SOULCODE ACADEMY

ENGENHARIA DE DADOS – BC17

DANILO MUSSATO FERRARI

EMILSON CARDOSO MOREIRA

LILIA DE BAKKER GOMES DA GRAÇA

STÉPHANIE ROSSI ROSSANO PIRAJÁ

COMBUSTÍVEIS

Brasil

2022

DANILO MUSSATO FERRARI

EMILSON CARDOSO MOREIRA

LILIA DE BAKKER GOMES DA GRAÇA

STÉPHANIE ROSSI ROSSANO PIRAJÁ

COMBUSTÍVEIS

Documentação apresentada ao Curso de Engenharia de Dados – BC17 da SoulCode Academy como requisito de aprovação no referido curso.

Orientador: Prof. Adriano Gomes.

Brasil

2022

Figuras

Sumário

COMBUSTÍVEIS

Danilo Mussato Ferrari[1]
Emilson Cardoso Moreira[2]
Lilia de Bakker Gomes da Graça[3]
Stéphanie Rossi Rossano Pirajá[4]

Resumo

O referido trabalho tem como objetivo coletar, tratar e disponibilizar dados dos datasets relacionados aos Combustíveis, no período que contempla os segundos semestres dos anos de 2019, 2020 e 2021 sobre os valores de vendas desses combustíveis em todas as regiões do Brasil e as bandeiras dos postos que comercializam esses produtos. A série temporal tem como o cenário nacional o antes, o durante e o "pós" pandemia mundial de Covid-19. Neste projeto foram utilizadas diversas ferramentas e processos, ambientados na nuvem que serão abordados no decorrer deste trabalho. Findado o trabalho de extração, transformação e carregamento dos dados, foi feita algumas análises e insights utilizando o dataset tratado.

Palavras-chave: *dataset*; *dataframe*; Pandas; PySpark.

Abstract

This work aims to collect, process and make available data from the datasets related to Fuels, in the period that includes the second semesters of the years 2019, 2020 and 2021 on the sales values of these fuels in all regions of Brazil

[1] Graduando em Tecnologia de Análise e Desenvolvimento de Sistemas, Graduado em Engenharia de Controle e Automação e Técnico em Eletrônica. E-mail: danilomferrari@terra.com.br

[2] Mestrando em Gestão e Tecnologias Aplicadas à Educação, MBA em Administração Estratégica, Graduado em Sistemas de Informação e Licenciado em Matemática. E-mail: emilsoncardoso@gmail.com

[3] Graduanda em Engenharia de Agrimensura e Cartografia. E-mail: liliabakker.lb@gmail.com

4 Graduanda em Tecnologia de Ciências de Dados. E-mail: tefi.sp@gmail.com

and the flags of the stations that sell these products. The time series has as the national scenario the before, during and "post" world pandemic of Covid-19. In this project, several tools and processes were used, set in the cloud that will be addressed in the course of this work. After the work of extracting, transforming and loading the data, some analyzes and insights were made using the treated dataset.several mists were used that were sent to the pandemic in the course of this work. Finally, generating some important ones to demonstrate the data contained in the final dataset.

Keywords: *dataset*; *dataframe*; Pandas; PySpark; ETL; Python; SQL; NoSQL

Introdução

Como forma de conclusão do curso de Engenharia de Dados da SoulCode Academy foi solicitado a realização de um projeto final a fim de utilizarmos os conceitos vistos durante o curso como forma de avaliação. No escopo do projeto foi especificado a obrigatoriedade de trabalhar com, no mínimo, dois *datasets* (um disponibilizado pelo orientador/professor Adriano Gomes e outro(s) escolhido(s) pela equipe) além disso, foi apontado também como parte obrigatória a utilização das seguintes tecnologias: Google Cloud Platform (Cloud Storage), Python, Pandas, PySpark, SparkSQL, Apache Beam, Data Studio, Big Query e Bancos de Dados SQL e NoSQL.

O tema definido pelos professores para a nossa equipe foi Combustíveis.

No decorrer desta documentação serão apresentados todos os procedimentos tomados pela equipe para a realização e entrega do Projeto Final.

Objetivo

Analisar as métricas dos combustíveis no Brasil nos segundos semestres dos anos 2019, 2020 e 2021 (Período que representa o cenário antes e durante a Pandemia de Covid-19).

A seguir, serão apresentados todos os passos/procedimentos tomados pela equipe para entrega do Projeto Final.

Apresentação da Solução Técnica - *Workflow*

Atendendo ao requisito obrigatório do projeto, apresentamos a seguinte solução técnica, conforme figura 1:

Figura 1 - *Workflow*

Sobre os *Datasets*

Foi disponibilizado pelo orientador/professor Adriano Gomes o *dataset* de nome: equipe1.csv, no formato: csv, de tamanho: 77,5 MB, baixado do: Google Classroom (local onde o prof. disponibilizou), contendo 472.856 linhas e 16 colunas. Para padronizar a nomenclatura o arquivo foi renomeado para: combustiveis-2021-02.csv.

Importamos mais dois *datasets* de nomes: combustiveis-2019-02.pq e combustiveis-2020-02.pq, no formato: *parquet*, de tamanho: 7,28 MB e 3,30 MB (respectivamente), baixado do: GitHub, contendo 507.299 linhas e 16 colunas para o arquivo do ano 2019 e 222.637 linhas e 16 colunas para o arquivo do ano 2020.

Todos os *datasets* contendo os seguintes atributos: Regiao – Sigla, Estado – Sigla, Municipio, Revenda, CNPJ da Revenda, Nome da Rua, Numero Rua, Complemento, Bairro, Cep, Produto, Data da Coleta, Valor de Venda, Valor de Compra, Unidade de Medida e Bandeira.

Google Storage - Bucket

Os Buckets são os contêineres básicos que armazenam seus dados. Tudo que é armazenado no Cloud Storage deve estar contido em um Bucket.

Foram criadas as seguintes estruturas de armazenamento:

- projeto-final-equipe1/Originais - com o objetivo de armazenar os *datasets* originais: combustiveis-2019-02.pq, combustiveis-2020-02.pq e combustiveis-2021-02.csv.
- projeto-final-equipe1/Tratados – com o objetivo de armazenar o *dataset* tratado (pronto para ser utilizado pela equipe de Ciência e Análise de Dados): Combustiveis-Tratados.pq/ (part-00000-2403fbf2-1828-4cf6-9318-bd8b281cc077-c000.snappy.parquet).
- projeto-final-equipe1/Pre-Tratado – com o objetivo de armazenar o *dataset* tratado pelo Pandas e PySpark, antes de disponibilizar os dados no banco de dados MongoDB: Combustiveis.pq.
- projeto-final-equipe1/Pipeline – com o objetivo de armazena os *datasets* gerados pelo processo *pipeline* (parte do projeto intitulado como Requisitos Desejáveis): CombustiveisAno2019-00000-of-00001, CombustiveisAno2020-00000-of-00001, CombustiveisAno2021-00000-of-00001.

