```
import pandas as pd
```

Os tópicos a seguir foram retirados de estudos do curso: "Data Science e Machine Learning - Asimov Academy, e de outros locais de sites da internet.

2. Analisando dados com Pandas

2.1. Conceitos básicos de Pandas

A Ciência de Dados, ou a Análise de Dados, é um ramo que vem ganhando cada vez mais notoriedade, várias empresas de pequeno a grande porte, como a Netflix, Airbnb e Google já possuem atividades de tomada de decisão baseadas em dados. Nesse cenário, a linguagem Python é bastante utilizada devido a sua versatilidade e simplicidade, contando com uma vasta quantidade de bibliotecas, e entre elas, o Pandas.

O que é Pandas?

Pandas é uma biblioteca de código aberto (open source), construída sobre a linguagem Python, e que providencia uma abordagem rápida e flexível, com estruturas robustas para se trabalhar com dados relacionais (ou rotulados), e tudo isso de maneira simples e intuitiva.

De maneira geral, o Pandas pode ser utilizado para várias atividades e processos, entre eles: **limpeza e tratamento de dados, análise exploratória de dados (EDA)**, suporte em atividades de Machine Learning, consultas e queries em bancos de dados relacionais, visualização de dados, webscraping e muito mais. E além disso, também possui ótima integração com várias outras bibliotecas muito utilizadas em Ciência de Dados, tais como: Numpy, Scikit-Learn, Seaborn, Altair, Matplotlib, Plotly, Scipy e outros.

2.4. Series

A estrutura principal do Pandas é composta por dois tipos de objetos: *Series e DataFrames*, vamos falar um pouco sobre o primeiro tipo;

As *Series* nada mais são que uma espécie de arranjo unidimensional, como uma lista por exemplo, mas que possui algumas características diferentes, uma delas é que possui rótulos para cada elemento do array, ou seja, **uma Série é um array unidimensional capaz de armazenar qualquer tipo de dado e vem com um índice que nos ajuda a localizar esses dados rapidamente.** Para facilitar, pense na *Serie* como uma coluna de uma tabela no Excel.

Uma série tem 4 partes importantes:

- Os elementos em si
- O índice que contém a referência para acessar os elementos
- O tipo dos elementos
- Um nome

Elementos e Tipos

Os elementos podem ser de qualquer tipo, ou seja, podemos ter uma série com números e strings, por exemplo.

Abaixo, criamos duas séries de exemplo de forma bem parecida como criamos a lista, com a exceção de que as criamos a partir da classe Series do pandas:

```
serie = pd.Series([42, 99, -1])
serie
```

```
0    42
1    99
2    -1
dtype: int64
```

```
serie2 = pd.Series(['radiohead', 2.3, True])
serie2
```

```
0    radiohead
1          2.3
2         True
dtype: object
```

Acessando elementos

Numa lista, acessamos os elementos por meio de índices posicionais, numéricos, certo?

Acessar o primeiro elemento: lista[0], o terceiro elemento: lista[2], e assim por diante.

Nas séries podemos acessar da mesma forma, porém, podemos acessar os elementos de uma *serie* com um índice posicional, mas não precisa ser assim, podemos criar um índice próprio que nem precisa ser numérico.

Vamos criar um caso de exemplo, queremos guardar as calorias de cada alimento que vamos ingerir... E com isso criamos uma serie com as calorias de uma banana, um prato feito e um big mac:

```
serie_sem_nome = pd.Series([200, 350, 550])
serie_sem_nome
```

```
0    200
1    350
```

```
2    550
dtype: int64
```

```
# Vamos agora dar nomes aos indices e assim saberemos quais calorias é de quais alimentos:
serie_com_nome = pd.Series([200, 350, 550], index=['banana', 'prato feito', 'big mac'])
serie_com_nome
```

```
banana         200
prato feito    350
big mac        550
dtype: int64
```

```
# Quantas calorias tem um big mac?
serie_com_nome['big mac']
```

```
np.int64(550)
```

Vamos para mais um exemplo, para conseguirmos entender sobre acessar os elementos de uma **serie**

```
labels = ['a', 'b', 'c']
minha_lista = [10, 20, 30]
d = {'a':10, 'b':20, 'c':30} # Um dicionário para utilizar no Series
```

Uma observação importante: Não preciso especificar qual será meu *data* ou meu *index*, o pandas identifica automáticamente quando eu coloco em ordem, veja os exemplos abaixo.

```
pd.Series(labels)
```

```
0    a
1    b
2    c
dtype: object
```

```
pd.Series(labels, minha_lista)
# A mesma coisa seria: pd.Series(data=labels, index=minha_lista)
```

```
10    a
20    b
30    c
dtype: object
```

Eu também posso utilizar dicionários no Series, e o interessante é que, automaticamente o *pandas* identifica a chave do dicionário como o indice do elemento. Vejamos abaixo:

```
pd.Series(d)
```

```
a    10
b    20
c    30
dtype: int64
```

```
d2 = {"banana": 34, "uva": 21, "laranja": 22} # Quantidade de frutas que eu tenho
pd.Series(d2)
```

```
banana     34
uva        21
laranja    22
dtype: int64
```

```
# Quantas bananas eu tenho?
d2['banana']
```

```
34
```

2.5. DataFrames

O DataFrame do Pandas é uma maneira de representar e trabalhar com dados tabulares. Ele pode ser visto como uma tabela que organiza os dados em linhas e colunas, criando uma estrutura de dados bidimensional. Um DataFrame pode ser criado do zero ou você pode usar outras estruturas de dados, como matrizes NumPy.

Em outras palavras: DataFrames são estruturas 2D (linhas e colunas) — parecida com uma planilha do Excel ou uma tabela SQL.

