

Estudo de Python e dados

Sergio Pedro Rodrigues Oliveira

17 August 2025

SUMÁRIO

| | | |
|----------|--|-----------|
| 1 | Objetivo | 1 |
| 2 | Básico sobre o DataFrame do Pandas | 1 |
| 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> | 15 |
| | Obtenção de subconjuntos de linhas com <code>ix</code> não funciona mais no Pandas v0.20 | 17 |
| 2.3.3 | Combinando tudo | 18 |
| | Obtendo subconjuntos de colunas | 18 |
| | Obtendo subconjuntos de colunas por intervalo | 20 |
| | Fatiando colunas | 23 |
| | Obtendo subconjuntos de linhas e de colunas | 26 |
| | Obtendo subconjuntos de várias linhas e de colunas | 27 |
| 2.4 | Cálculos agrupados e agregados | 29 |
| 2.4.1 | Médias agrupadas | 30 |
| 2.4.2 | Condatodes de frequência agrupados | 30 |
| 2.5 | Plotagem básica | 31 |
| 3 | Estrutura de dados do Pandas | 32 |
| 4 | Introdução à plotagem | 32 |

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 tupla¹ 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)
```

¹Uma 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 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. |
| 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 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

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 (nome da linha). |
| <code>iloc</code> | Subconjunto baseada no índice da linha (número 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.

Obtenção de subconjuntos com várias linhas. Assim como para as colunas, podemos selecionar várias linhas.

```
# Selecione a primeira, a centesima e a milésima linha
# Observe os colchetes duplos, semelhante a sintaxe usada para
# obter subconjuntos com várias colunas
print(df.loc[[0,99,999]])
```

| | country | continent | year | lifeExp | pop | gdpPercap |
|-----|-------------|-----------|------|---------|----------|-------------|
| 0 | Afghanistan | Asia | 1952 | 28.801 | 8425333 | 779.445314 |
| 99 | Bangladesh | Asia | 1967 | 43.453 | 62821884 | 721.186086 |
| 999 | Mongolia | Asia | 1967 | 51.253 | 1149500 | 1226.041130 |

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

`iloc` faz o mesmo que `loc`, mas é usado para obter subconjuntos com base no número de índice das linhas.

Em nosso exemplo atual, `iloc` e `loc` se comportarão exatamente do mesmo modo, pois os rótulos dos índices são os números das linhas. Tenha em mente, porém, que os rótulos dos índices não necessariamente têm de ser os números das linhas.

```
# Obtém a segunda linha
print(df.iloc[1])
```

```
country    Afghanistan
continent      Asia
year        1957
lifeExp     30.332
pop         9240934
gdpPercap   820.85303
Name: 1, dtype: object
```

```
# Obtém a centésima linha
print(df.iloc[99])
```

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

Observe que quando colocamos 1 na lista, na verdade, obtemos a segunda linha, e não a primeira. Isso está de acordo com o comportamento de indexação a partir do 0 de Python, o que significa que o primeiro item de um contêiner é o índice 0 (ou seja, o item 0 do contêiner).

Com `iloc` podemos passar `-1` para **obter a última linha** - algo que não era possível com `loc`.

```
# Usando -1 para obter a última linha
print(df.iloc[-1])
```

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

Como antes, é possível passar **uma lista de inteiros para obter várias linhas**.

```
# Obtém a primeira, a centésima e a milésima linha
print(df.iloc[[0, 99, 999]])
```

| | country | continent | year | lifeExp | pop | gdpPercap |
|-----|-------------|-----------|------|---------|----------|-------------|
| 0 | Afghanistan | Asia | 1952 | 28.801 | 8425333 | 779.445314 |
| 99 | Bangladesh | Asia | 1967 | 43.453 | 62821884 | 721.186086 |
| 999 | Mongolia | Asia | 1967 | 51.253 | 1149500 | 1226.041130 |

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

O atributo `ix` não funciona em versões de Pandas posteriores a v0.20, pois pode ser confuso. Apesar disso, faremos uma revisão rápida de `ix` nesta seção para que a explicação fique completa.

Podemos pensar em `ix` como uma combinação de `loc` e `iloc`, pois permite que tenhamos subconjuntos por rótulo ou por inteiro. Por padrão, ele procura rótulos. Se não puder encontrar o rótulo correspondente, ele recorrerá ao uso de indexação por inteiros. Isso pode ser causa de muitas confusões, e, assim, esse recurso foi removido.

O código que usa `ix` se parecerá exatamente com o código escrito quando `loc` e `iloc` são usados.