Segue figura 2 que tem como função demonstrar o container Originais da plataforma Google Cloud, da seção Google Storage – Bucket:

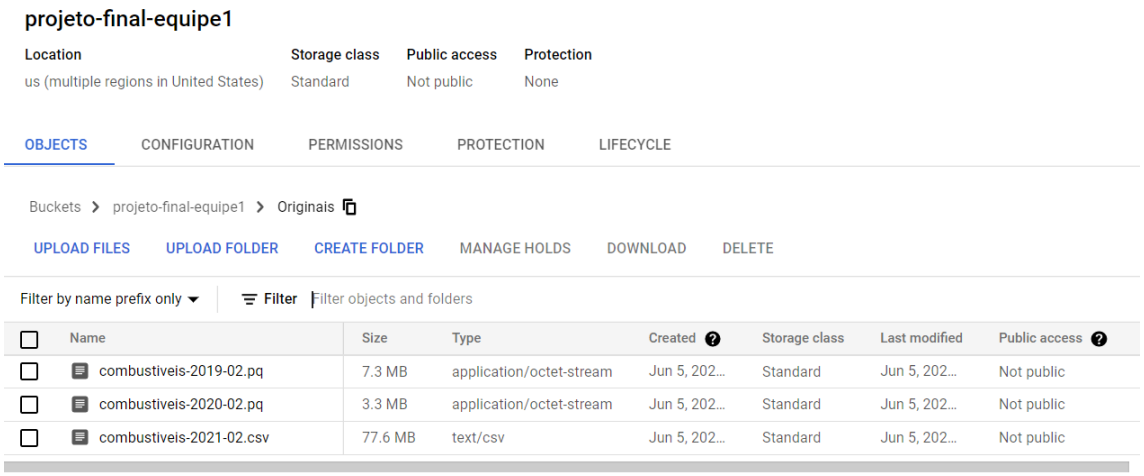

Figura 2 - *Bucket* container originais

IDE Utilizada

Foi utilizado a IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) Google Colab por se tratar de uma ferramenta em um ambiente em nuvem e fizemos uso da linguagem Python em todo nosso trabalho. O nome do nosso arquivo no Colab foi intitulado de: Projeto_Final_Equipe_1.ipynb.

Instalação e importação de bibliotecas

Para o desenvolvimento do Projeto Final, se fez necessário a instalação e importação de diversas bibliotecas. Pode-se observar que na figura 3, demonstramos todas as bibliotecas que foram instaladas, nas figuras 4 e 5 todas as bibliotecas que foram importadas e na figura 6 o código para conexão do Google Colab com o Gooble Drive.

```
[ ]  1 pip install mysql-connector # Fornecem conectividade ao servidor MySQL para programas clientes

[4]  1 pip install PyMySQL # Permite realizar conexão e interação com bancos de dados MySQL e MariaDB

[5]  1 pip install pyspark # É um mecanismo de processamento distribuído , na memória, que permite o pro

[6]  1 pip install gcsfs # Uma interface de sistema de arquivos Python para o Google Cloud Storage

[7]  1 pip install pymongo[srv] # Padrão do MongoDB para Python , é fácil de usar e oferece uma API intu

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pymongo[srv] in /usr/local/lib/python3.7/dist-packages (4.1.1)
Collecting dnspython<3.0.0,>=1.16.0
  Downloading dnspython-2.2.1-py3-none-any.whl (269 kB)
     |████████████████████████████████| 269 kB 13.3 MB/s
Installing collected packages: dnspython
Successfully installed dnspython-2.2.1
```

Figura 3 - Instalação de bibliotecas

Bibliotecas para manipulação dos dados

```
[ ]  1 import pandas as pd
```

```
[ ]  1 import numpy as np
```

Bibliotecas para fazer a conexão com o Banco de Dados MongoDB

```
[ ]  1 from pymongo import MongoClient
     2 from pymongo.mongo_client import MongoClient
     3 from pymongo import collection
```

Bibliotecas para manipulação dos dados em PySpark

```
[ ]  1 from pyspark import SparkConf
     2 from pyspark.sql import SparkSession
     3 from pyspark.sql.types import *
     4 import pyspark.sql.functions as F
```

Figura 4 - Importação de bibliotecas parte 01

Bibliotecas para conexão com o GCP

```
     1 import gcsfs
     2 from google.cloud import storage
     3 import os
```

Biblioteca para fazer a conexão com o Banco de Dados MySQL

```
[ ]  1 import mysql.connector
     2 import pymysql
```

```
[ ]  1 from sqlalchemy import create_engine
     2 import sqlalchemy
```

Figura 5 - Importação de bibliotecas parte 02

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Figura 6 - Conexão com o Google Drvie

Estabelecendo Conexões

Atendendo as solicitações de integração com o ambiente do Google Cloud e com bancos de dados, se fez necessário criar códigos de conexão com o GCP (Google Cloud Platform), com o banco de dados MySQL e com o banco de dados MongoDB Atlas. Na figura 7 é demonstrada os códigos de conexão.

Conector GCP

```
[ ]  1 serviceAccount = "/content/drive/MyDrive/ProjetoFinal/projeto-final-352219-6702b572d951.json"
     2 os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = serviceAccount
```

Conexão MySQL

```
[ ]  1 # Realizando conexão com MySQL no GCP
     2 conexao = mysql.connector.connect(host='35.233.109.131',user='root',password='4d9Uu.2ymgZ0PZbD', db='projeto-final')
     3 cursor = conexao.cursor()
     4 engine = create_engine("mysql+pymysql://root:4d9Uu.2ymgZ0PZbD@35.233.109.131/projeto-final")
```

Conexão com o MongoDB Atlas

```
[ ]  1 CONNECTION_STRING = "mongodb+srv://soulcode:a1b2c3@projetopessoalbc17.blwk4.mongodb.net/?retryWrites=true&w=majority"
```

```
[ ]  1 client = MongoClient(CONNECTION_STRING)
```

```
[ ]  1 dbname = client['Projeto_Final']
```

```
[ ]  1 collection_name = dbname['Combustíveis']
```

