

MVP3 - Seguros (Python)

[Import notebook](#)

Análise das Informações de uma Seguradora

O objetivo desta análise é identificar:

- Prêmios Emitidos Total por mês
- Sinistros Emitidos por mês
- Quais os produtos que mais vendem
- Comparação PRÊMIOS EMITIDOS X SINISTROS AVISADOS - Os 3 produtos mais vendidos

O código do programa dividi-se em:

1 - Leitura e Gravação dos arquivos nos diretórios

- 1.1 - Área das Funções
- 1.2 - Programa Principal

2 - Criar schemas - Bronze, Prata, Ouro

- 2.1 - Programa Principal

3 - Atualizar schemas

- 3.1 - Atualizar BRONZE
 - 3.1.1 - Atualizar Bronze - Área de Funções
 - 3.1.2 - Atualizar Bronze - Programa Principal
- 3.2 - Atualizar PRATA
 - 3.2.1 - Atualizar PRATA - Área de Funções
 - 3.2.2 - Atualizar PRATA - Programa Principal
- 3.3 - Atualizar OURO
 - 3.3.1 - Atualizar OURO - Área de Funções
 - 3.3.2 - Atualizar OURO - Programa Principal

4 - Área dos Gráficos

- 4.1 - Área das Funções
- 4.2 - Programa Principal
 - 4.2.1 - Prêmios Emitidos - GERAL
 - 4.2.2 - Sinistros Avisados - GERAL
 - 4.2.3 - Prêmios Emitidos X Sinistros Pagos - TOP 3 Produtos

```
; import requests  
import zipfile  
import io  
import os  
import shutil  
import csv  
import datetime  
  
!pip install pandas matplotlib  
import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
  
from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, sum as _sum  
  
!pip install tabulate  
from tabulate import tabulate  
  
from pyspark.sql.functions import col, concat, lpad, when
```

```
9/site-packages (from pandas) (1.21.5)  
Requirement already satisfied: pytz>=2020.1 in /databricks/python3/lib/python3.  
9/site-packages (from pandas) (2021.3)  
Requirement already satisfied: pyparsing>=2.2.1 in /databricks/python3/lib/pytho  
n3.9/site-packages (from matplotlib) (3.0.4)  
Requirement already satisfied: packaging>=20.0 in /databricks/python3/lib/python  
3.9/site-packages (from matplotlib) (21.3)  
Requirement already satisfied: cycler>=0.10 in /databricks/python3/lib/python3.  
9/site-packages (from matplotlib) (0.11.0)  
Requirement already satisfied: kiwisolver>=1.0.1 in /databricks/python3/lib/pyth  
on3.9/site-packages (from matplotlib) (1.3.2)  
Requirement already satisfied: pillow>=6.2.0 in /databricks/python3/lib/python3.  
9/site-packages (from matplotlib) (9.0.1)  
Requirement already satisfied: fonttools>=4.22.0 in /databricks/python3/lib/pyth  
on3.9/site-packages (from matplotlib) (4.25.0)  
Requirement already satisfied: six>=1.5 in /databricks/python3/lib/python3.9/sit  
e-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)  
WARNING: You are using pip version 21.2.4; however, version 25.0.1 is available.  
You should consider upgrading via the '/local_disk0/.ephemeral_nfs/envs/pythonEn  
v-369e9284-e9b7-409e-b13a-80bc734b098e/bin/python -m pip install --upgrade pip'  
command.
```

```
Collecting tabulate  
  Downloading tabulate-0.9.0-py3-none-any.whl (35 kB)  
Installing collected packages: tabulate  
Successfully installed tabulate-0.9.0  
WARNING: You are using pip version 21.2.4; however, version 25.0.1 is available.
```

You should consider upgrading via the '/local_disk0/.ephemeral_nfs/envs/pythonEnv-369e9284-e9b7-409e-b13a-80bc734b098e/bin/python -m pip install --upgrade pip' command.

1 - Leitura e gravação dos arquivos nos diretórios

1.1 - Área das Funções

7

```
# Apagar e criar os diretórios no DBFS
def apagar_e_criar_diretorios(dir_zip,dir_csv):

    dbutils.fs.rm(dir_zip, recurse=True)
    dbutils.fs.rm(dir_csv, recurse=True)

    print("Diretórios apagados com sucesso.")

    dbutils.fs.mkdirs(dir_zip)
    dbutils.fs.mkdirs(dir_csv)

    print("Diretórios Criados.")
```

8

```
# Função para baixar e extrair arquivos ZIP
def baixar_e_extrair_arquivos(dir_zip, dir_csv):

    import requests
    import zipfile
    import os
    # Definindo a URL do arquivo
    prefixo_url = 'https://raw.githubusercontent.com/phfgoes1969/PUC-RJ-MVP3/main/'

    nomes_arquivos = ['RAMOS.zip', 'SUB_RAMOS.zip', 'DOCFAT.zip',
    'SINISTROS.zip',
    'SINISTROS_PGT0.zip']

    for nome in nomes_arquivos:
        arquivo = nome
        url = f"{prefixo_url}{arquivo}"

        # Fazendo o download do arquivo
        response = requests.get(url)
        response.raise_for_status() # Verifica se houve algum erro na requisição

        # Salvando o arquivo no diretório especificado
        caminho_destino = "/dbfs" + dir_zip + '/' + arquivo
        # Diretório onde os arquivos serão extraídos
        diretorio_extracao = "/dbfs" + dir_csv

        # Certifique-se de que o diretório de destino existe
        os.makedirs(os.path.dirname(caminho_destino), exist_ok=True)

        with open(caminho_destino, 'wb') as f:
            f.write(response.content)

        response = requests.head(url)

        if response.status_code == 200:
            print(f'O arquivo {arquivo} está disponível na URL.')
        else:
            print(f'O arquivo {arquivo} não está disponível na URL. Status code: {response.status_code}')

        # Descompactar o arquivo ZIP
        with zipfile.ZipFile(caminho_destino, 'r') as zip_ref:
            zip_ref.extractall(diretorio_extracao)
```

```
print(f"Arquivo {caminho_destino} descompactado em  
{diretorio_extracao}")
```

