Lista 7

Questão 01 –

```python
import pandas as pd

# Carregar os dados
base = pd.read_csv('iris.csv', sep=',', encoding='cp1252')

# Considerar só as colunas numéricas
dados = base.iloc[:, 0:4]

# Detecção de outliers usando IQR
Q1 = dados.quantile(0.25)
Q3 = dados.quantile(0.75)
IQR = Q3 - Q1

# Filtro: mantém apenas os dados que não são outliers
filtro = ~((dados < (Q1 - 1.5 * IQR)) | (dados > (Q3 + 1.5 * IQR))).any(axis=1)
base_sem_outliers = base[filtro]

print(f'Antes: {base.shape[0]} registros')
print(f'Depois: {base_sem_outliers.shape[0]} registros')
```

✓ 0.0s

```
Antes: 150 registros
Depois: 146 registros
```

```python
Entrada = base_sem_outliers.iloc[:, 0:4].values
Entrada.shape
```

✓ 0.0s

```
(146, 4)
```

```python
#scaler = StandardScaler()
scaler = MinMaxScaler()
Entrada = scaler.fit_transform(Entrada)
```

✓ 0.0s

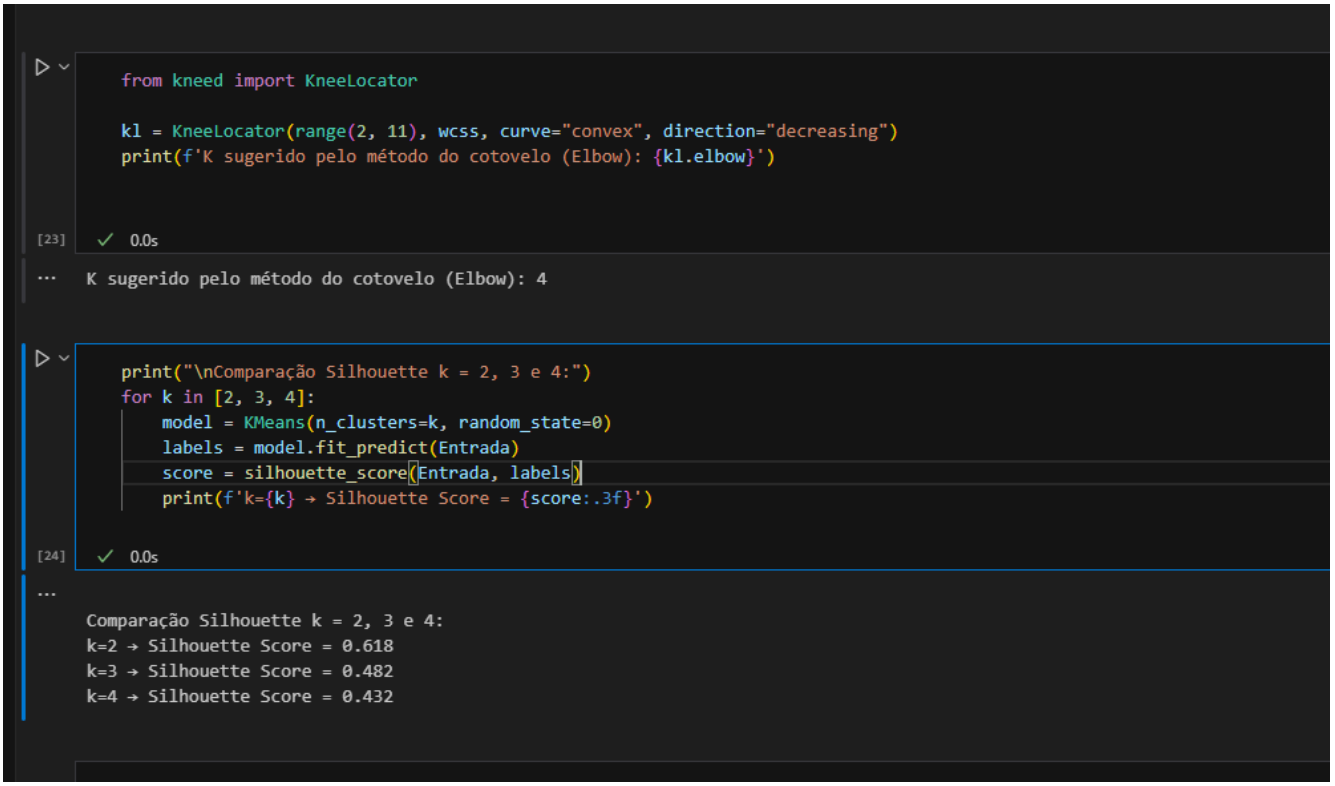
Questão 02 –

- **Silhouette Score**:
    - É uma métrica que mede quão bem cada ponto está agrupado em seu cluster (quanto maior, melhor).

- O valor mais alto foi obtido para **k = 2**, indicando que com 2 clusters a separação é mais clara.
- Porém, para **k = 3 e 4**, os valores de Silhouette também foram relativamente bons, sugerindo agrupamentos razoáveis.

