MVP | Sprint: Engenharia de Dados

PUC-Rio

Pós-Graduação em Ciência de Dados e Analytics

Isabela Ayres

Construindo um pipeline de dados utilizando tecnologias na nuvem, envolvendo a busca, coleta, modelagem, carga e análise dos dados.

---

1. Contexto e Objetivo

A Ciência de Dados oferece uma ampla gama de aplicações, desde projetos altamente sofisticados até tarefas cotidianas. Por exemplo, pode ser aplicada em situações sofisticadas, como a previsão de ações no mercado financeiro para orientar decisões de investimento. Por outro lado, ela também pode simplificar atividades comuns, como otimizar listas de compras para uma ida ao supermercado.

O foco deste projeto é estabelecer um pipeline de dados utilizando tecnologias em nuvem, abrangendo as etapas de pesquisa, coleta, modelagem, carregamento e análise de dados. Um pipeline de dados é essencialmente um sistema que permite a automação do fluxo de informações, desde a obtenção inicial dos dados até a análise e tomada de decisões com base neles. Ele cria uma estrutura eficiente para processar, transformar e extrair insights valiosos a partir dos dados.

Nesse contexto, este MVP tem como objetivo analisar preços de produtos em supermercados, compará-los com um valor de referência e fornecer orientações sobre a conveniência da compra.

Para alcançar esse objetivo, várias tecnologias e ferramentas foram utilizadas, incluindo Web Scraping (processo de extração de informações de websites de forma automatizada) e soluções em nuvem para pesquisa, armazenamento e análise de dados.

Na etapa de Web Scraping, foi utilizada a biblioteca BeautifulSoup em Python, uma ferramenta que simplifica a extração de dados de páginas da web. Ela atua como um assistente que navega pelo código HTML de um site, permitindo a coleta de informações de forma automatizada.

Já para o processo de carregamento, transformação e análise dos dados, foram utilizados os serviços em nuvem da AWS (Amazon Web Services), que é uma plataforma de computação em nuvem que oferece uma variedade de serviços e recursos para hospedar, armazenar, processar e analisar dados.

2. Busca e coleta de dados

A busca pelos dados foi aplicada diretamente no site da rede de supermercados Guanabara. Em seguida, para coletar esses dados, foi criado um robô em Python, que automatizou a tarefa de Web Scraping, conforme código abaixo.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import datetime

# Configuração para não exibir os warnings
import warnings
warnings.filterwarnings("ignore")

# Salva o dia e hora atual
current_time = datetime.datetime.now().strftime('%d/%m/%y %H:%M')

# Lista para percorrer todos os setores de produtos
codigo_setor = [42, 82, 32, 152, 203, 2, 92, 52, 62, 112, 102, 12]

dicionario_setor = {42:'acougue', 82:'bebidas', 32:'biscoito',
152:'bombom', 203:'infantil', 2:'cereais', 92:'conservas', 52:'embutidos',
62:'frios', 112:'higienepessoal', 102:'limpeza', 12:'padaria'}

#Listas para armazenar dados por setor
acougue = []
bebidas = []
biscoito = []
bombom = []
infantil = []
cereais = []
conservas = []
embutidos = []
frios = []
higienepessoal = []
limpeza = []
padaria = []

# Cria df inicial
df = pd.DataFrame()

# Função para ler e armazenar dados
for setor in codigo_setor:
    url = "https://www.supermercadosguanabara.com.br/produtos/" +
str(setor)
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    Product = soup.find_all('div', {'class': 'name'})
    Price = soup.find_all('div', {'class': 'price'})


    df = pd.DataFrame({"Date": current_time,
                       "Sector": dicionario_setor[setor],
                       "Product": Product[16:],
                       "Price": Price,
                          }).append(df)

