

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

**BRUNO CUNHA ZAGO**

**USO DE MACHINE LEARNING PARA PREVER ACIDENTES EM RODOVIAS  
BRASILEIRAS COM FOCO EM CONTRATOS DE MANUTENÇÃO**

Belo Horizonte

2023

**BRUNO CUNHA ZAGO**

**USO DE MACHINE LEARNING PARA PREVER ACIDENTES EM RODOVIAS  
BRASILEIRAS COM FOCO EM CONTRATOS DE MANUTENÇÃO**

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  
2023

## SUMÁRIO

<b>1. Introdução .....</b>	<b>4</b>
<b>1.1. Contextualização .....</b>	<b>4</b>
<b>1.2. O problema proposto .....</b>	<b>6</b>
<b>1.3. Objetivos.....</b>	<b>7</b>
<b>2. Coleta de Dados.....</b>	<b>7</b>
<b>3. Processamento/Tratamento de Dados .....</b>	<b>18</b>
<b>4. Análise e Exploração dos Dados .....</b>	<b>32</b>
<b>5. Criação de Modelos de Machine Learning.....</b>	<b>41</b>
<b>6. Interpretação dos Resultados .....</b>	<b>53</b>
<b>7. Apresentação dos Resultados .....</b>	<b>57</b>
<b>8. Links .....</b>	<b>58</b>
<b>REFERÊNCIAS .....</b>	<b>59</b>
<b>APÊNDICE .....</b>	<b>60</b>

## **1. Introdução**

### **1.1. Contextualização**

No Brasil, desde a década de 1920, quando as primeiras estradas federais foram criadas, o governo federal é responsável pela construção e manutenção das rodovias federais. Com o início do desenvolvimento econômico do país, na década de 1950, a malha rodoviária federal cresceu significativamente, com a construção de importantes rodovias, tais como a BR-101, BR-116 e BR-040, que conectam regiões do país de norte a sul e de leste a oeste. Nos anos seguintes, a rede de rodovias federais continuou a ser expandida e modernizada, com a construção de pontes, viadutos e túneis, além da adoção de tecnologias para melhorar a segurança viária. No entanto, apesar dos avanços, a malha rodoviária federal brasileira ainda apresenta desafios em termos de infraestrutura, segurança e logística, especialmente em regiões mais afastadas e de difícil acesso.

O Brasil é um dos países com maior concentração rodoviária de transporte de cargas e passageiros entre as principais economias mundiais, e mais de 70% da produção nacional é transportada por suas rodovias. Atualmente, a malha rodoviária brasileira possui 1.721 milhões de quilômetros (Fonte: Anuário CNT do Transporte 2021), sendo que apenas 213 mil estão pavimentados (Fonte: Anuário CNT do Transporte 2021). A malha rodoviária federal corresponde a 74.1 mil km (Fonte: SNV 202301B), dos quais 65.8 km são pavimentados (Fonte: SNV 202301B) e 8.2 km não pavimentados (Fonte: SNV 202301B). Em 2018, o investimento público federal em manutenção, duplicação, adequação e construção de rodovias somou R\$ 7,65 bilhões, visando aumentar a capacidade de tráfego, ordenar o trânsito urbano e garantir segurança e conforto para os usuários. Desse total, R\$ 4,3 bilhões são destinados à manutenção da malha federal administrada pelo DNIT.

No entanto, uma pesquisa realizada pela Confederação Nacional do Transporte (CNT) em 2020 mostrou que 61,9% das rodovias avaliadas apresentaram algum tipo de problema no estado geral (Fonte: Anuário CNT do Transporte 2021).

Os acidentes em rodovias são uma questão de grande preocupação em todo o mundo, causando mortes, lesões e prejuízos materiais. No Brasil, a preocupação com a segurança no trânsito é constante e os acidentes em rodovias são uma das principais causas de mortes violentas no país. Desde a década de 1990, a Polícia Rodoviária Federal (PRF) registra os acidentes em rodovias no Brasil, e os dados mostram que o número de acidentes vem apresentando um aumento significativo ao longo dos anos, em decorrência do aumento da frota de veículos e do fluxo de pessoas e mercadorias em outras modalidades de transporte, como ferrovias e hidrovias, que poderiam contribuir para uma maior diversificação do sistema de transporte e redução da dependência das rodovias.

Além disso, é importante destacar que a construção e manutenção de infraestrutura de transporte requerem altos investimentos, que nem sempre são possíveis em um contexto de crise econômica e orçamento limitado. Nesse sentido, é fundamental a busca por soluções criativas e inovadoras, que permitam maximizar os recursos disponíveis e garantir a eficiência e segurança do sistema de transporte.

Manter as rodovias em boas condições é essencial para garantir a segurança no trânsito e evitar acidentes. Estradas mal conservadas, com buracos, falta de sinalização e outras irregularidades, comprometem a segurança dos motoristas e passageiros, podendo levar a acidentes graves e prejuízos materiais. Por isso, é necessário investir em manutenção e conservação das rodovias para garantir a segurança e a fluidez do trânsito, além de aumentar a vida útil do pavimento e reduzir custos de manutenção a longo prazo.

De acordo com um estudo da CNT, intitulado "Acidentes Rodoviários e Infraestrutura", em locais com placas de sinalização legíveis, o índice de mortes em acidentes é de 7,1%, enquanto em locais com placas ilegíveis esse número sobe para 20,7%, quase três vezes mais. Em trechos sem placas indicativas de limite de velocidade, o índice de mortes por 10 km de extensão é de 19,9, enquanto em trechos com sinalização adequada esse número cai para 10,2. Além disso, aproximadamente 50% dos acidentes ocorrem em trechos com problemas ou falta de pintura na faixa.

É importante ressaltar que fatores como a condição do motorista, do veículo, institucional e as condições climáticas podem ser decisivos na ocorrência de um acidente. Entretanto, os dados apontam que nem sempre a imprudência dos motoristas é a única causa dos acidentes, como frequentemente relatado nos boletins de ocorrência. Por isso, é essencial investir em manutenção e sinalização adequadas nas rodovias para garantir a segurança dos usuários e reduzir o número de acidentes nas estradas.

## **1.2. O problema proposto**

O trânsito rodoviário no Brasil é um tema preocupante e que demanda soluções efetivas para reduzir a quantidade de acidentes. Com isso em mente, este trabalho tem como objetivo prever o número de acidentes em rodovias federais causados por problemas na via, utilizando dados do DATATRAN, que contém estatísticas da Polícia Rodoviária Federal, e do SNV, que contém informações detalhadas sobre a malha viária federal brasileira.

A ideia é identificar padrões de ocorrência de acidentes em rodovias brasileiras e, a partir desses dados, estudar modelos de machine learning para estimar o número de acidentes por meio dos dados de contratos de manutenção rodoviária. Esses dados serão fornecidos por uma empresa prestadora de serviços de manutenção rodoviária em um arquivo CSV.

Os segmentos do SNV foram criados pelo DNIT para organizar e classificar as rodovias brasileiras de acordo com suas características. Cada segmento do SNV será avaliado para determinar o número de acidentes em cada um deles. Os contratos de manutenção são licitados com base na tabela do SNV vigente na época da licitação, podendo incluir um ou mais segmentos em um mesmo contrato.

Para este estudo, serão utilizados dados de acidentes ocorridos no estado de Minas Gerais durante o período de 2007 a 2022 e os contratos de manutenção avaliados terão período de 2007 a 2021. Com base nessa análise, será possível criar uma ferramenta que estima o número de acidentes com base nos dados de contratos de manutenção, contribuindo para a redução da ocorrência de acidentes em rodovias brasileiras.

### 1.3. Objetivos

O objetivo é utilizar dados do DATATRAN e do SNV para prever o número de acidentes em rodovias federais causados por problemas na via e identificar padrões de ocorrência de acidentes em rodovias brasileiras.

Em seguida, pretende-se estudar modelos de machine learning para estimar o número de acidentes por meio dos dados de contratos de manutenção rodoviária. Para isso, cada segmento do SNV será avaliado para determinar o número de acidentes em cada um deles.

O objetivo final é criar uma ferramenta que estima o número de acidentes com base nos dados de contratos de manutenção. Isso permitirá identificar quais segmentos apresentam maior número de acidentes e, assim, propor ações preventivas em locais específicos, auxiliando o DNIT e as empresas contratadas na tomada de decisão quanto à alocação de recursos e ações preventivas em trechos de rodovias que apresentam maior risco de acidentes.

## 2. Coleta de Dados

Durante a etapa de coleta de dados, o objetivo é reunir o conjunto de dados necessários para o treinamento do modelo de machine learning. As bases de dados foram coletadas tanto pela internet quanto diretamente com uma empresa privada.

### 2.1. Base de acidentes da Polícia Rodoviária Federal - DATATRAN

Os dados do datatran foram disponibilizados no portal da PRF através do link "<https://www.gov.br/prf/pt-br/acesso-a-informacao/dados-abertos/dados-abertos-acidentes>". Os dados foram obtidos no dia 06/04/2023.

Os arquivos contendo os dados estão disponíveis em arquivos de extensão ".zip" e são agrupados por ano. Dentro dos arquivos zipados, há arquivos do tipo ".csv".

Para baixar os arquivos, utilizou-se a biblioteca requests, e para extrair os arquivos zip, utilizou-se a biblioteca zipfile. Os nomes dos arquivos baixados são especificados em uma lista file\_names e os URLs dos arquivos são especificados em uma lista correspondente urls.

**Figura 1 - código para baixar e extrair os arquivos**

```
# Baixando e extraindo o arquivo
extract_path = r"C:\Users\bcz87" #caminho de extração
urls = [
    "https://drive.google.com/u/0/uc?id=1PRQjuV5gOn_nn6UNvaJyVURDIbSAK4-&export=download",
    "https://drive.google.com/u/0/uc?id=12xH8LX9aN2gObR766YN3cMcuycwY3Dz&export=download",
    "https://drive.google.com/u/0/uc?id=1esu6IiH5VTxfoedv6DBGDd01Gvi8785&export=download",
    "https://drive.google.com/u/0/uc?id=1pN3fn2wY34GH6cY-gKfbxRJJBFE01b_1&export=download",
    "https://drive.google.com/u/0/uc?id=1cM4IgGMIiR-u4gBIH5IEe3DcvBvUzedi&export=download",
    "https://drive.google.com/u/0/uc?id=1HPLwt5f_14RIX3tkjI4tUXyZ0ev52W0N&export=download",
    "https://drive.google.com/u/0/uc?id=16qooQ1_ySoW61CrtsBbreBVNPY1EkoYm&export=download",
    "https://drive.google.com/u/0/uc?id=1DyqR5FFcwGsamSag-fGm13feQt0Y-3Da&export=download",
    "https://drive.google.com/u/0/uc?id=1FpF5wTBsRDkEhLm3z2g8XDIXr9S09Uk8&export=download",
    "https://drive.google.com/u/0/uc?id=1p_7lw9RzkINfscYAZSmc-Z9Ci4ZPJyEr&export=download",
    "https://drive.google.com/u/0/uc?id=18Vz2prqKSLthrMmW-73vr0iDmKTL6xE&export=download",
    "https://drive.google.com/u/0/uc?id=1HHhgLF-kSR6Gde2q0aTXL3T5ieD33hpG&export=download",
    "https://drive.google.com/u/0/uc?id=1_yU6FRh8M7USjChQwyF20NtY48GTmEX&export=download",
    "https://drive.google.com/u/0/uc?id=1qkVatg0pC_zosuBs0NCsgEXDJvBbnTYC&export=download",
    "https://drive.google.com/u/0/uc?id=1_OSeHlyKJw8cIhMS_JzSg1R1YX8k6vSG&export=download",
    "https://drive.google.com/u/0/uc?id=1EFpZF5F6cB0DOhd2Uxnj7X948WE69a8e&export=download"
]
file_names = [
    "datatran2022.zip",
    "datatran2021.zip",
    "datatran2020.zip",
    "datatran2019.zip",
    "datatran2018.zip",
    "datatran2017.zip",
    "datatran2016.zip",
    "datatran2015.zip",
    "datatran2014.zip",
    "datatran2013.zip",
    "datatran2012.zip",
    "datatran2011.zip",
    "datatran2010.zip",
    "datatran2009.zip",
    "datatran2008.zip",
    "datatran2007.zip"
]

for url, file_name in zip(urls, file_names):
    response = requests.get(url)
    open(file_name, "wb").write(response.content)

    with zipfile.ZipFile(file_name, 'r') as zip_ref:
        zip_ref.extractall(extract_path)
```

Para carregar os arquivos, foi definida uma lista com o nome dos arquivos a serem carregados e o caminho onde os arquivos estão localizados. Foi criada uma lista vazia para armazenar os dataframes e, em seguida, foi feito um loop em cada arquivo da lista, lendo o arquivo usando a biblioteca Pandas e armazenando o dataframe na lista de dataframes. Por fim, todos os dataframes armazenados na lista dfs foram concatenados em um único dataframe.



Figura 2 - código para carregar os arquivos

```
# Carregando arquivo
# Lista com o nome dos arquivos
file_names = ['datatran2022.csv', 'datatran2021.csv', 'datatran2020.csv', 'datatran2019.csv',
              'datatran2018.csv', 'datatran2017.csv', 'datatran2016.csv', 'datatran2015.csv',
              'datatran2014.csv', 'datatran2013.csv', 'datatran2012.csv', 'datatran2011.csv',
              'datatran2010.csv', 'datatran2009.csv', 'datatran2008.csv', 'datatran2007.csv']

# Caminho dos arquivos
path = r"C:\Users\bcz87\\"

# Lista para armazenar os dataframes
dfs = []

# Loop para ler cada arquivo e armazená-lo na lista dfs
for file in file_names:
    df = pd.read_csv(path + file, sep=';', decimal=',', encoding = 'cp1252', low_memory=False)
    dfs.append(df)

# Concatenação dos dataframes na mesma tabela
datatran = pd.concat(dfs, ignore_index=True)

# Exibir as primeiras linhas da tabela
print(datatran.head())
```

Figura 3 - primeiras linhas do dataframe

```
id data_inversa dia_semana horario uf br km \
0 405151.0 2022-01-01 sábado 01:35:00 PI 316.0 415.0
1 405158.0 2022-01-01 sábado 02:40:00 PR 116.0 33.0
2 405172.0 2022-01-01 sábado 05:22:00 MS 163.0 393.0
3 405203.0 2022-01-01 sábado 07:00:00 RJ 101.0 457.0
4 405207.0 2022-01-01 sábado 09:00:00 MG 40.0 508.3

municipio causa_acidente \
0 MARCOLANDIA Ingestão de álcool pelo condutor
1 CAMPINA GRANDE DO SUL Ingestão de álcool pelo condutor
2 NOVA ALVORADA DO SUL Condutor deixou de manter distância do veículo...
3 ANGRA DOS REIS Reação tardia ou ineficiente do condutor
4 RIBEIRAO DAS NEVES Acumulo de água sobre o pavimento

tipo_acidente ... ilesos ignorados feridos veiculos \
0 Colisão traseira ... 1 0 1 2
1 Tombamento ... 0 0 1 1
2 Colisão traseira ... 1 0 1 2
3 Colisão frontal ... 1 0 1 2
4 Saída de leito carroçável ... 3 0 0 1

latitude longitude regional delegacia uop ano
0 -7.4328 -40.682619 SPRF-PI DEL04-PI UOP03-DEL04-PI NaN
1 -25.114403 -48.846755 SPRF-PR DEL01-PR UOP02-DEL01-PR NaN
2 -21.228445 -54.456296 SPRF-MS DEL02-MS UOP01-DEL02-MS NaN
3 -23.031498 -44.177153 SPRF-RJ DEL03-RJ UOP02-DEL03-RJ NaN
4 -19.760612 -44.134754 SPRF-MG DEL02-MG UOP01-DEL02-MG NaN

[5 rows x 31 columns]
```

Além disso, é possível visualizar informações sobre o dataframe, incluindo o número de linhas e colunas, o nome e o tipo de cada coluna.

**Figura 4 - informações sobre o dataframe**

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1981217 entries, 0 to 1981216
Data columns (total 31 columns):
#   Column                Dtype
---  -
0   id                    float64
1   data_inversa          object
2   dia_semana            object
3   horario               object
4   uf                   object
5   br                   object
6   km                   object
7   municipio            object
8   causa_acidente       object
9   tipo_acidente        object
10  classificacao_acidente object
11  fase_dia              object
12  sentido_via           object
13  condicao_meteorologica object
14  tipo_pista            object
15  tracado_via           object
16  uso_solo              object
17  pessoas               int64
18  mortos                int64
19  feridos_leves         int64
20  feridos_graves       int64
21  ileso                 int64
22  ignorados             int64
23  feridos               int64
24  veiculos              int64
25  latitude              object
26  longitude              object
27  regional              object
28  delegacia             object
29  uop                   object
30  ano                   float64
dtypes: float64(2), int64(8), object(21)
memory usage: 468.6+ MB

```

A PRF disponibiliza no site um dicionário com o nome e a descrição de cada variável. Anexando-se o tipo de dado, temos a tabela 1.

**Tabela 1 - Colunas do dataframe do Datatran**

Nome da Coluna	Descrição	Tipo
id	Variável com valores numéricos, representando o identificador do acidente.	float64
data_inversa	Data da ocorrência no formato dd/mm/aaaa.	object
dia_semana	Dia da semana da ocorrência. Ex.: Segunda, Terça, etc.	object
horario	Horário da ocorrência no formato hh:mm:ss.	object
uf	Unidade da Federação. Ex.: MG, PE, DF, etc.	object
br	Variável com valores numéricos representando o identificador da BR do acidente.	object
km	Identificação do quilômetro onde ocorreu o acidente, com valor mínimo de 0,1 km e com a casa decimal separada por ponto.	object
municipio	Nome do município de ocorrência do acidente.	object
causa_acidente	Identificação da causa presumível do acidente. Ex.: Falta de atenção, Velocidade incompatível, etc.	object
tipo_acidente	Identificação do tipo de acidente. Ex.: Colisão frontal, Saída de pista, etc.	object

		ject
classificacao_acidente	Classificação quanto à gravidade do acidente: Sem Vítimas, Com Vítimas Feridas, Com Vítimas Fatais e Ignorado.	object
fase_dia	Fase do dia no momento do acidente. Ex. Amanhecer, Pleno dia, etc.	object
sentido_via	Sentido da via considerando o ponto de colisão: Crescente e decrescente.	object
condicao_meteorologica	Condição meteorológica no momento do acidente: Céu claro, chuva, vento, etc.	object
tipo_pista	Tipo da pista considerando a quantidade de faixas: Dupla, simples ou múltipla.	object
tracado_via	Descrição do traçado da via: reta, curva ou cruzamento.	object
uso_solo	Descrição sobre as características do local do acidente: Urbano ou rural.	object
peessoas	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 (é a soma dos feridos leves com os graves).	int64
veiculos	Total de veículos envolvidos na ocorrência.	int64
latitude	Latitude do local do acidente em formato geodésico decimal.	object
longitude	Longitude do local do acidente em formato geodésico decimal.	object
regional	Não informado	object
delegacia	Não informado	object
uop	Não informado	object
ano	Não informado	float64

## 2.2. Base de informações sobre as rodovias federais brasileiras - SNV

Os dados do Sistema Nacional de Viação (SNV) são disponibilizados no portal do Departamento Nacional de Infraestrutura de Transportes (DNIT) por meio do link <https://www.gov.br/dnit/pt-br/assuntos/atlas-e-mapas/pnv-e-snv>. Os dados foram acessados em 06/04/2023.

O arquivo disponibilizado possui extensão ".zip" e contém arquivos em formato PDF, além do arquivo de nosso interesse, em formato ".csv".

Para obter o arquivo, utilizou-se a função "get" da biblioteca "requests", passando a URL do arquivo como parâmetro. Em seguida, o conteúdo do arquivo foi descompactado usando a biblioteca "zipfile" e extraído para uma pasta local.

**Figura 5 - Código para baixar e extrair os arquivos**

```
# Baixando o arquivo
url = "https://servicos.dnit.gov.br/dnitcloud/index.php/s/TYqwT6cQ2b7Tq5Q/download"
file_name = "pub_202301B.zip"

r = requests.get(url)
z = zipfile.ZipFile(io.BytesIO(r.content))
z.extractall()
```

Para carregar o arquivo em um dataframe pandas, utilizou-se o método "read\_excel", passando o caminho do arquivo como parâmetro e indicando a exclusão das duas primeiras linhas.

**Figura 6 - Código para carregar o arquivo**

```
# Lendo o arquivo
arquivo = "C:\\Users\\bcz87\\pub_202301B\\SNV_202301B.xls"
snv = pd.read_excel(arquivo, skiprows=[0, 1])

# Exibir as primeiras linhas da tabela
print(snv.head())
```

Para visualizar as primeiras linhas do dataframe, utilizou-se o método "head".

Figura 7 - Primeiras linhas do dataframe

```

BR UF Tipo de trecho Desc Coinc Código \
0 10 DF Eixo Principal - 010BDF0010
1 10 DF Eixo Principal - 010BDF0015
2 10 DF Eixo Principal - 010BDF0016
3 10 DF Eixo Principal - 010BDF0018
4 10 DF Eixo Principal - 010BDF0020

Local de Início Local de Fim \
0 ENTR BR-020(A)/030(A)/450/DF-001 (BRASÍLIA) ENTR DF-440
1 ENTR DF-440 ACESSO I SOBRADINHO
2 ACESSO I SOBRADINHO ACESSO II SOBRADINHO
3 ACESSO II SOBRADINHO ENTR DF-230
4 ENTR DF-230 ENTR DF-128

km inicial km final Extensão Superfície Federal Obras \
0 0.0 2.4 2.4 DUP NaN
1 2.4 6.0 3.6 DUP NaN
2 6.0 8.3 2.3 DUP NaN
3 8.3 18.2 9.9 DUP NaN
4 18.2 22.0 3.8 DUP NaN

Federal Coincidente Administração Ato legal \
0 010BDF0010;020BDF0010;030BDF0010 Convênio Adm.Federal/Estadual NaN
1 010BDF0015;020BDF0015;030BDF0015 Convênio Adm.Federal/Estadual NaN
2 010BDF0016;020BDF0016;030BDF0016 Convênio Adm.Federal/Estadual NaN
3 010BDF0018;020BDF0018;030BDF0018 Convênio Adm.Federal/Estadual NaN
4 010BDF0020;020BDF0020;030BDF0020 Convênio Adm.Federal/Estadual NaN

Estadual Coincidente Superfície Est. Coincidente Jurisdição Superfície \
0 NaN NaN NaN Federal PAV
1 NaN NaN NaN Federal PAV
2 NaN NaN NaN Federal PAV
3 NaN NaN NaN Federal PAV
4 NaN NaN NaN Federal PAV

Unidade Local
0 Brasília
1 Brasília
2 Brasília
3 Brasília
4 Brasília

```

Já as informações sobre o dataframe, incluindo o número de linhas e colunas, o nome e o tipo de cada coluna e a quantidade de valores não nulos em cada coluna, foram obtidas por meio do método "info".

Figura 8 - Informações sobre o dataframe

```

# Exibindo informações
snv.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7429 entries, 0 to 7428
Data columns (total 20 columns):
#   Column                                Non-Null Count  Dtype
---  ---
0   BR                                     7429 non-null   int64
1   UF                                     7429 non-null   object
2   Tipo de trecho                        7429 non-null   object
3   Desc Coinc                            7429 non-null   object
4   Código                                7429 non-null   object
5   Local de Início                       7429 non-null   object
6   Local de Fim                          7429 non-null   object
7   km inicial                            7429 non-null   float64
8   km final                              7429 non-null   float64
9   Extensão                              7429 non-null   float64
10  Superfície Federal                    7429 non-null   object
11  Obras                                 210 non-null    object
12  Federal Coincidente                   7429 non-null   object
13  Administração                         7429 non-null   object
14  Ato legal                             0 non-null      float64
15  Estadual Coincidente                  1554 non-null   object
16  Superfície Est. Coincidente           1551 non-null   object
17  Jurisdição                           7429 non-null   object
18  Superfície                            7429 non-null   object
19  Unidade Local                         5243 non-null   object
dtypes: float64(4), int64(1), object(15)
memory usage: 1.1+ MB

```

Uma das informações importantes para entender as colunas do dataframe está disponível em um dos arquivos PDF presentes no arquivo zipado. Com base nesse documento, foi possível montar a Tabela 2, que descreve as colunas do dataframe do SNV.

**Tabela 2 - Colunas do dataframe do SNV**

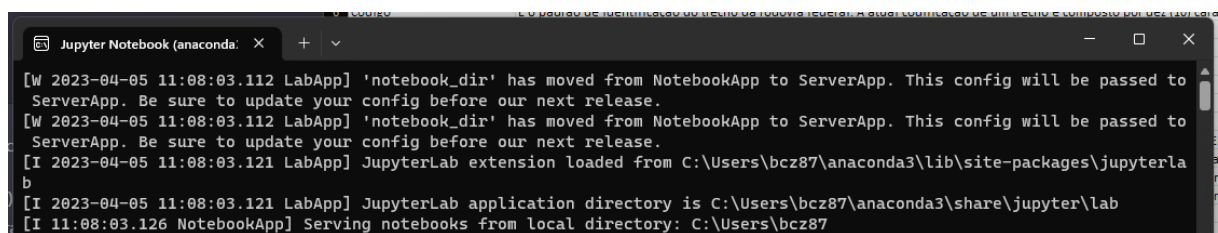
Nome da Coluna	Descrição	Tipo
BR	Código da rodovia cuja administração cabe ao Poder Executivo do País.	int64
UF	Unidade da Federação onde o segmento rodoviário está localizado.	object
Tipo de trecho	As rodovias pertencentes ao Subsistema Rodoviário Federal (SRF), parte integrante do SNV, são classificadas em dois tipos: trecho principal e trecho acessório. O SRF é composto por todas as rodovias administradas pela União, direta ou indiretamente, nos termos da Lei nº 12.379/2011.	object
Desc Coinc	Existem alguns casos de superposições de duas ou mais rodovias. Nestes casos, usualmente, adotase a rodovia representativa do trecho superposto à rodovia com o menor número, tendo em vista a operacionalidade dos sistemas computadorizados.	object
Código	É o padrão de identificação do trecho da rodovia federal. A atual codificação de um trecho é composto por dez (10) caracteres.	object
Local de Início	Campo indicativo do início do trecho rodoviário federal.	object
Local de Fim	Campo indicativo do fim do trecho rodoviário federal.	object
km inicial	Marco quilométrico inicial do trecho informado.	float64
km final	Marco quilométrico final do trecho informado.	float64
Extensão	Subtração do marco quilométrico final do trecho pelo marco quilométrico inicial.	float64
Superfície Federal	O campo de "SUPERFÍCIE FEDERAL" da planilha do SNV indica a situação da superfície da rodovia federal.	object
Obras	Os campos referentes a obras das rodovias são classificadas em três distintas descrições, que são listadas a seguir. Em Obras de Duplicação – EOD, Em Obras de Implantação – EOI e Em Obras de Pavimentação – EOP.	object
Federal Coincidente	Existem alguns casos de superposições de dois ou mais trechos de rodovias federais. Nestes casos, este campo indica o código da(s) rodovia(as) coincidente(s). Quando há mais de uma rodovia coincidente, um ponto e vírgula (;) separa as rodovias no campo.	object
Administração	Dividese em: Federal, Estadual ou Distrital, Municipal, Concessão Federal ou Concessão Estadual e Convênio de Administração	object
Ato legal	O campo "Ato Legal" indica os trechos inclusos na Medida Provisória/082. O trecho é indicado como pertencente à referida medida provisória quando não está nulo.	float64
Estadual Coincidente	São rodovias construídas pelos Estados ou municípios sobre a diretriz de uma rodovia federal planejada.	object
Superfície Est. Coincidente	Neste campo informase apenas a superfície da rodovia estadual coincidente com a rodovia federal.	object
Jurisdição	São as Rodovias Federais, cujos trechos estão sob regime de administração direta, ou delegada pelo DNIT aos Estados, Distrito Federal e Municípios.	object
Superfície	Neste campo é indicada a classificação das rodovias.	object

		je t
Unidade Local	Descreve o nome do município onde está a Unidade Local responsável por aquele trecho específico.	ob je t

### 2.3. Base de informações sobre contratos de manutenção rodoviária de uma determinada empresa - Obra

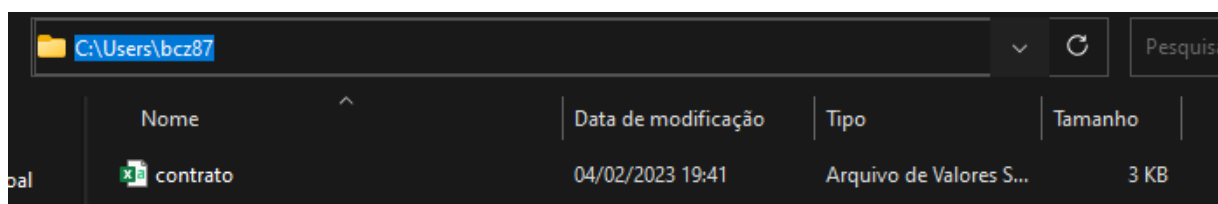
Os dados de contratos de manutenção foram fornecidos por uma empresa e estão no formato CSV. O arquivo foi salvo no diretório de trabalho do Jupyter Notebook através do Windows Explorer.

Figura 9 - Diretório padrão do Jupyter Notebook



```
[W 2023-04-05 11:08:03.112 LabApp] 'notebook_dir' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[W 2023-04-05 11:08:03.112 LabApp] 'notebook_dir' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[I 2023-04-05 11:08:03.121 LabApp] JupyterLab extension loaded from C:\Users\bcz87\anaconda3\lib\site-packages\jupyterlab
[I 2023-04-05 11:08:03.121 LabApp] JupyterLab application directory is C:\Users\bcz87\anaconda3\share\jupyter\lab
[I 11:08:03.126 NotebookApp] Serving notebooks from local directory: C:\Users\bcz87
```

Figura 10 - Local de salvamento do arquivo



Para carregar o arquivo CSV em um objeto pandas DataFrame, usou-se a função `read_csv()`, passando o caminho do arquivo como parâmetro.

Figura 11 - Código para carregar o arquivo

```
# Carregando o arquivo
file_path = r'C:\Users\bcz87\contrato.csv'
obra = pd.read_csv(file_path, sep=';', decimal=',')

# Exibir as primeiras linhas da tabela
print(obra.head())
```

Para verificar as primeiras linhas do dataframe, foi utilizado o método `head()`, que retorna as cinco primeiras linhas por padrão.

**Figura 12 - Primeiras linhas do dataframe**

```

      obra  uf  data_inicio  data_fim  br  km_inicial  km_final
0  G002  MG      42186      43719  267      62.0      98.7
1  G003  MG      42359      44260  458      97.2     147.2
2  G004  MG      42359      44183   50      65.5      77.0
3  G005  MG      42403      44221  262       0.0      72.2
4  G006  MG      42403      42909  452      58.4      91.8

```

Para obter informações sobre o dataframe, como o número de linhas e colunas, nome e tipo de cada coluna, e quantidade de valores não nulos em cada coluna, usou-se o método `info()`.

**Figura 13 - Informações sobre o dataframe**

```

# Exibindo informações
obra.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62 entries, 0 to 61
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  -
0   obra            62 non-null    object
1   uf              62 non-null    object
2   data_inicio     62 non-null    int64
3   data_fim        62 non-null    int64
4   br              62 non-null    int64
5   km_inicial      62 non-null    float64
6   km_final        62 non-null    float64
dtypes: float64(2), int64(3), object(2)
memory usage: 3.5+ KB

```

A descrição de cada coluna foi obtida diretamente com a empresa e está apresentada na Tabela 3.

**Tabela 3 - Colunas do dataframe da obra**

Nome da Coluna	Descrição	Tipo
obra	Código interno de obra	object
uf	Unidade Federativa da rodovia	object
data_inicio	Data de início da execução dos serviços	int64



data_fim	Data de fim da execução dos serviços	int64
br	Número da Rodovia	int64
km_inicial	Quilometro inicial da rodovia	float64
km_final	Quilometro final da rodovia	float64

### 3. Processamento/Tratamento de Dados

Na etapa de Processamento/Tratamento de Dados, os dados brutos são transformados e preparados para análise. Isso inclui a limpeza e validação dos dados, a eliminação de dados ausentes ou duplicados, a padronização de formatos e a conversão de tipos de dados. Também pode envolver a criação de novas variáveis e a combinação de dados de diferentes fontes. O objetivo é preparar os dados para que possam ser utilizados com eficácia nas etapas subsequentes de análise e modelagem. É uma etapa fundamental para garantir a qualidade e confiabilidade dos resultados obtidos.

#### 3.1. Limpeza, validação e conversão de tipos de dados

##### 3.1.1. Dataframe Datatran

Na Figura 4, é apresentado o tipo de dado de cada coluna. No entanto, alguns tipos de dados não coincidem com o tipo de dado necessário para a análise. Assim, foram realizadas conversões de tipos de dados.

Primeiramente, a coluna 'br' é convertida para números decimais usando a função 'pd.to\_numeric' com o argumento 'errors='coerce' para converter valores que não possam ser convertidos em NaN. Em seguida, as linhas que contêm valores NaN na coluna 'br' são removidas usando o método 'dropna' com o argumento 'subset=['br']'. Por fim, a coluna 'br' é convertida para números inteiros usando o método 'astype' com o argumento 'int'. O objetivo é garantir que a coluna 'br' seja convertida corretamente para números e que as linhas com valores inválidos sejam removidas, tornando os dados mais limpos e adequados para análise.

**Figura 14 - Conversão do tipo de dado da coluna 'br'**

```
# Convertendo a coluna br para números decimais
datatran['br'] = pd.to_numeric(datatran['br'], errors='coerce')

# Retirando linhas que contêm valores NaN da coluna br
datatran = datatran.dropna(subset=['br'])

# Convertendo a coluna br para números inteiros
datatran['br'] = datatran['br'].astype(int)
```

Em seguida, o próximo código converte os valores da coluna 'km' em números decimais usando a função 'pd.to\_numeric', com a opção 'errors='coerce'' usada para substituir qualquer valor não numérico por NaN. As linhas que contêm valores NaN na coluna 'km' são removidas usando o método 'dropna' do pandas, com o parâmetro 'subset' especificando a coluna 'km'. A conversão é útil para trabalhar com dados numéricos e tornar os dados adequados para análise.

**Figura 15 - Conversão do tipo de dado da coluna 'km'**

```
# Convertendo a coluna km para números decimais
datatran['km'] = pd.to_numeric(datatran['km'], errors='coerce')

# Retirando linhas que contêm valores NaN da coluna km
datatran = datatran.dropna(subset=['km'])
```

A coluna 'data\_inversa' do DataFrame é convertida para o tipo datetime usando a função 'to\_datetime()' do pandas. Essa conversão é útil para trabalhar com dados de datas e horários em análises de dados.

**Figura 16 - Conversão do tipo de dado da coluna data\_inversa**

```
# converter a coluna 'data_inversa' para o tipo datetime
datatran['data_inversa'] = pd.to_datetime(datatran['data_inversa'])
```

Na Figura 17, é apresentado o tipo de dado corrigido para as colunas alvo, demonstrando que os dados estão com as unidades corretas para as etapas subsequentes.

Figura 17 - Tipo de dado corrigido

```
# Exibindo informações
datatran.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1980323 entries, 0 to 1981216
Data columns (total 31 columns):
#   Column              Dtype
---  -
0    id                  float64
1    data_inversa         datetime64[ns]
2    dia_semana           object
3    horario              object
4    uf                  object
5    br                  int32
6    km                  float64
7    municipio            object
8    causa_acidente       object
9    tipo_acidente        object
10   classificacao_acidente object
11   fase_dia              object
12   sentido_via           object
13   condicao_meteorologica object
14   tipo_pista            object
15   tracado_via           object
16   uso_solo              object
17   pessoas              int64
18   mortos               int64
19   feridos_leves         int64
20   feridos_graves        int64
21   ileso                 int64
22   ignorados             int64
23   feridos               int64
24   veiculos              int64
25   latitude              object
26   longitude             object
27   regional              object
28   delegacia             object
29   uop                  object
30   ano                  float64
dtypes: datetime64[ns](1), float64(3), int32(1), int64(8), object(18)
memory usage: 475.9+ MB
```

Como o objetivo do estudo é prever acidentes relacionados à via, é necessário estudar os valores únicos da coluna 'causa\_acidente'. A função 'unique()' é utilizada para imprimir na tela os valores únicos encontrados na coluna 'causa\_acidente' do DataFrame. Dentre os valores únicos, aqueles relacionados à via são selecionados e filtrados na tabela 'datatran' utilizando o método 'isin()', excluindo as linhas que não possuem a causa de acidente listada.

**Figura 18 - Valores únicos da coluna causa\_acidente**

```
# Valores únicos da coluna causa_acidente
print(datatran['causa_acidente'].unique())

['Ingestão de álcool pelo condutor'
'Condutor deixou de manter distância do veículo da frente'
'Reação tardia ou ineficiente do condutor'
'Acumulo de água sobre o pavimento' 'Mal súbito do condutor' 'Chuva'
'Ausência de reação do condutor' 'Manobra de mudança de faixa'
'Pista Escorregadia' 'Ausência de sinalização' 'Condutor Dormindo'
'Velocidade Incompatível'
'Acessar a via sem observar a presença dos outros veículos'
'Conversão proibida' 'Transitar na contramão' 'Ultrapassagem Indevida'
'Desrespeitar a preferência no cruzamento' 'Acostamento em desnível'
'Demais falhas mecânicas ou elétricas'
'Carga excessiva e/ou mal acondicionada' 'Problema com o freio'
'Ingestão de álcool e/ou substâncias psicoativas pelo pedestre'
'Curva acentuada' 'Estacionar ou parar em local proibido'
'Avárias e/ou desgaste excessivo no pneu'
'Pedestre cruzava a pista fora da faixa' 'Obstrução na via'
'Acesso irregular' 'Pista esburacada' 'Animais na Pista'
'Falta de acostamento' 'Entrada inopinada do pedestre'
'Acumulo de areia ou detritos sobre o pavimento'
'Pedestre andava na pista' 'Desvio temporário' 'Transitar no acostamento'
'Problema na suspensão' 'Objeto estático sobre o leito carroçável'
'Deficiência do Sistema de Iluminação/Sinalização'
'Trafegar com motocicleta (ou similar) entre as faixas'
'Condutor usando celular'
'Restrição de visibilidade em curvas horizontais'
'Ingestão de álcool ou de substâncias psicoativas pelo pedestre'
'Declive acentuado' 'Frear bruscamente' 'Iluminação deficiente'
'Área urbana sem a presença de local apropriado para a travessia de pedestres'
'Demais Fenômenos da natureza' 'Demais falhas na via'
'Faixas de trânsito com largura insuficiente' 'Obras na pista'
'Retorno proibido' 'Afundamento ou ondulação no pavimento'
'Falta de elemento de contenção que evite a saída do leito carroçável'
'Ingestão de substâncias psicoativas pelo condutor' 'Neblina'
'Condutor desrespeitou a iluminação vermelha do semáforo'
'Sistema de drenagem ineficiente' 'Acumulo de óleo sobre o pavimento'
'Sinalização mal posicionada' 'Pista em desnível' 'Transitar na calçada'
'Fumaça' 'Redutor de velocidade em desacordo'
'Deixar de acionar o farol da motocicleta (ou similar)'
'Restrição de visibilidade em curvas verticais' 'Semáforo com defeito'
'Faróis desregulados' 'Modificação proibida' 'Participar de racha'
'Sinalização encoberta' 'Falta de Atenção à Condução'
'Não guardar distância de segurança' 'Defeito Mecânico no Veículo'
'Desobediência às normas de trânsito pelo condutor' 'Ingestão de Álcool'
'Mal Súbito' 'Agressão Externa' 'Falta de Atenção do Pedestre'
'Defeito na Via' 'Desobediência às normas de trânsito pelo pedestre'
'Fenômenos da Natureza' 'Restrição de Visibilidade'
'Ingestão de Substâncias Psicoativas'
'Deficiência ou não Acionamento do Sistema de Iluminação/Sinalização do Veículo'
'Sinalização da via insuficiente ou inadequada' 'Outras'
'Falta de atenção' 'Ingestão de álcool' 'Velocidade incompatível'
'Ultrapassagem indevida' 'Dormindo' 'Desobediência à sinalização'
'Defeito mecânico em veículo' 'Defeito na via' '(null)']
```

Foram identificados os valores únicos da coluna 'causa\_acidente' relacionados à via, que incluem 'Animais na Pista', 'Defeito na Via', 'Fenômenos da Natureza', 'Objeto estático sobre o leito carroçável', 'Pista Escorregadia', 'Restrição de Visibilidade', 'Sinalização da via insuficiente ou inadequada', 'Ausência de sinalização' e 'Chuva'. Para analisar apenas os acidentes relacionados à via, foi realizada uma seleção dessas causas, definidas na lista 'causas', e a tabela 'datatran' foi filtrada pelas causas selecionadas usando o método isin(). Dessa forma, foram excluídas da tabela as linhas que não possuem a causa de acidente listada, tornando a análise mais precisa e focada nos acidentes de interesse.

**Figura 19 - Filtro pelas causas de acidentes**

```
# Seleção das causas de acidentes relacionadas a via
causas = ['Animais na Pista',
          'Defeito na Via',
          'Fenômenos da Natureza',
          'Objeto estático sobre o leito carroçável',
          'Pista Escorregadia',
          'Restrição de Visibilidade',
          'Sinalização da via insuficiente ou inadequada',
          'Ausência de sinalização',
          'Chuva']

# Filtro da tabela datatran pelas causas selecionadas
datatran = datatran[datatran['causa_acidente'].isin(causas)]
```

É adicionada uma nova coluna chamada 'mes\_ano' no DataFrame 'datatran', preenchida com valores no formato de mês e ano (MM/AAAA) da coluna 'data\_inversa', que foi previamente convertida para o tipo datetime usando a função 'pd.to\_datetime()'. Essa operação permite que os dados sejam agrupados por período mensal.

**Figura 20 - Definindo períodos mensais**

```
# Definindo períodos mensais
datatran['mes_ano'] = datatran['data_inversa'].dt.strftime('%m/%Y')
```

Por último, é salvo o dataframe em um arquivo CSV.

**Figura 21 - Salvando o arquivo concatenado em .csv**

```
# Salvando o arquivo concatenado em .csv
datatran.to_csv(r"C:\Users\bcz87\datatran.csv", index=False, sep=';', decimal=',', encoding = 'cp1252')
```

### 3.1.2. Dataframe SNV

No dataframe SNV, as colunas estão em seus tipos de dados esperados. Entretanto, é necessário padronizar o nome das colunas com as do dataframe DATATRAN, para evitar divergências entre variáveis correspondentes de diferentes dataframes. As colunas "BR", "UF", "km inicial", "km final", "Código" e "Extensão" são renomeadas para "br", "uf", "km\_inicial", "km\_final", "codigo" e "extensao", respectivamente.

Figura 22 - Renomeando colunas

```
# Renomeando colunas
snv = snv.rename(columns={"BR": "br",
                          "UF": "uf",
                          "km inicial": "km_inicial",
                          "km final": "km_final",
                          "Código": "codigo",
                          "Extensão": "extensao"})
```

Conforme observado na Figura 13, o dataframe de obra não inclui uma coluna para o tipo de trecho. No entanto, no SNV, o trecho principal é definido como a seção da rodovia estabelecida por lei que liga duas localidades descritas na relação criadora. Esse trecho inclui as faixas da rodovia, além de eventuais acostamentos, canteiros divisórios, obras de arte especiais (OAE) ou obras de arte correntes (OAC), interseções e outros dispositivos relacionados ao eixo da rodovia. Por outro lado, o trecho acessório representa a parte da rodovia que auxilia o trecho principal em sua operação e utilização. Os tipos de trechos acessórios incluem acesso, anel rodoviário, contorno, variante e travessia urbana, esta última não faz parte do eixo principal. É importante destacar que cada tipo de trecho tem sua própria codificação e regras de criação.

Devido a essa diversidade de tipos de trechos, é possível que haja duplicidade de dados nas colunas uf, br e km inicial quando o tipo de trecho é alterado. Se alocarmos os acidentes por trechos sem levar em consideração o tipo, podemos perder a precisão dos dados. Portanto, optou-se por filtrar a coluna tipo de trecho pelo valor "Eixo Principal", que representa 94,1% dos tipos de trecho. Isso permite que os dados sejam mais precisos e confiáveis ao serem analisados.

Figura 23 - Tipo de trecho filtrado

```
# Valores únicos da coluna Tipo de trecho
print(snv['Tipo de trecho'].unique())

['Eixo Principal' 'Acesso' 'Contorno' 'Travessia Urbana' 'Variante' 'Anel']

print(snv["Tipo de trecho"].value_counts())
print('/n')
print(snv["Tipo de trecho"].value_counts(normalize=True))

Eixo Principal      6987
Acesso              165
Contorno            153
Anel                48
Variante            44
Travessia Urbana    32
Name: Tipo de trecho, dtype: int64
/n
Eixo Principal      0.940503
Acesso              0.022210
Contorno            0.020595
Anel                0.006461
Variante            0.005923
Travessia Urbana    0.004307
Name: Tipo de trecho, dtype: float64
```

```
# Filtro da tabela snv pelo trecho do tipo Eixo Principi
snv = snv[snv["Tipo de trecho"] == "Eixo Principal"]
```

Por fim, a tabela SNV foi salva em um novo arquivo CSV denominado snv.csv.

Figura 24 - Salvamento do dataframe em arquivo CSV

```
# Salvar o resultado em um novo arquivo CSV
snv.to_csv('snv.csv', index=False, sep=';', decimal=',', encoding = 'cp1252')
```

### 3.1.2. Dataframe Obra

Realizamos a conversão das colunas "data\_inicio" e "data\_fim" do dataframe "obra" para o formato de data do Python. Para isso, utilizamos a função "apply" para aplicar uma função lambda em cada valor das colunas. Dentro da função lambda, a função "xlrd.xldate\_as\_datetime" foi utilizada para converter os valores da coluna, que estão em formato de data do Excel, para o formato de data do Python.



Figura 25 - Conversão das colunas data\_inicio e data\_fim

```
# converter as colunas 'data_inicio' e 'data_fim' para o formato de data do Python
obra['data_inicio'] = obra['data_inicio'].apply(lambda x: xlrd.xldate_as_datetime(x, 0))
obra['data_fim'] = obra['data_fim'].apply(lambda x: xlrd.xldate_as_datetime(x, 0))
```

```
# Exibindo informações
print(obra.dtypes)
```

```
obra                object
uf                  object
data_inicio    datetime64[ns]
data_fim        datetime64[ns]
br                int64
km_inicial       float64
km_final         float64
dtype: object
```

Posteriormente, criamos uma nova coluna chamada "extensao" no DataFrame "obra". Essa coluna contém a diferença entre as colunas "km\_final" e "km\_inicial", que representam as quilometragens final e inicial da obra, respectivamente. Com isso, a nova coluna "extensao" apresenta o valor da extensão total da obra em quilômetros.

Figura 26 - Criação da coluna extensao

```
# Criação da nova coluna extensão
obra['extensao'] = obra['km_final'] - obra['km_inicial']

# Exibir as primeiras linhas da tabela
print(obra.head())
```

	obra	uf	data_inicio	data_fim	br	km_inicial	km_final	extensao
0	G002	MG	2015-07-01	2019-09-11	267	62.0	98.7	36.7
1	G003	MG	2015-12-21	2021-03-05	458	97.2	147.2	50.0
2	G004	MG	2015-12-21	2020-12-18	50	65.5	77.0	11.5
3	G005	MG	2016-02-03	2021-01-25	262	0.0	72.2	72.2
4	G006	MG	2016-02-03	2017-06-23	452	58.4	91.8	33.4

Por fim, realizamos a exportação do dataframe para um arquivo CSV com o nome obra.csv.

Figura 27 - Salvamento do dataframe em arquivo CSV

```
# Salvar o resultado em um novo arquivo CSV
obra.to_csv('obra.csv', index=False, sep=';', decimal=',', encoding = 'cp1252')
```

### 3.2. Combinação de dados de diferentes dataframes

A etapa de combinação de dados é fundamental para unir informações de diferentes tabelas e obter uma estrutura de dados única para realizar análises completas e precisas.

Para começar, é necessário criar um novo dataframe combinado a partir da cópia do dataframe original, usando o método "copy()" do pandas. Em seguida, adiciona-se uma nova coluna chamada "index" ao dataframe, usando o método "reset\_index()" do pandas, que redefine o índice do dataframe para uma sequência numérica crescente, adicionando uma nova coluna com os valores antigos do índice.

**Figura 28 - Cópia do dataframe e criação da coluna index**

```
# Criar cópia do DataFrame
datatran_snv_obra = datatran.copy()

# criar coluna index
datatran_snv_obra = datatran_snv_obra.reset_index()
```

Em seguida, é necessário verificar em qual segmento de SNV ocorreu cada acidente e incluir uma nova coluna com o código correspondente. Para isso, os dataframes "datatran\_snv\_obra" e "snv" são combinados usando as colunas "uf" e "br" como referência, por meio da função "merge" do pandas. Depois, é feita uma seleção das linhas em que o valor da coluna "km" está dentro do intervalo das colunas "km\_inicial" e "km\_final".

Posteriormente, é feita uma seleção das colunas "index" e "codigo" do dataframe combinado e, em seguida, é realizada uma nova mesclagem entre os dataframes "datatran\_snv\_obra" e "merged", utilizando a coluna "index" como chave de junção e mantendo todas as linhas do dataframe "datatran\_snv\_obra".

**Figura 29 - Inclusão da coluna codigo**

```
# Merge das tabelas datatran e snv
merged = pd.merge(datatran_snv_obra, snv, on=['uf', 'br'])
merged = merged[(merged['km'] >= merged['km_inicial']) & (merged['km'] < merged['km_final'])]

# Selecionar apenas as colunas "index" e "codigo" da tabela merged
merged = merged[['index', 'codigo']]

# Mesclar as tabelas usando a coluna de índice como chave de junção
datatran_snv_obra = pd.merge(datatran_snv_obra, merged, on="index", how="left")
```

A próxima coluna a ser adicionada é a de obra, que corresponde ao código da obra no caso de acidentes que aconteceram dentro do período contratual e nos locais contratados para manutenção. Nesse sentido, é feito um merge das tabelas

"datatran\_snv\_obra" e "obra" usando as colunas "uf" e "br" como referência. Em seguida, são realizadas algumas filtrações na tabela "merged", como a data do acidente, a quilometragem da rodovia e a extensão da obra. Posteriormente, são selecionadas apenas as colunas "index" e "obra" da tabela "merged" e, por fim, é feito um novo merge das tabelas "datatran\_snv\_obra" e "merged" utilizando a coluna de índice como chave de junção e adicionando a coluna "obra" na tabela "datatran\_snv\_obra".

**Figura 30 - Inclusão da coluna obra**

```
# Merge das tabelas datatran e obra
merged = pd.merge(datatran_snv_obra, obra, on=['uf', 'br'])
merged = merged[(merged['data_inversa'] >= merged['data_inicio']) & (merged['data_inversa'] <= merged['data_fim']) & (merged['km'] > 0)]

# Selecionar apenas as colunas "index" e "obra" da tabela merged
merged = merged[['index', 'obra']]

# Mesclar as tabelas usando a coluna de índice como chave de junção
datatran_snv_obra = pd.merge(datatran_snv_obra, merged, on="index", how="left")
```

Por último, o dataframe é salvo em arquivo csv com o nome "datatran\_snv\_obra". Essas etapas são essenciais para a organização e análise adequada dos dados, permitindo obter insights precisos e relevantes para o estudo em questão.

**Figura 31 - Salvamento do arquivo**

```
# Salvar o resultado em um novo arquivo CSV
datatran_snv_obra.to_csv('datatran_snv_obra.csv', index=False, sep=';', decimal=',', encoding = 'cp1252')
```

### 3.3. Criação de novos dataframes

#### 3.3.1. Dataframe de acidentes por SNV

Vamos agora criar o DataFrame para nossa análise preditiva. Criamos uma nova tabela chamada "acidente\_snv", que contém duas colunas: "codigo" e "mes\_ano". A coluna "codigo" é preenchida com os códigos únicos da coluna "código" da tabela "snv", enquanto a coluna "mes\_ano" é preenchida com todas as datas mensais entre janeiro de 2007 e dezembro de 2022.

Para preencher as colunas, utilizamos a função "np.repeat" para repetir os códigos únicos para cada mês/ano na lista de datas e a função "np.tile" para repetir a lista de datas para cada código único. Por fim, utilizamos o método "reset\_index" para resetar o índice da tabela resultante.

**Figura 32 - Criação do dataframe**

```
# Lista de códigos únicos da coluna "código" da tabela snv
codigos_unicos = snv['código'].unique()

# Lista de todas as datas no intervalo de janeiro de 2007 até dezembro de 2022
datas = pd.date_range(start='01/01/2007', end='12/31/2022', freq='M').strftime('%m/%Y').tolist()

# Cria uma nova tabela com as colunas "Código" e "mes_ano"
acidente_snv = pd.DataFrame({'código': np.repeat(codigos_unicos, len(datas)),
                             'mes_ano': np.tile(datas, len(codigos_unicos))})

# Reset o índice da tabela resultante
acidente_snv = acidente_snv.reset_index(drop=True)
```

A seguir, enriquecemos a tabela com os dados do SNV por meio de um merge entre as tabelas "acidente\_snv" e "snv", baseado na coluna "codigo". Utilizamos a opção "how='left'" para manter todas as linhas da tabela "acidente\_snv" e apenas as linhas da tabela "snv" que têm valores correspondentes na coluna "codigo".

**Figura 33 - Merge dos dataframes**

```
# Junção entre acidente_snv e snv
acidente_snv = pd.merge(acidente_snv, snv, on='codigo', how='left')
```

Padronizamos as variáveis em ambos os datasets ao remover as variáveis não presentes no DataFrame obra.

**Figura 34 - Removendo Colunas**

```
: # Removendo colunas
acidente_snv = acidente_snv.drop(["Tipo de trecho", "Desc Coinc", "Local de Início", "Local de Fim", "Superfície Federal", "Obras"])
```

Adicionamos a coluna de número de acidentes, realizando um agrupamento por código e mês/ano na tabela datatran\_snv\_obra. O resultado é uma tabela com as colunas "codigo", "mes\_ano" e "num\_acidente", que indica o número de acidentes em cada código e mês/ano. Na segunda etapa, realizamos um merge entre a tabela resultante da primeira etapa e a tabela "acidente\_snv" pela combinação das colunas "codigo" e "mes\_ano". O

resultado é uma nova tabela "acidente\_snv" que contém o número de acidentes para cada código e mês/ano.

**Figura 35 - Incluindo coluna de número de acidentes**

```
# Etapa 1 - Groupby na tabela datatran_snv_obra pelas colunas codigo e mes_ano
datatran_grouped = datatran_snv_obra.groupby(['codigo', 'mes_ano']).count().reset_index()[['codigo', 'mes_ano', 'index']]
datatran_grouped.rename(columns={'index': 'num_acidente'}, inplace=True)

# Etapa 2 - Merge entre datatran_grouped e acidente_snv pelas colunas codigo e mes_ano
acidente_snv = pd.merge(acidente_snv, datatran_grouped, on=['codigo', 'mes_ano'], how='left')

# Exibindo informações
print(acidente_snv.head())
print("\n")
print(acidente_snv.info())

# Calcular a proporção de valores nulos na coluna "num_acidente"
print("\nProporção de valores nulos na coluna 'num_acidente':")
print(acidente_snv['num_acidente'].isnull().mean())
```

	codigo	mes_ano	br	uf	km_inicial	km_final	extensao	num_acidente
0	040BMG0090	01/2007	40	MG	0.0	44.1	44.1	NaN
1	040BMG0090	02/2007	40	MG	0.0	44.1	44.1	1.0
2	040BMG0090	03/2007	40	MG	0.0	44.1	44.1	NaN
3	040BMG0090	04/2007	40	MG	0.0	44.1	44.1	NaN
4	040BMG0090	05/2007	40	MG	0.0	44.1	44.1	1.0

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 168576 entries, 0 to 168575
Data columns (total 8 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   codigo      168576 non-null object
1   mes_ano     168576 non-null object
2   br          168576 non-null int64
3   uf          168576 non-null object
4   km_inicial  168576 non-null float64
5   km_final    168576 non-null float64
6   extensao    168576 non-null float64
7   num_acidente 8130 non-null  float64
dtypes: float64(4), int64(1), object(3)
memory usage: 11.6+ MB
None

Proporção de valores nulos na coluna 'num_acidente':
0.9517724943052391
```

Identificamos os trechos e períodos sem acidentes, substituindo os valores nulos por zeros.

**Figura 36 - Substituindo os valores nulos por zero**

```
# Substituindo os valores nulos por zero
acidente_snv['num_acidente'].fillna(0, inplace=True)
```

Convertemos as colunas 'mes\_ano' para um objeto datetime usando a função `pd.to_datetime()`, especificando o formato de entrada como '%m/%Y', que indica o mês e o ano em que ocorreram os acidentes.

Em seguida, adicionamos duas novas colunas ao DataFrame "acidente\_snv": 'mes', que contém o número do mês correspondente ao acidente, e 'ano', que contém o ano

correspondente ao acidente. Essas novas colunas são obtidas através da extração dos valores do objeto datetime da coluna 'mes\_ano'.

**Figura 37 - Criando as colunas Mês e Ano**

```
# Criando as colunas Mês e Ano
acidente_snv['mes_ano'] = pd.to_datetime(acidente_snv['mes_ano'], format='%m/%Y')
acidente_snv['mes'] = acidente_snv['mes_ano'].dt.month
acidente_snv['ano'] = acidente_snv['mes_ano'].dt.year
```

Por fim, salvamos o novo DataFrame em um arquivo CSV.

**Figura 38 - Salvamento do arquivo**

```
# Salvar o resultado em um novo arquivo CSV
acidente_snv.to_csv('acidente_snv.csv', index=False, sep=';', decimal=',', encoding = 'cp1252')
```

### 3.3.2. Dataframe de acidentes por Obra

Para criar o dataframe de acidentes por obra, foi disponibilizado um arquivo CSV contendo todos os meses de execução de obra. A partir desse ponto, os códigos são os mesmos utilizados na criação do dataframe de acidentes por SNV e incluem as seguintes etapas:

- Junção dos dataframes acidente\_obra e obra: essa etapa permite combinar as informações de acidentes com as informações de obra, de modo a incluir o código único de cada obra na tabela de acidentes.
- Remoção das colunas não presentes no dataframe SNV: essa etapa é realizada para padronizar as variáveis em ambos os datasets, garantindo que apenas as informações relevantes sejam mantidas na tabela.
- Inclusão da coluna número de acidentes: para isso, é realizado um agrupamento por código e mês/ano na tabela datatran\_snv\_obra. O resultado é uma tabela com as colunas código, mês/ano e num\_acidente, que indica o número de acidentes em cada código e mês/ano. Na segunda etapa, é realizado um merge entre a tabela resultante da primeira etapa e a tabela

acidente\_snv pela combinação das colunas código e mês/ano. O resultado é uma nova tabela acidente\_snv que contém o número de acidentes para cada código e mês/ano.

- Substituição dos valores nulos por zero na coluna de número de acidentes: como a ausência de valores nessa coluna indica que não houve acidentes naquele código e mês/ano, é importante substituir os valores nulos por zero para que os dados fiquem mais claros e legíveis.
- Salvamento do dataframe em um arquivo CSV: por fim, o novo dataframe é salvo em um arquivo CSV para uso futuro.

**Figura 39 - Códigos para criação do dataframe de acidentes por obra**

```
# Baixando e extraindo
file_path = r'C:\Users\bcz87\obra_mes_ano.csv'
acidente_obra = pd.read_csv(file_path, sep=';', decimal=',')
```

```
# Junção das tabelas acidente_obra e obra
acidente_obra = pd.merge(acidente_obra, obra, on='obra', how='left')
```

```
# Removendo colunas
acidente_obra = acidente_obra.drop(["data_inicio", "data_fim"], axis=1)
```

```
# Etapa 1 - Groupby na tabela datatran_snv_obra pelas colunas código e mes_ano
datatran_grouped = datatran_snv_obra.groupby(['obra', 'mes_ano']).count().reset_index()[['obra', 'mes_ano', 'index']]
datatran_grouped.rename(columns={'index': 'num_acidente'}, inplace=True)

# Etapa 2 - Merge entre datatran_grouped e acidente_snv pelas colunas código e mes_ano
acidente_obra = pd.merge(acidente_obra, datatran_grouped, on=['obra', 'mes_ano'], how='left')
```

```
# Substituindo valores nulos por zero
acidente_obra['num_acidente'].fillna(0, inplace=True)
```

```
# Salvar o resultado em um novo arquivo CSV
acidente_obra.to_csv('acidente_obra.csv', index=False, sep=';', decimal=',', encoding = 'cp1252')
```

#### 4. Análise e Exploração dos Dados

A análise exploratória de dados é uma etapa fundamental em qualquer projeto de análise de dados, incluindo este trabalho de conclusão de curso. Essa etapa consiste em investigar e compreender os dados disponíveis, a fim de identificar padrões, tendências, anomalias e outras informações relevantes que possam fornecer insights sobre o problema em questão. A análise exploratória de dados geralmente envolve a aplicação de técnicas estatísticas e de visualização de dados para extrair informações úteis e relevantes a partir dos dados brutos.

Foi exibido estatísticas sobre o DataFrame `acidente_snv`. A função `describe()` calcula a contagem, a média, o desvio padrão, o valor mínimo, os quartis e o valor máximo de cada coluna numérica do DataFrame. Isso ajuda a ter uma noção geral dos valores presentes no DataFrame e identificar possíveis problemas ou padrões nos dados.

Com base nos dados apresentados, podemos observar que a extensão média dos trechos avaliados é de 21,01 km, com uma variação de 0,1 km para o menor trecho até 207,3 km para o maior trecho. A quantidade de acidentes por trecho varia de 0 a 13, com média de 0,062 acidentes por mês.

**Figura 40 - Estatísticas sobre o DataFrame**

```
# Exibindo informações
acidente_snv.describe()
```

	br	km_inicial	km_final	extensao	num_acidente	mes	ano
count	168576.000000	168576.000000	168576.000000	168576.000000	168576.000000	168576.000000	168576.000000
mean	293.462415	308.656891	329.669499	21.012608	0.061954	6.500000	2014.500000
std	132.749100	239.058098	237.831608	18.846457	0.316766	3.452063	4.609786
min	40.000000	0.000000	0.400000	0.100000	0.000000	1.000000	2007.000000
25%	153.000000	105.500000	127.500000	6.500000	0.000000	3.750000	2010.750000
50%	342.000000	265.400000	284.400000	16.550000	0.000000	6.500000	2014.500000
75%	381.000000	477.900000	499.600000	29.900000	0.000000	9.250000	2018.250000
max	499.000000	978.210000	1014.810000	207.300000	13.000000	12.000000	2022.000000

Foi criado um gráfico de matriz usando o Seaborn's `pairplot`, que exibe histogramas para cada variável em relação à variável de número de acidentes. Ele começa ajustando a distância entre as subparcelas usando a função `subplots_adjust`. Em seguida, ele cria uma figura com nove subplots usando a função `subplots` do Matplotlib e especifica o número de linhas, colunas e o tamanho da figura. Em seguida, ele define uma lista de colunas a serem



plotadas e faz um loop pelos subplots criados. Para cada subplot, é criado um histograma para a variável correspondente e o título do subplot é definido como o nome da variável.

Com base nos histogramas é possível verificar que a maioria dos trechos apresenta zero acidentes, o que já era esperado. Além disso, há uma concentração de acidentes nos quilômetros iniciais e finais dos trechos, possivelmente devido à variação na extensão dos trechos, com alguns tendo poucos quilômetros.

Também é possível observar uma concentração de trechos com alto número de acidentes nos primeiros e últimos meses do ano, o que pode ser devido a condições climáticas ou períodos de férias e feriados. Em relação a 2017, foi identificado um aumento no número de trechos com muitos acidentes por mês.

**Figura 41 - Histogramas para cada variável em relação à variável de número de acidentes**

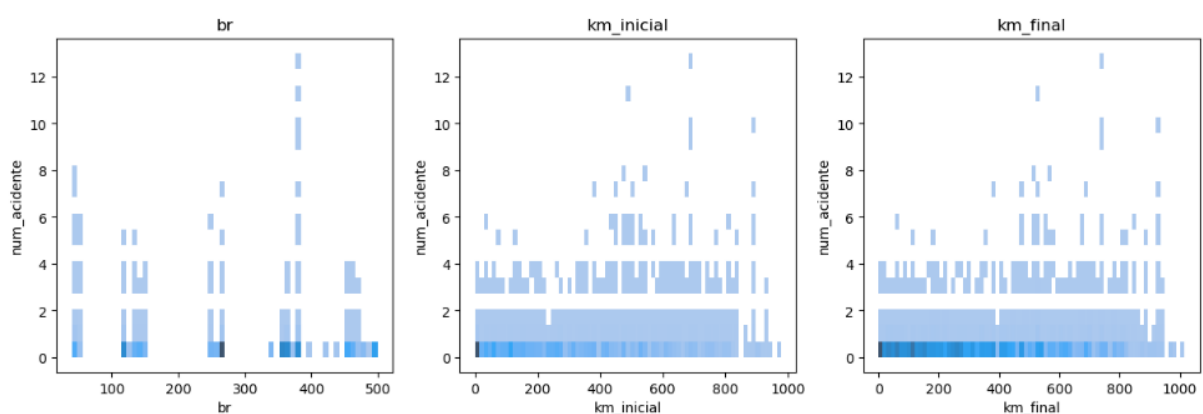
```
# Ajusta a distância entre as subplots
plt.subplots_adjust(wspace=0.5, hspace=0.5)

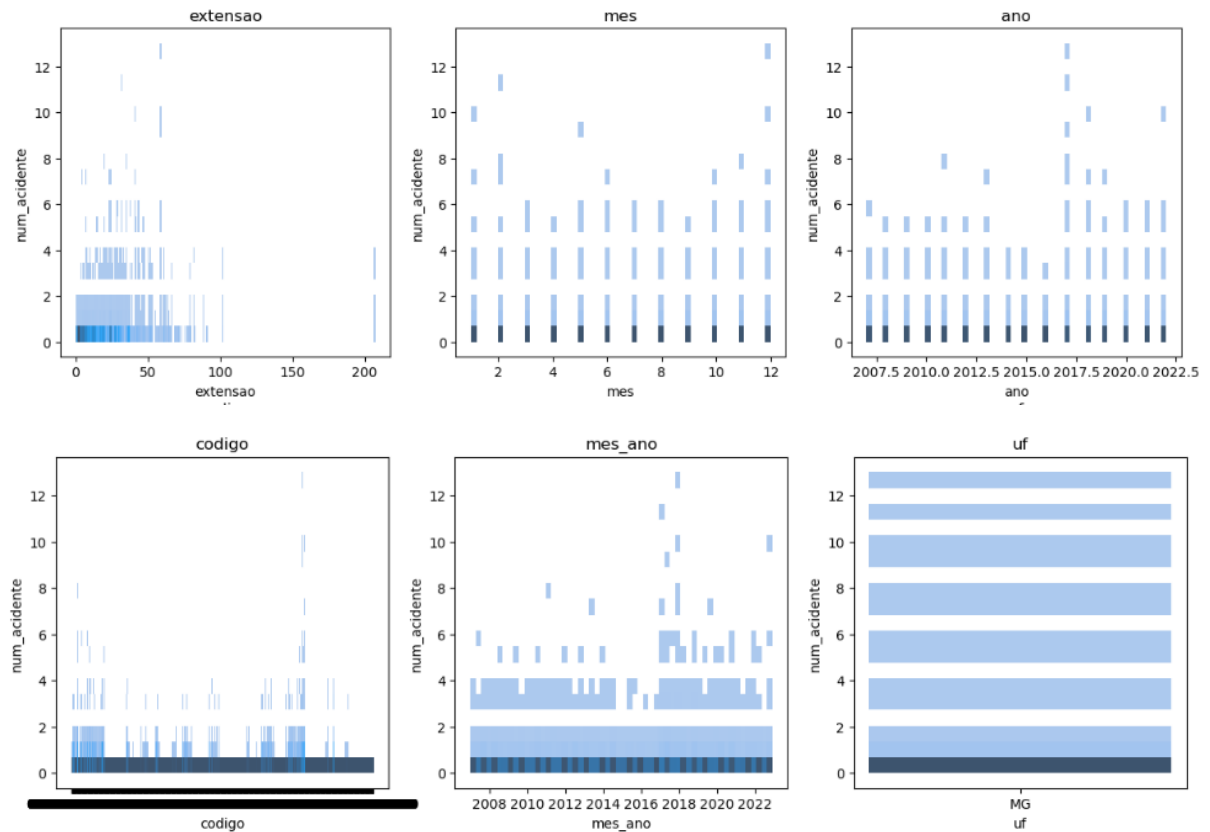
# Cria uma figura com os subplots
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))

# Define as colunas para o pairplot
cols = ['br', 'km_inicial', 'km_final', 'extensao', 'mes', 'ano', 'codigo', 'mes_ano', 'uf']

# Loop pelos subplots e cria o pairplot para cada coluna
for i, ax in enumerate(axs.flatten()):
    if i < len(cols):
        sns.histplot(data=acidente_snv, x=cols[i], y='num_acidente', ax=ax)
        ax.set_title(cols[i])
```

<Figure size 640x480 with 0 Axes>





Utilizei boxplots para visualizar a distribuição e identificar possíveis outliers em cada variável do conjunto de dados. Para cada boxplot, examinei a posição da mediana, a amplitude interquartil (IQR), que é a diferença entre o terceiro quartil (Q3) e o primeiro quartil (Q1), e os valores mínimo e máximo. Verifiquei que a variável extensão apresentava um grande número de outliers, sugerindo uma grande variabilidade nos valores da amostra. Já a variável número de acidentes apresentava uma alta frequência de valores iguais a zero.

**Figura 42 - Boxplots das variáveis do dataframe**

```
# Cria uma figura com 10 subplots, uma para cada variável
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(15, 6))

# Cria um boxplot para cada variável
axs[0, 0].boxplot(acidente_snv['br'])
axs[0, 0].set_title('BR')

axs[0, 1].boxplot(acidente_snv['km_inicial'])
axs[0, 1].set_title('Km Inicial')

axs[0, 2].boxplot(acidente_snv['km_final'])
axs[0, 2].set_title('Km Final')

axs[0, 3].boxplot(acidente_snv['extensao'])
axs[0, 3].set_title('Extensão')

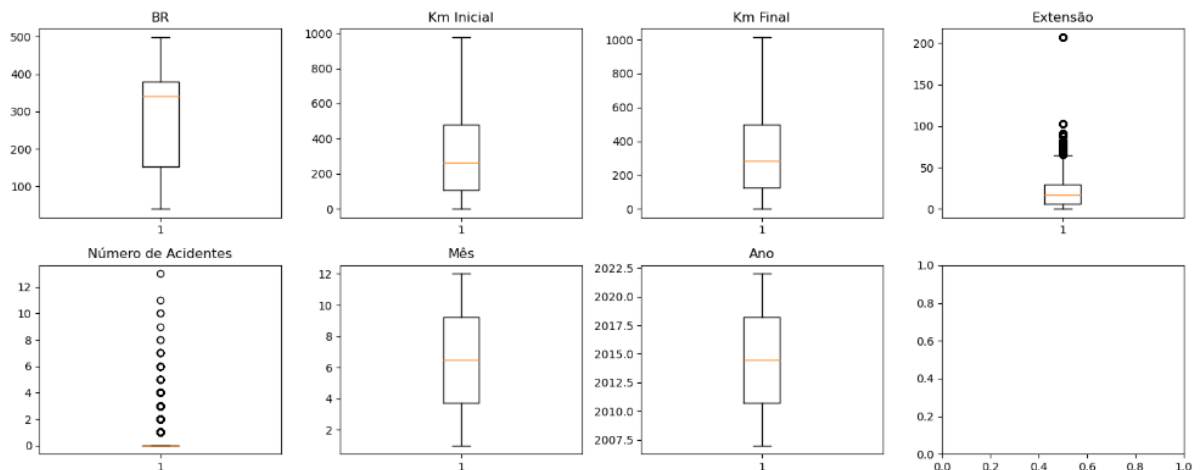
axs[1, 0].boxplot(acidente_snv['num_acidente'])
axs[1, 0].set_title('Número de Acidentes')

axs[1, 1].boxplot(acidente_snv['mes'])
axs[1, 1].set_title('Mês')

axs[1, 2].boxplot(acidente_snv['ano'])
axs[1, 2].set_title('Ano')

# Ajusta o espaçamento entre os subplots
plt.tight_layout()

# Exibe o gráfico
plt.show()
```



Uma matriz de correlação é uma ferramenta analítica que permite visualizar as relações entre pares de variáveis em um conjunto de dados. Essas relações são medidas por meio do coeficiente de correlação, que varia de -1 a 1. Valores próximos a 1 indicam uma correlação positiva forte, enquanto valores próximos a -1 indicam uma correlação negativa forte. É possível usar a matriz de correlação para identificar padrões e relacionamentos nos dados, bem como selecionar variáveis para análises posteriores ou modelagem.

Ao analisar a matriz de correlação no contexto do meu TCC, observei uma correlação altíssima entre as variáveis `km_final` e `km_inicial`, o que era esperado, já que essas variáveis estão relacionadas à distância percorrida em um trecho de rodovia. Porém, em relação à

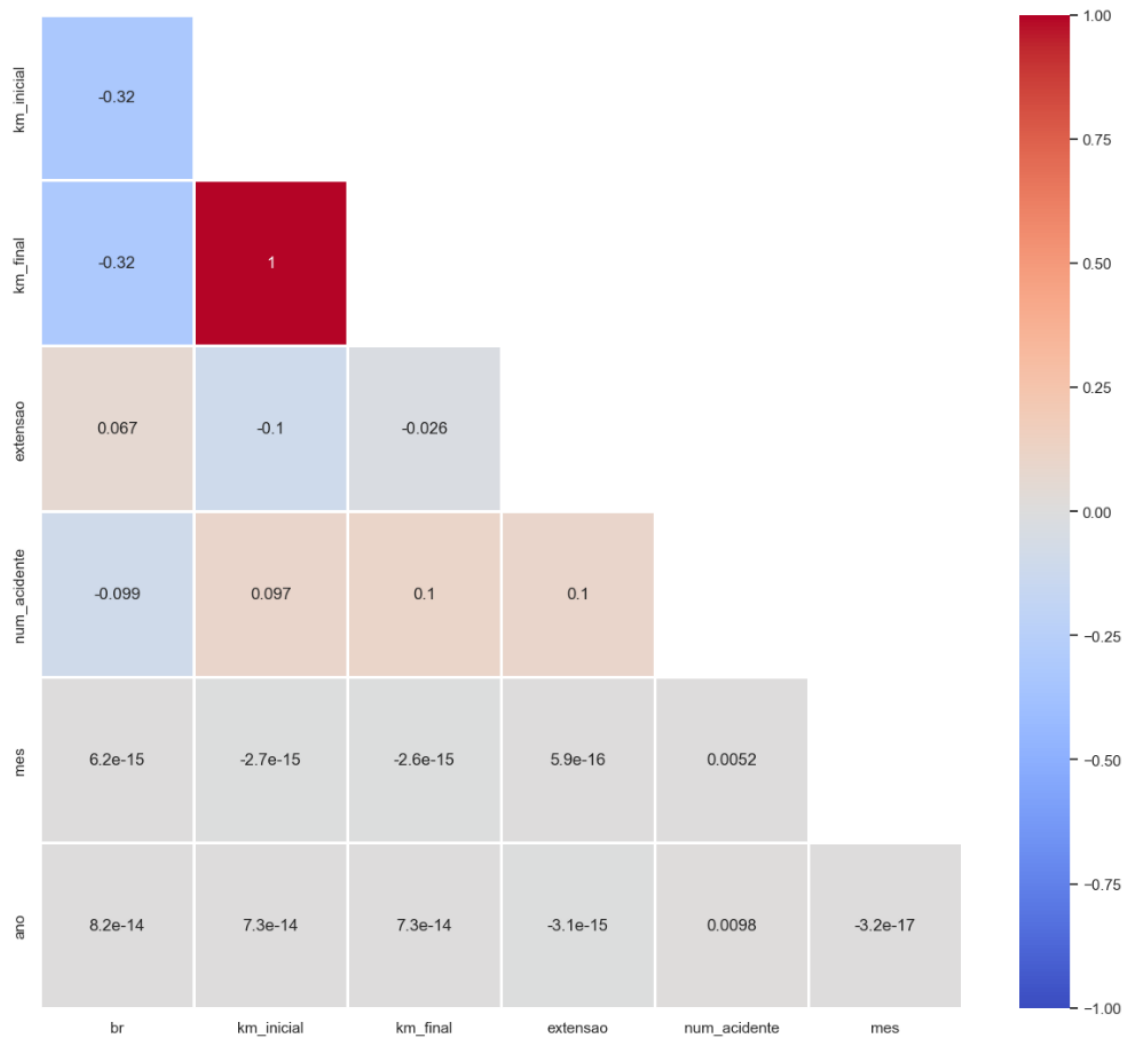
variável acidentes, suas correlações mais fortes foram com a extensão e km\_final, embora a correlação tenha sido baixa, de apenas 0,1.

As demais correlações foram ainda menores, o que indica que pode haver dificuldades para o aprendizado de máquina com base nesses dados.

**Figura 43 - Matriz de correlação**

```
# Função de correlação.
def correlacoes(acidente_snv):
    corr = acidente_snv.corr().drop(acidente_snv.corr().index[0], axis='index').drop(acidente_snv.corr().index[-1], axis='columns')
    mascara = np.triu(np.ones(corr.shape)).astype(bool)
    for i in range(len(mascara)):
        for j in range(mascara.shape[1]):
            if i == j:
                mascara[i, j] = False
    sns.set(font_scale=1.0)
    sns.set_style('whitegrid')
    plt.figure(figsize=(15, 13))
    mapa_calor = sns.heatmap(corr, mask=mascara, annot=True, vmin=-1, vmax=1, cmap='coolwarm', linewidths=1)
    plt.show()

# Impressão das correlações
correlacoes(acidente_snv)
```



Ao analisar o conjunto de dados de acidentes em rodovias, foi possível identificar as rodovias que apresentam os maiores números de acidentes. Os resultados indicam que as rodovias BR-381, BR-040 e BR-116 apresentam os maiores números de acidentes, nesta ordem.

**Figura 44 - Quantidade de acidente por BR**

```
# Quantidade de acidentes
acidente_snv.groupby('br')['num_acidente'].sum().sort_values(ascending=False).head(10)
```

```
br
381    2561.0
40     2100.0
116    1404.0
262    1152.0
365     957.0
50      525.0
267     365.0
251     303.0
153     233.0
135     217.0
Name: num_acidente, dtype: float64
```

Quanto aos trechos mais perigosos de cada uma dessas rodovias, temos:

- BR-381 de ENTR BR-262/381 (FIM CONTORNO BETIM) até ENTR MG-155 com 209 acidentes;
- BR-040 de ENTR MG-432 (P/ESMERALDAS) até ENTR BR-135(B)/262(A)/381(A) (ANEL RODOVIÁRIO DE BELO HORIZONTE) com 232 acidentes;
- BR-116 de ENTR BR-482 (FERVEDOURO) até ENTR BR-265(A)/356 (MURIAÉ) com 114 acidentes;

Dois dos trechos são na região metropolitana de BH.

**Figura 45 - Trechos com maior número de acidentes nas BRs 381, 040 e 116**

```
# Seleciona os acidentes na BR 381 com o código desejado
acidente_381 = acidente_snv.loc[(acidente_snv['br'] == 381) & (acidente_snv['codigo'])]

# Realiza o merge com a tabela 'snv'
acidente_381 = pd.merge(acidente_381, snv[['codigo', 'Local de Início', 'Local de Fim']], on='codigo')

# Agrupa por 'br' e 'codigo', soma o número de acidentes e ordena de forma decrescente
acidente_381 = acidente_381.groupby(['br', 'codigo', 'Local de Início', 'Local de Fim'])['num_acidente'].sum().sort_values(ascending=False)

# Imprime o resultado
print(acidente_381)
```

br	codigo	Local de Início	Local de Fim	num_acidente
381	381BMG0490	ENTR BR-262/381 (FIM CONTORNO BETIM)	ENTR MG-155	209.0
	381BMG0670	ENTR BR-265(B) (P/NEPOMUCENO)	ENTR MG-167(A) (P/TRÊS CORAÇÕES)	197.0
	381BMG0790	ENTR MG-295 (CAMBUÍ)	ENTR MG-460 (P/TOLEDO)	174.0
	381BMG0770	ENTR BR-459 (P/POUSO ALEGRE)	ENTR MG-295 (CAMBUÍ)	141.0
	381BMG0630	ENTR MG-332 (SANTO ANTÔNIO DO AMPARO)	ENTR BR-354 (PERDÕES)	100.0
	381BMG0530	ENTR MG-431 (P/ITAITIAIUÇU)	ENTR MG-040 (ITAGUARA)	100.0
	381BMG0170	ACESSO À GOV. VALADARES	ENTR R SÃO LUIZ (PERIQUITO)	92.0
	381BMG0480	ENTR AV CAMPOS OURIQUE (INÍCIO CONTORNO BETIM)	ENTR BR-262/381 (FIM CONTORNO BETIM)	87.0
	381BMG0730	ENTR BR-267 (P/PALMELA)	ENTR MG-458 (CAREACU)	86.0
	381BMG0250	ENTR MG-320 (P/JAGUARAÇU)	ENTR BR-120(A) (DESEMBARGADOR DRUMOND) (P/ ITABIRA)	85.0

Name: num\_acidente, dtype: float64

```
# Seleciona os acidentes na BR 040 com o código desejado
acidente_040 = acidente_snv.loc[(acidente_snv['br'] == 40) & (acidente_snv['codigo'])]

# Realiza o merge com a tabela 'snv'
acidente_040 = pd.merge(acidente_040, snv[['codigo', 'Local de Início', 'Local de Fim']], on='codigo')

# Agrupa por 'br' e 'codigo', soma o número de acidentes e ordena de forma decrescente
acidente_040 = acidente_040.groupby(['br', 'codigo', 'Local de Início', 'Local de Fim'])['num_acidente'].sum().sort_values(ascending=False)

# Imprime o resultado
print(acidente_040)
```

br	codigo	Local de Início	Local de Fim	num_acidente
40	040BMG0360	ENTR MG-432 (P/ESMERALDAS)	ENTR BR-135(B)/262(A)/381(A) (ANEL RODOVIÁRIO DE BELO HORIZONTE)	232.0
	040BMG0330	ENTR MG-238 (P/SETE LAGOAS)	ENTR MG-432 (P/ESMERALDAS)	204.0
	040BMG0270	ENTR MG-231	ENTR MG-424 (P/SETE LAGOAS)	165.0
	040BMG0490	ENTR BR-383(B)/482 (CONSELHEIRO LAFIETE)	ENTR MG-275 (P/CARANDAÍ)	112.0
	040BMG0510	ENTR MG-275 (P/CARANDAÍ)	ACESSO ALTO DOCE (INÍCIO PISTA DUPLA)	102.0
	040BMG0570	ENTR BR-499 (SANTOS DUMONT)	ENTR ANT UNIÃO E INDÚSTRIA (B. TRIUNFO)	97.0
	040BMG0170	ENTR BR-365	ENTR MG-220 (TRÊS MARIAS)	94.0
	040BMG0400	ENTR BR-356(A) (P/BELO HORIZONTE)	ENTR BR-356(B)	93.0
	040BMG0090	DIV GO/MG	ENTR MG-188(B) (P/SÃO SEBASTIÃO)	72.0
	040BMG0130	ENTR MG-410 (P/PORTO DIAMANTE)	ENTR MG-181 (JOÃO PINHEIRO)	65.0

Name: num\_acidente, dtype: float64

```
# Seleciona os acidentes na BR 116 com o código desejado
acidente_116 = acidente_snv.loc[(acidente_snv['br'] == 116) & (acidente_snv['codigo'])]

# Realiza o merge com a tabela 'snv'
acidente_116 = pd.merge(acidente_116, snv[['codigo', 'Local de Início', 'Local de Fim']], on='codigo')

# Agrupa por 'br' e 'codigo', soma o número de acidentes e ordena de forma decrescente
acidente_116 = acidente_116.groupby(['br', 'codigo', 'Local de Início', 'Local de Fim'])['num_acidente'].sum().sort_values(ascending=False)

# Imprime o resultado
print(acidente_116)
```

br	codigo	Local de Início	Local de Fim	num_acidente
116	116BMG1350	ENTR BR-482 (FERVEDOURO)	ENTR BR-265(A)/356 (MURIAÉ)	114.0
	116BMG1450	ENTR BR-267(B) (P/TEBAS)	ENTR BR-393(A)	82.0
	116BMG1195	ACESSO ITANHOMI	ACESSO P/FERNANDES TOURINHO E SOBRÁLIA	82.0
	116BMG1030	MEDINA (ACESSO SUL)	ENTR BR-367 (P/ ITAQBIM)	67.0
	116BMG1410	ENTR MG-454 (P/RECREIO)	ENTR BR-120/267(A) (LEOPOLDINA)	66.0
	116BMG1230	ENTR BR-458(B) (P/IAPÚ)	ENTR MG-425 (P/ENTRE FOLHAS)	63.0
	116BMG1130	ENTR BR-342(B)/418/MG-217 (TEÓFILO OTONI)	ACESSO ITAMBACURI	62.0
	116BMG1050	ENTR BR-367 (P/ ITAQBIM)	PADRE PARAÍSO (ACESSO SUL)	59.0
	116BMG1370	ENTR BR-265(B)	ENTR MG-285 (LARANJAL)	58.0
	116BMG1150	ACESSO ITAMBACURI	ENTR MG-311 (P/PESCADOR)	57.0

Name: num\_acidente, dtype: float64

Com base na análise dos dados, observou-se que os meses mais perigosos em termos de número de acidentes são os três primeiros (janeiro, fevereiro e março) e os três últimos (outubro, novembro e dezembro). Isso pode estar relacionado a vários fatores, como o aumento do tráfego nas estradas durante os feriados de fim de ano e férias de verão, bem como as condições climáticas adversas que ocorrem com maior frequência nesses meses.

**Figura 46 - Quantidade de acidente por mês**

```
# Quantidade de acidentes por mês
acidente_snv.groupby('mes')['num_acidente'].sum().sort_values(ascending=False).head(10)
```

mes	num_acidente
10	1006.0
2	947.0
11	944.0
12	930.0
3	870.0
1	854.0
7	837.0
8	835.0
5	816.0
6	814.0

Name: num\_acidente, dtype: float64

Com base na análise do número de acidentes por ano, pode-se verificar que o ano de 2017 apresentou o maior número de ocorrências, totalizando 1414 acidentes. Na sequência, temos os anos de 2018, 2020 e 2019, com 921, 799 e 770 acidentes, respectivamente. É interessante destacar que esses são praticamente os últimos anos levantados, indicando uma tendência de aumento no número de acidentes ao longo do tempo.

**Figura 47 - Quantidade de acidente por ano**

```
# Quantidade de acidentes por ano
acidente_snv.groupby('ano')['num_acidente'].sum().sort_values(ascending=False).head(10)
```

ano	num_acidente
2017	1414.0
2018	921.0
2020	799.0
2019	770.0
2007	715.0
2008	696.0
2012	642.0
2011	623.0
2010	599.0
2013	586.0

Name: num\_acidente, dtype: float64

Quanto aos números de acidentes por mês e ano, é possível observar que os meses mais perigosos são os três primeiros e os três últimos dos anos de 2017 e 2018, com destaque para o ano de 2017.

**Figura 48 - Quantidade de acidente por mês e ano**

```
# Quantidade de acidentes por mês/ano
acidente_snv.groupby('mes_ano')['num_acidente'].sum().sort_values(ascending=False).head(10)
```

mes_ano	num_acidente
2017-02-01	155.0
2017-11-01	148.0
2017-12-01	146.0
2017-03-01	133.0
2018-01-01	129.0
2018-02-01	129.0
2017-04-01	124.0
2017-05-01	120.0
2017-10-01	112.0
2017-06-01	110.0

Name: num\_acidente, dtype: float64



## 5. Criação de Modelos de Machine Learning

### 5.1. Etapas Iniciais

Na etapa de preparação dos dados para a modelagem de Machine Learning, realizamos a seleção das variáveis que seriam utilizadas como entrada no modelo. Para isso, selecionamos as variáveis referentes ao mês, ano, número da rodovia (BR), unidade federativa (UF), quilômetro inicial e final do trecho e extensão da rodovia. Além disso, selecionamos a variável alvo, que é o número de acidentes em cada trecho de rodovia. Essa etapa é crucial para a criação de um modelo que seja capaz de identificar padrões e fazer previsões precisas, garantindo assim que as decisões tomadas a partir dos resultados obtidos sejam confiáveis e eficientes.

**Figura 49 - Seleção das variáveis**

```
# Selecionando y e X
y = acidente_snv['num_acidente']
X = acidente_snv[['mes', 'ano', 'br', 'uf', 'km_inicial', 'km_final', 'extensao']]
```

Nesta etapa do trabalho, foi realizada a codificação das colunas categóricas do conjunto de dados utilizando a técnica de OneHotEncoder. Inicialmente, foi selecionada a coluna "uf" para ser codificada, que representa a unidade federativa onde ocorreram os acidentes. Em seguida, foi criado um objeto OneHotEncoder para realizar a codificação, com os parâmetros "sparse=False" e "handle\_unknown='ignore'", para que não ocorram erros caso haja valores desconhecidos.

Após a codificação, foram obtidas as categorias codificadas para a coluna "uf", que foram utilizadas para criar um novo conjunto de dados com as colunas codificadas. Foram criados dataframes para as colunas codificadas, com o nome de cada coluna seguindo o padrão "uf\_categoria". Em seguida, as colunas codificadas foram combinadas com as outras variáveis independentes, utilizando a função "concat" do pandas. Essa etapa é fundamental para garantir que as colunas categóricas sejam representadas de forma numérica, permitindo a utilização dos dados em algoritmos de aprendizado de máquina.

**Figura 50 - Codificação de colunas**

```
# OneHotEncoder
# Codificar as colunas "UF" usando OneHotEncoder
enc = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_enc = enc.fit_transform(X[['uf']])

# Obter as categorias codificadas
uf_categories = enc.categories_[0]

# Criar dataframes para as colunas codificadas
uf_encoded = pd.DataFrame(X_enc[:, :len(uf_categories)], index=X.index, columns=[f'uf_{category}' for category in uf_categories])

# Combinar as colunas codificadas com as outras variáveis independentes
X = pd.concat([X.drop(columns=['uf']), uf_encoded], axis=1)
```

O código a seguir é responsável por realizar o processo de normalização dos dados das variáveis independentes. A normalização é uma técnica de pré-processamento comum em machine learning que tem como objetivo garantir que todas as variáveis possuam a mesma escala e, dessa forma, nenhuma variável possua maior peso ou importância que as outras na modelagem do algoritmo de machine learning. Para isso, o código utiliza a classe `StandardScaler`, presente na biblioteca `scikit-learn`, que realiza a normalização dos dados pela subtração da média e divisão pelo desvio padrão. Ao final da normalização, o novo conjunto de dados normalizados é atribuído à variável `X`, que contém as variáveis independentes utilizadas no modelo de machine learning

**Figura 51 - Normalização dos dados**

```
# StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Foi feita a divisão dos dados em conjuntos de treino e teste. Essa etapa é fundamental para avaliar a performance do modelo de forma mais realista e evitar problemas como `overfitting`. A função `train_test_split()` da biblioteca `Scikit-Learn` é utilizada para dividir o dataset em dois conjuntos, um para treino e outro para teste. É possível definir o tamanho de cada conjunto através dos parâmetros `test_size` e `train_size`. Nesse caso, foi definido que 70% dos dados serão utilizados para treino e 30% para teste. Além disso, é possível definir uma semente (`random_state`) para garantir que a divisão seja feita sempre da mesma forma, o que é importante para a reprodutibilidade dos resultados.

**Figura 52 - Divisão do dataset em treino e teste**

```
# Model Selection - Splitter Functions
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, train_size=0.7, random_state=42)
```

## 5.2. Algoritmos

Ao escolher algoritmos para classificação em Machine Learning, é importante considerar diversos fatores, como a natureza dos dados, a quantidade de dados disponíveis, a simplicidade do modelo e a precisão desejada. Além disso, é importante considerar as limitações de cada algoritmo e como elas podem afetar a escolha do melhor modelo para um determinado problema. A escolha dos algoritmos para classificação deve ser baseada em uma análise cuidadosa desses fatores e em experimentos empíricos que permitam avaliar o desempenho de diferentes modelos em relação aos objetivos do projeto. Neste trabalho, foram escolhidos os algoritmos KNN, SGD Classifier, Random Forest Classifier, Decision Tree Regressor e Regressão Logística para realizar a classificação dos dados de acidentes de trânsito, levando em consideração suas características e vantagens em relação aos objetivos do projeto.

### 5.2.1. KNN

Foi realizado uma busca por meio do Grid Search, com o objetivo de encontrar os melhores hiperparâmetros para o algoritmo KNN (K-Nearest Neighbors). Para isso, foram definidos valores diferentes para cada hiperparâmetro do algoritmo, como o número de vizinhos mais próximos (`n_neighbors`), o peso de cada vizinho (`weights`), o algoritmo a ser utilizado (`algorithm`), a métrica de distância a ser usada (`metric`) e o valor de `p` utilizado na métrica de Minkowski (`p`). A função `GridSearchCV` do `scikit-learn` foi utilizada para criar um objeto que realiza a busca exaustiva dos melhores hiperparâmetros. O modelo foi treinado com o conjunto de treinamento e, após a execução do Grid Search, foi possível imprimir os melhores hiperparâmetros encontrados para o modelo. Essa é uma técnica importante para otimizar o desempenho do algoritmo e escolher a combinação ideal de hiperparâmetros para cada conjunto de dados.

**Figura 53 - Busca de hiperparâmetros pelo Grid Search**

```
# Grid Search
# Definir os valores dos hiperparâmetros a serem testados
param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'p': [1, 2],
    'metric': ['cityblock', 'minkowski', 'euclidean']
}

# Criar um objeto GridSearchCV com os valores dos hiperparâmetros a serem testados
grid = GridSearchCV(KNeighborsClassifier(), param_grid)

# Treinar o modelo com o conjunto de treinamento
grid.fit(X_train, y_train)

# Imprimir os melhores hiperparâmetros encontrados
print("Melhores hiperparâmetros:", grid.best_params_)

Melhores hiperparâmetros: {'algorithm': 'ball_tree', 'metric': 'minkowski', 'n_neighbors': 9, 'p': 2, 'weights': 'uniform'}
```

Em seguida treinamos um modelo de classificação KNN com os hiperparâmetros otimizados pelo Grid Search e avaliamos seu desempenho com as métricas de acurácia, precisão, recall e F1 Score. Primeiramente, o modelo é treinado com o conjunto de treinamento usando os hiperparâmetros definidos na etapa de otimização do Grid Search. Em seguida, o modelo é utilizado para fazer previsões com o conjunto de teste. Por fim, as métricas são calculadas comparando as previsões feitas pelo modelo com as classes reais do conjunto de teste. A acurácia representa a proporção de previsões corretas em relação ao total de previsões, enquanto a precisão mede a proporção de verdadeiros positivos em relação ao total de predições positivas. O recall mede a proporção de verdadeiros positivos em relação ao total de instâncias positivas, e o F1 Score é uma média harmônica entre a precisão e o recall. As métricas de avaliação são importantes para comparar e escolher entre modelos diferentes e para avaliar a qualidade geral do modelo em relação aos objetivos do projeto.

**Figura 54 - Avaliação do desempenho do algoritmo**

```
# Treinar o modelo
clf = KNeighborsClassifier(algorithm='ball_tree', metric='minkowski', n_neighbors=9, p=2, weights='uniform')
clf.fit(X_train, y_train)

# Fazer previsões com o conjunto de teste
y_pred = clf.predict(X_test)

# Avaliar o desempenho do modelo
print("Acurácia: {:.2f}%".format(accuracy_score(y_test, y_pred)*100))
print("Precisão: {:.2f}%".format(precision_score(y_test, y_pred, average='weighted')*100))
print("Recall: {:.2f}%".format(recall_score(y_test, y_pred, average='weighted')*100))
print("F1 Score: {:.2f}%".format(f1_score(y_test, y_pred, average='weighted')*100))

Acurácia: 95.11%
Precisão: 91.90%
Recall: 95.11%
F1 Score: 93.07%
```

A matriz de confusão é uma ferramenta que auxilia na análise do desempenho de um modelo de classificação. Ela representa visualmente o número de acertos e erros do modelo em cada classe. No código apresentado, é calculada a matriz de confusão a partir das previsões do modelo KNN treinado anteriormente. Em seguida, é criado um gráfico de matriz de confusão usando a biblioteca seaborn, que permite visualizar de forma clara e objetiva o desempenho do modelo. O gráfico apresenta a distribuição de previsões do modelo em relação às classes reais.

**Figura 55 - Matriz de confusão**

```
# Calcular a matriz de confusão
cm = confusion_matrix(y_test, y_pred)

# Criar um gráfico de matriz de confusão
plt.figure(figsize=(4, 3))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predito')
plt.ylabel('Real')
plt.title('Matriz de Confusão')
plt.show()
```



A validação cruzada é uma técnica importante para avaliar a capacidade de generalização de um modelo de aprendizado de máquina. Nesse sentido, foi utilizada a técnica de KFold com 10 folds para avaliar o desempenho dos modelos de classificação. Para isso, o código em questão calcula o resultado da técnica de validação cruzada para o algoritmo KNN, utilizando o método `cross_val_score` da biblioteca Scikit-learn. O resultado dessa avaliação é armazenado em um dataframe e posteriormente é gerada uma descrição estatística dos resultados. O objetivo é verificar a estabilidade dos resultados do modelo e identificar possíveis problemas de overfitting ou underfitting. O resultado da validação cruzada é uma medida importante para avaliar o desempenho do modelo de classificação, visto que permite a estimativa de como o modelo irá se comportar em dados desconhecidos.

Figura 56 - Cross Validation

```
# KFold + cross-validation score
resultados_knn_clf = []
for i in range(5):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    score = cross_val_score(clf, X, y, cv = kfold)
    resultados_knn_clf.append(score.mean())

df_resultados_knn_clf = pd.DataFrame(resultados_knn_clf, columns=['resultados_knn'])
df_resultados_knn_clf.describe()
```

resultados_knn	
count	5.000000
mean	0.950718
std	0.000100
min	0.950586
25%	0.950663
50%	0.950740
75%	0.950746
max	0.950853

### 5.2.2. SGD Classifier

Após realizar a busca exaustiva dos melhores hiperparâmetros para o algoritmo KNN por meio do Grid Search, foi repetido o mesmo processo com o algoritmo SGD Classifier. Foram definidos valores diferentes para cada hiperparâmetro do algoritmo, como a função de perda a ser utilizada (loss), o tipo de penalização (penalty), o valor de regularização (alpha) e o número máximo de iterações (max\_iter).

Figura 57 - Códigos do algoritmo SGD Classifier

```
# Grid Search
# Definir os valores dos hiperparâmetros a serem testados
param_grid = {
    'loss': ['hinge', 'log_loss', 'modified_huber', 'squared_hinge', 'perceptron'],
    'penalty': ['l2', 'l1', 'elasticnet', None],
    'alpha': [0.0001, 0.001, 0.01, 0.1],
    'max_iter': [5, 10, 20, 50, 100, 1000]
}

# Criar um objeto GridSearchCV com os valores dos hiperparâmetros a serem testados
grid = GridSearchCV(SGDClassifier(random_state=42), param_grid)

# Treinar o modelo com o conjunto de treinamento
grid.fit(X_train, y_train)

# Imprimir os melhores hiperparâmetros encontrados
print("Melhores hiperparâmetros:", grid.best_params_)

Melhores hiperparâmetros: {'alpha': 0.0001, 'loss': 'hinge', 'max_iter': 5, 'penalty': 'l2'}
```

```
# Treinar o modelo
clf = SGDClassifier(random_state=42, alpha=0.0001, loss='hinge', max_iter=5, penalty='l2')
clf.fit(X_train, y_train)

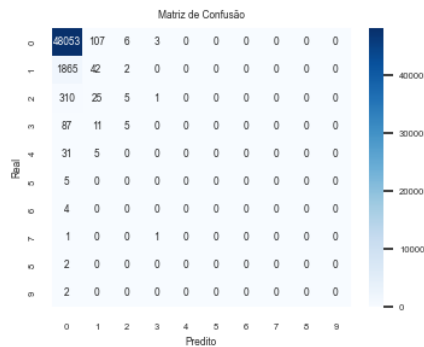
# Fazer previsões com o conjunto de teste
y_pred = clf.predict(X_test)

# Avaliar o desempenho do modelo
print("Acurácia: {:.2f}%".format(accuracy_score(y_test, y_pred)*100))
print("Precisão: {:.2f}%".format(precision_score(y_test, y_pred, average='weighted')*100))
print("Recall: {:.2f}%".format(recall_score(y_test, y_pred, average='weighted')*100))
print("F1 Score: {:.2f}%".format(f1_score(y_test, y_pred, average='weighted')*100))
```

Acurácia: 95.25%  
 Precisão: 90.72%  
 Recall: 95.25%  
 F1 Score: 92.93%

```
# Calcular a matriz de confusão
cm = confusion_matrix(y_test, y_pred)

# Criar um gráfico de matriz de confusão
sns.set(font_scale=0.5)
plt.figure(figsize=(4, 3))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predito')
plt.ylabel('Real')
plt.title('Matriz de Confusão')
plt.show()
```



```
# KFold + cross-validation score
resultados_sgd_classifier_clf = []
for i in range(5):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    score = cross_val_score(clf, X, y, cv = kfold)
    resultados_sgd_classifier_clf.append(score.mean())

df_resultados_sgd_classifier = pd.DataFrame(resultados_sgd_classifier_clf, columns=['resultados_sgd_classifier'])
df_resultados_sgd_classifier.describe()
```

resultados_sgd_classifier	
count	5.000000e+00
mean	9.517725e-01
std	7.700000e-09
min	9.517725e-01
25%	9.517725e-01
50%	9.517725e-01
75%	9.517725e-01
max	9.517725e-01

### 5.2.3. Random Forest Classifier

Realizamos uma busca pelos melhores hiperparâmetros para o algoritmo Random Forest Classifier por meio do Grid Search. Assim como no caso do KNN, foram definidos

valores diferentes para cada hiperparâmetro do algoritmo, como o número de árvores na floresta (`n_estimators`), `criterion`, o número mínimo de amostras necessárias para dividir um nó interno (`min_samples_split`), o número mínimo de amostras necessárias em uma folha (`min_samples_leaf`), entre outros.

**Figura 58 - Códigos do algoritmo Random Forest Classifier**

```
# Grid Search
# Definir os valores dos hiperparâmetros a serem testados
param_grid = {
    'n_estimators': [25, 50, 75, 100],
    'criterion': ['gini', 'entropy', 'log_loss'],
    'min_samples_split': [2, 4, 6, 8, 10],
    'min_samples_leaf': [1, 3, 5, 7]
}

# Criar um objeto GridSearchCV com os valores dos hiperparâmetros a serem testados
grid = GridSearchCV(RandomForestClassifier(random_state=42), param_grid)

# Treinar o modelo com o conjunto de treinamento
grid.fit(X_train, y_train)

# Imprimir os melhores hiperparâmetros encontrados
print("Melhores hiperparâmetros:", grid.best_params_)

Melhores hiperparâmetros: {'criterion': 'gini', 'min_samples_leaf': 7, 'min_samples_split': 2, 'n_estimators': 100}

# Treinar o modelo
clf = RandomForestClassifier(random_state=42, criterion='gini', min_samples_leaf=7, min_samples_split=2, n_estimators=100)
clf.fit(X_train, y_train)

# Fazer previsões com o conjunto de teste
y_pred = clf.predict(X_test)

# Avaliar o desempenho do modelo
print("Acurácia: {:.2f}%".format(accuracy_score(y_test, y_pred)*100))
print("Precisão: {:.2f}%".format(precision_score(y_test, y_pred, average='weighted')*100))
print("Recall: {:.2f}%".format(recall_score(y_test, y_pred, average='weighted')*100))
print("F1 Score: {:.2f}%".format(f1_score(y_test, y_pred, average='weighted')*100))

Acurácia: 95.20%
Precisão: 92.14%
Recall: 95.20%
F1 Score: 93.12%

# Calcular a matriz de confusão
cm = confusion_matrix(y_test, y_pred)

# Criar um gráfico de matriz de confusão
plt.figure(figsize=(4, 3))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predito')
plt.ylabel('Real')
plt.title('Matriz de Confusão')
plt.show()
```



```
# KFold + cross-validation score
resultados_random_forest_classifier_clf = []
for i in range(5):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    score = cross_val_score(clf, X, y, cv = kfold)
    resultados_random_forest_classifier_clf.append(score.mean())

df_resultados_random_forest_classifier = pd.DataFrame(resultados_random_forest_classifier_clf, columns=['resultados_random_forest_
df_resultados_random_forest_classifier.describe()
```

resultados_random_forest_classifier	
count	5.000000
mean	0.950243
std	0.000067
min	0.950165
25%	0.950201
50%	0.950254
75%	0.950254
max	0.950343

### 5.2.4. Decision Tree Regressor

Foram realizados os mesmos passos para o algoritmo Decision Tree Regressor. Inicialmente, realizamos uma busca pelos melhores hiperparâmetros para o modelo de regressão utilizando o Grid Search. Foram definidos valores diferentes para o critério.

**Figura 59 - Códigos do algoritmo Decision Tree Regressor**

```
# Grid Search
# Definir os valores dos hiperparâmetros a serem testados
param_grid = {
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error'],
}

# Criar um objeto GridSearchCV com os valores dos hiperparâmetros a serem testados
grid = GridSearchCV(DecisionTreeRegressor(), param_grid)

# Treinar o modelo com o conjunto de treinamento
grid.fit(X_train, y_train)

# Imprimir os melhores hiperparâmetros encontrados
print("Melhores hiperparâmetros:", grid.best_params_)

Melhores hiperparâmetros: {'criterion': 'squared_error'}
```

```
# Treinar o modelo
clf = DecisionTreeRegressor(criterion='squared_error')
clf.fit(X_train, y_train)

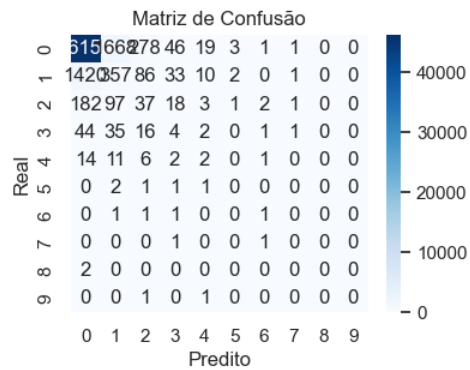
# Fazer previsões com o conjunto de teste
y_pred = clf.predict(X_test)

# Avaliar o desempenho do modelo
print("Acurácia: {:.2f}%".format(accuracy_score(y_test, y_pred)*100))
print("Precisão: {:.2f}%".format(precision_score(y_test, y_pred, average='weighted')*100))
print("Recall: {:.2f}%".format(recall_score(y_test, y_pred, average='weighted')*100))
print("F1 Score: {:.2f}%".format(f1_score(y_test, y_pred, average='weighted')*100))

Acurácia: 92.05%
Precisão: 92.63%
Recall: 92.05%
F1 Score: 92.34%
```

```
# Calcular a matriz de confusão
cm = confusion_matrix(y_test, y_pred)

# Criar um gráfico de matriz de confusão
plt.figure(figsize=(4, 3))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predito')
plt.ylabel('Real')
plt.title('Matriz de Confusão')
plt.show()
```



```
# KFold + cross-validation score
resultados_decision_tree_regressor_clf = []
for i in range(5):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    score = cross_val_score(clf, X, y, cv=kfold)
    resultados_decision_tree_regressor_clf.append(score.mean())

df_resultados_decision_tree_regressor = pd.DataFrame(resultados_decision_tree_regressor_clf, columns=['resultados_decision_tree_regressor'])
df_resultados_decision_tree_regressor.describe()
```

resultados_decision_tree_regressor	
count	5.000000
mean	-0.572805
std	0.017057
min	-0.588545
25%	-0.584833
50%	-0.578678
75%	-0.565233
max	-0.546737

## 5.2.5. Regressão Logística

Foram realizados os mesmos passos para a Regressão Logística. Inicialmente, foi realizada uma busca por meio do Grid Search para encontrar os melhores hiperparâmetros do modelo, tais como a estratégia de solução (solver).



```
# KFold + cross-validation score
resultados_logistic_regression_clf = []
for i in range(5):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    score = cross_val_score(clf, X, y, cv = kfold)
    resultados_logistic_regression_clf.append(score.mean())

df_resultados_logistic_regression = pd.DataFrame(resultados_logistic_regression_clf, columns=['resultados_logistic_regression'])
df_resultados_logistic_regression.describe()
```

resultados_logistic_regression	
count	5.000000e+00
mean	9.517725e-01
std	7.700000e-09
min	9.517725e-01
25%	9.517725e-01
50%	9.517725e-01
75%	9.517725e-01
max	9.517725e-01

## 6. Interpretação dos Resultados

Com o objetivo de avaliar diferentes algoritmos de aprendizado de máquina para a tarefa de classificação de imagens de radiografia torácica, foram realizados experimentos com os algoritmos KNN Classifier, SGD Classifier, Logistic Regression, Random Forest Classifier e Decision Tree Regressor. Inicialmente, realizou-se a busca exaustiva pelos melhores hiperparâmetros para cada algoritmo, utilizando o Grid Search e a validação cruzada com 5 folds. Em seguida, os modelos foram treinados e avaliados utilizando métricas de desempenho, como acurácia, precisão, recall e F1-score. Por fim, os resultados foram comparados e analisados em conjunto, a fim de identificar o algoritmo que apresentou o melhor desempenho para a tarefa em questão. Para isso, utilizou-se a técnica de concatenação de resultados, a partir da qual os resultados dos diferentes modelos foram combinados em um único Data-Frame e analisados estatisticamente. Os resultados obtidos são apresentados na figura abaixo.

**Figura 61 - Comparação de resultados**

```
# Criar uma lista com os DataFrames a serem concatenados
dataframes = [df_resultados_knn_clf,
               df_resultados_sgd_classifier,
               df_resultados_logistic_regression,
               df_resultados_random_forest_classifier,
               df_resultados_decision_tree_regressor]

# Concatenar os DataFrames ao longo do eixo das colunas (axis=1)
resultados = pd.concat(dataframes, axis=1)

# Exibir o resultado
print(resultados.describe())
```

	resultados_knn	resultados_sgd_classifier \
count	5.000000	5.000000e+00
mean	0.950718	9.517725e-01
std	0.000100	7.700000e-09
min	0.950586	9.517725e-01
25%	0.950663	9.517725e-01
50%	0.950740	9.517725e-01
75%	0.950746	9.517725e-01
max	0.950853	9.517725e-01

	resultados_logistic_regression	resultados_random_forest_classifier \
count	5.000000e+00	5.000000
mean	9.517725e-01	0.950243
std	7.700000e-09	0.000067
min	9.517725e-01	0.950165
25%	9.517725e-01	0.950201
50%	9.517725e-01	0.950254
75%	9.517725e-01	0.950254
max	9.517725e-01	0.950343

	resultados_decision_tree_regressor
count	5.000000
mean	-0.572805
std	0.017057
min	-0.588545
25%	-0.584833
50%	-0.578678
75%	-0.565233
max	-0.546737

Após avaliar os resultados obtidos com os modelos SGD Classifier e Regressão Logística, verificou-se que ambos atingiram uma acurácia de 95,18%, sendo considerados os melhores modelos para a previsão de acidentes em contratos de manutenção rodoviária. Dessa forma, optou-se por utilizar o modelo de Regressão Logística na previsão de acidentes. O próximo passo consiste na seleção das variáveis mais relevantes para o modelo, as quais serão utilizadas como entrada para a predição.

**Figura 62 - Seleção das variáveis**

```
# Selecionando y_obra e X_obra
y_obra = acidente_obra['num_acidente']
X_obra = acidente_obra[['mes', 'ano', 'br', 'uf', 'km_inicial', 'km_final', 'extensao']]
```

Uma das tarefas realizadas foi a codificação das variáveis categóricas, como a coluna "UF", para que fossem incluídas no modelo. Para isso, foi utilizado o método OneHotEncoder do pacote scikit-learn. Este método cria uma coluna para cada categoria presente na coluna original e preenche as células com valores binários 0 e 1. Após a codificação, foram criados dataframes para as colunas codificadas e em seguida foram combinados com as outras variáveis independentes, de modo a formar um único dataframe.

**Figura 63 - Codificação**

```
# OneHotEncoder
# Codificar as colunas "UF" usando OneHotEncoder
enc = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_obra_enc = enc.fit_transform(X_obra[['uf']])

# Obter as categorias codificadas
uf_categories = enc.categories_[0]

# Criar dataframes para as colunas codificadas
uf_encoded = pd.DataFrame(X_obra_enc[:, :len(uf_categories)], index=X_obra.index, columns=[f'uf_{category}' for category in uf_categories])

# Combinar as colunas codificadas com as outras variáveis independentes
X_obra = pd.concat([X_obra.drop(columns=['uf']), uf_encoded], axis=1)
```

Realizado a padronização das variáveis independentes para garantir que todas as variáveis tenham a mesma escala, a fim de que nenhuma delas influencie mais ou menos o resultado da previsão de acidentes.

**Figura 64 - Normalização das variáveis**

```
# StandardScaler
scaler = StandardScaler()
X_obra = scaler.fit_transform(X_obra)
```

Os resultados indicam que o modelo de previsão apresenta uma acurácia razoável, de 74,41%, o que significa que ele acerta a classificação das observações em cerca de 3 em cada

4 casos. Já a precisão de 64,16% indica que, das classificações positivas feitas pelo modelo, apenas cerca de 2 em cada 3 estão corretas. O recall de 74,41% significa que o modelo é capaz de identificar cerca de 3 em cada 4 casos de forma correta, enquanto o F1 Score de 67,98% indica um bom equilíbrio entre a precisão e o recall. Em geral, os resultados sugerem que o modelo tem potencial para ser útil na previsão de acidentes em contratos de manutenção rodoviária, mas podem ser melhorados em termos de precisão..

**Figura 65 - Predição de acidentes**

```
# Fazer previsões com o conjunto de teste
y_obra_pred = clf.predict(X_obra)

# Avaliar o desempenho do modelo
print("Acurácia: {:.2f}%".format(accuracy_score(y_obra, y_obra_pred)*100))
print("Precisão: {:.2f}%".format(precision_score(y_obra, y_obra_pred, average='weighted')*100))
print("Recall: {:.2f}%".format(recall_score(y_obra, y_obra_pred, average='weighted')*100))
print("F1 Score: {:.2f}%".format(f1_score(y_obra, y_obra_pred, average='weighted')*100))
```

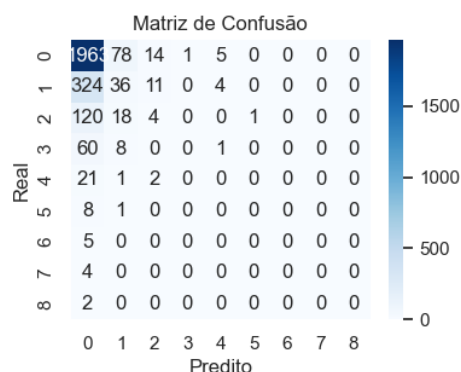
Acurácia: 74.41%  
 Precisão: 64.16%  
 Recall: 74.41%  
 F1 Score: 67.98%

A matriz de confusão apresenta um modelo que não conseguiu prever nenhuma das outras nove classes além da primeira, com 100% de acurácia apenas para esta classe. Isso indica um desbalanceamento de classes, onde a classe majoritária (primeira classe) está dominando o modelo.

**Figura 66 - Matriz de confusão**

```
# Calcular a matriz de confusão
cm = confusion_matrix(y_obra, y_obra_pred)

# Criar um gráfico de matriz de confusão
plt.figure(figsize=(4, 3))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predito')
plt.ylabel('Real')
plt.title('Matriz de Confusão')
plt.show()
```



Adicionado a coluna com as previsões de acidentes nos contratos de manutenção rodoviária na tabela 'acidente\_obra'.

Figura 67 - Adicionando a previsão ao dataframe

```
# Adicionar a coluna com as previsões na tabela
acidente_obra['num_acidente_pred'] = y_obra_pred
```

Este trecho de código salva o resultado final da análise de previsão de acidentes em um novo arquivo CSV.

Figura 68 - Salvando o dataframe em uma arquivo CSV

```
# Salvar o resultado em um novo arquivo CSV
acidente_obra.to_csv('acidente_obra.csv', index=False, sep=';', decimal=',', encoding = 'cp1252')
```

A partir da análise estatística descritiva do conjunto de dados de acidentes em obras de manutenção rodoviária, é possível observar que os valores mínimos para o número de acidentes reais e previstos foram zero, indicando que há trechos em que não ocorreram acidentes. Além disso, a média do número de acidentes por mês em cada trecho é baixa, sendo de 0,402 para o caso real e de 0,094 para os estimados, indicando que a maioria dos trechos não teve acidentes durante o período analisado. O valor máximo de 8 para o número de acidentes reais e 5 para os previstos mostram que houve trechos com um número elevado de acidentes em relação aos demais.

```
acidente_obra.describe()
```

	br	km_inicial	km_final	extensao	num_acidente	mes	ano	num_acidente_pred
count	2692.000000	2692.000000	2692.000000	2692.000000	2692.000000	2692.000000	2692.000000	2692.000000
mean	286.961738	250.970468	323.086397	72.115929	0.402303	6.471397	2015.285661	0.093611
std	138.512331	244.883630	251.268563	26.406480	0.913442	3.470682	3.408807	0.402708
min	40.000000	0.000000	4.900000	4.900000	0.000000	1.000000	2007.000000	0.000000
25%	146.000000	62.000000	118.600000	52.000000	0.000000	3.000000	2013.000000	0.000000
50%	364.000000	152.900000	235.700000	73.600000	0.000000	6.000000	2016.000000	0.000000
75%	369.000000	374.100000	471.300000	90.400000	0.000000	9.250000	2018.000000	0.000000
max	494.000000	857.200000	949.800000	118.900000	8.000000	12.000000	2021.000000	5.000000



## 7. Apresentação dos Resultados

**Data Science Workflow Canvas\***

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

**Title:** USO DE MACHINE LEARNING PARA PREVER ACIDENTES EM RODOVIAS BRASILEIRAS COM FOCO EM CONTRATOS DE MANUTENÇÃO

<p><b>1 Problem Statement</b> What problem are you trying to solve? What larger issues do the problem address?</p> <p>Acidentes são um grande problema do país.</p> <p>Os contratos de manutenção ajudam a evitar acidentes.</p>	<p><b>2 Outcomes/Predictions</b> What prediction(s) are you trying to make? Identify applicable predictor (x) and/or target (y) variables.</p> <p>Prever número de acidentes em trechos rodoviários com base em dados de rodovia, uf, km inicial e final do trecho, extensão, mês e ano.</p>	<p><b>3 Data Acquisition</b> Where are you sourcing your data from? Is there enough data? Can you work with it?</p> <p>Base de acidentes da PRF, base de trechos rodoviários do DNIT e base de contratos de manutenção de uma determinada empresa.</p>
<p><b>4 Modeling</b> What models are appropriate to use given your outcomes?</p> <p>Acredito que classificação seja o melhor para o modelo.</p> <p>Testar quantos algoritmos for possível.</p>	<p><b>5 Model Evaluation</b> How can you evaluate your model's performance?</p> <p>Cross Validation.</p>	<p><b>6 Data Preparation</b> What do you need to do to your data in order to run your model and achieve your outcomes?</p> <p>Sentar em frente ao pc e criar o código, esta nas minhas mãos.</p>

### ✓ Activation

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

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

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

## 8. Links

Aqui você deve disponibilizar os links para o vídeo com sua apresentação de 5 minutos e para o repositório contendo os dados utilizados no projeto, scripts criados, etc.

Link para o vídeo: <youtube.com/...>

Link para o repositório: <github.com/...>

## REFERÊNCIAS

Um projeto/relatório técnico de Ciência de Dados não requer revisão bibliográfica. Portanto, a inclusão das referências não é obrigatória. No entanto, caso você deseje incluir referências relacionadas às tecnologias ou às metodologias usadas em seu trabalho, relacione-as de acordo com o modelo a seguir.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

## APÊNDICE

### **Programação/Scripts**

Cole aqui seus scripts em Python e/ou R.

### **Gráficos**

Cole aqui workflows (KNIME), gráficos e figuras que você tenha gerado e não colocou no texto principal.

### **Tabelas**

Cole aqui tabelas de dados que você tenha gerado e não colocou no texto principal.