

Devisnan Bispo


Documentação com informações sobre a atividade de teste.

A primeira atividade teste pediu para realizar um web-Scraping. Iniciei os preparativos Escolhi a ferramenta IDE e Linguagem para realizar o desafio foram as seguintes respectivamente Pycharm e Python, a escolha do Pycharm se dá mais pelo suporte completo para a linguagem python não sendo necessário fazer inúmeros plugins na IDE, mas considere algo plausível o uso do VScode que eu utilizo e gosto muito também.

explicando a organização dos diretórios:

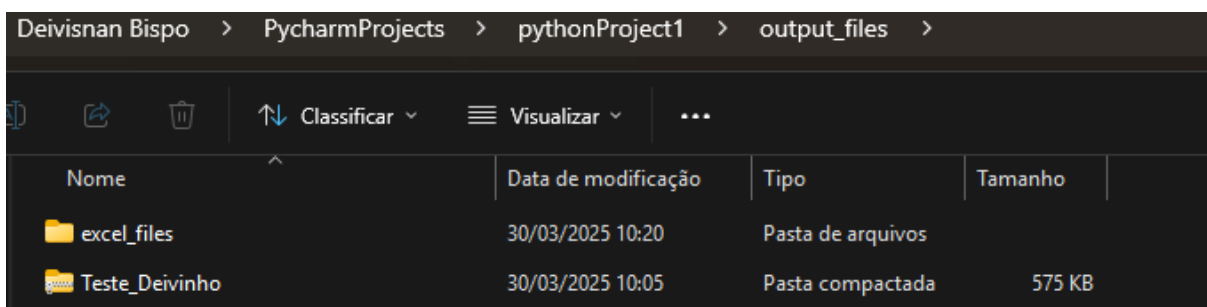
A organização é composta por duas pastas que são necessária para guardar temporariamente os arquivos, **Extracted_files** fica salvo os anexos logo depois esses anexos são compactados automaticamente e são transferidos para pasta **input_files** isso ocorre Quando damos o comando **python web_scraping_ans.py** no terminal.

Em seguida, no segundo comando que é **python Transformacao_dados.py** ele vai acessar a pasta **input_files** irá pegar os anexos zipados dentro da pasta e irá fazer a transformação de dados para CSV.



| | | | | |
|--------|---------------------|------------------|----------------------|------|
| folder | .idea | 30/03/2025 10:04 | Pasta de arquivos | |
| folder | downloads | 29/03/2025 21:37 | Pasta de arquivos | |
| folder | extracted_files | 29/03/2025 19:49 | Pasta de arquivos | |
| folder | input_files | 30/03/2025 10:00 | Pasta de arquivos | |
| folder | output_files | 30/03/2025 10:05 | Pasta de arquivos | |
| folder | venv | 29/03/2025 09:19 | Pasta de arquivos | |
| file | Transformacao_dados | 30/03/2025 00:47 | Arquivo Fonte Pyt... | 6 KB |
| file | web_scraping_ans | 30/03/2025 09:58 | Arquivo Fonte Pyt... | 4 KB |

OBS. Eu criei uma pasta extra na qual também irá guardar um arquivo EXCEL com os dados tratados e organizados de forma que fique mais legível nas planilhas a pasta teste irá gerar o nome de acordo com o nome do usuário que está na máquina.



| | | | | |
|--|----------------|---------------------|-------------------|---------|
| Devisnan Bispo > PycharmProjects > pythonProject1 > output_files > | | | | |
| ↑↓ Classificar ▾ ≡ Visualizar ▾ ... | | | | |
| Nome | ^ | Data de modificação | Tipo | Tamanho |
| folder | excel_files | 30/03/2025 10:20 | Pasta de arquivos | |
| file | Teste_Deivinho | 30/03/2025 10:05 | Pasta compactada | 575 KB |

WEB-SCRAPING

Primeiro fui verificar o site e logo em seguida comecei a desenvolver o código.

