escola britânica de artes criativas & tecnologia

Analista de Dados

Coleta de Dados II

Módulo | Análise de Dados: Coleta de Dados II

Caderno de Aula

Professor André Perez

Tópicos

1. Web Crawling;
2. Web Scraping;
3. Web API.

Aulas

0. Estruturas de dados

- **Não estruturado**: texto, imagem, aúdio, etc.
- **Semi estruturado**: html, json, etc.
- **Estruturado**: tabelas, planilhas, etc.

1. Web Crawling

1.1. HTTP

O HTTP (hypertext transfer protocol) é um protocolo de transferência de hipertexto (texto, imagens, vídeos, etc.). É o protocolo padrão de transferência de informação pela internet:

```
http://www.google.com/
```

Cliente / Servidor é a arquitetura da internet. Nela um **cliente** (navegador web, código Python, etc.) utiliza um **método** HTTP para interagir com um servidor (requisitar dados, enviar dados, etc.). O **servidor**, por sua vez, envia uma resposta para o cliente com um **código de retorno** indicando se a interação ocorreu com sucesso.

Métodos são as operações que podemos realizar com o protocolo para interagir com um

servidor, você pode encontrar uma lista completa neste link. Os mais importantes são:

- **GET**: Requisitar dados, (acessar uma página web, carregar o feed do instagram, etc.);
- **POST**: Enviar dados (login, cadastro, mensagem whatsapp, tweet do Twitter).

Códigos de retorno são as números de 0 a 1000 que recebemos como resposta do servidor ao realizar uma operação qualquer, você pode encontrar uma lista completa neste link. Os mais importantes são:

- Entre **200** e **299**: Sucesso;
- Entre **400** e **499**: Erro do cliente;
- Entre **500** e **599**: Erro do servidor.

Código **200** (sucesso) é o mais comum e o **404** (não encontrado) o mais famoso!

1.2. Pacote requests

Pacote Python para interagir com a web através do protocolo HTTP. A documentação pode ser encontrada neste (link).

In [ ]:
```
import requests

print(requests.__version__)
```

Método:

In [ ]:
```
resposta = requests.get('http://www.google.com')
```

Código de retorno:

In [ ]:
```
print(resposta.status_code)
```

Em geral, é bom verificar o código:

In [ ]:
```
import requests
from requests.exceptions import HTTPError

conteudo = None
URL = 'http://www.google.com/andre'

try:
  resposta = requests.get(URL)
  resposta.raise_for_status()
except HTTPError as exc:
  print(exc)
else:
  conteudo = resposta.text

print(conteudo)
```

1.3. Web Crawl

Aplicação que interage com a web de forma automatizada, também conhecido como *web spider* ou *bot*. O pacote **requests** funciona bem para interaçõces simples (apenas metodos

HTTP) já os pacotes **selenium** (documentação) e **scrapy** (documentação) permitem que você navegue pela internet.

> **Web crawling** é um tema que **requer muito cuidado**, em geral você precisa de autorização do website para acessa-lo automaticamente. Empresas que usam a tecnologia (já trabalhei em uma) possuem departamento jurídicos dedicados ao assunto.

> A maioria dos sites fornece um arquivo chamado `robots.txt` informando como um web crawler pode interagir com a página.

Exemplo: Função de web crawler.

In [ ]:
```
import requests
from requests.exceptions import HTTPError

def crawl_website(url: str) -> str:

  try:
    resposta = requests.get(url)
    resposta.raise_for_status()
  except HTTPError as exc:
    print(exc)
  else:
    return resposta.text
```

Exemplo: Wikipedia robots.txt

In [ ]:
```
URL = 'https://en.wikipedia.org/robots.txt'

conteudo = crawl_website(url=URL)
print(conteudo)
```

Exemplo: Extrair a página da Wikipédia sobre web crawlers.

In [ ]:
```
URL = 'https://en.wikipedia.org/wiki/Web_crawler'

conteudo = crawl_website(url=URL)
print(conteudo)
```

2. Web Scraping

2.1. Formato HTML

Um arquivo **texto** semi-estruturado, organizado por **tags**, você pode encontrar uma lista completa neste link.

Exemplo: Arquivo `lotr.html`.

