# Automatizando Relatórios com Python

Este projeto ensina a capturar dados da web (web scraping), gerar um arquivo CSV com os dados coletados e produzir um relatório estatístico automatizado com Python.

---

## Estrutura de Pastas do Projeto

Ao final da configuração e execução dos scripts, sua pasta do projeto ficará assim:

```
OFICINA/
├── venv/             # Ambiente virtual do Python
├── enviar_email.py   # Script para envio do relatório por e-mail
├── relatorio.py      # Script para gerar o relatório estatístico
└── scraper.py        # Script para coletar os dados dos livros (web scraping)
```

---

## Preparando o Ambiente

### 1. Verifique se o Python está instalado

Execute no terminal ou PowerShell:

```
python --version
```

Se não estiver instalado, acesse: python.org/downloads

### 2. Crie um ambiente virtual

Dentro da pasta do projeto:

```
python -m venv venv
```

Ative o ambiente virtual:

- **Windows:**

  ```
  venv\Scripts\activate
  ```

- **Linux/macOS:**

```
source venv/bin/activate
```

### 3. Instale as bibliotecas necessárias

```
pip install requests beautifulsoup4
```

---

## 🚀 Passo a Passo da Automação

### 1. Coletar dados do site

Siga os passos abaixo para capturar os dados dos livros do site Books to Scrape:

1. **Importe as bibliotecas necessárias**
   No início do seu script, importe as bibliotecas que serão usadas:

```
import requests
from bs4 import BeautifulSoup
import csv
```

2. **Defina a URL alvo**
   Especifique o endereço do site que será acessado:

```
url = 'https://books.toscrape.com/'
```

3. **Faça a requisição HTTP**
   Utilize o `requests` para obter o conteúdo da página:

```
resposta = requests.get(url)
resposta.encoding = 'UTF-8'
```

4. **Analise o HTML com BeautifulSoup**
   Converta o conteúdo HTML para um objeto manipulável:

```
soup = BeautifulSoup(resposta.text, 'html.parser')
```

5. **Encontre os elementos dos livros**
   Busque todos os artigos que representam livros:

```
livros = soup.find_all('article', class_='product_pod')
```

6. **Crie e escreva no arquivo CSV**
   Abra (ou crie) um arquivo CSV e escreva os dados coletados:

```
with open('livros.csv', 'w', newline='', encoding='UTF-8') as arquivo_csv:
     escritor = csv.writer(arquivo_csv)
     escritor.writerow(['titulo', 'preco', 'link'])
     for livro in livros:
          titulo = livro.h3.a['title']
          link = livro.h3.a['href']
          preco = livro.find('p', class_='price_color').text
          link_completo = 'https://books.toscrape.com/' + link
          escritor.writerow([titulo, preco, link_completo])
```

Pronto! Agora você terá um arquivo `livros.csv` com os títulos, preços e links dos livros da página inicial do site.

## 2. Gerar relatório estatístico dos livros

Agora que você já tem o arquivo `livros.csv`, vamos criar um relatório automatizado com estatísticas dos livros coletados. O relatório irá mostrar:

- Quantidade total de livros
- Livro mais caro (com título, preço e link)
- Livro mais barato (com título, preço e link)
- Valor total dos livros
- Média de preço dos livros

Veja o passo a passo:

1. **Leia os dados do arquivo CSV**
   Utilize o módulo `csv` para ler os dados salvos anteriormente.

2. **Calcule as métricas desejadas**
   Some os preços, encontre o livro mais caro e o mais barato, e calcule a média.

3. **Escreva o relatório em um arquivo de texto**
   Gere um arquivo `relatorio.txt` com as informações calculadas.

Exemplo de código:

```
import csv

livros = []
local_arquivo = 'livros.csv'

# Lê os dados do CSV
```

```
with open(local_arquivo, 'r', encoding='UTF-8') as arquivo_csv:
     leitor = csv.DictReader(arquivo_csv)
     for linha in leitor:
          preco = float(linha['preco'].replace('£', ''))
          livros.append({
                'preco': preco,
                'titulo': linha['titulo'],
                'link': linha['link']
          })

# Calcula as métricas
total_livros = len(livros)
valor_total = sum(livro['preco'] for livro in livros)
media_valor = valor_total / total_livros
livro_mais_caro = max(livros, key=lambda livro: livro['preco'])
livro_mais_barato = min(livros, key=lambda livro: livro['preco'])

# Escreve o relatório
with open('relatorio.txt', 'w', encoding='UTF-8') as relatorio:
     relatorio.write('--------- Relatório de Livros ---------\n')
     relatorio.write(f"Quantidade de livros: {total_livros}\n")
     relatorio.write(f"Livro mais caro: {livro_mais_caro['titulo']} (£
{livro_mais_caro['preco']:.2f})\n")
     relatorio.write(f"Link: {livro_mais_caro['link']}\n")
     relatorio.write(f"Livro mais barato: {livro_mais_barato['titulo']} (£
{livro_mais_barato['preco']:.2f})\n")
     relatorio.write(f"Link: {livro_mais_barato['link']}\n")
     relatorio.write(f"Valor total dos livros: £{valor_total:.2f}\n")
     relatorio.write(f"Média de preço: £{media_valor:.2f}\n")
```

Após rodar esse script, você encontrará um arquivo chamado `relatorio.txt` com um resumo estatístico dos livros coletados. Isso facilita a análise dos dados de forma rápida e automatizada!

---

## ✉ Enviando o relatório por e-mail (usando smtp4dev)

Agora vamos automatizar o envio do relatório gerado por e-mail, utilizando um servidor SMTP local de testes (smtp4dev). Isso é útil para simular o envio sem realmente entregar o e-mail, ideal para desenvolvimento.

### 1. Pré-requisitos

- O servidor **smtp4dev** deve estar rodando em sua máquina.
- O arquivo `relatorio.txt` já deve ter sido gerado.
- Verifique a porta configurada no smtp4dev (comum: 25, 2525 ou 1025).

### 2. Código do `enviar_email.py`

```
import smtplib
from email.message import EmailMessage

# Caminho do relatório
```

```
arquivo_relatorio = 'relatorio.txt'

# Lê o conteúdo do relatório
with open(arquivo_relatorio, 'r', encoding='utf-8') as file:
    conteudo = file.read()

# Cria a mensagem de e-mail
mensagem = EmailMessage()
mensagem['Subject'] = '📊 Relatório de Livros Automatizado'
mensagem['From'] = 'relatorio@automacao.com'
mensagem['To'] = 'destinatario@exemplo.com'
mensagem.set_content(conteudo)

# Envia pelo servidor SMTP local (smtp4dev)
try:
    with smtplib.SMTP('localhost', 25) as smtp:  # Altere a porta se necessário
        smtp.send_message(mensagem)
        print("☑ E-mail enviado com sucesso (simulado pelo smtp4dev).")
except ConnectionRefusedError:
    print("❌ Erro: Não foi possível conectar ao servidor SMTP local. Verifique
se o smtp4dev está em execução.")
```

### 3. Executando o smtp4dev com Docker

Se preferir usar Docker, execute:

```
docker run -p 5000:80 -p 25:25 rnwood/smtp4dev
```

Depois, acesse http://localhost:5000 para visualizar os e-mails recebidos.

---

Pronto! Agora você pode enviar o relatório por e-mail de forma automatizada e segura para testes.