Python para Ciência de Dados
Data Scraping

1. Introdução

Muitos dos principais sites e redes sociais fornecem APIs para acessar seus dados de uma forma mais estruturada: Twitter, Facebook, Google, Twitter, etc., que quase sempre a melhor abordagem para se fazer scraping. Entretanto, isso não acontece com todos os sites.

Nesses casos, precisamos fazer web scraping, que é uma técnica para extraçaõ de informações de sites web, transformando dados não estruturados em dados estruturados.

Portanto, além dos dados de arquivos CSV, também podemos, e precisamos, utilizar dados de partes de uma página web, utlizando web scraping, que é uma técnica para extraçaõ de informações de sites web, transformando dados não estruturados em dados estruturados. Para isso, utilizamos uma biblioteca BeautifulSoup, junto com o Pandas e o Requests.

2. Relembrando

Sempre que acessamos um site, o navegador faz uma solicitação chamada de solicitação GET, que recebe arquivos (HTML, CSS, JS, imagens, ...) do servidor, e renderizar a referida página. Lembrando que em html as tags são aninhadas e podem aparecer dentro de outras tags.

De acordo com a posição em relação à outra tag, as mesmas podem ser classificadas como:

- child (filha): uma tag dentro de outra tag.
- parent (pai): uma tag que tem outras tags dentro.
- sibling (irmão): uma tag que está aninhada dentro do mesmo pai que outra tag.

Exemplo:

```
<html><!--tag pai-->
  <head><!--tag filha-->
  </head>
  <body>
    <p><!--tag irmã-->
      texto
    </p>
    <p><!--tag irmã-->
      texto
    </p>
  </body>
</html>
```

As tags mais comuns são as seguintes:

- a: links que redirecionam para uma outra página.
- div: divide uma área na página.
- table: cria uma tabela.

- form: cria um formulário.

Essas tags podem ter atributos, que podem ser bastantes úteis quando realziamos scraping, por exemplo:

- class: usado para definir estilos iguais para elementos com o mesmo nome de classe.
- id: especifica um ID exclusivo para um elemento HTML.

3. A biblioteca requests

Agora vamos aprender a utilizar o Requests, uma biblioteca HTTP para Python simples e elegante, para baixar a página. Assim, não teremos a necessidade para adicionar query string nas suas URLs manualmente, ou de codificar seus dados de formulário POST.

Se quiser mais detalhes sobre essa biblioteca, pode consultar:

https://requests.readthedocs.io/pt_BR/latest/index.html

Exemplo:

```
import requests as rq

#definindo a url a ser baixada. Só por diversão...
url = "http://pudim.com.br/"

#usando a função get para obter os dados da página
pagina = rq.get(url)

#os dados da página
print(pagina.text)
```

```
<html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Pudim</title>
    <link rel="stylesheet" href="estilo.css">
</head>
<body>
<div>
    <div class="container">
        <div class="image">
            <img src="pudim.jpg" alt="">
        </div>
        <div class="email">
            <a href="mailto:pudim@pudim.com.br">pudim@pudim.com.br</a>
        </div>
    </div>
</div>
<script>
    (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
                (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
            m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
    })(window,document,'script','//www.google-analytics.com/analytics.js','ga');

    ga('create', 'UA-28861757-1', 'auto');
    ga('send', 'pageview');

</script>
</body>
</html>
```

3.1. Algumas das principais funções adicionais do requests

encoding	Retorna a codificação usada para decodificar r.text
headers	Retorna um dicionário de cabeçalhos de resposta
json()	Retorna um objeto JSON do resultado (se o resultado foi gravado no formato JSON, caso contrário, isso gera um erro)
status_code	Retorna um número que indica o status (200 está OK, 404 não foi encontrado)
text	Retorna o conteúdo da resposta, em unicode
url	Retorna o URL da resposta

4. Beautiful Soup

Também faremos a extração de dados de arquivos HTML e XML. Para isso, vamos utilizar o Beautiful Soup, uma biblioteca Python, que trabalha com seu analisador favorito para fornecer maneiras idiomáticas de navegar, pesquisar e modificar a árvore de análise. Geralmente, economiza horas ou dias de trabalho para os programadores.

