

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
PÓS-GRADUAÇÃO LATO SENSU EM CIÊNCIA DE DADOS E BIG DATA

DANIEL ELITON COSTA

Análise sobre os acidentes nas rodovias federais do Brasil entre os anos de 2017 a
2020

Belo Horizonte - MG
2021

DANIEL ELITON COSTA

Análise sobre os acidentes nas rodovias federais do Brasil entre os anos de 2017 a
2020

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 - MG
2021

LISTA DE ILUSTRAÇÕES

Figura 1 — Coleta de dados PRF (Agrupados por ocorrência)	9
Tabela 1 — Descrição dos campos/colunas dos datasets df_ocorrencias_XXXX "Agrupados por ocorrência"	9
Figura 2 — Coleta de dados PRF (Agrupados por pessoa)	11
Tabela 2 — Descrição dos campos/colunas dos datasets df_pessoas_XXXX "Agrupados por pessoa"	11
Figura 3 — Coleta de dados IBGE	12
Tabela 3 — Descrição dos campos/colunas dos dataset de IBGE	12
Figura 4 — Coleta de dados ANBIMA	13
Tabela 4 — Coleta dos dados AMBIMA	13
Figura 5 — Junção dos dataframes ocorrências	14
Figura 6 — Verificando dados faltosos em df_ocorrencias	14
Figura 7 — Verificação de tipos de dados de df_ocorrencias	15
Figura 8 — Remoção de nulos e duplicados	15
Figura 9 — Conversão de tipos de dados e correção de valores	16
Figura 10 — Criação da coluna "Classificacao"	16
Figura 11 — Junção dos dataframes pessoas	17
Figura 12 — Verificando dados faltosos em df_pessoas	17
Figura 13 — Resumo estatístico dos dados	17
Figura 14 — Limpeza de df_pessoas	18
Figura 15 — Conversão de tipos de dados	19
Figura 16 — Verificando dados faltosos em df_IBGE	19
Figura 17 — Verificação de tipos de dados de df_IBGE	20
Figura 18 — Tratamento de df_IBGE	20
Figura 19 — Verificando dados faltosos em df_feriados	21
Figura 20 — Verificação de tipos de dados de df_feriados	21
Figura 21 — Remoção da coluna "Dia da Semana"	21
Figura 22 — Junção de todos os dataframes	22
Figura 23 — Verificando dados faltosos em df	22
Figura 24 — Tratamento de df_feriado	22
Figura 25 — Correção da coluna dia_semana	23
Figura 26 — Correção da coluna "sexo"	23
Figura 27 — Salvar dados normalizados	23
Figura 28 — Quantidade de acidentes por ano	24
Figura 29 — Quantidades de acidentes por dia da semana	25
Figura 30 — Quantidades de acidentes por tipo de acidentes	26

Figura 31 — Quantidade de acidentes por fase do dia	27
Figura 32 — Fases do dia e Classificação dos Acidentes.....	28
Figura 33 — Quantidade de Acidentes por estado	29
Figura 34 — Mapa de calor - Estados e Proporção da Classificação dos Acidentes	30
Figura 35 — Quantidade de Acidentes por hora	31
Figura 36 — Condições meteorológicas dos acidentes	32
Figura 37 — Mapa de calor - Condições Meteorológicas e Classificação dos Acidentes	32
Figura 38 — BR com maiores índices de acidentes	33
Figura 39 — Quantidade de pessoas envolvido em acidentes por sexo.....	34
Figura 40 — Acidentes por sexo e classificação dos acidentes	35
Figura 41 — Mapa de calor - Sexo e Classificação dos acidentes	36
Figura 42 — Rank dos 10 municípios com maior número de acidentes	37
Figura 43 — Rank das regiões com maior número de acidentes	38
Figura 44 — Rank das microrregiões com maior número de acidentes	39
Figura 45 — Rank das mesorregiões com maior número de acidentes	40
Figura 46 — Rank dos feriados com maior número de acidentes.....	41
Figura 47 — Mapa de calor de acidentes do Brasil	42
Figura 48 —	43
Figura 49 — Treinar modelo Árvores Aleatórias	44
Figura 50 — Configuração de parâmetro que forneceu os melhores resultados ..	44
Figura 51 — Predições de Floresta Aleatória	45
Figura 52 — Acurácia Floresta Aleatória	45
Figura 53 — Matriz de confusão Floresta Aleatória.....	46
Figura 54 — Principais métricas de classificação Floresta Aleatória	46
Figura 55 — Treinar modelo K-Means	47
Figura 56 — Configuração de parâmetro que forneceu os melhores resultados ..	47
Figura 57 — Predições de K-Means	47
Figura 58 — Acurácia K-Means	48
Figura 59 — Matriz de confusão K-Means	48
Figura 60 — Principais métricas de classificação K-Means	49
Figura 61 — Configuração de parâmetro que forneceu os melhores resultados ..	49
Figura 62 — Predições de regressão logística	49
Figura 63 — Acurácia regressão logística	50
Figura 64 — Matriz de confusão Regressão logística	50
Figura 65 — Principais métricas de classificação Regressão logística	51
Figura 66 — Modelo de Canvas para projetos de Machine Learning	52

LISTA DE ABREVIATURAS E SIGLAS

ANBIMA	Associação Brasileira das Entidades dos Mercados Financeiro e de Capitais
API	Application Programming Interface
IBGE	Instituto Brasileiro de Geografia e Estatística
JSON	JavaScript Object Notation
OMS	Organização Mundial da Saúde
PRF	Polícia Rodoviária Federal
UF	Unidade Federativa
OPS	Organização Pan-Americana de Saúde
REST	Representational State Transfer

SUMÁRIO

1	INTRODUÇÃO	7
1.1	CONTEXTUALIZAÇÃO	7
1.2	O PROBLEMA PROPOSTO	7
2	COLETA DE DADOS	8
2.1	PRF	8
2.1.1	Agrupados por ocorrência	8
2.1.2	Agrupados por pessoa	10
2.2	IBGE	11
2.3	ANBIMA	12
3	PROCESSAMENTO/TRATAMENTO DOS DADOS	14
3.1	TRATAMENTO DE DF_OCORRENCIAS	14
3.2	TRATAMENTO DE DF_PESSOAS	16
3.3	TRATAMENTO DE DF_IBGE	19
3.4	TRATAMENTO DE DF_FERIADOS	20
3.5	CRIAÇÃO E TRATAMENTO DE DF	21
4	ANÁLISE/EXPLORAÇÃO DOS DADOS	24
5	CRIAÇÃO DE MODELOS DE ML	43
5.1	FLORESTA ALEATÓRIA (RANDOM FOREST)	43
5.2	K-MEANS	46
5.3	REGRESSÃO LOGÍSTICA	49
6	INTERPRETAÇÃO DOS RESULTADOS	52
7	LINKS	54
	REFERÊNCIAS	55

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Estradas são consideradas uma das formas mais comuns de viajar e transportar mercadorias no Brasil. No entanto, as estradas brasileiras são consideradas extremamente perigosas e inseguras. Segundo a OPS (OPS, 2019) mostra que o Brasil está na 9^a posição entre os países com maior número de mortes nas Américas, atrás apenas de Santa Lucia, República Dominicana, Venezuela, Belize, Guiana, Paraguai, El Salvador e Equador.

Existem três principais categorias de estradas no Brasil: estradas locais, estaduais e federais. A Polícia Rodoviária Federal também conhecida como PRF é responsável pelo monitoramento das estradas federais, que mantêm registros de tempo, local e tipo de acidentes em um conjunto de dados que está disponível no site público da própria PRF (PRF, 2021).

Nos feriados o fluxo de veículos aumenta o que pode aumentar os números de acidentes. A Associação Brasileira das Entidades dos Mercados Financeiro e de Capitais também conhecida como ANBIMA (ANBIMA, 2021), mantém registros de todos os feriados nacionais de 2001 até 2078, dados estes que foi usado para ver o feriado nacional com maior número de acidentes.

No Cidades (IBGE - CIDADES, 2021) é o sistema agregador de informações do Instituto Brasileiro de Geografia e Estatística mais conhecido como IBGE (IBGE, 2021) sobre municípios e estados do Brasil. Nele se encontra pesquisas e infográficos do IBGE. Para este trabalho se utilizou a API REST pública para coletar dados dos municípios, regiões, microrregiões e mesorregiões.

1.2 O PROBLEMA PROPOSTO

O objetivo principal é tentar prever um acidente com as seguintes classificações, sem vítimas, vítimas feridas e vítimas fatais. Consideraremos informações sobre a hora do acidente (momento do dia, dia da semana e etc), o local em que ocorreu o acidente (número da via, quilômetro, estado etc.) e as características do acidente (como causa do acidente e tipo de acidente).

Se pudermos prever, com certa confiança, o tipo de vítimas com base apenas em o local e a hora do acidente, pode-se supor que haja um problema nas estradas, e pode-se prever as áreas mais perigosas. Mesmo que isso não seja possível, as previsões podem ajudar a identificar as áreas mais problemáticas e ajudar nas medidas de prevenção.

2 COLETA DE DADOS

Para coleta dos dados foi utilizado a ferramenta google colaboratory (COLAB, 2021) popularmente conhecida como *colab*, um serviço de nuvem gratuito hospedado pela *google* para incentivar a pesquisa de aprendizado de máquina e inteligência artificial. Similar ao famoso *jupyter* notebook, o colab é uma lista de células que podem conter textos explicativos ou códigos executáveis na linguagem *python* e suas respectivas saídas.

Para tratamento dos dados foi usado dataframe que faz parte do pacote do *pandas* (PANDAS, 2021) desenvolvido na linguagem *python*, que é largamente usado em machine learning e inteligência artificial, ele fornece ferramentas com grande poder para manipulação de dados.

Para enriquecer dos dados coletados da PRF, foi utilizado a API pública do IBGE consumindo o endpoint do cadastro de cidades, também com o mesmo propósito foi utilizado um dataset sobre feriados nacionais disponibilizado pela ANBIMA que foi baixado diretamente do site do mesmo diretamente para o colab.

2.1 PRF

Os dados de acidentes coletados são referentes aos acidentes em rodovias federais agrupados por ocorrência e agrupados por pessoa, todos os datasets foram baixados do site público de dados da PRF e hospedados no repositório do projeto.

2.1.1 Agrupados por ocorrência

Este grupo de datasets tem como objetivo trazer dados sobre os acidentes com o foco na ocorrência, não informando dados sobre as vítimas, ao importar do repositório para o dataframe como mostra a Figura 1, já foi incluso as colunas que serão usadas no projeto, eliminando assim eliminação de campos na etapa de tratamento.

Figura 1 — Coleta de dados PRF (Agrupados por ocorrência)

```

cols = ['id', 'data_inversa', u'dia_semana', 'horario', u'municipio', 'uf', 'br', 'km', u'causa_acidente',
        u'tipo_acidente', u'fase_dia', u'condicao_meteorologica', u'tracado_via', u'uso_solo',
        u'classificacao_acidente', 'pessoas', 'mortos', 'feridos_leves', 'tipo_pista',
        'feridos_graves', 'ilesos', 'feridos', 'veiculos', u'sentido_via', 'latitude', 'longitude']

url_oco_2017 = 'https://github.com/callacius/TCC_Puc_mg/blob/main/datasets/baixados/PRF/datatranc2017.zip?raw=true'
url_oco_2018 = 'https://github.com/callacius/TCC_Puc_mg/blob/main/datasets/baixados/PRF/datatranc2018.zip?raw=true'
url_oco_2019 = 'https://github.com/callacius/TCC_Puc_mg/blob/main/datasets/baixados/PRF/datatranc2019.zip?raw=true'
url_oco_2020 = 'https://github.com/callacius/TCC_Puc_mg/blob/main/datasets/baixados/PRF/datatranc2020.zip?raw=true'

df_ocorrencias_2017 = pd.read_csv(url_oco_2017, sep=';', encoding="ISO-8859-1", compression='zip', usecols=cols)
df_ocorrencias_2018 = pd.read_csv(url_oco_2018, sep=';', encoding="ISO-8859-1", compression='zip', usecols=cols)
df_ocorrencias_2019 = pd.read_csv(url_oco_2019, sep=';', encoding="ISO-8859-1", compression='zip', usecols=cols)
df_ocorrencias_2020 = pd.read_csv(url_oco_2020, sep=';', encoding="ISO-8859-1", compression='zip', usecols=cols)

```

Fonte: O autor (2021)

A tabela 1 apresenta a descrição dos campos/colunas do dataset da PRF agrupado por ocorrência e que integram a amostra de dados.

Tabela 1 — Descrição dos campos/colunas dos datasets df_ocorrencias_XXXX "Agrupados por ocorrência"

(continua)

Nome da coluna/campo	Descrição	Tipo
id	Variável com valores numéricos, representando o identificador do acidente.	Numérico
data_inversa	Data da ocorrência no formato dd/mm/aaaa.	Data
dia_semana	Dia da semana da ocorrência. Ex.: Segunda, terça, etc.	Texto
horario	Horário da ocorrência no formato hh:mm:ss.	Hora
uf	Unidade da Federação. Ex.: MG, PE, DF, etc.	Texto
br	Variável com valores numéricos, representando o identificador da BR do acidente.	Texto
km	Identificação do quilômetro onde ocorreu o acidente, com valor mínimo de 0,1 km e com a casa decimal separada por ponto.	Numérico
municipio	Nome do município de ocorrência do acidente	Texto
causa_acidente	Identificação da causa principal do acidente. Neste conjunto de dados são excluídos os acidentes com a variável causa principal igual a “Não”.	Texto
tipo_acidente	Identificação do tipo de acidente. Ex.: Colisão frontal, Saída de pista, etc. Neste conjunto de dados são excluídos os tipos de acidentes com ordem maior ou igual a dois. A ordem do acidente demonstra a sequência cronológica dos tipos presentes na mesma ocorrência.	Texto
fase_dia	Fase do dia no momento do acidente. Ex. Amanhecer,	Texto

Tabela 1 — Descrição dos campos/colunas dos datasets df_ocorrencias_XXXX "Agrupados por ocorrência"
(conclusão)

Nome da coluna/campo	Descrição	Tipo
	Pleno dia, etc.	
sentido_via	Sentido da via considerando o ponto de colisão: Crescente e decrescente.	Texto
condicao_meteorologica	Condição meteorológica no momento do acidente: Céu claro, chuva, vento, etc.	Texto
tipo_pista	Tipo da pista considerando a quantidade de faixas: Dupla, simples ou múltipla.	Texto
tracado_via	Descrição do traçado da via.	Texto
uso_solo	Descrição sobre as características do local do acidente: Urbano=Sim; Rural=Não.	Texto
pessoas	Total de pessoas envolvidas na ocorrência.	Numérico
mortos	Total de pessoas mortas envolvidas na ocorrência.	Numérico
feridos_leves	Total de pessoas com ferimentos leves envolvidas na ocorrência.	Numérico
feridos_graves	Total de pessoas com ferimentos graves envolvidas na ocorrência.	Numérico
ilesos	Total de pessoas ilesas envolvidas na ocorrência.	Numérico
feridos	Total de pessoas feridas envolvidas na ocorrência (é a soma dos feridos leves com os graves).	Numérico
veiculos	Total de veículos envolvidos na ocorrência.	Numérico
latitude	Latitude do local do acidente em formato geodésico decimal.	Geográfico
longitude	Longitude do local do acidente em formato geodésico decimal.	Geográfico

Fonte: O autor (2021)

2.1.2 Agrupados por pessoa

Este grupo de datasets tem como objetivo trazer dados sobre os acidentes com o foco nas pessoas envolvidas, trazendo dados como idade, estado físico entre outros, ao importar do repositório para o dataframe como mostra a Figura 2, já foi incluso as colunas que serão usadas no projeto, eliminando assim este processo na etapa de tratamento.

