LIsta de IA #7

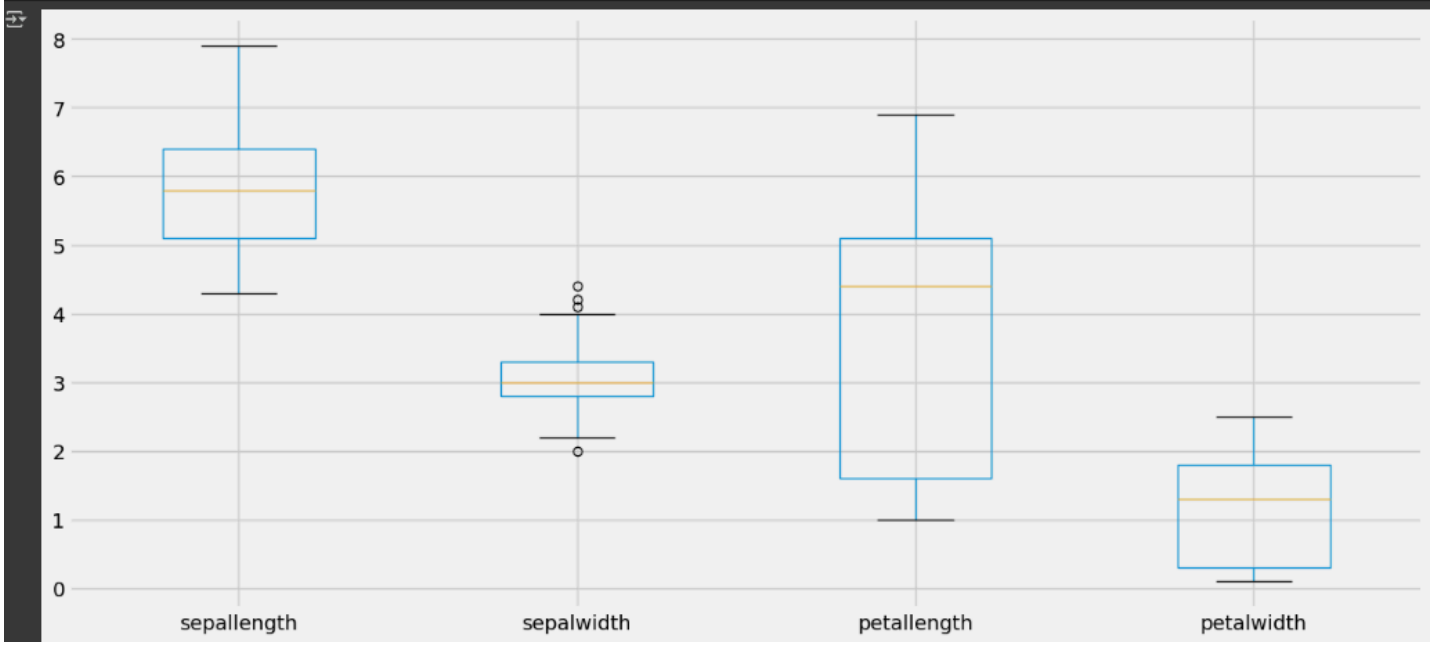
https://drive.google.com/file/d/1qzHYbZLq1sSuo2eLWcsNc-Su9p1_ER_I/view?usp=sharing

Questão 1)

```python
from sklearn.preprocessing import MinMaxScaler
# Inicializa o MinMaxScaler
scaler = MinMaxScaler()

colunas_para_normalizar = base.iloc[:, :-1]

# Ajusta e transforma os dados com o MinMaxScaler
base_normalizada = pd.DataFrame(scaler.fit_transform(colunas_para_normalizar), columns=colunas_para_normalizar.columns)

# Adiciona a última coluna não normalizada ao DataFrame normalizado
base_normalizada[base.columns[-1]] = base[base.columns[-1]]

# Exibe o DataFrame normalizado com a última coluna inalterada
print(base_normalizada)

base_normalizada.describe()
```

```
     sepallength  sepalwidth  petallength  petalwidth           class
0       0.222222    0.625000     0.067797    0.041667     Iris-setosa
1       0.166667    0.416667     0.067797    0.041667     Iris-setosa
2       0.111111    0.500000     0.050847    0.041667     Iris-setosa
3       0.083333    0.458333     0.084746    0.041667     Iris-setosa
4       0.194444    0.666667     0.067797    0.041667     Iris-setosa
..           ...         ...          ...         ...             ...
142     0.666667    0.416667     0.711864    0.916667             NaN
143     0.555556    0.208333     0.677966    0.750000  Iris-virginica
144     0.611111    0.416667     0.711864    0.791667  Iris-virginica
```

```python
# gera um bloxplot para cada atributo
base.boxplot(figsize=(15,7))
plt.show()
```



