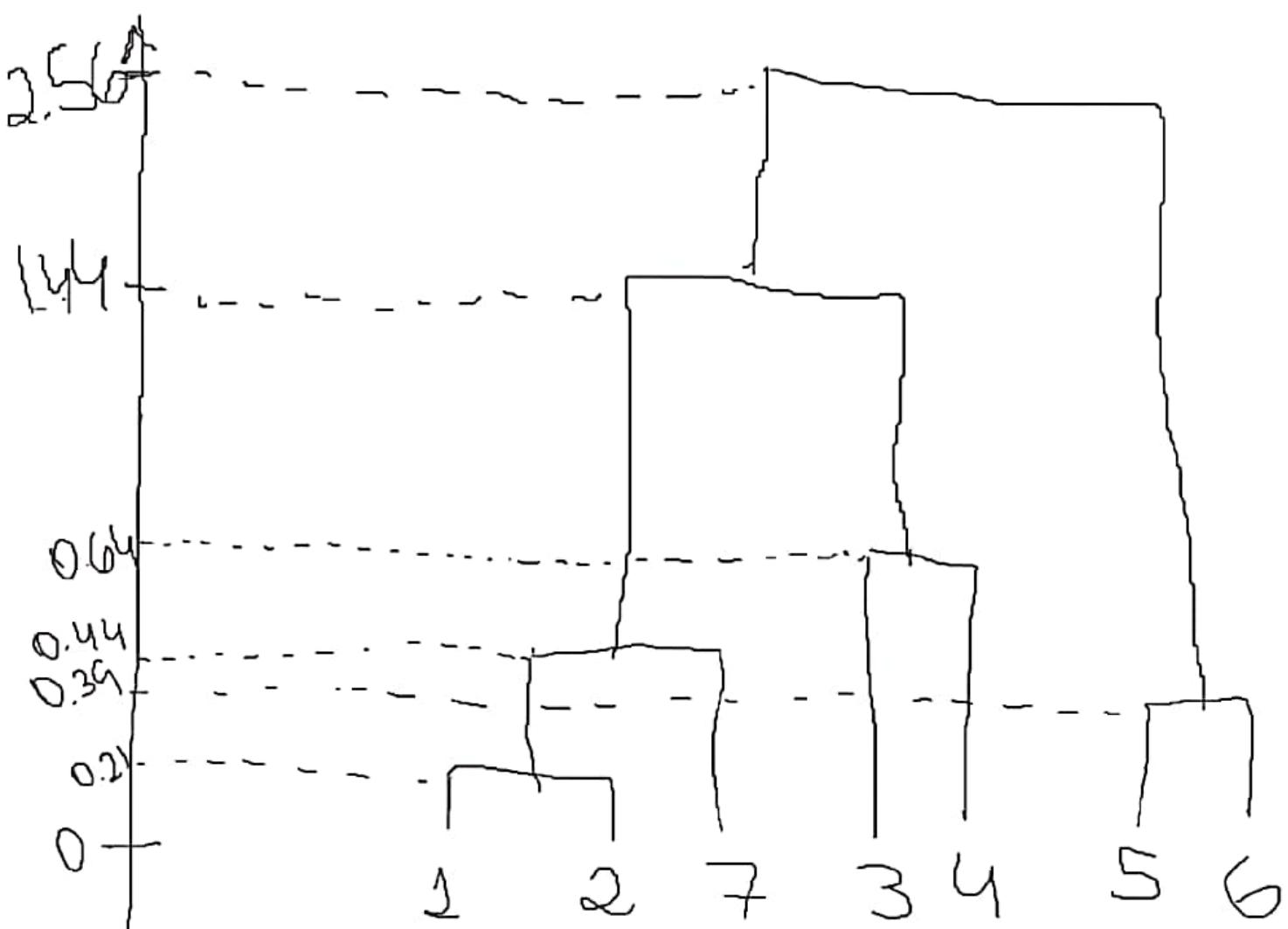
34

(12347)

(5,6)