```
# Primeira linha
df.ix[0]
```

```
# Centésima linha
df.ix[99]
```

```
# Primeira, centésima e milésima linhas
df.ix[[0,99,999]]
```

2.3.3 Combinando tudo

Os atributos `loc` e `iloc` podem ser usados para obter subconjuntos de colunas, linhas ou de ambos.

A sintaxe geral de `loc` e `iloc` faz uso de colchetes com uma vírgula. A parte à esquerda da vírgula são os valores das linhas para o subconjunto; a parte à direita são os valores das colunas. Ou seja, `df.loc[[rows],[columns]]` ou `df.iloc[[rows],[columns]]`.

Obtendo subconjuntos de colunas

Se quiser usar técnicas para obter subconjuntos somente de colunas, use a sintaxe de fatiamento (slicing) de Python. Temos de fazer isso porque, se estivermos gerando subconjuntos de colunas, teremos todas as linhas da coluna especificadas. Portanto precisamos de um método para capturar todas as linhas.

A sintaxe de fatiamento de Python usa dois-pontos, isto é, `:`. Se tivermos apenas dois-pontos sozinhos, o atributo se referirá a tudo. Assim, se quisermos obter somente a primeira coluna usando a sintaxe `loc` ou de `iloc`, podemos escrever algo como `df.loc[:, [columns]]` para obter o subconjunto da(s) coluna(s).

```
# Obtendo um subconjunto de colunas com loc
# Observe a posição dos dois-pontos
# Ele é usado para selecionar todas as linhas
subset = df.loc[:, ['year', 'pop']]
print(subset.head())
```

| | year | pop |
|---|------|----------|
| 0 | 1952 | 8425333 |
| 1 | 1957 | 9240934 |
| 2 | 1962 | 10267083 |
| 3 | 1967 | 11537966 |
| 4 | 1972 | 13079460 |

```
# Obtendo um subconjunto de colunas com iloc
# iloc nos permitirá usar inteiros
# -1 selecionará a última coluna
subset = df.iloc[:, [2, 4, -1]]
print(subset.head())
```

| | year | pop | gdpPercap |
|---|------|----------|------------|
| 0 | 1952 | 8425333 | 779.445314 |
| 1 | 1957 | 9240934 | 820.853030 |
| 2 | 1962 | 10267083 | 853.100710 |
| 3 | 1967 | 11537966 | 836.197138 |
| 4 | 1972 | 13079460 | 739.981106 |

Obtendo subconjuntos de colunas por intervalo

Podemos usar a função embutida `range` para criar um intervalo de valores em Python. Desse modo, é possível especificar os valores de início e fim, e Python criará automaticamente um intervalo com os valores entre eles.

Por padrão, todo valor entre o início e o fim (**inclusive à esquerda, não inclusive a direita**) será criado, a menos que você especifique um passo.

Em Python 3, a função `range` devolve um gerador.

Se estiver usando Python 2, a função `range` devolverá uma lista e a função `xrange` devolve um gerador.

Se observarmos o código apresentado antes, veremos que subconjuntos de colunas foram obtidos usando uma lista de inteiros. Como `range` devolve um gerador, é preciso convertê-lo em uma lista antes.

```
list(range(5))
```

Observe que, quando `range(5)` é chamado, cinco inteiros são devolvidos: 0-4.

```
# Cria um intervalo de inteiros de 0 a 4 inclusive, [0,5)
small_range = list(range(5))
print(small_range)
```

```
[0, 1, 2, 3, 4]
```

```
# Obtém um subconjunto de dataframe usando o intervalo
subset = df.iloc[:,small_range]
print(subset.head())
```

| | country | continent | year | lifeExp | pop |
|---|-------------|-----------|------|---------|----------|
| 0 | Afghanistan | Asia | 1952 | 28.801 | 8425333 |
| 1 | Afghanistan | Asia | 1957 | 30.332 | 9240934 |
| 2 | Afghanistan | Asia | 1962 | 31.997 | 10267083 |
| 3 | Afghanistan | Asia | 1967 | 34.020 | 11537966 |
| 4 | Afghanistan | Asia | 1972 | 36.088 | 13079460 |