Criando DataFrames

```
# Criando DataFrames a partir de dicionários:

dados = {
'Nome': ['Ana', 'Bruno', 'Carlos'],
'Idade': [25, 30, 35],
'Cidade': ['São Paulo', 'Rio de Janeiro', 'Belo Horizonte']
}

df = pd.DataFrame(dados)
print(df)
```

```
     Nome  Idade          Cidade
0     Ana     25       São Paulo
1   Bruno     30  Rio de Janeiro
2  Carlos     35  Belo Horizonte
```

```
# Criando DataFrames a partir de listas:
df2 = pd.DataFrame({'Calorias':[200, 350, 550], 'Gordura (%)':[0, 6, 15]}, index=
```

```
['banana', 'prato feito', 'big mac'])
print(df2)
print()
```

```
             Calorias  Gordura (%)
banana            200            0
prato feito       350            6
big mac           550           15
```

Acessando DataFrames

Quando trabalhamos com Series, acessamos atraves do indice, já nos DataFrammes, acessamos atraves do nome da coluna

```
df[['Cidade']] # Retornar a consulta como DataFrame
```

border="1" class="dataframe">

	Cidade
0	São Paulo
1	Rio de Janeiro
2	Belo Horizonte

```
df['Cidade'] # Retornar a consulta como Serie
```

```
0         São Paulo
1    Rio de Janeiro
2    Belo Horizonte
Name: Cidade, dtype: object
```

Adicionando colunas

```
df['new'] = df['Idade']
```

```
df
```

border="1" class="dataframe">

	Nome	Idade	Cidade	new
0	Ana	25	São Paulo	25
1	Bruno	30	Rio de Janeiro	30
2	Carlos	35	Belo Horizonte	35

```
# Quantas calorias eu consumiria ao comer duas vezes:
df2['Total_Calorias'] = df2['Calorias'] * 2
```

```
df2
```

border="1" class="dataframe">

	Calorias	Gordura (%)	Total_Calorias
banana	200	0	400
prato feito	350	6	700
big mac	550	15	1100

Deletando colunas do DataFrame

Um observação curiosa é que, no Pandas, se não for especificado, ele ira trabalhar com uma especie de copia do DataFrame, e você trabalhará com isso, até que seja especificado que você quer que seja o original. O que isso quer dizer?

Vamos atentar ao exemplo abaixo:

```
df.drop('new', axis=1)

# axis=0 corresponde a linhas
# axis=1 corresponde a colunas
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
1	Bruno	30	Rio de Janeiro
2	Carlos	35	Belo Horizonte

O comando: *df.drop('new', axis=1)* diz para o Pandas apagar a coluna que tem o nome 'new'. Porém, ele trabalha com o 'como seria' se for feito dessa maneira.

Se eu for acessar novamente esse dataframe, a coluna 'new' ainda estará lá:

```
df
```

border="1" class="dataframe">

	Nome	Idade	Cidade	new
0	Ana	25	São Paulo	25
1	Bruno	30	Rio de Janeiro	30
2	Carlos	35	Belo Horizonte	35

Então, como eu apago definitivamente a coluna?

Existem diversas maneiras, uma delas é salvar em um novo DataFrame:

```
df3 = df.drop('new', axis=1)
df3
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
1	Bruno	30	Rio de Janeiro
2	Carlos	35	Belo Horizonte

Ou então, utilizar a flag da própria função que permite fazer isso:

```
df.drop('new', axis=1, inplace=True) # Está dizendo que quero deletar do DataFrame Original sem cópias
```

```
df
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
1	Bruno	30	Rio de Janeiro
2	Carlos	35	Belo Horizonte

🔍 Principais operações com DataFrame:

>

Comando	O que faz	Exemplo
`.head()`	Mostra as 5 primeiras linhas	`df.head()`
`.tail()`	Mostra as 5 últimas linhas	`df.tail()`
`.shape`	Retorna (linhas, colunas)	`df.shape`
`.columns`	Mostra os nomes das colunas	`df.columns`
`.index`	Mostra os índices	`df.index`
Acessar uma coluna	Retorna uma Series	`df['Nome']`
Acessar várias colunas	Retorna outro DataFrame com colunas escolhidas	`df[['Nome', 'Idade']]`
Acessar uma linha	Retorna uma linha específica	`df.loc[0]`
Filtrar linhas	Retorna linhas com condição lógica	`df[df['Idade'] > 25]`
Adicionar coluna	Cria uma nova coluna	`df['Altura'] = [1.65, 1.80, 1.70]`
Remover coluna	Remove uma coluna	`df.drop('Cidade', axis=1)`
Ordenar por coluna	Ordena as linhas	`df.sort_values('Idade')`

📝 Exemplo completo:

```python
dados2 = {
    'Nome': ['Ana', 'Bruno', 'Carla'],
    'Idade': [25, 30, 22],
    'Cidade': ['São Paulo', 'Rio de Janeiro', 'Curitiba']
}

df_dados2 = pd.DataFrame(dados2)

print("DataFrame completo:\n", df_dados2)
print("----------------------------------------")
print("\nSomente a coluna Nome:\n", df_dados2['Nome'])
print("----------------------------------------")
print("\nSomente as duas primeiras linhas:\n", df_dados2.head(2))
print("----------------------------------------")
print("\nFiltrando quem tem Idade maior que 23 anos:\n",
df_dados2[df_dados2['Idade'] > 23])
print("----------------------------------------")
print("\nDataFrame ordenado pela Idade:\n", df_dados2.sort_values('Idade'))
print("----------------------------------------")
```

```
DataFrame completo:
     Nome  Idade          Cidade
0    Ana     25       São Paulo
1  Bruno     30  Rio de Janeiro
2  Carla     22        Curitiba
----------------------------------------

Somente a coluna Nome:
 0      Ana
1    Bruno
2    Carla
Name: Nome, dtype: object
----------------------------------------

Somente as duas primeiras linhas:
     Nome  Idade          Cidade
0    Ana     25       São Paulo
1  Bruno     30  Rio de Janeiro
----------------------------------------

Filtrando quem tem Idade maior que 23 anos:
     Nome  Idade          Cidade
0    Ana     25       São Paulo
1  Bruno     30  Rio de Janeiro
----------------------------------------

DataFrame ordenado pela Idade:
     Nome  Idade          Cidade
2  Carla     22        Curitiba
0    Ana     25       São Paulo
1  Bruno     30  Rio de Janeiro
----------------------------------------
```

2.6. Iloc e Filtros

O que é .iloc[]?

.iloc[] é usado para selecionar linhas e colunas pelo número da posição (índice inteiro). Significa: "integer-location based indexing".

Ou seja: você não usa o nome da coluna ou do índice personalizado — usa apenas números (0, 1, 2, ...).

