```
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]`

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