Lista 7 IA

808238 – Breno Pires Santos

Questão 1

In [103…

```
import numpy as np
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import seaborn as sns
from scipy.stats import
zscore
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from kneed import DataGenerator, Kneelocator
```

```
df = pd.read_csv("Iris.csv", sep=',', encoding='utf-8')
```

```
df
```

	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows x 5 columns

Dados Ausentes e Redundantes

```
missing_index = df[df.isnull().any(axis=1)J.index
          if len(missing_indexs) > 0:
            print('Dados Ausentes:')
            display(df.iloc[missing_index])
          else:
            print('Não há dados ausentes')
```

```
Não há dados ausentes
```

```
column_names = df.columns[:-1]
          df_redundantes = df[df.duplicated(subset=column_names,keep=False)]
          if len(df_duplicates) > 0:
            print('Dados Redundantes:')
            display(df_duplicates)
          else:
            print('Não há dados redundantes')
```

	sepallength	sepalwidth	petallength	petalwidth	class
9	4.9	3.1	1.5	0.1	Iris-setosa
34	4.9	3.1	1.5	0.1	Iris-setosa
37	4.9	3.1	1.5	0.1	Iris-setosa
101	5.8	2.7	5.1	1.9	Iris-virginica
142	5.8	2.7	5.1	1.9	Iris-virginica

Removendo Dados Redundantes

```
In [109...  def delRedundantes( df_dataset ):
            df_dataset = df_dataset.drop_duplicates(keep = 'first')
```

```
  return df_dataset

df = delRedundantes (df)
```

Verificando Dados Inconsistentes

In [110…
```
df_redundantes = df[df.duplicated(subset=column_names,keep=False)J
if len(df_redundantes)>0:
  print('Dados Inconsistentes:')
  display(df_redundantes)
else:
  print('Não há dados inconsistentes')
```

```
Não há dados inconsistentes
```

Removendo Dados Inconsistentes

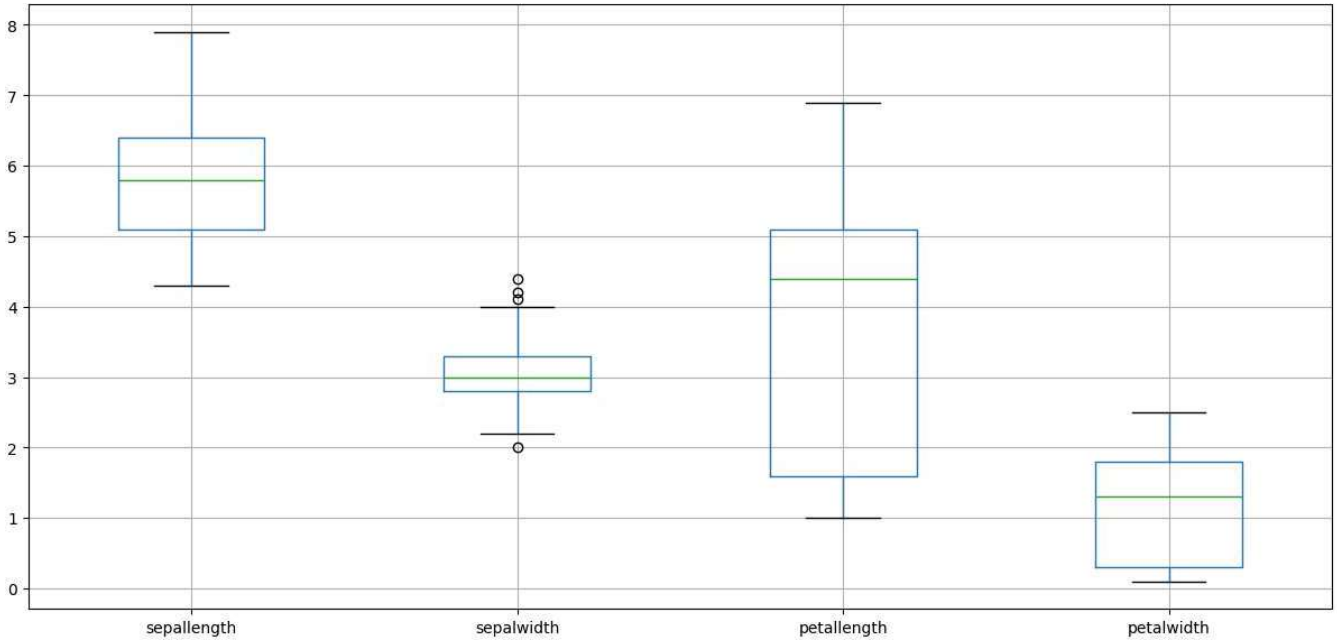
In [111…
```
def removerincons(df_dataset):
  df_dataset = df_dataset.drop_duplicates(subset=column_names,keep=False)
  return df dataset
# deLinconsistencias

df = removerincons (df)
df_redundantes = df[df.duplicated(subset=column_names,keep=False)]
if len(df_redundantes)>0:
  display(df_redundantes)
else:
  print('Não há dados redundantes ou inconsistentes')
```

```
Não há dados redundantes ou inconsistentes
```