1.2 - Programa Principal

10

```
# Apagar Diretórios, caso necessário  
  
dir_zip = 'dbfs:/FileStore/tables/arquivos_zip' # Diretório para armazenar  
os arquivos ZIP  
dir_csv = 'dbfs:/FileStore/tables/arquivos_csv' # Diretório para armazenar  
  
apagar_e_criar_diretorios(dir_zip,dir_csv)  
  
dir_zip = "/FileStore/tables/arquivos_zip"  
dir_csv = "/FileStore/tables/arquivos_csv"  
  
apagar_e_criar_diretorios(dir_zip,dir_csv)
```

Diretórios apagados com sucesso.
Diretórios Criados.
Diretórios apagados com sucesso.
Diretórios Criados.

11

```
# Listar todos os arquivos e diretórios no caminho especificado VERIFICAÇÃO
arquivos_csv = dbutils.fs.ls("dbfs:/FileStore/tables/arquivos_csv")

arquivos_zip = dbutils.fs.ls("dbfs:/FileStore/tables/arquivos_zip")

# Lista área ZIP
print('Área arquivos_zip:')
if arquivos_zip:
    for arquivo in arquivos_zip:
        print(arquivo.path)
else:
    print('Diretório Vazio')

# Lista área CSV
print('Área arquivos_csv:')
if arquivos_csv:
    for arquivo in arquivos_csv:
        print(arquivo.path)
else:
    print('Diretório Vazio')
```

Área arquivos_zip:
Diretório Vazio
Área arquivos_csv:
Diretório Vazio

12

```
# Baixar os arquivos do GIT e gravo na pasta arquivos_csv

dir_zip = "/FileStore/tables/arquivos_zip"
dir_csv = "/FileStore/tables/arquivos_csv"
baixar_e_extrair_arquivos(dir_zip, dir_csv)

print('Movendo para a área correta:')
dbutils.fs.mv("file:/dbfs/FileStore/tables/arquivos_csv",
"dbfs:/FileStore/tables/arquivos_csv", recurse=True)
```

O arquivo RAMOS.zip está disponível na URL.
Arquivo /dbfs/FileStore/tables/arquivos_zip/RAMOS.zip descompactado em /dbfs/FileSto

```
re/tables/arquivos_csv  
O arquivo SUB_RAMOS.zip está disponível na URL.  
Arquivo /dbfs/FileStore/tables/arquivos_zip/SUB_RAMOS.zip descompactado em /dbfs/FileStore/tables/arquivos_csv  
O arquivo DOCFAT.zip está disponível na URL.  
Arquivo /dbfs/FileStore/tables/arquivos_zip/DOCFAT.zip descompactado em /dbfs/FileStore/tables/arquivos_csv  
O arquivo SINISTROS.zip está disponível na URL.  
Arquivo /dbfs/FileStore/tables/arquivos_zip/SINISTROS.zip descompactado em /dbfs/FileStore/tables/arquivos_csv  
O arquivo SINISTROS_PGT0.zip está disponível na URL.  
Arquivo /dbfs/FileStore/tables/arquivos_zip/SINISTROS_PGT0.zip descompactado em /dbfs/FileStore/tables/arquivos_csv  
Movendo para a área correta:  
Out[6]: True
```

13

```
print('Apagando a área arquivos_zip...')  
# Apago o diretório arquivos_zip. Não é mais usado  
dbutils.fs.rm(dir_zip, recurse=True)
```

```
Apagando a área arquivos_zip...  
Out[7]: True
```

14

```
# Lista, para conferência, os arquivos que estão no arquivos_csv  
arquivos_csv = dbutils.fs.ls("dbfs:/FileStore/tables/arquivos_csv")  
print('Área arquivos_csv:')  
if arquivos_csv:  
    for arquivo in arquivos_csv:  
        print(arquivo.path)  
else:  
    print('Diretório Vazio')
```

```
Área arquivos_csv:  
dbfs:/FileStore/tables/arquivos_csv/DOCFAT.csv  
dbfs:/FileStore/tables/arquivos_csv/RAMOS.csv  
dbfs:/FileStore/tables/arquivos_csv/SINISTROS.csv  
dbfs:/FileStore/tables/arquivos_csv/SINISTROS_PGT0.csv  
dbfs:/FileStore/tables/arquivos_csv/SUB_RAMOS.csv
```

2 - Criar schemas Bronze, Prata, Ouro

2.1 - Programa Principal

18

```
# Apagando todo os Schemas  
# Dropar o esquema BRONZE  
spark.sql("DROP SCHEMA IF EXISTS BRONZE CASCADE")  
  
# Dropar o esquema PRATA  
spark.sql("DROP SCHEMA IF EXISTS PRATA CASCADE")  
  
# Dropar o esquema OURO  
spark.sql("DROP SCHEMA IF EXISTS OURO CASCADE")
```

Out[9]: DataFrame[]

