**Importar bibliotecas**

* **Pandas (import pandas as pd)**

**Quando usar**: Sempre que estiver manipulando e analisando dados em formato de tabelas, como planilhas Excel ou arquivos CSV. O pandas oferece estruturas de dados como DataFrame e Series, que facilitam o trabalho com grandes quantidades de informações.

**Exemplos de uso**:

* + Leitura de dados de arquivos CSV, Excel ou SQL.
  + Manipulação de dados: filtragem, agrupamento, pivotagem, etc.
  + Análise descritiva, como cálculo de médias, medianas, sumários, etc.
  + Limpeza de dados: tratamento de valores ausentes, substituições, formatações
* **Matplotlib (import matplotlib.pyplot as plt)**

**Quando usar**: Para gerar visualizações de dados, como gráficos de linha, barras, dispersão (scatter plots), histogramas e boxplots. O matplotlib é útil para visualizar tendências e padrões em seus dados.

**Exemplos de uso**:

* + Criação de gráficos simples e avançados a partir de dados numéricos.
  + Customização de gráficos, como títulos, eixos, cores, legendas, etc.
  + Salvamento de gráficos em formato de imagem (PNG, JPEG, etc.)
* **LogisticRegression (from sklearn.linear\_model import LogisticRegression)**

**Quando usar**: Você deve importar LogisticRegression quando estiver construindo modelos de **classificação binária ou multiclasse**. A **Regressão Logística** é um modelo estatístico que é amplamente usado para prever a probabilidade de uma classe (ou evento) com base em uma ou mais variáveis independentes. É especialmente útil quando a variável de saída é categórica (0/1, Verdadeiro/Falso, Sucesso/Fracasso).

**Exemplos de uso**:

* + Prever se um cliente irá comprar ou não um produto.
  + Prever se um paciente tem ou não uma determinada doença (classificação binária).
  + Prever categorias de objetos, como espécies de flores em datasets multiclasse.
* **Shuffle (from sklearn.utils import shuffle)**

**Quando usar**: Importar shuffle da biblioteca sklearn.utils é útil quando você deseja **embaralhar ou randomizar a ordem dos dados**. Isso é particularmente importante em **aprendizado de máquina** para evitar **overfitting** (superajuste) e garantir que o modelo não aprenda padrões indesejados na ordem dos dados. O embaralhamento é comum ao dividir um conjunto de dados em treino e teste, garantindo que ambos tenham uma distribuição aleatória de observações.

**Exemplos de uso**:

* + Embaralhar dados antes de dividir em conjuntos de treino e teste.
  + Evitar que o modelo de aprendizado de máquina dependa de uma sequência específica nos dados.
* **train\_test\_split (from sklearn.model\_selection import train\_test\_split)**

A função train\_test\_split é importada do módulo model\_selection da biblioteca scikit-learn. Esta função é utilizada para dividir um conjunto de dados em duas partes: um conjunto de treinamento e um conjunto de teste.

* **KNeighborsClassifier (from sklearn.neighbors import KneighborsClassifier)**

**from sklearn.neighbors:** Este trecho indica que estamos importando algo do módulo neighbors da biblioteca scikit-learn. O módulo neighbors contém várias implementações de algoritmos de aprendizado de máquina que utilizam a ideia de vizinhança (ou proximidade) para fazer previsões.

**import KNeighborsClassifier:** A classe KNeighborsClassifier é uma implementação do algoritmo K-Nearest Neighbors (KNN) para classificação. O KNN é um dos algoritmos de aprendizado supervisionado mais simples e populares, sendo amplamente utilizado para problemas de classificação.

O algoritmo KNN funciona da seguinte forma:

1. Para classificar um novo ponto de dados, ele calcula a distância entre esse ponto e todos os pontos de dados no conjunto de treinamento.
2. Em seguida, ele identifica os k pontos mais próximos (vizinhos) ao ponto a ser classificado.
3. A classe do novo ponto é determinada pela maioria das classes dos vizinhos mais próximos. Ou seja, se a maioria dos k vizinhos pertence à classe A, o novo ponto será classificado como classe A.

* **ConfusionMatrixDisplay (from sklearn.metrics import ConfusionMatrixDisplay)**

Importando a classe ConfusionMatrixDisplay do módulo metrics, que contém várias funções e classes úteis para avaliar modelos de aprendizado de máquina.

ConfusionMatrixDisplay é usada para exibir a matriz de confusão de maneira visual.

**Carregar dados**

* **Datasets existentes (from sklearn.datasets import load\_xxxxxx)**

**Quando usar**: Importar **conjuntos de dados existentes** a partir da biblioteca scikit-learn.

**Exemplos:**

* **load\_iris**, **load\_wine**, **load\_breast\_cancer**, **load\_digits**: Datasets reais, comumente usados em tarefas de classificação.
* **load\_diabetes**: Usado para regressão.
* **fetch\_20newsgroups**: Focado em **NLP** (classificação de textos).
* **make\_classification**, **make\_regression**, **make\_blobs**: Geradores de **dados sintéticos**, úteis para testar e visualizar algoritmos sem depender de conjuntos de dados reais.
* **Datasets carregados**

**Ler dados**

* **variavel = load\_variavel()**

**varivel**

**variável=load\_variavel()** **-** O método carrega o conjunto de dados xxx e os armazena na variável

O dataset é retornado como um objeto do tipo Bunch, que é parecido com um dicionário, contendo tanto os dados como as classes, além de informações adicionais.

**Variável** - Quando você simplesmente digita variavel em um ambiente interativo, ele mostra o conteúdo da variável. Isso ajuda a visualizar o que está carregado no dataset, incluindo dados e metadados.

**Criar dataframe**

* **df = pd.DataFrame(variavel.data, columns=variavel.feature\_names)**

**df['target']= variavel.target**

**df.sample(5, random\_state=42)**

**df = pd.DataFrame(variavel.data,columns=variavel.feature\_names)**

A função pd.DataFrame() cria um DataFrame a partir dos dados do conjunto de dados iris. O parâmetro variavel.data contém os valores do conjunto de dados.

O parâmetro columns=variavel.feature\_names define os nomes das colunas do DataFrame, que são os nomes das variáveis dos conjunto de dados.

**df['target']= variavel.target**

A linha df['target'] = variavel.target adiciona uma nova coluna chamada target ao DataFrame df. Esta coluna contém os valores-alvo (targets) do conjunto de dados, que representam as classes que se quer classificar.

**df.sample(5, random\_state=42)**

A função df.sample(5, random\_state=42) seleciona aleatoriamente 5 linhas do DataFrame. O parâmetro random\_state=42 garante que a amostragem seja reproduzível (ou seja, as mesmas 5 linhas serão selecionadas se o script for executado novamente com este valor).

**Preparar os dados**

* **X = variavel.data**

A variável X recebe os valores das variáveis independentes (ou características) do conjunto de dados.

O conjunto contém características que são usadas para prever a classe do valor dependente.

* **y = variavel.target**

A variável y recebe os valores da variável dependente, que é o alvo (*target*) no conjunto de dados. O alvo neste caso são as classes relacionadas ao conjunto de dados.

Esta variável contém a resposta que queremos prever com base nas características contidas em X.

* **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42, stratify=y)**

**X e y:** Os dados de entrada (X, que contém as características) e a variável de saída (y, que contém as classes) são passados como argumentos para a função.

**test\_size=0.20**: Este parâmetro define a proporção dos dados que será reservada para o conjunto de teste. Neste caso, 20% dos dados serão usados para teste, enquanto os 80% restantes serão usados para treinamento.

**random\_state=42:** Este parâmetro garante que a divisão dos dados seja reproduzível. O valor 42 é um número arbitrário que define a semente do gerador de números aleatórios. Ao usar o mesmo valor de random\_state, você garante que a mesma divisão dos dados ocorra a cada execução do código.

**stratify=y:** Este parâmetro garante que a divisão dos dados preserve a proporção das classes no conjunto de dados original. Ou seja, se, por exemplo, 30% são da classe A, 40% são da B, e 30% são C, essa mesma proporção será mantida tanto no conjunto de treinamento quanto no de teste. Isso é especialmente importante em conjuntos de dados desbalanceados, onde algumas classes podem ter muito mais amostras do que outras.

**X\_train:** Contém as características (features) do conjunto de treinamento.

**X\_test:** Contém as características (features) do conjunto de teste.

**y\_train:** Contém os rótulos (classes) do conjunto de treinamento.

**y\_test:** Contém os rótulos (classes) do conjunto de teste.

**Treinar o modelo**

* **knn = KNeighborsClassifier(n\_neighbors=3)**

**knn.fit(X\_train,y\_train)**

**knn.score(X\_test, y\_test)**

**KNeighborsClassifier(n\_neighbors=3):** Aqui, uma nova instância do classificador KNN é criada.

n\_neighbors=3: Este parâmetro define o número de vizinhos mais próximos que o algoritmo considerará ao fazer a previsão. No caso, o modelo usará 3 vizinhos para determinar a classe de um novo ponto de dados.

**knn.fit(X\_train, y\_train):** O método fit é chamado para treinar o modelo KNN com os dados de treinamento.

X\_train: As características (features) do conjunto de treinamento.

y\_train: Os rótulos (classes) correspondentes do conjunto de treinamento.

Durante o treinamento, o modelo armazena os pontos de dados e suas classes para que possa fazer previsões posteriormente com base nessa informação.

**knn.score(X\_test, y\_test):** O método score é utilizado para avaliar o desempenho do modelo treinado.

X\_test: As características do conjunto de teste, que o modelo não viu durante o treinamento.

y\_test: Os rótulos (classes) verdadeiros do conjunto de teste.

O método retorna a precisão (accuracy) do modelo, que é a proporção de previsões corretas feitas pelo modelo em relação ao total de previsões. Um valor de precisão mais alto indica que o modelo está fazendo boas previsões.

**Avaliando o modelo**

* **ConfusionMatrixDisplay.from\_estimator(knn, X\_test, y\_test, display\_labels=variavel.target\_names)**

**knn:** O modelo treinado que estamos avaliando (a instância do classificador KNN).

**X\_test:** As características do conjunto de teste. O modelo usará esses dados para fazer previsões.

**y\_test:** Os rótulos reais (classes) correspondentes ao conjunto de teste. Esses rótulos são usados para comparar as previsões do modelo.

**display\_labels=variavel.target\_names:** Este parâmetro define os rótulos que serão usados na matriz de confusão. variavel.target\_names é um array que contém os nomes das classes no conjunto de dados. Isso ajuda a tornar a matriz de confusão mais legível, pois em vez de ver números (0, 1, 2), você verá os nomes das classes.

**Testando o modelo (inferência)**

* **knn\_pred = knn.predict([[5.1, 3.5, 1.4, 0.2]])**

**knn.predict(...):** Este método é usado para fazer previsões com o modelo KNN. Ele aceita um ou mais exemplos de entrada e retorna as classes previstas correspondentes.

[[5.1, 3.5, 1.4, 0.2]]: A entrada é uma lista aninhada (uma lista contendo outra lista) que representa as características do novo item que se quer classificar:

A entrada deve estar em formato de lista, mesmo que haja apenas um exemplo, para que o método predict funcione corretamente.

**knn\_pred**: O resultado da predição é armazenado na variável knn\_pred. Esse resultado será a classe prevista para o novo item, representando a classe que o modelo acredita ser a correta com base nas características fornecidas.

* **print('nova predição:\nclasse = ', knn\_pred)**

**print('classe = ', variavel.target\_names[knn\_pred])**