Detectando Outliers

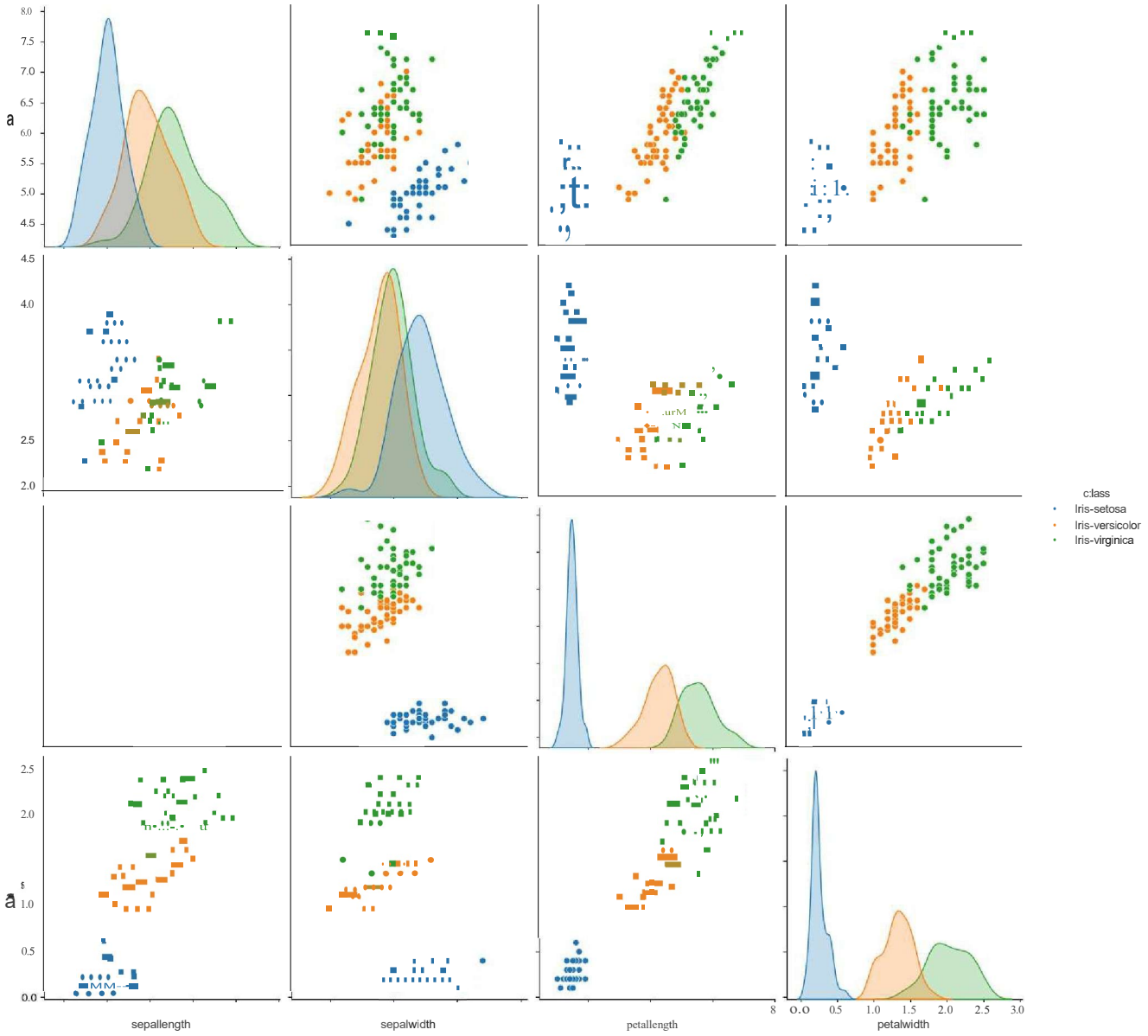
Boxplot:

In [112…
```
df.boxplot(figsize=(15,7))
plt.show()
```



Scatter

In [113…
```
sns.pairplot(df, hue='class', height=3.5);
plt.show()
```



Removendo Outliers

```
In [114...  z_scores = zscore(df[list(column_names)])
            z_df = pd.DataFrame(z_scores, columns=list(column_names))
            outlier_mask = (np.abs(z_df) > 3).any(axis=1)
            outliers = df[outlier_mask]

            print(f"{len(outliers)} Outlier:  (z-score > 3):") display(outliers)

            df = df[ outlier_mask].reset_index(drop=True)
```

```
1 Outlier: (z-score > 3):
```

	sepallength	sepalwidth	petallength	petalwidth	class
15	5.7	4.4	1.5	0.4	Iris-setosa

2. Encontrando agrupamentos utilizando Silhouette e Elbow

Normalização dos Dados

```
In [115...  Entrada= df.iloc[:, 0:4].values
            scaler = MinMaxScaler()
```

```
Entrada= scaler.fit_transform(Entrada)
```

