Web Scrapping utilizando Python, SQL e Power BI

Matheus Fellipe do Carmo Barros

No meu artigo conceitual eu apresentei o desafio de fazer um processo de "raspagem" (scrapping) de dados em um site de uma prefeitura, para a obtenção dos dados referentes às proposições apresentadas pelos vereadores e a posterior apresentação desses dados em um relatório que facilitasse a visualização dos resultados obtidos.

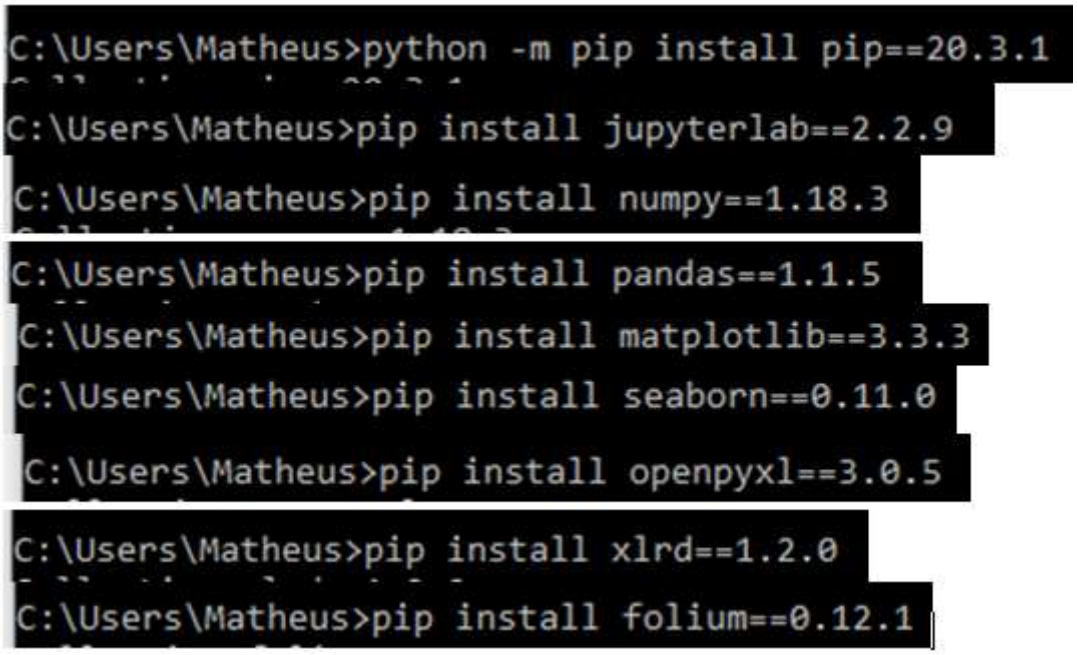
Na busca de resolver esse desafio eu cheguei a essa solução: Utilizar Python para realizar o web scrapping no site e posterior tratamento dos dados, conexão e criação de um banco de dados SQL Server. Já tendo armazenado os dados no banco realizei a integração do banco de dados com o Power BI e posteriormente criei um dashboard com filtros e gráficos dinâmicos que me possibilitou visualizar o histórico das proposições geradas pelos vereadores e separar essas proposições em diversas categorias como: assunto, ano, autor, situação. Assim eu consegui ter uma clara noção do trabalho realizado pelos vereadores ao logo dos anos e, por exemplo, temas mais abordados, períodos do ano com maior volume de apresentações e etc.

No desenvolvimento da solução eu realizei as seguintes etapas:

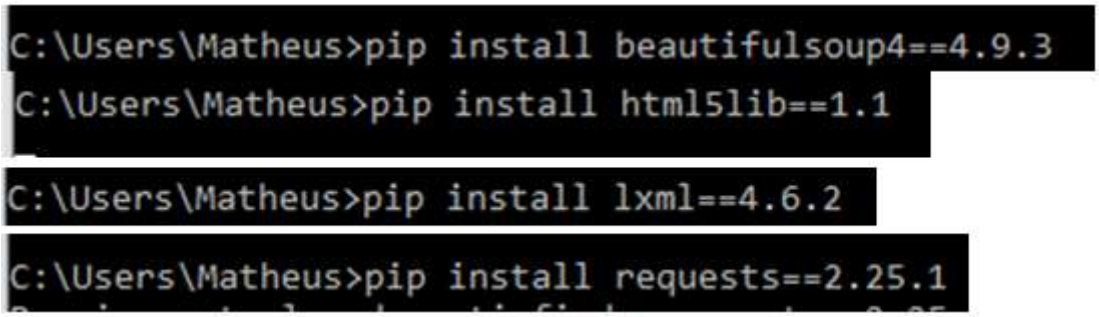
1. Web Scrapping
2. Criação e conexão com banco de dados SQL Server
3. Carregamento da base de dados do banco SQL Server no Power BI
4. Criação de medidas utilizando DAX no Power BI
5. Criação de tabelas auxiliares utilizando o Power Query no Power BI
6. Criação do dashboard no Power BI para a visualização dos resultados obtidos

1 Web Scrapping

Eu já tinha a versão 3.7.4 do Python instalado no meu computador. Com isso criei uma pasta para armazenar o meu projeto. Através do Prompt de comando eu instalei as bibliotecas que foram sendo requeridas ao longo do projeto, sendo elas:



Além dessas bibliotecas de uso geral eu precisei instalar algumas específicas para a utilização da técnica de web scraping:



Abri um novo notebook no Jupyter e comecei os trabalhos. Primeiramente eu realizei a importação das primeiras bibliotecas que eu precisei:

```
In [ ]: # Importando bibliotecas

        import time
        import pandas as pd

        from urllib.error import URLError, HTTPError
        from urllib.request import Request, urlopen

        import bs4

        import ssl
        ssl._create_default_https_context = ssl._create_unverified_context
```

Nessa etapa me deparei com um erro de certificado expirado. Após realizar pesquisas encontrei uma solução de usar o ***import ssl*** *e o* ***ssl.create*** para corrigir o erro.

Para garantir uma consulta sem erros de links e respostas eu criei uma função que me garantia isso. Primeiramente eu preciso saber qual o **User-agent** do navegador que eu utilizei, no caso o Chrome. Uma boa definição para o user-agent seria que ele é a identificação que o navegador passa para os sites, e que estes usam para entregar o suporte ou layout adequado.

Para capturar o "**User Agent**" eu abri uma pagina web no navegador Chrome, apertei F12 e depois F5, cliquei em **Headers**, escolhi o site na lista e copiei o **user-agent** na lista que se abriu, após isso eu criei uma variável para armazenar o valor do user-agent:

```
In [1]: # carregando variável com valor do "agente"
        agente = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36"
```

Criei uma varável **headers** para carregar o agente que será utilizado na requisição:

```
# Carregue o agente que será usado na requisição
headers = {'User-Agent': agente}
```

Feito isso eu já podia criar a função que realiza a consulta Web com tratamento de possíveis erros. Na função eu passo a url desejada e coloco uma exceção para caso encontre um erro ela passe.

```
In [10]: def ConsultaWebB(url):
             try:
                 req = Request(url, headers = headers)
                 response = urlopen(req)
                 return response.read()

             except:
                 pass
```

Depois de criar a função para realizar a consulta eu criei uma função para pegar a página encontrada e transformá-la em um objeto utilizável. Para isso eu utilizei a biblioteca BeautifulSoup que me garante conseguir acessar as tag´s sem me preocupar em ficar quebrando textos.

```
In [27]: def captura_html_pagina(url):
             html = ConsultaWebB(url)
             soup = bs4.BeautifulSoup(html, 'html.parser')
             return soup
```

Eu precisei coletar os dados do cabeçalho da proposição. Porém não sendo possível utilizar **import pandas as pd pd.read_html(url)** por não existir uma TAG com o nome *table,* eu utilizei o **html.findAll** para localizar as tags html que continham o conteúdo desejado. O **dt = html.find_all('dt')** retorna todo type html, depois o **x = dt[i].get_text()** armazena só a informação contida em cada linha do **dt**.

```
In [ ]: # Função para capturar as informações da "tabelinha cabeçalho" da página

        def Cabecalho(html):
            dt = html.find_all('dt')
            dd = html.find_all('dd')
            dic = {}
            for i in range(len(dt)):
                x = dt[i].get_text()
                y = dd[i].get_text()
                dic[x] = y
            return dic
```

Feito isso o próximo passo foi criar uma função que busca o conteúdo da proposição. Assim criei uma função que recebe como parâmetros a proposição desejada e o ano. Nessa função eu crio uma variável para receber a url do site em que será feita a busca, nessa url eu realizo uma concatenação que passa a proposição desejada e o ano. Depois eu crio uma variável que recebe a consulta da página em um objeto utilizável e um dicionário que recebe o conteúdo do cabeçalho da proposição. Feito isso a função ficou assim

```
In [ ]: # Funcao para capturar o conteúdo da página e transformar em um dicionário

def Conteudo(proposicao, ano):
    url = 'https://www.legislador.com.br//LegisladorWEB.ASP?WCI=ProposicaoTexto&ID=3&TPProposicao=1&nrProposicao='
        +str(proposicao)+'&aaProposicao='+str(ano)
    html = captura_html_pagina(url)
    dic = Cabecalho(html)
    dic['Proposição'] = proposicao
    dic['Ano'] = ano
    dic['Texto'] = html.p.get_text()
    return dic
```

Feito tudo isso eu criei uma função que realizará a busca das proposições. Essa função recebe como parâmetros uma variável que indica o ano inicial da busca, a quantidade de anos que serão buscados, o ano atual sendo buscado, a quantidade de erros admissíveis na busca e um valor para os segundos de espera entre cada busca.

```
In [ ]: def TabelaResultados(inicar_em, quantidade, ano, erros_admissiveis, segundos_espera):

    ultima_consulta = inicar_em + quantidade - 1
    # erros
    erros = 0

    # variaveis para loop
    i = 1
    lista = []

    while inicar_em <= ultima_consulta and erros <= erros_admissiveis:

        try:
            x = Conteudo(inicar_em,ano)
            lista = lista + [Conteudo(inicar_em,ano)]
        except:
            erros += 1
            pass

        time.sleep(segundos_espera)

        # carregamento incremental das variáveis
        inicar_em+=1
        i+=1
    return pd.DataFrame(lista)
```

2 Criação e conexão com banco de dados SQL Server

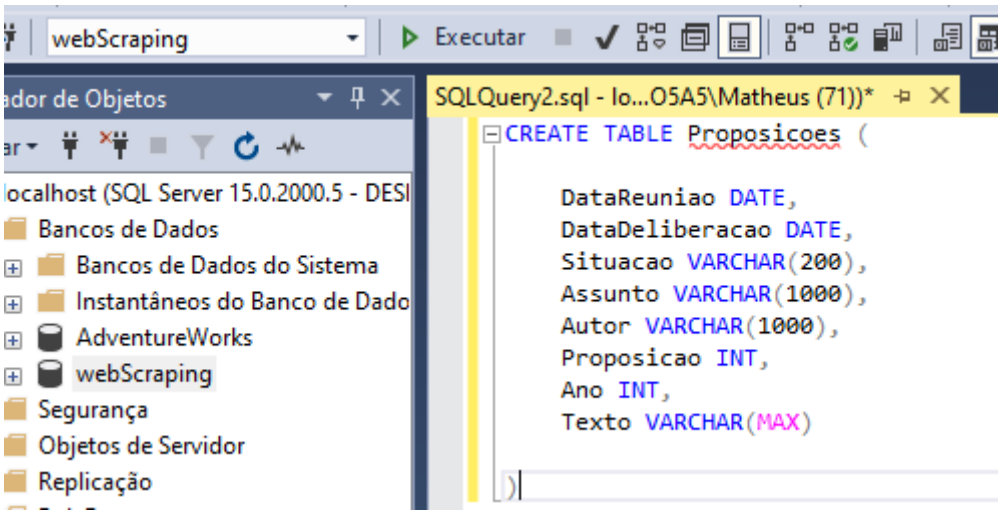
Nessa etapa eu utiizei o SQL Server e o Microsoft Management Studio. Primeiramente eu precisei instalar a biblioteca pyodbc

```
C:\Users\Matheus>pip install pyodbc==4.0.30
```

Para realizar a conexão com o banco de dados eu poderia utilizar uma conexão com senha ou uma conexão com autenticação do Windows (que foi a que eu utilizei)

In [ ]:
```
'''
# Com senha
conn = pyodbc.connect('Trusted_Connection=yes',
                      driver = '{ODBC Driver 17 for SQL Server}',
                      server = 'localhost',
                      database = 'webScraping',
                      UID='sa',
                      PWD='SuaSenha')
'''

# Com autenticação do Windows
conn = pyodbc.connect('Trusted_Connection=yes',
                      driver = '{ODBC Driver 17 for SQL Server}',
                      server = 'localhost',
                      database = 'webScraping')

query = '''
    select
        *
    from Proposicoes
'''
sql_query = pd.read_sql_query(query,conn)
sql_query
```

Feito a conexão com o SQL Server eu criei a base **Indaial** e a tabela **Proposicoes**



Com o banco criado eu criei as funções que me permitiram realizar seleção, limpeza e carregamento de dados no banco.

A função de seleção ficou assim:

In [ ]:
```
def SQLSelect(query):

    conn = pyodbc.connect('Trusted_Connection=yes',
                          driver = '{ODBC Driver 17 for SQL Server}',
                          server = 'localhost',
                          database = 'webScraping')

    out = pd.read_sql_query(query,conn)
    return out
```

Função para limpeza dos dados

In [ ]:
```
def SQLTruncate(NomeTabela):

    conn = pyodbc.connect('Trusted_Connection=yes',
                          driver = '{ODBC Driver 17 for SQL Server}',
                          server = 'localhost',
                          database = 'webScraping')

    cursor = conn.cursor()

    cursor.execute(f'''

                   TRUNCATE TABLE {NomeTabela}

                   ''')

    conn.commit()
    cursor.close()
```

Nesse projeto eu utilizei a ideia de seguir o seguinte passo a passo: Definir os dados que eu iria buscar e logo após capturar a HTML. Tendo sucesso na captura do html eu gero uma tabela, gravo no banco e indico o próximo alvo de busca. Assim para realizar essa atualização incremental dos dados no banco cada consulta tem que ser realizada na forma de **proposicao_anterior + 1**, com isso fiz:

```
def InsereProximaProposicao(ano):

    # Busca última proposição cadastrada
    dados_ano = SQLSelect(f'select Proposicao = max(Proposicao) from Proposicoes where Ano = {ano}')
    ultima_proposicao = dados_ano['Proposicao'].loc[0]

    # Verifica se foi identificado lançamento naquele ano
    if ultima_proposicao == None:
        proxima_proposicao = 1
    else:
        proxima_proposicao = int(ultima_proposicao) + 1

    # Captura e Insere dados na tabela
    dados = Conteudo(proxima_proposicao,ano)
    tabela = pd.DataFrame([dados])
    SQLInsertProposicoes(tabela)
```

```
InsereProximaProposicao(1996)
SQLSelect('select *from Proposicoes')
```

Para tratar o erro de quando chegar na ultima proposição fiz essa função:

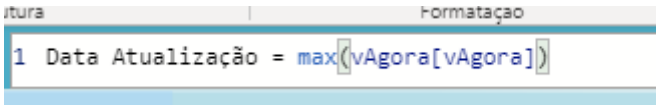
```python
def BuscaGravaDadosAno(ano, quantidade = 10, erros_admissiveis = 2, segundos_espera = 0):

    # erros
    erros = 0

    # variaveis para loop
    i = 1
    lista = []

    while erros <= erros_admissiveis:

        try:
            InsereProximaProposicao(ano)
        except:
            erros += 1
            pass

        time.sleep(segundos_espera)

        # carregamento incremental das variáveis
        i +=1
```

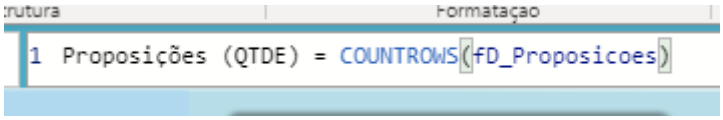
3 Carregamento da base de dados do banco SQL Server no Power BI

Utilizando Power BI eu realizei o carregamento dos dados

4 Criação de medidas utilizando DAX no Power BI

Comecei criando as medidas que seriam necessárias no processo (Proposições QTDE, Proposições aprovadas % e Proposições aprovadas QTDE), depois criei umas que julguei que seriam interessantes (Data Atualização e Rodapé)





```
Proposições Aprovadas (%) =
DIVIDE(
    [Proposições Aprovadas (QTDE)],
    [Proposições (QTDE)]
)
```

```
Proposições Aprovadas (QTDE) = CALCULATE([Proposições (QTDE)], fD_Proposicoes[Situacao] ="Proposição Aprovada")
```

5 Criação de tabelas auxiliares utilizando o Power Query no Power BI

Primeiro eu criei a tabela principal fD_Proposicoes

```
= Sql.Database("localhost", "webScraping", [Query="
```

Próximo passo foi criar tabelas auxiliares a fD_Proposicoes

Primeiro eu criei a dCalendario

Editor Avançado

dCalendario

```
let
    vDataIni = vDataMinima,
    vDataFim = Date.EndOfYear(vDataHoje),
    dias = Duration.Days(vDataFim - vDataIni)+1,
    lista_datas = List.Dates(vDataIni, dias, #duration(1,0,0,0)),
    TabelaDatas = Table.FromList(lista_datas, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
    #"Personalização Adicionada" = Table.AddColumn(TabelaDatas, "Datas", each

        Table.FromRecords({[
            MesNum = Date.Month([Column1]),
            AnoNum = Date.Year([Column1]),
            Ano_Mes = Date.ToText([Column1], "yyyy-MM")

        ]})


    ),
    #"Datas Expandido" = Table.ExpandTableColumn(#"Personalização Adicionada", "Datas", {"MesNum", "AnoNum", "Ano_Mes"}, {"MesNum", "AnoNum", "Ano_Mes"}),
    #"Tipo Alterado" = Table.TransformColumnTypes(#"Datas Expandido",{{"MesNum", Int64.Type}, {"AnoNum", Int64.Type}, {"Ano_Mes", type text}, {"Column1", type date}}),
    #"Colunas Renomeadas" = Table.RenameColumns(#"Tipo Alterado",{{"Column1", "Data"}})
in
    #"Colunas Renomeadas"
```

Criei uma fC_Autores a partir da fD_Proposicoes

```
= fD_Proposicoes[[Linha], [Autor]]
```

Criei uma d_Autores a partir da fC_Autores

```
= fC_Autores[[Autor]]
```

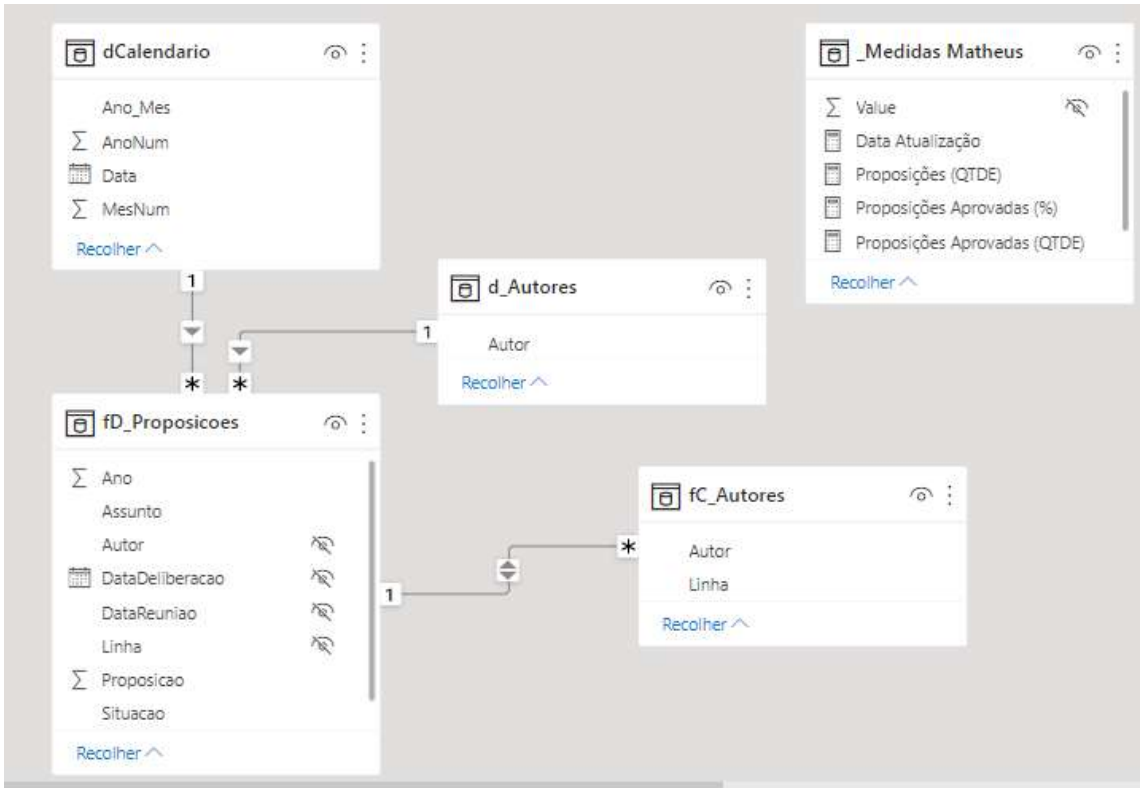
Uma consulta para buscar a data do dia

```
= Date.From(DateTime.LocalNow())
```

E uma consulta para buscar a data do inicio da busca

```
= Date.AddYears(Date.StartOfYear(vDataHoje), -26)
```

As tabelas ficaram relacionadas da seguinte maneira



A solução encontrada por mim foi um dashboard com filtros e gráficos dinâmicos. Esses filtros e gráficos me possibilita visualizar o histórico das proposições geradas pelos vereadores e separar essas proposições em diversas categorias como: assunto, ano, autor, situação. Assim eu consegui ter uma clara noção do trabalho realizado pelos vereadores ao logo dos anos e, por exemplo, temas mais abordados, períodos do ano com maior volume de apresentações