Relatório do Projeto

Entendimento do negócio

Nesse documento será explicada a implantação de um modelo preditivo da severidade de acidentes de trânsito nos Estados Unidos.

Definições do problema

O problema encontrado é referente aos casos de acidentes de trânsito nos Estados Unidos da América. Muitos acidentes de trânsito podem ter resultados severos ou fatais, e então faz-se necessária uma investigação para entender melhor as principais razões desses acidentes graves e, com a ajuda de algoritmos, prever a gravidade mais provável de ocorrer nessas circunstâncias, e então ser tomada as devidas medidas para o atendimento ao acidente em questão.

Escopo

Com as informações diversas sobre as características dos acidentes que ocorrem no território americano, será feito uma análise do comportamento dos dados em diversas ocasiões, como a variação da severidade do acidente durante os dias da semana, por exemplo. terminada a análise, será feita a preparação dos dados para previsão da severidade em determinado estado escolhido

Plano

Será utilizado um conjunto de dados, disponível no endereço https://www.kaggle.com/sobhanmoosavi/us-accidents, onde são demonstradas informações sobre diversos acidentes em todo o território principal dos Estados Unidos. Características sobre os acidentes, tais como dia e hora, temperatura, umidade, pressão, latitude e longitude, a visibilidade e a severidade do acidente são presentes no dataset.

Figura - Trecho do conjunto de dados

	ID	Severity	Start_Lng	Start_Lat	Distance_mi	Side	City	County	State	Timezone	...	year	month	dayWeek	day	hour	Duration
0	A-1	3.0	-84.058723	39.8651	0.0	R	Dayton	Montgomery	OH	US/Eastern	...	2016.0	2.0	Monday	8.0	5.0	6.
1	A-10	3.0	-82.925194	40.1006	0.0	R	Westerville	Franklin	OH	US/Eastern	...	2016.0	2.0	Monday	8.0	8.0	1.
2	A-100	2.0	-84.139359	39.7499	0.0	R	Dayton	Montgomery	OH	US/Eastern	...	2016.0	2.0	Thursday	11.0	8.0	1.
3	A-1000	2.0	-121.070541	38.6531	0.0	R	El Dorado Hills	El Dorado	CA	US/Pacific	...	2016.0	6.0	Thursday	23.0	10.0	1.
4	A-10000	3.0	-121.577354	38.5744	0.0	R	West Sacramento	Yolo	CA	US/Pacific	...	2017.0	1.0	Friday	6.0	16.0	1.

Com esses dados em mãos, inicia-se a fase de análise das informações, onde são relacionadas as características numéricas e categóricas, gerando gráficos para se ter o melhor entendimento do comportamento dos dados. Tendo feita esta análise, inicia-se o

processo de modelagem dos dados, que inclui a limpeza,imputação de dados faltantes, retirada de colunas que não agregam para a previsão, dentre outros métodos.
Após a limpeza, a tabela de dados será dividida em tabela de treino e tabela de teste. Com a tabela de trein, será introduzido um algoritmo de classificação, a fim de prever o resultado da severidade de acordo com as características da tabela de treino. com o modelo treinado, o mesmo será testado, com a tabela de teste, gerando assim resultados sobre a acurácia do modelo para as diversas severidades, podendo ser demonstrada uma matriz de confusão para melhor entendimento dos resultados atingidos. com o modelo validado, será posto em produção um script para a introdução de dados randômicos,utilizando do modelo para prever a severidade dos acidentes.

Equipe Pessoal

Vítor Veiga - Administrador formado pela UECE, Pós Graduando em Ciências de Dados e Inteligência de Negócios pela UNICHRISTUS. Atividades Realizadas: Análise Exploratória; Análise Estatísticas; Seleção de Features e Tratamento de dados; Análise de resultados do modelo e Definição de estratégia de uso.

Mariana Guimarães - Cientista da Computação formada pela UNIP, Pós Graduanda em Ciências de Dados e Inteligência de Negócios pela UNICHRISTUS. Atividades Realizadas: Responsável pela implementação do Banco de Dados; Implementação do código de Machine Learning; Análise de Resultados do modelo e Definição de estratégia de uso.

Luana Lima - Cientista da Computação formada pela UNIFOR, Pós Graduanda em Ciências de Dados e Inteligência de Negócios pela UNICHRISTUS. Atividades Realizadas: Seleção de Features e Tratamento de dados; Visualização da amostragem e distribuição das colunas de seleção; responsável pela parte de visualização; Análise de resultados do modelo e Definição de estratégia de uso

Pedro Caracas - Engenheiro Civil formado pela UNIFOR, Pós Graduando em Ciências de Dados e Inteligência de Negócios pela UNICHRISTUS. Atividades Realizadas: Contribuição para o código de produção, Análise de resultados do modelo; Definição de estratégia de uso e elaboração do relatório.

Métricas

Como métricas de avaliação der esultado do modelo, será utilizada a quantidade de acertos de cada categoria de severidade, sendo verificada , por meio de uma matriz de confusão, a quantidade de acidentes de severidade 3 que realmente foram qualificados pelo modelo como de categoria 3 e que foram classificadas erroneamente nas outras categorias, por exemplo.

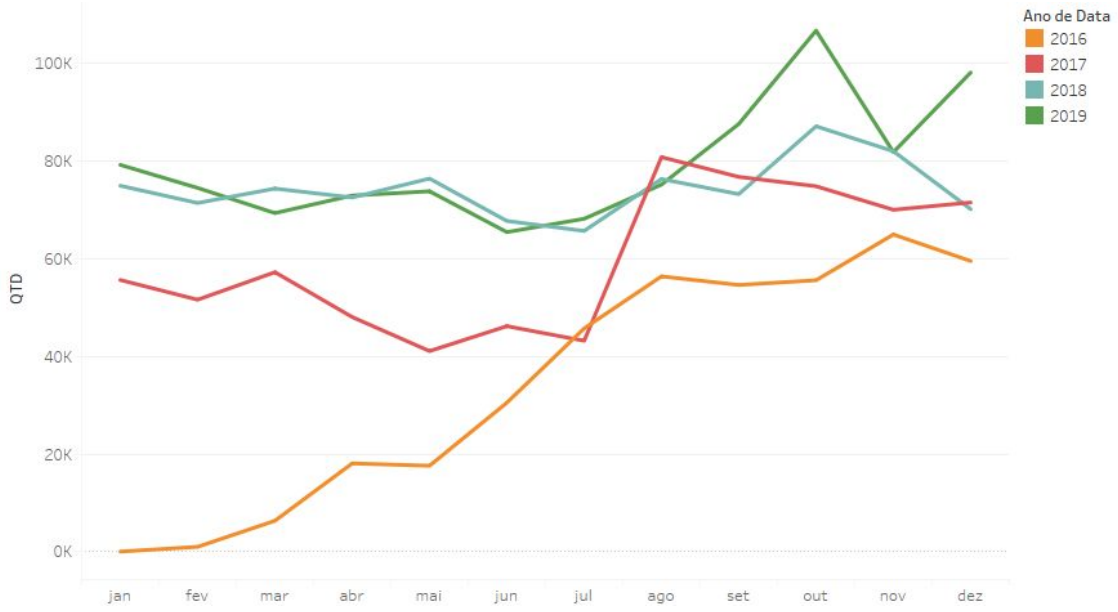
Análise dos dados

Para informações detalhadas sobre os dados, consulte a descrição no repositório do Kaggle.

A gravidade é um número entre 1 e 4, que indica o impacto que o acidente teve no trânsito. Sendo que 1 indica o menor impacto possível e 4 indica o maior impacto possível no trânsito.

