Aula 05 - Extração de dados

Módulos externos em Python

Assim como temos a biblioteca padrão do Python, com módulos importantes como `math`, `os.path`, `subprocess` e `random`, a linguagem possibilita o desenvolvimento de novas bibliotecas e novos pacotes pelos desenvolvedores. Essa é uma das maiores vantagens do Python: como a comunidade de desenvolvedores é gigantesca, temos à nossa disposição inúmeros pacotes para podermos importar e utilizar nos nossos projetos.

Qualquer um pode desenvolver um módulo novo. Apesar de esse não ser o foco do nosso curso, basta uma pesquisa rápida no Google para ver como é simples desenvolver um módulo próprio, para ser reaproveitado por nós mesmos ou então para distribui-lo na comunidade.

Quando queremos distribuir um pacote, podemos fazer isoladamente ou então utilizar uma plataforma que faz a curadoria desses pacotes. Em Python, a principal plataforma é conhecida por **PyPI**, ou *Python Package Index*. Esse portal é administrado pelos próprios mantenedores da linguagem, que fazem uma curadoria dos pacotes, verificando potenciais problemas de segurança e garantindo a estabilidade desses módulos. Pense no PyPI como se fosse uma loja de aplicativos, como a Play Store.

Instalando o pip

Para utilizarmos esses pacotes, precisamos instalá-los nas nossas máquinas. Para isso, podemos utilizar um programa chamado `pip`. Esse programa normalmente vem instalado juntamente com o Python. Para verificar se você possui uma instalação do pip no seu computador, abra o prompt de comando e entre com a seguinte instrução:

```
python -m pip --version
```

Caso o pip esteja instalado, aparecerá uma versão do programa. Caso contrário, recomendo desinstalar a sua distribuição do Python e instalar novamente, tomando o cuidado de marcar a caixa "instalar pip" quando for solicitado.

Duas observações:

- Para o comando apresentado acima funcionar, é necessário ter incluído o Python nas suas variáveis de ambiente. Para isso, abra o menu do Windows e digite "Editar as variáveis de ambiente para sua conta". Selecione a variável **Path** e clique em "Editar". Na janela que aparecer, clique em "Novo" e insira aí o caminho para a sua instalação do python (p.ex., *D:\Python*).
- Esta página fala sobre a instalação do Python e do pip em Windows. Para a instalação no Mac, veja este link.

Instalando os pacotes

Com o pip instalado, é possível instalar os pacotes que estiverem cadastrados no PyPI. Para instalar um novo pacote, basta abrir o prompt de comando e inserir a seguinte instrução:

```
python -m pip install <nome_do_modulo>
```

O pip vai, então, proceder com o download e a instalação do pacote. Para verificar se foi instalado corretamente você pode proceder de duas formas:

- Abrir o terminal interativo do Python e chamar o pacote com um `import`:

```
D:\Projects>python
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>>
```

- Mandar exibir todos os pacotes já instalados:

```
python -m pip freeze
```

Dica: Com o aumento no número de projetos que vamos participar, é comum utilizarmos muitos pacotes externos. Para complicar um pouco mais, é possível usar versões diferentes de um mesmo pacote em projetos diferentes. Sendo assim, caso você vá trabalhar com vários projetos em seu computador (o que é bem comum), recomendo que você estude um pouco sobre ambientes virtuais (ou *virtual envs*). Este link dá algumas dicas de como utilizar essa solução.

Utilizando os pacotes externos e dicas

Com os pacotes externos corretamente instalados, para usá-los basta importar da mesma forma que importamos um módulo da biblioteca padrão do Python, utilizando a instrução `import` e suas variações.

Assim como a biblioteca padrão do Python, os pacotes mais populares e maduros possuem vastas documentações que são esclarecedoras e bem completas. Recomendo que sempre que usar um pacote, escreva o código sempre acompanhado da documentação adequada.

Abaixo mostro alguns pacotes que normalmente utilizamos na área de Ciência de Dados, com os respectivos links para as documentações:

- requests
- beautifulsoup
- scikit-learn
- pandas

- numpy
- matplotlib
- seaborn

Durante o nosso curso, como estamos utilizando o Google Colab para escrever nossos notebooks, não vamos precisar instalar esses pacotes -- eles já estão instalados no servidor que vamos usar.

Relembrando o processo de mineração de dados

Como vimos na última aula, o processo de mineração de dados envolve seis diferentes fases:

1. Entendimento do negócio
2. Entendimento dos dados
3. Preparação dos dados
4. Modelagem
5. Avaliação
6. Entrega (ou *deploy*)

Nossas próximas aulas vão tratar do processo de desenvolvimento para permitir executar as fases 2 e 3. Normalmente, esse processo é chamado de **ETL**, do inglês *Extraction, Transformation and Load*, ou Extração, Transformação e Carregamento. Essas são as três etapas básicas para preparar os dados para posterior análise:

- Extração: coleta dos dados crus a partir de sistemas ERP, fontes de dados online, planilhas do excel, bancos de dados corporativos, etc.;
- Transformação: limpeza e organização dos dados, nomeação de colunas, exclusão de dados inválidos, balanceamento de dados, etc.;
- Carga: armazenamento dos dados limpos em um banco de dados local ou em um arquivo formatado de dados.

Na aula de hoje vamos falar especificamente da primeira etapa de um ETL: a extração dos dados.

Como podemos extrair os dados

Como falamos, os dados podem ser extraídos de várias formas, dependendo do contexto do problema que temos que resolver:

- Extração de dados locais, armazenados em um arquivo de texto, arquivo separado por vírgulas, de extensão .json ou armazenados em um banco de dados local;
- Extração de dados em um servidor, armazenados nos mesmos formatos dos arquivos locais;

- Extração de dados na web, através do acesso a uma API ou utilizando *web scraping* para leitura de textos em páginas da web.