```
# Vamos utilizar o dataFrame anterior: dados2
df_dados2
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
1	Bruno	30	Rio de Janeiro
2	Carla	22	Curitiba

Utilizando o *iloc*:

Atenção: O índice começa do zero, e como consequencia o fatiamento é igual ao range() do Python, por exemplo:

Dado os seguintes valores: (5, 2, 6, 8). Faça um fatiamento de (0, 2).

Resultado: *(5, 2)*

Quando usar .iloc[]?

- ✔️ Quando você quer selecionar pelo número da posição da linha/coluna, e não pelo nome.
- ✔️ Muito útil em laços, funções ou manipulações onde não sabe o nome da coluna.

---

1. Selecionar uma linha especifica:

```
df.iloc[0]
# Vai selecionar a primeira linha do dataFrame
```

```
Nome            Ana
Idade            25
Cidade    São Paulo
Name: 0, dtype: object
```

2. Selecionar uma célula específica (linha, coluna):

```
df.iloc[0, 1] # Primeira linha, segunda coluna

# O resultado será a idade de Ana, 25
```

```
np.int64(25)
```

3. Selecionar um intervalo de linhas (slice):

```
df.iloc[0:2] # Vai retornar as duas primeiras linhas
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
1	Bruno	30	Rio de Janeiro

4. Selecionar uma coluna específica (todas as linhas, coluna 0):

```
df.iloc[:, 0] # Retorna a coluna 'Nome' e todas as linhas da coluna
```

```
0      Ana
1    Bruno
2    Carla
Name: Nome, dtype: object
```

5. Selecionar um "pedaço" específico:

```
df.iloc[0:2, 0:2]    # Linhas 0 e 1, Colunas 0 e 1
```

border="1" class="dataframe">

	Nome	Idade
0	Ana	25
1	Bruno	30

📌 Diferença entre .loc[] e .iloc[]:

>

Método	Baseado em...	Exemplo
`.loc[]`	**Nome** (label)	`df.loc[0, 'Nome']`
`.iloc[]`	**Posição (número)**	`df.iloc[0, 0]`

Filtros:

O que é um Filtro no Pandas?

Filtrar um DataFrame significa selecionar linhas que atendem a uma condição específica. Saber fazer filtros em DataFrames é uma das habilidades mais importantes em Pandas — e a maioria das análises de dados começa por aqui.

Por exemplo:

- Quero ver só as pessoas com idade maior que 25.
- Quero ver quem mora em "São Paulo".

```
dados3 = {
    'Nome': ['Ana', 'Bruno', 'Carla', 'Daniel'],
    'Idade': [25, 30, 22, 28],
    'Cidade': ['São Paulo', 'Rio de Janeiro', 'Curitiba', 'São Paulo']
}

df3 = pd.DataFrame(dados3)
df3
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
1	Bruno	30	Rio de Janeiro
2	Carla	22	Curitiba
3	Daniel	28	São Paulo

Como fazer filtros?

1. Filtro simples (condição única)

```
# Quero apenas as colunas que tenham idade maior que 25 anos
df3[df3['Idade'] > 25]
```

border="1" class="dataframe">

	Nome	Idade	Cidade
1	Bruno	30	Rio de Janeiro
3	Daniel	28	São Paulo

2. Filtro com Igualdade:

```
# Mostre só quem mora em São Paulo:
df3[df3['Cidade'] == 'São Paulo']
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
3	Daniel	28	São Paulo

3. Filtro com múltiplas condições (AND):

```
# Quem tem mais de 25 anos e mora em São Paulo:
df3[(df3['Idade'] > 25) & (df3['Cidade'] == 'São Paulo')]
```

border="1" class="dataframe">

	Nome	Idade	Cidade
3	Daniel	28	São Paulo

4. Filtros com múltiplas condições (OR):

```
# Quem mora em Curitiba ou São Paulo:
df3[(df3['Cidade'] == 'Curitiba') | (df3['Cidade'] == 'São Paulo')]
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
2	Carla	22	Curitiba
3	Daniel	28	São Paulo

5. Filtro com isin() (valor está em uma lista):

```
# Quem mora em Curitiba ou Rio de Janeiro:
df3[df3['Cidade'].isin(['Curitiba', 'Rio de Janeiro'])]
```

border="1" class="dataframe">

	Nome	Idade	Cidade
1	Bruno	30	Rio de Janeiro
2	Carla	22	Curitiba

6. Filtro com not(~):

```
# Quem não mora em São Paulo
df3[~df3['Cidade'].isin(['São Paulo'])]
```

border="1" class="dataframe">

	Nome	Idade	Cidade
1	Bruno	30	Rio de Janeiro
2	Carla	22	Curitiba

Dicas importantes:

Para combinar condições:

- AND: &
- OR: |
- NOT: ~

Sempre coloque parênteses nas condições:

```
(df3['Idade'] > 25) & (df3['Cidade'] == 'São Paulo')
```

```
0    False
1    False
2    False
3     True
dtype: bool
```

.isin([]) é ótimo para verificar vários valores de uma vez.

2.7. Operações com índices

O que é um Índice no Pandas?

Em um DataFrame ou Series, o índice é a "etiqueta" que identifica cada linha.

