Portfólio de Aprendizado

SDC Academy

Autor: José Gilberto Araújo da Silva

2023

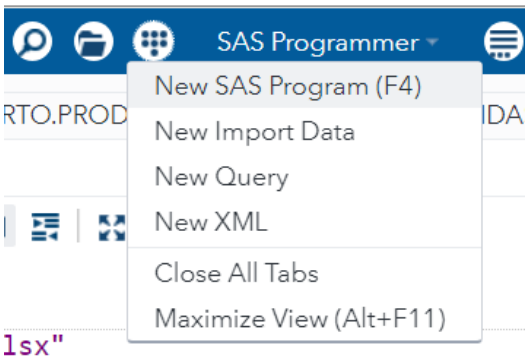
Sumário

Atividade 01
Acesso aos Dados

Antes de qualquer coisa, é necessário acessar os dados que serão utilizados para a pesquisa. Essa atividade contém 2 partes principais:
1 – Importar os arquivos para o SAS, usando o comando proc import
2 – Importar os arquivos para o Python, usando pandas e pyodbc

Importando os Arquivos para o SAS

Primeiramente, após acessar a máquina virtual, salva como favorito no navegador Chrome na barra de tarefas, criamos um novo programa .sas usando F4 ou usando a opção no menu:



Para importarmos os dados, como falado anteriormente, usamos o comando proc import:

```
*Importação da aba Vendas do arquivo DadosACADEMY.xlsx;

proc import datafile="/sasdata/sdcacademy/gilberto/DadosACADEMY.xlsx"
                    out=gilberto.VENDAS dbms=XLSX replace;
          sheet="Vendas";
RUN;
```

Acima mostra um exemplo de como importar um arquivo excel para o SAS usando esse comando.

1. A opção DATAFILE= sinaliza onde podemos encontrar o arquivo que será importado, é importante lembrar que utilizamos todo o caminho, incluindo a extensão do arquivo, xlsx nesse caso.

2. A opção OUT= especifica o nome da tabela de saída, na imagem fica dentro da biblioteca com o nome do usuário (gilberto) e então cria uma tabela com o nome VENDAS.

3. A opção DBMS= indica o tipo de dado que será importado, usamos XLSX para arquivos excel, geralmente.

4. Já o REPLACE indica que queremos que caso o dataset já exista, nesse caso o dataset VENDAS, seja substituído quando executarmos o programa.

5. A opção SHEET, que não fica dentro da declaração PROC IMPORT, indica qual aba queremos importar. Passamos uma string chamada “Vendas”, para importar a aba de mesmo nome.

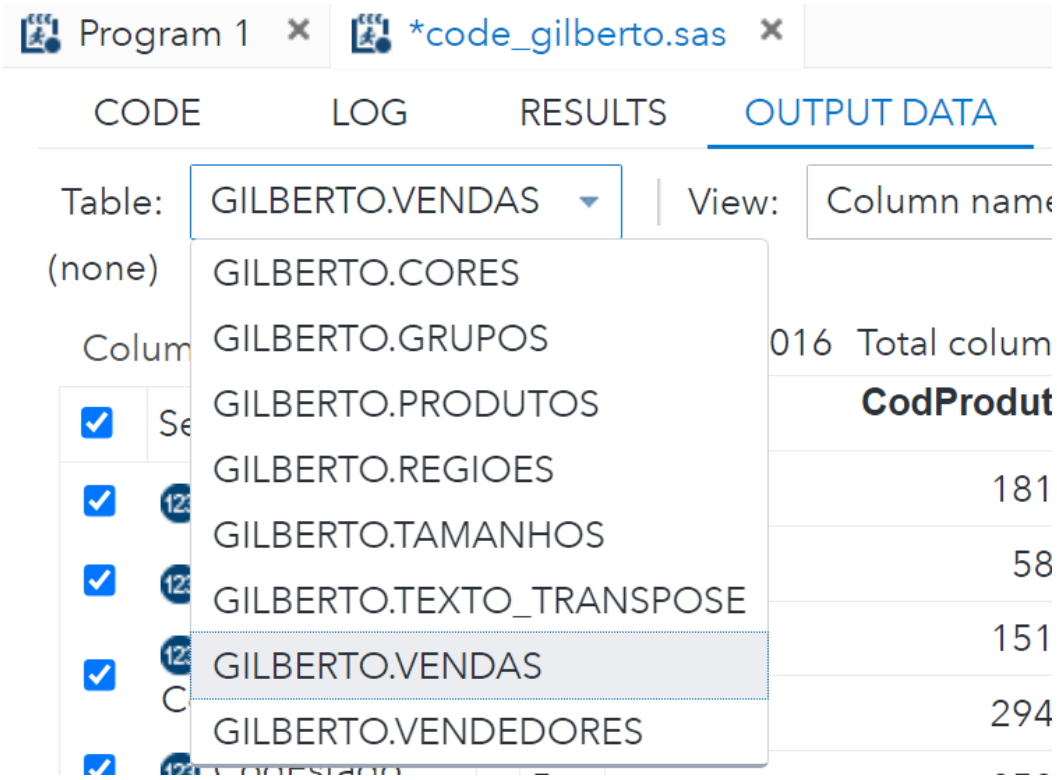
Faremos isso para cada uma das abas do arquivo, executando o mesmo comando e alterando a opção **SHEET**=Nome_da_Aba. Os nomes das abas no arquivo são: Vendas, Vendedor, Produtos, Grupos, Tamanhos e Regioes.

Para arquivos de texto plano, ou seja, .txt, utilizamos quase todas as mesmas opções, porém é necessário trocar o **DBMS=** para TAB e usar a opção **DELIMITER=** para indicar o separador das colunas, no primeiro caso a string “|” e no segundo a string “;”.

```
*Importação dos dados do arquivo Cores.txt;

proc import datafile="/sasdata/sdcacademy/gilberto/Cores.txt"

                out=gilberto.cores dbms=tab replace;

        delimiter="|";

run;
```

```
*Importação do arquivo Texto_transpose.txt;
proc import datafile="/sasdata/sdcacademy/gilberto/Texto_transpose.txt"
                out=gilberto.texto_transpose dbms=tab replace;
        delimiter=";";
run;
```

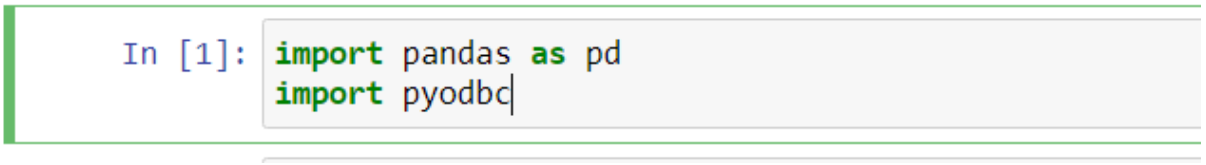
Ao final das importações, teremos na biblioteca gilberto essas tabelas:



Acessando os dados em Python

Já para importar os arquivos da pasta localizada em C:\Users\gilberto\Desktop\SDC AcademY\Dados, primeiro é necessário importar as bibliotecas que serão usadas para acessar esses arquivos:

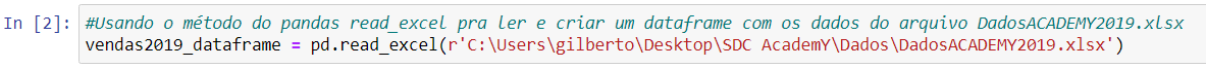
```
import pandas as pd
import pyodbc
```



Acessando Arquivos do Excel usando Pandas

No segundo bloco utilizamos o método read_excel da biblioteca pandas do Python para ler o arquivo DadosACADEMY2019.xlsx e criar um dataframe com os dados. A letra “r” na frente da string sinaliza para que os caracteres “\” não sejam interpretados pelo Python, visto que esse caractere que é usado no caminho dos arquivos e pastas no Windows, também é utilizado em Python para algumas funcionalidades.

vendas2019_dataframe = pd.read_excel(r'C:\Users\gilberto\Desktop\SDC AcademY\Dados\DadosACADEMY2019.xlsx')



Após isso, usamos print na variável vendas2019, para que possamos ver os resultados (Não é obrigatório, porém é uma forma de verificar se o método anterior funcionou de maneira adequada de maneira que seja possível ver as linhas e colunas):

```
print(vendas2019_dataframe)
```

```
     CodProduto  CodCor  CodTamanho  CodEstado  DataVenda  Vendedor  \
0          2909       7           1         22 2019-05-02         4
1          1223       2           2         18 2019-02-15         2
2           166       6           3         11 2019-02-24         4
3          1781       4           2         11 2019-02-16         2
4          2671       3           2         24 2019-08-24         2
..          ...     ...         ...        ...        ...       ...
494        1060       7           2         17 2019-07-01         3
495        1985       9           1         18 2019-03-04         4
496        2165       8           3          9 2019-05-31         5
497         275       1           4         13 2019-03-30         1
498           7       1           3         20 2019-08-08         2

     QtdeVendida
0             33
1            166
2            189
3            182
4            116
..           ...
494           39
495          106
496           88
497          126
498          191

[499 rows x 7 columns]
```

Acessando arquivos do Microsoft Access (.accdb)

Em seguida, iremos importar os arquivos do banco de dados do Microsoft Acess, com a extensão .accdb. Para isso usaremos a biblioteca Python pyodbc em conjunto com o Jupyter Notebook.

```
conexao= pyodbc.connect(r'"Driver={Microsoft Access Driver (*.mdb, *.accdb)};

                DBQ=C:\Users\gilberto\Desktop\SDC AcademY\Dados\Estados.accdb;"')

cursor = conexao.cursor()

comando_sql = f'SELECT * FROM Estados'

estados_dataframe = pd.read_sql_query(comando_sql, conexao)

conexao.close()
```

```
In [3]: #Conectando ao arquivo Estados.accdb usando o método connect do pyodbc
conexao = pyodbc.connect(r'''Driver={Microsoft Access Driver (*.mdb, *.accdb)};
                         DBQ=C:\Users\gilberto\Desktop\SDC AcademY\Dados\Estados.accdb;''')

#Criando um objeto cursor para executar comandos SQL
cursor = conexao.cursor()

#Comando SQL para exibir todas as linhas e colunas
comando_sql = f'SELECT * FROM Estados'

# Executando a consulta SQL para carregar os resultados em um DataFrame
estados_dataframe = pd.read_sql_query(comando_sql, conexao)

# Fechando a conexão estabelecida na primeira linha, já que não é mais necessária
conexao.close()
```

```
C:\Anaconda\lib\site-packages\pandas\io\sql.py:761: UserWarning: pandas only support SQLAlchemy connectable(engine/connection)
ordatabase string URI or sqlite3 DBAPI2 connectionother DBAPI2 objects are not tested, please consider using SQLAlchemy
  warnings.warn(
```

1. Na primeira linha, usaremos o método connect que criará uma conexão com o banco de dados. Como argumento, primeiro inserimos o nome do driver que será utilizado, e em seguida (após o “;”) DBQ identifica o caminho onde está localizado o arquivo.

2. Na segunda linha, usamos o método cursor para criar um objeto que será usado para executar os comandos SQL.

3. Na terceira linha definimos uma query em SQL que será usada na próxima linha como argumento do método read_sql_query para selecionar todas as linhas e todas as colunas do arquivo Estados.accdb.

4. Na quarta linha criamos um dataframe usando o método read_sql_query da biblioteca pandas, tornando assim mais fácil a manipulação e exploração de dados. O primeiro argumento é nosso comando SQL definido na terceira linha e o segundo é a conexão que fizemos com o banco de dados na primeira.