A extração de dados armazenados em arquivos, de qualquer extensão (.txt, .csv, .json, .xml, etc.), é relativamente simples. Normalmente as bibliotecas de manipulação de dados (como pandas) já possuem funções prontas para isso.

Não focaremos nesse curso a extração de dados oriundos de bancos de dados locais ou em servidor, já que isso exige um conhecimento em bancos de dados que não conseguimos cobrir.

Portanto, trabalharemos na aula de hoje com extração de dados da web, ou seja, como podemos acessar páginas online ou APIs para extrair os dados que precisamos e salvar nas

Tipos de dados

Os dados podem ser obtidos nos mais diversos formatos, mas os mais comuns atualmente são:

- csv, ou *comma-separated values*: é um formato que armazena dados separados por vírgula (ou algum outro separador -- no Brasil os computadores são configurados para usar, normalmente, o ponto-e-vírgula). É uma formato bem simples, e muitas vezes ineficiente. Por exemplo, quando queremos colocar textos mais longos, que utilizam vírgulas, precisamos tomar algumas precauções. Além disso, esse formato não permite dados muito complexos, com vários subníveis. Portanto, o csv é um formato usado apenas para dados muito simples, como tabelas com valores numéricos.

```
nome,matricula,idade,curso
João,1234,20,Administração
Fernando,1235,21,Engenharia
Luísa,1233,19,Direito
Amanda,1236,20,Direito
```

- json, ou *javascript object notation*: é um formato compacto e de padrão aberto, criado especificamente para troca de dados rápida. Ele foi concebido de forma a ser simples para processá-lo, e ao mesmo tempo ser legível para os humanos. Atualmente é o formato de troca de dados mais comumente aceito para praticamente qualquer tipo de dado. O formato se assemelha muito ao de um dicionário (ou a uma lista de dicionários) em Python. Para mais informações sobre a sintaxe do formato, recomendo uma lida no site oficial.

```
[
  {
    "nome": "João",
    "matricula": 1234,
    "idade": 20,
    "curso": "Administração"
```

```
  },
  {
    "nome": "Fernando",
    "matricula": 1235,
    "idade": 21,
    "curso": "Engenharia"
  },
  {
    "nome": "Luísa",
    "matricula": 1233,
    "idade": 19,
    "curso": "Direito"
  },
  {
    "nome": "Amanda",
    "matricula": 1236,
    "idade": 20,
    "curso": "Direito"
  }
]
```

- html, ou *hypertext markup language*: é uma linguagem de marcação utilizada no desenvolvimento de páginas web. É um formato extremamente popular, pois ele consegue ser interpretado pelos navegadores da web, como o Chrome e o Safari. Apesar de não ser um formato para transporte de dados, quando utilizamos técnicas de *web scraping* é bem comum termos que manipular arquivos html.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Título da Página</title>
  </head>
  <body>
    <h1>Este é um título</h1>
    <p>Este é um parágrafo.</p>
  </body>
</html>
```

- xml, ou *eXtensible markup language*: é um formato que define regras para codificar diferentes tipos de documentos. A legibilidade desse formato tende a ser bem prejudicada, portanto ultimamente ele tem perdido espaço para o format json. No entanto, ainda é bastante utilizado em contextos específicos, como por exemplo na geração de notas fiscais eletrônicas.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<alunos>
  <aluno>
    <nome>João</nome>
    <matricula>1234</matricula>
    <idade>20</idade>
    <curso>Administração</curso>
  </aluno>
  <aluno>
    <nome>Fernando</nome>
    <matricula>1235</matricula>
    <idade>21</idade>
    <curso>Engenharia</curso>
  </aluno>
  <aluno>
    <nome>Luísa</nome>
    <matricula>1233</matricula>
    <idade>19</idade>
    <curso>Direito</curso>
  </aluno>
  <aluno>
    <nome>Amanda</nome>
    <matricula>1236</matricula>
    <idade>20</idade>
    <curso>Direito</curso>
  </aluno>
</alunos>
```

Como os dados .xml possuem um uso muito específico atualmente e é raramente usado em mineração de dados, não vamos focar nele dentro deste curso. Portanto, vamos focar nos outros três formatos.

Sobre APIs

Uma das formas mais comuns de se extrair dados online é através do consumo de **APIs**. API é uma sigla para *Application Programming Interface*, ou Interface de Programação de Aplicações. É uma interface utilizada por muitos desenvolvedores para permitirem o acesso a dados ou a funções por outros desenvolvedores.

Podemos pensar na `matplotlib`, por exemplo, como uma API desenvolvida para permitir o uso de funções que geram gráficos na tela.

Uma API web, por sua vez, é uma API desenvolvida de forma que possamos acessar seus dados através de um acesso a uma página da web. Esses dados podem ser acessados

programaticamente, facilitando a sua manipulação e permitindo a execução automática desse código.

Com o acesso a uma API financeira, por exemplo, podemos montar um programa que acompanha as movimentações diárias de ativos e informa dados corporativos, como lucro, patrimônio líquido, etc. Neste link tem um exemplo bem interessante sobre como podemos usar uma API em Python para extrair dados financeiros.

Protocolo HTTP

Para podermos acessar qualquer página na web, incluindo as APIs, precisamos utilizar um protocolo específico, chamado **HTTP**, ou *hypertext transfer protocol*. Esse é o protocolo padrão para podermos transferir dados e solicitações entre diferentes páginas na web. Não é o nosso foco discutir sobre o protocolo HTTP neste curso, mas precisamos falar de dois assuntos principais: como nos comunicamos com as páginas e quais são as possíveis respostas.