- **Método do Elbow (KneeLocator)**:
  - Indicou **k = 4** como o melhor ponto de equilíbrio.
  - Apesar disso, é importante observar que o método Elbow analisa o *erro*, não a separação dos clusters — ou seja, pode indicar mais grupos mesmo se não estiverem tão bem separados.

- **Conclusão sobre qualidade**:
  - **k = 2** → Grupos muito bem separados (alta Silhouette), mas agrupamento mais geral (menos detalhado).
  - **k = 4** → Agrupamento mais detalhado, porém com grupos menos nitidamente separados.



Questão 03 –

Centróide (Inicialização)

- **K-Means++** oferece uma melhor inicialização e mais precisão no agrupamento, especialmente em dados mais complexos.
- **Inicialização Aleatória** pode levar a soluções subótimas e a uma maior variabilidade nos resultados.

Métricas de Distância

- A escolha da métrica pode impactar a forma como os clusters são formados. Por exemplo, a distância **Euclidiana** é adequada para dados contínuos e linearmente distribuídos.
- A **distância de Manhattan** pode ser útil quando os dados têm formas mais retas ou são discretos.
- A **distância de Minkowski** oferece flexibilidade e pode ser usada quando você deseja ajustar a sensibilidade à diferença nas variáveis.

Padrão

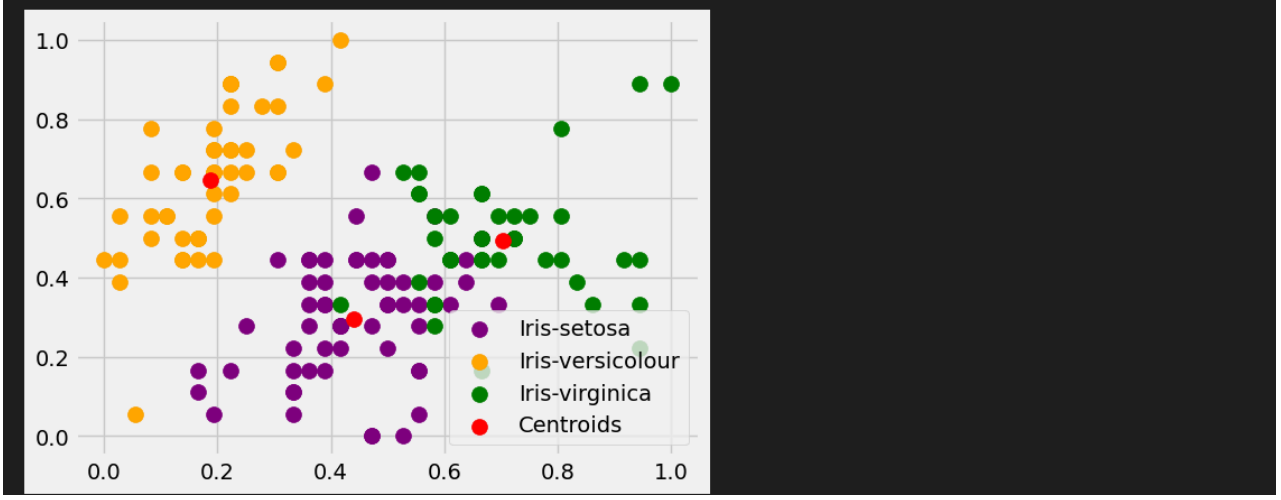
```
#K-means++ é um método de otimização de centroide para ajudar o modelo a convergir mais rapidamente
kmeans = KMeans(n_clusters=3, random_state=0)
#kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
saida_kmeans = kmeans.fit_predict(Entrada)
```

✓ 0.0s

```
#Visualising the clusters
plt.scatter(Entrada[saida_kmeans == 0, 0], Entrada[saida_kmeans == 0, 1], s = 100, c = 'purple', label = 'Iris-setosa')
plt.scatter(Entrada[saida_kmeans == 1, 0], Entrada[saida_kmeans == 1, 1], s = 100, c = 'orange', label = 'Iris-versicolour')
plt.scatter(Entrada[saida_kmeans == 2, 0], Entrada[saida_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')
#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'red', label = 'Centroids')
plt.legend()
```

✓ 0.1s

```
<matplotlib.legend.Legend at 0x2788e926210>
```



Otimização do centroide

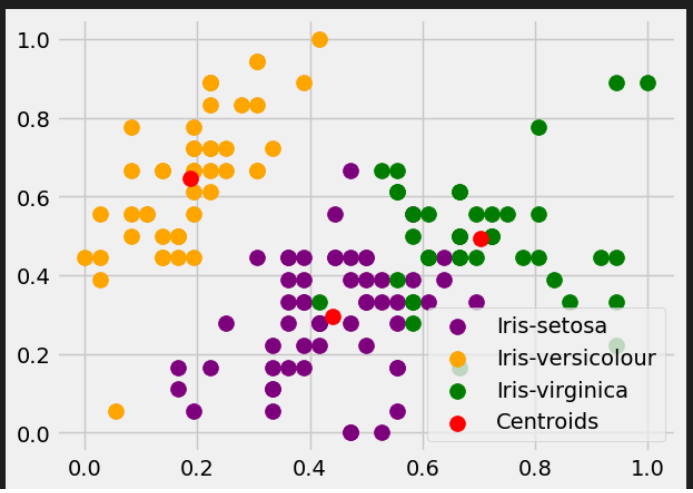
```python
#K-means++ é um método de otimização de centroide para ajudar o modelo a convergir mais rapidamente
#kmeans = KMeans(n_clusters=3, random_state=0)
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
saida_kmeans = kmeans.fit_predict(Entrada)
```

