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

Eduardo de Freitas Rocha

CLASSIFICAÇÃO DO RISCO DE ACIDENTES NAS RODOVIAS FEDERAIS BRASILEIRAS

Belo Horizonte
2021

Eduardo de Freitas Rocha

CLASSIFICAÇÃO DO RISCO DE ACIDENTES NAS RODOVIAS FEDERAIS BRASILEIRAS

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.

Belo Horizonte
2021

SUMÁRIO

1. Introdução

1.1. Contextualização

Entre as enormes transformações que a sociedade atual tem vivido nos últimos tempos, sem dúvida alguma, uma das principais é a enorme geração/fabricação de dados que necessitam ser processados, analisados e armazenados de forma bastante racional e produtiva.

Evidentemente, o ponto central dessa transformação deve-se ao acentuado progresso da tecnologia, em seu sentido amplo, que permitiu que um volume e uma variedade considerável de dados fossem tratados em uma velocidade recorde. Aqui, verificamos um resumo simplista da definição de **Big Data**, definição esta criada pelo grupo Gartner, por volta de 2001, conhecido como os três Vs[1].

Com o passar do tempo, mais duas características agregaram o conceito de Big Data: valor e veracidade, formando-se assim os chamados 5Vs, principais atributos da atual definição de Big Data. Abaixo, um resumo dos 5Vs retirados do Wikipedia[2]:

- **Volume**: relacionado à grande quantidade de dados gerados;
- **Variedade:** as fontes de dados são muito variadas, o que aumenta a complexidade das análises;
- **Velocidade**: devido ao grande volume e variedade de dados, todo o processamento deve ser ágil para gerar as informações necessárias;
- **Veracidade:** a veracidade está ligada diretamente ao quanto uma informação é verdadeira;
- **Valor:** este conceito está relacionado com o valor obtido desses dados, ou seja, com a “informação útil”.

Dentro de todo esse universo de dados, no sentido amplo, aquilo que mais surpreende é a presença dessa transformação em praticamente todos os setores da sociedade, muitas vezes agregando um valor considerável.

Finanças, medicina, educação, bioinformática, políticas públicas, segurança, robótica são apenas alguns exemplos concretos onde é possível verificar a presença de dados massivos e toda a estrutura ao redor deles.

Outro conceito intrinsecamente ligado ao Big Data e que se correlaciona diretamente a ele é o de **Ciência de Dados**(*data science,* em inglês). Trata-se de uma área interdisciplinar que objetiva estudar e analisar dados estruturados e não-estruturados, com objetivo de detectar padrões, extrair um significado, fazer uma percepção dos dados, de forma a fazer uma análise preditiva, ou seja, prever algo.

Como dito, a utilização e aplicação desses dados massivos é diversa, em inúmeras áreas. Neste contexto, tentaremos abordar neste estudo uma análise a respeito de acidentes nas rodovias federais brasileiras.

Segundo o Ministério da Infraestrutura, a extensão total da malha rodoviária federal é de é de 75.553 mil km, dos quais 65.528 mil km (87%) correspondem a rodovias pavimentadas e 10.025 mil km (13%) correspondem a rodovias não pavimentadas[3]. Em 2019, por exemplo, houve um registro de 67.446 acidentes, sendo que mais da metade desses acidentes houve feridos. Tal fato merece uma análise e estudos bem aprofundados sobre o tema.

1.2. O problema proposto

A proposta deste estudo é a utilização de técnicas de pré-processamento, análise de dados e aprendizagem de máquina supervisionada para prever, de acordo com informações coletadas, trechos de rodovias federais brasileiras com potencialidade de ocorrência de acidentes graves ou não graves, de acordo com características especificadas. A frente, explicarei com mais detalhes o conceito estabelecido de acidentes graves.

Utilizando a ferramenta dos 5-Ws, apresento uma visão geral dos problemas e possíveis tentativas de soluções a serem estudadas:

- **(Why?) Por que analisar os dados sobre acidentes nas rodovias federais brasileiras?**

Sendo o principal meio de transporte utilizado pelos brasileiros, evidentemente a quantidade de acidentes em decorrência nas rodovias é bem maior comparado aos demais modais. Entretanto, o grau de letalidade, associado aos acidentes com graves feridos são assustadores no Brasil. Somente em 2019, tivemos 5.333 vidas perdidas; no período acumulado de 2007 a 2019 foram 94.081 vidas. Isso sem contar com o número de acidentes com feridos graves que deixa inúmeras sequelas e corroboram para o colapso do nosso sistema de saúde. Com a disponibilidade dos dados acerca dos acidentes, acredito que podemos usá-los em benefício da sociedade, sendo mais um aliado nas estratégias para dirimir esta situação caótica e extremamente complicada.

- **(Who?) De quem são os dados analisados?**

Os datasets utilizados são oficiais do governo federal brasileiro, especificamente do Departamento de Polícia Rodoviária Federal, disponível no site desse órgão. Tais datasets se originam do compilado de boletins de ocorrência feitos pelos policiais rodoviários em cada acidente.

- **(What?): Quais os objetivos com essa análise?**

O objetivo do estudo é prever e identificar trechos das rodovias federais brasileiras que possuem risco grave de acidentes, de acordo com os dados disponibilizados pela Polícia Rodoviária Federal (PRF).

- **(Where?): Quais os aspectos geográficos da análise?**

As bases utilizadas englobam todas as rodovias federais brasileira (popularmente conhecidas como “Brs”), ou seja, todo o Brasil. A título de curiosidade, existe toda uma lógica por trás da nomenclatura de cada rodovia. Estabelecida pelo chamado Plano Nacional de Viação(PVN), os números que sucedem a sigla BR possuem significado, sendo o primeiro algarismo indicador do sentido da rodovia, tipo radial, longitudinal, transversal, etc[4].

(When?): Qual o período está sendo analisado?

Nesse trabalho, optei por analisar todos os acidentes nas rodovias federais brasileiras ocorridos em todo o ano de 2019. Oficialmente, foram 67.446 acidentes.

1.3 Ferramentas adotadas

Em todo o trabalho, será utilizada a linguagem de programação Python, juntamente às principais bibliotecas disponíveis para análise de dados, visualizações gráficas e aprendizado de máquina, como Pandas, Numpy, Seaborn, Statistics, Scikit-learn(sklearn), entre outros.

2. Coleta de Dados

Nesse projeto, os datasets utilizados foram retirados diretamente o site da Polícia Rodoviária Federal, podendo ser consultado pelo link https://portal.prf.gov.br/dados-abertos-acidentes. Na verdade, foram usados dois datasets, ambos em formato ".csv " (em inglês, *Comma-separated values*).

A leitura de dos datasets utilizados é feita através do comando pd.read_csv. Mostro, a seguir, como foram dados os comandos, em python:

```
# Leitura dos datasets datatran2019 e acidentes2019_PESSOA.csv  pelo Pandas

acidentes = pd.read_csv("datatran2019.csv", sep=';',encoding='latin-1', parse_dates=['data_inversa'])
acidentes_pessoa = pd.read_csv("acidentes2019_PESSOA.csv", sep=';',encoding='latin-1', parse_dates=['data_inversa'])
```

Figura 1 – leitura dos datasets

O principal deles a ser utilizado no estudo é o "**datatran2019.csv**"(dataframe "acidentes"), retirado do campo "Agrupados por ocorrência" no site da PRF. Ele é composto por 67.446 linhas e um total de 30 colunas. Agrupa individualmente o registro de todos os acidentes de trânsito ocorridos no ano de 2019. Ou seja, em cada linha desse dataset ocorreu um acidente. Cada acidente é representado no dataset pela primeira coluna, chamada de "id".

Apresento, agora, uma lista descritiva das colunas/atributos do dataset[5].

Nome coluna/variável	Descrição	Tipo
id	Variável que representa o identificador do acidente	float64
data-inversa	Data da ocorrência no formato dd/mm/aaaa	datatime
dia_semana	Dia da semana da ocorrência	object
horario	Horário da ocorrência no formato hh:mm:ss	object
uf	Unidade da Federação	object
br	Representa o identificador da BR do acidente	float64
km	Identificação do km onde ocorreu o acidente, com valor mínimo de 0,1km.	object
municipio	Nome do município de ocorrência do acidente	object
causa_acidente	Identificação da causa principal do acidente	object
tipo_acidente	Identificação do tipo de acidente.	object
classificação_acidente	Classificação quanto à gravidade do acidente	object
fase_dia	Fase do dia no momento do acidente	object
sentido_via	Sentido da via considerando o ponto de colisão	object
condicao_meteorologica	Condição meteorológica no momento do acidente	object
tipo_pista	Tipo de pista considerando a quantidade de faixas	object
tracado_via	Descrição do traçado da via	object
uso_solo	Características do local do acidente: urbano = sim, Rural = não	object
pessoas	Total de pessoas envolvidas na ocorrência	int64
mortos	Total de pessoas mortas envolvidas na ocorrência	int64
feridos_leves	Total de pessoas com ferimentos leves envolvidas na ocorrência	int64
feridos_graves	Total de pessoas com ferimentos graves envolvidas na ocorrência	int64
ilesos	Total de pessoas ilesas envolvidas na ocorrência	int64
ignorados	Total de pessoas envolvidas na ocorrência e que não se soube o estado físico	int64

feridos	Total de pessoas feridas envolvidas na ocorrência(feridos_leves + feridos_graves)	int64
veiculos	Total de veículos envolvido na ocorrência	int64
latitude	Latitude do local do acidente em formato geodésico decimal	object
longitude	Longitude do local do acidente me formato geodésico decimal	object
regional	Local geográfico da regional da PRF	object
delegacia	Local geográfico da Delegacia da PRF	object
uop	Dado interno da PRF	object

Abaixo, a função “df.info()” em python que traz um resumo pequeno dos dados, em complemento aos já apresentados. É bem simplista, mas é possível ter uma noção dos tipos de dados, das observações, da quantidade de atributos e quantas entradas são nulas em cada atributo.

```
[6]: # Verificando as informações sobre os dataframe, incluindo a memória
     acidentes.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67446 entries, 0 to 67445
Data columns (total 30 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       67446 non-null  float64
 1   data_inversa             67446 non-null  datetime64[ns]
 2   dia_semana               67446 non-null  object
 3   horario                  67446 non-null  object
 4   uf                       67446 non-null  object
 5   br                       67351 non-null  float64
 6   km                       67351 non-null  object
 7   municipio                67446 non-null  object
 8   causa_acidente           67446 non-null  object
 9   tipo_acidente            67446 non-null  object
 10  classificacao_acidente   67446 non-null  object
 11  fase_dia                 67446 non-null  object
 12  sentido_via              67446 non-null  object
 13  condicao_metereologica   67446 non-null  object
 14  tipo_pista               67446 non-null  object
 15  tracado_via              67446 non-null  object
 16  uso_solo                 67446 non-null  object
 17  pessoas                  67446 non-null  int64
 18  mortos                   67446 non-null  int64
 19  feridos_leves            67446 non-null  int64
 20  feridos_graves           67446 non-null  int64
 21  ilesos                   67446 non-null  int64
 22  ignorados                67446 non-null  int64
 23  feridos                  67446 non-null  int64
 24  veiculos                 67446 non-null  int64
 25  latitude                 67446 non-null  object
 26  longitude                67446 non-null  object
 27  regional                 67446 non-null  object
 28  delegacia                67446 non-null  object
 29  uop                      64006 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(8), object(19)
memory usage: 15.4+ MB
```

Figura 2 – Informações do dataset “datatran2019” em Python

O outro dataset utilizado neste trabalho é o “acidentes2019_PESSOA.csv”(dataframe “acidentes_pessoa”) Ele também é originário das ocorrências de acidente de trânsito, entretanto, neste dataset há uma

discriminação individual das pessoas envolvidas no acidente, em cada linha do dataset.

```
[7]: # conhecendo o dataframe acidentes_pessoa
     acidentes_pessoa.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 162273 entries, 0 to 162272
Data columns (total 35 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   id                      162273 non-null  float64
 1   pesid                   162272 non-null  float64
 2   data_inversa            162273 non-null  datetime64[ns]
 3   dia_semana              162273 non-null  object
 4   horario                 162273 non-null  object
 5   uf                      162273 non-null  object
 6   br                      162042 non-null  float64
 7   km                      162042 non-null  object
 8   municipio               162273 non-null  object
 9   causa_acidente          162273 non-null  object
 10  tipo_acidente           162273 non-null  object
 11  classificacao_acidente  162273 non-null  object
 12  fase_dia                162273 non-null  object
 13  sentido_via             162273 non-null  object
 14  condicao_metereologica  162273 non-null  object
 15  tipo_pista              162273 non-null  object
 16  tracado_via             162273 non-null  object
 17  uso_solo                162273 non-null  object
 18  id_veiculo              162273 non-null  int64
 19  tipo_veiculo            162273 non-null  object
 20  marca                   154122 non-null  object
 21  ano_fabricacao_veiculo  152353 non-null  float64
 22  tipo_envolvido          162273 non-null  object
 23  estado_fisico           162273 non-null  object
 24  idade                   146514 non-null  float64
 25  sexo                    162273 non-null  object
 26  ilesos                  162273 non-null  int64
 27  feridos_leves           162273 non-null  int64
 28  feridos_graves          162273 non-null  int64
 29  mortos                  162273 non-null  int64
 30  latitude                162273 non-null  object
 31  longitude               162273 non-null  object
 32  regional                162273 non-null  object
 33  delegacia               162273 non-null  object
 34  uop                     153754 non-null  object
dtypes: datetime64[ns](1), float64(5), int64(5), object(24)
memory usage: 43.3+ MB
```

Figura 3: Informações do dataset "acidentes_pessoa" em Python

Perceba, na figura acima, que neste dataset há outros atributos envolvendo diretamente as características das pessoas e dos veículos envolvidos nos acidentes, tipo sexo, idade, estado físico, tipo do veículo, ano de fabricação.

A união dos datasets será demonstrada em tópico posterior, já que será necessário fazer um tratamento e exploração do dataset “acidentes2019_PESSOA”.

3. Processamento/Tratamento de Dados

A seguir, apresentarei os tratamentos que foram dados a algumas variáveis do dataset principal, “datatran2019”.

3.1 Dados ausentes

Uma importante consideração se faz necessária em relação aos datasets “datatran2019” e “acidentes2019_PESSOA”. Conforme mencionado anteriormente, ambos são oriundos das ocorrências dos acidentes de trânsito. Os sistemas da PRF para emissão do Boletim de Ocorrência são todos eletrônicos, de forma que os campos geradores dos atributos são, em sua maioria, com preenchimento fixos. Esse fato, aliado à capacidade intelectual do policial rodoviário, faz com que os datasets gerados praticamente não gerem *outliers* e possam ser tratados de forma mais limpa, sem necessidade de um grande retrabalho por parte do cientista de dados, no que se refere à limpeza dos dados.

Isso é percebido verificando a quantidade de valores nulos em todo o dataset, pelo comando ***“isnull().sum()”.*** Vejamos:

```
[7]: acidentes.isnull().sum() # VERIFICANDO VALORES FALTANTES

[7]: id                          0
     data_inversa                0
     dia_semana                  0
     horario                     0
     uf                          0
     br                         95
     km                         95
     municipio                   0
     causa_acidente              0
     tipo_acidente               0
     classificacao_acidente      0
     fase_dia                    0
     sentido_via                 0
     condicao_metereologica      0
     tipo_pista                  0
     tracado_via                 0
     uso_solo                    0
     pessoas                     0
     mortos                      0
     feridos_leves               0
     feridos_graves              0
     ilesos                      0
     ignorados                   0
     feridos                     0
     veiculos                    0
     latitude                    0
     longitude                   0
     regional                    0
     delegacia                   0
     uop                      3440
     dtype: int64
```

Figura 4: Informações sobre valores faltantes no dataframe “acidentes”

Verifica-se, através dessa Figura 4, a presença de 3.440 registros de valores faltantes do atributo “uop”, além de 95 registros dos atributos “br” e “km”.

No primeiro caso, “uop”, trata-se de um atributo com utilização interna da PRF. Ele diz repeito a algum dado da unidade física da PRF, não havendo uma correlação com o objeto do estudo, ou seja, verificação de locais com maior probabilidade de acidente grave. Dessa forma, esse atributo será removido em sua totalidade no uso dos algoritmos de aprendizado de máquina, não sendo necessário remover os itens faltantes, para não prejudicar os demais registros.

No tocante aos atributos “br” e “km”, ambos possuem apenas 95 registros com valores faltantes, o que representa aproximadamente 0,2% de todos os registros do dataset. Por ser um valor insignificante em relação ao total de acidentes, decidi eliminar os registros com os valores faltantes, através do método ***“dropna()”.*** Entretanto, tais alterações serão feitas no decorrer da Análise e Exploração de Dados, próximo item do estudo.

3.2 Dados duplicados

Outra verificação que se faz mister em uma análise de dados é a presença de valores duplicados. Através do comando ***"duplicated()"***, percebe-se que não há valores duplicados no dataset "datatran2019". Vejamos:

```
: acidentes[acidentes.duplicated() == True].shape[0]
  print(f"O número de dados duplicados é: {acidentes[acidentes.duplicated() == True].shape[0]}")

O número de dados duplicados é: 0
```

Figura 5: verificando valores duplicados

3.3 Definindo a classe(alvo) do dataset

Inicialmente, com uma análise superficial dos dados, os objetivos que eu buscava em relação ao estudo diziam respeito apenas a quantidade de mortos nas rodovias federais brasileiras, que geralmente são os números que mais assustam. Resolvi, então, agrupar o atributo "mortos", com a função ***"groupby()"*** e plotar um gráfico de barras(***plot.bar***) para ter uma noção de como esse dado estava distribuído.

```
[9]: # Agrupando acidentes por mortes

     agrupar_morte = acidentes.groupby(['mortos']).size()
     print(agrupar_morte)
     # verificando em um gráfico de barras
     agrupar_morte.plot.bar(color = 'red')

     mortos
     0    62854
     1     4063
     2      389
     3       91
     4       32
     5       12
     6        4
     7        1
     dtype: int64
[9]: <AxesSubplot:xlabel='mortos'>
```

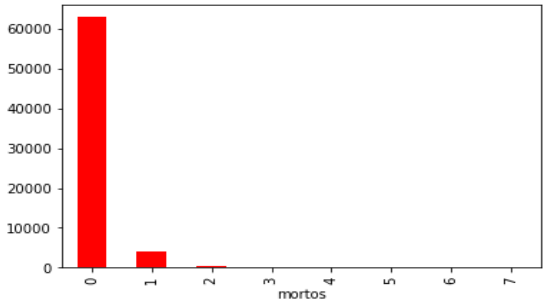


Figura 6: atributo "mortos"

Nas representações acima, é possível verificar um enorme desbalanceamento dos dados, ou seja, a porcentagem de acidentes em que não houve morte supera vultosamente as mortes. Abaixo, na Figura 7, reuni os atributos “mortos” e “pessoas” do dataset, de uma forma mais esclarecedora:

```
# Verificando o total de mortes
Total_mortos = acidentes['mortos'].sum()
Total_pessoas = acidentes['pessoas'].sum()
print('Mortos = ', Total_mortos)
print('Pessoas envolvidas em acidentes = ', Total_pessoas)
```

```
Mortos =  5333
Pessoas envolvidas em acidentes =  162273
```

Figura 7: nº de “mortos” e total de envolvidos

Sabendo dos outros atributos presentes no dataset, resolvi verificar, em porcentagem, qual a verdeira realidade das pessoas envolvidas em acidentes, usando o seguinte comando:

```
#Observando a porcentagem das mortes em relação ao número de pessoas envolvidas em acidentes
todos_envolvidos = acidentes['pessoas'].sum()
acidentes1 = round(acidentes[['ilesos','mortos','feridos_leves','feridos_graves', 'ignorados']].sum()/todos_envolvidos * 100, 1)
acidentes1
```

```
ilesos           42.3
mortos            3.3
feridos_leves    37.3
feridos_graves   11.4
ignorados         5.7
dtype: float64
```

Figura 8: pessoas envolvidas em acidentes

Nessa Figura 8, reuni os cinco atributos do dataset que, em conjunto, representa o atributo “pessoa”. Ou seja, a soma dos atributos “ilesos”, “mortos”, “feridos_leves”, “feridos_graves” e “ignorados”, em cada linha, representa o valor de “pessoas” do dataset. Detalhando no python usando o comando ***“loc”***.

```
acidentes.loc[10:15,['pessoas', 'mortos', 'feridos_graves', 'feridos_leves', 'ilesos','ignorados']]
```

	pessoas	mortos	feridos_graves	feridos_leves	ilesos	ignorados
10	3	0	2	0	1	0
11	4	0	0	4	0	0
12	2	0	0	1	1	0
13	1	0	0	1	0	0
14	2	0	0	2	0	0
15	2	0	2	0	0	0

Figura 9: pessoas envolvidas em acidentes

A partir da compilação desses dados, consegui verificar um número considerável e de suma importância para o estudo: a proporção de feridos graves(“feridos_graves”) unido com a de mortos representa cerca de 14,7% do total de pessoas envolvidas em acidentes.

Resolvi também criar um gráfico de setores para melhor representar esses valores. Com o título de “% DA GRAVIDADE DOS ACIDENTES EM 2019”, é possível verificar o quão importante é estudar melhor o comportamento das rodovias brasileiras.

```
# Verificando em um gráfico de pizza
labels = 'Ignorada','Nenhuma', 'Leve','Fatal','Grave'
percentagem = [ 5.7, 42.3,37.3,3.3,11.4]

fig1, ax1 = plt.subplots()
ax1.pie(percentagem, labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)

ax1.axis('equal')

#Título do gráfico
ax1.set_title('% DA GRAVIDADE DOS ACIDENTES EM 2019')

plt.show()
```

Figura 10: comando de criação do gráfico

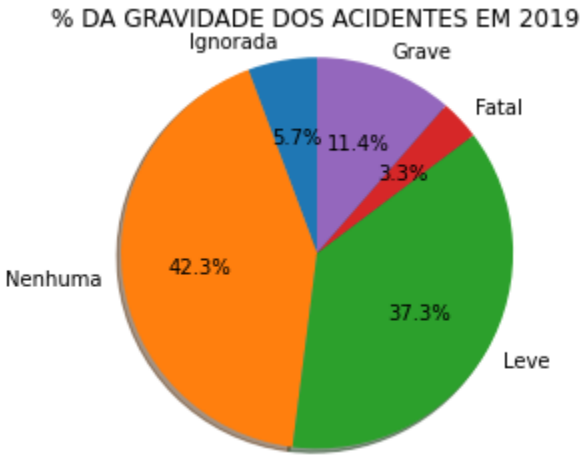


Figura 11: gráfico de setor

Evidentemente que 5.333 mortes em rodovias federais em um ano apenas é algo assustador, comparado aos demais países com um mínimo de educação no trânsito e com respeito às leis. Entretanto, a proporção de feridos graves aliada a de mortos ultrapassa qualquer limite e merece um estudo bem mais aprofundado.

A partir de todo este contexto e análise dos dados, optei por definir a minha classe, ou seja, atributo alvo que desejo classificar, da seguinte forma: onde houver o registro de valores diferentes de zero nos atributos “mortos” e “feridos_graves” será atribuído o valor booleano “True”. Caso contrário, nos registros em que tais atributos o valor seja zero, receberá o booleano “False”.

Essa classe será denominada **“risco”**. Perceba que o objetivo desta classe não é quantificar o número de mortos e/ou pessoas com ferimentos graves por acidente, mas sim verificar se houve, pelo menos uma pessoa, no registro, que esteja nessa qualificação. Sendo positivo, o retorno será “True”, senão “False”.

Em Python, existem inúmeras formas de executar esse comando. Fiz a opção por uma talvez mais longa, entretanto bem detalhada e explicativa. Criei um dataframe, “**risco_grave”**, a partir do dataset “datatran_2019” somente com os atributos mortos e feridos_graves. Vejamos:

```
# ****************** CRIANDO O DATAFRAME 'risco_grave" COM APENAS OS ATRIBUTOS mortos e feridos_graves
risco_grave = pd.read_csv("datatran2019.csv", sep=';',encoding='latin-1', usecols=[18,20])
risco_grave.head()
```

	mortos	feridos_graves
0	0	0
1	0	0
2	0	0
3	0	1
4	0	1

Figura 12: novo dataframe risco_grave

Após, acrescentei a classe(atributo alvo) “risco” a esse novo dataframe, através do comando ***“sum(axis)”***. Observe, na figura 12, que, além de acrescentar a nova coluna, o comando soma os valores, fazendo, assim, a união entre os dois atributos.

```
#criando a coluna 'risco', que será a CLASSE do nosso estudo,  para unir(SOMAR) os valores de 'mortos' + 'feridos_graves'
risco_grave['risco'] = risco_grave.sum(axis=1)
risco_grave.loc[40:50]
```

	mortos	feridos_graves	risco
40	0	0	0
41	0	0	0
42	0	0	0
43	0	0	0
44	0	0	0
45	0	0	0
46	0	1	1
47	1	1	2
48	1	1	2
49	0	0	0
50	0	1	1

Figura 13: criando a classe “risco”

Finalizando o tratamento ao dataframe “risco_grave”, exclui os atributos “mortos” e “feridos_graves” pelo método “***drop()***” e, logo em seguida, transformei os dados númericos em booleanos, através do comando “***astype(bool)***”. Veja na Figura 14:

```
[38]: # RETIRANDO OS ATRIBUTOS 'mortos' e 'feridos_graves'
      # TRANSFORMANDO O DATAFRAME EM BOOLEANO
      risco_grave = risco_grave.drop(columns=['mortos', 'feridos_graves'])
      risco_grave = risco_grave.astype(bool)
      risco_grave.head(10)

      ## VALORES = 0 >>>> ASSUMEM O VALOR FALSE
      ## VALORES DIFERENTES DE 0 >>>> VALOR TRUE >> REPRESENTA OS ACIDENTES FATAIS OU COM RISCO GRAVE
```

[38]:

	risco
0	False
1	False
2	False
3	True
4	True
5	False
6	False
7	False
8	True
9	True

Figura 14: transformando em booleano

Como esse novo dataframe “risco_grave” é oriundo do dataset “datatran2019”, assim como o dataframe “acidentes”, bastou unir os dois dataframes pelo comando ***“concat()”.*** Agora, a classe “risco” já está inserida no dataframe “acidentes”, sendo possível a continuação da análise dos dados.

```
# UNINDO O DATAFRAME "acidentes" com o "risco_grave"

risco_acidentes = pd.concat([acidentes,risco_grave], axis=1, join='inner')
```

Figura 15: unindo os dataframes “acidentes” e “risco_grave”

```
[140]: risco_acidentes.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67446 entries, 0 to 67445
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      67446 non-null  float64
 1   data_inversa            67446 non-null  datetime64[ns]
 2   dia_semana              67446 non-null  object
 3   horario                 67446 non-null  object
 4   uf                      67446 non-null  object
 5   br                      67351 non-null  float64
 6   km                      67351 non-null  object
 7   municipio               67446 non-null  object
 8   causa_acidente          67446 non-null  object
 9   tipo_acidente           67446 non-null  object
 10  classificacao_acidente  67446 non-null  object
 11  fase_dia                67446 non-null  object
 12  sentido_via             67446 non-null  object
 13  condicao_metereologica  67446 non-null  object
 14  tipo_pista              67446 non-null  object
 15  tracado_via             67446 non-null  object
 16  uso_solo                67446 non-null  object
 17  pessoas                 67446 non-null  int64
 18  mortos                  67446 non-null  int64
 19  feridos_leves           67446 non-null  int64
 20  feridos_graves          67446 non-null  int64
 21  ilesos                  67446 non-null  int64
 22  ignorados               67446 non-null  int64
 23  feridos                 67446 non-null  int64
 24  veiculos                67446 non-null  int64
 25  latitude                67446 non-null  object
 26  longitude               67446 non-null  object
 27  regional                67446 non-null  object
 28  delegacia               67446 non-null  object
 29  uop                     64006 non-null  object
 30  risco                   67446 non-null  bool
dtypes: bool(1), datetime64[ns](1), float64(2), int64(8), object(19)
memory usage: 15.5+ MB
```

Figura 16: dataframe “risco_acidentes” com a classe “risco”

4. Análise e Exploração dos Dados

4.1 União dos Datasets

Conforme explanado no início, o dataset complementar “acidentes2019_PESSOA” também é oriundo das ocorrências de acidentes de trânsito. Entretanto, ele é composto de 35 colunas, 5 a mais do que o objeto do estudo, e 162.273 linhas.

No âmbito do objetivo deste estudo, apenas um atributo desse dataset será necessário: o **tipo de veículo(tipo_veiculo)**. Isso porque a maioria dos atributos já constam no datatran2019 e as características do motorista e do passageiro, além da marca do veículo, não são essenciais, a meu ver, à detecção de um risco do acidente ser grave ou não. Ou seja, eu acredito que apenas a informação do tipo de veículo é relevante para a classificação, já que seria possível dar o tratamento correto em relação à segurança da rodovia, de acordo com o veículo.

Para tanto, criei um dataframe, chamado **tipo_veiculo**, usando o dataset “acidentes2019_PESSOA”, apenas com os atributos “id” e “tipo_veículo” , da seguinte forma:

```
# ****************** CRIANDO UM DATAFRAME COM APENAS OS ATRIBUTOS tipo veiculo
tipo_veiculo = pd.read_csv("acidentes2019_PESSOA.csv", sep=';',encoding='latin-1',usecols=[0,19])
tipo_veiculo.head(10)
```

	id	tipo_veiculo
0	182256.0	Caminhão
1	182263.0	Caminhão
2	182277.0	Caminhão-trator
3	182289.0	Caminhão-trator
4	182307.0	Caminhão-trator
5	182307.0	Caminhão-trator

Figura 17: dataframe com atributos “id” e “tipo_veiculo”

Algumas situações mereceram ser tratadas nesse novo dataframe. A primeira delas é a quantidade de itens que o atributo tipo_veiculo possui. Vejamos, com o uso da função ***“groupby”***, na figura 18:

```
[51]: agrupando_tipo= tipo_veiculo.groupby('tipo_veiculo').size()
      print(agrupando_tipo)

tipo_veiculo
Automóvel              70650
Bicicleta               2086
Caminhonete            14228
Caminhão               11517
Caminhão-trator        12068
Camioneta               4160
Carro de mão               6
Carroça-charrete          96
Ciclomotor               441
Micro-ônibus            1489
Motocicleta            31858
Motoneta                3746
Não Informado             39
Outros                   662
Quadriciclo                3
Reboque                   43
Semireboque              119
Trator de esteira          1
Trator de rodas           71
Trator misto               5
Trem-bonde                 8
Triciclo                  23
Utilitário              1900
Ônibus                  7054
dtype: int64
```

Figura 18: verificando itens agrupados

Pelas informações da figura, nota-se uma segregação do tipo de veículos com o uso de termos estritamente técnicos, de relevância considerável para a PRF. Ao todo, temos 22 tipos de veículos, além de dois itens chamados de “Não Informado” e “Outros”, representando dados não exatos. Resolvi agregar(usando a função ***replace***) alguns destes tipos, porque acredito ter pouca relevância, em questão de acidentes, a diferença entre uma motoneta e uma motocicleta, por exemplo.

Além disso, uni os tipos “Não Informado” e “Outros” ao tipo “Automóvel”, já que este representa a moda dos valores e a quantidade de itens não é tão considerável. O resultado foi o seguinte:

AGREGANDO ALGUNS VALORES DESSES VEÍCULOS, DA SEGUINTE FORMA:

1 - Bicicleta = Bicileta + Carro de mão

2 - Carroça = Carroça + charrete

3 - Reboque = Reboque + Semireboque

4 - Motocicleta = Motocicleta + Motoneta + Triciclo + Quadriciclo + Ciclomotor

5 - Automóvel = Automóvel + Não Informado + Outros

6 - Caminhonete = Caminhonete + Camioneta

7 - Trator = Trator de esteira + Trator de rodas + Trator misto

8 - Trem = Trem-bonde

9 - Caminhão = Caminhão + Caminhão-Trator

10 - Ônibus = Ônibus + Micro-ônibus + Utilitário

```
# Tratando a coluna 'tipo_veiculo':
tipo_veiculo.tipo_veiculo.replace(['Carro de mão', 'Carroça-charrete', 'Semireboque'],['Bicicleta', 'Carroça', 'Reboque'], inplace=True)
tipo_veiculo.tipo_veiculo.replace(['Ciclomotor','Triciclo','Quadriciclo', 'Motoneta'],'Motocicleta', inplace=True)
tipo_veiculo.tipo_veiculo.replace(['Caminhão-trator','Camioneta','Trem-bonde'],['Caminhão','Caminhonete','Trem'], inplace=True)
tipo_veiculo.tipo_veiculo.replace(['Trator de rodas','Trator misto','Trator de esteira'],'Trator', inplace=True)
tipo_veiculo.tipo_veiculo.replace(['Utilitário','Micro-ônibus'],'Ônibus', inplace=True)
tipo_veiculo.tipo_veiculo.replace(['Não Informado','Outros'],'Automóvel', inplace=True)
```

```
agrupando_tipo= tipo_veiculo.groupby('tipo_veiculo').size()
print(agrupando_tipo)
```

```
tipo_veiculo
Automóvel      71351
Bicicleta       2092
Caminhonete    18388
Caminhão       23585
Carroça           96
Motocicleta    36071
Reboque           162
Trator             77
Trem                8
Ônibus          10443
dtype: int64
```

Figura 19: agrupando os itens do atributo

Após a agregação dos tipos de veículos, em 10 principais, foi necessário fazer outra adequação. Atribui o número, conforme a figura 19 acima, para cada tipo do veículo. Foi usado o comando ***replace()***. Automaticamente, esse atributo passou a ser do tipo inteiro, que contribui para a análise futura dos algoritmos de aprendizado de máquina.

```
# TRANSFORMANDO CADA TIPO DE VEÍCULO EM NÚMERO
tipo_veiculo.tipo_veiculo.replace(['Bicicleta', 'Carroça', 'Reboque', 'Motocicleta'],[1, 2, 3, 4], inplace=True)
tipo_veiculo.tipo_veiculo.replace(['Automóvel', 'Caminhonete','Trator','Trem'], [5,6,7,8], inplace=True)
tipo_veiculo.tipo_veiculo.replace(['Caminhão', 'Ônibus'],[9,10], inplace=True)
```

```
agrupando_tipo= tipo_veiculo.groupby('tipo_veiculo').size()
print(agrupando_tipo)
```

```
tipo_veiculo
1      2092
2        96
3       162
4     36071
5     71351
6     18388
7        77
8         8
9     23585
10    10443
dtype: int64
```

Figura 20: transformando cada tipo_veiculo em um nº

Esse dataframe **“tipo_veiculo”** possui 162.273 linhas(já que ele é oriundo do dataset “acidentes2019_PESSOA”).

O dataframe objeto do estudo possui 67.446 linhas, com uma “id” única para cada registro. Essa diferença ocorre devido ao fato de dataset “acidentes2019_PESSOA” englobar todas pessoas, e não somente o registro do acidente. Por isso, há nesse dataset vários “id” repetidos, já que, em um mesmo acidente, pode haver várias pessoas e vários tipos de veículos.

Assim, foi necessário que eu estabelecesse alguma forma de selecionar somente um **“tipo_veiculo”** para cada “id” **,** ficando de acordo com as mesmas 67.446 linhas do dataset objeto do estudo. Optei por criar uma ordem de classificação quando uma “id” possui mais de um tipo.

Considerando que veículos de grande porte possuem mais chance de causar um acidente mais grave, com um maior número de envolvidos, estabeleci a seguinte padronização:

10-Ônibus > 9-Caminhão > 8-Trem > 7-Trator > 6-Caminhonete > 5-Automóvel > 4-Motocicleta > 3-Reboque > 2-Carroça > 1-Bicicleta.

Quando houver mais de uma linha com a mesma “id”, haverá uma seleção da “id” mais forte, ou seja, seguindo a ordem acima. Assim, o dataframe “tipo_veiculo” conterá as mesmas 67.446 linhas do dataset “**datatran2019.csv**”.

Esse procedimento foi efetuado da seguinte forma:

- criei o dataframe “dataframesuniao” pela junção de ‘risco_acidentes”(principal) e ‘tipo_veiculo” através do comando ***merge().***
- através do comando ***sort-values(),*** foi selecionado o atributo “tipo_veiculo” na ordem decrescente de valores.

```
# Fazendo a união de dois datasets diferentes pelo id, através do comando .merge()
dataframesuniao = risco_acidentes.merge(tipo_veiculo, on='id', how='inner' )
```

```
dataframesuniao = dataframesuniao.sort_values('tipo_veiculo', ascending=False)
```

```
dataframesuniao[['id' ,'tipo_veiculo' ]].head()
```

	id	tipo_veiculo
56929	210025.0	10
56047	209613.0	10
13934	189325.0	10
150103	255082.0	10
150102	255082.0	10

Figura 21: unindo os dataframes risco_acidentes e "tipo_veiculo"

- exclui as linhas que possuíam o atributo "id" duplicados, utilizando o comando ***drop_duplicates()***

```
# deletando os ids duplicados e mantendo o primeiro para existir apenas 1  rgistro para cada id com a coluna nova (tipo_veiculo)
df = dataframesuniao.drop_duplicates(subset='id', keep='first')
```

Figura 22: retirando "id" duplicadas

- nesse comando, usei o parâmetro *"keep=first"* para que o dataframe criado, "df", seguisse a preferência estabelecida por mim, ou seja, mantivesse a id com maior valor na coluna principal "tipo_veiculo". Vejamos as informações desse novo dataframe.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 67446 entries, 56929 to 129752
Data columns (total 32 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      67446 non-null  float64
 1   data_inversa            67446 non-null  datetime64[ns]
 2   dia_semana              67446 non-null  object
 3   horario                 67446 non-null  object
 4   uf                      67446 non-null  object
 5   br                      67351 non-null  float64
 6   km                      67351 non-null  object
 7   municipio               67446 non-null  object
 8   causa_acidente          67446 non-null  object
 9   tipo_acidente           67446 non-null  object
 10  classificacao_acidente  67446 non-null  object
 11  fase_dia                67446 non-null  object
 12  sentido_via             67446 non-null  object
 13  condicao_metereologica  67446 non-null  object
 14  tipo_pista              67446 non-null  object
 15  tracado_via             67446 non-null  object
 16  uso_solo                67446 non-null  object
 17  pessoas                 67446 non-null  int64
 18  mortos                  67446 non-null  int64
 19  feridos_leves           67446 non-null  int64
 20  feridos_graves          67446 non-null  int64
 21  ilesos                  67446 non-null  int64
 22  ignorados               67446 non-null  int64
 23  feridos                 67446 non-null  int64
 24  veiculos                67446 non-null  int64
 25  latitude                67446 non-null  object
 26  longitude               67446 non-null  object
 27  regional                67446 non-null  object
 28  delegacia               67446 non-null  object
 29  uop                     64006 non-null  object
 30  risco                   67446 non-null  bool
 31  tipo_veiculo            67446 non-null  int64
dtypes: bool(1), datetime64[ns](1), float64(2), int64(9), object(19)
memory usage: 16.5+ MB
```

Figura 23: dataframes unidos e com "id" única

Finalmente, após todo o tratamento, reunimos os dois datasets de acordo com o objetivo do estudo. Agora, o dataframe principal(chamado "df") possui as mesmas 67.446 linhas, representando em cada linha um acidente, além de 32 colunas, sendo adicionadas a coluna "risco", que representa a classe do estudo, além do atributo "tipo_veiculo", proveniente do dataset **"acidentes2019_PESSOA"**.

4.2 Seleção de Atributos

O estudo agora conta com apenas o dataframe "df" com 32 colunas(atributos e classe) e 67.446 linhas. Conforme mencionado o objetivo do estudo é classificar

determinado trecho de uma rodovia como sendo grave ou não, de acordo com alguns dados estabelecidos.

Um grande número de atributos pode tornar o trabalho de classificação muito complexo. De acordo com o estudo do aprendizado de máquina e do objeto do negócio, é possível inferir alguns atributos que não são relevantes para esta classificação.

Dentre esses eles, temos aqueles que são estritamente relacionados ao controle interno da PRF. São eles "regional", "delegacia" e "uop". Todos foram retirados do dataframe.

Outro atributo que também foi retirado é o "causa_acidente". Trata-se de um fato que aconteceu após o acidente, que não caracteriza necessariamente a rodovia, mas sim um possível comportamento do motorista no momento do acidente. Além do mais, ele possui características semelhantes ao atributo "tipo_acidente", que se adéqua melhor ao trabalho em questão.

Outros atributos que, a princípio, seriam de suma importância ao trabalho inicial, mas que deixaram de ser relevantes são "pessoas", "mortos", "feridos_leves", "feridos_graves", "ilesos", "ignorados" e "feridos". Com a criação da classe, denominada "risco", esses atributos passaram diretamente ou indiretamente a serem contemplados no alvo do trabalho, sendo, portanto, descartados.

O atributo "município" também foi excluída do dataframe. É cediço que este atributo possui a função apenas de identificar a localização do acidente em sentido amplo. A localização exata do acidente, em tese, é definida pelos atributos "br" e "km". Com esses dois atributos, já é possível verificar o local do acidente.

Pelo mesmo motivo explicado, também retirei os atributos "latitude" e "longitude". Ou seja, ambos já são contemplados nos dois atributos "br" e "km", tornando, até certo ponto, redundantes.

Por fim, eliminei também os atributos "data_inversa" e "id". Este representa apenas o identificador do acidente, utilizado para dar um número sequencial e diferente a cada acidente ocorrido. Sua utilização não acrescenta em nada a capacidade preditiva do modelo. Já o atributo "data_inversa" possui também um número considerável de valores únicos, a saber 365(um ano). Como será necessário codificar os valores dos atributos em numéricos, para os algoritmos de aprendizagem de máquina, a presença de inúmeros valores únicos pode prejudicar a execução de certos algoritmos de classificação.

A seguir, o comando direto que eliminou todos os atributos mencionados:

```
# ELIMINANDO ATRIBUTOS DESNECESSÁRIOS AO ESTUDO
acidentes = df.drop(columns=['id', 'data_inversa','municipio','causa_acidente', 'pessoas', 'mortos', 'feridos_leves','feridos_graves'])
acidentes = acidentes.drop(columns=['ilesos','ignorados', 'feridos', 'latitude', 'longitude', 'regional', 'delegacia', 'uop'])
```

```
acidentes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 67446 entries, 56929 to 129752
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   dia_semana              67446 non-null  object
 1   horario                 67446 non-null  object
 2   uf                      67446 non-null  object
 3   br                      67351 non-null  float64
 4   km                      67351 non-null  object
 5   tipo_acidente           67446 non-null  object
 6   classificacao_acidente  67446 non-null  object
 7   fase_dia                67446 non-null  object
 8   sentido_via             67446 non-null  object
 9   condicao_metereologica  67446 non-null  object
 10  tipo_pista              67446 non-null  object
 11  tracado_via             67446 non-null  object
 12  uso_solo                67446 non-null  object
 13  veiculos                67446 non-null  int64
 14  risco                   67446 non-null  bool
 15  tipo_veiculo            67446 non-null  int64
dtypes: bool(1), float64(1), int64(2), object(12)
memory usage: 8.3+ MB
```

Figura 24: eliminando atributos

4.3 Transformação do Tipos de Atributos

Verificando a figura 24, acima, percebe-se que os atributos do dataframe "acidentes" são, em sua maioria do tipo *object*, além do atributo alvo que é do tipo booleano (*bool)*. Este fato é de difícil compreensão para os algoritmos de classificação. Assim, foi necessário transformar os valores dos atributos de object e bool para inteiro(int64).

Os atributos "km" e "horário", por conterem uma alta quantidade de valores únicos, foram transformados em numéricos com o uso do utilitário labelEncoder da biblioteca scikit-learn.

```
# Transformação dos atributos categóricos em atributos numéricos, passando o índice de cada coluna CATEGÓRICA
labelencoder = LabelEncoder()
acidentes['km'] = labelencoder.fit_transform(acidentes['km'])
acidentes['horario'] = labelencoder.fit_transform(acidentes['horario'])
```

Figura 25: transformando atributos 'km' e 'horário' em numéricos

Já os demais atributos cujo tipo é object foram convertidos em numéricos com a substituição manual dos valores por números.

```
fase_dia_values = {'Plena Noite' : 0, 'Amanhecer' : 1, 'Pleno dia' : 2, 'Anoitecer' : 3}

sentido_via_values = {'Crescente' : 0, 'Decrescente' : 1, 'Não Informado' : 2}

condicao_metereologica_values = {'Céu Claro' : 0, 'Nublado' : 1, 'Chuva' : 2, 'Garoa/Chuvisco' : 3, 'Ignorado' : 4,
                                 'Nevoeiro/Neblina' : 5, 'Vento' : 6, 'Sol' : 7,  'Granizo' : 8, 'Neve': 9}

tipo_pista_values = {'Múltipla' : 0, 'Dupla' : 1, 'Simples' : 2}

tracado_via_values = {'Curva' : 0, 'Reta' : 1, 'Viaduto' : 2, 'Interseção de vias' : 3, 'Não Informado' : 4, 'Rotatória' : 5,
                      'Desvio Temporário' : 6, 'Retorno Regulamentado' : 7, 'Túnel' : 8, 'Ponte' : 9}

uso_solo_values = { 'Sim' : 1, 'Não' : 0 }

risco_values = { False : 0, True : 1}



acidentes.replace({'dia_semana': dia_semana_values,
                   'uf': uf_values,
                   'tipo_acidente': tipo_acidente_values,
                   'classificacao_acidente': classificacao_acidente_values,
                   'fase_dia': fase_dia_values,
                   'sentido_via': sentido_via_values,
                   'condicao_metereologica' : condicao_metereologica_values,
                   'tipo_pista' : tipo_pista_values,
                   'tracado_via' : tracado_via_values,
                   'uso_solo' : uso_solo_values,
                   'risco' : risco_values,

                  }, inplace=True)
```

Figura 26: transformando demais atributos em numéricos

Com o novo tratamento, todas os atributos e a classe passaram a ser numéricos.

```
acidentes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 67351 entries, 56929 to 129752
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   dia_semana              67351 non-null  int64
 1   horario                 67351 non-null  int32
 2   uf                      67351 non-null  int64
 3   br                      67351 non-null  float64
 4   km                      67351 non-null  int32
 5   tipo_acidente           67351 non-null  int64
 6   classificacao_acidente  67351 non-null  int64
 7   fase_dia                67351 non-null  int64
 8   sentido_via             67351 non-null  int64
 9   condicao_metereologica  67351 non-null  int64
 10  tipo_pista              67351 non-null  int64
 11  tracado_via             67351 non-null  int64
 12  uso_solo                67351 non-null  int64
 13  veiculos                67351 non-null  int64
 14  risco                   67351 non-null  int64
 15  tipo_veiculo            67351 non-null  int64
dtypes: float64(1), int32(2), int64(13)
memory usage: 8.2 MB
```

Figura 27: dataframe ‘acidentes’ todo numérico

4.4 Balanceamento da Variável Alvo

Ao criar a variável alvo (classe), verificou-se um detalhe de extrema importância que não pode ser descartado nos problemas de classificação de aprendizado de máquina: o **desbalanceamento da base de dados**. Verifica-se que existe uma grande desproporção entre a quantidade de acidentes com chamado risco ‘grave’ e a quantidade de acidentes com risco ‘não-grave’.

Este fato pode comprometer consideravelmente a performance de algoritmos de aprendizado de máquina, deixando o modelo de classificação enviesado. Assim, normalmente, o modelo tende a classificar os novos dados, ou seja, aqueles que serão inseridos para análise, como sendo a da classe que possui mais exemplos, priorizando essa classe majoritária.

Observe como a classe estava inicialmente distribuída:

```
# Verificando a quantidade de valores únicos da coluna risco
acidentes.risco.value_counts()
```

```
0    49108
1    18243
Name: risco, dtype: int64
```

```
ax = sns.countplot(x="risco", data=acidentes)
ax.set_xlabel('Risco Grave')
ax.set_ylabel('Quantidade')
```

```
Text(0, 0.5, 'Quantidade')
```

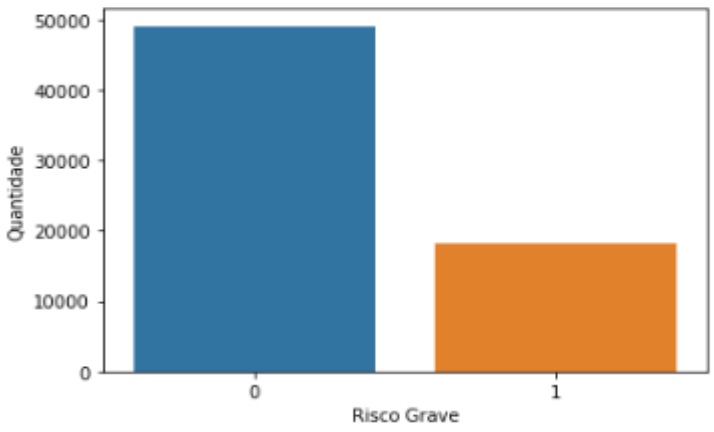


Figura 28: Classe(atributo alvo) desbalanceada

Para solucionar esse problema de desbalanceamento, de forma manual, criei um novo dataframe que, aleatoriamente, reduziu a quantidade de registros da classe majoritária. É o chamado **undersampling**.

Essa técnica foi utilizada, primeiramente, criando um novo dataframe, “data2”, oriundo do dataframe “acidentes”, com 18.243 registros com a classe alvo 0. Isso quer dizer que esse dataframe possui somente registros “sem risco grave”.

Logo após, adicionei a esse novo dataframe os 18.243 registros com o “risco grave”, com o método ***append()***.

```
# Selecionando um dataframe oriundo de "acidentes"  com  somente valores 0, ou seja, sem risco grave.
data2 = acidentes[acidentes.risco==0].sample(18243)
```

```
# adicionando ao dataframe criado a mesma quantidade de valores 1, com risco grave.
data = data2.append(acidentes[acidentes.risco==1].sample(18243))
```

```
data.risco.value_counts()
```

```
1    18243
0    18243
Name: risco, dtype: int64
```

Figura 29: Criando dataframe balanceado

```
[153]: ax = sns.countplot(x="risco", data=data)
       ax.set_xlabel('Risco Grave')
       ax.set_ylabel('Quantidade')
```

```
[153]: Text(0, 0.5, 'Quantidade')
```

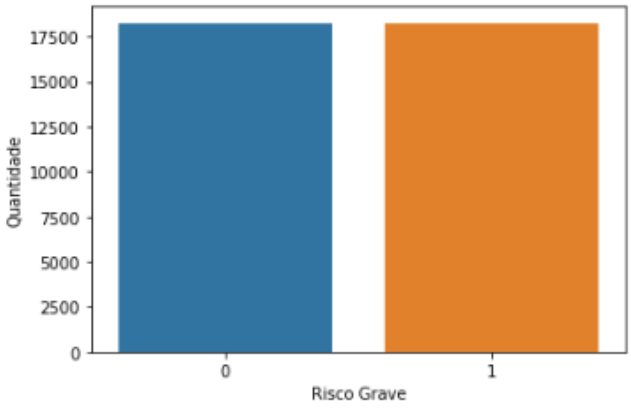


Figura 30: representação gráfica do dataframe balanceado

Portanto, aplicando a técnica de balanceamento das bases e dando os demais tratamentos relatados ao dataframe objeto do estudo, pode-se dizer que, em tese, o dataframe está apto a ser trabalhado por algoritmos de aprendizado de máquina.

5. Criação de Modelos de Machine Learning

Este estudo visa, conforme detalhado em tópicos anteriores, detectar os trechos nas rodovias federais brasileiras onde haverá um risco grave de acidente, conforme algumas condições estabelecidas.

Esse objetivo se dá, neste escopo, através do aprendizado de máquina (machine learning). Podemos dizer que a doutrina divide o aprendizado de máquina em três principais categorias. Aprendizagem supervisionada, aprendizagem não-supervisionada e aprendizagem por reforço.

Aprendizagem supervisionada é a tarefa de encontrar uma função a partir de dados de treinamento rotulados. O objetivo é encontrar os parâmetros ótimos que ajustem um modelo que possa prever rótulos desconhecidos em outros objetos (o conjunto teste) [6].

Na aprendizagem não supervisionada há menos informações acerco dos objetos, ou seja, o conjunto de treinamento não é rotulado. O objetivo, nesse contexto, é observar algumas similaridades entre os objetos e incluí-los em grupos apropriados[6].

Já na aprendizagem por reforço não há um conjunto de treinamento, rotulado ou não. Aqui o algoritmo recebe feedback da análise de dados, orientando o usuário para o melhor resultado. Nele, o sistema não é treinado com o conjunto de dados de amostra. Em vez disso, o sistema aprende por meio de tentativa e erro[7].

Este trabalho lida diretamente ao aprendizado de máquina supervisionado, pois aqui teremos um algoritmo que aprenderá com o conjunto de dados de treinamento, uma vez que os dados de treinamento são rotulados com a classe ao qual pertencem.

Após o treinamento, a próxima etapa é denominada de teste. O modelo será validado com o uso de dados rotulados, mas que não foram usados na etapa de treinamento. O objetivo dessa nova etapa é testar a generalização do modelo, ou seja, verificar se o modelo está bem treinado a prever a saída certa para novas instâncias. Com o modelo treinado e testado, outros dados não rotulados (ou seja,

sem a classe ao qual pertencem) servem de entrada para esse modelo. A saída é a predição das classes dos dados.

Para a tarefa de classificação, inúmeros algoritmos podem ser usados. No âmbito desse estudo, utilizarei três, no intuito de comparar qual apresenta melhores resultados de acordo com o objetivo do trabalho. São eles: **Support Vector Machine(SVM)**, **K-Nearest Neighbor(KNN)** e o **Random Forest**.

5.1 Support Vector Machine – SVM

O SVM, traduzido Máquina de Vetor de Suporte, em termos gerais, é um algoritmo que busca uma linha de separação entre duas classes distintas analisando os dois pontos, um de cada grupo, mais próximos da outra classe. Assim, o algoritmo escolhe o denominado hiperplano(reta) em maiores dimensões entre dois grupos que se distanciam mais de cada um[8].

Na figura abaixo, o hiperplano H3 representa a reta ótima

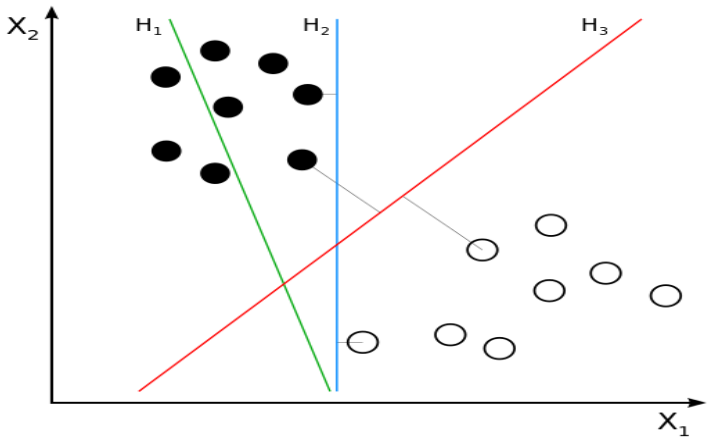


Figura 31: SVM separando hiperplanos[9]

O SVM padrão toma como entrada um conjunto de dados e prediz, para cada entrada dada, qual de duas possíveis classes a entrada faz parte, o que faz do SVM um classificador linear binário não probabilístico. O que ele realmente faz é encontrar uma linha de separação, o hiperplano, entre os dados de duas classes. Essa linha busca maximizar a distância entre os pontos mais próximos em relação a cada uma das classes[10].

5.2 K-Nearest Neighbor – KNN

O algoritmo K Vizinhos mais Próximos(traduzido) armazena todos os casos disponíveis e classifica novos casos por maioria de votos de seus vizinhos k. O caso que está sendo atribuído à classe é mais comum entre os seus k vizinhos mais próximos medidos por uma função de distância[11]. Essa distância pode ser medida por funções, como Euclidiana, Manhattan, Minkowski e distância de Hamming.

A variável k é o principal parâmetro a ser selecionado no modelo. É ela que direciona a quantidade de vizinhos. Abaixo, uma representação gráfica que exemplifica, de forma bastante simples, mas bem explicativa, como funciona esse algoritmo.

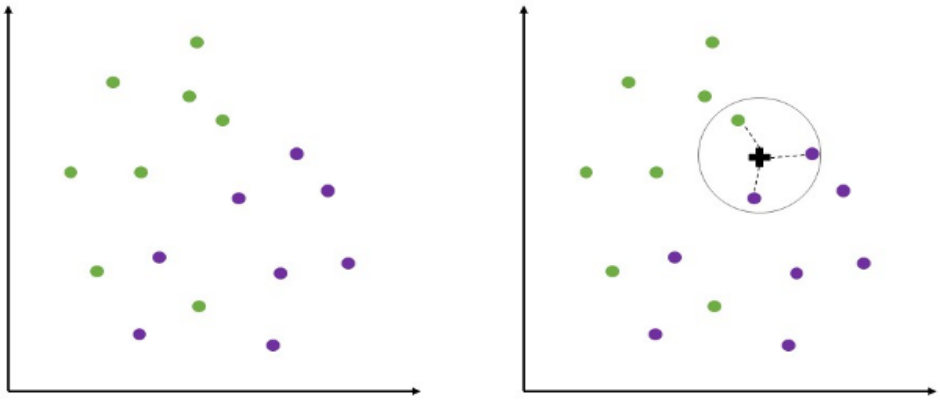


Figura 32: Classificação 2-d usando KNN quando k = 3

No lado esquerdo da figura acima temos um gráfico 2-d de dezesseis pontos: 8 são verdes e 8 roxos. A figura da direita mostra a classificação de um novo ponto(cruz preta), usando KNN quando k = 3. Verifica-se os três pontos mais próximos sendo feito a contagem de cada cor nesse intervalo de três pontos. Para o exemplo, dois são roxos e um verde. Assim, a classificação da cruz preta será roxa[12].

5.3 Random Forest (Florestas aleatórias)

O algoritmo Random Forest consistem em um conjunto de árvores de decisão geradas dentro de um mesmo objeto. Cada objeto (conjunto de árvores) passa por um mecanismo de votação (bagging), que elege a classificação mais votada. Essa classificação encontra-se nos nós terminais das mesmas[13].

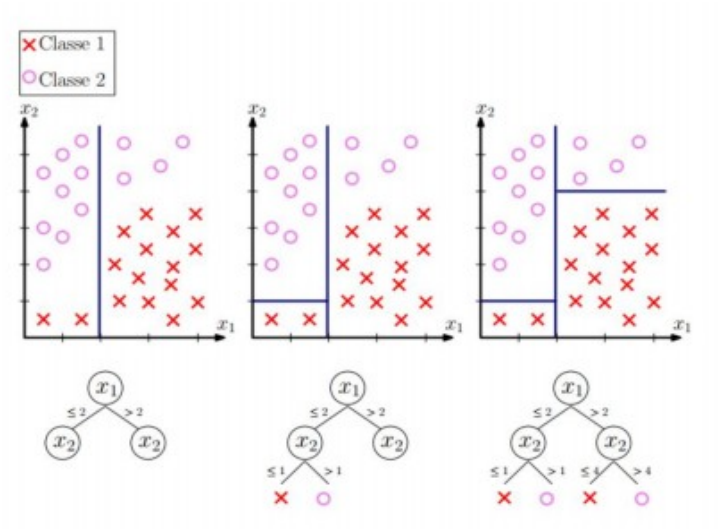


Figura 33: gerando uma árvore de decisão[13]

Aqui, o problema de classificação é, em termos simples, separar as superfícies de decisão em um espaço cujo número de dimensões é igual ao número de parâmetros de entrada no classificador. Ou seja, o classificador random forest separa as superfícies de decisão por meio da criação de uma sequência de hiperplanos paralelos aos eixos, assim como está na figura 33. [13].

Um algoritmo de floresta aleatória adiciona aleatoriedade extra ao modelo, ao criar as árvores. Ele não busca a melhor característica em ao fazer uma partição de nodos, mas sim a melhor característica em um **subconjunto** aleatório das características. Esse processo cria uma grande diversidade, o que normalmente leva a geração de modelos melhores[14].

5.4 Utilização dos modelos

De forma manual, para iniciar os testes dos modelos de aprendizado de máquina, se faz necessário, primeiramente, dividir o dataframe, de forma que de um lado tenhamos os atributos e, de outro, nosso alvo.

```
# Deletando a variável alvo e alimentando a mesma em Y
X_train = data.drop('risco',axis=1)
y = data['risco']
```

Figura 34: separando os atributos da classe

Além disso, é preciso separar os dados que serão utilizados nos algoritmos em treino e teste. Isso pode ser feito de forma automática pelo comando ***train_test_split()*** .Essa modificação está exposta na próxima figura:

```
X_treino, X_teste, y_treino, y_teste = train_test_split(X_train, y)
```

Figura 35: separando dados em treino e teste

Agora, optei por utilizar os três modelos de aprendizagem de máquina em sua forma *default,* **padrão**. Ou seja, não inseri nenhum parâmetro do modelo e também não utilizei nenhuma técnica de pré-processamento. É o método ***fit()*** que treina os modelos e constrói os algoritmos de aprendizagem de máquina. *V*ejamos:

Support Vector Macchine

```
svm1 = svm.SVC().fit(X_treino, y_treino)
```

```
previsoes = svm1.predict(X_teste)
acuracia = accuracy_score(y_teste, previsoes)
precisao = precision_score(y_teste, previsoes)
revocacao = recall_score(y_teste, previsoes)
confusao = confusion_matrix(y_teste, previsoes)

print('Matriz de Confusão')
print(confusao)
print()
print('Acurácia = ' , acuracia)
print('Precisão = ' , precisao)
print('Revocação = ' , revocacao)
print()
```

```
Matriz de Confusão
[[2491 2092]
 [2226 2313]]

Acurácia =  0.5266388949791713
Precisão =  0.5250851305334847
Revocação =  0.5095836087243887
```

Figura 36: aplicação do modelo Support Vector Macchine

K-Nearest Neighbor

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_treino, y_treino)
previsoes = knn.predict(X_teste)
acuracia = accuracy_score(y_teste, previsoes)
precisao = precision_score(y_teste, previsoes)
revocacao = recall_score(y_teste, previsoes)
confusao = confusion_matrix(y_teste, previsoes)

print('Matriz de Confusão')
print(confusao)
print()
print('Acurácia = ' , acuracia)
print('Precisão = ' , precisao)
print('Revocação = ' , revocacao)
print()
```

```
Matriz de Confusão
[[2371 2212]
 [2147 2392]]

Acurácia =  0.5221442666081999
Precisão =  0.5195482189400521
Revocação =  0.5269883234192554
```

Figura 37: aplicação do modelo K-Nearest Neighbor

Random Forest

```
rf = RandomForestClassifier(n_estimators = 1)
rf.fit(X_treino, y_treino)
```

```
RandomForestClassifier(n_estimators=1)
```

```
previsoes = rf.predict(X_teste)
acuracia = accuracy_score(y_teste, previsoes)
precisao = precision_score(y_teste, previsoes)
revocacao = recall_score(y_teste, previsoes)
confusao = confusion_matrix(y_teste, previsoes)

print('Matriz de Confusão')
print(confusao)
print()
print('Acurácia = ' , acuracia)
print('Precisão = ' , precisao)
print('Revocação = ' , revocacao)
print()
```

```
Matriz de Confusão
[[2849 1733]
 [1676 2864]]

Acurácia =  0.626288094716071
Precisão =  0.6230150097889928
Revocação =  0.6308370044052863
```

Figura 38: aplicação do modelo Random Forest

Nitidamente verifica-se que tais modelos não possuem boas métricas. A partir de então, resolvi utilizar algumas ferramentas e técnicas para tentar melhorar o desempenho dos algoritmos.

A primeira delas é a **Pipeline,** uma Classe contida na biblioteca Scikit-Learn. Ela cria um **objeto** com um ou mais transformadores em sequência e também um estimador no final. Transformadores são aqueles que transformam nossos dados, como o próprio nome diz, e estimadores são os nossos modelos, preditores[15].

Dessa forma, os Pipelines encapsulam várias etapas de uma só vez. Como vantagem, podemos dizer que ele melhora a produtividade dos algoritmos, facilita o uso de técnicas de validação e seleção de modelos e evita erros de manipulação de conjuntos de treino e teste[16].

Nesse trabalho, utilizando o Pipeline, criei três objetos, com um transformador e os algoritmos a serem utilizados, da seguinte forma:

```
pipe_random_forest = Pipeline([
        ('scl', StandardScaler()),
        ('clf', RandomForestClassifier())
        ]
)
```

```
pipe_random_forest.steps
```

```
[('scl', StandardScaler()), ('clf', RandomForestClassifier())]
```

```
pipe_svm = Pipeline([
        ('scl', StandardScaler()),
        ('clf', svm.SVC())
        ]
)
```

```
pipe_knn = Pipeline([
        ('scl', StandardScaler()),
        ('clf', KNeighborsClassifier())
        ]
)
```

Figura 39: criando um objeto Pipeline para cada algoritmo

A título de informação, o algoritmo svm é subdividido em outros tipos. Utilizei uma classe denominada **Supporte Vector Classification**(SVC) neste estudo.

Perceba, na figura 35, que usei para todos os objetos criados o comando ***StandardScaler().*** Trata-se de um pré-processador também contido na biblioteca Scikit-learn, utilizado para normalizar os dados do dataframe.

Essa tarefa, chamada de **standartization,** é muito útil para aumentar a eficiência dos modelos de aprendizado de máquina. A ideia do modelo é

normalizar/padronizar todos os atributos do dataframe, individualmente, antes de aplicar o modelo de aprendizagem de máquina. Isso porque a maioria dos algoritmos de machine learning assume que os dados estão padronizados na hora de gerar o modelo, ou seja, partem do princípio de que os dados estão na mesma escala.

Dessa forma, com o uso do Pipeline, estabeleci um transformador de normalidade e os algoritmos de aprendizado de máquina com o objetivo de dar um ganho de escala e produtividade ao trabalho.

No intuito de definir qual seria o modelo com melhor desempenho, utilizei a técnica do **grid search** nos três modelos. Essa ferramenta é uma pesquisa exaustiva, automatizada, realizada sobre valores de parâmetros específicos de um modelo.

O Grid Search ajusta (tuning) os hiperparâmetros de cada um dos modelos, fazendo de maneira sistemática diversas combinações dos parâmetros e, depois de avaliá-los, os armazenará em um único objeto. Permite, assim, determinar os melhores valores de parâmetro e maximizar a qualidade de cada modelo.

Hiperparâmetros são parâmetros que não são aprendidos diretamente nos estimadores. Assim, é necessário o uso dessa ferramenta para otimizar os parâmetros de cada modelo[17]. Utilizei, nesse trabalho, o GridSearchCV pertencente à biblioteca Scikit-learn.

Abaixo, a forma utilizada para criar um GridSearch para cada estimador:

```
valores = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

grid_params_rf = [{
    'clf__criterion': ['gini', 'entropy'],
    'clf__min_samples_leaf': valores,
    'clf__max_depth': valores,
    'clf__min_samples_split': valores[1:]
}]

grid_params_svm = [{
    'clf__kernel': ['linear', 'rbf'],
    'clf__C': valores
}]

grid_params_knn = [{
    'clf__n_neighbors': valores,
}]
```

Figura 40: definindo os parâmetros

```
gs_rf = GridSearchCV(
    estimator=pipe_random_forest,
    param_grid=grid_params_rf,
    scoring='accuracy',
    verbose=100,
    cv=10
)
```

```
gs_svm = GridSearchCV(
    estimator=pipe_svm,
    param_grid=grid_params_svm,
    scoring='accuracy',
    verbose=100,
    cv=10,
)
```

```
gs_knn = GridSearchCV(
    estimator=pipe_knn,
    param_grid=grid_params_knn,
    scoring='accuracy',
    verbose=100,
    cv=10,
)
```

Figura 41: criando um GridSearch para cada modelo

6. Apresentação dos Resultados

A utilização do GridSearch testa exaustivamente todos as combinações de parâmetros possíveis. Essa etapa, toda automatizada, levou um tempo significativo.

```
gs_svm.fit(X_train,y)
gs_knn.fit(X_train,y)
gs_rf.fit(X_train,y)
```

Figura 42: rodando os algoritmos

Na verdade, as validações do GridSearch levaram a um considerável gasto computacional(o script ficou bem poluído com a utilização da ferramenta), entretanto, subsidia de maneira bem detalhada e mais segura a tomada de decisão para o algoritmo correto.

Em todos os algoritmos, está imbutido, junto ao GridSearch, técnicas de **validação cruzada(Cross-validation)**. Isso quer dizer que todos eles possuem uma ótima capacidade de generalização. Ou seja, são criados diferentes conjuntos de treino e teste, para dar uma certeza que os modelos performaram bem.

Vejamos os resultados para cada algoritmo:

Support Vector Macchine

```
# Melhores parametros e scoring
print('Melhores parâmetros: %s' % gs_svm.best_params_)
print('Melhores Acurácia: %.3f' % gs_svm.best_score_)
```

```
Melhores parâmetros: {'clf__C': 1, 'clf__kernel': 'rbf'}
Melhores Acurácia: 0.691
```

Figura 43: resultado para SVM

K-Nearest Neighbor

```
# Melhores parametros e scoring
print('Melhores parâmetros: %s' % gs_knn.best_params_)
print('Melhores Acurácia: %.3f' % gs_knn.best_score_)
```

```
Melhores parâmetros: {'clf__n_neighbors': 10}
Melhores Acurácia: 0.653
```

Figura 44: resultado para KNN

Random Forest

```
# Melhores parametros e scoring
print('Melhores parâmetros: %s' % gs_rf.best_params_)
print('Melhores Acurácia: %.3f' % gs_rf.best_score_)
```

```
Melhores parâmetros: {'clf__criterion': 'gini', 'clf__max_depth': 10, 'clf__min_samples_leaf': 3, 'clf__min_samples_split': 9}
Melhores Acurácia: 0.713
```

Figura 45: resultado para o random forest

6.1 Métricas de Validação

Acurácia é uma das métricas de validação de um modelo de aprendizado de máquina e indica a performance geral do modelo, ou seja, entre todas as classificações realizadas pelo modelo, quantas o algoritmo efetuou corretamente. Em termos leigos, ela representa o número de previsões corretas de um modelo.

Como o dataframe utilizado foi balanceado, a acurácia se torna a principal métrica do modelo, sendo a referência utilizada para a definição.

Verifica-se, portanto, que o algoritmo **Random Forest,** após o uso das ferramentas aludidas, apresentou a maior acurácia(melhor perfomance) entre os demais e será o modelo a ser utilizado para a classificação das rodovias.

Vejamos a comparação entre a acurácia de cada modelo, antes e após a utilização das ferramentas(Ajustado nas figuras):

Variação da acurácia			
MODELO	Padrão	Ajustado	Ganho
SVM	0,5267	0,691	31,19%
KNN	0,5221	0,653	25,07%
Random Forest	0,6263	0,713	13,84%

Figura 46: acurácia dos modelos

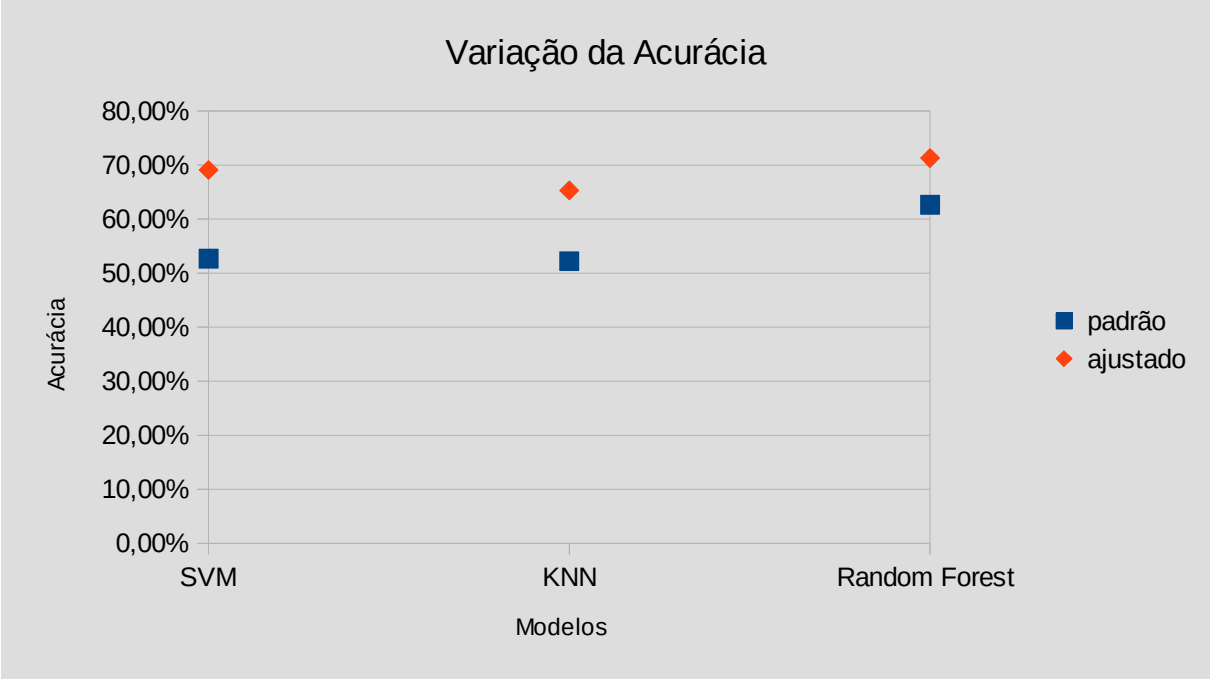


Figura 47: comparação dos modelos

É notável o ganho de performance entre os modelos após a utilização do GridSearch, combinado com o Pipeline. Além do mais, esse ajuste diminui consideravelmente a possibilidade de erros por parte do cientista de dados, conforme informado.

Mas, em relação ao trabalho efetuado, outra métrica muito importante deve ser atentamente analisada nesse estudo. Trata-se da **revocação/sensibilidade** (*recall*, em inglês). Tal métrica será exemplificada na execução real desse trabalho.

O modelo Random Forest obteve aproximadamente 71,30% de taxa de acurácia. Essa proporção indica o acerto do modelo, ou seja, classificar corretamente o risco de acidente de um trecho de rodovia como ou não.

Entretanto, o modelo, como qualquer outro, apresentou erros, que podem ser divididos em dois tipos:

1. informar que determinado trecho da rodovia apresenta um risco grave de acidente, sendo que, na realidade, esse trecho não traz risco; e
2. apontar que uma certa localidade da rodovia não apresenta risco, sendo que, na verdade, esse ponto da rodovia é muito arriscado.

A **revocação** é definida justamente nessa segunda situação. Ela aponta a razão entre a quantidade de exemplos **classificados** corretamente como de risco grave e a quantidade de exemplos que **realmente** são graves.

No âmbito do desse trabalho, é necessário que essa métrica seja a maior possível, já que o objetivo é preservar a vida. O modelo classificar um trecho como sem risco, sendo que há um risco iminente de acidentes pode gerar uma falsa percepção por parte do motorista e do setor público que o trecho é seguro, sendo que não é. A consequência desse fato é uma maior probabilidade de acidente grave.

O contrário, teoricamente, não gera tanta repercussão, quando o objetivo é a segurança. Ou seja, afirmar que um trecho possui risco, sendo que nesse trecho não existe, caracteriza um erro, entretanto, menos grave que o outro exemplo.

Vejamos como essa métrica se comportou, em relação ao modelo Random Forest padrão e o ajustado pelo GridSearch e a utilização do Pipeline:

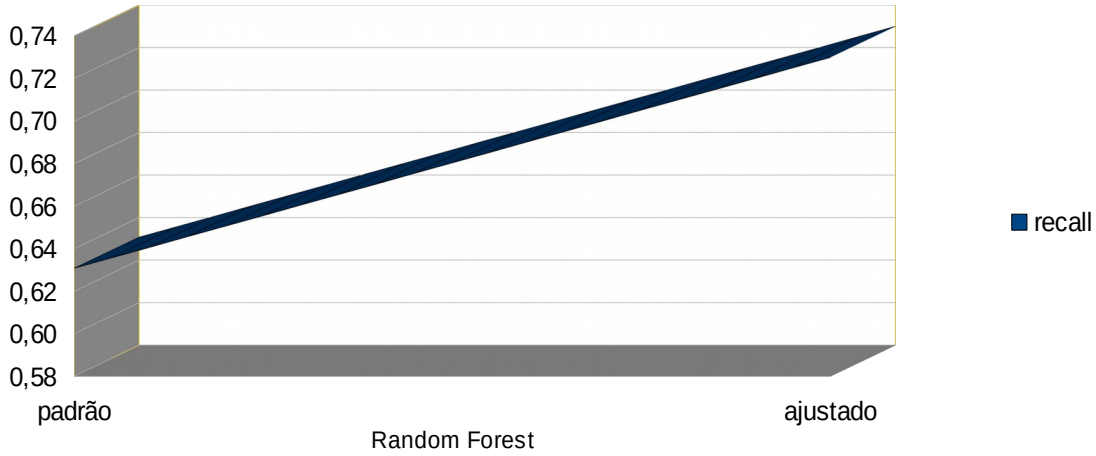


Figura 48: variação da revocação após o ajuste

Veja que esse ganho de revocação justifica o maior gasto computacional, já que permite, em tese, classificar corretamente trechos que realmente possuem risco grave de acidente.

Evidentemente, a busca pela melhora nesses índices de acurácia e, principalmente revocação, deve ser plenamente perseguidos, devido à eliminação dos riscos de acidentes graves. Mas, é cediço que a técnica de aprendizado de máquina é iterativa, de forma os modelos devem constantemente ser melhorados.

Todavia, o modelo Random Forest com 71,30% de acurácia e 73% de revocação pode ser considerado bem-sucedido, para um primeiro trabalho acadêmico.

6.2 Considerações finais

Modelo definido, a título de curiosidade, persisti o modelo para o disco, de forma que ele possa ser utilizado, em momento posterior, com dados novos, ou seja, provenientes de situações ainda não definidas.

```
# Importando a biblioteca joblid, utilizada para persistir o modelo em disco
import joblib
```

```
# Persistindo o melhor modelo em disco.
joblib.dump(gs_rf, 'model.pkl')
```

```
['model.pkl']
```

Figura 49: modelo salvo no disco

Logo após, carreguei somente o modelo a partir do disco para a memória e fiz um teste de classificação, de acordo com os atributos.

```
model = joblib.load('model.pkl')
```

```
print("Atributos do Modelo:\n\nClasses:{}\nEstimator:{}".format(model.classes_,model.estimator))
```

```
Atributos do Modelo:

Classes:[0 1]
Estimator:Pipeline(steps=[('scl', StandardScaler()), ('clf', RandomForestClassifier())])
```

Figura 50: modelo carregado

A última etapa consiste no teste do modelo, uma simulação, situação em que são informados os de atributos e teríamos a saída a classe do algoritmo, sendo 0(sem risco grave) ou 1(com risco grave).

```
teste = np.array([[1,425,16,116.0,3175,1,0,3,0,7,2,1,1,2,7]])
```

```
model.predict(teste)
```

```
array([0], dtype=int64)
```

Figura 51: teste de classificação

Evidentemente, para uma aplicação, seria necessário uma série de readaptações para transformar a classificação de forma amigável a um terceiro.

No âmbito do escopo desse trabalho, a ideia seria utilizar o modelo de classificação em duas vertentes:

- **Governamental**: políticas públicas para propor maior segurança em determinados trechos das rodovias após a utilização do modelo de classificação.
- **Usuário final:** alguma função dentro de aplicativos de localização(GoogleMaps ou Waze, por exemplo) que informassem ao motorista um alerta quando ele estivesse em determinado trecho, sob aquelas características(atributos).

Por fim, acredito que o objetivo do trabalho foi atingido. Efetivamente, foi possível realizar um projeto de Aprendizado de Máquina, desde a definição do problema, incluindo coleta e tratamento de dados, até a aplicação de diferentes modelos com o fito de classificar os trechos das rodovias brasileiras com maior índice de acidentes.

Considero o tema escolhido como bem relevante e capaz de ser abstraído para diversos campos de atuação, já que tratou-se de datasets relacionais, com boa quantidade de registros, sendo possível ainda uni-los com diferentes técnicas. A definição do atributo alvo(classe) pode ser algo trivial em alguns casos, entretanto, neste estudo essa definição surgiu a partir do conhecimento dos atributos e de como eles poderiam gerar algum valor a um modelo de aprendizado de máquina. Após o surgimento da classe, verificou que o modelo dos algoritmos seria de classificação. Percebi também a importância do conhecimento acerca dos modelos de aprendizagem de máquina, ou seja, do funcionamento em torno de como eles efetivamente “aprendem” e, aliado a isso, a importância dos **parâmetros** que cada modelo possui, sendo significativo o teste desses parâmetros na busca da configuração ótima do modelo(GridSearch).

Abaixo, um fluxo de trabalho(workflow) baseado no modelo Canvas proposto por Vasandani, para resumir todo o trabalho:

Título: **Classificação do Risco de Acidentes nas Rodovias Federais Brasileiras**		
1 – Definição do problema: Há um número significativo de acidentes nas rodovias federais brasileiras. A proporção de mortos e feridos nesses acidentes no Brasil é uma das maiores do mundo. Isso é tratado como calamidade pública. Praticamente todos os acidentes e as características que os envolvem são registrados pela PRF. A ideia é verificar a possibilidade de um algoritmo de aprendizado de máquina ajudar a diminuir esses números.	**2 – Resultados e previsões** O registro dos acidentes nos boletins de ocorrência identificam inúmeros atributos que podem ser utilizados como variáveis preditoras, tais como horário do acidente, fase do dia, condições climáticas, localização do acidente, tipo do veículo. A variável alvo, chamada de 'risco' de classe binária, indica se determinado trecho haverá risco de um acidente ser grave(True) ou não(False).	**3 – Fontes de dados** Os dois datasets utilizados foram retirados do site governamental da PRF e representam os acidentes de trânsito ocorridos em 2019. Ambos provém da mesma base, mas com colunas e registros diferentes.
4 – Escolha do modelo: Como a classe do estudo é discreta e binária, serão usados modelos de **classificação**. Utilizou-se três algoritmos, foram eles: – Support Vector Macchine – K-Nearest Neighbor – Random Forest Os algoritmos foram testados em sua forma padrão e, logo após, com a utilização da ferramenta Grid Search(ajustado).	**5 – Métricas de avaliação** Para este trabalho específico as duas métricas utilizadas nos modelos de aprendizagem de máquina foram a Acurácia e a Revocação(recall). O Random Forest ajustado foi o modelo com maior performance, sendo o escolhido.	**6 – Preparação dos dados** Pelas características dos datasets, não foi observado dados duplicados e um número quase insignificante de dados ausentes. A classe é resultante de uma combinação de duas colunas do dataset original. Atributos desnecessários e redundantes foram removidos .

7. Links

Link para o vídeo: https://www.youtube.com/watch?v=egXy1cR_840

Link para o repositório: https://github.com/edurocha81/TCC-CIENCIA-DE-DADOS-PUCMG

REFERÊNCIAS

[1] Oracle.com. Big data definido. Disponível em: https://www.oracle.com/br/big-data/what-is-big-data/#:~:text=Simplificando%2C%20big%20data%20%C3%A9%20um,simplesmente%20n%C3%A3o%20consegue%20gerenci%C3%A1%2Dlos. Acesso em 09/04/2021.

[2] Wikipedia. Big Data. Disponível em: https://pt.wikipedia.org/wiki/Big_data. Acesso em 09/04/2021.

[3] Ministério da Infraestrutura. Disponível em https://www.gov.br/infraestrutura/pt-br/assuntos/transporte-terrestre/rodovias-federais/rodovias-federais-informacoes-gerais-sistema-federal-de-viacao. Acesso em 09/04/2021.

[4] Confederação Nacional do Transporte. Disponível em https://www.cnt.org.br/agencia-cnt/como-funciona-nomenclatura-rodovias-federais . Acesso em 09/04/2021.

[5] Departamento de Polícia Rodoviária Federal. Disponível em https://portal.prf.gov.br/dados-abetos-dicionario-acidentes . Acesso em 09/04/2021.

[6] Medium. Disponível em https://medium.com/machina-sapiens/algoritmos-de-aprendizagem-de-m%C3%A1quina-qual-deles-escolher-67040ad68737. Acesso em 20/04/2021.

