PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Sérgio Rosemberg Encarnação Junior

Classificação para a ação de fiscalização utilizando modelos de Machine Learning

Rio de Janeiro
2021

Sérgio Rosemberg Encarnação Junior

Classificação para a ação de fiscalização utilizando modelos de Machine Learning

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Rio de Janeiro
2021

SUMÁRIO

1. Introdução

1.1. Contextualização

No intuito de desenvolver um trabalho de ciência de dados e analisando a importância que a ação de fiscalização possui para a qualidade dos produtos e serviços que são prestados por empresas para a população, realizei uma pesquisa sobre o assunto de forma que me possibilitasse ter mais informações para análises e previsões.

Neste sentido, as agências fiscalizadoras têm, entre outras ações, que fiscalizar se dentro de sua área de atuação existem empresas que estão prestando um bom serviço ou disponibilizando bons produtos.

A ação de fiscalização busca coibir práticas ilícitas, punindo os maus fornecedores de combustível, que violam os direitos dos consumidores. A fiscalização assegura que os direitos dos consumidores estão sendo respeitados, porém existem custos e recursos envolvidos nas ações de fiscalização, onde deve-se tentar minimizar estes custos e melhorar o desempenho operacional destas ações.

No mercado de combustíveis, o papel da ANP - Agência Nacional do Petróleo, Gás Natural e Biocombustíveis é de garantir que os combustíveis produzidos e comercializados no País atendam a todos os critérios de qualidade e preço exigidos pelas normas e regulamentações impostas tanto pela ANP quanto por outros órgãos que regulam o comércio desse importante produto.

Tem-se noticiado algumas parcerias entre a ANP - Agência Nacional do Petróleo, Gás Natural e Biocombustíveis e os PROCONS (Autarquia de Proteção e Defesa do Consumidor) para que a ação em conjunto possa ampliar a fiscalização, trazendo benefícios aos consumidores que terão no momento de abastecer seus veículos a garantia que o combustível será de boa qualidade.

Baseado nisto e buscando analisar um conjunto de dados neste contexto, ao pesquisar nos portais dados abertos de agências do governo, encontramos alguns datasets disponibilizados.

No caso de fiscalização e qualidade dos produtos e serviços, encontramos conjuntos de dados abertos da ANP - Agência Nacional do Petróleo, Gás Natural e Biocombustíveis.

No site de dados abertos da ANP encontramos uma descrição de um plano de dados abertos (PDA), disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos:

"Atendendo ao Decreto nº 8.777/2016, a ANP publicou em agosto de 2018 seu Plano de Dados Abertos – PDA, com validade 2018 a 2020, o qual estabeleceu a divulgação dos dados brutos, quando possível, e dos relatórios mais requisitados pela sociedade, em formato aberto, de fácil interpretação e processamento, com o objetivo de dar transparência e entregar, de forma simples e com periodicidade conhecida, os dados custodiados pela ANP, que podem então ser visualizados, estudados e trabalhados pelos cidadãos, acadêmicos, jornalistas e agentes econômicos.

Como seguimento, ainda em 2018, a ANP publicou, sob forma de projeto piloto, um conjunto de dados abertos importantes para a sociedade e mercado regulado, compreendendo dados de produção de petróleo e gás natural, fiscalização de contratos de conteúdo local, participações governamentais, programa de monitoramento da qualidade de combustíveis, rodadas de licitações e série histórica de pesquisa de preços, assim como o registro dos preços informados voluntariamente pelos postos de revenda de combustíveis."

Com estes dados, pretendemos trabalhar a analisar alguns conjuntos de dados que tenham como objetivo classificar se determinada revenda de combustível precisa ser fiscalizada ou não, baseado em atributos comuns para estas revendas, inclusive analisando no aspecto da qualidade de seus produtos, além de preços de compra e venda.

1.2. O problema proposto

Ações de fiscalização geram custo e muitas vezes não são efetivas. No contexto de fiscalização de revendas de combustível, é necessário ser mais efetivo para que as ações de fiscalizações identifiquem situações que precisam ser resolvidas. Revendas de combustíveis que, dentre outras situações, podem estar vendendo combustível de má qualidade para a sociedade, causando prejuízo para o cidadão.

Como classificar quais revendas de combustíveis precisam ser fiscalizadas, baseado em atributos comuns para estas revendas e enriquecendo com algumas características de qualidade?

(Why?) Por que esse problema é importante?

A fiscalização assegura que os direitos dos consumidores estão sendo respeitados. Tentar identificar quais revendas devem ser fiscalizados com maior assertividade poupa despesas e gera efetividade.

(Who?) De quem são os dados analisados? De um governo? Um ministério ou secretaria? Dados de clientes?

Os dados analisados são do Plano de Dados Abertos – PDA da ANP - Agência Nacional do Petróleo, Gás Natural e Biocombustíveis.

(What?): Quais os objetivos com essa análise? O que iremos analisar?

Os motivos que geram a indicação de fiscalização nas revendas de combustível, além de verificar a influência de demais fatores como histórico de multas, reclamações, conformidade e os valores de compra e venda para o consumidor.

(Where?): Trata dos aspectos geográficos e logísticos de sua análise.

Os estados do país.

(When?): Qual o período está sendo analisado? A última semana? Os últimos 6 meses? O ano passado?

Todo o ano de 2019, além de alguns dados históricos durante o período do ano de 2016 a 2019.

1.3. Objetivos

O objetivo é classificar com a maior assertividade possível quais revendas de combustível devem ser fiscalizadas, baseado no histórico de alguns motivos considerados importantes como não conformidade dos produtos, reclamações, multas, valores de compra e venda.

2. Coleta de Dados

Os dados utilizados foram obtidos diretamente da plataforma de dados abertos da ANP - Agência Nacional do Petróleo, Gás Natural e Biocombustíveis, disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos

Dentre os dados disponíveis, utilizei 2 conjuntos de dados principais:

1) Ações de fiscalizações ocorridas em 2019, disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/acoes-de-fiscalizacao

Os dados das ações de fiscalização serão usados como nosso alvo, pois nele temos os registros de ações de fiscalizações já ocorridas nas revendas de combustível.

Segundo o site, estes dados foram disponibilizados com o intuito de dar maior transparência a ações de fiscalização. Tais dados são atualizados mensalmente, com prazo de dois meses entre o mês da fiscalização e o mês da publicação, devido ao atendimento de exigências legais e aspectos operacionais.

Cabe mencionar que escolhi o ano de 2019 para que os dados analisados não tenham interferência da queda de fiscalizações devido a pandemia do corona vírus decretada no início do ano de 2020.

O arquivo em formato .csv, acessado em 17/07/2021, está disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/arquivos-acoes-de-fiscalizacao/acoes-de-fiscalizacao.csv

Metadados das ações de fiscalização, acessado em 17/07/2021, disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/arquivos-acoes-de-fiscalizacao/acoes-de-fiscalizacao-metadados.pdf

Nome da coluna/campo	Descrição	Tipo
UF	Sigla da Unidade da Federação onde está estabelecido o agente fiscalizado.	Texto
Município	Nome do município onde está estabelecido o agente fiscalizado	Texto

Bairro	Nome do bairro onde está estabelecido o agente fiscalizado.	Texto
Endereço	Logradouro onde o agente fiscalizado está estabelecido, bem como demais complementos necessários à correta identificação do local.	Texto
CNPJ/CPF	Nº do Cadastro Nacional da Pessoa Jurídica (CNPJ) do agente fiscalizado. Caso o estabelecimento não tenha CNPJ, o campo apresentará o nº do Cadastro de Pessoa Física (CPF) do responsável pelo estabelecimento.	Texto
Agente Econômico	Nome devidamente registrado sob o qual a pessoa jurídica fiscalizada se individualiza e exerce suas atividades.	Texto
Segmento Fiscalizado	Atividade econômica integrante da indústria do petróleo, do gás natural e dos biocombustíveis exercida pelo agente fiscalizado.	Texto
Data DF	Data do Documento de Fiscalização (DF).	Data
Número do Documento	Nº do Documento de Fiscalização utilizado pela autoridade competente da ANP ou do órgão público conveniado designado para as atividades de fiscalização.	Número
Procedimento de Fiscalização	Procedimento de Fiscalização	Texto

Resultado	Descrição dos fatos verificados durante a fiscalização, conforme o procedimento de fiscalização adotado	Texto

2) Série histórica de preços e combustíveis do ano de 2019

Disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/serie-historica-de-precos-de-combustiveis

Série histórica de pesquisa de preços do 1º e 2º semestres de 2019 sendo os registros dos preços informados voluntariamente pelos postos de revenda de combustíveis. O arquivo em formato .csv, acessado em 17/07/2021, está disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/shpc/dsas/ca/ca-2019-01.csv para o 1º semestre de 2019 e https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/shpc/dsas/ca/ca-2019-02.csv para o 2º semestre de 2019.

Metadados da série histórica de preços de combustível do ano de 2019, acessado em 17/07/2021, disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/shpc/metadados-serie-historica-precos-combustiveis.pdf

Nome da coluna/campo	Descrição	Tipo
Regiao - Sigla	Região	ALFANUMERICO
Estado - Sigla	Estado	ALFANUMERICO
Municipio	Município	ALFANUMERICO
CNPJ da Revenda	Número do Cadastro Nacional de Pessoa Jurídica da revenda	NUMERICO
Nome da Rua	Nome do logradouro	ALFANUMERICO
Numero Rua	Numero do logradouro	ALFANUMERICO
Complemento	Complemento do logradouro	ALFANUMERICO
Bairro	Bairro do logradouro	ALFANUMERICO
Cep	Cep do logradouro	ALFANUMERICO
Produto	Produto	ALFANUMERICO
Data da Coleta	Data da coleta	DATA
Valor de Venda	Preço ao consumidor final	NUMERICO

Valor de Compra	Preço de distribuição	NUMERICO
Unidade de Medida	Unidade de Medida	ALFANUMERICO
Bandeira	Bandeira da revenda	ALFANUMERICO

Com a junção destes 2 conjuntos de dados já é possível realizar a análise dos dados.

Para o enriquecimento das informações contidas nestes conjuntos de dados, inclusive no contexto de análise de fiscalização e qualidade dos produtos e serviços prestados pelo estabelecimento, foram incluídos mais 3 conjuntos de dados (multas aplicadas a partir do ano de 2016, conformidade do produto vendido e reclamações realizadas pela população aos procons):

3) Multas aplicadas com vencimento a partir de 2016, disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/multas-aplicadas-com-vencimento-a-partir-de-2016

Relação de multas aplicadas decorrentes de autuações em ações de fiscalização da ANP. Tem uma periodicidade de extração anual e seus dados serão utilizados na análise para indicar um histórico de multa para os registros analisados. O arquivo em formato .csv, acessado em 17/07/2021, está disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/mav/multas-aplicadas-2016a2019.csv

Metadados das Multas Aplicadas de 2016 a 2019, acessado em 17/07/2021, disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/mav/metadados-multas-aplicadas-2016a2019.pdf

Nome da coluna/campo	Descrição	Tipo
ANO	Ano de aplicação da multa	NÚMERO INTEIRO
STATUS PROCESSO	Situação na qual o processo administrativo se encontra, com os valores possíveis: Aguardando análise de recurso, aguardando pagamento, aguardando pagamento (2ª. instancia), débito extinto rd, débito	TEXTO

	residual, dívida ativa, dívida ativa/execução fiscal, exigibilidade suspensa, parcelamento em andamento, parcelamento rescindido, processo pago.	
SUPERINTENDÊNCIA	Área da ANP responsável pela autuação	TEXTO
NUMERO DO PROCESSO	Número do processo administrativo constituído	TEXTO
AUTO DE INFRAÇÃO	Número do auto de infração	TEXTO
CNPJ/CPF	CPF ou CNPJ do autuado	TEXTO
RAZÃO SOCIAL	Razão social do autuado	TEXTO
DATA TRANSITO JULGADO	Data em que o processo foi transitado em julgado	DATA
VENCIMENTO	Data de vencimento da multa	DATA
VALOR DA MULTA APLICADA	Valor da autuação, em reais	Número real
VALOR TOTAL PAGO	Valor recebido pela ANP, em reais	Número real

4) Não conformidade na qualidade dos combustíveis pelo PMQC – Programa de Monitoramento da Qualidade dos Combustíveis, disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/pmqc-programa-de-monitoramento-da-qualidade-dos-combustiveis

Utilizaremos as informações de não conformidade para avaliar a qualidade dos produtos nas revendas de combustível.

O Programa de Monitoramento da Qualidade dos Combustíveis (PMQC) da ANP faz um monitoramento constante da conformidade de gasolina, do etanol e do óleo diesel comercializados na revenda em território brasileiro.

Realizado por laboratórios contratados pela ANP, por meio de licitações públicas, o PMQC não tem caráter fiscalizatório e não gera autuações (ao contrário das ações de fiscalização, realizadas por servidores da Agência). Os principais objetivos do programa são o levantamento dos indicadores gerais da qualidade dos combustíveis do País e a identificação

de focos de não conformidade, visando à simetria de informações e a orientar e aperfeiçoar a atuação da área de fiscalização da Agência. Os arquivos em formato .csv são disponibilizados por mês. Como nossa análise será para todo o ano de 2019, a relação dos arquivos com seus respectivos links podem ser encontrados abaixo (acessados em 17/07/2021):

Dezembro - csv

Novembro - csv

Outubro - csv

Setembro - csv

Agosto - csv

Julho - csv

Junho - csv

Maio - csv

Abril - csv

Março - csv

Fevereiro - csv

Janeiro - csv

Metadados do PMQC - Programa de Monitoramento da Qualidade dos Combustíveis, acessado em 17/07/2021, disponível em https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/pmqc/pmqc-metadados.pdf

Nome da coluna/campo	Descrição	Tipo
DataColeta	Data no formato ISSO 8601 (AAAA-MM-DD) que a coleta de combustível foi realizada.	DATA
IdNumeric	Identificador único das amostras.	INTEIRO
GrupoProduto	Identifica a família de combustível (Gasolina, Diesel ou Etanol).	TEXTO
Produto	Identifica o produto específico de uma dada família de combustível. Por exemplo,	TEXTO

	para Gasolina há os produtos: Gasolina C Comum, Gasolina C Aditivada e Gasolina C Premium.	
RazaoSocialPosto	Razão Social do Posto Revendedor de Combustível.	TEXTO
CnpjPosto	CNPJ do Posto Revendedor de Combustível.	TEXTO
Distribuidora	Distribuidora associada ao Posto Revendedor de Combustível	TEXTO
Endereço	Endereço do posto de revenda de combustíveis	TEXTO
Complemento	Complemento do endereço	TEXTO
Bairro	Bairro	TEXTO
Município	Município	TEXTO
Latitude	Latitude do Posto Revendedor de Combustível	PONTO FLUTUANTE
Longitude	Longitude do Posto Revendedor de Combustível	PONTO FLUTUANTE
Uf	Unidade da Federação.	TEXTO
RegiaoPolitica	Regiões políticas do Brasil.	TEXTO
Ensaio	Nome do ensaio físico-químico realizado no combustível	TEXTO
Resultado	Resultado do ensaio físico-químico.	TEXTO, INTEIRO OU PONTO FLUTUANTE
UnidadeEnsaio	Unidade do ensaio físico-químico	TEXTO
Conforme	Avaliação de Conformidade do ensaio físico-químico. 1. Não: Ensaio Não Conforme 2. Sim: Ensaio Conforme	BOOLEANO

5) Reclamações registradas nos PROCONS – Sindec

Utilizaremos estes dados para analisar situações de reclamações realizadas contra as revendas de combustível.

O Sistema Nacional de Informações de Defesa do Consumidor – Sindec é um sistema informatizado que integra processos e procedimentos, relativos ao atendimento aos consumidores nos Procons, visando proporcionar um instrumento de gestão adequado ao dinamismo que é típico de seus setores de atendimento.

Ele é resultado de um trabalho integrado, feito segundo a lógica da parceria, constituindo um instrumento que permite amplificar a voz de milhões de consumidores em todo o Brasil. O Sindec integra hoje 26 Procons estaduais e 351 Procons municipais. Como vários desses Procons contam com mais de uma unidade, o Sistema opera em 675 unidades espalhadas por 448 cidades brasileiras. Esses Procons atendem a uma média mensal de 216 mil consumidores.

Na nomenclatura do Sindec, demanda refere-se a todos os tipos de atendimentos realizados pelo Procon. São classificados como demandas desde os procedimentos mais céleres de atendimento, como aqueles realizados por telefone ou por carta encaminhada ao fornecedor, até os processos administrativos instaurados, que na nomenclatura do Sindec são chamados de Reclamação.

O Cadastro Nacional de Reclamações Fundamentadas, assim, é o cadastro formado pelas Reclamações finalizadas pelos Procons integrados ao Sindec, no período de 12 meses.

De todas as demandas registradas no sistema, somente uma parcela é tratada por meio de processos administrativos (Reclamações), já que a maior parte dos Procons se utiliza preponderantemente de tipos de atendimento mais céleres para resolução dos problemas enfrentados pelos consumidores.

Os dados dos atendimentos estão disponíveis em https://dados.gov.br/dataset/cadastro-nacional-de-reclamacoes-fundamentadas-procons-sindec1

O arquivo em formato compactado .zip contendo outro arquivo no formato.csv, acessado em 18/07/2021, está disponível em http://dados.mj.gov.br/dataset/8ff7032a-d6db-452b-89f1-d860eb6965ff/resource/c2cce323-24c2-4430-8918-e24b2966213c/download/crf2019-dados-abertos.zip

Metadados dos dados, acessado em 18/07/2021, disponível em http://dados.mj.gov.br/dataset/8ff7032a-d6db-452b-89f1-

d860eb6965ff/resource/d87543d6-cf9d-4752-8f3c-1b0aa075dc45/download/dicionariodadossindec3-0.pdf

Nome da coluna/campo	Descrição	Tipo
AnoCalendario	Ano calendário de publicação do cadastro de reclamações fundamentadas	Inteiro
DataArquivamento	Data de arquivamento das reclamações	Data
DataAbertura	Data de abertura das reclamações	Data
CodigoRegiao	Código identificador da região do Procon	Texto
Regiao	Região do Procon	Texto
UF	Unidade da Federação do Procon	Texto
strRazaoSocial	Razão social do fornecedor (empresa) na base de dados do Sindec	Texto
strNomeFantasia	Nome fantasia do fornecedor na base de dados do Sindec (nome comercial / popular / fachada)	Texto
Tipo	Código identificador do tipo da pessoa: 1 – Pessoa Jurídica (CNPJ) 0 – Pessoa Física (CPF)	Inteiro
NumeroCNPJ	Número do CNPJ - Cadastro Nacional de Pessoa Jurídica ou CPF - Cadastro de Pessoa Física	Texto
RadicalCNPJ	Aplica-se para pessoa jurídica e serve para agrupar as informações de um mesmo fornecedor (matriz e filiais), sendo os oitos primeiros	Texto

	dígitos do número do CNPJ - Exemplo: a matriz (central) do banco e suas filiais (agências)	
RazaoSocialRFB	Razão social do fornecedor na base de dados da RFB – Receita Federal do Brasil. Obs.: somente para os CNPJs (NumeroCNPJ) válidos na base da RFB	Texto
NomeFantasiaRFB	Nome fantasia do fornecedor na base de dados da RFB – Receita Federal do Brasil Obs.: somente para os CNPJs (NumeroCNPJ) válidos na base da RFB	Texto
CNAEPrincipal	Código identificador da Classificação Nacional de Atividades Econômicas principal do fornecedor. Obs.: somente para os CNPJs (NumeroCNPJ) válidos na base da RFB	Texto
DescCNAEPrincipal	Descrição da Classificação Nacional de Atividades Econômicas principal do fornecedor. Obs.: somente para os CNPJs (NumeroCNPJ) válidos na base da RFB	Texto
Atendida	Código identificador da reclamação fundamentada atendida ou não pela empresa/fornecedor: S – Atendida N – NÃO Atendida	Texto

Código identificador do assunto	Código identificador do assunto	Inteiro
DescricaoAssunto	Descrição dos assuntos do Sindec (produto ou serviço objeto da reclamação)	Texto
CodigoProblema	Código identificador do problema	Inteiro
DescricaoProblema	Descrição dos problemas do Sindec (especificação da lesão sofrida pelo consumidor)	Texto
SexoConsumidor	Código identificador do sexo do consumidor: M – Masculino F – Feminino N – Não se aplica (são as reclamações (de ofício) em que o Procon é o reclamante)	Texto
FaixaEtariaConsumidor	Faixa etária do consumidor distribuída da seguinte forma: - até 20 anos - entre 21 e 30 anos - entre 31 e 40 anos - entre 41 e 50 anos - entre 51 e 60 anos - entre 61 e 70 anos - mais de 70 anos - Nao Informada (data de nascimento não informada no cadastro do consumidor) - Não se aplica (são as reclamações (de ofício) em que o Procon é o reclamante)	Texto
CEPConsumidor	Código identificador do CEP do consumidor (Código de Endereçamento Postal). Obs.: Não se aplica (são as	Texto

	reclamações (de ofício) em que o Procon é o reclamante)	

Podemos observar na tabela que os tipos de dados definidos no metadados, alguns tipos como numéricos, na verdade devem ser usados o tipo ponto flutuante.

Nos metadados dos 5 conjuntos de dados, alguns tipos foram definidos como TEXTO ou ALFANUMÉRICO. Neste caso, trataremos ambos os casos como ALFANUMÉRICO.

Para baixar os arquivos e tratá-los, usaremos o editor “Visual Studio Code” na versão 1.59.0 utilizando a extensão “Jupyter Notebook” e o Python 3.8.1.

Foram criados procedimentos em python para baixar os arquivos necessários dos sites de dados abertos. O código completo para baixar os arquivos dos 5 conjuntos de dados pode ser encontrado no apêndice.

3. Processamento/Tratamento de Dados

Para o tratamento dos dados nos conjuntos vamos utilizar o pandas dataframe, carregando cada conjunto de dados em um dataframe diferente.

Pandas DataFrame é uma estrutura de dados bidimensional com os dados alinhados de forma tabular em linhas e colunas, mutável em tamanho e potencialmente heterogênea.

3.1 Dataframe Fiscalizações

Carregamos o dataframe utilizando o procedimento carregar_dados_de_fiscalizacoes().

```
# carrega os dados de fiscalizações no dataframe
def carregar_dados_de_fiscalizacoes():
    df_fiscalizacoes = pd.read_csv("dados\\fiscalizacoes\\acoes-de-fiscalizacao.csv", encoding="UTF-8", sep=";", skipinitialspace=True, dtype=str)
    return df_fiscalizacoes
```

Ao carregar o dataframe utilizando a função "*read_csv*" do pandas, utilizamos alguns parâmetros para o correto carregamento dos dados:

encoding: Usa a codificação de caracteres UTF-8. Desta forma os valores são carregados com acentuações.

sep: Define o separador das colunas. Para todos os arquivos .csv, o separador é o "ponto e vírgula" (;).

skipinitialspace: Se definido como True, limpa caracteres de espaço antes do valor da coluna.

dtype: define que as colunas serão carregadas com o tipo definido, neste caso string (str).