Formas de comunicação

O protocolo HTTP tem dois principais métodos de comunicação com uma página, **GET** e **POST**.

O método **POST** é normalmente usado para envio de informações, como por exemplo quando queremos fazer o upload de um arquivo local em uma página, ou queremos inserir uma foto em um site. Como estamos querendo extrair dados, e não inseri-los, não vamos discutir esse método no curso.

Já o método **GET** solicita uma representação do recurso que for especificado. As solicitações usando GET só devem recuperar dados. Praticamente todas as solicitações que fazemos na web como usuários, como por exemplo acessar o site do Google, são feitas utilizando o método GET "por baixo dos panos". Ou seja, o navegador lê o endereço que queremos acessar e faz uma solicitação GET para o servidor que está armazenando esse site.

Com Python podemos fazer solicitações GET (e POST também) utilizando o módulo externo `requests`. Com esse módulo conseguimos fazer praticamente todas as requisições do protocolo HTTP.

Respostas das páginas

Ao fazer uma solicitação GET utilizando o protocolo HTTP, o site retorna diversos dados, além da informação que solicitamos. Um dado importante para a nossa análise é o código de resposta da página.

O código de resposta é um número de três dígitos entre 100 e 599, que representam como foi o processo de comunicação com a página. As respostas são agrupadas em cinco classes:

- Respostas de informação (100 a 199);
- Respostas de sucesso (200 a 299);

- Redirecionamentos (300 a 399);
- Erros do cliente (400 a 499);
- Erros do servidor (500 a 599).

As classes mais importantes para o nosso curso são as de sucesso (2XX), os erros do cliente (4XX) e os erros do servidor (5XX). As respostas mais importantes são indicadas abaixo:

- 200: OK - requisição bem sucedida;
- 400: Bad Request - o servidor não entendeu a requisição pois ela está com uma sintaxe inválida;
- 401: Unauthorized - o cliente deve se autenticar para obter a resposta adequada;
- 403: Forbidden - o cliente está autenticado, porém não tem direito a acessar o conteúdo;
- 404: Not Found - o servidor não consegue encontrar o recurso solicitado;
- 429: Too Many Requests - o usuário enviou muitas requisições num dado tempo;
- 500: Internal Server Error - o servidor encontrou uma situação com a qual não sabe lidar;
- 503: Service Unavailable - o servidor não está pronto para manipular a requisição, ele pode estar em manutenção ou sobrecarregado.

Aplicação 01 - SWAPI

SWAPI é uma API pública com dados dos filmes da franquia *Star Wars*. Essa API já está descontinuada, ou seja, não estão adicionando informações novas há um tempo. No entanto, como ela está disponível sem nenhum tipo de autenticação, ela é um bom ponto de partida para as nossas aplicações.

Acessando a página de documentação da API é possível identificar quais são as requisições aceitas, bem como os recursos disponíveis. Nesse caso, temos recursos como `people`, `films` e `starships`.

Como em praticamente todos os nossos processos de extração de dados online, devemos começar com o uso do módulo `requests`.

```
import json  # módulo da lib padrão do python, para manipular arquivos json
from pprint import pprint # módulo para apresentar de forma elegante os json

import requests

# Definindo os dados básicos da API:
SWAPI = {
    "main_url": "https://swapi.dev/api/",
    "resources": ["people", "films", "starships", "vehicles", "species", "planets"]
}

# Vamos coletar os dados de uma entrada para ver como a informação é retornada:
req = requests.get(f"{SWAPI['main_url']}people/1")
req
```

```
<Response [200]>
```

```
# Podemos pegar algumas informações de req:
print(req.status_code)
print(req.headers)
print(req.url)
print(req.text)
```

```
200
{'Server': 'nginx/1.16.1', 'Date': 'Tue, 07 Sep 2021 21:02:14 GMT', 'Content-Type':
https://swapi.dev/api/people/1
{"name":"Luke Skywalker","height":"172","mass":"77","hair_color":"blond","skin_color
```

```
# Caso o tipo do resultado seja json, podemos usar a função json() para
# converter o conteúdo de texto.
if "json" in req.headers["Content-Type"]:
  pprint(req.json())
```

```
{'birth_year': '19BBY',
 'created': '2014-12-09T13:50:51.644000Z',
 'edited': '2014-12-20T21:17:56.891000Z',
 'eye_color': 'blue',
 'films': ['https://swapi.dev/api/films/1/',
           'https://swapi.dev/api/films/2/',
           'https://swapi.dev/api/films/3/',
           'https://swapi.dev/api/films/6/'],
 'gender': 'male',
 'hair_color': 'blond',
 'height': '172',
 'homeworld': 'https://swapi.dev/api/planets/1/',
 'mass': '77',
 'name': 'Luke Skywalker',
 'skin_color': 'fair',
 'species': [],
 'starships': ['https://swapi.dev/api/starships/12/',
               'https://swapi.dev/api/starships/22/'],
 'url': 'https://swapi.dev/api/people/1/',
 'vehicles': ['https://swapi.dev/api/vehicles/14/',
              'https://swapi.dev/api/vehicles/30/']}
```

Podemos salvar esse resultado em json (e em qualquer outro formato, na verdade) em um arquivo no nosso computador, para usarmos posteriormente.

Isso é uma prática relativamente comum: muitas vezes o processo de extração de dados é extenso, e precisamos armazenar vários arquivos antes de começar a limpá-los. Dessa forma, o ideal é ir armazenando o conteúdo salvo no computador.