Questão 2)

QUESTÃO 2

```
[137] !pip install kneed
```

```
Requirement already satisfied: kneed in /usr/local/lib/python3.11/dist-packages (0.8.5)
Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.11/dist-packages (from kneed) (2.0.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from kneed) (1.14.1)
```

```
[138] import plotly.graph_objects as go
      from kneed import DataGenerator, KneeLocator
      from sklearn.cluster import KMeans
      from sklearn.preprocessing import StandardScaler
      from sklearn.metrics import silhouette_score
      from sklearn.preprocessing import MinMaxScaler
```

```
[139] X_teste.shape
```

```
(30, 4)
```

```
[140] base = pd.concat([pd.DataFrame(X_treino), pd.DataFrame(y_treino)], axis=1)
      base.columns = ['sepallength', 'sepalwidth', 'petallength', 'petalwidth', 'class']

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

```
(117, 4)
```

```
[141] scaler = MinMaxScaler()
      Entrada = scaler.fit_transform(Entrada)
```

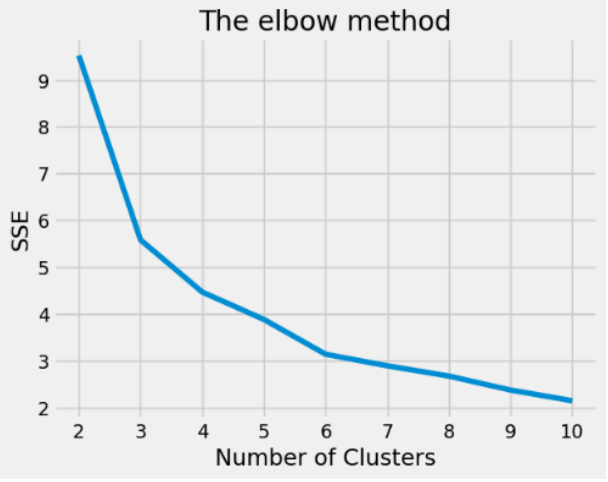
```python
limit = int((Entrada.shape[0]//2)**0.5)
for k in range(2, limit+1):
    model = KMeans(n_clusters=k)
    model.fit(Entrada)
    pred = model.predict(Entrada)
    score = silhouette_score(Entrada, pred)
    print('Silhouette Score k = {}: {:<.3f}'.format(k, score))
```

```
Silhouette Score k = 2: 0.632
Silhouette Score k = 3: 0.490
Silhouette Score k = 4: 0.429
Silhouette Score k = 5: 0.346
Silhouette Score k = 6: 0.329
Silhouette Score k = 7: 0.303
```

```python
wcss = []
for i in range(2, 11):
  kmeans = KMeans(n_clusters=i, random_state=10)
  kmeans.fit(Entrada)
  wcss.append(kmeans.inertia_)
wcss
```

```
[9.52469165550992,
 5.581647205886675,
 4.466456878115109,
 3.8804159892421026,
 3.1369288029624816,
 2.8888900314854657,
 2.6698402734693536,
 2.369980528083048,
 2.1416455838256843]
```

```python
plt.style.use("fivethirtyeight")
plt.plot(range(2, 11), wcss)
plt.xticks(range(2, 11))
plt.title('The elbow method')
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```



