4️⃣

Lab04 (treinamento e testes)

- Lógica por trás da estratégia de treinamento e teste utilizando a função `train_test_split` do Scikit-learn
  - A estratégia de treinamento e teste é uma abordagem comum para avaliar o desempenho de um modelo de machine learning. Ela divide o conjunto de dados disponível em dois subconjuntos:
    1. **Conjunto de Treinamento**:
       - Este subconjunto é usado para treinar o modelo. Ele contém uma parte dos dados, geralmente a maioria, que o modelo utilizará para aprender os padrões nos dados.
    2. **Conjunto de Teste**:
       - Este subconjunto é usado para avaliar o desempenho do modelo após o treinamento. Ele contém uma parte dos dados que o modelo não viu durante o treinamento e é usado para medir quão bem o modelo generaliza para novos dados não vistos.
  - Esses conjuntos de treinamento e teste são então usados para treinar o modelo (por exemplo, regressão linear) com os dados de treinamento e avaliar seu desempenho com os dados de teste. Ao dividir os dados dessa maneira, podemos obter uma estimativa do desempenho do modelo em dados não vistos, o que é crucial para avaliar sua capacidade de generalização. Isso nos permite entender se o modelo está aprendendo padrões úteis nos dados ou apenas memorizando o conjunto de treinamento

1. **Importação de Bibliotecas**:
   - `from sklearn import datasets, linear_model`: Importa o módulo `datasets` do Scikit-learn, que fornece conjuntos de dados de exemplo, e o modelo

`linear_model`, que contém implementações de modelos de regressão linear.

  - A regressão linear é um método estatístico utilizado para modelar a relação entre uma variável dependente (também conhecida como variável resposta ou alvo) e uma ou mais variáveis independentes (também conhecidas como variáveis preditoras ou características). O objetivo da regressão linear é encontrar a melhor linha reta que descreve a relação entre as variáveis, de forma a prever ou estimar os valores da variável dependente com base nos valores das variáveis independentes. A regressão linear é comumente usada para modelar relações lineares entre variáveis em muitas áreas, como estatísticas, ciências sociais, economia, engenharia e ciência de dados.

- `import numpy as np`: Importa a biblioteca NumPy, frequentemente usada para operações numéricas em Python.
- `from sklearn.model_selection import KFold`: Importa a classe `KFold` do Scikit-learn, que permite a implementação da estratégia de validação cruzada K-Folds.
  - O objetivo da validação cruzada K-Folds é estimar o desempenho do modelo em dados não vistos de maneira mais precisa e robusta, utilizando todos os dados disponíveis tanto para treinamento quanto para teste. Isso é especialmente útil quando o conjunto de dados é limitado e não pode ser dividido em conjuntos de treinamento e teste grandes o suficiente para uma avaliação confiável. Funcionamento:
    - Divisão dos Dados: Os dados são divididos em K subconjuntos (folds) de tamanho aproximadamente igual.
    - Treinamento e Teste do Modelo: O modelo é treinado K vezes, cada vez usando K-1 dos subconjuntos como conjunto de treinamento e o subconjunto restante como conjunto de teste.Por exemplo, na primeira iteração, o primeiro fold é usado como conjunto de teste e os K-1 folds restantes são usados como conjunto de treinamento. Na segunda iteração, o segundo fold é usado como conjunto de teste e assim por diante.Isso resulta em K modelos treinados e avaliados.

        - Avaliação do Desempenho: O desempenho do modelo é avaliado calculando-se uma métrica de desempenho em cada iteração, como a acurácia, precisão, recall, F1-score, coeficiente de determinação (R^2), entre outras, dependendo do tipo de problema.A métrica de desempenho final é calculada como a média ou a mediana das métricas de desempenho obtidas nas K iterações.
    - Além disso, a validação cruzada K-Folds ajuda a reduzir a variância na estimativa do desempenho do modelo, fornecendo uma avaliação mais estável e geral do modelo. Isso é importante para garantir que o modelo tenha uma capacidade de generalização consistente em diferentes subconjuntos de dados.
- `from sklearn.model_selection import train_test_split`: Importa a função `train_test_split` do Scikit-learn, que permite dividir os dados em conjuntos de treinamento e teste.
    - A função `train_test_split` é uma ferramenta essencial para a avaliação de modelos de machine learning, pois divide os dados em duas partes aleatórias e não sobrepostas: um **conjunto de treinamento** e um **conjunto de teste**. Você especifica a proporção dos dados que deseja alocar para cada conjunto, geralmente usando o parâmetro `test_size`. Por exemplo, `test_size=0.2` reservará 20% dos dados para o conjunto de teste e usará os 80% restantes para treinamento.

```python
# Importação de Bibliotecas
import pandas as pd
from sklearn import datasets, linear_model
from matplotlib import pyplot as plt
import numpy as np
from sklearn.model_selection import KFold # Para estratégia de Cross-Validadion (K-Folds)
from sklearn.model_selection import train_test_split # Para estratégia Train/Test Split
```

✓ 0.0s

2. Leitura de Dataset - Diabetes, do SKLearn:

- `columns = "age sex bmi map tc ldl hdl tch ltg glu".split()`: Define os nomes das colunas do conjunto de dados.
- `diabetes = datasets.load_diabetes()`: Carrega o conjunto de dados de diabetes fornecido pelo Scikit-learn.
- `df = pd.DataFrame(diabetes.data, columns=columns)`: Converte os dados do conjunto de dados em um DataFrame do Pandas, usando os nomes das colunas definidos anteriormente.
- `y = diabetes.target`: Define a variável alvo `y` como o atributo `target` do conjunto de dados, que contém os valores a serem previstos pelo modelo de regressão.

```
# Leitura de Dataset - Diabetes, do SKLearn

columns = "age sex bmi map tc ldl hdl tch ltg glu".split() # Declara os nomes das colunas
diabetes = datasets.load_diabetes() # Carrega o dataset diabetes de sklearn
df = pd.DataFrame(diabetes.data, columns=columns) # Carrega o dataset como um data frame
y = diabetes.target # define a variavel target variable (varivavel dependente, no modelo de regressão)
✓ 0.0s
```

3. **Treinamento e Testes - Estratégia: Train/Test Split (20% para Testes)**:
   - `X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.2)`: Divide os dados em conjuntos de treinamento e teste, onde 80% dos dados são usados para treinamento (`X_train`, `y_train`) e 20% para teste (`X_test`, `y_test`). `df` contém os recursos (variáveis independentes) e `y` contém o alvo (variável dependente) a ser previsto.
     - Após dividir os dados, os conjuntos resultantes (`X_train`, `X_test`, `y_train`, `y_test`) são retornados pela função.
       - `X_train`: Contém as características (ou atributos) do conjunto de treinamento.
       - `X_test`: Contém as características do conjunto de teste.
       - `y_train`: Contém os rótulos (ou alvos) correspondentes ao conjunto de treinamento.
       - `y_test`: Contém os rótulos correspondentes ao conjunto de teste.

- `print(X_train.shape, y_train.shape)`: Imprime a forma (número de linhas, número de colunas) dos conjuntos de treinamento (`X_train`, `y_train`), que agora contêm 80% dos dados.
- `print(X_test.shape, y_test.shape)`: Imprime a forma (número de linhas, número de colunas) dos conjuntos de teste (`X_test`, `y_test`), que agora contêm 20% dos dados.

```
# Treinamento e Testes - Estratégia: Train/Test Split (20% para Testes)

X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.2)
print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)

✓ 0.0s
(353, 10) (353,)
(89, 10) (89,)
```

Resultado: 353 amostras no conjunto de treinamento, com 10 características cada, e há 89 amostras no conjunto de teste, com 10 características cada. O tamanho dos alvos correspondentes (valores a serem previstos) corresponde ao número de amostras em cada conjunto, sem especificar o número de características (pois é apenas uma dimensão, não necessita de duas dimensões como os recursos). Esses conjuntos de dados agora estão prontos para serem usados no treinamento e na avaliação de modelos de regressão linear.

4. **Construção e Aplicação do Modelo: Regressão Linear**:
   - `lm = linear_model.LinearRegression()`: Cria uma instância do modelo de regressão linear.
   - `model = lm.fit(X_train, y_train)`: Treina o modelo de regressão linear com os dados de treinamento (`X_train`, `y_train`).
   - `predictions = lm.predict(X_test)`: Faz previsões usando o modelo treinado nos dados de teste (`X_test`). As previsões são armazenadas na variável `predictions`.

- `predictions[0:5]`: Exibe as primeiras cinco previsões geradas pelo modelo.

```
# Construção e Aplicação do Modelo: Regressão Linear

lm = linear_model.LinearRegression()
model = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
predictions[0:5]
```

✓ 0.0s

```
array([ 81.14256015, 133.67165487,  90.22210267, 186.57975309,
        65.58522479])
```

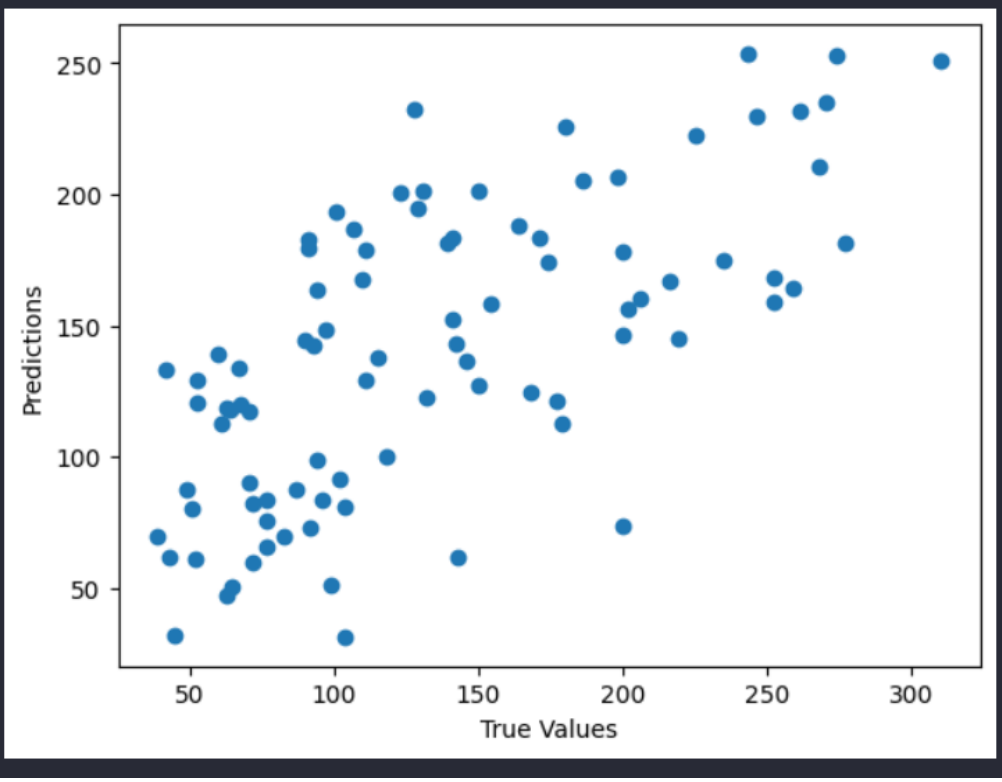
Resultado: O array `predictions` contém as previsões do modelo para os dados de teste. Cada valor no array representa a previsão do modelo para um exemplo de teste específico.

5. **Exibição do Modelo: Scatter Plot**:
   - `plt.scatter(y_test, predictions)`: Plota um gráfico de dispersão com os valores reais (`y_test`) no eixo x e as previsões (`predictions`) no eixo y.
   - `plt.xlabel("True Values")`: Define o rótulo do eixo x como "True Values" (Valores Reais).
   - `plt.ylabel("Predictions")`: Define o rótulo do eixo y como "Predictions" (Previsões).

```
# Exibição do Modelo: Scatter Plot

plt.scatter(y_test, predictions)
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

✓ 0.1s

```
Text(0, 0.5, 'Predictions')
```



Resultado: Um gráfico de dispersão é exibido, mostrando como as previsões do modelo se comparam aos valores reais. Idealmente, os pontos no gráfico deveriam estar alinhados próximos a uma linha diagonal, o que indicaria que as previsões do modelo são consistentes com os valores reais.

6. **Exibição de percentual de acurácia**:

- `print("Score:", model.score(X_test, y_test))`: Calcula e imprime o coeficiente de determinação (R^2) do modelo nos dados de teste. O coeficiente de determinação é uma medida da qualidade das previsões do modelo,

variando de 0 a 1, onde 1 indica uma correspondência perfeita entre as previsões e os valores reais.

```
# Exibição de percentual de acurácia

print ("Score:", model.score(X_test, y_test))
```

✓ 0.0s

```
Score: 0.43395442926796O9
```

Resultado: O coeficiente de determinação calculado (`0.43395442926796O9`) representa o percentual de variabilidade nos dados de teste que é explicado pelo modelo. Quanto mais próximo de 1, melhor o desempenho do modelo.

7. **Treinamento e Testes - Estratégia: Cross Validation (2 folds)**:
   - `kf = KFold(n_splits=2)`: Cria uma instância do KFold com 2 folds.
   - `kf.get_n_splits(X)`: Retorna o número de iterações de separação (splitting) do cross-validator.
   - `print(kf)`: Imprime informações sobre o objeto KFold.

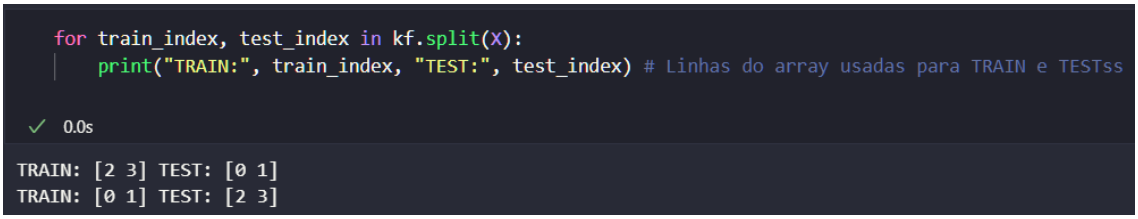
```
# Treinamento e Testes - Estratégia: Cross Validation (2 folds)

X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]]) # Cria um array como dataset exemplo
y = np.array([1, 2, 3, 4]) # Cria um outro array como dataset exemplo
kf = KFold(n_splits=2) # Define a separação (split) em 2 folds (k=2)

kf.get_n_splits(X) # Retorna o numero de iterações de separação (splitting) do cross-validator
print(kf)
```

✓ 0.0s

```
KFold(n_splits=2, random_state=None, shuffle=False)
```

Resultado: O objeto KFold é exibido, mostrando o número de splits (divisões), o estado aleatório e se há embaralhamento dos dados.

- O loop `for train_index, test_index in kf.split(X):` itera sobre cada split (divisão) gerado pelo KFold.
- `print("TRAIN:", train_index, "TEST:", test_index)`: Imprime os índices das linhas do array de dados (`X`) usadas para treinamento e teste em cada fold.



Resultado: Para cada iteração do loop, os índices das linhas usadas para treinamento e teste em cada fold são exibidos. Isso mostra como os dados são divididos em folds durante a validação cruzada.