5. Por último, já que o dataframe já foi criado, fechamos a conexão com o banco de dados por questões de boas práticas.

Podemos imprimir o resultado na tela para verificar se tudo ocorreu como o planejado:

```
print(estados_dataframe)
```

```
In [4]: print(estados_dataframe)
```

```
    CodEstado                 Nome Sigla         Capital  PercImposto
0         1.0                 Acre    AC      Rio Branco         0.12
1         2.0              Alagoas    AL          Maceió         0.09
2         3.0                Amapá    AP          Macapá         0.13
3         4.0             Amazonas    AM          Manaus         0.08
4         5.0                Bahia    BA        Salvador         0.15
5         6.0                Ceará    CE       Fortaleza         0.13
6         7.0     Distrito Federal    DF        Brasília         0.12
7         8.0       Espírito Santo    ES         Vitória         0.12
8         9.0                Goiás    GO         Goiânia         0.11
9        10.0             Maranhão    MA        São Luís         0.09
10       11.0         Mato Grosso    MT          Cuiabá         0.17
11       12.0   Mato Grosso do Sul    MS    Campo Grande         0.18
12       13.0        Minas Gerais    MG  Belo Horizonte         0.12
13       14.0                Pará    PA           Belém         0.10
14       15.0             Paraíba    PB     João Pessoa         0.09
15       16.0              Paraná    PR        Curitiba         0.09
16       17.0           Pernambuco    PE          Recife         0.10
17       18.0               Piauí    PI        Teresina         0.16
18       19.0      Rio de Janeiro    RJ  Rio de Janeiro         0.14
19       20.0  Rio Grande do Norte    RN           Natal         0.11
20       21.0    Rio Grande do Sul    RS    Porto Alegre         0.17
21       22.0             Rondônia    RO     Porto Velho         0.13
22       23.0              Roraima    RR       Boa Vista         0.11
23       24.0       Santa Catarina    SC   Florianópolis         0.13
24       25.0            São Paulo    SP       São Paulo         0.13
25       26.0              Sergipe    SE         Aracaju         0.18
26       27.0             Tocantins    TO          Palmas         0.15
```

Acessando os dados de um arquivo CSV (Comma Separated Values)

Para acessar os dados do arquivo Departamentos.csv localizado em C:\Users\gilberto\Desktop\SDC AcademY\Dados\Departamentos.csv e criar um dataframe utilizamos o método read_csv da biblioteca pandas:

departamentos_dataframe = pd.read_csv(r'C:\Users\gilberto\Desktop\SDC AcademY\Dados\Departamentos.csv', sep=";")

```
In [4]: #Criando um dataframe a partir do arquivo Departamentos.csv
departamentos_dataframe = pd.read_csv(r'C:\Users\gilberto\Desktop\SDC AcademY\Dados\Departamentos.csv', sep=";")
```

O primeiro argumento do método é o caminho do arquivo, já o segundo especifica o separador das colunas, o que é necessário caso seja diferente do padrão, que seria a vírgula (,).

Podemos usar print para verificar se tudo ocorreu de maneira adequada:

print(departamentos_dataframe)

```
print(departamentos_dataframe)
```

```
   CodDepto              Descricao
0         1     Artigos Esportivos
1         2              Casamentos
2         3                  Roupas
3         4                Infantil
4         5             Ferramentas
5         6  Utensílios Domésticos
6         7               Alimentos
7         8             Automotivos
8         9              Jardinagem
9        10                 Náutica
```

Atividade 02

Exploração dos Dados

Essa atividade contém 3 partes principais:

1 – Explorar os dados das tabelas criadas usando SAS usando os comandos proc contents, proc means e proc freq

2 – Explorar os dados dos dataframes criados usando python usando os métodos info, describe e crosstab

3 – Criar um modelo de dados usando a ferramenta brModelo

Explorando dados de uma tabela usando SAS

Utilizamos principalmente os comandos proc contents, proc means e proc freq para examinar os dados da tabela.

1 - proc contents

```
proc contents data=gilberto.vendas;
run;
```

Após executar esse código, veremos diversas informações sobre o conteúdo dessa tabela:

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	12
First Data Page	1
Max Obs per Page	1360
Obs in First Data Page	1303
Number of Data Set Repairs	0
Filename	/sasdata/sdcacademy/gilberto/vendas.sas7bdat
Release Created	9.0401M6
Host Created	Linux
Inode Number	67553357
Access Permission	rw-rw-r--
Owner Name	gilberto
File Size	832KB
File Size (bytes)	851968

Em um primeiro momento, é ideal observar a última tabela exibida:

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
2	CodCor	Num	8	BEST.		CodCor
4	CodEstado	Num	8	BEST.		CodEstado
1	CodProduto	Num	8	BEST.		CodProduto
3	CodTamanho	Num	8	BEST.		CodTamanho
5	DataVenda	Num	8	MMDDYY10.		DataVenda
7	QtdeVendida	Char	5	$5.	$5.	QtdeVendida
6	Vendedor	Char	1	$1.	$1.	Vendedor

Nela podemos observar o nome de cada coluna, o tipo, o comprimento, o formato, o informat e o rótulo, casa haja algum. Podemos usar esses dados para observar e depois alterar o tipo de dados de uma coluna, por exemplo de char(string) para num(Número). Usaremos esses nomes das colunas (Variable) como base para o nosso modelo de dados futuramente. Faremos isso para cada uma das tabelas.

2 - proc means

```
proc means data=gilberto.vendas;
run;
```

Após executar esse código, iremos receber uma tabela com informações descritivas com base nos dados da tabela, de forma estatística. É importante lembrar que serão levadas em conta somente colunas em que o tipo seja num (Número).

The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
CodProduto	CodProduto	15013	1502.40	862.3097190	1.0000000	3000.00
CodCor	CodCor	15013	5.5151535	2.8897243	1.0000000	10.0000000
CodTamanho	CodTamanho	15012	2.5073275	1.1207847	1.0000000	4.0000000
CodEstado	CodEstado	15013	14.0961833	7.7960322	1.0000000	27.0000000
DataVenda	DataVenda	15012	22396.10	278.9962123	21915.00	22875.00

Variable mostra o nome da coluna.
Label mostra o rótulo de cada coluna.
N mostra uma contagem do total de linhas utilizado para fazer os cálculos.
Mean (Média) mostra a média aritmética.
Std Dev mostra o desvio padrão, quanto maior, maior será a dispersão dos dados.

Minimum mostra o menor valor encontrado.
Maximum mostra o maior valor encontrado.

Executamos esse mesmo código para todas as tabelas, a fim de observar alguma coisa fora do normal ou dados que possam estar incompletos ou incorretos.
Também é possível adicionar várias opções para personalizar os resultados.

3 - proc freq

```
proc freq data=gilberto.vendas;
run;
```

Vendedor				
Vendedor	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	3022	20.13	3022	20.13
2	2949	19.64	5971	39.77
3	2943	19.60	8914	59.38
4	3071	20.46	11985	79.83
5	3024	20.14	15009	99.97
A	1	0.01	15010	99.98
B	1	0.01	15011	99.99
C	1	0.01	15012	99.99
X	1	0.01	15013	100.00
Frequency Missing = 3				

Após executar o código acima, nos resultados podemos ver uma tabela igual a mostrada acima para cada uma das variáveis.

A primeira coluna mostra cada um dos valores encontrados na tabela.
Frequency mostra quantas vezes cada valor aparece na tabela.
Percent mostra a porcentagem que cada valor aparece, em relação ao total.
Cumulative Frequency mostra a contagem de observações até determinada categoria.
Cumulative Percent mostra a porcentagem cumulativa em relação ao total até determinada categoria.

Por último, também podemos ver quantas observações (linhas) possuem valores ausentes, no caso da tabela acima são 3.
Executaremos esse mesmo código para cada uma das tabelas a fim de,

primeiramente, observar valores ausentes e valores que aparecem poucas vezes na tabela. Dessa forma podemos “limpar” os dados recebidos e organizá-los de modo a tornar nossa análise mais precisa.

Explorando os dados dos dataframes criados usando pandas

Utilizamos os métodos info, describe e crosstab para saber mais informações sobre os dataframes criados a partir dos dados.

1 - Método info

vendas2019_dataframe.info()

```
In [6]: #Executando o método info na tabela de Vendas
        vendas2019_dataframe.info()
```

Ao executarmos esse método, o Jupyter Notebook nos mostra a tabela abaixo:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499 entries, 0 to 498
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   CodProduto   499 non-null    int64
 1   CodCor       499 non-null    int64
 2   CodTamanho   499 non-null    int64
 3   CodEstado    499 non-null    int64
 4   DataVenda    499 non-null    datetime64[ns]
 5   Vendedor     499 non-null    int64
 6   QtdeVendida  499 non-null    int64
dtypes: datetime64[ns](1), int64(6)
memory usage: 27.4 KB
```

Nela podemos observar:
O tipo de index e a numeração, nesse caso do número 0 ao 498.
O nome das colunas.
Um contador para dados não nulos para cada coluna.
O tipo de dado de cada coluna.

Usamos esses dados para fazer a limpeza de dados nulos, caso existam, e também para fazer nosso modelo de dados.

2 - Método describe

vendas2019_dataframe.describe()

```
In [7]: #Executando o método describe na tabela de Vendas
        vendas2019_dataframe.describe()
```

Bem parecido com o proc means. Após executar esse código, o Jupyter Notebook mostra essa tabela:

	CodProduto	CodCor	CodTamanho	CodEstado	Vendedor	QtdeVendida
count	499.00000	499.000000	499.000000	499.000000	499.000000	499.00000
mean	1505.53507	5.418838	2.545090	14.004008	2.877756	100.88978
std	877.37977	2.782426	1.138709	7.810763	1.423796	59.91153
min	6.00000	1.000000	1.000000	1.000000	1.000000	1.00000
25%	722.50000	3.000000	2.000000	7.000000	2.000000	48.00000
50%	1545.00000	5.000000	3.000000	14.000000	3.000000	99.00000
75%	2262.00000	8.000000	4.000000	21.000000	4.000000	154.50000
max	3000.00000	10.000000	4.000000	27.000000	5.000000	200.00000

Para cada uma das tabelas com valor numérico, será gerado:

Count, uma contagem para a quantidade de valores não nulos.
Mean, a média aritmética dos valores.
Std, o desvio padrão.
Min, o menor valor encontrado.
25% mostra o primeiro quartil.
50% mostra a mediana.
75% mostra o terceiro quartil.
Max mostra o maior valor encontrado.

Usaremos esses dados para uma análise exploratória, como, por exemplo, ver o valor máximo encontrado em uma coluna ou observar o menor valor encontrado.

3 - Método crosstab

Já usando o método crosstab do pandas, iremos observar a relação entre as colunas, por exemplo:

vendas_por_estado = pd.crosstab(index=vendas2019_dataframe["CodProduto"], columns=vendas2019_dataframe["CodEstado"], margins=True, margins_name="Total")