Se quiser mais detalhes sobre essa biblioteca, pode consultar:

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

4.1. Analisando um documento

Para analisar um documento, passe-o para o construtor BeautifulSoup. Você pode passar uma string ou um arquivo aberto. Para isso, vamos:

1) importar a biblioteca BeautifulSoup;
2) criar uma instância da classe BeautifulSoup para analisar o documento;
3) usar o método prettify, do BeautifulSoup, para exibir o conteúdo HTML da página.

```
import requests as rq
from bs4 import BeautifulSoup as bs

#define a url a ser analisada
url = "https://www.bondfaro.com.br/celular-e-smartphone.html?mn=1"

#obtém a página, sem fazer a verificação do certificado
pagina = rq.get(url,verify = False)

#analisa a página obtida
analisador_pagina = bs(pagina.content, 'html.parser')

#mostra a página do analisador
print(analisador_pagina.prettify()[:1000])
```

```
C:\Anaconda3\lib\site-packages\urllib3\connectionpool.py:858: InsecureRequestWarning: Unverified HTTP
ding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/adva
gs
  InsecureRequestWarning)
<!DOCTYPE html>
<!--[if IE 7]>
<html lang="en" class="no-js ie7"/>
<![endif]-->
<!--[if IE 8]>
<html lang="en" class="no-js ie8"/>
<![endif]-->
<!--[if IE 9]>
```

4.2. Buscando uma determinada tag

Podemos extrai da página uma única tag ou todas as ocorrências de uma mesma tag.

```
import requests as rq
from bs4 import BeautifulSoup as bs
url = "https://www.bondfaro.com.br/celular-e-smartphone.html?mn=1"
pagina = rq.get(url,verify = False)
analisador_pagina = bs(pagina.content, 'html.parser')

#buscando a primeira tag a, que define um hiperlink. Usado para vincular de uma página para outra.
analisador_pagina.find('a')

#buscando todas as tags a
analisador_pagina.find_all('a')
```

```
orias</a>,
 <a class="item" href="/celular-e-smartphone.html?mn=1" title="Celulares">Celulares</a>,
 <a class="item" href="/tv.html?mn=1" title="TV">TV</a>,
 <a class="item" href="/eletronicos.html?mn=1" title="Eletrônicos">Eletrônicos</a>,
 <a class="item" href="/informatica.html?mn=1" title="Informática">Informática</a>,
 <a class="item" href="/eletrodomesticos.html?mn=1" title="Eletrodomésticos">Eletrodomésticos</a>,
 <a class="item" href="/casa-e-decoracao.html?mn=1" title="Casa e Decoração">Casa e Decoração</a>,
 <a class="item" href="/esporte-e-lazer.html?mn=1" title="Esporte e Lazer">Esporte e Lazer</a>,
 <a class="item" href="/games.html?mn=1" title="Games">Games</a>,
 <a class="item" href="/construcao-e-ferramentas.html?mn=1" title="ConstruÃ§Ã£o">ConstruÃ§Ã£o</a>,
 <a class="item" href="/categorias?mn=1">Mais Categorias</a>,
 <a class="clear-term" href="#" id="clear-term"></a>,
 <a class="item" data-ga-action="celulares" data-ga-ctg="Menu_home" data-ga-lbl="celulares" href="/ce
mn=1" title="Celulares">Celulares</a>,
 <a data-ga-action="Celular e Smartphone" data-ga-ctg="Menu_home" data-ga-lbl="celulares" href="/celu
=1">Celular e Smartphone</a>,
 <a data-ga-action="iPhone" data-ga-ctg="Menu_home" data-ga-lbl="celulares" href="/celular-e-smartpho
```

4.3. Buscando tags com atributos específicos

Mas são muitas tags mesmo assim. Então vamos analisar e buscar tags com classes específicas. Entretanto, "class", é uma palavra reservada no Python. Usar class como argumento de palavra-chave fornecerá um erro de sintaxe. No Beautiful Soup 4.1.2, você pode pesquisar por classe CSS usando o argumento de palavra-chave class_.

```
import requests as rq
from bs4 import BeautifulSoup as bs
url = "https://www.bondfaro.com.br/celular-e-smartphone.html?mn=1"
pagina = rq.get(url,verify = False)
analisador_pagina = bs(pagina.content, 'html.parser')

#analisando a página, vemos que a classe price traz os dados dos celulares
dados_celulares = analisador_pagina.find_all(class_='price')
for x in dados_celulares:
    print(x.prettify())
```

```
<p class="price">
 <a href="/celular-e-smartphone--smartphone-xiaomi-redmi-note-7-64gb.html">
  a partir de
  <br/>
  <span class="value">
   R$ 939,00
  </span>
  em
  <span class="sellers">
   9 lojas
  </span>
 </a>
</p>
```