Ele funciona como o "endereço" de cada dado — como se fosse o número da linha em uma tabela do Excel

```
# Exemplo:
df3
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
1	Bruno	30	Rio de Janeiro
2	Carla	22	Curitiba
3	Daniel	28	São Paulo

Os números a esquerda (0, 1, 2 e 3) são os índices. Eles não fazem parte dos seus dados — são um "rótulo" para acessar linhas.

Uma dica importante e interresante é que você sempre pode usar ***índices numéricos* com .iloc[]** e ***índices "nomeados"* com .loc[]**.

Operações Comuns com índices:

1. Acessar um índice:

```
# Ele mostrara os indices
# Caso os indices tenham sido gerados de maneira padrão, que é quando não definimos um index
# e o próprio pandas cria um rangeIndex, ele mostrará exatamente isso no resultado.
df3.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

```
# Por exemplo, as colunas foram definidas antecipadamente, então quando eu peço para ver...
df3.columns
```

```
Index(['Nome', 'Idade', 'Cidade'], dtype='object')
```

2. Definir um índice personalizado:

```
#### Vamos definir que a coluna 'Nome' será os index
df3.set_index('Nome', inplace=True)
```

```
df3
```

border="1" class="dataframe">

Nome	Idade	Cidade
Ana	25	São Paulo
Bruno	30	Rio de Janeiro
Carla	22	Curitiba
Daniel	28	São Paulo

3. Resetar o índice

O índice volta a ser numérico (0, 1, 2) e a coluna 'Nome' volta a ser coluna normal.

```
df3.reset_index(inplace=True)
```

```
df3
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25	São Paulo
1	Bruno	30	Rio de Janeiro
2	Carla	22	Curitiba
3	Daniel	28	São Paulo

4. Renomear índices:

```
df3.index = ['A', 'B', 'C', 'D']
```

```
df3
```

border="1" class="dataframe">

	Nome	Idade	Cidade
A	Ana	25	São Paulo
B	Bruno	30	Rio de Janeiro
C	Carla	22	Curitiba
D	Daniel	28	São Paulo

5. Filtrar usando índice:

```
df3_copy = df3.set_index('Nome')
print(df3_copy.loc['Ana']) # Acessa a lista da Ana
```

```

```

```
Idade            25
Cidade    São Paulo
Name: Ana, dtype: object
```

RESUMO RÁPIDO:

>

Operação	Código	O que faz
Ver índice	`df.index`	Mostra o índice atual
Definir coluna como índice	`df.set_index('coluna')`	Transforma uma coluna no novo índice
Resetar índice	`df.reset_index()`	Volta para índice padrão (0,1,2...)
Ordenar pelo índice	`df.sort_index()`	Ordena DataFrame pelo índice
Renomear índice manualmente	`df.index = ['a', 'b', 'c']`	Define índices novos
Filtrar pelo índice	`df.loc['rótulo']`	Acessa linhas específicas usando índice

---

2.9. Tratamento de dados ausentes

O que são Dados Ausentes?

São valores faltantes, nulos ou indefinidos em um conjunto de dados.

No Pandas, eles aparecem geralmente como:

- NaN (Not a Number)
- None (Python)

Dicas importantes:

- Nunca elimine dados ausentes automaticamente — às vezes eles são importantes.
- Decida o que fazer caso a caso: remover? preencher? ignorar?
- Valores ausentes podem enviesar uma análise estatística se não forem tratados.

```python
# Exemplo
import numpy as np

dados_ausentes = {'Nome': ['Ana', 'Bruno', 'Carla', 'Daniel'],
          'Idade': [25, np.nan, 22, 28],
          'Cidade': ['São Paulo', 'Rio de Janeiro', None, 'São Paulo']}

df_ausentes = pd.DataFrame(dados_ausentes)
```

```
df_ausentes
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25.0	São Paulo
1	Bruno	NaN	Rio de Janeiro
2	Carla	22.0	None
3	Daniel	28.0	São Paulo

1. Identificar dados ausentes:

```
df_ausentes.isnull() # Mostra True onde há valores ausentes
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	False	False	False
1	False	True	False
2	False	False	True
3	False	False	False

```
# Contar os valores ausentes
df_ausentes.isnull().sum()
```

```
Nome      0
Idade     1
Cidade    1
dtype: int64
```

2. Filtrar linhas com dados ausentes:

```
df_ausentes[df_ausentes['Idade'].isnull()]   # Filtra linhas onde 'Idade' é NaN
```

border="1" class="dataframe">

	Nome	Idade	Cidade
1	Bruno	NaN	Rio de Janeiro

3. Remover dados ausentes

```
# Remove qualquer linha que contenha pelo menos um NaN:
df_ausentes.dropna()
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25.0	São Paulo
3	Daniel	28.0	São Paulo

```
#  Remove só se todos os valores da linha forem NaN:
df_ausentes.dropna(how='all')
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25.0	São Paulo
1	Bruno	NaN	Rio de Janeiro
2	Carla	22.0	None
3	Daniel	28.0	São Paulo

4. Preencher valores ausentes:

```
# Com valor especifico:
df_ausentes['Idade'] = df_ausentes['Idade'].fillna(0)   # Preenche NaN da coluna
'Idade' com 0
```

```
df_ausentes
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25.0	São Paulo
1	Bruno	0.0	Rio de Janeiro
2	Carla	22.0	None
3	Daniel	28.0	São Paulo

```
# Com a média, mediana ou moda:
df_ausentes['Idade'] = df_ausentes['Idade'].fillna(df_ausentes['Idade'].mean())  #
Preenche com a média
```

```
df_ausentes
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25.0	São Paulo
1	Bruno	25.0	Rio de Janeiro
2	Carla	22.0	None
3	Daniel	28.0	São Paulo

```
# Utilização em texto:
df_ausentes['Cidade'] = df_ausentes['Cidade'].fillna('Não informado')
```

```
df_ausentes
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25.0	São Paulo
1	Bruno	25.0	Rio de Janeiro
2	Carla	22.0	Não informado
3	Daniel	28.0	São Paulo

5. Substituir todos os NaN's do DataFrame

```
df_ausentes.fillna('Sem dado')  # Substitui todos os NaN do DataFrame
```

border="1" class="dataframe">

	Nome	Idade	Cidade
0	Ana	25.0	São Paulo
1	Bruno	Sem dado	Rio de Janeiro
2	Carla	22.0	Sem dado
3	Daniel	28.0	São Paulo

2.10. Groupby

O método groupby() do Pandas é uma das ferramentas mais poderosas e essenciais para análises agrupadas — muito usado em relatórios, resumos, dashboards e análise de dados reais.

O que é o .groupby()?

O .groupby() é usado para:

1. Agrupar dados de um DataFrame com base em uma ou mais colunas.
2. Fazer alguma operação de agregação (como soma, média, contagem, etc.) em cada grupo.