Figura 2 — Coleta de dados PRF (Agrupados por pessoa)

```

cols = ['id', 'pesid', 'uf', 'u'id_veiculo', u'tipo_veiculo', u'marca',
        'ano_fabricacao_veiculo', u'tipo_envolvido', u'estado_fisico', 'idade', u'sexo']

url_pes_2017 = 'https://github.com/callacius/TCC_Puc_mg/blob/main/datasets/baixados/PRF/acidentes2017.zip?raw=true'
url_pes_2018 = 'https://github.com/callacius/TCC_Puc_mg/blob/main/datasets/baixados/PRF/acidentes2018.zip?raw=true'
url_pes_2019 = 'https://github.com/callacius/TCC_Puc_mg/blob/main/datasets/baixados/PRF/acidentes2019.zip?raw=true'
url_pes_2020 = 'https://github.com/callacius/TCC_Puc_mg/blob/main/datasets/baixados/PRF/acidentes2020.zip?raw=true'

df_pessoas_2017 = pd.read_csv(url_pes_2017, sep=';', encoding="ISO-8859-1", compression='zip', usecols=cols)
df_pessoas_2018 = pd.read_csv(url_pes_2018, sep=';', encoding="ISO-8859-1", compression='zip', usecols=cols)
df_pessoas_2019 = pd.read_csv(url_pes_2019, sep=';', encoding="ISO-8859-1", compression='zip', usecols=cols)
df_pessoas_2020 = pd.read_csv(url_pes_2020, sep=';', encoding="ISO-8859-1", compression='zip', usecols=cols)

```

Fonte: O autor (2021)

A tabela 2 apresenta a descrição dos campos/colunas do dataset da PRF agrupado por pessoas e que integram a amostra de dados.

Tabela 2 — Descrição dos campos/colunas dos datasets df_pessoas_XXXX "Agrupados por pessoa"

Nome da coluna/campo	Descrição	Tipo
id	Variável com valores numéricos, representando o identificador do acidente.	Numérico
pestd	Variável com valores numéricos, representando o identificador da pessoa envolvida.	Numérico
id_veiculo	Variável com valores numéricos, representando o identificador do veículo envolvido.	Numérico
tipo_veiculo	Tipo de veículo conforme Art.96 do Código de Trânsito Brasileiro.	Texto
marca	Descrição da marca do Veículo.	Texto
ano_fabricacao_veiculo	Ano de fabricação do veículo, formato aaaa	Numérico
tipo_envolvido	Tipo do envolvido conforme sua participação no evento.	Texto
idade	Idade do envolvido	Numérico
sexo	Sexo do envolvido.	Texto

Fonte: O autor (2021)

2.2 IBGE

Os dados de cidades foram coletados da API pública do IBGE onde se passa o parâmetro {UF} e API retorna um json com uma lista com todos os dados dos municípios que fazem parte da UF. Para importar diretamente da API para um dataframe se usou uma biblioteca json_normalize do python que gera um dataframe diretamente do json retornado como mostra na figura 3, com esta biblioteca foi

possível passar somente as colunas que vão ser usado no projeto, eliminando assim este processo na etapa de tratamento.

Figura 3 — Coleta de dados IBGE

```
cols = [u'id', u'nome', u'microrregiao.mesorregiao.UF.sigla', u'regioao-imediata.regioao-intermediaria.UF.regioao.id',
        u'regioao-imediata.regioao-intermediaria.UF.regioao.sigla', u'regioao-imediata.regioao-intermediaria.UF.regioao.nome',
        u'microrregiao.id', u'microrregiao.nome', u'microrregiao.mesorregiao.id', u'microrregiao.mesorregiao.nome']

ufs = ['AC','AL','AP','AM','BA','CE','DF','ES','GO','MA','MT','MS','MG','PA','PB','PR','PE','PI','RJ','RN','RS',
       'RO','RR','SC','SP','SE','TO']

df_ibge = pd.DataFrame(columns=cols)

for uf in ufs:
    url = f"https://servicodados.ibge.gov.br/api/v1/localidades/estados/{uf}/municpios"
    data = requests.get(url)
    df = pd.DataFrame(json_normalize(data.json()),columns=cols)
    df_ibge = pd.concat([df_ibge, df])
```

Fonte: O autor (2021)

A tabela 3 apresenta a descrição dos campos/colunas do dataset do IBGE e que integram a amostra de dados.

Tabela 3 — Descrição dos campos/colunas dos dataset de IBGE

Nome da coluna/campo	Descrição	Tipo
id	Identificador IBGE	Numérico
nome	Nome do município	Texto
microrregiao.mesorregiao.UF.sigla	Unidade federativa	Texto
regiao-imediata.regioao-intermediaria.UF.regioao.id	Identificador da região	Numérico
regiao-imediata.regioao-intermediaria.UF.regioao.sigla	Sigla da região	Texto
regiao-imediata.regioao-intermediaria.UF.regioao.nome	Nome da região	Texto
microrregiao.id	Identificador da microrregião	Numérico
microrregiao.nome	Nome da microrregião	Texto
microrregiao.mesorregiao.id	Identificador da mesorregião	Numérico
microrregiao.mesorregiao.nome	Nome da mesorregião	Texto

Fonte: O autor (2021)

2.3 ANBIMA

Os dados de feriados nacionais foram coletados diretamente do site da ANBIMA no formato xls que foi importado diretamente para um dataframe como mostra a Figura 4.

Figura 4 — Coleta de dados ANBIMA

```
[ ] url = 'https://www.anbima.com.br/feriados/arqs/feriados_nacionais.xls'  
df_feriados = pd.read_excel(url, nrows=936)
```

Fonte: O autor (2021)

A tabela 4 apresenta a descrição dos campos/colunas do dataset da ANBIMA e que integram a amostra de dados.

Tabela 4 — Coleta dos dados AMBIMA

Nome da coluna/campo	Descrição	Tipo
Data	Data do feriado	data
Dia da Semana	Dia da semana	Texto
Feriado	Nome do feriado	Texto

Fonte: O autor (2021)

3 PROCESSAMENTO/TRATAMENTO DOS DADOS

O tratamento dos dados foi feito separadamente por fontes de dados e a junção a posteriori. Este tratamento de dados foi dividido em ocorrências, pessoas, IBGE e feriados.

3.1 TRATAMENTO DE DF_OCORRENCIAS

Antes de começar o tratamento dos dados de ocorrências, foi acrescentado a coluna ano conforme o ano do arquivo com extensão csv coletado diretamente do site público da PRF, depois foi feito a junção de todos os dataframes, *df_ocorrecia_2017*, *df_ocorrecia_2018*, *df_ocorrecia_2019* e *df_ocorrecia_2020* em um único dataframe *df_ocorrecias*.

Figura 5 — Junção dos dataframes ocorrências

```
# Acrescentar o ano conforme o ano coletado no dataset
df_ocorrecias_2017['ano'] = 2017
df_ocorrecias_2018['ano'] = 2018
df_ocorrecias_2019['ano'] = 2019
df_ocorrecias_2020['ano'] = 2020

# Concatenando os datasets para formar um unico dataset "df_ocorrecias"
df_ocorrecias = pd.concat([df_ocorrecias_2017,
                           df_ocorrecias_2018,
                           df_ocorrecias_2019,
                           df_ocorrecias_2020], sort = False)
```

Fonte: O autor (2021)

Para começarmos o tratamento de *df_ocorrecias*, conforme pode ser verificado na Figura 6, verificamos a existência de dados faltosos, pelo retorno da célula do colab pode se constatar que existe 513 registros faltosos nas colunas br e km.

Figura 6 — Verificando dados faltosos em *df_ocorrecias*

```
# Verificando os dados faltosos
df_ocorrecias.isnull().sum().sort_values(ascending=False)
```

br	513
km	513
ano	0
sentido_via	0

Fonte: O autor (2021)

Agora vamos verificar os tipos de dados das colunas conforme mostra na Figura 7 para podermos analisar qual coluna vai necessitar de conversão.

Figura 7 — Verificação de tipos de dados de df_ocorrencias

```
# Verificando os tipos
df_ocorrencias.dtypes
```

id	float64
data_inversa	object
dia_semana	object
horario	object
uf	object

Fonte: O autor (2021)

Logo após verificar os dados faltosos e tipos de dados, vamos começar o tratamento de df_ocorrencias, conforme mostrado na Figura 8 começamos eliminando os registros com valores nulos e duplicados, e como mostra na figura 9 foi feito a conversão dos tipos de dados de algumas colunas, também foi alterado a grafia dos valores da coluna municípios segundo a base de dados do IBGE, coluna esta essencial para junção com df_IBGE.

Figura 8 — Remoção de nulos e duplicados

```
antesLinhas = df_ocorrencias.shape[0]
antesColunas = df_ocorrencias.shape[1]

df_ocorrencias = df_ocorrencias[df_ocorrencias['br'].notna()]
df_ocorrencias = df_ocorrencias[df_ocorrencias['km'].notna()]

# Eliminar dados duplicados
df_ocorrencias.drop_duplicates()

# Reindexar os registros, por conta da remoção das duplicatas
df_ocorrencias.reset_index(drop=True, inplace=True)

print('Quantidade de registros antes da limpeza')
print(f'{antesLinhas} linhas e {antesColunas} colunas.')
print('Quantidade de registros após a limpeza')
print(f'{df_ocorrencias.shape[0]} linhas e {df_ocorrencias.shape[1]} colunas.')
print('Quantidade de registros removidos')
print(f'{antesLinhas - df_ocorrencias.shape[0]} registros.')

Quantidade de registros antes da limpeza
289852 linhas e 27 colunas.
Quantidade de registros após a limpeza
289339 linhas e 27 colunas.
Quantidade de registros removidos
513 registros.
```

Fonte: O autor (2021)

Figura 9 — Conversão de tipos de dados e correção de valores

```
# Correção de vírgula para ponto e alterar os campos que estão como string.
# e devem ser normalizados para float, a fim de utilizá-los em eventuais plots geográficos.
df_ocorrencias['latitude'] = df_ocorrencias['latitude'].str.replace(',', '.')
df_ocorrencias['longitude'] = df_ocorrencias['longitude'].str.replace(',', '.')
df_ocorrencias['km'] = df_ocorrencias['km'].str.replace(',', '.')

df_ocorrencias['latitude'] = df_ocorrencias['latitude'].astype(float)
df_ocorrencias['longitude'] = df_ocorrencias['longitude'].astype(float)
# Este campo data inversa vamos conversão para datetime, e alterar seu nome para data ficando assim mais legível.
df_ocorrencias['data_inversa'] = df_ocorrencias['data_inversa'].astype('datetime64')
df_ocorrencias.rename(columns={'data_inversa': 'data'}, inplace = True)
# df_ocorrencias['horario'] = df_ocorrencias['horario'].dt.strftime('%H:%M:%S')
df_ocorrencias['horario'] = pd.to_datetime(df_ocorrencias['horario']).dt.strftime('%H:%M:%S', errors='ignore').dt.time
df_ocorrencias['id'] = df_ocorrencias['id'].astype(np.int64)
df_ocorrencias['br'] = df_ocorrencias['br'].astype(np.int64)
df_ocorrencias['km'] = df_ocorrencias['km'].astype(np.float)

# Colocar todos os dados da coluna "municipio" para minúsculo para fazer o left join com dataset "df_ibge"
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8').str.lower()

# Corrigir gafraria de alguns valores da coluna "municipio"
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.replace('muquem de sao francisco', 'muquem do sao francisco')
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.replace('fortaleza do tabocao', 'tabocao')
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.replace('belem de sao francisco', 'belem do sao francisco')
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.replace('santana do livramento', 'santana do livramento')
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.replace('poxoreo', 'poxoreu')
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.replace('santa teresinha', 'santa terezinha')
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.replace('sao sebastiao de lagoa de roca', 'sao sebastiao de lagoa de roca')
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.replace('augusto severo', 'campo grande')
df_ocorrencias['municipio'] = df_ocorrencias['municipio'].str.replace('santana do livramento', 'santana do livramento')
conditions = [
    (df_ocorrencias['municipio'] == 'santa terezinha') & (df_ocorrencias['uf'] == 'PB'),
    (df_ocorrencias['municipio'] == 'santa teresinha') & (df_ocorrencias['uf'] == 'BA'),
    (df_ocorrencias['municipio'] == 'santa teresinha') & (df_ocorrencias['uf'] == 'SC'),
    (df_ocorrencias['municipio'] == 'santa teresinha') & (df_ocorrencias['uf'] == 'PE'),
    (df_ocorrencias['municipio'] == 'santa teresinha') & (df_ocorrencias['uf'] == 'MT')
]
choices = ['santa teresinha', 'santa teresinha', 'santa teresinha', 'santa teresinha', 'santa teresinha']
df_ocorrencias['municipio'] = np.select(conditions, choices, default=df_ocorrencias['municipio'])
```

Fonte: O autor (2021)

Na figura 10, criamos a coluna *classificacao* com valores 0, 1 ou 2 com base na coluna já existente *classificacao_acidente*, onde os valores são *Sem Vítimas*, *Vítimas Feridas* ou *Vítimas Fatais*, e logo em seguida foi criado um arquivo com extensão csv com os dados tratados até o momento com o nome *ocorrencias.csv*.