4.4. Mais detalhes do site analisado

Se utilizarmos algums funções específicas, podemos encontrar facilmente informações úteis em cada site analisado. Imagine o seguinte:

- Neste site do bondfaro, estamos interessados em saber o preço de diversos modelos de celulares da marca Samsung;
- Para isso, precisamos inspecionar o código, para saber os os preços são apresentados no HTML;
- Neste caso específico, os dados que nós queremos estão armazenados nas tags <li> com atributos da classe chamdo “prd-new”, que possui estas tags filhas:
    - A tag p, com atributo class = ‘title’, guarda o modelo do celular;
    - A tag p span com o atributo ‘value’ guarda o preço do celular;
    - A tag p span com o atributo ‘sellers’ guarda os vendedores do celular.

Assim podemos buscar exatamente por esses elementos usando a função find_all.

```
import requests as rq
import pandas as pd
import re
from bs4 import BeautifulSoup as bs
from IPython.display import display
#desabilitando os avisos
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

url = 'https://www.bondfaro.com.br/celular-e-smartphone.html?kw=samsung&page=1'
dadosPagina =  rq.get(url,verify=False)
analisador =  bs(dadosPagina.text,'html.parser')

celulares = []
dados = analisador.find_all('li',class_=re.compile("prd-new"))
for x in dados:
    modelos = x.find('p',class_='title').text
    precos = x.find('span',class_='value').text
    lojas = x.find('span',class_='sellers').text
    celulares.append(dict(zip(['Modelo','Preço','Lojas'],
                              [modelos,precos,lojas])))
display(celulares)
```

```
[{'Modelo': 'Smartphone Samsung Galaxy A30 SM-A305GT 64GB',
  'Preço': 'R$ 969,00',
  'Lojas': '17 lojas'},
 {'Modelo': 'Smartphone Samsung Galaxy A70 SM-A705MN 128GB',
  'Preço': 'R$ 1.754,10',
```

Ainda utilizamos o seguinte:

- **re.compile()**: compilar um padrão de expressão regular para um objeto expressão regular , que pode ser usado para combinar com os seus match(), search() e outros métodos;
- **text:** que devolve o o resultado da busca no formato texto, sem as tags;
- **append :** adiciona seu argumento como um único elemento ao final de uma lista. O comprimento da lista aumenta em um.
- **dict():**cria um dicionário, que é um dicionário é uma coleção desordenada, mutável e indexada;

- **zip():** retorna um objeto zip, que é um iterador de tuplas em que o primeiro item em cada iterador passado é emparelhado e, em seguida, o segundo item em cada iterador passado é emparelhado etc.

4.5. Construindo um data frame usando os dados coletados

O resultado do código acima, é uma lista de dicionários, cada um com as informações referentes a um modelo específico de celular Samsung.

Vamos facilitar a nossa vida, vamos criar um data frame com o resultado, de onde podemos obter estatísticas sobre a variação de preços, disponibilidade de lojas etc..

Usamos também, Estrutura de dados tabular bidimensional, mutável em tamanho e potencialmente heterogênea, com eixos rotulados (linhas e colunas). As operações aritméticas são alinhadas nos rótulos de linha e coluna.

```
import requests as rq
import pandas as pd
import re
from bs4 import BeautifulSoup as bs
from IPython.display import display
#desabilitando os avisos
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

url = 'https://www.bondfaro.com.br/celular-e-smartphone.html?kw=samsung&page=1'
dadosPagina =  rq.get(url,verify=False)
analisador =  bs(dadosPagina.text,'html.parser')

celulares = []
dados = analisador.find_all('li',class_=re.compile("prd-new"))
for x in dados:
    modelos = x.find('p',class_='title').text
    precos = x.find('span',class_='value').text
    lojas = re.sub(' lojas','',x.find('span',class_='sellers').text)
    celulares.append(dict(zip(['Modelo','Preço','Lojas Disponíveis'],
                              [modelos,precos,lojas])))

dataCelulares = pd.DataFrame(celulares,columns=['Modelo','Preço','Lojas Disponíveis'])
display(dataCelulares)
```

	Modelo	Preço	Lojas Disponíveis
0	Smartphone Samsung Galaxy A30 SM-A305GT 64GB	R$ 987,00	17
1	Smartphone Samsung Galaxy A10 SM-A105M 32GB	R$ 658,00	17
2	Smartphone Samsung Galaxy A50 SM-A505GT 64GB	R$ 1.329,05	13
3	Smartphone Samsung Galaxy A70 SM-A705MN 128GB	R$ 1.749,00	15