> Para ser cientista de dados, você precisa de dados. Na verdade, como cientista de dados, você gastará uma (boa) fração de seu tempo adquirindo, limpando e transformando base de dados. Neste capítulo, veremos diferentes maneiras de se obter dados em Python e colocá-los nos formatos corretos.

Lendo arquivos

Você pode ler e gravar arquivos diretamente em seu código. A linguagem Python torna o trabalho com arquivos bastante simples.

Arquivos de Texto

O primeiro passo para trabalhar com um arquivo de texto é obtê-lo. Usualmente, este arquivo será enviado à você. No diretório do curso, algumas planilhas estarão disponíveis como exemplo. Vamos nos concentrar em arquivos com extensão `.csv`:

In [ ]:
```
# Primeiro, voce deve alocar o arquivo em uma determinada pasta de destino
# Seja no seu computador ou na nuvem

import pandas as pd
nomes_br = pd.read_csv("/content/nomes_br2019.csv")

# "/content/" representa o endereco da pasta onde esta localizado o arquivo
# "nomes_br2019" e o nome do arquivo
# ".csv" e a extensão do arquivo
```

Teremos um `data frame` da seguinte forma:

In [ ]:
```
# 0     ALINE   F       530550
# 1     ARAO    M       3526
# 2     ARON    M       3442
# 3     ADA       F     5583
# 4     ABADE   M       57
# ...
```

> Tenha sempre cuidado no preenchimento do endereço e a extensão do arquivo. Na grande maioria dos casos, o caminho será especificado da seguinte forma: `pasta onde o arquivo se encontra/nome do arquivo.csv'`.

> Note que o índice da primeira linha é **zero**. Esse é o padrão do Python.

Arquivos com extensão `.csv` tem padrões diferentes. Por uma questão regional, eventualmente utilizamos arquivos deste tipo com separadores diferentes. Assim, simplesmente devemos adicionar o seguinte opcional:

In [ ]:
```
# Separador = ;
dados_pixar = pd.read_csv("/content/dados_pixar.csv", sep = ";")
```

Neste ferramental do Python para leitura de arquivos, temos muitas opções. Não vamos explorá-las aqui mas pode-se citar que podemos limitar o número de linhas (importante para arquivos muito grandes), pular determinadas linhas, selecionar e nomear colunas e por aí vai. Aqui você pode consultar uma ajuda (em inglês):

In [ ]:
```
?pd.read_csv
```

Planilhas de Texto

Usualmente, também trabalhamos bastante com planilhas de texto. Elas são uma fonte de análise de dados acessíveis à todos e facilitam muito na visualização do banco de dados. Trazê-las para o Python também é muito simples:

In [ ]:
```
dados_starwars = pd.read_excel("/content/dados_starwars.xlsx")
```

In [ ]:
```
# 0     Luke Skywalker  172.0   77.0    Loiro ...
# 1              C-3PO  167.0   75.0    NaN       ...
# 2              R2-D2  96.0    32.0    NaN       ...
# ...
```

> Esta função suporta diversos tipos de extensões: `xls`, `xlsx`, `xlsm`, `xlsb`, `odf`, `ods` e `odt`.

> Se você abrir este arquivo em um editor de planilhas, verá que as células das linhas 2 e 3 da coluna 4 estarão vazias (ou em branco). O Python utiliza `NaN` para representar um **valor faltante**. Grande parte do tempo você irá se deparar com este tipo de ocorrência.

Assim como a função anterior, esta também possui um grande conjunto de argumentos opcionais que nos auxiliam nas mais diversas leituras. Para saber mais, veja:

In [ ]:
```
?pd.read_excel
```

Raspando dados da Web

Outra maneira de obter dados é raspando-os de páginas da web. Buscar páginas da web, ao que parece, é bem fácil. Por outro lado, obter informações estruturadas significativas delas, bem menos.

Para obter dados do `HTML`, usaremos a biblioteca `Beautiful Soup`, que constrói uma árvore com os vários elementos de uma página da web e fornece uma interface simples para acessá-los. Essa é uma das mais difíceis tarefas de obtenção de dados. O Python fica limitado à páginas bem estruturadas. Para além disso, você irá necessitar de muita experiência e paciência.

Vamos estudar um exemplo. A *Wikipedia* disponibiliza um oceano de informações. Suponha que desejamos algumas informações valiosas sobre o maravilhoso Estado do Rio de Janeiro disponibilizadas neste link):

In [ ]:
```
import requests
from bs4 import BeautifulSoup

# retirando o texto do HTML de uma pag na web
url = requests.get('https://pt.wikipedia.org/wiki/Rio_de_Janeiro_(estado)').text

# fazendo a busca
busca = BeautifulSoup(url,'lxml')
```

Note que o resultado da busca é exatamente o montante de linhas de códigos que aparecem quando inspecionamos uma página da *web*. Para visualizá-lo manualmente, abra o link no seu navegador de preferência, clique com o botão direito do *mouse* e depois, clique em 'Inspecionar'.