	(12347)	(5,6)
(12347)	0.00	2.56
(5,6)	2.56	0.00

→ todos juntos



## Exercício 1.3) k-medoides

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
X = np.array(list(range(1,8)))

matriz_distancias = np.array([[0.00, 0.21, 1.26, 1.18, 2.36, 2.09, 0.44],
[0.21, 0.00, 1.44, 1.39, 2.56, 2.28, 0.42],
[1.26, 1.44, 0.00, 0.64, 1.23, 0.89, 1.26],
[1.18, 1.39, 0.64, 0.00, 1.23, 1.05, 1.40],
[2.36, 2.56, 1.23, 1.23, 0.00, 0.39, 2.46],
[2.09, 2.28, 0.89, 1.05, 0.39, 0.00, 2.14],
[0.44, 0.42, 1.26, 1.40, 2.46, 2.14, 0.00]])
```

Neste caso só precisamos da matriz de distâncias, não dos valores dos atributos.

In [3]:

```
# X = np.array([[1.11, 1.05],
#               [0.94, 0.93],
#               [2.36, 1.22],
#               [2.04, 1.78],
#               [3.23, 2.09],
#               [3.08, 1.73],
#               [1.25, 0.63]])

# def distancia_euclidiana(x,y):
#     return np.sqrt((x[0]-y[0])**2+(x[1]-y[1])**2)

# matriz_distancias = np.array([[distancia_euclidiana(x,y) for x in X] for y in X])
```

In [4]:

```
indice_medoides = [2,3]
print("indices dos medoides iniciais\n")
print(indice_medoides)
```

indices dos medoides iniciais

[2, 3]

In [5]:

```

for i in range(5):
    print("iteração "+str(i+1)+"\n")

distancias_aos_medoideos = {}
for j, x in enumerate(X):
    distancias_aos_medoideos[str(x)] = []
    for ind_med in indice_medoideos:
        distancias_aos_medoideos[str(x)].append(matriz_distancias[j,ind_med])

matriz = pd.DataFrame(distancias_aos_medoideos).T
matriz['mais_prox'] = matriz.apply(np.argmin, axis=1)
matriz = matriz.reset_index()
print("matriz de distâncias dos exemplos até os medoídes\n")
print(matriz)