Avaliando Silhouette Score

In [116…

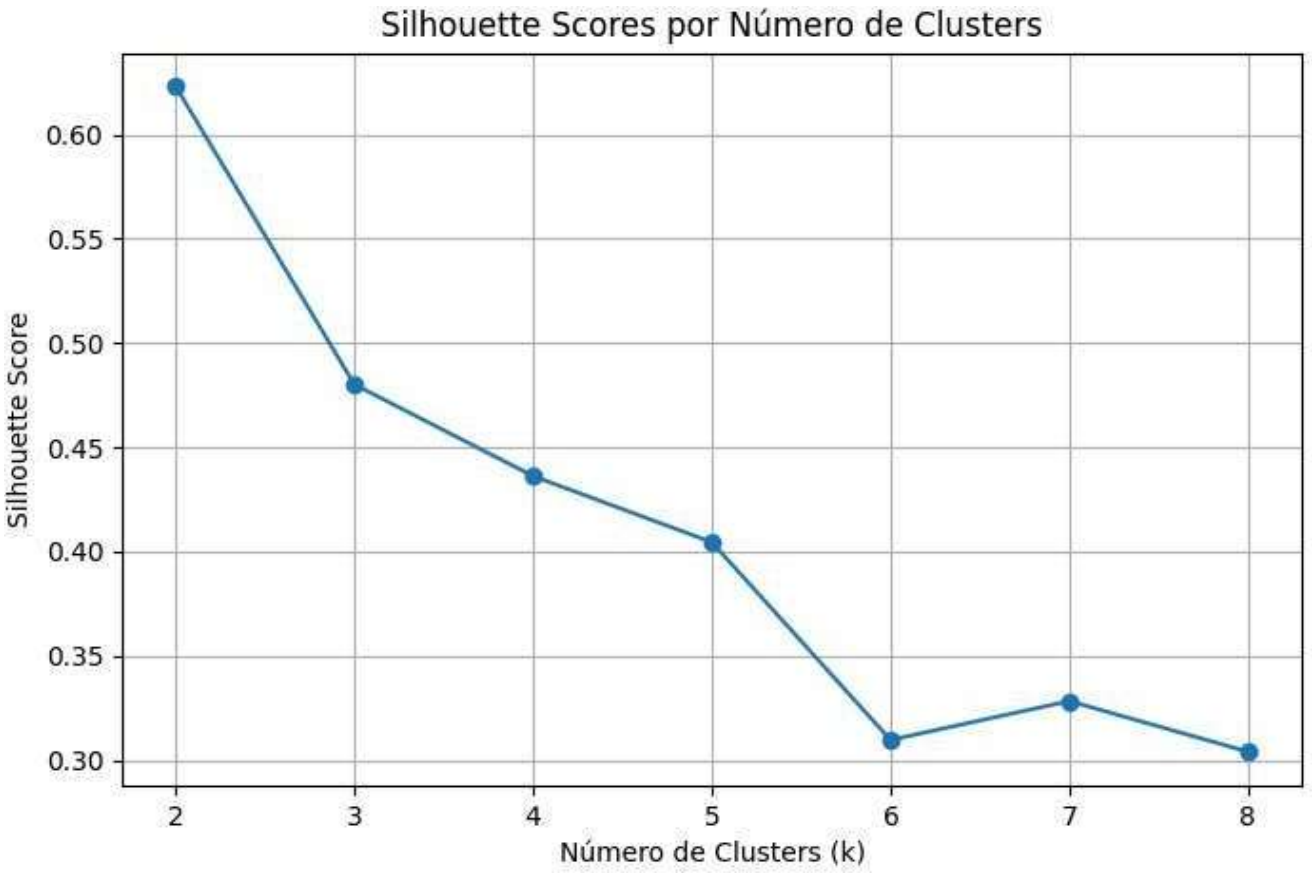
```
limit = int((Entrada.shape[0] // 2) ** 0.5)
sil_scores = []
k_range = range(2, limit+1)

for k in k_range:
    model = KMeans(n_clusters=k, random_state=42)
    labels = model.fit_predict(Entrada)
    score = silhouette_score(Entrada, labels)
    sil_scores.append(score)
    print(f"Silhouette Score k = {k}:{score:.3f}")
```

```
Silhouette Score k = 2: 0.623
Silhouette Score k = 3: 0.480
Silhouette Score k = 4: 0.436
Silhouette Score k = 5: 0.404
Silhouette Score k = 6: 0.310
Silhouette Score k = 7: 0.328
Silhouette Score k = 8: 0.304
```

Gráfico Silhouette:

In [117…

```
plt.figure(figsize=(8, 5))
plt.plot(k_range, sil_scores, marker='o')
plt.title("Silhouette Scores p/clusters")
plt.xlabel("Clusters: ") plt.ylabel("Silhouette
Score")
plt.grid(True)
plt.show()
```