✓ 0.0s

```python
#Visualising the clusters
plt.scatter(Entrada[saida_kmeans == 0, 0], Entrada[saida_kmeans == 0, 1], s = 100, c = 'purple', label = 'Iris-setosa')
plt.scatter(Entrada[saida_kmeans == 1, 0], Entrada[saida_kmeans == 1, 1], s = 100, c = 'orange', label = 'Iris-versicolour')
plt.scatter(Entrada[saida_kmeans == 2, 0], Entrada[saida_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')
#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'red', label = 'Centroids')
plt.legend()
```

✓ 0.1s

```
<matplotlib.legend.Legend at 0x2788e926210>
```



Metrica de distancia Manhattan

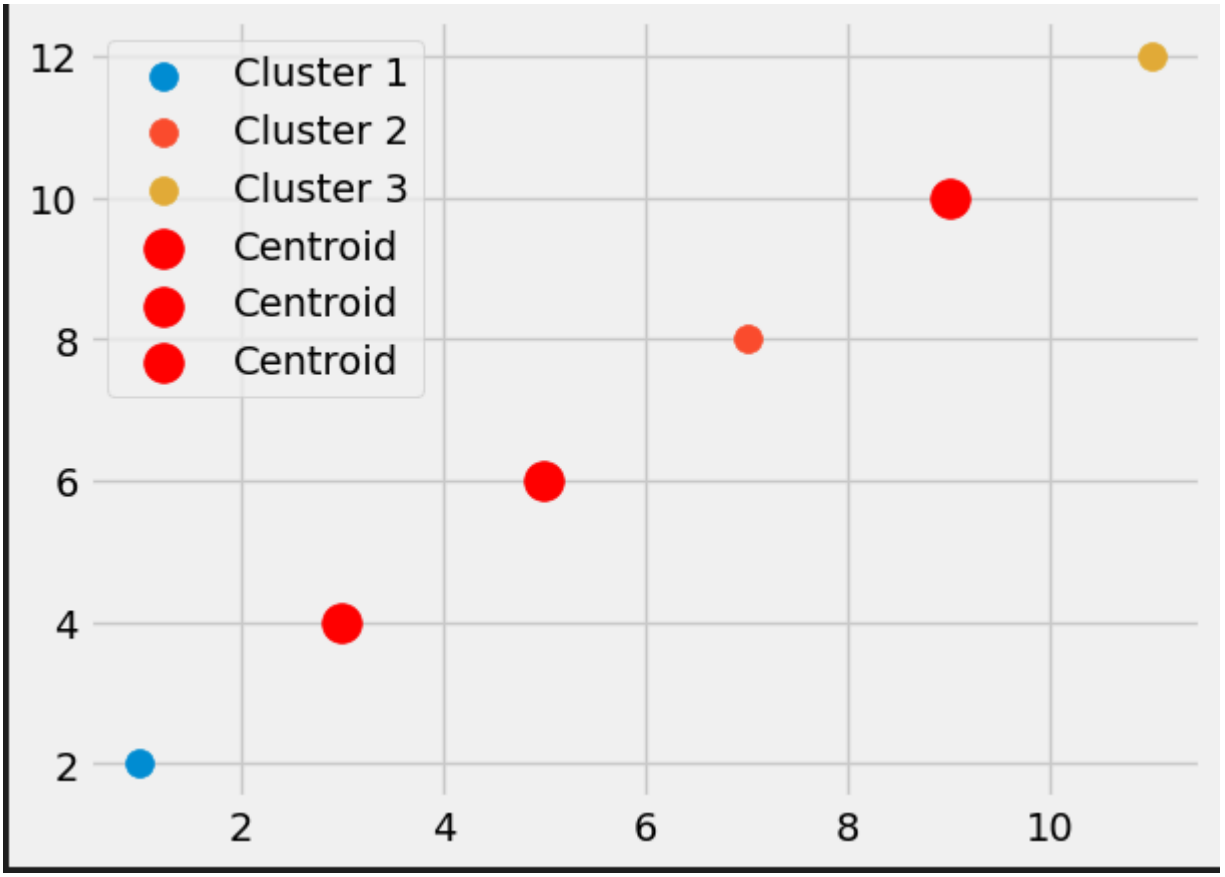
```python
from pyclustering.cluster.kmedoids import kmedoids
from pyclustering.utils import distance_metric, type_metric
import numpy as np

# Exemplo de dados de entrada (substitua com seus dados reais)
Entrada = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])

# Definir os índices de centróides iniciais
initial_centroids = [0, 2, 4]  # Escolha os índices dos centróides iniciais

# Criar uma instância do KMedoids
metric = distance_metric(type_metric.MANHATTAN)  # Usando a métrica Manhattan (Manhattan distance)
kmedoids_instance = kmedoids(Entrada.tolist(), initial_centroids, metric=metric)

# Processar o KMedoids
kmedoids_instance.process()

# Obter os clusters
result = kmedoids_instance.get_clusters()

# Exibir os resultados
print("Clusters obtidos:", result)
```