```
# Cria um intervalo de 3 a 5 inclusive, [3,5] ou [3,6)
small_range = list(range(3,6))
print(small_range)
subset = df.iloc[:,small_range]
print(subset.head())
```

```
[3, 4, 5]
   lifeExp    pop  gdpPercap
0  28.801  8425333  779.445314
1  30.332  9240934  820.853030
2  31.997 10267083  853.100710
3  34.020 11537966  836.197138
4  36.088 13079460  739.981106
```

Pergunta (Desafio 1):

O que acontecerá se você especificar um intervalo que estiver além do número de colunas existente?

Resposta:

Erro do tipo `IndexError`.

Mais uma vez, observe que os valores são especificados de modo que **o intervalo é inclusivo à esquerda, mas não a direita**.

```
# Cria um intervalo de 0 a 5 inclusive, com inteiros alternados [0,6)
small_range = list(range(0,6,2))
subset = df.iloc[:,small_range]
print(subset.head())
```

| | country | year | pop |
|---|-------------|------|----------|
| 0 | Afghanistan | 1952 | 8425333 |
| 1 | Afghanistan | 1957 | 9240934 |
| 2 | Afghanistan | 1962 | 10267083 |
| 3 | Afghanistan | 1967 | 11537966 |
| 4 | Afghanistan | 1972 | 13079460 |

Converter um gerador em uma lista é um pouco complicado; podemos usar a sintaxe de fatiamento de Python para dar um jeito nisso.

Fatiando colunas

A sintaxe de **fatiamento de Python**, `:`, é semelhante à sintaxe de `range`. Em vez de usar uma função que especifique os valores de início, fim e o passo, delimitados por vírgula, separamos os valores com dois-pontos.

Se você entendeu o que estava acontecendo com a função `range` que usamos antes, o fatiamento então poderá ser visto como um atalho para fazer o mesmo.

Exemplo `range`:

```
small_range = list(range(3))
subset = df.iloc[:,small_range]
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 |

Exemplo fatiamento de Python:

```
# Fatia as três primeiras colunas
subset = df.iloc[:,3]
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 |

Exemplo range:

```
small_range = list(range(3,6))
subset = df.iloc[:,small_range]
print(subset.head())
```

| | lifeExp | pop | gdpPercap |
|---|---------|----------|------------|
| 0 | 28.801 | 8425333 | 779.445314 |
| 1 | 30.332 | 9240934 | 820.853030 |
| 2 | 31.997 | 10267083 | 853.100710 |
| 3 | 34.020 | 11537966 | 836.197138 |
| 4 | 36.088 | 13079460 | 739.981106 |

Exemplo fatiamento de Python:

```
# Fatia as colunas de 3 a 5 inclusive, [3,6)
subset = df.iloc[:,3:6]
print(subset.head())
```

| | lifeExp | pop | gdpPercap |
|---|---------|----------|------------|
| 0 | 28.801 | 8425333 | 779.445314 |
| 1 | 30.332 | 9240934 | 820.853030 |
| 2 | 31.997 | 10267083 | 853.100710 |
| 3 | 34.020 | 11537966 | 836.197138 |
| 4 | 36.088 | 13079460 | 739.981106 |

Exemplo range:

```
small_range = list(range(0,6,2))  
subset = df.iloc[:,small_range]  
print(subset.head())
```

| | country | year | pop |
|---|-------------|------|----------|
| 0 | Afghanistan | 1952 | 8425333 |
| 1 | Afghanistan | 1957 | 9240934 |
| 2 | Afghanistan | 1962 | 10267083 |
| 3 | Afghanistan | 1967 | 11537966 |
| 4 | Afghanistan | 1972 | 13079460 |

Exemplo fatiamento de Python:

```
# Fatia as cinco primeiras colunas alternadamente  
subset = df.iloc[:,6:2]  
print(subset.head())
```

| | country | year | pop |
|---|-------------|------|----------|
| 0 | Afghanistan | 1952 | 8425333 |
| 1 | Afghanistan | 1957 | 9240934 |
| 2 | Afghanistan | 1962 | 10267083 |
| 3 | Afghanistan | 1967 | 11537966 |
| 4 | Afghanistan | 1972 | 13079460 |

Obtendo subconjuntos de linhas e de colunas

Temos usado dois-pontos, :, em `loc` e em `iloc` à esquerda da vírgula. Quando fazemos isso, selecionamos todas as linhas de nosso dataframe. No entanto, podemos optar por colocar valores à esquerda da vírgula se quisermos selecionar linhas específicas, além de colunas específicas.