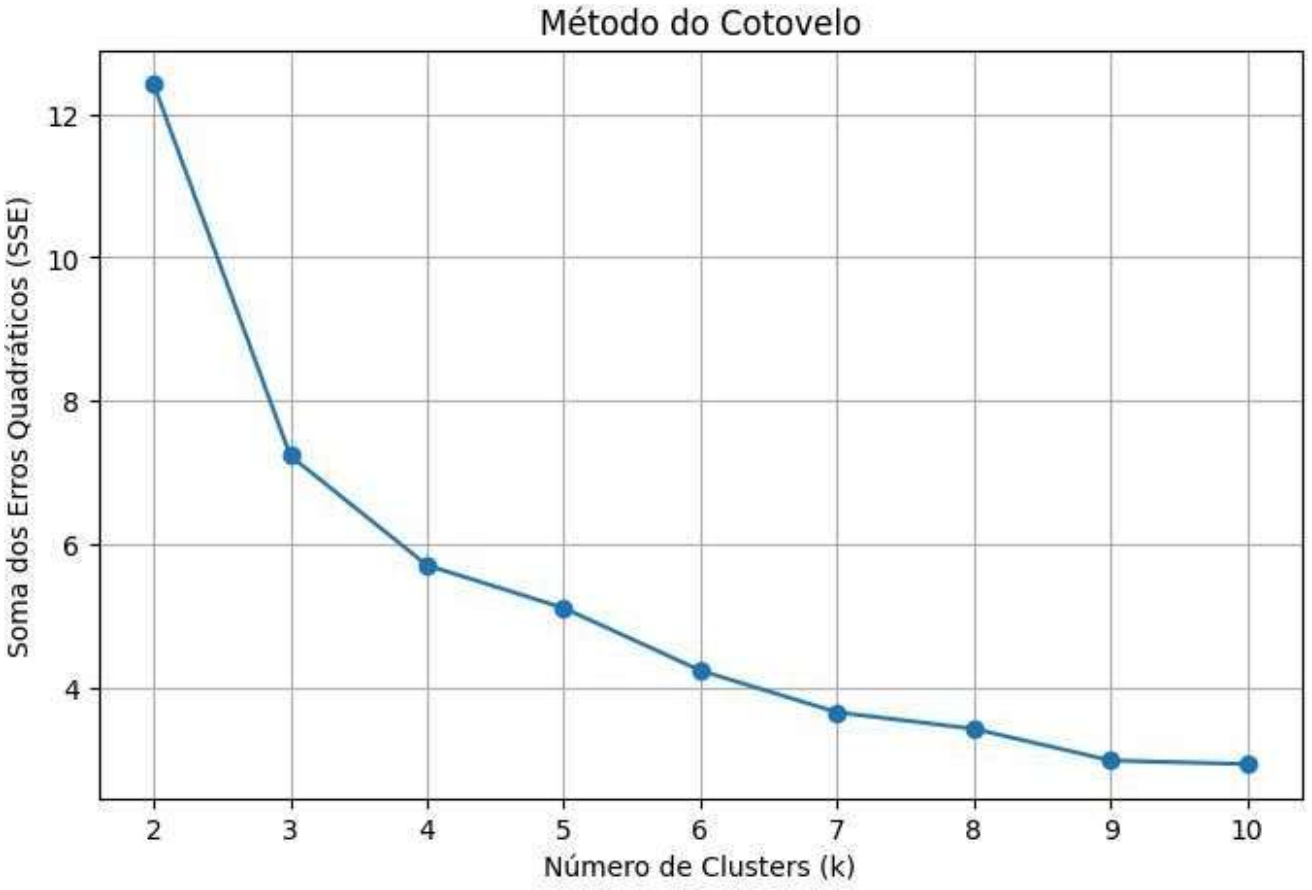


Avaliando o Elbow Method

In [118...
```
wcss = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(Entrada)
    wcss.append(kmeans.inertia_)
```

Gráfico Elbow

In [119...
```
plt.figure(figsize=(8, 5))
plt.plot(range(2, 11), wcss, marker='o')
plt.title("Método do Cotovelo")
plt.xlabel("Número de Clusters (k)")
plt.ylabel("Soma dos Erros Quadráticos (SSE)")
plt.grid(True)
plt.show()
```



Localiza o cotovelo

In [120...
```
kl = Kneelocator(range(2, 11), wcss, curve=''convex", direction="decreasing")
k_otimo = int(kl.elbow)
print(f"Valor ótimo de k encontrado pelo método do cotovelo: {k_otimo}")
```

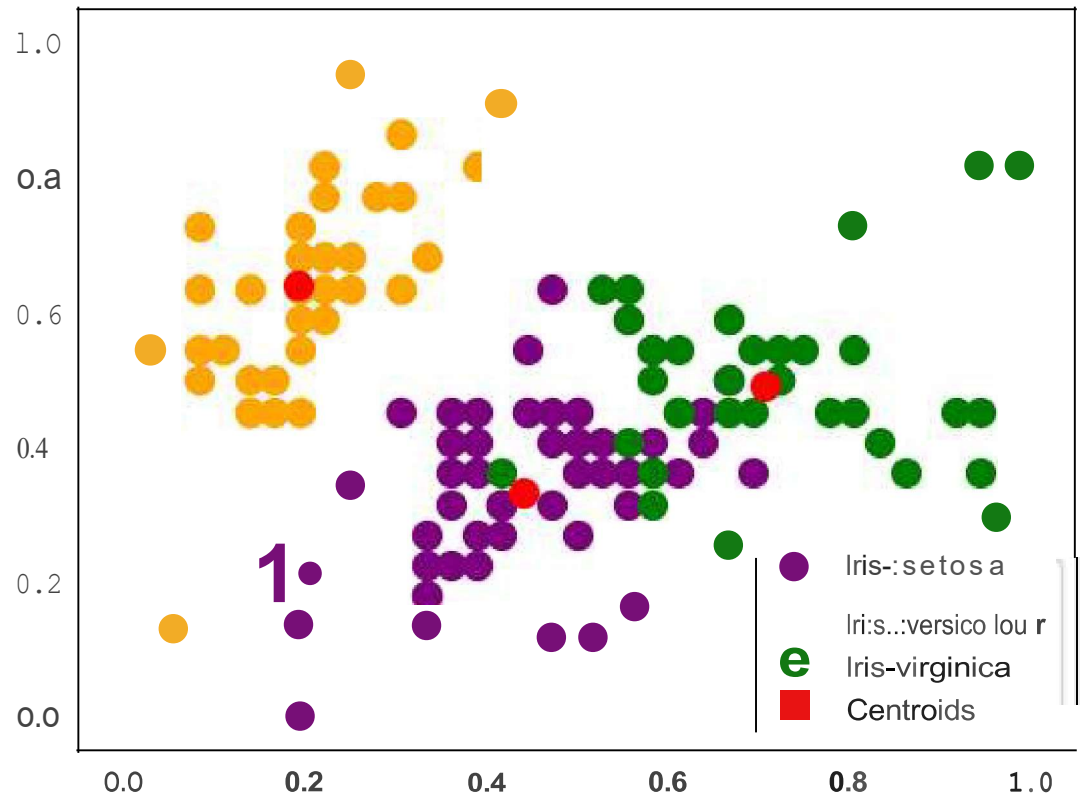
```
Valor ótimo de k encontrado pelo método do cotovelo: 4
```

Kmeans

In [121...
```
kmeans = KMeans(n_clusters=3, random_state=0)
saida_kmeans = kmeans.fit_predict(Entrada)
```

In [122...
```
plt.scatter(Entrada[saida_kmeans == 0, 0], Entrada[saida_kmeans == 0, 1], s = 100, e = 'pur
plt.scatter(Entrada[saida_kmeans == 1, 0], Entrada[saida_kmeans == 1, 1], s = 100, e = 'orê
plt.scatter(Entrada[saida_kmeans == 2, 0], Entrada[saida_kmeans == 2, 1], s = 100, e = 'grE
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, e = 'red'
plt.legend()
```

Out[122_ <matplotlib.legend.Legend at 0x7a9988991610>



Conclusão

A análise do Silhouette Score mostrou que usar 2 clusters deu o melhor resultado, com pontuação de 0.623. Quando tentamos mais clusters, os números foram piores, o que sugere que 2 grupos funcionam melhor.Por outro lado, o método do cotovelo apontou que 4 clusters seria o número ideal, já que a partir daí a redução no erro (SSE) não melhora muito.No fim das contas, pra não complicar demais e ainda ter um agrupamento decente, o melhor seria ficar com 3 ou 4 clusters.

3 – Hiperparametros do K-Means

O parâmetro init define como os centróides iniciais são escolhidos no KMeans, o que pode afetar fortemente os resultados. O modo 'k-means++', padrão, seleciona os centróides de forma estratégica para acelerar a convergência e evitar soluções ruins, sendo geralmente recomendado. Já o modo 'random' escolhe os centróides aleatoriamente, o que pode gerar resultados inconsistentes, especialmente em bases grandes ou mal distribuídas.

O n_init determina quantas vezes o algoritmo será executado com diferentes pontos de partida. O resultado com a menor inércia é mantido. Até 2023, o valor padrão era 10, mas agora pode ser 'auto', o que escolhe o número ideal automaticamente. Um valor maior ajuda a evitar mínimos locais ruins, mas aumenta o tempo de processamento.

Quanto à métrica de distância, o KMeans clássico sempre usa a distância Euclidiana para medir a proximidade entre os pontos e os centróides. Para usar outras métricas como Manhattan (L1), coseno, etc., é necessário recorrer a variantes como KMedoids (do scikit-learn-extra), DBSCAN ou AgglomerativeClustering.

O parâmetro max_iter define o número máximo de iterações que o algoritmo pode realizar antes de parar. O valor padrão é 300, o que costuma ser suficiente, mas pode ser aumentado se os dados forem complexos ou se houver muitos clusters, evitando que o algoritmo pare antes de convergir.

Por fim, o parâmetro tol controla a tolerância para considerar a convergência. Ele define o quanto os centróides podem se mover entre iterações. Valores muito pequenos tornam o critério mais rigoroso, podendo gerar mais iterações sem ganhos significativos na qualidade.

3. Explicação das Métricas: Silhouette Score e SSE

1. SSE (Soma dos Erros Quadráticos)

A SSE (Soma dos Erros Quadráticos) é uma métrica fundamental no método do cotovelo para avaliar a qualidade de agrupamentos. Ela mede o quão distantes os pontos estão dos centróides de seus respectivos clusters. Para isso, calcula-se a distância de cada ponto ao centróide do seu cluster, eleva-se esse valor ao quadrado e soma-se tudo. O resultado final representa o erro total do modelo — quanto menor, melhor o ajuste dos pontos aos clusters.

Fórmula:

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

- *k:* número de clusters
- Ci: conjunto de pontos no cluster i
- *μi:* centróide do cluster i

- *x:* ponto no cluster
- $\|x - \mu_i\|^2$: distância euclidiana ao quadrado do ponto ao centróide

Interpretação:

A interpretação da SSE está ligada à coesão dos clusters: quanto menor for esse valor, mais próximos os pontos estão dos seus centróides, indicando agrupamentos mais compactos. No entanto, à medida que aumentamos o número de clusters (k), a SSE tende a diminuir automaticamente, pois os grupos ficam menores e mais específicos. Por isso, o Elbow Method é usado para identificar um ponto de equilíbrio — o "cotovelo" do gráfico — onde aumentar k continua reduzindo a SSE, mas com ganhos cada vez menores, sugerindo o número ideal de clusters.