Ele funciona assim: Dividir ➡️ Agrupar ➡️ Agregar

Vamos ao exemplo:

```
dados3 = {'Categoria': ['A', 'B', 'A', 'B', 'A', 'C'],
          'Valor': [10, 20, 15, 25, 10, 30]}

df3 = pd.DataFrame(dados3)
df3
```

border="1" class="dataframe">

	Categoria	Valor
0	A	10
1	B	20
2	A	15
3	B	25
4	A	10
5	C	30

1. Agregar por categoria e somar valores:

```
df3.groupby('Categoria')['Valor'].sum() # O pandas irá somar os valores de cada categoria
```

```
Categoria
A    35
B    45
C    30
Name: Valor, dtype: int64
```

2. Calcular a média dos valores por categoria:

```
df3.groupby('Categoria')['Valor'].mean()
```

```
Categoria
A    11.666667
B    22.500000
C    30.000000
Name: Valor, dtype: float64
```

3. Contar em quantas entradas há em cada categoria:

```
df3.groupby('Categoria')['Valor'].count()
```

```
Categoria
A    3
B    2
C    1
Name: Valor, dtype: int64
```

4. Obter várias estatisticas ao mesmo tempo:

```
df3.groupby('Categoria')['Valor'].agg(['sum', 'mean', 'count'])
```

border="1" class="dataframe">

Categoria	sum	mean	count
A	35	11.666667	3
B	45	22.500000	2
C	30	30.000000	1

Agrupando por mais de uma coluna

```
dados4 = {'Categoria': ['A', 'A', 'B', 'B', 'C', 'C'],
          'Subcategoria': ['X', 'Y', 'X', 'Y', 'X', 'Y'],
          'Valor': [10, 15, 20, 25, 30, 35]}

df4 = pd.DataFrame(dados4)
df4
```

border="1" class="dataframe">

	Categoria	Subcategoria	Valor
0	A	X	10
1	A	Y	15
2	B	X	20
3	B	Y	25
4	C	X	30
5	C	Y	35

```
df4.groupby(['Categoria', 'Subcategoria'])['Valor'].sum()
```

```
Categoria  Subcategoria
A          X               10
           Y               15
B          X               20
           Y               25
C          X               30
           Y               35
Name: Valor, dtype: int64
```

2.11. Merge, concat e Join

Todas essas funções são usadas para combinar/juntar DataFrames. Mas cada uma tem um jeito específico de fazer isso.

1. Concat:

É simplesmente juntar um embaixo ou ao lado do outro.

Pense assim: `É como empilhar várias planilhas ou colar blocos`.

Serve para:

- Empilhar DataFrames (em cima/embaixo ou lado a lado).
- Não se importa com chaves/colunas iguais.

```
df_1 = pd.DataFrame({'A': ['A1', 'A2'], 'B': ['B1', 'B2']})
df_2 = pd.DataFrame({'A': ['A3', 'A4'], 'B': ['B3', 'B4']})
```

```
df_1
```

border="1" class="dataframe">

	A	B
0	A1	B1
1	A2	B2

```
df_2
```

border="1" class="dataframe">

	A	B
0	A3	B3
1	A4	B4

Juntar DataFrames em linhas (vertical):

```
pd.concat([df_1, df_2])
```

border="1" class="dataframe">

	A	B
0	A1	B1
1	A2	B2
0	A3	B3
1	A4	B4

Juntar DataFrames lado a lado (horizontal):

```
pd.concat([df_1, df_2], axis=1)
```

border="1" class="dataframe">

	A	B	A	B
0	A1	B1	A3	B3
1	A2	B2	A4	B4

2. merge() - Juntar como no SQL (por chave):

Serve para:

- Juntar DataFrames com base em uma coluna em comum (chave).
- Igual ao JOIN do SQL.

Pense assim: *Juntar duas tabelas onde existe uma coluna em comum, tipo ID, CPF, produto, etc.*

Dicas importantes:

- Sempre confira se as colunas-chaves estão com nomes idênticos antes de fazer merge.
- Para merges com nomes de colunas diferentes: `|pd.merge(df1, df2, left_on='coluna1', right_on='coluna2')|`
- No concat(), os dados podem ficar desalinhados se as colunas forem diferentes.

```
# Exemplo:
df_1 = pd.DataFrame({'ID': [1, 2, 3], 'Nome': ['Ana', 'Bruno', 'Carla']})
df_2 = pd.DataFrame({'ID': [1, 2, 4], 'Cidade': ['SP', 'RJ', 'MG']})
```

```
df_1
```

border="1" class="dataframe">

	ID	Nome
0	1	Ana
1	2	Bruno
2	3	Carla

```
df_2
```

border="1" class="dataframe">

	ID	Cidade
0	1	SP
1	2	RJ
2	4	MG

```
pd.merge(df_1, df_2, on='ID')
```

border="1" class="dataframe">

	ID	Nome	Cidade
0	1	Ana	SP
1	2	Bruno	RJ

✔️ Só aparecem os IDs que existem nos dois DataFrames.

Tipos de Merge:

>

Tipo de Merge	Descrição	Código
`inner` (padrão)	Só o que existe nos dois	`pd.merge(df1, df2, on='ID', how='inner')`
`left`	Tudo do DF1 + combinações do DF2	`how='left'`
`right`	Tudo do DF2 + combinações do DF1	`how='right'`
`outer`	Todos os dados dos dois DF	`how='outer'`

3. join() - Juntar usando o índice como chave:

Serve para:

- Juntar DataFrames pelos seus índices (não colunas).

```
# Exemplo:
df_1 = pd.DataFrame({'Nome': ['Ana', 'Bruno', 'Carla']}, index=[1,2,3])
df_2 = pd.DataFrame({'Idade': [25,30,22]}, index=[1,2,3])
```

```
df_1
```

border="1" class="dataframe">

	Nome
1	Ana
2	Bruno
3	Carla

```
df_2
```

border="1" class="dataframe">

	Idade
1	25
2	30
3	22

```
df_1.join(df_2)
```

border="1" class="dataframe">

	Nome	Idade
1	Ana	25
2	Bruno	30
3	Carla	22

✔️ Juntou automaticamente pelos índices iguais.