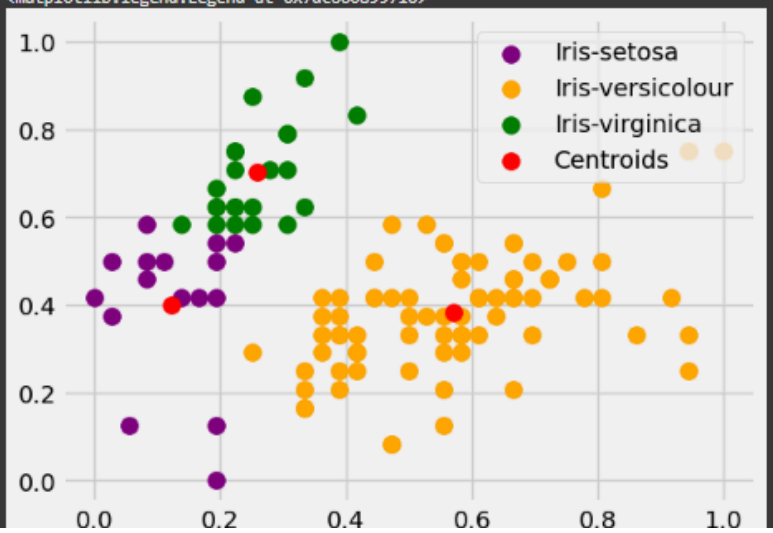
```
[146] kl = KneeLocator(range(2, 11), wcss, curve="convex", direction="decreasing")
      kl.elbow
```

```
np.int64(4)
```

```
[147] kmeans = KMeans(n_clusters=3, random_state=0)
      saida_kmeans = kmeans.fit_predict(Entrada)
```

```
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')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'red', label = 'Centroids')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7dc600899710>
```




```
for i in range(3):  # Supondo 3 clusters
    cluster_data = base[saida_kmeans == i]
    print(f"Cluster {i + 1}:")
    print(cluster_data.describe())
```

```
Cluster 1:
       sepallength  sepalwidth  petallength  petalwidth
count    16.000000   16.000000    16.000000    16.00000
mean      4.743750    2.962500     1.637500     0.33125
std       0.263233    0.406407     0.703207     0.27500
min       4.300000    2.000000     1.100000     0.10000
25%       4.575000    2.975000     1.300000     0.20000
50%       4.750000    3.050000     1.400000     0.20000
75%       5.000000    3.200000     1.525000     0.30000
max       5.100000    3.400000     3.500000     1.00000
Cluster 2:
       sepallength  sepalwidth  petallength  petalwidth
count    78.000000   78.000000    78.000000   78.000000
mean      6.353846    2.916667     4.985897    1.691026
std       0.605536    0.313926     0.786860    0.417533
min       5.200000    2.200000     3.500000    1.000000
25%       5.825000    2.700000     4.400000    1.300000
50%       6.300000    2.900000     4.900000    1.600000
75%       6.700000    3.100000     5.575000    2.000000
max       7.900000    3.800000     6.900000    2.500000
Cluster 3:
       sepallength  sepalwidth  petallength  petalwidth
count    23.000000   23.000000     23.00000   23.000000
mean      5.234783    3.686957      1.50000    0.265217
std       0.244222    0.284931      0.17581    0.115242
min       4.800000    3.400000      1.20000    0.100000
25%       5.100000    3.500000      1.40000    0.200000
50%       5.200000    3.600000      1.50000    0.200000
75%       5.400000    3.850000      1.55000    0.350000
max       5.800000    4.400000      1.90000    0.600000
```

Questão 3)

```python
from sklearn.model_selection import ParameterGrid

param_grid = {
    'n_clusters': [2, 3, 4, 5],
    'init': ['k-means++', 'random'],
    'n_init': [10, 20],
    'algorithm': ['lloyd', 'elkan']
}

melhor_score = -1
melhor_config = None

for params in ParameterGrid(param_grid):
    model = KMeans(**params, random_state=42)
    model.fit(X)
    score = silhouette_score(X, model.labels_)
    if score > melhor_score:
        melhor_score = score
        melhor_config = params

print("Melhor configuração:", melhor_config)
print("Melhor silhouette score:", melhor_score)
```

```
Melhor configuração: {'algorithm': 'lloyd', 'init': 'k-means++', 'n_clusters': 3, 'n_init': 10}
Melhor silhouette score: 0.6496036940701934
```

