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

Vitor Santos CaÇula

ESTUDO DE CASOS DE INFRAÇÕES E ACIDENTES DE TRÂNSITO EM RODOVIAS FEDERAIS DE SÃO PAULO

São Paulo – SP
2024

ESTUDO DE CASOS DE INFRAÇÕES E ACIDENTES DE TRÂNSITO EM RODOVIAS FEDERAIS DE SÃO PAULO

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

São Paulo- SP

2024

SUMÁRIO

1. Introdução

O transporte rodoviário tem um papel muito importante em todo o mundo, no Brasil, além de interligar a população, o papel econômico é essencial, visto que as vias rodoviárias são muito utilizadas para transportes de cargas que abastecem todos os estados do país.

Mas, devido tantas locomoções, é comum haver ocorrências de infrações e acidentes de trânsito, que acabam deixando as BRs cada vez mais perigosas, havendo vários mortos e feridos.

Entendendo o comportamento das infrações e acidentes diante as principais rodovias do país, pode-se auxiliar a liderar campanhas de conscientização da educação no trânsito, descobrir as principais áreas de melhorias no trânsito, aumentando a assertividade das equipes de fiscalização em momentos de maior necessidade. Diante disso, o trabalho irá abordar a segurança nas rodovias federais do estado de São Paulo entre 2020 e 2021.

1.1. Contextualização

Acidentes e infrações de trânsito nas rodovias paulistas são um desafio constante à segurança e mobilidade viária do estado. São Paulo possui uma das maiores redes rodoviárias do Brasil, tornando o trânsito tenso e complicado em muitas áreas.

Infelizmente, a combinação destes fatores conduz frequentemente a acidentes e infrações, afetando os condutores, a economia e o sistema de saúde pública.

Primeiramente, é importante ressaltar que as infrações de trânsito ocorrem com frequência nas rodovias paulistas. Isso inclui excesso de velocidade, ultrapassagens perigosas, uso de celular ao dirigir, dirigir sob efeito de álcool ou drogas, entre outros comportamentos imprudentes. As Autoridades de trânsito têm feito esforços para combater estas violações através de verificações mais rigorosas e de sensibilização, mas o problema persiste.

Em relação aos acidentes, as rodovias paulistas registram muitos acidentes, dos quais causam ferimentos graves ou até mortes. Existem muitas causas diferentes, incluindo fatores como velocidade excessiva, desatenção do motorista e más

condições climáticas.

O governo do estado de São Paulo tem tomado medidas para melhorar a segurança nas rodovias, como alargamento e modernização de trechos de estradas, instalação de radares de controle de velocidade e promoção de campanhas educativas. Além disso, investem em atividades de sensibilização para incentivar um comportamento mais responsável na estrada. Contudo, reduzir acidentes e infrações nas rodovias de São Paulo é um desafio constante que requer a cooperação de todos os envolvidos, desde motoristas até órgãos reguladores e autoridades.

1.2. O problema proposto

Por que esse problema é importante?

O estudo de casos de acidentes e infrações nas rodovias é de extrema importância devido à necessidade de aprimorar a segurança viária e econômica do país. Essas ocorrências não só causam tragédias humanas, mas também têm impactos econômicos significativos devido a custos médicos, danos a veículos e perturbações na circulação de mercadorias.

De quem são os dados analisados?

Os dados analisados neste estudo foram obtidos a partir do site do governo federal, que disponibiliza informações sobre as ocorrências de acidentes e infrações em rodovias federais, e a fonte de tabela de infrações disponibilizada no site do DETRAN. Esses dados são coletados e registrados por órgãos competentes, como a Polícia Rodoviária Federal (PRF), e o Departamento Estadual de Transito (DETRAN).

Quais os objetivos com essa análise? O que iremos analisar?

O objetivo da análise é entender o comportamento das ocorrências e auxiliar a liderar campanhas de conscientização da educação no trânsito.
Descobrir as principais áreas de melhorias no trânsito, aumentando a assertividade das equipes de fiscalização em momentos de maior necessidade.

Trata-se de aspectos geográficos e logísticos de sua análise?

A análise será baseada em dados de ocorrências de acidentes e infrações ocorridos em diferentes regiões geográficas de São Paulo. Essa análise permitirá uma compreensão mais abrangente das ocorrências e uma identificação das particularidades relacionadas a cada rodovia (BRs).

Qual o período está sendo analisado?

O período analisado compreende os anos de 2020 a 2021.

1.3. Objetivos

O objetivo do trabalho é saber o desempenho das ocorrências nas rodovias, períodos do ano, horário do dia e gravidade dos eventos. Conhecendo este comportamento, podem ser implementadas medidas preventivas, tornando as inspeções mais eficazes e ajudando a reduzir os riscos na estrada. Para alcançar esse objetivo geral, serão estabelecidos os seguintes objetivos específicos:

1. Realizar uma análise aprofundada dos períodos com maiores ocorrências de acidentes e infrações, assim podendo destacar meses, dias da semana e períodos do dia mais propensos a ter casos de ocorrências.
2. Investigar a severidade das ocorrências, e podermos ter visibilidade em relação a periculosidade nas estradas.
3. Observar quais são as ocorrências de acidentes e infrações com

maiores números de casos.

4. Analisar as rodovias com maiores números de ocorrências de acidentes e infrações.
5. Visualizar a distribuição geográfica das ocorrências dos acidentes nas rodovias de São Paulo.

Por meio desses objetivos, conseguimos entender o comportamento das ocorrências e obter diversas respostas que auxiliaram a tomada de decisão para a segurança viária de São Paulo.

2. Coleta de Dados

Para realizar a coleta de análise dos dados foi utilizado a linguagem de programação Python versão 3.11.5, e a IDE (Integrated Development Environment) utilizada foi Jupyter Notebook versão 8.15.0

```
print("Versão do Jupyter Noyebook:", IPython.__version__)
print("Versão do Python:", sys.version)
```

```
Versão do Jupyter Noyebook: 8.15.0
Versão do Python: 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)]
```

Figura 1: Coleta de Dados/ Verificando a versão do Jupyter e Python.

Para obter os dados de Acidentes em rodovias federais para estudo, foi utilizado dados fornecidos pelo governo, no seguinte link:
https://www.gov.br/prf/pt-br/acesso-a-informacao/dados-abertos/dados-abertos-acidentes

No site os Datasets utilizados são:
“Agrupados por ocorrência (detran2020)”
“Agrupados por ocorrência (detran2021)”

Nome	Tipo	Descrição
id	float64	Variável com valores numéricos, representando o identificador do acidente.
data_inversa	object	Data da ocorrência no formato dd/mm/aaaa.
dia_semana	object	Dia da semana da ocorrência. Ex.: Segunda, Terça, etc.
horario	object	Horário da ocorrência no formato hh:mm:ss.
uf	object	Unidade da Federação. Ex.: MG, PE, DF, etc.
br	float64	Variável com valores numéricos representando o identificador da BR do
km	object	Valor representante do Kilômetro da BR do acidente.
municipio	object	Nome do município de ocorrência do acidente.
causa_acidente	object	causa_acidente Identificação da causa presumível do acidente. Ex.: Falta de
tipo_acidente	object	tipo_acidente Identificação do tipo de acidente. Ex.: Colisão frontal, Saída de
classificacao_acidente	object	classificação_acidente Classificação quanto à gravidade do acidente: Sem
fase_dia	object	Fase do dia no momento do acidente. Ex. Amanhecer, Pleno dia
sentido_via	object	Sentido da via considerando o ponto de colisão: Crescente e
condicao_metereologica	object	Condição meteorológica no momento do acidente: Céu claro,
tipo_pista	object	Tipo da pista considerando a quantidade de faixas: Dupla,simples ou múltipla.
tracado_via	object	Descrição do traçado da via: reta, curva ou cruzamento.
uso_solo	object	Descrição sobre as características do local do acidente:Urbano ou rural.
pessoas	int64	Total de pessoas envolvidas na ocorrência.
mortos	int64	Total de pessoas mortas envolvidas na ocorrência.
feridos_leves	int64	Total de pessoas com ferimentos leves envolvidas na ocorrência.
feridos_graves	int64	Total de pessoas com ferimentos graves envolvidas na ocorrência.
ilesos	int64	Total de pessoas ilesas envolvidas na ocorrência.
ignorados	int64	Total de pessoas envolvidas na ocorrência e que não se soube o estado físico.
feridos	int64	Total de pessoas feridas envolvidas na ocorrência (é a soma dos feridos leves
veiculos	int64	Total de veículos envolvidos na ocorrência.
latitude	object	Latitude do local do acidente.
longitude	object	Longitude do local do acidente.

Figura 2: Coleta de Dados/ Dicionário de dados Acidentes .

Figura 3: Coleta de Dados/ Fonte de dados Acidentes.

Para obter os dados de infrações para estudo, foi utilizado dados fornecidos pelo governo, no seguinte link:

https://www.gov.br/prf/pt-br/acesso-a-informacao/dados-abertos/dados-abertos-infracoes

No site os Datasets utilizados são:

“Infracoes_2020”

“Infracoes_2021”

Nome	Tipo	Descrição
Número do Auto	object	Variável com valores numéricos, representando o identificador do acidente.
Data da Infração (DD/MM/AAAA)	object	Data da ocorrência no formato dd/mm/aaaa.
Indicador de Abordagem	object	Identifica se houve abordagem do veículo: C (houve abordagem), S (não houve abordagem).
Assinatura do Auto	object	Variável que informa se o infrator assinou o auto de infracao. S (sim), Vazio (não).
Indicador Veiculo Estrangeiro	object	Indicador Veiculo Estrangeiro
Sentido Trafego	object	Sentido da via considerando o ponto de colisão: Crescente e decrescente.
UF Placa	object	Unidade da Placa Ex.: MG, PE, DF, etc.
UF Infração	object	Unidade da Infração Ex.: MG, PE, DF, etc.
BR Infração	int64	Variável com valores numéricos representando o identificador da BR do acidente.
Km Infração	int64	Identificação do quilômetro onde ocorreu o acidente, com valor mínimo de 0,1 km e com a casa decimal separada por ponto.
Município	object	Nome do município de ocorrência do acidente.
Código da Infração	int64	Variavel informa o código da infração
Descrição Abreviada Infração	object	Descrição Abreviada Infração
Enquadramento da Infração	object	Enquadramento da Infração
Início Vigência da Infração	object	Data de inicio de vigência no formato dd/mm/aaaa.
Fim Vigência Infração	object	Data de fim de vigência no formato dd/mm/aaaa.
Medição Infração	object	Registro da medição realizada em radares, etilômetros, balanças e trenas.
Descrição Especie Veículo	object	Descrição da Especie Veículo
Descrição Marca Veículo	object	Descrição da marca Veículo
Hora Infração	int64	Hora de ocorrência da infração
Medição Considerada	object	Medição Considerada para o registro da infração
Excesso Verificado	object	Excesso verificado nas infrações onde são utilizados
Qtd Infrações	int64	Quantidade de infrações

Figura 4: Coleta de Dados/ Dicionário de dados Infrações .

Figura 5: Coleta de Dados/ Onde são encontradas as bases.

Para obter os dados de tipos de infrações para estudo, foi utilizado dados fornecidos pelo governo, no seguinte link:

https://www.detran.sp.gov.br/wps/portal/portaldetran/cidadao/infracoes/servicos/consultaTabelaInfracoes

Nome	Tipo	Descrição
Código da Infração	int64	Variavel informa o código da infração
Infracao	object	Descrição da Infração
Gravidade	object	Gravidade da Infração
Responsavel	object	Responsavel pela Infração cometida

Figura 6: Coleta de Dados/ Dicionário de dados Tipos infrações.

Figura 7: Coleta de dados / Onde são encontradas as bases.

3. Processamento/Tratamento de Dados

Nesse projeto, as etapas para processamento de dados serão:

Importação das bibliotecas

Foram importadas as seguintes bibliotecas para realização dos processamentos e tratamentos necessários:

Importação das bibliotecas

```
import pandas as pd
import numpy as np
import seaborn as sns
from datetime import date
import plotly.graph_objs as go
```

Figura 8: Processamento Tratamento de dados / Importação de bibliotecas.

Obtenção dos dados

Para obtenção dos dados de infrações foi necessário realizar a importação separada por mês, devido a base ser segmentada mensalmente.

Em seguida, para as bases de Infrações e Acidentes de 2020 à 2021, realizamos a concatenação das bases de forma Anual:

```
#LENDO BASE DE ACIDENTES DE 2020 À 2021
DFacidentes2020 = pd.read_csv("datatran2020.csv",sep =";",encoding='cp1252',low_memory=False)
DFacidentes2021 = pd.read_csv("datatran2021.csv",sep =";",encoding='cp1252',low_memory=False)

#LENDO BASE DE INFRAÇÕES DE 2020 À 2021

DfInfracoes2020Jan = pd.read_csv("infracoes_2020_01.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Fev = pd.read_csv("infracoes_2020_02.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Mar = pd.read_csv("infracoes_2020_03.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Abr = pd.read_csv("infracoes_2020_04.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Mai = pd.read_csv("infracoes_2020_05.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Jun = pd.read_csv("infracoes_2020_06.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Jul = pd.read_csv("infracoes_2020_07.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Ago = pd.read_csv("infracoes_2020_08.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Set = pd.read_csv("infracoes_2020_09.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Out = pd.read_csv("infracoes_2020_10.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Nov = pd.read_csv("infraçoes_2020_11.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2020Dez = pd.read_csv("infraçoes_2020_12.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)

DfInfracoes2021Jan = pd.read_csv("infraçoes2021_01.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Fev = pd.read_csv("infraçoes2021_02.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Mar = pd.read_csv("infraçoes2021_03.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Abr = pd.read_csv("infraçoes2021_04.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Mai = pd.read_csv("infraçoes2021_05.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Jun = pd.read_csv("infraçoes2021_06.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Jul = pd.read_csv("infraçoes2021_07.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Ago = pd.read_csv("infraçoes2021_08.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Set = pd.read_csv("infraçoes2021_09.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Out = pd.read_csv("infraçoes2021_10.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Nov = pd.read_csv("infraçoes2021_11.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
DfInfracoes2021Dez = pd.read_csv("infraçoes2021_12.csv",sep =";",decimal = '.',encoding='cp1252',low_memory=False)
```

Figura 9: Processamento Tratamento de dados / Obtenção dos dados de Acidentes e Infrações.

Concatenando bases:

```
DfAcidentes = pd.concat([DFacidentes2020,DFacidentes2021])

DfInfracoes2020 = pd.concat([DfInfracoes2020Jan,DfInfracoes2020Fev,DfInfracoes2020Mar,DfInfracoes2020Abr,DfInfracoes2020Mai,DfInf
                            ,DfInfracoes2020Jul,DfInfracoes2020Ago,DfInfracoes2020Set,DfInfracoes2020Out,DfInfracoes2020Nov,DfInf

DfInfracoes2021 = pd.concat([DfInfracoes2021Jan,DfInfracoes2021Fev,DfInfracoes2021Mar,DfInfracoes2021Abr,DfInfracoes2021Mai,DfInf
                            ,DfInfracoes2021Jul,DfInfracoes2021Ago,DfInfracoes2021Set,DfInfracoes2021Out,DfInfracoes2021Nov,DfInf
```

Figura 10: Processamento Tratamento de dados / Concatenando as bases de Acidentes e Infrações.

3.1 Tratando Base de acidentes:

Excluindo colunas desnecessárias

Para as bases de Infrações selecionamos apenas as colunas que serão necessárias para o estudo, e para a base de acidentes realizamos a exclusão de algumas colunas desnecessárias:

```
#Deletando colunas
del DfAcidentes['regional']
del DfAcidentes['delegacia']
del DfAcidentes['uop']
```

Figura 11: Processamento Tratamento de dados / Excluindo colunas desnecessárias.

Filtrando os Datasets com o Estado de foco do estudo

Para o nosso estudo será necessário filtrar as bases de acidentes com o estado de São Paulo, que é o foco da nossa análise:

Filtrando o Datasets com a Uf de foco do nosso estudo (SP)

```
estado =['SP']
DFacidentes2020_SP = DFacidentes2020[DFacidentes2020['uf'].isin(estado)]
DFacidentes2021_SP = DFacidentes2021[DFacidentes2021['uf'].isin(estado)]
```

Figura 12: Processamento Tratamento de dados / Filtrando UF.

Verificando Valores ausentes nos datasets:

Identificamos pouca quantidade de valores nulos nas colunas br, km e uop na base de acidentes. Iremos realizar a limpeza do dataset excluindo esses valores. Já para o dataset de infrações não foi identificado valores nulos.

```
#Representação de valores ausentes
(DFacidentes2021_SP.isnull().sum()/DFacidentes2021_SP.shape[0]).sort_values(ascending = False)
```

```
br                      0.001860
km                      0.001860
uop                     0.000233
uso_solo                0.000000
delegacia               0.000000
regional                0.000000
longitude               0.000000
latitude                0.000000
veiculos                0.000000
feridos                 0.000000
ignorados               0.000000
ilesos                  0.000000
feridos_graves          0.000000
feridos_leves           0.000000
mortos                  0.000000
pessoas                 0.000000
id                      0.000000
data_inversa            0.000000
tipo_pista              0.000000
condicao_metereologica  0.000000
sentido_via             0.000000
fase_dia                0.000000
classificacao_acidente  0.000000
tipo_acidente           0.000000
causa_acidente          0.000000
municipio               0.000000
```

Figura 13: Processamento Tratamento de dados / verificando valores ausentes.

Eliminando Valores ausentes nos datasets:

Limpando os valores ausentes com a função `dropna`

```
DFacidentes2020_SP_Clean = DFacidentes2020_SP.dropna()
DFacidentes2021_SP_Clean = DFacidentes2021_SP.dropna()
```

Figura 15: Processamento Tratamento de dados / excluindo calores ausentes.

Tratando coluna de horário do Dataset de acidentes:

Iremos realizar o tratamento da coluna de hora do Dataset de Acidentes para ser compatível com a coluna de horário do Dataset de Infrações, isso facilitara análises futuras.

```
DFacidentes2020_SP_Clean['Hora'] = DFacidentes2020_SP_Clean['horario'].str.split(':').str[0]#tratando coluna de horario para o f
DFacidentes2021_SP_Clean['Hora'] = DFacidentes2021_SP_Clean['horario'].str.split(':').str[0]
#Deletando coluna antiga
del DFacidentes2021_SP_Clean['horario']
del DFacidentes2020_SP_Clean['horario']
```

Figura 16: Processamento Tratamento de dados / Tratando coluna de horário.

Concatenando Base de acidentes:

```
DFacidentesSP = pd.concat([DFacidentes2020_SP_Clean,DFacidentes2021_SP_Clean])
```

Figura 17: Processamento Tratamento de dados / concatenando bases.

Removendo caracteres de .0 da coluna BR da base de acidentes:

Para melhor visualização dos dados de BR foi necessário o tratamento da coluna.

```
DFacidentesSP['br'] = DFacidentesSP['br'].astype('str').str.replace('.0','')
```

Figura 18: Processamento Tratamento de dados / tratando coluna.

Convertendo coluna de horas da base de acidentes para o tipo Int:

```
DFacidentesSP['Hora'] = pd.to_numeric(DFacidentesSP['Hora'])
```

Figura 19: Processamento Tratamento de dados / tratando coluna.

Criando coluna de Período do dia para base de Acidentes:

Criando função que ira percorrer a linha de horas do Dataset e definir um período.

```
def f(Hora):
    if Hora >= 0 and Hora <= 5:
        return 'Madrugada'
    elif Hora >= 6 and Hora <= 11:
        return 'Manhã'
    elif Hora >= 12 and Hora <= 17:
        return 'Tarde'
    else:
        return "Noite"
```

Figura 20: Processamento Tratamento de dados / tratando coluna.

Criando coluna de Período através da aplicação da função Período:

```
DFacidentesSP['Periodo'] = DFacidentesSP['Hora'].apply(lambda Hora: f(Hora))
```

Figura 21: Processamento Tratamento de dados / tratando coluna.

Criando coluna com Mês/Ano para base de Acidentes:

Convertendo coluna para o tipo Data.

```
DFacidentesSP["data_inversa"] = pd.to_datetime(DFacidentesSP["data_inversa"])
```

Figura 22: Processamento Tratamento de dados / tratando coluna.

Criando função que ira percorrer a linha de Data do Dataset e definir Mês e Ano.

```
def f(Data):
    Data = Data.strftime("%b %Y")
    return Data
```

Figura 23: Processamento Tratamento de dados / tratando coluna.

Criando coluna de AnoMes através da aplicação da função AnoMes.

```
DFacidentesSP['AnoMes'] = DFacidentesSP['data_inversa'].apply(lambda Data: f(Data))
```

Figura 24: Processamento Tratamento de dados / tratando coluna.

Criando coluna com Dia da Semana para base de Acidentes:

Através da função WeekDay da biblioteca Datetime conseguimos definir o dia da semana para as datas da nossa base, utilizaremos o Loop for para percorrer os registros.

```
Dia_da_semana = []

for i in DFacidentesSP['data_inversa']:
  Dia_da_semana.append(i.weekday())

DFacidentesSP['Dia_da_semana'] = Dia_da_semana

DFacidentesSP.loc[DFacidentesSP['Dia_da_semana'] == 0, 'Dia'] = "Segunda-feira"
DFacidentesSP.loc[DFacidentesSP['Dia_da_semana'] == 1, 'Dia'] = "Terca-feira"
DFacidentesSP.loc[DFacidentesSP['Dia_da_semana'] == 2, 'Dia'] = "Quarta-feira"
DFacidentesSP.loc[DFacidentesSP['Dia_da_semana'] == 3, 'Dia'] = "Quinta-feira"
DFacidentesSP.loc[DFacidentesSP['Dia_da_semana'] == 4, 'Dia'] = "Sexta-feira"
DFacidentesSP.loc[DFacidentesSP['Dia_da_semana'] == 5, 'Dia'] = "Sabado"
DFacidentesSP.loc[DFacidentesSP['Dia_da_semana'] == 6, 'Dia'] = "Domingo"
```

Figura 25: Processamento Tratamento de dados / tratando coluna.

3.2 Tratando Base de Infrações:

Excluindo colunas desnecessárias

Para as bases de Infrações selecionamos apenas as colunas que serão necessárias para o estudo, e para a base de acidentes realizamos a exclusão de algumas colunas desnecessárias:

```
Infracoes2020 = DfInfracoes2020[['Data da Infração (DD/MM/AAAA)','UF Infração','BR Infração','Código da Infração','Descrição Abre
Infracoes2021 = DfInfracoes2021[['Data da Infração (DD/MM/AAAA)','UF Infração','BR Infração','Código da Infração','Descrição Abre
```

Figura 11: Processamento Tratamento de dados / Excluindo colunas desnecessárias.

Concatenando as bases de infrações:

```
DfInfracoesSP = pd.concat([Infracoes2020_SP,Infracoes2021_SP])
```

Figura 26: Processamento Tratamento de dados / Concatenando bases.

Filtrando os Datasets com o Estado de foco do estudo

Para o nosso estudo será necessário filtrar as bases de infrações com o estado de São Paulo, que é o foco da nossa análise:

Filtrando o Datasets com a Uf de foco do nosso estudo (SP)

```
estado =['SP']
Infracoes2020_SP = Infracoes2020[Infracoes2020['UF Infração'].isin(estado)]
Infracoes2021_SP = Infracoes2021[Infracoes2021['UF Infração'].isin(estado)]
```

Figura 27: Processamento Tratamento de dados / Filtrando UF.

Verificando Valores ausentes nos datasets:

```
(Infracoes2021_SP.isnull().sum()/Infracoes2021_SP.shape[0]).sort_values(ascending = False)
```

```
Data da Infração (DD/MM/AAAA)    0.0
UF Infração                      0.0
BR Infração                      0.0
Código da Infração               0.0
Descrição Abreviada Infração     0.0
Hora Infração                    0.0
dtype: float64
```

```
(Infracoes2020_SP.isnull().sum()/Infracoes2020_SP.shape[0]).sort_values(ascending = False)
```

```
Data da Infração (DD/MM/AAAA)    0.0
UF Infração                      0.0
BR Infração                      0.0
Código da Infração               0.0
Descrição Abreviada Infração     0.0
Hora Infração                    0.0
dtype: float64
```

Figura 28: Processamento Tratamento de dados / verificando valores ausentes.

Criando coluna de Período do dia para base de Infrações:

```
#Função para atribuir o periodo em seus respectivos horários
def f(Hora):
    if Hora >= 0 and Hora <= 5:
        return 'Madrugada'
    elif Hora >= 6 and Hora <= 11:
        return 'Manhã'
    elif Hora >= 12 and Hora <= 17:
        return 'Tarde'
    else:
        return "Noite"
```

Figura 29: Processamento Tratamento de dados / tratamento coluna.

Aplicando função para a criação da coluna de período.

```
DfInfracoesSP['Periodo'] = DfInfracoesSP['Hora Infração'].apply(lambda Hora: f(Hora))
```

Figura 30: Processamento Tratamento de dados / tratamento coluna.

Criando coluna com Mês/Ano para base de Infrações:

Convertendo para coluna data

```
#Convertendo para colna data
DfInfracoesSP["Data da Infração (DD/MM/AAAA)"] = pd.to_datetime(DfInfracoesSP["Data da Infração (DD/MM/AAAA)"])
```

Figura 31: Processamento Tratamento de dados / tratamento coluna.

Função para atribuir o formato de AnoMes:

```
#Função para atribuir o formato de AnoMes
def f(Data):
    Data = Data.strftime("%b %Y")
    return Data
```

Figura 32: Processamento Tratamento de dados / tratamento coluna.

Aplicando a função na coluna

```
#Aplicando a função na coluna
DfInfracoesSP['AnoMes'] = DfInfracoesSP['Data da Infração (DD/MM/AAAA)'].apply(lambda Data: f(Data))
```

Figura 33: Processamento Tratamento de dados / tratamento coluna.

Criando coluna com Dia da Semana para base de Infrações

Através da função WeekDay da biblioteca Datetime conseguimos definir o dia da semana para as datas da nossa base, utilizaremos o Loop For para percorrer os registros.

```
Dia_da_semana = []

for i in DfInfracoesSP['Data da Infração (DD/MM/AAAA)']:
  Dia_da_semana.append(i.weekday())

DfInfracoesSP['Dia_da_semana'] = Dia_da_semana

DfInfracoesSP.loc[DfInfracoesSP['Dia_da_semana'] == 0, 'Dia'] = "Segunda-feira"
DfInfracoesSP.loc[DfInfracoesSP['Dia_da_semana'] == 1, 'Dia'] = "Terca-feira"
DfInfracoesSP.loc[DfInfracoesSP['Dia_da_semana'] == 2, 'Dia'] = "Quarta-feira"
DfInfracoesSP.loc[DfInfracoesSP['Dia_da_semana'] == 3, 'Dia'] = "Quinta-feira"
DfInfracoesSP.loc[DfInfracoesSP['Dia_da_semana'] == 4, 'Dia'] = "Sexta-feira"
DfInfracoesSP.loc[DfInfracoesSP['Dia_da_semana'] == 5, 'Dia'] = "Sabado"
DfInfracoesSP.loc[DfInfracoesSP['Dia_da_semana'] == 6, 'Dia'] = "Domingo"
```

Figura 34: Processamento Tratamento de dados / tratamento coluna.

3.3 Enriquecendo o Dataset com a base de tipos de infrações.

O dataset de tipos de infrações irá enriquecer as nossas bases de infrações trazendo uma descrição mais objetiva sobre a infração, e o tipo de gravidade. Fundamental para a etapa de análise de dados

Lendo base de Tipos de infrações:

```
TiposInfracoes = pd.read_csv("TiposInfracoes4.csv", sep =";",encoding='ISO-8859-1')
```

Figura 35: Processamento Tratamento de dados / Enriquecendo dataset.

Renomeando colunas:

```
TiposInfracoes.columns = ['Código da Infração','Infracao','artigo','Gravidade','Responsavel','tipo','pontos','valor']
```

Figura 36: Processamento Tratamento de dados / Enriquecendo dataset.

Excluindo colunas desnecessárias

```
del TiposInfracoes['artigo']
del TiposInfracoes['tipo']
del TiposInfracoes['pontos']
del TiposInfracoes['valor']
```

Figura 37: Processamento Tratamento de dados / Enriquecendo dataset.

Cruzando a base de infrações com os tipos de Infrações

Para cruzarmos as duas bases utilizamos as duas colunas em comum entre os dois datasets, que nesse caso será a Código de Infração.

```
DfInfracoesSPa = pd.merge(DfInfracoesSP,TiposInfracoes, on = 'Código da Infração', how = 'left')
```

Figura 38: Processamento Tratamento de dados / Enriquecendo dataset.

3.4 Tratando Outliers

Acidentes:

Através do Histograma é possível notar indícios de presenças de outliers. Por exemplo a variável pessoas, mortos, veículos, feridos, leves, feridos, graves, ilesos, ignorados, feridos da base de Acidentes.

Os valores não seguem uma distribuição, e distorcem toda a representação gráfica. Para confirmar, há duas maneiras rápidas que auxiliam a detecção de outliers. São elas:

- Resumo estatístico por meio do método describe().
- Plotar boxplots para a variável.

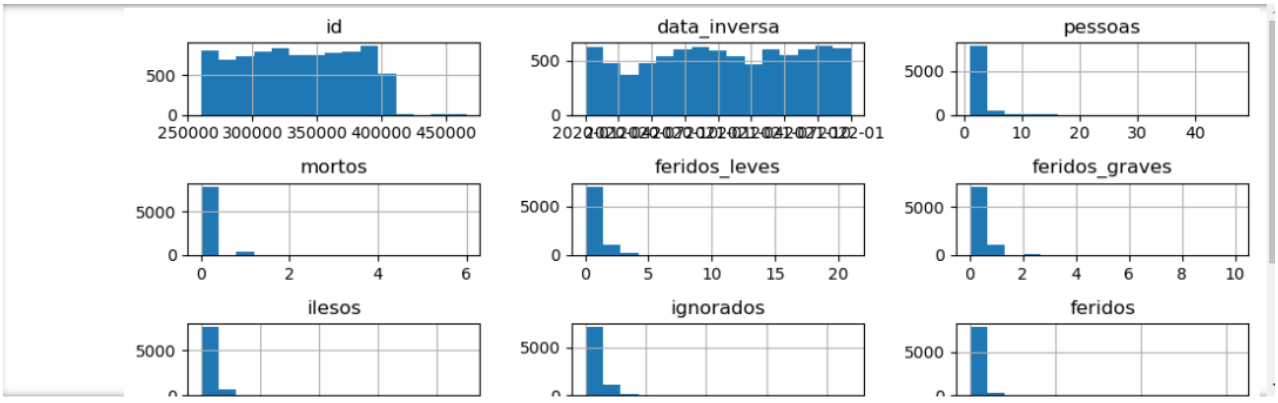


Figura 39: Processamento Tratamento de dados / Outliers.

```
DFacidentesSP[['pessoas','mortos','veiculos','feridos_leves','feridos_graves','ilesos','ignorados','feridos']].describe()
```

	pessoas	mortos	veiculos	feridos_leves	feridos_graves	ilesos	ignorados	feridos
count	8322.000000	8322.000000	8322.000000	8322.000000	8322.000000	8322.000000	8322.000000	8322.000000
mean	2.264840	0.055155	1.644076	0.868061	0.163542	1.016462	0.161620	1.031603
std	1.663939	0.256553	0.813874	0.865913	0.450723	1.391371	0.440911	0.940655
min	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000
50%	2.000000	0.000000	2.000000	1.000000	0.000000	1.000000	0.000000	1.000000
75%	3.000000	0.000000	2.000000	1.000000	0.000000	1.000000	0.000000	1.000000
max	47.000000	6.000000	12.000000	21.000000	10.000000	45.000000	11.000000	31.000000

Figura 40: Processamento Tratamento de dados / Outliers.

```
DFacidentesSP[['pessoas','mortos','veiculos','feridos_leves','feridos_graves','ilesos','ignorados','feridos']].plot(kind='box', v
plt.show()
```

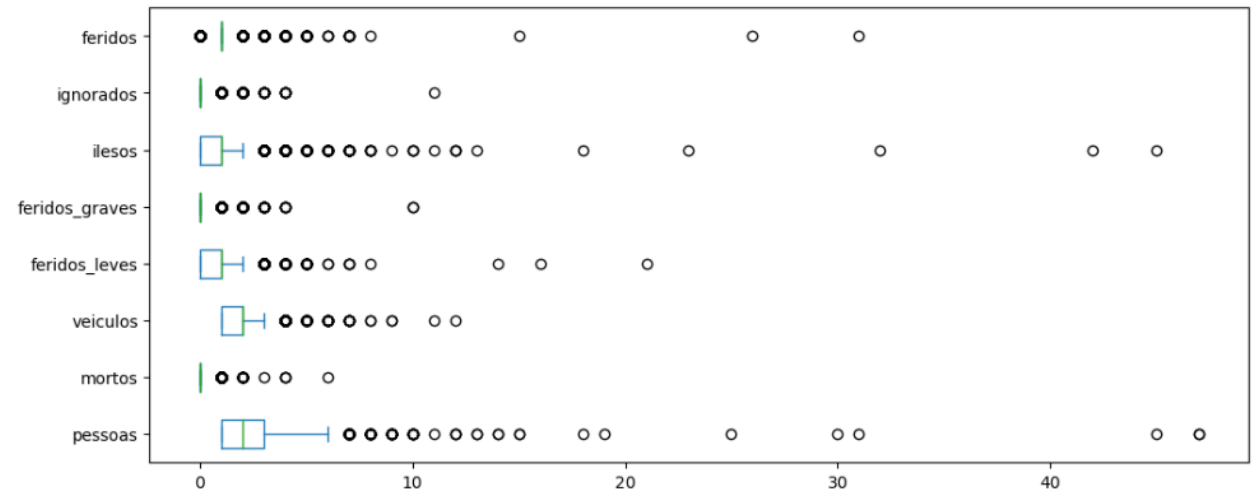


Figura 41: Processamento Tratamento de dados / Outliers.

Olhando o resumo estatístico acima, podemos confirmar que possuímos Outliers em todas as variáveis numéricas de nossa base.

- A variável pessoas por exemplo possui 75% dos valores abaixo de 3, porém seu valor máximo é de 47. Iremos realizar a limpeza desses Outliers com o método de Amplitude Interquartil

- Dessa forma, considerando os critérios do método de amplitude interquartil, nos permite identificar os valores de inferiores e superiores de cada variável, que são considerados outliers nessa amostra.

Remoção de Outliers

Já que identificamos *outliers* nas variáveis, vamos agora limpar do *DataFrame* os valores que ultrapassam os limites e plotar novamente o histograma.

```
Variaveis = ['pessoas','mortos','veiculos','feridos_leves','feridos_graves','ilesos','ignorados','feridos']
for coluna in Variaveis:
    Q1 = np.percentile(DFacidentesSP[coluna],25)
    Q3 = np.percentile(DFacidentesSP[coluna],75)

    #print("Q1 {} = {}".format(Q1,coluna))
    #print("Q3 {} = {}".format(Q3,coluna))

    C = 1.5
    IIQ= Q3-Q1
    LI = Q1 - (C*IIQ)
    LS = Q3 + (C*IIQ)

    print("IIQ {}: {}".format(coluna,IIQ))
    print("Limites: ")
    print("Inferior {}: {}".format(coluna,LI))
    print("Superior {}: {}".format(coluna,LS))
    print("\n")
```

Figura 42: Processamento Tratamento de dados / Outliers.

```
DFacidentesSP_Clean = DFacidentesSP.copy()
for coluna in Variaveis:
    Q1 = np.percentile(DFacidentesSP[coluna],25)
    Q3 = np.percentile(DFacidentesSP[coluna],75)

    C = 1.5
    IIQ= Q3-Q1
    LI = Q1 - (C*IIQ)
    LS = Q3 + (C*IIQ)

    DFacidentesSP_Clean.drop(DFacidentesSP_Clean.loc[(DFacidentesSP_Clean[coluna] < LI) | (DFacidentesSP_Clean[coluna] > LS)].inc

DFacidentesSP_Clean.hist(bins =15,figsize=(10,5))
plt.tight_layout()
```

Figura 43: Processamento Tratamento de dados / Outliers.

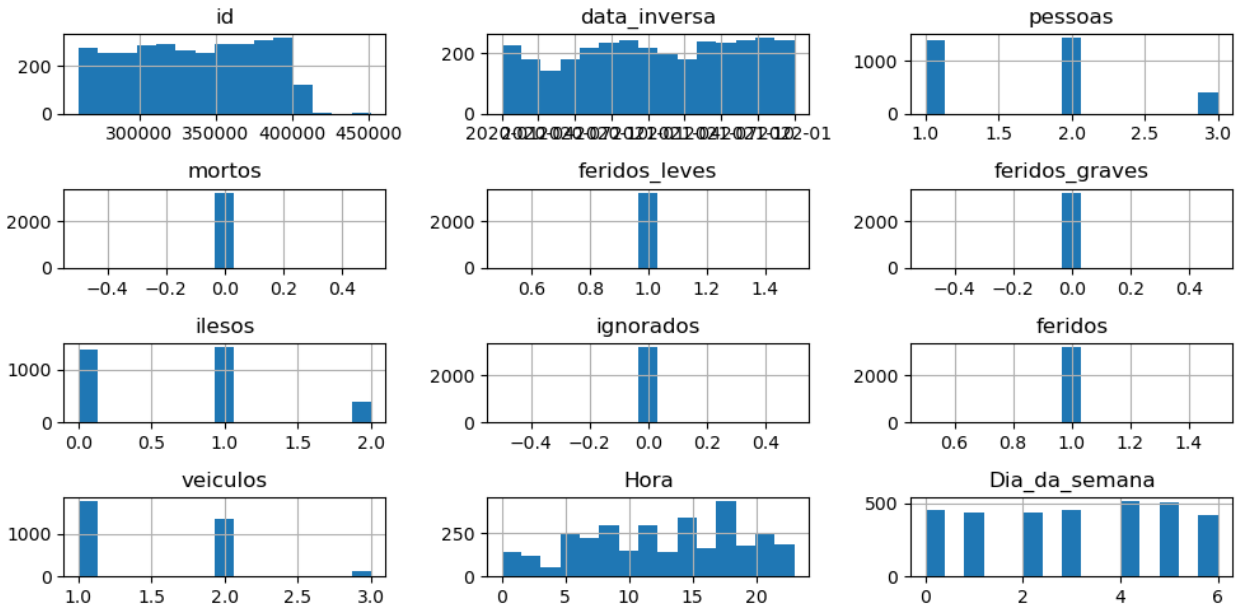


Figura 44: Processamento Tratamento de dados / Outliers.

Após a remoção é possível enxergar uma maior distribuição das varáveis.

```
DFacidentesSP_Clean[['pessoas','mortos','veiculos','feridos_leves','feridos_graves','ilesos','ignorados','feridos']].describe()
```

	pessoas	mortos	veiculos	feridos_leves	feridos_graves	ilesos	ignorados	feridos
count	3228.000000	3228.0	3228.000000	3228.0	3228.0	3228.000000	3228.0	3228.0
mean	1.692069	0.0	1.499690	1.0	0.0	0.692069	0.0	1.0
std	0.677661	0.0	0.580384	0.0	0.0	0.677661	0.0	0.0
min	1.000000	0.0	1.000000	1.0	0.0	0.000000	0.0	1.0
25%	1.000000	0.0	1.000000	1.0	0.0	0.000000	0.0	1.0
50%	2.000000	0.0	1.000000	1.0	0.0	1.000000	0.0	1.0
75%	2.000000	0.0	2.000000	1.0	0.0	1.000000	0.0	1.0
max	3.000000	0.0	3.000000	1.0	0.0	2.000000	0.0	1.0

Figura 45: Processamento Tratamento de dados / Outliers.

```
DFacidentesSP_Clean[['pessoas','mortos','veiculos','feridos_leves','feridos_graves','ilesos','ignorados','feridos']].plot(kind='b
plt.show()
```

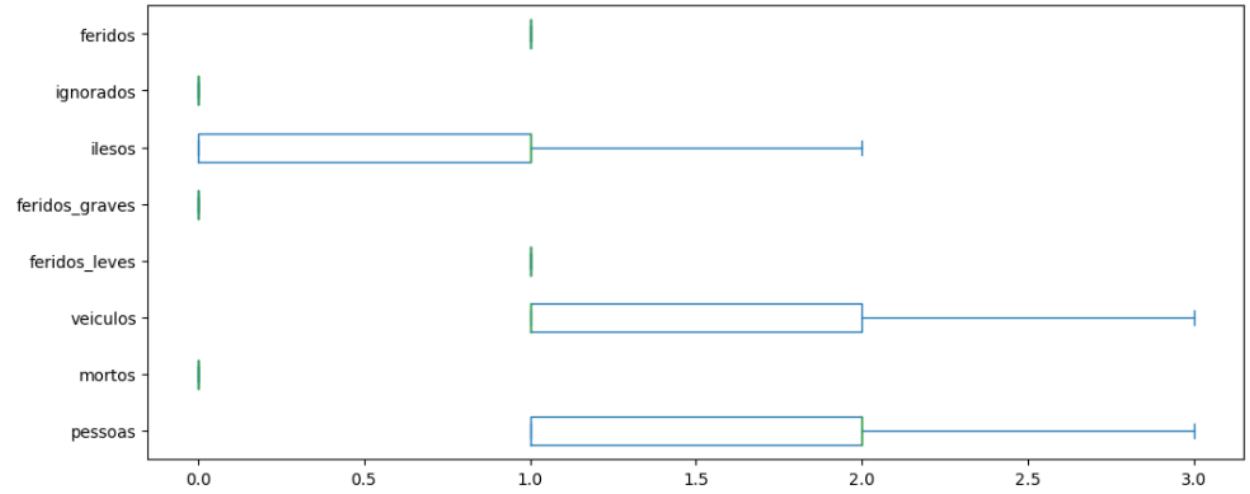


Figura 46: Processamento Tratamento de dados / Outliers.

Infrações:

Através do histograma é possível verificar que não há variáveis quantitativas para a identificação de outliers.

```
DfInfracoesSPa.hist(bins =15,figsize=(10,5))
plt.tight_layout()
```

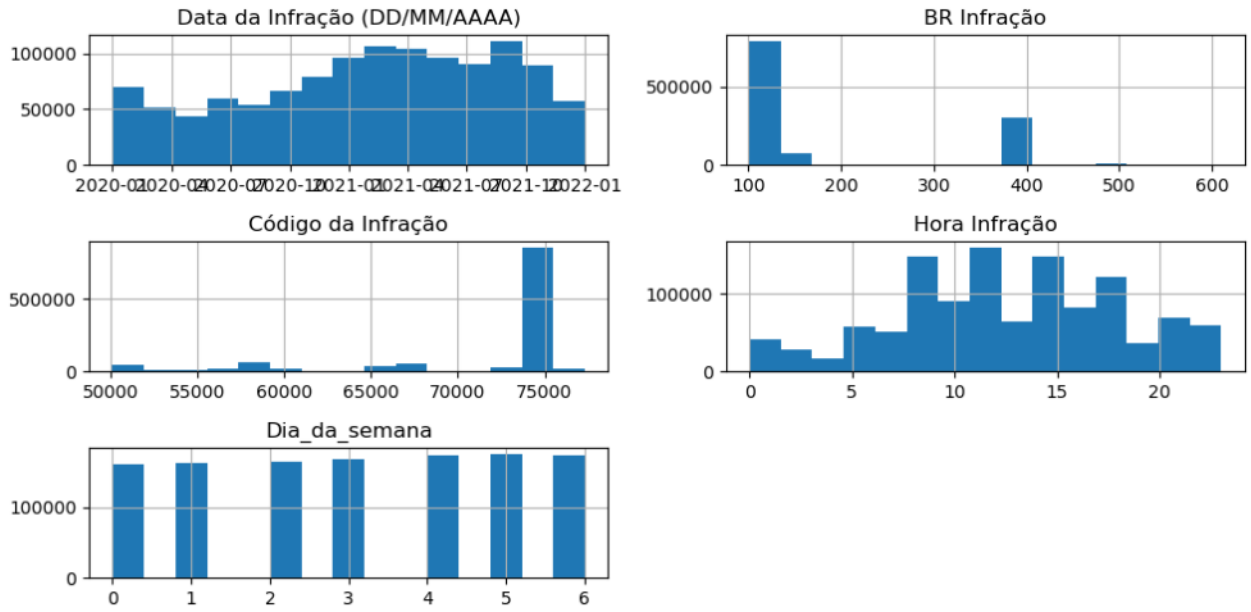


Figura 47: Processamento Tratamento de dados / Outliers.

4. Análise Exploratória

4.1 Infrações

Correlações entre as variáveis:

Correlação significa que existe uma relação entre duas coisas. No nosso contexto, estamos buscando relação ou semelhança entre duas variáveis.

Utilizando o Mapa de calor conseguimos observar que não possuímos fortes correlações referentes às variáveis do dataset.

```
#Criando base com a quantidade de infrações por hora e dia da semana
InfracoesSoma = DfInfracoesSPa[['Dia_da_semana','Hora Infração']].value_counts().reset_index()
InfracoesSoma = pd.DataFrame(InfracoesSoma)
InfracoesSoma.columns =['DiaSemana','Hora','Volume']
```

```
#Checando correlação entre as variáveis da baase criada
InfracoesSomaCorr = InfracoesSoma.corr()
sns.heatmap(InfracoesSomaCorr,cmap = sns.cubehelix_palette(as_cmap=True), fmt='.2f', square=True, linecolor='white', annot=True);
```

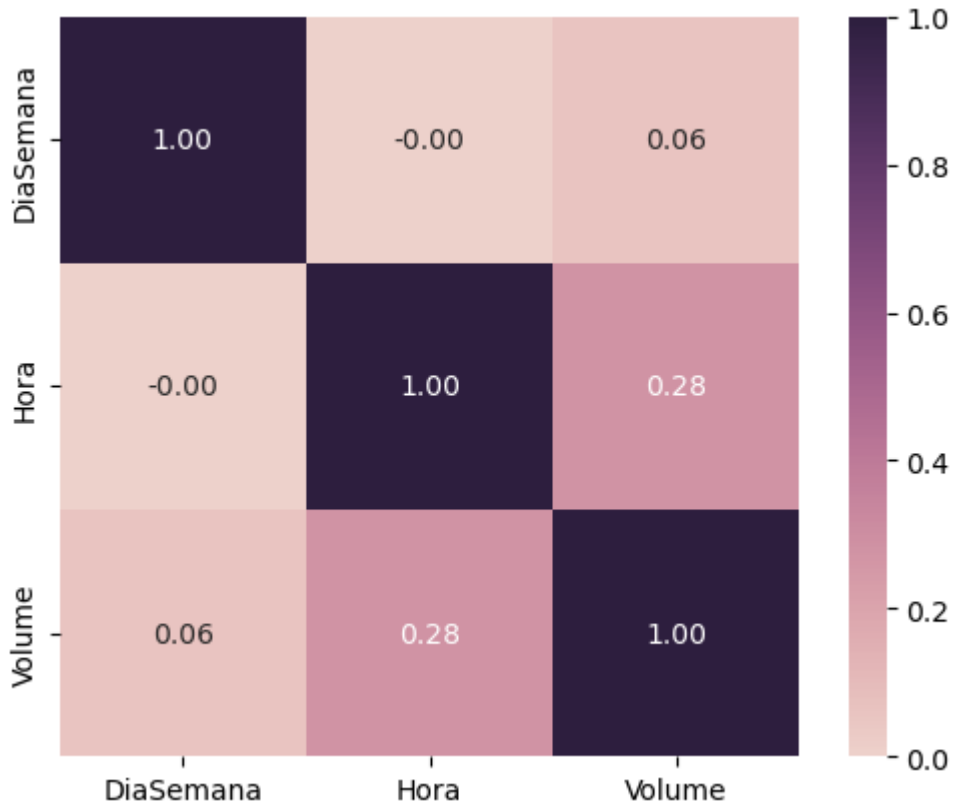


Figura 48: Analise exploratória de dados/ Correlação.

Em quais períodos do ano tivemos o maior índice de infrações?

Vamos analisar o volume de infrações em linha do tempo plotando os gráficos de barras dos dois anos.

```
#Criando dataset
data = DfInfracoesSPa['AnoMes'].value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['AnoMes','Volume']
#Filtrando dataset
data1 = data.loc[data['AnoMes'].str.contains('2020')]
data2 = data.loc[data['AnoMes'].str.contains('2021')]

data = data[['AnoMes','Volume']].groupby('AnoMes').sum().sort_values('Volume',ascending=True)
data1= data1[['AnoMes','Volume']].groupby('AnoMes').sum().sort_values('Volume',ascending=True)
data2 = data2[['AnoMes','Volume']].groupby('AnoMes').sum().sort_values('Volume',ascending=True)

#Plot
data.plot(kind = 'bar', color ='lightblue',figsize=(15,5))

plt.title("Volume Por Ano e Mês")
plt.tight_layout()

data1.plot(kind = 'bar', color ='lightblue',figsize=(15,5))
plt.title("Volume Por Mês - 2020")

data2.plot(kind = 'bar', color ='lightblue',figsize=(15,5))
plt.title("Volume Por Mês - 2021")
plt.tight_layout()
```

Figura 49: Analise exploratória de dados/ Graficos.

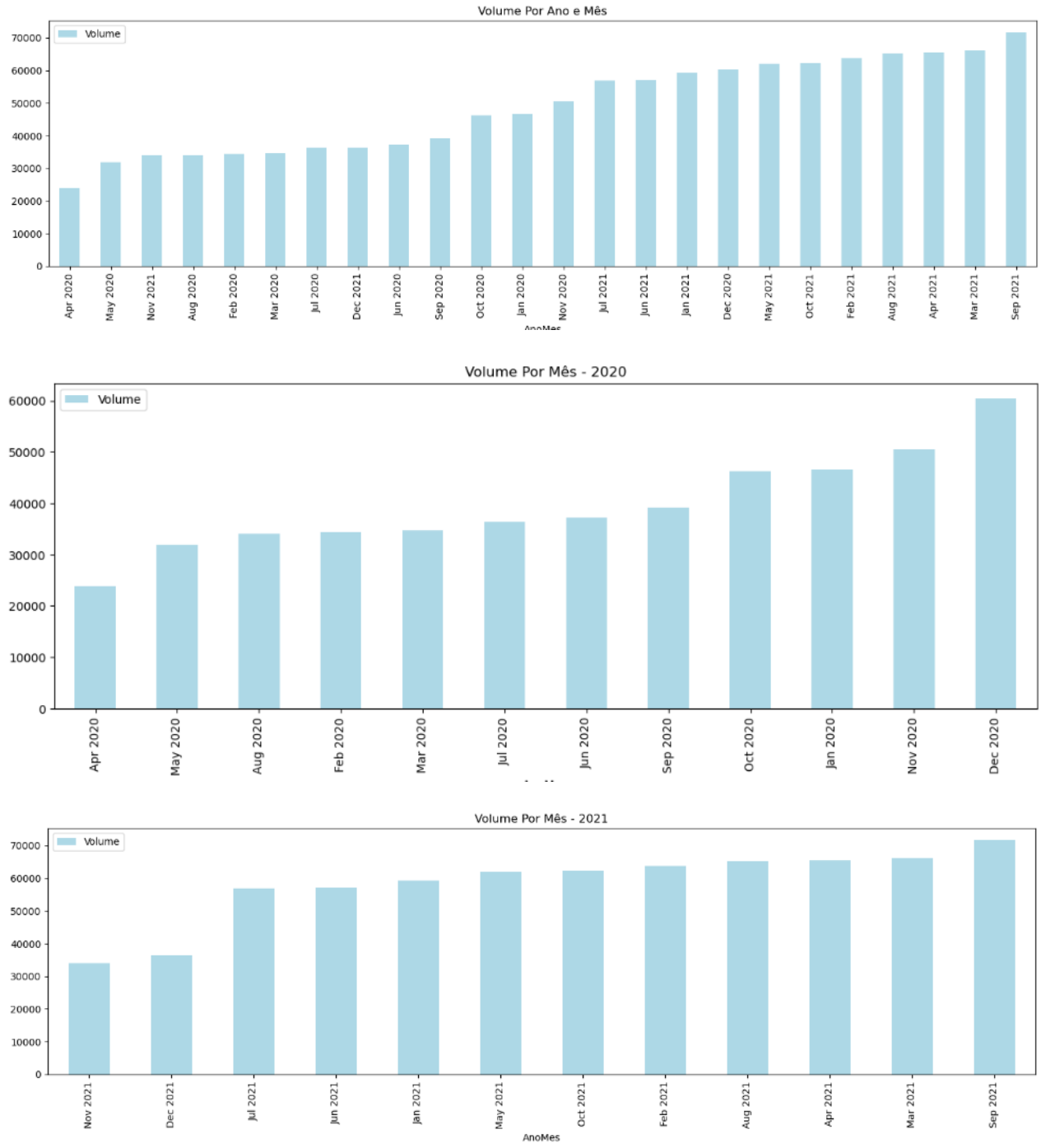
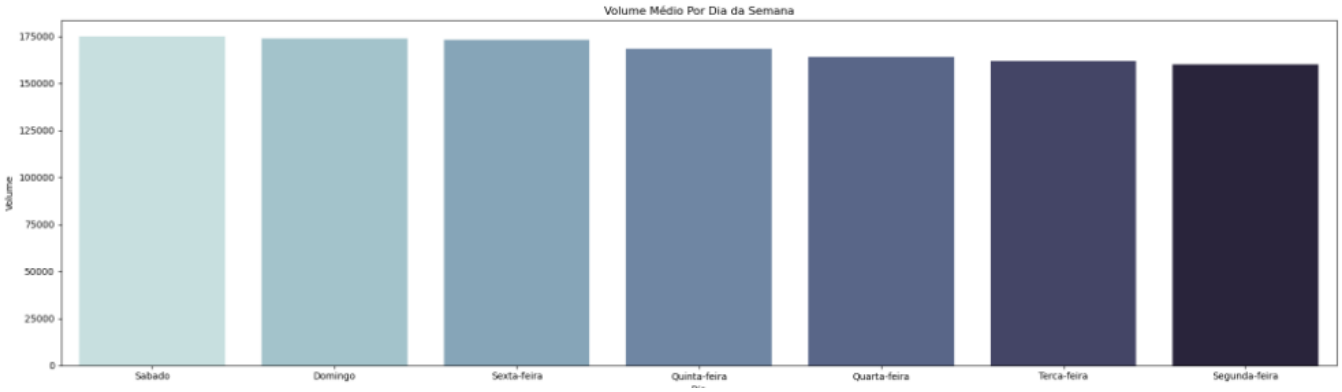


Figura 50: Analise exploratória de dados/ Graficos.

É possível observar uma crescente nos casos de infrações de 2020 à 2021. Tendo os meses de dezembro e setembro com maior quantidade de ocorrências.

Nota-se que devido à pandemia de COVID-19 e lockdown, possivelmente tendo afetado os dados do período de 2020



Em quais dias da semana temos aumento de infrações?

Podemos abrir a análise por dia e observar os dias da semana com a média de infrações mais altas.

Figura 51: Analise exploratória de dados/ Graficos.

```
#Criando Dataset
data1 = DfInfracoesSPa['Dia'].value_counts().reset_index()
data1 = pd.DataFrame(data1)
data1.columns = ['Dia','Volume']
data1 = data1[['Dia','Volume']].groupby('Dia').mean().sort_values('Dia',ascending = True).reset_index()
#Plot
plt.figure(figsize=(20,6))
sns.barplot(x='Dia',
            y='Volume',
            data=data1,
            palette ='ch:start=.2,rot=-.3',
            order=data1.sort_values('Volume',ascending = False).Dia)
plt.title("Volume Médio Por Dia da Semana")
print(data1)
plt.tight_layout()
```

```
             Dia    Volume
0         Domingo  173928.0
1   Quarta-feira  163952.0
2   Quinta-feira  168444.0
3         Sabado  174787.0
4  Segunda-feira  159927.0
5    Sexta-feira  173075.0
6    Terca-feira  161972.0
```

Figura 52: Analise exploratória de dados/ Graficos.

Observa-se que o volume de infrações sobe a partir do início do final de semana, tendo os dias com mais ocorrências Sábado, Domingo e Sexta respectivamente.

Em quais os períodos do dia temos mais ocorrências?

Adentrando nos dias, podemos analisar os períodos dos dias e suas médias de infrações.

```
#Criando Dataset
fig, (ax,ax1,ax2) = plt.subplots(nrows=1,ncols=3)

data = DfInfracoesSPa[['AnoMes','Periodo']].groupby(['AnoMes','Periodo']).value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['AnoMes','Periodo','Volume']
#Filtrando Dataset
data1 = data.loc[data['AnoMes'].str.contains('2020')]
data2 = data.loc[data['AnoMes'].str.contains('2021')]

data = data[['Periodo','Volume']].groupby('Periodo').mean()
data1= data1[['Periodo','Volume']].groupby('Periodo').mean()
data2 = data2[['Periodo','Volume']].groupby('Periodo').mean()

#Plot
ax.set_title("Volume Médio Por Período 2020", size = 12)
ax1.set_title("Volume MédioPor Período 2021",size = 12)
ax2.set_title("Volume Médio Por Período Total",size = 12)


data1.plot(kind = 'bar', color ='lightblue', ax= ax,figsize=(15,5))
data2.plot(kind = 'bar', color ='lightblue', ax= ax1,figsize=(15,5))
data.plot( kind = 'bar', color ='lightblue',ax= ax2,figsize=(15,5))


plt.tight_layout()
data2
```

Figura 53: Analise exploratória de dados/ Graficos.

Periodo	Volume
Madrugada	5665.916667
Manhã	19670.666667
Noite	11197.750000
Tarde	21847.916667

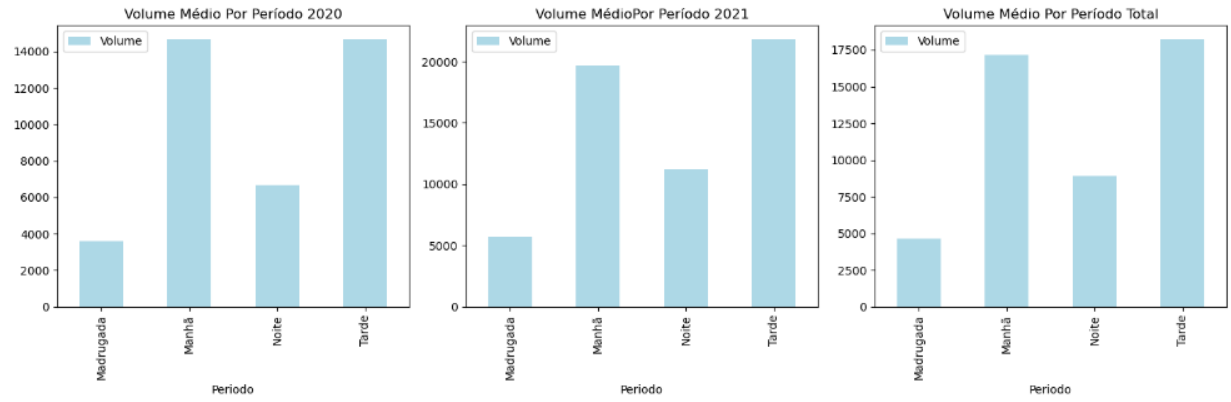
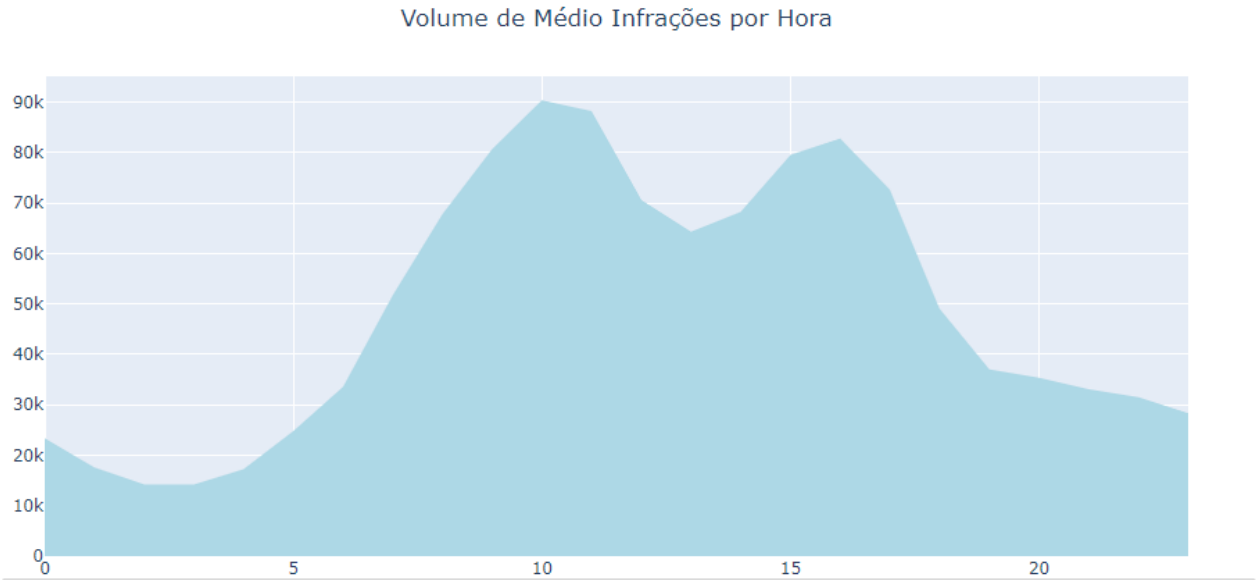


Figura 54: Analise exploratória de dados/ Graficos.

Em sua totalidade, entre 2020 e 2021 o período médio de maior ocorrência das infrações é no período da **Tarde**, seguido do período da **Manhã**. O que faz

correlação com os horários de picos de tráfego das rodovias de São Paulo serem entre **7h** e **10h** da manhã e das **17h** às **20h**, divulgados pela EBC - Empresa Brasil de Comunicação.

Em quais horários do dia temos picos de ocorrências de infrações?




```
#Criando dataset
data2 = DfInfracoesSPa['Hora Infração'].value_counts().reset_index()
data2 = pd.DataFrame(data2)
data2.columns = ['Hora Infração','Volume']
data2 = data2[['Hora Infração','Volume']].groupby('Hora Infração').mean().sort_values('Hora Infração',ascending = True).reset_inc
#Plot
fig = go.Figure()
fig.add_trace(go.Scatter(x=data2['Hora Infração'], y=data2['Volume'], fill='tonexty',fillcolor='lightblue',mode= 'none'))

fig.update_layout(
    title={
        'text': "Volume de Médio Infrações por Hora",
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
plt.show()
fig.show()
plt.tight_layout();
```

Figura 55: Analise exploratória de dados/ Graficos.

Figura 56: Analise exploratória de dados/ Graficos

Nota-se que o horário de pico de ocorrências de infrações são respectivamente 10h, 11h e 16h.

Quais são as severidades das infrações registradas?

Vamos contar os registros de infrações por Gravidade e plotar um gráfico de rosca para vermos a representação de cada categoria de infrações.

```
#Criando dataset
data = DfInfracoesSPa.Gravidade.value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['Gravidade','Volume']
#Plot
fig = px.pie(data, values="Volume", names="Gravidade",
             color_discrete_sequence=px.colors.sequential.RdBu,
             opacity=0.7, hole=0.5,width=800, height=400)

fig.update_layout(
    title={
        'text': "Representação(%) de Infrações por Tipo ",
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})

fig.show()
```

Figura 57: Analise exploratória de dados/ Graficos

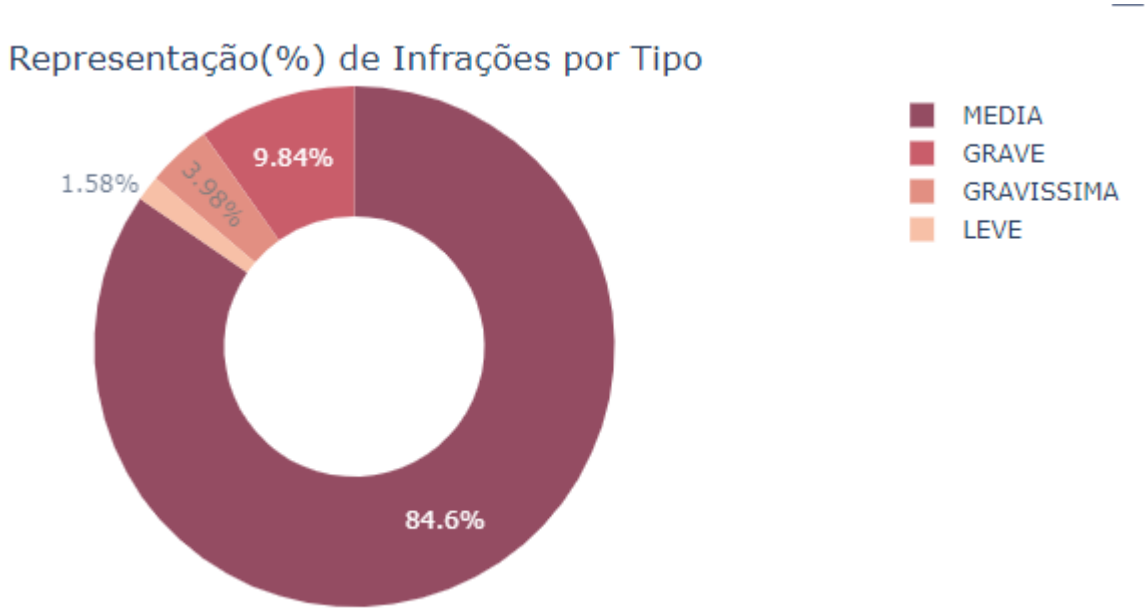


Figura 58: Analise exploratória de dados/ Graficos

Os tipos de infrações de maior ocorrência é a de categoria **Média** com uma cerca de 85% das ocorrências, seguida das **Graves** e a de menor ocorrência são as **leves**.

Quais são as infrações com maiores ocorrências?

Vamos montar um ranking com as 3 infrações com maiores volumes de ocorrências no período.

```
#Criando dataset
data = DfInfracoesSPa['Infracao'].value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['Infracao','Volume']
data = data.sort_values('Volume',ascending = False).head(3)
#Plot
plt.figure(figsize=(15,5))
sns.barplot(
            x='Infracao'
            ,y='Volume'
            ,data = data
            ,palette ='ch:start=.2,rot=-.3')
plt.title("Ranking de Infrações",size = 15)
plt.xlabel("Infracao", size = 15)
plt.ylabel("Volume", size = 15)
plt.tight_layout()
print(data)
```

```
                                             Infracao  Volume
0  Estacionar em desacordo com as posições estabe...  771413
1  Conduzir ciclo sem segurar o guidom com ambas ...   76302
2  Usar veículo para arremessar sobre os veículos...   24399
```

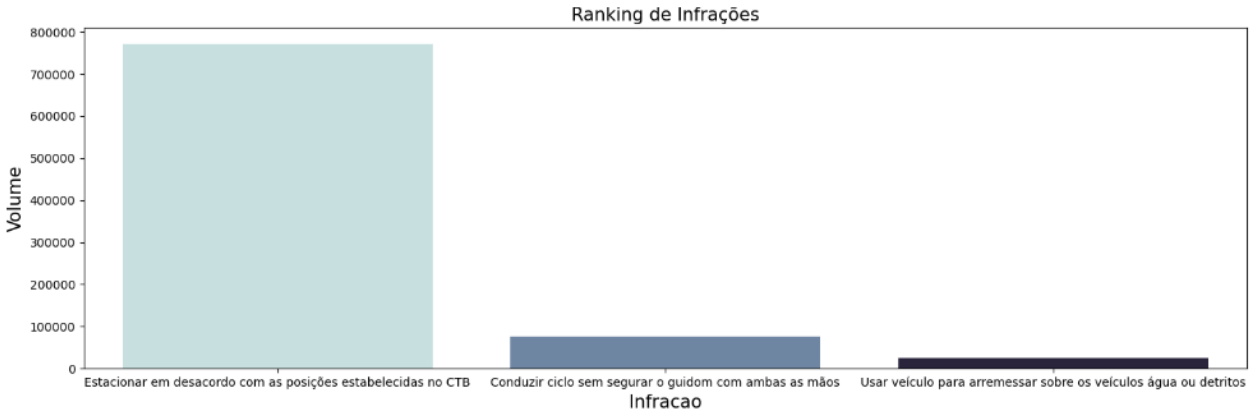


Figura 59: Analise exploratória de dados/ Graficos

Entre todas as infrações cometidas nas rodovias de São Paulo, as mais cometidas são respectivas:

1- **Estacionar em desacordo com as posições estabelecidas no CTB**

2- **Conduzir ciclo sem segurar o guidom com ambas as mãos**

3- **Usar veículo para arremessar sobre os veículos: água ou detritos**

Quais são as rodovias de São Paulo com maiores ocorrências de infrações?

Vamos montar um ranking com as 3 BRs com maiores volumes de ocorrências no período.

```
#Criando dataset
data = DfInfracoesSPa['BR Infração'].value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['BR','Volume']
data = data.sort_values('Volume',ascending = False).head(3)

#Plot
data.plot( kind='bar'
            ,x='BR'
            ,y='Volume'
            ,figsize=(5,4)
           , color ='lightblue'
            )
plt.title("Ranking de BRs",size = 10)
plt.xlabel("BR", size = 10)
plt.ylabel("Volume", size = 10)
plt.tight_layout()
```

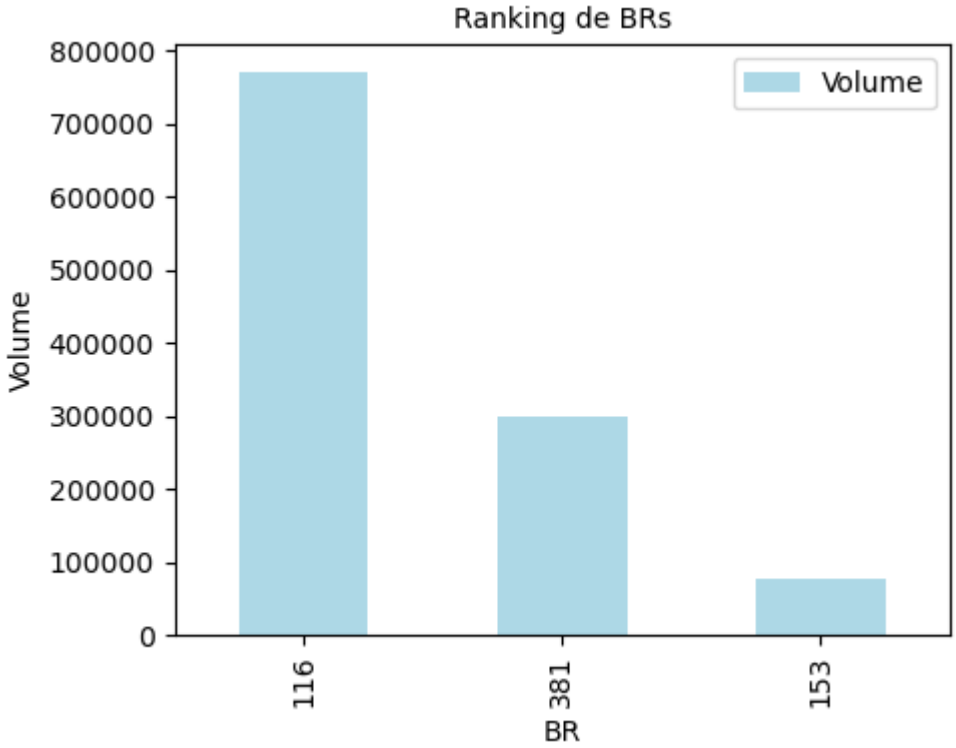


Figura 60: Analise exploratória de dados/ Graficos

Nota-se que a rodovia de São Paulo com mais ocorrências de infrações nos anos de 2020 e 2021 é a **BR 116**

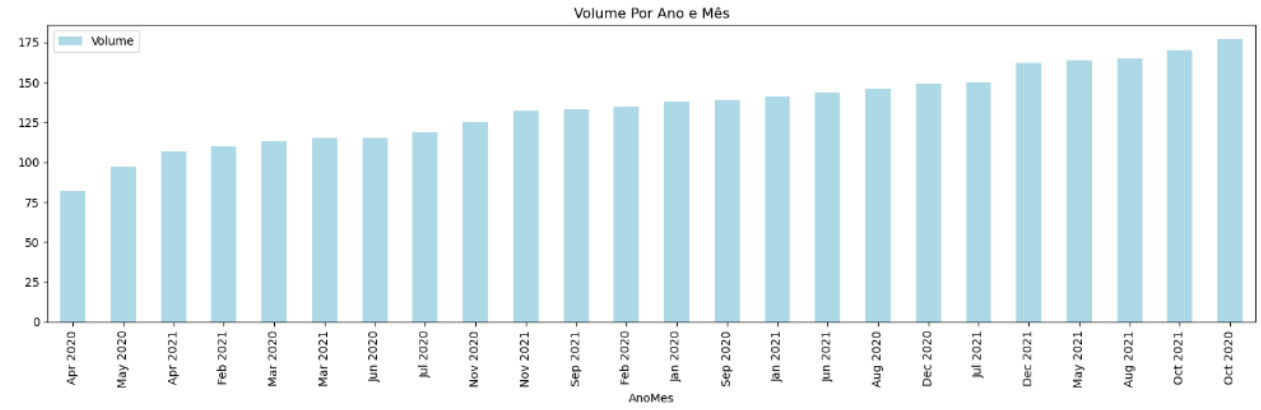
4.2 Acidentes

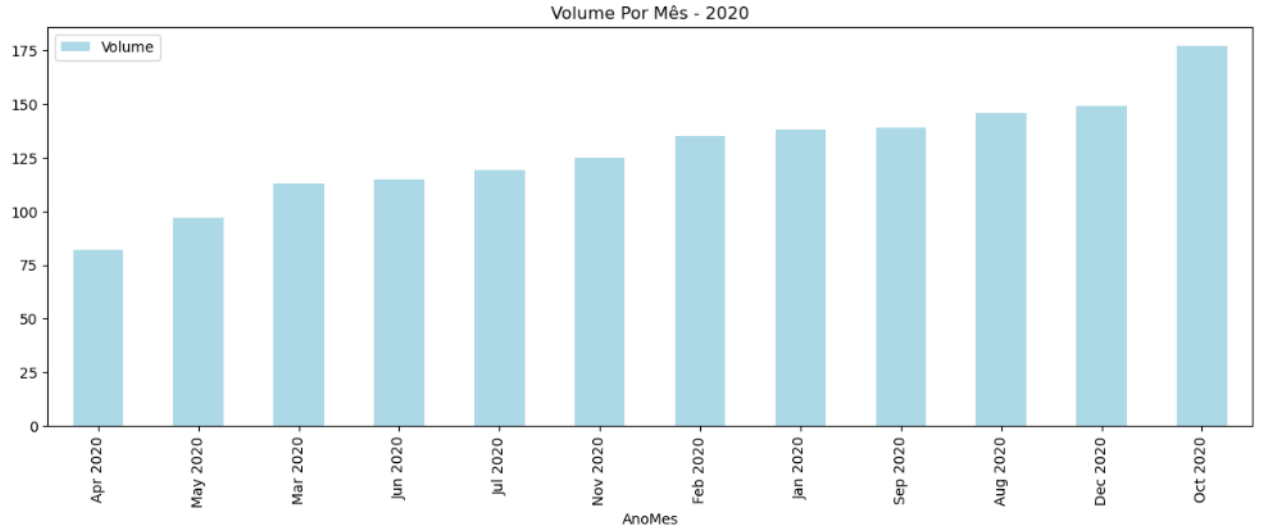
Em quais períodos do ano tivemos o maior índice de acidentes?

Vamos analisar o volume de acidentes em linha do tempo plotando os gráficos de barras dos dois anos.

```
#Criando dataset
data = DFacidentesSP_Clean['AnoMes'].value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['AnoMes','Volume']
#Filtrando Dataset
data1 = data.loc[data['AnoMes'].str.contains('2020')]
data2 = data.loc[data['AnoMes'].str.contains('2021')]

data = data[['AnoMes','Volume']].groupby('AnoMes').sum().sort_values('Volume',ascending=True)
data1= data1[['AnoMes','Volume']].groupby('AnoMes').sum().sort_values('Volume',ascending=True)
data2 = data2[['AnoMes','Volume']].groupby('AnoMes').sum().sort_values('Volume',ascending=True)

#Plot
data.plot(kind = 'bar', color ='lightblue',figsize=(15,5))

plt.title("Volume Por Ano e Mês")
plt.tight_layout()

data1.plot(kind = 'bar', color ='lightblue',figsize=(15,5))
plt.title("Volume Por Mês - 2020")

data2.plot(kind = 'bar', color ='lightblue',figsize=(15,5))
plt.title("Volume Por Mês - 2021")
plt.tight_layout()
```

Figura 61: Analise exploratória de dados/ Graficos





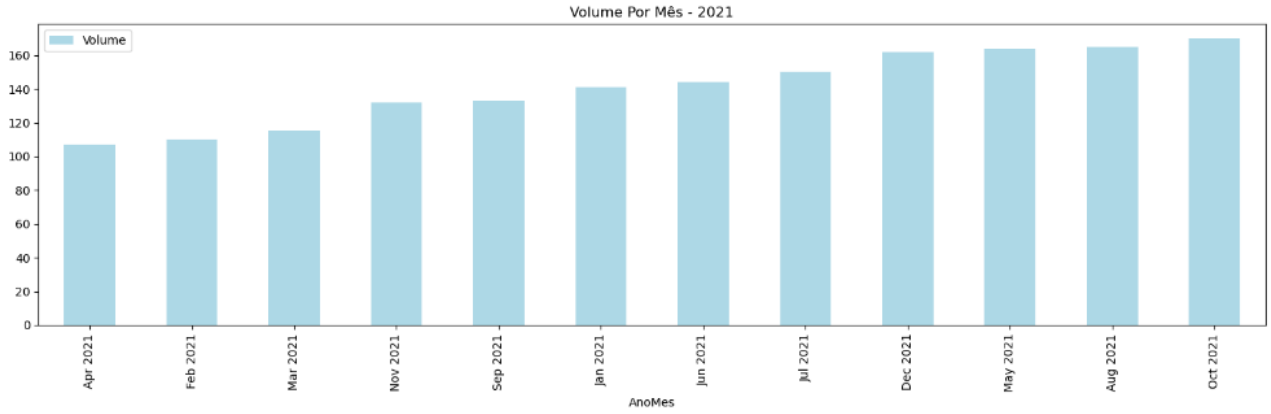


Figura 62: Analise exploratória de dados/ Graficos

Observando os casos de acidentes de 2020 à 2021, nota-se que outubro é o mês com maior quantidade de ocorrências nos dois anos, e abril o mês com menor ocorrência.

Qual o comportamento dos registros de infrações e acidentes durante o período?

Análise em linha do tempo a movimentação das ocorrências de acidentes sobre as ocorrências de infrações.

```
#Vamos separar a base de acidentes por Data e volume de ocorrências
TotalAcidentes = DFacidentesSP_Clean.AnoMes.value_counts().reset_index()
TotalAcidentes = pd.DataFrame(TotalAcidentes)
TotalAcidentes.columns = ['Data','Volume']
#Vamos separar a base de infrações por Data e volume de ocorrências
TotalInfracoes = DfInfracoesSPa.AnoMes.value_counts().reset_index()
TotalInfracoes = pd.DataFrame(TotalInfracoes)
TotalInfracoes.columns = ['Data','Volume']
```

```
#Vamos juntar as duas bases cruzando por Data
Total = pd.merge(TotalInfracoes,TotalAcidentes, on = 'Data', how = 'left')
```

```
#Renomeando colunas
Total.columns = ['Data','TotalInfracoes','TotalAcidentes']
```

```
#Plotando gráfico de coluna e linha.
ax =  Total.plot.bar(y='TotalInfracoes', ylabel='TotalInfracoes', figsize=(25,6))
Total.plot(y='TotalAcidentes', x = 'Data', c='red', ax=ax, use_index=True, secondary_y=True, mark_right=False)
ax.right_ax.set_ylabel('TotalAcidentes');
```

Figura 63: Analise exploratória de dados/ Graficos

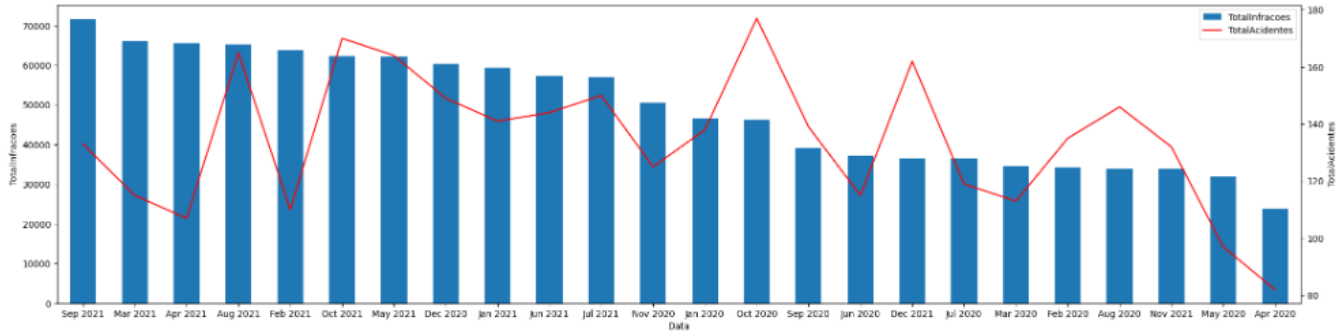
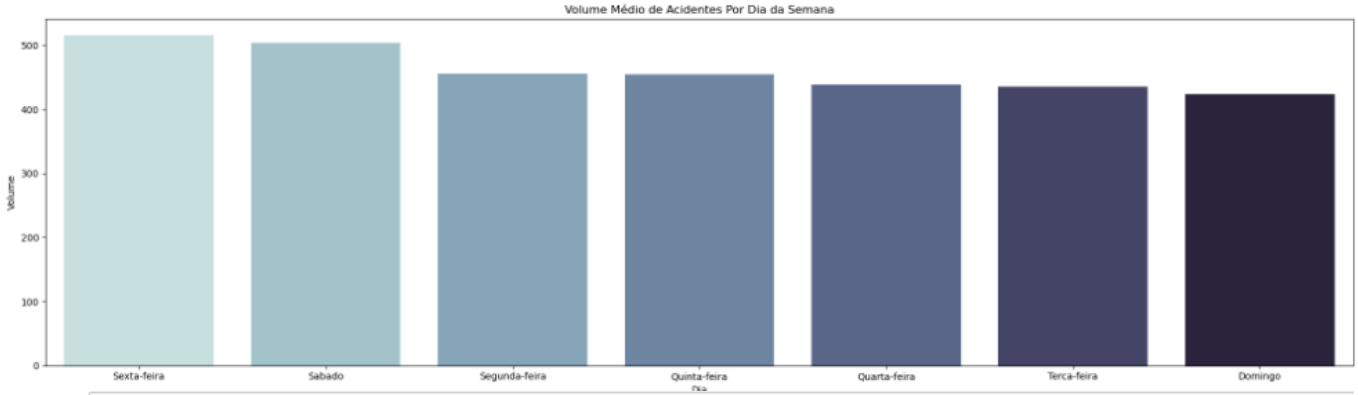


Figura 64: Analise exploratória de dados/ Graficos

Em quais dias da semana temos aumento de acidentes?

Podemos abrir a análise por dia e observar os dias da semana com a média de acidentes mais altas.




```
data1 = DFacidentesSP_Clean['Dia'].value_counts().reset_index()
data1 = pd.DataFrame(data1)
data1.columns = ['Dia','Volume']
data1 = data1[['Dia','Volume']].groupby('Dia').mean().sort_values('Dia',ascending = True).reset_index()
plt.figure(figsize=(20,6))
sns.barplot(x='Dia',
            y='Volume',
            data=data1,
            palette ='ch:start=.2,rot=-.3',
            order=data1.sort_values('Volume',ascending = False).Dia)
plt.title("Volume Médio de Acidentes Por Dia da Semana")
print(data1)
plt.tight_layout()
```

```
            Dia  Volume
0        Domingo   424.0
1   Quarta-feira   439.0
2   Quinta-feira   455.0
3         Sabado   504.0
4  Segunda-feira   456.0
5    Sexta-feira   515.0
6    Terca-feira   435.0
```

Figura 65: Analise exploratória de dados/ Graficos

Figura 66: Analise exploratória de dados/ Graficos

Observa-se que o volume de infrações sobe a partir do início do final de semana, tendo os dias com mais ocorrências Sexta e Sábado respectivamente.

Em quais os períodos do dia temos mais ocorrências?

Vamos analisar o comportamento desses casos por período do dia.

```
#Criando dataset
fig, (ax,ax1,ax2) = plt.subplots(nrows=1,ncols=3)

data = DFacidentesSP_Clean[['AnoMes','Periodo']].groupby(['AnoMes','Periodo']).value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['AnoMes','Periodo','Volume']
#Filtrado Dataset
data1 = data.loc[data['AnoMes'].str.contains('2020')]
data2 = data.loc[data['AnoMes'].str.contains('2021')]

data = data[['Periodo','Volume']].groupby('Periodo').mean()
data1= data1[['Periodo','Volume']].groupby('Periodo').mean()
data2 = data2[['Periodo','Volume']].groupby('Periodo').mean()
#Plot
ax.set_title("Volume Por Período 2020", size = 12)
ax1.set_title("Volume Por Período 2021",size = 12)
ax2.set_title("Volume Por Período Total",size = 12)


data1.plot(kind = 'bar', color ='lightblue', ax= ax,figsize=(15,5))
data2.plot(kind = 'bar', color ='lightblue', ax= ax1,figsize=(15,5))
data.plot( kind = 'bar', color ='lightblue',ax= ax2,figsize=(15,5))


plt.tight_layout()

data2
```

Figura 67: Analise exploratória de dados/ Graficos

Periodo	Volume
Madrugada	19.000000
Manhã	43.666667
Noite	36.083333
Tarde	42.333333

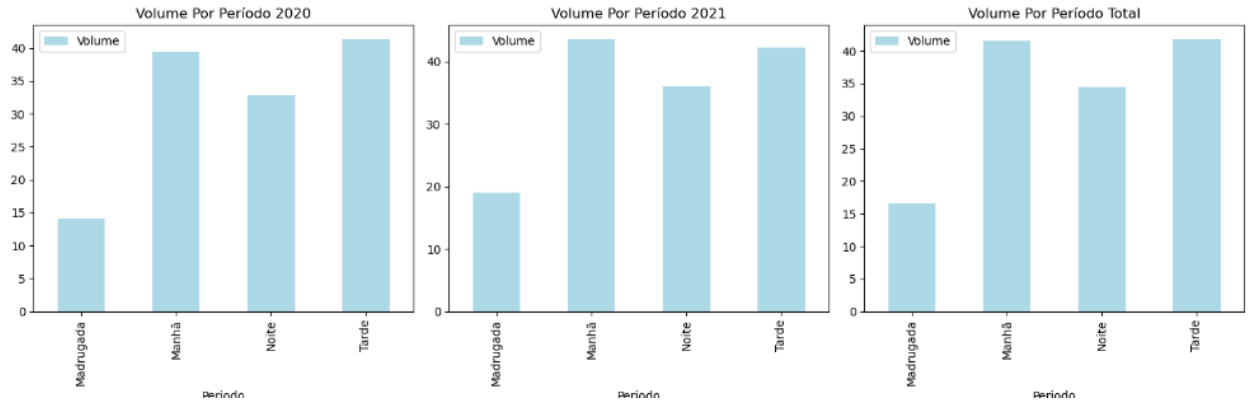


Figura 68: Analise exploratória de dados/ Graficos

Em sua totalidade, entre 2020 e 2021 o período médio de maior ocorrência de acidentes é no período da **Manhã**, seguido do período da **Tarde**, com médias quase iguais de acidentes.

Em quais horários do dia temos picos de ocorrências de acidentes?

```
#Criando dataset
data2 = DFacidentesSP_Clean['Hora'].value_counts().reset_index()
data2 = pd.DataFrame(data2)
data2.columns = ['Hora','Volume']
data2 = data2[['Hora','Volume']].groupby('Hora').mean().sort_values('Hora',ascending = True).reset_index()
#Plot
fig = go.Figure()
fig.add_trace(go.Scatter(x=data2['Hora'], y=data2['Volume'], fill='tonexty',fillcolor='lightblue',mode= 'none'))

fig.update_layout(
    title={
        'text': "Volume de Médio Acidentes por Hora",
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})

plt.show()
fig.show()
plt.tight_layout();
```

Figura 69: Analise exploratória de dados/ Graficos

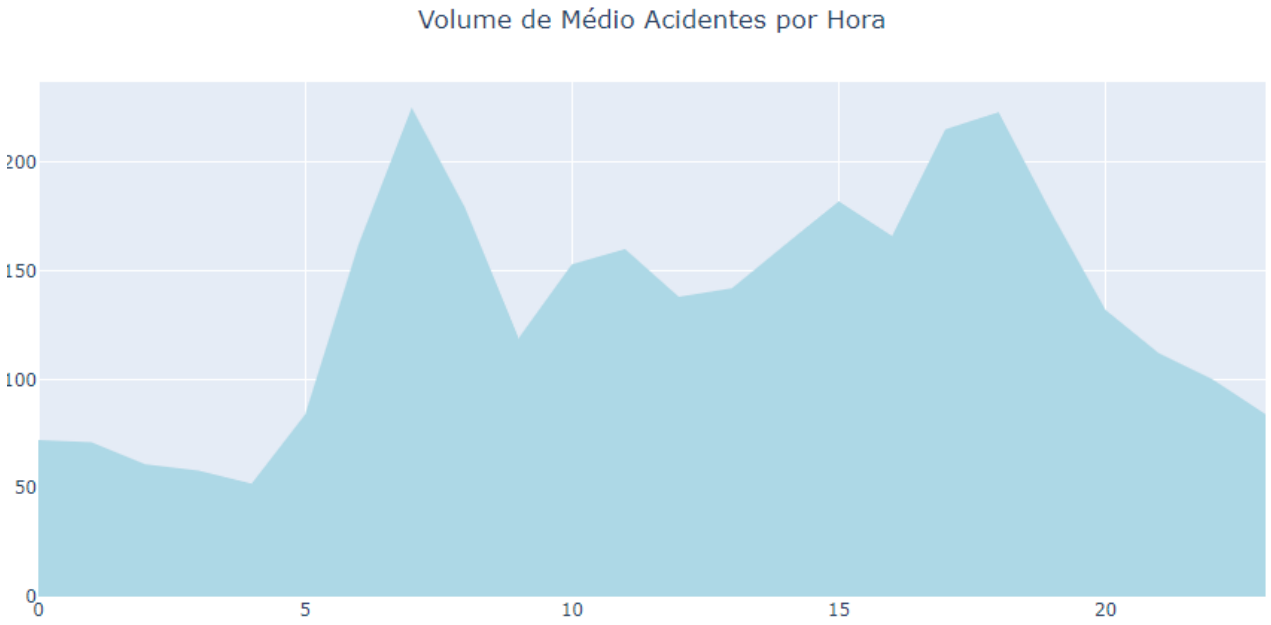


Figura 70: Analise exploratória de dados/ Graficos

Nota-se que o horário de pico de ocorrências de acidentes são respectivamente 7h e 18h.

Qual a Correlação existente entre as variáveis?

Utilizando o Mapa de calor conseguimos observar que se possuímos fortes correlações referentes às variáveis do dataset.

```
#Criando dataset com as váriaveis para correlação
AcidentesSoma = DFacidentesSP_Clean[['pessoas','mortos','feridos_leves','feridos_graves','ilesos','ignorados','feridos','veiculos
AcidentesSoma = pd.DataFrame(AcidentesSoma)
AcidentesSoma.columns =['pessoas','mortos','feridos_leves','feridos_graves','ilesos','ignorados','feridos','veiculos','Hora','Vol
```

```
#Plotando correlação
AcidentesSomaCorr = AcidentesSoma.corr()
sns.heatmap(AcidentesSomaCorr,cmap = sns.cubehelix_palette(as_cmap=True), fmt='.2f', square=True, linecolor='white', annot=True);
```

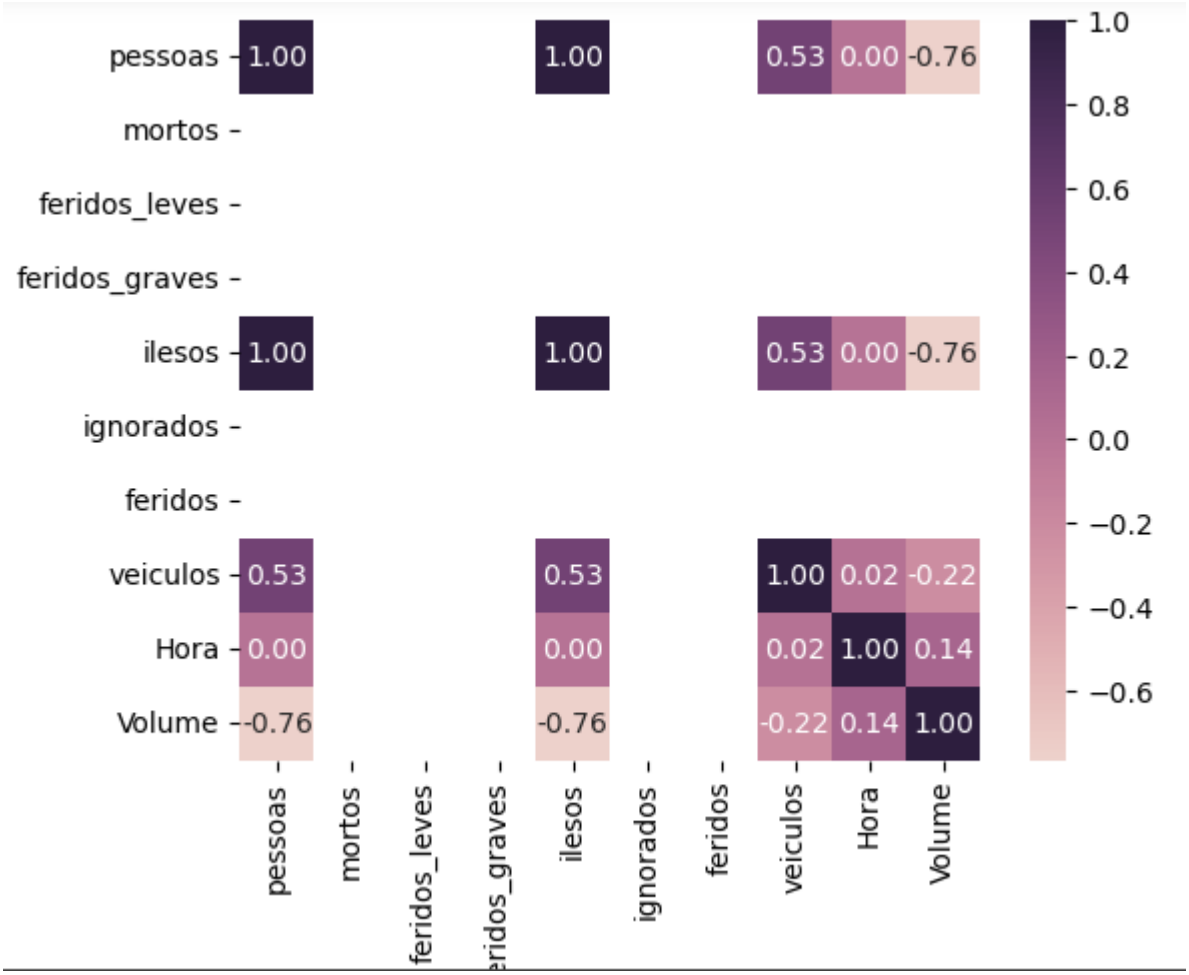


Figura 71: Analise exploratória de dados/ Graficos

Com o mapa de calor conseguimos perceber uma correlação a variável Volume e Pessoas, e nos descreve que quanto maior o volume de acidentes, menor o volume de pessoas ilesas.

Qual a representação de pessoas envolvidas nos acidentes?

```
#Criando dataset
data = DFacidentesSP_Clean.pessoas.value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['Pessoas','Volume']

fig = px.pie(data, values="Volume", names="Pessoas",
             color_discrete_sequence=px.colors.sequential.RdBu,
             opacity=0.7, hole=0.5,width=800, height=400)
#Plot
fig.update_layout(
    title={
        'text': "Representação(%) de Qtd de pessoas em acidentes ",
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})

fig.show()
```

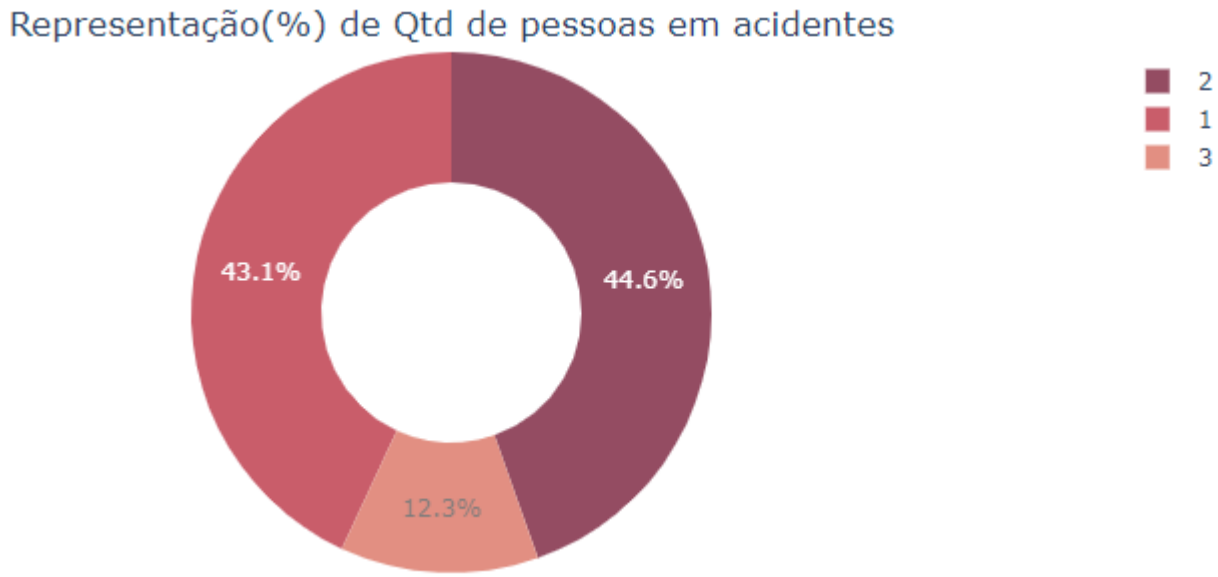


Figura 72: Analise exploratória de dados/ Graficos

Quais são as causas de acidentes com maiores ocorrências?

Vamos montar um ranking com as Causas com maiores volumes de ocorrências no período.

```
#Criando dataset
data = DFacidentesSP_Clean['causa_acidente'].value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['causa_acidente','Volume']
data = data.sort_values('Volume',ascending = False).head(5)
#Plot
plt.figure(figsize=(18,5))
sns.barplot(x='causa_acidente',
            y='Volume',
            data=data,
            palette ='ch:start=.2,rot=-.3',
            order=data.sort_values('Volume',ascending = False).causa_acidente)
plt.title("Ranking Causa de Acidentes")
plt.tight_layout()
```

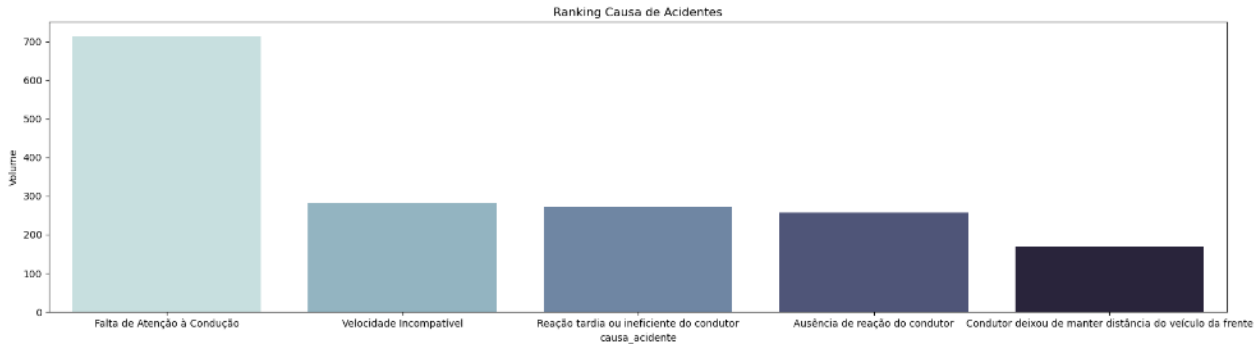


Figura 73: Analise exploratória de dados/ Graficos

A Falta de atenção à condução é destacada como a maior causa de ocorrências de acidentes, segui da Velocidade incompatível.

Quais são as rodovias de São Paulo com maiores ocorrências de infrações?

Vamos montar um ranking com as 3 BRs com maiores volumes de ocorrências no período.

```
#Criando dataset
data = DFacidentesSP_Clean['br'].value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['BR','Volume']
data = data.sort_values('Volume',ascending = False).head(3)
#Plot
data.plot( kind='bar'
            ,x='BR'
            ,y='Volume'
            ,figsize=(5,4)
           , color ='lightblue'
            )
plt.title("Ranking de BRs",size = 10)
plt.xlabel("BR", size = 10)
plt.ylabel("Volume", size = 10)
plt.tight_layout()
```

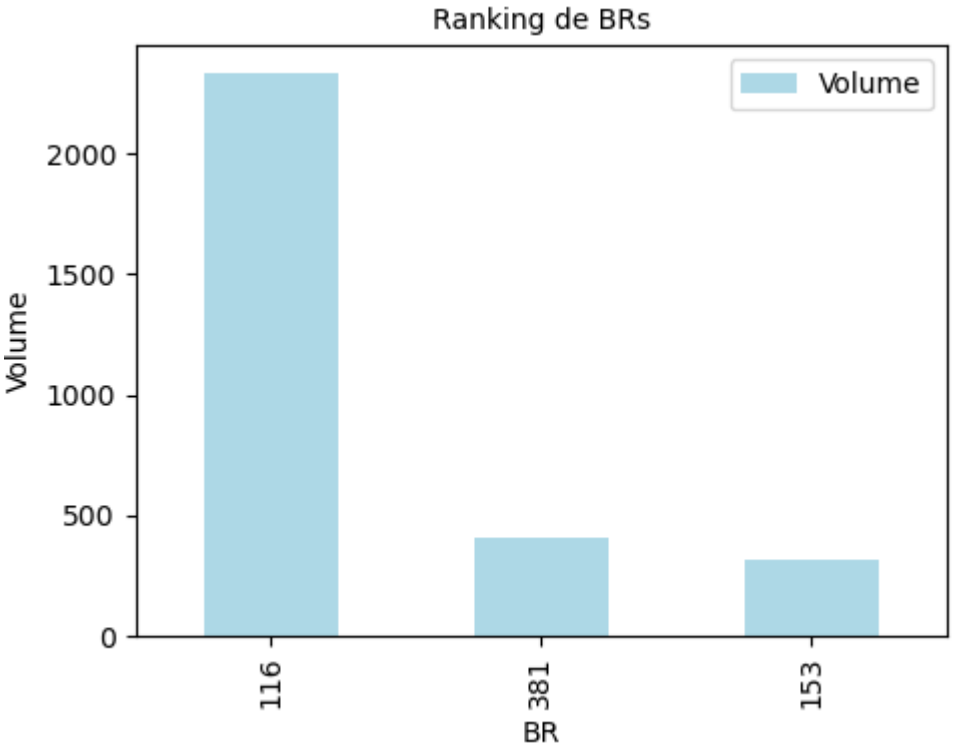


Figura 74: Analise exploratória de dados/ Graficos

Mapa

Através do mapa plotado abaixo é possível observar a dispersão de ocorrências de acidentes ao redor de São Paulo. Foram criadas duas colunas no dataset:

Qtd que categoriza o volume de acidentes como Baixo, Médio, Alto.

Cores categoriza em cores com base no Qtd, para complementar a legenda do gráfico de Mapa.

Legenda

Amarelo - Volume de acidentes menor ou igual à 1 ocorrências.

Laranja - Volume de acidentes entre 2 e 10 ocorrências.

Vermelho - Volume de acidentes maior que 10 ocorrências.

```
#Criando dataset
data = DFacidentesSP_Clean[['latitude','longitude']].value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['latitude','longitude','Acidentes']
data.describe()
```

	Acidentes
count	1518.000000
mean	2.126482
std	2.438393
min	1.000000
25%	1.000000
50%	1.000000
75%	2.000000
max	22.000000

Figura 75: Analise exploratória de dados/ Graficos

```
#Colunas para popular valores das cores
data['Qtd'] = 'N/A'
data['Cores'] = 'N/A'

#Definindo as categorias de volume de acidentes
data.loc[data['Acidentes']<=1, ['Qtd','Cores']] = ['Baixo','beige']
data.loc[(data['Acidentes']>=2) & (data['Acidentes'] <= 10), ['Qtd','Cores']] = ['Alto','lightred']
data.loc[data['Acidentes']>10, ['Qtd','Cores']] = ['Super Alto','red']

#Convertendo os valores de longitude e latitude para float, e definindo uma amostra de 500 registros para plotar no mapa
data_plot = data[['latitude','longitude','Acidentes','Cores']].sort_values('Acidentes', ascending = False)
data_plot = data_plot.replace(',','.',regex=True).sample(200)
data_plot['latitude'] = data_plot['latitude'].astype(float)
data_plot['longitude'] = data_plot['longitude'].astype(float)


import folium

mapa = folium.Map()
for index, location_info in data_plot.iterrows():
    folium.Marker([location_info['latitude'],location_info['longitude']], popup = location_info[['Acidentes']]
                  ,icon = folium.Icon(color = location_info['Cores'],icon='warning-sign'), zoom_scale =25).add_to(mapa)
mapa
```

Figura 76: Analise exploratória de dados/ Graficos

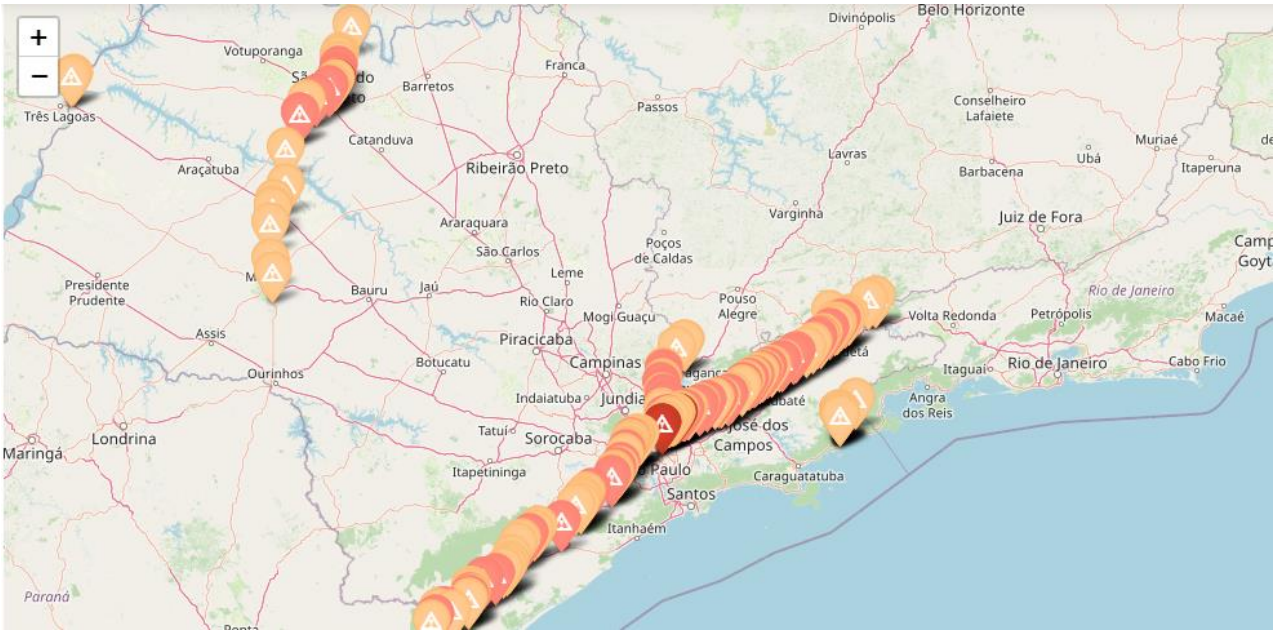


Figura 77: Analise exploratória de dados/ Graficos

Observando o mapa também podemos identificar o tipo da via em que ocorrem mais acidentes.

Existe um tipo de via na qual há mais ocorrência de acidentes?

```
#Criando dataset
data = DFacidentesSP_Clean['tracado_via'].value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['tracado_via','Acidentes']
```

```
data = data.sort_values('Acidentes',ascending = False).head(3)
#Plot
data.plot( kind='bar'
           ,x='tracado_via'
           ,y='Acidentes'
           ,figsize=(5,4)
           )
plt.title("Acidentes por Traçado da via ",size = 10)
plt.xlabel("Tipo", size = 10)
plt.ylabel("Acidentes", size = 10)
plt.tight_layout()
```

Figura 78: Analise exploratória de dados/ Graficos

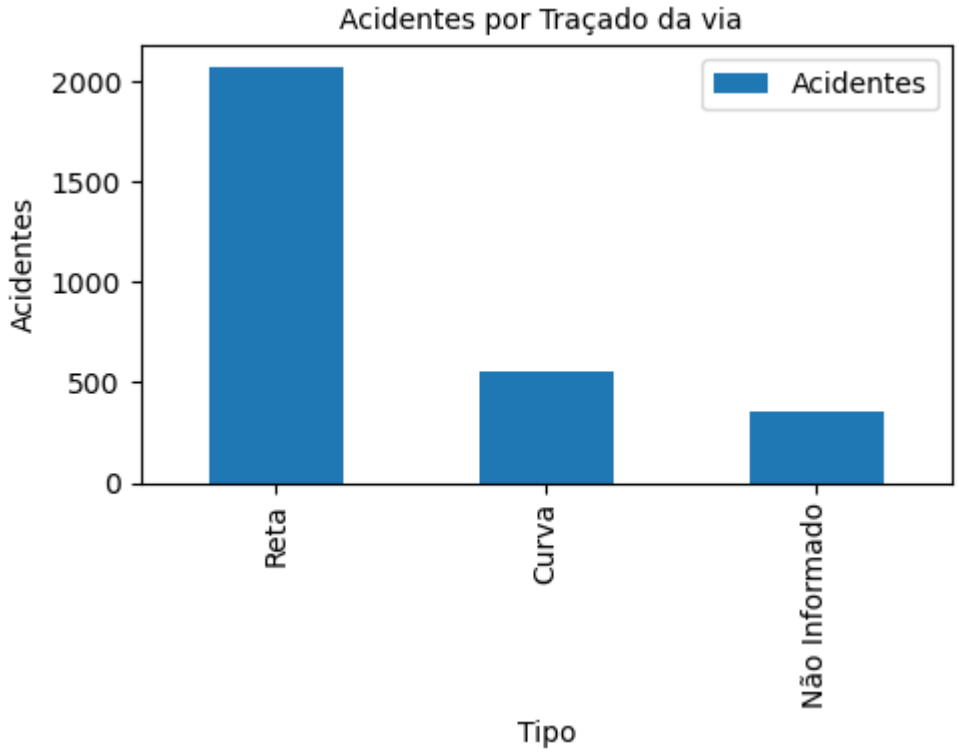


Figura 79: Analise exploratória de dados/ Graficos

Nota-se que o tipo de via em que mais ocorrem acidentes em SP são em linha **reta**, seguido das **curvas**.

Qual o comportamento das ocorrências de acidentes sobre as condições meteorológicas?

Podemos observar o comportamento dos acidentes em relação às condições meteorológica.

```
#Criando dataset
data = DFacidentesSP_Clean['condicao_metereologica'].value_counts().reset_index()
data = pd.DataFrame(data)
data.columns = ['condicao_metereologica','Volume']
data = data.sort_values('Volume',ascending = False).head(3)
data= data[['condicao_metereologica','Volume']].groupby('condicao_metereologica').mean().reset_index()
data = data.sort_values('condicao_metereologica',ascending = True)
#Plot
data.plot( kind='bar'
            ,x='condicao_metereologica'
            ,y='Volume'
            ,figsize=(5,4)
            )
plt.title("Acidentes por Condição Metereologica ",size = 10)
plt.xlabel("Condição", size = 10)
plt.ylabel("Volume", size = 10)
plt.tight_layout()
```

Figura 80: Analise exploratória de dados/ Graficos

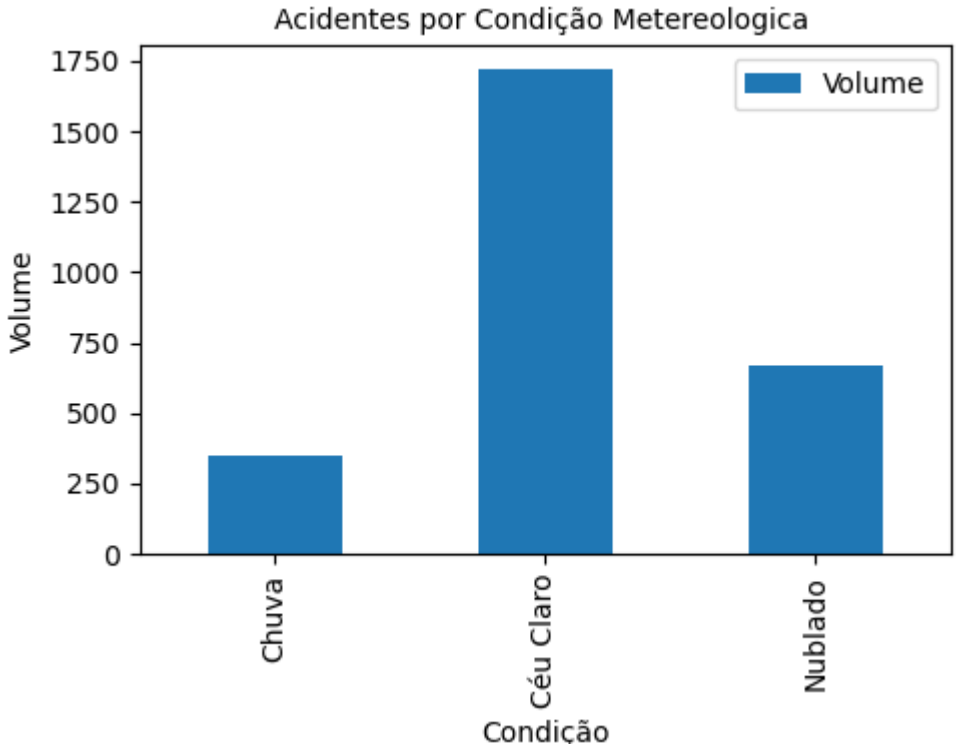
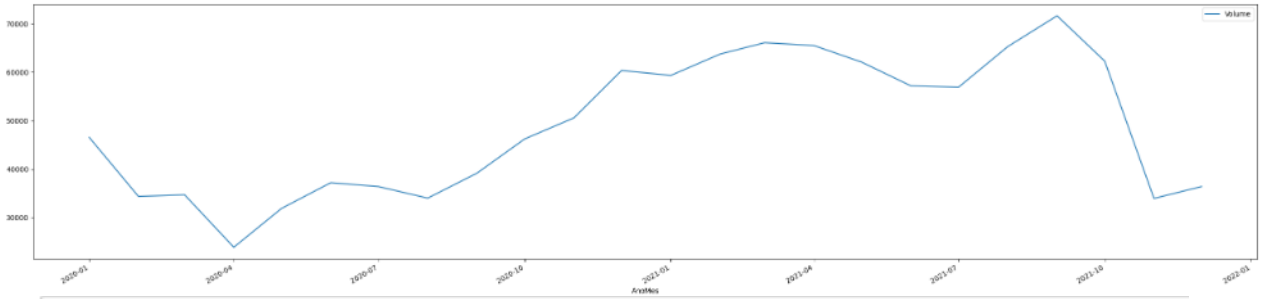


Figura 81: Analise exploratória de dados/ Graficos

As ocorrências de acidentes no período de **2020** à **2021** aconteceram em sua maioria à **céu claro**.

5. Criação de Modelos de Machine Learning

Primeiramente iremos criar as bases de acidentes e infrações em séries temporais com as quantidades indexadas por Ano Mês:

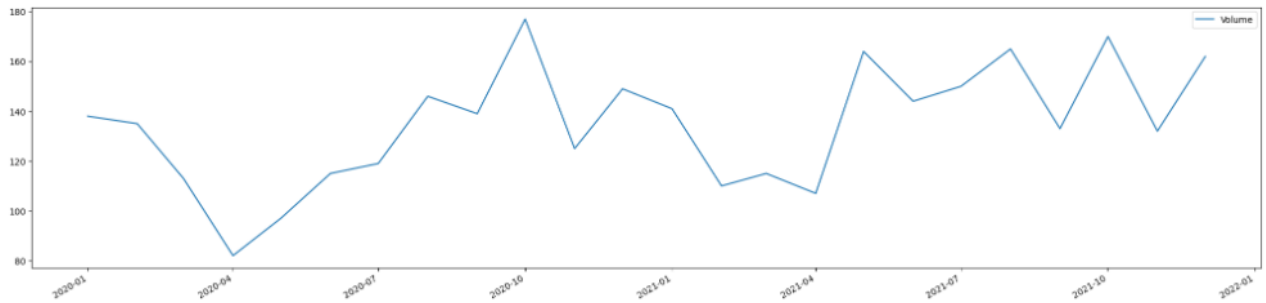



```
#Plotando Série Temporal de Acidentes
Acidentes = DFacidentesSP_Clean['AnoMes'].value_counts().reset_index()
Acidentes = pd.DataFrame(Acidentes)
Acidentes.columns = ['AnoMes','Volume']
Acidentes['AnoMes'] = pd.to_datetime(Acidentes['AnoMes'])

Acidentes.plot(y='Volume', x = 'AnoMes', xlabel='AnoMes',figsize=(25,6))
```

Figura 82: Machine Learning/ Séries temporais.

```
#Plotando Série Temporal de Infrações
Infracoes = DfInfracoesSPa['AnoMes'].value_counts().reset_index()
Infracoes = pd.DataFrame(Infracoes)
Infracoes.columns = ['AnoMes','Volume']
Infracoes['AnoMes'] = pd.to_datetime(Infracoes['AnoMes'])

Infracoes.plot(y='Volume', x = 'AnoMes',xlabel='AnoMes',figsize=(25,6))
plt.tight_layout();
```

Figura 83: Machine Learning/ Séries temporais.



Indexando as bases de Acidentes e Infrações Por Data e suas quantidades

```
#Indexando as bases de Acidentes e Infrações Por Data e suas quantidades
Acidentes = Acidentes.sort_values('AnoMes', ascending = True)
Acidentes.set_index('AnoMes', inplace = True)

Infracoes = Infracoes.sort_values('AnoMes', ascending = True)
Infracoes.set_index('AnoMes', inplace = True)
```

Figura 84: Machine Learning/ Séries temporais.

Vamos juntar as duas bases cruzando por Data

```
df = pd.merge(Acidentes,Infracoes, on = 'AnoMes', how = 'left')
df = df.sort_values('AnoMes', ascending = True)
#Renomear as colunas
df.columns = ['Acidentes','Infracoes']
```

Figura 85: Machine Learning/ Séries temporais.

Autocorrelação (ACF) e Autocorrelação Parcial (PACF)

A autocorrelação e a autocorrelação parcial são medidas de associação entre valores de séries atuais e anteriores e indicam quais valores de série anteriores são mais úteis para prever valores futuros. Com esse conhecimento, é possível determinar a ordem dos processos no modelo ARIMA.

Vamos plotar os gráficos de ACF e PACF:

```
# Vamos plotar os graficos de ACF e PACF
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(16, 8))

for i, column in enumerate(df.columns):
  plot_acf(df[column], ax=ax[0,i], title = f"Autocorrelação {column}")
  plot_pacf(df[column], ax=ax[1,i], lags = 11, title = f"Autocorrelação Parcial {column}")

plt.show()
```

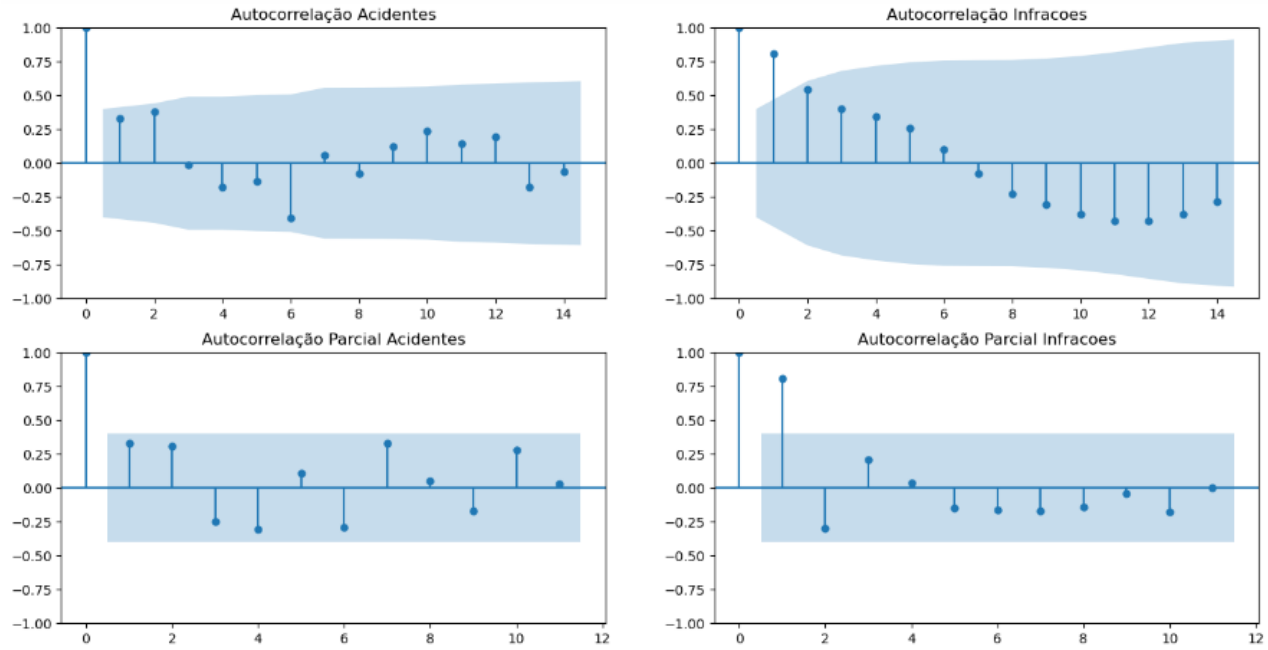


Figura 85: Machine Learning/ Séries temporais.

Os valores de Autocorrelação Acidentes ficam dentro do intervalo de confiança a partir do Lag 1. Sendo assim, o valor p estará em 0.
No caso da Autocorrelação Infrações o valor de p estará entre 0 e 1.

Observando a Autocorrelação Parcial, em Acidentes apenas o primeiro valor tem uma Correlativa alta significante diante os outros. O valor de q estará em 0.
No caso da Autocorrelação Parcial Infrações o valor de q estará entre 0 e 1.

Decompondo a Série de Acidentes e Infrações

vamos decompor as séries de Acidentes e Infrações para entender os comportamentos de sazonalidade, tendência e resíduo

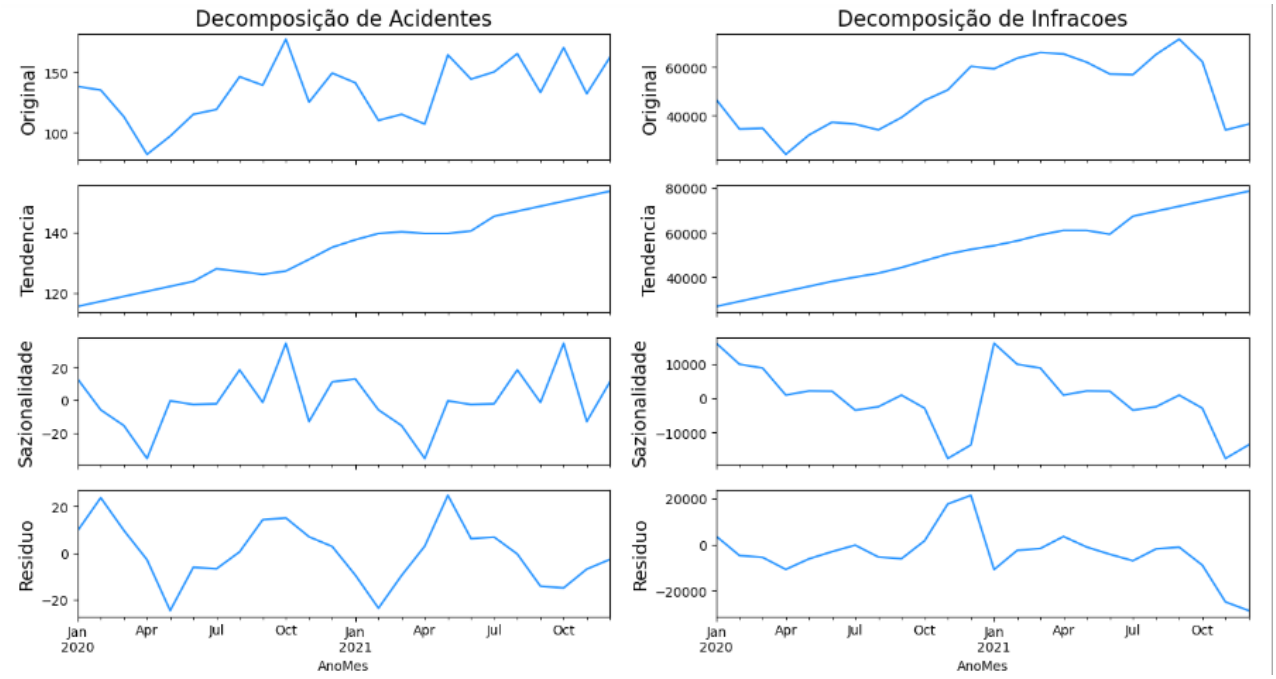


Figura 86: Machine Learning/ Séries temporais.

Sazonalidade

Observamos que há certa sazonalidade sobre ambas séries.

Tendência

No gráfico de tendência, observa-se que a tendência era Crescente para ambas séries.

Teste Dickey-Fuller

O teste de Dickey-Fuller serve para confirmarmos se a série é probabilisticamente estacionário, o que é verdadeiro para valores p menores que 0,05%.

```python
# Confirma através do teste que as series não são estacionárias

def TesteAdf(df):
    resultado = adfuller(df.values)
    print(f"Teste Adfuller da série '{df.name}'")
    print(f"ADF Statistics: {round(resultado[0], 2)}")
    print(f"p-value: {round(resultado[1], 2)}")
    print("# n_lags: " , resultado[2])
    print("Núm. Observações: " , resultado[3])
    print(f"Valores Críticos:")
    for key, value in resultado[4].items():
         print(f"\t{key}: {round(value,2)} ")
    print( "\n")
    return ("A série NÃO é estacionária" if resultado[1] > 0.05 else "A série é estacionária")

df.apply(TesteAdf, axis = 0)
```

```
Teste Adfuller da série 'Acidentes'
ADF Statistics: -0.04
p-value: 0.96
# n_lags:  9
Núm. Observações:  14
Valores Críticos:
        1%: -4.01
        5%: -3.1
        10%: -2.69


Teste Adfuller da série 'Infracoes'
ADF Statistics: -1.44
p-value: 0.56
# n_lags:  2
Núm. Observações:  21
Valores Críticos:
        1%: -3.79
        5%: -3.01
        10%: -2.65



Acidentes    A série NÃO é estacionária
Infracoes    A série NÃO é estacionária
```

Figura 86: Machine Learning/ Séries temporais.

Conforme verificado, o valor-p é maior que 0,05%, a série então é dada como não estacionária, e, por isso, iremos aplicar as diferenciações necessárias para deixarmos as séries estacionárias.

Diferenciação a série Acidentes:

```
# Diferenciação a série 'Acidentes'
df_Acidentes_diff = df['Acidentes'].diff(3).dropna()

fig = plt.figure(figsize=(8,4))
ts_ax = plt.subplot2grid((1, 1), (0, 0), colspan=1)

p_value = adfuller(df_Acidentes_diff)[1]
ts_ax.set_title('Analise da Série "Acidentes" com diff = 3\n Dickey-Fuller: p={0:.5f}'.format(p_value))
df_Acidentes_diff.plot(ax=ts_ax)
plt.show()
```

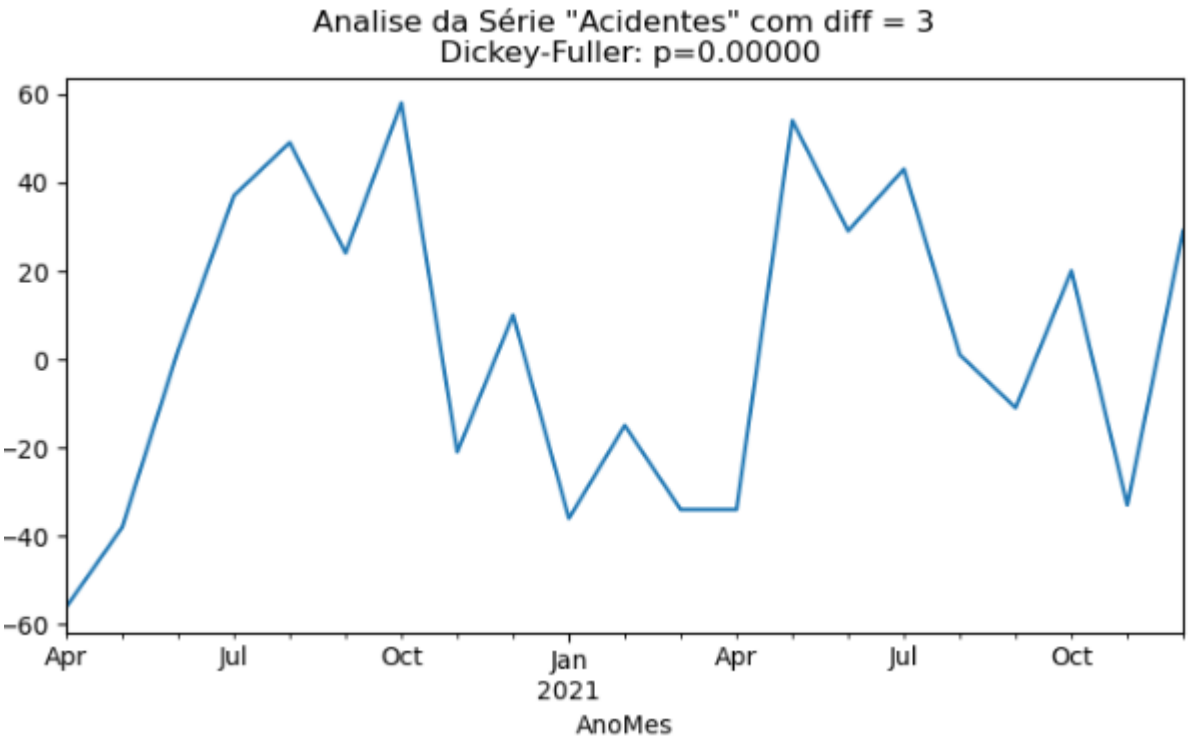


Figura 87: Machine Learning/ Séries temporais.

Foram aplicadas 3 diferenciações na série para obtermos o valor de p < 0,05%.

Diferenciação a série Infrações

```
# Diferenciação a série 'Infracoes'
df_Infracoes_diff = df['Infracoes'].diff(2).dropna()

fig = plt.figure(figsize=(8,4))
ts_ax = plt.subplot2grid((1, 1), (0, 0), colspan=1)

p_value = adfuller(df_Infracoes_diff)[1]
ts_ax.set_title('Analise da Série "Infracoes" com diff = 2\n Dickey-Fuller: p={0:.5f}'.format(p_value))
df_Infracoes_diff.plot(ax=ts_ax)
plt.show()
```

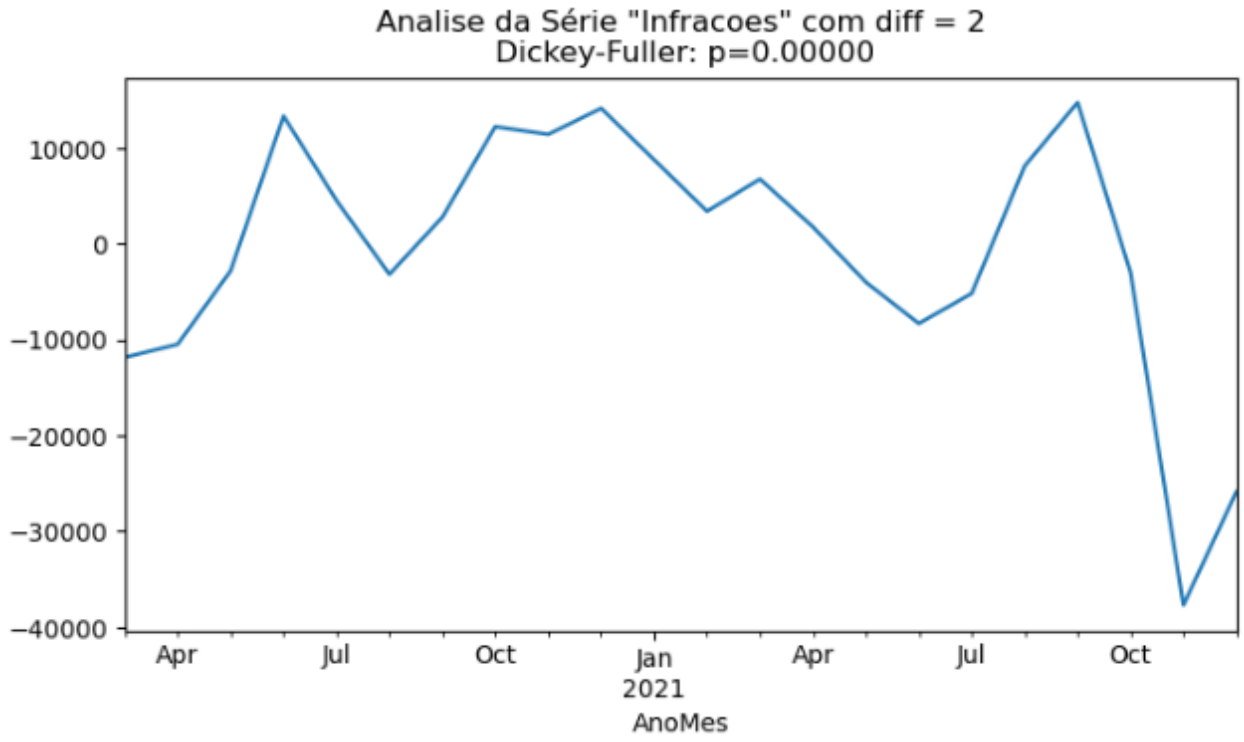


Figura 88: Machine Learning/ Séries temporais.

Foram aplicadas 2 diferenciações na série para obtermos o valor de p < 0,05%.

5.1 SARIMAX

5.1.1 Acidentes

Definindo Função para plotar o modelo e as métricas:

```
#Função para plotar o modelo e as métricas
def Plotagem(Base, Teste, Pred):
  # Raiz do erro quadratico médio
  rmse = round(mean_squared_error(Teste, Pred, squared=False),2)
  # Erro médio absoluto
  mae = round(mean_absolute_error(Teste, Pred),2)
  # Plotagem
  pd.concat([Base, Pred], axis = 1).plot(figsize=(16, 6), title = f"RMSE: {rmse}\n MAE: {mae}")
```

Figura 89: Machine Learning/ Séries temporais.

Dividindo bases

Para a dividir as bases, utilizamos o percentual de 80% para treino e 20% para teste.
A divisão em meses ficou da seguinte forma:
Base Total de **24 Meses**

80% Base de Treino = **19 Meses**

20% Base de Treino = **5 Meses**

```
#Divindo Dataset entre Treino e Teste
Acidentes_train = df.Acidentes.loc['2020-01-01':'2021-07-01']
Acidentes_test = df.Acidentes.loc['2021-08-01':]
```

```
# Define o intervalo que será utilizado para predição
inicio = len(Acidentes_train)
fim = len(Acidentes_train) + len(Acidentes_test) - 1
```

Figura 90: Machine Learning/ Séries temporais.

Montando modelo

Optou-se pelo modelo por permitir diferenciar dados por frequência sazonal, mas também por diferenciação não sazonal, nesse modelo acabamos não utilizando variáveis exógenas, e mesmo assim obtivemos uma boa performance.

Preparando modelo, criando predição e Plotagem de resultados:

```
# Prepara e ajusta o modelo SARIMAX
modelo_sarimax = SARIMAX(Acidentes_train, freq= 'MS', order=(1,1,1), seasonal_order=(1,1,1,12)).fit()

# Previsão do modelo
modelo_sarimax_prev = modelo_sarimax.predict(inicio, fim).rename("Previsão SARIMAX")

#Resultados
Plotagem(df['Acidentes'], Acidentes_test, modelo_sarimax_prev)
```

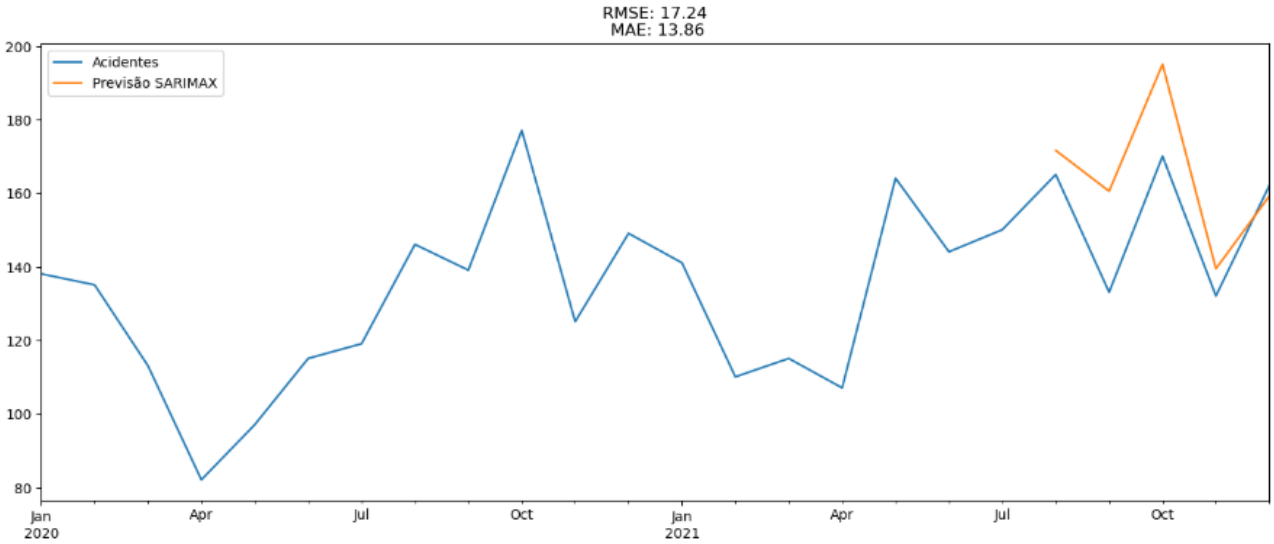


Figura 91: Machine Learning/ Séries temporais.

Plotando comparativa entre predição e base teste, para melhor visualização:

```
pd.concat([Acidentes_test,modelo_sarimax_prev], axis =1 ).plot(figsize=(8,4))
```

```
<Axes: >
```

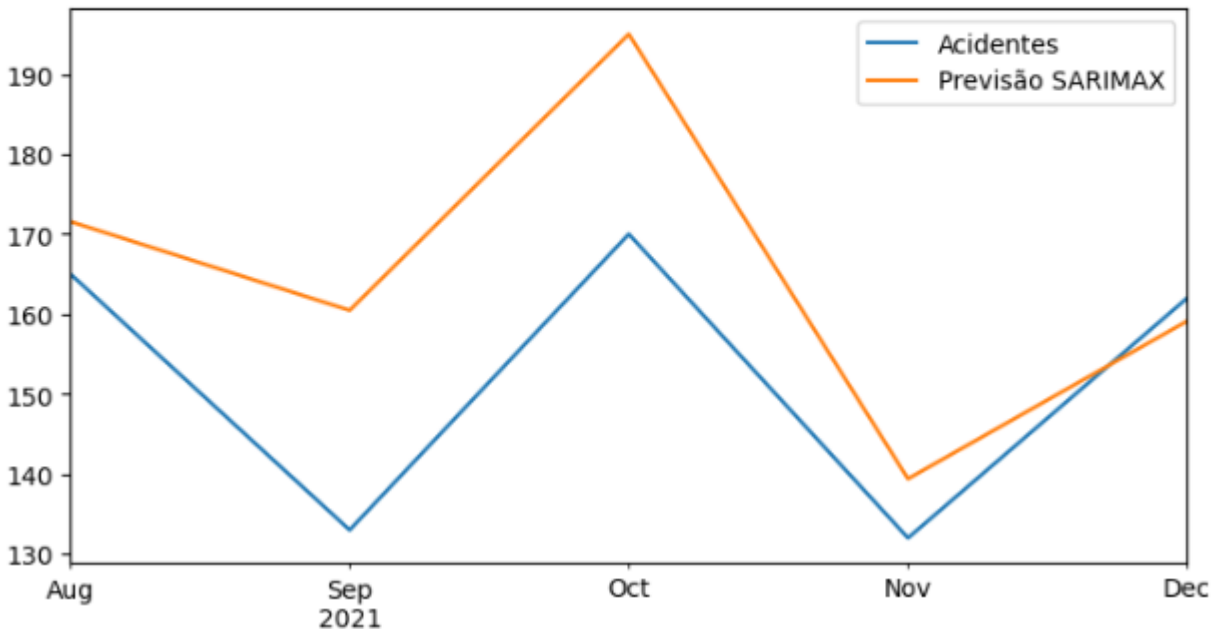


Figura 92: Machine Learning/ Séries temporais.

5.1.2 Infrações

Dividindo bases

Para a dividir as bases, utilizamos o percentual de 80% para treino e 20% para teste.
A divisão em meses ficou da seguinte forma:
Base Total de **24 Meses**

80% Base de Treino = **19 Meses**

20% Base de Treino = **5 Meses**

```
#Divindo Dataset entre Treino e Teste
Infra_train = df.Infracoes.loc['2020-01-01':'2021-07-01']
Infra_test = df.Infracoes.loc['2021-08-01':]
```

```
# Define o intervalor que será utilizado para predição
inicio = len(Infra_train)
fim = len(Infra_train) + len(Infra_test) - 1
```

Figura 93: Machine Learning/ Séries temporais.

Montando modelo

Optou-se pelo modelo por permitir diferenciar dados por frequência sazonal, mas também por diferenciação não sazonal, nesse modelo acabamos não utilizando variáveis exógenas, e mesmo assim obtivemos uma boa performance.

Preparando modelo, criando predição e Plotagem de resultados:

```
# Prepara e ajusta o modelo SARIMAX
modelo_sarimax = SARIMAX(Infra_train, freq= 'MS', order=(5,3,1), seasonal_order=(1,1,1,12)).fit()

# Previsão do modelo
modelo_sarimax_prev = modelo_sarimax.predict(inicio, fim).rename("Previsão SARIMAX")

#Resultados
Plotagem(df['Infracoes'], Infra_test, modelo_sarimax_prev)
```

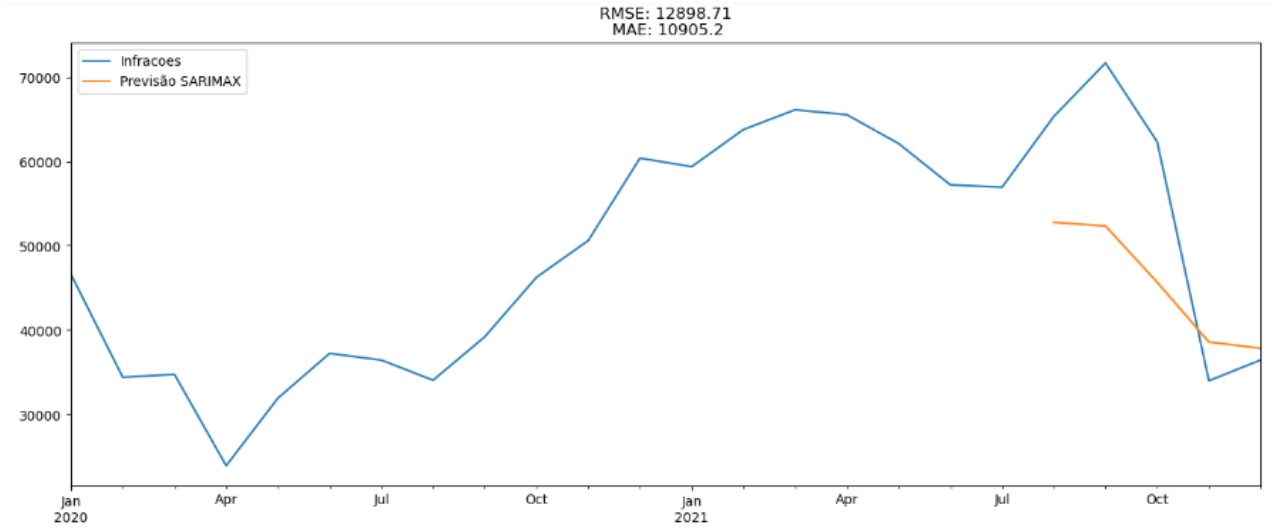


Figura 94: Machine Learning/ Séries temporais.

Plotando comparativa entre predição e base teste, para melhor visualização:

```
pd.concat([Infra_test,modelo_sarimax_prev], axis =1 ).plot(figsize=(8,4))
```

```
<Axes: >
```

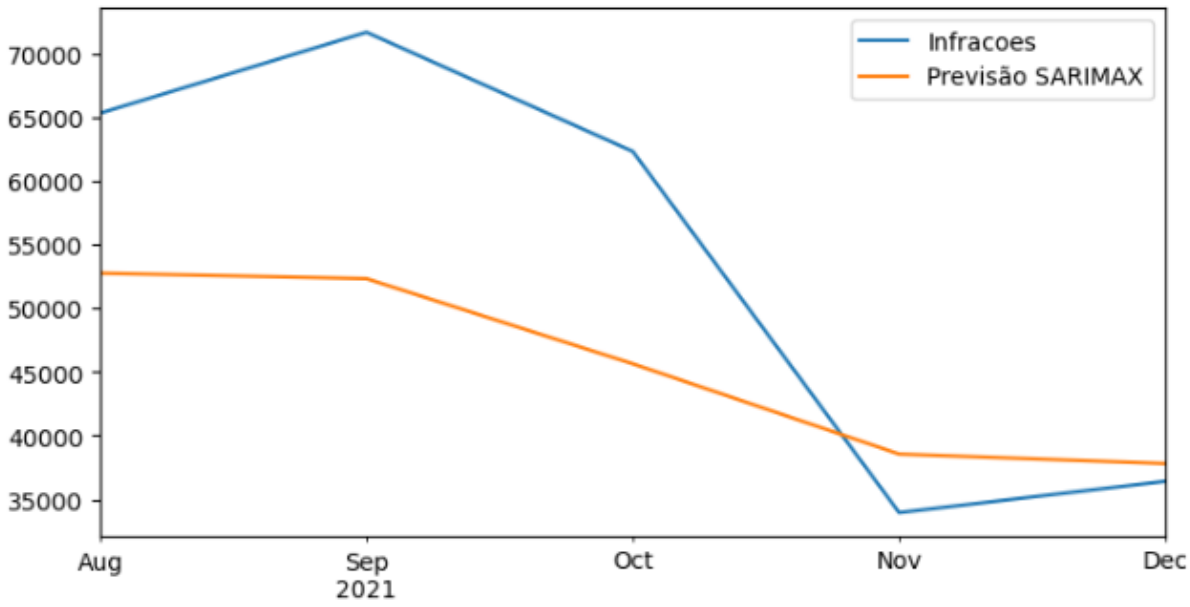


Figura 95: Machine Learning/ Séries temporais.

Mesmo sendo um período incomum conseguimos definir que tivemos boa compatibilidade das predições dos modelos com as bases de testes. Utilizaremos os resultados das métricas de performance para comparar com os demais modelos no

final.

5.2 ARIMA

5.2.1 Acidentes

Dividindo as bases de treino e teste

```
#Dividindo os datasets de Treino e teste em 80/20
Acidentes_train = df.Acidentes.loc['2020-01-01':'2021-07-01']
Acidentes_test = df.Acidentes.loc['2021-08-01':]
```

Figura 96: Machine Learning/ Séries temporais

Confirma através do teste que as series não são estacionárias

```
Base = Acidentes_train.values
Resultado = adfuller(Base)
print('ADF Statistic: %f' % Resultado[0])
print('p-value:%f'% Resultado[1])
print("Critical Values:")
for key, values in Resultado[4].items():
    print('\t%s: %.3f' % (key, values))
```

```
ADF Statistic: -2.828547
p-value:0.054310
Critical Values:
        1%: -4.069
        5%: -3.127
        10%: -2.702
```

Figura 97: Machine Learning/ Séries temporais

Como podemos ver, o valor-p não é menor que 0,05%, a série de treino então é dada como não estacionária, precisaremos fazer a diferenciação.

Aplicando a diferenciação na série e removendo dados nulos

Agora vamos aplicar a diferenciação da série e validarmos se teremos uma série estacionária

```
# Aplicando a diferenciação na série e removendo dados nulos
Acidentes_train_diff = Acidentes_train.diff(2)
Acidentes_train_diff = Acidentes_train_diff.dropna()
```

```
# Confirma através do teste que as series não são estacionárias
Base = Acidentes_train_diff.values
Resultado = adfuller(Base)
print('ADF Statistic: %f' % Resultado[0])
print('p-value:%f'% Resultado[1])
print("Critical Values:")
for key, values in Resultado[4].items():
    print('\t%s: %.3f' % (key, values))
```

```
ADF Statistic: -4.066544
p-value:0.001099
Critical Values:
        1%: -4.223
        5%: -3.189
        10%: -2.730
```

Figura 98: Machine Learning/ Séries temporais

Como podemos observar, o valor de p é menor que 0,05%, logo temos uma série estacionária.

Decompondo série diferenciada

```
# Decompondo a série diferenciada
decomposicao = seasonal_decompose(Acidentes_train_diff, period = 8)
Plot = decomposicao.plot()
```

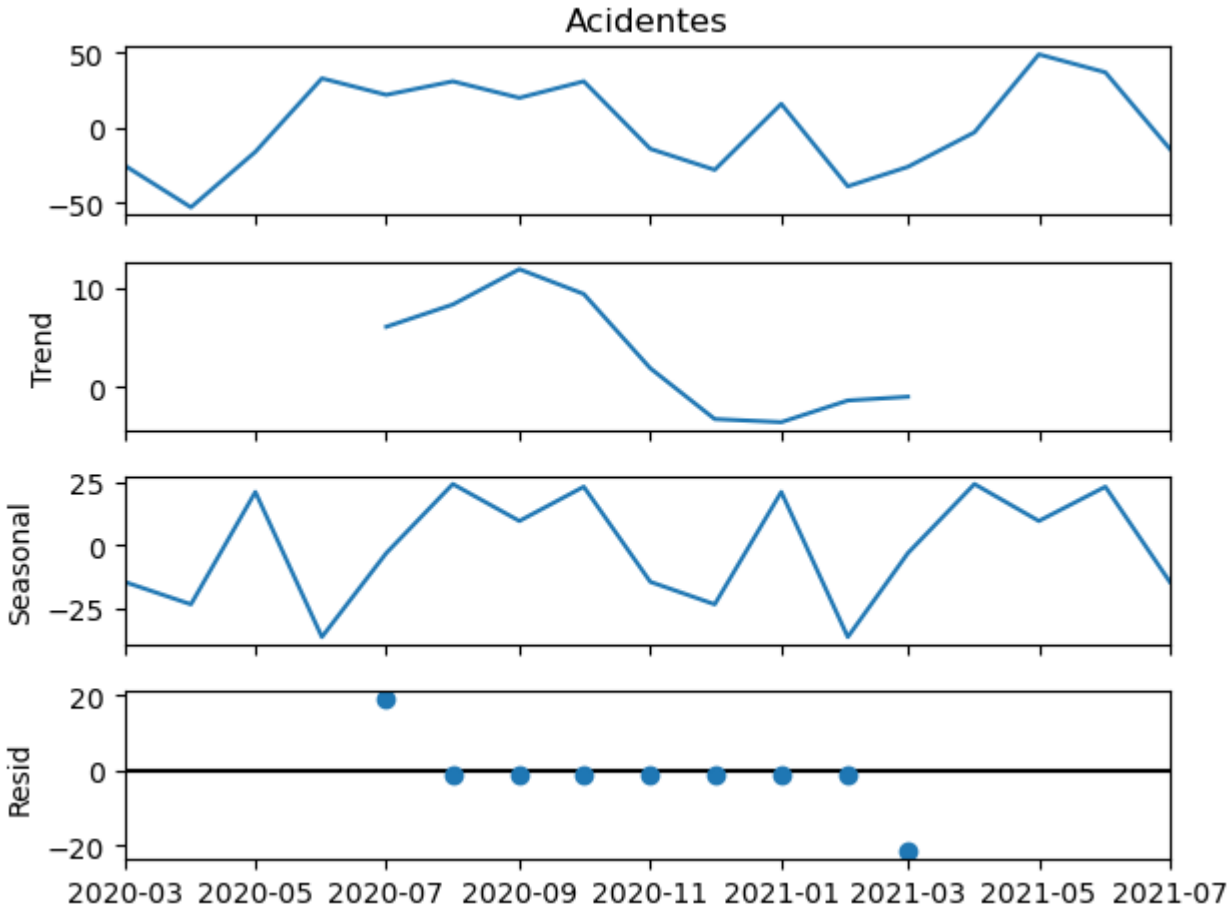


Figura 99: Machine Learning/ Séries temporais

Plotando Gráficos de ACF e PACF

```
# Vamos plotar os graficos de ACF e PACF
plot_acf(Acidentes_train, title = f"Autocorrelação Acidentes")
plot_pacf(Acidentes_train, lags = 8, title = f"Autocorrelação Parcial Acidentes")
plt.show()
```

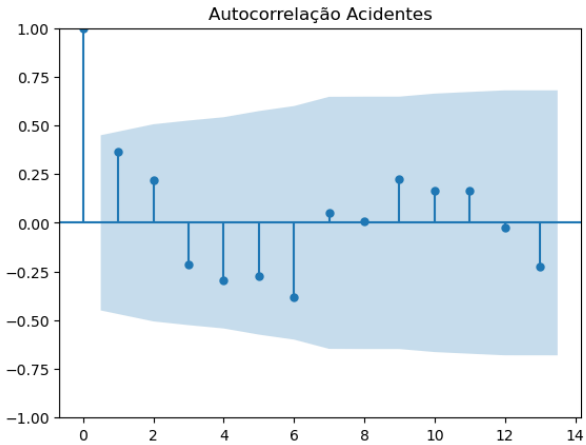


Figura 100: Machine Learning/ Séries temporais

Os valores de Autocorrelação Acidentes ficam dentro do intervalo de confiança a partir do Lag 1. Sendo assim, o valor p estará em 0.

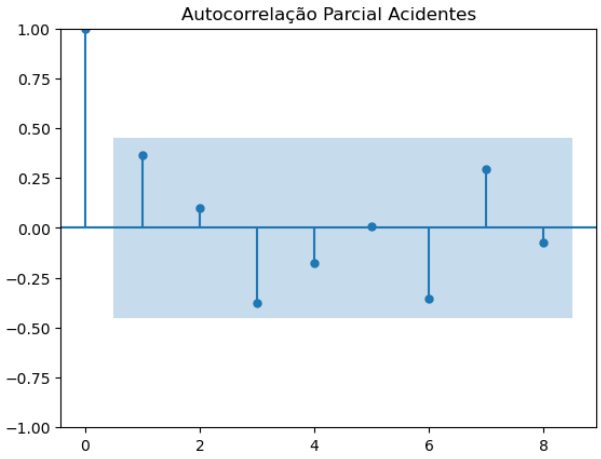


Figura 101: Machine Learning/ Séries temporais

Observando a Autocorrelação Parcial, em Acidentes apenas o primeiro valor tem uma Correlativa alta significante diante os outros. O valor de q estará em 0.

Montando modelo

```
#Criando o modelo autoarima
autoarima_Acidentes = auto_arima(Acidentes_train, trace=True, seasonal = True, stationary = False, start_p = 0 , start_q = 0 ,m=
autoarima_Acidentes_Pred= autoarima_Acidentes.fit(Acidentes_train)
autoarima_Acidentes_Pred.aic()
```

```
 ARIMA(0,2,0)(0,0,0)[1]             : AIC=179.557, Time=0.05 sec
 ARIMA(0,2,1)(0,0,0)[1]             : AIC=inf, Time=0.05 sec
 ARIMA(0,2,2)(0,0,0)[1]             : AIC=inf, Time=0.07 sec
 ARIMA(0,2,3)(0,0,0)[1]             : AIC=inf, Time=0.06 sec
 ARIMA(1,2,0)(0,0,0)[1]             : AIC=169.453, Time=0.03 sec
 ARIMA(1,2,1)(0,0,0)[1]             : AIC=inf, Time=0.06 sec
 ARIMA(1,2,2)(0,0,0)[1]             : AIC=inf, Time=0.08 sec
 ARIMA(1,2,3)(0,0,0)[1]             : AIC=inf, Time=0.13 sec
 ARIMA(2,2,0)(0,0,0)[1]             : AIC=169.398, Time=0.04 sec
 ARIMA(2,2,1)(0,0,0)[1]             : AIC=inf, Time=0.09 sec
 ARIMA(2,2,2)(0,0,0)[1]             : AIC=inf, Time=0.18 sec
 ARIMA(2,2,3)(0,0,0)[1]             : AIC=inf, Time=0.26 sec
 ARIMA(3,2,0)(0,0,0)[1]             : AIC=171.246, Time=0.05 sec
 ARIMA(3,2,1)(0,0,0)[1]             : AIC=inf, Time=0.08 sec
 ARIMA(3,2,2)(0,0,0)[1]             : AIC=inf, Time=0.19 sec
 ARIMA(4,2,0)(0,0,0)[1]             : AIC=171.396, Time=0.06 sec
 ARIMA(4,2,1)(0,0,0)[1]             : AIC=inf, Time=0.09 sec
 ARIMA(5,2,0)(0,0,0)[1]             : AIC=172.067, Time=0.13 sec

Best model:  ARIMA(2,2,0)(0,0,0)[1]
Total fit time: 1.723 seconds
```

Figura 102: Machine Learning/ Séries temporais

O modelo de melhor performance, visando o menor AIC foi o ARIMA(2,2,0)(0,0,0)[1].

Preparando modelo, criando predição e Plotagem de resultados:

```
# Previsão do modelo
Forecast = autoarima_Acidentes_Pred.predict()
Forecast = pd.DataFrame(Forecast, index = Acidentes_test.index, columns= ['Predicao Auto Arima'])
#Resultados
Plotagem(df['Acidentes'], Acidentes_test, Forecast)
```

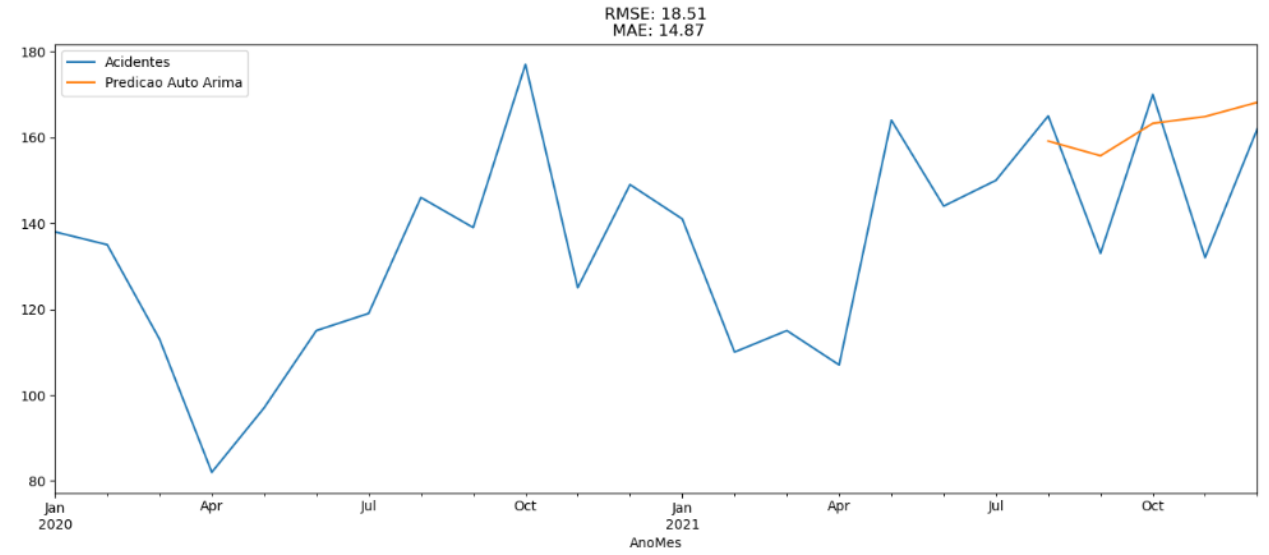


Figura 103: Machine Learning/ Séries temporais

Plotando comparativa entre predição e base teste, para melhor visualização:

```
: pd.concat([Acidentes_test,Forecast], axis =1 ).plot(figsize=(8,4))
```

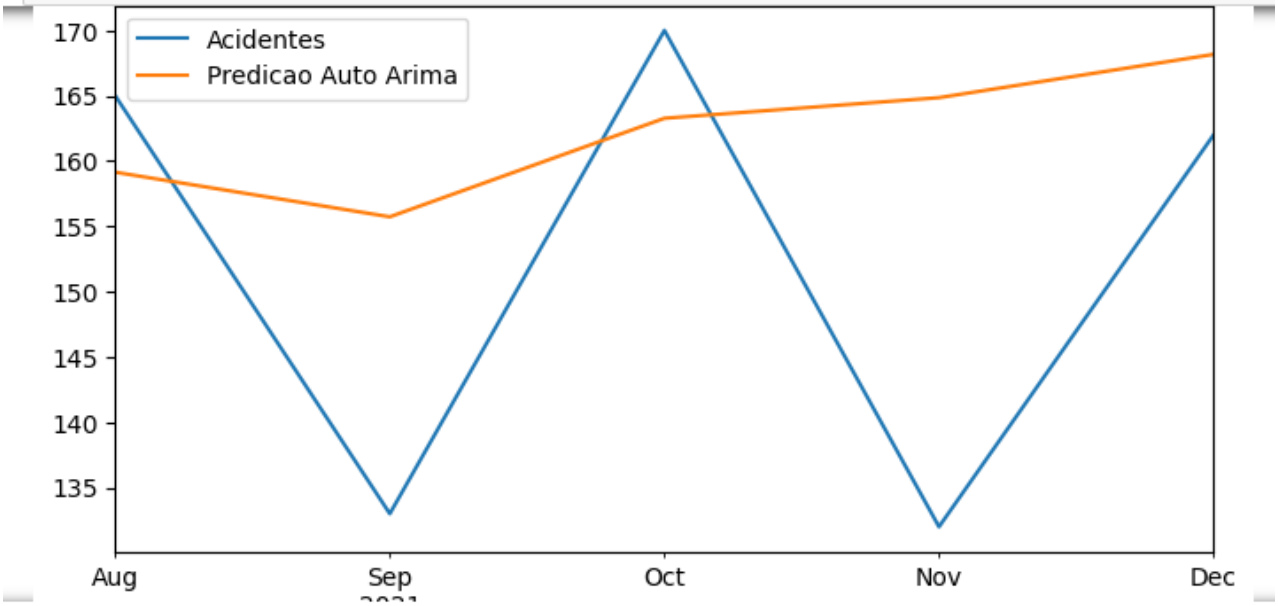


Figura 104: Machine Learning/ Séries temporais

5.2.2 Infrações

Dividindo as bases de treino e teste

```
#Dividindo os datasets de Treino e teste em 80/20
Infracoes_train = df.Infracoes.loc['2020-01-01':'2021-07-01']
Infracoes_test = df.Infracoes.loc['2021-08-01':]
```

Figura 105: Machine Learning/ Séries temporais

Confirma através do teste que as series não são estacionárias

```
# Confirma através do teste que as series não são estacionárias
Base = Infracoes_train.values
Resultado = adfuller(Base)
print('ADF Statistic: %f' % Resultado[0])
print('p-value:%f'% Resultado[1])
print("Critical Values:")
for key, values in Resultado[4].items():
    print('\t%s: %.3f' % (key, values))
```

```
ADF Statistic: -2.219505
p-value:0.199243
Critical Values:
        1%: -4.223
        5%: -3.189
        10%: -2.730
```

Figura 107: Machine Learning/ Séries temporais

Como podemos ver, o valor-p não é menor que 0,05%, a série de treino então é dada como não estacionária, precisaremos fazer a diferenciação.

Aplicando a diferenciação na série e removendo dados nulos

Agora vamos aplicar a diferenciação da série e validarmos se teremos uma série estacionária

```
# Aplicando a diferenciação na série e removendo dados nulos
Infracoes_train_diff = Infracoes_train.diff(3)
Infracoes_train_diff = Infracoes_train_diff.dropna()
```

```
# Confirma através do teste que as series não são estacionárias
az = Infracoes_train_diff.values
result2 = adfuller(az)
print('ADF Statistic: %f' % result2[0])
print('p-value:%f'% result2[1])
print("Critical Values:")
for key, values in result2[4].items():
    print('\t%s: %.3f' % (key, values))
```

```
ADF Statistic: -4.917525
p-value:0.000032
Critical Values:
        1%: -4.473
        5%: -3.290
        10%: -2.772
```

Figura 106: Machine Learning/ Séries temporais

Como podemos observar, o valor de p é menor que 0,05%, logo temos uma série estacionária.

Decompondo série diferenciada

```
# Decompondo a série diferenciada
decomposicao3 = seasonal_decompose(Infracoes_train_diff, period = 8)
imagem3 = decomposicao3.plot()
```

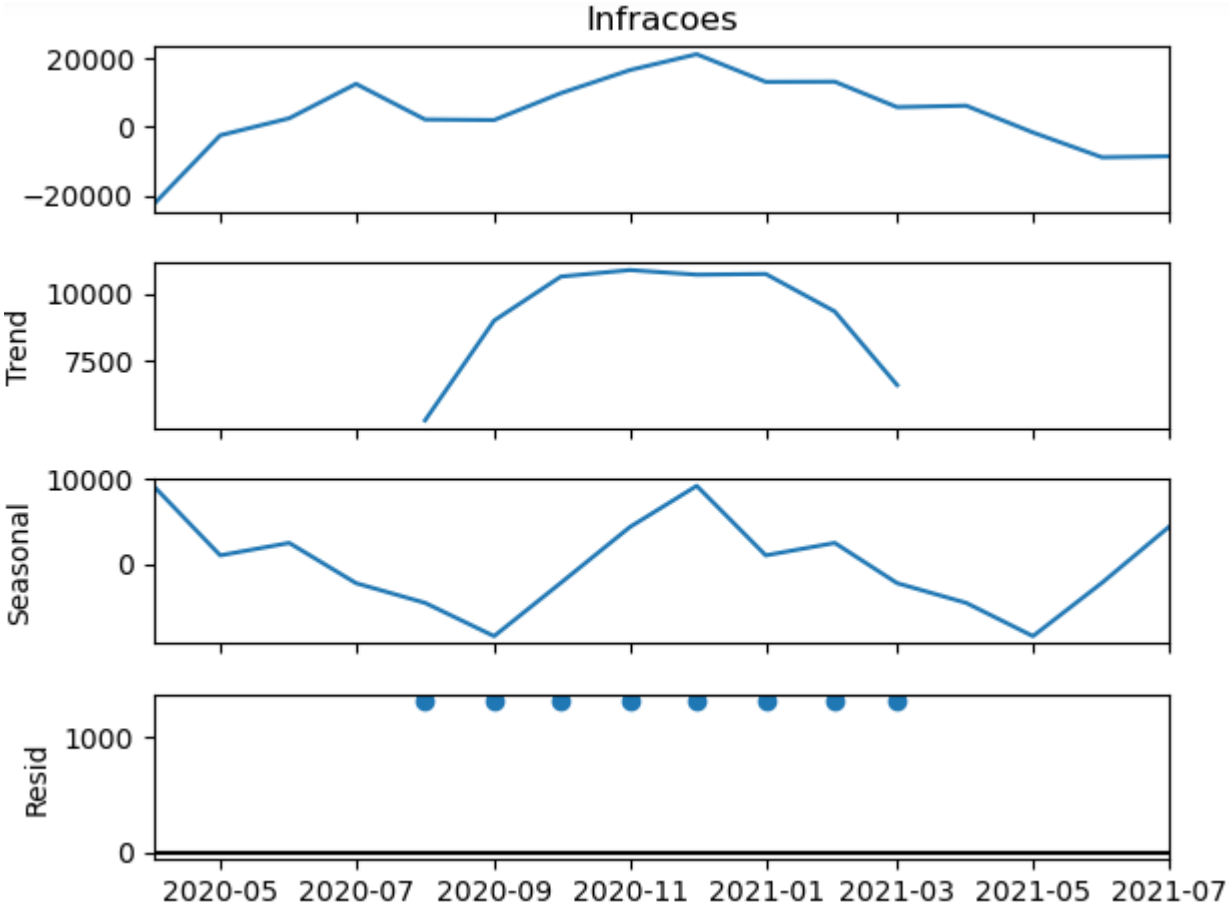


Figura 107: Machine Learning/ Séries temporais

Plotando Gráficos de ACF e PACF

```
# Vamos plotar os graficos de ACF e PACF

plot_acf(Infracoes_train, title = f"Autocorrelação Infracoes")
plot_pacf(Infracoes_train, lags = 8, title = f"Autocorrelação Parcial Infracoes")

plt.show()
```

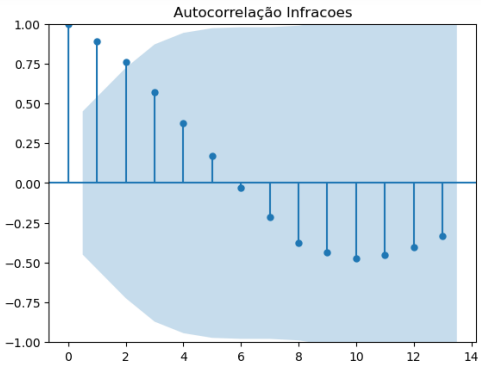


Figura 108: Machine Learning/ Séries temporais

Observando a Autocorrelação Infrações o valor de p iniciará entre em 2.

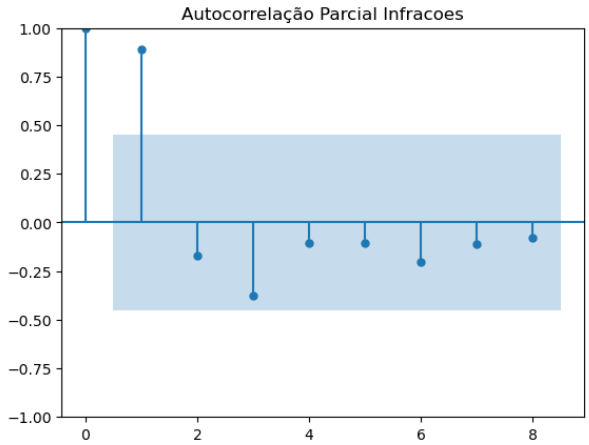


Figura 109: Machine Learning/ Séries temporais

No caso da Autocorrelação Parcial Infrações o valor de q estará entre em 1.

Montando modelo

```
#Criando o modelo do autoarima
autoarima_Infracoes = auto_arima(Infracoes_train, trace=True, seasonal = True, stationary = False, start_p = 2 , start_q = 1 ,m=
autoarima_Infracoes_Pred = autoarima_Infracoes.fit(Infracoes_train)
autoarima_Infracoes_Pred.aic()
```

```
 ARIMA(0,3,0)(0,0,0)[1]             : AIC=351.098, Time=0.03 sec
 ARIMA(0,3,1)(0,0,0)[1]             : AIC=inf, Time=0.04 sec
 ARIMA(0,3,2)(0,0,0)[1]             : AIC=inf, Time=0.06 sec
 ARIMA(1,3,0)(0,0,0)[1]             : AIC=339.460, Time=0.03 sec
 ARIMA(1,3,1)(0,0,0)[1]             : AIC=inf, Time=0.08 sec
 ARIMA(1,3,2)(0,0,0)[1]             : AIC=inf, Time=0.14 sec
 ARIMA(2,3,0)(0,0,0)[1]             : AIC=344.230, Time=0.03 sec
 ARIMA(2,3,1)(0,0,0)[1]             : AIC=inf, Time=0.11 sec
 ARIMA(2,3,2)(0,0,0)[1]             : AIC=inf, Time=0.19 sec
 ARIMA(3,3,0)(0,0,0)[1]             : AIC=337.830, Time=0.05 sec
 ARIMA(3,3,1)(0,0,0)[1]             : AIC=327.684, Time=0.06 sec
 ARIMA(3,3,2)(0,0,0)[1]             : AIC=inf, Time=0.20 sec
 ARIMA(4,3,0)(0,0,0)[1]             : AIC=325.971, Time=0.05 sec
 ARIMA(4,3,1)(0,0,0)[1]             : AIC=325.459, Time=0.16 sec
 ARIMA(5,3,0)(0,0,0)[1]             : AIC=325.371, Time=0.32 sec

Best model:  ARIMA(5,3,0)(0,0,0)[1]
Total fit time: 1.567 seconds

325.3707013949057
```

Figura 110: Machine Learning/ Séries temporais

O modelo de melhor performance, visando o menor AIC foi o ARIMA(5,3,0)(0,0,0)[1]

.

Preparando modelo, criando predição e Plotagem de resultados:

```
Forecast = autoarima_Infracoes_Pred.predict()
Forecast = pd.DataFrame(Forecast, index = Infracoes_test.index, columns= ['Predicao Auto Arima'])
Plotagem(df['Infracoes'], Infracoes_test, Forecast)
```

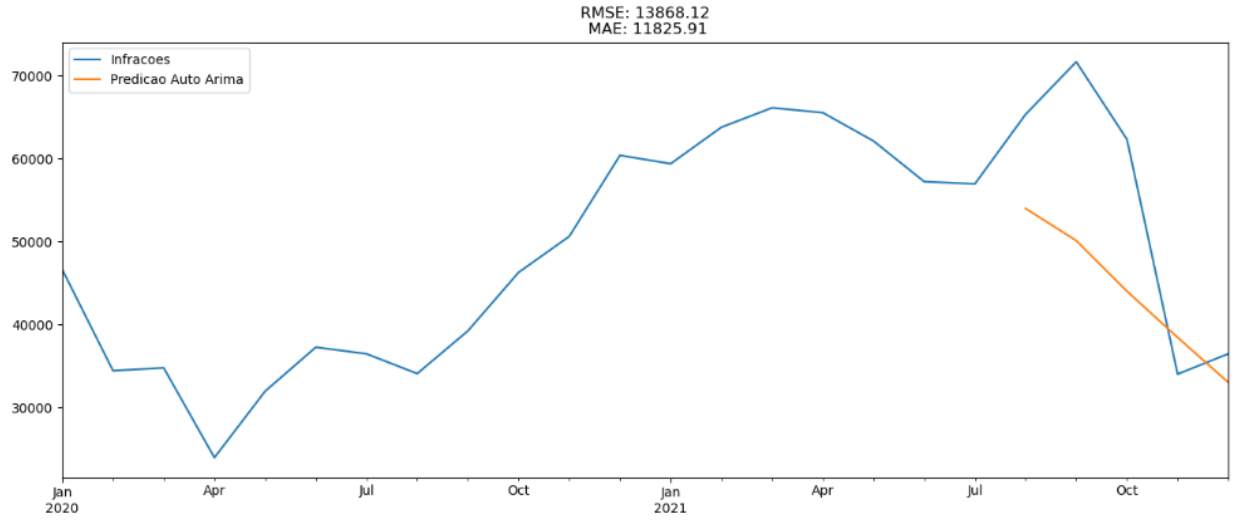


Figura 111: Machine Learning/ Séries temporais

Plotando comparativa entre predição e base teste, para melhor visualização:

```
pd.concat([Infracoes_test,Forecast], axis =1 ).plot(figsize=(8,4));
```

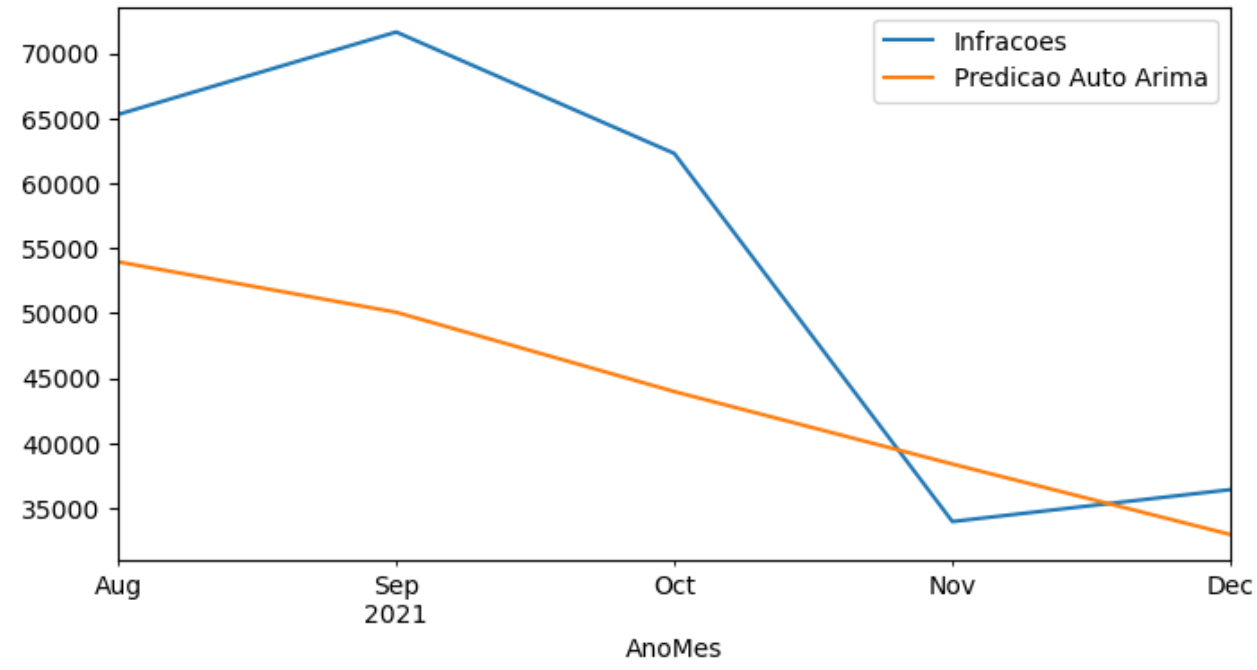


Figura 112: Machine Learning/ Séries temporais

Mesmo sendo um período incomum conseguimos definir que tivemos uma

performance baixa em relação as predições dos modelos. Utilizaremos os resultados das métricas de performance para comparar com os demais modelos no final.

5.3 Holt-Winters

5.3.1 Acidentes

Como já foi analisado anteriormente a sazonalidade da série, temos as informações necessárias para partirmos diretamente para montagem do modelo.

Montando modelo

Optou-se pelo modelo pela capacidade de lidar com padrões sazonais complicados, adicionando os efeitos de inclinação e sazonalidade.

Preparando modelo, criando predição e Plotagem de resultados:

```
# Aplicando modelo Holt-Winters - Acidentes
HW_acid = ExponentialSmoothing(Acidentes_train, seasonal_periods=7, trend="add", seasonal='add',damped=True, use_boxcox=True, ini
HW_Pred = HW_acid.fit(optimized = True);
```

```
forecast_HW_Pred = HW_Pred.forecast(5).rename("Holt Winters Seasonal - Acidentes")
Plotagem(df['Acidentes'], Acidentes_test, forecast_HW_Pred)
```

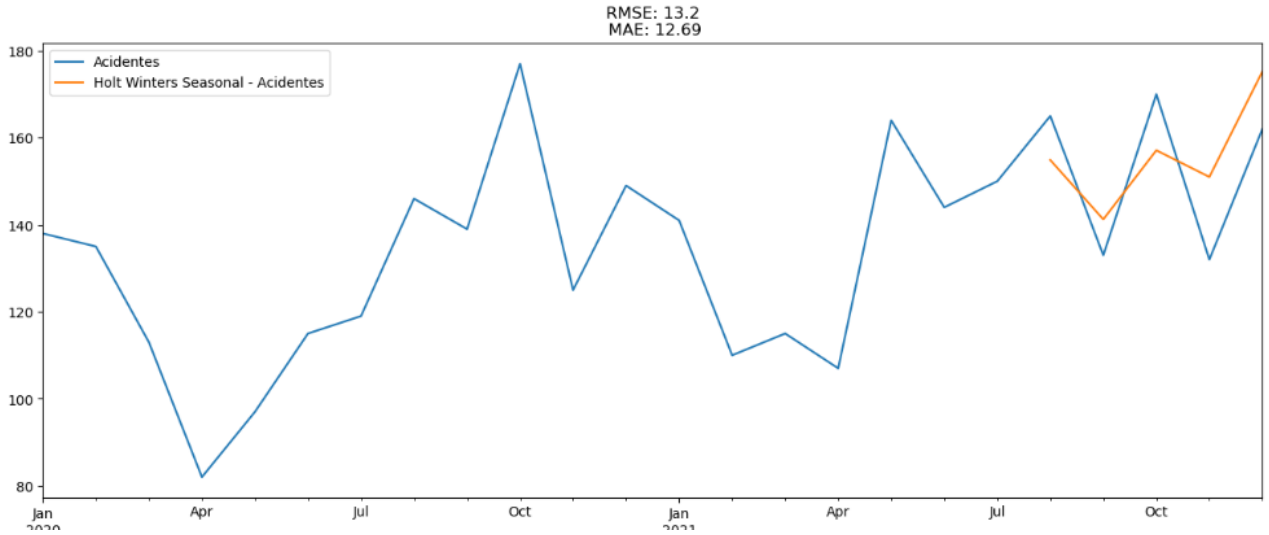


Figura 113: Machine Learning/ Séries temporais.

Plotando comparativa entre predição e base teste, para melhor visualização:

```
pd.concat([Acidentes_test,forecast_HW_Pred], axis =1 ).plot(figsize=(8,4));
```

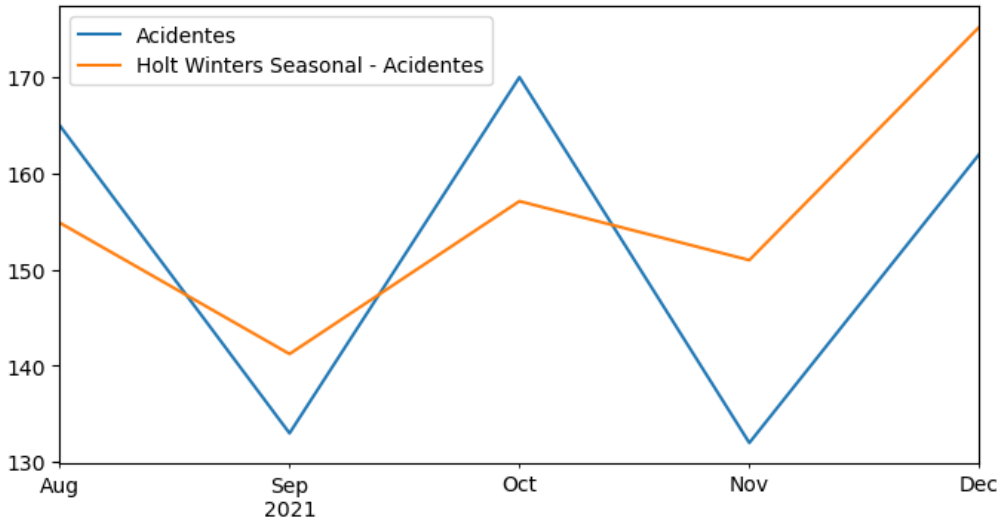


Figura 114: Machine Learning/ Séries temporais.

5.3.2 Infrações

Montando modelo

Preparando modelo, criando predição e Plotagem de resultados:

```
# Aplicando modelo Holt-Winters - Infrações

HW_Infra = ExponentialSmoothing(Infracoes_train, seasonal_periods=9,trend= "add", seasonal='add', use_boxcox=True, initializatior
HW_Infra_Pred = HW_Infra.fit(optimized = True)
```

```
forecast_HW_Pred = HW_Infra_Pred.forecast(5).rename("Holt Winters Seasonal - Infracoes")
#Resultados
Plotagem(df['Infracoes'], Infracoes_test, forecast_HW_Pred)
```

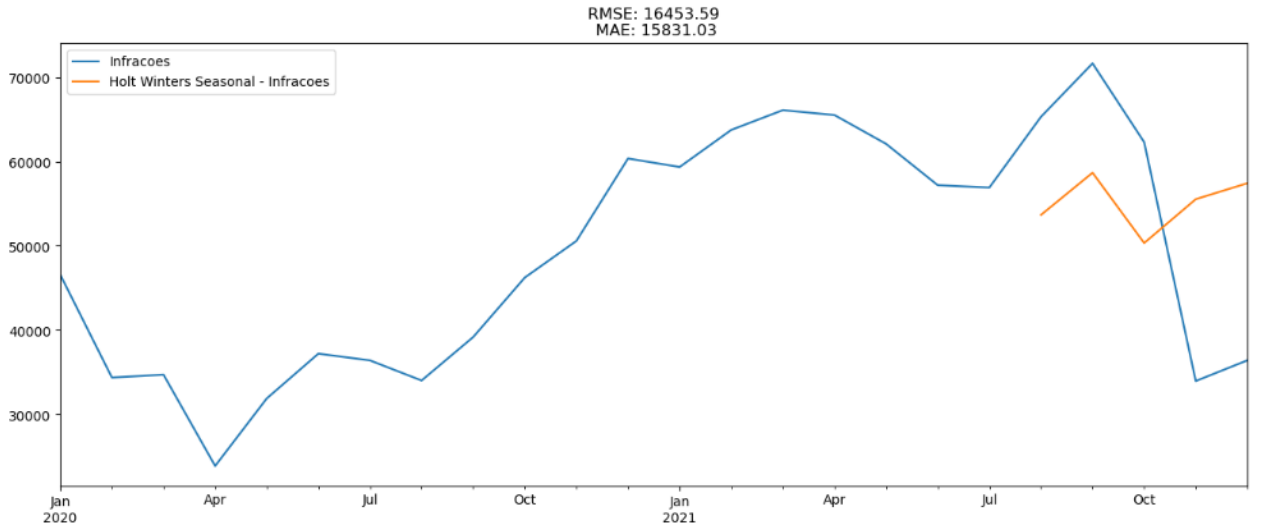


Figura 115: Machine Learning/ Séries temporais.

Plotando comparativa entre predição e base teste, para melhor visualização:

```
pd.concat([Infracoes_test,forecast_HW_Pred], axis =1 ).plot(figsize=(8,4));
```

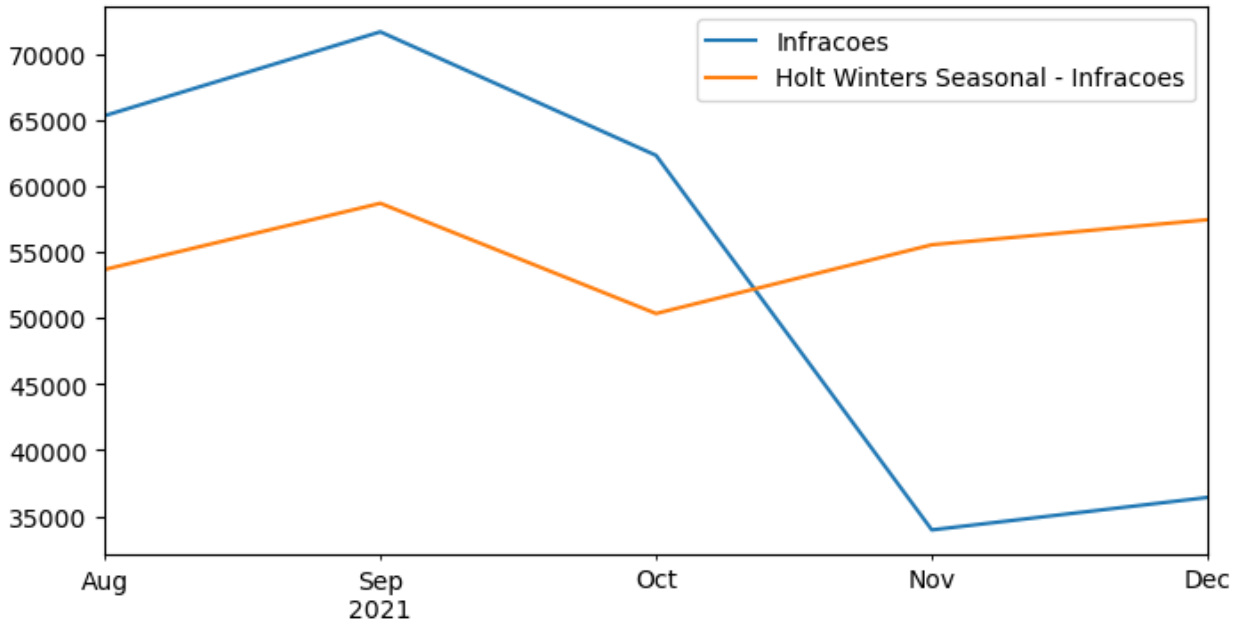


Figura 116: Machine Learning/ Séries temporais.

Mesmo sendo um período incomum conseguimos definir que tivemos uma boa performance em relação as predições dos modelos, tivemos uma maior precisão na série de acidentes, não tivemos a mesma performance na base de infrações. Utilizaremos os resultados das métricas de performance para comparar com os demais modelos no final.

6. Interpretação dos Resultados

Analisando os datasets de ocorrências de infrações e acidentes nas rodovias de São Paulo de 2020 à 2021, conseguimos extrair bastantes insights:

Vimos que o no segundo semestre dos anos de **2020** e **2021** são os períodos em que há mais ocorrências de infrações de trânsito nas rodovias de SP, sendo dezembro e setembro os meses com maior quantidade de infrações, e outubro o Mês com maior quantidade de acidentes,

podemos definir essa alta de infrações no mês de dezembro devido o período de festas, e férias das escolas, o que aumentam muito mais as locomoções com desatenções pelas vias nessa época do ano, devido ao ânimo das celebrações e festividades (vale ressaltar que tivemos uma queda nas ocorrências nos primeiros meses de 2020, devido às restrições de Lockdown, que foi se excedendo durante o final de 2020 e início de 2021).

As ocorrências de infrações e acidentes aumentam na chegada do final de semana, tendo as maiores médias de ocorrências na sexta-feira e sábado, o inicio do final de semana em São Paulo é marcado por haver muitas movimentações em baladas e festas, onde uma vez havendo um ambiente de celebração e animação, aumentam os riscos da falta de atenção à condução, que como já destacamos, é a maior causa de ocorrências de acidentes em São Paulo, seguido pela Velocidade incompatível.

Afunilando os casos de acidentes conseguimos analisar que 45% dos acidentes envolvem 2 pessoas, 43% envolvem 1 pessoa e 12% envolvem 3 pessoas, onde podemos definir que a maioria das ocorrências envolvem múltiplas pessoas, reforçando o ponto que destacamos acima, onde citamos as locomoções para festividades e lazer onde envolvem mais que uma pessoa.

No caso das infrações, as mais cometidas são: Estacionar em desacordo com as posições estabelecidas no CTB, conduzir ciclo sem segurar o guidom com ambas as mãos e Usar veículo para arremessar sobre os veículos: água ou detritos. A severidade das ocorrências em foram definidas como Média gravidade, mas podemos dizer que são totais potenciadores de gerar um acidente pela falta de atenção.

Os horários do dia de maior ocorrência de acidentes e infrações é o período da Tarde (12h às 17h59), é quando há um aumento do fluxo de veículos, por conta das pessoas que estão voltando do trabalho para casa, os motoristas já estão cansados, seja do batente no escritório, seja das horas seguidas ao volante, também podemos destacar o final da tarde como um horário de saída para pessoas realizarem as atividades finais do dia. Tudo isso é um fator resultante para as ocorrências de infrações e acidentes.

E por último conseguimos mencionar as vias com maior periculosidade em São Paulo através das maiores quantidades de ocorrências de infrações e acidentes nas BRs: BR116, BR381 e BR153.

Interpretação dos resultados de Machine Learning

Os modelos foram implementados buscando uma melhor performance através da redução da RMSE (Raiz do erro médio quadrático), que mede a diferença média entre os valores previstos por um modelo de previsão e os valores reais, e a redução do MAE (Erro Médio Absoluto) que mede a média da diferença absoluta entre os valores previstos pelo modelo e os valores observados, quanto menos os valores de RMSE e MAE melhor o desempenho do modelo.

Os resultados das métricas calculadas em cada modelo para as bases foram coletadas:

	Métricas	
Modelos	**Acidentes**	**Infrações**
SARIMAX	MAE: 13.86 RMSE: 17.24	MAE: 10905.2 RMSE: 12898.71
Exponential Smoothing (Holt Winters)	MAE: 12.69 RMSE: 13.2	MAE: 15831.03 RMSE: 16453.59
Auto ARIMA	MAE: 14.87 RMSE: 18.57	MAE: 11825.91 RMSE: 13868.12

Figura 117: Interpretação de Resultados/ Modelos.

SARIMAX

modelo SARIMAX, uma variação do modelo ARIMA que permite lidar com séries temporais com tendência de crescimento, sazonalidade marcada e a consideração de variáveis exógenas.

De todos os modelos testados, nota-se que ele teve a melhor aderência do modelo para as séries, que esta ligado a facilidade do modelo para com bases que possuem grandes tendências e sazonalidades. Obteve a melhor taxa de MAE e RMSE para série de infrações, e uma performance razoável na série de acidentes.

Modelo Sarimax - Acidentes SP

```
pd.concat([Acidentes_test,modelo_sarimax_prev], axis =1 ).plot(figsize=(8,4))
```

```
<Axes: >
```

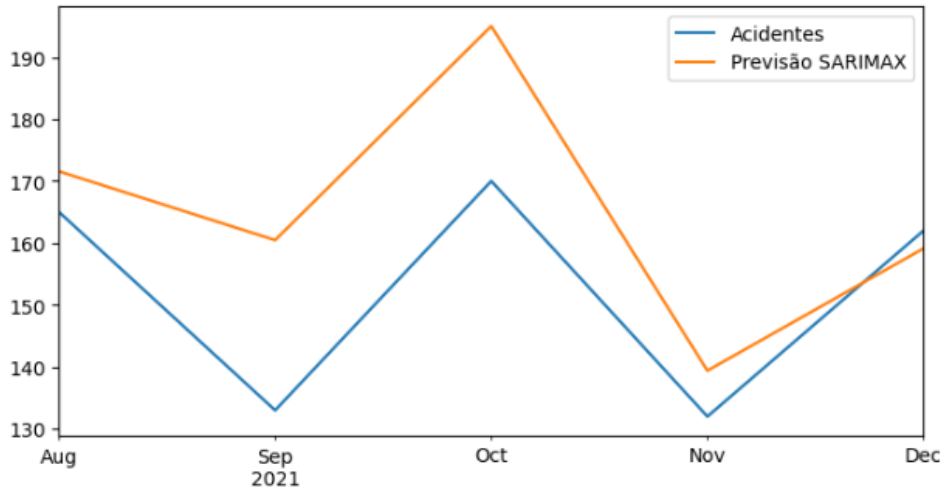


Figura 118: Interpretação de Resultados/ Modelos.

Modelo Sarimax – Infrações SP

```
pd.concat([Infra_test,modelo_sarimax_prev], axis =1 ).plot(figsize=(8,4))
```

```
<Axes: >
```

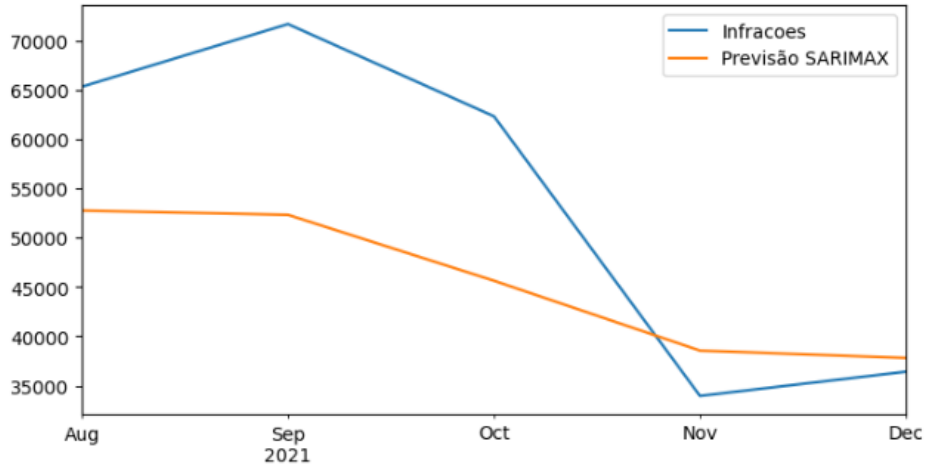


Figura 119: Interpretação de Resultados/ Modelos.

Holt-Winters

O Holt-Winters é um modelo de suavização exponencial para representar séries temporais
tendências e sazonalidade. Este é um dos métodos mais utilizados para previsões de curto prazo demanda devido à sua simplicidade, baixos custos operacionais, boa precisão, capacidade de ajustar-se automaticamente às mudanças na série.

Observando os resultados apresentados das métricas, o modelo obteve a melhor performance para a série de acidentes, porém teve a pior performance sobre a série de infrações, isto pode ser compreendido pelo fato do modelo ser apropriado para série sazonal, e o a base de infrações não apresentar muita sazonalidade, por isso que seu índice foi o de menor performance.

Holt-Winters - Acidentes

```
pd.concat([Acidentes_test,forecast_HW_Pred], axis =1 ).plot(figsize=(8,4));
```

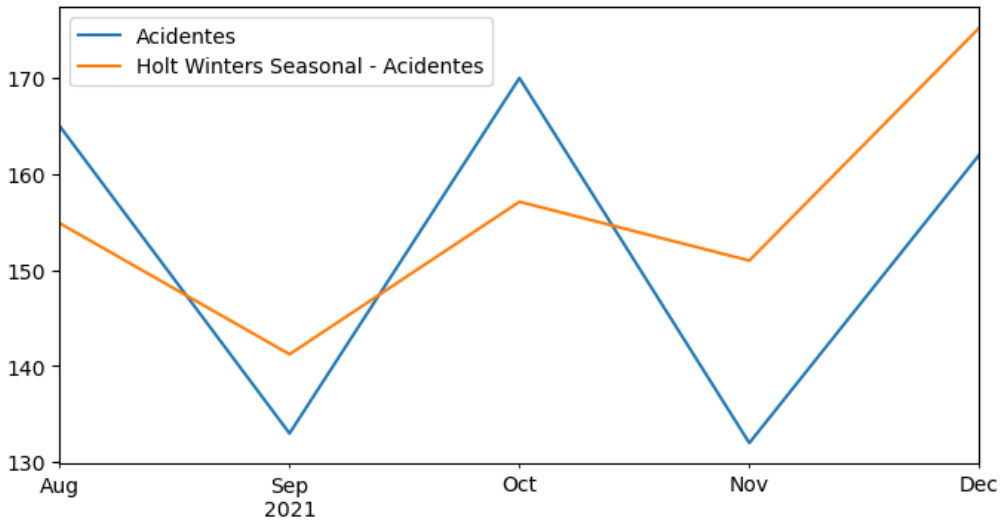


Figura 120: Interpretação de Resultados/ Modelos.

Holt Winters - Infrações

```
pd.concat([Infracoes_test,forecast_HW_Pred], axis =1 ).plot(figsize=(8,4));
```

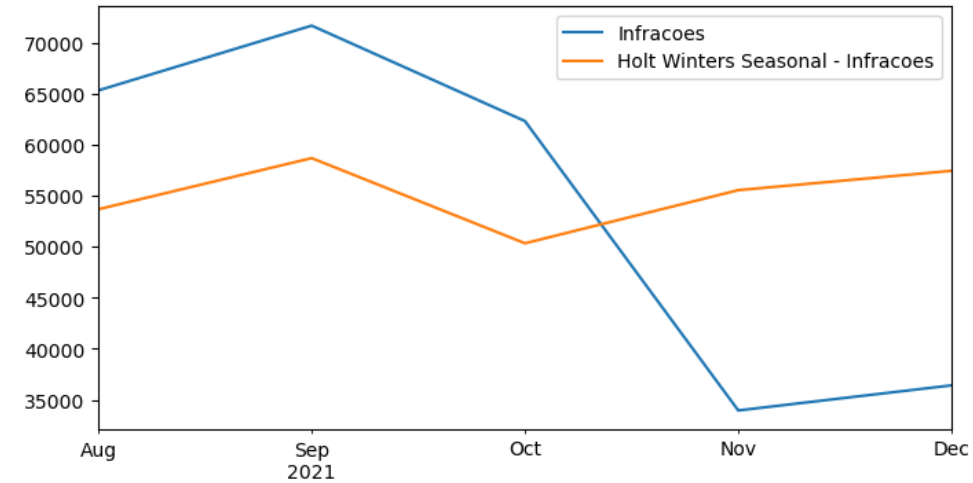


Figura 121: Interpretação de Resultados/ Modelos.

ARIMA

Para o modelo utilizamos o AutoArima que combina testes de raiz unitária, minimização do AIC e MLE para obter um modelo ARIMA. Tivemos que diferenciar as bases para obtermos uma série estacionária, e consequentemente uma melhor performance do modelo.

Observando os resultados das métricas apresentadas, tivemos a pior performance para a série de acidentes, onde a predição foi quase linear diante a base de testes.

No caso da base de infrações tivemos um resultado razoável, porem longe de ser eficaz.

Arima – Acidentes

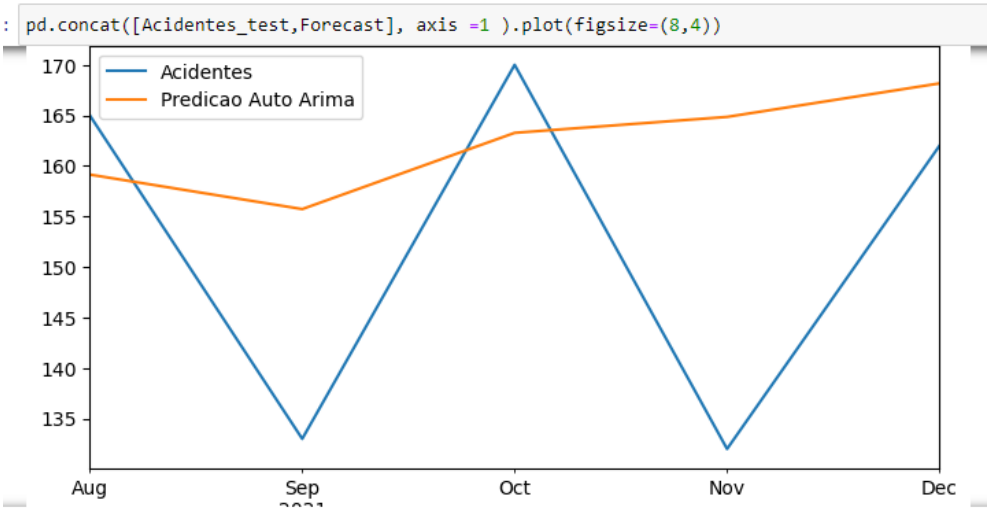
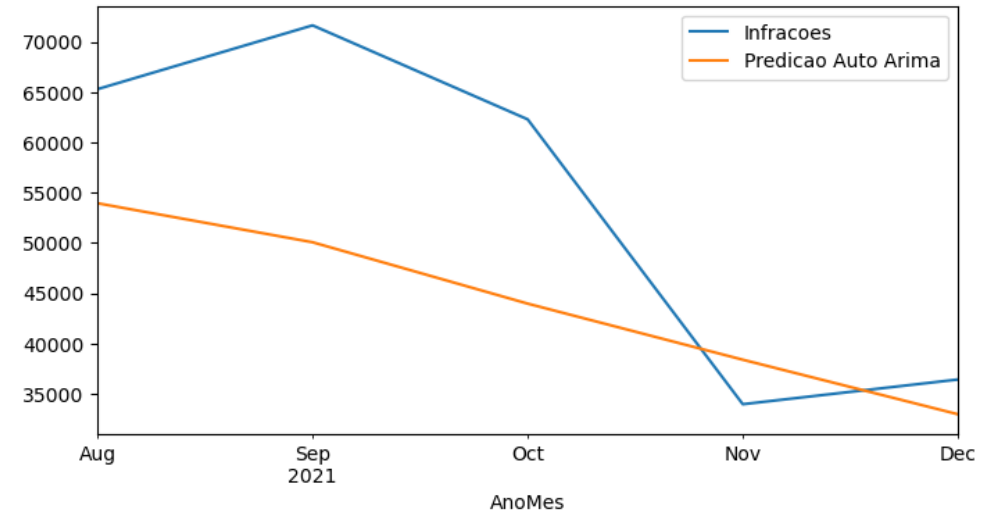


Figura 122: Interpretação de Resultados/ Modelos.

Arima - Infrações

```
pd.concat([Infracoes_test,Forecast], axis =1 ).plot(figsize=(8,4));
```



Através das métricas da RMSE (Raiz do erro médio quadrático), que mede a diferença média entre os valores previstos por um modelo de previsão e os valores reais, e a redução do MAE (Erro Médio Absoluto) que mede a média da diferença absoluta entre os valores previstos pelo modelo e os valores observados, conseguimos obter um desempenho geral dos modelos e definir os que tiveram a melhor aderência com as séries:

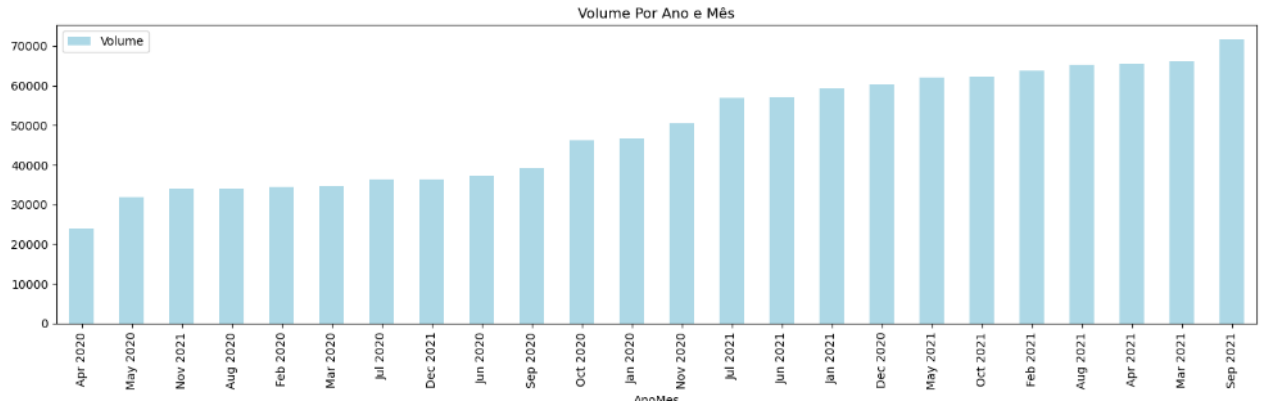
Como visto na tabela de Desempenho geral, o modelo com melhor performance para a série de infrações foi o SARIMAX, e para Acidentes foi o modelo Holt-Winters.

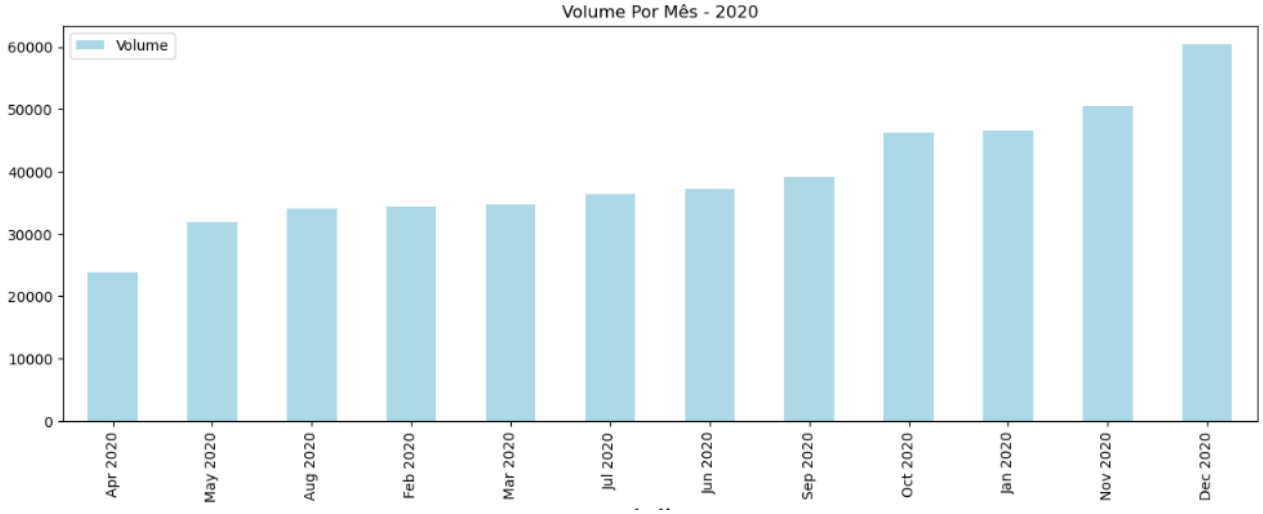
7. Apresentação dos Resultados

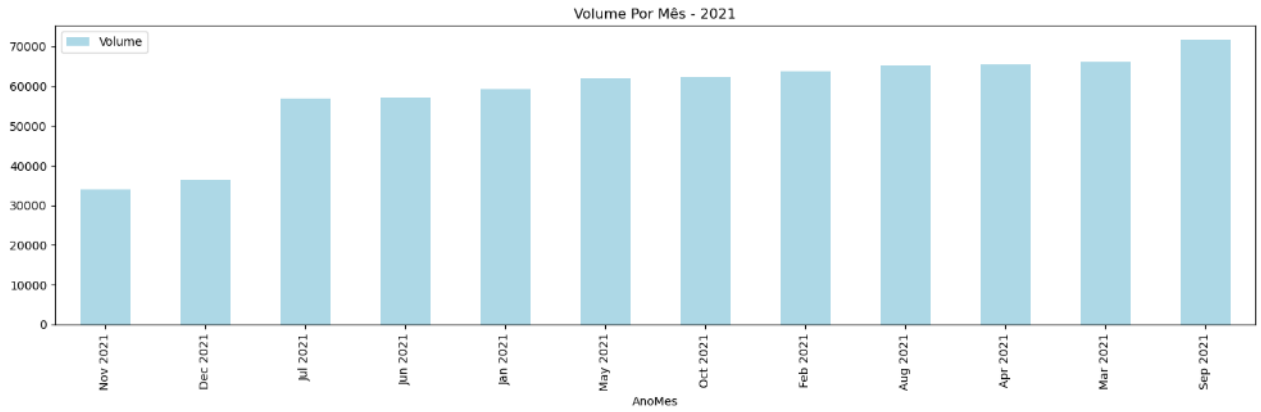
Conforme apresentado no módulo de Análise exploratória, para entendermos os comportamentos das ocorrências de infrações e acidentes de trânsito, foram elaboradas perguntas de negócio, e com a ajuda de Dashboards plotados no Jupyter Notebook, extraímos os principais insights:

Dashboards:

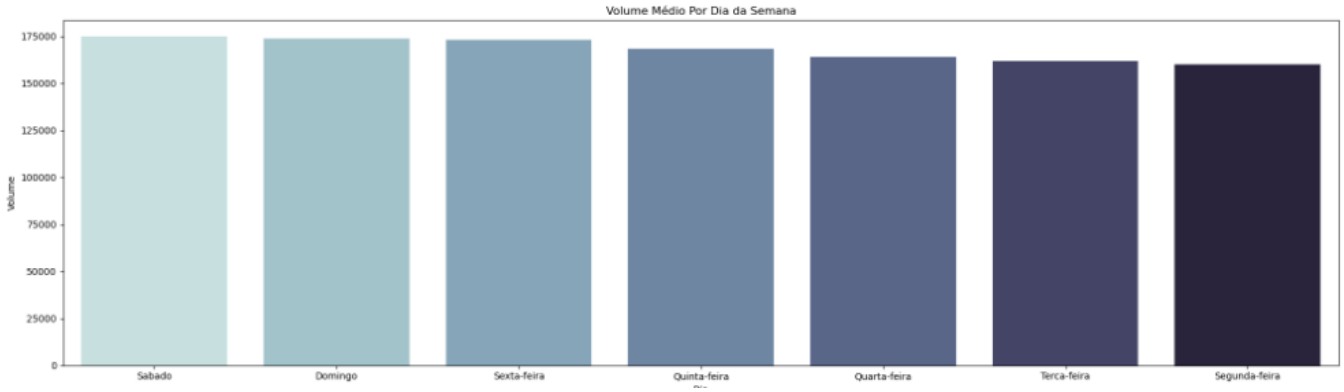
Em quais períodos do ano tivemos o maior índice de infrações?





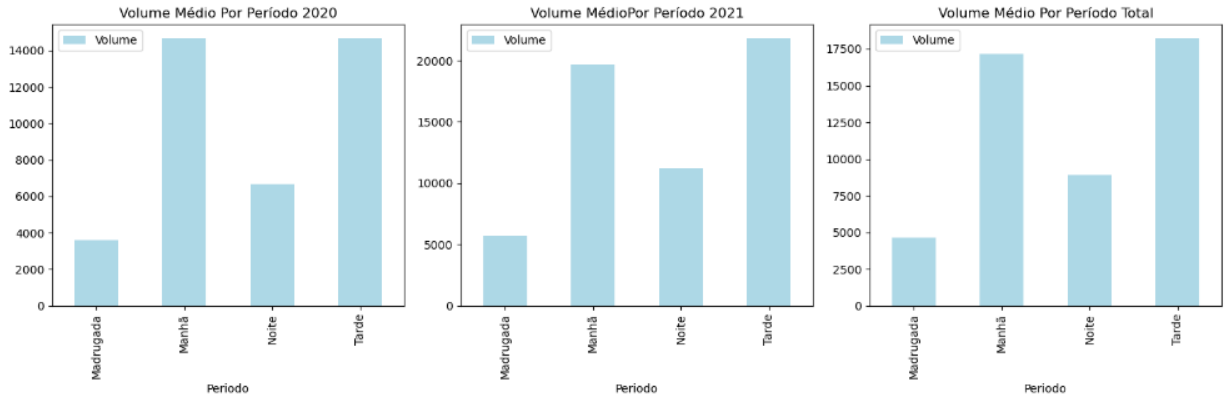


Em quais dias da semana temos aumento de infrações?

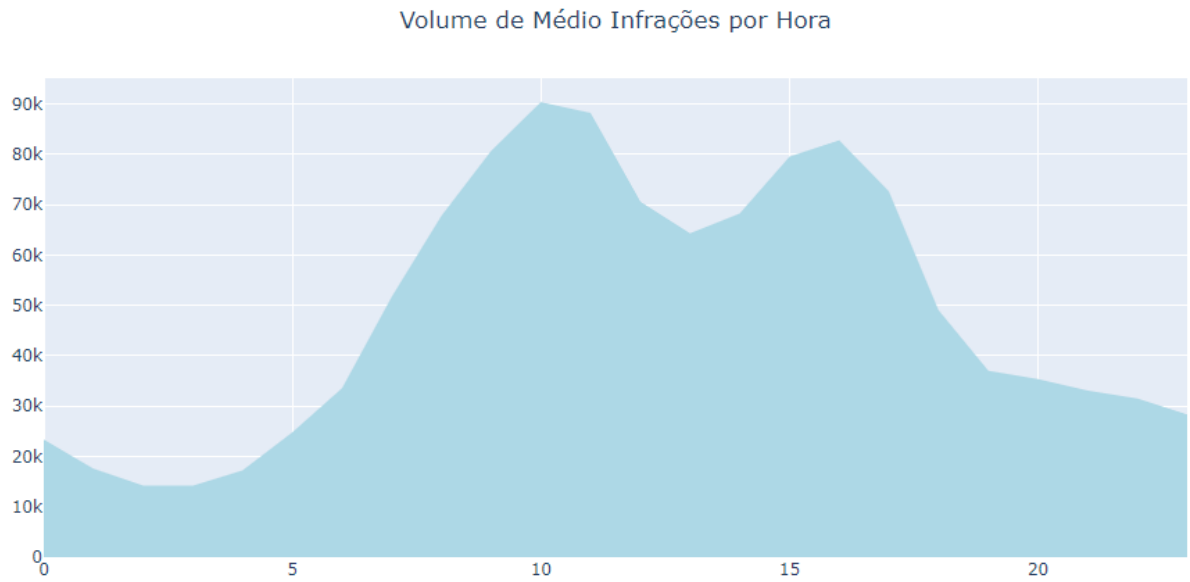


Em quais os períodos do dia temos mais ocorrências?

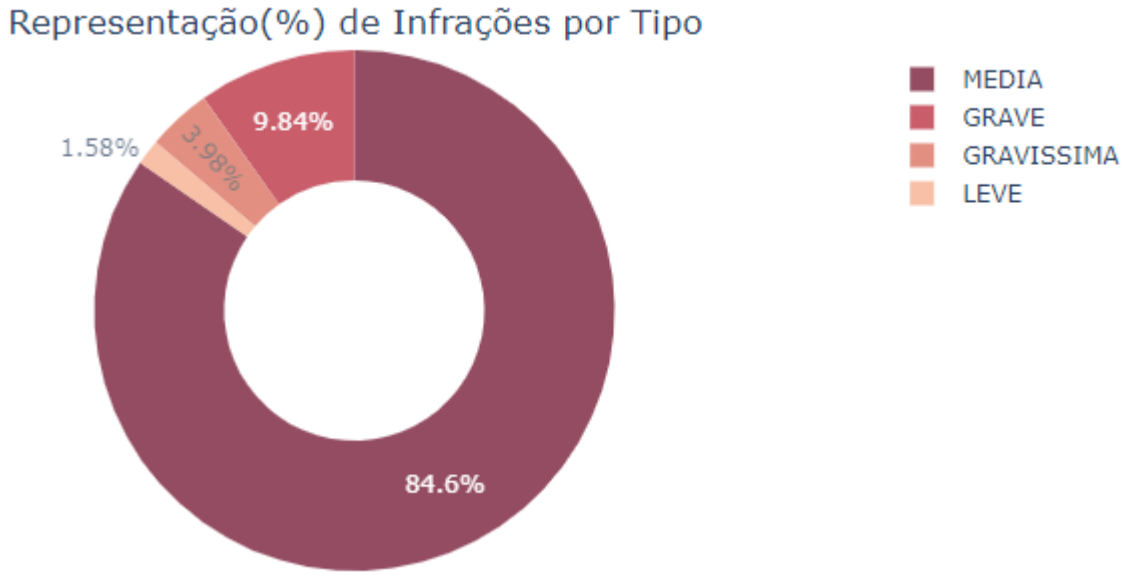
Periodo	Volume
Madrugada	5665.916667
Manhã	19670.666667
Noite	11197.750000
Tarde	21847.916667



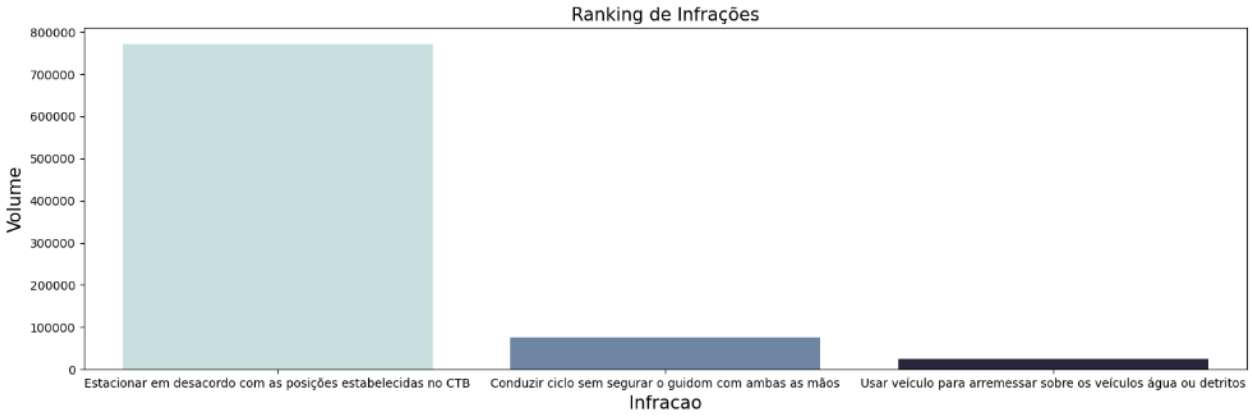
Em quais horários do dia temos picos de ocorrências de infrações?



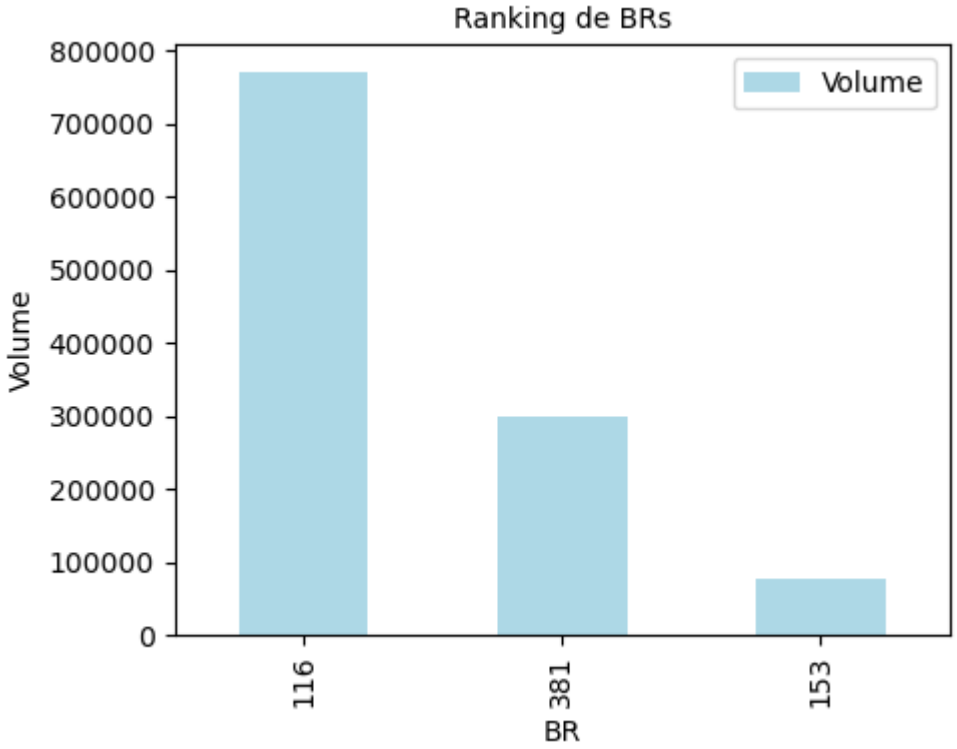
Quais são as severidades das infrações registradas?



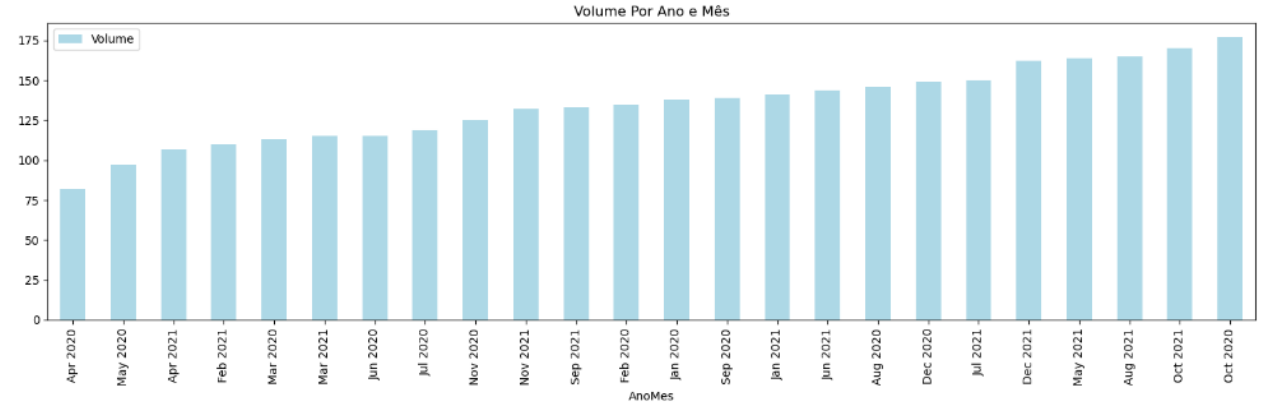
Quais são as infrações com maiores ocorrências?

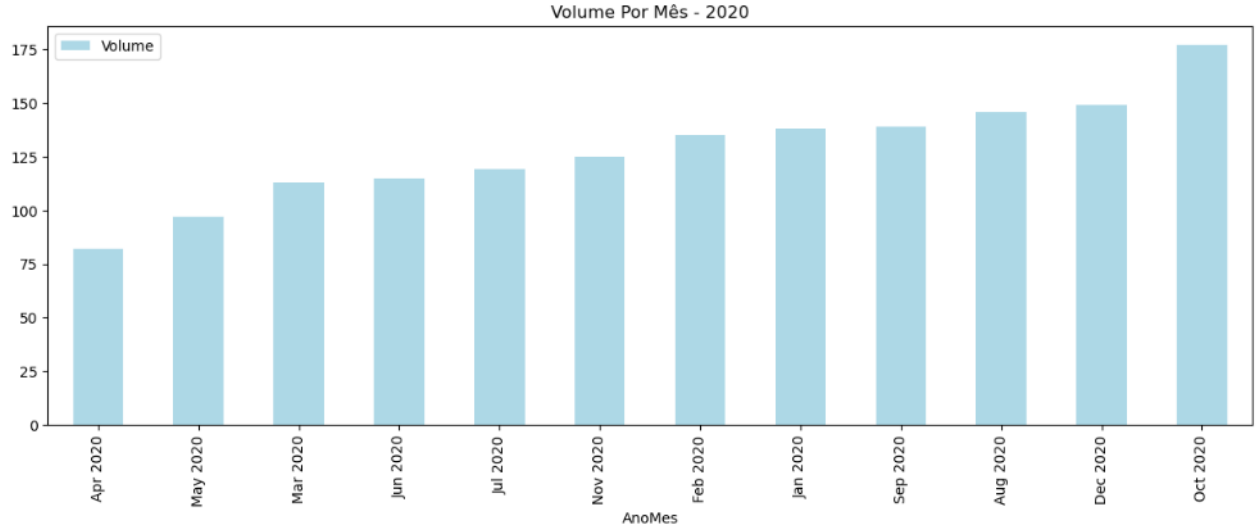


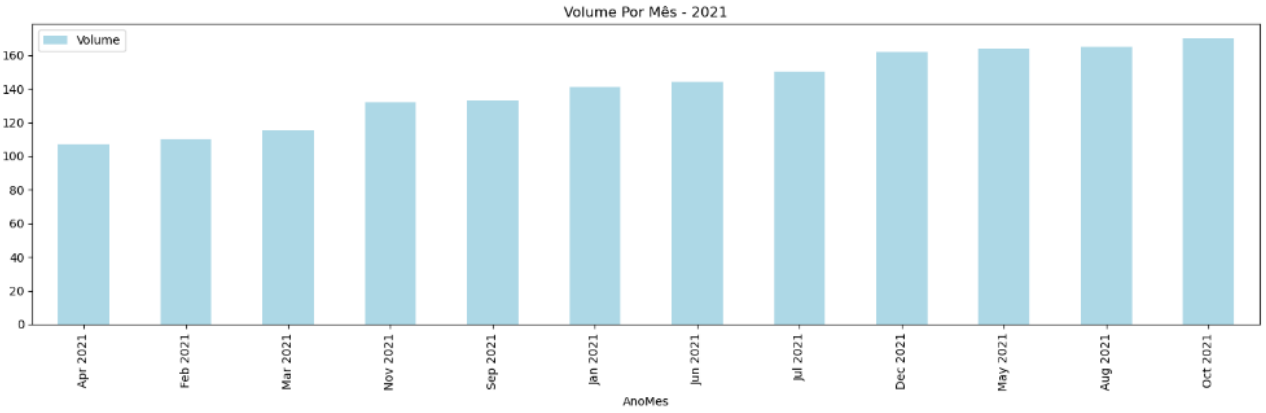
Quais são as rodovias de São Paulo com maiores ocorrências de infrações?



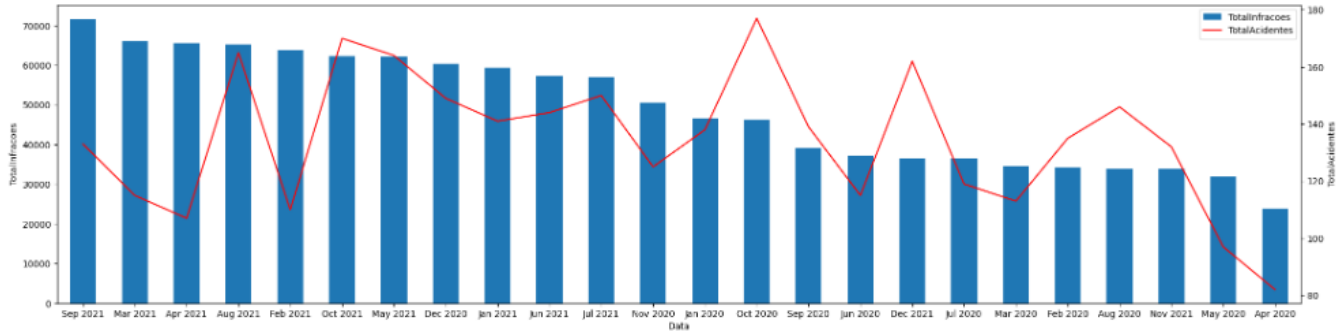
Em quais períodos do ano tivemos o maior índice de acidentes?



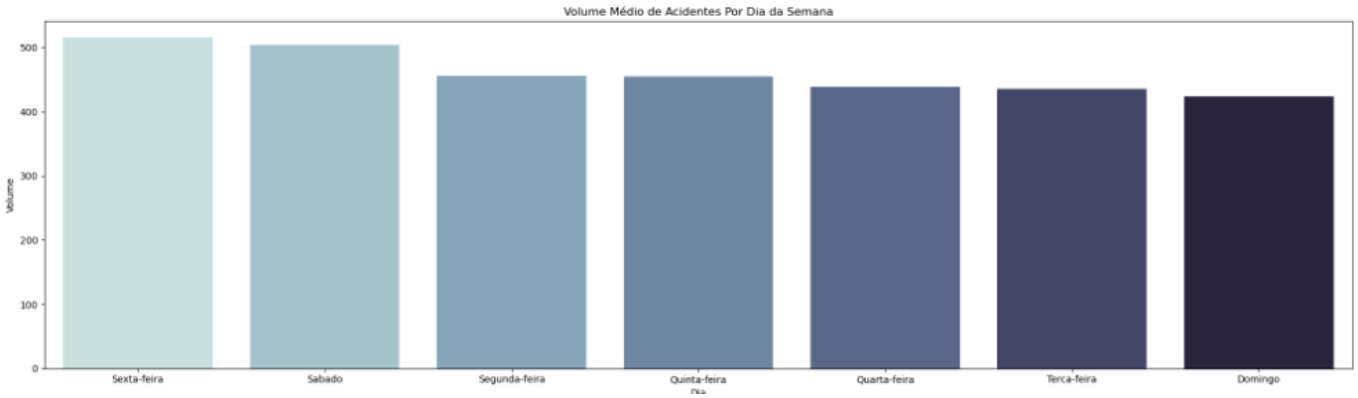




Qual o comportamento dos registros de infrações e acidentes durante o período?

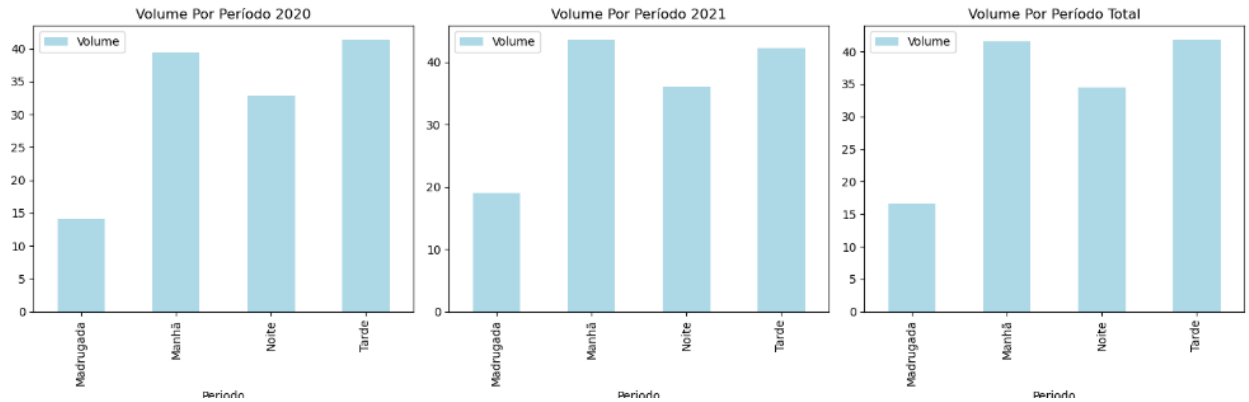


Em quais dias da semana temos aumento de acidentes?

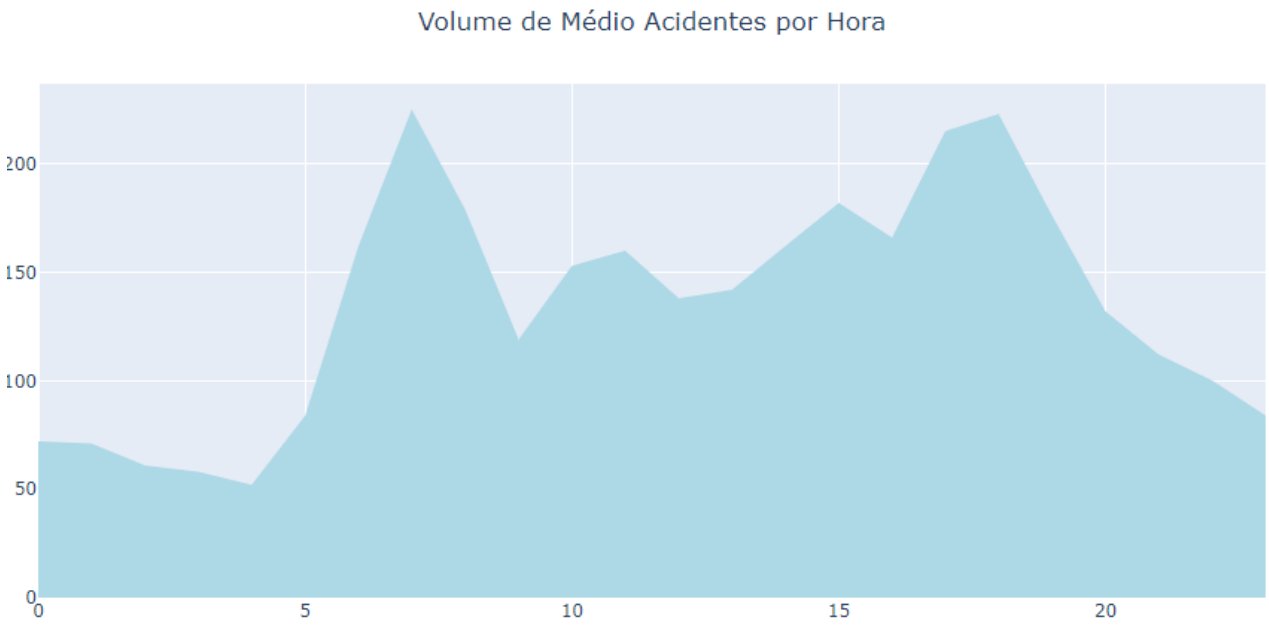


Em quais os períodos do dia temos mais ocorrências?

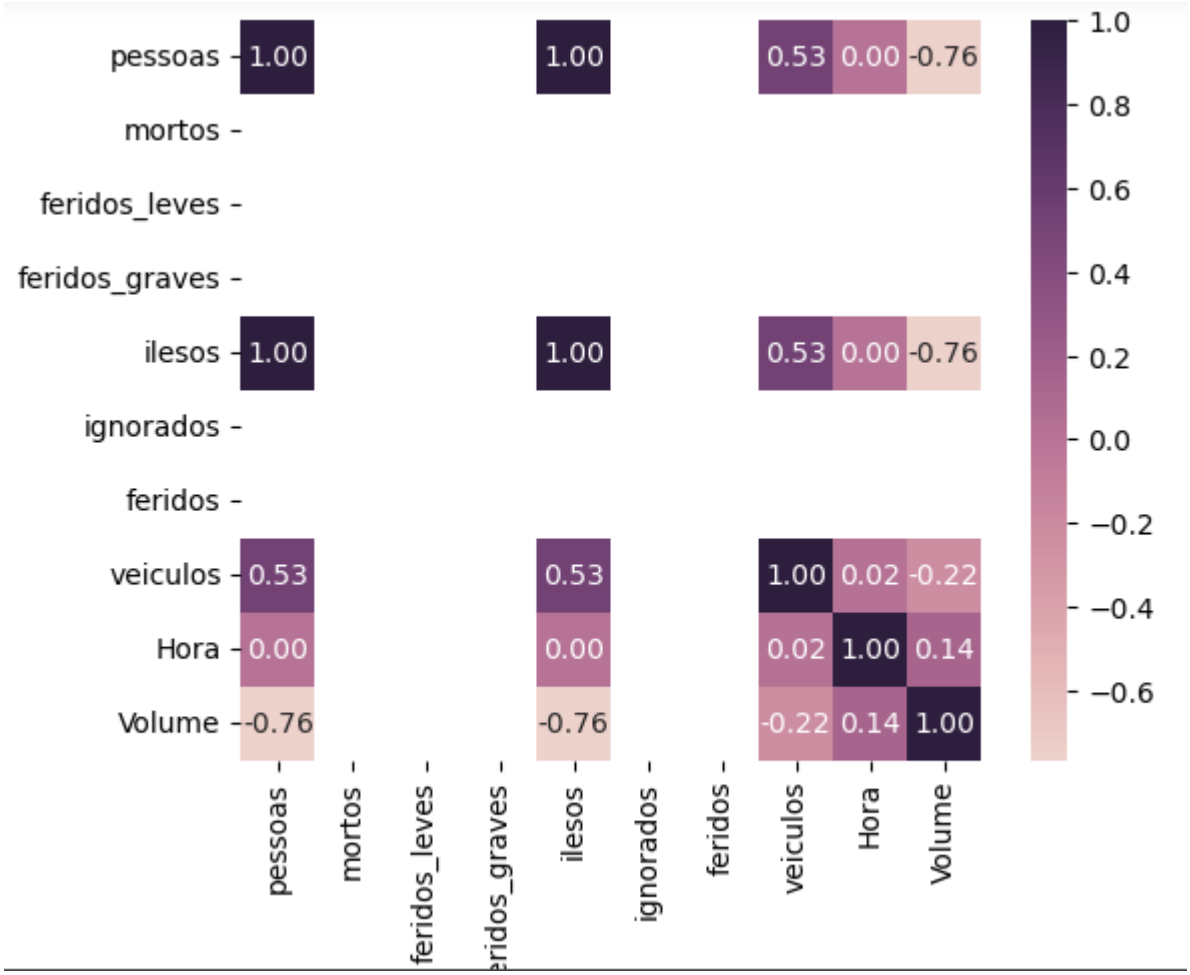
Periodo	Volume
Madrugada	19.000000
Manhã	43.666667
Noite	36.083333
Tarde	42.333333



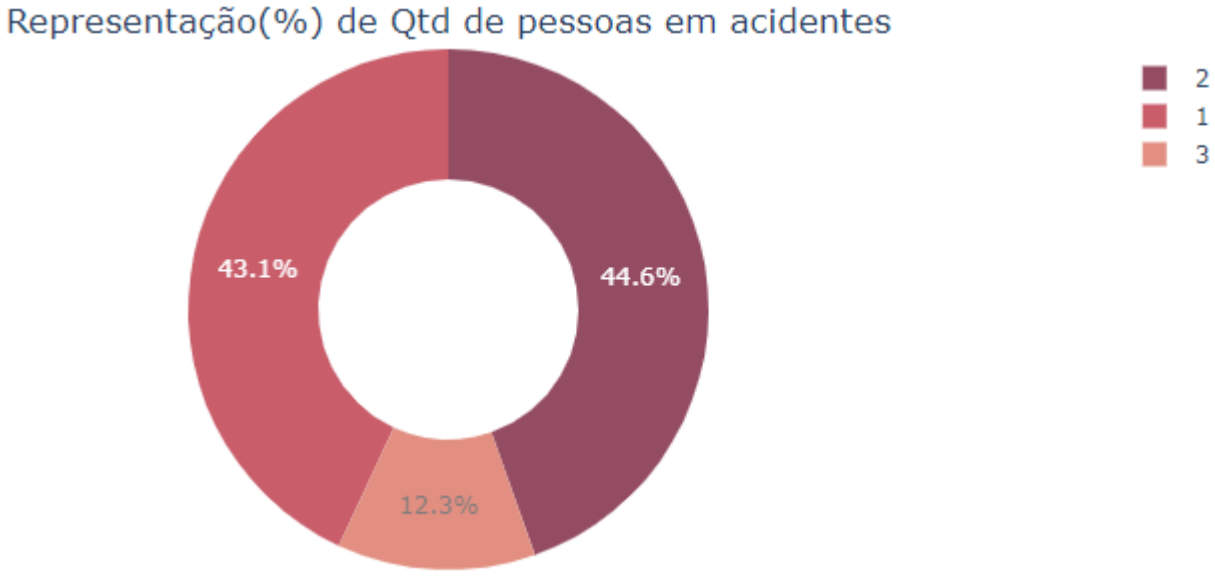
Em quais horários do dia temos picos de ocorrências de acidentes?



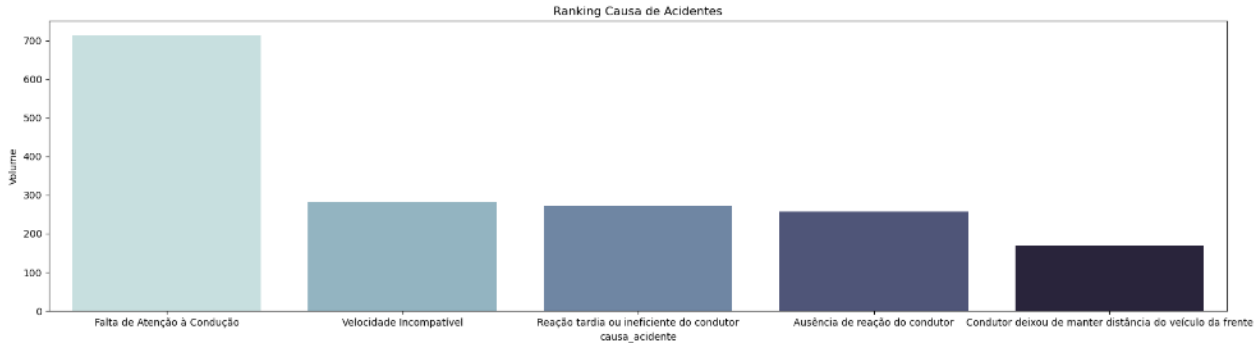
Qual a Correlação existente entre as variáveis?



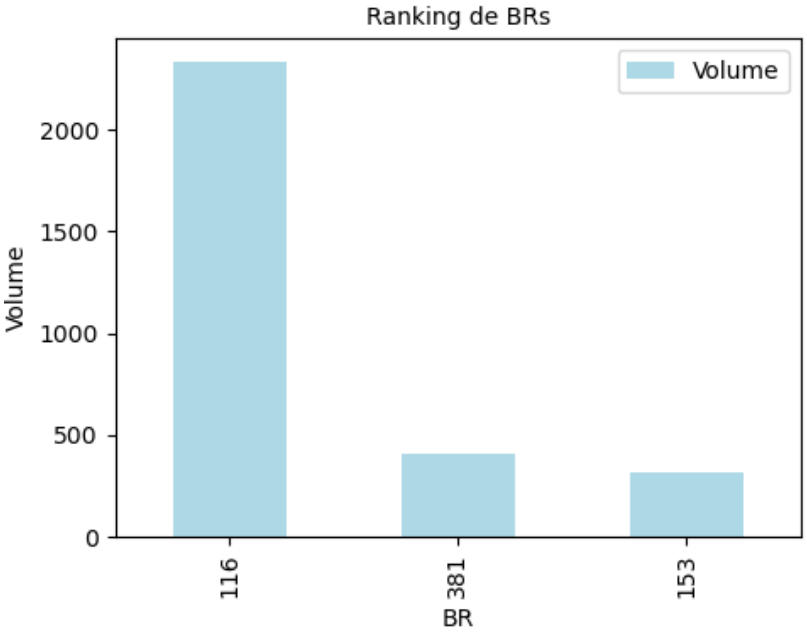
Qual a representação de pessoas envolvidas nos acidentes?



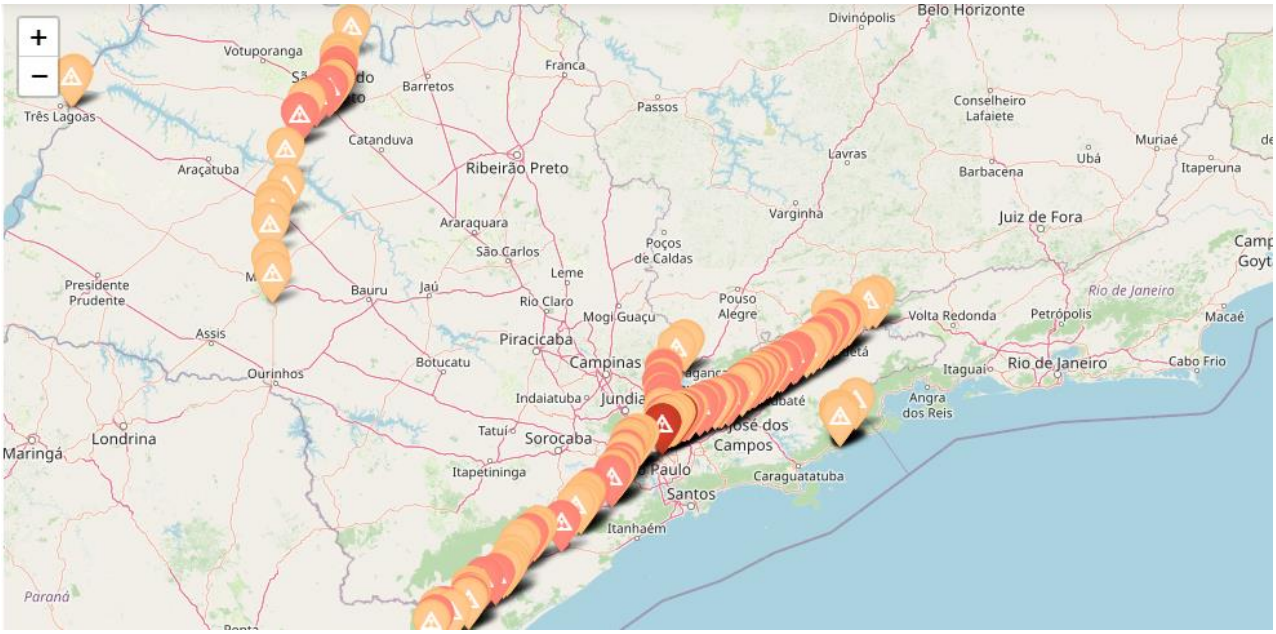
Quais são as causas de acidentes com maiores ocorrências?



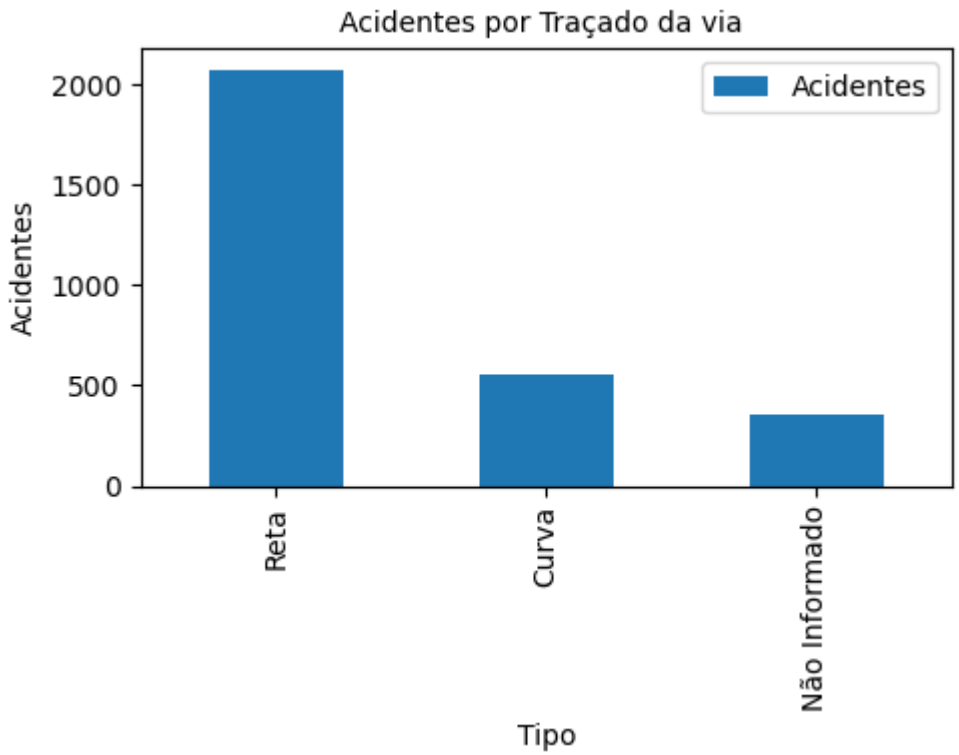
Quais são as rodovias de São Paulo com maiores ocorrências de infrações?



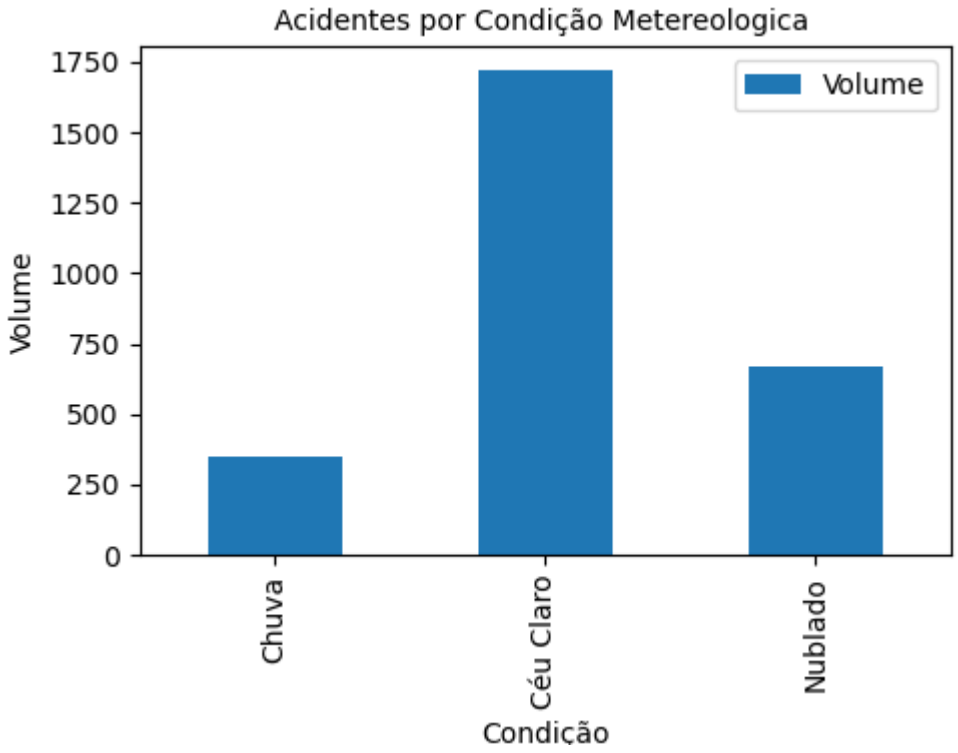
Mapa



Existe um tipo de via na qual há mais ocorrência de acidentes?



Qual o comportamento das ocorrências de acidentes sobre as condições meteorológicas?



- Vimos que o no segundo semestre dos anos de **2020** e **2021** são os períodos em que há mais ocorrências de infrações de trânsito nas rodovias de SP, sendo os em dezembro e setembro os meses com maior quantidade de infrações. E em outubro o Mês com maior quantidade de acidentes.
- As ocorrências de infrações e acidentes aumentam na chegada do final de semana, tendo as maiores médias de ocorrências na Sexta-feira e Sábado.
- Os horários do dia de maior ocorrência de acidentes e infrações é o período da Tarde (12h às 17h59).
- As severidades das infrações durante o período em São Paulo são de Média gravidade.
- Em relação aos acidentes, 45% dos acidentes envolvem 2 pessoas, 43% envolvem 1 pessoa e 12% envolvem 3 pessoas.
- Há mais ocorrências de infrações e acidentes na BR116, BR381 e BR153.
- Há uma forte correlação negativa entre as variáveis Volume e Pessoas, e nos descreve que quanto maior o volume de acidentes, menor o volume de pessoas ilesas.

- A Falta de atenção à condução é destacada como a maior causa de ocorrências de acidentes, seguida Velocidade incompatível.
- Entre todas as infrações cometidas nas rodovias de São Paulo, as mais cometidas são respectivas: 1- Estacionar em desacordo com as posições estabelecidas no CTB 2- Conduzir ciclo sem segurar o guidom com ambas as mãos 3- Usar veículo para arremessar sobre os veículos: água ou detritos
- O tipo de via em que mais ocorrem acidentes em SP são em linha reta, seguido das curvas.

<table>
<tr>
<td>PREDICTION TASK
Type of task? Entity on which predictions are made? Possible outcomes? Wait time before observation?

A tarefa de machine learning consiste na predição das séries temporais de infrações e acidentes que ocorreram nas rodovias federais do estado de São Paulo, entre 2020 e 2021. Foram utilizados os modelos SARIMAX, Holt Winters e ARIMA. Analisando o MAE e RMSE o modelo que obteve a menor performance foi o ARIMA, já o que performou melhor para o forecast de Acidentes foi o Holt Winters, já para Infrações foi o SARIMAX;</td>
<td>DECISIONS
How are predictions turned into proposed value for the end-user? Mention parameters of the process / application that does that.

Os modelos possibilitam termos resultados de predições de desempenho das ocorrências de infrações e acidentes de rodovias federais de São Paulo, possibilitando descobrir as principais áreas de melhorias no trânsito, aumentando a assertividade das equipes de fiscalização em momentos de maior necessidade.</td>
<td>VALUE PROPOSITION
Who is the end-user? What are their objectives? How will they benefit from the ML system? Mention workflow/interfaces.

O objetivo do trabalho é saber o desempenho das ocorrências nas rodovias, períodos do ano, horário do dia e gravidade dos eventos. Conhecendo este comportamento, podem ser implementadas medidas preventivas, tornando as inspeções mais eficazes e ajudando a reduzir os riscos na estrada.</td>
<td>DATA COLLECTION
Strategy for initial train set & continuous update. Mention collection rate, holdout on production entities, cost/constraints to observe outcomes.

As fontes de dados de acidentes e infrações foram os arquivos disponibilizados pelo Governo Federal, (Polícia Rodoviária federal). A fonte de tipos de infrações foi o site oficial do Portal do DETRAN.</td>
<td>DATA SOURCES
Where can we get (raw) information on entities and observed outcomes? Mention database tables, API methods, websites to scrape, etc.

As fontes de dados de acidentes e infrações foram os arquivos disponibilizados pelo Governo Federal, (Polícia Rodoviária federal). A fonte de tipos de infrações foi o site oficial do Portal do DETRAN.</td>
</tr>
<tr>
<td>IMPACT SIMULATION
Can models be deployed? Which test data to assess performance? Cost/gain values for (in)correct decisions? Fairness constraint?
Para a aplicação dos modelos foram necessários: Excluir as linhas vazias, verificar duplicidades, verificar descrições estatísticas.As métricas foram analisar a autocorrelação e autocorrelação parcial, e amplitude interquartil para remover outliers.</td>
<td>MAKING PREDICTIONS
When do we make real-time / batch pred.? Time available for this + featurization + post-processing? Compute target?
Após os tratamentos necessários e raspagem de dados as predições foram realizadas de forma rápida.</td>
<td rowspan="2"></td>
<td>BUILDING MODELS
How many prod models are needed? When would we update? Time available for this (including featurization and analysis)?
Foram utilizados os modelos SARIMAX, Holt Winters e ARIMA. Os modelos podem ser atualizados após a inclusão de 12 meses de dados.</td>
<td>FEATURES
Input representations available at prediction time, extracted from raw data sources.

A predição utilizada foi de séries temporais, separamos de uma grande base detalhada de Acidentes e Infrações, a variável X é a linha do tempo (AnoMes) e a variável Y a quantidade somada de ocorrências de acidentes e infrações agrupadas por AnoMes.</td>
</tr>
<tr>
<td></td>
<td>MONITORING
Metrics to quantify value creation and measure the ML system's impact in production (on end-users and business)?
O monitoramento dos modelos deve ser iniciado a partir das inclusões de novos meses de dados de infrações e acidentes.</td>
<td></td>
<td></td>
</tr>
</table>

8. Links

Link para o vídeo:

https://www.youtube.com/watch?v=kVkltopwxmk

Link para o repositório GitHub:

https://github.com/Vitor-Santos-Cacula/TCC-CienciaDeDados

REFERÊNCIAS

GOVERNO FEDERAL. Polícia Rodoviária Federal - Acidentes, 2020. Disponível em:

https://www.gov.br/prf/pt-br/acesso-a-informacao/dados-abertos/dados-abertos-acidentes

GOVERNO FEDERAL. Polícia Rodoviária Federal - Infrações, 2020. Disponível em:

https://www.gov.br/prf/pt-br/acesso-a-informacao/dados-abertos/dados-abertos-infracoes

DETRANSP. Portal Detran – Consulta Tabela de Infrações

https://www.detran.sp.gov.br/wps/portal/portaldetran/cidadao/infracoes/servicos/consultaTabelaInfracoes

OTEXTS – Arima modelling

https://otexts.com/fpp2/arima-r.html

ILOS – Modelo SARIMAX

https://ilos.com.br/introducao-ao-uso-de-analytics-em-previsao-de-vendas-com-aplicacao-de-um-modelo-sarimax-em-uma-empresa-de-bens-de-consumo/

MARIOFILHO -MAE

https://mariofilho.com/mae-erro-medio-absoluto-em-machine-learning/

SAP - Raiz do erro médio quadrático

https://help.sap.com/docs/SAP_ANALYTICS_CLOUD/00f68c2e08b941f081002fd3691d86a7/12cda9ce48b049a8adda7a6f3c240fa4.html

Benjamin, Etienne. MEDIUM – Python Séries Temporais

https://towardsdatascience.com/time-series-in-python-part-2-dealing-with-seasonal-data-397a65b74051

VASCONCELOS, Paulo. Technoblog Hotmart. Dicas para criar um modelo de previsão de séries temporais. Disponível em: < https://medium.com/techbloghotmart/dicas-para-criar-um modelo-de-previs%C3%A3o-de-s%C3%A9ries-temporais-d4bb2e32e148>.