Solução:

Uso do software LibreOffice, único que consegui visualizar as abas de dados da planilha original: vendas-combustiveis-m3.xls

Interface gráfica do usuário, Aplicativo, Tabela, Excel

Descrição gerada automaticamente

Notas:

Tentativa de ler em python, com algumas bibliotecas, o arquivo em formato .XLS falhou

Foi necessário um trabalho manual de separar as abas em arquivos em formato .CSV

As abas DPCache\_m3, DPCache\_m3\_2 e DPCache\_m3\_3 apresentaram dados e colunas equivalentes, assim juntei todas numa única aba para leitura das informações.

As abas DPCache\_m3\_4, DPCache\_m3\_5 e DPCache\_m3\_6 apresentaram quebra nas formações nas linhas com ano 2020, acabei desprezando essas abas pois a única informação diferente era a coluna de “SEGMENTO“ em relação as outras que tinha a coluna “COMBUSTIVEL (product)”.

Interface gráfica do usuário, Aplicativo, Tabela, Excel

Descrição gerada automaticamente

**Código Python**

Bibliotecas:

from pyspark.sql.functions import split, col, row\_number, expr, row\_number, current\_timestamp

from pyspark.sql import SparkSession

from pyspark.sql.window import Window

from pyspark import HiveContext

from pyspark.sql import Row, functions as F, types as T

from pyspark.sql.window import Window

from pyspark.sql.functions import col, lit, split, input\_file\_name, substring, regexp\_replace, trim, when, add\_months, to\_date, concat, coalesce, months\_between, max, sum, count, avg

from datetime import datetime, timedelta

from functools import reduce

Código de leitura e tratamento de dados

Nota: os valores totalizados por ano e mês não ficaram batidos com a planilha, portanto o indicador de Variação acumulado também não bateu

df = spark.read.load('abfss://landing-zone@romagnoledatalake1.dfs.core.windows.net/z\_Lab/anp\_prod\_ok.csv', format='csv', header=True, sep=";")

colunas\_a\_manter = ['product', 'year', 'unit', 'uf']

#realizando pivot das colunas de Meses para Linhar

df\_melted = df.select(colunas\_a\_manter + [expr("stack(12, 'JAN', JAN, 'FEV', FEV, 'MAR', MAR, 'ABR', ABR, 'MAI', MAI, 'JUN', JUN, 'JUL', JUL, 'AGO', AGO, 'SET', SET, 'OUT', OUT, 'NOV', NOV, 'DEZ', DEZ) as (month, volume)")])

#tipagem de campo e geração de campo Mês numérico (para ordenação)

df = df\_melted\

    .withColumn('year', col('year').cast('int'))\

    .withColumn('volume',when(col('volume').isNotNull(), col('volume')).otherwise(0).cast('float'))\

    .withColumn('created\_at',current\_timestamp())\

    .withColumn('month\_no',\

        when(col('month') == 'JAN', 1).\

        when(col('month') == 'FEV', 2).\

        when(col('month') == 'MAR', 3).\

        when(col('month') == 'ABR', 4).\

        when(col('month') == 'MAI', 5).\

        when(col('month') == 'JUN', 6).\

        when(col('month') == 'JUL', 7).\

        when(col('month') == 'AGO', 8).\

        when(col('month') == 'SET', 9).\

        when(col('month') == 'OUT', 10).\

        when(col('month') == 'NOV', 11).\

        when(col('month') == 'DEZ', 12).\

        otherwise(0).cast('int'))

#filtro de registros apenas com valores válidos

df = df.filter(df['volume'].cast('string').rlike('^[0-9.]+$'))

#display(df.limit(12)

df\_pivot = df.groupBy("month", "month\_no").pivot("year").agg(sum("volume")).orderBy("month\_no")

agg\_cols = df\_pivot.columns[1:]

rollup\_df = df\_pivot.rollup().sum()

renamed\_df = reduce(

    lambda rollup\_df, idx: rollup\_df.withColumnRenamed(rollup\_df.columns[idx], agg\_cols[idx]),

    range(len(rollup\_df.columns)), rollup\_df

)

renamed\_df = renamed\_df.withColumn('month', lit('Total'))

df\_pivot.unionByName(

    renamed\_df

)

colunas = df\_pivot.columns[2:-1]

# Crie uma janela de especificação para a acumulação

window\_spec = Window.orderBy()  # Você pode especificar a ordem conforme necessário

# Calcule a soma acumulativa das duas últimas colunas

for coluna in colunas[-2:]:

    df\_pivot = df\_pivot.withColumn(coluna + '\_cumsum', sum(col(coluna)).over(window\_spec))

# Calcule a nova coluna acumulativa com base nas duas últimas colunas

df\_pivot = df\_pivot.withColumn('variacao acumulado 19-20',

                   when((col(colunas[-2] + '\_cumsum') + col(colunas[-1] + '\_cumsum') == 0), "n/d")

                   .otherwise((col(colunas[-1] + '\_cumsum') / col(colunas[-2] + '\_cumsum') - 1) \* 100))

# Mostre o DataFrame resultante

df\_pivot.show()

#display(final\_df.limit(12))

Resultado do dataframe:

Tabela

Descrição gerada automaticamente

Arquivo disponível na “consume zone”



Extração dados finais em formato Parquet:

Tabela

Descrição gerada automaticamente

Script SQL para gerar tabela no Datawarehouse

IF NOT EXISTS (SELECT \* FROM sys.external\_file\_formats WHERE name = 'SynapseParquetFormat')

    CREATE EXTERNAL FILE FORMAT [SynapseParquetFormat]

    WITH ( FORMAT\_TYPE = PARQUET)

GO

IF NOT EXISTS (SELECT \* FROM sys.external\_data\_sources WHERE name = 'consume-dominiodatalake1\_dfs\_core\_windows\_net')

    CREATE EXTERNAL DATA SOURCE [consume-zone\_romagnoledatalake1\_dfs\_core\_windows\_net]

    WITH (

        LOCATION = 'abfss://consume-zone@dominiodatalake1.dfs.core.windows.net'

    )

GO

CREATE EXTERNAL TABLE dbo.XLS\_CSV\_teste (

    [product] nvarchar(4000),

    [year] int,

    [unit] nvarchar(4000),

    [uf] nvarchar(4000),

    [month] nvarchar(4000),

    [volume] real,

    [created\_at] datetime2(7),

    [month\_no] int

    )

    WITH (

    LOCATION = 'TI/XLS\_CVS\_teste/\*\*',

    DATA\_SOURCE = [consume-zone\_romagnoledatalake1\_dfs\_core\_windows\_net],

    FILE\_FORMAT = [SynapseParquetFormat]

    )

GO

SELECT TOP 100 \* FROM dbo.XLS\_CSV\_teste

GO