

# **Estudo de Python e dados**

Sergio Pedro Rodrigues Oliveira

05 August 2025

# SUMÁRIO

<b>1</b>	<b>Objetivo</b>	<b>1</b>
<b>2</b>	<b>Básico sobre o DataFrame do Pandas</b>	<b>1</b>
2.1	Introdução . . . . .	1
2.2	Carregando seu primeiro conjunto de dados . . . . .	2
2.3	Observando colunas, linhas e células . . . . .	7
2.3.1	Obtendo subconjuntos de colunas . . . . .	7
	Obtendo subconjuntos de colunas pelo nome . . . . .	7
	Obter subconjuntos de colunas pela posição dos índices não funciona mais no Pandas v0.20 . . . . .	9
2.3.2	Obtendo subconjuntos de linhas . . . . .	10
	Obtendo subconjuntos de linhas pelo rótulo dos índices: <code>loc</code> . . . . .	10
	Obtendo subconjuntos de linhas pelo número das linhas: <code>iloc</code> . . . . .	14
	Obtenção de subconjuntos de linhas com <code>ix</code> não funciona mais no Pandas v0.20 . . . . .	14
2.3.3	Combinando tudo . . . . .	14
2.4	Cálculos agrupados e agregados . . . . .	15
2.5	Plotagem básica . . . . .	15
<b>3</b>	<b>Estrutura de dados do Pandas</b>	<b>16</b>
<b>4</b>	<b>Introdução à plotagem</b>	<b>16</b>

## LISTA DE FIGURAS

## LISTA DE TABELAS

1	Informações do método <code>info()</code> do Pandas . . . . .	5
2	Tipos do Pandas versus tipos de Python . . . . .	6
3	Diferentes métodos para indexação de linhas (ou de colunas). . . . .	10

# 1 Objetivo

O objetivo deste estudo é explorar e documentar as funcionalidades essenciais das principais bibliotecas científicas do Python, como NumPy, Pandas e outras, através de exemplos práticos e casos de uso selecionados. Pretende-se consolidar o conhecimento sobre a manipulação, análise e visualização de dados, servindo como um guia de referência pessoal para futuros projetos de programação científica.

## 2 Básico sobre o DataFrame do Pandas

### 2.1 Introdução

O Pandas é uma biblioteca Python de código aberto para análise de dados. Ele dá a Python a capacidade de trabalhar com dados do tipo planilha, permitindo **carregar**, **manipular**, **alinhar** e **combinar dados** rapidamente, entre outras funções.

Para proporcionar esses recursos mais sofisticados ao Python, o Pandas introduz dois novos tipos de dados: **Series** e **DataFrame**.

- **DataFrame**

Representa os dados de planilhas ou retangulares completos.

- **Series**

Corresponde a única coluna do **DataFrame**.

- Também podemos pensar em um **DataFrame** do Pandas como um **dicionário** ou uma coleção de objetos **Series**.

Por que você deveria usar uma linguagem de programação como Python e uma ferramenta como o Pandas para trabalhar com dados? Tudo se reduz à automação e à reprodutibilidade.

Objetivos do capítulo:

1. Carga de um arquivo de dados simples e delimitado.
2. Como contar quantas linhas e colunas foram carregadas.
3. Como delimitar quais tipos de dados foram carregados.
4. Observação de diferentes porções de dados criando subconjuntos de linhas e colunas.

## 2.2 Carregando seu primeiro conjunto de dados

Dado um conjunto de dados inicialmente o carregamos e começamos a observar sua estrutura e conteúdo.

O modo mais simples de observar um conjunto de dados é analisar e criar subconjuntos de linhas e colunas específicas. Podemos ver quais tipos de informação estão armazenadas em cada coluna, e começar a procurar padrões por meio de estatísticas descritivas agregadas.

Como o **Pandas** não faz parte da biblioteca-padrão de Python, devemos dizer antes ao Python que carregue a biblioteca (`import`):

```
import pandas as pd
```

Quando trabalhamos com funções **Pandas**, usar o alias `pd` para `pandas` é uma prática comum.

