

# Resumo do Capítulo: Extraindo Dados de Recursos Online

## O que é a Mineração de Dados na Web (Web Mining)?

Às vezes você pode não ter recebido dados suficientes para conduzir uma análise profunda. Nesses casos, você precisa fazer mais pesquisas. Isso permite que você tenha em consideração mais fatores, identifique mais padrões e tire conclusões inesperadas. Os analistas podem enriquecer seus dados complementando-os com os dados da internet. Primeiro, eles encontram recursos que podem ser relevantes e então recuperam todos os dados necessários. Esse processo é chamado de **Mineração de Dados na Web**

## Algumas Coisas que um Analista Deve Saber sobre a Internet. Navegadores. HTML. HTTP

A internet é uma rede de computadores que trocam dados. Ela possui regras para *troca e representação de informação na internet (levando em consideração a forma como os computadores as exibem)*. Para que isso aconteça, várias coisas foram inventadas:

1. Uma linguagem para criar documentos: **HTML**
2. Aplicativos para visualizar esses documentos: **navegadores**
3. Regras gerais para transferência de documentos: **HTTP**

## Protocolos de transferência

Para que o processo de troca de dados funcione é necessário que haja certas regras que regem como um computador envia dados para outro. A troca de dados na internet é baseada no princípio de "requisição-resposta": um navegador gera uma requisição, então o servidor analisa-a e envia uma resposta. As regras de formulação das

requisições e respostas são determinadas pelo que conhecemos como um protocolo de transferência - nesse caso, HTTP.

A maioria dos sites hoje usa um protocolo de transferência de dados com segurança aprimorada chamado **HTTPS**. Esse protocolo garante que toda a comunicação entre seu navegador e um site é criptografada.

Quando você acessa um site, seu navegador envia uma requisição HTTP para o servidor. O servidor, por sua vez, formula uma resposta: o código HTML para a página relevante. Uma requisição gerada por um navegador pode incluir o seguinte:

- O método HTTP: determina a operação que precisa ser realizada. Existem vários métodos, os mais populares são GET e POST. O primeiro solicita dados do servidor, enquanto o segundo envia-os.
- Caminho: o segmento do endereço após o nome do site (em *example.com/hello* o caminho é */hello*).
- A versão do protocolo HTTP usado para enviar a requisição (por exemplo, HTTP/1.1).
- Cabeçalhos da requisição, que são usados para enviar informação adicional ao servidor.
- Corpo da requisição. Por exemplo, o corpo de uma requisição POST são os dados enviados. Nem todas as requisições têm um corpo.

A resposta pode incluir:

- A versão do HTTP.
- O código e a mensagem da resposta (por exemplo, "200 OK" se tudo correr bem, ou "404 Not Found" se o caminho requisitado não puder ser encontrado).
- Cabeçalhos que contêm informação adicional para o navegador.
- O corpo da resposta (por exemplo, quando você abrir o site, você verá o código HTML para esta página no corpo da resposta).

## Introdução a HTML

Para exportar uma lista de bens do site de uma loja online, primeiro você precisa obter o código da página e seu conteúdo.

Com HTML, cada objeto na página deve ser marcado para ser exibido corretamente. Essa marcação envolve colocar blocos de informação dentro de comandos chamados "tags". Essas tags dizem aos navegadores como exibir a informação dentro delas.

Um elemento HTML é composto *de tags* e o *conteúdo* dentro delas. Uma tag HTML consiste de um nome cercado de parênteses angulares (< e >). Um elemento começa com uma *tag de abertura* com o nome da tag e termina com uma *tag de fechamento* com uma barra e o nome da tag. O elemento é referido pelo nome da tag.

Aqui está a estrutura típica de uma página HTML:

## 1. <html> ... </html>

A tag <html> apresenta cada documento HTML e indica seu início, enquanto </html> marca seu fim. O <head> e o <body> do documento HTML são encontrados entre essas tags.

## 2. <head> ... </head>

Estas tags marcam o cabeçalho do documento. As tags que declaram o título do documento (<title>) e a metainformação (<meta>) são colocadas dentro destas tags.

## 3. <body> ... </body>

A tag <body> marca o início do corpo de uma página HTML. Todos os conteúdos da página (cabeçalhos, parágrafos de texto, tabelas, imagens) são colocados dentro do corpo.

Para tornar a marcação mais clara, os desenvolvedores deixam comentários dentro de tags especiais <!-- --> no código da página. Isso ajuda muito os analistas, e nós também vamos deixar comentários no código dos nossos exemplos.