Figura 10 — Criação da coluna "Classificacao"

```
value = {
    'Sem Vítimas': 0,
    'Com Vítimas Feridas': 1,
    'Com Vítimas Fatais': 2
}
df_ocorrencias['classificacao'] = df_ocorrencias.classificacao_acidente.apply(lambda x: value[x])

df_ocorrencias.to_csv(f"{path}/ocorrencias.csv")
```

Fonte: O autor (2021)

3.2 TRATAMENTO DE DF_PESSOAS

Antes de começar o tratamento de *df_pessoas*, foi feito a junção de todos os dataframes, *df_pessoas_2017*, *df_pessoas_2018*, *df_pessoas_2019* e *df_pessoas_2020* em um único dataframe *df_pessoas* conforme mostrado na Figura 11.

Figura 11 — Junção dos dataframes pessoas

```
# Concatenando os datasets para formar um unico dataset df_pessoas
df_pessoas = pd.concat([df_pessoas_2017,
                       df_pessoas_2018,
                       df_pessoas_2019,
                       df_pessoas_2020], sort = False)
```

Fonte: O autor (2021)

Para começarmos o tratamento do df_ocorrencias, conforme pode ser verificado na Figura 12 e Figura 13, verificamos a existência de dados faltosos e estatísticas descritivas das colunas do tipo integer.

Figura 12 — Verificando dados faltosos em df_pessoas

```
# Verificando os dados faltosos
df_pessoas.isnull().sum().sort_values(ascending=False)

idade          63309
pesid            4
sexo              0
estado_fisico      0
tipo_envolvido      0
id                0
dtype: int64
```

Fonte: O autor (2021)

Figura 13 — Resumo estatístico dos dados

```
# resumo númerico ou estatístico dos dados
df_pessoas.describe().astype(int).T

   count    mean     std    min    25%    50%    75%    max
id    679112  167706  95787     8   84687  169701  250430  352488
pesid  679108  371056 214399     1  183451  372892  556654  785255
idade  615803      40      65     -1      27      37      48     2019
```

Fonte: O autor (2021)

Conforme mostra na figura 14, todos os valores nulos da coluna *idade* foi preenchido com a mediana dos valores da coluna, todos os valores inválidos como idade maior ou igual 100 ou -1 foram removidos, foram eliminados 6076 registros.

Figura 14 — Limpeza de df_pessoas

```

    antesLinhas = df_pessoas.shape[0]
    antesColunas = df_pessoas.shape[1]

    # Os valores null's do dataset, recebem como idade o valor da média das idades
    # (mesmo valor da mediana no caso)
    df_pessoas['idade'].fillna(value= 37, inplace= True)
    # A variável f recebe o index ou posição de todos os registros em que a idade
    # é maior do que 100
    f = df_pessoas[df_pessoas['idade'] >= 100].index
    # Em seguida, é utilizada como máscara para deletar todos os registros
    # selecionados
    df_pessoas.drop(f, axis = 0, inplace = True)
    # Em seguida, todos os registros com a flag -1 no campo idade, tem a mesma
    # substituída por um valor null
    df_pessoas['idade'].replace(-1,np.nan, inplace= True)
    # Que é interpolado visando redistribuir os mesmos no range de idade de forma
    # o mais natural possível
    df_pessoas['idade'].interpolate(inplace= True)

    df_pessoas = df_pessoas[df_pessoas['pesid'].notna()]

    # Eliminar dados duplicados
    df_pessoas.drop_duplicates()

    # Reindexar os registros, por conta da remoção das duplicatas
    df_pessoas.reset_index(drop=True, inplace=True)

    print('Quantidade de registros antes da limpeza')
    print(f'{antesLinhas} linhas e {antesColunas} colunas.')
    print('Quantidade de registros após a limpeza')
    print(f'{df_pessoas.shape[0]} linhas e {df_pessoas.shape[1]} colunas.')
    print('Quantidade de registros removidos')
    print(f'{antesLinhas - df_pessoas.shape[0]} registros.')

```

↳ Quantidade de registros antes da limpeza
679112 linhas e 6 colunas.
Quantidade de registros após a limpeza
673036 linhas e 6 colunas.
Quantidade de registros removidos
6076 registros.

Fonte: O autor (2021)

Como mostra na figura 15 foi feito a conversão dos tipos de dados de algumas colunas e logo em seguida foi criado um arquivo com extensão csv com os dados tratados até o momento com o nome pessoas.csv.

Figura 15 — Conversão de tipos de dados

```
df_pessoas.dtypes

id           float64
pesid        float64
tipo_envolvido   object
estado_fisico    object
idade         float64
sexo          object
dtype: object

# Convertendo dados do dataset df_pessoas
df_pessoas['id'] = df_pessoas['id'].astype(np.int64)
df_pessoas['pesid'] = df_pessoas['pesid'].astype(np.int64)
df_pessoas['idade'] = df_pessoas['idade'].astype(np.int64)

df_pessoas.to_csv(f'{path}/pessoas.csv')
```

Fonte: O autor (2021)

3.3 TRATAMENTO DE DF_IBGE

Para começarmos o tratamento de df_ibge, conforme pode ser verificado na Figura 16 e Figura 17, verificamos a existência de dados faltosos e os tipos de dados das colunas.

Figura 16 — Verificando dados faltosos em df_IBGE

```
# Verificando os dados faltosos
df_ibge.isnull().sum().sort_values(ascending=False)

microrregiao.mesorregiao.nome      0
microrregiao.mesorregiao.id       0
microrregiao.nome                  0
microrregiao.id                   0
regiao-imediata.regiao-intermediaria.UF.regiao.nome 0
regiao-imediata.regiao-intermediaria.UF.regiao.sigla 0
regiao-imediata.regiao-intermediaria.UF.regiao.id   0
microrregiao.mesorregiao.UF.sigla 0
nome                           0
id                            0
dtype: int64
```

Fonte: O autor (2021)

Figura 17 — Verificação de tipos de dados de df_IBGE

df_ibge.dtypes	
id	object
nome	object
microrregiao.mesorregiao.UF.sigla	object
regiao-imediata.regiao-intermediaria.UF.regiao.id	object
regiao-imediata.regiao-intermediaria.UF.regiao.sigla	object
regiao-imediata.regiao-intermediaria.UF.regiao.nome	object
microrregiao.id	object
microrregiao.nome	object
microrregiao.mesorregiao.id	object
microrregiao.mesorregiao.nome	object
dtype: object	

Fonte: O autor (2021)

Conforme mostra na figura 18 foi criado a coluna *município*, coluna esta que será usado para junção com outros dataframes, foi renomeado as colunas deixando assim mais legível e logo em seguida foi criado um arquivo com extensão csv com os dados tratados até o momento do dataframe df_IBGE com o nome ibge.csv.

Figura 18 — Tratamento de df_IBGE

```
# Criação da coluna "município" e copiando os valores de "nome"
df_ibge['municipio'] = df_ibge['nome']

# Renomeando as colunas para ficar mais legível
df_ibge.rename(columns={'id':'ibge',
                       'microrregiao.mesorregiao.UF.sigla':'uf',
                       'microrregiao.id':'microrregiao',
                       'microrregiao.nome':'microrregiao_nome',
                       'microrregiao.mesorregiao.id':'mesorregiao',
                       'microrregiao.mesorregiao.nome':'mesorregiao_nome',
                       'regiao-imediata.regiao-intermediaria.UF.regiao.id':'regiao',
                       'regiao-imediata.regiao-intermediaria.UF.regiao.sigla':'regiao_sigla',
                       'regiao-imediata.regiao-intermediaria.UF.regiao.nome':'regiao_nome'}, inplace = True)

df_ibge['ibge'] = df_ibge['ibge'].astype(np.int64)
df_ibge['regiao'] = df_ibge['regiao'].astype(np.int64)
df_ibge['microrregiao'] = df_ibge['microrregiao'].astype(np.int64)
df_ibge['mesorregiao'] = df_ibge['mesorregiao'].astype(np.int64)

# Normalizando os valores de município retirando acentos para posteriormente
# facilitar o merge com outros datasets
df_ibge['municipio'] = df_ibge['municipio'].str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8').str.lower()
# Remover aspas simples
df_ibge['municipio'] = df_ibge['municipio'].str.replace("'", '')

df_ibge.to_csv(f'{path}/ibge.csv')
```

Fonte: O autor (2021)

3.4 TRATAMENTO DE DF_FERIADOS

Para começarmos o tratamento do df_feriados, conforme pode ser verificado na Figura 19 e Figura 20, verificamos a existência de dados faltosos e os tipos de dados das colunas.

Figura 19 — Verificando dados faltos em df_feriados

```
# Verificando os dados faltos
df_feriados.isnull().sum().sort_values(ascending=False)

Feriado      0
Dia da Semana 0
Data          0
dtype: int64
```

Fonte: O autor (2021)

Figura 20 — Verificação de tipos de dados de df_feriados

```
df_feriados.dtypes

Data           datetime64[ns]
Dia da Semana   object
Feriado         object
dtype: object
```

Fonte: O autor (2021)

Conforme mostra na figura 21 foi removido a coluna *Dia da Semana* por já conter essa coluna no dataframe df_ocorrencias, em seguida foi criado um arquivo com extensão csv com os dados tratados até o momento do dataframe df_feriados com o nome feriados.csv.

Figura 21 — Remoção da coluna "Dia da Semana"

```
# Removendo a coluna "Dia da Semana" por esta informação já conter em outro dataframe
df_feriados.pop('Dia da Semana')
# Renomenado as colunas para lower para padronizar com outros datasets
df_feriados.columns= df_feriados.columns.str.lower()

df_feriados.to_csv(f'{path}/feriados.csv')
```

Fonte: O autor (2021)

3.5 CRIAÇÃO E TRATAMENTO DE DF

Para a criação do dataframe df foi feito a junção de todos os dataframes anteriores conforme vemos abaixo na Figura 22.

Figura 22 — Junção de todos os dataframes

```
# Criando um único dataset
df = pd.merge(df_ocorrencias, df_pessoas)
df = pd.merge(df, df_ibge, on=["municipio","uf"], how='left')
df = pd.merge(df, df_feriados, on=['data'], how='left')
```

Fonte: O autor (2021)

Para começarmos o tratamento do df, conforme pode ser verificado na Figura 23, verificamos a existência de dados faltosos.

Figura 23 — Verificando dados faltosos em df

```
# Verificando os dados faltosos
df.isnull().sum().sort_values(ascending=False)
```

feriado	647873
classificacao_acidente	0
mortos	0
pessoas	0
uso_solo	0
tracado_via	0
tipo_pista	0
condicao_metereologica	0
sentido_via	0
fase_dia	0
tipo_acidente	0
feridos_graves	0
causa_acidente	0

Fonte: O autor (2021)

Conforme mostra na figura 24, foi preenchido os valores faltosos da coluna *feriado* para *Dia Normal*, em seguida foi preenchido os valores da coluna *municipio* com os valores da coluna nome e remover a coluna nome pois está redundante com a coluna *municipio*.

Figura 24 — Tratamento de df_feriado

```
# Preencher dados incompletos da coluna feriado
df['feriado'].fillna('Dia Normal', inplace=True)
# Preencher a coluna município com valores acentuados de nome e apagar a coluna nome
df['municipio'] = df['nome']
df.pop('nome')
df.tail()
```

Fonte: O autor (2021)

Para padronizar foi alterado os valores da coluna *dia_semana* para primeira maiúscula e remoção do hífen como mostra na Figura 25.

Figura 25 — Correção da coluna dia_semana

```
# Para padronizar, vamos deixar tudo em title (primeira letra maiúscula)
# e depois deixar os dias sem o hífen
df['dia_semana'] = df['dia_semana'].apply(lambda x: str(x).title().split('-')[0])
trim = pd.Series(df['dia_semana'])
trim = trim.str.strip()
df['dia_semana'] = trim
df['dia_semana'] = df['dia_semana'].str.title()
df['dia_semana'].unique()

array(['Domingo', 'Segunda', 'Terça', 'Sexta', 'Quarta', 'Quinta',
       'Sábado'], dtype=object)
```

Fonte: O autor (2021)

Para a coluna *sexo*, foi alterado a valor da coluna *Ignorado* para *Não Informado* porque ambos os valores são iguais. (Figura 26)

Figura 26 — Correção da coluna "sexo"

```
[ ] df['sexo'].unique()

array(['Masculino', 'Feminino', 'Não Informado', 'Ignorado'], dtype=object)

df['sexo'] = df['sexo'].str.replace('Ignorado', 'Não Informado')
df['sexo'].unique()

array(['Masculino', 'Feminino', 'Não Informado'], dtype=object)
```

Fonte: O autor (2021)

Após a normalização dos dados de df, foi criado um arquivo com extensão csv com os dados tratados com o nome completo_normalizado.csv, dados estes que será usado pelas próximas etapas. (Ver Figura 27)

Figura 27 — Salvar dados normalizados

```
df.to_csv(f'{path}/completo_normalizado.csv')
```

Fonte: O autor (2021)

4 ANÁLISE/EXPLORAÇÃO DOS DADOS

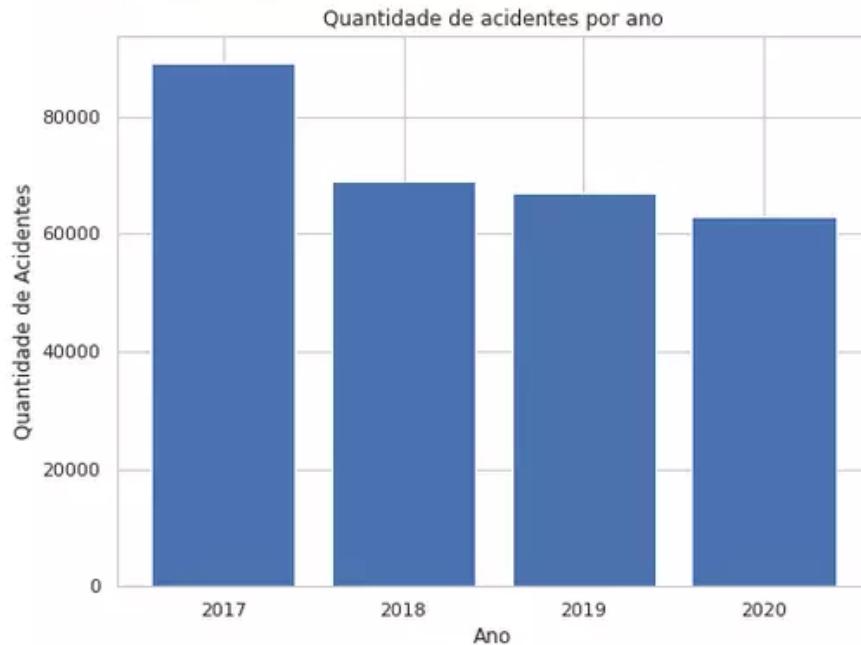
A análise e exploração dos dados foi efetuada por meio de scripts em python utilizando google colab e com bibliotecas *pandas*, *seaborn*, *matplotlib* e *folium*.

A quantidade de acidentes, vem diminuindo conforme podemos ver na figura Figura 28, conforme podemos constatar pelos anos usados neste estudo, os acidentes de 2017 a 2020, percebe-se uma tendência de queda na quantidade de acidentes por ano.

Figura 28 — Quantidade de acidentes por ano

```
ano = df.groupby(['id','ano'],as_index=False).count().iloc[:,range(2)]
plt.figure(figsize=(8,6))
plt.bar(np.arange(4), ano['ano'].value_counts())
ticks = plt.xticks(np.arange(4), ano['ano'].value_counts().index)
plt.title('Quantidade de acidentes por ano')
plt.ylabel('Quantidade de Acidentes')
plt.xlabel('Ano')
```

Text(0.5, 0, 'Ano')



Fonte: O autor (2021)

A maioria dos acidentes acontece nos finais de semana, sendo o sábado o dia com maior número de acidentes. Sexta-feira é o dia da semana com maior número de acidentes, enquanto, terça-feira é o dia com menor número de acidentes conforme podemos verificar na Figura 29.

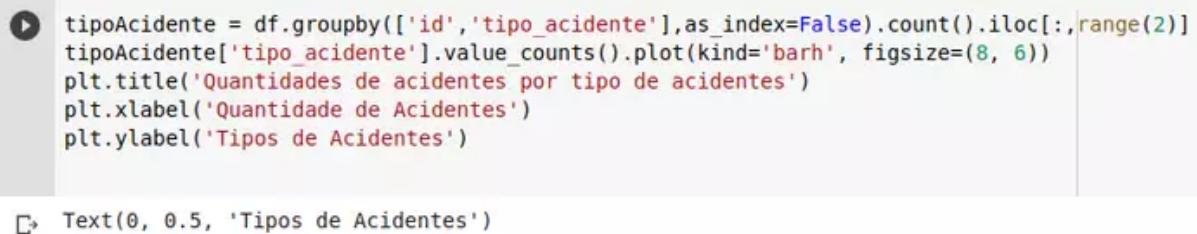
Figura 29 — Quantidades de acidentes por dia da semana



Fonte: O autor (2021)

A colisão traseira é o tipo de acidente mais comum, com mais de 50.000 acidentes registrados. A saída de leito carroçável e colisão traseira são a segunda e a terceira causa mais comum de acidentes, com mais de 35.000 registros conforme podemos verificar na Figura 30.

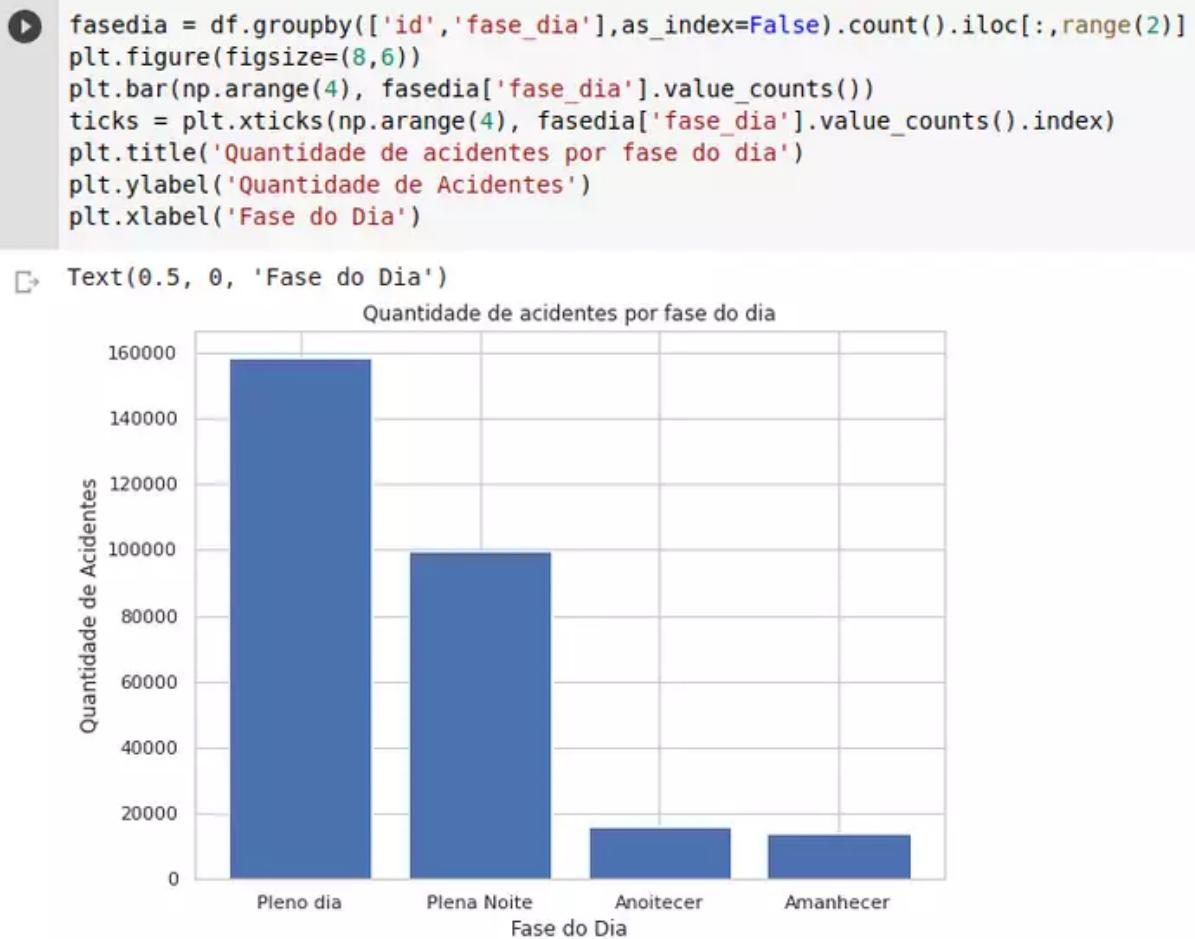
Figura 30 — Quantidades de acidentes por tipo de acidentes



Fonte: O autor (2021)

A maioria dos acidentes ocorreu durante Pleno dia, enquanto ao amanhecer aconteceu menos acidentes conforme podemos verificar na Figura 31.

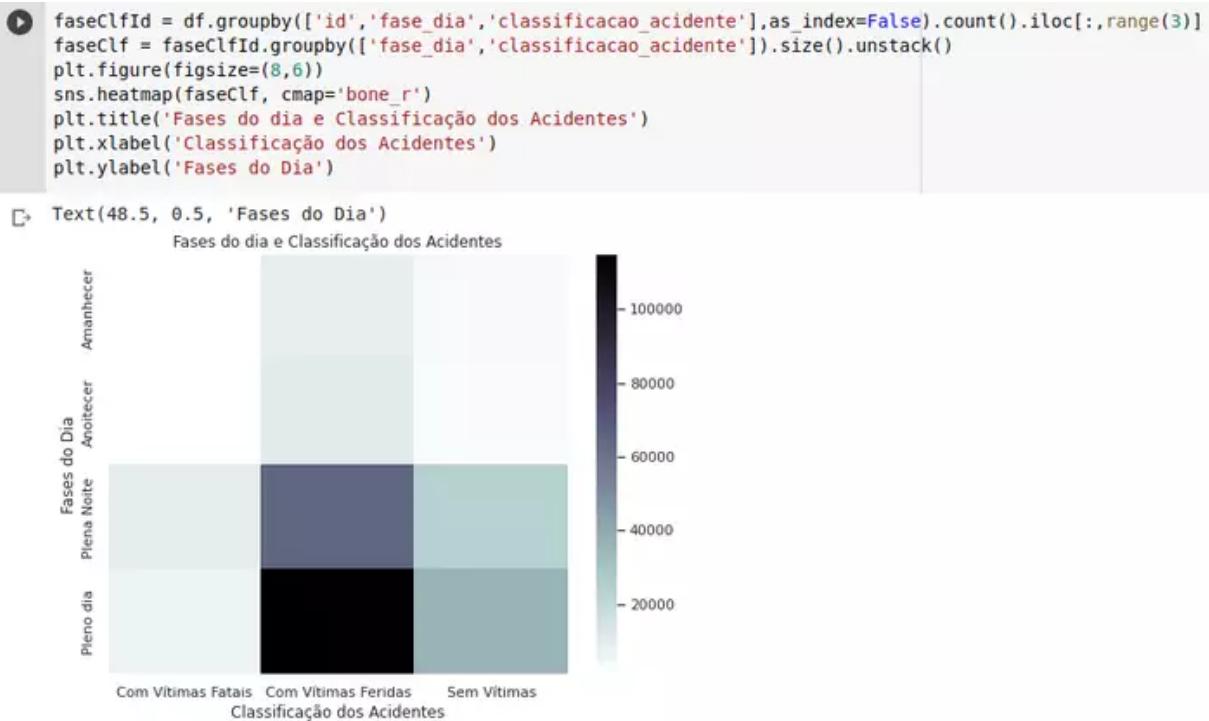
Figura 31 — Quantidade de acidentes por fase do dia



Fonte: O autor (2021)

Enquanto a maioria dos acidentes com vítimas feridas ocorreu durante o dia, os acidentes durante a noite com vítimas fatais foram maiores do que durante o dia, provavelmente devido à uma pior visibilidade causando assim acidentes com mais vítimas fatais.

Figura 32 — Fases do dia e Classificação dos Acidentes



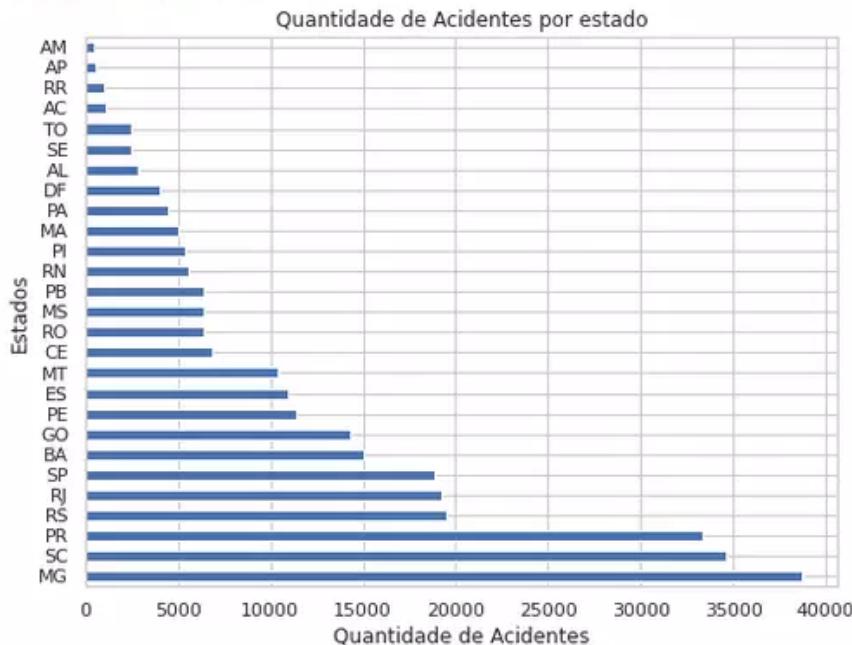
Fonte: O autor (2021)

Podemos perceber que os estados de MG, SC e PR são os estados que mais têm acidentes e que o estado de MG tem quase o dobro do RS que está na quarta posição conforme podemos verificar na Figura 33.

Figura 33 — Quantidade de Acidentes por estado

```
estados = df.groupby(['id','uf'],as_index=False).count().iloc[:,range(2)]
estados['uf'].value_counts().plot(kind='barh', figsize=(8, 6))
plt.title('Quantidade de Acidentes por estado');
plt.xlabel('Quantidade de Acidentes')
plt.ylabel('Estados')
```

Text(0, 0.5, 'Estados')



Fonte: O autor (2021)

O estado da BA ocupa a sétima posição do ranking em acidentes, mas conforme podemos conferir na figura 34, comparando a proporção da classificação dos acidentes podemos perceber que o único estado onde a quantidade proporcional de acidentes fatais, o que representa que na maioria dos acidentes no estado da BA são acidentes com vítimas fatais, e que a quantidade de vítimas se compara aos estados de SC e PR com números muitos maiores de acidentes.

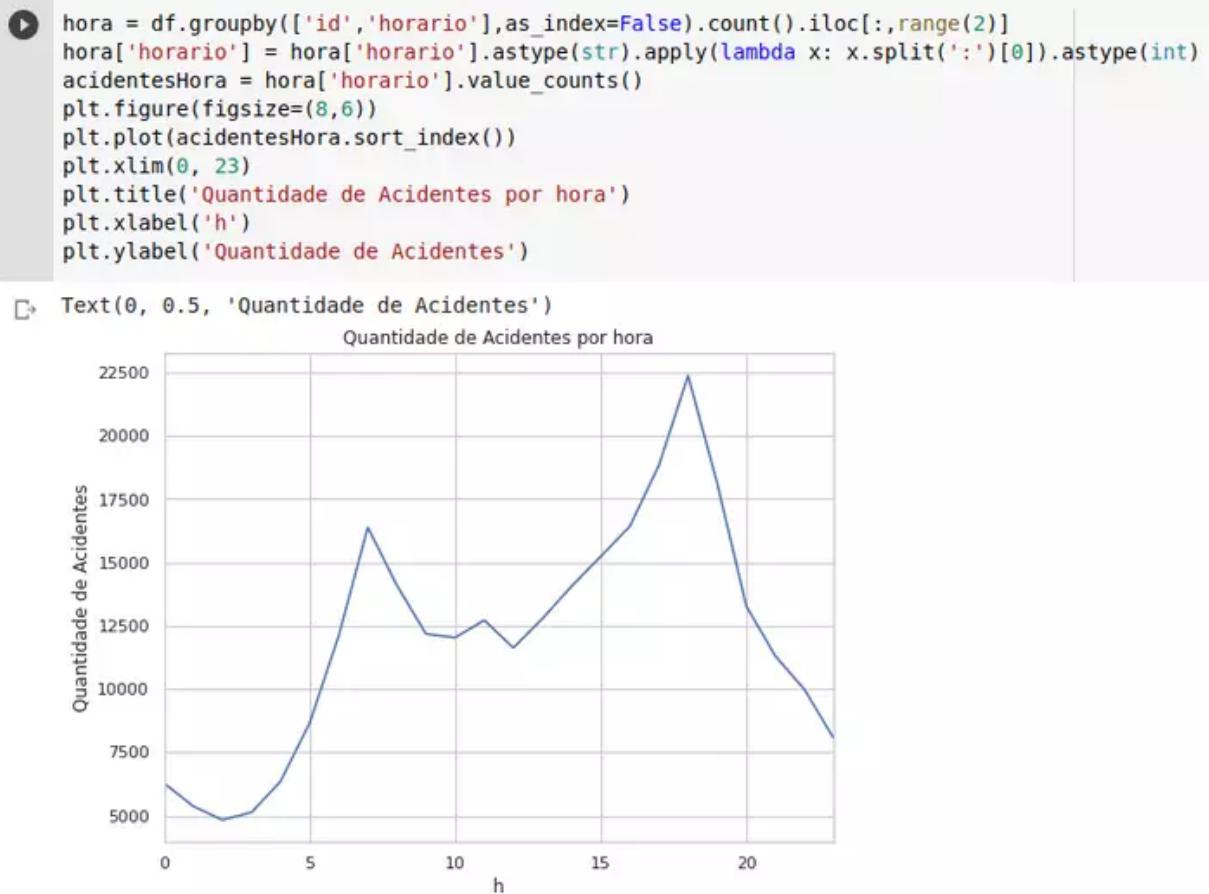
Figura 34 — Mapa de calor - Estados e Proporção da Classificação dos Acidentes



Fonte: O autor (2021)

Durante as horas do dia podemos acompanhar a quantidade de acidentes, das 0h às 5h é o período do dia com menor números de acidentes enquanto o período das 15h às 20h é o período do dia em que ocorre mais acidentes conforme podemos verificar na Figura 35.

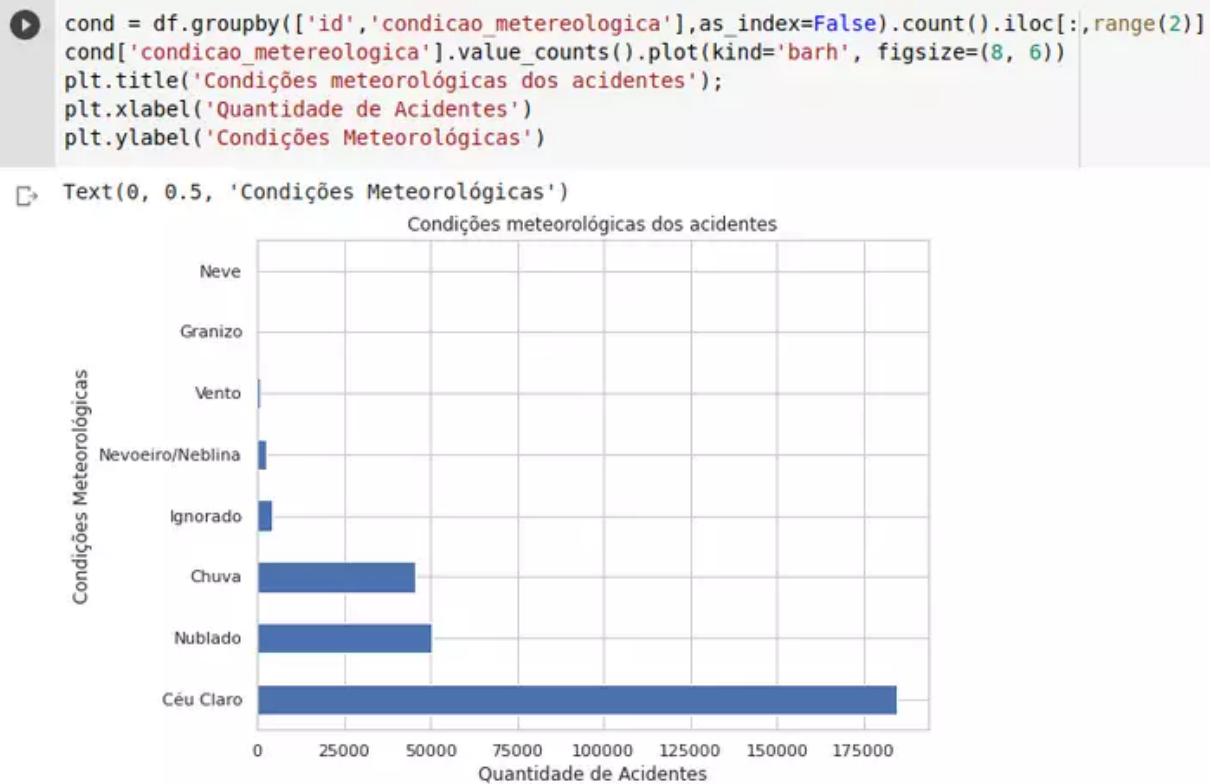
Figura 35 — Quantidade de Acidentes por hora



Fonte: O autor (2021)

As condições meteorológicas onde acontece mais acidentes seria com céu claro com o mostra na Figura 36 e podemos também verificar na Figura 37, o mapa de calor onde mostra a classificação dos acidentes conforme as condições meteorológicas.

Figura 36 — Condições meteorológicas dos acidentes



Fonte: O autor (2021)

Figura 37 — Mapa de calor - Condições Meteorológicas e Classificação dos Acidentes



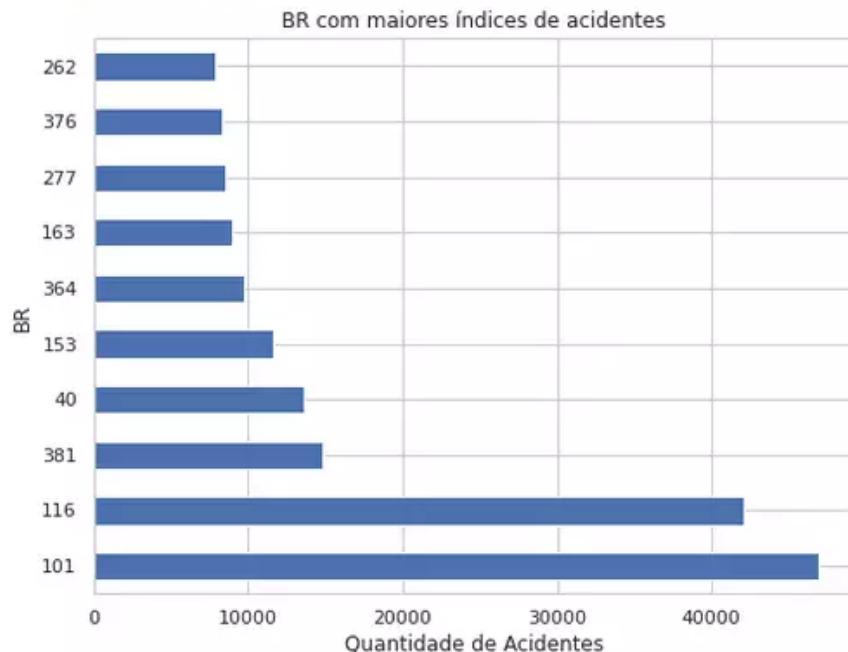
Fonte: O autor (2021)

As duas rodovias com maior número de acidentes em relação às demais, é a BR-101 com 4.650 km de extensão que sai do município de Touros, no Rio Grande do Norte, e termina em São José do Norte, no Rio Grande do Sul e a BR-116 com 4.486 km de extensão, que sai do município de fortaleza, no Ceará, e termina em Jaguarão, no Rio Grande do Sul, sendo as duas considerado como principais eixos rodoviários do país. (Ver Figura 38).

Figura 38 — BR com maiores índices de acidentes

```
▶ br = df.groupby(['id','br'],as_index=False).count().iloc[:,range(2)]
  plt.figure(figsize=(8,6))
  bri['br'].value_counts().head(10).plot.barh()
  plt.title('BR com maiores índices de acidentes');
  plt.xlabel('Quantidade de Acidentes')
  plt.ylabel('BR')
```

□ Text(0, 0.5, 'BR')

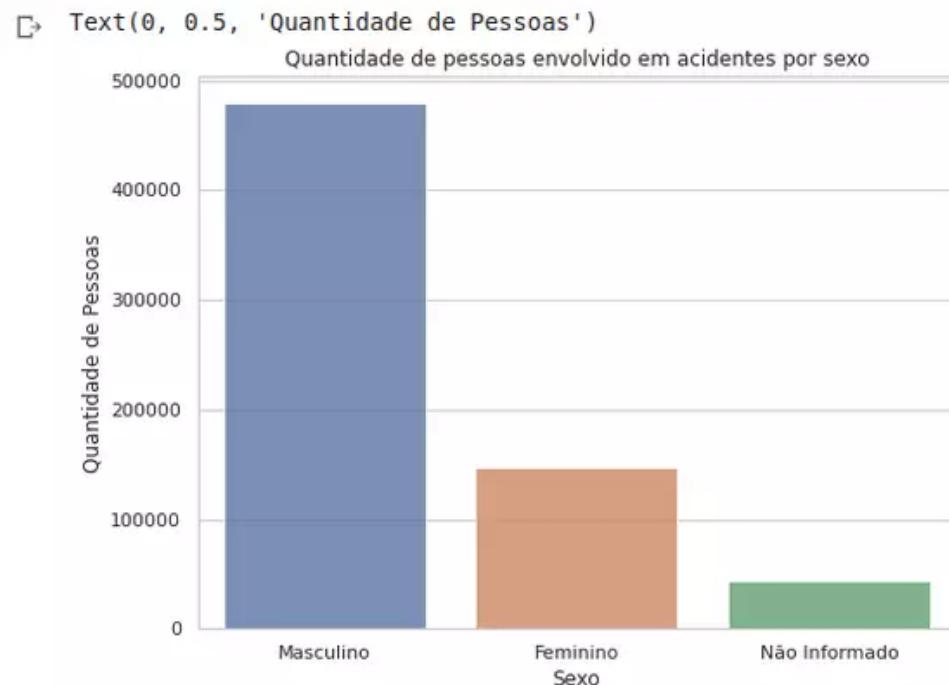


Fonte: O autor (2021)

A quantidade de acidentes envolvendo pessoas do sexo Masculino foi muito superior ao pessoas do sexo feminino ou não informado conforme podemos verificar na Figura 39, na Figura 40 e Figura 42.

Figura 39 — Quantidade de pessoas envolvido em acidentes por sexo

```
sex = df['sexo'].value_counts()
plt.figure(figsize=(8,6))
sns.barplot(sexo.index, sexo.values, alpha=0.8)
plt.title('Quantidade de pessoas envolvido em acidentes por sexo')
plt.xlabel('Sexo')
plt.ylabel('Quantidade de Pessoas')
```

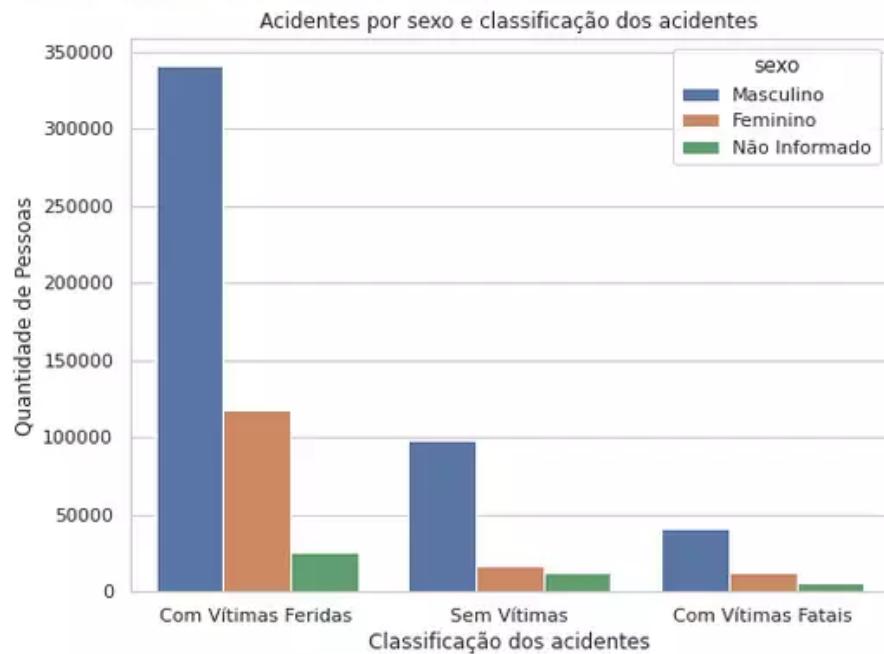


Fonte: O autor (2021)

Figura 40 — Acidentes por sexo e classificação dos acidentes

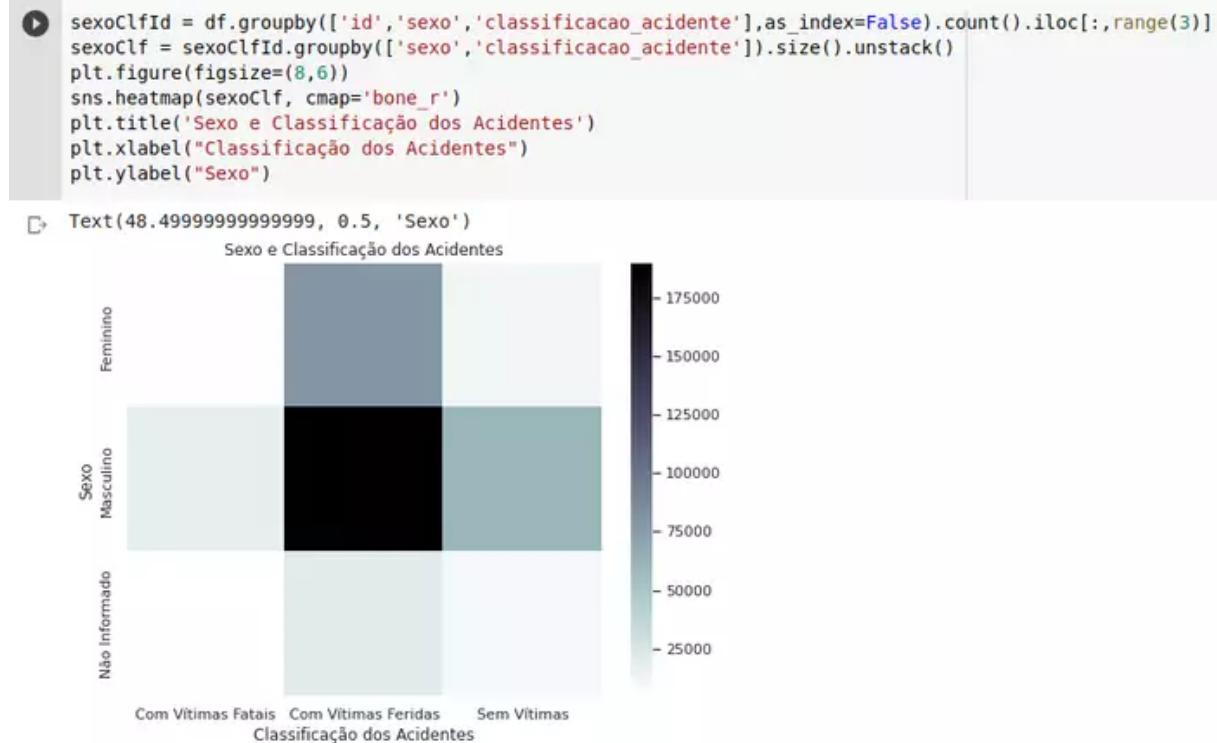
```
plt.figure(figsize=(8,6))
sns.countplot(x='classificacao_acidente', data=df, hue='sexo')
plt.title('Acidentes por sexo e classificação dos acidentes')
plt.xlabel('Classificação dos acidentes')
plt.ylabel('Quantidade de Pessoas')
```

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



Fonte: O autor (2021)

Figura 41 — Mapa de calor - Sexo e Classificação dos acidentes



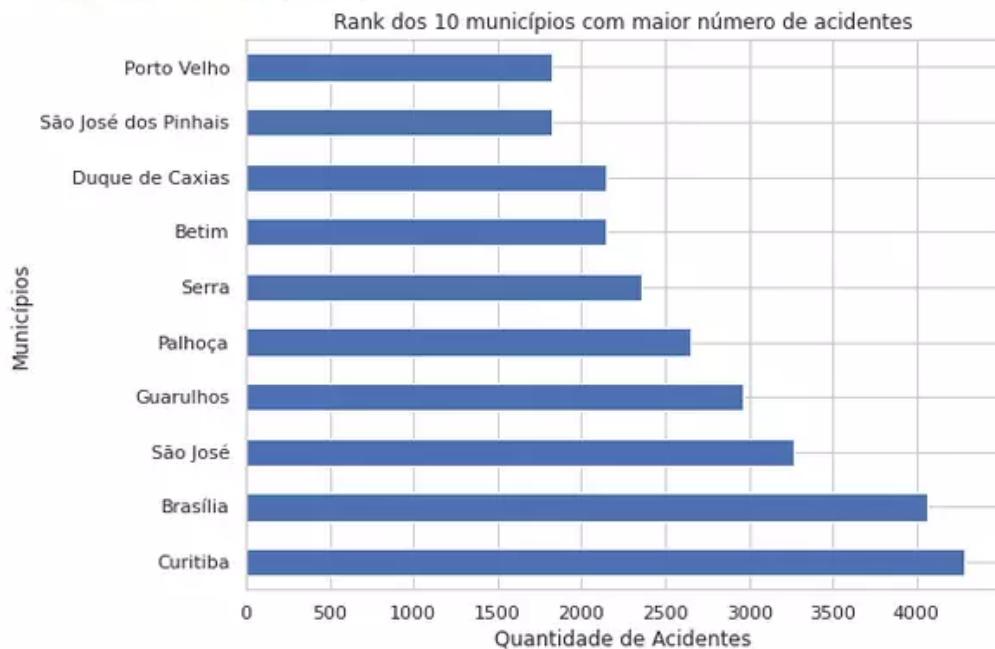
Fonte: O autor (2021)

Dos municípios com maior quantidade de acidentes como podemos visualizar na figura 42, Curitiba (SC) está na frente dos demais municípios, seguido por Brasília (DF), São José (SC) e Guarulhos (SP).

Figura 42 — Rank dos 10 municípios com maior número de acidentes

```
▶ mun = df.groupby(['id','municipio'],as_index=False).count().iloc[:,range(2)]
  plt.figure(figsize=(8,6))
  mun['municipio'].value_counts().head(10).plot.barh()
  plt.title('Rank dos 10 municípios com maior número de acidentes');
  plt.xlabel('Quantidade de Acidentes')
  plt.ylabel('Municípios')
```

▷ Text(0, 0.5, 'Municípios')



Fonte: O autor (2021)

A região Sudeste é a região onde mais contém acidentes, na segunda posição e muito próximo está a região sul, seguida por Nordeste, Centro-Oeste e Norte.(Figura 43)

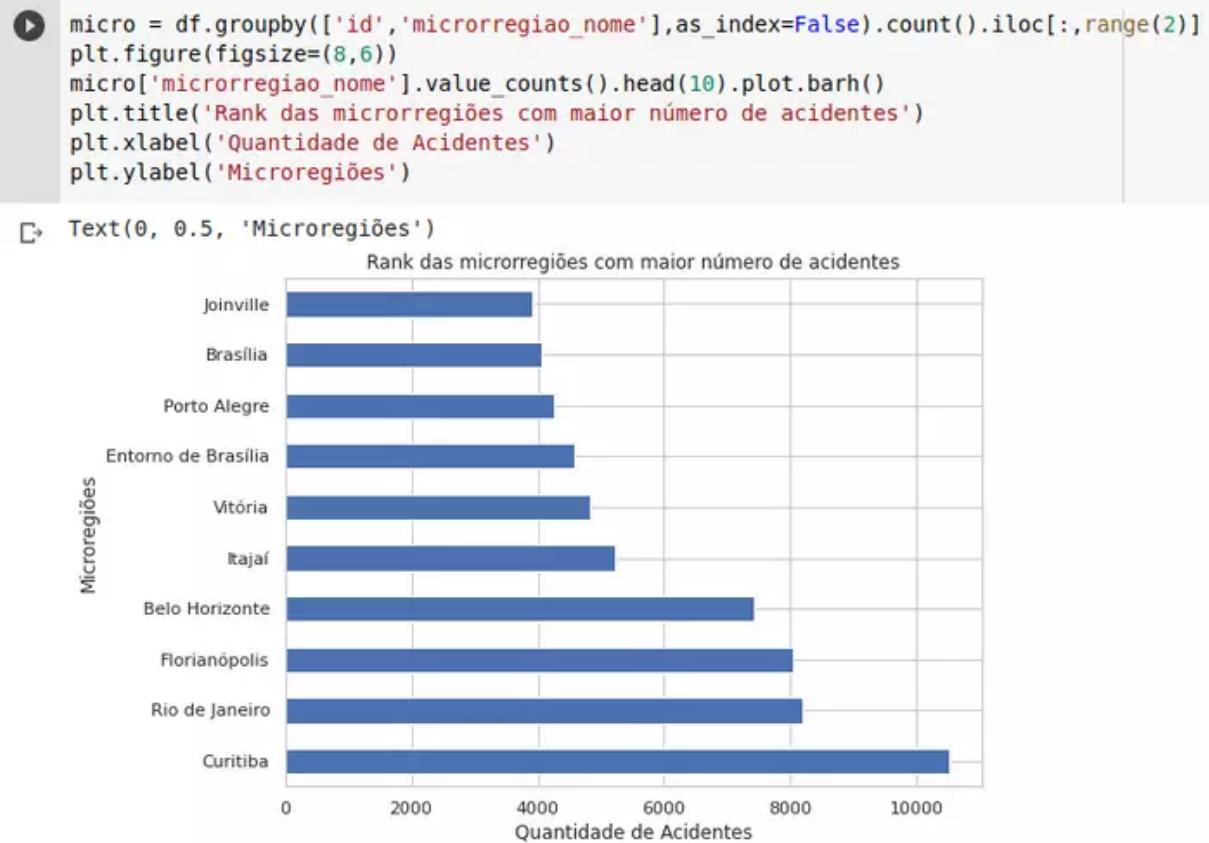
Figura 43 — Rank das regiões com maior número de acidentes



Fonte: O autor (2021)

Das microrregiões do Brasil, Curitiba é microrregião que contém mais acidentes, seguida por Rio de Janeiro, Florianópolis e Belo Horizonte. (Figura 44)

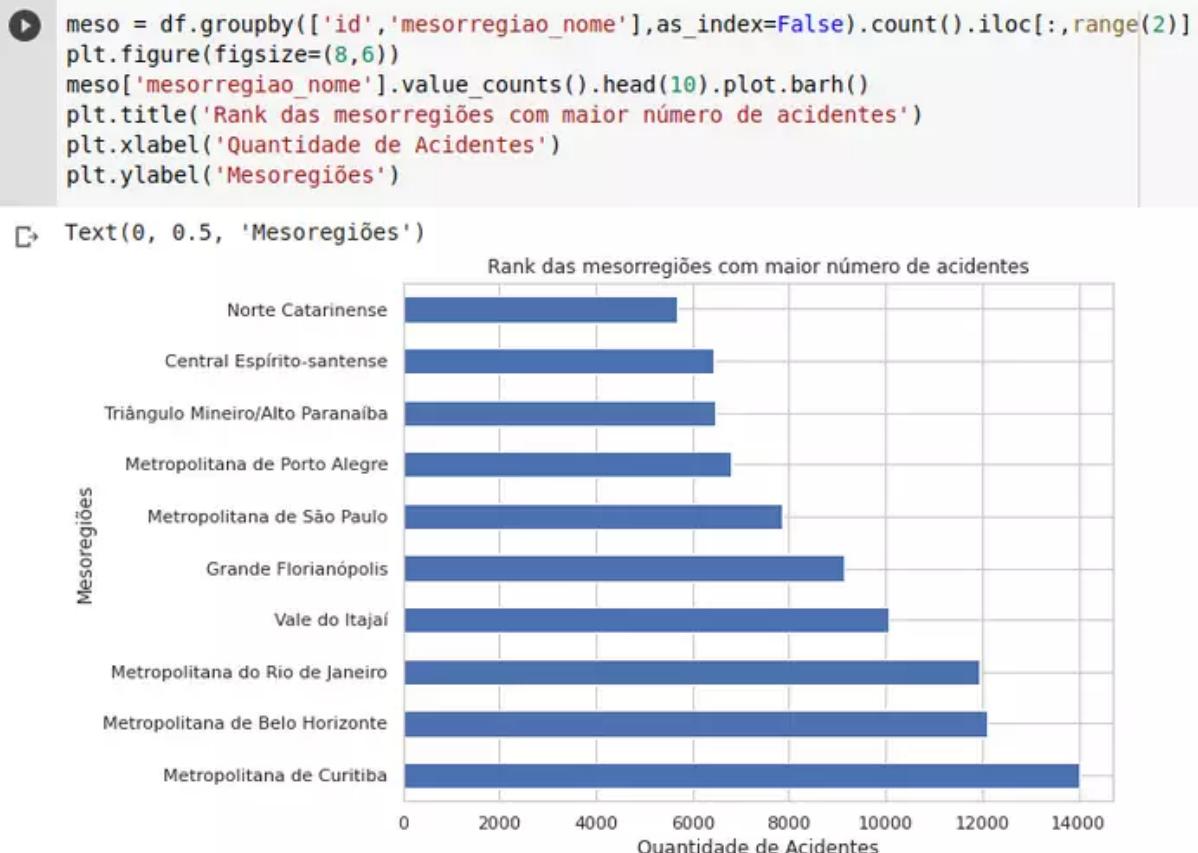
Figura 44 — Rank das microrregiões com maior número de acidentes



Fonte: O autor (2021)

Entre as Mesorregiões com maior número de acidentes em primeiro lugar aparece a Metropolitana de Curitiba seguida por Metropolitana de Belo Horizonte, Metropolitana do Rio de Janeiro e Vale do Itajaí.(Figura 45)

Figura 45 — Rank das mesorregiões com maior número de acidentes



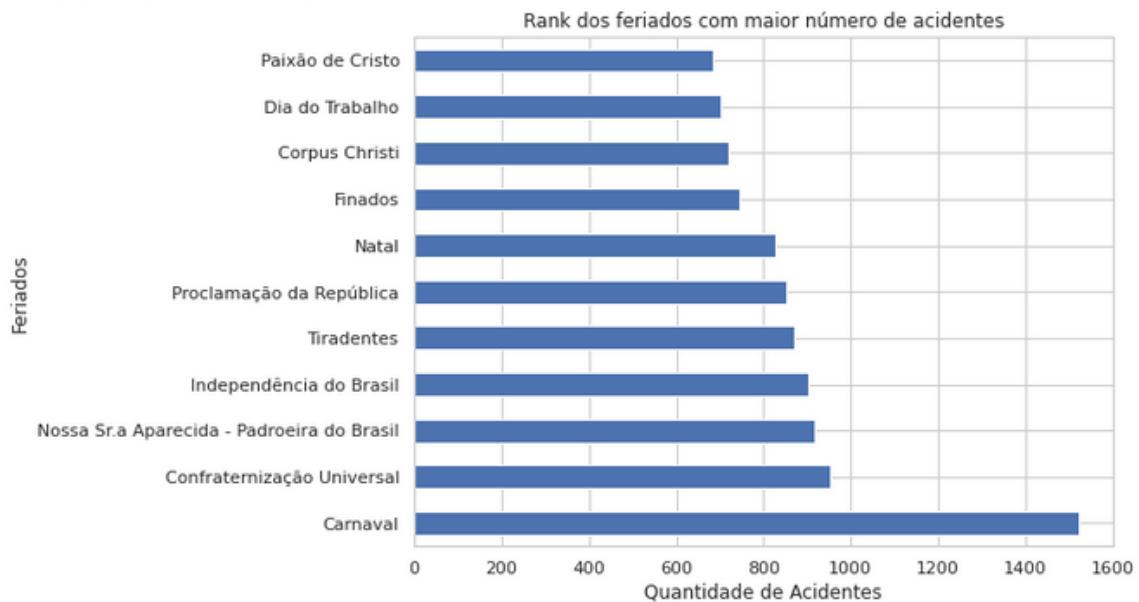
Fonte: O autor (2021)

Entre os feriados, o feriado com maior número de acidentes é o carnaval, em seguida Confraternização Universal, Nossa Sra. Aparecida e Independência do Brasil. (Figura 46)