Resumo:

>

Método	Usa chave?	Usa índice?	Exemplo útil	Parecido com
`concat()`	Não (só empilha)	Não	Empilhar ou lado a lado	"Copiar e colar"
`merge()`	Sim (coluna)	Não	Juntar por coluna comum	SQL JOIN
`join()`	Não	Sim (índice)	Juntar baseado no índice	SQL JOIN por índice

---

2.12. Operações com DataFrames

Depois que você entende bem o que é um DataFrame, o próximo passo natural é aprender a fazer operações com DataFrames — algo que você usará muito no dia a dia de análise de dados.

O que são "operações com DataFrames"?

São ações que podemos realizar entre colunas, entre linhas, entre dois DataFrames ou com constantes.

Exemplos:

- Soma de colunas
- Subtração entre DataFrames
- Operações aritméticas
- Comparações
- Aplicação de funções

```
df = pd.DataFrame({
    'Produto': ['A', 'B', 'C'],
    'Preço': [10, 20, 15],
    'Quantidade': [2, 1, 3]
})
```

```
df
```

border="1" class="dataframe">

	Produto	Preço	Quantidade
0	A	10	2
1	B	20	1
2	C	15	3

Tipos de operações comuns:

1. Operações aritmeticas comuns

```
# Quero o valor total dos produtos (A quantidade multiplicado pelo preço
df['Total'] = df['Preço'] * df['Quantidade']
```

```
df
```

border="1" class="dataframe">

	Produto	Preço	Quantidade	Total
0	A	10	2	20
1	B	20	1	20
2	C	15	3	45

2. Operações Aritméticas com Constantes:

```
df['Preço_com_Desconto'] = df['Preço'] * 0.9   # 10% de desconto
```

```
df
```

border="1" class="dataframe">

	Produto	Preço	Quantidade	Total	Preço_com_Desconto
0	A	10	2	20	9.0
1	B	20	1	20	18.0
2	C	15	3	45	13.5

3. Operações entre DataFrames (Mesmo formato):

```
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})

df1 + df2
```

border="1" class="dataframe">

	A	B
0	6	10
1	8	12

4. Operações lógicas (Comparações):

```
df['Caro'] = df['Preço'] > 15 # Se 'Preço' for maior que 15, significa que ele é caro, então retorna 'True'
```

```
df
```

border="1" class="dataframe">

	Produto	Preço	Quantidade	Total	Preço_com_Desconto	Caro
0	A	10	2	20	9.0	False
1	B	20	1	20	18.0	True
2	C	15	3	45	13.5	False

5. Estatísticas rápidas:

```
df['Preço'].sum()     # Soma total
```

```
np.int64(45)
```

```
df['Preço'].mean()    # Média
```

```
np.float64(15.0)
```

```
df['Preço'].min()     # Mínimo
```

```
np.int64(10)
```

```
df['Preço'].max()     # Máximo
```

```
np.int64(20)
```

```
df['Preço'].std()     # Desvio padrão
```

```
np.float64(5.0)
```

6. Aplicando funções com `.apply()`

```
df['Preço_formatado'] = df['Preço'].apply(lambda x: f'R${x:.2f}')
```

```
df
```

border="1" class="dataframe">

	Produto	Preço	Quantidade	Total	Preço_com_Desconto	Caro	Preço_formatado
0	A	10	2	20	9.0	False	R$10.00
1	B	20	1	20	18.0	True	R$20.00
2	C	15	3	45	13.5	False	R$15.00

Outras coisas:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Produto             3 non-null      object 
 1   Preço               3 non-null      int64  
 2   Quantidade          3 non-null      int64  
 3   Total               3 non-null      int64  
 4   Preço_com_Desconto  3 non-null      float64
 5   Caro                3 non-null      bool   
dtypes: bool(1), float64(1), int64(3), object(1)
memory usage: 255.0+ bytes
```

```
df.memory_usage()
```

```
Index                132
Produto               24
Preço                 24
Quantidade            24
Total                 24
Preço_com_Desconto    24
Caro                   3
dtype: int64
```

```
# Gerar um novo dataframe copiando algum já existente
df_copiar = df.copy()
```

```
df_copiar
```

border="1" class="dataframe">

	Produto	Preço	Quantidade	Total	Preço_com_Desconto	Caro
0	A	10	2	20	9.0	False
1	B	20	1	20	18.0	True
2	C	15	3	45	13.5	False

2.13. Séries Temporais no Pandas

O que são Séries Temporais?

Série temporal é um conjunto de dados onde o fator tempo (data/hora) é essencial.

Exemplo de séries temporais:

- Preços de ações ao longo dos dias.
- Vendas mensais de uma empresa.
- Temperatura diária de uma cidade.
- Frequência de acesso a um site por hora.

Dicas Importantes:

- Trabalhe sempre com datas no tipo datetime64:
  `pd.to_datetime(df['coluna_de_data'])`
- Use resample() para agrupar dados temporais (muito usado em dashboards).
- Para análise de tendências e sazonalidade, séries temporais são essenciais!

```
# Exemplo:
datas = pd.date_range(start='2024-01-01', periods=5, freq='D')
vendas = [100, 150, 200, 130, 170]
```

```
df = pd.DataFrame({'Data': datas, 'Vendas': vendas})
```

```
df
```

border="1" class="dataframe">

	Data	Vendas
0	2024-01-01	100
1	2024-01-02	150
2	2024-01-03	200
3	2024-01-04	130
4	2024-01-05	170

1. Transformando coluna em índice temporal:

```
df.set_index('Data', inplace=True) # Agora o índice é temporal – o Pandas reconhece a coluna "Data" como um objeto de tempo.
```

```
df.index # O índice é do tipo 'DatetimeIndex
```

```
DatetimeIndex(['2024-01-01', '2024-01-02', '2024-01-03', '2024-01-04',
               '2024-01-05'],
              dtype='datetime64[ns]', name='Data', freq=None)
```