Figura 7 - Conexões

Leitura dos *Datasets*

Depois de ter estabelecido as conexões com o ambiente GCP e com os bancos de dados, para atender um dos requisitos do projeto, se fez necessário ler os *datasets* armazenados no GCP (transformando-os em *dataframes*) para podermos manipulá-los. Este processo foi realizado utilizando a biblioteca Pandas. Os códigos de leituras dos *datasets* seguem na figura 8:

Combustiveis 2019-02

```
[ ]  1 df2019 = pd.read_parquet("gs://projeto-final-equipe1/Originais/combustiveis-2019-02.pq")
```

Combustiveis 2020-02

```
[ ]  1 df2020 = pd.read_parquet("gs://projeto-final-equipe1/Originais/combustiveis-2020-02.pq")
```

Combustiveis 2021-02

```
[ ]  1 df2021 = pd.read_csv("gs://projeto-final-equipe1/Originais/combustiveis-2021-02.csv",encoding="unicode_escape", sep=";")
```

Figura 8 - Leitura dos *datasets*

Inserção dos Dados no Bando de Dados MySQL

Depois de termos transformados os três *datasets* em três *dataframes* agimos da seguinte forma (utilizando a biblioteca Pandas):

1. Renomeamos os nomes das colunas para padronizar (conforme figura 9);
2. Verificamos as modificações (conforme figura 10);
3. Inserimos os dados no banco de dados MySQL (conforme figura 11).

A inserção foi feita sequencial em uma tabela chamada combustiveis que após as três inserções ficou com 1.202.792 registros.

```
#Renomeando as colunas para padrozinar na inserção do MySQL
df2019.columns = ['Regiao_Sigla', 'Estado_Sigla', 'Municipio', 'Revenda', 'CNPJ_Revenda', 'Rua', 'Numero',
             'Complemento', 'Bairro', 'CEP', 'Produto', 'Data_Coleta', 'Valor_Venda', 'Valor_Compra',
             'Unidade_Medida', 'Bandeira']
```

```
# Renomeando as colunas para padrozinar na inserção do MySQL
df2020.columns = ['Regiao_Sigla', 'Estado_Sigla', 'Municipio', 'Revenda', 'CNPJ_Revenda', 'Rua', 'Numero',
             'Complemento', 'Bairro', 'CEP', 'Produto', 'Data_Coleta', 'Valor_Venda', 'Valor_Compra',
             'Unidade_Medida', 'Bandeira']
```

```
# Renomeando as colunas para padrozinar na inserção do MySQL
df2021.columns = ['Regiao_Sigla', 'Estado_Sigla', 'Municipio', 'Revenda', 'CNPJ_Revenda', 'Rua', 'Numero',
             'Complemento', 'Bairro', 'CEP', 'Produto', 'Data_Coleta', 'Valor_Venda', 'Valor_Compra',
             'Unidade_Medida', 'Bandeira']
```

Figura 9 - Renomeando colunas via Pandas

```
#Verificando as alterações
df2019.columns
```

```
Index(['Regiao_Sigla', 'Estado_Sigla', 'Municipio', 'Revenda', 'CNPJ_Revenda',
       'Rua', 'Numero', 'Complemento', 'Bairro', 'CEP', 'Produto',
       'Data_Coleta', 'Valor_Venda', 'Valor_Compra', 'Unidade_Medida',
       'Bandeira'],
      dtype='object')
```

```
# Verificando as alterações
df2020.columns
```

```
Index(['Regiao_Sigla', 'Estado_Sigla', 'Municipio', 'Revenda', 'CNPJ_Revenda',
       'Rua', 'Numero', 'Complemento', 'Bairro', 'CEP', 'Produto',
       'Data_Coleta', 'Valor_Venda', 'Valor_Compra', 'Unidade_Medida',
       'Bandeira'],
      dtype='object')
```

```
# Verificando as alterações
df2021.columns
```

```
Index(['Regiao_Sigla', 'Estado_Sigla', 'Municipio', 'Revenda', 'CNPJ_Revenda',
       'Rua', 'Numero', 'Complemento', 'Bairro', 'CEP', 'Produto',
       'Data_Coleta', 'Valor_Venda', 'Valor_Compra', 'Unidade_Medida',
       'Bandeira'],
      dtype='object')
```

Figura 10 - Verificando as alterações das colunas

```
# Inserindo os dados no banco MySQL
df2019.to_sql('combustiveis', engine, if_exists='append', index= False)
```

```
# Inserindo os dados no banco MySQL
df2020.to_sql('combustiveis', engine, if_exists='append', index= False)
```

```
# Inserindo os dados no banco MySQL
df2021.to_sql('combustiveis', engine, if_exists='append', index= False)
```

Figura 11 - Inserindo dados no banco MySQL

Leitura dos Dados do MySQL

Com o objetivo de tratar os dados, se fez necessário criar um novo *dataset* (chamado de dfproj) contendo as informações armazenadas do banco de dados MySQL, desta forma foi feito a leitura utilizando a biblioteca Pandas e a constatação que o novo *dataframe* contém as informações, ambos códigos podem ser verificados na figura 12.

```
# Leitura dos dados MySQL em Dataframe
dfproj = pd.read_sql( "select * from combustiveis", conexao)
```

```
# Verificando a leitura dos dados
dfproj.head(5)
```

	id	Regiao_Sigla	Estado_Sigla	Municipio	Revenda	CNPJ_Revenda	Rua	Numero	Complemento	Bairro	CEP	Produto
0	7216753	S	RS	CANOAS	METROPOLITANO COMERCIO DE COMBUSTIVEIS LTDA	88.587.589/0001-17	AVENIDA GUILHERME SCHELL	6340	None	CENTRO	92310-000	GASOLINA
1	7216754	S	RS	CANOAS	METROPOLITANO COMERCIO DE COMBUSTIVEIS LTDA	88.587.589/0001-17	AVENIDA GUILHERME SCHELL	6340	None	CENTRO	92310-000	ETANOL
2	7216755	S	RS	CANOAS	METROPOLITANO COMERCIO DE COMBUSTIVEIS LTDA	88.587.589/0001-17	AVENIDA GUILHERME SCHELL	6340	None	CENTRO	92310-000	GNV
3	7216756	NE	BA	ITABUNA	LOPES LEMOS COMERCIO DE COMBUSTIVEIS LTDA	00.231.792/0001-05	RODOVIA BR 101	SN	KM 503 5	MANOEL LEAO	45601-402	GASOLINA

