**Conceitos em Pandas**

Arquivo contendo as anotações sobre pandas, seus conceitos básicos e outras coisas.

Anotações baseadas em pesquisas, retiradas também de trechos das aulas do curso da Stack Academy – Data Science, e também com ajuda do ChatGPT

**Sumário**

[1. O que é o Pandas? 2](#_Toc209201107)

[2. Tipos de arquivos suportados pelo pandas 2](#_Toc209201108)

[3. Exemplo básico de um DataFrame: 4](#_Toc209201109)

[4. Métodos comuns em Pandas 4](#_Toc209201110)

[5. Alterando o dataset 6](#_Toc209201111)

# O que é o Pandas?

Pandas é uma **biblioteca do Python** feita para **análise e manipulação de dados**.  
Pensa nele como um “Excel turbinado dentro do Python”: com ele você consegue **ler, transformar, analisar e exportar dados** de maneira muito mais flexível e automatizada do que em planilhas manuais.

**Para que serve na prática?**  
No dia a dia, usamos Pandas para:

* **Ler dados** de diferentes fontes: CSV, Excel, SQL, JSON, entre outros.
* **Explorar dados**: ver as primeiras linhas, descrever estatísticas, entender o formato.
* **Transformar dados**: limpar colunas, tratar valores nulos, converter tipos.
* **Analisar dados**: agrupar, filtrar, ordenar, calcular médias/somas, gerar tabelas resumidas.
* **Exportar dados**: salvar de volta em CSV, Excel ou até mandar para um banco.

**Principais estruturas do Pandas**

* **Series**: uma coluna de dados (como uma coluna do Excel).
* **DataFrame**: uma tabela com linhas e colunas (como uma planilha inteira) que suporta diferentes tipos de dados (inteiro, float, string, etc)

# Tipos de arquivos suportados pelo pandas

O **Pandas** consegue ler e gravar dados em vários formatos. Vou te listar os **principais tipos de arquivos suportados** que realmente usamos no mercado:

**📄 Arquivos de texto/planilhas**

* **CSV** → pd.read\_csv() e df.to\_csv()
* **TSV** (tab separado) → também com read\_csv(sep="\t")
* **TXT** → idem, se for estruturado com separadores
* **Excel (.xls, .xlsx, .xlsm, .xlsb, .odf, .ods, .odt)**
  + pd.read\_excel() e df.to\_excel()
  + Suporta múltiplas abas de uma planilha.

**🗄️ Bancos de dados**

* **SQL (MySQL, PostgreSQL, SQLite, etc.)**
  + pd.read\_sql() e df.to\_sql() (precisa de conexão via SQLAlchemy ou driver).

**🌐 Estruturados em texto**

* **JSON** → pd.read\_json() e df.to\_json()
* **HTML (tabelas)** → pd.read\_html()
* **XML** → pd.read\_xml()

**📦 Outros formatos comuns em ciência de dados**

* **Parquet** → pd.read\_parquet() e df.to\_parquet()
* **Feather** → pd.read\_feather() e df.to\_feather()
* **ORC** → pd.read\_orc() e df.to\_orc()
* **Stata (.dta)** → pd.read\_stata() e df.to\_stata()
* **SPSS (.sav)** → pd.read\_spss()
* **HDF5 (.h5)** → pd.read\_hdf() e df.to\_hdf()
* **Pickle (.pkl)** → pd.read\_pickle() e df.to\_pickle()

👉 Ou seja: **se é um formato tabular ou semi-estruturado, o Pandas provavelmente consegue lidar**.

💡 **No dia a dia**:

* CSV é o mais comum (troca de dados entre sistemas).
* Excel é onipresente em empresas (relatórios, planilhas financeiras, etc).
* SQL é essencial (quase sempre os dados “moram” em um banco).
* Parquet está se tornando padrão em ambientes de Big Data (porque é compacto e rápido).

# Exemplo básico de um DataFrame:



# Métodos comuns em Pandas

O Pandas tem centenas de métodos, mas no **dia a dia de trabalho** usamos sempre um conjunto pequeno que resolve 80% dos casos. Abaixo vou listar alguns exemplos que são mais comumente usadas no dia a dia e que é sempre bom ter ciência de suas utilizações.

Vale também observar que, **df**, que se encontra no inicio do comando será o nome do dataframe que você estiver utilizando.