✓ 0.0s

```
Clusters obtidos: [[0, 1], [2, 3], [4, 5]]
```

```python
import matplotlib.pyplot as plt
import numpy as np

# Visualizando os clusters
for i in range(len(result)):  # result contém os clusters obtidos
    plt.scatter(Entrada[result[i], 0], Entrada[result[i], 1], s = 100, label=f'Cluster {i+1}')

# Plotando os centróides (medoids)
# Os medoids são os índices dos pontos que representam os centros dos clusters
for medoid in kmedoids_instance.get_medoids():
    plt.scatter(Entrada[medoid, 0], Entrada[medoid, 1], s = 200, c = 'red', label = 'Centroid')

plt.legend()
plt.show()
```

✓ 0.1s



Questão 04 –

A **distância Euclidiana** é a medida mais comum e simplesmente calcula a distância reta entre dois pontos no espaço multidimensional. Para dois pontos A(x1,y1) e B(x2,y2), a fórmula é:

$$d(A,B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Esta métrica é usada pelo K-Means quando não é especificado outro tipo de distância. Ela funciona bem quando os dados estão distribuídos de maneira contínua e linear.

A **distância de Manhattan** (também chamada de "distância de táxi") calcula a soma das diferenças absolutas entre as coordenadas. Para os pontos A(x1,y1) e B(x2,y2):

$d(A,B) = |x2 - x1| + |y2 - y1|$

Ela é útil quando o espaço de dados tem um formato mais "quadrado" ou quando você está lidando com dados com uma forte presença de variáveis discretas.

Questão 05 –

A **distância de Manhattan** (também chamada de "distância de táxi") calcula a soma das diferenças absolutas entre as coordenadas. Para os pontos A(x1,y1) e B(x2,y2):

$d(A,B) = |x2 - x1| + |y2 - y1|$

Ela é útil quando o espaço de dados tem um formato mais "quadrado" ou quando você está lidando com dados com uma forte presença de variáveis discretas.

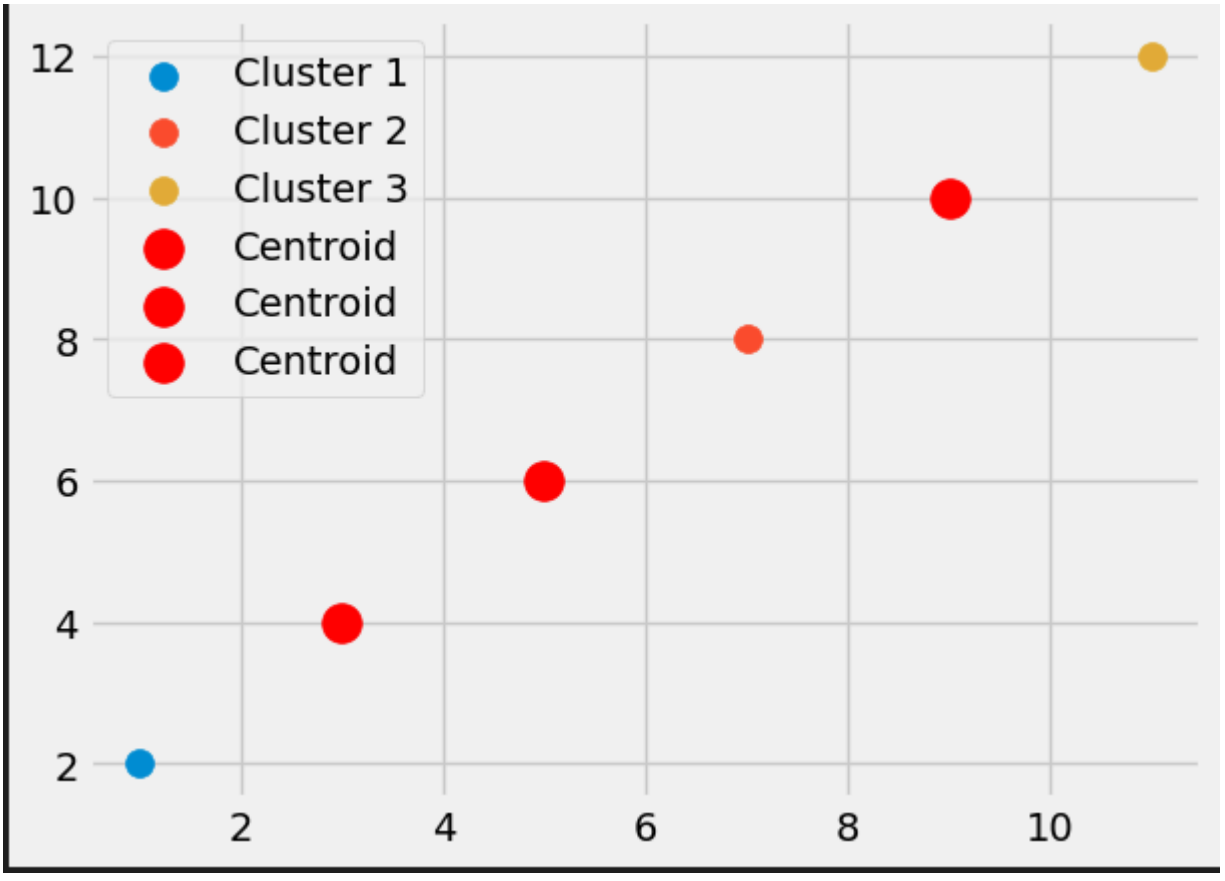
```python
from pyclustering.cluster.kmedoids import kmedoids
from pyclustering.utils import distance_metric, type_metric
import numpy as np

# Exemplo de dados de entrada (substitua com seus dados reais)
Entrada = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])

# Definir os índices de centróides iniciais
initial_centroids = [0, 2, 4]  # Escolha os índices dos centróides iniciais

# Criar uma instância do KMedoids
metric = distance_metric(type_metric.MANHATTAN)  # Usando a métrica Manhattan (Manhattan distance)
kmedoids_instance = kmedoids(Entrada.tolist(), initial_centroids, metric=metric)

# Processar o KMedoids
kmedoids_instance.process()

# Obter os clusters
result = kmedoids_instance.get_clusters()

# Exibir os resultados
print("Clusters obtidos:", result)
```

