1. Escreva uma função que receba uma lista de números e retorne outra lista com os números ímpares.

def encontrar\_impares(lista\_numeros):  
 impares = []  
 for numero in lista\_numeros:  
 if numero % 2 != 0:  
 impares.append(numero)

return impares  
# Exemplo de uso

numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

lista\_impares = encontrar\_impares(numeros)

print(f'Lista original: {numeros}')

print(f'Números ímpares: {lista\_impares}')

2. Escreva uma função que receba uma lista de números e retorne outra lista com os números primos presentes.

def is\_primo(num):

if num <= 1:

return False

# Checa divisores de 2 até a raiz quadrada do número

for i in range(2, int(num\*\*0.5) + 1):

if num % i == 0:

return False

return True

def encontrar\_primos(lista\_numeros):

primos = []

for numero in lista\_numeros:

if is\_primo(numero):

primos.append(numero)

return primos

# Exemplo de uso

numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13]

lista\_primos = encontrar\_primos(numeros)

print(f'Lista original: {numeros}')

print(f'Números primos: {lista\_primos}')

3. Escreva uma função que receba duas listas e retorne outra lista com os elementos que estão presentes em apenas uma das listas.

def elementos\_em\_uma\_lista(lista1, lista2):

set1 = set(lista1)

set2 = set(lista2)

# Encontra elementos que estão em set1, mas não em set2

apenas\_na\_lista1 = set1.difference(set2)

# Encontra elementos que estão em set2, mas não em set1

apenas\_na\_lista2 = set2.difference(set1)

# Converte de volta para lista e combina

return list(apenas\_na\_lista1.union(apenas\_na\_lista2))

# Exemplo de uso

lista\_a = [1, 2, 3, 4]

lista\_b = [3, 4, 5, 6]

elementos\_unicos = elementos\_em\_uma\_lista(lista\_a, lista\_b)

print(f'Lista 1: {lista\_a}')

print(f'Lista 2: {lista\_b}')

print(f'Elementos em apenas uma das listas: {elementos\_unicos}')

4. Dada uma lista de números inteiros, escreva uma função para encontrar o segundo maior valor na lista.

def segundo\_maior\_valor(lista\_numeros):

if len(lista\_numeros) < 2:

return "A lista deve ter pelo menos dois elementos."

maior = -float('inf')

segundo\_maior = -float('inf')

for numero in lista\_numeros:

if numero > maior:

segundo\_maior = maior

maior = numero

elif numero > segundo\_maior and numero != maior:

segundo\_maior = numero

if segundo\_maior == -float('inf'):

return "Todos os elementos são iguais."

return segundo\_maior

# Exemplo de uso

numeros = [10, 5, 20, 15, 25, 20]

segundo\_maior = segundo\_maior\_valor(numeros)

print(f'Lista original: {numeros}')

print(f'O segundo maior valor é: {segundo\_maior}')

5. Crie uma função que receba uma lista de tuplas, cada uma contendo o nome e a idade de uma pessoa, e retorne a lista ordenada pelo nome das pessoas em ordem alfabética.

def ordenar\_por\_nome(lista\_pessoas):

# A função lambda pega o primeiro elemento da tupla (o nome) para a ordenação

return sorted(lista\_pessoas, key=lambda x: x[0])

# Exemplo de uso

pessoas = [('João', 30), ('Ana', 25), ('Pedro', 35), ('Maria', 28)]

pessoas\_ordenadas = ordenar\_por\_nome(pessoas)

print(f'Lista original: {pessoas}')

print(f'Lista ordenada por nome: {pessoas\_ordenadas}')

6. Como identificar e tratar outliers em uma coluna numérica usando desvio padrão ou quartis?

Outliers são valores que se desviam significativamente dos outros dados. Duas formas comuns de identificá-los e tratá-los são:

* **Método do Desvio Padrão:** Assume que os dados seguem uma distribuição normal (ou similar). Valores que estão a uma certa distância da média (geralmente 2 ou 3 desvios padrão) são considerados outliers.
  + **Identificação:** Calcule a média (mean) e o desvio padrão (std) da coluna. Os limites superior e inferior são definidos como média ± (N \* desvio\_padrão), onde N é um multiplicador (ex: 3). Valores fora desse intervalo são outliers.
  + **Tratamento:** Você pode **remover** as linhas com outliers, **substituí-los** pela média/mediana (técnica de "imputação") ou **limitar** os valores ao limite superior/inferior (técnica de "capping").
* **Método dos Quartis (IQR):** Mais robusto e não assume uma distribuição específica. É baseado no Intervalo Interquartil (IQR).
  + **Identificação:** Calcule o primeiro quartil (**Q1**) e o terceiro quartil (**Q3**). O **IQR** é a diferença entre eles (Q3 - Q1). Os limites superior e inferior são definidos como Q1 - 1.5 \* IQR e Q3 + 1.5 \* IQR. Valores fora desse intervalo são outliers.
  + **Tratamento:** As estratégias de tratamento são as mesmas do método do desvio padrão: remoção, substituição ou capping.

7. Como concatenar vários DataFrames (empilhando linhas ou colunas), mesmo que tenham colunas diferentes? Dica: Utiliza-se pd.concat() especificando axis=0 (linhas) ou axis=1 (colunas). Quando há colunas diferentes, os valores ausentes são preenchidos com NaN.

import pandas as pd

df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})

df2 = pd.DataFrame({'C': [5, 6], 'D': [7, 8]})

df3 = pd.DataFrame({'A': [9, 10], 'C': [11, 12]})

# Concatenar linhas com colunas diferentes

df\_linhas = pd.concat([df1, df3], axis=0)

print("Concatenação por linhas:")

print(df\_linhas)

# Resultado: NaN em colunas não presentes nos DFs originais

# Concatenar colunas com índices diferentes

df\_colunas = pd.concat([df1, df2], axis=1)

print("\nConcatenação por colunas:")

print(df\_colunas)

# Resultado: Combinações lado a lado

8. Utilizando pandas, como realizar a leitura de um arquivo CSV em um DataFrame e exibir as primeiras linhas?

import pandas as pd

# Supondo que você tem um arquivo chamado 'dados.csv'

# df = pd.read\_csv('dados.csv')

# Exemplo com um DataFrame fictício para demonstração

data = {'Nome': ['Maria', 'João', 'Ana', 'Pedro'],

'Idade': [28, 35, 22, 41],

'Cidade': ['SP', 'RJ', 'BH', 'SP']}

df = pd.DataFrame(data)

# Exibe as primeiras 5 linhas por padrão

print(df.head())

# Exibe um número específico de linhas (ex: 3)

print(df.head(3))

9. Utilizando pandas, como selecionar uma coluna específica e filtrar linhas em um “DataFrame” com base em uma condição?

import pandas as pd

data = {'Nome': ['Maria', 'João', 'Ana', 'Pedro'],

'Idade': [28, 35, 22, 41],

'Cidade': ['SP', 'RJ', 'BH', 'SP']}

df = pd.DataFrame(data)

# Selecionar a coluna 'Idade'

idades = df['Idade']

print("Coluna 'Idade':")

print(idades)

# Filtrar linhas onde a idade é maior que 30

df\_filtrado = df[df['Idade'] > 30]

print("\nPessoas com mais de 30 anos:")

print(df\_filtrado)

10.Utilizando pandas, como lidar com valores ausentes (NaN) em um DataFrame?

Valores ausentes (geralmente representados como NaN) são um problema comum. O pandas oferece várias formas de lidar com eles.

* **Identificação:**
  + .isnull() ou .isna(): Retorna um DataFrame ou Series de booleanos (True se for NaN, False caso contrário).
  + .isnull().sum(): Conta o número de NaNs em cada coluna.
* **Tratamento:**
  + **Remover (.dropna()):**
    - df.dropna(): Remove todas as linhas que contêm pelo menos um valor NaN.
    - df.dropna(axis=1): Remove todas as colunas que contêm pelo menos um valor NaN.
  + **Preencher (.fillna()):**
    - df.fillna(0): Preenche todos os NaNs com um valor fixo (neste caso, 0).
    - df.fillna(df.mean()): Preenche os NaNs de cada coluna numérica com a média dessa coluna.
    - df.fillna(method='ffill'): Preenche os NaNs com o valor da linha anterior (forward-fill).
    - df.fillna(method='bfill'): Preenche os NaNs com o valor da próxima linha (backward-fill).