```
In [9]: #Criando uma tabela de contingência usando o código dos produtos como linhas e o código dos estados como colunas
        vendas_por_estado = pd.crosstab(index=vendas2019_dataframe["CodProduto"],
                                        columns=vendas2019_dataframe["CodEstado"],
                                        margins=True,
                                        margins_name="Total")
```

Index indica o que iremos usar de índice, nesse caso usaremos o código dos produtos, que serão nossas linhas.
Columns indica o que iremos usar para as colunas, nesse caso, usaremos o código do estado de cada produto.
Margins indica se desejamos ou não visualizar as somas das linhas ou das colunas na margem da tabela.
Margins_name indica qual será o nome da coluna e da linha que mostrará as informações acima. Nesse caso, seria Total.

Há diversas formas de usar esse método, acima, porém, usaremos para visualizar a quantidade de vendas de certo produto por estado.

CodEstado	1	2	3	4	5	6	7	8	9	10	...	19	20	21	22	23	24	25	26	27	Total
CodProduto																					
6	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
15	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2972	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2985	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	1
2994	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0	0	0	0	1
3000	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
Total	18	19	19	25	15	14	19	21	14	20	...	18	19	17	20	14	19	22	21	15	499

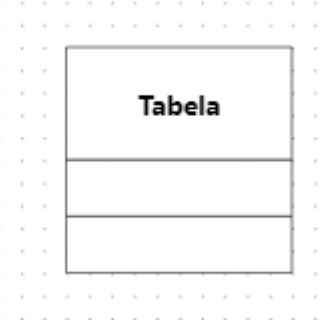
Acima está a tabela mostrada pelo Jupyter Notebook após a execução do método. Podemos observar, por exemplo, que o produto com código 6 foi vendido apenas

uma vez. E que, no total, o estado correspondente ao código 1, vendeu um total de 18 produtos em 2019.

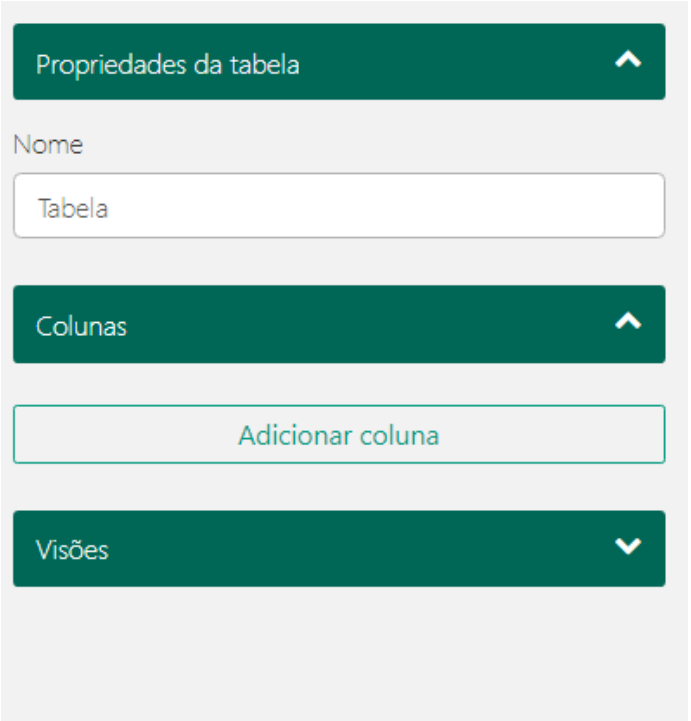
É possível também salvar as tabelas criadas pelo crosstab em uma variável, assim depois podemos usar os métodos info e describe para receber ainda mais informações a fim de analisar os dados.

Criando um modelo de dados usando BrModelo

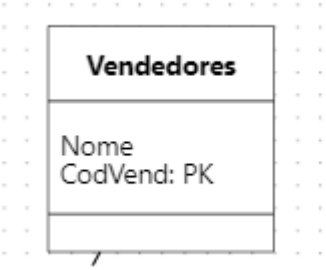
Por fim, usaremos o site https://app.brmodeloweb.com/ para criar nosso modelo de dados baseado na nossa análise. É necessário criar uma conta no site e depois criar um novo modelo lógico.



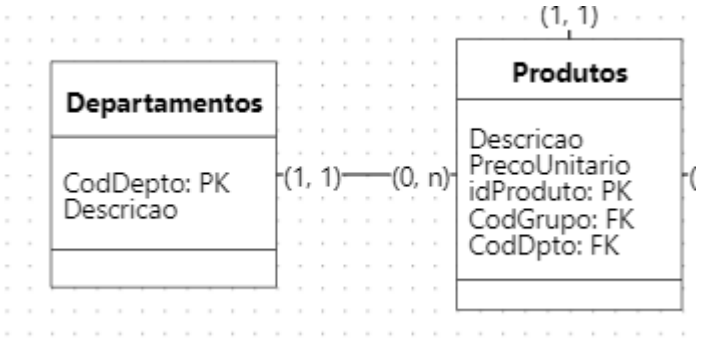
A princípio, teremos uma tabela em branco. Clicamos em cima da tabela depois de arrastá-la para o nosso quadro e então veremos esse menu:



O nome da nossa tabela será correspondente ao nome da tabela contendo os nossos dados, já o nome das colunas corresponderá ao nome das colunas dessa tabela de dados, por exemplo na tabela vendedores:



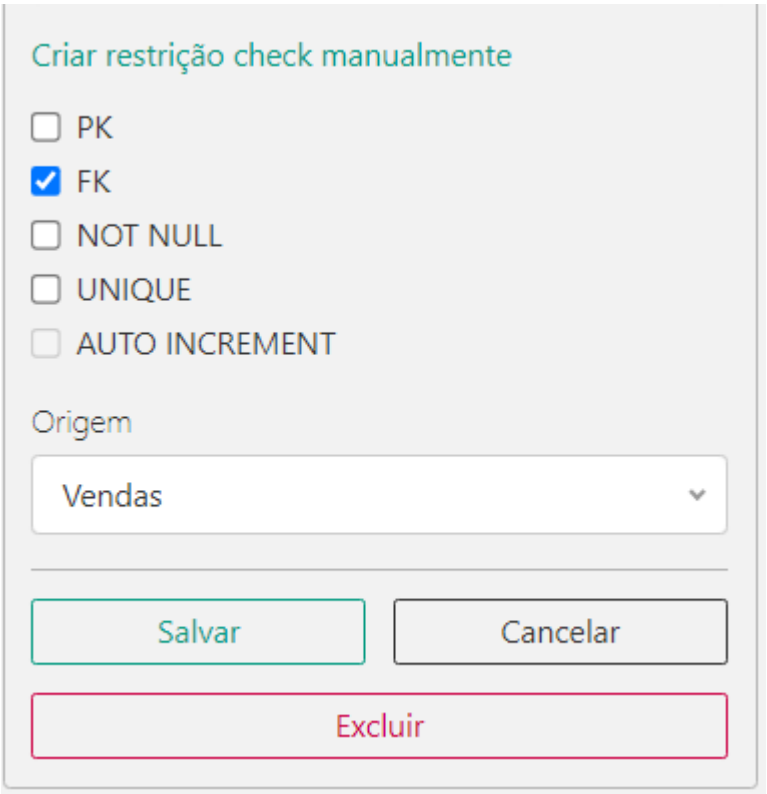
Temos o nome da nossa tabela e dois atributos: o nome do vendedor e o Código do Vendedor. O PK após o nome do nosso significa que temos uma primary key(Chave Primária), ou seja: Um identificador único para a nossa tabela. Nesse caso, um vendedor tem um código único que torna possível identificá-lo.



Já no exemplo acima, temos alguns atributos com FK na frente do nome, o que significa que aquilo é uma foreign key (Chave Estrangeira) que é uma chave em uma tabela que faz referência a uma chave primária em outra tabela. Por exemplo, um produto tem um código de departamento que identifica a qual departamento ele pertence e um departamento tem um código único (Chave Primária) que o identifica.

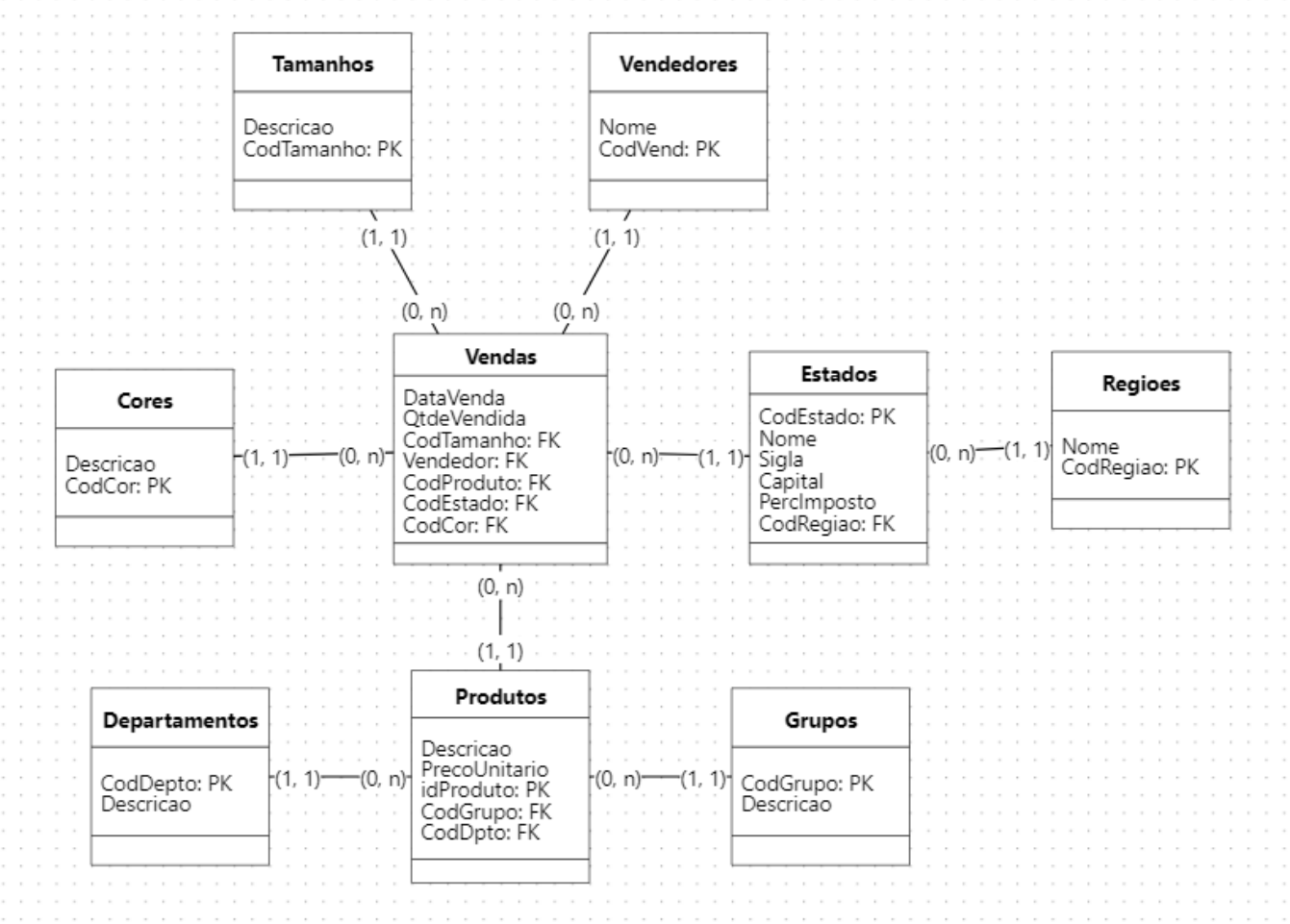
Já falando sobre relacionamentos entre tabelas, no exemplo temos uma relação de 0 para n (0 para vários) entre departamentos e produtos, o que significa que um departamento pode ter 0 ou vários produtos e um relacionamento de 1 para 1 entre produtos e departamentos, o que significa que um produto deve ter um único departamento.

Teremos que fazer esses relacionamentos entre cada uma das tabelas. Para isso, nas opções da tabela selecionamos essas configurações:



Identificamos o atributo como uma chave estrangeira marcando a opção FK e depois selecionamos a tabela onde está a chave primária que dá origem a esse relacionamento. Caso ocorra algum bug, apague tanto a chave primária da tabela de origem quanto a estrangeira da tabela atual e crie as duas novamente.

Por fim teremos um modelo assim:



Após todos esses passos, podemos analisar o tipo de dados de cada tabela e organizá-los e limpá-los de forma que seja possível unir todos para gerarmos uma tabela única com os dados que precisaremos para nossos gráficos.

Atividade 03

Preparação dos Dados

Essa atividade contém:

1 – Transpose dos dados usando SAS

2 – Data Cleaning e Data Quality usando SAS

3 - Join de todas as tabelas (SAS e não SAS) usando Python e pd.merge

Transpose de dados usando SAS

Total rows: 5 Total columns: 62 Rows 1-5

	Nome	04JAN2019	07JAN2019	09JAN2019	13.
1	Carla	108	8	55	
2	Igor	.	67	.	
3	João	.	.	.	
4	Marcelo	.	55	.	
5	Patricia	.	.	.	

Como é possível ver na imagem acima, os dados do arquivo Texto_transpose.txt consistem de 5 linhas (rows) contendo o nome de cada um dos vendedores na primeira coluna e um número que aparenta ser a quantidade de vendas para cada uma das colunas de data.
Dessa maneira é difícil visualizar os dados da tabela, então de modo a simplificar tanto a visualização dos dados como operações futuras (soma de quantidade vendida e afins), iremos usar um proc transpose de modo a mudar a formatação dessa tabela.