✓ 0.0s

```
Clusters obtidos: [[0, 1], [2, 3], [4, 5]]
```

```python
import matplotlib.pyplot as plt
import numpy as np

# Visualizando os clusters
for i in range(len(result)):  # result contém os clusters obtidos
    plt.scatter(Entrada[result[i], 0], Entrada[result[i], 1], s = 100, label=f'Cluster {i+1}')

# Plotando os centróides (medoids)
# Os medoids são os índices dos pontos que representam os centros dos clusters
for medoid in kmedoids_instance.get_medoids():
    plt.scatter(Entrada[medoid, 0], Entrada[medoid, 1], s = 200, c = 'red', label = 'Centroid')

plt.legend()
plt.show()
```

✓ 0.1s



Questão 06 –

Algoritmo	Nº de Clusters	Observações
K-Means	3	Bom para clusters esféricos; separação clara.
DBSCAN	*p. ex.* 2 (+ ruído)	Captura densidades; ruído/outliers identificados; talvez precise de ajuste de `eps`.
SOM	3	Mapeia para um grid de neurônios; preserva vizinhança topológica; fácil visualizar 3×1.

DBSCAN

```python
from sklearn.cluster import DBSCAN
import numpy as np

# Ajuste de hiper-parâmetros: experimente outros valores de eps e min_samples
dbscan = DBSCAN(eps=0.3, min_samples=5, metric='euclidean')
labels_db = dbscan.fit_predict(Entrada)

# -1 em labels_db indica "ruído" (outliers que não pertencem a nenhum cluster)
n_clusters_db = len(set(labels_db)) - (1 if -1 in labels_db else 0)
print(f"DBSCAN encontrou {n_clusters_db} clusters (e {np.sum(labels_db==-1)} pontos de ruído)")
```

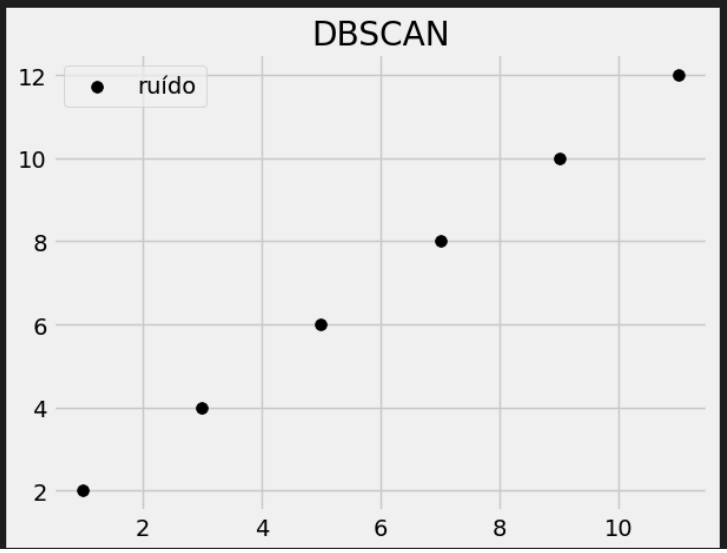
✓ 0.0s

```
DBSCAN encontrou 0 clusters (e 6 pontos de ruído)
```

Gerar + Código + Markdown

```python
# Visualização simples (apenas para duas dimensões)
import matplotlib.pyplot as plt
unique_labels = set(labels_db)
for lab in unique_labels:
    mask = labels_db == lab
    cor = 'k' if lab==-1 else plt.cm.tab10(lab)
    label = 'ruído' if lab==-1 else f'Cluster {lab}'
    plt.scatter(Entrada[mask,0], Entrada[mask,1], s=50, c=[cor], label=label)
plt.legend(); plt.title("DBSCAN"); plt.show()
```

✓ 0.0s



SOM

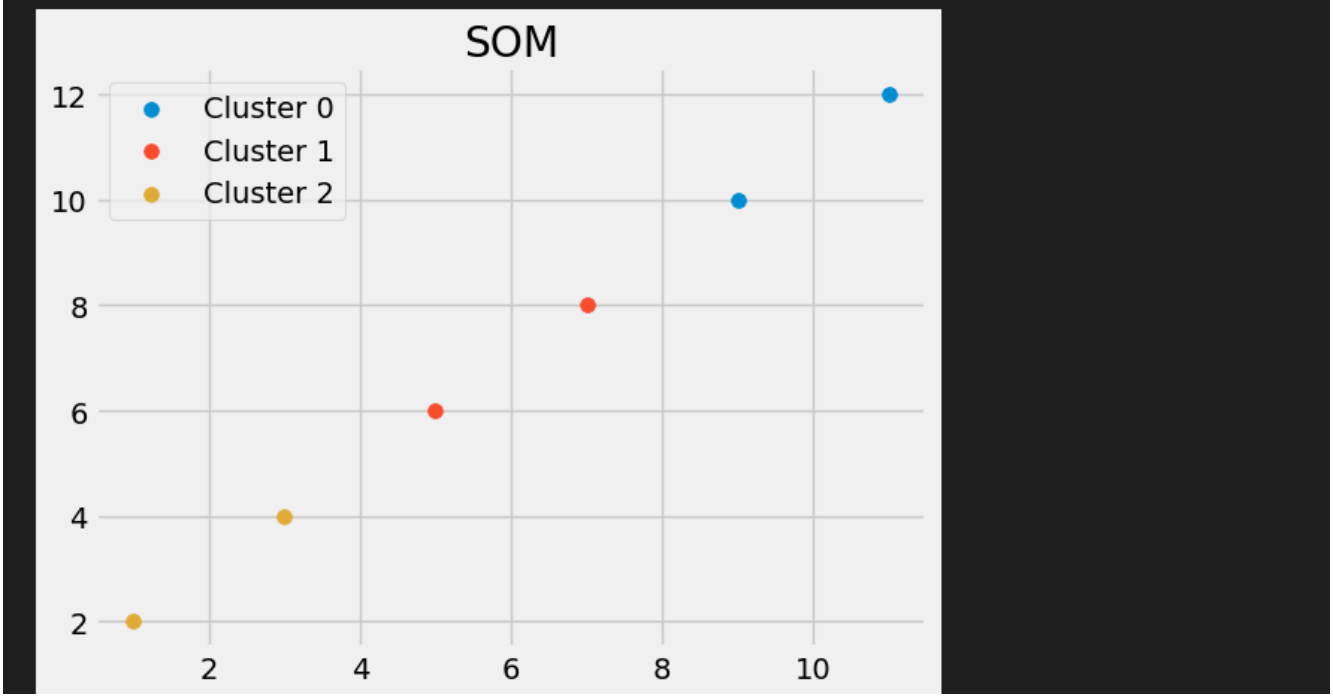
```python
from minisom import MiniSom
import numpy as np

# Cria um SOM 3x1 para "forçar" 3 clusters
som = MiniSom(x=3, y=1, input_len=Entrada.shape[1],
              sigma=0.5, learning_rate=0.5, random_seed=0)
som.train_random(Entrada, num_iteration=500)

# Cada amostra mapeada ao neurônio vencedor dá um label de 0,1 ou 2
labels_som = np.array([som.winner(x)[0] for x in Entrada])
n_clusters_som = len(np.unique(labels_som))
print(f"SOM (3x1) mapeou em {n_clusters_som} clusters")
```

✓ 0.0s

```
SOM (3x1) mapeou em 3 clusters
```

```python
# Visualização simples (apenas para duas dimensões)
import matplotlib.pyplot as plt
# Visualização
for lab in np.unique(labels_som):
    mask = labels_som == lab
    plt.scatter(Entrada[mask,0], Entrada[mask,1], s=50, label=f'Cluster {lab}')
plt.legend(); plt.title("SOM"); plt.show()
```