Após carregar os dados no dataframe, verificamos suas colunas com o procedimento "*info()*". Este procedimento informa um resumo da estrutura do dataframe, como nome das colunas, seus tipos e quantidade total de registros.

Dataframe Fiscalizacoes

```
df_fiscalizacoes = carregar_dados_de_fiscalizacoes()
df_fiscalizacoes.info()
```

✓ 0.9s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214939 entries, 0 to 214938
Data columns (total 11 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   UF                          214939 non-null  object
 1   Município                   214939 non-null  object
 2   Bairro                      214394 non-null  object
 3   Endereço                    214939 non-null  object
 4   CNPJ/CPF                    214934 non-null  object
 5   Agente Econômico            214603 non-null  object
 6   Segmento Fiscalizado        214923 non-null  object
 7   Data DF                     214939 non-null  object
 8   Número do Documento         214939 non-null  object
 9   Procedimento de Fiscalização 214939 non-null  object
 10  Resultado                   214939 non-null  object
dtypes: object(11)
memory usage: 18.0+ MB
```

O dataframe possui 11 colunas e 214.939 registros.

A coluna que interessa para a ligação entre todos os dataframes é a que contém a informação em comum entre eles e que no caso, para todos os dataframes que iremos trabalhar é a coluna que identifica o CNPJ.

No dataframe Fiscalizações, aqui definido como df_fiscalizacoes, a coluna é “CNPJ/CPF”. Podemos visualizar uma amostra de seu conteúdo com o código df_fiscalizacoes['CNPJ/CPF'].

```
df_fiscalizacoes['CNPJ/CPF']
✓ 0.1s
0         01575860000116
1         01575860000116
2         01575860000116
3         01575860000116
4         01575860000116
              ...
214934    24375589000170
214935    24375589000170
214936    24375589000170
214937    24375589000170
214938    24375589000170
Name: CNPJ/CPF, Length: 214939, dtype: object
```

Como os valores de CNPJ precisam estar formatados para que todos os dataframes possam ser unidos posteirormente em um único dataframe, vamos formatar a coluna com a máscara do CNPJ (99.999.999/9999-99) usando o código df_fiscalizacoes['CNPJ'] = df_fiscalizacoes['CNPJ/CPF'].str[0:2] + '.' + df_fiscalizacoes['CNPJ/CPF'].str[2:5] + '.' + df_fiscalizacoes['CNPJ/CPF'].str[5:8] + '/' + df_fiscalizacoes['CNPJ/CPF'].str[8:12] + '-' + df_fiscalizacoes['CNPJ/CPF'].str[12:14].

Este código usa o procedimento *str* do dataframe df_fiscalizações na coluna CNPJ/CPF, extraindo, conforme a posição passada como parâmetro, um pedaço da string e concatenando com outros pedaços, formando uma única string formatada.

Agora temos uma nova coluna CNPJ no dataframe com o valor formatado corretamente.

```
df_fiscalizacoes['CNPJ']
✓ 0.5s
0         01.575.860/0001-16
1         01.575.860/0001-16
2         01.575.860/0001-16
3         01.575.860/0001-16
4         01.575.860/0001-16
                 ...
214934    24.375.589/0001-70
214935    24.375.589/0001-70
214936    24.375.589/0001-70
214937    24.375.589/0001-70
214938    24.375.589/0001-70
Name: CNPJ, Length: 214939, dtype: object
```

Convertemos a coluna “Data DF” para o formato data utilizando a função do pandas *pd.to_datetime()*

```
df_fiscalizacoes['Data DF'] = pd.to_datetime(df_fiscalizacoes['Data DF'])
✓ 0.7s
```

Selecionamos apenas os registros que sejam anteriores ao ano de 2019.

```
df_fiscalizacoes = df_fiscalizacoes[df_fiscalizacoes['Data DF'] <= '2019-12-31']
✓ 0.1s
```

Verificamos novamente os dados do dataframe atualizado e constatamos que a seleção de registros menores ou iguais ao ano de 2019 funcionou.

```
df_fiscalizacoes.info()
✓ 0.1s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 137755 entries, 25 to 214938
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   UF                          137755 non-null  object
 1   Município                   137755 non-null  object
 2   Bairro                      137213 non-null  object
 3   Endereço                    137755 non-null  object
 4   CNPJ/CPF                    137750 non-null  object
 5   Agente Econômico            137459 non-null  object
 6   Segmento Fiscalizado        137755 non-null  object
 7   Data DF                     137755 non-null  datetime64[ns]
 8   Número do Documento         137755 non-null  object
 9   Procedimento de Fiscalização 137755 non-null  object
 10  Resultado                   137755 non-null  object
 11  CNPJ                        137750 non-null  object
dtypes: datetime64[ns](1), object(11)
memory usage: 13.7+ MB
```

Agora temos 137.755 registros.

Extraimos o mês da coluna Data DF com o código pd.DatetimeIndex(df_fiscalizacoes['Data DF']).month para uma nova coluna. Este código extrai o mês da coluna com o formato de data.

```
df_fiscalizacoes['Mes'] = pd.DatetimeIndex(df_fiscalizacoes['Data DF']).month
✓ 0.5s
```

Contamos a quantidade de registros agrupados por mês e incluímos em uma nova coluna com o nome de Ocorrencias Fiscalizacoes.

```
df_fiscalizacoes['Ocorrencias Fiscalizacoes'] = df_fiscalizacoes['Mes'].groupby(df_fiscalizacoes['CNPJ']).transform('count')
✓ 0.7s
```

Agora filtramos o dataframe selecionando somente as colunas CNPJ e Ocorrencias Fiscalizações.

```
df_fiscalizacoes = df_fiscalizacoes[['CNPJ','Ocorrencias Fiscalizacoes']]
✓ 0.5s
```

Nosso dataframe ficou com 2 colunas.

```
df_fiscalizacoes
✓ 0.1s
```

	CNPJ	Ocorrencias Fiscalizacoes
25	05.938.540/0001-34	4.0
26	05.938.540/0001-34	4.0
27	05.938.540/0001-34	4.0
28	05.938.540/0001-34	4.0
35	03.608.766/0004-94	2.0
...	...	...
214934	24.375.589/0001-70	10.0
214935	24.375.589/0001-70	10.0
214936	24.375.589/0001-70	10.0
214937	24.375.589/0001-70	10.0
214938	24.375.589/0001-70	10.0

137755 rows × 2 columns

Removemos todos os registros que possuem CNPJ duplicado com o código df_fiscalizacoes.drop_duplicates(subset =['CNPJ'], inplace = True). O parâmetro *subset* informa qual coluna queremos verificar os registros duplicados e o parâmetro *inplace* com o valor *True* indica que o resultado da operação será efetivado no próprio dataframe.

```
df_fiscalizacoes.drop_duplicates(subset =['CNPJ'], inplace = True)
✓ 0.7s
```

Com os registros duplicados removidos, o dataframe reduziu para 21.240.

df_fiscalizacoes

✓ 0.5s

	CNPJ	Ocorrencias Fiscalizacoes
25	05.938.540/0001-34	4.0
35	03.608.766/0004-94	2.0
45	03.608.766/0005-75	2.0
54	03.608.766/0006-56	2.0
65	03.987.364/0002-86	13.0
...	...	...
214889	02.280.133/0051-53	6.0
214895	16.960.698/0001-27	12.0
214907	22.533.231/0001-01	13.0
214920	09.429.072/0001-23	9.0
214929	24.375.589/0001-70	10.0

21240 rows × 2 columns

Finalizamos o tratamento do dataframe df_fiscalizacoes.

3.2 Dataframe Reclamações

Carregamos o dataframe df_reclamacoes utilizando o procedimento carregar_dados_de_reclamacoes().

df_reclamacoes = pd.read_csv("dados\\reclamacoes\\CRF2019 Dados Abertos.csv", encoding="UTF-8", sep=";", skipinitialspace=True, error_bad_lines=False, dtype=str)

```
# carrega os dados de reclamações no dataframe
def carregar_dados_de_reclamacoes():
    df_reclamacoes = pd.read_csv("dados\\reclamacoes\\CRF2019 Dados Abertos.csv", encoding="UTF-8", sep=";", skipinitialspace=True, error_bad_lines=False, dtype=str)
    return df_reclamacoes
```

Para o carregamento do conjunto de dados de reclamações, incluímos um novo parâmetro na função *read_csv* do pandas:

error_bad_lines: Em alguns casos, as linhas de registros não correspondem a quantidade original de colunas. Caso isto aconteça, este parâmetro com o valor *False* ignora as linhas com erro no momento de carregar o dataframe.

Após o carregamento, verificamos as colunas e quantidade de registros presentes no dataframe df_reclamacoes com o código df_reclamacoes.info()

Dataframe Reclamacoes

```
df_reclamacoes = carregar_dados_de_reclamacoes()
df_reclamacoes.info()
```

✓ 0.2s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17555 entries, 0 to 17554
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   AnoCalendario         17555 non-null  object
 1   DataArquivamento      17552 non-null  object
 2   DataAbertura          17552 non-null  object
 3   CodigoRegiao          17555 non-null  object
 4   Regiao                17555 non-null  object
 5   UF                    17555 non-null  object
 6   strRazaoSocial        17555 non-null  object
 7   strNomeFantasia       14034 non-null  object
 8   Tipo                  17555 non-null  object
 9   NumeroCNPJ            16028 non-null  object
 10  RadicalCNPJ           15974 non-null  object
 11  RazaoSocialRFB        14572 non-null  object
 12  NomeFantasiaRFB       7651 non-null   object
 13  CNAEPrincipal         14572 non-null  object
 14  DescCNAEPrincipal     14502 non-null  object
 15  Atendida              17555 non-null  object
 16  CodigoAssunto         17541 non-null  object
 17  DescricaoAssunto      17541 non-null  object
 18  CodigoProblema        45 non-null     object
```

Observamos que o dataframe possui 23 colunas e 17.555 registros.

Novamente precisamos tratar algumas colunas para extrair a quantidade de reclamações por CNPJ.

A coluna NumeroCNPJ está sem a quantidade total de caracteres, ou seja, sem ‘zeros ’ à esquerda.

```
df_reclamacoes['NumeroCNPJ']
✓ 0.4s
0        40432544000147
1                   191
2          5914650000166
3         13110340000117
4          5423963000111
               ...
17550     33254319000100
17551      3130170000189
17552       360305000104
17553       360305000104
17554     60746948000112
Name: NumeroCNPJ, Length: 17555, dtype: object
```

Para tratá-lo, primeiro vamos preencher os 0 ‘zeros’ à esquerda na coluna.

Para isso, aplicamos uma função que para cada registro da coluna NumeroCNPJ que será completado com zeros à esquerda até termos 14 caracteres.

```
df_reclamacoes['NumeroCNPJ'] = df_reclamacoes['NumeroCNPJ'].apply(lambda x: '{0:0>14}'.format(x))
✓ 0.3s
```

Após o tratamento, a coluna NumeroCNPJ apresenta ao valores completos com 14 posições.

```
df_reclamacoes['NumeroCNPJ']
✓ 0.1s
0        40432544000147
1        00000000000191
2        05914650000166
3        13110340000117
4        05423963000111
              ...
17550    33254319000100
17551    03130170000189
17552    00360305000104
17553    00360305000104
17554    60746948000112
Name: NumeroCNPJ, Length: 17555, dtype: object
```

Usamos o código para formatar a coluna com a máscara do cnpj criando uma nova coluna CNPJ.

df_reclamacoes['CNPJ'] = df_reclamacoes['NumeroCNPJ'].str[0:2] + '.' + df_reclamacoes['NumeroCNPJ'].str[2:5] + '.' + df_reclamacoes['NumeroCNPJ'].str[5:8] + '/' + df_reclamacoes['NumeroCNPJ'].str[8:12] + '-' + df_reclamacoes['NumeroCNPJ'].str[12:14]

A nova coluna CNPJ está com a formatação correta.

```
df_reclamacoes['CNPJ']
✓ 0.1s
0        40.432.544/0001-47
1        00.000.000/0001-91
2        05.914.650/0001-66
3        13.110.340/0001-17
4        05.423.963/0001-11
              ...
17550    33.254.319/0001-00
17551    03.130.170/0001-89
17552    00.360.305/0001-04
17553    00.360.305/0001-04
17554    60.746.948/0001-12
Name: CNPJ, Length: 17555, dtype: object
```

Agora criamos uma nova coluna Ocorrencias Reclamacoes com a contagem de números de CNPJ. Para realizarmos esta contagem, agrupamos a coluna cnpj com o procedimento *groupby* e depois contamos cada ocorrencia do grupo com o procedimento *transform*, passando como parâmetro 'count'.

```
df_reclamacoes['Ocorrencias Reclamacoes'] = df_reclamacoes['NumeroCNPJ'].groupby(df_reclamacoes['CNPJ']).transform('count')
✓ 0.4s
```

Eliminamos as demais colunas e deixamos somente as colunas CNPJ e Ocorrencias Reclamacoes. Podemos realizar esta operação atribuindo o dataframe df_reclamacoes somente com as colunas que nos interessam.

```
df_reclamacoes = df_reclamacoes[['CNPJ', 'Ocorrencias Reclamacoes']]
✓ 0.3s
```

Verificamos o dataframe df_reclamacoes com as novas colunas.

```
df_reclamacoes
✓ 0.1s
```

	CNPJ	Ocorrencias Reclamacoes
0	40.432.544/0001-47	370
1	00.000.000/0001-91	133
2	05.914.650/0001-66	609
3	13.110.340/0001-17	1
4	05.423.963/0001-11	203
...	...	...
17550	33.254.319/0001-00	39
17551	03.130.170/0001-89	33
17552	00.360.305/0001-04	217
17553	00.360.305/0001-04	217
17554	60.746.948/0001-12	167

17555 rows × 2 columns

Eliminamos os registros duplicados pelo CNPJ com o procedimento *drop_duplicates*.

```
df_reclamacoes.drop_duplicates(subset =['CNPJ'], inplace = True)
✓ 0.4s
```

Agora temos 4.043 registros

```
df_reclamacoes
✓ 0.1s
```

	CNPJ	Ocorrencias Reclamacoes
0	40.432.544/0001-47	370
1	00.000.000/0001-91	133
2	05.914.650/0001-66	609
3	13.110.340/0001-17	1
4	05.423.963/0001-11	203
...	...	...
17495	27.957.205/0001-16	1
17496	21.983.278/0001-04	1
17535	16.613.061/0001-64	1
17539	61.400.453/0001-08	1
17540	49.318.538/0001-38	1

4043 rows × 2 columns

3.3 Dataframe Multas

Carregamos o dataframe df_multas utilizando o procedimento carregar_dados_de_multas().

```
# carrega os dados de multas de 2016 a 2019 no dataframe
def carregar_dados_de_multas():
    # O parâmetros skiprows foi necessário para ignorar as 4 primeiras linhas dos arquivo que possuem informações desnecessárias para a análise
    df_multas = pd.read_csv("dados\multas\multas_2016_a_2019.csv", encoding="UTF-8", sep=";", skiprows=4, skipinitialspace=True)
    return df_multas
```

Como o conjunto de dados de multas tem as 4 primeiras linhas sem informação de dados, precisamos incluir um novo parâmetro no procedimento *read_csv* do pandas:

skiprows: Ignora as quantidades de linhas iniciais definidas nos parâmetros. No caso, ignoramos as 4 primeiras linhas.

```
df_multas = carregar_dados_de_multas()
✓ 0.9s
```

Após o carregamento podemos verificar as colunas e a quantidade de registros do dataframe df_multas com o código df_multas.info().

```
df_multas.info()
✓ 0.4s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16440 entries, 0 to 16439
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Status Processo           16440 non-null  object
 1   Superintendência          16359 non-null  object
 2   Número do Processo        16440 non-null  int64
 3   Auto de Infração          16440 non-null  object
 4   CNPJ/CPF                  16440 non-null  int64
 5   Razão Social              16440 non-null  object
 6   Data Transito Julgado     15122 non-null  object
 7   Vencimento                16440 non-null  object
 8   Valor da Multa Aplicada   16440 non-null  object
 9   Valor Total Pago (*) (**) 16440 non-null  object
dtypes: int64(2), object(8)
memory usage: 1.3+ MB
```

O dataframe possui 10 colunas e 16.440 registros de multas.

Verificamos a coluna CNPJ/CPF.

```
df_multas['CNPJ/CPF']
✓ 0.4s
0          9256390000130
1          2454508000195
2          8952971000143
3          4434150000407
4          7789937000137
                ...
16435      2913444000739
16436      2276842000104
16437     68316801000102
16438      3662454000116
16439      7610990000129
Name: CNPJ/CPF, Length: 16440, dtype: int64
```

Percebemos que a coluna precisa ser tratada corretamente, logo devemos inserir os zeros à esquerda e formatar com a máscara de cnpj.

```
df_multas['CNPJ/CPF'] = df_multas['CNPJ/CPF'].apply(lambda x: '{0:0>14}'.format(x))
✓ 0.5s
```

Em seguida precisamos formatar os valores em uma nova coluna CNPJ. Código para criar uma coluna CNPJ com o valor formatado:

df_multas['CNPJ'] = df_multas['CNPJ/CPF'].str[0:2] + '.' + df_multas['CNPJ/CPF'].str[2:5] + '.' + df_multas['CNPJ/CPF'].str[5:8] + '/' + df_multas['CNPJ/CPF'].str[8:12] + '-' + df_multas['CNPJ/CPF'].str[12:14]

Atualizamos o dataframe df_multas somente com os atributos que nos interessam (CNPJ e Valor da multa aplicada).

```
df_multas = df_multas[['CNPJ','Valor da Multa Aplicada']]
✓ 0.4s
```

Convertemos o tipo da coluna valor multa aplicada para float, trocando vírgula por ponto. Para esta operação, usamos o procedimento *replace* do dataframe, onde passamos como parâmetro o caractere que queremos procurar no texto e caractere que queremos que

o substitua, no caso específico seriam a *vírgula* ‘,’ e o *ponto* ’.’. Ao final convertemos o tipo da coluna com o procedimento *astype*().

```
df_multas['Valor da Multa Aplicada'] = df_multas['Valor da Multa Aplicada'].str.replace(',', '.').astype(float)
✓ 0.4s
```

Somamos o valor total da multa aplicada para uma nova coluna chamada Total multa. Novamente utilizamos o procedimento *transform*, porém agora com o parâmetro ‘*sum*’ para somar os valores de cada registro do grupo.

```
df_multas['Total multa'] = df_multas['Valor da Multa Aplicada'].groupby(df_multas['CNPJ']).transform('sum')
✓ 0.4s
```

Removemos a coluna Valor da Multa Aplicada, deixando somente as colunas CNPJ e Total multa.

```
df_multas = df_multas[['CNPJ','Total multa']]
✓ 0.3s
```

Removemos as linhas duplicadas com *drop_duplicates*().

```
df_multas = df_multas.drop_duplicates()
✓ 0.2s
```

Verificando o resultado do dataframe df_multas.

df_multas

✓ 0.1s

	CNPJ	Total multa
0	09.256.390/0001-30	33000.00
1	02.454.508/0001-95	25000.00
2	08.952.971/0001-43	33878.63
3	04.434.150/0004-07	40000.00
4	07.789.937/0001-37	10000.00
...	...	...
16430	14.345.959/0001-73	20000.00
16433	64.567.878/0001-96	20000.00
16435	02.913.444/0007-39	40000.00
16436	02.276.842/0001-04	20000.00
16438	03.662.454/0001-16	24000.00

11281 rows × 2 columns

Finalizamos o tratamento do dataframe df_multas com 2 colunas e 11.281 registros.

3.4 Dataframe PMQC

Carregamos o dataframe df_PMQC utilizando o procedimento carregar_dados_PMQC().

```
# carrega os vários arquivos PMQC que estão no diretório dados\PMQC e concatena seus valores no dataframe final
def carregar_dados_PMQC():
    caminho = r'dados\PMQC'
    todos_os_arquivos = glob.glob(caminho + "/*.csv")
    # inicializo um vetor para incluir cada dataframe do pandas que leu um arquivo .csv
    li = []

    for nome_arquivo in todos_os_arquivos:
        df = pd.read_csv(nome_arquivo, encoding="UTF-8", sep=";", usecols=['DataColeta','CnpjPosto','GrupoProduto','Uf','RegiaoPolitica','Ensaio','Resultado','Conforme'])
        li.append(df)

    df_PMQC = pd.concat(li)
    return df_PMQC
```

Como o conjunto de dados do PMQC está separado por mês, precisamos de um procedimento para ler de um diretório todos os arquivos .csv. Cada arquivo contido no diretório é carregado pelo procedimento *read_csv* do pandas e incluído em um vetor com o procedimento *append()*. Ao final da leitura dos arquivos, as estruturas e dados carregados no vetor são adicionados a um único dataframe com o procedimento *concat().*

Na leitura de cada arquivo .csv, usamos um parâmetro para filtrar as colunas que queremos carregar na estrutura do dataframe:

usecols: parâmetro com os nomes das colunas que se deseja carregar na estrutura. Se a coluna existir no arquivo .csv mas não estiver inserida neste parâmetro, a mesma não será carregada.

Carregamos o dataframe df_PMQC

```
df_PMQC = carregar_dados_PMQC()
✓ 2.2s
```

Após termos o dataframe carregado, vamos verificar suas informações.

```
df_PMQC.info()
✓ 0.2s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 758313 entries, 0 to 59928
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   DataColeta      758313 non-null  object
 1   GrupoProduto    758313 non-null  object
 2   CnpjPosto       758313 non-null  object
 3   Uf              758313 non-null  object
 4   RegiaoPolitica  758313 non-null  object
 5   Ensaio          758313 non-null  object
 6   Resultado       758313 non-null  object
 7   Conforme        758313 non-null  object
dtypes: object(8)
memory usage: 52.1+ MB
```

Temos 8 colunas e 758.313 registros. Para o tratamento deste dataframe, vamos conferir a coluna CnpjPosto com o código df_PMQC['CnpjPosto'].

```
df_PMQC['CnpjPosto']
✓ 0.3s
0         09.386.909/0001-02
1         09.386.909/0001-02
2         09.386.909/0001-02
3         09.386.909/0001-02
4         09.386.909/0001-02
                 ...
59924     62.420.575/0001-10
59925     62.420.575/0001-10
59926     62.420.575/0001-10
59927     62.420.575/0001-10
59928     62.420.575/0001-10
Name: CnpjPosto, Length: 758313, dtype: object
```

A coluna já se encontra completa e formatada com a máscara do cnpj.

Vamos renomear a coluna CnpjPosto para somente CNPJ. Para isso, utilizamos o procedimento *rename* onde definimos como parâmetro as colunas a serem renomeadas e seus novos nomes.

```
df_PMQC.rename(columns={'CnpjPosto':'CNPJ'}, inplace=True)
✓ 0.2s
```

Precisamos também alterar o tipo da coluna DataColeta para o formato de data. Novamente utilizamos o procedimento *to_datetime()* do pandas.

```
df_PMQC['DataColeta'] = pd.to_datetime(df_PMQC['DataColeta'])
✓ 0.1s
```

Criamos um nova coluna Mes a partir da data da coleta.

```
df_PMQC['Mes'] = pd.DatetimeIndex(df_PMQC['DataColeta']).month
✓ 0.1s
```

Removemos as linhas duplicadas, de forma que para cada CNPJ, por mês, tenha apenas um tipo de Grupo Produto, considerando a coluna conformidade. Isto é necessário devido aos vários registros da coluna Ensaio, que não é importante para a nossa análise.

```
df_PMQC.drop_duplicates(subset =['CNPJ','Mes','GrupoProduto'], inplace = True)
✓ 0.3s
```

Removemos os atributos do dataframe df_PMQC, deixando somente os atributos CNPJ, MES, GrupoProduto e Conforme.

```
df_PMQC = df_PMQC[['CNPJ','Mes','GrupoProduto','Conforme']]
✓ 0.4s
```

Verificamos o dataframe df_PMQC.

```
df_PMQC
✓ 0.1s
```

	CNPJ	Mes	GrupoProduto	Conforme
0	09.386.909/0001-02	4	Etanol	Sim
8	00.306.597/0071-00	4	Etanol	Sim
17	00.306.597/0086-96	4	Etanol	Sim
25	11.726.230/0001-59	4	Etanol	Sim
33	00.306.597/0002-88	4	Etanol	Sim
...	...	...	...	...
59879	43.101.310/0001-05	3	Óleo Diesel	Sim
59889	10.945.290/0001-08	3	Óleo Diesel	Sim
59899	61.389.383/0001-26	3	Óleo Diesel	Sim
59909	61.167.185/0001-18	3	Óleo Diesel	Sim
59919	62.420.575/0001-10	3	Óleo Diesel	Sim

84003 rows × 4 columns

Tratamos todos os dados do dataframe e agora temos 4 colunas e 84.003 registros.

3.5 Dataframe Serie

Carregamos o dataframe df_serie utilizando o procedimento carregar_dados_serie_historica_2019().

```
# carrega os dados de da série histórica do primeiro e segundo semestres de 2019 no dataframe
# Foi incluido o parâmetro , skipinitialspace=True para retirar os espaço em branco antes dos valores das colunas (Principalmente CNPJ)
def carregar_dados_serie_historica_2019():
    df_serie_primeiro_sem_2019 = pd.read_csv("dados\serie\sem_2019-1_CA.csv", encoding="UTF-8", sep=";", skipinitialspace=True)
    df_serie_segundo_sem_2019 = pd.read_csv("dados\serie\sem_2019-2_CA.csv", encoding="UTF-8", sep=";", skipinitialspace=True)
    return pd.concat([df_serie_primeiro_sem_2019,df_serie_segundo_sem_2019])
```

Para carregar os 2 conjuntos de dados da série histórica, criamos 2 dataframes. O primeiro recebe o 1º semestre de 2019 e o segundo recebe o 2º semestre de 2019. Ao final concatenamos os 2 dataframes utilizando a função concat do pandas.

```
df_serie = carregar_dados_serie_historica_2019()
```

Os dados são carregados no dataframe df_serie.

Verificamos o dataframe df_serie.

```
df_serie.info()
✓ 0.7s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1004029 entries, 0 to 507298
Data columns (total 16 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   Regiao - Sigla     1004029 non-null  object
 1   Estado - Sigla     1004029 non-null  object
 2   Municipio          1004029 non-null  object
 3   Revenda            1004029 non-null  object
 4   CNPJ da Revenda    1004029 non-null  object
 5   Nome da Rua        1004029 non-null  object
 6   Numero Rua         1003332 non-null  object
 7   Complemento        259350 non-null   object
 8   Bairro             1000240 non-null  object
 9   Cep                1004029 non-null  object
 10  Produto            1004029 non-null  object
 11  Data da Coleta     1004029 non-null  object
 12  Valor de Venda     1004029 non-null  object
 13  Valor de Compra    381869 non-null   object
 14  Unidade de Medida  1004029 non-null  object
 15  Bandeira           1004029 non-null  object
dtypes: object(16)
memory usage: 130.2+ MB
```

O dataframe possui 16 colunas e 1.004.029 registros. Como já visto no tratamento dos dataframes anteriores, vamos renomear 2 colunas, principalmente a coluna que indica o cnpj, neste caso, a coluna Cnpj da Revenda.

Renomeamos as colunas “Estado – Sigla” e “CNPJ da Revenda”.

```
df_serie.rename(columns={'Estado - Sigla':'Uf','CNPJ da Revenda':'CNPJ'}, inplace=True)
```

Atualizamos a coluna Data da Coleta para o tipo data.

```
df_serie['Data da Coleta'] = pd.to_datetime(df_serie['Data da Coleta'])
```

Criamos uma nova coluna “Mes” a partir da extração do mês da coluna “Data da Coleta”.

```
df_serie['Mes'] = pd.DatetimeIndex(df_serie['Data da Coleta']).month
```

Formatamos os tipos das colunas 'Valor de Venda' e 'Valor de Compra' para float, alterando vírgula por ponto.

```
df_serie['Valor de Venda'] = df_serie['Valor de Venda'].str.replace(',', '.').astype(float)
df_serie['Valor de Compra'] = df_serie['Valor de Compra'].str.replace(',', '.').astype(float)
✓ 0.9s
```

Criamos a coluna GrupoProduto, onde vamos definir dentro dos produtos do dataframe df_serie a qual grupo de produtos ele pertence.

Para começar, vamos importar a biblioteca numpy.

```
import numpy as np
```

Vamos criar a lista de condições. Como temos 3 tipos de grupo de produtos, serão 3 condições de acordo com os produtos. Não vamos considerar o produto GNV, pois o mesmo não possui classificação de conformidade no dataframe df_PMQC.

Verificamos os tipo únicos de produtos que existem na coluna “Produto” com o código df_serie['Produto'].unique()

```
df_serie['Produto'].unique()
✓ 0.1s
array(['GASOLINA', 'ETANOL', 'DIESEL S10', 'GNV', 'DIESEL'], dtype=object)
```

Observamos que temos os registros de 5 itens comuns na coluna “Produto”:

'GASOLINA', 'ETANOL', 'DIESEL S10', 'GNV', 'DIESEL'

Criamos uma lista de condições para separar cada produto, sendo que pelas opções únicas que temos, os itens DIESEL S10 e DIESEL são do mesmo grupo de produtos.

```
condicoes = [
    (df_serie['Produto'] == 'GASOLINA'),
    (df_serie['Produto'] == 'ETANOL'),
    (df_serie['Produto'] == 'DIESEL S10') | (df_serie['Produto'] == 'DIESEL'),
    (df_serie['Produto'] == 'GNV')
    ]
✓ 0.3s
```

Verificamos quais são os grupos de produtos existentes no dataframe df_PMQC para que possamos identificar corretamente a qual cada produto do dataframe df_serie pertence ao grupo do dataframe df_PMQC.

```
df_PMQC['GrupoProduto'].unique()
✓ 0.1s
array(['Etanol', 'Gasolina', 'Óleo Diesel'], dtype=object)
```

Temos os grupos 'Etanol', 'Gasolina', 'Óleo Diesel' para o dataframe df_PMQC. Não existe um grupo para o produto GNV.

Criamos uma lista com os valores de grupos de produtos em que cada produto da coluna “Produto” do dataframe df_serie pode ser encaixado.

```
valores = ['Gasolina', 'Etanol', 'Óleo Diesel', 'GNV']
```

Precisamos criar uma nova coluna chamada GrupoProduto no dataframe df_serie e para isto utilizaremos o procedimento *select* da biblioteca numpy.

O NumPy é uma biblioteca para a linguagem de programação Python, que suporta o processamento de grandes, multi-dimensionais arranjos e matrizes, juntamente com uma grande coleção de funções matemáticas de alto nível para operar sobre estas matrizes. Ele

contém um procedimento *select* que é usado para criar colunas sendo seus valores de dados definidos por condições.

Primeiro importamos a biblioteca numpy.

```
import numpy as np
```

Criamos a nova coluna GrupoProduto no dataframe df_serie e usamos o código np.select para colocar o valor correto de acordo com as condições definidas na lista condições:

Se o produto do dataframe df_serie for igual a 'GASOLINA', então seu GrupoProduto será 'Gasolina';

Se o produto do dataframe df_serie for igual a 'ETANOL', então seu GrupoProduto será 'Etanol';

Se o produto do dataframe df_serie for igual a 'DIESEL S10' ou 'DIESEL', então seu GrupoProduto será 'Óleo Diesel';

Se o produto do dataframe df_serie for igual a 'GNV', então seu GrupoProduto será 'GNV';

Executamos o código df_serie['GrupoProduto'] = np.select(condicoes, valores) para a criação da nova coluna GrupoProduto no dataframe df_serie.

```
df_serie['GrupoProduto'] = np.select(condicoes, valores)
```

Removemos colunas desnecessárias do dataframe df_serie, mantendo somente as colunas que nos interessam.

```
df_serie = df_serie[['CNPJ','Mes','GrupoProduto','Uf','Valor de Venda','Valor de Compra','Bandeira']]
```

Verificamos o dataframe df_serie atualizado.

df_serie

✓ 0.5s

	CNPJ	Mes	GrupoProduto	Uf	Valor de Venda	Valor de Compra	Bandeira
0	49.051.667/0001-02	3	Gasolina	SP	4.199	3.5766	PETROBRAS DISTRIBUIDORA S.A.
1	49.051.667/0001-02	3	Etanol	SP	2.899	2.3513	PETROBRAS DISTRIBUIDORA S.A.
2	49.051.667/0001-02	3	Óleo Diesel	SP	3.349	2.8841	PETROBRAS DISTRIBUIDORA S.A.
3	49.051.667/0001-02	3	GNV	SP	2.439	NaN	PETROBRAS DISTRIBUIDORA S.A.
4	88.587.589/0001-17	2	Gasolina	RS	4.399	3.8550	BRANCA
...	...	...	...	...	...	...	...
507294	28.039.566/0001-46	12	Óleo Diesel	SP	3.799	NaN	IPIRANGA
507295	28.470.480/0001-73	12	Gasolina	MG	4.939	4.3180	BRANCA
507296	28.470.480/0001-73	12	Etanol	MG	3.339	2.7697	BRANCA
507297	28.470.480/0001-73	12	Óleo Diesel	MG	3.799	3.4160	BRANCA
507298	28.470.480/0001-73	12	Óleo Diesel	MG	3.899	3.4700	BRANCA

1004029 rows × 7 columns

Temos ao total 7 colunas sendo que a coluna CNPJ já está formatada corretamente.

3.6 Juntando os 5 dataframes

Agora que finalizamos o tratamento dos dados do dataframe df_serie, vamos verificar como ficou cada um dos 5 dataframes para que possamos juntá-los em um único dataframe.

A figura 1 representa como ficou a estrutura de cada dataframe

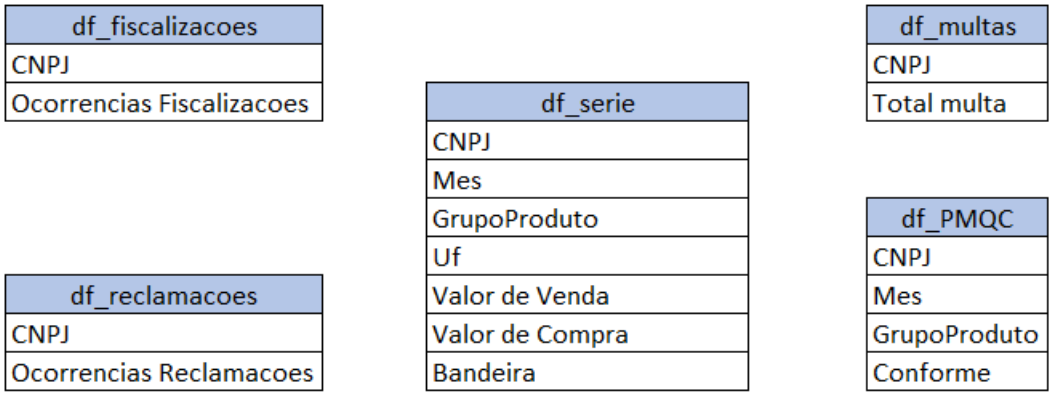


Figura 1 - Estrutura de cada dataframe

Para realizar a correta junção dos dataframes, vamos incluir os registros dos demais dataframes no dataframe df_serie, começando com o dataframe df_PMQC.

Para esta junção precisamos unir os 2 dataframes de acordo com o batimento de 3 colunas existentes em ambos os dataframes: CNPJ, Mes e GrupoProduto.

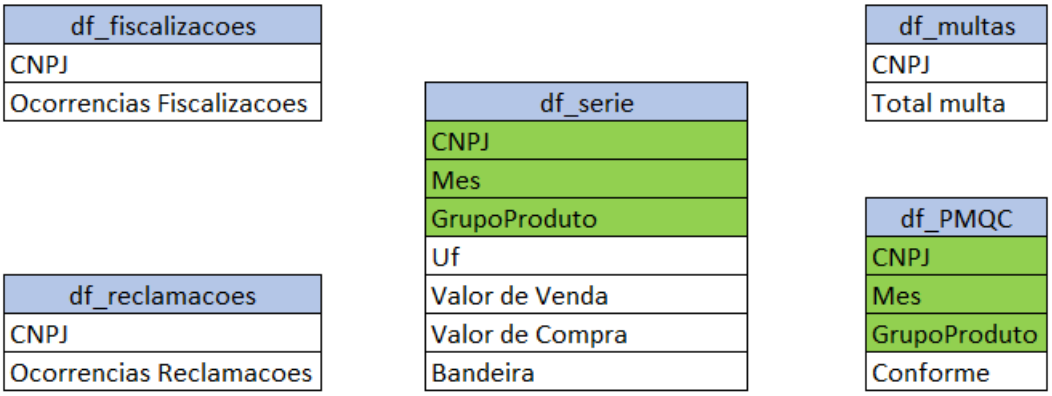


Figura 2 - Colunas selecionadas para realizar a junção dos dois dataframes

Na figura 2 marcamos exatamente as colunas existentes em cada dataframe para realizar a junção. O dataframe df_serie receberá os valores do dataframe df_PMQC que se correspondem nas 3 colunas (CNPJ, Mes e GrupoProduto).

Para isso, usamos o código *merge* do pandas.

```
df_serie = pd.merge(df_serie, df_PMQC, on=['CNPJ','Mes','GrupoProduto'], how='left' )
✓ 0.7s
```

Passamos como parâmetros os dataframes df_serie e df_PMQC, relacionamos os dataframes no parâmetro “on” com as colunas “CNPJ”, “Mes” e “GrupoProduto” e definimos para qual dataframe será preenchido com os valores no parâmetro “*how*” que neste caso está definido como ‘*left’*, ou seja, será o dataframe df_serie.

Após o uso do código merge, podemos visualizar como ficou o dataframe df_serie, usando o procedimento *info().*

```
df_serie.info()
✓ 0.4s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1004029 entries, 0 to 1004028
Data columns (total 8 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   CNPJ            1004029 non-null  object
 1   Mes             1004029 non-null  int64
 2   GrupoProduto    1004029 non-null  object
 3   Uf              1004029 non-null  object
 4   Valor de Venda  1004029 non-null  float64
 5   Valor de Compra 381869 non-null   float64
 6   Bandeira        1004029 non-null  object
 7   Conforme        67684 non-null    object
dtypes: float64(2), int64(1), object(5)
memory usage: 68.9+ MB
```

Continuamos com 1.004.029 registros, sendo que a nova coluna “Conforme” possui 67.684 registros que se relacionam com as colunas CNPJ, Mes e GrupoProduto respectivemente.

Agora vamos juntar os outros 3 dataframes restantes no dataframe df_serie.

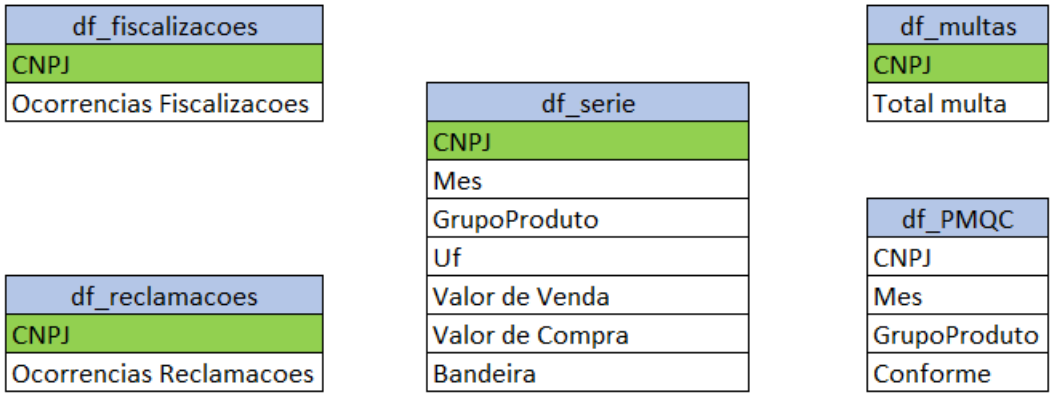


Figura 3 – Colunas selecionadas para realizar a junção dos 4 dataframes

A junção entre os dataframes será pela coluna CNPJ, comum entre todos os dataframes.

Para realizar esta operação, usamos novamente o código *merge* do pandas, unindo os dataframes df_serie com df_multas.

```
df_serie = pd.merge(df_serie, df_multas, on='CNPJ', how='left' )
✓ 0.3s
```

No parâmetro “on” realizamos a ligação entre os dataframes somente com a coluna ‘CNPJ’, sendo os registros do dataframe df_multas sendo inseridos no dataframe df_serie.

Verificamos a estrutura do dataframe df_serie.

```
df_serie.info()
✓ 0.3s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1004029 entries, 0 to 1004028
Data columns (total 9 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   CNPJ            1004029 non-null  object
 1   Mes             1004029 non-null  int64
 2   GrupoProduto    1004029 non-null  object
 3   Uf              1004029 non-null  object
 4   Valor de Venda  1004029 non-null  float64
 5   Valor de Compra 381869 non-null   float64
 6   Bandeira        1004029 non-null  object
 7   Conforme        67684 non-null    object
 8   Total multa     172717 non-null   float64
dtypes: float64(3), int64(1), object(5)
memory usage: 76.6+ MB
```

Continuamos com 1.004.029 registros e temos uma nova coluna “Total multa” com 172.717 registros que se relacionam com a coluna CNPJ do dataframe df_multas.

Realizamos o mesmo código de *merge* para o dataframe df_serie com o dataframe df_reclamacoes.

```
df_serie = pd.merge(df_serie, df_reclamacoes, on='CNPJ', how='left')
✓ 0.3s
```

Após o merge, o dataframe df_serie passa a ter uma nova coluna “Ocorrencias Reclamacoes” oriunda do df_reclamacoes.

Conferimos a nova coluna “Ocorrencias Reclamacoes” no dataframe df_serie.

```
df_serie.info()
✓ 0.3s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1004029 entries, 0 to 1004028
Data columns (total 10 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   CNPJ                   1004029 non-null  object
 1   Mes                    1004029 non-null  int64
 2   GrupoProduto           1004029 non-null  object
 3   Uf                     1004029 non-null  object
 4   Valor de Venda         1004029 non-null  float64
 5   Valor de Compra        381869 non-null   float64
 6   Bandeira               1004029 non-null  object
 7   Conforme               67684 non-null    object
 8   Total multa            172717 non-null   float64
 9   Ocorrencias Reclamacoes  420 non-null      float64
dtypes: float64(4), int64(1), object(5)
memory usage: 84.3+ MB
```

Ao total temos 420 reclamações vinculadas ao CNPJ no dataframe df_serie.

No final uniremos a coluna correspondente as fiscalizações realizadas entre os dataframes df_serie e df_fiscalizacoes.

```
df_serie = pd.merge(df_serie, df_fiscalizacoes, on='CNPJ', how='left')
✓ 0.4s
```

Conferimos a estrutura do dataframe df_serie após todas as junções com os outros 4 dataframes.

```
df_serie.info()
✓ 0.3s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1004029 entries, 0 to 1004028
Data columns (total 11 columns):
 #   Column                  Non-Null Count    Dtype
---  ------                  --------------    -----
 0   CNPJ                    1004029 non-null  object
 1   Mes                     1004029 non-null  int64
 2   GrupoProduto            1004029 non-null  object
 3   Uf                      1004029 non-null  object
 4   Valor de Venda          1004029 non-null  float64
 5   Valor de Compra         381869 non-null   float64
 6   Bandeira                1004029 non-null  object
 7   Conforme                67684 non-null    object
 8   Total multa             172717 non-null   float64
 9   Ocorrencias Reclamacoes 420 non-null      float64
 10  Ocorrencias Fiscalizacoes 401466 non-null float64
dtypes: float64(5), int64(1), object(5)
memory usage: 91.9+ MB
```

Temos a nova coluna “Ocorrencias Fiscalizacoes” com 401.466 registros.

Verificamos que o dataframe possui informações nulas em registros que atrapalham na interpretação e precisam ser tratadas. Para verificar se estas colunas estão com valor nulo, usamos o procedimento *isnull().sum()* do dataframe. Este procedimento busca os valores nulos nas colunas e as soma, exibindo a quantidade total de ocorrências para cada coluna.

```
df_serie.isnull().sum()
✓ 0.3s
CNPJ                          0
Mes                           0
GrupoProduto                  0
Uf                            0
Valor de Venda                0
Valor de Compra          622160
Bandeira                      0
Conforme                 936345
Total multa              831312
Ocorrencias Reclamacoes  1003609
Ocorrencias Fiscalizacoes 602563
dtype: int64
```

Temos muitos valores nulos para as coluna “Valor de compra”, “Conforme”, “Total multa”, “Ocorrencias Reclamacoes” e “Ocorrencias Fiscalizacoes”

Como nosso objetivo é analisar se determinado registro possui ou não indicação de multa, reclamação ou fiscalização, vamos tratar estas colunas com o procedimento *where* do numpy.

O procedimento “where” do numpy recebe 3 parâmetros: condição, o valor a ser atribuído a coluna caso a condição seja verdadeira, o valor a ser atribuído a coluna caso a condição seja falsa.

Com o código where do numpy, atualizamos todos os registros em uma nova coluna “Multado” em que a coluna Total multa for maior que zero ‘0’ com o valor “Sim” ou “Não” quando for NaN.

Desta forma definimos que os registros que não tem informação de multa ou informação zerada como não multado (Não). Os demais registros que possuem informação de multa maior que 0 serão definidos como multados (Sim).

```
df_serie['Multado'] = np.where(df_serie['Total multa'] > 0, 'Sim', 'Não')
✓ 0.1s
```

Criamos uma nova coluna “Reclamado” incluímos a informação se existe reclamação ou não de acordo com os valores da coluna “Ocorrencias Reclamacoes”. Deste modo podemos definir os registros que não possuem reclamação dos que realmente possuem.

```
df_serie['Reclamado'] = np.where(df_serie['Ocorrencias Reclamacoes'] > 0, 'Sim', 'Não')
✓ 0.1s
```

Por fim, criamos uma nova coluna 'Fiscalizado' e incluímos a informação se já foi fiscalizado ou não de acordo com a coluna “Ocorrencias Fiscalizacoes”. Esta coluna é o nosso foco mais importante, pois ela será no alvo para a análise de Machine Learning.

```
df_serie['Fiscalizado'] = np.where(df_serie['Ocorrencias Fiscalizacoes'] > 0, 'Sim', 'Não')
✓ 0.1s
```

Com as novas colunas tratadas, podemos eliminar as linhas que não agregam na análise pois possuem muitos registros NaN.

```
df_serie.isnull().sum()
✓ 0.4s
CNPJ                        0
Mes                         0
GrupoProduto                0
Uf                          0
Valor de Venda              0
Valor de Compra        622160
Bandeira                    0
Conforme               936345
Total multa            831312
Ocorrencias Reclamacoes  1003609
Ocorrencias Fiscalizacoes  602563
Multado                     0
Reclamado                   0
Fiscalizado                 0
dtype: int64
```

Removemos as linhas da coluna “Conforme” igual a NaN pois esta coluna é importante para a análise de conformidade e não deve ter valores nulos. Para remover estas linhas, utilizamos o código com o procedimento *dropna* do dataframe.

```
df_serie.dropna(subset = ["Conforme"], inplace=True)
✓ 0.5s
```

Para nossa análise, não podemos ter atributos vazios "NaN", logo precisamos eliminar as linhas da coluna “Valor da Compra” que estão nesta situação.

```
df_serie.dropna(subset = ["Valor de Compra"], inplace=True)
✓ 0.4s
```

Verificamos como ficou nosso dataframe após a limpeza destas linhas. Para esta verificação, usamos o código que apresenta uma amostra dos valores do dataframe.

```
df_serie
✓ 0.1s
```

Python

	CNPJ	Mes	GrupoProduto	Uf	Valor de Venda	Valor de Compra	Bandeira	Conforme	Total multa	Ocorrencias Reclamacoes	Ocorrencias Fiscalizacoes	Reclamado	Fiscalizado
17	00.001.388/0002-26	4	Etanol	DF	3.399	3.2640	RAIZEN	Sim	0.0	NaN	15.0	Não	Sim
19	00.001.388/0002-26	4	Gasolina	DF	4.399	4.0975	RAIZEN	Sim	0.0	NaN	15.0	Não	Sim
22	00.001.388/0002-26	4	Óleo Diesel	DF	3.699	3.5178	RAIZEN	Sim	0.0	NaN	15.0	Não	Sim
23	00.001.388/0002-26	4	Óleo Diesel	DF	4.029	3.5501	RAIZEN	Sim	0.0	NaN	15.0	Não	Sim
24	00.001.388/0002-26	5	Etanol	DF	3.399	2.8643	RAIZEN	Sim	0.0	NaN	15.0	Não	Sim
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1003227	97.531.250/0001-90	11	Gasolina	SP	3.899	3.6700	BRANCA	Sim	5000.0	NaN	7.0	Não	Sim
1003318	97.550.033/0001-47	8	Etanol	MS	3.599	2.9110	TAURUS	Sim	0.0	NaN	NaN	Não	Não
1003320	97.550.033/0001-47	8	Gasolina	MS	4.539	3.9350	TAURUS	Sim	0.0	NaN	NaN	Não	Não
1003323	97.550.033/0001-47	8	Óleo Diesel	MS	3.649	3.2000	TAURUS	Sim	0.0	NaN	NaN	Não	Não
1003324	97.550.033/0001-47	8	Óleo Diesel	MS	3.759	3.2500	TAURUS	Sim	0.0	NaN	NaN	Não	Não

26816 rows × 13 columns

Temos 26.816 linhas no nosso dataframe. Observamos que 2 colunas, “Ocorrencias Reclamacoes” e “Ocorrencias Fiscalizacoes”, possuem muitos registros nulos mas que já foram tratados respectivamente nas novas colunas “Reclamado” e “Fiscalizado”, logo podemos eliminar as colunas “Ocorrencias Reclamacoes” e “Ocorrencias Fiscalizacoes”.

Para eliminar as colunas “Ocorrencias Fiscalizacoes” e “Ocorrencias Reclamacoes” usamos o procedimento *drop* do dataframe.

```
df_serie.drop(['Ocorrencias Fiscalizacoes','Ocorrencias Reclamacoes'], axis=1, inplace=True)
✓ 0.3s
```

Verificamos que o nosso dataframe está totalmente limpo e sem valores nulos.

```
df_serie.isnull().sum()
```

```
CNPJ              0
Mes               0
GrupoProduto      0
Uf                0
Valor de Venda    0
Valor de Compra   0
Bandeira          0
Conforme          0
Multado           0
Reclamado         0
Fiscalizado       0
dtype: int64
```

```
df_serie
✓ 0.2s
```

	CNPJ	Mes	GrupoProduto	Uf	Valor de Venda	Valor de Compra	Bandeira	Conforme	Total multa	Reclamado	Fiscalizado
17	00.001.388/0002-26	4	Etanol	DF	3.399	3.2640	RAIZEN	Sim	0.0	Não	Sim
19	00.001.388/0002-26	4	Gasolina	DF	4.399	4.0975	RAIZEN	Sim	0.0	Não	Sim
22	00.001.388/0002-26	4	Óleo Diesel	DF	3.699	3.5178	RAIZEN	Sim	0.0	Não	Sim
23	00.001.388/0002-26	4	Óleo Diesel	DF	4.029	3.5501	RAIZEN	Sim	0.0	Não	Sim
24	00.001.388/0002-26	5	Etanol	DF	3.399	2.8643	RAIZEN	Sim	0.0	Não	Sim
...	...	...	...	...	...	...	...	...	...	...	...
1003227	97.531.250/0001-90	11	Gasolina	SP	3.899	3.6700	BRANCA	Sim	5000.0	Não	Sim
1003318	97.550.033/0001-47	8	Etanol	MS	3.599	2.9110	TAURUS	Sim	0.0	Não	Não
1003320	97.550.033/0001-47	8	Gasolina	MS	4.539	3.9350	TAURUS	Sim	0.0	Não	Não
1003323	97.550.033/0001-47	8	Óleo Diesel	MS	3.649	3.2000	TAURUS	Sim	0.0	Não	Não
1003324	97.550.033/0001-47	8	Óleo Diesel	MS	3.759	3.2500	TAURUS	Sim	0.0	Não	Não

26816 rows × 11 columns

Finalizamos o tratamento do nosso dataframe com 26.816 registros e podemos analisar e explorar os dados.

4. Análise e Exploração dos Dados

Para a exploração dos nossos dados, vamos utilizar nosso dataframe tratado df_serie e analisar alguns aspectos.

Valores de Compra e Venda de Combustíveis

Os valores de compra e venda de combustíveis estão incluídos no nosso conjunto de dados e conforme informação do Plano de Dados Abertos - PDA foram informados voluntariamente pelos postos de revenda de combustíveis.

Existe alguma correlação entre os valores de compra e venda existentes em nosso conjunto de dados?

A correlação é uma forma descritiva que mede se existe e qual o grau de dependência entre variáveis (o quanto uma variável interfere em outra), sendo que essa relação de dependência pode ou não ser causal. Essa medida de grau de relação é medida através de coeficientes. O coeficiente que usaremos será o de Pearson, também chamado de "coeficiente de correlação produto-momento" que mede o grau de correlação através do cálculo de direção positiva ou negativa. Este coeficiente assume apenas valores entre -1 e 1.

A análise de correlação vai retornar três possíveis cenários: correlação positiva; correlação negativa; e não há correlação.

Correlação positiva: quando duas variáveis que possuem correlação crescem ou decrescem juntas, ou seja, que possuem uma relação direta;

Correlação negativa: quando duas variáveis que possuem correlação, mas quando uma variável cresce a outra decresce, ou vice-versa;

Não ter correlação: quando o crescimento ou decrescimento de uma variável não tem efeito sobre outra variável.

Como interpretamos os valores do resultado?

0,9 a 1 (positivo ou negativo): correlação muito forte;

0,7 a 09 (positivo ou negativo): correlação forte;

0,5 a 0,7 (positivo ou negativo): correlação moderada;

0,3 a 0,5 (positivo ou negativo): correlação fraca;

0 a 0,3 (positivo ou negativo): não possui correlação.

Verificamos a correlação utilizando o procedimento "corr" do dataframe.

```
df_serie['Valor de Venda'].corr(df_serie['Valor de Compra'])
✓ 0.4s
0.9605723795351239
```

A correlação encontrada foi de 0.96, o que pode ser considerada uma correlação muito forte.

Avaliamos a correlação do Valor de venda com o total de multa.

```
df_serie['Valor de Venda'].corr(df_serie['Total multa'])
✓ 0.1s
0.0045183374779765185
```

A correlação foi muito baixa, 0,004, indicando que o os valores de multa não possuem qualquer correlação com os valores de venda.

Vamos verificar quais são os valores máximos e mínimos de venda, incluindo o estado.

Usaremos o procedimento "*nlargest*" do dataframe. Este procedimento retorna a quantidade de linhas definidas no primeiro parâmetro ordenadas de forma descendente, ou seja, do maior para o menor da coluna definida no segundo parâmetro.

```
df_serie_gasolina = df_serie[['Mes','GrupoProduto','Uf','Valor de Venda','Valor de Compra','Bandeira','Conforme','Multado','Reclamado','Fiscalizado']]
df_serie_gasolina = df_serie_gasolina[df_serie_gasolina['GrupoProduto'] == 'Gasolina']
df_serie_gasolina.nlargest(5,'Valor de Venda')
```

	Mes	GrupoProduto	Uf	Valor de Venda	Valor de Compra	Bandeira	Conforme	Multado	Reclamado	Fiscalizado
864452	11	Gasolina	RJ	5.499	4.5186	PETROBRAS DISTRIBUIDORA S.A.	Sim	Sim	Não	Sim
377210	5	Gasolina	PA	5.450	4.2383	IPIRANGA	Sim	Não	Não	Não
396199	5	Gasolina	PA	5.450	4.5045	IPIRANGA	Sim	Não	Não	Não
415378	5	Gasolina	PA	5.450	4.5045	IPIRANGA	Sim	Não	Não	Não
175951	7	Gasolina	RJ	5.399	4.2200	PETROBRAS DISTRIBUIDORA S.A.	Sim	Não	Não	Sim

Como resultado temos que o GrupoProduto Gasolina teve o valor mais caro vendido em 2019 no estado do Rio de Janeiro no valor de R$ 5.49. Interessante observar que neste caso o mesmo registro indica que o revendedor do maior valor de venda da gasolina estava com o valor "Sim" para a conforme e embora não tenha registro de reclamação (o campo Reclamado está igual a "Não"), o mesmo possui multas (o campo Multado está igual a "Sim") e já foi fiscalizado (o campo Fiscalizado está igual a "Sim").

Para verificar o valor de venda mais baixo, vamos utilizar o procedimento "*nsmallest*" que retorna os registros ordenados de forma ascendente conforme a coluna passada no parâmetro.

```
df_serie_gasolina.nsmallest(5,'Valor de Venda')
```

	Mes	GrupoProduto	Uf	Valor de Venda	Valor de Compra	Bandeira	Conforme	Multado	Reclamado	Fiscalizado
121459	2	Gasolina	SP	3.559	3.2500	BRANCA	Sim	Não	Não	Não
140492	2	Gasolina	SP	3.559	3.2500	BRANCA	Sim	Não	Não	Não
122519	11	Gasolina	SP	3.599	3.3500	BRANCA	Sim	Sim	Não	Sim
141009	2	Gasolina	SP	3.599	3.4781	BRANCA	Sim	Não	Não	Sim
109970	4	Gasolina	SP	3.649	3.3799	BRANCA	Sim	Não	Não	Não

Estes valores parecem ser bem distantes da média do preço de venda da Gasolina. Vamos verificar se existem valores discrepantes (conhecidos como “outliers”) no valor de venda.

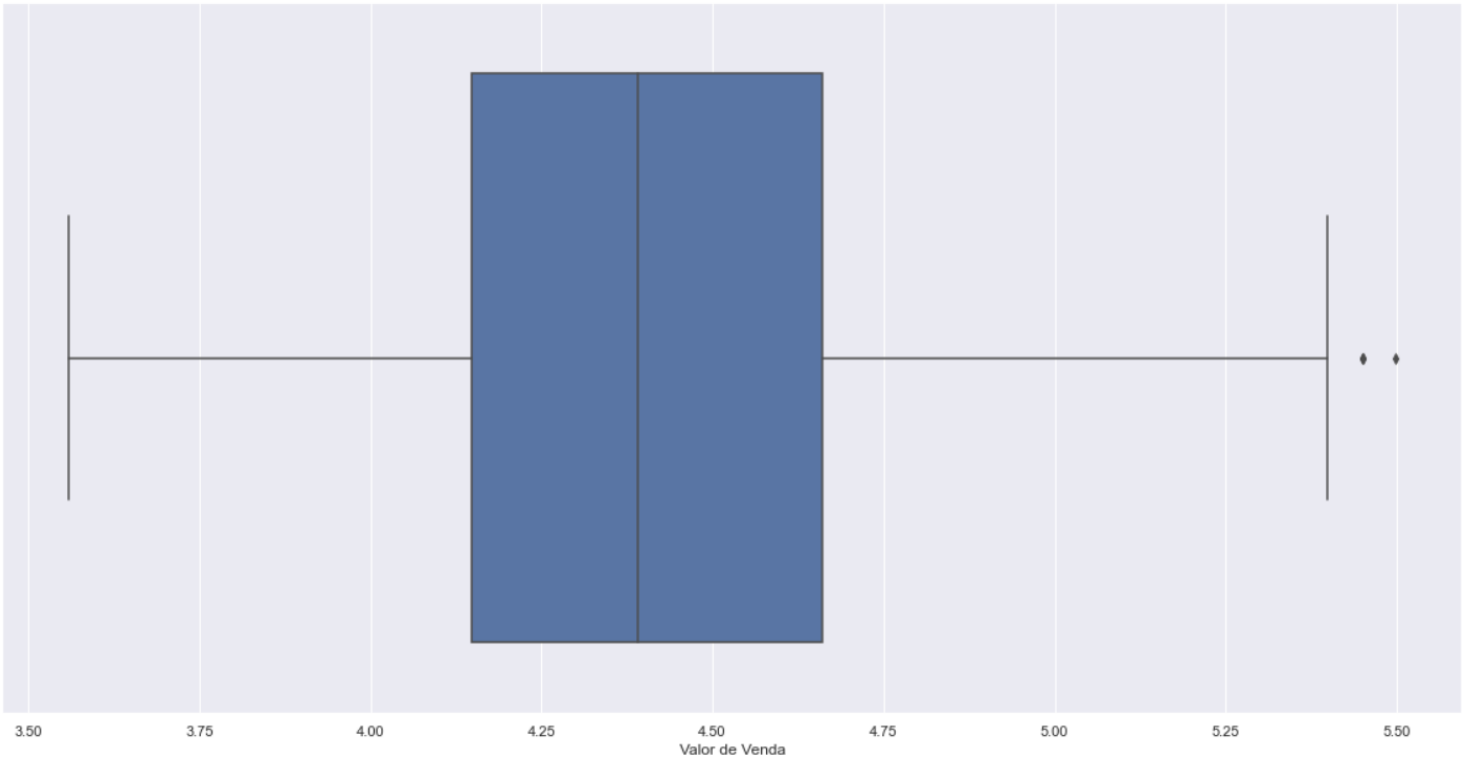
“Um outlier é uma observação que se diferencia tanto das demais observações que levanta suspeitas de que aquela observação foi gerada por um mecanismo distinto” (Hawkins, 1980), ou seja, os outliers são dados que se distanciam radicalmente de todos os valores que fogem da normalidade e que podem causar desequilíbrio nos resultados obtidos. Um conjunto de dados pode apresentar um ou vários outliers.

Usaremos o BoxPlot presente na biblioteca seaborn para representar graficamente se temos outliers.

```
import seaborn as sb
```

Com o código ax = sb.boxplot(x=df_serie_gasolina["Valor de Venda"]) criamos o gráfico do BoxPlot.

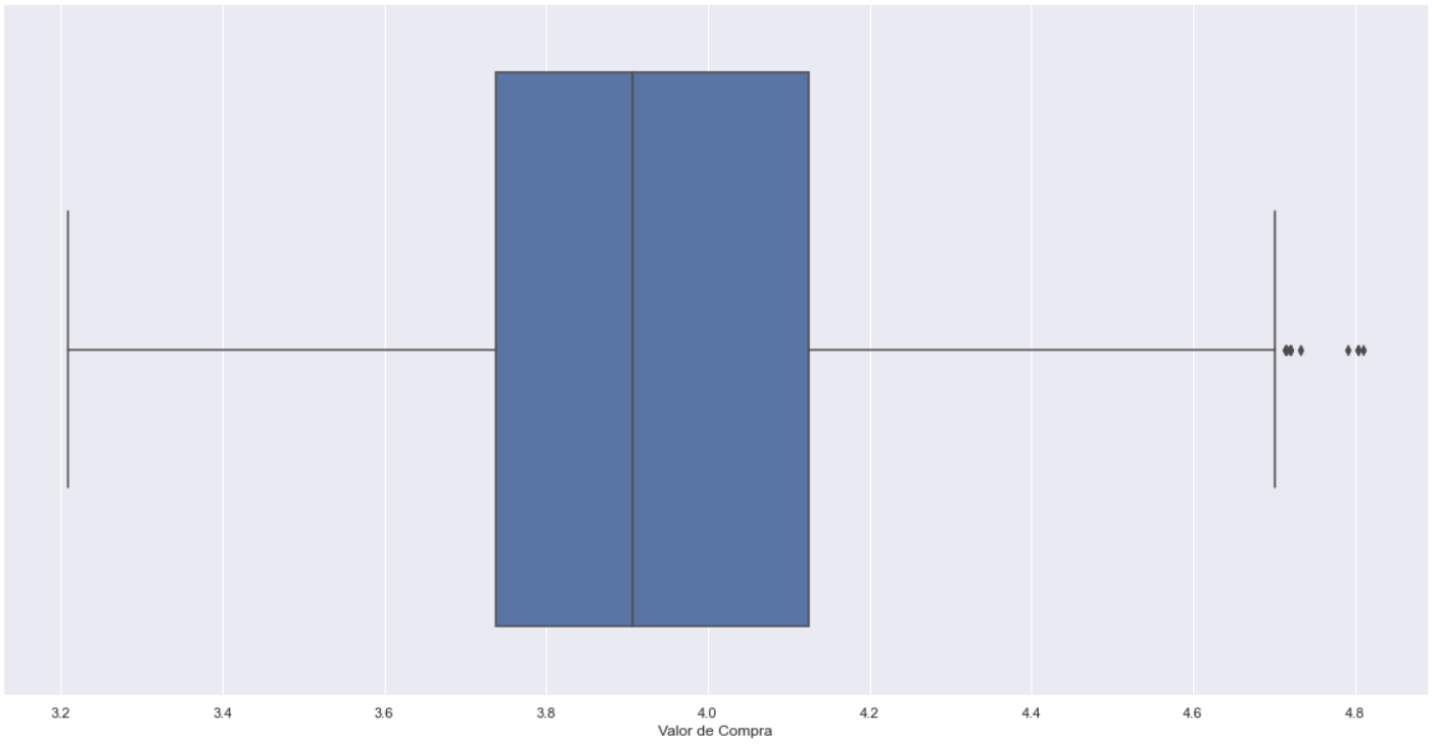
```
ax = sb.boxplot(x=df_serie_gasolina["Valor de Venda"])
✓ 0.5s
```



Observamos que para o valor de venda da gasolina temos poucos valores discrepantes. Estes valores discrepantes estão representados no gráfico acima como “pontos” próximos do valor de 5,50. São representações discrepantes dos maiores valores de venda da gasolina. Ao visualizarmos os valores mínimos de venda da gasolina, não encontramos “pontos” que representem valores discrepantes.

Como existe forte correlação entre os valores de venda e compra, vamos analisar os valores de compra da gasolina.

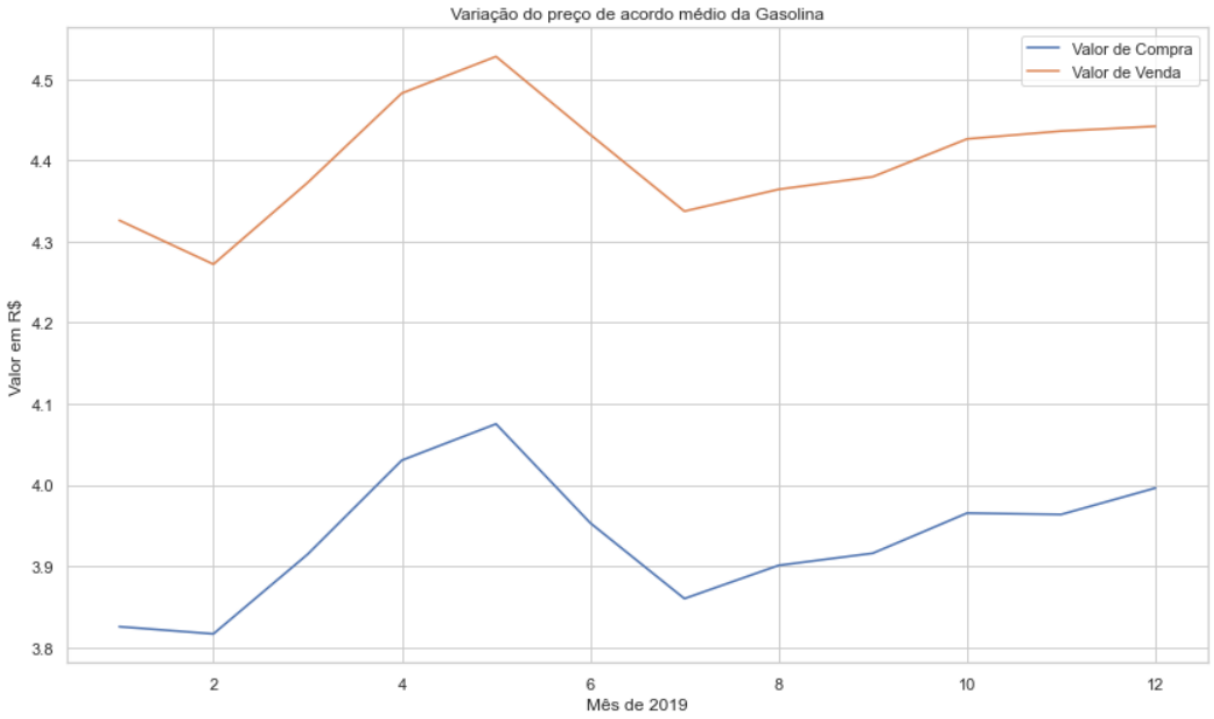
```
ax = sb.boxplot(x=df_serie_gasolina["Valor de Compra"])
✓ 2.5s
```



Vemos que os valores de compra da gasolina apresentam alguns poucos dos maiores valores discrepantes próximos do valor R$ 4,8 e nenhum valor menor de compra que seja discrepante.