```
proc transpose data=gilberto.texto_transpose
out=work.transposed(rename=(_name_=Data col1=Quantidade));

        by nome;

run;
```

É possível fazer isso com o código acima.

1. A opção DATA= indica o arquivo de origem, nesse caso seria a tabela texto_transpose.

2. A opção OUT= indica o arquivo de saída que será gerado após executarmos o código.

3. A opção RENAME= renomeia as colunas criadas. Os argumentos são separados por espaços. Para renomear, primeiro indicamos o nome antigo e, depois do sinal de =, indicamos o novo nome.

4.A opção BY indica a coluna que iremos utilizar para fazer o agrupamentos, no nosso caso, queremos agrupar os valores pelo nome do vendedor, de modo que teremos várias linhas contendo: Um nome, uma data e uma quantidade.

Após executarmos o código, obtemos a tabela transposed:

Total rows: 305 Total columns: 3

	Nome	Data	Quantidade
1	Carla	04JAN2019	108
2	Carla	07JAN2019	8
3	Carla	09JAN2019	55
4	Carla	13JAN2019	179
5	Carla	14JAN2019	234
6	Carla	16JAN2019	174
7	Carla	17JAN2019	106
8	Carla	20JAN2019	162
9	Carla	23JAN2019	265

Opcional

Após observar a tabela transposed, é possível ver que algumas data possuem valores ausentes, representados pelo ponto (**.**).
Podemos limpar a tabela desses dados incompletos usando um data step:

```
data texto_transpose_limpo;

        set work.transposed;

        if not missing(Quantidade) then output;

run;
```

Na primeira linha, definimos o nome que será usado para a tabela de saída do data step: texto_transpose_limpo.
Na segunda, utilizamos **set** para definir de onde virão nossos dados.
Por fim usamos um **if** para verificar se o valor da coluna Quantidade não está ausente, seguido de um output (saída) explícito para esses valores a fim de adicioná-los à tabela de saída.

Data Cleaning e Data Quality usando SAS

Data cleaning é o ato de limpar as tabelas de quaisquer dados incorretos, incompletos,duplicados, corrompidos ou formatados de maneira incorreta das tabelas SAS.
Já o Data Quality observa alguns aspectos dos dados que devem ser: corretos, precisos, completos, integrados e consistentes.

Após analisar nossos dados e perceber inconsistências devemos corrigi-las.

Após analisarmos a tabela Estados, percebemos que alguns estados (Rio Grande do Sul, Santa Catarina e Paraná) possuem um código de região 5. Observando a nossa tabela de regiões, é possível verificar que, atualmente, só há 4 regiões:

	CodRegiao	Nome
1	1	Nordeste
2	2	Norte
3	3	Centro Oeste
4	4	Sudeste

Logo, deduzimos que há algo errado. Para consertar isso, devemos adicionar a região com código 5, que seria a sul.

```
*Adicionando uma região a tabela Regioes;

proc sql;

        *Inserindo o código de região 5 e o nome sul a tabela regioes;

         insert into gilberto.regioes

                values(5, "Sul");

quit;
```

Na primeira linha usamos um **proc sql**.
Na segunda usamos **insert into** e definimos o nome da tabela em que o valor será inserido.
Na terceira, passamos os valores a serem adicionados.
Por fim, usamos **quit** para encerrar esse passo.

Agora temos todas as regiões de todos os estados da tabela Estados.

Observando a tabela Tamanhos, podemos observar que há um valor que se repete.

Total rows: 15 Total columns: 2

	CodGrupo	Descricao
1	1	Nacional
2	2	Importados
3	3	Fabricação Própria
4	4	Consignação
5	5	Grupo 5
6	6	Grupo 6
7	7	Grupo 7
8	8	Grupo 8
9	9	Grupo 9
10	10	Grupo 10
11	10	Grupo 11
12	12	Grupo 12
13	13	Grupo 13
14	14	Grupo 14
15	15	Grupo 15

Para corrigir isso, utilizamos esse código:

```
*Ajustando um código de grupo incorreto;

data gilberto.grupos_corrigido;

        set gilberto.grupos;

        if Descricao="Grupo 11" then CodGrupo=11;

run;
```

O **if** avalia se a descrição da linha atual é igual a “Grupo 11” e, caso seja, redefine o código do grupo como o número 11.

Por fim temos a tabela Vendas, podemos observar algumas inconsistências: tipos de dados incorretos, alguns valores ausentes, códigos de vendedores inválidos, etc.

Para corrigir todos os erros, utilizamos esse código:

```
*Usando um data step para modificar os dados da tabela vendas;

*Rename para mudar o nome das tabela corrigidas;

data gilberto.vendas_limpo(rename=(Vendedor_num=Vendedor
QtdeVendida_num=QtdeVendida));

        set gilberto.vendas;

        *Usando where para selecionar apenas linhas que não tenham nenhum valor
ausente;

        where not missing(CodProduto)

        and not missing(CodCor)

        and not missing(CodTamanho)

        and not missing(CodEstado)

        and not missing(DataVenda)

        and not missing(Vendedor)

        and not missing(QtdeVendida);

        *Criando colunas com valores númericos para a coluna Vendedores e
QtdeVendida;

        Vendedor_Num=input(Vendedor, 8.);

        QtdeVendida_Num=input(QtdeVendida, 8.);

        /*Verificando se foi possível a conversão de char para num, caso não seja
possível

        isso significa que o código de vendedor é inválido, portanto também o
removemos do

        dataset */

        if not missing(Vendedor_Num) then output;
```

```
        *Removendo as tabelas Vendedor e QtdeVendida;

        drop Vendedor QtdeVendida;

run;
```

No código acima, a opção **where** nos permite filtrar as linhas da nossa tabela. Usamos a função **missing** para verificar se em alguma das colunas há algum valor ausente e selecionar apenas linhas em que todos os valores estejam presentes. Após isso utilizamos a função **input** para transformar os dados das colunas Quantidades e Vendedor em valores numéricos.
Após a conversão acima, verificamos se foi possível fazer a conversão de char para num na coluna vendedores. Caso não seja possível, sabemos que temos um código de vendedor inválido (De acordo com nossa tabela Vendedores) e então usamos um if para selecionar apenas as linhas que contenham um valor numérico na coluna Vendedor_num.
Por fim, de modo a simplificar nosso **merge** futuramente, iremos utilizar um **rename** na nossa tabela de saída, de modo a renomear as tabelas numéricas criadas com os nomes das tabelas originais. Também usamos **drop** para remover as tabelas originais da nossa nova tabela.
A nossa tabela final: vendas_limpo deve conter somente valores válidos, para verificar isso podemos usar **proc means** e **proc freq**:

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
2	CodCor	Num	8	BEST.	CodCor
4	CodEstado	Num	8	BEST.	CodEstado
1	CodProduto	Num	8	BEST.	CodProduto
3	CodTamanho	Num	8	BEST.	CodTamanho
5	DataVenda	Num	8	MMDDYY10.	DataVenda
7	QtdeVendida	Num	8		
6	Vendedor	Num	8		

Como mostrado acima, agora todas as nossas colunas possuem valores numéricos.

The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
CodProduto	CodProduto	14989	1502.18	862.2661195	1.0000000	3000.00
CodCor	CodCor	14989	5.5151778	2.8902831	1.0000000	10.0000000
CodTamanho	CodTamanho	14989	2.5073721	1.1207589	1.0000000	4.0000000
CodEstado	CodEstado	14989	14.0926012	7.7981673	1.0000000	27.0000000
DataVenda	DataVenda	14989	22396.09	279.0051270	21915.00	22875.00
Vendedor		14989	3.0085396	1.4185686	1.0000000	5.0000000
QtdeVendida		14989	102.2181600	103.6572199	1.0000000	6000.00

Acima, podemos ver que em N todos os 14989 resultados da nossa tabela foram levados em conta durante o cálculo dos valores, o que significa que não temos valores ausentes.

Por fim, observando os resultados do **proc freq** podemos confirmar que não há mais valores ausentes na nossa tabela, visto que não há indicação de Frequency Missing nas tabelas do resultado.

Join de todas as tabelas (SAS e não SAS) usando Python e pd.merge

Por fim, é necessário unir todas as nossas tabelas. Para isso usaremos a biblioteca pandas e o método merge em conjunto com a biblioteca saspy para acessar as tabelas SAS.

1 - Importando a biblioteca SASPY e conectando ao SAS

Para começar é necessário importar a biblioteca saspy:

import saspy

```
import saspy
```

Adicionamos a linha de código acima junto às outras importações de biblioteca.

Em seguida iremos criar uma variável chamada sas e utilizaremos o método do saspy SASsession para iniciar a nossa conexão. Esse método toma como argumento o nome da conexão previamente configurada, no nosso caso, vmacademy2:

sas= saspy.SASsession(cfgname ="vmacademy2");

```
In [14]: sas= saspy.SASsession(cfgname ="vmacademy2");
```

Após executar esse código, surgirá uma caixa de texto abaixo que solicitará o nome e a senha. Ao inserir essas informações, deverá aparecer essa mensagem:

```
SAS Connection established.
```

Isso indica que tudo ocorreu de maneira correta e que foi possível estabelecer a conexão com o SAS.

Analisando as colunas, é possível perceber que a tabela vendas2019 e a tabela vendas do SAS contém as mesmas colunas, o que significa que ambas contém informações sobre as vendas. Dito isso, é necessário unir as duas para que seja possível visualizar os dados de todas as vendas. Para isso utilizamos esse código para transformar a tabela sas em um dataframe:

vendas_dataframe = sas.sasdata(libref='gilberto', table='vendas_limpo').to_df()

```
In [15]: #Criando um dataframe a partir da tabela SAS de vendas pós limpeza e análise dos dados
         vendas_dataframe = sas.sasdata(libref='gilberto', table='vendas_limpo').to_df()
```

Como argumento temos:

LIBREF que indica o nome da biblioteca que queremos acessar.
TABLE que indica o nome da tabela que queremos acessar.

Por fim, utilizamos o método to_df para transformar os dados dessa tabela em um dataframe.

concat = pd.concat([vendas2019_dataframe, vendas_dataframe], ignore_index=True)

```
In [17]: #Concatenando o dataframe vendas com o dataframe vendas2019
         concat = pd.concat([vendas2019_dataframe, vendas_dataframe], ignore_index=True)
```

Para a concatenação, passamos para o método concat uma lista com as duas tabelas a serem concatenadas. Já o ignore_index diz para a nova tabela ignorar o índice das tabelas anteriores e criar um novo começando do 0 até o último índice da tabela concatenada.

Agora que temos todas as linhas juntas, precisamos adicionar informações sobre essas linhas. Para isso, usaremos o método merge do pandas:

merge_estados_vendas = pd.merge(concat, estados_dataframe, on="CodEstado")

```
In [19]: #Fazendo merge da tabela estados e da tabela vendas
         merge_estados_vendas = pd.merge(concat, estados_dataframe, on="CodEstado")
```

Para o método merge, passamos as duas colunas que queremos relacionar e após isso passamos um argumento “on” para nomear a coluna que será usada para fazer esse relacionamento.
Acima, vendo a tabela de vendas e a tabela de estados, é possível observar que ambas possuem uma coluna chamada “CodEstado”, usaremos isso para o nosso merge.
Após a execução do código acima, teremos uma tabela armazenada na variável merge_estados_vendas com as linhas da tabela da variável concat e uma junção das colunas da tabela de vendas com as colunas da tabela estados.

Ou seja, agora as linhas terão informações sobre: O nome do estado, a sigla, a capital, a porcentagem de imposto e o código da região.

Esse é o objetivo do merge, adicionar o máximo possível de informação as nossas linhas, pegando informações de outras tabelas e adicionando a nossa tabela principal.

Utilizamos o mesmo processo para a nossa tabela de vendedores:

#Criando um dataframe a partir da tabela SAS Vendedores

vendedores_dataframe = sas.sasdata(libref='gilberto', table='vendedores').to_df()

#Fazendo merge do dataframe anterior com com o dataframe vendedores de acordo com o código do vendedor

merge_vendedores = pd.merge(merge_estados_vendas, vendedores_dataframe, left_on="Vendedor", right_on="CodVend")

#Renomeando as colunas com nomes iguais

merge_vendedores.rename(columns={'Nome_x': 'NomeEstado', 'Nome_y': 'NomeVendedor'}, inplace=True)

Usamos os argumentos left_on e right_on para definir o nome da coluna em comum, quando as colunas em comum das tabelas possuírem um nome diferente. Também

utilizamos o método rename para renomear as colunas de forma a especificar o máximo possível que informação ela contém. Para esse método passamos o argumentos colums seguido de um dicionário contendo o nome original seguido do novo nome. O argumento inplace do método declara se queremos que o nome seja atualizado na nossa tabela atual e por padrão cria uma tabela nova com os nomes atualizados.

```
#Criando um dataframa da tabela de Cores do SAS

cores_dataframe = sas.sasdata(libref='gilberto', table='Cores').to_df()

#Fazendo o merge do dataframe anterior com o dataframe Cores, de acordo com o código de cor

merge_cores = pd.merge(merge_vendedores, cores_dataframe, on="CodCor")

#Renomeando a coluna Descricao

merge_cores.rename(columns={'Descricao': 'DescricaoCor'}, inplace=True)
```

Fazemos o merge da tabela cores de acordo com o código de cor.

```
#Criando um dataframe da tabela SAS Regioes

regioes_dataframe = sas.sasdata(libref='gilberto', table='Regioes').to_df()

#Fazendo merge do dataframe anterior com o dataframe Regioes

regioes_merge = pd.merge(merge_cores, regioes_dataframe, on="CodRegiao")

#Renomando a coluna Nome

regioes_merge.rename(columns={'Nome': 'NomeRegiao'}, inplace=True)
```

Depois o merge da tabela regiões de acordo com o código da região.

```
#Criando um dataframe da tabela SAS Tamanhos

tamanhos_dataframe = sas.sasdata(libref='gilberto', table='Tamanhos').to_df()

#Fazendo o merge do dataframe anterior com o dataframe Tamanhos
```

```
tamanhos_merge = pd.merge(regioes_merge, tamanhos_dataframe,
on="CodTamanho")

#Renomeando a coluna Descrição

tamanhos_merge.rename(columns={'Descricao': 'DescricaoTamanho'},
inplace=True)
```

O mesmo para a tabela tamanhos, de acordo com o código de tamanho.

```
#Criando um dataframe da tabela SAS Produtos

produtos_dataframe = sas.sasdata(libref='gilberto', table='Produtos').to_df()

#Fazendo o merge da tabela anterior como o dataframe Produtos

produtos_merge = pd.merge(tamanhos_merge, produtos_dataframe,
left_on="CodProduto", right_on="idProduto")

#Renomeando a coluna Descricao

produtos_merge.rename(columns={'Descricao': 'DescricaoProduto'}, inplace=True)
```

O mesmo para a tabela de produtos.

```
#Criando um dataframe da tabela SAS Grupos

grupos_dataframe = sas.sasdata(libref='gilberto', table='Grupos_corrigido').to_df()

#Fazendo o merge da tabela anterior como o dataframe Grupos

grupos_merge = pd.merge(produtos_merge, grupos_dataframe, on="CodGrupo")

#Renomeando a coluna Descricao

grupos_merge.rename(columns={'Descricao': 'DescricaoGrupo'}, inplace=True)
```

Relacionamos a tabela de grupos também.

```
#Fazendo o merge do dataframe anterior com o dataframe departamentos

departamentos_merge = pd.merge(grupos_merge, departamentos_dataframe, on="CodDepto")

#Renomeando a coluna Descricao

departamentos_merge.rename(columns={'Descricao': 'DescricaoDepto'}, inplace=True)
```

Por fim a tabela de departamentos.

É possível verificar que agora teremos 24 colunas usando o método info.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15488 entries, 0 to 15487
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   CodProduto        15488 non-null  float64
 1   CodCor            15488 non-null  float64
 2   CodTamanho        15488 non-null  float64
 3   CodEstado         15488 non-null  float64
 4   DataVenda         15488 non-null  datetime64[ns]
 5   Vendedor          15488 non-null  float64
 6   QtdeVendida       15488 non-null  float64
 7   NomeEstado        15488 non-null  object
 8   Sigla             15488 non-null  object
 9   Capital           15488 non-null  object
 10  PercImposto       15488 non-null  float64
 11  CodRegiao         15488 non-null  float64
 12  CodVend           15488 non-null  float64
 13  NomeVendedor      15488 non-null  object
 14  DescricaoCor      15488 non-null  object
 15  NomeRegiao        15488 non-null  object
 16  DescricaoTamanho  15488 non-null  object
 17  idProduto         15488 non-null  float64
 18  DescricaoProduto  15488 non-null  object
 19  CodGrupo          15488 non-null  float64
 20  CodDepto          15488 non-null  float64
 21  PrecoUnitario     15488 non-null  float64
 22  DescricaoGrupo    15488 non-null  object
 23  DescricaoDepto    15488 non-null  object
dtypes: datetime64[ns](1), float64(13), object(10)
memory usage: 3.0+ MB
```

No total temos 15488 linhas e, como podemos verificar, não há valores nulos no nosso dataframe/tabela.
Porém, ainda há detalhes a serem arrumados. Por exemplo: A coluna do código de cor atualmente está definida como um valor float, ou seja, números decimais. Quando convertido para CSV ao invés do código “1” teremos um código “10”.
Para arrumar isso, é necessário mudar o tipo de dados das colunas usando esse código:

```
#Mudando o tipo de dados das colunas para int - Números inteiros
```

```
departamentos_merge['CodProduto'] =
departamentos_merge['CodProduto'].astype(int)

departamentos_merge['CodCor'] = departamentos_merge['CodCor'].astype(int)

departamentos_merge['CodTamanho'] =
departamentos_merge['CodTamanho'].astype(int)

departamentos_merge['CodEstado'] =
departamentos_merge['CodEstado'].astype(int)

departamentos_merge['Vendedor'] = departamentos_merge['Vendedor'].astype(int)

departamentos_merge['QtdeVendida'] =
departamentos_merge['QtdeVendida'].astype(int)

departamentos_merge['CodRegiao'] =
departamentos_merge['CodRegiao'].astype(int)

departamentos_merge['CodVend'] = departamentos_merge['CodVend'].astype(int)

departamentos_merge['idProduto'] = departamentos_merge['idProduto'].astype(int)

departamentos_merge['CodGrupo'] = departamentos_merge['CodGrupo'].astype(int)

departamentos_merge['CodDepto'] = departamentos_merge['CodDepto'].astype(int)

departamentos_merge['PrecoUnitario'] =
departamentos_merge['PrecoUnitario'].astype(int)
```

Há uma coluna (PercImposto) em que não será necessário fazer essa conversão, visto que a porcentagem de imposto por estado é dada em valores decimais. Usando o método info no nosso dataframe, é possível ver se tudo ocorreu adequadamente:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15488 entries, 0 to 15487
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   CodProduto        15488 non-null  int32
 1   CodCor            15488 non-null  int32
 2   CodTamanho        15488 non-null  int32
 3   CodEstado         15488 non-null  int32
 4   DataVenda         15488 non-null  datetime64[ns]
 5   Vendedor          15488 non-null  int32
 6   QtdeVendida       15488 non-null  int32
 7   NomeEstado        15488 non-null  object
 8   Sigla             15488 non-null  object
 9   Capital           15488 non-null  object
 10  PercImposto       15488 non-null  float64
 11  CodRegiao         15488 non-null  int32
 12  CodVend           15488 non-null  int32
 13  NomeVendedor      15488 non-null  object
 14  DescricaoCor      15488 non-null  object
 15  NomeRegiao        15488 non-null  object
 16  DescricaoTamanho  15488 non-null  object
 17  idProduto         15488 non-null  int32
 18  DescricaoProduto  15488 non-null  object
 19  CodGrupo          15488 non-null  int32
 20  CodDepto          15488 non-null  int32
 21  PrecoUnitario     15488 non-null  int32
 22  DescricaoGrupo    15488 non-null  object
 23  DescricaoDepto    15488 non-null  object
dtypes: datetime64[ns](1), float64(1), int32(12), object(10)
memory usage: 2.2+ MB
```

Pronto. Agora exportamos esses dados para um arquivo CSV usando o seguinte código:

```
departamentos_merge.to_csv('SDCAcademy.csv',  decimal=',', float_format='%.3f')
```

Chamamos o método to_csv para converter nosso dataframe pra um arquivo CSV.
O primeiro argumento indica o nome do arquivo.
O segundo (decimal) indica o separador que queremos usar para os valores decimais. Vírgula no nosso caso.
O terceiro indica como queremos formatar os valores floats, nesse caso, com 3 casas decimais após a vírgula.

Atividade 04

Análise dos Dados e Relatórios

Para a atividade final, devemos fazer uma análise dos nossos dados e criar relatórios para que possamos apresentá-los.

1 – Importar os arquivos para o PowerBI

2 – Criação das colunas adicionais

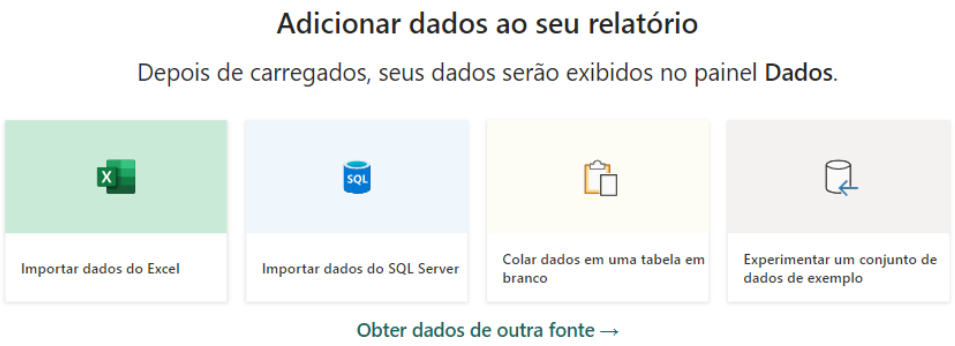
3 – Criação dos gráficos relacionados às Vendas

4 – Criação dos gráficos relacionados às Comissões, Impostos e Faturamento

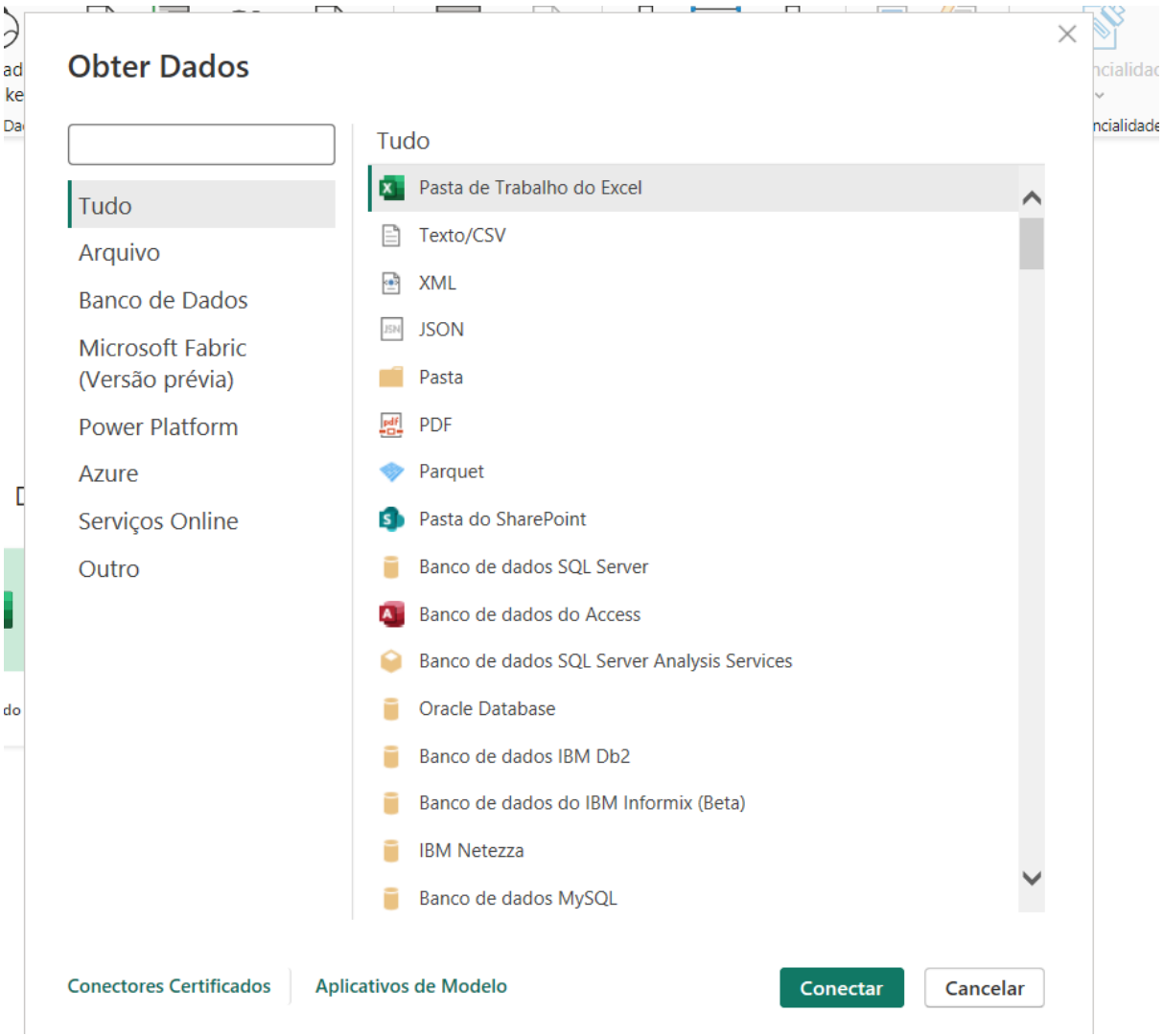
Importando os dados para o PowerBI

O arquivo contendo nossos dados estará na pasta C:\Users\gilberto\Desktop\SDC AcademY\Codigo_e_Portifolio. De forma a termos uma organização melhor, devemos movê-lo para C:\Users\gilberto\Desktop\SDC AcademY\Dados.

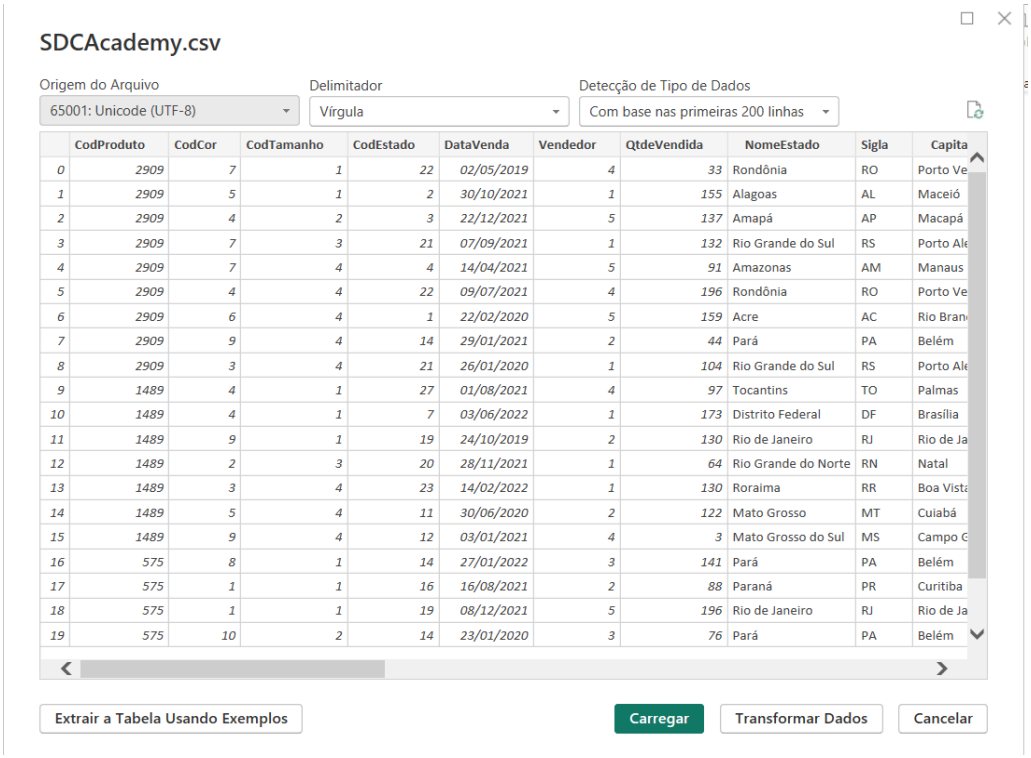
Após isso iremos abrir o PowerBI. Essa tela será apresentada:



Clicamos na opção “Obter dados de outra fonte”.



Na tela seguinte clicamos em Texto/CSV e navegamos até onde se encontra o arquivo com nossos dados. Após selecionar nosso arquivo e confirmar, será apresentada essa tela:



	CodProduto	CodCor	CodTamanho	CodEstado	DataVenda	Vendedor	QtdeVendida	NomeEstado	Sigla	Capita
0	2909	7	1	22	02/05/2019	4	33	Rondônia	RO	Porto Ve
1	2909	5	1	2	30/10/2021	1	155	Alagoas	AL	Maceió
2	2909	4	2	3	22/12/2021	5	137	Amapá	AP	Macapá
3	2909	7	3	21	07/09/2021	1	132	Rio Grande do Sul	RS	Porto Ale
4	2909	7	4	4	14/04/2021	5	91	Amazonas	AM	Manaus
5	2909	4	4	22	09/07/2021	4	196	Rondônia	RO	Porto Ve
6	2909	6	4	1	22/02/2020	5	159	Acre	AC	Rio Bran
7	2909	9	4	14	29/01/2021	2	44	Pará	PA	Belém
8	2909	3	4	21	26/01/2020	1	104	Rio Grande do Sul	RS	Porto Ale
9	1489	4	1	27	01/08/2021	4	97	Tocantins	TO	Palmas
10	1489	4	1	7	03/06/2022	1	173	Distrito Federal	DF	Brasília
11	1489	9	1	19	24/10/2019	2	130	Rio de Janeiro	RJ	Rio de Ja
12	1489	2	3	20	28/11/2021	1	64	Rio Grande do Norte	RN	Natal
13	1489	3	4	23	14/02/2022	1	130	Roraima	RR	Boa Vista
14	1489	5	4	11	30/06/2020	2	122	Mato Grosso	MT	Cuiabá
15	1489	9	4	12	03/01/2021	4	3	Mato Grosso do Sul	MS	Campo G
16	575	8	1	14	27/01/2022	3	141	Pará	PA	Belém
17	575	1	1	16	16/08/2021	2	88	Paraná	PR	Curitiba
18	575	1	1	19	08/12/2021	5	196	Rio de Janeiro	RJ	Rio de Ja
19	575	10	2	14	23/01/2020	3	76	Pará	PA	Belém

Clicamos em “Carregar”. Essa tela será apresentada:

Podemos clicar no ícone de tabela na parte superior esquerda para visualizar os dados carregados:

Column1	CodProduto	CodCor	CodTamanho	CodEstado	DataVenda	Vendedor	QtdeVendida	NomeEstado	Sigla	Capital	PercImposto	CodRe
134	381	3	2	10	terça-feira, 10 de agosto de 2021	4	5	Maranhão	MA	São Luís	0,09	
168	850	10	1	10	terça-feira, 15 de março de 2022	4	152	Maranhão	MA	São Luís	0,09	
570	427	2	1	10	quinta-feira, 23 de dezembro de 2021	4	71	Maranhão	MA	São Luís	0,09	
606	2102	5	4	10	sábado, 29 de agosto de 2020	4	167	Maranhão	MA	São Luís	0,09	
652	2084	6	1	10	quarta-feira, 16 de dezembro de 2020	4	194	Maranhão	MA	São Luís	0,09	
695	884	9	4	10	domingo, 1 de março de 2020	4	91	Maranhão	MA	São Luís	0,09	
984	736	9	4	10	terça-feira, 4 de fevereiro de 2020	4	49	Maranhão	MA	São Luís	0,09	
1132	2386	7	4	10	terça-feira, 29 de setembro de 2020	4	110	Maranhão	MA	São Luís	0,09	
1229	595	1	1	10	segunda-feira, 6 de junho de 2022	4	71	Maranhão	MA	São Luís	0,09	
1481	1116	5	2	10	domingo, 13 de março de 2022	4	70	Maranhão	MA	São Luís	0,09	
1581	17	1	2	10	segunda-feira, 28 de fevereiro de 2022	4	99	Maranhão	MA	São Luís	0,09	
1887	2270	3	1	10	domingo, 25 de julho de 2021	4	120	Maranhão	MA	São Luís	0,09	
2003	521	10	3	10	domingo, 25 de abril de 2021	4	104	Maranhão	MA	São Luís	0,09	
2030	1571	9	2	10	segunda-feira, 16 de maio de 2022	4	186	Maranhão	MA	São Luís	0,09	
2046	1985	2	2	10	sexta-feira, 10 de junho de 2022	4	93	Maranhão	MA	São Luís	0,09	
2114	157	8	3	10	quinta-feira, 4 de março de 2021	4	37	Maranhão	MA	São Luís	0,09	
2227	2732	8	1	10	quarta-feira, 29 de junho de 2022	4	45	Maranhão	MA	São Luís	0,09	
2266	2504	1	3	10	segunda-feira, 13 de setembro de 2021	4	33	Maranhão	MA	São Luís	0,09	
2322	857	5	3	10	domingo, 18 de julho de 2021	4	96	Maranhão	MA	São Luís	0,09	
2366	2383	3	1	10	quinta-feira, 3 de março de 2022	4	80	Maranhão	MA	São Luís	0,09	
2574	1998	8	3	10	sexta-feira, 7 de fevereiro de 2020	4	31	Maranhão	MA	São Luís	0,09	
2644	893	6	3	10	segunda-feira, 11 de julho de 2022	4	3	Maranhão	MA	São Luís	0,09	
2726	1492	9	2	10	segunda-feira, 10 de agosto de 2020	4	69	Maranhão	MA	São Luís	0,09	
2733	2532	4	1	10	terça-feira, 30 de novembro de 2021	4	195	Maranhão	MA	São Luís	0,09	
2734	2532	6	1	10	quinta-feira, 2 de janeiro de 2020	4	156	Maranhão	MA	São Luís	0,09	
2746	1961	4	4	10	quinta-feira, 21 de maio de 2020	4	26	Maranhão	MA	São Luís	0,09	
2747	1961	4	4	10	domingo, 17 de abril de 2022	4	120	Maranhão	MA	São Luís	0,09	

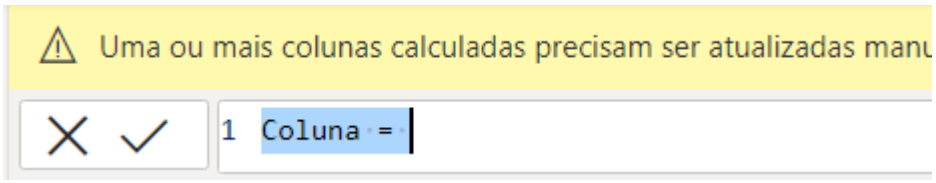
Com isso concluímos a exportação dos nossos dados para o PowerBI.

Criação das colunas adicionais

Precisaremos mostrar os dados sobre a comissão para cada uma das vendas. Porém, como é possível observar na nossa tabela atual, não possuímos uma coluna que contenha essa informação. Será necessário obter essas informações.

Para isso clicamos na opção “Ferramentas da Tabela” e selecionamos a opção “Nova coluna”:

Após isso surgirá uma nova coluna na tabela. Iremos primeiro definir o nome:



```
1 Comissão =
```

Agora, depois do símbolo de igual, iremos fazer o cálculo ou expressão que definirá os valores para a nossa coluna.

As instruções para calcular a comissão são:

***Comissão= calcular 5% sobre o valor líquido da venda**

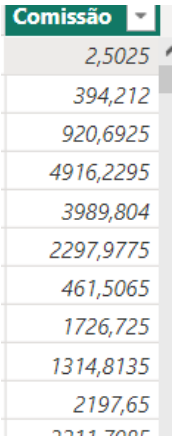
De acordo com a tabela, temos acesso ao preço unitário de cada produto vendido, quantidade vendida e porcentagem de imposto por estado.

Para calcular o valor bruto de uma venda, multiplicamos o preço unitário pela quantidade vendida. Para calcular o valor líquido, iremos subtrair a porcentagem paga em imposto do valor bruto. Após isso teremos o valor líquido e poderemos multiplicar por 0,05 para obtermos a comissão por venda, representando 5% desse valor líquido.

((SDCAcademy[QtdeVendida] * SDCAcademy[PrecoUnitario]) - (SDCAcademy[QtdeVendida] * SDCAcademy[PrecoUnitario] * SDCAcademy[PercImposto])) * 0.05

No cálculo acima, os parênteses são usados para que possamos demarcar a ordem das operações. Então, por exemplo, o valor bruto será calculado (primeira linha), o imposto será calculado (segunda e terceira linhas), esses dois valores serão subtraídos e só então serão multiplicados por 0,05, gerando o valor da comissão

Após colar o cálculo acima após o sinal de igual e apertar enter, teremos uma coluna assim:



Também será necessário mostrar os dados sobre impostos e faturamento.
Para isso usaremos o mesmo procedimento de criar uma nova coluna, porém com esse cálculo para os impostos:

SDCAcademy[QtdeVendida] * SDCAcademy[PrecoUnitario] * SDCAcademy[PercImposto]

```
1  Imposto = SDCAcademy[QtdeVendida] * SDCAcademy[PrecoUnitario] * SDCAcademy[PercImposto]
```

E esse para o faturamento:

SDCAcademy[PrecoUnitario] * SDCAcademy[QtdeVendida]