Os analistas frequentemente fazem análise sintática de tabelas. Essas são geralmente colocadas em elementos do tipo *tabela*, entre as tags <table> e </table>. A tag de abertura <table> marca o início da tabela, enquanto a tag de fechamento </table> marca seu final. Dentro deste elemento, os conteúdos da tabela são divididos em linhas pelas tags <tr> (table row, linha de tabela), e as linhas, por sua vez, são

divididas em células pelas tags `<td>` (table data, dados da tabela). A primeira linha geralmente contém os cabeçalhos das colunas, em vez de células comuns. Eles são colocados entre as tags `<th>` (table headings, cabeçalhos da tabela).

Texto é frequentemente colocado dentro de um elemento *p* (parágrafo). O início do parágrafo é marcado com a tag `<p>` e o final - com a tag `</p>`.

É muito comum a tag de bloco `<div>` (divisão), que pode agrupar vários elementos. *div* é muito útil porque pode incorporar qualquer número de elementos, até mesmo de tipos diferentes (por exemplo, um cabeçalho com uma imagem e um par de parágrafos de texto), e atribuir a eles comportamento ou características comuns.

Você também pode colocar **atributos** dentro das tags para fornecer mais informação sobre como o elemento deve se comportar. Diferentes tipos de informação exigem diferentes atributos.

O nome do atributo informa ao navegador a que característica se refere o atributo, enquanto o valor especifica o que deve acontecer com essa característica. Na maioria das vezes, você precisará trabalhar com os atributos `id` e `class`. O atributo *id* fornece um identificador único para um elemento. O valor do atributo *class* é um nome que pode ser compartilhado entre vários elementos, assim como vários membros de uma família podem compartilhar o mesmo sobrenome.

## Ferramentas de desenvolvedor

Todo navegador moderno tem uma barra de ferramentas para **desenvolvedores web**, um canivete suíço para qualquer desenvolvedor. Aqui você pode dar uma olhada no código da página inteira ou de um elemento particular, ver o estilo de cada elemento da página e até mesmo alterar a visualização deles no seu computador. Você pode acessá-lo pressionando Control+Shift+i.

## Sua Primeira Requisição GET

Para obter dados do servidor, nós usaremos o método `get()`, e para enviar requisições HTTP, precisamos da biblioteca **Requests**. Importamos a biblioteca:

```
import requests
```

O método *get()* funciona como um navegador. Vamos passar o link como um argumento para ele . O método enviará uma requisição GET para o servidor, processará os dados que receber do servidor e retornará uma **resposta**, um objeto que contém a resposta do servidor para a requisição.

```
req = requests.get(URL) # salvando o objeto de resposta como a variável req
```

Um objeto *resposta* contém a resposta do servidor: o código de status, o conteúdo da requisição e o código da própria página HTML. Os atributos de objetos *resposta* permitem obter do servidor apenas dados relevantes. Por exemplo, um objeto *resposta* com o atributo *text* retornará apenas o conteúdo textual da requisição:




```
print(req.text) # o nome do atributo é colocado depois do objeto de resposta e separado de  
le por um ponto
```

O atributo *status\_code* informa se o servidor respondeu ou se ocorreu um erro.

```
print(req.status_code)
```

Infelizmente, nem todas as requisições retornaram com dados. Às vezes, as requisições retornam erros; cada erro tem um código especial dependendo do seu tipo. Aqui estão os erros mais comuns:

### Códigos de erro

 Error code	 Name	 Implication
<u>200</u>	OK	Tudo está ótimo
<u>302</u>	Found	A página foi movida
<u>400</u>	Bad Request	Erro na sintaxe da requisição
<u>404</u>	Not Found	A página não pode ser encontrada

Aa Error code	≡ Name	≡ Implication
<u>500</u>	Internal Server Error	Erro por parte do servidor
<u>502</u>	Bad Gateway	Erro no intercâmbio de dados entre servidores
<u>503</u>	Server Unavailable	O servidor está temporariamente incapaz de processar requisições

## Expressões Regulares

Para pesquisar strings em textos grandes você precisará de uma ferramenta poderosa, as expressões regulares. Uma **expressão regular** é uma regra para pesquisar substrings (fragmentos de texto dentro de strings). É possível criar regras complexas para que uma expressão regular retorne várias substrings.