Figura 46 — Rank dos feriados com maior número de acidentes

```
feriados = df.groupby(['id','feriado'],as_index=False).count().iloc[:,range(2)]
feriados = feriados[(feriados['feriado'] != "Dia Normal")]
plt.figure(figsize=(8,6))
feriados['feriado'].value_counts().plot.barh()
plt.title('Rank dos feriados com maior número de acidentes')
plt.xlabel('Quantidade de Acidentes')
plt.ylabel('Feriados')

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

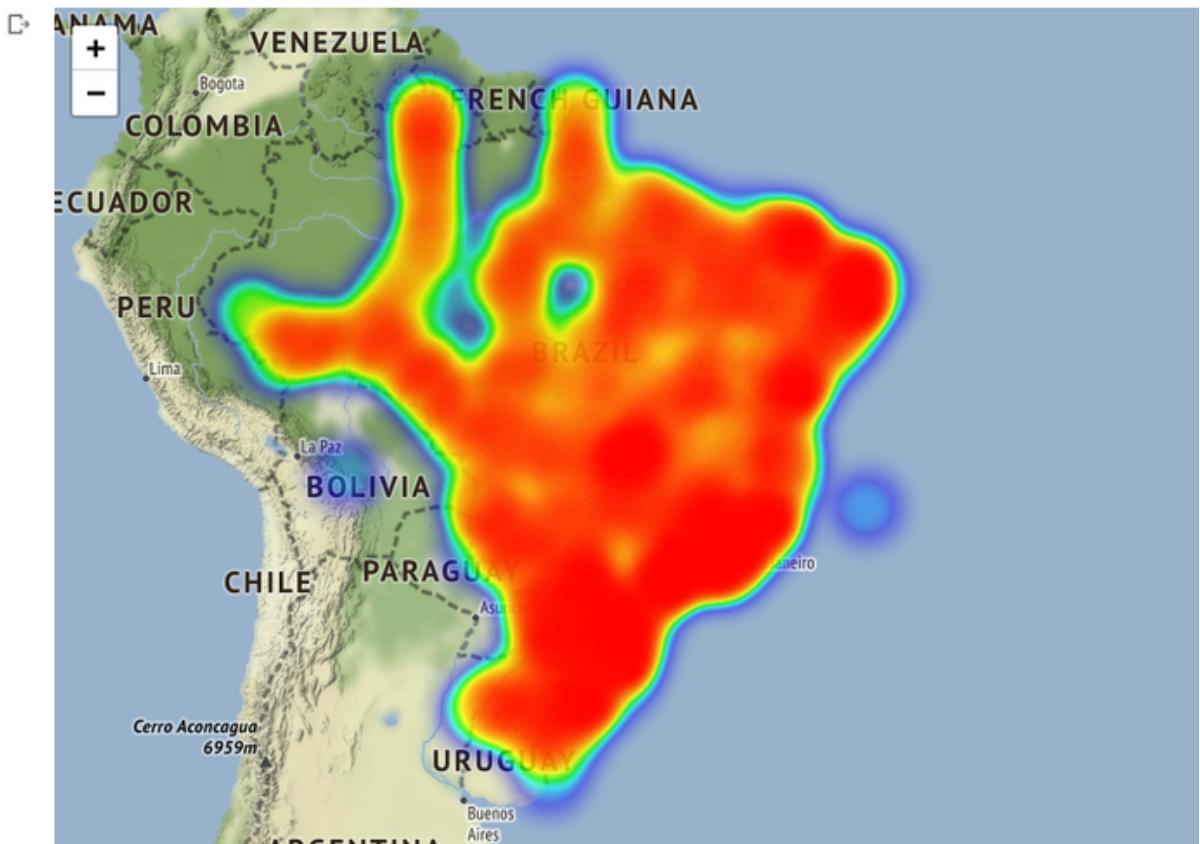


Fonte: O autor (2021)

Com o mapa de color foi possível visualizar no mapa as regiões onde se concentra os maiores números de acidentes como podemos verificar na Figura 47.

Figura 47 — Mapa de calor de acidentes do Brasil

```
coor = df.groupby(['id','latitude','longitude'],as_index=False).count().iloc[:,range(3)]
coordenadas=[]
for lat,lng in zip(coor['latitude'].values,coor['longitude'].values):
    coordenadas.append([lat,lng])
mapa = folium.Map(location=[-15.788497,-47.879873],zoom_start=3,tiles='Stamen Terrain')
mapa.add_child(plugins.HeatMap(coordenadas))
mapa
```



Fonte: O autor (2021)

5 CRIAÇÃO DE MODELOS DE ML

Após a coleta dos dados, do processo de tratamento e de uma análise e exploração dos dados, a partir deste ponto, a proposta do trabalho é efetuar a aplicação de modelos de machine learning para a classificação do acidente.

Para aplicação dos modelos de machine learning usou-se a biblioteca `sklearn` (SCIKIT-LEARN, 2021) que é uma biblioteca de aprendizado de máquina de código aberto para a linguagem Python. Ela inclui vários algoritmos de classificação, que para este estudo foi usado seus principais classificadores.

Para otimização dos hiperparâmetros foi utilizado a ferramenta `GridSearchCV` do `sklearn`, que é usada para automatizar o processo de ajuste dos parâmetros de um algoritmo, pois ele fará de maneira sistemática diversas combinações dos parâmetros e depois de avaliá-los os armazenará num único objeto.

Antes de treinar os modelos como podemos verificar na Figura 48, foi criado um novo dataframe com menos colunas, foi usado a ferramenta `train_test_split` do `sklearn`, para separar dados de treino e teste.

Figura 48

```
np.random.seed = 42

cols = ['id', 'dia_semana', 'uf', 'br', 'fase_dia', 'condicao_meteorologica',
        'tracado_via', 'idade', 'sexo', 'regiao', 'microrregiao', 'mesorregiao',
        'feriado', 'classificacao']
df_acor = df[cols]

x = pd.get_dummies(df_acor.drop(['classificacao'], axis=1))
y = df_acor['classificacao']

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.2,
                                                    stratify = y)
```

Fonte: O autor (2021)

5.1 FLORESTA ALEATÓRIA (RANDOM FOREST)

Floresta Aleatória (Random Forest) é um algoritmo de aprendizagem de máquina flexível e fácil de usar que produz excelentes resultados a maioria das vezes, mesmo sem ajuste de hiperparâmetros. É também um dos algoritmos mais utilizados, devido à sua simplicidade e o fato de que pode ser utilizado para tarefas de classificação e também de regressão.

Figura 49 — Treinar modelo Árvores Aleatórias

```

model = RandomForestClassifier(verbose=1, n_jobs=-1)

parameters = {
    'n_estimators': [10, 100],
    'criterion': ['gini', 'entropy'],
    'max_features': ['sqrt', 'auto', 'log2', None],
    'class_weight': ['balanced', None],
}

scorer = make_scorer(score_func=accuracy_score)
grid_obj = GridSearchCV(model, parameters, scoring=scorer)
grid_fit = grid_obj.fit(x_train, y_train)

modelo = grid_fit.best_estimator_

pickle.dump(modelo, open('RandomForest.sav', 'wb'))

```

Parallel(n_jobs=-1): Done 10 out of 10 | elapsed: 9.25 finished
 [Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
 [Parallel(n_jobs=4)]: Done 10 out of 10 | elapsed: 0.4s finished
 [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
 [Parallel(n_jobs=-1)]: Done 10 out of 10 | elapsed: 9.3s finished
 [Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
 [Parallel(n_jobs=4)]: Done 10 out of 10 | elapsed: 0.5s finished
 [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
 [Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 37.5s
 [Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 1.4min finished
 [Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
 [Parallel(n_jobs=4)]: Done 42 tasks | elapsed: 1.4s
 [Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed: 3.1s finished
 [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
 [Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 34.2s

Fonte: O autor (2021)

Conforme podemos verificar na Figura 50 a melhor configuração de parâmetro encontrado pelo *GridSearchCV* do *sklearn* para este modelo e na Figura 51 foi feito a sua predição.

Figura 50 — Configuração de parâmetro que forneceu os melhores resultados

```

model

```

RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
 criterion='gini', max_depth=None, max_features='log2',
 max_leaf_nodes=None, max_samples=None,
 min_impurity_decrease=0.0, min_impurity_split=None,
 min_samples_leaf=1, min_samples_split=2,
 min_weight_fraction_leaf=0.0, n_estimators=100,
 n_jobs=-1, oob_score=False, random_state=None, verbose=1,
 warm_start=False)

Fonte: O autor (2021)

Figura 51 — Predições de Floresta Aleatória

```
▶ y_pred = model.predict(x_test)

↳ [Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done 42 tasks      | elapsed:    1.4s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    3.2s finished
```

Fonte: O autor (2021)

O nível de exatidão dos resultados, pode ser verificado na Figura 52.

Figura 52 — Acurácia Floresta Aleatória

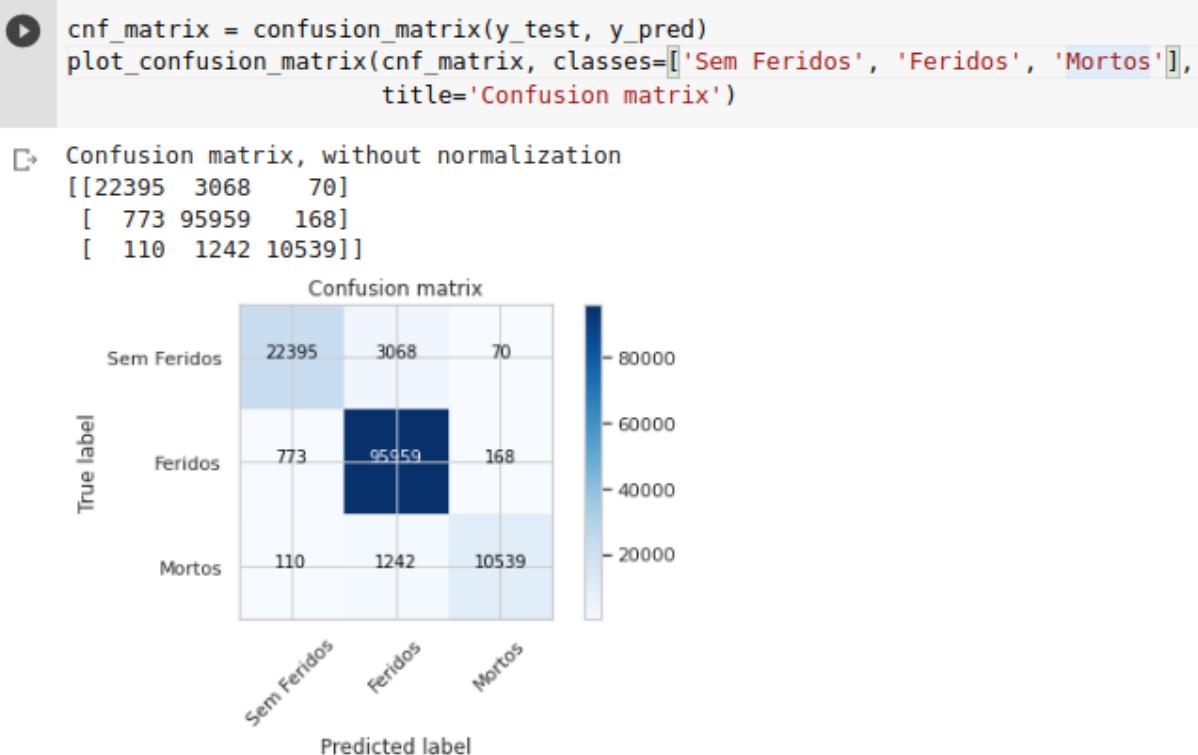
```
▶ accuracy = accuracy_score(y_test, y_pred) * 100
print("A acurácia foi %.2f%%" % accuracy)

↳ A acurácia foi 95.96%
```

Fonte: O autor (2021)

Conforme podemos verificar na Figura 64 números de falsos positivos, falsos negativos, verdadeiros positivos e verdadeiros negativos do modelo de classificação Floresta Aleatória.

Figura 53 — Matriz de confusão Floresta Aleatória



Fonte: O autor (2021)

Figura 54 — Principais métricas de classificação Floresta Aleatória

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.96	0.88	0.92	25533
1	0.96	0.99	0.97	96900
2	0.98	0.89	0.93	11891
accuracy			0.96	134324
macro avg	0.97	0.92	0.94	134324
weighted avg	0.96	0.96	0.96	134324

Fonte: O autor (2021)

5.2 K-MEANS

A ideia do algoritmo K-Means é fornecer uma classificação de informações de acordo com os próprios dados. Esta classificação, como será vista a seguir, é baseada em análise e comparações entre os valores numéricos dos dados. Desta

maneira, o algoritmo automaticamente vai fornecer uma classificação automática sem a necessidade de nenhuma supervisão humana, ou seja, sem nenhuma pré-classificação existente. Por causa desta característica, o K-Means é considerado como um algoritmo de mineração de dados não supervisionado.

Figura 55 — Treinar modelo K-Means

```
model = KNeighborsClassifier()

parameters = {
    'n_neighbors': [3,5,11,19],
    'weights': ['uniform','distance'],
    'metric': ['euclidean','manhattan']
}

scorer = make_scorer(score_func=accuracy_score)
grid_obj = GridSearchCV(model, parameters, scoring=scorer)
grid_fit = grid_obj.fit(x_train, y_train)

modelo = grid_fit.best_estimator_

pickle.dump(modelo, open('KMeans.sav', 'wb'))
```

Fonte: O autor (2021)

Conforme podemos verificar na Figura 56 a melhor configuração de parâmetro encontrado pelo *GridSearchCV* do *sklearn* para este modelo e na Figura 57 foi feito a sua predição.

Figura 56 — Configuração de parâmetro que forneceu os melhores resultados

```
model
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='manhattan',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='distance')
```

Fonte: O autor (2021)

Figura 57 — Predições de K-Means