19

```
spark.sql("CREATE SCHEMA IF NOT EXISTS BRONZE")  
  
spark.sql("CREATE SCHEMA IF NOT EXISTS PRATA")  
  
spark.sql("CREATE SCHEMA IF NOT EXISTS OURO")
```

Out[10]: DataFrame[]

20

```
# Listar todos os esquemas  
schemas = spark.sql("SHOW SCHEMAS")  
schemas.show()
```

► schemas: pyspark.sql.dataframe.DataFrame = [databaseName: string]

```
+-----+  
|databaseName|  
+-----+  
|    bronze|  
|    default|  
|      ouro|  
|     prata|  
+-----+
```

3 - Atualizar Schemas

3.1 - Atualizar BRONZE

3.1.1 - Atualizar Bronze - Área de Funções

25

```
# Ler os arquivos CSVs e criar as tabelas

def ler_e_criar_tabelas(diretorio_csv, esquema_bronze):
    # Listar os arquivos no diretório
    arquivos = dbutils.fs.ls(diretorio_csv)

    for arquivo in arquivos:
        if arquivo.path.endswith(".csv"):
            # Extrair o nome do arquivo sem a extensão
            nome_tabela = arquivo.name.split(".")[0]

            # Ler o arquivo CSV
            df = spark.read.format("csv") \
                .option("header", "true") \
                .option("inferSchema", "true") \
                .option("delimiter", ";") \
                .load(arquivo.path)

            # Criar a tabela no esquema BRONZE
            df.write.mode("overwrite").saveAsTable(f"{esquema_bronze}." \
{nome_tabela}")
            print(f"Tabela {esquema_bronze}.{nome_tabela} criada com \
sucesso.")
```

3.1.2 - Atualizar Bronze - Programa Principal

27

```
# Ler o diretório CSV e criar as tabelas
diretorio_csv = "dbfs:/FileStore/tables/arquivos_csv"
esquema_bronze = "BRONZE"
ler_e_criar_tabelas(diretorio_csv, esquema_bronze)
```

Tabela BRONZE.DOCFAT criada com sucesso.
Tabela BRONZE.RAMOS criada com sucesso.
Tabela BRONZE.SINISTROS criada com sucesso.
Tabela BRONZE.SINISTROS_PGTO criada com sucesso.
Tabela BRONZE.SUB_RAMOS criada com sucesso.

3.2 - Atualizar PRATA

3.2.1 - Atualizar PRATA - Área das funções

30

```
def ler_tabela(tabela,df_bronze):

    # Inicializar a sessão Spark
    spark = SparkSession.builder.appName("LerTabela").getOrCreate()

    # Ler a tabela bronze.FATO_DOCFAT ou bronze.FATO_SINISTROS (Depende do
    # parâmetro)
    df_bronze = spark.table("bronze."+tabela)

    # Retornar o DataFrame
    return df_bronze
```

31

```
# Ler as demais tabelas
def ler_e_gravar_tabelas_bronze_para_prata():
    # Inicializar a sessão Spark
    spark =
SparkSession.builder.appName("TransferirBronzeParaPrata").getOrCreate()

    # Selecionar as tabelas que serão transferidas
    tabelas_dim = ['RAMOS', 'SUB_RAMOS']

    # Ler cada tabela e gravar na respectiva tabela prata.DIM_
    for tabela in tabelas_dim:
        tabela_bronze = f"bronze.{tabela}"
        tabela_prata = f"prata.DIM_{tabela}"

        # Ler a tabela bronze
        df_bronze = spark.table(tabela_bronze)

        # Gravar o DataFrame na tabela prata
        df_bronze.write.mode("overwrite").saveAsTable(tabela_prata)

        print(f" Tabela {tabela_bronze} gravada como {tabela_prata}")
```

32

```
def ajustar_am_comp(df, df_name):
    from pyspark.sql.functions import col, when, year, month, concat, lpad

    if df_name == "df_docfat_bronze":
        print('Ajustando informação AM_COMP da DOCFAT')
        df_ajustado = df.withColumn(
            "AM_COMP",
            when((col("AM_COMP").isNull() | (col("AM_COMP") == "0")),
            concat(year(col("DT_VIG_INI")), lpad(month(col("DT_VIG_INI")), 2, '0'))).otherwise(col("AM_COMP"))
        )
    else:
        print('Ajustando informação AM_COMP da SINISTROS')
        df_ajustado = df.withColumn(
            "AM_COMP",
            concat(year(col("DT_AVISO")), lpad(month(col("DT_AVISO")), 2, '0'))
        )

    return df_ajustado
```

