Relatório 8 - Prática: Web Scraping com Python p/ Ciência de Dados (II)

Breno Augusto Oliveira Abrantes

Descrição da Atividade

O objetivo principal desta atividade foi aprender a coletar dados da web utilizando web scraping com a biblioteca BeautifulSoup em Python. Durante a prática, diferentes aspectos da estrutura HTML foram analisados, e técnicas para manipular e extrair informações relevantes foram implementadas.

1. Tópico 1 - Estrutura HTML e instalação da biblioteca

Neste primeiro momento, abordamos a estrutura básica de um documento HTML, que é a linguagem de marcação utilizada para formatar o conteúdo das páginas da web. Elementos como <head>, <body>, <footer>, e <title> foram apresentados, assim como suas respectivas funções no layout da página. Isso foi essencial para compreender como o conteúdo de uma página web é organizado e pode ser extraído para análise.

Para iniciar o processo de web scraping, instalamos a biblioteca BeautifulSoup e o lxml, utilizado para o parsing do HTML:

```
pip install beautifulsoup4

pip install lxml
```

A BeautifulSoup permite navegar e manipular o conteúdo HTML com facilidade, ajudando a isolar e extrair partes específicas da página, como títulos, parágrafos e tabelas.

2. Tópico 2 - Utilizando BeautifulSoup para leitura de arquivos HTML locais

Neste tópico, o foco foi na manipulação de arquivos HTML locais, que podem ser carregados no código Python utilizando a função open(). Um exemplo de como ler e processar um arquivo HTML seria:

```
from bs4 import BeautifulSoup
```

```
with open('home.html', 'r') as html_file:
    content = html_file.read()

soup = BeautifulSoup(content, 'lxml')
print(soup.prettify())  # Exibe o HTML de forma organizada
```

Utilizando o método soup.prettify(), podemos formatar o HTML para que ele seja mais legível. Para extrair informações específicas, como títulos de cursos em uma página, podemos utilizar o método find_all():

```
courses_html_tags = soup.find_all('h5')
for course in courses_html_tags:
    print(course.text)  # Exibe apenas o texto contido nas tags <h5>
```

A seguir, um exemplo de extração de informações de uma div com a classe 'card':

```
course_cards = soup.find_all('div', class_='card')

for course in course_cards:
    course_name = course.h5.text
    course_price = course.a.text.split('\n')[-1]
    print(f'{course_name} costs {course_price}')
```

Esse código gera uma lista de cursos e seus respectivos preços, simulando uma situação real de scraping de conteúdo. Isso é útil em casos de coleta de dados em massa, como preços de produtos ou informações de serviços.

3. **Tópico 3 - Scraping de um site real**

No terceiro tópico, foi realizado o scraping de um site real, o **TimesJobs**, para coletar informações sobre vagas de emprego. O processo começou buscando a página HTML com a biblioteca requests:

```
from bs4 import BeautifulSoup
```

```
import requests

html_text = requests.get('https://www.timesjobs.com/candidate/job-search.html?searchType=personalizedSearch&txtKeywords=Python').text
soup = BeautifulSoup(html_text, 'lxml')
```

Em seguida, buscamos as informações desejadas, como o nome da empresa e as habilidades necessárias para a vaga:

```
job = soup.find('li', class_='clearfix job-bx wht-shd-bx')
company_name = job.find('h3', class_='joblist-comp-name').text
skills = job.find('span', class_='srp-skills').text
print(f"Company Name: {company_name}")
print(f"Required Skills: {skills}")
```

O código acima retorna as informações da primeira vaga de emprego encontrada na página. Utilizando um laço de repetição, podemos iterar sobre várias vagas e extrair os dados relevantes:

```
jobs = soup.find_all('li', class_='clearfix job-bx wht-shd-bx')

for job in jobs:
    company_name = job.find('h3', class_='joblist-comp-name').text.strip()
    skills = job.find('span', class_='srp-skills').text.strip()
    published_date = job.find('span', class_='sim-posted').span.text
    print(f"
    Company Name: {company_name}
    Required Skills: {skills}
    Published Date: {published_date}
    ")
```

Esse código exibe as informações de cada vaga encontrada, incluindo o nome da empresa, as habilidades exigidas e a data de publicação. A seguir, um exemplo de como filtrar as vagas com base nas habilidades que o usuário não possui:

```
unfamiliar_skill = input('Enter a skill you are not familiar with: ')


for job in jobs:
    company_name = job.find('h3', class_='joblist-comp-name').text.strip()
    skills = job.find('span', class_='srp-skills').text.strip()
    if unfamiliar_skill not in skills:
        print(f"Company Name: {company_name}")
        print(f"Required Skills: {skills}")
```

Por fim, salvamos os resultados em arquivos de texto:

```
with open(f'posts/{index}.txt', 'w') as f:
    f.write(f"Company Name: {company_name.strip()} \n")
    f.write(f"Required Skills: {skills.strip()} \n")
    f.write(f"Published Date: {published_date} \n")
```

Este processo permitiu organizar e automatizar a coleta de dados de várias vagas de emprego, facilitando análises posteriores.

Criado uma nova função para apagar os txts antigos e inserir os novos dados a cada filtragem:

```
def clear_posts():
    folder = 'posts'
    if os.path.exists(folder):
        for filename in os.listdir(folder):
            file_path = os.path.join(folder, filename)
            if os.path.isfile(file_path) and file_path.endswith('.txt'):
                os.remove(file_path)
```

Uma variavel para salvar nossa pasta com os txts(folder), para cada arquivo dentro na pasta e o caminho do diretório para acessá-las (file_path = os.path.join(folder,

filename)), se o caminho corresponder, encontrar pasta e o arquivo terminar com .txt, eles serão excluídos.

Assim, é possível fazer sempre novos arquivos com as informações que filtramos no terminal, exemplos como: django, css, html5 etc. Irei deixar cada um desses testes.

Conclusões

Usamos Web scraping com Python e BeautifulSoup de uma maneira eficiente para coletar dados de sites. Navegando pela estrutura HTML de uma página e extrair informações específicas, como vimos com a coleta de vagas de emprego, permitecriar pipelines automáticos de dados úteis, desde análise de preços até monitoramento de informações públicas. Ao longo da atividade, foi possível entender como manipular elementos HTML e estruturar os dados coletados de forma que possam ser utilizados em análises futuras.

https://github.com/brenooabrantes/BT-Machine-

Learning-Lamia/tree/main/CARD8

Link com o projeto acima.

Referências

- https://docs.python-requests.org/en/latest/
- https://docs.python.org/3/library/os.html
- https://www.crummy.com/software/BeautifulSoup/bs4/doc/