indice_medoideos = []
for grupo in [0,1]:
    lista_aux = list(matriz.query("mais_prox == @grupo").index)
    matriz_aux = matriz_distancias[([[[item]*len(lista_aux) for item in lista_aux]], len(lista_aux)*[lista_aux])
    indice_medoideos.append(lista_aux[np.argmin(np.sum(matriz_aux, axis=0))])
print("\nindice dos medoides atualizados (índice que minimiza a soma das distâncias dos")
print(indice_medoideos)
print("\n")

```

iteração 1

matriz de distâncias dos exemplos até os medoídes

	index	0	1	mais_prox
0	1	1.26	1.18	1
1	2	1.44	1.39	1
2	3	0.00	0.64	0
3	4	0.64	0.00	1
4	5	1.23	1.23	0
5	6	0.89	1.05	0
6	7	1.26	1.40	0

índice dos medoides atualizados (índice que minimiza a soma das distâncias dos elementos daquele cluster até ele

[2, 0]

iteração 2

matriz de distâncias dos exemplos até os medoídes

	index	0	1	mais_prox
0	1	1.26	0.00	1
1	2	1.44	0.21	1
2	3	0.00	1.26	0
3	4	0.64	1.18	0
4	5	1.23	2.36	0
5	6	0.89	2.09	0
6	7	1.26	0.44	1

índice dos medoides atualizados (índice que minimiza a soma das distâncias

dos elementos daquele cluster até ele

[5, 1]

iteração 3

matriz de distâncias dos exemplos até os medoídes

index	0	1	mais_prox
0	1	2.09	0.21
1	2	2.28	0.00
2	3	0.89	1.44
3	4	1.05	1.39
4	5	0.39	2.56
5	6	0.00	2.28
6	7	2.14	0.42

indice dos medoides atualizados (índice que minimiza a soma das distâncias dos elementos daquele cluster até ele

[5, 1]

iteração 4

matriz de distâncias dos exemplos até os medoídes

index	0	1	mais_prox
0	1	2.09	0.21
1	2	2.28	0.00
2	3	0.89	1.44
3	4	1.05	1.39
4	5	0.39	2.56
5	6	0.00	2.28
6	7	2.14	0.42

índice dos medoides atualizados (índice que minimiza a soma das distâncias dos elementos daquele cluster até ele

[5, 1]

iteração 5

matriz de distâncias dos exemplos até os medoídes

index	0	1	mais_prox
0	1	2.09	0.21
1	2	2.28	0.00
2	3	0.89	1.44
3	4	1.05	1.39
4	5	0.39	2.56
5	6	0.00	2.28
6	7	2.14	0.42

índice dos medoides atualizados (índice que minimiza a soma das distâncias dos elementos daquele cluster até ele

[5, 1]



O algoritmo converge (os centroídes param de ser atualizados a partir da iteração 3. Ficamos com os grupos:

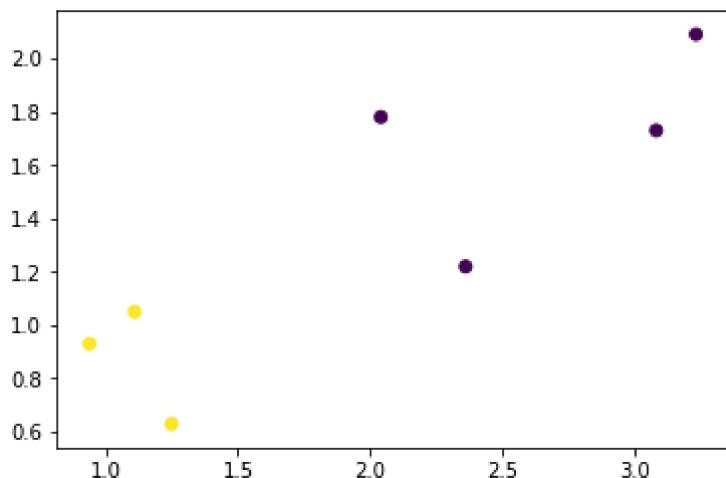
Grupo 0 = {exemplos 3, 4, 5, 6}

Grupo 1 = {exemplos 1, 2, 7}

Visualmente:

In [6]:

```
X = np.array([[1.11, 1.05],  
              [0.94, 0.93],  
              [2.36, 1.22],  
              [2.04, 1.78],  
              [3.23, 2.09],  
              [3.08, 1.73],  
              [1.25, 0.63]])  
  
plt.scatter(X[:,0],X[:,1], c = matriz['mais_prox'])  
plt.show()
```



## Exercício 1.4) k-médias

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
X = np.array([[1.11, 1.05],
              [0.94, 0.93],
              [2.36, 1.22],
              [2.04, 1.78],
              [3.23, 2.09],
              [3.08, 1.73],
              [1.25, 0.63]])
```

In [3]:

```
def distancia_euclidiana(x,y):
    return np.sqrt((x[0]-y[0])**2+(x[1]-y[1])**2)
```

In [4]:

```
centroides = np.array([X[2,:],X[3,:]])
print("centroides iniciais\n")
print(centroides)
```

centroides iniciais

```
[[2.36 1.22]
 [2.04 1.78]]
```

In [5]:

```

for i in range(5):
    print("iteração "+str(i+1)+"\n")

distancias_aos_centroides = {}
for x in X:
    distancias_aos_centroides[str(x)] = []
    for centr in centroides:
        distancias_aos_centroides[str(x)].append(distancia_euclidiana(x,centr))

matriz = pd.DataFrame(distancias_aos_centroides).T
matriz['mais_prox'] = matriz.apply(np.argmin, axis=1)
matriz = matriz.reset_index()
print("matriz de distâncias dos exemplos até os centroídes\n")
print(matriz)

centroides = np.array([
    np.mean(X[matriz.query("mais_prox == 0").index], axis=0),
    np.mean(X[matriz.query("mais_prox == 1").index], axis=0)])
print("\nzentroídes atualizados (médias de cada componente)\n")
print(centroides)
print("\n")

```

iteração 1

matriz de distâncias dos exemplos até os centroídes

	index	0	1	mais_prox
0	[1.11 1.05]	1.261507	1.182286	1
1	[0.94 0.93]	1.449310	1.390144	1
2	[2.36 1.22]	0.000000	0.644981	0
3	[2.04 1.78]	0.644981	0.000000	1
4	[3.23 2.09]	1.230366	1.229715	1
5	[3.08 1.73]	0.882326	1.041201	0
6	[1.25 0.63]	1.257060	1.395206	0

zentroídes atualizados (médias de cada componente)

```
[[2.23      1.19333333]
 [1.83      1.4625      ]]
```

iteração 2

matriz de distâncias dos exemplos até os centroídes

	index	0	1	mais_prox
0	[1.11 1.05]	1.129134	0.829793	1
1	[0.94 0.93]	1.316603	1.037138	1
2	[2.36 1.22]	0.132707	0.582843	0
3	[2.04 1.78]	0.616667	0.380666	1
4	[3.23 2.09]	1.343135	1.534196	0
5	[3.08 1.73]	1.005242	1.278302	0
6	[1.25 0.63]	1.130374	1.014621	1

zentroídes atualizados (médias de cada componente)

```
[[2.89  1.68  ]
 [1.335 1.0975]]
```

iteração 3

matriz de distâncias dos exemplos até os centroídes

	index	0	1	mais_prox
0	[1.11 1.05]	1.888200	0.229959	1
1	[0.94 0.93]	2.089258	0.429047	1
2	[2.36 1.22]	0.701783	1.032294	0
3	[2.04 1.78]	0.8555862	0.981240	0
4	[3.23 2.09]	0.532635	2.139178	0
5	[3.08 1.73]	0.196469	1.856093	0
6	[1.25 0.63]	1.947332	0.475164	1

centroídes atualizados (médias de cada componente)

```
[[2.6775 1.705 ]
 [1.1     0.87 ]]
```

iteração 4

matriz de distâncias dos exemplos até os centroídes

	index	0	1	mais_prox
0	[1.11 1.05]	1.698847	0.180278	1
1	[0.94 0.93]	1.902507	0.170880	1
2	[2.36 1.22]	0.579682	1.307708	0
3	[2.04 1.78]	0.641897	1.308320	0
4	[3.23 2.09]	0.673410	2.454649	0
5	[3.08 1.73]	0.403276	2.158703	0
6	[1.25 0.63]	1.787003	0.283019	1

centroídes atualizados (médias de cada componente)

```
[[2.6775 1.705 ]
 [1.1     0.87 ]]
```

iteração 5

matriz de distâncias dos exemplos até os centroídes

	index	0	1	mais_prox
0	[1.11 1.05]	1.698847	0.180278	1
1	[0.94 0.93]	1.902507	0.170880	1
2	[2.36 1.22]	0.579682	1.307708	0
3	[2.04 1.78]	0.641897	1.308320	0
4	[3.23 2.09]	0.673410	2.454649	0
5	[3.08 1.73]	0.403276	2.158703	0
6	[1.25 0.63]	1.787003	0.283019	1

centroídes atualizados (médias de cada componente)

```
[[2.6775 1.705 ]
 [1.1     0.87 ]]
```

O algoritmo converge (os centroídes param de ser atualizados a partir da iteração 3. Ficamos com os grupos:

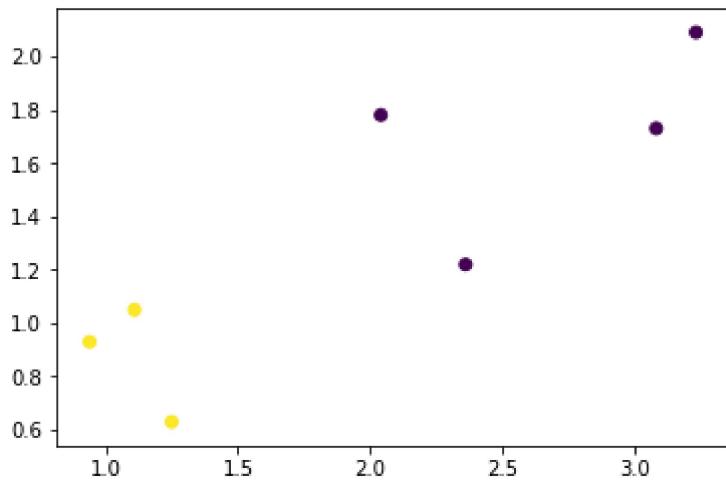
Grupo 0 = {exemplos [2.36, 1.22], [2.04, 1.78], [3.23, 2.09], [3.08, 1.73]}