Com a biblioteca carregada, podemos usar a função `read_csv` para carregar um arquivo de dados **CSV**. Para acessar a função `read_csv` do Pandas, usamos a notação de ponto.

```
# Por padrão, a função read_csv lerá um arquivo separado por vírgula;
# Nosso dados Gapminder estão separados por tabulações;
# Podemos usar o parâmetro sep a representar uma tabulação com \t
import pandas as pd # Importa a biblioteca pandas como 'pd'.

# --- Carregamento e Inspeção Inicial ---
df = pd.read_csv('./Data/Cap_01/gapminder.tsv', sep='\t')
# Carrega o arquivo TSV em um DataFrame, usando tabulação como separador.

# Usamos o método head para que Python nos mostre as 5 primeiras linhas
print(df.head())
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106

- Função `type()`:

Podemos verificar se estamos trabalhando com um **DataFrame** do Pandas usando a função embutida `type` (isto é, se ele vem diretamente de Python, e não de algum pacote, como o Pandas).

A função `type()` é conveniente quando começamos a trabalhar com vários tipos diferentes de objetos Python e precisamos saber em qual objeto estamos trabalhando no momento.

```
print(type(df))
```

```
<class 'pandas.core.frame.DataFrame'>
```

- Atributo `shape`:

No momento, o conjunto de dados que carregamos esta salvo como um objeto **DataFrame** do **Pandas**, e é relativamente pequeno.

Todo objeto **DataFrame** tem um atributo `shape` que nos dará o número de linhas e de colunas desse objeto.

O atributo `shape` devolve uma tupla<sup>1</sup> na qual o primeiro valor é o número de linhas e o segundo é a quantidade de colunas.

Com base nesse resultado anterior, podemos ver que nosso conjunto de dados Gapminder tem 1704 linhas e 6 colunas.

Como `shape` é um atributo de **DataFrame**, e não uma função ou um método, não há parênteses após o ponto. Se você cometer o erro de colocar parênteses depois do atributo `shape`, um erro será devolvido.

```
# Obtém o número de linhas e colunas  
print(df.shape)
```

```
(1704, 6)
```

---

<sup>1</sup>Uma tupla é semelhante a uma `list`, pois ambas podem armazenar informações heterogêneas. A principal diferença é que o conteúdo de uma tupla é “imutável”, o que significa que ela não pode ser alterada. As tuplas também são criadas com parênteses, `()`.

- Atributo `columns`:

Em geral, quando observamos um conjunto de dados pela primeira vez, queremos saber quantas linhas e colunas há (acabamos de fazer isso).

Para ter uma noção de quais informações ele contém, devemos observar as colunas.

Os nomes das colunas, assim como `shape`, são especificados usando o atributo `columns` do objeto `dataframe`.

```
# Obtém os nomes das colunas
print(df.columns)
```

```
Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='object')
```

- Atributo `dtypes`:

O objeto `DataFrame` do **Pandas** é semelhante a objetos do tipo `DataFrame` que se encontra em outras linguagens (por exemplo, Julia e R).

Toda coluna (**Series**) deve ser do mesmo tipo, enquanto cada linha pode conter tipos variados.

Em nosso exemplo atual, podemos esperar que a coluna `country` só contenha strings e que `year` contenha inteiros. No entanto, é melhor garantir que isso seja verdade usando o atributo `dtypes` ou o método `info()`.

O atributo `dtypes` de um `DataFrame` **Pandas** retorna uma **Series** que descreve o tipo de dado de cada coluna do `DataFrame`. Ele é útil para inspecionar os tipos de dados inferidos ou atribuídos às suas colunas, o que é crucial para operações corretas e eficientes.

```
# Obtém o dtype de cada coluna
print(df.dtypes)
```

```
country      object
continent    object
year          int64
lifeExp      float64
pop           int64
gdpPercap    float64
dtype: object
```



- Método `info()`:

O método `info()` de um **DataFrame Pandas** é uma ferramenta essencial para obter um resumo conciso e detalhado do seu **DataFrame**. Ele imprime um resumo conciso do **DataFrame**, incluindo:

Table 1: Informações do método `info()` do Pandas

Informação	Descrição
Tipo de índice	Informações sobre o índice (por exemplo, <code>RangeIndex</code> ).
Número de entradas (linhas)	Quantas linhas seu <b>DataFrame</b> possui.
Número de colunas	Quantas colunas seu <b>DataFrame</b> tem.
Contagem de valores não nulos por coluna	Para cada coluna, informa quantos valores não são nulos.
Dtype (tipo de dado) de cada coluna	Isso é crucial para identificar dados faltantes. Semelhante ao atributo <code>dtype</code> , mas apresentado de forma mais organizada.
Uso de memória	A quantidade de memória que o <b>DataFrame</b> está utilizando.