No nosso caso, como estamos usando o Colab, o arquivo fica salvo no menu "Arquivos", no ícone de pasta na barra lateral.

Para armazenar um arquivo localmente utilizando Python, nós usamos as funções `open()` e `close()`. Ou seja, abrimos um arquivo com a função `open()`, fazemos as operações necessárias de leitura e escrita, e depois fechamos o arquivo com a função `close()`.

No entanto, caso ocorra um erro de execução enquanto o arquivo estiver aberto, o arquivo pode ser danificado ou até mesmo corrompido. Portanto, é mais seguro utilizar a instrução `with`

`open()`.

Com essa instrução, caso haja algum erro, o próprio interpretador do Python toma as devidas providências para encerrar o arquivo de forma segura.

A função `open()` utiliza dois parâmetros principais:

- `file`: nome do arquivo que será aberto (ou criado). Caso esse arquivo esteja localizado em outra pasta, é necessário colocar o caminho adequado. Recomendo o uso dos módulos `os.path` ou `pathlib`) para navegar entre pastas;
- `mode`: modo como o arquivo será aberto. Os modos mais comuns são:
  - `"r"`: abertura para leitura apenas (valor padrão);
  - `"w"`: abertura para escrita apenas, ignorando o conteúdo já armazenado no arquivo, caso ele já exista;
  - `"a"`: abertura para escrita apenas, incluindo o que for escrito no final do arquivo;

É uma boa prática abrir um arquivo **apenas** para escrita ou **apenas** para leitura. Dessa forma é possível controlar melhor como o arquivo está sendo manipulado.

```
# Vamos construir uma função para poder reaproveitar em outros projetos.
# Veja que essas funções não estão "protegidas", ou seja, não estão
# levando em consideração possíveis cenários de erro, como:
# - Arquivo não existente na hora de ler
# - Tipo de dado do conteúdo incompatível com o formato json
# - Nome de arquivo com formato inválido
# - etc.
# Não é nosso objetivo tratar esses casos aqui, mas recomendo
# estudarem sobre tratamento de erros e exceções (estrutura try/except).

def salvar(nome_arquivo, conteudo, e_json):
  with open(nome_arquivo, "w") as arquivo:
    if e_json:
      # Usamos a função json.dump() para salvar arquivos.
      # O parâmetro `indent` serve para definir
      # a indentação dos dados no arquivo
      json.dump(conteudo, arquivo, indent=4)
    else:
      arquivo.write(conteudo)

def ler(nome_arquivo, e_json):
  with open(nome_arquivo, "r") as arquivo:
    conteudo = json.load(arquivo) if e_json else arquivo.read()

  return conteudo


salvar("exemplo.json", req.json(), True)

pprint(ler("exemplo.json", True))
```

```
     {'birth_year': '19BBY',
      'created': '2014-12-09T13:50:51.644000Z',
      'edited': '2014-12-20T21:17:56.891000Z',
      'eye_color': 'blue',
      'films': ['https://swapi.dev/api/films/1/',
                'https://swapi.dev/api/films/2/',
```

```
                'https://swapi.dev/api/films/3/',
                'https://swapi.dev/api/films/6/'],
    'gender': 'male',
    'hair_color': 'blond',
    'height': '172',
    'homeworld': 'https://swapi.dev/api/planets/1/',
    'mass': '77',
    'name': 'Luke Skywalker',
    'skin_color': 'fair',
    'species': [],
    'starships': ['https://swapi.dev/api/starships/12/',
                  'https://swapi.dev/api/starships/22/'],
    'url': 'https://swapi.dev/api/people/1/',
    'vehicles': ['https://swapi.dev/api/vehicles/14/',
                 'https://swapi.dev/api/vehicles/30/']}
```

Vamos montar, agora, uma função que seja mais completa. O nosso objetivo para esse exercício é extrair todas as pessoas armazenadas na API. Para isso, vamos investigar a URL https://swapi.dev/api/people/.

```
{
    "count": 82,
    "next": "https://swapi.dev/api/people/?page=2",
    "previous": null,
    "results": [<pessoas>]
}
```

Observe o campo `next`. Esse campo indica que há uma continuação nos dados. Isso é feito para agilizar as consultas, limitando o número de resultados por consulta.

Dessa forma, nosso script precisa consultar a página inicial, armazenar os resultados em uma lista, e continuar consultando páginas enquanto o campo `next` não for nulo.

Atenção! Esse campo `next` é específico dessa API. Nem sempre o processo de consulta é dessa forma. É sempre necessário analisar os dados antes de iniciar a extração, e ir testando à medida em que desenvolve o seu código.

```
# Vamos montar uma função que possa coletar apenas um recurso,
# ou todos os recursos caso não seja passado nenhum parâmetro.
def swapi(resources_to_extract=None):
  resources = resources_to_extract or SWAPI["resources"]

  for resource in resources:
    print(f"Collecting data from {resource}...")
    url = f"{SWAPI['main_url']}{resource}/"
    full_data = []

    while url is not None:
      print(url)
      req = requests.get(url)
      data = req.json()

      full_data.extend(data["results"])
```

```
        url = data["next"]

    salvar(f"swapi_{resource}.json", full_data, True)

    print(f"Data from {resource} collected.")
    print(f"Collected {len(full_data)} items.\n")

# Testando com apenas um recurso para agilizar...
swapi(["people"])
```

```
Collecting data from people...
https://swapi.dev/api/people/
https://swapi.dev/api/people/?page=2
https://swapi.dev/api/people/?page=3
https://swapi.dev/api/people/?page=4
https://swapi.dev/api/people/?page=5
https://swapi.dev/api/people/?page=6
https://swapi.dev/api/people/?page=7
https://swapi.dev/api/people/?page=8
https://swapi.dev/api/people/?page=9
Data from people collected.
Collected 82 items.
```

```
# Agora que funcionou, vamos rodar para todos os recursos:
swapi()
```

```
Collecting data from people...
https://swapi.dev/api/people/
https://swapi.dev/api/people/?page=2
https://swapi.dev/api/people/?page=3
https://swapi.dev/api/people/?page=4
https://swapi.dev/api/people/?page=5
https://swapi.dev/api/people/?page=6
https://swapi.dev/api/people/?page=7
https://swapi.dev/api/people/?page=8
https://swapi.dev/api/people/?page=9
Data from people collected.
Collected 82 items.

Collecting data from films...
https://swapi.dev/api/films/
Data from films collected.
Collected 6 items.

Collecting data from starships...
https://swapi.dev/api/starships/
https://swapi.dev/api/starships/?page=2
https://swapi.dev/api/starships/?page=3
https://swapi.dev/api/starships/?page=4
Data from starships collected.
Collected 36 items.

Collecting data from vehicles...
https://swapi.dev/api/vehicles/
https://swapi.dev/api/vehicles/?page=2
https://swapi.dev/api/vehicles/?page=3
https://swapi.dev/api/vehicles/?page=4
Data from vehicles collected.
```

```
Collected 39 items.

Collecting data from species...
https://swapi.dev/api/species/
https://swapi.dev/api/species/?page=2
https://swapi.dev/api/species/?page=3
https://swapi.dev/api/species/?page=4
Data from species collected.
Collected 37 items.

Collecting data from planets...
https://swapi.dev/api/planets/
https://swapi.dev/api/planets/?page=2
https://swapi.dev/api/planets/?page=3
https://swapi.dev/api/planets/?page=4
https://swapi.dev/api/planets/?page=5
https://swapi.dev/api/planets/?page=6
Data from planets collected.
Collected 60 items.
```

Aplicação 02 - NYTimes

Agora que já fizemos a extração dos dados de uma API sem autenticação, vamos ver um cenário em que os dados não estão disponíveis livremente.

Isso é bem comum em vários casos. Muitas empresas fornecem os seus dados para consulta e pesquisa, mas exigem que o usuário faça uma autenticação prévia, para controlar a forma como esses dados estão sendo utilizados e a frequência com que esses dados são extraídos.

Um outro ponto importante para a autenticação é com relação às requisições POST. Por exemplo, podemos utilizar a API do Github para criar um novo repositório ou adicionar usuários a um repositório existente. No entanto, para isso ocorrer é necessário que o usuário se identifique.

A autenticação também é usada em dados privados. Por exemplo, um banco pode oferecer uma API para que seus clientes consultem informações financeiras da sua empresa. Para evitar que qualquer um acesse todas as informações financeiras, uma autenticação é necessária.

Nessa aula vamos extrair os dados de uma API pública bem comum para pesquisas de dados, principalmente para modelos de processamento de linguagem natural: a API de notícias do New York Times.

Qualquer um pode se cadastrar e obter um *token* de autenticação. Para conseguir um, faça os seguintes passos:

- Crie uma conta em https://developer.nytimes.com/accounts/create;
- Entre em https://developer.nytimes.com e faça login;
- Entre em https://developer.nytimes.com/my-apps/new-app e crie uma app. Você pode selecionar quais APIs você quer utilizar. No nosso exemplo vamos precisar das APIs *Article Search* e *Top Stories*;

- Copie a chave que aparece em *API Keys* e guarde em um lugar seguro.

ATENÇÃO!!! Essa chave é única e não deve ser compartilhada com **ninguém**. Lembre-se que essa é a sua identidade dentro do ambiente do NYT. Se você distribuir a chave e outra pessoa acessar a API maliciosamente, você poderá ser envolvido. Portanto, **nunca disponibilize esse token em locais públicos, como no repositório do Github!** Caso queira publicar o seu código que utiliza um token, recomendo criar um arquivo de configuração que seja excluído do seu repositório. Assim você mantém o token apenas localmente. O módulo `configparser` é bem interessante para montar arquivos de configuração em Python.

Para utilizar qualquer uma das APIs disponibilizadas pelo NYT, podemos acessar a documentação em https://developer.nytimes.com/apis. Lá conseguimos identificar todos os pontos de entrada, parâmetros e resultados de cada chamada.

Vamos aplicar nossos conhecimentos criando duas funções: uma para pegar as notícias mais importantes de uma determinada categoria, usando a API *Top Stories*; e uma para pesquisar por artigos de um determinado tema e que mencione o Brasil, usando a API *Article Search*.

Para fazer as consultas, ao invés de usar a função `requests.get()`, vamos precisar usar a função `requests.request()`, que permite incluir novos parâmetros, como o modo de consulta (`GET` ou `POST`) e o token de autenticação.