Podemos analisar o acompanhamento desta correlação entre os meses de 2019:

```
grafico = df_serie_gasolina[['Valor de Compra','Valor de Venda','Mes']].groupby(df_serie_gasolina['Mes']).mean().plot(
                        title="Variação do preço de acordo médio da Gasolina",
                        x="Mes",
                        y=['Valor de Compra','Valor de Venda'],figsize=(14,8))
grafico.set_xlabel("Mês de 2019")
grafico.set_ylabel("Valor em R$")
plt.show()
```



Ainda que sejam poucos os valores discrepantes, vamos considerá-los no nosso conjunto de dados, pois o valor de venda de combustível muito alto ou muito baixo também é um motivo para a ação de fiscalização.

Analisando por estados (UF)

Vamos realizar nossa análise agrupando os valores por estado – coluna “Uf” do nosso conjunto de dados.

Para os valores de conformidade, podemos verificá-los pela coluna “Conforme”, usamos o procedimento *value_counts()* que retorna a quantidade.

```
df_serie['Conforme'].groupby(df_serie['Uf']).value_counts()
✓ 0.6s
```

Uf	Conforme	
AL	Sim	401
AP	Sim	147
BA	Sim	1716
CE	Sim	769
DF	Sim	406
ES	Sim	745
GO	Sim	65
MA	Sim	201
MG	Sim	2585
MS	Sim	607
PA	Sim	778
PB	Sim	402
PE	Sim	1307
	Não	5
PR	Sim	1261
RJ	Sim	3789
	Não	9
RN	Sim	338
RS	Sim	1121
SC	Sim	818
SE	Sim	268
SP	Sim	9078

Name: Conforme, dtype: int64

Observamos que temos muitos valores “Sim” para conformidade e pouquíssimos valores “Não”, presentes somente para os estados de Pernambuco (PE) e Rio de Janeiro (RJ).

Para os valores de reclamações temos o seguinte resultado:

```
df_serie['Reclamado'].groupby(df_serie['Uf']).value_counts()
✓ 0.6s
Uf  Reclamado
AL  Não          401
AP  Não          147
BA  Não         1716
CE  Não          769
DF  Não          406
ES  Não          745
GO  Não           65
MA  Não          201
MG  Não         2585
MS  Não          607
PA  Não          778
PB  Não          402
PE  Não         1312
PR  Não         1261
RJ  Não         3798
RN  Não          329
    Sim            9
RS  Não         1121
SC  Não          818
SE  Não          268
SP  Não         9078
Name: Reclamado, dtype: int64
```

Temos uma quantidade muito grande de valores “Não” e somente 9 valores “Sim” concentrados no estado do Rio Grande do Norte (RN).

Tanto para os valores de conformidade quanto para os valores de reclamações temos pouca variação dos dados. Como estes dados estão refletindo as informações disponibilizadas nos portais de dados abertos que temos no momento, demostrando que existem poucas reclamações específicas contra revendas de combustível nos PROCONS e como demonstrado nesta reportagem, disponível em https://portallubes.com.br/2019/01/percentual-de-conformidade-do-combustivel/ , o percentual de conformidade do combustível em postos de bandeira branca e bandeirados chegam a 96,7% e 98,6% respectivamente, vamos manter estas colunas.

Para a coluna “Multado”, temos valores mais variados:

```
df_serie['Multado'].groupby(df_serie['Uf']).value_counts()
✓ 0.4s
```

```
Uf  Multado
AL  Não         300
    Sim         101
AP  Não         109
    Sim          38
BA  Não        1108
    Sim         608
CE  Não         679
    Sim          90
DF  Sim         251
    Não         155
ES  Não         656
    Sim          89
GO  Não          42
    Sim          23
MA  Não         191
    Sim          10
MG  Não        2341
    Sim         244
MS  Não         470
    Sim         137
PA  Não         681
    Sim          97
PB  Não         326
    Sim          76

show more (open the raw output data in a text editor) ...

    Sim         124
```

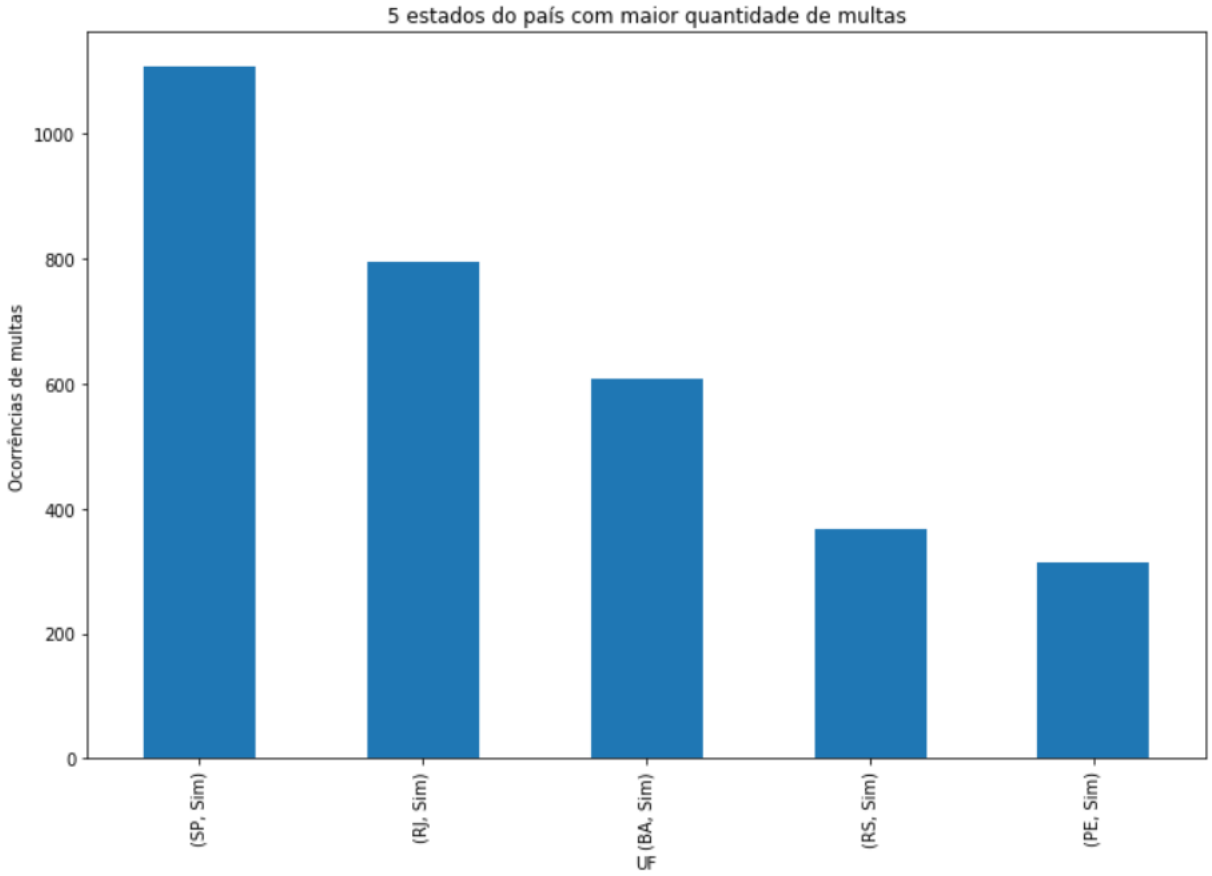
Podemos verificar os 5 estados que mais possuem multas utilizando um gráfico de barras. Para ordenar os valores, usamos o procedimento *sort_values* do dataframe com o parâmetro *ascending = False* para que os valores sejam ordenados do maior para o menor.

```
df_serie_multado_sim = df_serie[df_serie["Multado"] == 'Sim']
df_serie_multado_sim = df_serie_multado_sim['Multado'].groupby(df_serie_multado_sim['Uf']).value_counts().sort_values(ascending=False)
df_serie_multado_sim
```

✓ 0.1s

Uf	Multado	
SP	Sim	1108
RJ	Sim	795
BA	Sim	608
RS	Sim	366
PE	Sim	314
DF	Sim	251
MG	Sim	244
PR	Sim	160
MS	Sim	137
SC	Sim	124
AL	Sim	101
PA	Sim	97
CE	Sim	90
ES	Sim	89
PB	Sim	76
RN	Sim	39
AP	Sim	38
SE	Sim	32
GO	Sim	23

```
grafico = df_serie_multado_sim.sort_values(ascending=False).head(5).plot(kind='bar',
                                    figsize=(12,8),
                                    title="5 estados do país com maior quantidade de multas")
grafico.set_xlabel("UF")
grafico.set_ylabel("Ocorrências de multas")
plt.show()
```



Agora vamos verificar os estados com maior quantidade de ocorrências de fiscalização:

Realizamos a contagem para a coluna fiscalizado agrupado pela coluna “Uf” com o procedimento *value_counts()*. Observamos que existem quantidades de valores variados e distribuídos entre os estados.

```
df_serie['Fiscalizado'].groupby(df_serie['Uf']).value_counts()
✓ 0.5s
```

Uf	Fiscalizado	
AL	Sim	262
	Não	139
AP	Sim	116
	Não	31
BA	Sim	1350
	Não	366
CE	Não	422
	Sim	347
DF	Sim	406
ES	Sim	409
	Não	336
GO	Não	34
	Sim	31
MA	Não	161
	Sim	40
MG	Não	1955
	Sim	630
MS	Não	401
	Sim	206
PA	Sim	455
	Não	323
PB	Sim	255
	Não	147

Para verificar os 5 estados com maior ocorrência de fiscalizações, selecionamos somente os registros da coluna Fiscalizado igual a “Sim”. Com o resultado da seleção, agrupamos os valores pela coluna “Uf”, contamos com *values_count* utilizando o procedimento *sort_values* para ordenar de forma descrescente com o parâmetro *ascending = False* e utilizamos o procedimento *head()* com o parâmetro igual a 5.

O *head()* retorna as primeiras linhas do dataframe, com o parâmetro 5, ele retorna as 5 primeiras linhas.

```
df_serie_fiscalizado_sim = df_serie[df_serie["Fiscalizado"] == 'Sim']
df_serie_fiscalizado_sim = df_serie_fiscalizado_sim['Fiscalizado'].groupby(df_serie_fiscalizado_sim['Uf']).value_counts()
df_serie_fiscalizado_sim.sort_values(ascending=False).head(5)
```

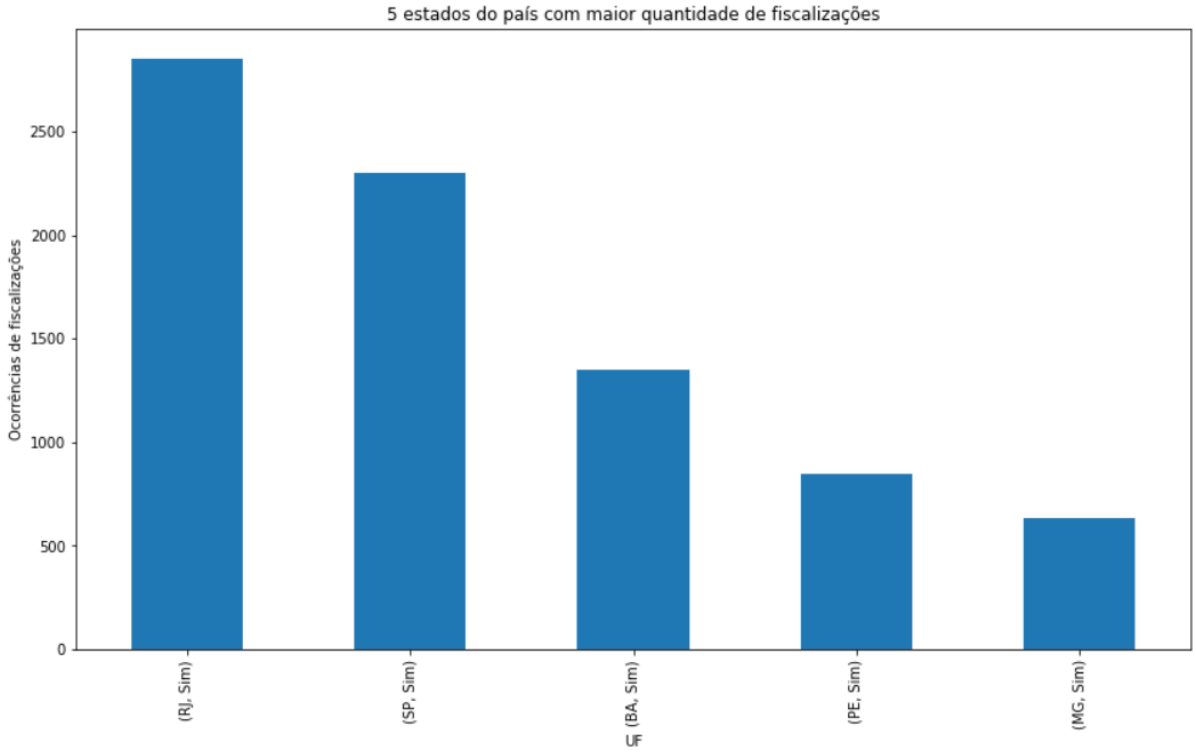
✓ 0.4s

```
Uf  Fiscalizado
RJ  Sim           2855
SP  Sim           2299
BA  Sim           1350
PE  Sim            844
MG  Sim            630
Name: Fiscalizado, dtype: int64
```

Em forma de gráfico de barras utilizando o procedimento *plot*:

```
grafico = df_serie_fiscalizado_sim.sort_values(ascending=False).head(5).plot(kind='bar',
                                   figsize=(14,8),
                                   title="5 estados do país com maior quantidade de fiscalizações")
grafico.set_xlabel("UF")
grafico.set_ylabel("Ocorrências de fiscalizações")
plt.show()
```

✓ 0.3s



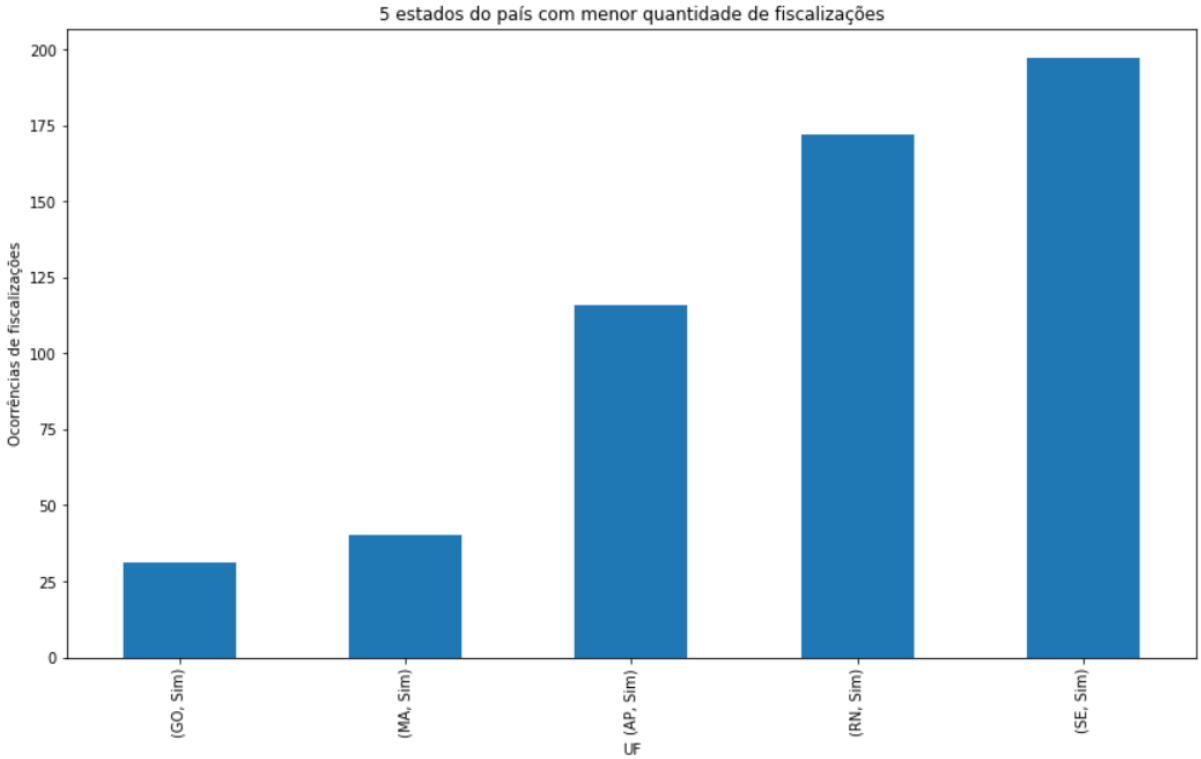
Conseguimos visualizar que os estados do Rio de Janeiro (RJ), São Paulo (SP), Bahia (BA), Pernambuco (PE) e Minas Gerais (MG) possuem as maiores ocorrências de fiscalizações.

E quais seriam os 5 estados com menor ocorrência de fiscalizações?

Invertemos o parâmetro de ordenação do procedimento *sort_values* para *ascending = True* e teremos a resposta.

```
grafico = df_serie_fiscalizado_sim.sort_values(ascending=True).head(5).plot(kind='bar',
                                   figsize=(14,8),
                                   title="5 estados do país com menor quantidade de fiscalizações")
grafico.set_xlabel("UF")
grafico.set_ylabel("Ocorrências de fiscalizações")
plt.show()
```

✓ 0.2s



Os 5 estados com menores ocorrências de fiscalização são Goiás (GO), Maranhão (MA), Amapá (AP), Rio Grande do Norte (RN) e Sergipe (SE).

Vamos verificar as medidas estatísticas do dataframe df_serie com o procedimento “describe(include=all)” do próprio dataframe.

O parâmetro “include=all” passado no procedimento indica que todas as colunas devem ser incluídas no resultado.

```
df_serie.describe(include='all')
✓ 0.1s
```

	CNPJ	Mes	GrupoProduto	Uf	Valor de Venda	Valor de Compra	Bandeira	Conforme	Multado	Reclamado	Fiscalizado
count	26816	26816.000000	26816	26816	26816.000000	26816.000000	26816	26816	26816	26816	26816
unique	3925	NaN	3	20	NaN	NaN	43	2	2	2	2
top	13.384.278/0001-51	NaN	Óleo Diesel	SP	NaN	NaN	BRANCA	Sim	Não	Não	Não
freq	60	NaN	10671	9078	NaN	NaN	8956	26803	22114	26807	14934
mean	NaN	6.434181	NaN	NaN	3.791009	3.362112	NaN	NaN	NaN	NaN	NaN
std	NaN	3.417889	NaN	NaN	0.588713	0.541402	NaN	NaN	NaN	NaN	NaN
min	NaN	1.000000	NaN	NaN	2.239000	1.950000	NaN	NaN	NaN	NaN	NaN
25%	NaN	4.000000	NaN	NaN	3.459000	3.047100	NaN	NaN	NaN	NaN	NaN
50%	NaN	6.000000	NaN	NaN	3.749000	3.321500	NaN	NaN	NaN	NaN	NaN
75%	NaN	9.000000	NaN	NaN	4.199000	3.769000	NaN	NaN	NaN	NaN	NaN
max	NaN	12.000000	NaN	NaN	5.499000	4.810800	NaN	NaN	NaN	NaN	NaN

Conseguimos observar alguns aspectos interessantes, como o GrupoProduto e o estado que aparece mais vezes são o Óleo Diesel e São Paulo (SP) respectivamente. Embora cada revenda de combustível possa representar uma bandeira, a bandeira Branca é a que também mais aparece.

Em relação as colunas “Conforme”, “Reclamado” e “Fiscalizado”, temos o valor *Sim* mais frequentes para o Conforme, e *Não* para Reclamado e Fiscalizado. A variação das informações da coluna Fiscalizado devem estar balanceadas para que tenhamos uma melhor avaliação dos modelos de Machine Learning.

Para finalizar, verificamos o balanceamento da nossa coluna alvo “Fiscalizado”.

Com o procedimento do dataframe *value_counts()*, os valores “Sim” e “Não” da coluna “Fiscalizado” estão bem balanceados:

```
df_serie['Fiscalizado'].value_counts()
✓ 0.4s
Não    14934
Sim    11882
Name: Fiscalizado, dtype: int64
```

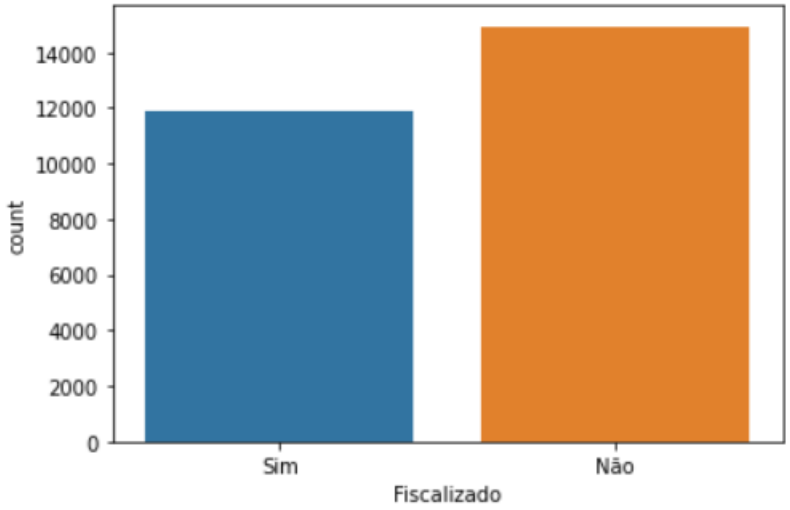
Em forma de gráfico de barras, utilizando o *seaborn countplot*.

O *countplot* do *seaborn* mostra de forma de barras os valores totalizados de colunas categóricas binárias.

```
ax = sb.countplot(x="Fiscalizado", data=df_serie)

coluna_alvo = df_serie.Fiscalizado.value_counts()
print('Não:', coluna_alvo[0])
print('Sim:', coluna_alvo[1])
print('Proporção:', round(coluna_alvo[1] / coluna_alvo[0], 2), ': 1')
```

✓ 0.2s

```
Não: 14934
Sim: 11882
Proporção: 0.8 : 1
```



Percebemos que existe um pequeno desbalanceamento nos valores da classe alvo “Fiscalizado”. Esta diferença está numa proporção de 0.8 : 1, considerando que temos uma quantidade total de 26.816 registros, com a classe “Não” sendo levemente majoritária à classe “Sim”. Como a diferença proporcional das classes não é muito grande e temos uma boa quantidade total de registros, decidimos avançar para as análises com o conjunto de dados original, sem realizar métodos de balanceamento de classes.

5. Criação de Modelos de Machine Learning

Neste trabalho utilizamos os modelos de aprendizado supervisionado de Machine Learning com a biblioteca “scikit-learn”, usada frequentemente para implementações de algoritmos de Machine Learning em Python. Como o objetivo é classificar situações em que se deve fiscalizar ou não, usaremos modelos classificadores mais usados em classificação binária (Sim e Não).

Antes de iniciar o uso dos modelos, precisamos realizar 2 procedimentos: Transformar os dados categóricos em numéricos do dataframe df_serie e depois separá-lo em conjunto de treinamento e outro de teste. Existem no scikit-learn as bibliotecas LabelEncoder e train_test_split que auxiliam neste passo.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

O dataframe df_serie tem as colunas Mes, Valor de Compra e Valor de Venda que são do tipo inteiro e float respectivamente e não precisam passar pelo processo de transformação. As demais colunas são consideradas categóricas e devem passar pelo processo de transformação para numérico antes de serem utilizadas nos modelos de Machine Learning.

Para a transformação dos dados, instanciamos uma variável “le” com o LabelEncoder. Foram criados também 2 vetores: “atributos” contendo os valores das colunas que serão usadas como atributos (features) no processo de classificação e o vetor “classe_alvo” que contém os valores da classe que queremos classificar, neste caso específico é a coluna “Fiscalizado”.

Para o vetor de atributos, foi realizada a transformação dos dados categóricos em numéricos aplicando o procedimento fit_transform do LabelEnconder.

```
le = LabelEncoder()

atributos = df_serie.iloc[:,0:10].values
atributos[:,0] = le.fit_transform(atributos[:,0])
atributos[:,2] = le.fit_transform(atributos[:,2])
atributos[:,3] = le.fit_transform(atributos[:,3])
atributos[:,6] = le.fit_transform(atributos[:,6])
atributos[:,7] = le.fit_transform(atributos[:,7])
atributos[:,8] = le.fit_transform(atributos[:,8])
atributos[:,9] = le.fit_transform(atributos[:,9])

classe_alvo = df_serie.iloc[:,10].values
```

Com as variáveis “atributos” e “classe_alvo” preenchidas, foi realizado o processo de separação com o procedimento “train_test_split”, na proporção de 80% para treinamento e 20% para teste.

```
X_treinamento, X_teste, Y_treinamento, Y_teste =  train_test_split(atributos, classe_alvo, test_size = 0.2, random_state = 1)
```

As 4 variáveis que foram criadas com esta separação para treinamento e teste: X_treinamento, X_teste, Y_treinamento e Y_teste, contêm respectivamente a separação de 80% dos dados que serão usados para treinamento dos modelos e 20% dos dados que serão usados para teste dos modelos de machine learning. Para o parâmetro random_state foi utilizado o valor igual a 1 para que sempre que o procedimento for usado tenha sempre a mesma saída.

Para cada modelo será analisado:

Matriz de confusão

A matriz de confusão é uma tabela que mostra as seguintes frequências de classificação para a classe do modelo:

Verdadeiro positivo (true positive — TP): ocorre quando no conjunto real, a classe que estamos buscando foi prevista corretamente.

Falso positivo (false positive — FP): ocorre quando no conjunto real, a classe que estamos buscando prever foi prevista incorretamente.

Falso negativo (false negative — FN): ocorre quando no conjunto real, a classe que não estamos buscando prever foi prevista incorretamente.

Verdadeiro negativo (true negative — TN): ocorre quando no conjunto real, a classe que não estamos buscando prever foi prevista corretamente.

A matriz de confusão pode ser representada desta forma:

		Valores preditos	
		Sim	Não
Valores reais	Sim	TP	FP
	Não	FN	TN

Figura 4 - Representação da matriz de confusão

Métricas

Precisão (precision)

Indica a proporção de números verdadeiros positivos que realmente estão corretos. É a razão da predição correta (TP) sobre a soma entre a predição correta (TP) e o número de exemplos classificados nesta classe, mas que pertencem a outras (FP).

$$\text{Precisão} = \frac{TP}{TP + FP}$$

Revocação (recall)

Indica a proporção de números positivos que foram identificados corretamente. É a razão entre verdadeiros positivos sobre a soma de verdadeiros positivos com negativos falsos.

$$\text{Revocação} = \frac{TP}{TP + FN}$$

Pontuação F1 (F1 score)

Indica a média harmônica ou ponderada entre precisão e revocação.

$$\text{Pontuação F1} = 2 * \frac{\text{precisão} * \text{revocação}}{\text{precisão} + \text{revocação}}$$

Acurácia (accuracy)

Indica a proporção de acertos das predições possíveis. É a razão entre o somatório das predições corretas (verdadeiros positivos com verdadeiros negativos) sobre o somatório das predições.

$$\text{Acurácia} = \frac{TP + TN}{TP + FP + TN + FN} = \frac{\text{predições corretas}}{\text{todas as predições}}$$

Validação cruzada (cross validation)

Para avaliar a generalização do modelo, é usada a técnica de validação cruzada. Esta técnica busca estimar o quão preciso é o modelo na prática, ou seja, o seu desempenho para um novo conjunto de dados.

O conceito da técnica de validação cruzada é o particionamento do conjunto de dados em 'n' subconjuntos mutuamente exclusivos do mesmo tamanho e, a partir daí, um subconjunto é utilizado para teste e os demais restantes são utilizados para estimação dos parâmetros. O processo é realizado 'n' vezes alternando o subconjunto de dados, assim a cada interação temos um conjunto diferente de dados para treino e teste.

Teste	Treino	Treino	Treino	Treino
Treino	Teste	Treino	Treino	Treino
Treino	Treino	Teste	Treino	Treino
Treino	Treino	Treino	Teste	Treino
Treino	Treino	Treino	Treino	Teste

Figura 5 - Conceito da técnica de validação cruzada

Como existem 'n' subconjuntos, o resultado da acurácia deve vai ser a média dos valores de cada um desses subconjuntos.

Modelos de Machine Learning

5.1 Modelo Naive Bayes

O Naive Bayes é um algoritmo probabilístico simples que utiliza dados de treino para formar um modelo classificador baseada na independência dos atributos (features).

Começamos importando o modelo da biblioteca scikit-learn.

```
from sklearn.naive_bayes import GaussianNB
```

Criamos o classificador do modelo Gaussian Naives Bayes como classificador_nb e o treinamos com a nossa base de treinamento usando seu procedimento “fit”

```
classificador_nb = GaussianNB()
classificador_nb.fit(X_treinamento, Y_treinamento)
```

O classificador_nb finaliza o treino com os dados de treinamento e agora pode realizar a predição com os dados de teste.

```
predicao_nb = classificador_nb.predict(X_teste)
```

Criamos a variável “predicao_nb” que recebe um vetor com o resultado da predição do classificador_nb utilizando como parâmetro os dados de testes X_teste (dados dos atributos).

Podemos visualizar uma amostra deste resultado na variável predicao_nb.

```
predicao_nb
✓ 0.3s
array(['Não', 'Não', 'Não', ..., 'Sim', 'Sim', 'Não'], dtype='<U3')
```

Com o resultado da predição, usamos uma matriz de confusão para avaliar o modelo.

Importamos os procedimentos *confusion_matrix*, *ConfusionMatrixDisplay* e *classification_report* da biblioteca scikit-learn.

```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import ConfusionMatrixDisplay
```

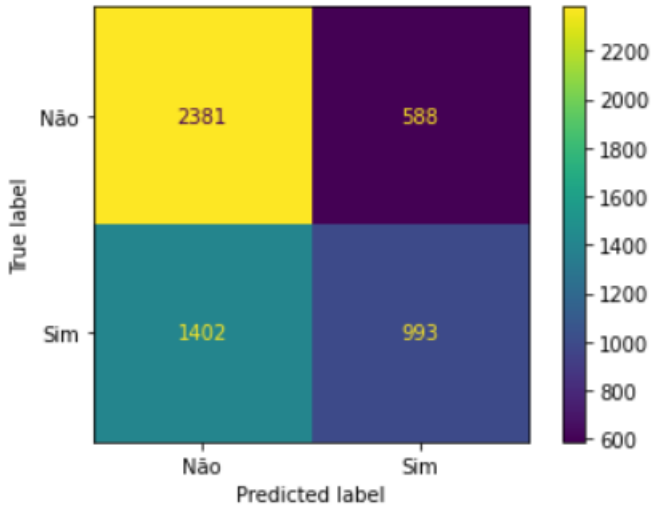
Com o procedimento *confusion_matrix* passamos os parâmetros Y_teste e predicao_nb para recebermos como resultado a matriz de confusão que avalia o modelo. Este resultado é armazenado na variável confusao_nb.

```
confusao_nb = confusion_matrix(Y_teste, predicao_nb)
✓ 0.4s
```

Podemos visualizar o resultado da matriz de confusão diretamente da variável confusao_nb ou com o gráfico do ConfusionMatrixDisplay.

```
print(confusao_nb)
✓ 0.3s
[[2381  588]
 [1402  993]]
```

```
cmd_nb = ConfusionMatrixDisplay(confusao_nb, display_labels=classificador_nb.classes_).plot()
✓ 0.2s
```



Observamos que a matriz de confusão possui mais valores na classe “Não” do que na classe “Sim”, o que pode ter sido provocado pelo leve desbalanceamento da classe alvo. O modelo tende a classificar melhor a classe “Não”. Seus valores de falso positivo e falso negativo também são bem altos, apresentando um risco da utilização do modelo na classificação.

Como vamos analisar os cálculos das métricas resultantes, usamos o *classification_report* que nos retorna um relatório detalhado e armazenamos na variável relatório_nb.

```
relatorio_nb = classification_report(Y_teste, predicao_nb)
✓ 0.2s
```

Visualizamos o relatório e temos acesso às métricas:

```
print(relatorio_nb)
✓ 0.3s
```

	precision	recall	f1-score	support
Não	0.63	0.80	0.71	2969
Sim	0.63	0.41	0.50	2395
accuracy			0.63	5364
macro avg	0.63	0.61	0.60	5364
weighted avg	0.63	0.63	0.61	5364

Destas informações, podemos destacar que a acurácia do modelo foi baixa e pela revocação entre as classes “Sim” e “Não” demonstra que o modelo possui um viés para o “Não” (0.80) que é praticamente o dobro da classe “Sim” (0.41).

Para realizamos a validação cruzada no nosso modelo precisamos importar o procedimento “*cross_val_score*” da biblioteca scikit-learn.

```
from sklearn.model_selection import cross_val_score
```

Utilizamos o procedimento cross_val_score, passando como parâmetros o classificador_nb, os vetores de atributos e classe_alvo, a métrica que queremos de retorno (scoring) que será a acurácia e a quantidade de subconjuntos que serão particionados os dados (cv) que será o valor 5.

```
validacao_nb = cross_val_score(classificador_nb, atributos, classe_alvo, scoring='accuracy', cv=5)
print(validacao_nb.mean())
✓ 0.4s
0.6343231991287835
```

Como resultado da validação cruzada retornamos a média dos valores de acurácia de cada subconjunto. O valor da média da acurácia retornada pela validação cruzada demonstra uma baixa generalização do modelo.

5.2 Modelo árvore de decisão (Decision Tree Classifier)

O modelo de árvore de decisão é um algoritmo de aprendizagem de máquina supervisionado que se baseia na ideia de divisão dos dados em grupos homogêneos podendo ser utilizado em um cenário de classificação.

Importamos o procedimento "*DecisionTreeClassifier*" e "*plot_tree*" da biblioteca scikit-learn.

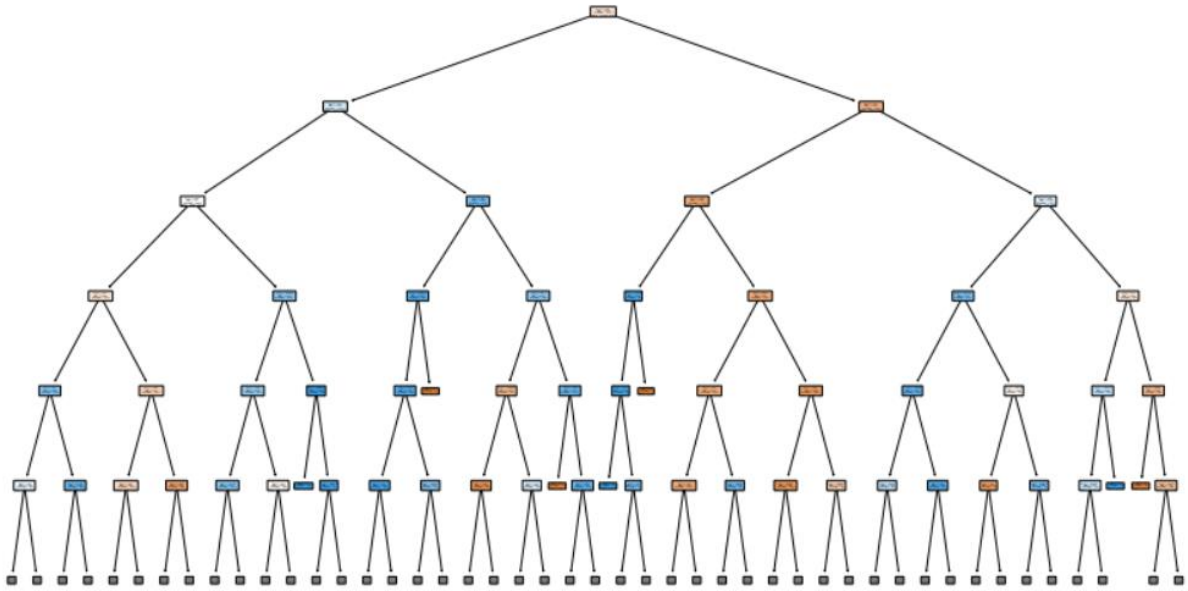
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
```

Após a importação da biblioteca, criamos uma variável "*classificador_arvore*" e realizamos o treinamento do modelo com os conjuntos de dados separados para treinamento (X_treinamento e Y_Treinamento).

```
classificador_arvore = DecisionTreeClassifier(random_state = 1)
classificador_arvore.fit(X_treinamento,Y_treinamento)
✓ 0.2s
```

Com o modelo treinado podemos visualizar graficamente a árvore gerada com o procedimento *plot_tree*.

```
plt.figure(figsize=(14,8))
plot_tree(classificador_arvore, max_depth=5, filled=True, rounded=True)
plt.show()
```



Com o modelo classificador da árvore, realizamos a predição usando o conjunto de teste.

```
predicao_arvore = classificador_arvore.predict(X_teste)
```

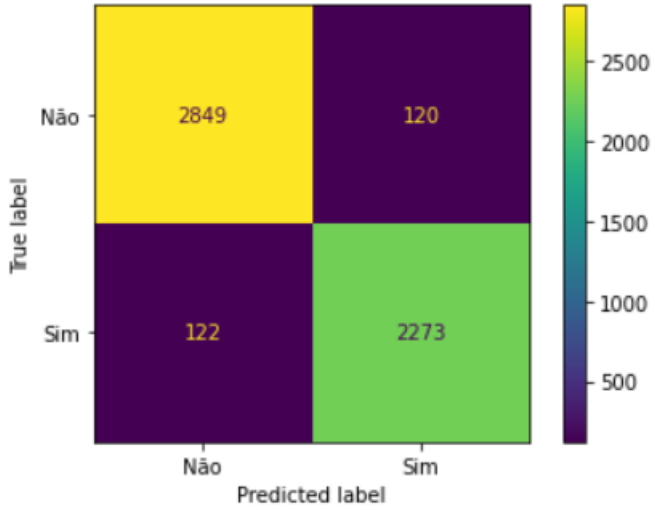
A variável predicao_arvore contém o resultado da predição e será usado para verificar a matriz de confusão.

```
confusao_arvore = confusion_matrix(Y_teste, predicao_arvore)
print(confusao_arvore)
cmd_arvore = ConfusionMatrixDisplay(confusao_arvore, display_labels=classificador_arvore.classes_).plot()
```

✓ 0.2s

```
[[2849  120]
 [ 122 2273]]
```

Usamos o resultado da matriz de confusão da variável confusao_arvore para visualizá-la em formato gráfico.



A matriz de confusão da árvore de decisão apresenta valores baixos com um bom equilibrio para os casos de falso negativo e falso positivo.

Utilizamos também o resultado da predição do modelo da arvore de decisão para gerar o relatório com as métricas do modelo.

```
relatorio_arvore = classification_report(Y_teste, predicao_arvore)
print(relatorio_arvore)
```
✓ 0.2s

```
              precision    recall  f1-score   support

         Não       0.96      0.96      0.96      2969
         Sim       0.95      0.95      0.95      2395

    accuracy                           0.95      5364
   macro avg       0.95      0.95      0.95      5364
weighted avg       0.95      0.95      0.95      5364
```

Na avaliação da acurácia, o modelo teve um resultado muito bom, inclusive nas métricas de precisão e revocação, sendo a classe “Não” (0.96) um pouco melhor do que a classe “Sim” (0.95).

Para avaliar a generalização, realizamos a validação cruzada no modelo e verificamos o cálculo da acurácia. Como usamos o parâmetro de subconjuntos igual a 5, nosso resultado da acurácia também será uma média do resultado de cada subconjunto.

```
validacao_arvore = cross_val_score(classificador_arvore, atributos, classe_alvo, scoring='accuracy', cv=5)
print(validacao_arvore.mean())
```
✓ 0.8s

```
0.8428932783427975
```

A média da acurácia de desempenho na validação cruzada do modelo é muito boa, apresentando o valor de 0.84.

5.3 Random Forest

Random Forest é um algoritmo de aprendizagem supervisionada. É uma combinação de árvores de decisão onde a idéia principal é que a combinação dos modelos de aprendizado aumenta o resultado geral. Como o modelo de árvore de decisão apresentou bons resultados em sua avaliação, vamos utilizar o modelo random forest para verificar se seu resultado será melhor.

Começamos importando o procedimento “*RandomForestClassifier*” da biblioteca scikit-learn e criar o classificador_rf para treinar o modelo a partir dos dados de treinamento utilizando o procedimento “*fit*”.

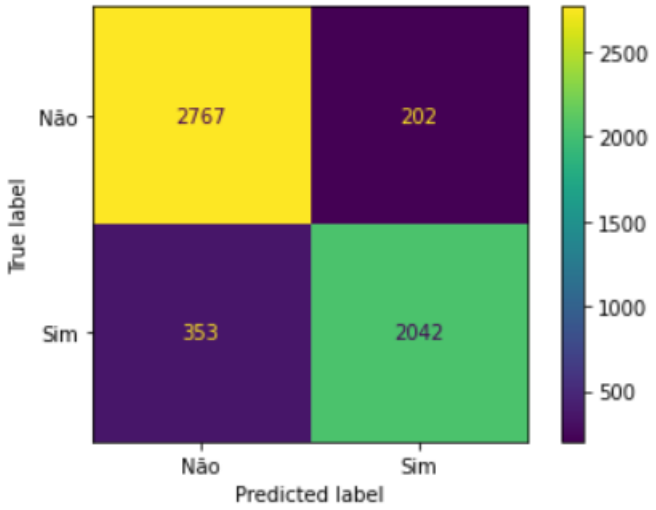
```
from sklearn.ensemble import RandomForestClassifier

classificador_rf = RandomForestClassifier()
classificador_rf.fit(X_treinamento, Y_treinamento)
✓ 5.3s
RandomForestClassifier()
```

Realizamos uma predição no classificador_rf com os dados de teste e visualizamos o vetor com o resultado.

```
predicao_rf = classificador_rf.predict(X_teste)
print(predicao_rf)
✓ 0.1s
['Não' 'Não' 'Não' ... 'Não' 'Não' 'Não']
```

Geramos a matriz de confusão a partir da predição dos dados de teste e visualizamos o resultado da matriz de forma gráfica.

```
confusao_rf = confusion_matrix(Y_teste, predicao_rf)
print(confusao_rf)
cmd_rf = ConfusionMatrixDisplay(confusao_rf, display_labels=classificador_rf.classes_).plot()
✓ 0.3s
[[2767  202]
 [ 353 2042]]
```



Observamos pela matriz de confusão que o modelo também apresenta uma boa classificação das classes “Não” e “Sim”, sendo seus valores de falso positivo e falso negativo ligeiramente piores que os da árvore de decisão.

Como resultado da predição, podemos visualizar suas métricas usando o procedimento “classification_report”.

```
relatorio_rf = classification_report(Y_teste, predicao_rf)
print(relatorio_rf)
```
✓ 0.2s

```
              precision    recall  f1-score   support

         Não       0.89      0.93      0.91      2969
         Sim       0.91      0.85      0.88      2395

    accuracy                           0.89      5364
   macro avg       0.90      0.89      0.89      5364
weighted avg       0.89      0.89      0.89      5364
```

Verificamos que a acurácia do modelo é boa, tendo uma precisão com valor semelhante, porém os valores de revocação para as classes “Não” e “Sim” apresentam uma diferença, sendo um pouco maior para a classe “Não” (0.93) comparada à classe “Sim” (0.85). Com estes valores, o modelo Random Forest tende a ter mais viés para a classe “Não” do que o modelo de árvore de decisão.

Para validar a generalização do modelo aplicamos a validação cruzada para obtermos a acurácia com o conjunto de dados original.