Figura 12 - Leitura e verificação do novo *dataset*

Pré-Análise Utilizando o Pandas

Com o *dataframe* criado, contendo todos os dados, se faz necessário a pré-análise com o objetivo de identificar quais são as colunas e os seus tipos de dados (figura 13), se existem valores nulos (figura 14) e verificar se existem inconsistências nas colunas (segue exemplo na figura 15), além de verificar algumas frequências (figuras 16 e 17) e porcentagens que alguns dados são apresentados (figuras 18 e 19).

```
2 dfproj.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1202792 entries, 0 to 1202791
Data columns (total 17 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   id              1202792 non-null  int64
 1   Regiao_Sigla    1202792 non-null  object
 2   Estado_Sigla    1202792 non-null  object
 3   Municipio       1202792 non-null  object
 4   Revenda         1202792 non-null  object
 5   CNPJ_Revenda    1202792 non-null  object
 6   Rua             1202792 non-null  object
 7   Numero          1202191 non-null  object
 8   Complemento     283835 non-null   object
 9   Bairro          1199052 non-null  object
 10  CEP             1202792 non-null  object
 11  Produto         1202792 non-null  object
 12  Data_Coleta     1202792 non-null  object
 13  Valor_Venda     1202792 non-null  object
 14  Valor_Compra    217767 non-null   object
 15  Unidade_Medida  1202792 non-null  object
 16  Bandeira        1202792 non-null  object
dtypes: int64(1), object(16)
memory usage: 156.0+ MB
```

Figura 13 - Verificando informações do *dataframe*

```
2 dfproj.isnull().sum()
```

```
id                    0
Regiao_Sigla          0
Estado_Sigla          0
Municipio             0
Revenda               0
CNPJ_Revenda          0
Rua                   0
Numero              601
Complemento      918957
Bairro             3740
CEP                   0
Produto               0
Data_Coleta           0
Valor_Venda           0
Valor_Compra     985025
Unidade_Medida        0
Bandeira              0
dtype: int64
```

Figura 14 - Verificando valores nulos

```
2 dfproj["Estado_Sigla"].unique()
```

```
array(['RS', 'BA', 'PR', 'AC', 'AL', 'AM', 'AP', 'CE', 'DF', 'ES', 'GO',
       'TO', 'MA', 'MG', 'MS', 'MT', 'PA', 'PB', 'PE', 'PI', 'RJ', 'RN',
       'RO', 'RR', 'SC', 'SE', 'SP'], dtype=object)
```

Figura 15 - Exemplo de verificação de inconsistências na coluna

```
2 dfproj["Bandeira"].value_counts().head(5)
```

```
BRANCA                          423626
IPIRANGA                        203849
RAIZEN                          191651
PETROBRAS DISTRIBUIDORA S.A.    176376
VIBRA ENERGIA                   102831
Name: Bandeira, dtype: int64
```

Figura 16 - Frequência de bandeiras

```
2 dfproj["Produto"].value_counts()
```

```
GASOLINA              336675
ETANOL                300108
DIESEL S10            276109
DIESEL                164451
GASOLINA ADITIVADA    104444
GNV                    21005
Name: Produto, dtype: int64
```

Figura 17 - Frequência de tipos de combustíveis

```
2 dfproj["Bandeira"].value_counts(normalize=True).map("{:.2%}".format).head(5)
```

```
BRANCA                          35.22%
IPIRANGA                        16.95%
RAIZEN                          15.93%
PETROBRAS DISTRIBUIDORA S.A.    14.66%
VIBRA ENERGIA                    8.55%
Name: Bandeira, dtype: object
```

Figura 18 - Porcentagem das cinco maiores bandeiras

```
2 dfproj["Produto"].value_counts(normalize=True).map("{:.2%}".format)
```

```
GASOLINA              27.99%
ETANOL                24.95%
DIESEL S10            22.96%
DIESEL                13.67%
GASOLINA ADITIVADA     8.68%
GNV                    1.75%
Name: Produto, dtype: object
```

Figura 19 - Porcentagem dos combustíveis

Tratamento Utilizando o Pandas

Feitas as pré-análises, executamos as seguintes ações listadas a seguir, porém antes de iniciarmos as ações, fizemos uma cópia de segurança do *dataframe* (conforme figura 20).

```
2 dfproj_backup = dfproj.copy()
```

Figura 20 - Backup do *dataframe*

1. “Dropar” as colunas: ID, Rua, Bairro, Numero, Complemento, por não serem consideradas importantes para nosso objetivo (*Insights* finais), e por ter o CEP como informação redundante e dropar a coluna Valor_Compra por aproximadamente 82% dos valores serem nulos – conforme figura 21;
2. Trocar as "," por "." na coluna Valor_Venda – conforme figura 22.

```
2 dfproj.drop(columns=["id","Rua","Numero","Complemento","Bairro","Valor_Compra"], inplace=True)
```

Figura 21 - *Drop* das colunas

```
1 #Alterando "," por "." na coluna Valor_Venda
2 dfproj["Valor_Venda"] = dfproj["Valor_Venda"].str.replace(",",".")
```

Figura 22 - Alterando a "," por "." na coluna Valor_Venda

Depois da coluna id dropada utilizamos um comando para verificar se existem valores duplicados (figura 23) e a resposta foi: zero valores duplicados.

```
2 dfproj.duplicated().sum()

0
```

Figura 23 - Verificação de valores duplicados

Para uma melhor referência com os títulos das colunas, renomeamos conforme figura 24.

```
2 dfproj.columns=["Regiao","Estado","Municipio","Revenda","CNPJ","CEP","Produto",\
3                 "Data","Valor_Venda","Unidade_Medida","Bandeira"]
```

Figura 24 - Renomeando colunas

Após estes pré-tratamentos utilizando a biblioteca Pandas, salvamos este *dataframe* (transformando-o em um *dataset* do tipo *parquet*) dentro de Bucket no GCP – conforme figura 25.

```
dfproj.to_parquet("gs://projeto-final-equipe1/Pre-Tratado/Combustiveis.pq", index=False)
```

Figura 25 - Salvando arquivo pré-tratado no formato *parquet*

Lendo do Dados com PySpark

Uma das exigências do Projeto Final foi trabalhar com a biblioteca PySpark. Para atender esta demanda realizamos a conexão (figura 26), criar o esquema (figura 27) e executamos a leitura do arquivo no formato *parquet* (figura 28) salvo anteriormente no Bucket intitulado de Pre-Tratado.

```
spark = (
    SparkSession.builder
                .master ('local')
                .appName('projeto-final')
                .config('spark.ui.port','4050')
                .config('spark.jars','https://storage.googleapis.com/hadoop-lib/gcs/gcs-connector-hadc
                .getOrCreate()
```