Agora, iremos filtrar o código para retirar somente as tabelas. Em `HTML`, as classes de tabelas se chamam `table`:

In [ ]:
```
tabela = busca.find_all('table')
```

E teremos um amontoado de texto nada inteligível como este:

In [ ]:
```
# [<table class="box-Mais_fontes plainlinks metadata ambox
# ambox-content ambox-Refimprove" role="presentation"><tbody><tr>
# <td class="mbox-image"><div style="width:52px"><a class="image"
# href="/wiki/Ficheiro:Question_book-new.svg"><img alt="" data-file-height="399"
# data-file-width="512" decoding="async" height="39"
# ...
```

Vamos selecionar diferentes tabelas da página e transformá-las diretamente em um `data frame`:

In [ ]:
```
# tab 1: crescimento populacional
cresc_pop = pd.read_html(str(tabela))[6]

# tab 2: populacao por cor/raca
pop_cor = pd.read_html(str(tabela))[7]

# tab 3: lista de municipios mais populosos
munic_maispop = pd.read_html(str(tabela))[8]
```

> Note que a função `pd.read_html` lê várias tabelas de uma única vez. O valor entre colchetes significa que estamos selecionando uma tabela específica conforme a sua ordem. Por exemplo, `[6]` significa que estamos selecionando a *sétima* tabela.

Infelizmente, as tabelas não são perfeitamente transformadas/traduzidas e quase sempre se faz necessário a sua edição. No próximo capítulo, aprenderemos algumas ferramentas para nos auxiliar nesta nova etapa.

Usando APIs

Muitos sites e serviços da *Web* fornecem interfaces de programação de aplicativos (APIs), que permitem solicitar dados explicitamente em um formato estruturado. Isso evita (muito) trabalho de raspá-los! Além disso, a linguagem Python possui diversos colaboradores mundo a fora, inclusive no Brasil. Assim, importantes bases de dados são traduzidas por meio de bibliotecas e disponibilizadas à todos os usuários de forma direta e simples.

No Brasil, muitas bases de dados se destacam. Vamos destacar três delas:

- **IBGE**: O Instituto Brasileiro de Geografia e Estatística (IBGE) tem atribuições ligadas às geociências e estatísticas sociais, demográficas e econômicas, o que inclui realizar censos e organizar as informações obtidas nesses censos, para suprir órgãos das esferas governamentais federal, estadual e municipal, e para outras instituições e o público em geral;
- **BACEN**: O Banco Central do Brasil é uma das principais autoridades monetárias do país. Fornece dados através do Sistema Gerenciador de Séries Temporais (SGS);
- **IPEA**: O Instituto de Pesquisa Econômica Aplicada (Ipea) é uma fundação pública federal vinculada ao Ministério da Economia que fornece suporte técnico e institucional às ações do governo para a formulação e reformulação de políticas públicas e programas de desenvolvimento. Fornece dados econômicos e financeiros, demográficos, de renda, educação, saúde, habitação, trabalho através da base Ipeadata.

Inicialmente, precisamos instalar essas bibliotecas virtualmente:

In [ ]:
```
!pip install ibge sgs ipeadatapy
```

Para ilustrar, podemos buscar dados diretamente do IBGE sobre os 5570 municípios de nosso país:

In [ ]:
```
from ibge.localidades import *

municip = Municipios()
df_municip = pd.json_normalize(municip.json())
```

Note que a base de dados já vem completamente organizada. Algum colaborador já fez todo o trabalho complicado, facilitando nossas tarefas.

O BCB fornece milhares de informações econômicas. Vamos buscar a série temporal do valor da cesta básica no município do Rio de Janeiro de 01/01/1998 à 01/01/2024. O código SGS dessa série é `7491`. Algo que você pode encontrar através de uma busca manual pelo link:

In [ ]:
```
import sgs

# o codigo da serie de cesta basica no RJ
# e 7491
df_cesta = sgs.dataframe(7491, start = '01/01/1998', end = '01/01/2024')
```

Apenas 1 linha de código e o poder está em suas mãos!

Por fim, vamos buscar o *valor real* do salário mínimo ao longo dos anos na base Ipeadata:

In [ ]:
```
import ipeadatapy

# o codigo da serie de salario minimo na base
# e 'GAC12_SALMINRE12'
df_salmin = ipeadatapy.timeseries('GAC12_SALMINRE12')
```

> **Valor real** do salário mínimo significa que ele foi *deflacionado*. Ou seja, foi feito um cálculo na tentativa de remoção da inflação ao longo dos anos de forma que os valores do passado sejam comparáveis de forma direta com os mais recentes.

Referências Adicionais

- IBGE, BCB e Ipea são as bases de dados citadas nos exemplos;
- Pandas é a biblioteca primária que contém os tipos mais utilizados na ciência de dados para trabalhar e importar dados;
- Scrapy é uma biblioteca completa para construir raspadores de dados complicados;
- Kaggle hospeda uma grande coleção de conjuntos de dados (em inglês).