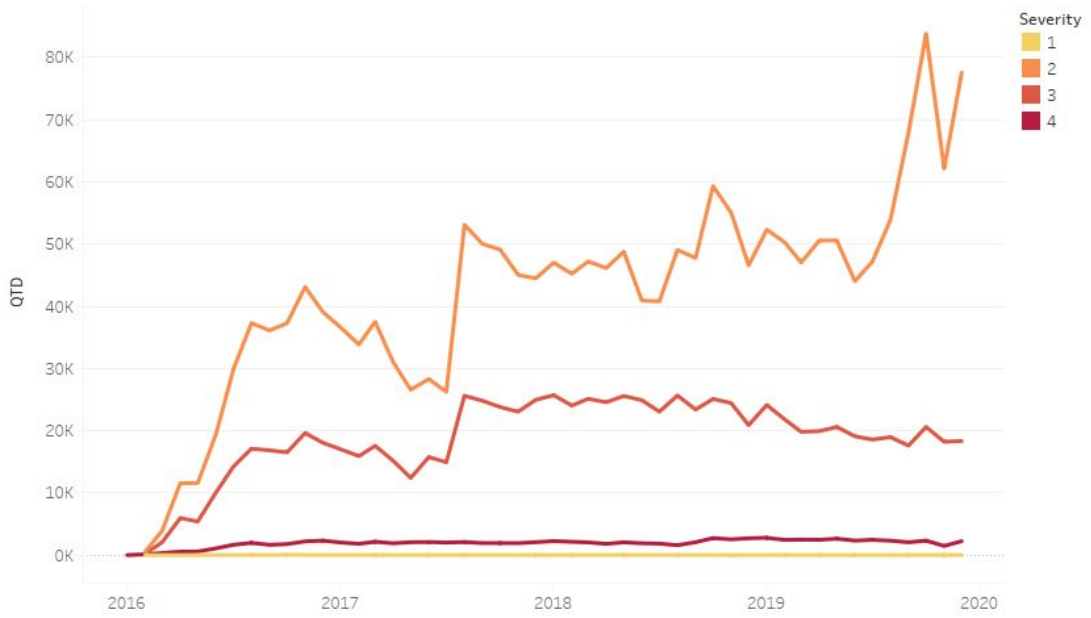
Examinaremos três grupos que impactam acabam impactando na gravidade do acidente: tempo, clima e infraestrutura.

Tempo



É interessante perceber que o primeiro ano (2016) é de crescimento, muito provavelmente decorrente do processo de coleta de dados que estava se iniciando.

Observando 2018 e 2019 podemos ver que existe uma certa sazonalidade, havendo sempre uma aumento no número de acidentes no fim do ano.

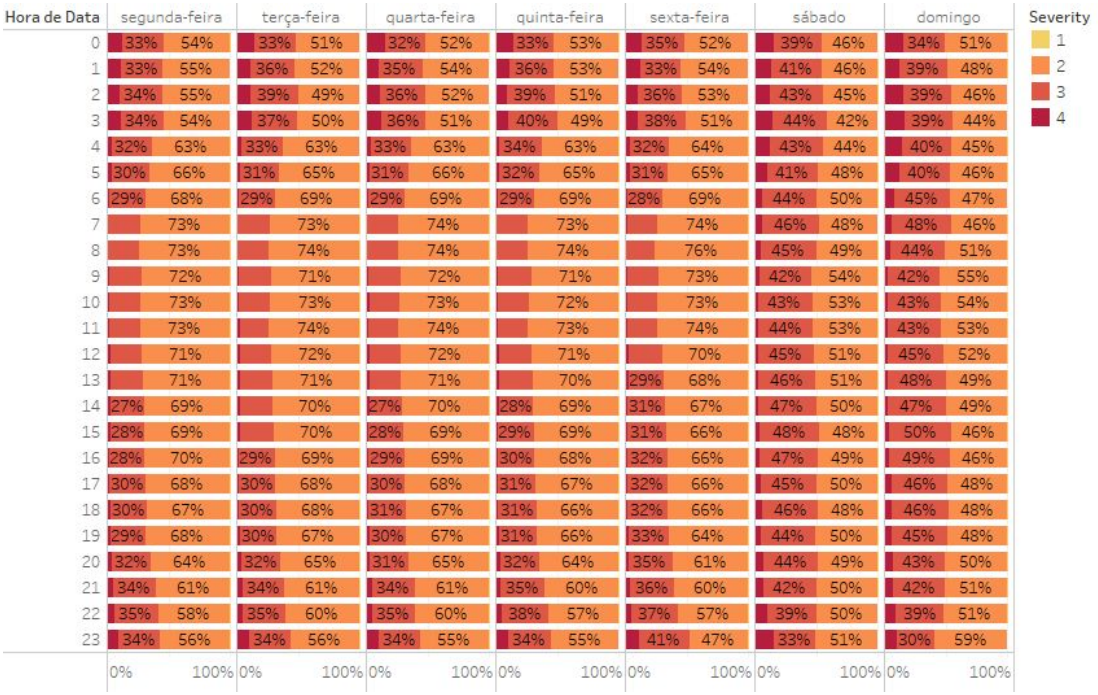


Os acidentes de gravidade 1 e 4 (os dois extremos) são os com menor incidência ao longo dos anos. É interessante perceber também, que para esses dois tipos de acidentes, não tem muita variação em relação a quantidade ao longo dos anos (nem para mais, nem para menos).

Já os acidentes de gravidade 2 são os que tem a maior incidência e vem aumentando a cada ano. Os acidentes de gravidade 3 tiveram alguns picos de crescimento no início, mas depois se estabilizaram.

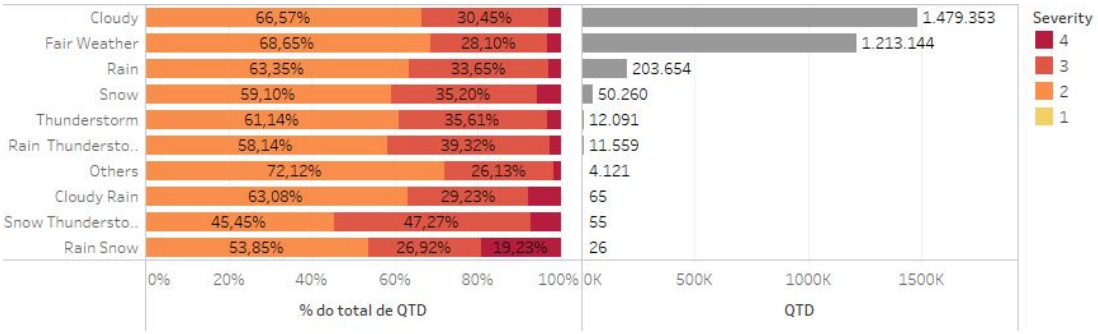
Hora..	seg	ter	qua	qui	sex	sáb	dom
0	3.419	3.218	3.284	3.186	3.425	3.294	3.357
1	2.634	2.274	2.359	2.434	2.604	2.563	3.132
2	2.716	2.357	2.233	2.352	2.692	3.201	3.393
3	2.836	2.478	2.411	2.507	2.855	2.547	2.966
4	9.966	8.680	8.903	9.042	10.550	3.203	3.250
5	16.016	15.463	15.464	14.534	15.375	3.496	3.518
6	31.633	34.279	33.541	30.584	29.039	5.200	4.803
7	50.263	57.722	55.176	51.185	45.931	7.048	6.180
8	52.059	60.311	57.997	53.263	46.995	7.575	6.074
9	30.927	35.020	34.554	31.976	29.227	8.966	6.798
10	27.146	28.722	28.481	27.324	27.814	10.266	8.114
11	25.874	26.807	26.781	26.067	28.688	11.633	9.480
12	22.225	22.848	23.040	22.933	25.590	12.397	10.133
13	22.412	22.868	23.515	23.266	26.734	12.469	10.763
14	24.098	24.805	24.699	25.108	28.563	12.101	10.868
15	29.526	30.588	30.869	31.117	35.098	10.922	9.535
16	36.196	38.205	37.860	38.746	42.035	10.308	9.226
17	38.136	41.693	41.347	42.241	41.663	8.665	7.896
18	27.786	30.952	30.823	30.883	31.702	8.090	7.175
19	18.649	20.160	20.021	20.990	22.217	7.070	6.306
20	13.134	13.677	13.682	14.335	15.734	5.741	5.430
21	8.674	9.007	8.857	9.541	10.392	5.807	5.461
22	7.216	7.662	7.542	8.231	8.745	4.588	4.315
23	3.713	3.929	4.029	4.293	4.081	3.199	3.472

Fazendo um relação entre dia da semana e hora, podemos ver que a maioria dos acidentes costumam acontecer na durante a semana, principalmente nos horários de pico (entre 6 e 9 da manhã, quando as pessoas costumam sair de casa para ir trabalhar, e entre 15 e 18, quando estão voltando do trabalho).

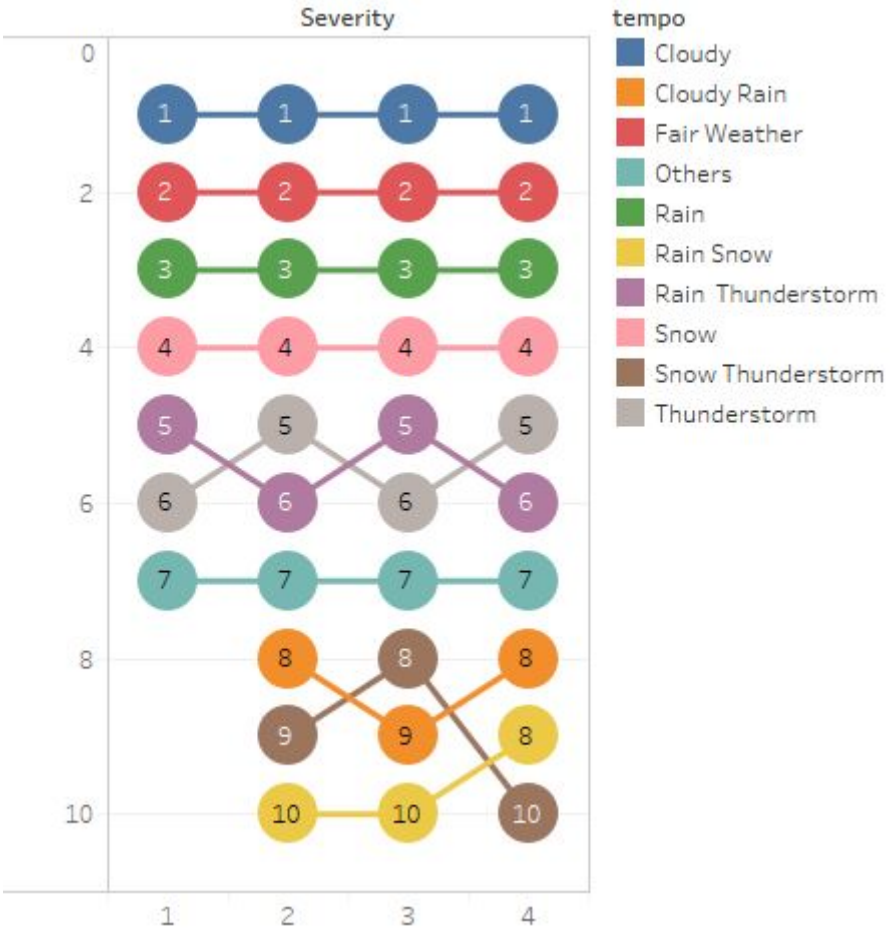


Mas se formos analisar o percentual de cada gravidade dentro dessa relação, podemos ver que durante a semana de madrugada e no fim de semana o percentual de acidentes de gravidade 3 e 4 aumenta consideravelmente. E que durante a semana nos horários de pico acontece predominantemente acidentes de gravidade 2.

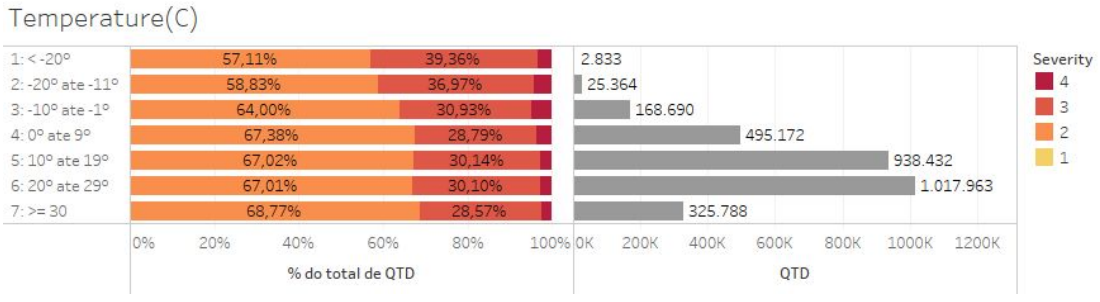
Clima



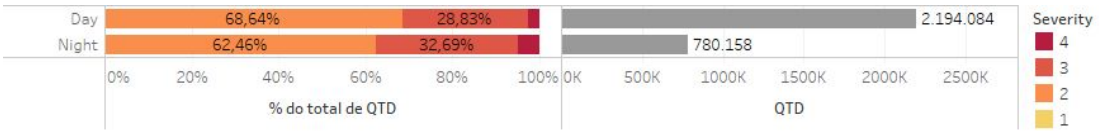
Os dois climas com maior ocorrência (Cloudy e Fair Weather) são provavelmente os que também ocorrem com mais frequência durante o ano. É interessante ver também que o percentual dos acidentes de maior gravidade (3 e 4) aumenta com os climas que ocorrem com menor frequência.



Analisando o Rank dos climas em cada gravidade, podemos perceber que não tem muita variação. As posições praticamente se mantém iguais nos quatro níveis de gravidades. Mas é interessante ver que as três últimas posições não apresentaram nenhum acidentes de gravidade 1, o que reforça ainda mais o que já foi dito, que os climas com menor ocorrência tem maior probabilidade de gerar acidentes de maior gravidade.

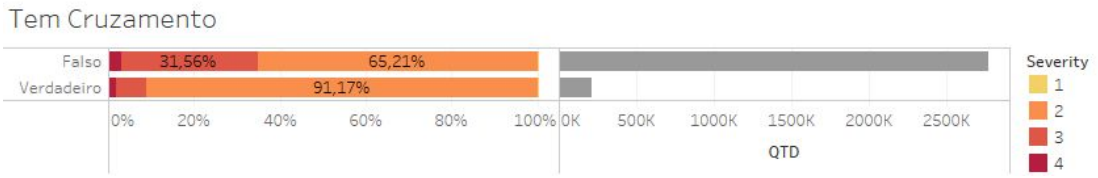


A temperatura também acaba influenciado para a gravidade, quanto menor a temperatura maior o percentual de acidentes de gravidade mais alta (3 e 4).

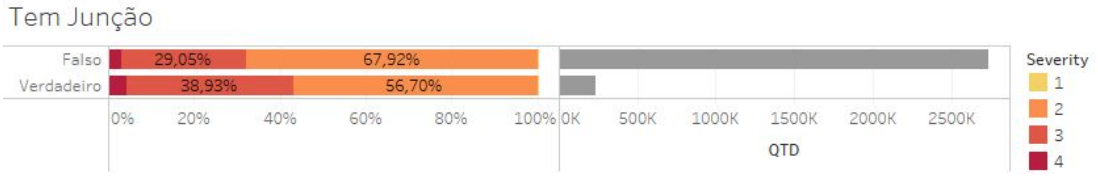


Infraestrutura

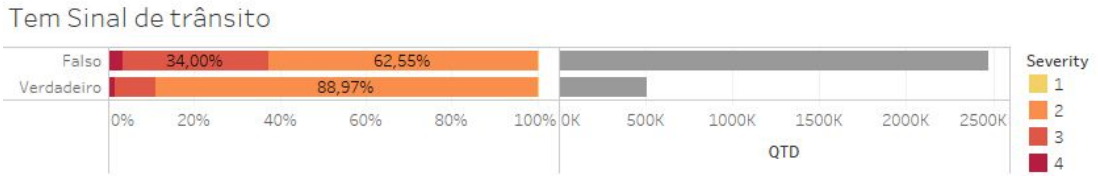
Não acontecem muitos acidentes nas vias com cruzamento e quando ocorrem têm uma grande possibilidade de ser de severidade 2.



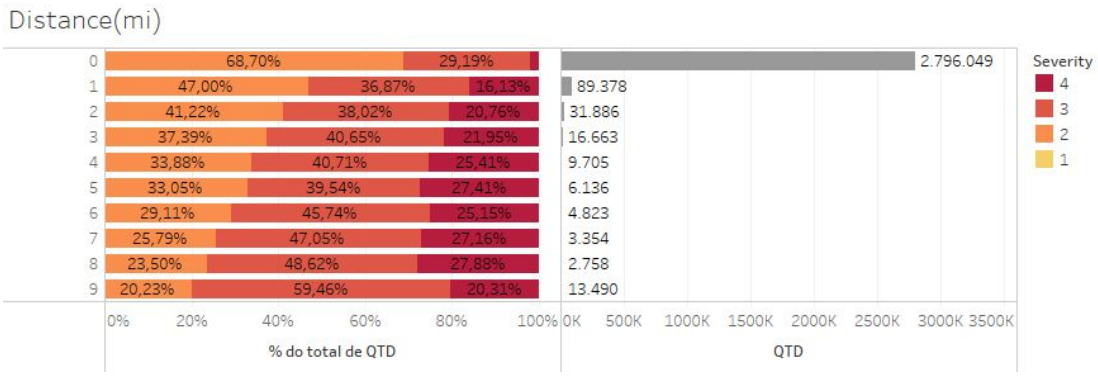
Apesar de ter poucos acidentes em vias com junção, esses acidentes acabam tendo uma severidade maior.

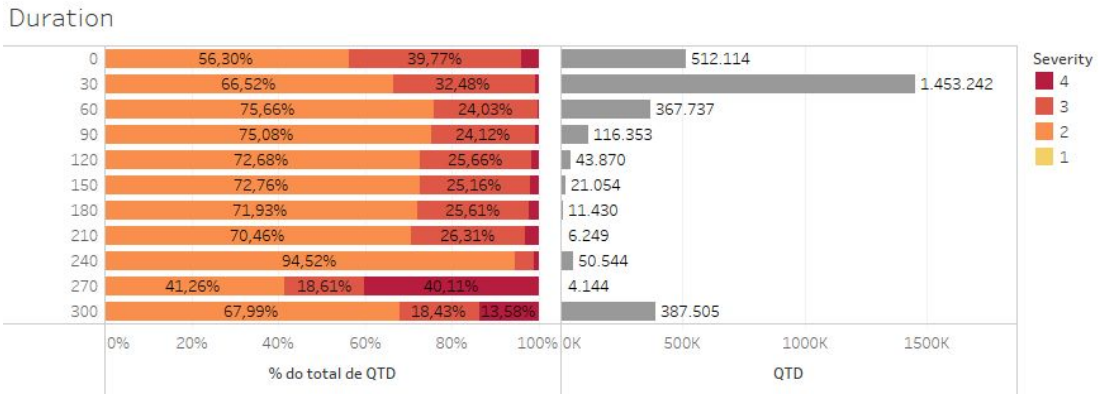


Acontecem bem mais acidentes em ruas que não possuem sinal de trânsito e esses acidentes tendem a ter uma severidade maior.



Quanto maior a distância e a duração que o acidente causou ao trânsito, maior o percentual de acidentes de gravidade 3 e 4.





Modelagem

Feature Engineering

Limpeza de dados

Esses campos foram removidos, pois durante a análise foi identificado que não tinham relevância para a definição da severidade: End_Lat, End_Lng, Weather_Timestamp, Airport_Code, Civil_Twilight, Nautical_Twilight, Description, Country, Give_Way, Bump, Traffic_Calming, Roundabout, No_Exit, Street, Source, Turning_Loop, Zipcode, Precipitation(in), Pressure(in), Wind_Chill(F), Visibility(mi), Wind_Speed(mph), Number, Astronomical_Twilight, County, City, Humidity(%), Amenity, Railway, Station, Stop, Wind_Direction, Timezone, Side

No campo Weather_Condition foi identificamos 121 tipos de climas. Analisando melhor esses campo, pude perceber duas coisas. A primeira, tinha muitos clima similares variando apenas a intensidade como por exemplo Light Snow e Heavy Snow. Com base nisso, agrupamos os climas em cinco grupos: Cloudy, Rain, Snow, Thunderstorm e Fair Weather. A segunda, que alguns clima eram a interseção entre dois tipos de climas, como por exemplo Heavy Thunderstorms and Snow e Heavy Thunderstorms and Rain.

Com base nisso, foi criado cinco colunas: Fair_Weather, Cloudy, Rain, Snow e Thunderstorms. Todas sendo preenchidas com 0 ou 1, para identificar se houve ou não presença daquele tipo de clima no acidente.

Tratamento dos campos numéricos

Nas colunas do tipo numérico, os dados nulos foram preenchidos com a sua média.

A coluna Temperature(F) que está em Fahrenheit foi transformada para Celsius.

Para tranar tar os outliers e o modelo por trabalhar melhor com os campos numéricos foi feito um agrupamento dos dados:

- Distance(mi)
    - 1 = valor menor que 1 mi
    - 2 = valor entre 1 min e 2 min
    - 3 = valor entre 2 min e 3 min
    - 4 = valor entre 3 min e 4 min
    - 5 = valor entre 4 min e 5 min
    - 6 = valor entre 5 min e 6 min
    - 7 = valor entre 6 min e 7 min
    - 8 = valor entre 7 min e 8 min
    - 9 = valor entre 8 min e 9 min
    - 10 = valor maior que 9 min
- Duration
    - 1 = valor menor que 30 min
    - 2 = valor entre 30 min e 60 min
    - 3 = valor entre 60 min e 90 min
    - 4 = valor entre 90 min e 120 min
    - 5 = valor entre 120 min e 150 min
    - 6 = valor entre 150 min e 180 min
    - 7 = valor entre 180 min e 210 min
    - 8 = valor entre 210 min e 240 min
    - 9 = valor entre 240 min e 270 min
    - 10 = valor entre 270 min e 300 min
    - 11 = valor maior que 300 min
- Humidity(%)
    - 1 = valor menor que 10
    - 2 = valor entre 10 e 20
    - 3 = valor entre 20 e 30
    - 4 = valor entre 30 e 40
    - 5 = valor entre 40 e 50
    - 6 = valor entre 50 e 60
    - 7 = valor entre 60 e 70
    - 8 = valor entre 70 e 80
    - 9 = valor entre 80 e 90
    - 10 = valor entre 90 e 100
    - 11 = valor maior que 100
- Temperature(C)
    - 1 = valores menor que -20° C
    - 2 = valor entre -20° C e -10° C
    - 3 = valor entre -10° C e 0° C
    - 4 = valor entre 0° C e 10° C
    - 5 = valor entre 10° C e 20° C
    - 6 = valor entre 20° C e 30° C
    - 7 = valor maior que 30° C

Na coluna de severidade, a feature em estudo, as severidades 1 e 2 foram fundidas, para formar a severidade 1, já que a diferença entre as originais severidades 1 e 2, de acordo com a descrição do dataset, é muito pequena. a severidade 3 original foi transformada em severidade 2, e a severidade 4 original foi transformada em severidade 3.

```
In [6]: #Deleta os campos que não são significantes
acidentes.drop(['End_Lat','End_Lng','Weather_Timestamp','Airport_Code','Civil_Twilight',
                'Nautical_Twilight','Description','Country','Give_Way','Bump','Traffic_Calming',
                'Roundabout','No_Exit','Street','Source','Turning_Loop','Zipcode','Precipitation(in)',
                'Pressure(in)','Wind_Chill(F)','Visibility(mi)','Wind_Speed(mph)','Number',
                'Astronomical_Twilight','County','City','Humidity(%)','Amenity','Railway','Station',
                'Stop','Wind_Direction','Timezone','Side'],axis = 1, inplace = True)

In [7]: #Removendo caracteres especiais das colunas
acidentes.rename(columns={'Temperature(F)': 'TemperatureF',
                          'Wind_Chill(F)': 'Wind_ChillF',
                          'Wind_Speed(mph)': 'Wind_Speed_mph',
                          'Distance(mi)': 'Distance_mi',
                          'Visibility(mi)': 'Visibility_mi'}, inplace = True)

In [8]: #Preparando e gerando os dados adicionais

acidentes['Start_Time'] = pd.to_datetime(acidentes['Start_Time'])
acidentes['End_Time'] = pd.to_datetime(acidentes['End_Time'])

acidentes['Hour'] = acidentes['Start_Time'].dt.hour
acidentes['Month'] = acidentes['Start_Time'].dt.month
acidentes['DayWeek'] = acidentes['Start_Time'].dt.strftime('%w') #%A
acidentes['Duration']= abs(round((acidentes['End_Time']-acidentes['Start_Time'])/np.timedelta64(1,'m')))

acidentes['TemperatureF'] = acidentes['TemperatureF'].astype(float)
acidentes['TemperatureC'] = (((acidentes['TemperatureF'].astype(float)) - 32) * 5/9)


#Depois de criar o Duration e TemperaturaC, nao precisa mais do start e endtime e TemperaturaF
acidentes = acidentes.drop(['Start_Time','End_Time','TemperatureF'],axis = 1)
```

```
In [9]: acidentes['Weather_Condition'] = acidentes['Weather_Condition'].fillna('')

#Cria os campos binarios referentes aos tipos de climas
acidentes['Fair_Weather'] = acidentes['Weather_Condition'].apply(lambda x: 1 if 'CLEAR' in x or 'FAIR' in x else

acidentes['Cloudy'] = acidentes['Weather_Condition'].apply(lambda x: 1 if 'CLOUDY' in x or 'CLOUDS' in x
                                                           or 'CLOUD' in x or 'OVERCAST' in x
                                                           or 'FOG' in x or 'HAZE' in x
                                                           or 'MIST' in x else 0)

acidentes['Rain'] = acidentes['Weather_Condition'].apply(lambda x: 1 if 'RAIN' in x or 'DRIZZLE' in x
                                                         or 'SHOWERS' in x else 0)

acidentes['Snow'] = acidentes['Weather_Condition'].apply(lambda x: 1 if 'SNOW' in x or 'SLEET' in x
                                                         or 'HAIL' in x or 'ICE' in x
                                                         or 'WINTRY' in x else 0)

acidentes['Thunderstorms'] = acidentes['Weather_Condition'].apply(lambda x: 1 if 'T-STORM' in x or 'THUNDERSTORM
                                                                  or 'THUNDERSTORM' in x or 'THUNDER' in x
                                                                  or 'TORNADO' in x or 'SQUALLS' in x else 0)

In [10]: #Preenche registros nulos com a mediana
acidentes['TemperatureC'].fillna((acidentes['TemperatureC'].median()), inplace=True)

#Preenche registros nulos com 0
acidentes['TMC'] = acidentes['TMC'].fillna('0')

In [11]: #Gera dados NaN para valores nulos e remove estas linhas
acidentes[acidentes['Sunrise_Sunset']==''] = np.nan
acidentes = acidentes[acidentes['Sunrise_Sunset'].notna()]
```

In [10]:
```
#Preenche registros nulos com a mediana
acidentes['TemperatureC'].fillna((acidentes['TemperatureC'].median()), inplace=True)

#Preenche registros nulos com 0
acidentes['TMC'] = acidentes['TMC'].fillna('0')
```

In [11]:
```
#Gera dados NaN para valores nulos e remove estas linhas
acidentes[acidentes['Sunrise_Sunset']==''] = np.nan
acidentes = acidentes[acidentes['Sunrise_Sunset'].notna()]
```

In [12]:
```
#Agrupando Duracao
acidentes.loc[(acidentes['Duration'] < 30), 'Duration_Group'] = 1
acidentes.loc[(acidentes['Duration'] >= 30)  & (acidentes['Duration'] <  60), 'Duration_Group'] = 2
acidentes.loc[(acidentes['Duration'] >= 60)  & (acidentes['Duration'] <  90), 'Duration_Group'] = 3
acidentes.loc[(acidentes['Duration'] >= 90)  & (acidentes['Duration'] < 120), 'Duration_Group'] = 4
acidentes.loc[(acidentes['Duration'] >= 120) & (acidentes['Duration'] < 150), 'Duration_Group'] = 5
acidentes.loc[(acidentes['Duration'] >= 150) & (acidentes['Duration'] < 180), 'Duration_Group'] = 6
acidentes.loc[(acidentes['Duration'] >= 180) & (acidentes['Duration'] < 210), 'Duration_Group'] = 7
acidentes.loc[(acidentes['Duration'] >= 210) & (acidentes['Duration'] < 240), 'Duration_Group'] = 8
acidentes.loc[(acidentes['Duration'] >= 240) & (acidentes['Duration'] < 270), 'Duration_Group'] = 9
acidentes.loc[(acidentes['Duration'] >= 270) & (acidentes['Duration'] < 300), 'Duration_Group'] = 10
acidentes.loc[(acidentes['Duration'] >= 300), 'Duration_Group'] = 11
```

In [13]:
```
#Agrupando Distance_mi
acidentes.loc[(acidentes['Distance_mi'] < 1), 'Distance_Group'] = 1
acidentes.loc[(acidentes['Distance_mi'] >= 1) & (acidentes['Distance_mi'] < 2), 'Distance_Group'] = 2
acidentes.loc[(acidentes['Distance_mi'] >= 2) & (acidentes['Distance_mi'] < 3), 'Distance_Group'] = 3
acidentes.loc[(acidentes['Distance_mi'] >= 3) & (acidentes['Distance_mi'] < 4), 'Distance_Group'] = 4
acidentes.loc[(acidentes['Distance_mi'] >= 4) & (acidentes['Distance_mi'] < 5), 'Distance_Group'] = 5
acidentes.loc[(acidentes['Distance_mi'] >= 5) & (acidentes['Distance_mi'] < 6), 'Distance_Group'] = 6
acidentes.loc[(acidentes['Distance_mi'] >= 6) & (acidentes['Distance_mi'] < 7), 'Distance_Group'] = 7
acidentes.loc[(acidentes['Distance_mi'] >= 7) & (acidentes['Distance_mi'] < 8), 'Distance_Group'] = 8
acidentes.loc[(acidentes['Distance_mi'] >= 8) & (acidentes['Distance_mi'] < 9), 'Distance_Group'] = 9
```

In [14]:
```
#Agrupando TemperatureC
acidentes.loc[(acidentes['TemperatureC'] < -20), 'TemperatureC_Group'] = 1
acidentes.loc[(acidentes['TemperatureC'] >= -20) & (acidentes['TemperatureC'] < -10), 'TemperatureC_Group'] = 2
acidentes.loc[(acidentes['TemperatureC'] >= -10) & (acidentes['TemperatureC'] <   0), 'TemperatureC_Group'] = 3
acidentes.loc[(acidentes['TemperatureC'] >=   0) & (acidentes['TemperatureC'] <  10), 'TemperatureC_Group'] = 4
acidentes.loc[(acidentes['TemperatureC'] >=  10) & (acidentes['TemperatureC'] <  20), 'TemperatureC_Group'] = 5
acidentes.loc[(acidentes['TemperatureC'] >=  20) & (acidentes['TemperatureC'] <  30), 'TemperatureC_Group'] = 6
acidentes.loc[(acidentes['TemperatureC'] >=  30), 'TemperatureC_Group'] = 7
```

In [15]:
```
#Agrupando serveridade
acidentes.loc[(acidentes['Severity'] == 1), 'Severity_Group'] = 1
acidentes.loc[(acidentes['Severity'] == 2), 'Severity_Group'] = 1
acidentes.loc[(acidentes['Severity'] == 3), 'Severity_Group'] = 2
acidentes.loc[(acidentes['Severity'] == 4), 'Severity_Group'] = 3
```

In [16]:
```
acidentes.loc[(acidentes['Sunrise_Sunset'] == 'DAY'), 'Night'] = 0
acidentes.loc[(acidentes['Sunrise_Sunset'] == 'NIGHT'), 'Night'] = 1
```

In [17]:
```
acidentes['TMC'] = acidentes['TMC'].astype('int32', copy=False)
acidentes['Hour'] = acidentes['Hour'].astype('int32', copy=False)
acidentes['Month'] = acidentes['Month'].astype('int32', copy=False)
acidentes['DayWeek'] = acidentes['DayWeek'].astype('int32', copy=False)
```

In [18]:
```
acidentes['Rain'] = acidentes['Rain'].astype('int32', copy=False)
acidentes['Snow'] = acidentes['Snow'].astype('int32', copy=False)
acidentes['Cloudy'] = acidentes['Cloudy'].astype('int32', copy=False)
acidentes['Fair_Weather'] = acidentes['Fair_Weather'].astype('int32', copy=False)
acidentes['Thunderstorms'] = acidentes['Thunderstorms'].astype('int32', copy=False)
```

In [19]:
```
acidentes['Duration_Group'] = acidentes['Duration_Group'].astype('int32', copy=False)
acidentes['Distance_Group'] = acidentes['Distance_Group'].astype('int32', copy=False)
acidentes['TemperatureC_Group'] = acidentes['TemperatureC_Group'].astype('int32', copy=False)
acidentes['Severity_Group'] = acidentes['Severity_Group'].astype('int32', copy=False)
```

In [20]:
```
acidentes['Night'] = acidentes['Night'].astype('int32', copy=False)
```

In [21]:
```
acidentes['Crossing'] = acidentes['Crossing'].astype('int32', copy=False)
acidentes['Junction'] = acidentes['Junction'].astype('int32', copy=False)
acidentes['Traffic_Signal'] = acidentes['Traffic_Signal'].astype('int32', copy=False)
```

In [22]:
```
acidentes = acidentes.drop(['TemperatureC','Distance_mi','Duration','Severity','Sunrise_Sunset','Weather_Conditi
```

In [23]:
```
#Criando index
acidentes = acidentes.reset_index()
acidentes['ID'] = acidentes.index
acidentes.drop('index',axis=1, inplace=True)
```

após esses tratamentos, tivemos um dataframe com vinte e uma colunas.

Salvando dados processados

Os dados tratados no processo anterior foram salvos em um banco mysql, em partes separadas para que o processamento nesse processo não ficasse muito elevado

```
In [24]: #Separando dataframe para insert segmentado
         acc1_ = acidentes.iloc[0:100]
         acc2_ = acidentes.iloc[100:400000]
         acc3_ = acidentes.iloc[400000:800000]
         acc4_ = acidentes.iloc[800000:1200000]
         acc5_ = acidentes.iloc[1200000:2000000]
         acc6_ = acidentes.iloc[2000000:2500000]
         acc7_ = acidentes.iloc[2500000:3000000]
```

Com isso, cada tabela de nome acc#_ foi repassada para o banco remoto pelo mêtodo append.

```
In [25]: #Criando tabela e inserindo dados tratados
         acc1_.to_sql('us_accident_prep', con = db_connection, if_exists = 'replace', index=False ,chunksize = 10000)

In [26]: acc2_.to_sql('us_accident_prep', con = db_connection, if_exists = 'append', index=False ,chunksize = 10000)

In [27]: acc3_.to_sql('us_accident_prep', con = db_connection, if_exists = 'append', index=False , chunksize = 10000)

In [28]: acc4_.to_sql('us_accident_prep', con = db_connection, if_exists = 'append', index=False , chunksize = 10000)

In [29]: acc5_.to_sql('us_accident_prep', con = db_connection, if_exists = 'append', index=False , chunksize = 10000)

In [30]: acc6_.to_sql('us_accident_prep', con = db_connection, if_exists = 'append', index=False , chunksize = 10000)

In [31]: acc7_.to_sql('us_accident_prep', con = db_connection, if_exists = 'append', index=False , chunksize = 10000)

In [32]: db_connection.execute("commit")
Out[32]: <sqlalchemy.engine.result.ResultProxy at 0x1f277598320>
```

Treinamento do modelos

A captura dos dados do banco é feita atraves de uma query na table us_accident_prep , onde o estado a ser estudado é passado como condição where

```
db_connection_str = 'mysql+pymysql://marianag_dev:senhasecreta@host8.hospedameusite.com.br/marianag_Acidentes_Po
db_connection = create_engine(db_connection_str)
```

```
In [3]: #Escolha um estado para treino e teste
        #(Estados com dados de serveridade balanceados e maior numero de registros 'NY', 'GA' e 'MN' e 'MD')
        state = 'NY'
        #Busca dados no bando de dados MySql
        query = "SELECT * FROM us_accident_prep where State = '" + state + "'"
```

Com isso, são excluídas as colunas ID e State, que não influenciam em nada o processo de treinamento, ja que ID é uma variável única de cada registro, e o State é sempre o mesmo, neste caso, NY

Após isso, é feito o carregamento do dataframe de pandas para a plataforma dask, que utiliza de processamento paralelo, para carregar as informações mais rapidamente. Feito isso, é utilizado o one-hot encoding das variáveis categóricas para se obter uma maior quantidade de colunas booleanas, em que a o algoritmo classificador melhor opera. depois são divididas as features entre y: categoria alvo para predição; e x: categorias utilizadas no

treinamento para prever a categoria y. após isso, a tabela foi dividida em duas tabelas: uma de treino que corresponde a 70% das linhas da tabela original; e 30% para a tabela de teste, que irá servir para testar a acurácia do modelo treinado
O algoritmo escolhido para a predição foi o random forest classifier , em que são geradas várias árvores de decisão, e então a média dos resultados dessas várias árvores é utilizado como resultado final para a classificação. o algoritmo foi escolhido entre os outros, pois , como nosso objetivo é prever a severidade de uma acidente, faz sentido utilizar um algoritmo classificador, e o random forest serve tanto para se ter uma melhor explicação de como funciona o processo de decisão dele, bem como foi o algoritmo com maior resultado dentre os vários algoritmos de classificação testados no dataframe. No caso desse estudo, foi utilizado um classificador com cem árvores de decisão diferentes.

```
In [5]: #Busca dados no bando de dados MySql
        acc = pd.read_sql(query, con=db_connection)

        acidentes_state = acc

In [6]:
        acidentes_state.shape

Out[6]: (137779, 21)

In [7]: acidentes_state = acidentes_state.drop(['ID','State'],axis = 1)

In [8]: acidentes_dask = dd.from_pandas(acidentes_state, npartitions=3)
        trn = dd.get_dummies(acidentes_dask.categorize()).compute()

In [9]: #Montando coluna de previsao e separando os datasets em teste e treino
        target = 'Severity_Group'#Outra opção seria DurationC
        y = trn[target]
        X = trn.drop(target, axis=1)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=21, stratify=y)

In [10]: #ML Treinando modelo utilizando randomForest
         clf=RandomForestClassifier(n_estimators=100)
         clf.fit(X_train,y_train)

Out[10]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=None, max_features='auto', max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=100,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)
```

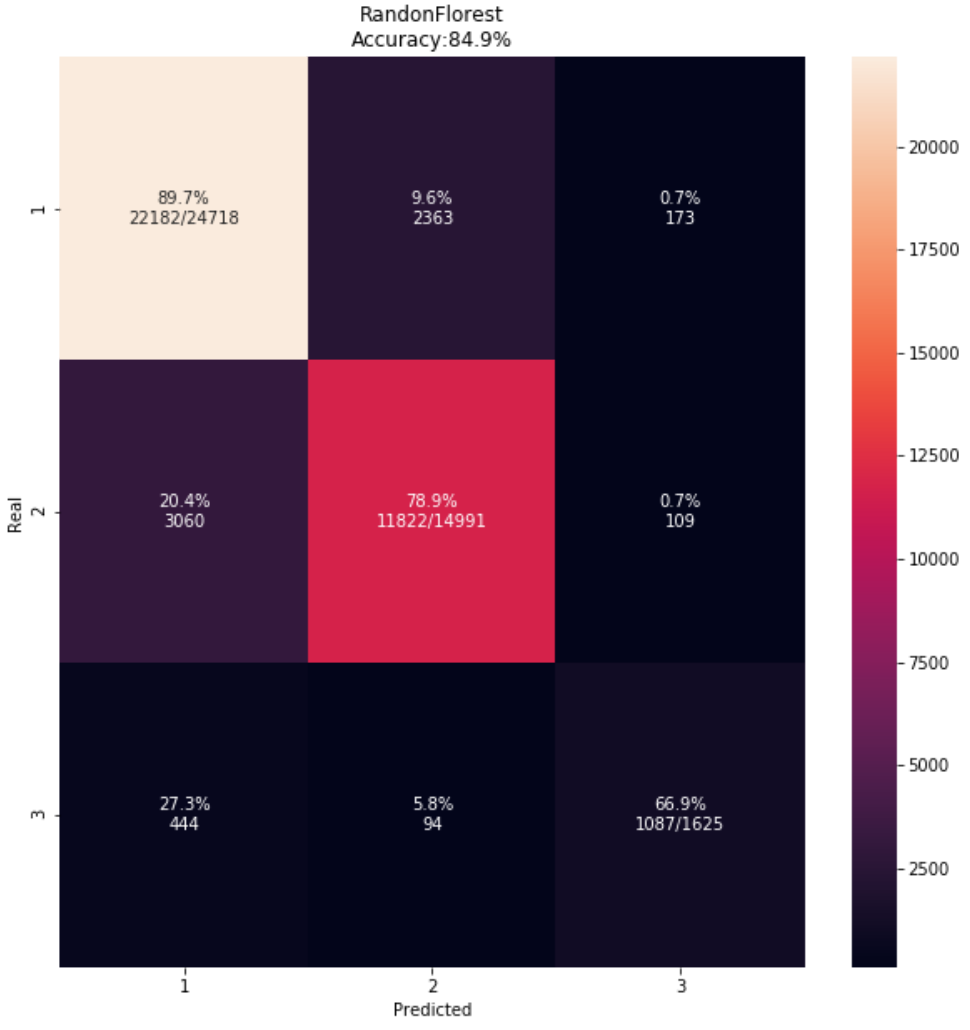
Avaliação do modelo

Tendo o modelo treinado, podemos avaliar com várias métricas. Nesse estudo, foram utilizados o acurracy score e uma matriz de confusão, para determinar o resultado do modelo nas diversas severidades. para o estado de nova iorque, o modelo teve uma taxa de acerto de aproximadamente 85%

```
In [11]: #Fazendo a previsão com modelo treinado usando dataset de teste
         y_pred=clf.predict(X_test)

In [12]: #Medindo acuracia do resultado
         acc=accuracy_score(y_test, y_pred)
         resultados=[]

In [13]: #Ajustando output
         resultados=[]
         resultados.append("Model RandonForest")
         resultados.append("Model Target:" + target)
         resultados.append("Accuracy:" + str(acc))
         resultados

Out[13]: ['Model RandonForest',
          'Model Target:Severity_Group',
          'Accuracy:0.8489621135142982']
```

No entanto, ao ver a matriz de confusão, percebe-se que a taxa de acerto pela severidade varia muito, onde se teve uma taxa de 89% de acerto para a severidade do tipo 1 , enquanto houve um taxa de acerto de 67% para a severidade 3. sendo isso, a alta taxa de acerto geral se deve ao fato que as severidades 1 e 2 compõem aproximadamente 99% dos acidentes no dataset, com cerca de 175 mil exemplos; já a severidade 3 possui apenas cerca de 1600 acidentes, o que justifica alta taxa de acerto geral em 89%

```
In [14]: #Verificando matriz de confusao
labels=[1,2,3]
ymap=None
figsize=(10,10)
if ymap is not None:
    y_pred = [ymap[yi] for yi in y_pred]
    y_test = [ymap[yi] for yi in y_true]
    labels = [ymap[yi] for yi in labels]
cm = confusion_matrix(y_test, y_pred, labels=labels)
cm_sum = np.sum(cm, axis=1, keepdims=True)
cm_perc = cm / cm_sum.astype(float) * 100
annot = np.empty_like(cm).astype(str)
nrows, ncols = cm.shape
for i in range(nrows):
    for j in range(ncols):
        c = cm[i, j]
        p = cm_perc[i, j]
        if i == j:
            s = cm_sum[i]
            annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
        elif c == 0:
            annot[i, j] = ''
        else:
            annot[i, j] = '%.1f%%\n%d' % (p, c)
cm = pd.DataFrame(cm, index=labels, columns=labels)
cm.index.name = 'Real'
cm.columns.name = 'Predicted'
fig, ax = plt.subplots(figsize=figsize)
sns.heatmap(cm, annot=annot, fmt='', ax=ax)
acc = accuracy_score(y_test, y_pred)*100
plt.title('RandonFlorest\nAccuracy:{0:.1f}%'.format(acc))
plt.show()
```



Implantação

Com as métricas visualizadas, o modelo foi salvo em uma variavel de nome clf e então salva em um arquivo por meio do pacote default do python chamado pickle.

```
In [15]: # Save to file in the current working directory
         pkl_filename = "Models\\"+state + ".pkl"
         with open(pkl_filename, 'wb') as file:
             pickle.dump(clf, file)
```

após isso, o arquivo resultante foi salvo em uma página web para download do modelo e posterior utilização. no Caso foram salvos os modelos treinados de NY, FL, GA, MN, MD e CA.

As tabelas de teste e predição foram salvas no banco relacional utilizado previamente, com as tables nomeadas de acordo com o estado estudado.

```
]: y_testpd = pd.DataFrame(y_test)
   y_predpd = pd.DataFrame(y_pred)

]: y_testpd = y_testpd.reset_index()
   y_testpd['SevPred'] = y_predpd

]: y_testpd.rename(columns={'level_0': 'Pred_Index',
                           'index': 'Test_Index',
                           }, inplace = True)

]: us_accident_PredTest = 'us_accident_PredTest_' + state
   y_testpd.to_sql(us_accident_PredTest, con = db_connection, if_exists = 'replace', index=False , chunksize = 1000
   db_connection.execute("commit");

]: us_accident_PredTest = 'us_accident_TableTest_' + state
   X_test.to_sql(us_accident_PredTest, con = db_connection, if_exists = 'replace', index=False , chunksize = 10000)
   db_connection.execute("commit");
```

Após isso, pode ser iniciado outra instância do python somente para a utilização dos dados
Importando pacotes no novo arquivo:

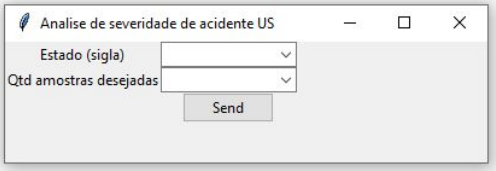
```
#Bibliotecas necessárias para projeto
import numpy as np
import pandas as pd
from pandas import DataFrame
import pymysql
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.io as pio
import dask.dataframe as dd
from sqlalchemy import create_engine
import pymysql as pymysql
import sklearn
import pickle
import tkinter
import seaborn as sns
from urllib.request import Request, urlopen
from random import randint
from folium.plugins import HeatMap
init_notebook_mode(connected=True)
```

É então aberta uma conexão com o banco de dados remoto

```
]: #Abre conexao com banco de dados remoto
db_connection_str = 'mysql+pymysql://marianag_dev:senhasecreta@host8.hospedameusite.com.br/marianag_Acidentes_Po
db_connection = create_engine(db_connection_str)
```

Após, é proporcionado um menu dropdown de seleção para que se escolha alguns dos estados a ser visualizado com novas amostras aleatórias, utilizando o pacote tkinter, sendo o estado escolhido assinalado a uma variável, bem como a quantidade de amostras aleatórias a serem utilizadas na simulação de imput

```
window = tk.Tk()
window.minsize(400, 100)
window.title("Analise de severidade de acidente US")

def chosingNumbers():
   window.destroy()

label = ttk.Label(window, text = "Estado (sigla)")
label.grid(column = 0, row = 0)

amostras = ttk.Label(window, text = "Qtd amostras desejadas")
amostras.grid(column = 0, row = 1)

namostras = tk.StringVar()
cmbobox = ttk.Combobox(window, width = 15, textvariable = namostras)
cmbobox['values'] = ("20","40","60","80","100","120")
cmbobox.grid(column = 1, row = 1)

mynumber = tk.StringVar()
combobox = ttk.Combobox(window, width = 15 , textvariable = mynumber)
combobox['values'] = ("NY","FL","GA","MD","MN","CA")
combobox.grid(column = 1, row = 0)


button = ttk.Button(window, text = "Send", command = chosingNumbers)
button.grid(column = 1, row = 3)



window.mainloop()
```



É então aberto o arquivo contendo o modelo treinado para o estado (assumindo que o modelo ja foi baixado previamente no endereço marianaguimaraes.site/*estado*.pkl; e salvo na subpasta \Models), e é estimada uma faixa para simulação de valores novos. Além disso é feita uma query no banco com o estado selecionado com a quantidade definida de valores aleatórios.

```
#Carrega modelo
state = mynumber.get()
pkl_filename = '.\\Models\\' + state + ".pkl"
with open(pkl_filename, 'rb') as file:
    model = pickle.load(file)
```

```
#Seleciona numero de registros pra simulação
teste_range = int(namostras.get())
```

```
#Busca base dos dados no bando de dados MySql
query = "SELECT * FROM us_accident_TableTest_"+state+ " ORDER BY RAND() LIMIT  " + str(teste_range)
acc = pd.read_sql(query, con=db_connection)
acidentes_state = acc
```

São então preparadas as tabelas de teste e de resultados. após isso, são estipulados os valores aleatórios de duração, temperatura e distância com base na faixa de valores vista no dataframe original, e após isso é feita a predição da severidade de acordo com esses valores aleatórios postos na tabela.

```
In [9]: #Prepara dataset de teste e resultados
x_force_teste = acidentes_state.iloc[teste_range-1:teste_range]
simulate = pd.DataFrame(np.repeat(x_force_teste.values,teste_range,axis=0))
simulate.columns = x_force_teste.columns
simulate['Start_Lng'] = acidentes_state['Start_Lng']
simulate['Start_Lat'] = acidentes_state['Start_Lat']
x_force_teste = simulate.copy()
simulate['Prev_Result'] = 0
```

```
In [10]: #Simulador de valores randomicos
x_force_teste['Duration_Group'] = np.random.randint(1, 11,size = teste_range)
x_force_teste['TemperatureC_Group']  = np.random.randint(1, 7, size = teste_range)
x_force_teste['Distance_Group'] = np.random.randint(1, 10, size = teste_range)
```

```
In [11]: #Faz as previsoes
y_pred = model.predict(x_force_teste)
#Verifica resultados
y_pred
```

```
Out[11]: array([2, 3, 1, 3, 2, 2, 1, 2, 1, 3, 2, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1,
       1, 2, 2, 3, 2, 2, 1, 2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 3, 2, 3, 3, 2, 3, 2, 2, 2, 2, 2, 1, 2], dtype=int64)
```

É então salva a parte de latitude, longitude e severidade do caso em um novo dataframe, para gerar as visualizações

```
: x_force_teste['Severity'] = y_pred
```

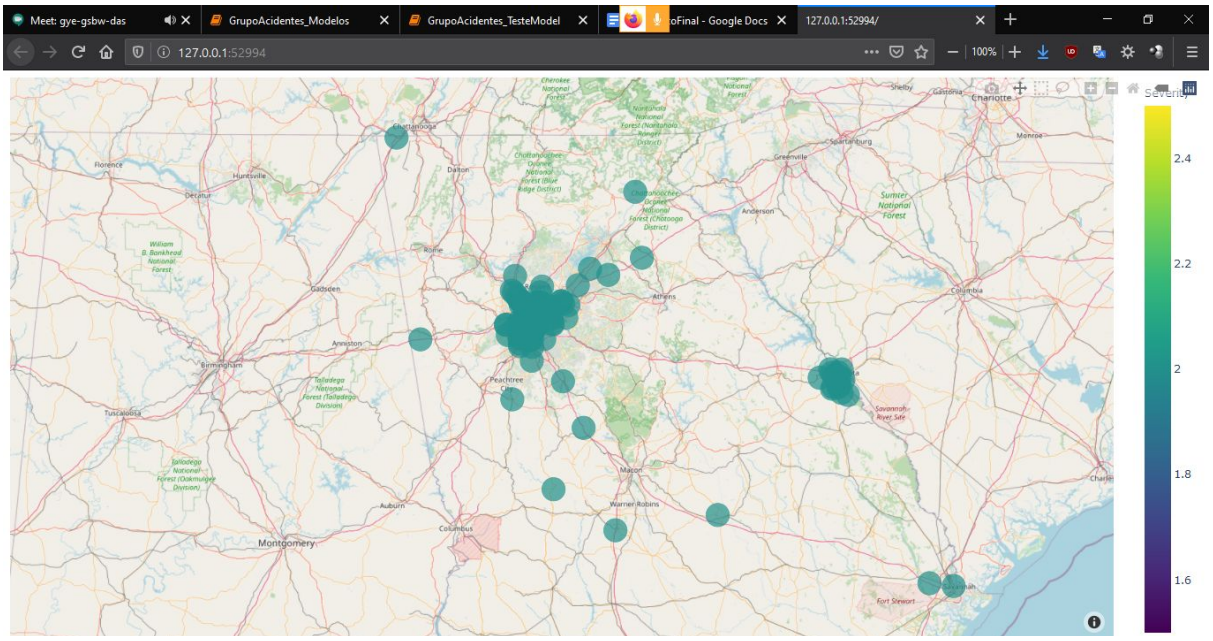
```
: x_result = x_force_teste[['Start_Lat','Start_Lng','Severity']]
x_result.sample(5)
```

:

	Start_Lat	Start_Lng	Severity
21	34.059410	-83.566805	1
1	33.722965	-84.502815	3
20	33.703450	-84.166910	2
46	33.766376	-84.527321	2
17	33.821548	-84.359383	2

Por fim, utilizando o pacote plotly para implementar um mapa indicando a localização dos acidentes e sua severidade de acordo com a cor e tamanho do marcador, em uma nova aba do navegador.

```
fig = px.scatter_mapbox(x_result,
                        lat = 'Start_Lat', lon = 'Start_Lng',
                        color = 'Severity', size = 'Severity',
                       color_continuous_scale=px.colors.sequential.Viridis
                       )
fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



O notebook pode ser convertido em um arquivo .py, que pede somente o estado a ser estudado e a quantidade de amostras a serem utilizadas, dando como resultado final um mapa semelhante ao já demonstrado.