Questão 4)

O silhouette index é definido como o valor médio Si
entre todos os pontos, dado pela Equação:

Silhouette Index = $\frac{1}{n}\sum_{i=e}^{n} Si$, sendo Si = $-\frac{distOUT(Xi) - distIN(Xi)}{max\{distOUT(Xi), distIN(Xi)}$

distOUT = distância média de Xi para os pontos dos cluster mais próximos
distIN = distância média de Xi para os pontos do próprio cluster de Xi.

O método do “Elbow” é uma técnica heurística que, de forma matemática, se baseia na análise da função que mede a soma dos erros quadráticos (ou soma dos quadrados das distâncias) dentro dos clusters – conhecida como WCSS (Within-Cluster Sum of Squares) – em função do número de clusters K.

WCSS(K) = $\sum_{i=1}^{K} \sum_{p \epsilon Ci} dist(p,\ Ci)^2$

Questão 5)

```
[222] from sklearn.metrics import davies_bouldin_score
```

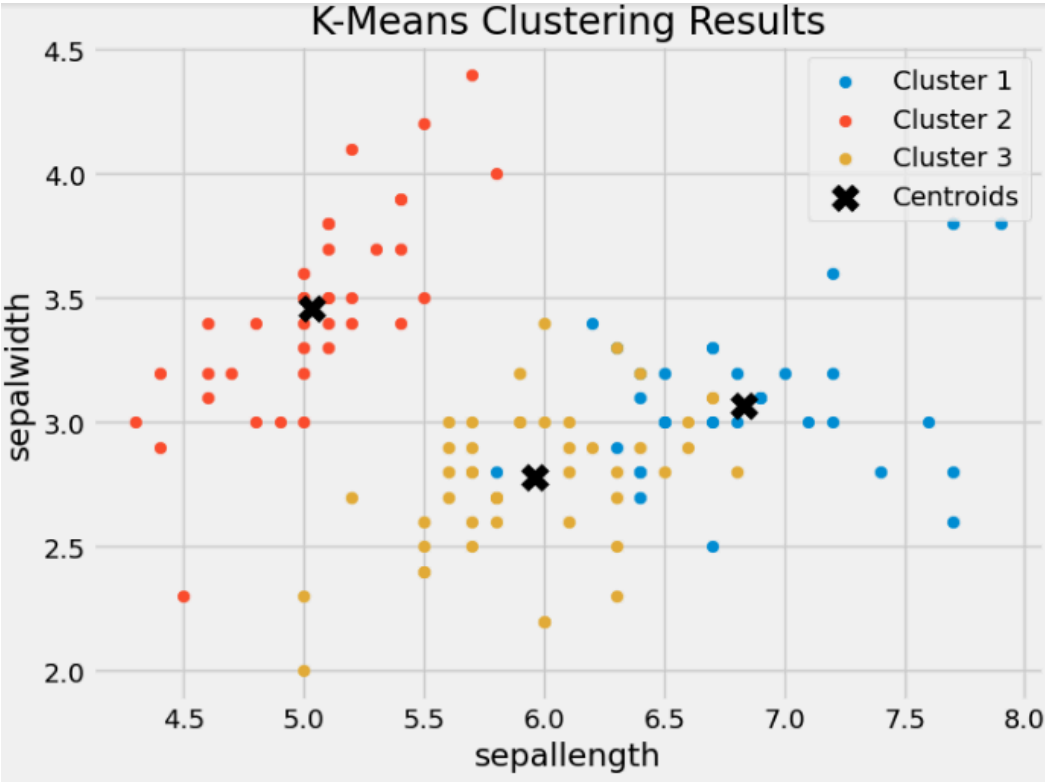
```
[223] X = base_nova[['sepallength', 'sepalwidth', 'petallength', 'petalwidth']].values
      labels = base_nova['cluster'].values
```

```
[224] db_index = davies_bouldin_score(X, labels)
      print(f"Davies-Bouldin Index: {db_index}")
```

```
Davies-Bouldin Index: 0.6819702008837346
```

```
[225] kmeans = KMeans(n_clusters=3, random_state=0)
      kmeans.fit(X)
      centroids = kmeans.cluster_centers_
```

```
plt.figure(figsize=(8, 6))
for i in range(3):
    cluster_data = X[labels == i]
    plt.scatter(cluster_data[:, 0], cluster_data[:, 1], label=f'Cluster {i + 1}')
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='k', marker='X', label='Centroids')
plt.xlabel('sepallength')
plt.ylabel('sepalwidth')
plt.title('K-Means Clustering Results')
plt.legend()
plt.show()
```



