

# Referência Rápida de pandas

DataFrames, seleção, agregação, mesclagem e mais

## DataFrames

### Criando DataFrames

```
import pandas as pd
df = pd.DataFrame({
    "name": ["Alice", "Bob", "Carol"],
    "age": [25, 30, 35],
    "score": [88, 92, 79]
})
```

### Inspeção

<b>df.head(n)</b>	Primeiras n linhas (padrão 5)
<b>df.tail(n)</b>	Últimas n linhas
<b>df.shape</b>	Tupla de (linhas, colunas)
<b>df.dtypes</b>	Tipo de dado de cada coluna
<b>df.info()</b>	Tipos de coluna, contagens não nulas
<b>df.describe()</b>	Estatísticas para colunas numéricas
<b>df.columns</b>	Nomes das colunas como Index
<b>df.index</b>	Rótulos das linhas

### Leitura de Dados

#### Leitores Comuns

```
df = pd.read_csv("data.csv")
df = pd.read_excel("data.xlsx")
df = pd.read_json("data.json")
df = pd.read_sql(query, connection)
```

#### Escrita de Dados

```
df.to_csv("out.csv", index=False)
df.to_excel("out.xlsx", index=False)
df.to_json("out.json", orient="records")
```

### Opções de Leitura

<b>sep=";"</b>	Delimitador personalizado
<b>header=None</b>	Sem linha de cabeçalho no arquivo
<b>usecols=[0, 2]</b>	Ler apenas colunas específicas
<b>nrows=100</b>	Ler as primeiras 100 linhas
<b>na_values=["N/A"]</b>	Tratar como NaN

### Seleção

#### Colunas

```
df["name"] # single column (Series)
df[["name", "age"]] # multiple columns (DataFrame)
df.name # attribute access (simple names)
```

#### Linhas com loc / iloc

```
df.loc[0] # row by label
df.loc[0:2, "name"] # rows 0-2, column "name"
df.iloc[0] # row by position
df.iloc[0:2, 0:2] # first 2 rows, 2 cols
```

#### loc vs iloc

<b>df.loc[row, col]</b>	Selecionar por **rótulo** (fim inclusivo)
<b>df.iloc[row, col]</b>	Selecionar por **posição** (fim exclusivo)
<b>df.at[row, col]</b>	Acesso escalar rápido por rótulo
<b>df.iat[row, col]</b>	Acesso escalar rápido por posição

### Filtragem

#### Filtragem Booleana

```
df[df["age"] > 25]
df[df["name"].str.contains("li")]
df[(df["age"] > 25) & (df["score"] > 80)]
df[df["name"].isin(["Alice", "Bob"])]
```

### Tratamento de Dados Ausentes

```
df.isna().sum() # NaN count per column
df.dropna() # drop rows with any NaN
df.fillna(0) # fill NaN with 0
df["col"].fillna(df["col"].mean())
```

### Ordenação

```
df.sort_values("age") # ascending
df.sort_values("age", ascending=False)
df.sort_values(["age", "score"]) # multi
```

### Agregação

#### Agregações Comuns

<b>df["col"].sum()</b>	Soma da coluna
<b>df["col"].mean()</b>	Média
<b>df["col"].median()</b>	Mediana
<b>df["col"].std()</b>	Desvio padrão
<b>df["col"].min() / .max()</b>	Mínimo / máximo
<b>df["col"].count()</b>	Contagem de não nulos
<b>df["col"].nunique()</b>	Número de valores únicos
<b>df["col"].value_counts()</b>	Frequência de cada valor

#### Múltiplas Agregações

```
df.agg({"age": "mean", "score": ["min", "max"]})
df.describe() # summary stats for all numeric
```

### GroupBy

#### Agrupamento Básico

```
df.groupby("dept")["salary"].mean()
df.groupby("dept").agg(
    avg_sal=("salary", "mean"),
    count=("salary", "count")
)
```

#### Múltiplos Grupos

```
df.groupby(["dept", "year"])["sales"].sum()
df.groupby("dept").size() # rows per group
```

#### Transform e Apply

```
df["z_score"] = df.groupby("dept")["salary"] \
    .transform(lambda x: (x - x.mean()) / x.std())
df.groupby("dept").apply(lambda g: g.nlargest(3, "salary"))
```

### Mesclagem

#### Merge (Join estilo SQL)

```
pd.merge(df1, df2, on="id") # inner
pd.merge(df1, df2, on="id", how="left")
pd.merge(df1, df2, left_on="uid",
    right_on="user_id")
```

#### Tipos de Join

<b>how="inner"</b>	Manter apenas linhas correspondentes (padrão)
<b>how="left"</b>	Manter todas as linhas esquerdas, NaN para sem correspondência
<b>how="right"</b>	Manter todas as linhas direitas
<b>how="outer"</b>	Manter todas as linhas de ambos os lados

#### Concatenação

```
pd.concat([df1, df2]) # stack rows
pd.concat([df1, df2], axis=1) # side by side
pd.concat([df1, df2], ignore_index=True)
```

### Tabelas Dinâmicas

#### Tabela Dinâmica

```
df.pivot_table(
    values="sales", index="region",
    columns="quarter", aggfunc="sum"
)
```

#### Reformatação

```
df.melt(id_vars=["name"],
    value_vars=["q1", "q2"],
    var_name="quarter", value_name="sales")
```

#### Tabulação Cruzada

```
pd.crosstab(df["dept"], df["gender"])
pd.crosstab(df["dept"], df["gender"],
    normalize="index") # row percentages
```

### Série Temporal

#### Opções Básicas de DateTime

```
df["date"] = pd.to_datetime(df["date"])
df["year"] = df["date"].dt.year
df["month"] = df["date"].dt.month
df["weekday"] = df["date"].dt.day_name()
```

#### Intervalos de Data e Reamostragem

```
pd.date_range("2025-01-01", periods=12, freq="ME")
df.set_index("date").resample("ME")["sales"].sum()
```

#### Atributos do Accessor

<b>.dt.year / .dt.month / .dt.day</b>	Extrair componentes de data
<b>.dt.hour / .dt.minute</b>	Extrair componentes de hora
<b>.dt.day_name()</b>	Nome do dia da semana (Segunda, etc.)
<b>.dt.days_in_month</b>	Dias naquele mês

### Padrões Comuns

#### Renomear Colunas

```
df.rename(columns={"old": "new"})
df.columns = ["a", "b", "c"] # replace all
```

#### Adicionar / Modificar Colunas

```
df["total"] = df["q1"] + df["q2"]
df["grade"] = df["score"].apply(
    lambda x: "A" if x >= 90 else "B"
)
```

#### Remover Colunas / Linhas

```
df.drop(columns=["temp"])
df.drop_duplicates(subset=["name"])
df.reset_index(drop=True)
```

#### Operações com Strings

```
df["name"].str.lower()
df["name"].str.contains("ali", case=False)
df["name"].str.split(" ").str[0] # first name
```