Figura 26 - Conexão com a *sparksession*

```
my_schema = (StructType([
                        StructField("Regiao",StringType(),nullable = False),
                        StructField("Estado",StringType(),nullable = False),
                        StructField("Municipio",StringType(),nullable = False),
                        StructField("Revenda",StringType(),nullable = False),
                        StructField("CNPJ",StringType(),nullable = False),
                        StructField("CEP",StringType(),nullable = False),
                        StructField("Produto",StringType(),nullable = False),
                        StructField("Data",StringType(),nullable = False),
                        StructField("Valor_Venda",StringType(),nullable = False),
                        StructField("Unidade_Medida",StringType(),nullable = False),
                        StructField("Bandeira",StringType(),nullable = False),
])
)
```

Figura 27 - Criação do esquema

```
dfSpark = (spark.read.format("parquet")
                .option("header", True)
                .option("delimiter", ",")
                .load("gs://projeto-final-equipe1/Pre-Tratado/Combustiveis.pq", schema=my_schema)
)
```

Figura 28 - Fazendo leitura no *bucket* GCP - *Spark*

Tratamento com PySpark

Com o objetivo de atender mais um critério do Projeto Final, utilizamos alguns métodos do *PySpark* para fazer verificações, conversões e criação, como ilustrado nas figuras 29, 30 e 31 respectivamente.

```
2 dfSpark.printSchema()

root
 |-- Regiao: string (nullable = true)
 |-- Estado: string (nullable = true)
 |-- Municipio: string (nullable = true)
 |-- Revenda: string (nullable = true)
 |-- CNPJ: string (nullable = true)
 |-- CEP: string (nullable = true)
 |-- Produto: string (nullable = true)
 |-- Data: string (nullable = true)
 |-- Valor_Venda: string (nullable = true)
 |-- Unidade_Medida: string (nullable = true)
 |-- Bandeira: string (nullable = true)
```

Figura 29 - Verificando tipo de dados no Spark

```
2 dfSpark = dfSpark.withColumn("Valor_Venda" ,dfSpark["Valor_Venda"].cast(FloatType()))
```

Figura 30 - Convertendo tipo de dados no Spark

```
2 dfSpark = dfSpark.withColumn(("Ano"),F.year("Data")).withColumn("Mes",F.month("Data"))
```

Figura 31 - Criando colunas no Spark

Filtrando e Agrupando com PySpark

Para atender mais um desafio do Trabalho Final criou-se filtros e agrupamentos utilizando a biblioteca PySpark. Na figura 32 é apresentado um filtro com agrupamento que tem como objetivo mostrar a média, valor máximo, o valor mínimo de cada produto do ano de 2019.

```
ano2019 = dfSpark["Ano"] == 2019
(dfSpark.filter(ano2019).groupBy("Produto").agg(F.mean("Valor_Venda").alias("Media_Preco"),
        F.max("Valor_Venda").alias("Preco_Max"),F.min("Valor_Venda").alias("Preco_Min")).show())
```

```
+----------+------------------+---------+---------+
|   Produto|       Media_Preco|Preco_Max|Preco_Min|
+----------+------------------+---------+---------+
|DIESEL S10| 3.732812434513274|    5.089|    2.999|
|    DIESEL|3.6475548843968655|     4.99|    2.999|
|    ETANOL| 3.161922439504206|     5.47|    2.099|
|       GNV|3.2474904105460367|    4.559|    2.489|
|  GASOLINA|4.4251792166388055|    5.859|    3.559|
+----------+------------------+---------+---------+
```

Figura 32 - Filtro e agrupamento utilizando PySpark

Utilizando o SparkSQL

Outro desafio foi a utilização do SparkSQL onde conseguimos trazer insight representando as bandeiras mais presentes em cada região, porém antes de utilizarmos o comando spark.sql precisamos definir uma consulta de visualização, ambos comandos são visualizados na figura 33.

```
[75]  #Criar tabela para realização de consultas
      dfSpark.createOrReplaceTempView("combustiveis")
```

Bandeiras mais presentes em cada região

```
spark.sql("WITH n AS (SELECT Bandeira, Regiao, count(Bandeira) AS Frequencia FROM combustiveis \
WHERE Regiao = 'N' GROUP BY Bandeira, Regiao ORDER BY Frequencia DESC LIMIT 10), \
ne AS (SELECT Bandeira, Regiao, count(Bandeira) AS Frequencia FROM combustiveis \
WHERE Regiao = 'NE' GROUP BY Bandeira, Regiao ORDER BY Frequencia DESC LIMIT 10), \
s AS (SELECT Bandeira, Regiao, count(Bandeira) AS Frequencia FROM combustiveis \
WHERE Regiao = 'S' GROUP BY Bandeira, Regiao ORDER BY Frequencia DESC LIMIT 10), \
se AS (SELECT Bandeira, Regiao, count(Bandeira) AS Frequencia FROM combustiveis \
WHERE Regiao = 'SE' GROUP BY Bandeira, Regiao ORDER BY Frequencia DESC LIMIT 10), \
co AS (SELECT Bandeira, Regiao, count(Bandeira) AS Frequencia FROM combustiveis \
WHERE Regiao = 'CO' GROUP BY Bandeira, Regiao ORDER BY Frequencia DESC LIMIT 10) \
\
SELECT n.Bandeira, n.Regiao, n.Frequencia, ne.Regiao, ne.Frequencia, s.Regiao, s.Frequencia, se.Regiao, se.Frequencia,\
co.Regiao, co.Frequencia FROM n INNER JOIN ne ON n.Bandeira = ne.Bandeira INNER JOIN s ON ne.Bandeira = s.Bandeira \
INNER JOIN se ON s.Bandeira = se.Bandeira INNER JOIN co ON se.Bandeira = co.Bandeira ORDER BY n.Frequencia DESC").show(truncate = False)
```

```
+--------------------------+------+---------+------+---------+------+---------+------+---------+------+---------+
|Bandeira                  |Regiao|Frequencia|Regiao|Frequencia|Regiao|Frequencia|Regiao|Frequencia|Regiao|Frequencia|
+--------------------------+------+---------+------+---------+------+---------+------+---------+------+---------+
|BRANCA                    |N     |15886    |NE    |81164    |S     |56841    |SE    |225338   |CO    |44397    |
|PETROBRAS DISTRIBUIDORA S.A.|N     |13544    |NE    |43511    |S     |24680    |SE    |76971    |CO    |17670    |
|IPIRANGA                  |N     |12961    |NE    |20428    |S     |51963    |SE    |104825   |CO    |13672    |
|VIBRA ENERGIA             |N     |7831     |NE    |25337    |S     |14765    |SE    |45559    |CO    |9339     |
|RAIZEN                    |N     |1651     |NE    |37717    |S     |24706    |SE    |112799   |CO    |14778    |
|ALESAT                    |N     |423      |NE    |7169     |S     |2760     |SE    |14920    |CO    |2095     |
+--------------------------+------+---------+------+---------+------+---------+------+---------+------+---------+
```