Questão 6)

```
[227] from sklearn.datasets import make_blobs

      centers = [[1, 1], [-1, -1], [1, -1]]
      X, labels_true = make_blobs(
          n_samples=750, centers=centers, cluster_std=0.4, random_state=0
      )

      X = StandardScaler().fit_transform(X)

[229] from sklearn import metrics
      from sklearn.cluster import DBSCAN

      db = DBSCAN(eps=0.3, min_samples=10).fit(X)
      labels = db.labels_

      n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
      n_noise_ = list(labels).count(-1)

      print("Estimated number of clusters: %d" % n_clusters_)
      print("Estimated number of noise points: %d" % n_noise_)

      Estimated number of clusters: 3
      Estimated number of noise points: 18
```

```
print(f"Homogeneity: {metrics.homogeneity_score(labels_true, labels):.3f}")
print(f"Completeness: {metrics.completeness_score(labels_true, labels):.3f}")
print(f"V-measure: {metrics.v_measure_score(labels_true, labels):.3f}")
print(f"Adjusted Rand Index: {metrics.adjusted_rand_score(labels_true, labels):.3f}")
print(
    "Adjusted Mutual Information:"
    f" {metrics.adjusted_mutual_info_score(labels_true, labels):.3f}"
)
print(f"Silhouette Coefficient: {metrics.silhouette_score(X, labels):.3f}")
```

```
Homogeneity: 0.953
Completeness: 0.883
V-measure: 0.917
Adjusted Rand Index: 0.952
Adjusted Mutual Information: 0.916
Silhouette Coefficient: 0.626
```

```
[231] !pip install sklearn-som
from sklearn_som.som import SOM
```

```
Requirement already satisfied: sklearn-som in /usr/local/lib/python3.11/dist-packages (1.1.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from sklearn-som) (2.0.2)
```

```
[232] iris_som = SOM(m=3, n=1, dim=2)
iris_som.fit(X)
```

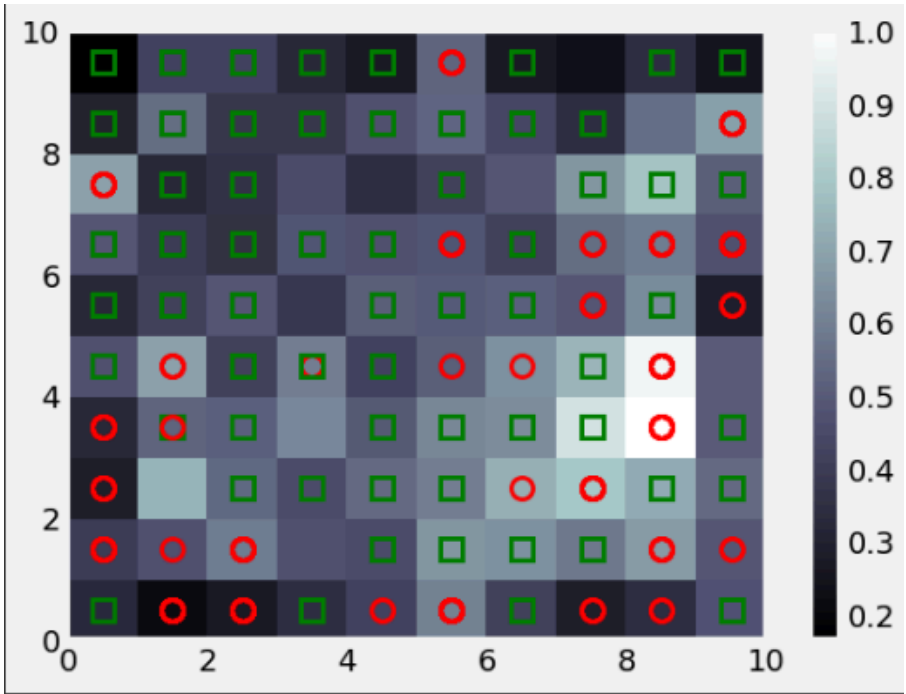
```
[233] predictions = iris_som.predict(X)
predictions
```

```
array([0, 2, 0, 1, 0, 2, 2, 1, 0, 0, 2, 2, 2, 1, 2, 0, 2, 2, 2, 1, 1, 1,
```

```
pip install MiniSom!
```

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
X = sc.fit_transform(X)

from minisom import MiniSom
som = MiniSom(x = 10, y = 10, input_len = 2, sigma = 1.0, learning_rate = 0.5)
som.random_weights_init(X)
som.train_random(data = X, num_iteration = 100)
```

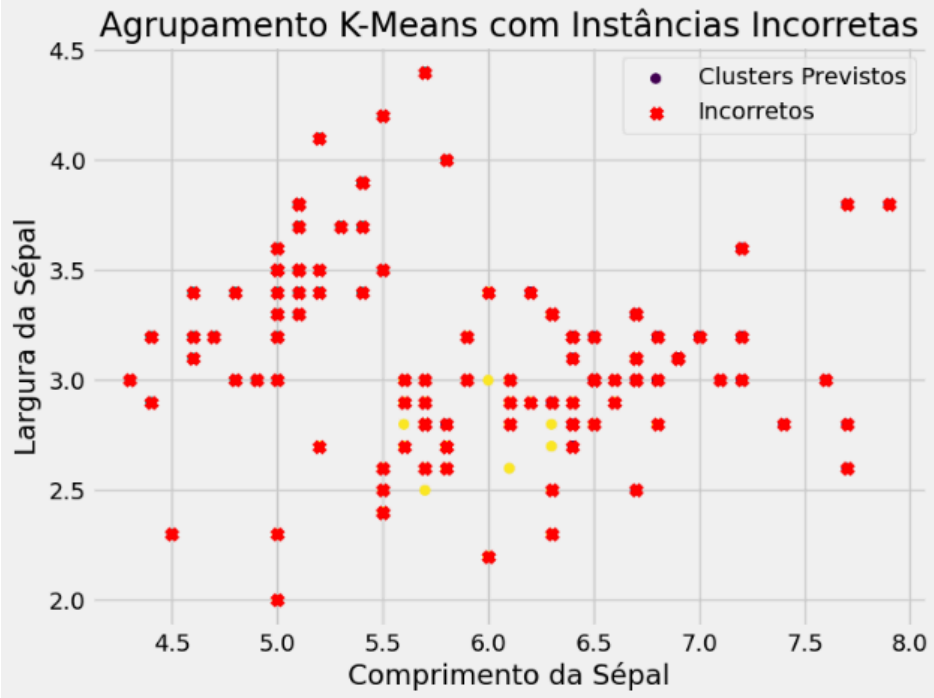
```
from pylab import bone, pcolor, colorbar, plot, show
bone()
pcolor(som.distance_map().T)
colorbar()
markers = ['o', 's']
colors = ['r', 'g']
for i, x in enumerate(X):
    w = som.winner(x)
    cluster_index = predictions[i]
    cluster_index = min(cluster_index, len(markers) - 1)
    cluster_index = min(cluster_index, len(colors) - 1)
    plot(w[0] + 0.5,
         w[1] + 0.5,
         markers[cluster_index],
         markeredgecolor = colors[cluster_index],
         markerfacecolor = 'None',
         markersize = 10,
         markeredgewidth = 2)
show()

outlier_neuron = (5,5)
if outlier_neuron in mappings and len(mappings[outlier_neuron]) > 0:
    frauds = mappings[outlier_neuron]
    frauds = sc.inverse_transform(frauds)
    print("Potential outliers:")
    print(frauds)
else:
    print("Warning: Outlier neuron not found or empty in mappings.")
```