# Tranformar para .csv e salvar arquivo
df.to_csv('CAMINHO/WebScraping-Guanabara.csv', index=False)
```

Notebook disponível no GitHub:
https://github.com/IsabelaAyres/datascience/blob/main/MVP_Data_Engineering/web_scraping.ipynb

3. Modelagem

Quanto à modelagem dos dados, optou-se pelo uso de um modelo flat, devido à natureza simples e direta das informações disponíveis. Nesse contexto, uma modelagem flat significa que os dados foram organizados em uma tabela simples, com poucas colunas e uma estrutura que reflete a simplicidade das informações obtidas. Essa abordagem foi escolhida para manter a simplicidade e clareza dos dados, sem a necessidade de estruturas complexas de banco de dados.

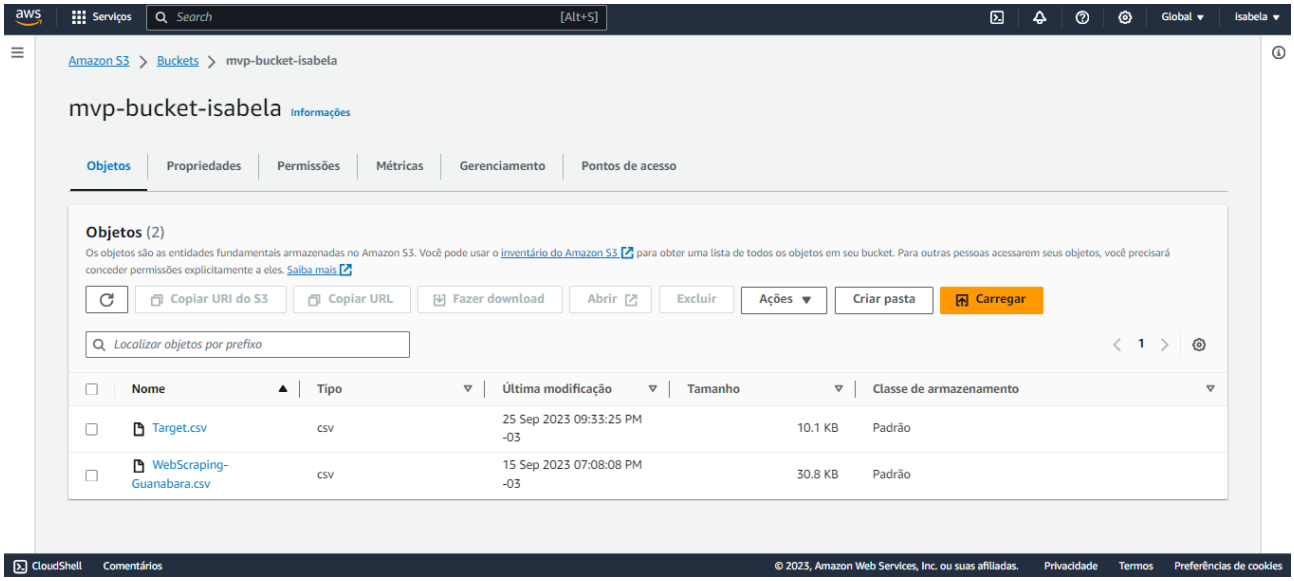
4. Catálogo de Dados

Abaixo a descrição detalhada de cada atributo.

- **dateexport:** É a data em que os dados foram coletados pelo robô de coleta;
- **sector**: Refere-se à categoria em que o produto está classificado, abrangendo áreas como limpeza, alimentos, produtos de higiene, entre outras;
- **product:** Neste campo, encontra-se uma descrição detalhada do produto em questão;
- **price:** Este campo apresenta o preço atual do produto;
- **targetprice:** Aqui está o preço alvo para o produto, que é o valor máximo estimado para a compra desse item. Quando o preço atual excede esse valor, o programa indica que a compra não é recomendada.

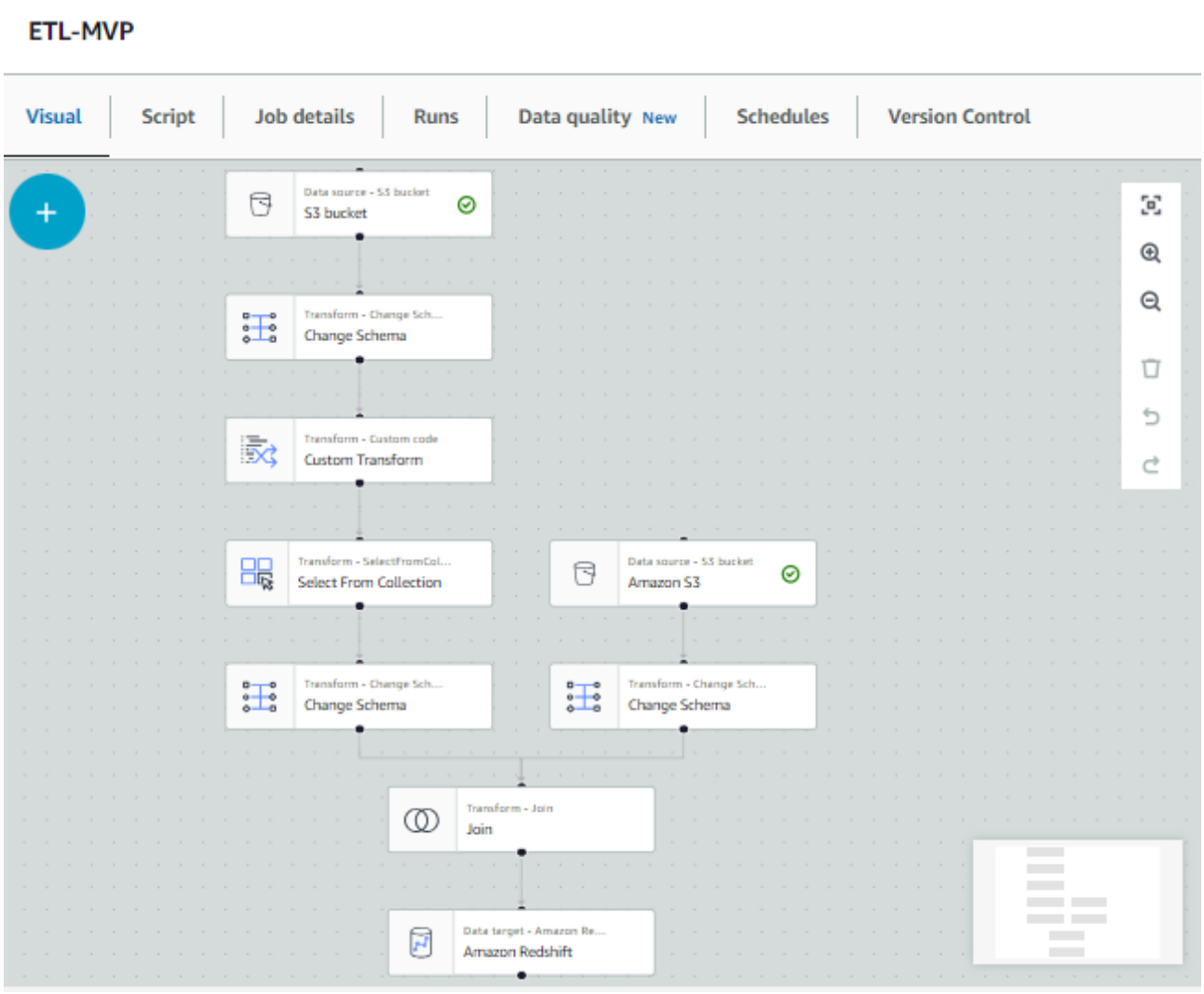
5. Carga

Após o processo de extração de dados e a conversão desses dados para o formato .csv realizada pelo robô de coleta, foi feito o armazenamento na nuvem, mais especificamente no Amazon S3 (Simple Storage Service), um serviço de armazenamento fornecido pela Amazon Web Services (AWS).



Na sequência, um Job no Amazon Glue foi configurado para realizar o processo de ETL (Extract, Transform, Load) nos dados adquiridos. O AWS Glue é uma ferramenta para realizar a preparação e transformação de dados. O processo envolveu diversas etapas essenciais, detalhadas abaixo.

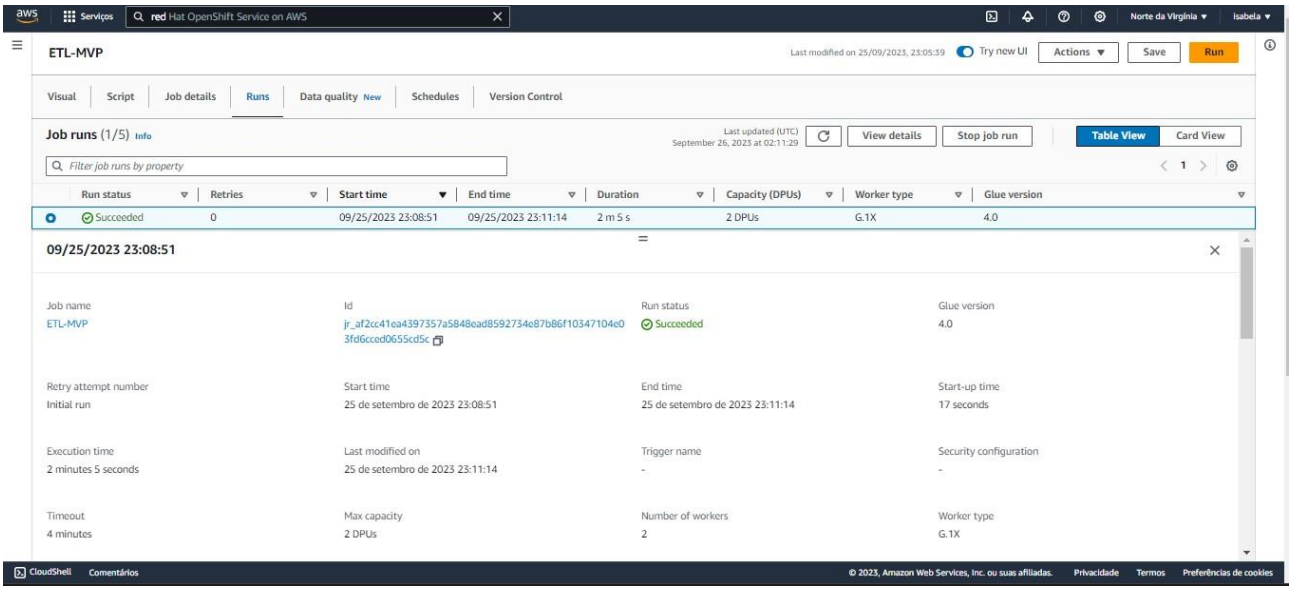
- **Leitura das tabelas .csv:** Inicialmente, o Job efetuou a leitura das tabelas em formato .csv, que continham os dados previamente coletados, bem como a tabela com preços-alvo;
- **Atualização do Schema:** Para garantir que as colunas dos dados estivessem corretamente formatadas e alinhadas com os tipos de dados apropriados, houve uma etapa de atualização do Schema. Isso é crucial para assegurar a qualidade e consistência dos dados ao longo do processo;
- **Transformação Customizada:** Uma transformação personalizada foi aplicada para tratar os textos presentes nos dados. Isso incluiu a remoção de partes não desejadas ou irrelevantes, visando a preparação dos dados para análises futuras;
- **Join das Tabelas:** Para facilitar a execução de comandos de comparação entre os preços dos produtos e seus respectivos preços-alvo, foi necessário realizar um processo de junção (Join) das tabelas envolvidas;
- **Carga no Amazon Redshift:** Finalmente, os dados tratados e transformados foram carregados no Amazon Redshift. O Amazon Redshift é um serviço de data warehousing, também fornecido pela Amazon Web Services (AWS).



Abaixo o código utilizado na etapa de Transformação Customizada.

```
def MyTransform (glueContext, dfc) -> DynamicFrameCollection:
    from pyspark.sql import functions as F

    # Converte o DynamicFrame em DataFrame
    df = dfc.select(list(dfc.keys())[0]).toDF()

    # Função para remover os padrões de texto
    def remove_patterns(col):
        patterns = [
            '<div class="name">',
            '</div>',
            '<div class="price"><div class="vertical-align"><span
class="number">',
            '</span> </div></div>',
            '</span>',
            '<span class="unity">cada'
        ]
        for pattern in patterns:
            col = F.regexp_replace(col, pattern, '')
        return col

    def remove_patternscomma(col):
        patterns = [
            ','
        ]
        for pattern in patterns:
            col = F.regexp_replace(col, pattern, '.')
        return col


    # Aplica a função de remoção às colunas 'Product' e 'Price'
    df = df.withColumn('Product', remove_patterns('Product'))
    df = df.withColumn('Price', remove_patterns('Price'))
    df = df.withColumn('Price', remove_patternscomma('Price'))

    # Converte o DataFrame resultante de volta em um DynamicFrame
    dyf_removed = DynamicFrame.fromDF(df, glueContext, "remove_patterns")

    return DynamicFrameCollection({"CustomTransform0": dyf_removed},
glueContext)
```

Na foto abaixo, pode-se observar o Job sendo excecutado com êxito após todas as configurações.



6. Análise

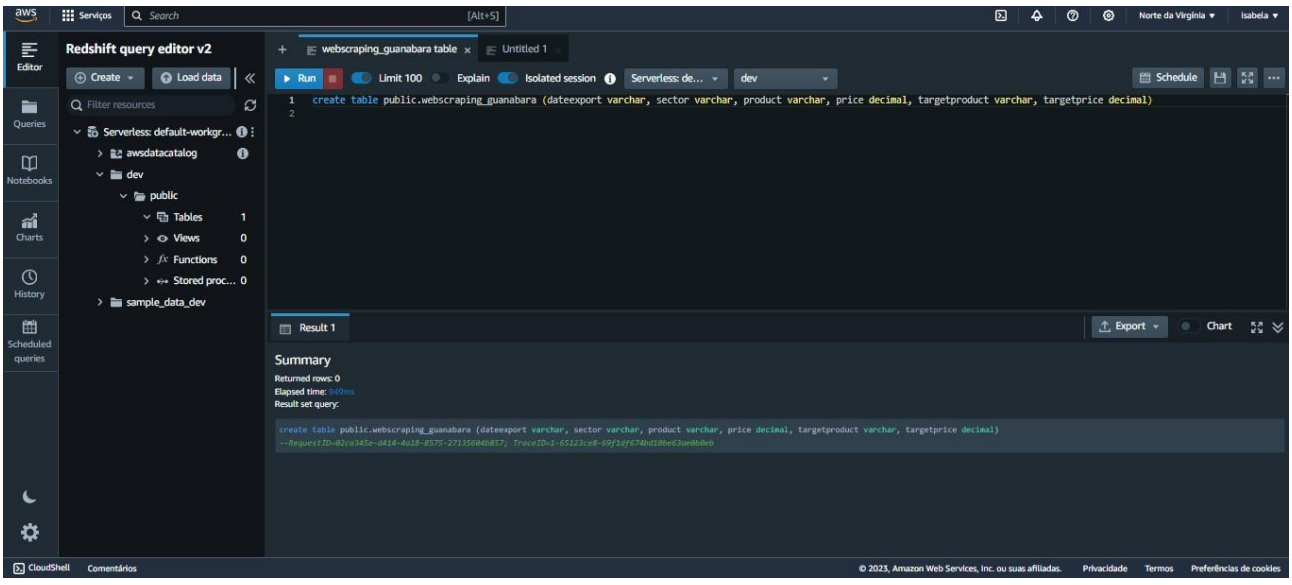
