FICHA TRABALHO 4

MODALIDADE:	Aprendizagem +	Não aplicável	
CURSO:	Técnico/a Especialista em Gestão de Informação e Ciência dos Dados		
UFCD:	Visualização de dados em Python	CÓDIGO UFCD:	10809
FORMADOR/A:	Bruno Silva	DATA:	

Para esta ficha de trabalho, vamos trabalhar com modelos de aprendizagem máquina, amis especificamente, trabalhar com o módulo Scikit-Learn.

O **Scikit-learn** (também conhecido como sklearn) é uma biblioteca de código aberto em Python focada na temática de aprendizagem de máquina (machine learning), construída sobre o NumPy, SciPy e Matplotlib.

Lançado em 2010, esta biblioteca ganhou um lugar de destaque no ecossistema de aprendizagem máquina em Python, pois, implementa vários algoritmos de modelagem de dados e aprendizado de máquina e fornece aplicações Python consistentes (oferece suporte a uma interface de modelo padronizada e concisa entre modelos).

Para além disso, esta biblioteca oferece uma variedade de algoritmos e ferramentas para análise de dados e modelagem preditiva. Com uma ampla gama de funcionalidades, desde algoritmos de classificação e regressão até clustering e pré-processamento de dados, o Scikit-Learn tornou-se uma ferramenta essencial para cientistas de dados e programadores.

Por exemplo, o Scikit-learn utiliza um modelo simples de fluxo de trabalho de ajuste/predição para seus algoritmos de classificação.

Algoritmos de aprendizagem

Ele fornece um conjunto abrangente de algoritmos de aprendizagem supervisionados e não supervisionados, cobrindo áreas como:

- **Classificação**: Identificar a qual categoria que um objeto pertence;
- **Regressão**: Prever um atributo de valor contínuo associado a um objeto;
- **Clustering**: Agrupamento automático de objetos semelhantes em conjuntos, com modelos como k-means;
- **Redução de Dimensões**: Redução do número de atributos nos dados para sumarização, visualização e seleção de recursos, com modelos como Análise de Componentes Principais;
- **Avaliação e Seleção de Modelos**: Comparar, validar e escolher parâmetros e modelos
- **Pré-processamento**: Extração e normalização de recursos, incluindo definição de atributos em dados de imagem e texto;

Alguns algoritmos disponíveis:

- Regressão Linear
- Regressão Logística
- KNN (k vizinhos mais próximos)
- Árvores de Decisão
- Random Forest
- SVM
- K-Means
- Gradient Boosting

Algoritmos de aprendizagem scikit-learn

A aprendizagem máquina é uma derivação da inteligência artificial dedicado à compreensão e à criação de métodos para imitar a maneira como os seres humanos aprendem.

Estes métodos incluem o uso de algoritmos e dados para melhorar o desempenho em algum conjunto de tarefas e geralmente enquadram em um dos três tipos mais comuns de aprendizagem:

- **Aprendizagem supervisionado:** um tipo de aprendizagem máquina que aprende a relação entre entrada e saída de dados;
- **Aprendizagem não supervisionado:** um tipo de aprendizagem máquina que aprende a estrutura subjacente de um conjunto de dados sem classificações;
- **Aprendizagem por reforço:** um método de aprendizagem máquina em que o agente de software aprende a executar determinadas ações em um ambiente que o leva à compensação máxima;

Instalação da biblioteca scikit-learn

Para instalar o scikit-learn deve abrir o terminal do Visual Studio Code e digitar a instrução:

- **Windows:** pip install scikit-learn
- **MacOS:** pip3 install scikit-learn

Nota Importante 1: No entanto, <u>se ainda não tiver as bibliotecas de ciência de dados instaladas</u>, deve usar o seguinte comando para instalar tudo de uma vez:

- **Windows:** pip install scikit-learn pandas numpy matplotlib
- **MacOS:** pip3 install scikit-learn pandas numpy matplotlib

Nota Importante 2: Esta biblioteca pode demorar um pouco (podemos falar talvez 1 a 2 minutos dependendo da ligação da internet), pois temos alguns pacotes com tamanhos de 5 a 21 MB. Poderá sempre consultar o estado da instalação para ver como corre a instalação:

```
Downloading joblib-1.6.0-py3-none-any.whl (306 kB)
Downloading cloudpickle-3.1.2-py3-none-any.whl (22 kB)
Downloading narwhals-2.25.0-py3-none-any.whl (467 kB)
Downloading numpy-2.5.2-cp314-cp314-macosx_14_0_arm64.whl (5.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.4/5.4 MB 3.3 MB/s  0:00:01
Downloading scipy-1.18.1-cp314-cp314-macosx_14_0_arm64.whl (20.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.5/20.5 MB 3.8 MB/s  0:00:05
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, numpy, narwhals, cloudpickle,
```

Depois da instalação, para poder testar o scikit-learn no python basta criar um novo ficheiro (VSCode ou Jupyter) com o seguinte código:

```python
#importar o módulo scikit-learn
import sklearn

#usar a função print() para mostrar qual a versão scikit-learn instalada
print(sklearn.__version__)
```

Resultado: Se aparecer o número da versão, significa que a instalação foi feita com sucesso

```
brunosilva@Air-de-Bruno ficha_3 % /u
testes.py
1.9.0
brunosilva@Air-de-Bruno ficha_3 %
```

Scikit-learn e os conceitos X (maiúsculo) e y (minúsculo)

Nos exemplos de Machine Learning vamos encontrar frequentemente os termos X (maiúsculo) e y (minúsculo). Estas letras são a convenção padrão para representar os dados que alimentam um modelo de Machine Learning. Estes dividem o conjunto de dados entre o que o modelo usa para aprender.

A diferença fundamental entre os dois é a seguinte:

O **X representa as variáveis explicativas ou independentes**. É tudo aquilo que o modelo vai analisar para tentar encontrar um padrão.

- **Formato**: Geralmente é uma matriz bidimensional (2D), como um DataFrame do Pandas ou um array do NumPy;
- **Estrutura**: As linhas representam as amostras (clientes, imóveis, exames) e as colunas representam as características de cada amostra (idade, preço, tamanho, sintomas).
- **Por que X maiúsculo?** Na matemática, matrizes são representadas por letras maiúsculas ☺

O **y representa a variável dependente ou resposta**. É o objetivo final, aquilo que quer que o modelo aprenda a prever no futuro.

- **Formato**: Geralmente é um vetor unidimensional (1D), ou seja, apenas uma única coluna de dados.
- **Exemplos**: Se o modelo vai prever se um cliente vai cancelar o plano, o y terá respostas como 0 (Não) ou 1 (Sim). Se for prever o preço de uma casa, o y conterá os valores em dinheiro.
- **Por que y minúsculo? Na matemática, vetores são representados por letras minúsculas.**

Portanto num exemplo prático, podemos referir que:

- X → características
- y → resultado que queremos prever

Neste exemplo:

- X = área + número de quartos
- y = preço

```
X = [
    [80, 2],
    [120, 3],
    [200, 4]
]

y = [
    150000,
    220000,
    350000
]
```

Scikit-learn: funções fit() e predict()

Nos exemplos de Machine Learning também vamos encontrar frequentemente as funções:

- **fit():** serve para treinar o modelo, ou seja, "O modelo aprende a partir destes dados";

```
modelo.fit(X, y)
```

- **predict():** serve para fazer as previsões " O modelo agora utiliza o que aprendeu para fazer uma previsão";

```
modelo.predict(novos_dados)
```

Utilizar os modelos de dados incluídos no scikit-learn

O Scikit-Learn inclui vários conjuntos de dados para fins de demonstração, que podem ser consultados na seguinte hiperligação: https://scikit-learn.org/stable/api/sklearn.datasets.html

Exemplos de Datasets do Scikit-Learn:

- **Wine**: Classificação de tipos de vinhos (**load_wine**);
  - 178 amostras
  - 13 características
  - 3 classes
  - **Problema**: classificação
- **Iris**: Classificação de tipos de flores (**load_iris**);
  - 150 amostras
  - 4 características
  - 3 classes
  - **Problema**: classificação
- **Breast Cancer:** Classificação de tumores como malignos ou benignos (load_breast_cancer);
  - 569 amostras
  - 30 características
  - 2 classes
  - **Problema**: classificação

- **Digits**: Reconhecimento de números escritos à mão (load_digits);
  - Imagens de números
  - 10 classes (0 a 9)
  - **Problema**: classificação

- **Diabetes**: Classificação do tipo de diabetes;
  - Aqui o objetivo não é prever uma classe, **mas um valor numérico.**

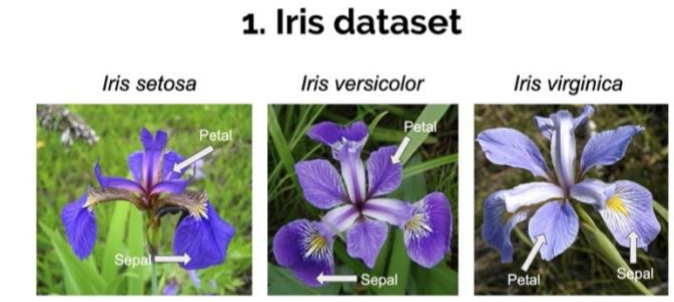
- **California Houses**: Previsão do valor de casas (fetch_california_housing);
  - Problema: **regressão**.

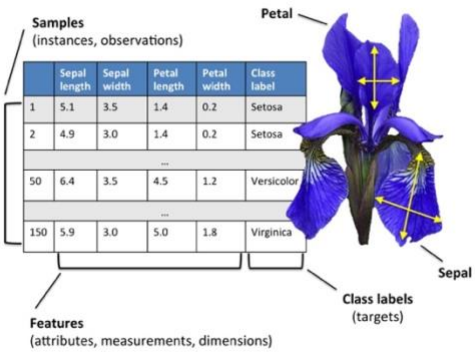
- Entre outros mais;

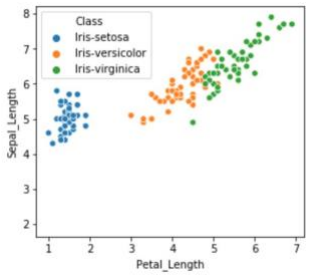
Como tal, vamos utilizar o nosso primeiro exemplo e utilizar um dataset que já vem incluído no Scikit-Learn: **o Iris**. Este dataset contém informações sobre flores e temos 4 caraterísticas:

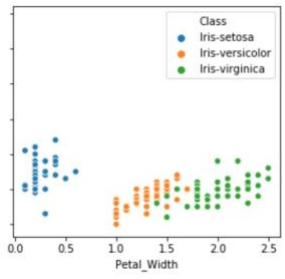
- comprimento da sépala;
- largura da sépala;
- comprimento da pétala;
- largura da pétala;

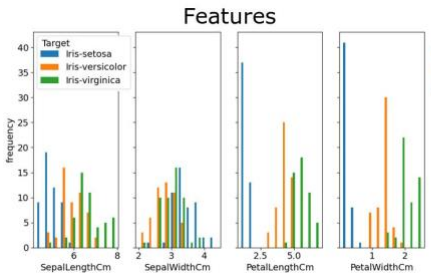
O objetivo é: prever a espécie da flor.











Passo 1: Carregar o Conjunto de Dados Iris

Como tal, vamos começar por importar o módulo do scikit-learn e de seguida aos **modelos** (datasets) e importar o **dataset específico** load_iris:

```
from sklearn.datasets import load_iris

# Carregar o conjunto de dados Iris
iris = load_iris()
```

A partir daqui podemos mostrar os dados que vem no dataset, como por exemplo:

- **Exibir informações sobre o conjunto de dados**

```
# Exibir informações sobre o conjunto de dados
print(iris)
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5.  3.4  1.5  0.2]
```

- **Mostrar os nomes das características**

```
# Mostrar os nomes das características
print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

- **Mostrar os nomes das espécies**

```
# Mostrar o nome das especies
print(iris.target_names)
```

```
['setosa' 'versicolor' 'virginica']
```

- **Mostrar informações do conjunto de dados e forma detelhada**

```
# Exibir informações sobre o conjunto de dados
print(iris.DESCR)
```

Resultado:

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica

:Summary Statistics:

============== ==== ==== ======= ===== ====================
                Min  Max   Mean    SD   Class Correlation
============== ==== ==== ======= ===== ====================
sepal length:   4.3  7.9   5.84   0.83    0.7826
sepal width:    2.0  4.4   3.05   0.43   -0.4194
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
============== ==== ==== ======= ===== ====================
```

Passo 2: Carregamento e Dividir a Exploração dos Dados (separar o X e y)

Vamos criar as variáveis e separar os dados em atributos (X) e rótulos (y), e de seguida, vamos ver os dados que carregou, mais especificamente, as colunas dos dados que carregou (sem os dados em si):

```
# Separa os dados em atributos (X) e rótulos (y)
X = iris.data
y = iris.target

# Mostrar os nomes das características e os nomes das classes
print("Nomes das Características:", iris.feature_names)
print("Nomes das Classes:", iris.target_names)
```

```
Nomes das Características: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Nomes das Classes: ['setosa' 'versicolor' 'virginica']
```

Se eventualmente utilizar a propriedade **shape** (como já visto com o módulo do pandas, podemos ver a dimensão dos dados que foram carregados):

```
#Ver a dimensão dos dados carregados (com a propriedade shape)
print(X.shape) #Mostra quantos exemplos carregou
print(y.shape) #Mostra caractristicas temos (com base nos exemplos que carregou)
```

```
(150, 4)
(150,)
```

Resumindo, temos 150 exemplos (respostas) carregadas com 4 colunas (características)

Passo 3: Dividir os dados em conjunto de Treino e Teste

Antes de começar esta parte temos de ter em atenção um ponto muito importante: **Não devemos normalmente treinar e avaliar o modelo exatamente nos mesmos dados.**

Antes de **construir o modelo**, **precisamos dividir os dados** em um **conjunto de treino** e um **conjunto de teste**. O conjunto de treino será usado para treinar o modelo, enquanto o conjunto de teste será usado para avaliar sua precisão.

Há várias maneiras de dividir os dados em conjuntos de treino e teste, mas o scikit-learn tem uma função integrada para fazer isso com o nome **train_test_split()**.

Como tal, vamos ter de adicionar (após a importação do módulo scikit-learn), a importação do modelo de seleção dos dados para dividir os dados em conjuntos de treino e teste

```
#carregar o modulo scikit-learn e ao mesmo tempo o conjunto de dados Iris
from sklearn.datasets import load_iris

# Para dividir os dados em conjuntos de treino e teste
from sklearn.model_selection import train_test_split
```

Código:

Para dividir os dados em conjuntos de treino e teste

from sklearn.model_selection import train_test_split

De seguida, vamos fazer a dividir os dados em conjunto de treinamento e teste, mais especificamente, 80% para treinamento e 20% para teste. Como tal, vamos colocar a seguinte instrução:

```
# Dividir os dados em conjunto de treinamento e teste
#(80% para treinamento, 20% para teste)

X_treino, X_teste, y_treino, y_teste = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42
)
```

Código:

Dividir os dados em conjunto de treinamento e teste (80% para treinamento, 20% para teste)

X_treino, X_teste, y_treino, y_teste = **train_test_split**(X, y, test_size=0.2, random_state=42)

O que significa o parâmetro test_size=0.2?

Significa que reservamos **20% dos dados para teste**.

Como temos 150 exemplos:

80% → 120 para treino

20% → 30 para teste

E random_state=42?

- Serve para tornar a divisão reproduzível, ou seja, sempre que executar o código com os mesmos dados, vamos obter a mesma divisão.
- O número 42 não tem nenhuma propriedade especial. Poderia ser outro número.

Passo 4: Criar um modelo de classificação e a Função fit()

As técnicas de classificação são uma parte essencial das aplicações de aprendizagem máquina e exploração de dados. Aproximadamente 70% dos problemas em ciência de dados são problemas de classificação.

Para criar modelos de classificação, podemos usar várias formas de classificação dos dados, como por exemplo:

- Regressão logística
- Máquina de vetor de suporte
- Classificador de árvore de decisão
- Entre outros modelos;

Neste exemplo, vamos usar a **Regressão Logística**, um **algoritmo simples e eficaz para problemas de classificação**, em que:

- Vamos **importar** o **modelo** da **regressão linear**;
- **Criar variável para e passar o modelo de classificação** (com um número de interações (**max_iter → define o número máximo de iterações (passos de otimização) que o algoritmo de otimização** pode executar para encontrar os melhores coeficientes do modelo) e depois treinar o modelo utilizando o conjunto de dados que foram treinados (feito no passo anterior);

- **Por último,** deve utilizar a **função fit()** para treinar o modelo utilizando o conjunto do treino que foi efetuado no passo anteriore;

```
from sklearn.linear_model import LogisticRegression

# Criar o modelo de classificação
model = LogisticRegression(max_iter=200)

# Treinar o modelo utilizando o conjunto de treinamento
model.fit(X_treino, y_treino)
```

```
▾ LogisticRegression ⓘ ⓘ
▸ Parameters
▸ Fitted attributes
```

Passo 5: Avaliar o modelo com a Função predict() + accuracy_score()

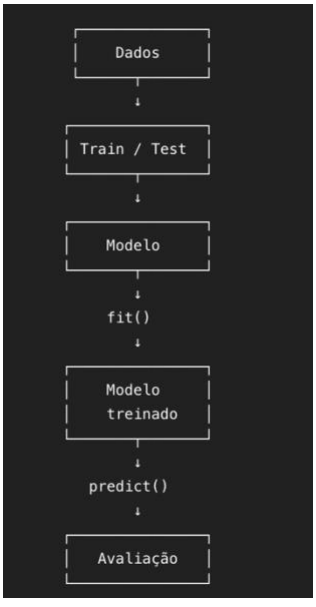
Agora que o modelo está treinado, podemos utilizar os dados que o modelo **não viu durante o treino** para fazer previsões sobre o conjunto de teste e avaliar a sua precisão.

```
# Fazer previsões sobre o conjunto de teste
previsoes = model.predict(X_teste)

from sklearn.metrics import accuracy_score

# Avaliar a precisão do modelo
accuracy = accuracy_score(y_teste, previsoes)

print(f"Precisão: {accuracy:.2%}")
```

```
Precisão: 100.00%
```

Isto significa que o modelo classificou corretamente aproximadamente 100% das flores no conjunto de teste.

Este pequeno programa já contém o fluxo fundamental de um projeto de Machine Learning:



Passo 6: Visualizar os resultados

Para entender melhor como nosso modelo está se saindo, podemos visualizar a matriz de confusão, que mostra as previsões corretas e incorretas feitas pelo modelo.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Gerar a matriz de confusão (com base no teste e previsões)
cm = confusion_matrix(y_teste, predictions)

# Criar a matriz de confusão
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.title('Matriz de Confusão')

plt.show()
```