**Exploração de dados**

* **df.head(n)** → mostra as primeiras n linhas (por padrão o valor é 5).
* **df.tail(n)** → mostra as últimas n linhas (por padrão o valor é 5).
* **df.info()** → mostra tipos de dados, colunas e memória.
* **df.describe()** → estatísticas descritivas (média, min, max, std).
* **df.shape** → retorna (linhas, colunas).
* **df**.**columns** → lista os nomes das colunas.
* **df.dtypes** → mostra os tipos de cada coluna.

**Seleção e filtragem**

* **df["coluna"]** → seleciona uma coluna.
* **df[["col1","col2"]]** → seleciona várias colunas.
* **df.loc[linha, coluna]** → seleção por rótulo.
* **df.iloc[linha, coluna]** → seleção por posição.
* **df[df["coluna"] > 10]** → filtragem com condição.
* **df.query("coluna > 10 and outra\_coluna == 'X'")** → filtragem estilo SQL.

**Transformação**

* **df.sort\_values(by="coluna")** → ordena linhas.
* **df.rename(columns={"antigo": "novo"})** → renomeia colunas.
* **df.drop(columns=["coluna"])** → remove coluna(s).
* **df.dropna()** → remove valores nulos.
* **df.fillna(valor)** → preenche nulos.
* **df.astype(tipo)** → converte tipo de coluna.
* **df.apply(func)** → aplica função a cada linha ou coluna.
* **df["coluna"].map(func)** → aplica função em uma série.
* **df.assign(nova=df["coluna"] \* 2)** → cria colunas derivadas.

**Agregação e estatísticas**

* **df["coluna"].mean()** → média.
* **df["coluna"].sum()** → soma.
* **df["coluna"].value\_counts()** → contagem de valores únicos.
* **df.groupby("coluna").mean()** → média por grupo.
* **df.groupby(["col1","col2"]).agg({"col3": "sum"})** → agregações mais complexas.
* **df.pivot\_table(values="col", index="col1", columns="col2", aggfunc="mean")** → tabela dinâmica.

**Junções e concatenações**

* **pd.concat([df1, df2])** → concatena DataFrames.
* **df.merge(df2, on="coluna")** → faz join (inner join padrão).
* **df.merge(df2, how="left", on="coluna")** → left join.

**Entrada e saída**

* pd.**read\_csv**("arquivo.csv") / df.**to\_csv**("arquivo.csv")
* pd.**read\_excel**("arquivo.xlsx") / df.**to\_excel**("arquivo.xlsx")
* pd.**read\_sql**(query, con) / df.**to\_sql**("tabela", con)

# Alterando o dataset

Existem vários jeitos, mas dá pra organizar em **6 grupos principais**:

**1. Alterar colunas**

* **Renomear**: df.rename(columns={"idade": "Idade"}, inplace=True)
* **Criar coluna nova**: df["idade\_mais10"] = df["idade"] + 10
* **Alterar valores com função**: df["nome"] = df["nome"].str.upper()

**2. Alterar linhas**

* **Remover linhas pelo índice**: df.drop(index=[0, 1], inplace=True)
* **Filtrar e manter só o que quero**: df = df[df["idade"] > 18]

**3. Alterar valores específicos**

* **Com .loc**: df.loc[0, "cidade"] = "São Paulo" # altera célula
* **Com condição**: df.loc[df["cidade"] == "SP", "cidade"] = "São Paulo"

**4. Tratar valores nulos**

* **Remover nulos**: df.dropna(inplace=True)
* **Preencher nulos**: df.fillna("Desconhecido", inplace=True)

**5. Alterar tipos de dados**

* **Converter tipo de coluna**: df["idade"] = df["idade"].astype(int)
* **Datas**: df["data\_nascimento"] = pd.to\_datetime(df["data\_nascimento"])

**6. Reestruturar o dataset**

* **Ordenar**: df.sort\_values(by="idade", ascending=False, inplace=True)
* **Resetar índices**: df.reset\_index(drop=True, inplace=True)
* **Adicionar linhas (append/concat)**:   
  novo = pd.DataFrame({"nome": ["João"], "idade": [25], "cidade": ["BH"]})  
  df = pd.concat([df, novo], ignore\_index=True)

**Resumo**:

* Alterar dataset = mexer em **colunas, linhas, valores, nulos, tipos ou estrutura**.
* A cada modificação, você pode usar inplace=True (altera direto) ou reatribuir (df = ...).

# Percorrendo linhas de um DataFrame