```
1  Faturamento = SDCAcademy[PrecoUnitario] * SDCAcademy[QtdeVendida]
```

Assim teremos todos os dados para a criação do nosso dashboard.

Criação dos gráficos relacionados às Vendas

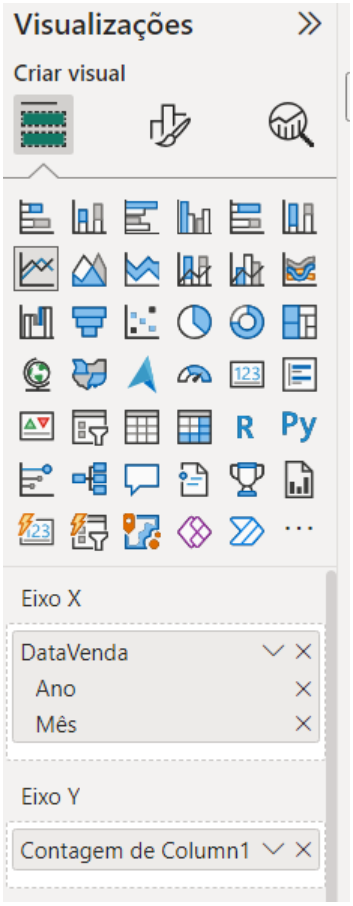
Primeiro, é necessário analisar o que foi pedido pela área de negócios:

Vendas: Produto/Periodo/Estado
Quais os melhores vendedores em cada região?
Qual o produto mais vendido?

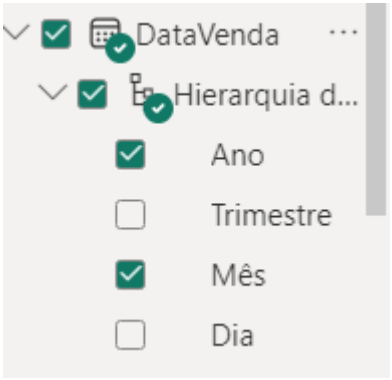
Sabendo disso, devemos criar uma visualização para cada uma dessas informações.

Começaremos pelas vendas por período:

De maneira geral, o gráfico de linhas é uma boa opção para demonstrar a queda ou aumento de vendas durante um certo período de tempo:



Como visto na imagem ao lado, clicamos no ícone do gráfico de linhas.
Selecionamos DataVenda por Mês e Ano para o Eixo X.

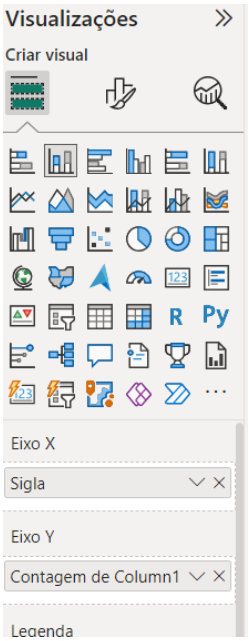


E contagem de Column1, a coluna que contém os nossos índices, para identificar cada uma das vendas durante esse período de tempo e popular nosso Eixo Y.

Esse é o resultado final:

Agora precisamos da quantidade de Vendas por Estado.

Para isso usaremos um gráfico de colunas.

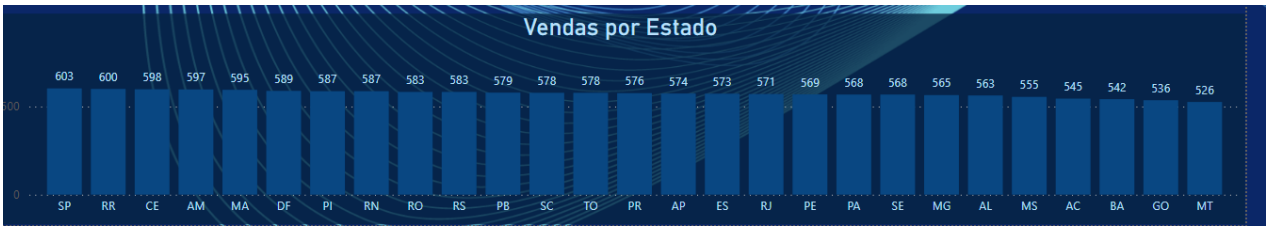


Iremos clicar no ícone do gráfico de colunas.

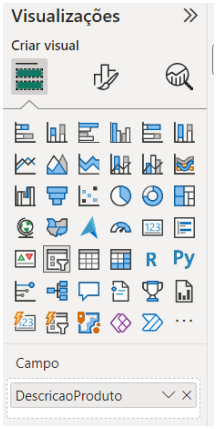
Para o Eixo X queremos a sigla do Estado.

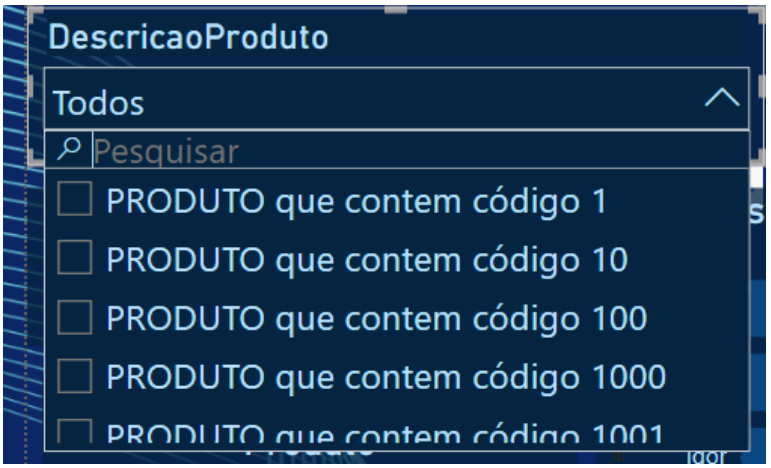
Para o Eixo Y queremos a contagem de índices (Vendas).

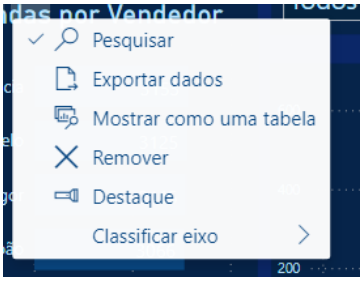
Esse será o nosso resultado final:



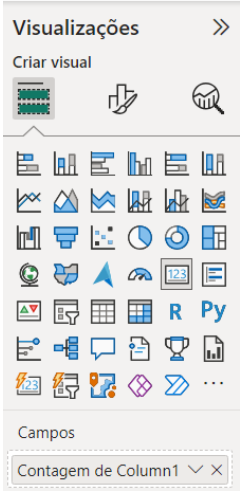
Precisamos também da quantidade de Vendas por Produto. Para isso, visto que é inviável colocar os 3000 códigos de produto em um único gráfico, iremos usar uma abordagem diferente: Usando um filtro e um cartão. Criamos uma Visualização de Filtro e definimos o campos como DescricaoProduto:







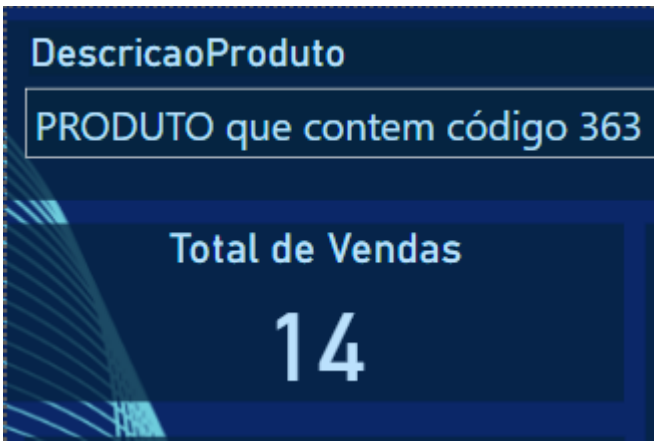
Após aplicar os estilos e ligar a opção de busca (Clicando nos 3 pontos do estilo e marcando a opção “Pesquisar”, ver imagem acima) , nosso filtro está completo. Resta o cartão.



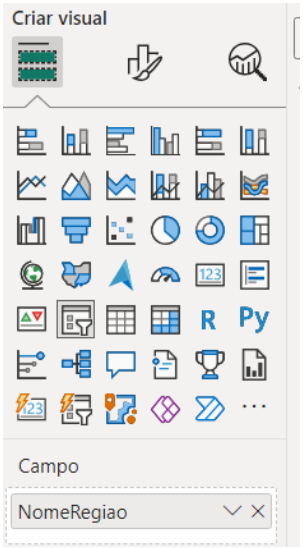
Para o cartão, vamos em visualizações e clicamos no ícone de cartão.

Para os campos, o que queremos é a contagem de Column1, ou seja, a contagem de Vendas.

Então quando filtramos, por exemplo, o produto com código 363, obtemos o total de Vendas deste produto:

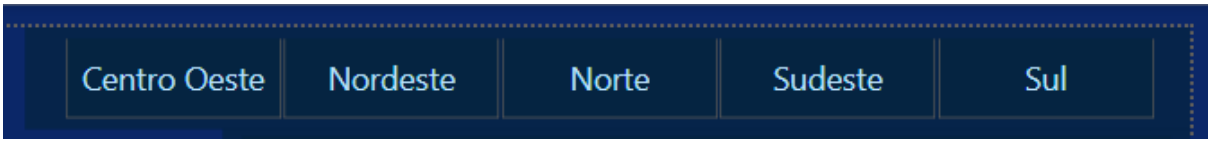


Para saber qual o melhor vendedor de cada região, é necessário criar um filtro de regiões:

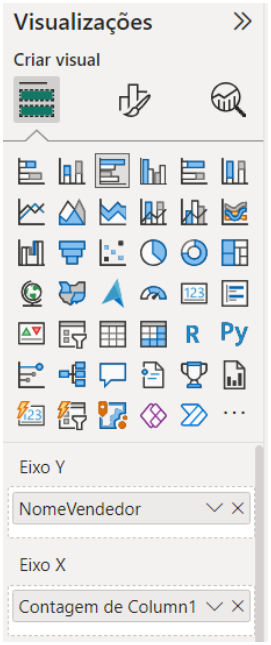


Para o Campo, iremos selecionar NomeRegiao.

Após aplicar os estilos esse é o resultado final:



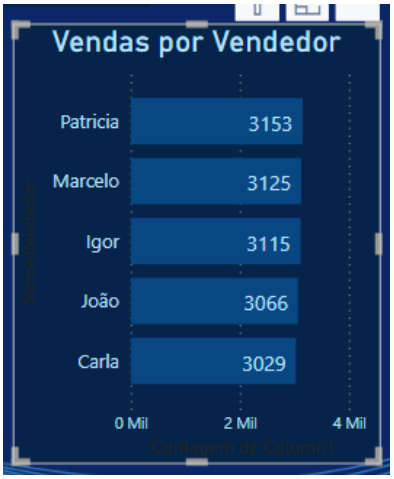
Criamos um gráfico de barras:



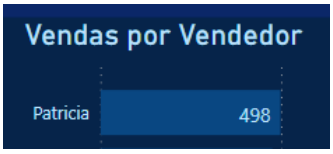
Para o Eixo Y, iremos usar a coluna NomeVendedor.

Para o Eixo X, usaremos a Colum1.

Após isso, teremos um gráfico contabilizando a quantidade de vendas que cada vendedor fez:

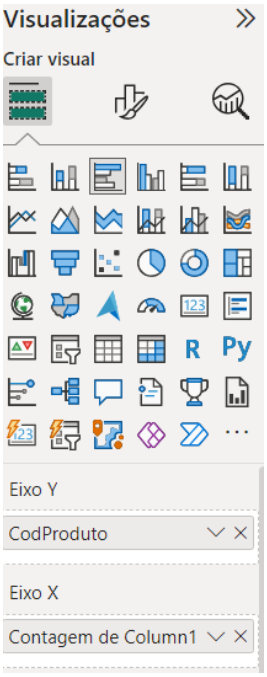


Selecionamos uma região no filtro:



Pronto, podemos saber quem mais fez Vendas de acordo com a região.

Por último, queremos saber qual o produto mais vendido.
Para isso, usaremos um gráfico de barras:

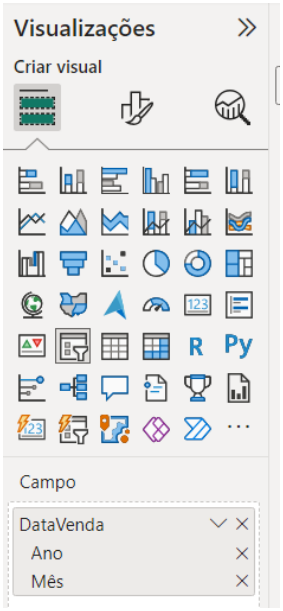


Usamos o código do produto para o Eixo Y.

E a contagem de Column1 (Vendas) para o Eixo X.

Sabemos agora qual o produto mais vendido.

Para finalizar, adicionamos um filtro por mês e ano.



Usamos os campos Ano e Mês da coluna DataVenda.

Pronto, temos nosso filtro por Ano e Mês.

Esse é nosso dashboard final com todas as informações sobre as Vendas:

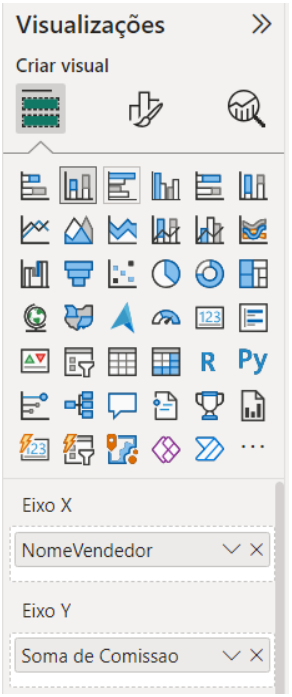
Criação dos gráficos relacionados a Comissões, Impostos e Faturamento

Podemos copiar e colar os filtros de Ano, Mês e Região feitos para o dashboard anterior.

Agora, veremos as informações que teremos que disponibilizar:

Comissão: Vendedor/Período/Região
Qual o Estado que gera maior faturamento?
Quanto foi pago de imposto por estado? Usar recurso de mapa.

Criamos um gráfico de colunas para mostrar as comissões por Vendedor:



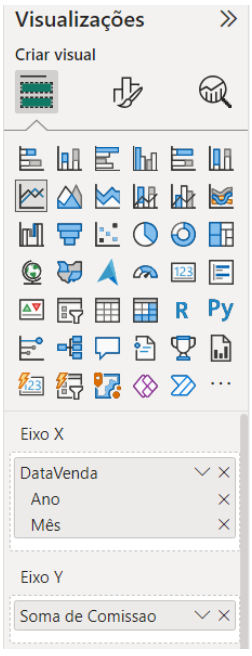
Usamos o NomeVendedor para o Eixo X, representando cada uma das colunas.
Usamos a soma da nossa coluna de Comissão para o Eixo Y, para definir o valor de cada coluna.

Após adicionar um título, o resultado é esse:

Para saber a quantidade de comissão de cada vendedor por região, só precisamos clicar no filtro de regiões e selecionar a região.

Por exemplo, para saber a comissão por vendedor na região Sudeste:

Para a comissão por período, usaremos um gráfico de linhas:



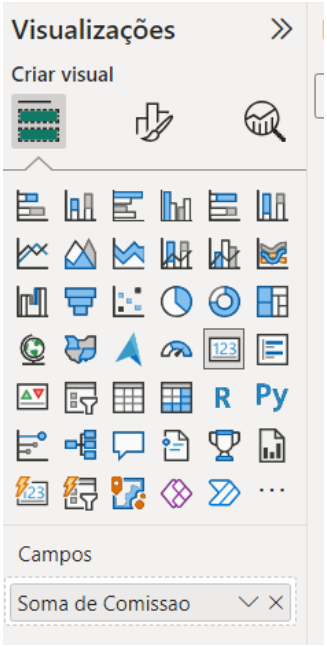
Para o Eixo X, usaremos DataVenda com Ano e Mês.

Para o Eixo X, usaremos a soma da coluna Comissão.

Esse é nosso resultado:

Caso seja necessário, podemos usar o filtro de ano e mês para ver um período mais específico.

Para mostrar a comissão total por região, iremos usar um cartão:



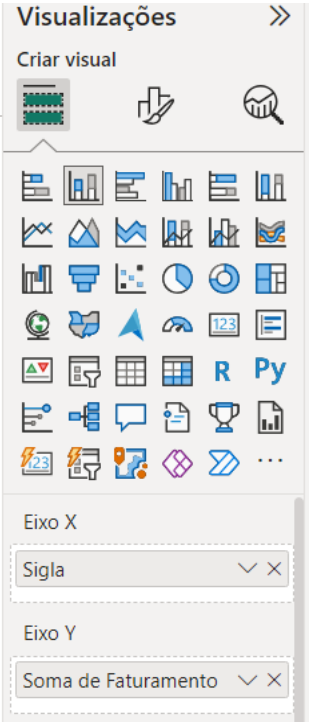
Utilizamos a soma da coluna Comissão como nosso campo.

Após adicionar um título, temos esse esse resultado:

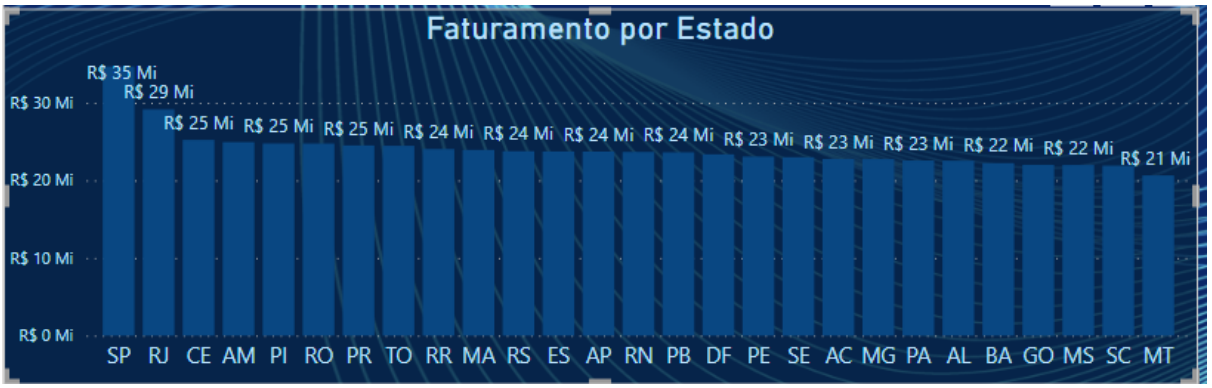
Para saber a comissão total de cada região, só precisamos clicar no filtro de regiões:

Acima temos a comissão total da região Nordeste.

Para sabermos qual o estado com o maior faturamento criaremos um gráfico de colunas.



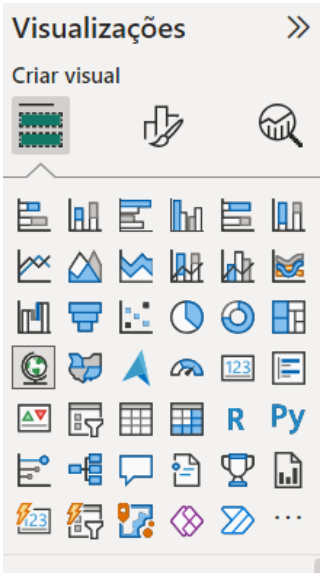
Usamos a sigla de cada estado para o Eixo X e a soma da coluna Faturamento para o eixo Y.



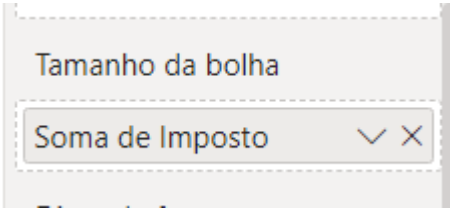
Pronto, agora sabemos que o estado com o maior faturamento é São Paulo.

Por último, precisamos saber quanto de imposto foi pago por cada estado e, de acordo com as instruções, é necessário usar o recurso do mapa.

Criamos uma visualização de mapa:

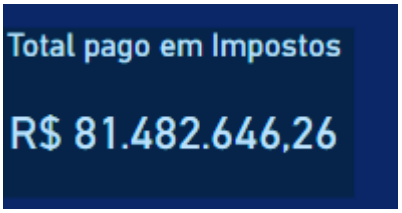


Para a localização, utilizamos o nome do estado.



Para o tamanho da bolha, usaremos a soma de Imposto. Quanto maior a quantidade de imposto, maior será o tamanho da bolha:

De modo a facilitar a visualização criaremos um cartão da quantidade de impostos pagos. Após criar uma visualização de cartão. Adicionamos o campo soma da coluna Impostos e o título.



Podemos clicar na bolha de cada estado para ver a quantidade de impostos pago por ele.

Por exemplo:

Esse será nosso dashboard final: