AED

March 30, 2025

1 Análise Exploratória de Dados

[1]:
```
import sklearn
print(sklearn.__version__)  # Verificando a versão instalada
```

```
1.6.1
```

[2]:
```
from sklearn import datasets

# Carregando o conjunto de dados
iris = datasets.load_iris()
print("Conjunto de dados carregado com sucesso!")
```

```
Conjunto de dados carregado com sucesso!
```

[3]:
```
# Converter para DataFrame
import pandas as pd
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# Adicionar coluna das espécies
df['species'] = iris.target
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2:␣
 ↪'virginica'})

# Mostrar as primeiras linhas
df.head()
```

[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	
3	4.6	3.1	1.5	0.2	
4	5.0	3.6	1.4	0.2	

	species
0	setosa
1	setosa
2	setosa

```
3  setosa
4  setosa
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

1.0.1 O que a célula anterior nos informa:

Número de linhas e colunas

Tipos de dados de cada coluna

Se há valores ausentes

[5]:
```
df.describe()
```

[5]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
max            2.500000
```

1.0.2 O que a célula anterior nos informa:

Média, mínimo, máximo, desvio padrão para cada característica

O intervalo de valores

[6]:
```
df.isnull().sum()
```

[6]:
```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
dtype: int64
```

A saída é 0 para todas as colunas porque o conjunto de dados Iris não contém valores ausentes.

[7]:
```
df['species'].value_counts()
```

[7]:
```
species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

Cada espécie (Setosa, Versicolor, Virginica) tem 50 amostras.

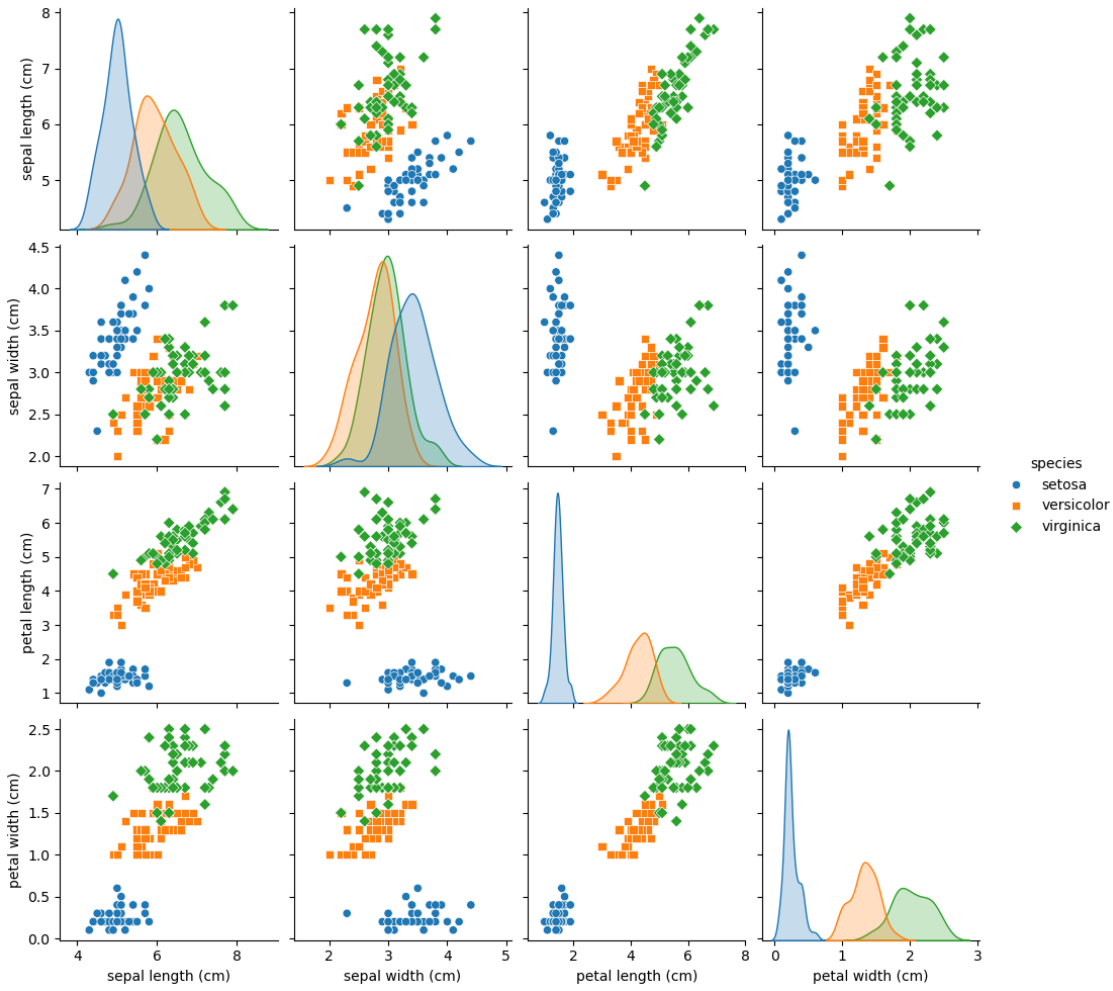
2 Visualizações

2.1 Etapa 1: Importar bibliotecas necessárias

[8]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

2.2 Etapa 2: Pairplot – Relacionamentos gerais de recursos

[9]:
```
sns.pairplot(df, hue="species", diag_kind="kde", markers=["o", "s", "D"])
plt.show()
```



2.2.1 O que a célula anterior nos informa:

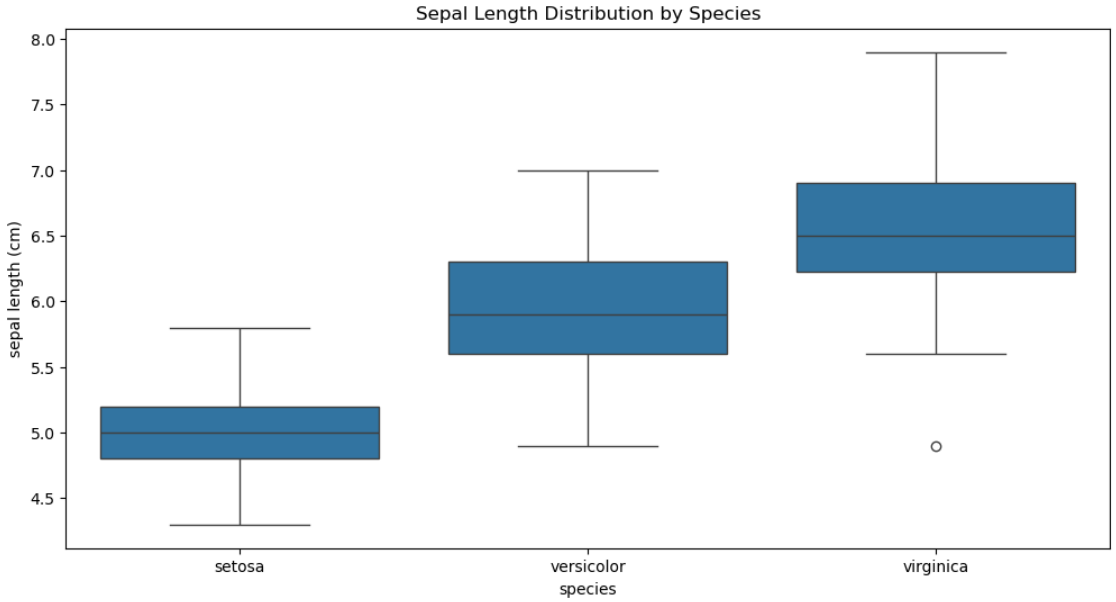
Diagramas de dispersão comparando todas as características

Cores diferentes representam espécies diferentes

Diagramas diagonais mostram a distribuição de cada característica

2.3 Etapa 3: Boxplots – Distribuições de recursos

[10]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x="species", y="sepal length (cm)")
plt.title("Sepal Length Distribution by Species")
plt.show()
```



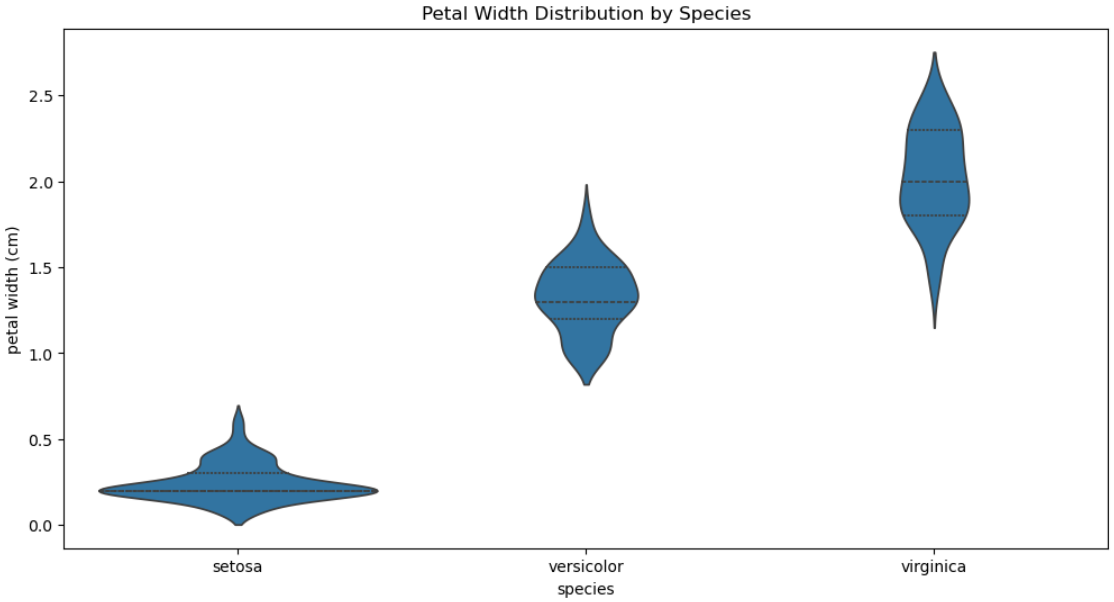
2.3.1 O que a célula anterior nos informa:

Como o comprimento da sépala varia entre as espécies

Se alguma espécie tem uma extensão maior ou valores atípicos

2.4 Etapa 4: Gráfico de violino – Distribuições de recursos com densidade

[11]:
```
plt.figure(figsize=(12, 6))
sns.violinplot(data=df, x="species", y="petal width (cm)", inner="quartile")
plt.title("Petal Width Distribution by Species")
plt.show()
```



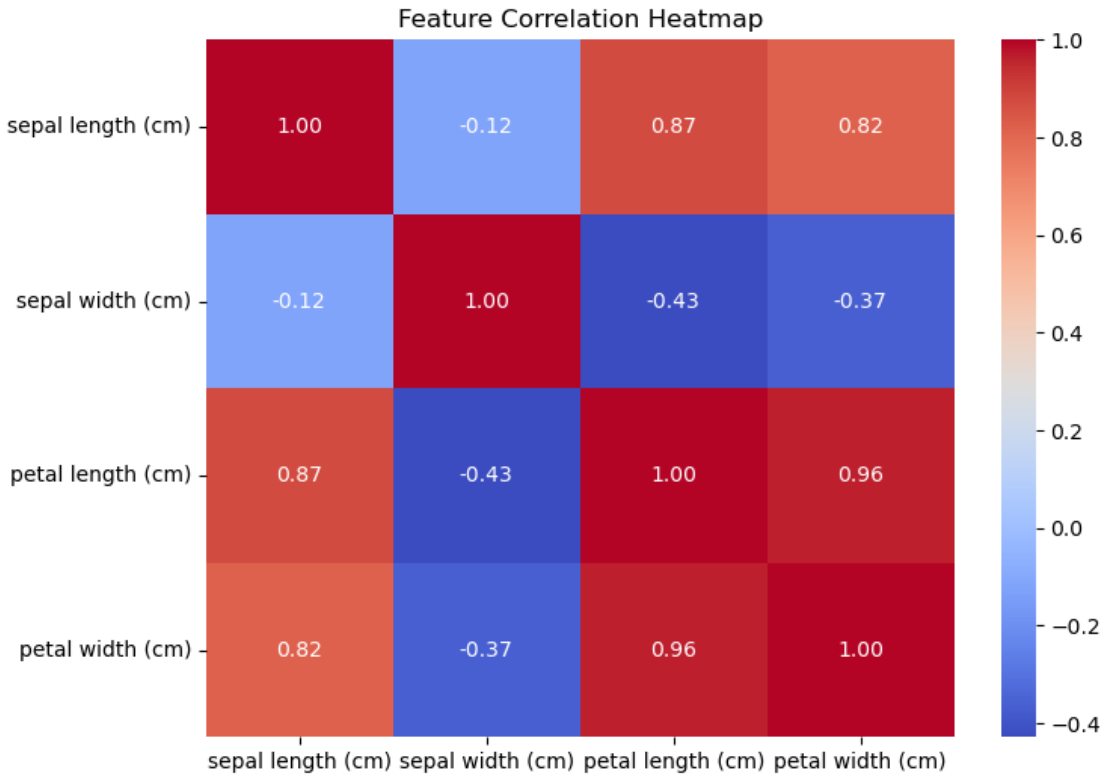
2.4.1 Por que usar um gráfico de violino:

Combina um boxplot e um gráfico de densidade

Mostra onde a maioria dos pontos de dados estão concentrados

2.5 Etapa 5: Mapa de calor – Correlação entre recursos

[12]:
```
plt.figure(figsize=(8,6))
sns.heatmap(df.drop(columns=["species"]).corr(), annot=True, cmap="coolwarm",␣
↪fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```



2.5.1 O que a célula anterior nos informa:

Quais recursos são altamente correlacionados (por exemplo, comprimento da pétala e largura da pétala)

Ajuda a decidir quais recursos podem ser redundantes

3 Cálculo da Acurácia

3.1 Etapa 1: Dividindo os Dados (Divisão de Treinamento-Teste)

Antes de treinar um modelo, dividimos o conjunto de dados em:

Conjunto de treinamento (por exemplo, 80%) – Usado para treinar o modelo

Conjunto de teste (por exemplo, 20%) – Usado para avaliar o desempenho do modelo

[13]:
```
from sklearn.model_selection import train_test_split

# Definir características (X) e alvo (y)
X = df.drop(columns=["species"])  # Características
y = df["species"]  # Alvo

# Dividir dados entre 80% treinamento e 20% teste
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

3.2 Etapa 2: Treinando um modelo

Classificador simples (regressão logística):

[14]:
```
from sklearn.linear_model import LogisticRegression

# Inicializar e treinar o modelo
model = LogisticRegression()
model.fit(X_train, y_train)
```

[14]:
```
LogisticRegression()
```

3.3 Etapa 3: Fazendo previsões

Após o treinamento, usamos o modelo para prever espécies no conjunto de teste:

[15]:
```
y_pred = model.predict(X_test)
```

3.4 Etapa 4: Calculando a pontuação de acurácia

Agora, calculamos a acurácia usando accuracy_score do Scikit-learn:

[16]:
```
from sklearn.metrics import accuracy_score

# Calcular acurácia
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 1.00
```

3.5 Etapa 5: Implementando KNN (K-Nearest Neighbors)

[20]:
```
# Importar o classificador KNN do sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Inicializar o classificador KNN
knn = KNeighborsClassifier(n_neighbors=3)

# Treinar o modelo nos dados de treinamento
knn.fit(X_train, y_train)

# Fazer previsões sobre os dados de teste
y_pred = knn.predict(X_test)

# Calcular acurácia
```

```python
accuracy = accuracy_score(y_test, y_pred)

# Print da acurácia
print(f"KNN Model Accuracy: {accuracy:.2f}")

# Matriz de confusão
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
KNN Model Accuracy: 1.00
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

3.6 Interpretação da Matriz de Confusão:

A primeira linha ([10, 0, 0]) significa que todas as 10 flores Setosa foram corretamente classificadas como Setosa.

A segunda linha ([0, 9, 0]) significa que todas as 9 flores Versicolor foram corretamente classificadas como Versicolor.

A terceira linha ([0, 0, 11]) significa que todas as 11 flores Virginica foram corretamente classificadas como Virginica.

Como não há classificações erradas (nenhum valor fora da diagonal), seu modelo previu perfeitamente todas as espécies.

3.6.1 Sobre a acurácia de 100%

O conjunto de dados Iris é frequentemente considerado um conjunto de dados "de brinquedo" ("toy dataset"), o que significa que é muito bem estruturado e separável com modelos simples. Como é um conjunto de dados pequeno e fácil de aprender, o modelo memorizou os exemplos de treinamento em vez de aprender padrões generalizáveis. Isso pode resultar em uma acurácia muito alta, como no caso deste modelo (100%).