```
# Obtém mais informações sobre nossos dados
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  -
0   country    1704 non-null   object
1   continent  1704 non-null   object
2   year       1704 non-null   int64
3   lifeExp    1704 non-null   float64
4   pop        1704 non-null   int64
5   gdpPercap  1704 non-null   float64
dtypes: float64(2), int64(2), object(2)
memory usage: 80.0+ KB
None
```

Table 2: Tipos do Pandas versus tipos de Python

Tipo do Pandas	Tipo de Python	Discrição
object	string	Cadeia de caracteres, usado para representar texto.
int64	int	Números inteiros.
float64	float	Números com decimais.
datetime64	datetime	<b>datetime</b> trata-se de uma biblioteca-padrão de Python (ou seja, não é carregado por padrão e deve ser importado). Representa pontos específicos no tempo.

## 2.3 Observando colunas, linhas e células

Agora que somos capazes de carregar um arquivo de dados simples, queremos inspecionar o seu conteúdo. Podemos exibir o conteúdo do dataframe com `print`, mas com os dados de hoje em dia, com frequência, haverá células demais para ser possível compreender todas as informações exibidas. Em vez disso, a melhor maneira de observar nossos dados é inspecioná-los por partes, observando vários subconjuntos dos dados.

Já vimos que podemos usar o método `head()` de um dataframe para observar as cinco primeiras linhas de nossos dados. Isso é conveniente para ver se os dados foram carregados de modo apropriado e para ter uma noção de cada uma das colunas, seus nomes e o conteúdo. Às vezes, porém, talvez queiramos ver somente linhas, colunas e valores específicos de nossos dados.

### 2.3.1 Obtendo subconjuntos de colunas

Se quiser analisar várias colunas, especifique-as com base nos nomes, nas posições ou em intervalos.

#### Obtendo subconjuntos de colunas pelo nome

Se quiser observar apenas uma coluna específica de nossos dados, podemos acessá-la usando colchetes.

```
# Obtém somente a coluna country e a salva em sua própria variável
country_df = df['country']

# Mostra as 5 primeiras observações
print(country_df.head())
```

```
0    Afghanistan
1    Afghanistan
2    Afghanistan
3    Afghanistan
4    Afghanistan
Name: country, dtype: object
```

```
# Mostra as 5 últimas observações
print(country_df.tail())
```

```
1699    Zimbabwe
1700    Zimbabwe
1701    Zimbabwe
1702    Zimbabwe
1703    Zimbabwe
Name: country, dtype: object
```

Para especificar várias colunas pelo nome, devemos passar uma `list` Python entre os colchetes. Isso pode parecer um pouco estranho, pois haverá dois conjuntos de colchetes.

```
# Observando country, continent e year
subset = df[['country', 'continent', 'year']]
print(subset.head())
```

	country	continent	year
0	Afghanistan	Asia	1952
1	Afghanistan	Asia	1957
2	Afghanistan	Asia	1962
3	Afghanistan	Asia	1967
4	Afghanistan	Asia	1972

```
# Mostra as 5 últimas observações
print(subset.tail())
```

	country	continent	year
1699	Zimbabwe	Africa	1987
1700	Zimbabwe	Africa	1992
1701	Zimbabwe	Africa	1997
1702	Zimbabwe	Africa	2002
1703	Zimbabwe	Africa	2007

Mais uma vez, é possível optar por exibir todo o dataframe subset usando `print`.

## **Obter subconjuntos de colunas pela posição dos índices não funciona mais no Pandas v0.20**

Ocasionalmente, talvez você queira obter uma coluna em particular com base em sua posição, e não em seu nome. Por exemplo, pode querer a primeira (“country”) e a terceira (“year”) colunas, ou somente a última (“gdpPercap”).

No **pandas** v0.20 não é mais possível passar uma lista de inteiros entre colchetes para obter subconjuntos de colunas. Por exemplo, `df[[1]]`, `df[[0,-1]]` e `df[list(range(5))]` não funcionam mais. Há outras formas de obter subconjuntos de colunas, mas não baseadas na técnica usada para obter subconjuntos de linhas.

### 2.3.2 Obtendo subconjuntos de linhas

Podemos obter subconjuntos de linhas de várias maneiras, pelos nomes ou pelos índices das linhas. A Table 3 apresenta uma visão geral rápida dos diversos métodos.

Table 3: Diferentes métodos para indexação de linhas (ou de colunas).

Método para obtenção de subconjuntos	Descrição
<code>loc</code>	Subconjunto baseado no rótulo do índice ( <b>nome</b> da linha).
<code>iloc</code>	Subconjunto baseada no índice da linha ( <b>número</b> da linha).
<code>ix</code> (não funciona mais no Pandas v0.20)	Subconjunto baseado no rótulo do índice ou no índice da linha.

#### Obtendo subconjuntos de linhas pelo rótulo dos índices: `loc`

Vamos observar uma parte de nossos dados Gapminder.