33

```
# Verifico se o campo VLR_PRE_TOT é númerico
def verificar_vlr_pre_tot_numerico(df):
    from pyspark.sql.functions import col, when
    from pyspark.sql.functions import col, when, year, month, concat, lpad

    # Tentar converter a coluna VLR_PRE_TOT para tipo numérico
    df_verificado = df.withColumn("VLR_PRE_TOT_NUM",
        col("VLR_PRE_TOT").cast("double"))

    # Verificar se a conversão foi bem-sucedida
    df_verificado = df_verificado.withColumn("IS_NUMERIC",
        when(col("VLR_PRE_TOT_NUM").isNotNull(), True).otherwise(False))

    # Contar o número de registros não numéricos
    num_non_numeric = df_verificado.filter(col("IS_NUMERIC") ==
        False).count()

    # Retornar True se todos os valores forem numéricos, caso contrário,
    False
    return num_non_numeric == 0
```

```
# "Seta" valor 0 para os casos NULL
def setar_vlr_pre_tot_null_para_zero(df, df_name):
    from pyspark.sql.functions import col, when

    if df_name == 'df_docfat_bronze':
        # Substituir valores NULL ou 'NULL' na coluna VLR_PRE_TOT por 0.00
        df_ajustado = df.withColumn(
            "VLR_PRE_TOT",
            when((col("VLR_PRE_TOT").isNull()) | (col("VLR_PRE_TOT") == 'NULL')), 0.00).otherwise(col("VLR_PRE_TOT"))
    )
    elif df_name == 'df_sinistros_bronze':
        # Substituir valores NULL ou 'NULL' na coluna VLR_SIN por 0.00
        df_ajustado = df.withColumn(
            "VLR_SIN",
            when((col("VLR_SIN").isNull()) | (col("VLR_SIN") == 'NULL')), 0.00).otherwise(col("VLR_SIN"))
    )
    else:
        # Se o nome do DataFrame não for reconhecido, retornar o DataFrame original
        print('Valores NULL não encontrados')
        df_ajustado = df

    # Retornar o DataFrame ajustado
    return df_ajustado
```

```
def analise_fato_docfat():
    print('Início verificação DOCFAT')

    spark =
SparkSession.builder.appName("InicializarDataFrame").getOrCreate()

    # Ler a tabela
    df_docfat_bronze = spark.table("bronze.DOCFAT")

    # Ajustar a informação AM_COMP do frame df_docfat_bronze

    df_docfat_bronze = ajustar_am_comp(df_docfat_bronze,"df_docfat_bronze")

    numerico = verificar_vlr_pre_tot_numerico(df_docfat_bronze)

    if numerico:
        print('Campo VLR_PRE_TOT está OK')
    else:
        print('Campo VLR_PRE_TOT sendo ajustado...')
        # Ajustar o campo VLR_PRE_TOT = 0 para todos os casos NULL
        df_docfat_bronze =
setar_vlr_pre_tot_null_para_zero(df_docfat_bronze,'df_docfat_bronze')
        print('Fim do Ajuste do VLR_PRE_TOT')

    # Listar para verificação prévia

    print('Fim verificação DOCFAT')

    print('Listagem para verificação e gravação da DOCFAT no schema PRATA')

    print('Iniciando gravação bronze.DOCFAT para prata.FATO_DOCFAT')
    # Gravar o DataFrame na tabela prata.FATO_DOCFAT

    df_docfat_bronze.write.mode("overwrite").saveAsTable("prata.FATO_DOCFAT")
    print('Fim gravação bronze.DOCFAT para prata.FATO_DOCFAT')

    # Verifica PROPOSTAS e FATURAS
    df_prata = spark.sql("SELECT COD_TIP_DE, AM_COMP, VLR_PRE_TOT,
DT_VIG_INI FROM prata.FATO_DOCFAT WHERE COD_TIP_DE IN ('PP', 'FT')")

    # Mostrar as primeiras linhas do DataFrame para verificar os dados
    print('Lista apólices e faturas')
    df_prata.show()

    # Verifica ENDOSSOS
    df_prata = spark.sql("SELECT COD_TIP_DE, AM_COMP, VLR_PRE_TOT,
```

```
DT_VIG_INI FROM prata.FATO_DOCFAT WHERE COD_TIP_DE LIKE 'E%")\n\n# Mostrar as primeiras linhas do DataFrame para verificar os dados\nprint('Lista Endossos')\ndf_prata.show()\n\n# Verifica APÓLICES ABERTAS e SUBGRUPOS\ndf_prata = spark.sql("SELECT COD_TIP_DE, AM_COMP, VLR_PRE_TOT,\nDT_VIG_INI FROM prata.FATO_DOCFAT WHERE COD_TIP_DE IN ('AB', 'SG')")\n\n# Mostrar as primeiras linhas do DataFrame para verificar os dados\nprint('Lista apólices abertas e sub-grupos')\ndf_prata.show()
```

```
def inclusao_vlr_sinistros(df_sinistros_bronze):

    from pyspark.sql import SparkSession
    from pyspark.sql.functions import sum as _sum

    # Inicializar a sessão Spark
    spark = SparkSession.builder.appName("IncluirVLR_SIN").getOrCreate()

    # Ler a tabela bronze.FATO_SINISTROS_PGTO
    df_sinistros_pgto = spark.table("bronze.SINISTROS_PGTO")

    # Agrupar por COD_SIN e somar VLR_SIN
    df_vlr_sin_sum =
    df_sinistros_pgto.groupBy("COD_SIN").agg(_sum("VLR_SIN").alias("VLR_SIN"))

    # Adicionar a coluna VLR_SIN ao DataFrame df_sinistros_bronze
    df_sinistros_bronze = df_sinistros_bronze.join(df_vlr_sin_sum,
df_sinistros_bronze["COD_SIN"] == df_vlr_sin_sum["COD_SIN"],
"left").drop(df_vlr_sin_sum["COD_SIN"])

    return df_sinistros_bronze
```

```
# Inclusão do produto (NUM_RM, NUM_SUB_RM) na FATO_SINISTROS
def inserir_produto(df_sinistros_bronze):
    spark = SparkSession.builder.appName("InserirProduto").getOrCreate()
    print('Incluindo Produto...')

    # Ler a tabela bronze.FATO_DOCFAT
    df_docfat = spark.table("bronze.DOCFAT")

    # Filtrar para garantir que NUM_SUB_RM não seja NULL
    df_docfat = df_docfat.filter(df_docfat["NUM_SUB_RM"].isNotNull())

    # Realizar a junção das tabelas usando o campo COD_DOC_EXTER
    df_sinistros_bronze = df_sinistros_bronze.join(df_docfat,
df_sinistros_bronze["COD_DOC_EXTER"] == df_docfat["COD_DOC"],
"left").select(df_sinistros_bronze["*"], df_docfat["NUM_SUB_RM"])

    print("Fim da Inclusão de Produto na tabela SINISTROS")

    # Retornar o DataFrame resultante
    return df_sinistros_bronze
```

```
def analise_fato_sinistros():

    print('Início verificação SINISTROS e SINISTROS_PGT0')

    spark =
    SparkSession.builder.appName("InicializarDataFrame").getOrCreate()

    df_sinistros_bronze = spark.table("bronze.SINISTROS")

    # Inclusão do Campo AM_COMP
    df_sinistros_bronze =
    ajustar_am_comp(df_sinistros_bronze,"df_sinistros_bronze")

    # Inclusão do Campo VLR_SIN
    df_sinistros_bronze = inclusao_vlr_sinistros(df_sinistros_bronze)

    df_sinistros_bronze =
    setar_vlr_pre_tot_null_para_zero(df_sinistros_bronze,'df_sinistros_bronze')

    #inclusão dos campos NUM_SUB_RM
    df_sinistros_bronze = inserir_produto(df_sinistros_bronze)

    # Gravar o DataFrame na tabela prata.FATO_DOCFAT

    df_sinistros_bronze.write.mode("overwrite").saveAsTable("prata.FATO_SINISTR
OS")

    df_aux = df_sinistros_bronze.select("COD_SIN","NUM_RM",
    "COD_DOC_EXTER","DT_AVISO","AM_COMP","VLR_SIN", "NUM_SUB_RM")
    df_aux.show()
```