```
y_pred = model.predict(x_test)
```

Fonte: O autor (2021)

O nível de exatidão dos resultados, pode ser verificado na Figura 58.

Figura 58 — Acurácia K-Means

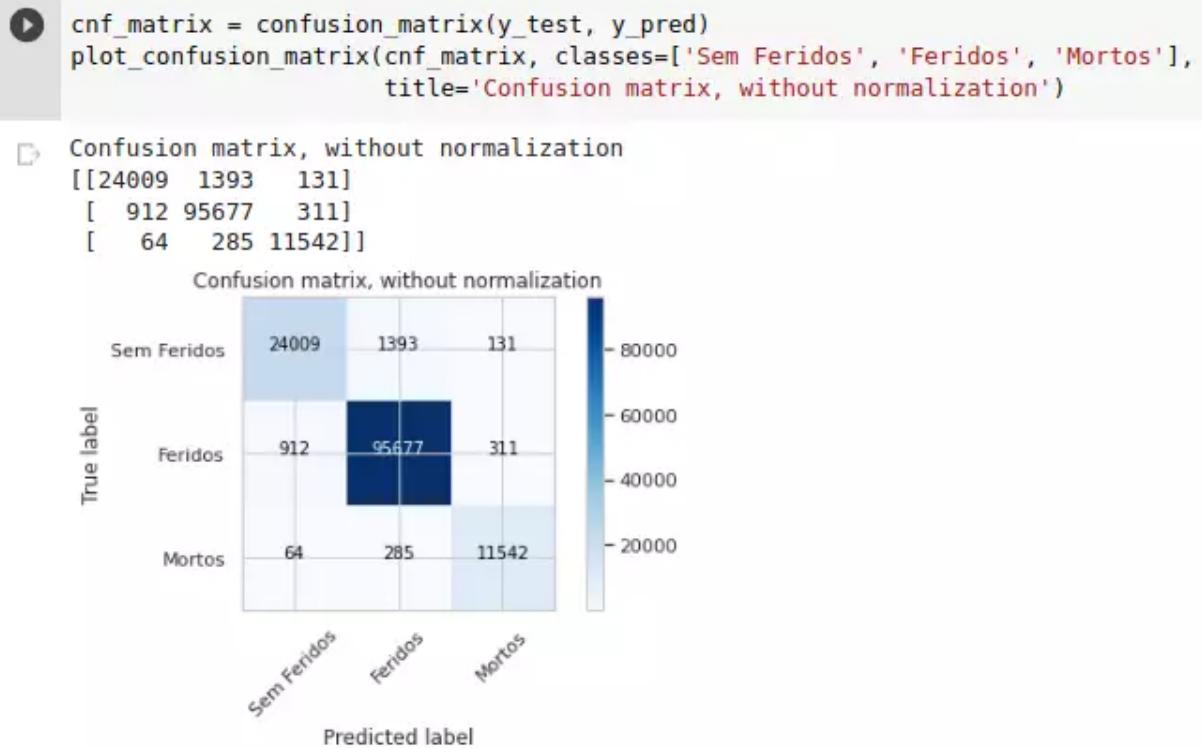
```
accuracy = accuracy_score(y_test, y_pred) * 100
print("A acurácia foi %.2f%%" % accuracy)

⇒ A acurácia foi 97.70%
```

Fonte: O autor (2021)

Conforme podemos verificar na Figura 59, números de falsos positivos, falsos negativos, verdadeiros positivos e verdadeiros negativos do modelo de classificação k-means.

Figura 59 — Matriz de confusão K-Means



Fonte: O autor (2021)

Figura 60 — Principais métricas de classificação K-Means

```
▶ print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.96	0.94	0.95	25533
1	0.98	0.99	0.99	96900
2	0.96	0.97	0.97	11891
accuracy			0.98	134324
macro avg	0.97	0.97	0.97	134324
weighted avg	0.98	0.98	0.98	134324

Fonte: O autor (2021)

5.3 REGRESSÃO LOGÍSTICA

Regressão logística é um algoritmo de classificação de aprendizado de máquina usado para prever a probabilidade de uma variável dependente categórica.

Conforme podemos verificar na Figura 61 a melhor configuração de parâmetro encontrado pelo *GridSearchCV* do *sklearn* para este modelo.

Figura 61 — Configuração de parâmetro que forneceu os melhores resultados

```
▶ model|
```

```
▶ LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                      intercept_scaling=1, l1_ratio=None, max_iter=100,
                      multi_class='auto', n_jobs=None, penalty='l2',
                      random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                      warm_start=False)
```

Fonte: O autor (2021)

Figura 62 — Predições de regressão logística

```
▶ y_pred = model.predict(x_test)
```

Fonte: O autor (2021)

O nível de exatidão dos resultados, pode ser verificado na Figura 63

Figura 63 — Acurácia regressão logística

```
accuracy = accuracy_score(y_test, y_pred) * 100
print("A acurácia foi %.2f%%" % accuracy)
```

↳ A acurácia foi 72.14%

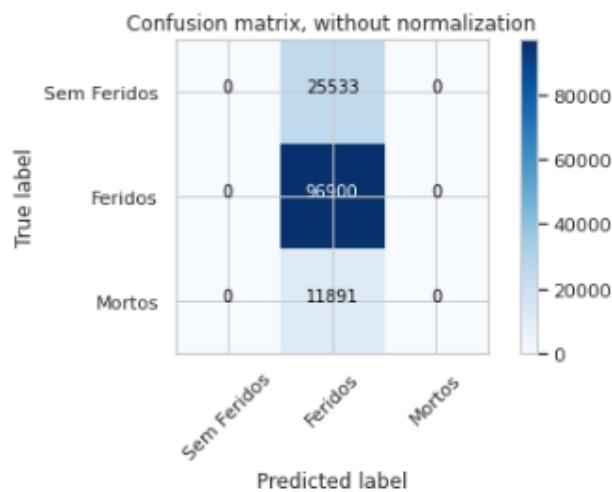
Fonte: O autor (2021)

Conforme podemos verificar na Figura 64, número de falsos positivos, falsos negativos, verdadeiros positivos e verdadeiros negativos do modelo de classificação regressão logística.

Figura 64 — Matriz de confusão Regressão logística

```
cnf_matrix = confusion_matrix(Y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=['Sem Feridos', 'Feridos', 'Mortos'],
                      title='Confusion matrix, without normalization')
```

↳ Confusion matrix, without normalization
[[0 25533 0]
 [0 96900 0]
 [0 11891 0]]



Fonte: O autor (2021)

Figura 65 — Principais métricas de classificação Regressão logística

```
print(classification_report(y_test, y_pred))
```

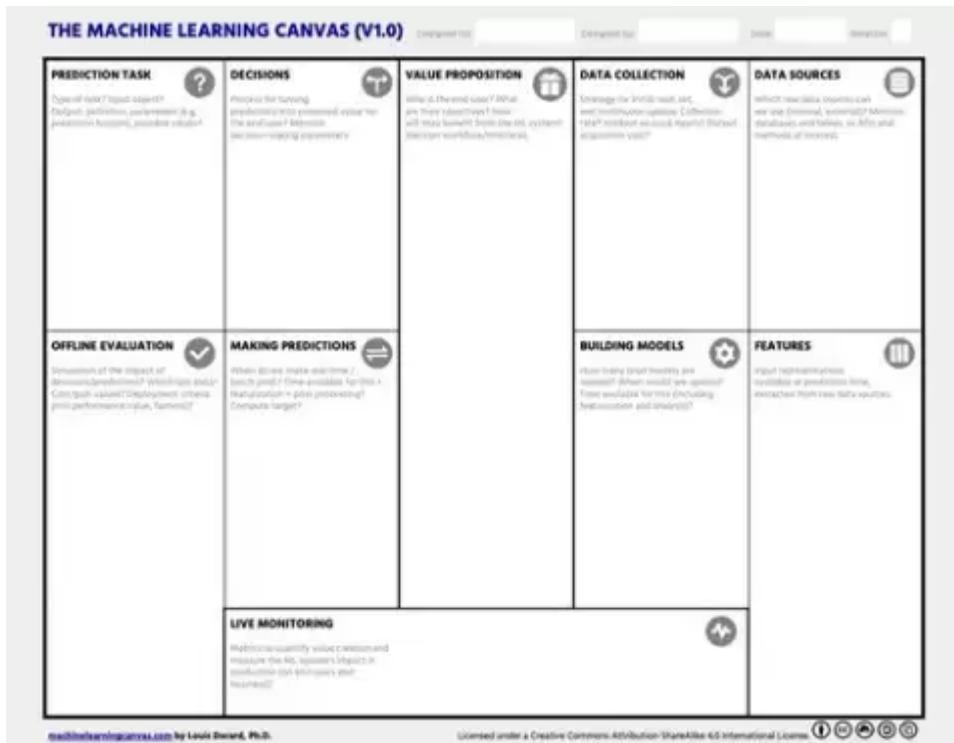
	precision	recall	f1-score	support
0	0.00	0.00	0.00	25533
1	0.72	1.00	0.84	96900
2	0.00	0.00	0.00	11891
accuracy			0.72	134324
macro avg	0.24	0.33	0.28	134324
weighted avg	0.52	0.72	0.60	134324

Fonte: O autor (2021)

6 INTERPRETAÇÃO DOS RESULTADOS

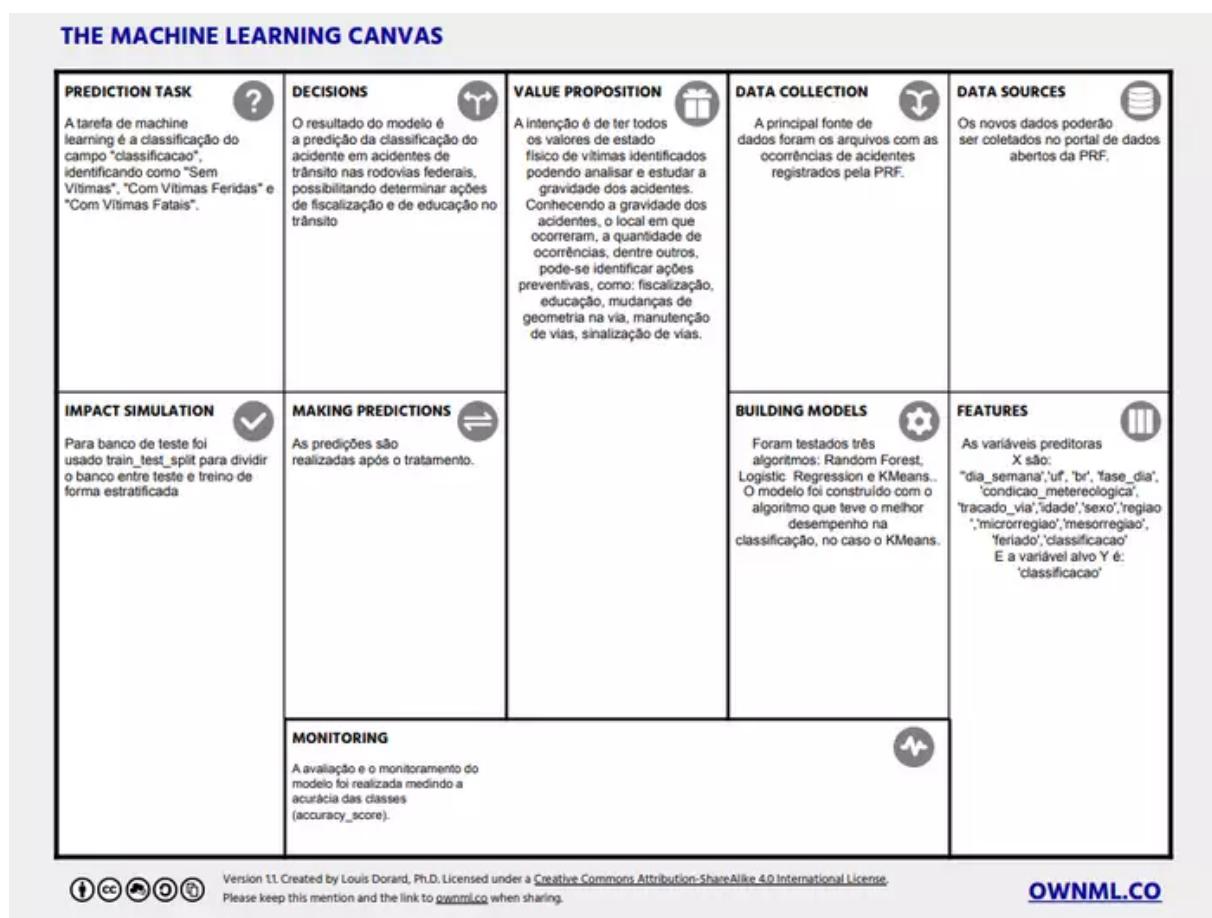
Para a apresentação dos resultados obtidos, foi utilizado o modelo de Canvas proposto por Dourard. (Figura 66)

Figura 66 — Modelo de Canvas para projetos de Machine Learning



Fonte: Dorard (2015)

Figura 67



Fonte: O autor (2021)

7 LINKS

A seguir, estão os links do vídeo de apresentação e do repositório contendo os dados utilizados no projeto.

Link para o vídeo: <https://youtu.be/wC-EzPu9iX8>

Link para o repositório: https://github.com/callacius/TCC_Puc_mg

REFERÊNCIAS

- ANBIMA. Associação Brasileira das Entidades dos Mercados Financeiro e de Capitais. 2021. Disponível em: <https://www.anbima.com.br>. Acesso em: 1 set. 2021.
- COLAB. Google Colaboratory. 2021. Disponível em: <https://colab.research.google.com/>. Acesso em: 1 set. 2021.
- DORARD, Louis. **Machine Learning Canvas**. 2015. Disponível em: <https://www.machinelearningcanvas.com/>. Acesso em: 25 out. 2021.
- IBGE - CIDADES. Sistema agregador de informações do IBGE sobre os municípios e estados do Brasil. 2021. Disponível em: <https://cidades.ibge.gov.br>. Acesso em: 1 set. 2021.
- IBGE. Instituto Brasileiro de Geografia e Estatística. 2021. Disponível em: <https://www.ibge.gov.br>. Acesso em: 1 set. 2021.
- NUMPY. fundamental package for scientific computing with Python. 2021. Disponível em: <https://numpy.org/>. Acesso em: 1 set. 2021.
- OPS, Organização Pan-Americana de Saúde. Estado de la seguridad vial en la Región de las Américas. **OPS**, Washington, D.C., 2019.
- PANDAS. Python Data Analysis Library. 2021. Disponível em: <https://pandas.pydata.org>. Acesso em: 1 set. 2021.
- PRF. **Acidentes em rodovias federais**. Departamento de Polícia Rodoviária Federal (DPRF). 2021. Disponível em: <https://www.gov.br/prf/pt-br/acesso-a-informacao/dados-abertos/dados-abertos-acidentes>. Acesso em: 1 set. 2021.
- SCIKIT-LEARN. Machine Learning in Python. 2021. Disponível em: <https://scikit-learn.org>. Acesso em: 1 set. 2021.
- SEABORN. statistical data visualization. 2021. Disponível em: <https://seaborn.pydata.org>. Acesso em: 1 set. 2021.