```
print(df.head())
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106

A esquerda do DataFrame exibido, vemos o que parece ser os números das linhas. Essa lista de valores sem coluna é o rótulo dos índices do dataframe.

Pense no rótulo dos índices como um nome de coluna, mas para linhas em vez de colunas. Por padrão, o Pandas preencherá os rótulos dos índices com os números das linhas (observe que a contagem começa em 0).

Um exemplo comum em que os rótulos dos índices das linhas não são iguais ao número das linhas ocorre quando trabalhamos com dados de séries temporais. Nesse caso, o rótulo dos índices será algum tipo de timestamp. Por exemplo, manteremos os valores default, que são os números das linhas.

Podemos usar o atributo `loc` do dataframe para obter subconjuntos de linhas com base no rótulo dos índices.

```
# Obtém a primeira linha
# Python começa a contar de 0
print(df.loc[0])
```

```
country      Afghanistan
continent      Asia
year          1952
lifeExp       28.801
pop           8425333
gdpPercap     779.445314
Name: 0, dtype: object
```

```
# Obtém a centesima linha
# Python começa a contar de 0
print(df.loc[99])
```

```
country      Bangladesh
continent      Asia
year          1967
lifeExp       43.453
pop           62821884
gdpPercap     721.186086
Name: 99, dtype: object
```

**Para obter a última linha**, uma alternativa seria passar -1 para `loc`, porém acarretaria num erro. Ao passar -1 para `loc` causará um erro, pois o código procurará a linha cujo rótulo de índice (nesse caso, número da linha) seja “-1”, e esse valor não existe no nosso exemplo.

Em vez disso, podemos usar um pouco de Python para calcular o número de linhas e passar esse valor para `loc`.

```
# Obtém a última linha (corretamente)
# Usar o primeiro valor dado por shape para obter o número de linhas
number_of_rows = df.shape[0]

# Subtrai 1 do valor, pois queremos obter o número do último índice
last_row_index = number_of_rows - 1

# Obtem agora o subconjunto usando o índice da última linha
print(df.loc[last_row_index])
```

```
country      Zimbabwe
continent     Africa
year          2007
lifeExp       43.487
pop          12311143
gdpPercap    469.709298
Name: 1703, dtype: object
```

Como alternativa, podemos usar o método `tail` para devolver a última linha, em vez de usar o default de 5.

```
# Método tail, devolvendo a última linha
print(df.tail(n=1))
```

```
      country continent  year  lifeExp      pop  gdpPercap
1703  Zimbabwe     Africa  2007   43.487 12311143  469.709298
```

Observe que, quando usamos `tail()` e `loc`, os resultados foram exibidos de modo diferentes. Vamos observar o tipo devolvido quando usamos esses métodos.



```
subset_loc = df.loc[0]
subset_head = df.head(n=1)

# type usando loc para uma linha
print("type usando loc para uma linha:")
print(type(subset_loc))

# type usando head para uma linha
print("\ntype usando head para uma linha:")
print(type(subset_head))
```

```
type usando loc para uma linha:
<class 'pandas.core.series.Series'>
```

```
type usando head para uma linha:
<class 'pandas.core.frame.DataFrame'>
```

No início desse capítulo, mencionamos que o Pandas introduziu dois novos tipos de dados em Python. Conforme o método que usamos e a quantidade de linhas retornada, o Pandas devolverá um objeto diferente. O modo como o objeto é exibido pela tela pode ser um indicador do tipo, mas sempre é melhor usar a função `type()` por garantia.

Obtendo subconjuntos de linhas pelo número das linhas: `iloc`

Obtenção de subconjuntos de linhas com `ix` não funciona mais no Pandas v0.20

### 2.3.3 Combinando tudo

## **2.4 Cálculos agrupados e agregados**

## **2.5 Plotagem básica**

### **3 Estrutura de dados do Pandas**

### **4 Introdução à plotagem**