```
# Usando loc
print(df.loc[42, 'country'])
```

Angola

```
# Usando iloc
print(df.iloc[42, 0])
```

Angola

Certifique-se de que não se esquecerá das diferenças entre `loc` (rótulo do índice, nome) e `iloc` (índice, número).

Observe agora como `ix` pode ser confuso. É bom que ele não esteja mais funcionando.

Obtendo subconjuntos de várias linhas e de colunas

Podemos combinar a sintaxe de obtenção de subconjuntos de linhas e de colunas com a sintaxe de subconjuntos de várias linhas e várias colunas a fim de obter fatias de nossos dados.

```
# Obtém a primeira, a centésima e a milésima linha
# da primeira, quarta e sexta coluna;
# As colunas que esperamos obter são
# country, lifeExp e gdpPercap
print(df.iloc[[0,99,999],[0,3,5]])
```

| | country | lifeExp | gdpPercap |
|-----|-------------|---------|-------------|
| 0 | Afghanistan | 28.801 | 779.445314 |
| 99 | Bangladesh | 43.453 | 721.186086 |
| 999 | Mongolia | 51.253 | 1226.041130 |

Atenção!!!

Em meu trabalho, tento passar os nomes das colunas para obter subconjuntos de dados, sempre que possível. Essa abordagem deixa o código mais legível, pois não será necessário observar o vetor de nomes das colunas para saber qual índice está sendo especificado.

Além disso, usar índices absolutos (número da coluna) pode resultar em problemas caso a ordem das colunas seja alterada por algum motivo.

Essa é somente uma regra geral, uma vez que haverá exceções em que usar a posição do índice será uma opção melhor (por exemplo, para concatenar dados).

```
# Se usarmos os nomes das colunas diretamente,
# o código será um pouco mais fácil de ler
# Observe agora que temos que usar loc ao invés de iloc
print(df.loc[[0,99,999],['country','lifeExp','gdpPercap']])
```

| | country | lifeExp | gdpPercap |
|-----|-------------|---------|-------------|
| 0 | Afghanistan | 28.801 | 779.445314 |
| 99 | Bangladesh | 43.453 | 721.186086 |
| 999 | Mongolia | 51.253 | 1226.041130 |

Usar loc sempre que possível!!!

Lembre-se de que podemos usar a sintaxe de fatiamento na parte referente às linhas dos atributos `loc` e `iloc`.

```
print(df.loc[10:13,['country','lifeExp','gdpPercap']])
```

| | country | lifeExp | gdpPercap |
|----|-------------|---------|-------------|
| 10 | Afghanistan | 42.129 | 726.734055 |
| 11 | Afghanistan | 43.828 | 974.580338 |
| 12 | Albania | 55.230 | 1601.056136 |
| 13 | Albania | 59.280 | 1942.284244 |

2.4 Cálculos agrupados e agregados

Se você já trabalhou com outras bibliotecas numéricas ou linguagens, saberá que muitos cálculos estatísticos básicos estarão disponíveis na biblioteca ou embutidos na linguagem. Vamos observar novamente nossos dados Gapminder.

```
print(df.head(n=10))
```

| | 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 |
| 5 | Afghanistan | Asia | 1977 | 38.438 | 14880372 | 786.113360 |
| 6 | Afghanistan | Asia | 1982 | 39.854 | 12881816 | 978.011439 |
| 7 | Afghanistan | Asia | 1987 | 40.822 | 13867957 | 852.395945 |
| 8 | Afghanistan | Asia | 1992 | 41.674 | 16317921 | 649.341395 |
| 9 | Afghanistan | Asia | 1997 | 41.763 | 22227415 | 635.341351 |

Perguntas estatísticas

Há várias perguntas iniciais que podemos nos fazer:

1. Para cada ano em nossos dados, qual era a expectativa de vida média? Qual é a expectativa de vida média, a população e o GDP?
2. E se estratificarmos os dados por continente e fizermos os mesmos cálculos?
3. Quantos países estão listados para cada continente?

2.4.1 Médias agrupadas

2.4.2 Condatodes de frequência agrupados

2.5 Plotagem básica

3 Estrutura de dados do Pandas

4 Introdução à plotagem