Importei as seguintes bibliotecas para realizar o Scraping:

```
import requests
from bs4 import BeautifulSoup
import zipfile
import os
import time
```

Aqui foi onde coloquei a URL da Página onde vai fazer o web scraping, foi necessário criar uma pasta temporária para armazenar os PDFs. Foi utilizar o comando `os.makedirs` para criar um diretório.

```
# URL da página de onde vamos fazer o web scraping
URL =
'https://www.gov.br/ans/pt-br/aceso-a-informacao/participacao-da-
sociedade/atualizacao-do-rol-de-procedimentos'

# Criar uma pasta temporária para armazenar os PDFs
PASTA_DOWNLOADS = 'downloads'
os.makedirs(PASTA_DOWNLOADS, exist_ok=True)
```

Logo a seguir está uma função estruturada para que a requisição seja feita várias vezes caso haja uma falha, foi necessário por um número fixo de tentativas e timeout pois se não o código poderia ficar num loop de requisição o que demoraria sua resposta ou até mesmo poderia travar, decidi escolher 10 tentativas pois percebi que é um número razoável e que atendia as requisições o timeout também foi escolhido um número aceitável depois de eu realizar alguns testes. E lembrando que as requisições não estão sendo mandada uma atrás da outra, caso haja falha tenta depois apenas de 5 segundos, isso garante que não sobrecarregue ou que caso o site esteja com algum problema se estabilize. se no fim não der certo com as tentativas definidas retornará uma mensagem de erro.

```
# Função para tentar a requisição várias vezes em caso de falha
def tentar_requisicao(url, tentativas=10, timeout=20):
    for tentativa in range(tentativas):
        try:
            response = requests.get(url, timeout=timeout)
            response.raise_for_status() # Verifica se a requisição
foi bem-sucedida
            return response
        except requests.exceptions.RequestException as e:
            print(f"Tentativa {tentativa + 1} falhou: {e}")
            if tentativa < tentativas - 1:
```

```

        print("Tentando novamente em 5 segundos...")
        time.sleep(5) # Aguarda 5 segundos antes de tentar
novamente
    else:
        print("Máximo de tentativas alcançado, erro
persistente.")
        raise

```

Aqui é o processo na qual será feita a procura do PDF utilizei um laço for para fazer essa leitura no link, logo em seguida precisei garantir que os links estavam em caminhos absolutos, caso eles estivessem relativos

```

# Encontrar e baixar os PDFs
pdf_links = []
for link in soup.find_all('a', href=True):
    texto_link = link.text.strip().lower()
    href = link['href'].lower()
    if ('anexo' in texto_link) and ('.pdf' in href): # Filtra
samente links .pdf
        pdf_links.append(link['href'])

# Garantir que os links são absolutos (caso estejam como links
relativos)
pdf_links = [link if link.startswith('http') else
f'https://www.gov.br{link}' for link in pdf_links]

```

Aqui neste trecho do código realizei o procedimento para baixar os PDFs que foram encontrados no link da página. há um tratamento a erros, caso ocorra algum erro no download do PDF

```

# Baixar os PDFs encontrados
pdf_files = []
for i, link in enumerate(pdf_links, 1):
    try:
        print(f"Iniciando download do PDF {i} do link: {link}") #
Imprime o link do PDF
        response = requests.get(link, timeout=10)
        response.raise_for_status() # Verifica se o download foi
bem-sucedido
        filename = os.path.join(PASTA_DOWNLOADS, f'anexo_{i}.pdf')
        with open(filename, 'wb') as file:
            file.write(response.content)
        pdf_files.append(filename)
    except:
        pass

```

```

        # Adicionar o link e o arquivo baixado no mapeamento
        mapeamento[link] = filename
        print(f"Download concluído: {filename}")
    except requests.exceptions.RequestException as e:
        print(f"Erro ao baixar o PDF {i}: {e}")

```

e por fim compactar em zip os pdfs baixados:

```

# Compactar os arquivos PDF em um arquivo ZIP
zip_filename = 'anexos.zip'
with zipfile.ZipFile(zip_filename, 'w') as zipf:
    for pdf in pdf_files:
        zipf.write(pdf, os.path.basename(pdf)) # Adiciona apenas o
        nome do arquivo ao ZIP

print(f"Compactação concluída: {zip_filename}")

# Limpeza dos arquivos temporários
for pdf in pdf_files:
    os.remove(pdf)
os.rmdir(PASTA_DOWNLOADS)

print("Processo finalizado com sucesso!")

```

TRANSFORMANDO DADOS EM CSV

Para transformar os anexos pdf em CSV primeiro importei as bibliotecas necessárias como pode ver no código abaixo imagem abaixo

```

import os
import zipfile
import pdfplumber
import pandas as pd
from pathlib import Path

```

```
# Caminhos relativos para as pastas
input_folder = Path.cwd() / 'input_files'
extract_folder = Path.cwd() / 'extracted_files'
output_folder = Path.cwd() / 'output_files'
excel_folder = output_folder / 'excel_files'

# Criando as pastas, caso não existam
input_folder.mkdir(parents=True, exist_ok=True)
extract_folder.mkdir(parents=True, exist_ok=True)
output_folder.mkdir(parents=True, exist_ok=True)
excel_folder.mkdir(parents=True, exist_ok=True)

# Caminho para o arquivo zip
zip_file_path = input_folder / 'anexos.zip'

# Função para extrair os arquivos do zip
def extract_zip(zip_file_path, extract_folder):
    if not zip_file_path.exists():
        print(f"Erro: O arquivo {zip_file_path} não  
foi encontrado.")
        return False
    with zipfile.ZipFile(zip_file_path, 'r') as  
        zip_ref:
        zip_ref.extractall(extract_folder)
    print(f"Arquivos extraídos para:  
{extract_folder}")
    return True

# Extraíndo o zip
if not extract_zip(zip_file_path, extract_folder):
    exit()

# Verifica PDFs extraídos
```

```

pdf_files = list(extract_folder.glob("*.pdf"))
    if not pdf_files:
        print("Nenhum arquivo PDF encontrado na pasta
              extraída.")
        exit()

pdf_file_path = pdf_files[0] # Pegando o primeiro
                             PDF encontrado

    # Função para extrair tabelas do PDF
def extract_pdf_tables(pdf_file_path):
    try:
        with pdfplumber.open(pdf_file_path) as pdf:
            all_data = []
            for page in pdf.pages:
                tables = page.extract_tables()
                for table in tables:
                    clean_table = []
                    for row in table:
                        clean_row =
[cell.replace("\n", " ").strip() if cell else ""
 for cell in row]

                    clean_table.append(clean_row)
            all_data.extend(clean_table)
            return all_data
    except Exception as e:
        print(f"Erro ao ler o PDF: {e}")
        return []

    # Extraíndo os dados do PDF
data = extract_pdf_tables(pdf_file_path)

```

```
# Verificando se há dados extraídos
    if not data:
        print("Nenhum dado extraído do PDF.")
        exit()

# Convertendo para DataFrame
df = pd.DataFrame(data)

# Removendo linhas totalmente vazias
df.dropna(how='all', inplace=True)

# Ajustando colunas (pegando os nomes da primeira
# linha se forem válidos)
    if all(df.iloc[0].notna()):
        df.columns = df.iloc[0] # Define os nomes das
                                # colunas
df = df[1:].reset_index(drop=True) # Remove a
    primeira linha usada como cabeçalho

# Substituindo abreviações por nomes completos
abbreviations = {
    'OD': 'Odontológica',
    'AMB': 'Ambulatorial',
}

df.replace(abbreviations, inplace=True)

# Padronizando os dados (removendo espaços extras
# e colocando tudo em maiúsculas)
df = df.apply(lambda col: col.map(lambda x:
x.strip().upper() if isinstance(x, str) else x))

# Removendo colunas desnecessárias (se houver
# colunas vazias ou irrelevantes)
df.dropna(axis=1, how='all', inplace=True)
```

```

# Ajustando a largura das colunas e a altura das
    linhas
    excel_file_path = excel_folder /
        'rol_de_procedimentos.xlsx'
    with pd.ExcelWriter(excel_file_path,
        engine='xlsxwriter') as writer:
        df.to_excel(writer, index=False,
            sheet_name='Procedimentos')

    # Acessando o objeto da planilha
        workbook = writer.book
    worksheet = writer.sheets['Procedimentos']

# Criando um formato de célula para a quebra de
    linha e altura de linha
cell_format = workbook.add_format({'text_wrap':
    True, 'valign': 'top'})

    # Ajustando a largura das colunas
        column_widths = {}
    for idx, col in enumerate(df.columns):
        max_len =
            df[col].astype(str).apply(len).max() #
Encontrando o comprimento máximo de cada coluna
        adjusted_width = max_len + 4 # Dando um
aumento de largura para garantir visibilidade
        column_widths[idx] = adjusted_width

    # Aumentando a largura das colunas
    for idx, width in column_widths.items():
        worksheet.set_column(idx, idx, width,
            cell_format)

# Ajustando a largura específica da coluna
    "PROCEDIMENTO" para 40

```



```
        procedimento_col_idx =
df.columns.get_loc("PROCEDIMENTO")  # Encontrando
        o índice da coluna "PROCEDIMENTO"
        worksheet.set_column(procedimento_col_idx,
        procedimento_col_idx, 40, cell_format)

# Ajustando a altura das linhas para 30 (40
        pixels)
        worksheet.set_default_row(30)

# Destacando os títulos das colunas com uma cor
        específica
        header_format = workbook.add_format({'bold':
True, 'bg_color': '#FFFF00', 'font_color':
'#000000'})  # Amarelo com texto preto
        for col_num, value in
        enumerate(df.columns.values):
            worksheet.write(0, col_num, value,
            header_format)

# Salvando os dados em um arquivo CSV na pasta de
        saída
        csv_file_path = output_folder /
        'rol_de_procedimentos.csv'
        df.to_csv(csv_file_path, index=False)

# Compactando o CSV em um arquivo ZIP na pasta de
        saída
        zip_output = output_folder /
        f'Teste_{os.getlogin()}.zip'
with zipfile.ZipFile(zip_output, 'w') as zipf:
    zipf.write(csv_file_path,
        'rol_de_procedimentos.csv')

# Removendo o arquivo CSV após a compactação
```

```

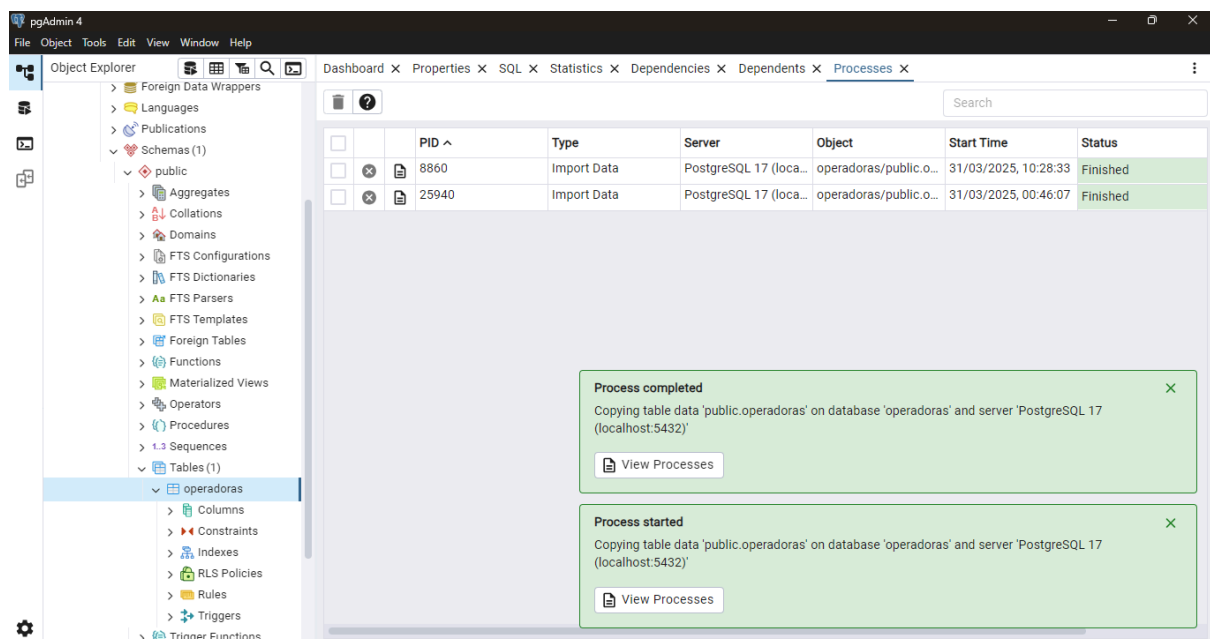
csv_file_path.unlink()

print(f"Arquivo Excel salvo em:
      {excel_file_path}")
print(f"CSV compactado em: {zip_output}")

```

ETAPA COM BANCO DE DADOS

Decidi trabalhar com postgree por ser mais robusto e mais familiaridade, foi desafiador pois tive que organizar os arquivos antes que continha vírgulas e estava gerando erro substituir por ponto, tive que analisar como iria verificar dados de cada trimestre já que estava separado os dados, logo então tive a ideia de criar uma nova coluna chamada trimestre e coloquei 1, 2 , 3 e 4 para primeiro trimestre, segundo trimestre, terceiro trimestre, e quarto trimestre. isso serviu útil para que a query de busca fosse criada corretamente trazendo os dados referente o trimestre e referente ao ano todo.



Query para verificar as 10 operadoras com mais despesas:

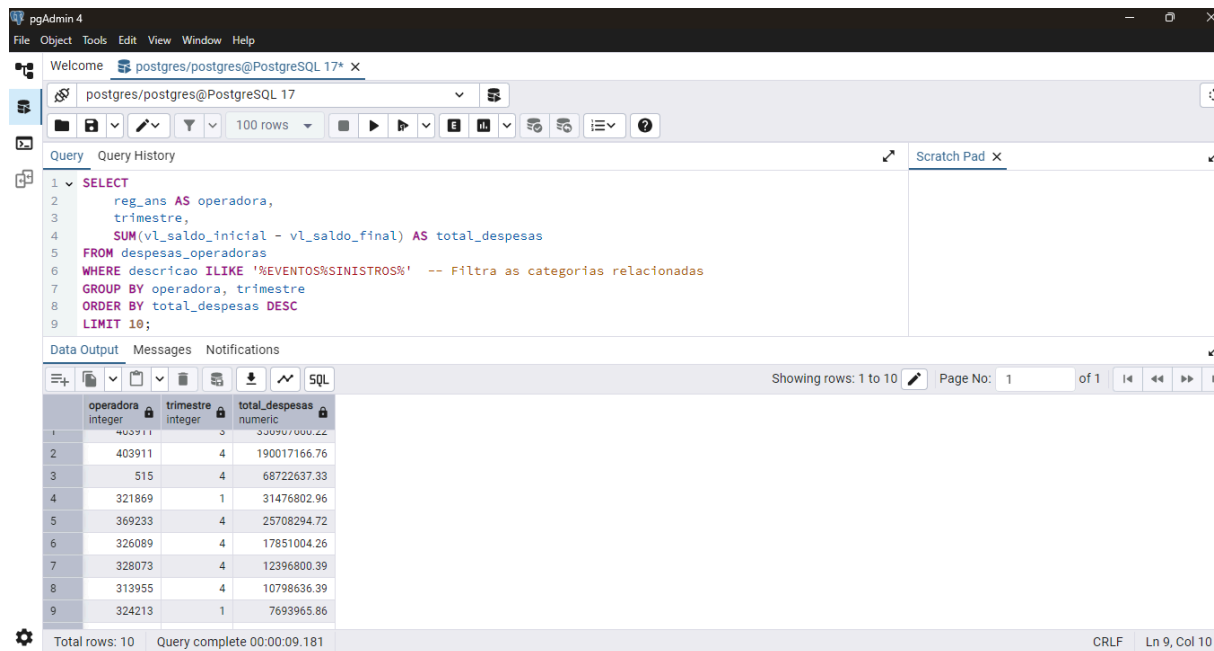
```
SELECT
    reg_ans AS operadora,
    trimestre,
    SUM(vl_saldo_inicial - vl_saldo_final) AS total_despesas
FROM despesas_operadoras
WHERE
    descricao ILIKE '%EVENTOS%SINISTROS%' -- Filtra as categorias
relacionadas
    AND trimestre = 4 -- Filtra apenas o último trimestre
GROUP BY operadora, trimestre
ORDER BY total_despesas DESC
LIMIT 10;
```

Query para verificar as 10 operadoras com mais despesas:

```
SELECT
    reg_ans AS operadora,
    trimestre,
    SUM(vl_saldo_inicial - vl_saldo_final) AS total_despesas
FROM despesas_operadoras
WHERE descricao ILIKE '%EVENTOS%SINISTROS%' -- Filtra
as categorias relacionadas
GROUP BY operadora, trimestre
```

ORDER BY total_despesas DESC
LIMIT 10;

imagem para ilustrar o resultado:



The screenshot shows the pgAdmin 4 interface. The query editor contains the following SQL code:

```
1 SELECT
2     reg_ans AS operadora,
3     trimestre,
4     SUM(vl_saldo_inicial - vl_saldo_final) AS total_despesas
5 FROM despesas_operadoras
6 WHERE descricao ILIKE '%EVENTOS% SINISTROS%' -- Filtra as categorias relacionadas
7 GROUP BY operadora, trimestre
8 ORDER BY total_despesas DESC
9 LIMIT 10;
```

The Data Output tab shows the results of the query, displaying 10 rows. The columns are operadora (integer), trimestre (integer), and total_despesas (numeric). The status bar at the bottom indicates 'Total rows: 10' and 'Query complete 00:00:09.181'.

| | operadora integer | trimestre integer | total_despesas numeric |
|---|----------------------|----------------------|---------------------------|
| 1 | 403911 | 3 | 330907000.22 |
| 2 | 403911 | 4 | 190017166.76 |
| 3 | 515 | 4 | 68722637.33 |
| 4 | 321869 | 1 | 31476802.96 |
| 5 | 369233 | 4 | 25708294.72 |
| 6 | 326089 | 4 | 17851004.26 |
| 7 | 328073 | 4 | 12396800.39 |
| 8 | 313955 | 4 | 10798636.39 |
| 9 | 324213 | 1 | 7693965.86 |