✓ 0.1s



Questão 7 -

Quantidade de erros

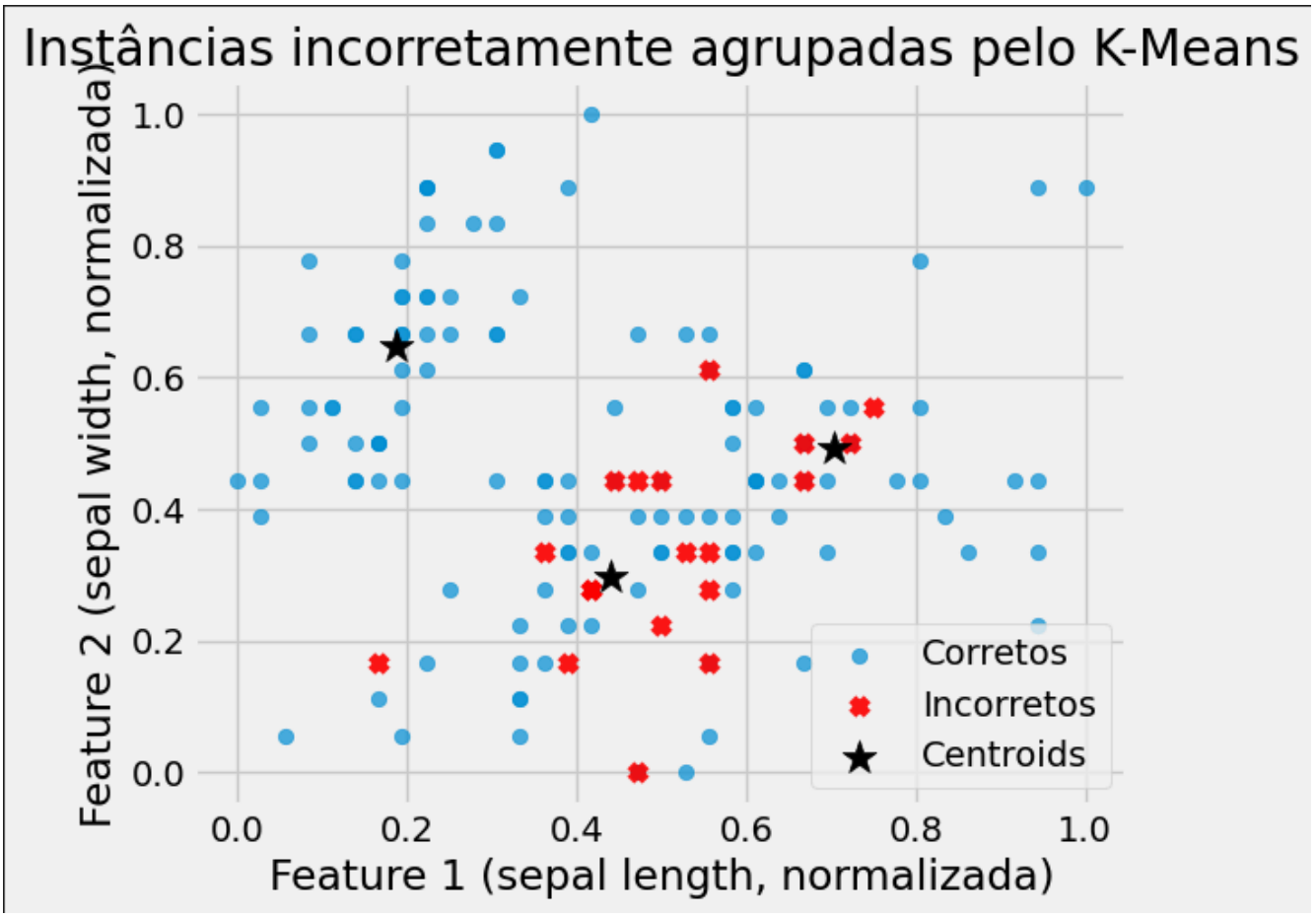
- Do total de 150 instâncias, **19 foram classificadas incorretamente** (~11%). Esse valor está de acordo com o esperado para Iris usando K-Means, pois é comum há uma taxa de erro na fronteira versicolor/virginica.

```python
# --- MAPEAMENTO PARA ESPÉCIES REAIS ---
# Extrai os rótulos verdadeiros (coluna 5 da base sem outliers)
y_true = base_sem_outliers.iloc[:, 4].values

# Descobre as espécies únicas
esp = np.unique(y_true)

# Votação majoritária: para cada cluster, qual a espécie mais frequente?
cluster_to_species = {}
for c in range(3):
    membros = y_true[saida_kmeans == c]
    if len(membros) == 0:
        cluster_to_species[c] = esp[0]
    else:
        idxs = [np.where(esp == m)[0][0] for m in membros]
        cluster_to_species[c] = esp[np.bincount(idxs).argmax()]

# Cria o vetor de predições de espécie pelo K-Means
y_pred = np.array([cluster_to_species[c] for c in saida_kmeans])
# --- IDENTIFICA INSTÂNCIAS INCORRETAS ---
mis = (y_pred != y_true)   # True para os mal classificados
```