```
Potential outliers:
[[-0.50267092 -0.75786841]
 [-0.50375148 -0.65585405]
 [-0.2789764  -0.79571625]]
```

Questão 7)

```
class_mapping = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
base_nova['true_label'] = base_nova['class'].map(class_mapping)

predicted_clusters = base_nova['cluster'].values
true_labels = base_nova['true_label'].values

plt.figure(figsize=(8, 6))
plt.scatter(base_nova['sepallength'], base_nova['sepalwidth'], c=predicted_clusters, cmap='viridis', label='Clusters Previstos')

incorrect_indices = np.where(predicted_clusters != true_labels)[0]
plt.scatter(base_nova.iloc[incorrect_indices]['sepallength'], base_nova.iloc[incorrect_indices]['sepalwidth'], marker='x', color='red', label='Incorretos')

plt.xlabel('Comprimento da Sépal')
plt.ylabel('Largura da Sépal')
plt.title('Agrupamento K-Means com Instâncias Incorretas')
plt.legend()
plt.show()
```



Questão 8)

Verificação e Tratamento de Valores Faltantes

- Executou-se **base.isnull().any(axis=1)** para identificar linhas com **NaN**.
- Resultado: não foram encontradas amostras com valores faltantes, indicando que a base original estava completa.

- Apesar disso, implementou-se a função **trataFaltantes** que, por classe, substituiria **NaN** pela média do mesmo atributo.

- Impacto: a etapa de imputação não modificou nenhum registro, mas garante robustez caso haja futuros dados faltantes .

3. Remoção de Duplicatas e Inconsistências

- Redundância: detectou-se pares de amostras com todos os atributos iguais (**drop_duplicates(keep='first')**).

- Inconsistência: buscou-se casos em que atributos eram idênticos mas as classes divergiam (**drop_duplicates(keep=False)**).

- Resultado: não havia duplicatas nem inconsistências após as verificações, mantendo-se as 150 amostras originais .

4. Estatísticas Descritivas

- Com **base.describe()**, obtiveram-se contagem, média, desvio-padrão, mínimo, máximo e quartis para cada atributo.

- **Observações principais:**

    - Médias típicas em torno de 5.8 (comprimento da sépala) e 3.0 (largura da sépala).

    - Amplitude natural sem valores extremos além do intervalo [4.3, 7.9] para sépala e [2.0, 4.4] para pétala.

5. Normalização (Min–Max Scaling)

- Aplicou-se **MinMaxScaler()** aos quatro atributos de entrada, transformando-os no intervalo [0, 1].

- A coluna de classe permaneceu inalterada.

- Justificativa: métodos de agrupamento baseados em distância (por ex., K-Means) sensíveis a escalas distintas de atributos .

6. Detecção de Outliers

- Boxplot: geração de boxplots para cada atributo, sem identificação de pontos extremamente fora do “bigode” dos gráficos.

- Pairplot: via **sns.pairplot**, visualizou-se boa separação entre as três classes, sem amostras isoladas além da dispersão esperada .

7. Análise da Distribuição das Classes

- Usou-se **np.unique(..., return_counts=True)** e **sns.countplot** para quantificar instâncias por classe.

- Resultado: 50 amostras para cada classe (**Iris-setosa, Iris-versicolor**, **Iris-virginica**), confirmando base perfeitamente balanceada .

8. Amostragem Holdout (Treino/Teste)

- Separou-se 80 % dos dados para treino (120 amostras) e 20 % para teste (30 amostras), com **random_state=42** para reprodutibilidade.

- Objetivo: avaliar posteriormente algoritmos de clustering e classificação em dados não vistos durante o treinamento .

Códigos:

https://drive.google.com/file/d/1qzHYbZLq1sSuo2eLWcsNc-Su9p1_ER_I/view?usp=sharing