Para começar a trabalhar com expressões regulares em Python, precisamos importar o módulo `re` (ou seja, regular expressions). Seguem-se duas etapas.

Primeiro criamos o padrão da expressão regular. É um algoritmo para descrever o que deve ser procurado no texto (por exemplo, todas as letras maiúsculas).

Depois esse padrão é passado para os métodos especiais do módulo `re`. Esses métodos procuram, substituem e removem símbolos. Em outras palavras, o padrão identifica o que deve ser procurado e como isso deve ser efetuado, enquanto o método define o que vai ser feito com as ocorrências encontradas.

A seguinte tabela contém os padrões mais simples de expressões regulares. Você pode criar expressões regulares mais complexas combinando esses padrões.

### Sintaxe de expressão regular

Aa Expressões Regulares	≡ Descrição	≡ Exemplo	≡ Explicação
[ ]	Caractere único contido entre colchetes	[a-]	a ou -
[^...]	Negação	[^a]	qualquer caractere exceto «a»

Aa Expressões Regulares	Descrição	Exemplo	Explicação
<code>[0-9]</code>	Intervalo	<code>[0-9]</code>	intervalo: qualquer dígito de 0 a 9
<code>.</code>	Qualquer caractere único, exceto uma nova linha	<code>a.</code>	<code>as</code> , <code>a1</code> , <code>a_</code>
<code>\d</code> (see <code>[0-9]</code> )	Qualquer dígito	<code>a\d</code>   <code>a[0-9]</code>	<code>a1</code> , <code>a2</code> , <code>a3</code>
<code>\w</code>	Qualquer letra, dígito ou <code>_</code>	<code>a\w</code>	<code>a_</code> , <code>a1</code> , <code>ab</code>
<code>[A-z]</code>	Qualquer letra latina	<code>a[A-z]</code>	<code>ab</code>
<code>[A-Я]</code>	Qualquer letra cirílica	<code>a[A-Я]</code>	<code>ая</code>
<code>?</code>	0 ou 1 entrada	<code>a?</code>	<code>a</code> ou nenhum
<code>+</code>	1 ou mais entradas	<code>a+</code>	<code>a</code> ou <code>aa</code> , ou <code>aaa</code>
<code>*</code>	0 e mais entradas	<code>a*</code>	nada ou <code>a</code> , ou <code>aa</code>
<code>^</code>	Início da string	<code>^a</code>	<code>a1234</code> , <code>abcd</code>
<code>\$</code>	Fim da string	<code>a\$</code>	<code>1a</code> , <code>ba</code>

As tarefas mais comuns dos analistas incluem:

- encontrar uma substring dentro de uma string
- dividir strings em substrings
- substituir partes de uma string por outras strings

Para completar essas tarefas, você precisará dos seguintes métodos do módulo `re`:

1. **`search(pattern, string)`** procura um `pattern` em uma `string`. Embora `search()` percorra toda a string para encontrar o padrão, ele retorna apenas a primeira substring que encontra:

```
import re
print(re.search(pattern, string))
```

O método `search()` retorna um objeto do tipo **`match`** (correspondência).

O parâmetro `span` define um intervalo de índices correspondentes ao padrão. O

parâmetro `match` indica o valor da própria substring.

Se não precisamos de qualquer informação sobre o intervalo, podemos retornar apenas a substring usando o método `group()`:

```
import re
print(re.search(pattern, string).group())
```

2. **`split(pattern, string)`** uma `string` em pontos onde o `pattern` aparece.

```
import re
print(re.split(pattern, string))
```

A string é dividida sempre que o padrão seja encontrado. O número de divisões pode ser controlado por meio do parâmetro **`maxsplit`** do método `split()`.

```
import re
print(re.split(pattern, string, maxsplit = num_split))
```

3. **`sub(pattern, repl, string)`** procura o `pattern` substring dentro de uma `string` e o substitui pela substring em `repl` (ou seja, replace).

```
import re
print(re.sub(pattern, repl, string))
```

4. **`findall(pattern, string)`** retorna uma lista de todas as substrings em uma `string` que corresponde ao `pattern`. Compare-o com o método `search()`, ue retorna apenas a primeira substring.

```
import re
print(re.findall(pattern, string))
```



**findall()** é um método particularmente útil porque permite determinar o número de substrings recorrentes em uma string com a função `len()`:

```
import re
print(len(re.findall(pattern, string)))
```

## Análise sintática de HTML

Extrair manualmente valores de dados puros de uma string contendo o código de uma página pode ser difícil. Para resolver isso, precisaremos da biblioteca **BeautifulSoup**. Os métodos da biblioteca BeautifulSoup transformam um arquivo HTML em uma estrutura de árvore. Em seguida, o conteúdo necessário pode ser encontrado por tags e atributos.

