Estudo de Python e dados

Sergio Pedro Rodrigues Oliveira

30 July 2025

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

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 contéudo.

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[1] na qual o primeiro valor é o número de linhas e o segundo é a quantidade de colunas.

  Com base nesse resultado anteior, 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)
```

[1] 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	Discrição
Tipo de índice	Informações sobre o índice (por exemplo, RangeIndex).
Número de entradas (linhas)	Quantas linhas seu DataFrame possui.
Número de colunas	Quantas colunas seu DataFrame tem.
Contagem de valores não nulos por coluna	Para cada coluna, informa quantos valores não são nulos. Isso é crucial para identificar dados faltantes.
Dtype (tipo de dado) de cada coluna	Semelhante ao atributo dtype, mas apresentado de forma mais organizada.
Uso de memória	A quantidade de memória que o DataFrame 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	`datetime` 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

2.3.1 Obtendo subconjuntos de colunas

Obtendo subconjuntos de colunas pelo nome

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

2.3.2 Obtendo subconjuntos de linhas

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