```
validacao_rf = cross_val_score(classificador_rf, atributos, classe_alvo, scoring='accuracy', cv=5)
print(validacao_rf.mean())
```
✓ 14.1s

```
0.8275664671751081
```

Observando que o resultado da média da acurácia do modelo Random Forest para a validação cruzada ficou abaixo do resultado do modelo árvore de decisão.

5.4 Modelo SVM (Support Vector Machine)

SVM é um algoritmo binário da classificação. Tomando como entrada um conjunto de dados, o algorítimo prediz para cada entrada dada, qual de duas possíveis classes a entrada faz parte. O SVM encontra uma linha de separação (hiperplano) entre dados de duas classes com o objetivo de maximizar a distância entre os pontos mais próximos em relação a cada uma das classes.

Começamos importando o procedimento “*svm*” da biblioteca scikit-learn.

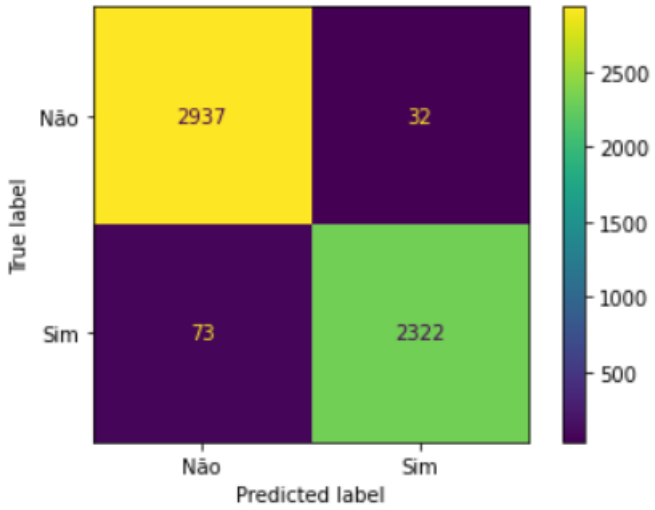
Com ele geramos nosso classificador_svm e o treinamos o conjunto de dados de treinamento com o procedimento “*fit*”.

```
from sklearn import svm

classificador_svm = svm.SVC(gamma='auto')
classificador_svm.fit(X_treinamento, Y_treinamento)
predicao_svm = classificador_svm.predict(X_teste)
confusao_svm = confusion_matrix(Y_teste, predicao_svm)
print(confusao_svm)
cmd_svm = ConfusionMatrixDisplay(confusao_svm, display_labels=classificador_svm.classes_).plot()
```

✓ 122.1s

```
[[2937   32]
 [  73 2322]]
```

Realizamos a predição com o procedimento "*predict*" do classificador_svm usando o conjunto de dados de teste e criamos a matriz de confusão.



A matriz de confusão do modelo svm apresentou bons valores para as classes "Não" e "Sim", sendo os valores de falso negativo e falso positivo mais baixos do que os do modelo de árvore de decisão.

Com a variável predicao_svm podemos gerar o relatório com as métricas do modelo svm. A variável relatorio_svm recebe o valor do procedimento "classification_report" com os parâmetros Y_teste e predicao_svm.

Os valores das métricas geradas pelo modelo de classificação svm podem ser visualizados ao imprimir a variável relatorio_svm.

```
relatorio_svm = classification_report(Y_teste, predicao_svm)
print(relatorio_svm)
```
✓ 0.2s

```
              precision    recall  f1-score   support

         Não       0.98      0.99      0.98      2969
         Sim       0.99      0.97      0.98      2395

    accuracy                           0.98      5364
   macro avg       0.98      0.98      0.98      5364
weighted avg       0.98      0.98      0.98      5364
```

Observamos que os valores das métricas para o modelo de classificação svm se apresentam bem altos. A acurácia do modelo tem o valor de 0.98, igualmente com os valores de precisão e revocação do modelo. A diferença de valor nas métricas precisão, revocação para as classes “Não” e “Sim” é muito pequena, sendo igual para a métrica F1 score, demonstrando que o modelo tende a classificar bem as classes.

Vamos avaliar a generalização do modelo com a validação cruzada com 5 subconjuntos de dados.

```
validacao_svm = cross_val_score(classificador_svm, atributos, classe_alvo, scoring='accuracy', cv=5)
print(validacao_svm.mean())
```
✓ 530.8s

```
0.8386050371653315
```

O resultado da média da acurácia na validação cruzada do modelo também é muito boa (0.83)

5.5 Modelo KNN (K-Nearest Neighbor)

O modelo KNN é um algoritmo que pode ser usado para classificação. Seu objetivo é determinar a qual grupo uma determinada amostra vai pertencer com base nas amostras vizinhas através de uma variável chamada de K, a qual é parte do nome do modelo sendo o principal parâmetro a ser selecionado.

Começamos importando o procedimento “*KNeighborsClassifier*” da biblioteca scikit-learn. Criamos a variável classificador_knn com o procedimento KNeighborsClassifier e o treinamos com o conjunto de treinamento “X_treinamento” e “Y_treinamento”.

```
from sklearn.neighbors import KNeighborsClassifier

classificador_knn = KNeighborsClassifier()
classificador_knn.fit(X_treinamento, Y_treinamento)
predicao_knn = classificador_knn.predict(X_teste)
confusao_knn = confusion_matrix(Y_teste, predicao_knn)
print(confusao_knn)
cmd_knn = ConfusionMatrixDisplay(confusao_knn, display_labels=classificador_knn.classes_).plot()
```

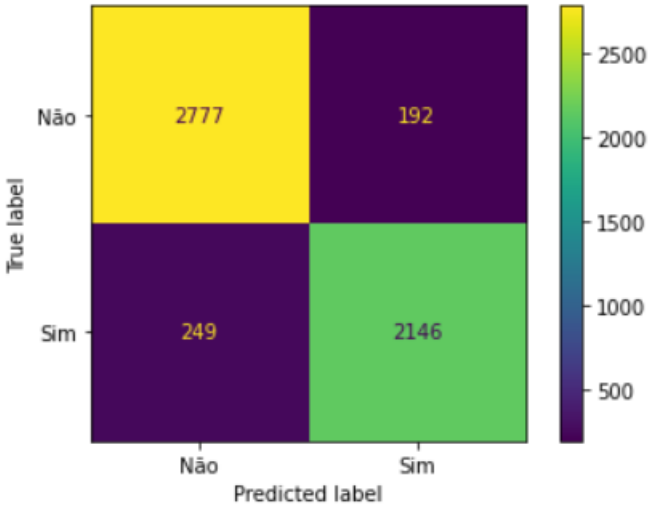
✓ 2.9s

```
[[2777  192]
 [ 249 2146]]
```

Geramos a variável predicao_knn para receber a predição do classificador_knn usando como parâmetro o conjunto de dados de teste “X_teste”.

Com o resultado da predição podemos calcular a matriz de confusão do modelo. Como não passamos o parâmetro k no momento de criar o classificador_knn, ele foi considerado com o valor de k igual a 5.

Visualizamos o resultado da matriz de confusão com o procedimento “*ConfusionMatrizDisplay*”.



A matriz de confusão do modelo apresenta bons valores para as classes “Não” e “Sim”. Os valores para classificação de falso positivo e falso negativo são baixos.

Geramos o relatório com as métricas da predição usando o procedimento “*classification_reports*”.

```
relatorio_knn = classification_report(Y_teste, predicao_knn)
print(relatorio_knn)
```

✓ 0.2s

```
              precision    recall  f1-score   support

         Não       0.92      0.94      0.93      2969
         Sim       0.92      0.90      0.91      2395

    accuracy                           0.92      5364
   macro avg       0.92      0.92      0.92      5364
weighted avg       0.92      0.92      0.92      5364
```

Verificamos que as métricas do modelo classificador knn tiveram bons resultados, sendo a acurácia e a precisão para as classes “Não” e “Sim” do modelo com o valor de 0.92. Existe uma pequena diferença nos valores de revocação e F1 score para as classes, sendo a classe “Não” um pouco maior do que a classe “Sim”.

Vamos agora verificar a média da acurácia para a generalização do modelo knn com a validação cruzada usando o procedimento “cross_val_score.”

```
validacao_knn = cross_val_score(classificador_knn, atributos, classe_alvo, scoring='accuracy', cv=5)
print(validacao_knn.mean())
```

✓ 1.4s

```
0.8257399382044758
```

A acurácia da generalização do modelo na validação cruzada foi de 0.82

Mas e se o valor de *k* variasse para a validação do modelo?

Vamos variar o valor de k de 1 a 10 e verificar seus resultados de validação cruzada montando uma estrutura de repetição.

```
for k in range(1, 11):
    classificador_knn = KNeighborsClassifier(n_neighbors = k)
    classificador_knn.fit(X_treinamento, Y_treinamento)
    validacao_knn = cross_val_score(classificador_knn, atributos, classe_alvo, scoring='accuracy', cv=5)
    print('Validação cruzada para a acurácia do modelo knn sendo k = ' + str(k) + ' -> ' + str(validacao_knn.mean()))
```

✓ 15.2s

```
Validação cruzada para a acurácia do modelo knn sendo k = 1 -> 0.8503516860839655
Validação cruzada para a acurácia do modelo knn sendo k = 2 -> 0.8483380268842928
Validação cruzada para a acurácia do modelo knn sendo k = 3 -> 0.8452429946787883
Validação cruzada para a acurácia do modelo knn sendo k = 4 -> 0.8341673128902805
Validação cruzada para a acurácia do modelo knn sendo k = 5 -> 0.8257399382044758
Validação cruzada para a acurácia do modelo knn sendo k = 6 -> 0.8075790384665389
Validação cruzada para a acurácia do modelo knn sendo k = 7 -> 0.7931846455878884
Validação cruzada para a acurácia do modelo knn sendo k = 8 -> 0.7818851736766808
Validação cruzada para a acurácia do modelo knn sendo k = 9 -> 0.7711083885595548
Validação cruzada para a acurácia do modelo knn sendo k = 10 -> 0.7632772637884097
```

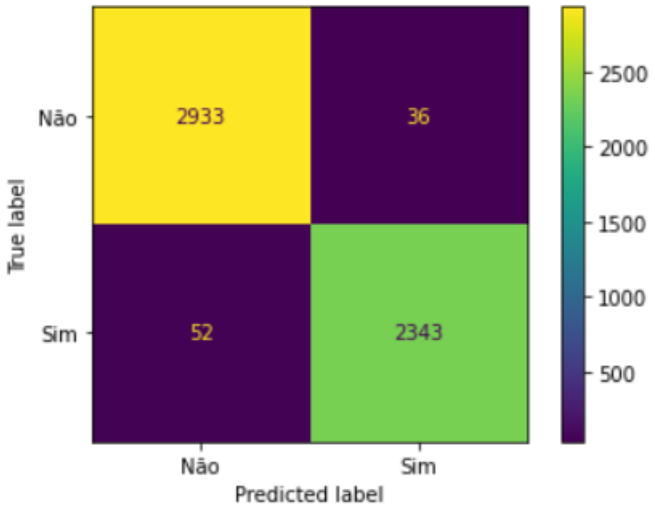
O modelo knn apresenta o melhor resultado de generalização da acurácia na validação cruzada com o valor de k igual a 1.

Como na validação cruzada com k igual a 1 apresentou o melhor desempenho, realizamos um novo treinamento, agora com o valor de k sendo igual a 1 no procedimento "*KNeighborsClassifier*".

```
classificador_knn_1 = KNeighborsClassifier(n_neighbors = 1)
classificador_knn_1.fit(X_treinamento, Y_treinamento)
predicao_knn_1 = classificador_knn_1.predict(X_teste)
confusao_knn_1 = confusion_matrix(Y_teste, predicao_knn_1)
print(confusao_knn_1)
cmd_knn_1 = ConfusionMatrixDisplay(confusao_knn_1, display_labels=classificador_knn_1.classes_).plot()
```

✓ 0.5s

```
[[2933   36]
 [  52 2343]]
```




```
relatorio_knn_1 = classification_report(Y_teste, predicao_knn_1)
print(relatorio_knn_1)
```

✓ 0.2s

```
              precision    recall  f1-score   support

         Não       0.98      0.99      0.99      2969
         Sim       0.98      0.98      0.98      2395

    accuracy                           0.98      5364
   macro avg       0.98      0.98      0.98      5364
weighted avg       0.98      0.98      0.98      5364
```

Visualizamos uma melhora significativa quando ajustamos o valor de k para 1.

Observamos pela matriz de confusão que o modelo com o k = 1 continua com uma boa classificação para as classes Não e Sim, sendo os valores de falso positivo e falso negativo menores e melhores, além de suas métricas de precisão, revocação e F1 score que também tiveram melhora em relação ao modelo treinado com k igual a 5.

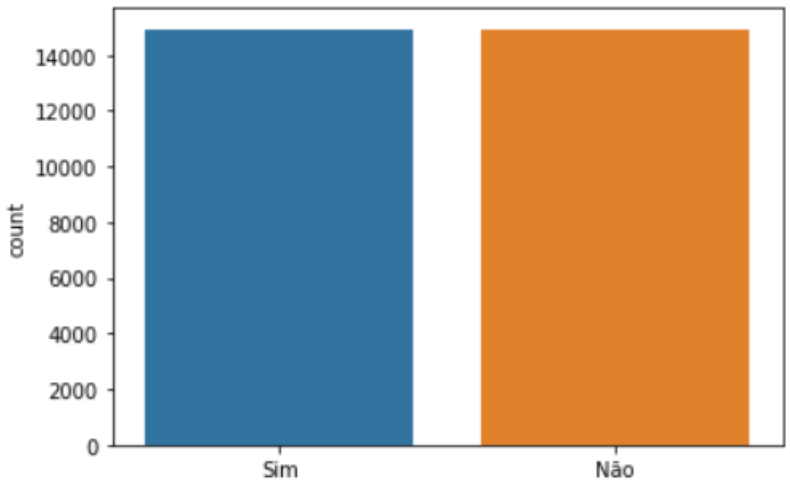
5.6 Validando o leve desbalanceamento

Mas será que os modelos Naive Bayes e Random Forest, que tiveram o pior desempenho entre os demais modelos foram influenciados pelo leve desbalanceamento entre as classes “Sim” e “Não”?

Vamos eliminar a diferença que causa o desbalanceamento aplicando um método de Oversampling, que consiste em criar sinteticamente novas observações da classe minoritária com o objetivo de igualar a proporção das categorias. Selecionamos o método SMOTE por ser conhecido como o mais sofisticado por consistir em criar observações intermediárias entre dados parecidos.

```
from imblearn.over_sampling import SMOTE

atributos_re = atributos
classe_alvo_re = classe_alvo

smote = SMOTE(random_state=1)
atributos_re, classe_alvo_re = smote.fit_resample(atributos_re, classe_alvo_re)
sb.countplot(x=classe_alvo_re)

X_treinamento_re, X_teste_re, Y_treinamento_re, Y_teste_re =  train_test_split(atributos_re, classe_alvo_re, test_size = 0.2, random_state = 1)
```

Realizamos o oversampling com SMOTE e visualizamos a classe Fiscalizado com *countplot()*

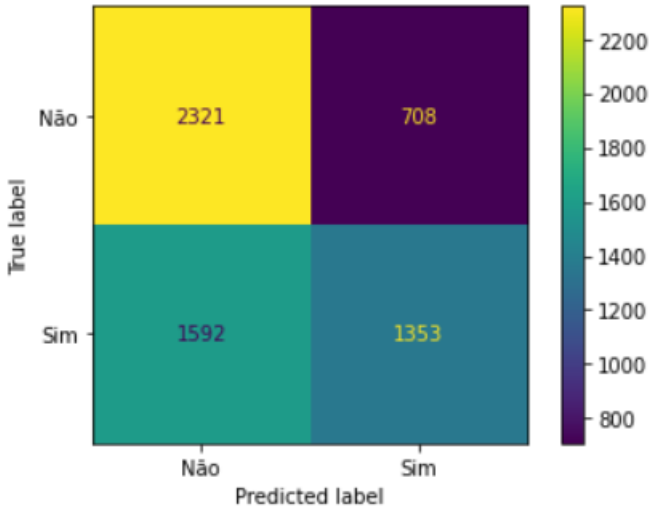


Agora que temos a classe totalmente balanceada, vamos realizar novamente o treinamento no modelo Naive Bayes.

```
classificador_nb_re = GaussianNB()
classificador_nb_re.fit(X_treinamento_re, Y_treinamento_re)
predicao_nb_re = classificador_nb_re.predict(X_teste_re)
confusao_nb_re = confusion_matrix(Y_teste_re, predicao_nb_re)
print(confusao_nb_re)
cmd_nb_re = ConfusionMatrixDisplay(confusao_nb_re, display_labels=classificador_nb_re.classes_).plot()
relatorio_nb_re = classification_report(Y_teste_re, predicao_nb_re)
print(relatorio_nb_re)
```

✓ 0.6s

```
[[2321  708]
 [1592 1353]]
              precision    recall  f1-score   support

         Não       0.59      0.77      0.67      3029
         Sim       0.66      0.46      0.54      2945

    accuracy                           0.61      5974
   macro avg       0.62      0.61      0.60      5974
weighted avg       0.62      0.61      0.61      5974
```



A Matriz de confusão demonstra grande frequência de classificação para falsos positivos e falsos negativos.

Verificamos que os valores das métricas do modelo não melhoraram muito após o oversampling. Valores como revocação e F1 score continuam tendo diferenças significativas entre as classes “Sim” e “Não”.

O valor da média da acurácia na validação cruzada para avaliar a generalização do modelo apresentou um valor baixo, 0.59.

```
validacao_nb_re = cross_val_score(classificador_nb_re, atributos_re, classe_alvo_re, scoring='accuracy', cv=5)
print(validacao_nb_re.mean())
```

✓ 0.4s

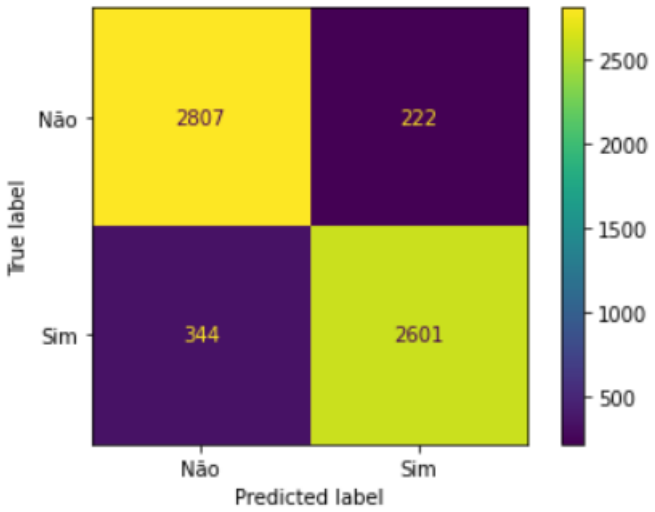
```
0.597461206833496
```

Agora vamos realizar novamente o treinamento no modelo Random Forest.

```
classificador_rf_re = RandomForestClassifier()
classificador_rf_re.fit(X_treinamento_re, Y_treinamento_re)
predicao_rf_re = classificador_rf_re.predict(X_teste_re)
confusao_rf_re = confusion_matrix(Y_teste_re, predicao_rf_re)
print(confusao_rf_re)
cmd_rf_re = ConfusionMatrixDisplay(confusao_rf_re, display_labels=classificador_rf_re.classes_).plot()
relatorio_rf_re = classification_report(Y_teste_re, predicao_rf_re)
print(relatorio_rf_re)
```

✓ 3.8s

```
[[2807  222]
 [ 344 2601]]
              precision    recall  f1-score   support

         Não       0.89      0.93      0.91      3029
         Sim       0.92      0.88      0.90      2945

    accuracy                           0.91      5974
   macro avg       0.91      0.90      0.91      5974
weighted avg       0.91      0.91      0.91      5974
```



Verificamos que os valores das métricas do modelo melhoraram um pouco em relação aos valores antes do oversampling. O valor da acurácia aumentou de 0.89 para 0.91 e os valores das métricas precisão, revocação e F1 score para a classe “Sim” também aumentaram de 0.91 para 0.92, 0.85 para 0.88 e 0.88 para 0.90 respectivamente.

O valor da média da acurácia na validação cruzada para avaliar a generalização do modelo apresentou um valor baixo, 0.78.