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(req.text, 'lxml')
```

O primeiro argumento apresenta dados que formarão uma estrutura de árvore. O segundo argumento é um analisador sintático. Ele define a maneira como uma página da Web se transforma na árvore. Existem vários analisadores sintáticos, e todos eles criam estruturas diferentes de um mesmo documento HTML. Escolhemos o analisador **lxml** por seu desempenho de alta velocidade. Mas, claro, existem outros analisadores, como `html.parser`, `xml` e `html5lib`.

Depois que o código é transformado em uma estrutura de árvore, os dados podem ser pesquisados usando vários métodos. O primeiro método de busca se chama **find()**. Ele percorre um documento HTML e encontra o primeiro elemento cujo nome foi passado como argumento e o retorna junto com as tags e seu conteúdo.

```
tag_content = soup.find(tag)
```

Para exibir o conteúdo sem tags, você precisará do método **text**. Ele retornará o resultado na forma de uma string:

```
tag_content.text
```

Existe ainda outro método de busca que é o **find\_all**. Ao contrário do método anterior, `find_all()` encontra *todas* as instâncias de um determinado elemento em um documento HTML e retorna uma lista:

```
tag_content = soup.find_all(tag)
```

Vamos extrair apenas o conteúdo dos parágrafos com a ajuda do método *text*:

```
for tag_content in soup.find_all(tag):  
    print(tag_content.text)
```

Os métodos `find()` e `find_all()` possuem um filtro extra para procurar elementos de página: o parâmetro **attrs** (atributos). É usado para busca por classes e identificadores. Seus nomes são especificados no painel de ferramentas do desenvolvedor da Web.

Você precisa passar para o parâmetro `attrs` um dicionário com os nomes e valores dos atributos.

```
soup.find(tag, attrs={"attr_name": "attr_value"})
```

## API

Às vezes precisamos requisitar informação de fontes externas que possuem uma estrutura muito mais complexa de que uma página HTML simples. Para evitar a necessidade de estudar sua estrutura para conseguir obter dados mais rapidamente, os analistas enviam requisições GET para aplicações de terceiros através de uma interface especial de transferência de dados chamada **API** (Application Programming Interface, o que literalmente significa uma interface de programação de aplicativos).

A biblioteca requests te permite passar parâmetros para uma URL. Se você estiver procurando certos conteúdos específicos em um site de múltiplas páginas, você precisa passar o dicionário PARAM para a palavra-chave **params** (parâmetros). Por exemplo:

```
URL = 'https://yandex.com/'
PARAM={"page": "4"}
req = requests.get(url = URL, params = PARAM)
```

Esta requisição deveria retornar a quarta página (conforme o catálogo) do site <https://yandex.com/>.

## JSON

Ao responder à sua requisição, o servidor retorna dados estruturados em um de vários formatos especiais, sendo o mais comum o **JSON** (JavaScript Object Notation). Ele se parece com uma mistura de dígitos, letras, dois pontos e chaves.

Veja como ficam os dados neste formato:

```
[
  {
    "name": "General Slocum",
    "date": "Junho 15, 1904"
  },
  {
    "name": "Camorta",
    "date": "Maio 6, 1902"
  },
  {
    "name": "Norge",
    "date": "Junho 28, 1904"
  }
]
```

Se JSON inclui vários elementos, eles são escritos entre colchetes `[ ... ]`, assim como nas listas. Um objeto JSON individual se parece com um dicionário: é envolto em chaves e tem pares `key : value`.

O JSON permite coletar dados em um objeto (uma lista de pares `key : value`) e, em seguida, criar uma string para ser passada em uma requisição. O receptor transforma essa string novamente em um objeto.

O python tem um módulo embutido para trabalhar com dados no formato JSON:

```
import json
```

Seu método **`json.loads()`** converte as strings que estão no formato JSON:

```
x = '{"name": "General Slocum", "date": "June 15, 1904"}'
y = json.loads(x)

print('Nome : {0}, data : {1}'.format(y['nome'], y['data']))
```

```
# Resposta
Nome : General Slocum, data : Junho 15, 1904
```