3.2.2 - Atualizar PRATA - Programa Principal

40

```
# Ajustar as informações da tabela FATO_DOCFAT

analise_fato_docfat()
```

| | | | | |
|--|----|--------|-----|---------------------|
| | SG | 202404 | 0.0 | 2024-04-30 00:00:00 |
| | SG | 202404 | 0.0 | 2024-04-30 00:00:00 |
| | AB | 202404 | 0.0 | 2024-04-30 00:00:00 |
| | SG | 202403 | 0.0 | 2024-03-31 00:00:00 |
| | AB | 202404 | 0.0 | 2024-04-30 00:00:00 |
| | AB | 202404 | 0.0 | 2024-04-30 00:00:00 |
| | AB | 202404 | 0.0 | 2024-04-30 00:00:00 |
| | AB | 202404 | 0.0 | 2024-04-30 00:00:00 |
| | AB | 202404 | 0.0 | 2024-04-30 00:00:00 |
| | AB | 202405 | 0.0 | 2024-05-12 00:00:00 |
| | AB | 202404 | 0.0 | 2024-04-30 00:00:00 |
| | AB | 202405 | 0.0 | 2024-05-12 00:00:00 |
| | AB | 202405 | 0.0 | 2024-05-16 00:00:00 |
| | AB | 202403 | 0.0 | 2024-03-31 00:00:00 |

only showing top 20 rows

41

```
# Análise das informações de Sinistros e Pagamento de Sinsitros
analise_fato_sinistros()
```

| | | | | | |
|---------------------|------------------------|----------------------|--------|---|--|
| 150.0 | 0 | | | | |
| 0101.14.03.20240037 | 14 0101.14.00.00000402 | 2024-10-11 00:00:... | 202410 | | |
| 0.0 | null | | | | |
| 0101.14.03.20240038 | 14 0101.14.00.00000549 | 2024-10-14 00:00:... | 202410 | | |
| 587.6 | 0 | | | | |
| 0101.14.03.20240039 | 14 0101.14.00.00000503 | 2024-11-04 00:00:... | 202411 | | |
| 574.4 | 0 | | | | |
| 0101.14.03.20240040 | 14 0101.14.00.00000515 | 2024-11-04 00:00:... | 202411 | | |
| 593.6 | 0 | | | | |
| 0101.14.03.20240041 | 14 0101.14.00.00000632 | 2024-11-04 00:00:... | 202411 | | |
| 0.0 | 0 | | | | |
| 0101.29.03.20240133 | 29 0101.93.00.00000130 | 2024-01-02 00:00:... | 202401 | 2 | |
| 000.0 | null | | | | |
| 0101.29.03.20240134 | 29 0101.93.00.00000144 | 2024-02-02 00:00:... | 202402 | 3 | |
| 000.0 | null | | | | |
| 0101.29.03.20240135 | 29 0101.93.00.00000144 | 2024-02-07 00:00:... | 202402 | 3 | |
| 000.0 | null | | | | |

only showing top 20 rows

42

```
# Ler as demais Tabelas  
ler_e_gravar_tabelas_bronze_para_prata()
```

Tabela bronze.RAMOS gravada como prata.DIM_RAMOS
Tabela bronze.SUB_RAMOS gravada como prata.DIM_SUB_RAMOS

_3.3 - Atualizar OURO

3.3.1 - Atualizar OURO - Área de funções

45

```
# Gravar as tabelas

def insere_fato_ouro(tabela):
    # Inicializar a sessão Spark
    spark = SparkSession.builder.appName("TransferirDados").getOrCreate()

    # Habilitar a evolução do esquema
    spark.conf.set("spark.databricks.delta.schema.autoMerge.enabled",
    "true")

    if tabela == "FATO_DOCFAT":
        # Ler a tabela prata.FATO_DOCFAT
        df_prata = spark.table("prata.FATO_DOCFAT")

        # Crio a informação PRODUTO
        df_prata = df_prata.withColumn("ID_NUM_PROD", concat(col("NUM_RM"),
        lpad(col("NUM_SUB_RM").cast("string"), 2, "0")))

        # Selecionar os campos específicos
        df_selecionado =
        df_prata.select(col("COD_DOC").alias("ID_COD_DOC"), "ID_NUM_PROD",
                        col ("NUM_RM").alias("ID_NUM_RM"),

        col("COD_TIP_DE").alias("ID_COD_TIP_DE"), "STA_DOC", "VLR_PRE_TOT",
        "AM_COMP")

        # Escrever os dados na tabela ouro.FATO_DOCFAT

        df_selecionado.write.mode("overwrite").saveAsTable("ouro.FATO_DOCFAT")

        # Mensagem de sucesso
        print(f"Dados da tabela prata.FATO_DOCFAT foram inseridos na tabela
        ouro.FATO_DOCFAT com sucesso.")

    elif tabela == "FATO_SINISTROS":
        # Ler a tabela prata.FATO_SINISTROS
        df_prata = spark.table("prata.FATO_SINISTROS")

        # Crio a informação PRODUTO
        df_prata = df_prata.withColumn("ID_NUM_PROD", concat(col("NUM_RM"),
        lpad(col("NUM_SUB_RM").cast("string"), 2, "0")))

        # Selecionar os campos específicos
        df_selecionado =
        df_prata.select(col("COD_SIN").alias("ID_COD_SIN"), "ID_NUM_PROD",
                        col("NUM_RM").alias("ID_NUM_RM"), "COD_DOC_EXTER", "VLR_SIN",
                        "AM_COMP")
```

```
# Escrever os dados na tabela ouro.FATO_SINISTROS

df_selecionado.write.mode("overwrite").saveAsTable("ouro.FATO_SINISTROS")

# Mensagem de sucesso
print(f"Dados da tabela prata.FATO_SINISTROS foram inseridos na
tabela ouro.FATO_SINISTROS com sucesso.")

elif tabela == "DIM_SUB_RAMOS":
    # Ler a tabela prata.DIM_RAMOS
    df_prata = spark.table("prata.DIM_SUB_RAMOS")

    df_prata = df_prata.withColumn("ID_NUM_PROD", concat(col("NUM_RM"),
    lpad(col("NUM_SUB_RM").cast("string"), 2, "0")))

    # Selecionar os campos específicos
    df_selecionado =
        df_prata.select(col("NUM_RM").alias("ID_NUM_RM"), "ID_NUM_PROD",
                        col("DSC_SUB_RM").alias
        ("DSC_PRODUTO"))

# Escrever os dados na tabela ouro.FATO_SINISTROS

df_selecionado.write.mode("overwrite").saveAsTable("ouro.DIM_SUB_RAMOS")

# Mensagem de sucesso
print(f"Dados da tabela prata.DIM_SUB_RAMOS foram inseridos na
tabela ouro.DIM_SUB_RAMOS com sucesso.")

elif tabela == "DIM_RAMOS":
    # Ler a tabela prata.DIM_RAMOS
    df_prata = spark.table("prata.DIM_RAMOS")

    # Selecionar os campos específicos
    df_selecionado = df_prata.select(col("NUM_RM").alias("ID_NUM_RM"),
    "NOM_RM", "ABV_RM")

    # Escrever os dados na tabela ouro.FATO_SINISTROS

df_selecionado.write.mode("overwrite").saveAsTable("ouro.DIM_RAMOS")

# Mensagem de sucesso
print(f"Dados da tabela prata.DIM_RAMOS foram inseridos na tabela
ouro.DIM_RAMOS com sucesso.")
```

46

```
def criar_dim_cod_tip_de():
    # Inicializar a sessão Spark
    spark =
SparkSession.builder.appName("CriarDIM_COD_TIP_DE").getOrCreate()

    # Ler a tabela prata.FATO_DOCFAT
    df_fato_docfat = spark.table("prata.FATO_DOCFAT")

    # Selecionar os campos específicos e criar a tabela de dimensão
    df_dim_cod_tip_de = df_fato_docfat.select("COD_TIP_DE").distinct() \
        .withColumnRenamed("COD_TIP_DE",
                            "ID_COD_TIP_DE") \
        .withColumn("DSC_COD_TIP_DE",
                    when(col("ID_COD_TIP_DE") == "AB", "APÓLICE
ABERTA").when(col("ID_COD_TIP_DE") == "FT",
"FATURA").when(col("ID_COD_TIP_DE") == "PP",
"PROPOSTA").when(col("ID_COD_TIP_DE") == "E1", "ENDOSO
COBRANÇA").when(col("ID_COD_TIP_DE") == "E5", "ENDOSO
CANCELAMENTO").when(col("ID_COD_TIP_DE") == "E4", "ENDOSO SEM
MOVIMENTO").when(col("ID_COD_TIP_DE") == "E8", "ENDOSO
ESTORNO").when(col("ID_COD_TIP_DE") == "SG", "SUB-
GRUPO").otherwise("OUTRO"))

    # Criar a nova tabela com o esquema especificado
    df_dim_cod_tip_de.write.mode("overwrite").saveAsTable("ouro.DIM_COD_TIP_DE"
)
```

3.3.1 - Atualizar OURO - Programa Principal

48

```
# Cria a tabela Dimensão DIM_COD_TIP_DE
criar_dim_cod_tip_de()
```

49

```
# Carrega OURO.FATO_DOCFAT  
insere_fato_ouro('FATO_DOCFAT')
```

Dados da tabela prata.FATO_DOCFAT foram inseridos na tabela ouro.FATO_DOCFAT com sucesso.

50

```
# Carrega OURO.FATO_SINISTROS  
insere_fato_ouro('FATO_SINISTROS')
```

Dados da tabela prata.FATO_SINISTROS foram inseridos na tabela ouro.FATO_SINISTROS com sucesso.

51

```
# Carrega OURO.DIM_RAMOS  
insere_fato_ouro('DIM_RAMOS')
```

Dados da tabela prata.DIM_RAMOS foram inseridos na tabela ouro.DIM_RAMOS com sucesso.

52

```
# Carrega OURO.DIM_SUB_RAMOS  
insere_fato_ouro('DIM_SUB_RAMOS')
```

Dados da tabela prata.DIM_SUB_RAMOS foram inseridos na tabela ouro.DIM_SUB_RAMOS com sucesso.

=====

4 - Área dos Gráficos

4.1 - Área dos Gráficos - Funções

56

```
# Identifico os 3 produtos mais vendidos ( VLR_PRE_TOT) da empresa

def top_3_produtos_mais_vendidos():
    spark =
    SparkSession.builder.appName("TopProdutosVendidos").getOrCreate()

    # Ler a tabela ouro.FATO_DOCFAT
    df_docfat = spark.table("ouro.FATO_DOCFAT")

    # Filtrar os códigos de tipo 'PP', 'FT', 'E1' e somar o campo
    VLR_PRE_TOT
    df_top_produtos = df_docfat.filter(col("ID_COD_TIP_DE").isin('PP',
    'FT', 'E1')) \
        .groupBy("ID_NUM_PROD") \

    .agg(_sum("VLR_PRE_TOT").alias("TOTAL_VENDAS")) \
        .orderBy(col("TOTAL_VENDAS").desc()) \
        .limit(3)

    # Ler a tabela ouro.SUB_RAMOS
    df_sub_ramos = spark.table("ouro.DIM_SUB_RAMOS")

    # Adicionar a descrição do produto
    df_top_produtos_com_desc = df_top_produtos.join(df_sub_ramos,
    (df_top_produtos["ID_NUM_PROD"] == df_sub_ramos["ID_NUM_PROD"]),
        "left") \
        .select(df_top_produtos["*"],
    df_sub_ramos["DSC_PRODUTO"])

    return df_top_produtos_com_desc
```

```
# Busco Todos os Sinistros Avisados durante o período. Se eu quiser ver todos,  
# a variavel top_3_produtos =[]  
  
def buscar_sinistros(produtos):  
    spark = SparkSession.builder.appName("BuscarSinistros").getOrCreate()  
  
    # Converter top_3_produtos para Pandas DataFrame  
    produtos_pd = produtos.toPandas()  
  
    # Criar a string para a cláusula IN  
    in_clause = ', '.join([f"({int(row['ID_NUM_PROD'])})" for _, row in  
    produtos_pd.iterrows()])  
  
    # Executar a consulta SQL para buscar sinistros e nome dos produtos  
    df_sinistros = spark.sql(f"""  
        SELECT SIN.AM_COMP, SUM(SIN.VLR_SIN) AS VLR_SIN_TOTAL  
        FROM ouro.FATO_SINISTROS SIN  
        WHERE (CAST(SIN.ID_NUM_PROD AS INT)) IN ({in_clause})  
        GROUP BY SIN.AM_COMP  
        ORDER BY SIN.AM_COMP  
    """)  
  
    return df_sinistros
```

```
# Busco Todos as Emissões duramnte o período. Se eu quiser ver todos,  
# a variavel top_3_produtos será carregada com todos osm produtos  
  
def buscar_emissoes(produtos):  
  
    spark = SparkSession.builder.appName("BuscarEmissoes").getOrCreate()  
  
    # Converter top_3_produtos para Pandas DataFrame  
    produtos_pd = produtos.toPandas()  
  
    # Criar a string para a cláusula IN  
    in_clause = ', '.join([f"({int(row['ID_NUM_PROD'])})" for _, row in  
    produtos_pd.iterrows()])  
  
  
    # Ler a tabela ouro.FATO_DOCFAT  
    df_docfat = spark.sql(f"""  
        SELECT AM_COMP, SUM(VLR_PRE_TOT) AS VLR_PRE_TOT_TOTAL  
        FROM ouro.FATO_DOCFAT  
        WHERE AM_COMP >= 202401 AND AM_COMP <= 202412 AND ID_COD_TIP_DE IN  
        ('PP', 'FT', 'E1')  
        AND (CAST(ID_NUM_PROD AS INT)) IN ({in_clause})  
        GROUP BY AM_COMP  
        ORDER BY AM_COMP  
    """)  
    return df_docfat
```

```
def gerar_grafico_barras(df,cor,df_name):
    # Converter para Pandas DataFrame
    df_pandas = df.toPandas()

    # Gerar o gráfico de barras
    plt.figure(figsize=(10, 6))
    if df_name == "df_docfat":
        bars = plt.bar(df_pandas["AM_COMP"], df_pandas["VLR_PRE_TOT_TOTAL"] / 1e6, color=cor) # Dividir por 1e6 para converter para milhões
        plt.ylabel("Total Prêmios Emitidos (em milhões)")
        plt.title("Total de Prêmios Emitidos por Ano/Mês ( Jan/2024 até Dez/2024)")
    else:
        bars = plt.bar(df_pandas["AM_COMP"], df_pandas["VLR_SIN_TOTAL"] / 1e6, color=cor)
        plt.ylabel("Total Sinistros Avisados (em milhões)")
        plt.title("Total de Sinistros Avisados por Ano/Mês ( Jan/2024 até Dez/2024)")

    plt.xlabel("Ano/Mês Competência")

    plt.xticks(rotation=45)

    # Adicionar os valores dentro de cada barra na posição horizontal
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width() / 2, height / 2,
f'{height:.2f}M', ha='center', va='center', color='black')

    plt.tight_layout()
    plt.show()
```

4.2 - Programa Principal

4.2.1 - Prêmios Emitidos - GERAL

```
# Recupero todos os prêmios e gero gráfico

# Inicializar a sessão Spark
spark = SparkSession.builder.appName("ObterProdutos").getOrCreate()

# Ler a tabela ouro.DIM_SUB_RM
df_dim_sub_rm = spark.table("ouro.DIM_SUB_RAMOS")

# Selecionar as colunas NUM_RM e NUM_SUB_RM
produtos = df_dim_sub_rm.select("ID_NUM_PROD")

df_docfat = buscar_emissoes(produtos)

cor ='lightgreen'
gerar_grafico_barras(df_docfat,cor,"df_docfat")
```

- ▶ df_dim_sub_rm: pyspark.sql.dataframe.DataFrame = [ID_NUM_RM: integer, ID_NUM_PROD: string ... 1 more field]
- ▶ df_docfat: pyspark.sql.dataframe.DataFrame = [AM_COMP: string, VLR_PRE_TOT_TOTAL: double]
- ▶ produtos: pyspark.sql.dataframe.DataFrame = [ID_NUM_PROD: string]

4.2.2 - Sinistros Avisados - GERAL

64

```
df_sinistros = buscar_sinistros(produtos)

cor ='lightblue'
gerar_grafico_barras(df_sinistros,cor,"df_sinistros")
```

- ▶ df_sinistros: pyspark.sql.dataframe.DataFrame = [AM_COMP: string, VLR_SIN_TOTAL: double]

4.2.3 - Relação Prêmios Emitidos X Sinistros Pagos - TOP 3 Produtos

66

```
# Relação PREMIOS EMITIDOS X SINISTROS AVISADOS dos 3 produtos mais vendidos

# Chamar a função para obter os 3 produtos mais vendidos
top_3_produtos = top_3_produtos_mais_vendidos()

# Converter top_3_produtos para Pandas DataFrame
top_3_produtos_pd = top_3_produtos.toPandas()

# Imprime os 3 produtos mais vendidos
print(tabulate(top_3_produtos_pd, headers="keys", tablefmt="fancy_grid"))

# Criar a string para a cláusula IN
in_clause = ', '.join([f"{{int(row['ID_NUM_PROD'])}})" for _, row in top_3_produtos_pd.iterrows()])

# Ler a tabela ouro.FATO_DOCFAT
df_docfat = buscar_emissoes(top_3_produtos)

# Ler a tabela ouro.FATO_SINISTROS
df_sinistros = buscar_sinistros(top_3_produtos)

# Converter para Pandas DataFrame
df_docfat_pandas = df_docfat.toPandas()
df_sinistros_pandas = df_sinistros.toPandas()

# Mesclar os DataFrames
df_combined = pd.merge(df_docfat_pandas, df_sinistros_pandas, on="AM_COMP",
how="outer").fillna(0)

# Configurar a largura das barras
bar_width = 0.35
index = np.arange(len(df_combined["AM_COMP"]))

# Gerar o gráfico de barras agrupadas
plt.figure(figsize=(12, 8))
bars1 = plt.bar(index, df_combined["VLR_PRE_TOT_TOTAL"] / 1e6, bar_width,
label='Prêmio Emitido Total', color='lightgreen') # Dividir por 1e6 para converter para milhões
bars2 = plt.bar(index + bar_width, df_combined["VLR_SIN_TOTAL"] / 1e6,
bar_width, label='Sinistros Avisados', color='lightblue') # Dividir por 1e6 para converter para milhões

plt.xlabel("Ano Mês")
plt.ylabel("Total (em milhões)")
```

```

plt.title("Total de Prêmio Emitido e Sinistro Avisado por Ano Mês  

Competência - TOP 3")
plt.xticks(index + bar_width / 2, df_combined["AM_COMP"], rotation=45)
plt.legend()

# Adicionar os valores dentro de cada barra na posição horizontal
for bar1, bar2 in zip(bars1, bars2):
    height1 = bar1.get_height()
    height2 = bar2.get_height()
    plt.text(bar1.get_x() + bar1.get_width() / 2, height1 / 2,
f'{height1:.2f}M', ha='center', va='center', color='black')
    plt.text(bar2.get_x() + bar2.get_width() / 2, height2 / 2,
f'{height2:.2f}M', ha='center', va='center', color='black')

plt.tight_layout()
plt.show()

```

- ▶ df_docfat: pyspark.sql.dataframe.DataFrame = [AM_COMP: string, VLR_PRE_TOT_TOTAL: double]
- ▶ df_sinistros: pyspark.sql.dataframe.DataFrame = [AM_COMP: string, VLR_SIN_TOTAL: double]
- ▶ top_3_produtos: pyspark.sql.dataframe.DataFrame = [ID_NUM_PROD: string, TOTAL_VENDAS: double ... 1 more field]

| | ID_NUM_PROD | TOTAL_VENDAS | DSC_PRODUTO |
|---|-------------|--------------|--------------------------------|
| 0 | 5300 | 4.5744e+07 | RESPONSABILIDADE CIVIL |
| 1 | 5301 | 1.59127e+07 | RCF-V - GARANTIA ◊NICA (DM/DC) |
| 2 | 9301 | 1.32632e+07 | VG |

=====:

