PUC Minas
Virtual

Pós-graduação Lato Sensu em Ciência de Dados e Big Data
Trabalho de Conclusão de Curso

CARACTERIZAÇÃO DE ACIDENTES DE TRÂNSITO EM TRECHOS DE RODOVIAS FEDERAIS E A APLICAÇÃO DE MODELOS DE MACHINE LEARNING PARA A CLASSIFICAÇÃO DO ESTADO FÍSICO DOS ENVOLVIDOS

Aluna: **Lilian Campos Soares**

Belo Horizonte
2021

Introdução

- Mortes e lesões por acidentes de trânsito são consideradas um problema de saúde pública
- Exigem ações para prevenção e redução de suas consequências.
- Conhecer os fatores contribuintes dos acidentes é fundamental para intervir preventivamente, avaliando as interações entre os elementos que compõem o sistema de trânsito (o homem, o veículo e a via), identificando as causas dos acidentes e agindo na redução de suas consequências.

Contextualização

- Tema: segurança viária
- Escopo:
  - Caracterização de acidentes de trânsito de uma UF brasileira nos anos de 2017 a 2020.
  - A UF será definida pela extensão de malha rodoviária federal e o total de acidentes.
  - Serão gerados datasets com os acidentes do período, efetuadas análises dos dados, produzidos de índices de acidentalidade e aplicados modelos de *machine learning* para a classificação do estado físico dos envolvidos.

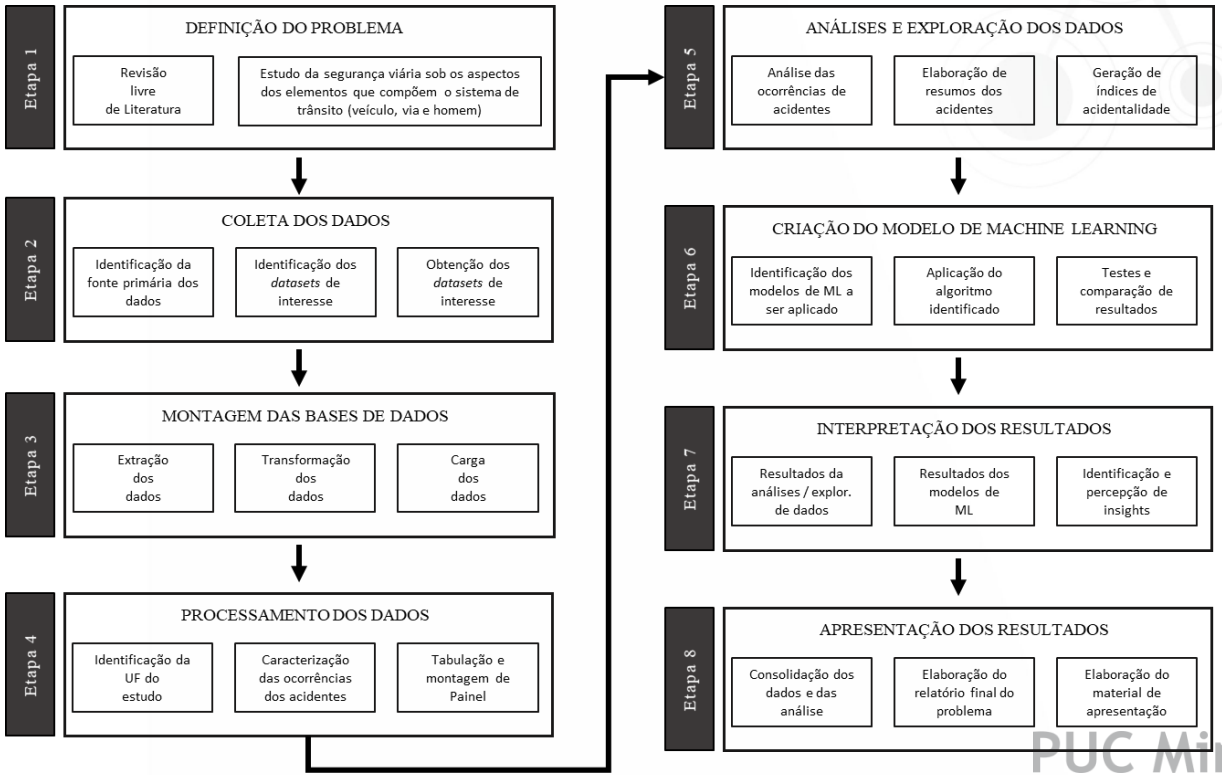
Problema proposto

- Partindo do problema do quadro alarmante da acidentalidade de trânsito no Brasil, a pesquisa a ser desenvolvida busca responder a seguinte questão:
    - De acordo com as informações registradas pela PRF na base de dados de acidentes nas rodovias federais nos anos de 2017 a 2020, qual é a extensão das lesões geradas pelos acidentes rodoviários em uma unidade da federação?

Estratificação do problema

W	Resposta
Por que?	Quadro alarmante da acidentalidade de trânsito no Brasil
Quem?	Informações registradas pela PRF em sua base de dados de acidentes nas rodovias federais
O que?	A extensão das lesões geradas pelos acidentes rodoviários em uma unidade da federação
Onde?	Unidade da federação brasileira que apresentou a mais alta média de acidentes por quilômetro de rodovia federal
Quando?	2017 a 2020

Metodologia



Metodologia

- Os procedimentos foram executados utilizando-se:
  - a linguagem de programação Python na versão 3.8.2, pelo Jupyter Notebook (na versão 6.1.4) do Anaconda3 para a coleta de dados, montagem das bases de dados, processamento dos dados, análise e exploração dos dados, criação do modelo de *machine learning* e interpretação dos resultados;
  - o Microsoft PowerBI Desktop (versão de dezembro de 2020) para o painel analítico;
  - o Microsoft Excel (para determinados suportes); o Microsoft Winword (para o relatório final) e o Microsoft Powerpoint (para o material de apresentação).

Bibliotecas Python

Coleta de dados, montagem das bases de dados, processamento dos dados, e análise e exploração dos dados

```
#Carregamento de bibliotecas gerais
import pandas as pd
import numpy as np
import seaborn as sns
import shutil
import os
import io
import datetime
from datetime import datetime
import matplotlib.pyplot as plt
```

```
# Carregamento de bibliotecas para a obtenção de arquivos
from time import sleep
from helium import click, kill_browser, start_chrome
from selenium import webdriver
import zipfile
```

```
# Instalação e carregamento de bibliotecas para descompactar rar
!pip install pyunpack
from pyunpack import Archive
!pip install patool
import patoolib
```

A interpretação de dados, usou basicamente as mesmas bibliotecas.

Criação do modelo de *machine learning*

```
#Carregamento de bibliotecas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#Carregamento do pacote com funções estatísticas
import scipy.stats as stats

#Carregamento do pacote Sklearn
import sklearn.metrics as m
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn import tree
from sklearn.model_selection import RandomizedSearchCV
```

```
#Teste com o algoritmo SGDClassifier
from sklearn.linear_model import SGDClassifier
```

```
#Comparação com o SGDClassifier
from sklearn.svm import SVC
```

```
#Importando biliotecas para trabalhar com o algoritmo Decision Tree
from sklearn.tree import DecisionTreeClassifier
```

```
#Teste com o algoritmo Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
```

```
#Predição dos dados de teste usando o modelo treinado e apresentação da matriz de confusão
from sklearn.metrics import confusion_matrix
```

```
#Criação de um relatório de classificação nos dados de predição para checar as métricas de acurácia
from sklearn.metrics import classification_report
```

```
#Confusion Matrix
import itertools
```

Coleta de dados

- Dados do Departamento Nacional de Infraestrutura Terrestre (DNIT)
  - extensão de malha rodoviária federal - SNV
  - modelagem do fluxo de veículos - VMDa
- Dados do Departamento de Polícia Rodoviária Federal (PRF)
  - ocorrências de acidentes registradas nos anos de 2017 a 2020
    - dados agrupados por ocorrência: arquivos “datatran”
    - dados agrupados por pessoa: arquivos “acidentes”

Coleta de dados

- Toda a coleta de dados foi realizada via Python, desde o download dos arquivos, descompactação, movimentação para as pastas do projeto

```python
# Criação do recurso para download dos arquivos de SNV do DNIT
# O arquivo "SNV_201801B.xls" se refere aos dados de rodovias de 2017
# O arquivo "SNV_201903A.xls" se refere aos dados de rodovias de 2018
# O arquivo "SNV_202001A.xls" se refere aos dados de rodovias de 2019
# O arquivo "SNV_202101A.xls" se refere aos dados de rodovias de 2020
opts = webdriver.ChromeOptions()
opts.set_capability("loggingPrefs", {"performance": "ALL"})

driver = start_chrome("http://servicos.dnit.gov.br/dnitcloud/index.php/s/oTpPRmYs5AAd
sleep(12)
click("SNV_201801B.xls")
sleep(12)
click("SNV_201903A.xls")
sleep(12)
click("SNV_202001A.xls")
sleep(12)
click("SNV_202101A.xls")
sleep(12)kill_browser()
```

```python
# Armazenamento dos arquivos SNV no diretório do trabalho e exclusão dos arquivos bai
path = r"C:\Users\lilia\Downloads\TCC PUC\xls"
os.mkdir(path)
shutil.copyfile(r"C:\Users\lilia\Downloads\SNV_201801B.xls", r"C:\Users\lilia\Downloa
shutil.copyfile(r"C:\Users\lilia\Downloads\SNV_201903A.xls", r"C:\Users\lilia\Downloa
shutil.copyfile(r"C:\Users\lilia\Downloads\SNV_202001A.xls", r"C:\Users\lilia\Downloa
shutil.copyfile(r"C:\Users\lilia\Downloads\SNV_202101A.xls", r"C:\Users\lilia\Downloa
os.remove(r"C:\Users\lilia\Downloads\SNV_201801B.xls")
os.remove(r"C:\Users\lilia\Downloads\SNV_201903A.xls")
os.remove(r"C:\Users\lilia\Downloads\SNV_202001A.xls")
os.remove(r"C:\Users\lilia\Downloads\SNV_202101A.xls")
#Como as planilhas possuíam figuras e células mescladas, foi feito um tratamento no excel e geradas tabelaa em formato csv d
```

```python
# Descompactação dos arquivos da PRF
descompactar = zipfile.ZipFile(r"C:\Users\lilia\Downloads\datatran2017.zip")
descompactar.extractall(r"C:\Users\lilia\Downloads")
descompactar.close()
descompactar = zipfile.ZipFile(r"C:\Users\lilia\Downloads\datatran2018.zip")
descompactar.extractall(r"C:\Users\lilia\Downloads")
descompactar.close()
descompactar = zipfile.ZipFile(r"C:\Users\lilia\Downloads\datatran2019.zip")
descompactar.extractall(r"C:\Users\lilia\Downloads")
descompactar.close()
descompactar = zipfile.ZipFile(r"C:\Users\lilia\Downloads\datatran2020.zip")
descompactar.extractall(r"C:\Users\lilia\Downloads")
descompactar.close()
descompactar = zipfile.ZipFile(r"C:\Users\lilia\Downloads\acidentes2018.zip")
descompactar.extractall(r"C:\Users\lilia\Downloads")
descompactar.close()
descompactar = zipfile.ZipFile(r"C:\Users\lilia\Downloads\acidentes2019.zip")
descompactar.extractall(r"C:\Users\lilia\Downloads")
descompactar.close()
descompactar = zipfile.ZipFile(r"C:\Users\lilia\Downloads\acidentes2020.zip")
descompactar.extractall(r"C:\Users\lilia\Downloads")
descompactar.close()
patoolib.extract_archive(r"C:\Users\lilia\Downloads\acidentes2017.rar", outdir=r"C:\Users\lilia\Downloads")
```

Montagem da base de dados

- A montagem da base de dados se deu a partir da preparação de dados para identificar a UF que será o objeto do estudo.
- Iniciou-se pelos dados do DNIT de extensão de malha rodoviária federal (SNV) e de fluxo de veículos (VMDa) visando construir um ranking de dados por UF.

Extensões de rodovias por UF - 2020

UF	TOTAL_Sem_planejada
MG	8860.8
BA	7593.4
RS	5800.2
PA	5121.5
MT	5098.3

```
#Verificação das colunas que tem UF e a Extensão total em KM (sem as planejadas) em 2017, 2018, 2019 e 2020
#resumo_snv_2017.info()
#resumo_snv_2018.info()
#resumo_snv_2019.info()
#resumo_snv_2020.info()
print("Extensões de rodovias por UF - 2017")
resumo_snv_2017 = resumo_snv_2017.sort_values(by ='TOTAL_Sem_planejada', ascending=False)
print(resumo_snv_2017[['UF', 'TOTAL_Sem_planejada']])
print("Extensões de rodovias por UF - 2018")
resumo_snv_2018 = resumo_snv_2018.sort_values(by ='TOTAL_Sem_planejada', ascending=False)
print(resumo_snv_2018[['UF', 'TOTAL_Sem_planejada']])
print("Extensões de rodovias por UF - 2019")
resumo_snv_2019 = resumo_snv_2019.sort_values(by ='TOTAL_Sem_planejada', ascending=False)
print(resumo_snv_2019[['UF', 'TOTAL_Sem_planejada']])
print("Extensões de rodovias por UF - 2020")
resumo_snv_2020 = resumo_snv_2020.sort_values(by ='TOTAL_Sem_planejada', ascending=False)
print(resumo_snv_2020[['UF', 'TOTAL_Sem_planejada']])
#O resultado mostra que o estado de MG é o que possui a maior extensão de rodovias federais em KM em todos os anos
```

Montagem da base de dados

- Após a junção dos arquivos da PRF, verificou-se a quantidade de acidentes por UF.

Totais de acidentes por UF – 2017 a 2020

MG	38.879
SC	34.807
PR	33.517
RS	19.602
RJ	19.372

```
#Verificação de dados de acidentes por UF: para definição da UF objeto do estudo
abcd = pd.concat([a,b,c,d], join="inner")
abcd.to_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\datatran2.csv", index=False)
datatran_abcd = pd.read_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\datatran2.csv", sep=',', decimal='.', encoding = 'utf_8')
print("Total de acidentes por UF - 2017 a 2020")
datatran_abcd.groupby('uf')['id'].count().sort_values(ascending=False)

#MG é a UF com a maior quantidade de registros de ocorrências de acidentes em todos os anos
#(a contagem neste ponto inclui registros que tem algum campo nulo ('0' / NaN))

#Conclusão: MG é o de maior extensão rodoviária federal, com a maior quantidade de acidentes por ano e em segundo lugar em v
```

Montagem da base de dados

- Foi realizada a produção de indicadores por ano/UF.

Tabela resumo por UF - Indicadores com base em dados de 2017

UF	km_rodovia	acidentes	acidentes_km	mortes	mortes_km
MG	9577.00	12730	1.3292	869	0.0907
PR	4053.40	10689	2.6370	612	0.1510
SC	2352.10	10665	4.5342	381	0.1620
RS	5799.40	6386	1.1011	391	0.0674
SP	1122.30	6011	5.3560	255	0.2272

Foi possível identificar a UF para o estudo:

(i) MG é a UF de maior extensão rodoviária federal;

(ii) MG apresentou a maior quantidade de acidentes por ano e do total do período;

(iii) MG apresentou a maior quantidade de mortes por ano e do total por período.

```
#Verificação de dados de acidentes por UF: tabela resumo por UF com indicadores (acidente/km; mortes/km)

print("Tabela resumo por UF - Indicadores com base em dados de 2017")
a.groupby('uf', as_index = False)['id'].count()
acidentes_2017 = pd.DataFrame(a.groupby('uf', as_index = False)['id'].count()).reset_index()
acidentes_2017.rename(columns={'id':'acidentes'}, inplace=True)
acidentes_2017.rename(columns={'uf':'UF'}, inplace=True)
acidentes_2017.drop(columns=['index'], axis=1, inplace=True)

a.groupby('uf', as_index = False)['mortos'].sum()
mortes_2017 = pd.DataFrame(a.groupby('uf', as_index = False) ['mortos'].sum()).reset_index()
mortes_2017.rename(columns={'mortos':'mortes'}, inplace=True)
mortes_2017.rename(columns={'uf':'UF'}, inplace=True)
mortes_2017.drop(columns=['index'], axis=1, inplace=True)

#print(acidentes_2017)
#print(mortes_2017)
acidentes_2017.drop(columns=['UF'], axis=1, inplace=True)
mortes_2017.drop(columns=['UF'], axis=1, inplace=True)

resumo_snv_2017_2 = pd.read_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\Resumo_2018_SNV201801B.csv", sep=';', decimal=',', en
resumo_snv_2017_2.drop(columns=['PLANEJADA','RNP_TRAVESSIA','RNP_LEITO_NATUAL','RNP_EM_OBRAS_IMP','RNP_IMPLANT','RNP_EM_OBRA
resumo_snv_2017_2['acidentes_km'] = '0'
resumo_snv_2017_2['mortes_km'] = '0'
resumo_snv_2017_2['acidentes_km'] = resumo_snv_2017_2.acidentes_km.astype('float64')
resumo_snv_2017_2['mortes_km'] = resumo_snv_2017_2.mortes_km.astype('float64')
resumo_snv_2017_2.rename(columns={'TOTAL_Sem_planejada':'km_rodovia'}, inplace=True)

temp = pd.concat([acidentes_2017, resumo_snv_2017_2], axis=1, sort=False).reset_index()
resumo_uf_2017 = pd.concat([temp, mortes_2017], axis =1, sort = False).reset_index()

resumo_uf_2017['acidentes_km'] = resumo_uf_2017['acidentes']/resumo_uf_2017['km_rodovia']
resumo_uf_2017['mortes_km'] = resumo_uf_2017['mortes']/resumo_uf_2017['km_rodovia']
resumo_uf_2017.drop(columns=['level_0'], axis=1, inplace=True)
resumo_uf_2017.drop(columns=['index'], axis=1, inplace=True)
resumo_uf_2017 = resumo_uf_2017.reindex(columns=['UF', 'km_rodovia', 'acidentes', 'acidentes_km', 'mortes', 'mortes_km'])
print(resumo_uf_2017)
resumo_uf_2017.to_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\resumo_uf_2017.csv", index=False, sep=';', decimal=',', encodin

#Em 2017, SP apresentou o mais alto índice de acidente/Km, seguido do DF, SC e RJ.
```

Processamento de dados

- Com as bases de “datatran” e “acidentes” montadas para a UF de MG, passou-se aos procedimentos de processamento dos dados, com tratamento e enriquecimento dos dados da base.

```python
#Tratamento dos arquivos datatran.CSV - conversão de atributos
from numpy import int64
datatran_mg['id'] = datatran_mg.id.astype('int64')
datatran_mg['br'] = datatran_mg.br.astype('int64')
datatran_mg['km'] = datatran_mg.km.astype('int64')
datatran_mg['uso_solo'] = datatran_mg.uso_solo.astype('string')
datatran_mg['data_inversa'] = datatran_mg.data_inversa.astype('datetime64')
#datatran_mg.info()
```

```python
#Tratamento dos arquivos datatran.CSV - configurando registros Urbano e Rural
datatran_mg['uso_solo'].replace(['Sim'], 'Urbano', inplace=True)
datatran_mg['uso_solo'].replace(['Não'], 'Rural', inplace=True)
```

```python
#Tratamento dos arquivos datatran.CSV - remoção de atributos que não serão utilizados
datatran_mg.drop(columns = ['sentido_via','regional', 'delegacia', 'uop'], axis=1, inplace=True)
datatran_mg.reset_index(inplace=True)
datatran_mg.drop(columns=['index'], axis=1, inplace=True)
```

```python
#Tratamento dos arquivos acidentes.CSV - conversão de atributos
from numpy import int64
acidentes_mg['id'] = acidentes_mg.id.astype('int64')
acidentes_mg['pesid'] = acidentes_mg.pesid.astype('int64')
acidentes_mg['br'] = acidentes_mg.br.astype('int64')
acidentes_mg['km'] = acidentes_mg.km.astype('int64')
acidentes_mg['uso_solo'] = acidentes_mg.uso_solo.astype('string')
acidentes_mg['data_inversa'] = acidentes_mg.data_inversa.astype('datetime64')
#acidentes_mg.info()
```

```python
#Tratamento dos arquivos acidentes.CSV - configurando registros Urbano e Rural
acidentes_mg['uso_solo'].replace(['Sim'], 'Urbano', inplace=True)
acidentes_mg['uso_solo'].replace(['Não'], 'Rural', inplace=True)
```

```python
#Tratamento dos arquivos acidentes.CSV - remoção de atributos que não serão utilizados
acidentes_mg.drop(columns = ['sentido_via','regional', 'delegacia', 'uop', 'id_veiculo'
acidentes_mg.reset_index(inplace=True)
acidentes_mg.drop(columns=['index'], axis=1, inplace=True)
```

Processamento de dados

- Foi incluída coluna ‘elemento_transito’ e o atributo informado conforme a causa do acidente.

```
#Enriquecimento de dados dos arquivos datatran.CSV - preenche elemento_transito
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Agressão Externa"), datatran_mg['elemento_trans
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Carga excessiva e/ou mal acondicionada"), datat
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Condutor Dormindo"), datatran_mg['elemento_tran
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Desobediência às normas de trânsito pelo condut
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Desobediência às normas de trânsito pelo pedest
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Falta de Atenção à Condução"), datatran_mg['ele
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Falta de Atenção do Pedestre"), datatran_mg['el
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Ingestão de Álcool"), datatran_mg['elemento_tra
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Ingestão de álcool e/ou substâncias psicoativas
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Ingestão de Substâncias Psicoativas"), datatran
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Ingestão de álcool pelo condutor"), datatran_mg
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Mal Súbito"), datatran_mg['elemento_transito'].
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Não guardar distância de segurança"), datatran_
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Ultrapassagem Indevida"), datatran_mg['elemento
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Velocidade Incompatível"), datatran_mg['element
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Ausência de reação do condutor"), datatran_mg['
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Manobra de mudança de faixa"), datatran_mg['ele
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Reação tardia ou ineficiente do condutor"), dat
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Transitar na contramão"), datatran_mg['elemento
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Conversão proibida"), datatran_mg['elemento_tra
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Avarias e/ou desgaste excessivo no pneu"), data
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Defeito Mecânico no Veículo"), datatran_mg['ele
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Deficiência ou não Acionamento do Sistema de Il
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Demais falhas mecânicas ou elétricas"), datatra
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Animais na Pista"), datatran_mg['elemento_trans
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Defeito na Via"), datatran_mg['elemento_transit
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Fenômenos da Natureza"), datatran_mg['elemento_
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Objeto estático sobre o leito carroçável"), dat
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Pista Escorregadia"), datatran_mg['elemento_tra
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Restrição de Visibilidade"), datatran_mg['eleme
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Sinalização da via insuficiente ou inadequada")
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Ausência de sinalização"), datatran_mg['element
datatran_mg['elemento_transito'] = np.where((datatran_mg['causa_acidente'] == "Chuva"), datatran_mg['elemento_transito'].repla
```

Análise e exploração dos dados

- Análise e exploração com o Python.

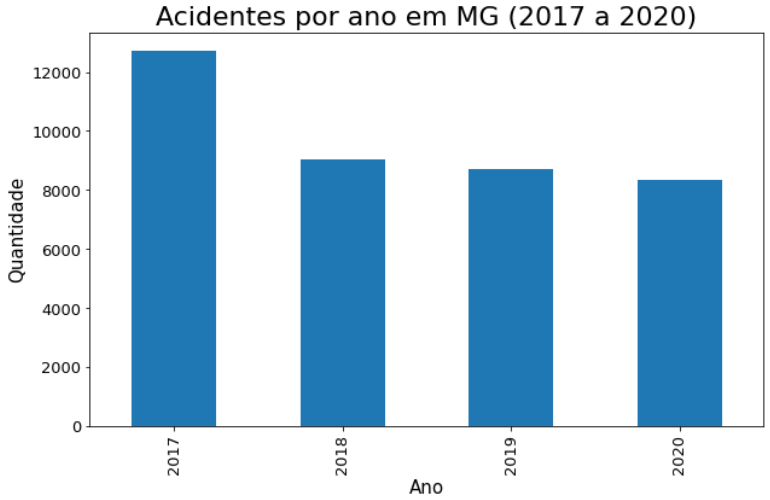
```
Quantidade_acidentes = datatran_mg.shape[0]
print("Acidentes em MG (2017 a 2020): " + str(Quantidade_acidentes))
```

```
Acidentes em MG (2017 a 2020): 38835
```

```
datatran_mg['data_inversa'] = datatran_mg.data_inversa.astype('datetime64')
Acidentes_por_ano = datatran_mg.groupby(datatran_mg['data_inversa'].dt.strfti
print("Acidentes por ano em MG (2017 a 2020): " + str(Acidentes_por_ano))
```

```
Acidentes por ano em MG (2017 a 2020): data_inversa
2017    12711
2018     9055
2019     8713
2020     8356
Name: id, dtype: int64
```

```
1 ax = Acidentes_por_ano.plot(kind='bar', figsize=(10,6), fontsize=13)
2 ax.set_alpha(0.8)
3 ax.set_title("Acidentes por ano em MG (2017 a 2020)", fontsize=22)
4 ax.set_ylabel("Quantidade", fontsize=15)
5 ax.set_xlabel("Ano", fontsize=15)
6 plt.show()
```



Análise e exploração dos dados

- Análise e exploração com o Python.

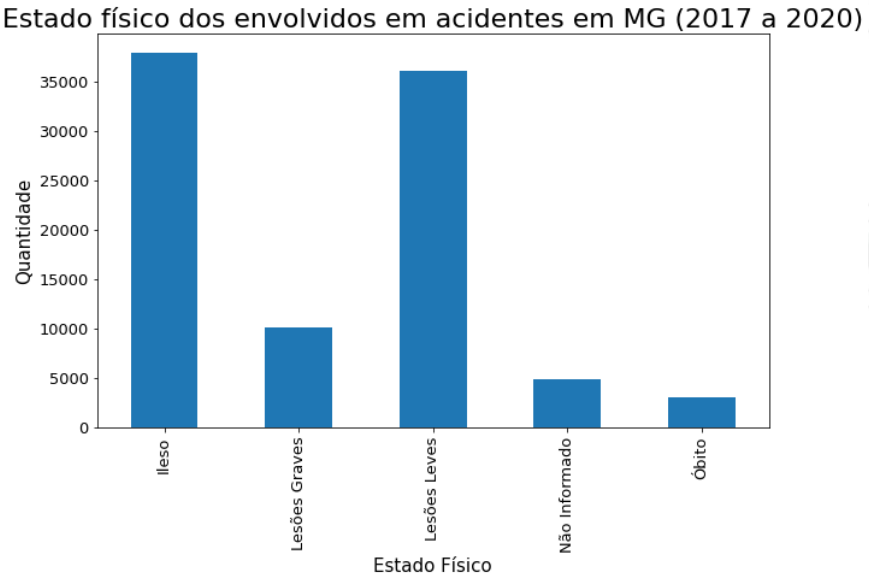
```
Estado_fisico = acidentes_mg.groupby(['estado_fisico']).size()
print("Estado físico dos envolvidos em MG (2017 a 2020): " + str(Estado_fisico))
```

```
Estado físico dos envolvidos em MG (2017 a 2020): estado_fisico
Ileso            37895
Lesões Graves    10022
Lesões Leves     36078
Não Informado     4837
Óbito             2957
dtype: int64
```

```
Quantidade_envolvidos = acidentes_mg.shape[0]
perc_ileso = (acidentes_mg['id'][acidentes_mg['estado_fisico'] == 'Ileso'].count() / Quantidade_er
perc_lesgr = (acidentes_mg['id'][acidentes_mg['estado_fisico'] == 'Lesões Graves'].count() / Quant
perc_leslv = (acidentes_mg['id'][acidentes_mg['estado_fisico'] == 'Lesões Leves'].count() / Quanti
perc_nainf = (acidentes_mg['id'][acidentes_mg['estado_fisico'] == 'Não Informado'].count() / Quant
perc_obito = (acidentes_mg['id'][acidentes_mg['estado_fisico'] == 'Óbito'].count() / Quantidade_er
#print(f"% Estado físico dos envolvidos em MG (2017 a 2020): \n Ileso- {perc_ileso},\n Lesões Grav
print("% Estado físico dos envolvidos em MG (2017 a 2020): \n Ileso " + "%.2f" % perc_ileso + "\n
```

```
% Estado físico dos envolvidos em MG (2017 a 2020):
 Ileso 41.28
 Lesões Graves 10.92
 Lesões Leves 39.31
 Não Informado 5.27
 Óbito 3.22
```

```
1 ax = Estado_fisico.plot(kind='bar', figsize=(10,6), fontsize=13);
2 ax.set_alpha(0.8)
3 ax.set_title("Estado físico dos envolvidos em acidentes em MG (2017 a 2020)", fontsize=22)
4 ax.set_ylabel("Quantidade", fontsize=15);
5 ax.set_xlabel("Estado Físico", fontsize=15);
6 plt.show()
```



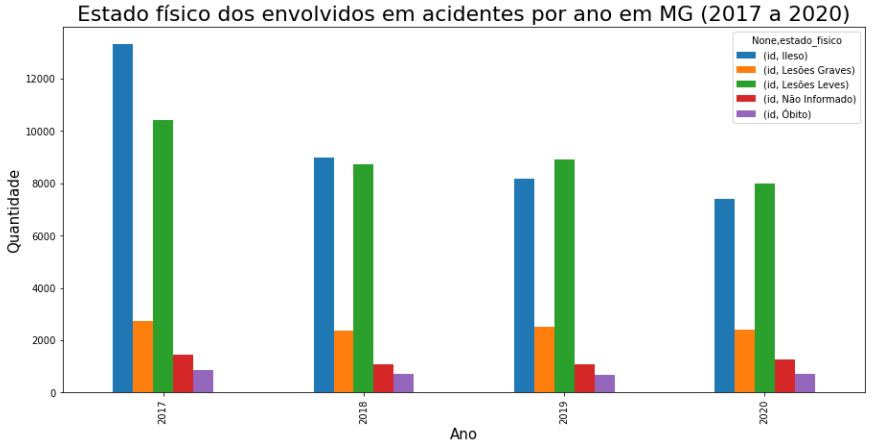
PUC Minas Virtual

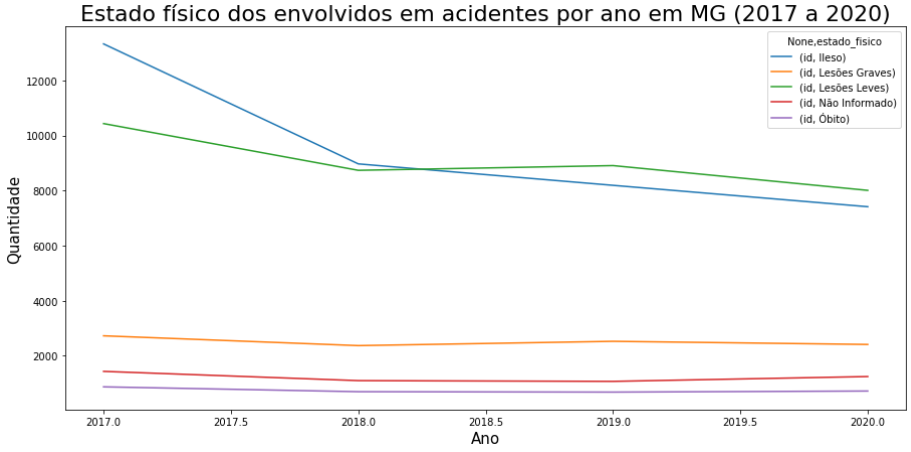
Análise e exploração dos dados

- Análise e exploração com o Python.

Estado físico dos envolvidos por ano em MG (2017 a 2020):

Ano	estado_fisico	
2017	Ileso	13326
	Lesões Graves	2723
	Lesões Leves	10432
	Não Informado	1431
	Óbito	869
2018	Ileso	8968
	Lesões Graves	2367
	Lesões Leves	8734
	Não Informado	1095
	Óbito	693
2019	Ileso	8189
	Lesões Graves	2523
	Lesões Leves	8906
	Não Informado	1068
	Óbito	678
2020	Ileso	7412
	Lesões Graves	2409
	Lesões Leves	8006
	Não Informado	1243
	Óbito	717





Análise e exploração dos dados

- Geração de índice de acidentalidade com o Python.

```
#Geração de indicadores relacionados aos acidentes de MG

#Classificação por gravidade dos acidentes
#print(str(datatran_mg.groupby(['br','classificacao_acidente']).size()))
datatran_mg['data_inversa'] = datatran_mg.data_inversa.astype('datetime64')
class_acid = pd.DataFrame(datatran_mg.groupby([datatran_mg['data_inversa'].dt.year.rename('Ano'), datatran_mg['br'], d
class_acid.rename(columns={'br':'BR'}, inplace=True)
class_acid.rename(columns={'id':'Acidentes'}, inplace=True)
class_acid['Sem_Vitimas'] = '0'
class_acid['Com_Vitimas_Feridas'] = '0'
class_acid['Com_Vitimas_Fatais'] = '0'
class_acid['BR'] = class_acid.BR.astype('object')
class_acid['Sem_Vitimas'] = class_acid.Sem_Vitimas.astype('int64')
class_acid['Com_Vitimas_Feridas'] = class_acid.Com_Vitimas_Feridas.astype('int64')
class_acid['Com_Vitimas_Fatais'] = class_acid.Com_Vitimas_Fatais.astype('int64')
class_acid.loc[class_acid['classificacao_acidente'] == 'Com Vitimas Fatais', 'Com_Vitimas_Fatais'] = class_acid['Acide
class_acid.loc[class_acid['classificacao_acidente'] == 'Com Vitimas Feridas', 'Com_Vitimas_Feridas'] = class_acid['Aci
class_acid.loc[class_acid['classificacao_acidente'] == 'Sem Vitimas', 'Sem_Vitimas'] = class_acid['Acidentes']
class_acid.drop(columns=['classificacao_acidente'], axis=1, inplace=True)
class_acid.drop(columns=['Acidentes'], axis=1, inplace=True)
class_acidentes = pd.DataFrame(class_acid.groupby([class_acid['Ano'], class_acid['BR']]).agg({'Sem_Vitimas' : 'sum', '
#class_acidentes.head(10)

#Extensões por trecho de BR
snv_2017_mg = pd.read_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\SNV_201801B_3.csv", sep=';', decimal=',', encoding =
snv_2018_mg = pd.read_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\snv_201903a_3.csv", sep=';', decimal=',', encoding =
snv_2019_mg = pd.read_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\SNV_202001A_3.csv", sep=';', decimal=',', encoding =
snv_2020_mg = pd.read_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\SNV_202101A_3.csv", sep=';', decimal=',', encoding = 'utf_8')
snv_2017_mg.drop(columns=['Tipo de trecho','Desc Coinc','Código','Local de Início','Local de Fim','km inicial','km final'], ax
snv_2018_mg.drop(columns=['Tipo de trecho','Desc Coinc','Código','Local de Início','Local de Fim','km inicial','km final'], ax
snv_2019_mg.drop(columns=['Tipo de trecho','Desc Coinc','Código','Local de Início','Local de Fim','km inicial','km final','Sug
snv_2020_mg.drop(columns=['Tipo de trecho','Desc Coinc','Código','Local de Início','Local de Fim','km inicial','km final','Sug

#Extensões por BR
snv_2017_mg_br = pd.DataFrame(snv_2017_mg.groupby('BR', as_index = False)['Extensão'].sum()).reset_index()
snv_2018_mg_br = pd.DataFrame(snv_2018_mg.groupby('BR', as_index = False)['Extensão'].sum()).reset_index()
snv_2019_mg_br = pd.DataFrame(snv_2019_mg.groupby('BR', as_index = False)['Extensão'].sum()).reset_index()
snv_2020_mg_br = pd.DataFrame(snv_2020_mg.groupby('BR', as_index = False)['Extensão'].sum()).reset_index()
snv_2017_mg_br.rename(columns={'index':'Ano'}, inplace=True)
```

```
#Gerando os indicadores - dados de 2019
temp_2019 = temp_indices.loc[(temp_indices['Ano'] == "2019")]
#print(temp_2019.loc[temp_2019['Ano'] == "2019", 'Extensão'].sum())
temp_2019['%_Extensão_BR'] = (temp_2019['Extensão'] / temp_2019['Extensão'].sum())*100
temp_2019['%_Acidente_BR'] = (temp_2019['Acidentes'] / temp_2019['Acidentes'].sum())*100
temp_2019['%_Feridos_BR'] = (temp_2019['Feridos'] / temp_2019['Feridos'].sum())*100
temp_2019['%_Mortes_BR'] = (temp_2019['Mortes'] / temp_2019['Mortes'].sum())*100
temp_2019['Acidente_KM'] = (temp_2019['Acidentes'] / temp_2019['Extensão'])
temp_2019['Mortes_KM'] = (temp_2019['Mortes'] / temp_2019['Extensão'])
temp_2019 = temp_2019.reindex(columns=['Ano', 'BR', 'Extensão', 'Acidentes', 'Envolvidos', 'Feridos', 'Mortes', '%_Extensão_BR
temp_2019.drop(columns=['index'], axis=1, inplace=True)
#print(temp_2019)
#temp_2019.head(50)

#Gerando os indicadores - dados de 2020
temp_2020 = temp_indices.loc[(temp_indices['Ano'] == "2020")]
#print(temp_2019.loc[temp_2020['Ano'] == "2020", 'Extensão'].sum())
temp_2020['%_Extensão_BR'] = (temp_2020['Extensão'] / temp_2020['Extensão'].sum())*100
temp_2020['%_Acidente_BR'] = (temp_2020['Acidentes'] / temp_2020['Acidentes'].sum())*100
temp_2020['%_Feridos_BR'] = (temp_2020['Feridos'] / temp_2020['Feridos'].sum())*100
temp_2020['%_Mortes_BR'] = (temp_2020['Mortes'] / temp_2020['Mortes'].sum())*100
temp_2020['Acidente_KM'] = (temp_2020['Acidentes'] / temp_2020['Extensão'])
temp_2020['Mortes_KM'] = (temp_2020['Mortes'] / temp_2020['Extensão'])
temp_2020 = temp_2020.reindex(columns=['Ano', 'BR', 'Extensão', 'Acidentes', 'Envolvidos', 'Feridos', 'Mortes', '%_Extensão_BR
temp_2020.drop(columns=['index'], axis=1, inplace=True)
#print(temp_2020)
#temp_2020.head(50)

#Consolidando os indicadores de acidentalidade em um único dataframe
indices_temp = pd.concat([temp_2017, temp_2018, temp_2019, temp_2020], sort = False).reset_index()
indices_temp.drop(columns=['index'], axis=1, inplace=True)
class_acidentes.drop(columns=['Ano'], axis=1, inplace=True)
class_acidentes.drop(columns=['BR'], axis=1, inplace=True)
indices_acidentes = pd.concat([indices_temp, class_acidentes], axis=1, sort = False).reset_index()
indices_acidentes.drop(columns=['index'], axis=1, inplace=True)
indices_acidentes['Gravidade'] = (indices_acidentes['Sem_Vitimas'] + (5 * indices_acidentes['Com_Vitimas_Feridas'])+(13 * indi
indices_acidentes.to_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\indices_acidentes.csv", index = False, header = True, sep=';',
```

PUC Minas Virtual

Análise e exploração dos dados

- Geração de índice de acidentalidade com o Python.

Foi gerado dataframe com os índices dos acidentes e armazenado em um arquivo csv.

	Ano	BR	Extensão	Acidentes	Envolvidos	Feridos	Mortes	%_Extensão_BR	%_Acidente_BR	Acidente_KM	%_Feridos_BR	%_Mortes_BR	Mortes_KM
0	2017	40	893.90	2374	5479	2494	145	8.7231	18.6767	2.6558	18.9586	16.6858	0.1622
1	2017	50	269.80	673	1333	619	13	2.6328	5.2946	2.4944	4.7054	1.4960	0.0482
2	2017	116	818.10	1626	3749	1841	170	7.9834	12.7921	1.9875	13.9947	19.5627	0.2078
3	2017	135	834.70	208	553	260	34	8.1454	1.6364	0.2492	1.9764	3.9125	0.0407
4	2017	146	726.80	114	256	128	6	7.0925	0.8969	0.1569	0.9730	0.6904	0.0083
5	2017	153	239.90	379	855	362	29	2.3411	2.9817	1.5798	2.7518	3.3372	0.1209
6	2017	251	1015.70	282	736	340	50	9.9117	2.2186	0.2776	2.5846	5.7537	0.0492
7	2017	262	1082.90	1381	3012	1495	97	10.5675	10.8646	1.2753	11.3645	11.1623	0.0896
8	2017	267	534.70	293	759	397	30	5.2179	2.3051	0.5480	3.0179	3.4522	0.0561
9	2017	354	774.50	106	241	90	4	7.5579	0.8339	0.1369	0.6842	0.4603	0.0052
10	2017	356	291.90	108	230	121	2	2.8485	0.8497	0.3700	0.9198	0.2301	0.0069

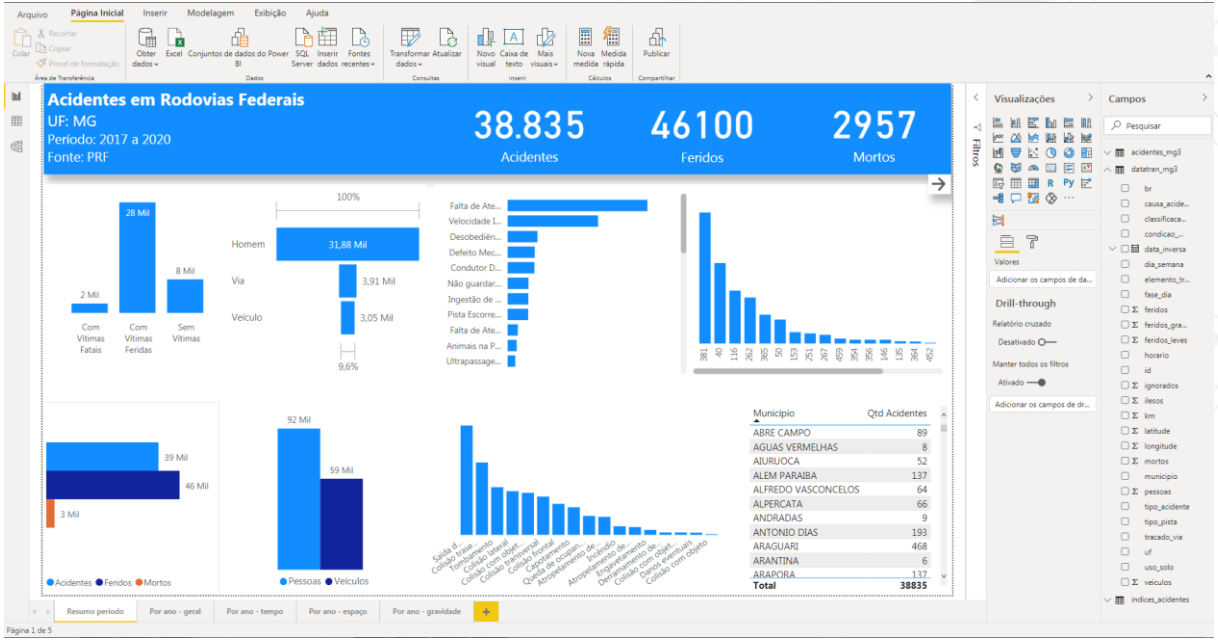
Sem_Vitimas	Com_Vitimas_Feridas	Com_Vitimas_Fatais	Gravidade
720	1544	110	11.0415
236	426	11	9.2995
385	1111	130	9.3265
58	125	25	1.2076
36	72	6	0.6522
144	210	25	6.3318
87	162	33	1.3055
399	902	80	5.4936
63	205	25	2.6426
35	67	4	0.5449
21	85	2	1.6170

São 66 registros de dados, sendo um para cada BR e por ano.
A BR-381, em todos os anos, apresentou o mais alto índice de gravidade dos acidentes.
Depois, estão as BRs “040, 050 e 116”, com os mais altos índices de gravidade dos acidentes.
Em termos de periculosidade (acidente/km), o ranking também é liderado pelas BRs “381, 040 e 050”, com destaque para a 381.

Análise e exploração dos dados

- Análise e exploração com o PowerBI.

Foram desenvolvidos 05 (cinco) dashboards, sendo 01 como um resumo geral do período de 2017 a 2020 e os outros 04 com filtros para seleção por ano e com temas específicos.



Criação do modelo de Machine Learning

- Classificação do estado físico dos envolvidos nos acidentes.

```
1 acidentes_mg['estado_fisico'].value_counts()
```

```
Ileso            37895
Lesões Leves     36078
Lesões Graves    10022
Não Informado     4837
Óbito             2957
Name: estado_fisico, dtype: int64
```

O atributo ‘estado_fisico’, presente nos registros de ocorrência agrupados por pessoas, trata-se da condição do envolvido conforme a gravidade das lesões e apresenta os valores de “Ileso”, “Lesões Leves”, “Lesões Graves”, “Não Informado” e “Óbito”.

A intenção é prever valores de “Ileso”, “Lesões Leves”, “Lesões Graves” e “Óbito”, de maneira que o valor de “Não Informado” não pode fazer parte dos valores que o modelo irá aprender, devendo o mesmo ser removido da amostra em que serão aplicados os algoritmos de machine learning.

Criação do modelo de Machine Learning

- Preparação da amostra para treinamento e teste.

Variáveis preditoras e variável alvo

```
#Tratamento dos registros que o estado_fisico está como Não Informado
f = fset.loc[fset['estado_fisico'] == 'Não Informado'].index
fset.drop(f, axis= 0, inplace = True)
```

```
#Alocação de variáveis preditoras
X = fset[['dia_semana','br','km','causa_acidente','tipo_acidente','classificacao_acidente','fase_dia', 'condicao_metereologi
#Alocação da variável target
Y = fset['estado_fisico'].values
```

Transformação dos dados não numéricos

```
#Importação do módulo preprocessing
from sklearn import preprocessing
```

```
#Transformação de dados não numéricos em dados numéricos - utilização
LE_Dia_Semana = preprocessing.LabelEncoder()
LE_Dia_Semana.fit(fset.dia_semana.unique())
X[:,0] = LE_Dia_Semana.transform(X[:,0])
```

```
LE_Causa = preprocessing.LabelEncoder()
LE_Causa.fit(fset.causa_acidente.unique())
X[:,3] = LE_Causa.transform(X[:,3])
```

```
LE_Tipo_Acidente = preprocessing.LabelEncoder()
LE_Tipo_Acidente.fit(fset.tipo_acidente.unique())
X[:,4] = LE_Tipo_Acidente.transform(X[:,4])
```

Normalização dos dados

```
#Normalização dos valores para uma melhor performance do algoritmo de classificação
#Foram efetuados testes com StandardScaler, MinMaxScaler e RobustScaler, mas optou-se pelo StandardScaler
```

```
#Normalização dos valores com StandardScaler
std = preprocessing.StandardScaler()
std.fit(X)
Xstd = std.transform(X)
print ('Média antes da normalização: {:.2f}'.format(X.mean()),'\nDesvio padrão antes da normalização: {:.
print('')
print ('Média depois da normalização: {:.2f}'.format(Xstd.mean()),'\nDesvio padrão depois da normalização
```

```
Média antes da normalização: 49.48
Desvio padrão antes da normalização: 143.08

Média depois da normalização: -0.00
Desvio padrão depois da normalização: 1.00
```

Criação do modelo de Machine Learning

- Divisão da amostra e verificação do 1º algoritmo – SGDClassifier.

Criação do conjunto de treinamento e teste

```
#Depois de normalizado, o array Xstd, que contém os valores para a predição, foi dividido em 4 partes:

#X_trainset: Uma de treino do modelo, onde o algoritmo irá 'apreender' a relação entre as variáveis preditoras
#Y_trainset: Uma parte também de treino do modelo, mas somente com as variáveis target, ou os valores que queremos prever
#X_testset: Uma parte para teste, onde o modelo irá exercer o aprendizado obtido com os arrays X_trainset e Y_trainset, ou i
#Y_testset: Uma parte para avaliar a performance do modelo, Y_testset contém as saídas corretas que desejamos prever,e é usa
```

```
#Divisão do dataset seguindo a proporção 70/30 (70% para treino e 30% para teste)
X_trainset, X_testset, Y_trainset, Y_testset = train_test_split(Xstd, Y, test_size=0.3, random_state=3)
```

Teste entre SGDClassifier e Linear SVC

```
em SGD com o SGDClassifier
import time
start = time.time()
clf = SGDClassifier(loss="hinge", penalty="l2")
clf.fit(X_trainset,Y_trainset)
stop = time.time()
print(f"Tempo de treinamento para linear SVM com SGD: {stop - start}s")

start = time.time()
clf = SVC(kernel='linear')
clf.fit(X_trainset,Y_trainset)
stop = time.time()
print(f"Tempo de treinamento para linear SVM sem SGD: {stop - start}s")
#SGD classifier executa bem mais rápido do que o Linear SVM
```

click to scroll output; double click to hide

```
Tempo de treinamento para linear SVM com SGD: 3.3240966796875s
Tempo de treinamento para linear SVM sem SGD: 953.1051347255707s
```

Teste com Regressão Logística e Linear SVM

```
#Regressão Logística (loss='log') com treinamento SGD do modelo
clf = SGDClassifier(loss="log", penalty="l2")
clf.fit(X_trainset,Y_trainset)
```

```
SGDClassifier(loss='log')
```

```
Y_Pred = clf.predict(X_testset)
print('Acurácia: {:.2f}'.format(accuracy_score(Y_testset, Y_Pred)))
```

```
Acurácia: 0.58
```

```
#Linear SVM (loss='hinge') com treinamento SGD do modelo
clf = SGDClassifier(loss="hinge", penalty="l2")
clf.fit(X_trainset,Y_trainset)
```

```
SGDClassifier()
```

```
Y_Pred = clf.predict(X_testset)
print('Acurácia: {:.2f}'.format(accuracy_score(Y_testset, Y_Pred)))
```

```
Acurácia: 0.59
```

Criação do modelo de Machine Learning

- Identificando os parâmetros mais adequados para o SGDClassifier.

Regressão logística com os parâmetros mais adequados

```
#Teste com o algoritmo SGDClassifier e os hyper parâmetros mais adequados
clf = SGDClassifier(loss="log", penalty="l1", alpha=0.01, max_iter=50)
clf.fit(X_trainset,Y_trainset)
```

```
SGDClassifier(alpha=0.01, loss='log', max_iter=50, penalty='l1')
```

```
Y_Pred = clf.predict(X_testset)
print('Acurácia: {:.2f}'.format(accuracy_score(Y_testset, Y_Pred)))
```

```
Acurácia: 0.60
```

Armazenamento predições

```
#Armazenando as predições com dados de treino e dados de teste
pTrain = clf.predict(X_trainset)
pTest = clf.predict(X_testset)
```

Encontrando parâmetros

```
#Encontrando os hyper parâmetros mais adequados por meio do método GridSearch
from sklearn.model_selection import GridSearchCV
params = {
    "loss" : ["hinge", "log", "squared_hinge", "modified_huber", "perceptron"],
    "alpha" : [0.0001, 0.001, 0.01, 0.1],
    "penalty" : ["l2", "l1", "elasticnet", "none"],
}

clf = SGDClassifier(max_iter=50)
grid = GridSearchCV(clf, param_grid=params, cv=10)
```

```
grid.fit(X_trainset,Y_trainset)
```

```
print(grid.best_params_)
```

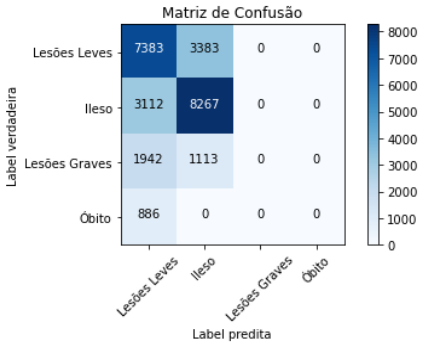
```
{'alpha': 0.01, 'loss': 'log', 'penalty': 'l1'}
```

```
grid_predictions = grid.predict(X_testset)

print('Acurácia: {:.2f}'.format(accuracy_score(Y_testset, grid_predictions)))
```

```
Acurácia: 0.60
```

Matriz de Confusão

```
              accuracy                           0.60     26086
             macro avg       0.30      0.35      0.32     26086
          weighted avg       0.51      0.60      0.55     26086

Matriz de confusão sem normalização
[[7383 3383    0    0]
 [3112 8267    0    0]
 [1942 1113    0    0]
 [ 886    0    0    0]]
```



Criação do modelo de Machine Learning

- Verificação do 2º algoritmo – DecisionTree.

Armazenamento predições

```
#Teste com o algoritmo Decision Tree
dtclf = DecisionTreeClassifier()
dtclf.fit(X_trainset,Y_trainset)
```

```
DecisionTreeClassifier()
```

```
#Armazenamento de previões em duas variáveis: uma para os dados de treino e outra para os dados de teste
Y_PredTE = dtclf.predict(X_testset)
Y_PredTR = dtclf.predict(X_trainset)
```

Encontrando parâmetros

```
#Encontrando os hyper parâmetros mais adequados por meio do método GridSearch
param_grid = {"criterion": ["gini", "entropy"],
              "min_samples_split": [2, 5, 10, 15, 20],
              "max_depth": [None, 2, 3, 5, 7, 10],
              "min_samples_leaf": [1, 3, 5, 7, 10],
              "max_leaf_nodes": [None, 3, 5, 7, 10, 15, 20],
              }

grid = GridSearchCV(dtclf, param_grid, cv=10, scoring='accuracy')
grid.fit(X_trainset, Y_trainset)
```

```
GridSearchCV(cv=10,
             estimator=DecisionTreeClassifier(criterion='entropy',
                                              min_samples_split=4),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 2, 3, 5, 7, 10],
                         'max_leaf_nodes': [None, 3, 5, 7, 10, 15, 20],
                         'min_samples_leaf': [1, 3, 5, 7, 10],
                         'min_samples_split': [2, 5, 10, 15, 20]},
             scoring='accuracy')
```

```
#O melhor score obtido entre todos os parâmetros
print(grid.best_score_)
```

```
0.7130583616107357
```

```
#Relação dos hyper parâmetros usados para gerar o melhor score
print(grid.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 10, 'max_leaf_nodes': None, 'min_samples_leaf': 10, 'min_samples_split': 15}
```

Decision Tree com os parâmetros mais adequados

```
#Teste com o algoritmo DecisionTree e os hyper parâmetros mais adeq
dtclf = DecisionTreeClassifier(criterion = 'gini', max_depth=10, ma
dtclf.fit(X_trainset,Y_trainset)
Y_PredTE = dtclf.predict(X_testset)
Y_PredTR = dtclf.predict(X_trainset)
print('Acurácia p/ dados de Teste: {:.2f}'.format(accuracy_score(Y_
print('Acurácia p/ dados de Treino: {:.2f}'.format(accuracy_score(Y_
print('Erros de classificação das amostras Teste: {}'.format((Y_tes
print('Erros de classificação das amostras Treino: {}'.format((Y_tr
print('Indice Jaccard p/ dados de Teste: {:.3f}%'.format(m.jaccard_
print('Indice Jaccard p/ dados de Treino: {:.3f}%'.format(m.jaccard_
```

```
Acurácia p/ dados de Teste: 0.71
Acurácia p/ dados de Treino: 0.72
Erros de classificação das amostras Teste: 7536
Amostras de teste "y": 26086
Erros de classificação das amostras Treino: 16989
Amostras de treino "y": 60866
Indice Jaccard p/ dados de Teste: 0.552%
Indice Jaccard p/ dados de Treino: 0.564%
```

Criação do modelo de Machine Learning

- Verificação do 2º algoritmo – DecisionTree.

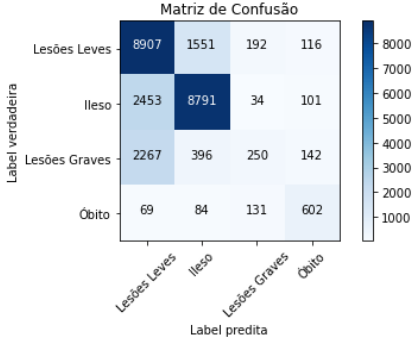
Matriz de Confusão

```
#Cálculo da matriz de confusão
cnf_matrix_dt = m.confusion_matrix(Y_testset, Y_PredTE, labels = fs
np.set_printoptions(precision=2)
```

```
print (m.classification_report(Y_testset, Y_PredTE))

# Gráfico da matriz de confusão sem normalização
plt.figure()
plot_confusion_matrix(cnf_matrix_dt, classes= fset.estado_fisico.ur
```

	precision	recall	f1-score	support
Ileso	0.81	0.77	0.79	11379
Lesões Graves	0.41	0.08	0.14	3055
Lesões Leves	0.65	0.83	0.73	10766
Óbito	0.63	0.68	0.65	886
accuracy			0.71	26086
macro avg	0.63	0.59	0.58	26086
weighted avg	0.69	0.71	0.68	26086

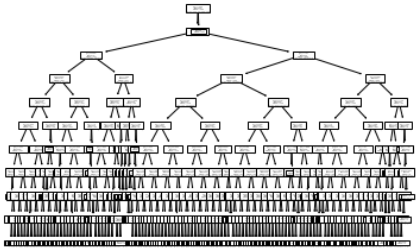
```
Matriz de confusão sem normalização
[[8907 1551  192  116]
 [2453 8791   34  101]
 [2267  396  250  142]
 [  69   84  131  602]]
```



Árvore gerada

```
from sklearn import tree
tree.plot_tree(dtclf)
```

```
Text(332.02732919254663, 69.18545454545455, 'gini = 0.556\nsamples = 13\nvalue = [7, 1, 5, 0]'),
Text(333.4136645962733, 69.18545454545455, 'X[13] <= -0.607\ngini = 0.5\nsamples = 26\nvalue = [0, 13, 13, 0]'),
Text(332.72049689440996, 49.418181818181836, 'gini = 0.473\nsamples = 13\nvalue = [0, 8, 5, 0]'),
Text(334.1068322981367, 49.418181818181836, 'gini = 0.473\nsamples = 13\nvalue = [0, 5, 8, 0]'),
Text(158.97909549689442, 187.7890909090909, 'gini = 0.0\nsamples = 9158\nvalue = [9158, 0, 0, 0]')]
```



Criação do modelo de Machine Learning

- Verificação do 3º algoritmo – RandomForest.

Armazenamento predições

```
#Teste com o algoritmo Random Forest Classifier
rfclf = RandomForestClassifier()
rfclf.fit(X_trainset,Y_trainset)
```

```
RandomForestClassifier()
```

```
#Armazenamento de previões em duas variáveis: uma para os dados de treino e outra para os dados de teste
Y_PredTE2 = fclf.predict(X_testset)
Y_PredTR2 = fclf.predict(X_trainset)
```

Encontrando parâmetros

```
#Encontrando os hyper parâmetros mais adequados por meio do método GridSearch
param_dist = {"n_estimators": list(range(10,210,10)),
              "max_depth": list(range(3,20)),
              "max_features": list(range(1, 10)),
              "min_samples_split": list(range(2, 11)),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}
```

```
n_iter_search = 50

random_search = RandomizedSearchCV(rfclf, param_distributions=param_dist, scoring='accuracy',
                                   n_iter=n_iter_search)
random_search.fit(X_trainset,Y_trainset)
```

```
RandomizedSearchCV(estimator=RandomForestClassifier(), n_iter=50,
                   param_distributions={'bootstrap': [True, False],
                                        'criterion': ['gini', 'entropy'],
                                        'max_depth': [3, 4, 5, 6, 7, 8, 9, 10,
                                                      11, 12, 13, 14, 15, 16,
                                                      17, 18, 19],
                                        'max_features': [1, 2, 3, 4, 5, 6, 7, 8,
                                                         9],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
                                        'n_estimators': [10, 20, 30, 40, 50, 60,
                                                         70, 80, 90, 100, 110,
                                                         120, 130, 140, 150,
                                                         160, 170, 180, 190,
                                                         200]},
                   scoring='accuracy')
```

```
#Identificando os melhores hiper parâmetros
print('Melhor número de estimadores:', random_search.best_estimator_.get_params()['n_estimators'])
print('Melhor min_samples_split:', random_search.best_estimator_.get_params()['max_depth'])
```

```
Melhor número de estimadores: 140
Melhor min_samples_split: 14
```

Decision Tree com os parâmetros mais adequados

```
#Teste com o algoritmo RandomForest e os hyper parâmetros mais adequados
rfclf = RandomForestClassifier(n_estimators=140, min_samples_split=9, max_features
rfclf.fit(X_trainset,Y_trainset)
Y_PredTE2 = rfclf.predict(X_testset)
Y_PredTR2 = rfclf.predict(X_trainset)
print('Acurácia p/ dados de Teste: {:.2f}'.format(accuracy_score(Y_testset, Y_Prec
print('Acurácia p/ dados de Treino: {:.2f}'.format(accuracy_score(Y_trainset, Y_Pr
print('Erros de classificação das amostras Teste: {}'.format((Y_testset != Y_PredT
print('Erros de classificação das amostras Treino: {}'.format((Y_trainset != Y_Pre
print('Indice Jaccard p/ dados de Teste: {:.3f}%'.format(m.jaccard_score(Y_testset
print('Indice Jaccard p/ dados de Treino: {:.3f}%'.format(m.jaccard_score(Y_trains
```

```
Acurácia p/ dados de Teste: 0.72
Acurácia p/ dados de Treino: 0.75
Erros de classificação das amostras Teste: 7283
Amostras de teste "y": 26086
Erros de classificação das amostras Treino: 15029
Amostras de treino "y": 60866
Indice Jaccard p/ dados de Teste: 0.563%
Indice Jaccard p/ dados de Treino: 0.604%
```

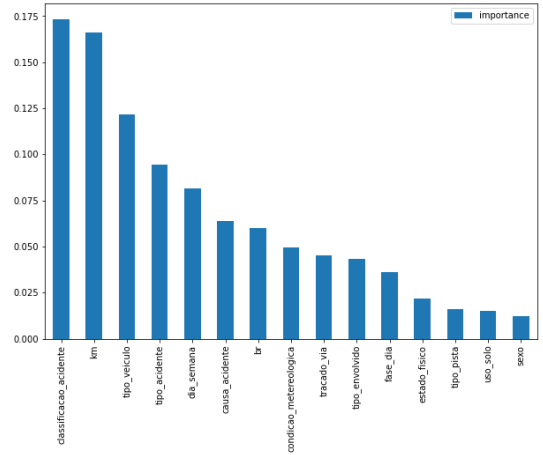
Criação do modelo de Machine Learning

- Verificação do 3º algoritmo – RandomForest.

Feature of importance

```
#Determinação de atributo (feature) de importância
feat_imps = pd.DataFrame({'importance': fclf.feature_importances_}, index=fset.columns[:-1])
feat_imps.sort_values(by='importance', ascending=False, inplace=True)
```

```
feat_imps.plot(kind='bar', figsize=(10,7))

plt.legend()
plt.show()
```



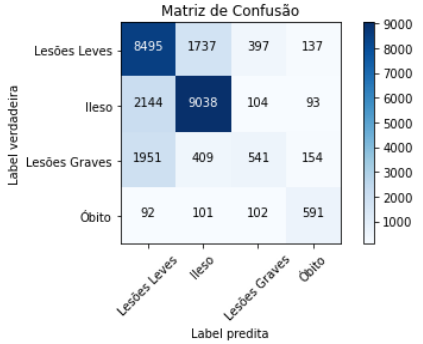
Matriz de Confusão

```
#Cálculo da matriz de confusão
cnf_matrix_rf = m.confusion_matrix(Y_testset, Y_PredTE2, labels = fset.est
np.set_printoptions(precision=2)
```

```
print (m.classification_report(Y_testset, Y_PredTE2))

# Gráfico da matriz de confusão sem normalização
plt.figure()
plot_confusion_matrix(cnf_matrix_rf, classes= fset.estado_fisico.unique(),
```

	precision	recall	f1-score	support
Ileso	0.80	0.79	0.80	11379
Lesões Graves	0.47	0.18	0.26	3055
Lesões Leves	0.67	0.79	0.72	10766
Óbito	0.61	0.67	0.64	886
accuracy			0.72	26086
macro avg	0.64	0.61	0.60	26086
weighted avg	0.70	0.72	0.70	26086

```
Matriz de confusão sem normalização
[[8495 1737  397  137]
 [2144 9038  104   93]
 [1951  409  541  154]
 [  92  101  102  591]]
```



Criação do modelo de Machine Learning

- De todos os algoritmos testados (SGDclassifier – log e hinge; DecisionTreeClassifier; e RandomForestClassifier), o **Random Forest** foi o que entregou o score de acurácia mais elevado, embora se desejasse um valor superior a **0.72**, foi o melhor que se obteve nos presentes testes.

	precision	recall	f1-score	support
Ileso	0.80	0.79	0.80	11379
Lesões Graves	0.47	0.18	0.26	3055
Lesões Leves	0.67	0.79	0.72	10766
Óbito	0.61	0.67	0.64	886
accuracy			0.72	26086
macro avg	0.64	0.61	0.60	26086
weighted avg	0.70	0.72	0.70	26086

Considerando os resultados obtidos, passou-se à implementação do modelo escolhido, com o Random Forest, para efetivar os resultados do algoritmo num conjunto de dados para estudo

```
#Montagem de um novo dataset para implementar o modelo escolhido de ML, o do RandomForest
acidentes_mg.reset_index(inplace = True)
fset2 = acidentes_mg[['dia_semana','br','km','causa_acidente','tipo_acidente','classificaca
fset2.head()
```

	dia_semana	br	km	causa_acidente	tipo_acidente	classificacao_acidente	fase_dia	condicao_metereologi
0	domingo	381	605.0	Condutor Dormindo	Saída de leito carroçável	Com Vítimas Feridas	Amanhecer	Céu Cla
1	domingo	381	605.0	Condutor Dormindo	Saída de leito carroçável	Com Vítimas Feridas	Amanhecer	Céu Cla
2	domingo	262	368.0	Velocidade Incompatível	Colisão com objeto estático	Com Vítimas Feridas	Plena Noite	Céu Cla
3	domingo	262	368.0	Velocidade Incompatível	Colisão com objeto estático	Com Vítimas Feridas	Plena Noite	Céu Cla
4	domingo	262	368.0	Velocidade Incompatível	Colisão com objeto estático	Com Vítimas Feridas	Plena Noite	Céu Cla

Criação do modelo de Machine Learning

- Após a implementação do modelo, foram executados procedimentos ajustar os datasets de acidentes e datatran com os valores da predição.

```
#Verificando o dataset novamente modificado de acidentes
acidentes_mg['estado_fisico'].value_counts()
```

```
Ileso            40160
Lesões Leves     37775
Lesões Graves    10382
Óbito             3472
Name: estado_fisico, dtype: int64
```

```
#Confirmando os valores alterados de acidentes
print('Total de pessoas envolvidas: ', acidentes_mg['id'].count())
print('Ilesos: ',acidentes_mg.ilesos.sum())
print('Feridos Leves: ',acidentes_mg.feridos_leves.sum())
print('Feridos Graves: ',acidentes_mg.feridos_graves.sum())
print('Mortos: ',acidentes_mg.mortos.sum())
```

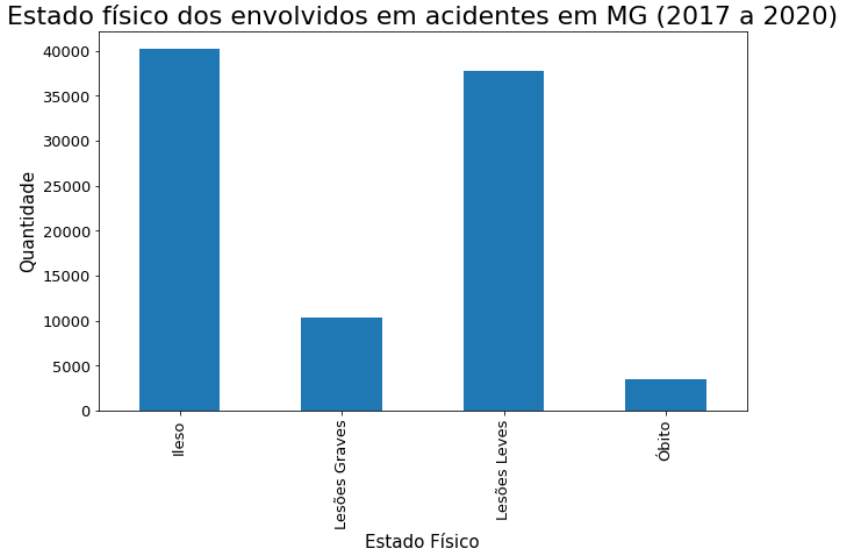
```
Total de pessoas envolvidas:  91789
Ilesos:  40160
Feridos Leves:  37775
Feridos Graves:  10382
Mortos:  3472
```

Estado físico real		Estado físico com predição	
Ileso	37895	Ileso	40160
Lesões Graves	10022	Lesões Graves	10382
Lesões Leves	36078	Lesões Leves	37775
Óbito	2957	Óbito	3472
Não Informado	4837	Não Informado	-----

Interpretação do resultado

- Esta etapa foi realizada com o Python e também com o PowerBI.

```
ax = Estado_fisico.plot(kind='bar', figsize=(10,6), fontsize=13);
ax.set_alpha(0.8)
ax.set_title("Estado físico dos envolvidos em acidentes em MG (2017 a 2020)", fontsize=22)
ax.set_ylabel("Quantidade", fontsize=15);
ax.set_xlabel("Estado Físico", fontsize=15);
plt.show()
```



Interpretação do resultado

- Esta etapa foi realizada com o Python e também com o PowerBI.

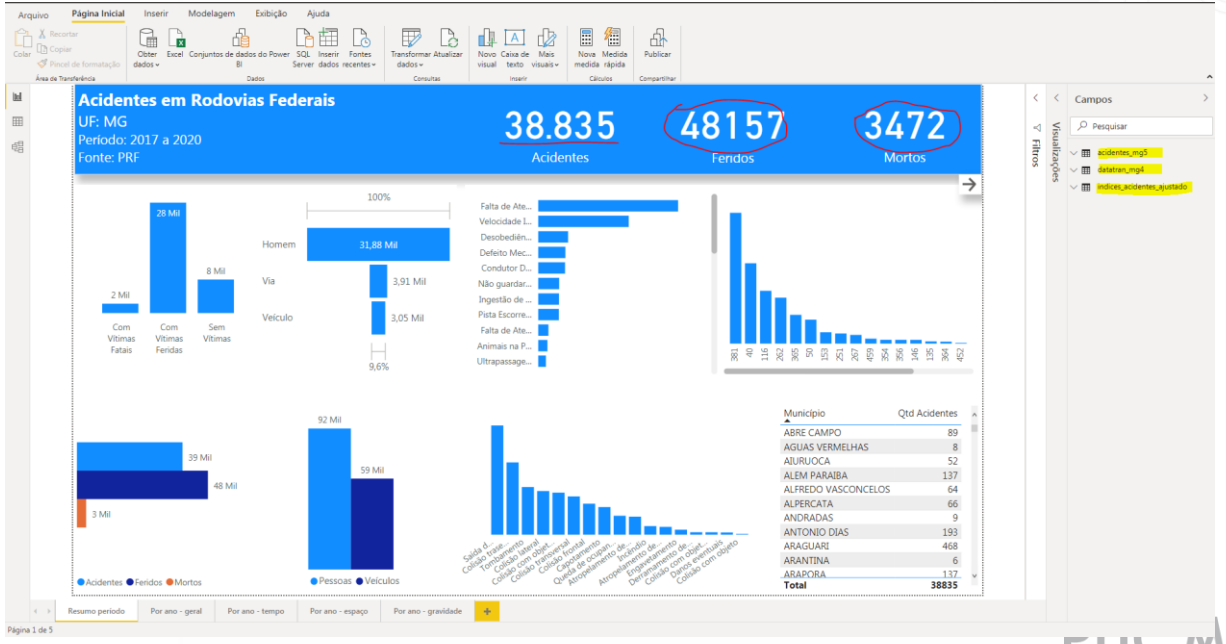
```python
#Consolidando os indicadores de acidentalidade em um único dataframe
indices_temp = pd.concat([temp_2017, temp_2018, temp_2019, temp_2020], sort = False).reset_index()
indices_temp.drop(columns=['index'], axis=1, inplace=True)
class_acidentes.drop(columns=['Ano'], axis=1, inplace=True)
class_acidentes.drop(columns=['BR'], axis=1, inplace=True)
indices_acidentes = pd.concat([indices_temp, class_acidentes], axis=1, sort = False).reset_index()
indices_acidentes.drop(columns=['index'], axis=1, inplace=True)
indices_acidentes['Gravidade'] = (indices_acidentes['Sem_Vitimas'] + (5 * indices_acidentes['Com_Vitimas_Feridas'])+(13 * ind
indices_acidentes.to_csv(r"C:\Users\lilia\Downloads\TCC PUC\csv\indices_acidentes_ajustado.csv", index = False, header = True,
```

```python
#Relação dos indicadores de acidentalidade ajustados
indices_acidentes.head(66)
#Embora as quantidades totais de classificação de acidentes não tenham sofrido mudanças por categoria (Sem Vítimas, Com Vítima
#ocorreram mudanças nos valores destas categorias por BR
#Isto poderá ser verificado comparando os arquivos de indices de acidentes
```

	Ano	BR	Extensão	Acidentes	Envolvidos	Feridos	Mortes	%_Extensão_BR	%_Acidente_BR	Acidente_KM	%_Feridos_BR	%_Mortes_BR	Mortes_KM	Se
0	2017	40	893.90	2374	5479	2600	182	8.7231	18.6767	2.6558	19.0086	17.9310	0.2036	
1	2017	50	269.80	673	1333	638	21	2.6328	5.2946	2.4944	4.6644	2.0690	0.0778	
2	2017	116	818.10	1626	3749	1935	204	7.9834	12.7921	1.9875	14.1468	20.0985	0.2494	
3	2017	135	834.70	208	553	263	39	8.1454	1.6364	0.2492	1.9228	3.8424	0.0467	
4	2017	146	726.80	114	256	137	8	7.0925	0.8969	0.1569	1.0016	0.7882	0.0110	
5	2017	153	239.90	379	855	377	31	2.3411	2.9817	1.5798	2.7563	3.0542	0.1292	
6	2017	251	1015.70	282	736	333	50	9.9117	2.2186	0.2776	2.4346	4.9261	0.0492	
7	2017	262	1082.90	1381	3012	1547	102	10.5675	10.8646	1.2753	11.3101	10.0493	0.0942	
8	2017	267	534.70	293	759	425	35	5.2179	2.3051	0.5480	3.1072	3.4483	0.0655	
9	2017	354	774.50	106	241	96	4	7.5579	0.8339	0.1369	0.7019	0.3941	0.0052	

Interpretação do resultado

- Esta etapa foi realizada com o Python e também com o PowerBI.



Apresentação dos resultados

- Foi utilizado o modelo de Canvas proposto por Dourard.

Decisions O resultado do modelo é a predição do estado físico de vítimas em acidentes de trânsito nas rodovias federais, possibilitando determinar ações de fiscalização e de educação no trânsito.	**ML task** A tarefa de machine learning é a classificação do estado físico das vítimas, identificando o estado dos que estavam como "Não Informado" na base de dados e que poderiam ser "Ilesos, Feridos Leves, Feridos Graves ou Óbito". A variável alvo é 'estado_fisico'. Dos algoritmos testados, o Random Forest, após ter sido aprimorado, foi o que apresentou mais alta acurácia.	**Value Propositions** A intenção é de ter todos os valores de estado físico de vítimas identificados podendo analisar e estudar a gravidade dos acidentes. Conhecendo a gravidade dos acidentes, o local em que ocorreram, a quantidade de ocorrências, dentre outros, pode-se identificar ações preventivas, como: fiscalização, educação, mudanças de geometria na via, manutenção de vias, sinalização de vias.	**Data Sources** A principal fonte de dados foram os arquivos com as ocorrências de acidentes registrados pela PRF.	**Collecting Data** Os novos dados poderão ser coletados no portal de dados abertos da PRF.
Making Predictions As predições são realizadas após o tratamento e o processamento dos dados e de forma muito rápida.	**Offline Evaluation** Técnicas de validação cruzada para avaliação do modelo, bem como as métricas de desvio-padrão, coeficiente de variação, valor mínimo e valor máximo dos parâmetros: 1. "accuracy" 2. "precision" 3. "recall" 4. "f1" 5. "jaccard"	-	**Features** As variáveis preditoras X são: 'dia_semana','br','km','causa_acidente','tipo_acidente','classificacao_acidente','fase_dia', 'condicao_metereologica', 'tipo_pista','tracado_via','uso_solo', 'tipo_veiculo', 'tipo_envolvido', 'sexo', 'elemento_transito'. E a variável alvo Y é: 'estado_fisico'.	**Building Models** Foram testados três algoritmos: SGDClassifier, DecisionTree Classifier e o RandomForest Classifier. O modelo foi construído com o algoritmo que teve o melhor desempenho na classificação, no caso o Random Forest.
	Live Evaluation and Monitoring A avaliação e o monitoramento do modelo foi realizada por meio da classificação do estado físico 'Não Informado', medindo a acurácia das classes (accuracy_score e jaccard_score), bem como pela medição do tempo gasto para classificação de novos registros pelo modelo.			

Pós-graduação Lato Sensu em Ciência de Dados e Big Data
Trabalho de Conclusão de Curso

CARACTERIZAÇÃO DE ACIDENTES DE TRÂNSITO EM TRECHOS DE RODOVIAS FEDERAIS E A APLICAÇÃO DE MODELOS DE MACHINE LEARNING PARA A CLASSIFICAÇÃO DO ESTADO FÍSICO DOS ENVOLVIDOS

Aluna: **Lilian Campos Soares**

Belo Horizonte
2021

PUC Minas Virtual