2. Silhouette Score

O Silhouette Score avalia a qualidade do agrupamento medindo o quão bem cada ponto se encaixa no seu próprio cluster em relação aos clusters vizinhos. Ele combina dois aspectos: a coesão (o quão próximo um ponto está dos outros no mesmo grupo) e a separação (o quão distante ele está dos pontos de outros grupos). Os valores vão de -1 a 1, onde quanto mais próximo de 1, melhor definido está o cluster — ou seja, os pontos estão bem agrupados e bem separados dos demais.

Fórmula:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

- a(i): distância média entre o ponto i e todos os outros pontos no **mesmo cluster**
- b(i): menor distância média do ponto i a **outros clusters** (i.e., cluster mais próximo)
- s(i) varia entre -1 e 1:

4. Métrica de avaliação diferente das 2 anteriores

Calinski-Harabasz

Definição:

O Índice de Calinski-Harabasz avalia a qualidade do agrupamento, levando em consideração a dispersão entre os clusters e a dispersão dentro dos clusters. Quanto maior o índice, melhor a separação entre os clusters, indicando que os clusters são bem definidos e distintos.

Fórmula:

DBI

$$s(i) = \frac{\text{Trac}(Bk) \div \text{Trac}(Wk)}{(k-1)\ (n-k)}$$

- Bk é a matriz de dispersão entre os clusters.
- Wk é a matriz de dispersão dentro dos clusters.
- k é o número de clusters.
- n é o número total de pontos.

Interpretação:

- **Valores mais altos indicam um melhor agrupamento, com clusters mais bem separados e mais coesos internamente.**
- Um valor muito baixo sugere que os clusters não estão bem separados ou são muito dispersos.
-

```
ln [123_ from sklearn.metrics import calinski_harabasz_score
ln [123_ from sklearn.cluster import KMeans

for k in range(2, 12):
    model = KMeans(n_clusters=k, random_state=42)
    labels = model.fit_predict(Entrada)
    dbi = davies_bouldin_score(Entrada, labels)
    print(f"Calinski-Harabasz Index k={k}:
    {ch_score:.3f}")
```

6. Aplicar DBSCAN e SOM e comparar com KMeans

DBSCAN

Agrupamento baseado em densidade. Identifica regiões densas de pontos e marca como outliers os pontos isolados.

In [124...
```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.3, min_samples=5)
dbscan_labels = dbscan.fit_predict(Entrada)
# Número de clusters encontrados (desconsiderando rótulo -1, que é ruído)
n_clusters_dbscan = len(set(dbscan_labels)) - (1 if -1 in dbscan_labels else 0)

print(f''Número de clusters encontrados pelo DBSCAN: {n_clusters_dbscan}")
```

```
Número de clusters encontrados pelo DBSCAN: 2
```

SOM (Self-Organizing Maps)

Rede neural não supervisionada que projeta os dados em um mapa 2D e organiza por similaridade.

In [125...
```
!pip install minisom
```

```
Requirement already satisfied: minisom in /usr/local/lib/python3.11/dist-packages (2.3.5)
```

In [126...
```
from minisom import MiniSom

som= MiniSom(x=2, y=2, input_len=Entrada.shape[1], sigma=0.5, learning_rate=0.5)
som.train_random(Entrada, 100)
som_labels = np.array([som.winner(x) for x in Entrada])
unique_som_labels = np.unique(som_labels, axis=0)
som_n_clusters = len(unique_som_labels)

print(f"Número de clusters encontrados pelo SOM: {som_n_clusters}")
```

```
Número de clusters encontrados pelo SOM: 3
```

7. Mostrando as instâncias agrupadas incorretamentes

In [127...
```
from scipy.stats import mede
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
```

Separar as características e os rótulos reais

In [128...
```
iris data= df.iloc[:, 0:4].values
iris_target = df['class']
```

Normalizar os dados

In [129…
```
scaler = MinMaxScaler()
iris_data= scaler.fit_transform(iris_data)
```

Aplicar K-Means com 3 clusters

In [130…
```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans_labels = kmeans.fit_predict(Entrada)
```

Redução para visualização 2D

In [131…
```
pca = PCA(n_components=2)
X_2d = pca.fit_transform(Entrada)
```

Mapear os rótulos previstos para as classes reais

In [132…
```
unique_labels = np.unique(iris_target)
labels = np.zeros_like(kmeans_labels)
for i in range(3):
    mask = (kmeans_labels == i)
    labels[mask] = mode([np.where(unique_labels == y)[0][0] for y in iris_target[mask]])[0]
```