Figura 33 - Comando SparkSQL

Fazendo o Processo de *Load*

Para finalizar o processo de ETL deste trabalho, realizou-se o carregamento (*load*) dos dados através da inserção do arquivo final no formato parquet no bucket da GCP intitulado Tratados (figura 34) e no banco de dados não relacional MondoDB (figura 35).

```
(dfSpark.write.format("parquet").option("header",True).
 save("gs://projeto-final-equipe1/Tratados/Combustiveis-Tratados.pq",mode="overwrite"))
```

Figura 34 - *Load* em parquet no *bucket* GCP

```
# Transformando o parquet tratado em df pandas
df_tratado = pd.read_parquet("gs://projeto-final-equipe1/Tratados/Combustiveis-Tratados.pq/part-00000-15ebaaa9-
```

```
#Transformando a coluna data para o formato string
df_tratado["Data"] = df_tratado["Data"].astype(str)
```

```
# Transformando o DataFrame em dicionário
dados = df_tratado.to_dict('records')
```

```
# Inserindo os dados no banco MongoDB
collection_name.insert_many(dados)
```

```
<pymongo.results.InsertManyResult at 0x7f4bb72da910>
```

Figura 35 - *Load* em parquet para o MongoDB

Pipeline

Foi criado um modelo de Pipeline (figura 36) onde é realizada a leitura do arquivo *parquet* com os dados tratados, em seguida feito um filtro para cada ano (2019, 2020 e 2021), a partir daí obtêm-se três arquivos separados. Para execução desse modelo de Pipeline foi usado o *DataFlow* (figura 37), uma ferramenta dentro da plataforma GCP.

```
#Criação da Pipeline
p1 = beam.Pipeline(options=pipeline_options)

combustiveis2019=(
    p1
    |"01.Leitura dos datasets">> beam.io.ReadFromParquet(path)
    |"02.Filtrar por ano">> beam.Filter(lambda record: record["Ano"] == 2019)
    |'03.Salvar resultado'>> beam.io.WriteToParquet("gs://projeto-final-equipe1/Pipeline/CombustiveisAno2019",schema=schema_map)
)
combustiveis2020=(
    p1
    |"11.Leitura dos datasets">> beam.io.ReadFromParquet(path)
    |"12.Filtrar por ano">> beam.Filter(lambda record: record["Ano"] == 2020)
    |'13.Salvar resultado'>> beam.io.WriteToParquet("gs://projeto-final-equipe1/Pipeline/CombustiveisAno2020",schema=schema_map)
)
combustiveis2021=(
    p1
    |"21.Leitura dos datasets">> beam.io.ReadFromParquet(path)
    |"22.Filtrar por ano">> beam.Filter(lambda record: record["Ano"] == 2021)
    |'23.Salvar resultado'>> beam.io.WriteToParquet("gs://projeto-final-equipe1/Pipeline/CombustiveisAno2021",schema=schema_map)
)
p1.run()
```

Figura 36 - Criação do modelo Pipeline

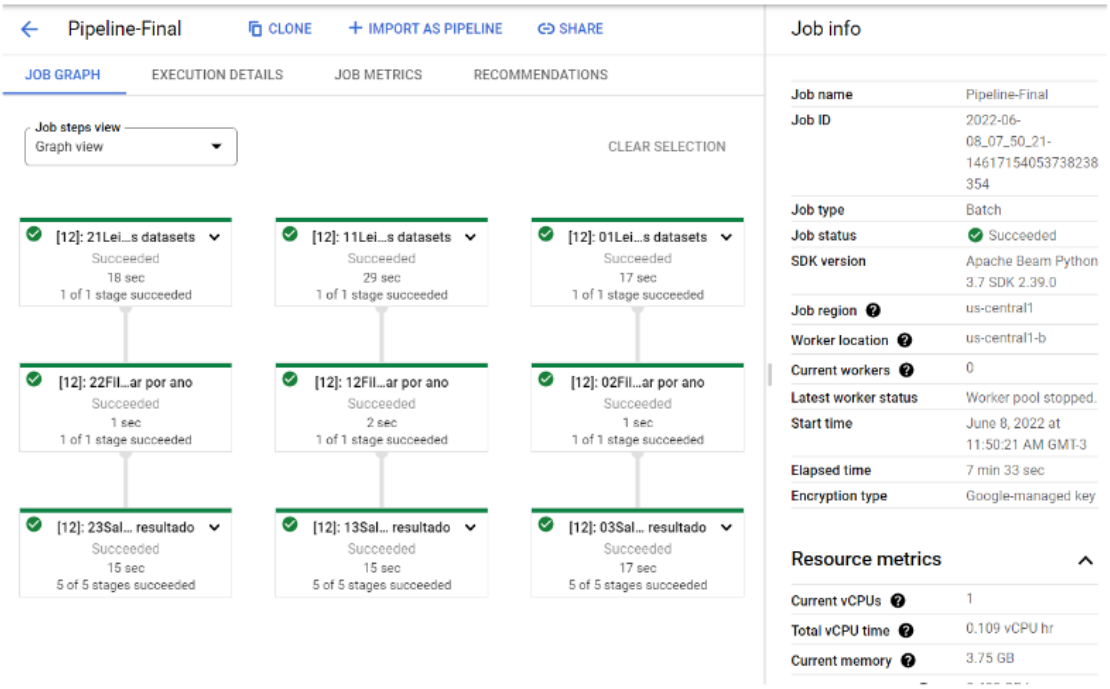

Figura 37 - Pipeline executada no *DataFlow*

Big Query

Um dos requisitos obrigatórios do nosso trabalho foi estruturar um *datawarehouse*, onde fizemos a estrutura no Big Query ferramenta utilizada dentro da plataforma do GCP. Fizemos a importação dos dados tratados do *Bucket* diretamente para o Big Query, com o objetivo de realizar consultas e *insights* primários (conforme figura 38).

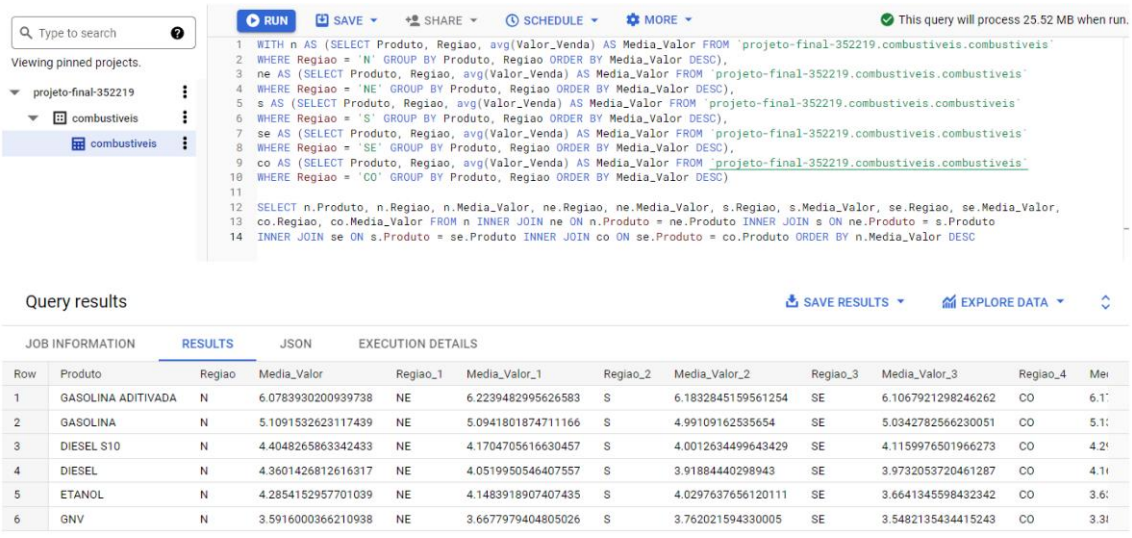

Figura 38 - Consulta Big Query

Plotagem com Pandas

Em cima do *insight* realizado na Big Query, realizamos a plotagem com Pandas utilizando a Função Pivot_table que permite realizar agrupamentos e funções de agregação de acordo com as análises que pretendemos sobre os dados. Os dados de combustíveis são distribuídos em seis categorias (conforme figura 39). É possível observar uma leve queda nos valores médios para todos os combustíveis no ano de 2020 em relação ao ano de 2019 (figura 40).

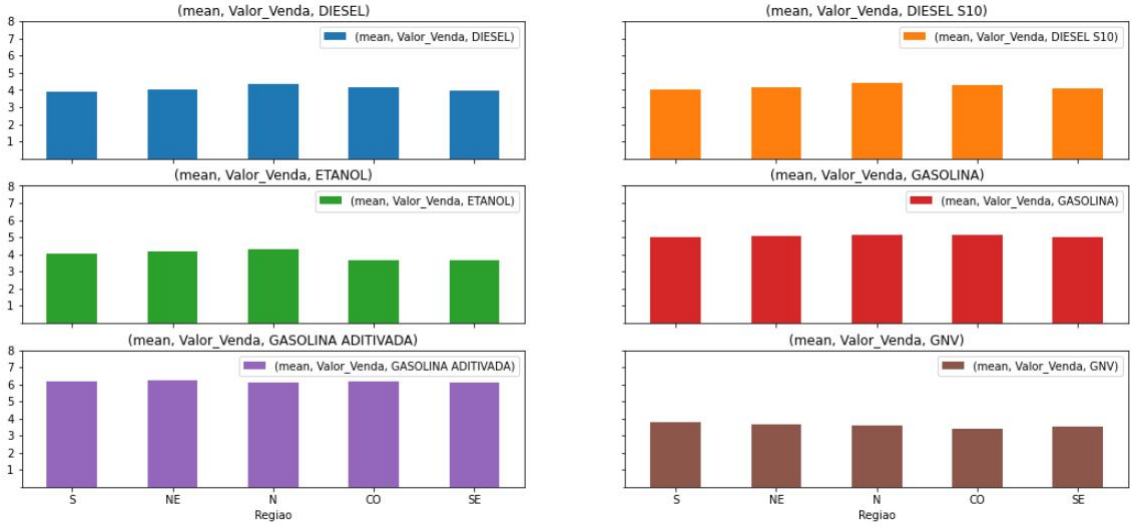

Figura 39 - Média de valor de venda de cada combustível por região

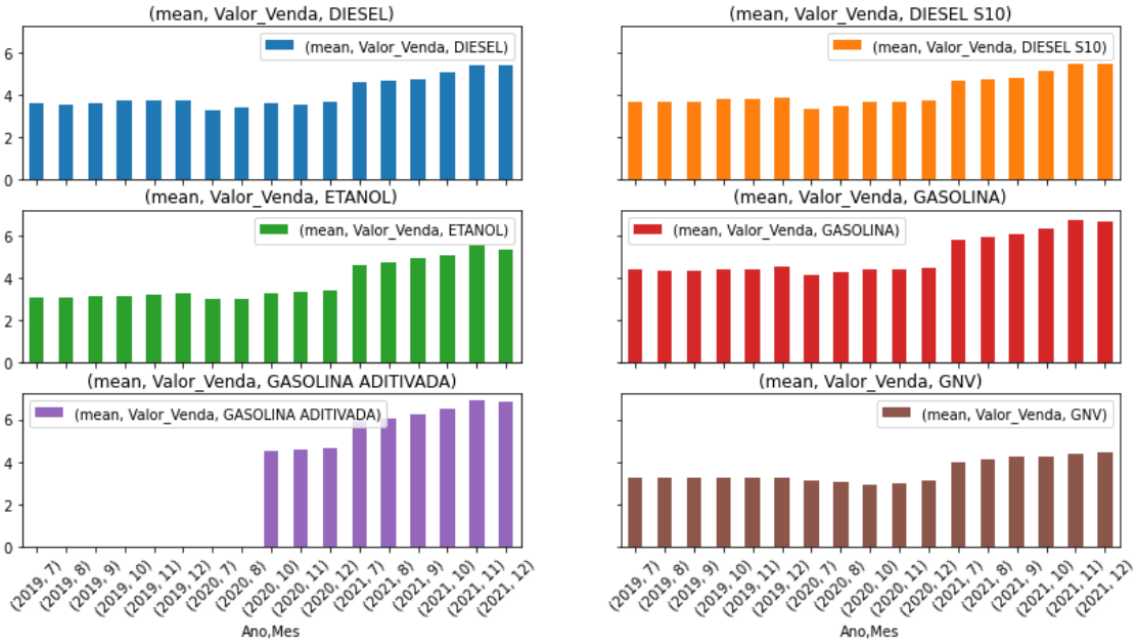

Figura 40 - Média de variação de combustíveis

Google Data Studio

Foi utilizada a ferramenta Google Data Studio para apresentar os *insights* referentes aos valores médios de vendas dos combustíveis, as incidências das bandeiras no Brasil, as variações das médias dos combustíveis por mês e um consolidado da variação dos segundos semestres dos anos de 2019, 2020 e 2021. As figuras 41 e 42 seguem como exemplo do trabalho realizado.