```
df
```

border="1" class="dataframe">

	Vendas
Data	
2024-01-01	100
2024-01-02	150
2024-01-03	200
2024-01-04	130
2024-01-05	170

2. Selecionando por datas:

```
# Com o index sendo um dataframe, também temos a capacidade de fazer o seguinte...
# Quero puxar apenas quando o dia for apenas 04:
df[df.index.day == 4]

# Mas também serve para se quisesse apenas o mês, ou o ano, enfim, para diversas coisas envolvendo períodos de tempo
```

border="1" class="dataframe">

Data	Vendas
2024-01-04	130

```
df.loc['2024-01-03']
```

```
Vendas    200
Name: 2024-01-03 00:00:00, dtype: int64
```

3. Filtro por intervalo de datas:

```
df.loc['2024-01-02':'2024-01-04']
```

border="1" class="dataframe">

Data	Vendas
2024-01-02	150
2024-01-03	200
2024-01-04	130

4. Resampling (Reamostragem)

Reagrupar dados por período — exemplo: somar vendas por mês:

```
df.resample('D').sum()    # 'D' = diário
df.resample('ME').sum()    # 'M' = mensal
```

border="1" class="dataframe">

Data	Vendas
2024-01-31	750

5. Extraindo partes da data:

```
df['Ano'] = df.index.year
df['Mes'] = df.index.month
df['Dia'] = df.index.day
```

```
df
```

border="1" class="dataframe">

Data	Vendas	Ano	Mes	Dia
2024-01-01	100	2024	1	1
2024-01-02	150	2024	1	2
2024-01-03	200	2024	1	3
2024-01-04	130	2024	1	4
2024-01-05	170	2024	1	5

2.14. Entrada e Saída de dados

Saber como fazer entrada e saída de dados (I/O — Input/Output) no Pandas é essencial, porque quase todo trabalho real de análise começa com:

- Ler dados de um arquivo (Excel, CSV, SQL, JSON etc.)
- Manipular no Python
- Salvar os dados processados para um novo arquivo.

O Pandas consegue ler arquivos de vários formatos, como por exemplo: ***CSV (o mais comum), Excel (.xlsx), JSON, Arquivo de texto delimitado, SQL, dentre outros***.

Como são diversos os modos de se ter uma entrada de dados (e consequentemente a saída dos dados), não vai ser possível mostrar todas as maneiras, por isso **irei focar apenas em CSV**.

Foi baixado um modelo de dataset CSV no 'Kaggle' para ser utilizado como prática.

```
# Lendo o arquivo CSV
df_csv = pd.read_csv('Books.csv')
```

```
df_csv
```

border="1" class="dataframe">

	title	author	pages	genre	description	published_date	publisher	language	average_rating	ratings_count	thumbnail
0	Fictional Points of View	Peter Lamarque	252	Literary Criticism	The volume focuses on a wide range of thinkers...	1996	Cornell University Press	en	No rating	0	http://books.google.com/books/content?id=rh-om...
1	Science Fiction and Fantasy Literature	R. Reginald, Douglas Menville, Mary A. Burgess	802	Reference	Science Fiction and Fantasy Literature, A Chec...	2010-09-01	Wildside Press LLC	en	No rating	0	http://books.google.com/books/content?id=P8zW2...
2	Library of Congress Subject Headings	Library of Congress. Cataloging Policy and Sup...	1662	Subject headings, Library of Congress	No description available	2004	Unknown Publisher	en	No rating	0	http://books.google.com/books/content?id=pEhkh...
3	Library of Congress Subject Headings	Library of Congress	1512	Subject headings, Library of Congress	No description available	2007	Unknown Publisher	en	No rating	0	http://books.google.com/books/content?id=FgAjF...
4	Fictional Space in the Modernist and Post-mode...	Carl Darryl Malmgren	248	Fiction	Fictional space is the imaginal expanse of fie...	1985	Bucknell University Press	en	No rating	0	http://books.google.com/books/content?id=KXzoz...
...	...	...	...	...	...	...	...	...	...	...	...
2044	The Index Card	Helaine Olen, Harol	256	Personal Finance	Simplifies personal	2016-01-05	Portfolio	en	4.0	30000	http://books.google.com

	title	author	pages	genre	description	published_date	publisher	language	average_rating	ratings_count	thumbnail
		d Pollack			finance to ten rules that ...						/books/content?id=8z4_D...
2045	The Road to Wealth	Suze Orman	608	Personal Finance	A comprehensive guide to managing money, inves...	2001-04-01	Riverhead Books	en	4.1	50000	http://books.google.com/books/content?id=zv0oD...
2046	The Success Principles	Jack Canfield	512	Self-Help	A guide to achieving personal and financial su...	2004-12-28	HarperCollins	en	4.2	100000	http://books.google.com/books/content?id=7zL_D...
2047	The Courage to Be Rich	Suze Orman	448	Personal Finance	Combines emotional and practical advice for bu...	1999-03-01	Riverhead Books	en	4.0	40000	http://books.google.com/books/content?id=2c3_D...
2048	The Money Manual	Tonya Raple y	256	Personal Finance	A millennial-focused guide to budgeting, savin...	2023-05-16	Adams Media	en	4.2	20000	http://books.google.com/books/content?id=3z3_D...

2049 rows × 11 columns

```
df_csv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2049 entries, 0 to 2048
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   title           2049 non-null   object
 1   author          2049 non-null   object
 2   pages           2049 non-null   object
 3   genre           2049 non-null   object
 4   description     2049 non-null   object
 5   published_date  2049 non-null   object
 6   publisher       2049 non-null   object
 7   language        2049 non-null   object
 8   average_rating  2049 non-null   object
 9   ratings_count   2049 non-null   int64 
 10  thumbnail       2049 non-null   object
dtypes: int64(1), object(10)
memory usage: 176.2+ KB
```

```
# Filtrando o DataFrame para mostrar os livros publicados acima de 1990 e menor que 2000:
df_filtrado = df_csv[(df_csv['published_date'] > '1990') | (df_csv['published_date'] < '2000') ]
```