Calcular a acurácia

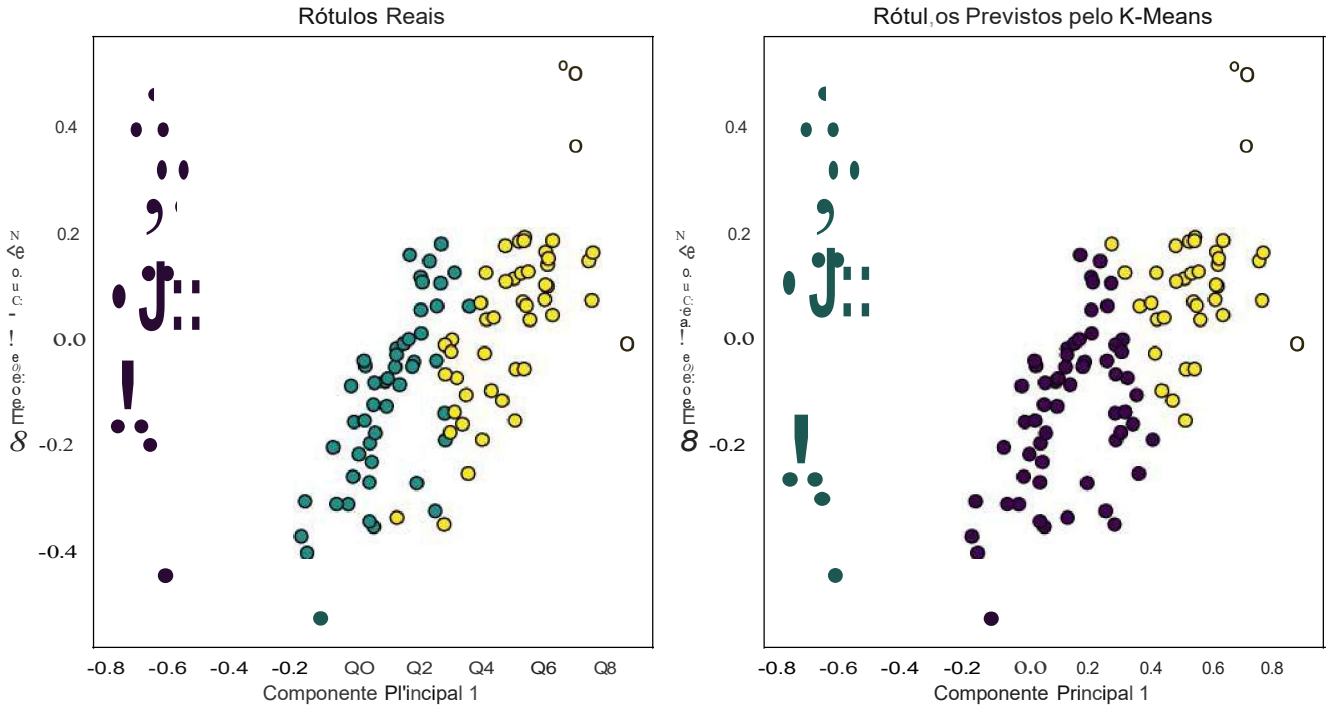
In [133…
```
accuracy = accuracy_score([np.where(unique_labels == y)[0][0] for y in iris_target], label
print(f"Acurácia do K-Means em relação aos rótulos reais: {accuracy * 100:.2f}%")
```

```
Acurácia do K-Means em relação aos rótulos reais: 89.04%
```

Visualizar os clusters e os rótulos reais

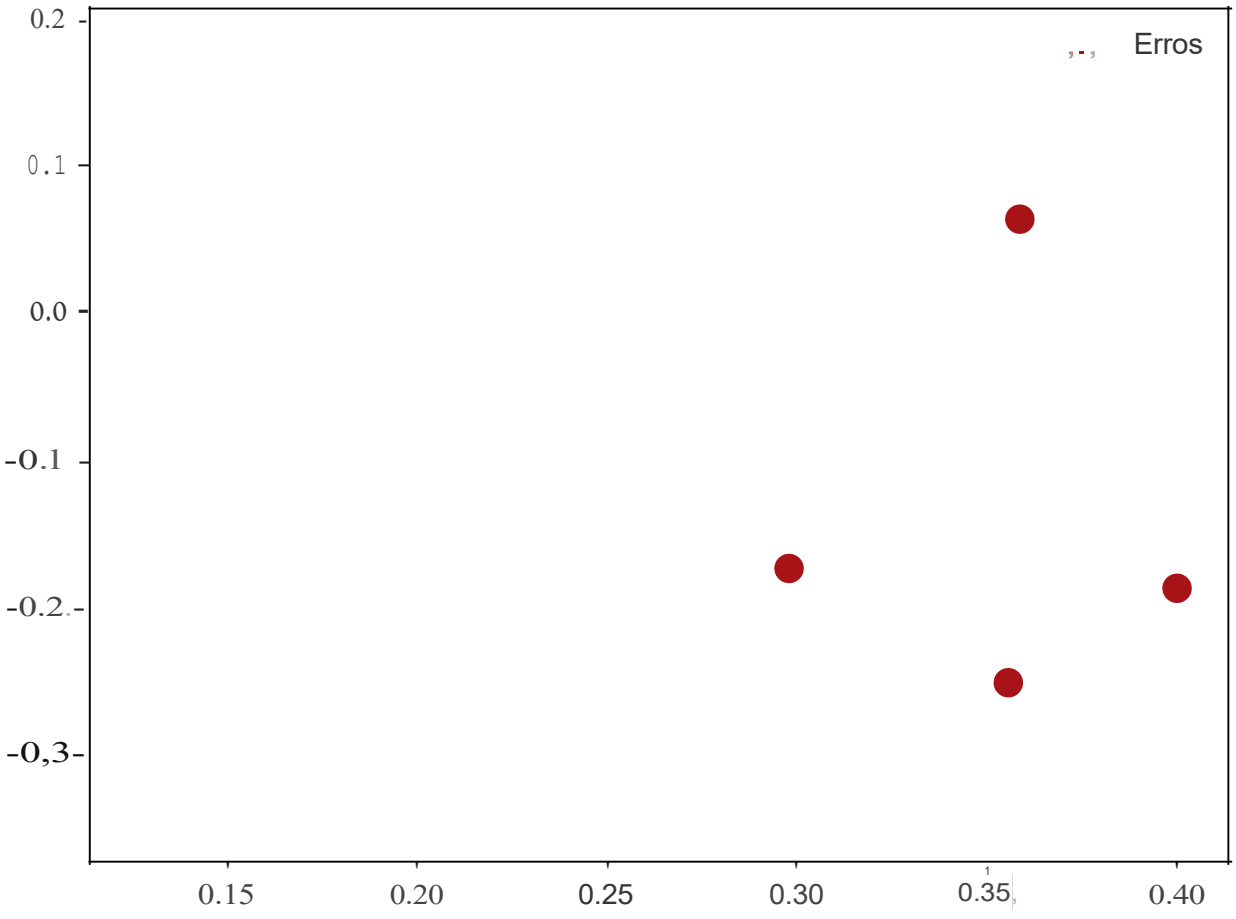
In [134…
```
plt.figure(figsize=(12, 6))

# Plot com os rótulos reais
plt.subplot(1, 2, 1)
plt.scatter(X_2d[:, 0], X_2d[:, 1], c=[np.where(unique_labels == y)[0][0] for y in iris tar
            cmap='viridis', edgecolor='k', s=50)
plt.title("Rótulos Reais")
plt.xlabel("Componente Principal 1")
plt.ylabel("Componente Principal 2")

# Plot com os rótulos previstos pelo K-Means
plt.subplot(1, 2, 2)
plt.scatter(X_2d[:, 0], X_2d[:, 1], c=kmeans_labels, cmap='viridis', edgecolor='k', s=50)
plt.title("Rótulos Previstos pelo K-Means")
plt.xlabel("Componente Principal 1")
plt.ylabel("Componente Principal 2")
```