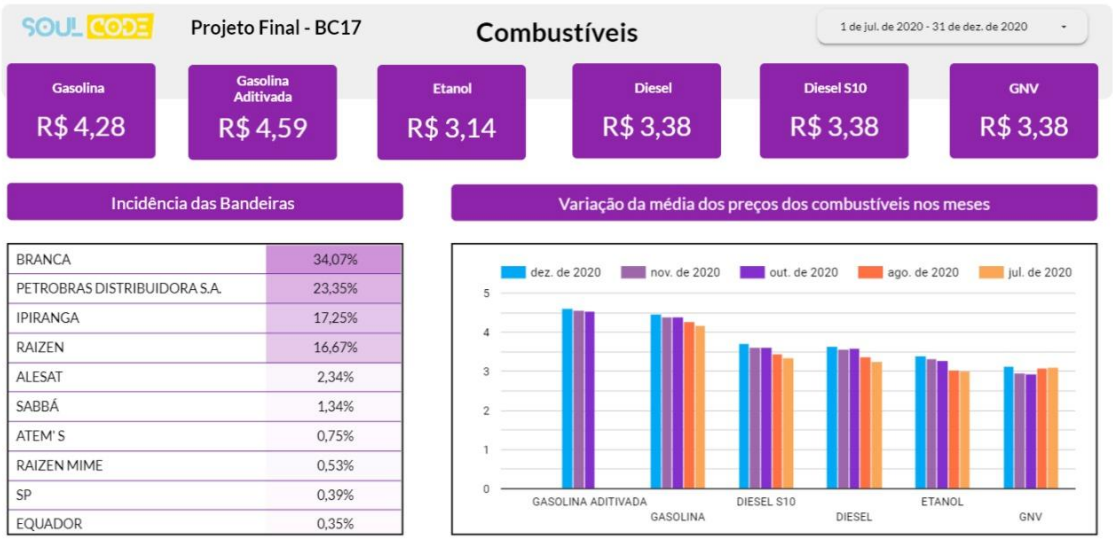

Figura 41 - Dashboard principal

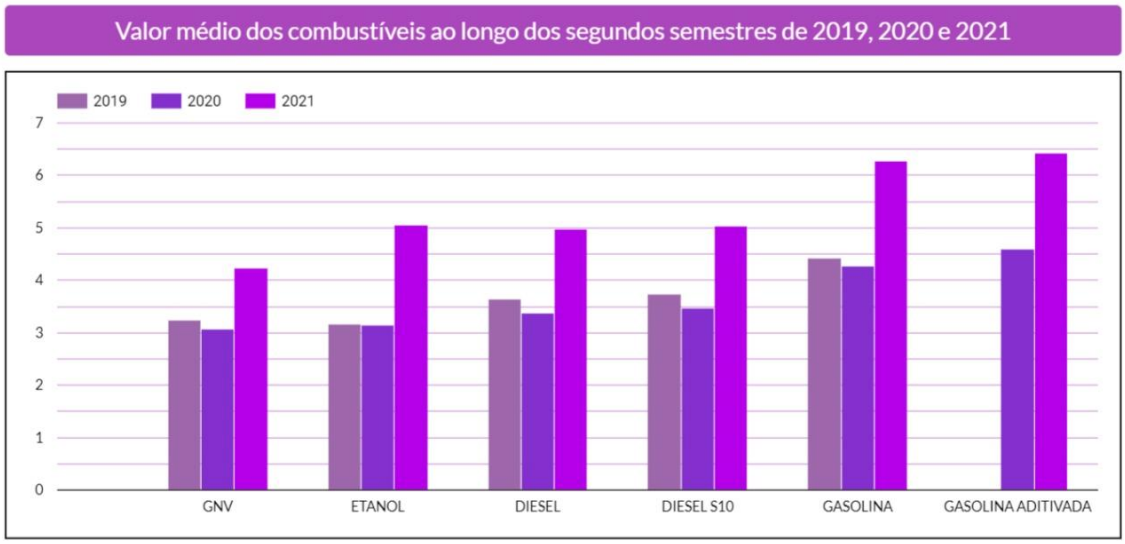

Figura 42 - Consolidado por combustíveis

Considerações Finais

Percebemos que houve um aumento gradual no segundo semestre de 2019 e em 2020, verificamos que, apesar do aumento gradual que se deu nos últimos meses, a média dos preços desses combustíveis se manteve ou até diminuiu comparado com o ano anterior. Porém, para os dados de 2021, é notável que houve um aumento significativo, representando no mínimo 34% de alta, sendo que o etanol, que foi o combustível que mais sofreu aumento, ficou 45% mais caro para o consumidor.

Finalizando nossas análises, fizemos um consolidado do valor médio de venda dos diferentes tipos de combustíveis nos segundos semestre de 2019, 2020 e 2021 e ficou visível o aumento significativo que constatamos em 2021.

Referências

APACHE_BEAM.IO.PARQUETIO module. [S. l.], sem data. Disponível em: https://beam.apache.org/releases/pydoc/2.11.0/apache_beam.io.parquetio.html. Acesso em: 8 jun. 2022.

SPADINI, Allan Segovia. **Data Lake vs Data Warehouse**. [*S. l.*], 16 ago. 2021. Disponível em: https://www.alura.com.br/artigos/data-lake-vs-data-warehouse?gclid=Cj0KCQjwwJuVBhCAARIsAOPwGATNCdh6Ri3F2NfzV4B07W4MLb0a8CMuI7JOZgHDx372hiIJEI5f0N0aAoBoEALw_wcB. Acesso em: 6 jun. 2022.

USER Guide. [S. l.], 2008-2020. Disponível em: https://pandas.pydata.org/pandas-docs/version/1.1/user_guide/index.html. Acesso em: 6 jun. 2022.

USER Guide. [S. l.], sem data. Disponível em: https://spark.apache.org/docs/latest/api/python/user_guide/index.html. Acesso em: 6 jun. 2022.