```
df_filtrado
```

border="1" class="dataframe">

	title	author	pages	genre	description	published_date	publisher	language	average_rating	ratings_count	thumbnail
0	Fictional Points of View	Peter Lamarque	252	Literary Criticism	The volume focuses on a wide range of thinkers...	1996	Cornell University Press	en	No rating	0	http://books.google.com/books/content?id=rh-om...
1	Science Fiction and Fantasy Literature	R. Reginald, Douglas Menville, Mary A. Burgess	802	Reference	Science Fiction and Fantasy Literature, A Chec...	2010-09-01	Wildside Press LLC	en	No rating	0	http://books.google.com/books/content?id=P8zW2...
2	Library of Congress Subject Headings	Library of Congress. Cataloging Policy and Sup...	1662	Subject headings, Library of Congress	No description available	2004	Unknown Publisher	en	No rating	0	http://books.google.com/books/content?id=pEhkh...
3	Library of Congress Subject Headings	Library of Congress	1512	Subject headings, Library of Congress	No description available	2007	Unknown Publisher	en	No rating	0	http://books.google.com/books/content?id=FgAjF...
4	Fictional Space in the Modernist and Post-mode...	Carl Darryl Malmgren	248	Fiction	Fictional space is the imaginal expanse of fie...	1985	Bucknell University Press	en	No rating	0	http://books.google.com/books/content?id=KXzoz...
...	...	...	...	...	...	...	...	...	...	...	...
2044	The Index Card	Helaine Olen, Harol	256	Personal Finance	Simplifies personal	2016-01-05	Portfolio	en	4.0	30000	http://books.google.com

	title	author	pages	genre	description	published_date	publisher	language	average_rating	ratings_count	thumbnail
		d Pollack			finance to ten rules that ...						/books/content?id=8z4_D...
2045	The Road to Wealth	Suze Orman	608	Personal Finance	A comprehensive guide to managing money, inves...	2001-04-01	Riverhead Books	en	4.1	50000	http://books.google.com/books/content?id=zv0oD...
2046	The Success Principles	Jack Canfield	512	Self-Help	A guide to achieving personal and financial su...	2004-12-28	HarperCollins	en	4.2	100000	http://books.google.com/books/content?id=7zL_D...
2047	The Courage to Be Rich	Suze Orman	448	Personal Finance	Combines emotional and practical advice for bu...	1999-03-01	Riverhead Books	en	4.0	40000	http://books.google.com/books/content?id=2c3_D...
2048	The Money Manual	Tonya Raplely	256	Personal Finance	A millennial-focused guide to budgeting, savin...	2023-05-16	Adams Media	en	4.2	20000	http://books.google.com/books/content?id=3z3_D...

2049 rows × 11 columns

```
# Criando um novo arquivo CSV:
# Vamos pegar a informação anterior gerada com o filtro e colocara para gerar um novo arquivo
```

```
df_filtrado.to_csv("book_filtrado.csv")
```

```
pd.read_csv('book_filtrado.csv') # Gerado e visualizado com sucesso
```

border="1" class="dataframe">

	Unnamed: 0	title	author	pages	genre	description	published_date	publisher	language	average_rating	ratings_count	thumbnail
0	0	Fictional Points of View	Peter Lamarque	252	Literary Criticism	The volume focuses on a wide range of thinkers...	1996	Cornell University Press	en	No rating	0	http://books.google.com/books/content?id=rh-om...
1	1	Science Fiction and Fantasy Literature	R. Reginald, Douglas Menville, Mary A. Burgess	802	Reference	Science Fiction and Fantasy Literature, A Chec...	2010-09-01	Wildside Press LLC	en	No rating	0	http://books.google.com/books/content?id=P8zW2...
2	2	Library of Congress Subject Headings	Library of Congress. Cataloging Policy and Sup...	1662	Subject headings, Library of Congress	No description available	2004	Unknown Publisher	en	No rating	0	http://books.google.com/books/content?id=pEhkh...
3	3	Library of Congress Subject Headings	Library of Congress	1512	Subject headings, Library of Congress	No description available	2007	Unknown Publisher	en	No rating	0	http://books.google.com/books/content?id=FgAjF...
4	4	Fictional Space in the Modernist and Post-mode...	Carl Darryl Malmgren	248	Fiction	Fictional space is the imaginal expanse of fie...	1985	Bucknell University Press	en	No rating	0	http://books.google.com/books/content?id=KXzoz...

	Unnamed: 0	title	author	pages	genre	description	published_date	publisher	language	average_rating	ratings_count	thumbnail
...	...	...	...	...	...	...	...	...	...	...	...	...
2044	2044	The Index Card	Helaine Olen, Harold Pollack	256	Personal Finance	Simplifies personal finance to ten rules that ...	2016-01-05	Portfolio	en	4.0	30000	http://books.google.com/books/content?id=8z4_D...
2045	2045	The Road to Wealth	Suze Orman	608	Personal Finance	A comprehensive guide to managing money, inves ...	2001-04-01	Riverhead Books	en	4.1	50000	http://books.google.com/books/content?id=zv0oD...
2046	2046	The Success Principles	Jack Canfield	512	Self-Help	A guide to achieving personal and financial su...	2004-12-28	HarperCollins	en	4.2	100000	http://books.google.com/books/content?id=7zL_D...
2047	2047	The Courage to Be Rich	Suze Orman	448	Personal Finance	Combines emotional and practical advice for bu...	1999-03-01	Riverhead Books	en	4.0	40000	http://books.google.com/books/content?id=2c3_D...
2048	2048	The Money Manual	Tonya Rapley	256	Personal Finance	A millennial-focused guide to budgeting, savin ...	2023-05-16	Adams Media	en	4.2	20000	http://books.google.com/books/content?id=3z3_D...

2049 rows × 12 columns

Referencias:

- Sobre Series:
  - https://pythonacademy.com.br/blog/series-no-pandas
- Sobre DataFrames:
  - https://pythonacademy.com.br/blog/dataframes-do-pandas
- https://www.alura.com.br/artigos/pandas-o-que-e-para-que-serve-como-instalar?srsltid=AfmBOoownB66j48peufL3Qc19kyDVEx4h1Id7PtWKpn9LGqSsJOU1dwZ