Grupo 1 = {exemplos [1.11, 1.05], [0.94, 0.93], [1.25 0.63]}

Visualmente:

In [6]:

```
plt.scatter(X[:,0],X[:,1], c = matriz['mais_prox'])
plt.show()
```



# Exercício 3

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
```

## Carregando dados

In [2]:

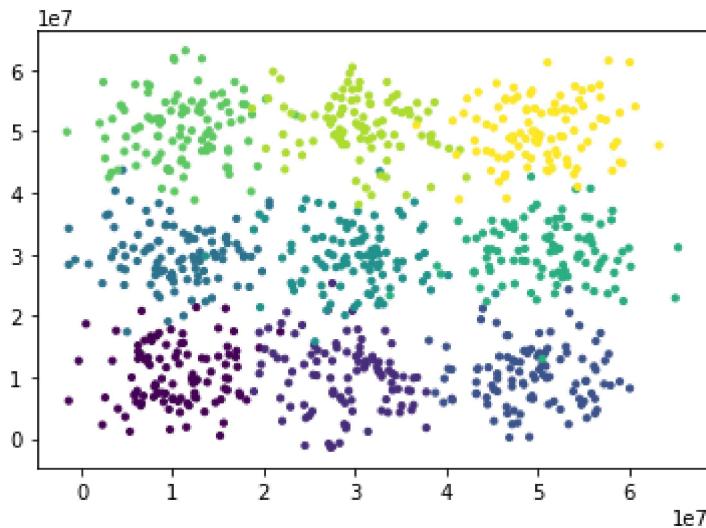
```
df = pd.read_csv("9Gauss.csv", sep=';', names=['x', 'y', 'golden_truth'], header=None)
```

In [3]:

```
X = np.array(df.iloc[:,[0,1]])
```

In [4]:

```
plt.scatter(X[:,0],X[:,1], s=10, c = df.golden_truth)
plt.show()
```



## Uma inicialização do k-Means

In [5]:

```
def distancia_euclidiana(x,y): # ao quadrado
    return ((x[0]-y[0])**2+(x[1]-y[1])**2)
```

In [6]:

```
centroides = np.column_stack([np.random.uniform(min(X[:,0]),max(X[:,0]), size=9),
                             np.random.uniform(min(X[:,1]),max(X[:,1]), size=9)])
```

In [7]:

```
J = np.inf
for i in range(100):

    distancias_aos_centroides = {}
    for j, x in enumerate(X):
        distancias_aos_centroides[j] = []
        for centr in centroides:
            distancias_aos_centroides[j].append(distancia_euclidiana(x,centr))

    matriz = pd.DataFrame(distancias_aos_centroides).T
    matriz['mais_prox'] = matriz.apply(np.argmin, axis=1)
    matriz = matriz.reset_index()

    J_new = sum([matriz.query("mais_prox == @grupo")[grupo].sum()
                 for grupo in range(len(centroides))])

    if J_new >= J: # neste caso não atualizou
        J = J_new
        break
    else:
        J = J_new
        centroides = np.array([
            np.mean(X[matriz.query("mais_prox == @grupo").index], axis=0)
            for grupo in range(len(centroides))])
```

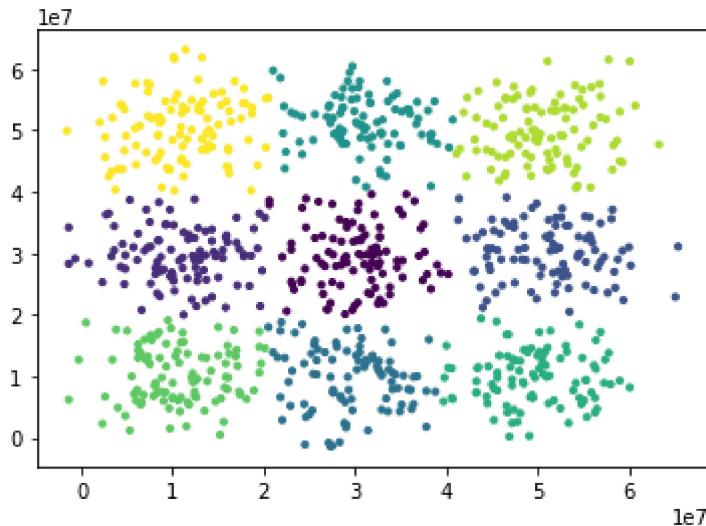
In [8]:

```
print(J)
print(i)
```

4.020265467671727e+16  
22

In [9]:

```
plt.scatter(X[:,0],X[:,1], s=10, c = matriz['mais_prox'])
plt.show()
```



## Várias inicializações

In [10]:

```

lista_J = []
for t in tqdm(range(100)):

    centroides = np.column_stack([np.random.uniform(min(X[:,0]),max(X[:,0]), size=9),
                                    np.random.uniform(min(X[:,1]),max(X[:,1]), size=9)])
    J = np.infty
    for i in range(100):

        distancias_aos_centroides = {}
        for j, x in enumerate(X):
            distancias_aos_centroides[j] = []
            for centr in centroides:
                distancias_aos_centroides[j].append(distancia_euclidiana(x,centr))

        matriz = pd.DataFrame(distancias_aos_centroides).T
        matriz['mais_prox'] = matriz.apply(np.argmin, axis=1)
        matriz = matriz.reset_index()

        J_new = sum([matriz.query("mais_prox == @grupo")[grupo].sum()
                     for grupo in range(len(centroides))])

        if J_new >= J: # neste caso não atualizou
            J = J_new
            break
        else:
            J = J_new
            centroides = np.array([
                np.mean(X[matriz.query("mais_prox == @grupo").index], axis=0)
                for grupo in range(len(centroides))])

    lista_J.append(J)
if J == min(lista_J):
    best = matriz['mais_prox']

```

66% |██████████| 66/100 [05:44<03:55, 6.92s/it] C:\Users\carlo\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3118: RuntimeWarning: Mean of empty slice.  
out=out, \*\*kwargs)  
C:\Users\carlo\Anaconda3\lib\site-packages\numpy\core\\_methods.py:78: RuntimeWarning: invalid value encountered in true\_divide  
ret, rcount, out=out, casting='unsafe', subok=False)  
100% |██████████| 100/100 [08:36<00:00, 6.22s/it]

In [11]:

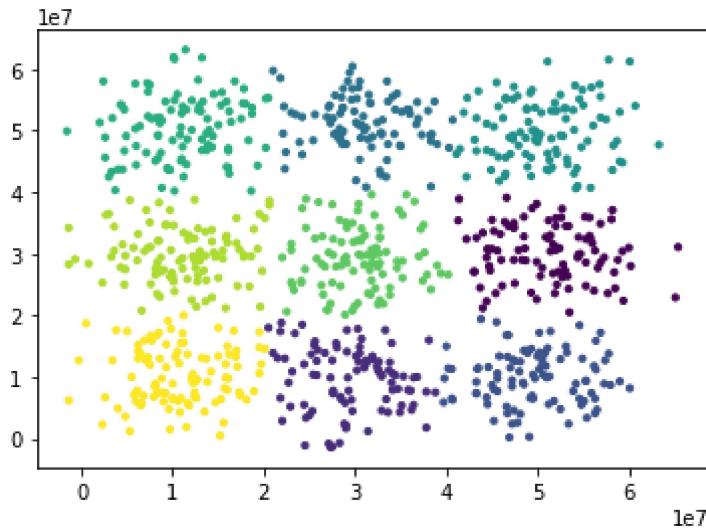
min(lista\_J)

Out[11]:

4.019175177941683e+16

In [12]:

```
plt.scatter(X[:,0],X[:,1], s=10, c = best)
plt.show()
```



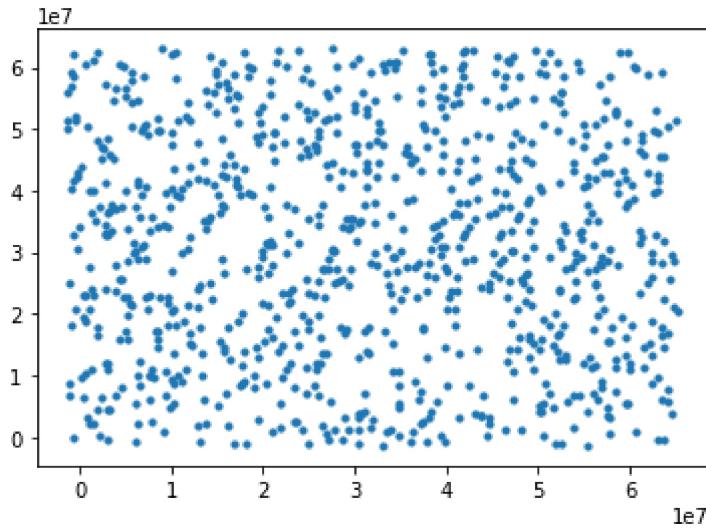
## Dados uniformemente gerados

In [13]:

```
X_uni = np.column_stack([np.random.uniform(min(X[:,0]),max(X[:,0]), size=900),
                           np.random.uniform(min(X[:,1]),max(X[:,1]), size=900)])
```

In [14]:

```
plt.scatter(X_uni[:,0],X_uni[:,1],s=10)
plt.show()
```



In [15]:

```
lista_J_uni = []
for t in tqdm(range(100)):

    centroides = np.column_stack([np.random.uniform(min(X[:,0]),max(X[:,0]), size=9),
                                    np.random.uniform(min(X[:,1]),max(X[:,1]), size=9)])
    J = np.infty
    for i in range(100):

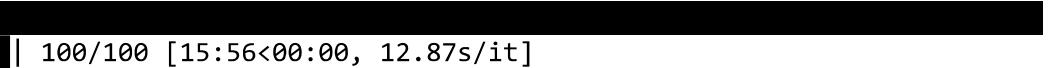
        distancias_aos_centroides = {}
        for j, x in enumerate(X_uni):
            distancias_aos_centroides[j] = []
            for centr in centroides:
                distancias_aos_centroides[j].append(distancia_euclidiana(x,centr))

        matriz = pd.DataFrame(distancias_aos_centroides).T
        matriz['mais_prox'] = matriz.apply(np.argmin, axis=1)
        matriz = matriz.reset_index()

        J_new = sum([matriz.query("mais_prox == @grupo")[grupo].sum()
                     for grupo in range(len(centroides))])

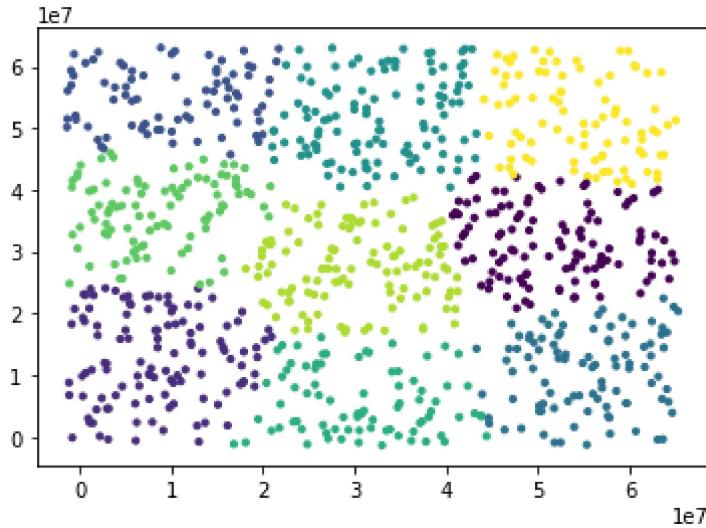
        if J_new >= J: # neste caso não atualizou
            J = J_new
            break
        else:
            J = J_new
            centroides = np.array([
                np.mean(X_uni[matriz.query("mais_prox == @grupo").index], axis=0)
                for grupo in range(len(centroides))])

    lista_J_uni.append(J)
if J == min(lista_J_uni):
    best = matriz['mais_prox']
```

100% |  | 100/100 [15:56<00:00, 12.87s/it]

In [16]:

```
plt.scatter(X_uni[:,0],X_uni[:,1],s=10, c=best)
plt.show()
```



In [17]:

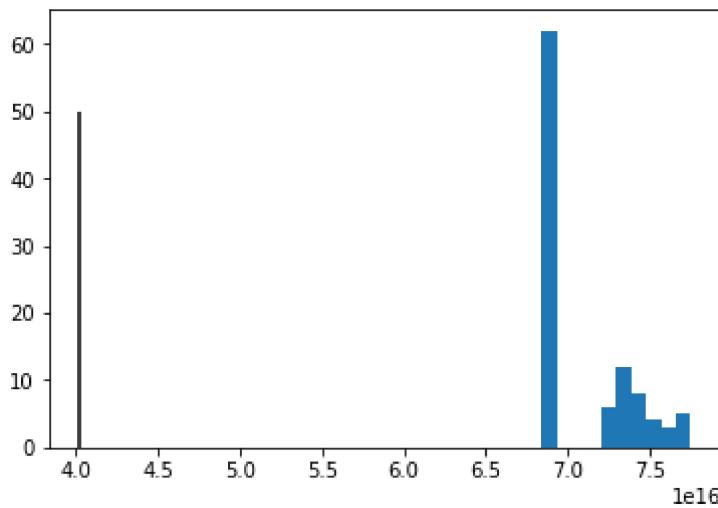
```
min(lista_J_uni)
```

Out[17]:

6.839661073509089e+16

In [18]:

```
plt.vlines(min(lista_J),0,50)
plt.hist(lista_J_uni)
plt.show()
```



In [19]:

```
sum(lista_J_uni > min(lista_J))
```

Out[19]:

100

Ficamos confiante que os cluster encontrados na 9Gauss fazem sentido e não são apenas encontrados ao

acaso.

## exercício 5)

transação	itens				
1	a	b	d	e	
2		b	c	d	
3	a	b		d	e
4	a		c	d	e
5		b	c	d	e
6		b		d	e
7			c	d	
8	a	b	c		
9	a		d	e	
10		b	d		

Itemsets de tamanho 1

itemset	σ	contagem de suporte
a	5	
b	7	
c	5	
d	9	
e	6	

Como todos itemsets de tamanho 1 tem suporte maior que minsup não temos nenhum para nos itemsets de tamanho 2.

itemset	σ
a b	3
a c	2
a d	4
a e	4
b c	3
b d	6
b e	4
c d	4
c e	2
d e	6

Como  $F_2$  não são todos, geramos  $L_3$  olhando a ordem lexicográfica. Combinando prefixos de tamanho 1 que sejam iguais.

$$(a \underline{b}) + (\underline{a} d) = (\underline{a} b d) \text{ OK!}$$

$$(a \underline{b}) + (\underline{a} \underline{c}) = \text{NÃO} / (a \underline{b}) + (\underline{b} d) = \text{NÃO}$$

itemsets de tamanho 3:

itemset	$\sigma$
a b d	2 x
a b e	2 x
a d e	4
b c d	2 x
b c e	1 x
b d e	4

Paramos aqui  
pois (ade) e  
(bde) não com-  
partilham prefixo.  
(Note que abde não  
pode já que abd  
não tem o sufi-  
ciente já).

Ficamos então com os itemsets frequentes:

$\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4 \cup \mathcal{F}_5$  com,

$$\mathcal{F}_1 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$$

$$\mathcal{F}_2 = \{\{a,b\}, \{a,d\}, \{a,e\}, \{b,c\}, \{b,d\}, \{b,e\}, \\ \{b,e\}, \{c,d\}, \{d,e\}\}$$

$$\mathcal{F}_3 = \{\{a,d,e\}, \{b,d,e\}\}$$

$$\mathcal{F}_4 = \mathcal{F}_5 = \emptyset$$