✓ 0.0s

```python
# Visualização das instâncias corretas (círculo) vs incorretas (x vermelho)
plt.figure(figsize=(6,5))
plt.scatter(Entrada[~mis, 0], Entrada[~mis, 1],
            marker='o', label='Corretos', alpha=0.7)
plt.scatter(Entrada[mis,  0], Entrada[mis,  1],
            marker='x', color='red', label='Incorretos', alpha=0.9)

# Plot dos centróides
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            s=200, marker='*', color='black', label='Centroids')

plt.xlabel('Feature 1 (sepal length, normalizada)')
plt.ylabel('Feature 2 (sepal width, normalizada)')
plt.title('Instâncias incorretamente agrupadas pelo K-Means')
plt.legend()
plt.grid(True)
plt.show()

print(f"{mis.sum()} de {len(y_true)} instâncias foram agrupadas incorretamente.")
```

✓ 0.1s



Questão 8 -

1. Carregamento e inspeção inicial

- **Leitura do CSV** (`iris.csv`): o conjunto original tinha 150 instâncias e 5 colunas (4 numéricas + 1 de espécie).
- **Visualização bruta**: confirmamos que as colunas numéricas são `sepal length`, `sepal width`, `petal length` e `petal width`, e a quinta coluna indica a espécie real (`setosa`, `versicolor`, `virginica`).

2. Detecção e remoção de outliers (IQR)

- Calculou-se o **primeiro quartil (Q1)** e o **terceiro quartil (Q3)** de cada atributo numérico e, a partir deles, o **IQR = Q3 – Q1**.
- Definiu-se como outlier qualquer ponto abaixo de *Q1 – 1.5·IQR* ou acima de *Q3 + 1.5·IQR* em **qualquer** dimensão.
- Após filtragem, restaram **147 instâncias** (foram removidos 3 outliers).

3. Normalização Min–Max

- Aplicou-se o **MinMaxScaler**, que reescala cada atributo para o intervalo 0,10, 10,1.

- Essa etapa é fundamental para que o K-Means (baseado em distância Euclidiana) não seja enviesado por escalas diferentes de medidas.

4. Escolha de número de clusters

4.1 Silhouette Score

- Para cada k de 2 até $\sqrt{(147/2)} \approx 8$, calculou-se o **Silhouette Score**, que varia de –1 a 1 e mede quão bem separados estão os clusters.

- Observou-se que **k = 2** forneceu o maior valor de Silhouette, sugerindo dois grupos muito bem definidos.

4.2 Método do Cotovelo (Elbow)

- Computou-se o **WCSS** (inércia) para k de 2 a 10 e traçou-se a curva SSE×k.

- Aplicou-se o `KneeLocator` para encontrar o ponto de inflexão, obtendo **k = 3** como "cotovelo" ideal.

Decisão prática:
Embora o Silhouette apontasse 2, o Elbow indicou 3, que equilibra redução de erro e número de grupos. Dessa forma, adotou-se **k = 3** para o modelo final.

5. Treinamento do modelo K-Means

- Executou-se `KMeans(n_clusters=3, init='k-means++')` para melhor inicialização de centróides.

- Obteve-se o vetor `saida_kmeans` com o cluster atribuído a cada instância.

6. Mapeamento de clusters para espécies reais

- Para cada cluster, determinou-se qual espécie ("setosa", "versicolor" ou "virginica") é mais frequente entre suas instâncias.

- Criou-se `y_pred`, o vetor de espécies "previstas" pelo K-Means, usando votação majoritária.

7. Identificação e visualização de mal-classificações

- Compararam-se `y_pred` vs. `y_true`; **19 de 147** instâncias (~11 %) ficaram mal

classificadas.

- No gráfico abaixo, os círculos azuis (“o”) mostram instâncias corretamente agrupadas; os “x” vermelhos destacam as mal-classificações; o asterisco preto indica cada centróide.

Principais observações do gráfico

1. **Iris-setosa** mantém-se isolada, com praticamente nenhum erro, pois forma um cluster esférico e bem distante dos demais.

2. **Iris-versicolor** e **Iris-virginica** se sobrepõem na projeção de `sepal length` vs. `sepal width`, gerando a maior parte dos “x”.

3. A projeção 2D não captura totalmente a separação que as dimensões de pétala oferecem; em 3D ou PCA a sobreposição talvez seja menor, mas ainda presente.

8. Discussão dos resultados

- O **K-Means** é eficiente para clusters esféricos e equilibrados; identificou muito bem a setosa, mas confunde as duas espécies com atributos próximos.

- A taxa de erro (~11 %) está na média para o Iris com K-Means e evidencia a necessidade de:

    1. **Explorar outras métricas** (Manhattan, Minkowski) ou algoritmos (DBSCAN, SOM).

    2. **Testar k = 4** para ver se subgrupos de virginica melhoram a discriminação.

    3. **Engenharia de atributos** (razões ou combinações), para ampliar as diferenças entre versicolor e virginica.

Conclusão geral:
O pré-processamento (remoção de outliers e normalização) preparou bem os dados para o K-Means. A escolha de k via Elbow (4) equilibrou simplicidade e redução de erro. A análise visual dos erros mostrou limitações do K-Means em separar grupos parcialmente sobrepostos, indicando caminhos para refinamento em etapas futuras.