[7] IBM. Disponível em https://www.ibm.com/br-pt/analytics/machine-learning?p1=Search&p4=43700052630834837&p5=b&gclid=Cj0KCQjwyZmEBhCpARIsALIzmnIWnjUwCUtWihnJIuifgHqkVWdq47yPVnxTVsISFqBQb9a_yOhzOM8aAuc3EALw_wcB&gclsrc=aw.ds. Acesso em 20/04/2021.

[8] Medium. Disponível em https://medium.com/turing-talks/turing-talks-12-classifica%C3%A7%C3%A3o-por-svm-f4598094a3f1. Acesso em 21/04/2021.

[9] Wikimedia Commons. Acessado no link https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_(SVG).svg em 21/04/2021.

[10] Wikipedia. Acesso: https://pt.wikipedia.org/wiki/M%C3%A1quina_de_vetores_de_suporte , em 21/04/2021

[11] Vooo – Insights . Em https://www.vooo.pro/insights/fundamentos-dos-algoritmos-de-machine-learning-com-codigo-python-e-r/ . Acesso em 21/04/2021.

[12] ICHI.PRO. Em https://ichi.pro/pt/como-construir-knn-do-zero-em-python-103503218478034. Acesso em 21/04/2021.

[13] Utilização De Classificador Random Forest Na Detecção De Falhas Em Máquinas Rotativas. Em http://www.monografias.poli.ufrj.br/monografias/monopoli10015019.pdf , acesso 21/04/2021.

[14] Medium. Acesso: https://medium.com/machina-sapiens/o-algoritmo-da-floresta-aleat%C3%B3ria-3545f6babdf8, em 21/04/2021.

[15] Medium. Acesso: https://medium.com/databootcamp/pipeline-da-scikit-learn-e71c79ec744b, em 24/04/2021.

[16] Minerando dados. Acesso: https://minerandodados.com.br/pipelines-machine-learning / , em 24/04/2021.

[17] Sikit learn. Acesso: https://scikit-learn.org/stable/modules/grid_search.html#grid-search , em 25/04/2021

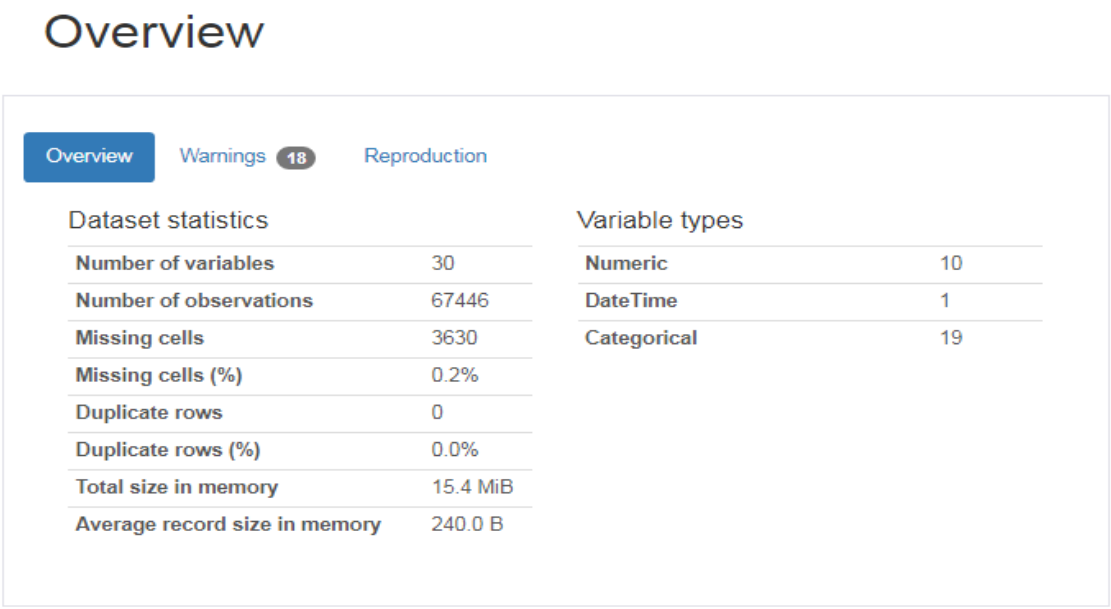
APÊNDICE

Programação/Scripts

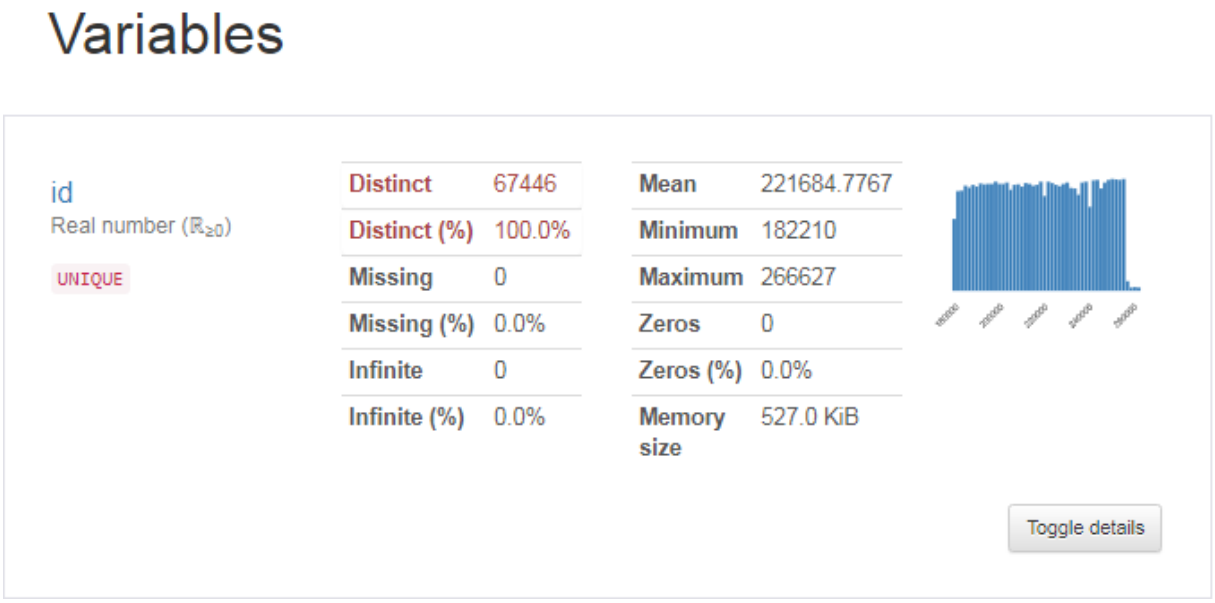
Todos os scripts e os datasets encontram-se no repositório do GitHub

Gráficos

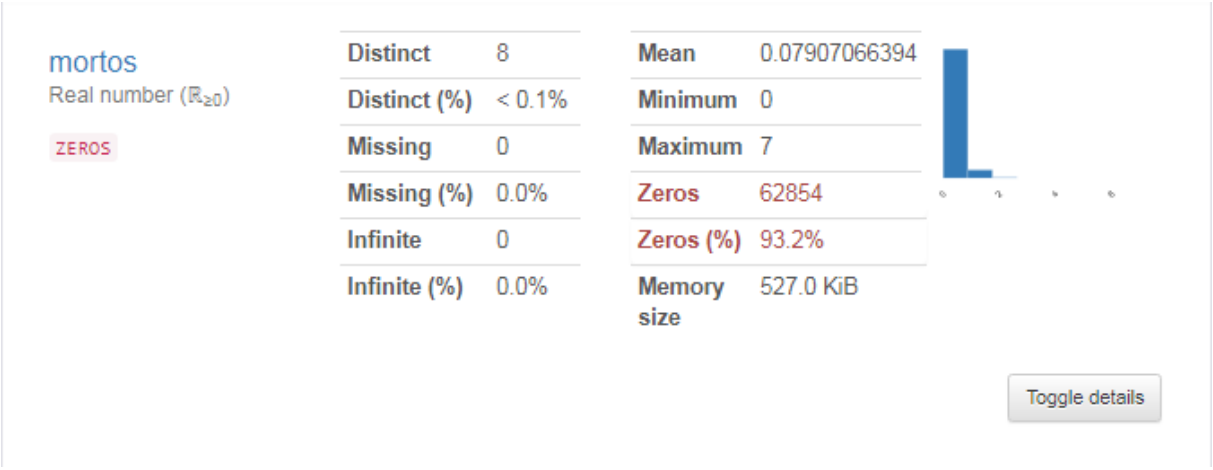
Apresento alguns gráficos e figuras que são gerados diretamente de um recurso do Pandas chamado **Pandas Profiling**. O comando encontra-se no script. Ele dá uma noção geral e inicial do dataset "**datatran2019"** explicando, de forma gráfica e bastante amigável as principais características do dataset.



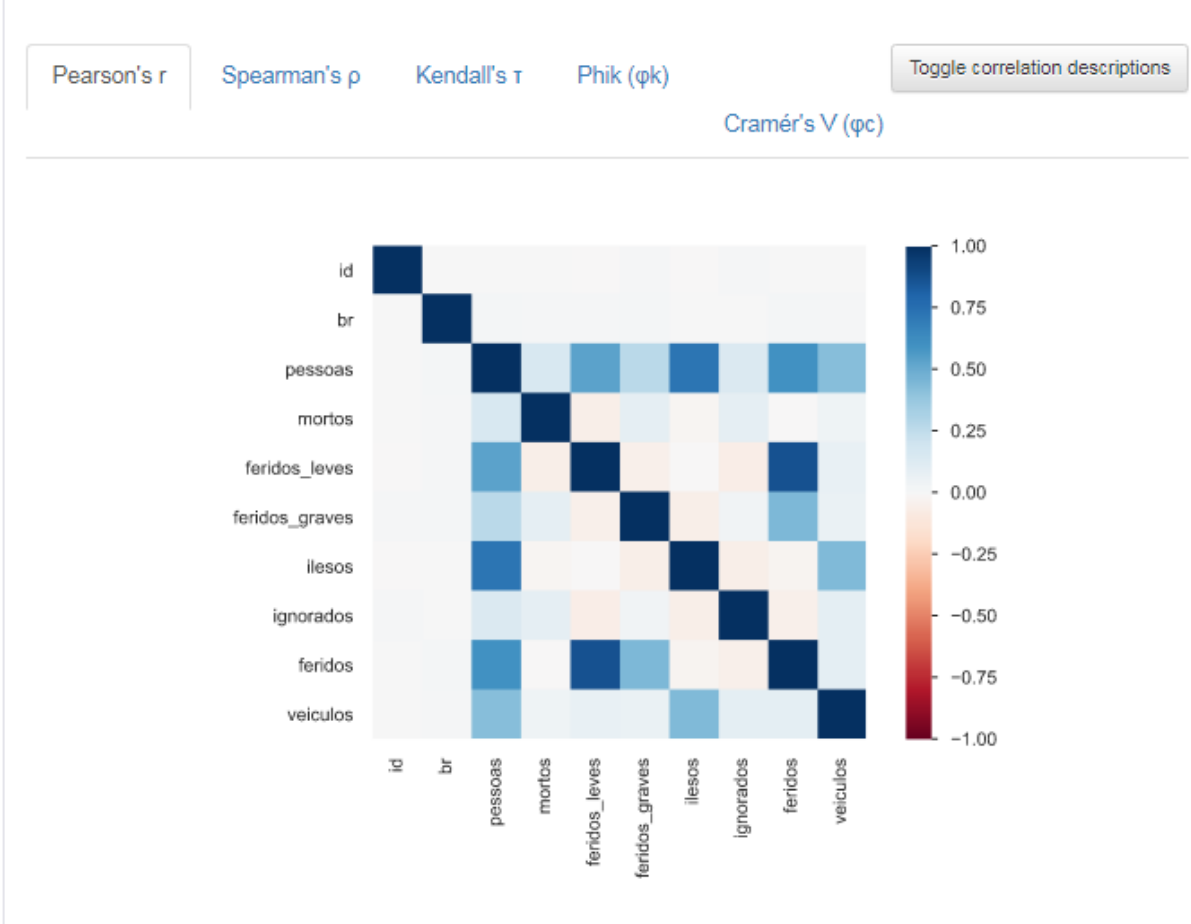
Visão global do Dataset



Apresentação das variáveis



Variável “mortos”



Correlações

Valores ausentes