```
validacao_rf_re = cross_val_score(classificador_rf_re, atributos_re, classe_alvo_re, scoring='accuracy', cv=5)
print(validacao_rf_re.mean())
✓ 15.2s
0.7832423396636274
```

Com estes valores, o modelo Random Forest ainda que tenha tido uma pequena melhora continua tendo desempenho inferior aos demais modelos, indicando que o leve desbalanceamento no conjunto de dados antes do oversampling não teve grande relevância nos resultados obtidos anteriormente. Desta forma concluiu-se que não será necessário treinar novamente os demais modelos utilizando o conjunto de dados com o método oversampling.

6. Interpretação dos Resultados

Realizamos vários testes com modelos classificadores para nosso conjunto de dados tendo como alvo a coluna Fiscalizado. Buscamos como objetivo os atributos que indiquem uma boa predição para a nossa classe fiscalizado de forma que as ações de fiscalizações sejam feitas com maior eficiência.

Após todos os testes realizados com o treinamento dos modelos e a validação cruzada realizada, podemos resumir os resultados em uma tabela:

Algoritmo	**Classe**	**Modelo**				**Validação cruzada (cross-validation)**
		Acurácia (accuracy)	**Precisão (precision)**	**Revocação (recall)**	**Pontuação F1 (F1 score)**	**Acurácia**
Gaussian Naive Bayes	Não	0.63	0.63	0.8	0.71	0.63
	Sim		0.63	0.41	0.5	
Gaussian Naive Bayes *Oversampling (SMOTE)*	Não	0.61	0.59	0.77	0.67	0.59
	Sim		0.66	0.46	0.54	
Árvore de decisão (Decision Tree Classifier)	Não	0.95	0.96	0.96	0.96	0.84
	Sim		0.95	0.95	0.95	
Random forest	Não	0.89	0.89	0.93	0.91	0.82
	Sim		0.91	0.85	0.88	
Random forest *Oversampling (SMOTE)*	Não	0.91	0.89	0.93	0.91	0.78
	Sim		0.92	0.88	0.90	
SVM (Support Vector Machine)	Não	0.98	0.98	0.99	0.98	0.83
	Sim		0.99	0.97	0.98	
KNN (K-Nearest Neighbor)* **valor de k = 5*	Não	0.92	0.92	0.94	0.93	0.82
	Sim		0.92	0.90	0.91	
KNN (K-Nearest Neighbor)* **valor de k = 1*	Não	0.98	0.98	0.99	0.99	0.85
	Sim		0.98	0.98	0.98	

O modelo Gaussian Naive Bayes teve o pior desempenho entre os modelos, mesmo quando aplicado o oversampling (SMOTE). Os demais modelos apresentam bons valores, então vamos compará-los:

O modelo de Árvore de decisão teve uma boa métrica de acurácia, sendo seus valores para as métricas de precisão, revocação e pontuação F1 (F1 score) também muito bons.

Com os bons resultados das métricas do modelo de árvore de decisão pensamos que o modelo de Random Forest, que realiza uma combinação de vários modelos de árvore de decisão para se obter um único resultado, o que o torna um algoritmo mais robusto e complexo, teria melhores resultados. Porém os resultados das métricas do modelo Random Forest ficaram abaixo do modelo de árvore de decisão, mesmo após aplicação do método oversampling (SMOTE). Neste ponto, percebeu-se que a aplicação do método oversampling para corrigir o leve desbalanceamento da classe não contribuiu para o desempenho geral dos modelos treinados, reforçando a decisão que este leve desbalanceamento e a quantidade de registros no conjunto de dados original não tem influência no desempenho dos modelos. Logo foi descartado o treinamento do conjunto de dados com o método oversampling nos demais modelos que já apresentavam bons resultados.

O modelo SVM apresentou resultados muito bons nas métricas de acurácia, precisão, revocação e pontuação F1 (F1 score), sendo sua média na acurácia da validação cruzada ligeiramente inferior ao valor da média da validação cruzada do modelo de árvore de decisão, sendo que para o modelo svm os valores de falso negativo e falso positivo foram mais baixos do que os do modelo de árvore de decisão.

O modelo knn (valor de k = 5) teve o resultado da métrica acurácia inferior aos modelos de árvore de decisão e svm, tendo os seus valores de métricas de precisão, revocação e pontuação F1 (F1 score) abaixo dos valores das mesmas métricas dos modelos svm e árvore de decisão. O valor da média da acurácia do modelo knn (valor de k =5) também teve um desempenho inferior aos valores de média das acurácias da validação cruzada dos modelos svm e árvore de decisão.

Ao realizar novos treinamentos e testes da validação cruzada para o modelo knn variando os valores do parâmetro k, percebemos que quando o valor de k é igual a 1, sua média do valor da acurácia para a validação cruzada teve um desempenho para generalização de 0.85, valor maior do que o desempenho dos demais modelos até o momento. Verificamos as métricas para o modelo knn (valor de k = 1) e obtivemos os melhores valores para acurácia, precisão, revocação e pontuação F1 (F1 score) em relação ao modelo knn (valor de k = 5).

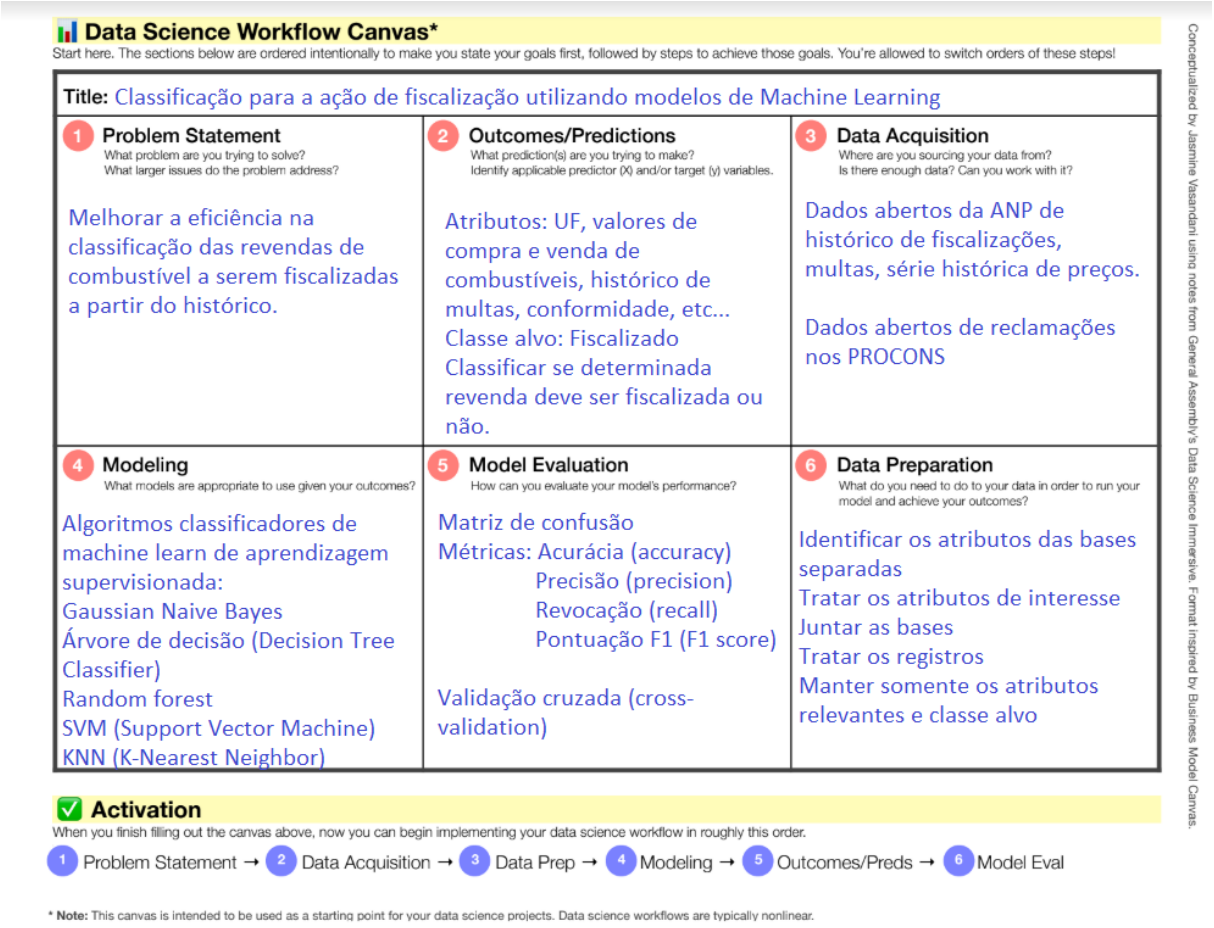
Realizar o treinamento com a validação cruzada variando o valor de k para o modelo knn demonstrou que podemos rever os valores usados anteriormente chegar a melhores desempenhos com os modelos. Para o nosso problema, o modelo knn (valor de k = 1)

apresentou melhor resultado das métricas, sendo o modelo mais equilibrado na métrica de revocação e com a maior média de generalização na acurácia da validação cruzada, além de ter os menores valores para as frequências de classificação da matriz de confusão para os falsos positivos e falsos negativos.

Toda esta iteração na ciência de dados se mostra necessária para revermos e validarmos os vários parâmetros, além de reavaliarmos ajustes, de forma que encontremos o resultado que melhor se ajuste para nossa necessidade.

7. Apresentação dos Resultados

Os resultados para execução deste trabalho foram seguidos pelas metas descritas no modelo Canvas de Vasandani.



Data Science Workflow Canvas*

Start here. The sections below are ordered intentionally to make you state your goals first, followed by steps to achieve those goals. You're allowed to switch orders of these steps!

Title: Classificação para a ação de fiscalização utilizando modelos de Machine Learning

1 Problem Statement
What problem are you trying to solve?
What larger issues do the problem address?

Melhorar a eficiência na classificação das revendas de combustível a serem fiscalizadas a partir do histórico.

2 Outcomes/Predictions
What prediction(s) are you trying to make?
Identify applicable predictor (X) and/or target (y) variables.

Atributos: UF, valores de compra e venda de combustíveis, histórico de multas, conformidade, etc...
Classe alvo: Fiscalizado
Classificar se determinada revenda deve ser fiscalizada ou não.

3 Data Acquisition
Where are you sourcing your data from?
Is there enough data? Can you work with it?

Dados abertos da ANP de histórico de fiscalizações, multas, série histórica de preços.

Dados abertos de reclamações nos PROCONS

4 Modeling
What models are appropriate to use given your outcomes?

Algoritmos classificadores de machine learn de aprendizagem supervisionada:
Gaussian Naive Bayes
Árvore de decisão (Decision Tree Classifier)
Random forest
SVM (Support Vector Machine)
KNN (K-Nearest Neighbor)

5 Model Evaluation
How can you evaluate your model's performance?

Matriz de confusão
Métricas: Acurácia (accuracy)
Precisão (precision)
Revocação (recall)
Pontuação F1 (F1 score)

Validação cruzada (cross-validation)

6 Data Preparation
What do you need to do to your data in order to run your model and achieve your outcomes?

Identificar os atributos das bases separadas
Tratar os atributos de interesse
Juntar as bases
Tratar os registros
Manter somente os atributos relevantes e classe alvo

Activation

When you finish filling out the canvas above, now you can begin implementing your data science workflow in roughly this order.

1 Problem Statement → 2 Data Acquisition → 3 Data Prep → 4 Modeling → 5 Outcomes/Preds → 6 Model Eval

* **Note:** This canvas is intended to be used as a starting point for your data science projects. Data science workflows are typically nonlinear.

Conceptualized by Jasmine Vasandani using notes from General Assembly's Data Science Immersive. Format inspired by Business Model Canvas.

Como já mencionado anteriormente, as ações de fiscalização geram custo e muitas vezes não são efetivas. No contexto de fiscalização de revendas de combustível, é necessário ser mais efetivo para que as ações de fiscalizações identifiquem situações que precisam ser resolvidas.

No objetivo é classificar com a maior assertividade possível quais revendas de combustível devem ser fiscalizadas, baseado no histórico de alguns motivos considerados importantes como não conformidade dos produtos, reclamações, multas, valores de compra e venda.

Coletamos os dados de fiscalizações, multas, conformidade e série histórica dos preços dos combustíveis do ano de 2019 diretamente da plataforma de dados abertos da ANP - Agência Nacional do Petróleo, Gás Natural e Biocombustíveis, e dados de Reclamações registradas no PROCONS.

Identificamos necessidades de tratamento para estes dados e os exploramos, analisando informações importantes, sendo necessário tratá-los até que estivessem adequados para serem analisados nos modelos de Machine Learning.

Utilizamos 5 modelos classificadores de aprendizado supervisionado: Naive Bayes, Arvore de decisão (Decision Tree Classifier), Random Forest, SVM (Support Vector Machine) e KNN (K-Nearest Neighbor).

Analisamos as métricas resultantes de cada modelo, sendo que percebemos a importância de variar seus parâmetros, reavaliar ajustes nos conjuntos de dados e realizar a validação cruzada para avaliar sua generalização, onde chegamos à conclusão que os modelos SVM e árvore de decisão possuem bons resultados em suas métricas, porém o modelo KNN (K-Nearest Neighbor) com o valor de k igual a 1 atenderia melhor o problema proposto.

8. Links

Link para o vídeo: https://youtu.be/gtAFpaG4M8s

Link para o repositório: https://github.com/sergiorej/TCC-PucMinas2021-CienciadeDados

REFERÊNCIAS

Ministério de Minas e Energia. **Dados Abertos**. Disponível em: <https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos>. Acesso em: 11/07/2021

ANP. **A - Série histórica de preços de combustíveis - revenda**. Disponível em: <https://dados.gov.br/dataset/serie-historica-de-precos-de-combustiveis-por-revenda>. Acesso em: 17/07/2021

ANP. **PMQC - Programa de Monitoramento da Qualidade dos Combustíveis**. Disponível em: <https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/pmqc-programa-de-monitoramento-da-qualidade-dos-combustiveis>. Acesso em: 17/07/2021

ANP. **Ações de Fiscalização**. Disponível em: <https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/acoes-de-fiscalizacao>. Acesso em: 17/07/2021

ANP. **Multas aplicadas com vencimento a partir de 2016**. Disponível em: <https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/multas-aplicadas-com-vencimento-a-partir-de-2016>. Acesso em: 17/07/2021

Ministério de Minas e Energia. **Cadastro Nacional de Reclamações Fundamentadas (PROCONS - Sindec)**. Disponível em: < https://dados.gov.br/dataset/cadastro-nacional-de-reclamacoes-fundamentadas-procons-sindec1>. Acesso em: 17/07/2021

Portallubes. **Percentual de conformidade do combustível em postos**. Disponível em: <https://portallubes.com.br/2019/01/percentual-de-conformidade-do-combustivel/>. Acesso em: 17/07/2021

Carboroil. **O papel da ANP na cadeia de combustíveis no Brasil**. Disponível em:

<https://carboroil.com.br/anp-agencia-nacional-petroleo-combustiveis-brasil/>. Acesso em: 14/07/2021

Instituto combustível legal. **ANP segue firme no combate ao mercado irregular de combustíveis no Rio de Janeiro**. Disponível em: <https://institutocombustivellegal.org.br/anp-segue-firme-no-combate-ao-mercado-irregular-de-combustiveis-no-rio-de-janeiro/>. Acesso em: 16/07/2021

Auto papo. **ANP treina funcionários do Procon para fiscalização de postos**. Disponível em: <https://autopapo.uol.com.br/noticia/anp-procon-fiscalizacao-de-postos/>. Acesso em: 17/07/2021

CASTRO, João Paullo. **Procon inicia treinamento com ANP para fiscalização em postos de combustíveis**. Disponível em: <https://agencia.ac.gov.br/procon-inicia-treinamento-com-anp-para-fiscalizacao-em-postos-de-combustiveis/>. Acesso em: 18/07/2021

GUIMARÃES, Amanda Munari. **Estatística: análise de correlação usando Python e R**. Disponível em: <https://medium.com/omixdata/estat%C3%ADstica-an%C3%A1lise-de-correla%C3%A7%C3%A3o-usando-python-e-r-d68611511b5a>. Acesso em: 10/08/2021

FILHO, Mário. **As Métricas Mais Populares para Avaliar Modelos de Machine Learning**. Disponível em: <https://www.mariofilho.com/as-metricas-mais-populares-para-avaliar-modelos-de-machine-learning/>. Acesso em: 11/08/2021

AZANK, Felipe. **Dados Desbalanceados — O que são e como lidar com eles**. Disponível em: <https://medium.com/turing-talks/dados-desbalanceados-o-que-s%C3%A3o-e-como-evit%C3%A1-los-43df4f49732b>. Acesso em: 13/08/2021

COELHO, Gustavo. **Como fazer cross-validation**. Disponível em: <https://gusrabbit.com/code/cross_validate/>. Acesso em: 11/08/2021

SANTANA, Rodrigo. **Validação Cruzada: Aprenda de forma simples como usar essa técnica**. Disponível em: <https://minerandodados.com.br/validacao-cruzada-aprenda-de-forma-simples-como-usar-essa-tecnica/>. Acesso em: 11/08/2021

APÊNDICE

Programação/Scripts

Procedimentos para baixar os arquivos (python):

```
import requests
import os
import zipfile

# Faz o download dos arquivos conforme a url fornecida e o coloca no ende-
reço do segundo parâmetro
def baixar_arquivo(url, endereco=None):
    if endereco is None:
        endereco = os.path.basename(url.split("?")[0])
    resposta = requests.get(url, stream=True)
    if resposta.status_code == requests.codes.OK:
        with open(endereco, 'wb') as novo_arquivo:
            for parte in resposta.iter_content(chunk_size=256):
                novo_arquivo.write(parte)
        print("Download finalizado. Arquivo salvo em: {}".format(endereco))
    else:
        resposta.raise_for_status()


# Seleciona os arquivos a serem baixados do site de dados abertos da ANP
def selecionar_arquivos_fiscalizacao_para_baixar():
    # Ações de fiscalização de 2019
    metadados_fiscalizacao = "https://www.gov.br/anp/pt-br/centrais-de-con-
teudo/dados-abertos/arquivos/arquivos-acoes-de-fiscalizacao/acoes-de-fiscali-
zacao-metadados.pdf"
    fiscalizacao = "https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-
abertos/arquivos/arquivos-acoes-de-fiscalizacao/acoes-de-fiscalizacao.csv"

    # cria o diretorio, caso não exista
    if not os.path.exists("dados\\fiscalizacoes\\"):
        os.makedirs("dados\\fiscalizacoes\\")

    print("Baixando metadados das ações de fiscalização de 2019")
    baixar_arquivo(metadados_fiscalizacao, "dados\\fiscalizacoes\\acoes-de-
fiscalizacao-metadados.pdf")

    print("Baixando acoes de fiscalizacao de 2019")
    baixar_arquivo(fiscalizacao, "dados\\fiscalizacoes\\acoes-de-fiscaliza-
cao.csv")


# Como estes arquivos são atualizados com uma certa frequencia, teremos sem-
pre os dados mais atualizados
```

```
def selecionar_arquivos_serie_historica_para_baixar():
    # Série histórica de preços de combustíveis - revenda -  2019
    metadados_serie_historica = "https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/shpc/metadados-levantamento-precos.pdf"
    serie_1_sem_2019 = "https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/shpc/dsas/ca/ca-2019-01.csv"
    serie_2_sem_2019 = "https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/shpc/dsas/ca/ca-2019-02.csv"

    # cria o diretorio, caso não exista
    if not os.path.exists("dados\\serie\\"):
        os.makedirs("dados\\serie\\")

    print("Baixando metadados da série histórica do primeiro semestre de 2019")
    baixar_arquivo(metadados_serie_historica, "dados\serie\metadados_serie_historica.pdf")

    print("Baixando série histórica do primeiro semestre de 2019")
    baixar_arquivo(serie_1_sem_2019, "dados\serie\sem_2019-1_CA.csv")

    print("Baixando série histórica do segundo semestre de 2019")
    baixar_arquivo(serie_2_sem_2019, "dados\serie\sem_2019-2_CA.csv")



def selecionar_arquivos_multas_para_baixar():
    # multas aplicadas pela anp entre 2016 e 2019
    metadados_multas_aplicadas_2016_a_2019 = "https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/mav/metadados-multas-aplicadas-2016a2019.pdf"
    multas_2016_a_2019 = "https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/arquivos/mav/multas-aplicadas-2016a2019.csv"

    # cria o diretorio, caso não exista
    if not os.path.exists("dados\\multas\\"):
        os.makedirs("dados\\multas\\")

    print("Baixando metadados multas aplicadas entre 2016 e 2019")
    baixar_arquivo(metadados_multas_aplicadas_2016_a_2019, "dados\multas\metadados_multas_aplicadas_2016_a_2019.pdf")

    print("Baixando multas aplicadas entre 2016 e 2019")
    baixar_arquivo(multas_2016_a_2019, "dados\multas\multas_2016_a_2019.csv")


def selecionar_arquivos_reclamacoes_para_baixar():
    # Reclamações registradas nos PROCONS – Sindec
```

```
    metadados_reclamacoes = "http://dados.mj.gov.br/dataset/8ff7032a-d6db-
452b-89f1-d860eb6965ff/resource/d87543d6-cf9d-4752-8f3c-1b0aa075dc45/down-
load/dicionariodadossindec3-0.pdf"
    reclamacoes = "http://dados.mj.gov.br/dataset/8ff7032a-d6db-452b-89f1-
d860eb6965ff/resource/c2cce323-24c2-4430-8918-e24b2966213c/download/crf2019-
dados-abertos.zip"

    # cria o diretorio, caso não exista
    if not os.path.exists("dados\\reclamacoes\\"):
        os.makedirs("dados\\reclamacoes\\")

    print("Baixando metadados de reclamacoes")
    baixar_arquivo(metadados_reclamacoes, "dados\\reclamacoes\\metadados_re-
clamacoes.pdf")

    print("Baixando dados de reclamacoes")
    baixar_arquivo(reclamacoes, "dados\\reclamacoes\\crf2019-dados-aber-
tos.zip")

    print("Extraindo dados de reclamacoes do arquivo zip")
    with zipfile.ZipFile("dados\\reclamacoes\\crf2019-dados-aber-
tos.zip", 'r') as zip_ref:
        zip_ref.extractall("dados\\reclamacoes\\")


def selecionar_arquivos_PMQC_para_baixar():
    # dados do programa de qualidade de 2019
    metadados_PMQC = "https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-
abertos/arquivos/pmqc/pmqc-metadados.pdf"

    PMQC_2019_01 = "http://www.anp.gov.br/arquivos/dadosaber-
tos/PMQC/PMQC_2019_01.csv"
    PMQC_2019_02 = "http://www.anp.gov.br/arquivos/dadosaber-
tos/PMQC/PMQC_2019_02.csv"
    PMQC_2019_03 = "http://www.anp.gov.br/arquivos/dadosaber-
tos/PMQC/PMQC_2019_03.csv"
    PMQC_2019_04 = "http://www.anp.gov.br/arquivos/dadosabertos/PMQC/2019-04-
pmqc.csv"
    PMQC_2019_05 = "http://www.anp.gov.br/arquivos/dadosabertos/PMQC/2019-05-
pmqc.csv"
    PMQC_2019_06 = "http://www.anp.gov.br/arquivos/dadosabertos/PMQC/2019-06-
pmqc.csv"
    PMQC_2019_07 = "http://www.anp.gov.br/arquivos/dadosabertos/PMQC/2019-07-
pmqc.csv"
    PMQC_2019_08 = "http://www.anp.gov.br/arquivos/dadosabertos/PMQC/2019-08-
pmqc.csv"
    PMQC_2019_09 = "http://www.anp.gov.br/arquivos/dadosabertos/PMQC/2019-09-
pmqc.csv"
```

```
    PMQC_2019_10 = "http://www.anp.gov.br/arquivos/dadosabertos/PMQC/2019-10-
pmqc.csv"
    PMQC_2019_11 = "http://www.anp.gov.br/arquivos/dadosabertos/PMQC/2019-11-
pmqc.csv"
    PMQC_2019_12 = "http://www.anp.gov.br/arquivos/dadosabertos/PMQC/2019-12-
pmqc.csv"

    # cria o diretorio, caso não exista
    if not os.path.exists("dados\\PMQC\\"):
        os.makedirs("dados\\PMQC\\")

    print("Baixando metadado PMQC - Programa de Monitoramento da Quali-
dade dos Combustíveis")
    baixar_arquivo(metadados_PMQC, "dados\PMQC\PMQC_metadados.pdf")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Janeiro de 2019")
    baixar_arquivo(PMQC_2019_01, "dados\PMQC\PMQC_2019_01.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Fevereiro de 2019")
    baixar_arquivo(PMQC_2019_02, "dados\PMQC\PMQC_2019_02.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Março de 2019")
    baixar_arquivo(PMQC_2019_03, "dados\PMQC\PMQC_2019_03.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Abril de 2019")
    baixar_arquivo(PMQC_2019_04, "dados\PMQC\PMQC_2019_04.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Maio de 2019")
    baixar_arquivo(PMQC_2019_05, "dados\PMQC\PMQC_2019_05.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Junho de 2019")
    baixar_arquivo(PMQC_2019_06, "dados\PMQC\PMQC_2019_06.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Julho de 2019")
    baixar_arquivo(PMQC_2019_07, "dados\PMQC\PMQC_2019_07.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Agosto de 2019")
    baixar_arquivo(PMQC_2019_08, "dados\PMQC\PMQC_2019_08.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Setembro de 2019")
```

```
    baixar_arquivo(PMQC_2019_09, "dados\PMQC\PMQC_2019_09.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Outubro de 2019")
    baixar_arquivo(PMQC_2019_10, "dados\PMQC\PMQC_2019_10.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Novembro de 2019")
    baixar_arquivo(PMQC_2019_11, "dados\PMQC\PMQC_2019_11.csv")

    print("Baixando PMQC - Programa de Monitoramento da Qualidade dos Combus-
tíveis - Dezembro de 2019")
    baixar_arquivo(PMQC_2019_12, "dados\PMQC\PMQC_2019_12.csv")
```