6.1 Qualidade de dados

Não foram identificadas discrepâncias ou problemas de qualidade nos dados. Os valores em cada coluna apresentam consistência, indicando que a etapa de coleta e tratamento foi conduzida de forma adequada.

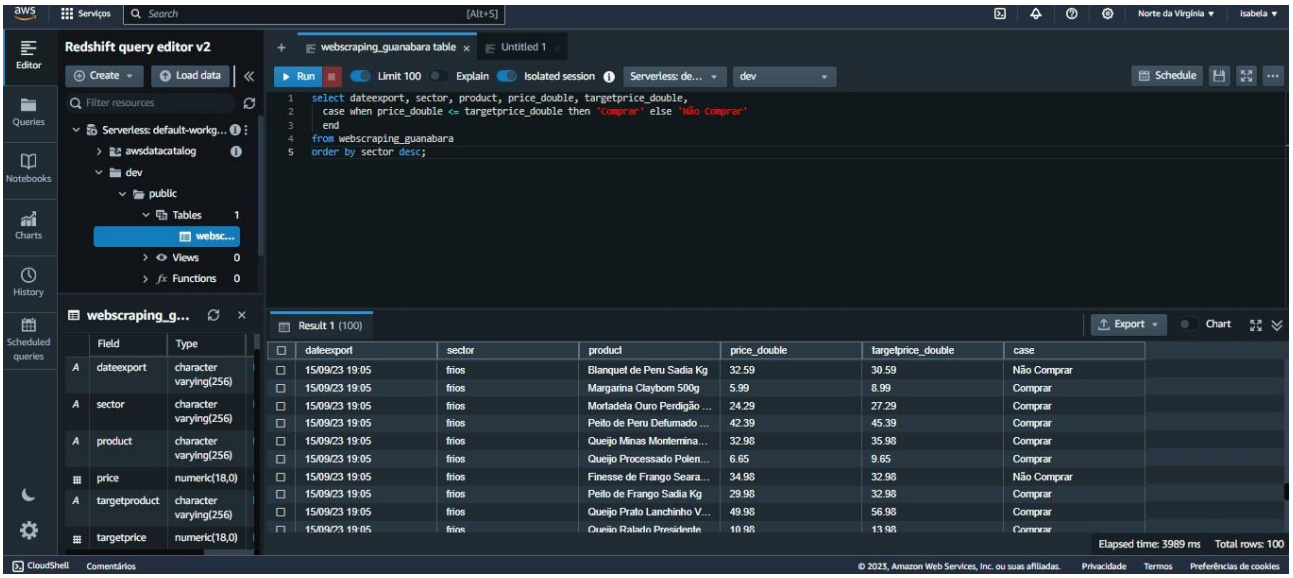
6.2 Solução dos problemas

Para atingir o objetivo do MVP em questão, que é indicar se um produto deve ser comprado ou não, foi criada uma query SQL no Amazon Redshift. Essa query adiciona uma coluna condicional à tabela, classificando os produtos com base em seus preços em relação aos preços alvo. Se o preço atual for menor ou igual ao preço alvo, o produto é marcado como 'Comprar', caso contrário, é marcado como 'Não Comprar'.

Primeiramente, foi criada uma tabela no Redshift para receber os dados transformados na etapa de ETL realizada anteriormente.



Na sequência, foi criada a consulta SQL para determinar a conveniência da compra por produto, considerando seu preço atual x alvo.



A análise dessa coluna demonstrou resultados satisfatórios, pois classificou corretamente os produtos com base no preço alvo estabelecido. Portanto, pode-se concluir que o trabalho alcançou seu objetivo.

Para implementações futuras, seria interessante considerar a criação de uma automação que dispare um e-mail com a lista de itens a serem comprados. Isso otimizaria ainda mais o processo de consulta e tomada de decisão, proporcionando uma experiência mais eficiente para o usuário.