In [ ]:
```
%%html
<html>
  <head>
    <!-- metadata -->
  </head>
  <body>
```

```
    <h3>Senhor dos Anéis</h3>
    <p>Filmes:</p>
    <ul>
      <li><b>2001:</b> O Senhor dos Anéis: A Sociedade do Anel</li>
      <li><b>2002:</b> O Senhor dos Anéis: As Duas Torres</li>
      <li><b>2003:</b> O Senhor dos Anéis: O Retorno do Rei</li>
    </ul>
  </body>
</html>
```

In [ ]:
```
%%writefile lotr.html
<html>
  <head>
    <!-- metadata -->
  </head>
  <body>
    <h3>Senhor dos Anéis</h3>
    <p>Filmes:</p>
    <ol>
      <li><b>2001:</b> O Senhor dos Anéis: A Sociedade do Anel</li>
      <li><b>2002:</b> O Senhor dos Anéis: As Duas Torres</li>
      <li><b>2003:</b> O Senhor dos Anéis: O Retorno do Rei</li>
    </ol>
  </body>
</html>
```

2.2. Pacote beautifulsoup4

Pacote Python para extrair informações de arquivos HTML. A documentação pode ser encontrada neste (link).

Exemplo: Extrair os filmes e anos do arquivo `lotr.html` em um dicionário.

In [ ]:
```
from bs4 import BeautifulSoup

pagina = BeautifulSoup(open('lotr.html', mode='r'), 'html.parser')
```

In [ ]:
```
filmes_li = pagina.find_all('li')
print(filmes_li)
```

In [ ]:
```
print(list(set(map(lambda filme_li: type(filme_li), filmes_li))))
```

In [ ]:
```
filmes = []

for filme_li in filmes_li:
  filme = filme_li.get_text()
  ano = int(filme.split(sep=':')[0].strip())
  titulo = ':'.join(filme.split(sep=':')[1:]).strip()
  filmes.append({'ano': ano, 'titulo': titulo})

for filme in filmes:
  print(filme)
```

2.3. Web Scrape

Aplicação que extrai conteúdo de páginas web de forma automatizada, em geral é aplicado

após o processo de web crawling.

Exemplo: Extrair todo o texto da página da Wikipédia sobre web crawlers e contar a ocorrência da palavra `crawler`.

In [ ]:
```
URL = 'https://en.wikipedia.org/wiki/Web_crawler'

conteudo = crawl_website(url=URL)
with open(file='wiki.html', mode='w', encoding='utf8') as arquivo:
  arquivo.write(conteudo)
```

In [ ]:
```
from bs4 import BeautifulSoup

pagina = BeautifulSoup(open('wiki.html', mode='r'), 'html.parser')
```

In [ ]:
```
texto = pagina.get_text()
print(texto)
```

In [ ]:
```
import re

ocorrencias = len(re.findall('crawler', texto, re.IGNORECASE))
print(ocorrencias)
```

3. Web API

3.1. Formato JSON

Um arquivo semi-estruturado, organizado por **chave/valor**, é equivalente a um dicionário Python.

Arquivo JSON: lotr.json

In [ ]:
```
%%writefile lotr.json
[
    {
        "ano": 2001,
        "titulo": "O Senhor dos Anéis: A Sociedade do Anel"
    },
    {
        "ano": 2002,
        "titulo": "O Senhor dos Anéis: As Duas Torres"
    },
    {
        "ano": 2003,
        "titulo": "O Senhor dos Anéis: O Retorno do Rei"
    }
]
```

2.2. Pacote json

Pacote nativo do Python para interagir com dados no formato json.

Exemplo: Arquivo JSON para dicionario Python.

In [ ]:
```
import json
```

```
data = json.load(open(file='lotr.json', mode='r'))
print(data)
```

Exemplo: Dicionario Python para formato JSON.

In [ ]:
```
import json

data_json = json.dumps(data, indent=2, ensure_ascii=False)
print(data_json)
```

2.3. Web API

Uma API (application programming interface) é uma interface de comunicação com uma aplicação no formato cliente/servidor. Uma REST API é uma API que segue o padrão HTTP e transfere dados (em geral) no formato JSON. APIs fechadas são pagas e exigem autenticação (via método HTTP POST). APIs abertas são gratuitas e podem exigir autenticação.

Alguns exemplos:

- Twitter;
- Governo Federal.

Exemplo: Extrair a taxa CDI da API da B3.

In [ ]:
```
import requests
from requests.exceptions import HTTPError

conteudo = None
URL = 'https://www2.cetip.com.br/ConsultarTaxaDi/ConsultarTaxaDICetip.aspx'

try:
  resposta = requests.get(URL)
  resposta.raise_for_status()
except HTTPError as exc:
  print(exc)
else:
  conteudo = resposta.text

print(conteudo)
print(type(conteudo))
```

In [ ]:
```
import json

data = json.loads(conteudo)
print(data)
```

In [ ]:
```
cdi = float(data['taxa'].replace(',', '.'))
print(cdi)
```