Out[134…
```
Text(0, 0.5, 'Componente Principal 2')
```



Marcar as instâncias incorretamente classificadas

```
ln [135_ incorrect = (labels != [np.where(unique_labels == y)[0][0] for y in iris_target])
plt.scatter(X_2d[incorrect, 0], X_2d[incorrect, 1], color='red', edgecolor='k', s=80, labe]
plt.legend()
plt.tight_layout()
plt.show()
```



8. Relatório sobre as etapas de pré-processamento e resultados obtidos

1. Pré-processamento dos Dados

- A base de dados utilizada foi o conjunto Iris, disponível no sklearn.datasets.
- Foram considerados quatro atributos numéricos, que representam as medidas das sépalas e pétalas das flores.
- **Normalização**: A técnica de normalização MinMaxScaler foi aplicada para ajustar os dados ao intervalo [0, 1], essencial para algoritmos que dependem de distâncias, como KMeans e DBSCAN.
- **Detecção e remoção de outliers**: A identificação de outliers foi feita utilizando o escore Z (z-score). Instâncias com valores absolutos superiores a 3 foram consideradas outliers e removidas da base.

2. **Agrupamento com KMeans**

- **Número de clusters ideal**: O valor ótimo de k foi determinado usando o método do cotovelo (Elbow Method), que indicou k = 4. No entanto, como a base já possui três classes conhecidas, também foi testado k = 3.
- **Análise da qualidade do agrupamento**: A métrica **Silhouette Score** foi aplicada, apresentando os seguintes resultados:
  - k = 2: 0.623
  - k = 3: 0.480
  - k = 4: 0.436
- A escolha de k = 3 foi mais representativa para a estrutura da base, uma vez que existem três espécies de flores no conjunto Iris: Setosa, Versicolor e Virginica.
- **Caracterização dos clusters**: O KMeans conseguiu identificar claramente a classe Setosa, enquanto as classes Versicolor e Virginica apresentaram maior sobreposição nos agrupamentos.

2. Conclusão

O algoritmo KMeans demonstrou um bom desempenho ao ser aplicado na base Iris, principalmente após as etapas de remoção de outliers e normalização dos atributos. O uso das métricas **Silhouette Score**, **Elbow Method** e **Davies-Bouldin Index** possibilitou uma avaliação robusta da qualidade dos agrupamentos gerados. A análise revelou que a escolha do número de clusters ideal foi influenciada pela estrutura real da base de dados, com k = 3 sendo o valor mais representativo, já que o conjunto Iris contém três espécies de flores.

Ao comparar com os resultados dos algoritmos alternativos **DBSCAN** e **SOM**, observou-se que ambos também geraram resultados relevantes. O DBSCAN foi eficaz na identificação de ruídos, enquanto o SOM organizou os dados de maneira topológica, refletindo as relações de similaridade de forma interessante.

Ao comparar os clusters gerados com as classes reais, os resultados indicaram que o agrupamento foi capaz de capturar a estrutura dos dados, especialmente em relação à clara separação da classe **Setosa**. Contudo, as pequenas confusões entre as classes **Virginica** e **Versicolor** evidenciam a complexidade inerente ao agrupamento não supervisionado, onde as fronteiras entre essas classes não são tão nítidas, o que é esperado em tarefas de agrupamento.