Uma observação: existem várias formas de se autenticar a uma API. Cada API é desenvolvida de uma forma. É importante sempre consultar a documentação da API para identificar qual deve ser o método de autenticação. No nosso caso, podemos consultar essa informação aqui. Navegue até o final da página e você encontrará as seguintes informações:

```
Authentication Requirements
API Key Authentication

Key location: query
Parameter name: api-key
```

Dessa forma, precisamos passar o parâmetro `api-key` com a chave que criamos para ter o acesso desejado.

```
# Vou colocar os imports novamente para manter a aplicação coesa,
# mas já importamos esses módulos anteriormente nesse notebook.
import requests
from pprint import pprint

# Esse campo está em branco intencionalmente!
# Insira aqui a chave que você criou no exemplo:
NYT_KEY = ""


def nytimes_topstories(section):
  url = f"https://api.nytimes.com/svc/topstories/v2/{section}.json"
  params = {"api-key": NYT_KEY}
```

```
  req = requests.request(
      "GET",
      url,
      params=params
  )

  return req.json()


def nytimes_article(query):
    begin_date = "20210901"
    filter_query = '"glocations:("BRAZIL")"'
    page = "0"
    sort = "relevance"
    url = "https://api.nytimes.com/svc/search/v2/articlesearch.json"
    params = {
        "q": query,
        "api-key": NYT_KEY,
        "begin_date": begin_date,
        "fq": filter_query,
        "page": page,
        "sort": sort
    }

    req = requests.request(
        "GET",
        url,
        params=params
    )

    return req.json()


pprint(nytimes_topstories("science"))
pprint(nytimes_article("science"))
```

A saída de streaming foi truncada nas últimas 5000 linhas.

```
                                          'additional shot to attain protection '
                                          'against severe Covid.\xa0',
                                'copyright': 'Emily Elconin/Getty Images',
                                'format': 'thumbLarge',
                                'height': 150,
                                'subtype': 'photo',
                                'type': 'image',
                                'url': 'https://static01.nyt.com/images/2021/09/03/
                                'width': 150},
                               {'caption': 'A man receiving a third shot of the '
                                          'Pfizer-BioNTech vaccine last month '
                                          'in Southfield, Mich., after the '
                                          'F.D.A. and the C.D.C. said that some '
                                          'immunocompromised people needed an '
                                          'additional shot to attain protection '
                                          'against severe Covid.\xa0',
                                'copyright': 'Emily Elconin/Getty Images',
                                'format': 'mediumThreeByTwo210',
                                'height': 140,
                                'subtype': 'photo',
                                'type': 'image',
                                'url': 'https://static01.nyt.com/images/2021/09/03/
                                'width': 210},
```

```
                    {'caption': 'A man receiving a third shot of the '
                                'Pfizer-BioNTech vaccine last month '
                                'in Southfield, Mich., after the '
                                'F.D.A. and the C.D.C. said that some '
                                'immunocompromised people needed an '
                                'additional shot to attain protection '
                                'against severe Covid.\xa0',
                     'copyright': 'Emily Elconin/Getty Images',
                     'format': 'Normal',
                     'height': 127,
                     'subtype': 'photo',
                     'type': 'image',
                     'url': 'https://static01.nyt.com/images/2021/09/03/
                     'width': 190}],
      'org_facet': ['Centers for Disease Control and Prevention',
                    'Food and Drug Administration',
                    'Pfizer Inc',
                    'BioNTech SE',
                    'Moderna Inc',
                    'Johnson & Johnson'],
      'per_facet': ['Biden, Joseph R Jr'],
      'published_date': '2021-09-03T10:46:04-04:00',
      'section': 'us',
      'short_url': 'https://nyti.ms/2WRCrAr',
      'subsection': '',
      'title': 'Health Officials Advise White House to Scale Back '
               'Booster Plan for Now',
      'updated_date': '2021-09-04T10:20:42-04:00',
      'uri': 'nyt://article/f1af9a3c-22d9-5ff1-a654-6ae0f756f273',
      'url': 'https://www.nytimes.com/2021/09/03/us/coronavirus-booster-s
     {'abstract': 'He and a colleague identified a mechanism by which '
                  'cells communicate, a Nobel-winning breakthrough '
                  'that paved the way for disease-fighting drugs.',
      'byline': 'By Sabrina Imbler',
```


Aplicação 03 - Box Office Mojo

Para a nossa terceira aplicação, vamos sair um pouco do campo das APIs. Nesse exemplo, o problema que temos que resolver é que os dados não estão disponíveis em uma API, e sim em uma página HTML.

Nem sempre conseguimos extrair os dados de um site. No entanto, dependendo de como ele foi desenvolvido conseguimos analisar o código HTML e programaticamente extrair esses dados.

Para saber se conseguimos extrair os dados, basta clicar com o botão direito sobre o dado e clicar na opção "Inspecionar". O seu navegador vai abrir uma caixa com o código HTML da página. Nesse momento, é importante parar e analisar padrões e como cada conteúdo está inserido na página. Eles podem estar dentro de `<div>`, dentro de `<table>`, etc. Normalmente essas tags também possuem um parâmetro `class`, que podemos utilizar para estreitar nossa busca.

Para fazer essa extração precisaremos do módulo `requests` para obter o conteúdo da página, e do módulo `bs4` ou (*BeautifulSoup*) para processar os dados da página.

Nesse exemplo vamos utilizar a página de maiores bilheterias no mundo do portal Box Office Mojo. Nosso objetivo aqui não é esgotar as funcionalidades do BeautifulSoup, mas sim dar uma visão geral de como conseguiríamos resolver esse problema em particular.

O código vai ser comentado à medida que for necessário explicar alguma função ou algum parâmetro específico. Ao contrário dos outros exemplos, vamos montar o nosso código em comandos separados para ir analisando os dados à medida que fazemos a exploração.

Observação: Veja que o portal apresenta 1000 filmes, separados de 200 em 200. Dá para acessar as páginas seguintes incluindo o parâmetro `offset` na URL:

- Exibição de 201 a 400: https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?area=XWW&offset=200
- Exibição de 401 a 600: https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?area=XWW&offset=400

Esse é um truque interessante para extrair dados da web. Sempre confira como é a formação da URL, assim você consegue automatizar várias consultas!

```
import requests
from bs4 import BeautifulSoup

boxoffice = requests.get("https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/")
boxoffice.status_code
```

```
200
```

```
# Consulta bem-sucedida, vamos pegar o conteúdo.
# Observe que o resultado que aparece aqui é diferente daquele obtido com
# a inspeção da página.
# Isso acontece porque a página é gerada dinamicamente usando javascript,
# e o módulo requests não utiliza essa extração dinâmica.
boxoffice.text
```

```
'<!doctype html><html class="a-no-js" data-19ax5a9jf="dingo"><head><script>var aPag
eStart = (new Date()).getTime();</script><meta charset="utf-8"/><meta name="viewpor
t" content="width=device-width, initial-scale=1.0" />\n                <meta charse
t="utf-8" />\n            <title dir="ltr">Top Lifetime Grosses - Box Office Mojo</
title><meta content="Top Lifetime Grosses" name="title" />\n            <meta conte
nt="Box Office Mojo" property="og:site_name" />\n            <meta content="http
s://m.media-amazon.com/images/G/01/boxofficemojo/logo/mojo-logo-bg.png" property="o
g:image"/>\n            <meta name="format-detection" content="telephone=no" />\n
<link href="https://m.media-amazon.com/images/G/01/boxofficemojo/v2/favicon_CB4489
```

```
# Inspecionando a página no navegador, vimos que todo o conteúdo que queremos
# está incluído em uma tabela, ou seja, precisamos o conteúdo na tag <table>.
# Antes disso, vamos rodar o nosso parser com o BeautifulSoup
soup = BeautifulSoup(boxoffice.text, "html.parser")
```

```
# De todo aquele conteúdo, vamos extrair a nossa tabela e analisar seu conteúdo.
# Isso é interessante de ser feito para facilitar a nossa leitura.
# O retorno da função find_all é uma lista com todas as tags encontradas.
# Sabemos que essa página só tem uma tag table, portanto vou pegar o único
```

```
# elemento da lista
table = soup.find_all("table")[-1]
table
```

```
<table class="a-bordered a-horizontal-stripes a-size-base a-span12 mojo-body-table m
</th><th class="a-text-left mojo-field-type-title a-nowrap"><span title="Title">Titl
</th><th class="a-text-right mojo-field-type-money a-nowrap"><span title="Worldwide
</th><th class="a-text-right mojo-field-type-money a-nowrap"><span title="Domestic L
</th><th class="a-text-right mojo-field-type-percent a-nowrap"><span title="Domestic
</th><th class="a-text-right mojo-field-type-money a-nowrap"><span title="Foreign Li
</th><th class="a-text-right mojo-field-type-percent a-nowrap"><span title="Foreign
</th><th class="a-text-left mojo-field-type-year a-nowrap"><span title="Year">Year</
</th></tr><tr><td class="a-text-right mojo-header-column mojo-truncate mojo-field-ty
```

```
# Veja que o nosso conteúdo está dentro das tags td, mas os dados
# não estão agrupados...
# Sabendo que a tabela possui 8 colunas, podemos montar uma estrutura de
# repetição para extrair esses dados e ir alocando em uma matriz de acordo
separated_data = table.find_all("td")

# Conferindo se todos os dados estão aqui...
# São 200 linhas, com 8 colunas cada, totalizando 1600 dados.
len(separated_data)
```

```
1600
```

```
movies = []
column_size = 8

for index, single_data in enumerate(separated_data):
  if index // column_size == len(movies):
    movies.append([])

  movies[-1].extend(single_data.contents)

movies
```

```
[['1',
  <a class="a-link-normal" href="/title/tt0499549/?ref_=bo_cso_table_1">Avatar</a
  '$2,847,246,203',
  '$760,507,625',
  '26.7%',
  '$2,086,738,578',
  '73.3%',
  <a class="a-link-normal" href="/year/2009/?ref_=bo_cso_table_1">2009</a>],
 ['2',
  <a class="a-link-normal" href="/title/tt4154796/?ref_=bo_cso_table_2">Avengers:
  '$2,797,501,328',
  '$858,373,000',
  '30.7%',
  '$1,939,128,328',
  '69.3%',
  <a class="a-link-normal" href="/year/2019/?ref_=bo_cso_table_2">2019</a>],
 ['3',
  <a class="a-link-normal" href="/title/tt0120338/?ref_=bo_cso_table_3">Titanic</
  '$2,201,647,264',
  '$659,363,944',
  '30%',
```

```
        '$1,542,283,320',
        '70%',
        <a class="a-link-normal" href="/year/1997/?ref_=bo_cso_table_3">1997</a>],
       ['4',
        <a class="a-link-normal" href="/title/tt2488496/?ref_=bo_cso_table_4">Star Wars
        '$2,069,521,700',
        '$936,662,225',
        '45.3%',
        '$1,132,859,475',
        '54.7%',
        <a class="a-link-normal" href="/year/2015/?ref_=bo_cso_table_4">2015</a>],
       ['5',
        <a class="a-link-normal" href="/title/tt4154756/?ref_=bo_cso_table_5">Avengers:
        '$2,048,359,754',
        '$678,815,482',
        '33.1%',
        '$1,369,544,272',
        '66.9%',
        <a class="a-link-normal" href="/year/2018/?ref_=bo_cso_table_5">2018</a>],
       ['6',
        <a class="a-link-normal" href="/title/tt0369610/?ref_=bo_cso_table_6">Jurassic
        '$1,670,516,444',
        '$652,385,625',
        '39%',
        '$1,018,130,819',
        '61%',
        <a class="a-link-normal" href="/year/2015/?ref_=bo_cso_table_6">2015</a>],
       ['7',
        <a class="a-link-normal" href="/title/tt6105098/?ref_=bo_cso_table_7">The Lion
        '$1,667,635,327',
        '$543,638,043',
        '32.6%',
        '$1,123,997,284',
        '67.4%',
        <a class="a-link-normal" href="/year/2019/?ref_=bo_cso_table_7">2019</a>],
       ['8',
        <a class="a-link-normal" href="/title/tt0848228/?ref_=bo_cso_table_8">The Aveng
```

```
# Conseguimos coletar todos os dados que precisamos!
# Vamos fazer mais alguns passos para poder organizar melhor essa lista.
# O primeiro passo é transformar as tags <a> em conteúdo:
for movie in movies:
  movie[1] = movie[1].contents[-1]
  movie[7] = movie[7].contents[-1]

movies
```

```
      [['1',
        'Avatar',
        '$2,847,246,203',
        '$760,507,625',
        '26.7%',
        '$2,086,738,578',
        '73.3%',
        '2009'],
       ['2',
        'Avengers: Endgame',
        '$2,797,501,328',
        '$858,373,000',
        '30.7%',
```

```
        '$1,939,128,328',
        '69.3%',
        '2019'],
       ['3',
        'Titanic',
        '$2,201,647,264',
        '$659,363,944',
        '30%',
        '$1,542,283,320',
        '70%',
        '1997'],
       ['4',
        'Star Wars: Episode VII - The Force Awakens',
        '$2,069,521,700',
        '$936,662,225',
        '45.3%',
        '$1,132,859,475',
        '54.7%',
        '2015'],
       ['5',
        'Avengers: Infinity War',
        '$2,048,359,754',
        '$678,815,482',
        '33.1%',
        '$1,369,544,272',
        '66.9%',
        '2018'],
       ['6',
        'Jurassic World',
        '$1,670,516,444',
        '$652,385,625',
        '39%',
        '$1,018,130,819',
        '61%',
        '2015'],
       ['7',
        'The Lion King',
        '$1,667,635,327',
        '$543,638,043',
        '32.6%',
        '$1,123,997,284',
        '67.4%',
        '2019'],
       ['8',
        'The Avengers',
        '$1,518,815,515',
```

```
# Nosso último passo, agora, é colocar cada filme dentro de um dicionário,
# para podermos salvar como json.
json_movies = []

for movie in movies:
  json_movies.append(
    {
      "rank": movie[0],
      "title": movie[1],
      "worldwide": movie[2],
      "domestic": movie[3],
      "domestic %": movie[4],
      "foreign": movie[5],
      "foreign %": movie[6],
```

```
      "year": movie[7]
    }
  )

# Por último, vamos salvar!
salvar("boxoffice.json", json_movies, True)
```

Observe que, mesmo com a análise feita e o código funcional, o ideal é condensar todo o código desenvolvido e consolidar em uma função (ou algumas funções), de forma a permitir a reutilização do código!

Pedido da P1

Com o que vimos até agora, conseguimos desenvolver um código para extrair os dados brutos da web. A P1 vai consistir no desenvolvimento de um projeto para fazer essa extração.

Pedido

O trabalho deve conter o seguinte:

- Introdução
    - Informação básica sobre a fonte dos dados: URL principal, forma de acesso, tipo de consulta que será feita (via API ou consultando o código HTML), etc.;
    - Caso seja uma API que exija autenticação, incluir os passos para obter uma chave de acesso;
    - Objetivo da extração, ou seja, o que se espera obter ao final da extração dos dados;
- Análise
    - Indicar funções que serão analisadas na API ou estruturas e tags que serão consultadas no código HTML;
- Extração
    - Implementar o código para extração e organização mínima dos dados;
    - Incluir as etapas intermediárias! É importante ver o processo de entendimento da base de dados;
    - Salvar os dados em um arquivo .json.

Avaliação

O trabalho será avaliado segundo os seguintes critérios:

- Complexidade da fonte de dados: quanto menos complexa a fonte, mais exigente será a avaliação dos outros critérios;
- Qualidade do código: o código está seguindo as regras discutidas em aula (nomeação de variáveis, espaçamentos, formato de funções, etc.)?
- Eficiência do código: as estruturas utilizadas são eficientes em termos de uso de memória ou processamento (ou seja, o código não apresenta redundâncias ou soluções

demoradas)?
- Eficácia do código: o projeto entrega o que foi pedido no início?
- Completude do projeto: o projeto inclui todas as informações solicitadas no pedido? O processo de análise e execução do código está detalhado?

Informações gerais:

- Prazo: 12/10/2021
- Grupos de no mínimo 2 e no máximo 3 pessoas
- Implementar utilizando um notebook (jupyter, Google Colab ou outros)
- Subir na tarefa do Teams o arquivo .ipynb com todo o processo
- Não se esqueçam de detalhar o processo! Utilizem as células de texto para fazer as explicações que considerarem necessárias

Dicas de fontes de dados

Seguem abaixo algumas dicas de fontes de dados. Não é necessário usar uma dessas fontes, estou colocando aqui apenas para inspiração. Use a criatividade!

- Dados com Campeonato Brasileiro no Globo Esporte
- Análise de repositórios do Github
- Composições de diferentes carteiras da XP Investimentos
- Dados das empresas listadas na Fortune 500
- Algumas dicas de APIs públicas: https://medium.com/@iara.ribeiro/apis-legais-para-seu-pr%C3%B3ximo-projeto-c9d177bad738