

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

João Ricardo Côre Dutra

**ANÁLISE DA FREQUÊNCIA DE CRUZAMENTO DE MOTOCICLETAS QUANTO
ÀS CONDIÇÕES CLIMÁTICAS AOS TIPOS DE DIA E HORÁRIOS NA AVENIDA
AFONSO PENA ESQUINA COM A RUA MARANHÃO NO MUNICÍPIO DE BELO
HORIZONTE**

Belo Horizonte
2023

João Ricardo Côre Dutra

**ANÁLISE DA FREQUÊNCIA DE CRUZAMENTO DE MOTOCICLETAS QUANTO
ÀS CONDIÇÕES CLIMÁTICAS AOS TIPOS DE DIA E HORÁRIOS NA AVENIDA
AFONSO PENA ESQUINA COM A RUA MARANHÃO NO MUNICÍPIO DE BELO
HORIZONTE**

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

Belo Horizonte

2023

SUMÁRIO

1. Introdução	4
1.1. Contextualização	4
1.2. O problema proposto	5
1.3. Objetivos	6
2. Coleta de Dados.....	7
2.1. Base de Contagem Volumétrica de Radares.....	7
2.2. Base Tipos de Dia e Horário.....	12
2.3. Base de Dados Climáticos.....	14
3. Processamento e Tratamento de Dados	17
4. Análise e Exploração dos Dados	28
5. Criação de Modelos de Machine Learning.....	40
6. Interpretação e Comunicação dos Resultados	54
7. Conclusão.....	57
8. Links	59
APÊNDICE	60

1. Introdução

1.1. Contextualização

Viver em sociedade é uma característica de muitos animais, assim como os demais primatas organizam seus grupos a fim de potencializar a sua capacidade de sobrevivência e melhorar sua qualidade de vida, o ser humano desde muito tempo vive em grupos de diversos tipos e tamanhos com o mesmo objetivo.

O homem moderno já surgiu vivendo em sociedade, herança dos seus ancestrais. Viver em células organizadas onde cada indivíduo desempenha um papel específico para contribuir de forma benéfica para os outros membros do grupo faz o ser humano potencializar a sua capacidade de dominar a natureza em benefício próprio.

As cidades são um dos exemplos de células sociais mais complexas em que o homem se organiza e vive. Dentro delas os cidadãos se organizam de forma a contribuir de diferentes maneiras em benefício próprio e para o coletivo, melhorando assim a qualidade de vida de todos. Entretanto, essa forma de se organizar pode gerar alguns malefícios para a vida humana.

Durante o século XX, pode-se observar um alto crescimento das cidades. Dois dos fatores que mais contribuíram para esse crescimento foi a queda de mortalidade proporcionada pelas inovações tecnológicas e também o êxodo rural principalmente nos países em desenvolvimento. Esse alto crescimento fez aparecer diversas metrópoles pelo mundo e no Brasil não foi diferente.

Durante a história do mundo pode-se observar diversas mazelas oriundas da formação e crescimento das cidades e formação de metrópoles. A pandemia da peste bubônica por exemplo se fundamenta na falta de higiene das cidades do século XIV. Alguns vírus que vieram acometer os humanos oriundos de grandes criações de aves e suínos também é outro exemplo, já que grandes criações animais são necessárias para alimentar muitas pessoas que não necessariamente trabalham na produção de alimentos.

Apesar de grandes metrópoles terem um papel fundamental no desenvolvimento de um país, viver em cidades muito grandes sem um planejamento urbano adequado pode causar diversos problemas, como o aumento da poluição do ar, problemas quanto ao trânsito

de pessoas e cargas por haver altos fluxos de veículos, fazendo gastar mais tempo que o necessário para se deslocar-se de um ponto a outro, também o aumento da violência por conta dos bolsões de pobreza oriundos de uma má gestão social e muitos outros.

1.2. O problema proposto

Muitas das vezes, o crescimento de uma cidade traz problemas para a sociedade urbana por falta de planejamento. Estudar e entender a intensidade de fluxo de veículos em uma cidade pode trazer diversos benefícios para o município que o faz, possibilitando uma melhor estruturação das vias públicas e do trânsito, gerando economia de tempo, de recursos e cuidado com o meio ambiente, portanto melhorando a vida dos cidadãos que ali residem.

Intuitivamente, pressupõe-se que os fluxos de alguns tipos de veículos podem variar em função de alguns fatores, como por exemplo o fluxo de caminhões pode ser maior em dias úteis quando comparados à finais de semana e feriados, assim como o fluxo de motocicletas também pode variar de acordo com o dia, horário e possivelmente condições climáticas. Prever e entender o comportamento do fluxo de veículos em um determinado ponto de uma cidade se mostra de extrema importância para melhor gestão de recursos, mitigação de acidentes, melhoria da qualidade de vida de uma população, tomadas de decisões quanto ao planejamento urbano, cuidado com o meio ambiente e muitos outros motivos.

Atualmente, a prefeitura da cidade de Belo Horizonte, disponibiliza através do portal BHTrans (<https://prefeitura.pbh.gov.br/bhtrans> acessado em 22 de novembro de 2022) diversos dados relacionados ao trânsito na cidade, dentre esses dados existe a contagem volumétrica de radares (<https://dados.pbh.gov.br/dataset/contagens-volumetricas-de-radares> acessado em 22 de novembro de 2022) onde estão presentes os registros de cada veículo que passou em frente aos radares de trânsito espalhados pela cidade. Nessa massa de dados estão presentes informações de data, horário, tipo de veículo e localização.

O fluxo de motocicletas em uma cidade impacta diretamente na qualidade e na segurança do trânsito como um todo e também esse fluxo pode ser bastante volátil em função de muitos fatores externos, logo, optou-se por medir o fluxo de motocicletas na Av. Afonso Pena, equina com a Rua Maranhão nos meses de maio e junho de 2022 a fim de conseguir entender como alguns fatores externos o influenciam. A Av. Afonso pena foi escolhida por ser

conhecidamente uma avenida bastante movimentada na cidade de Belo Horizonte, portanto, proporcionando uma quantidade significativa de dados a serem analisados. Os fatores escolhidos para serem estudados como influenciadores na intensidade de fluxo de motocicletas foram os tipos de dia (dia útil, final de semana e feriado), horários e condições climáticas (temperatura e precipitação).

1.3. Objetivos

Este trabalho tem como objetivo propor e avaliar a influência fatores externos que podem afetar a frequência de cruzamentos de motocicletas na Av. Afonso Pena, equina com a rua Maranhão em Belo Horizonte.

O estudo consistiu em calcular a frequência de cruzamento de motocicletas em cada hora de cada dia no intervalo do dia 01 de maio de 2022 até o dia 30 de junho de 2022 e depois classificar cada registro como acima da média ou abaixo da média da frequência com relação a todos os registros.

Os fatores escolhidos como possíveis influenciadores na frequência de cruzamentos de motocicletas no ponto estudado foram:

1. Tipo de dia: Onde pode ser sábado, domingo, feriado ou dia útil. Esse fator foi escolhido pois nos dias úteis as atividades econômicas e sociais ocorrem com maior intensidade, podendo influenciar na frequência da quantidade de motocicletas que cruzam por hora o ponto de estudo.
2. Hora: A hora em que a dada frequência é calculada é um possível fator de influência, pois existem horários ao longo das 24h de um dia em que as pessoas saem de casa para trabalhar ou para lazer. Também há os horários de funcionamento das atividades comerciais e econômicas, o que pode influenciar na maior quantidade de pessoas e transporte de cargas utilizando as vias públicas para se deslocarem.
3. Temperatura Ambiente: A temperatura pode ser um fator que influencia na decisão de uma pessoa utilizar ou não uma motocicleta, onde em dias muito quentes ou muito frios pode acontecer de optarem por outro tipo de transporte.
4. Precipitação: Assim como a temperatura, a precipitação também pode ser um fator que influencia na decisão da utilização de uma motocicleta para transporte ou outro tipo de veículo.

Os quatro fatores apresentados foram classificados em dois grupos: o primeiro é o de fatores climáticos, compreendendo dados de precipitação e temperatura ambiente, o segundo grupo com os dados de tipos de dias e horários, compreendendo o tipo de dia em que a medição da frequência está sendo feita e o respectivo horário.

Por fim, este trabalho procurou verificar o quanto e como esses grupos influenciou no aumento ou diminuição do fluxo de motocicletas no ponto estudado.

2. Coleta de Dados

Os dados utilizados neste trabalho tiveram origem em três fontes. A primeira foi a base de contagem volumétrica de radares fornecida pela prefeitura de Belo Horizonte através do portal da BHTrans. A segunda foi o tipo de dia, onde essa base foi gerada em código e a única informação externa agregada foi o dia do feriado de *Corpus Christi*. A terceira base, que são os dados climáticos, apesar de ter sido construída no próprio código, as informações de temperatura ambiente e precipitação vieram da *API open meteo* (<https://open-meteo.com/> acessado em 22 de novembro de 2022).

A base de contagem volumétrica de radares serviu para gerar a base de frequência de cruzamentos por hora de motocicletas na Av. Afonso Pena, equina com a rua Maranhão, que posteriormente serviu para gerar a base final com todos os dados.

As três bases que deram origem a base de dados final com todos os dados coletados e gerados, possuem registros orientados ao horário da observação, portanto foi criado uma coluna chamada “DATA_HORA_COMPARADOR”, que tem como objetivo servir de referência para a união das três bases de dados, também funcionando como uma chave primária, já que não existem mais de um registro para a mesma data e horário.

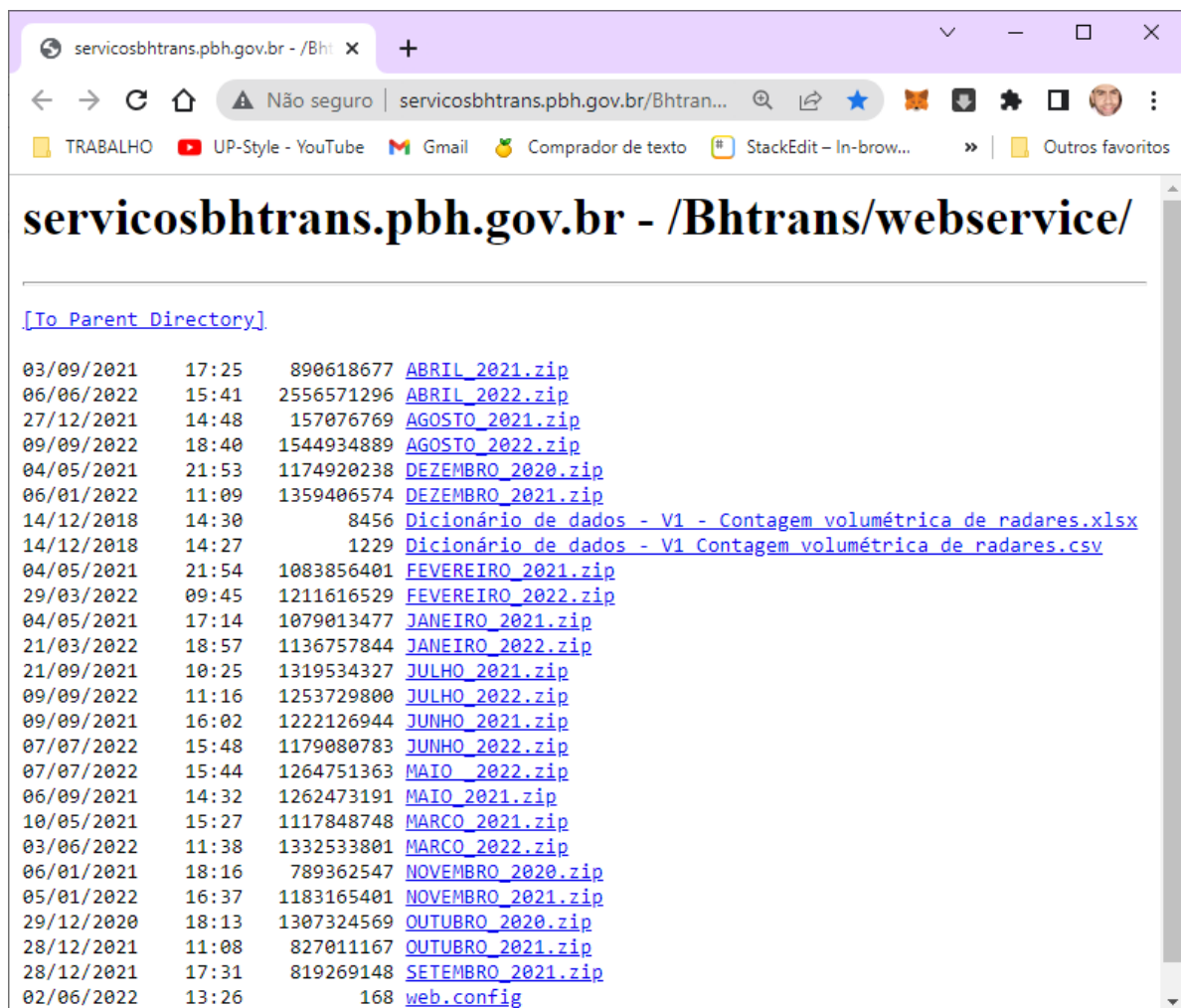
2.1. Base de Contagem Volumétrica de Radares

A base de contagem volumétrica de radares foi obtida na plataforma de dados do site da prefeitura de Belo Horizonte (<https://dados.pbh.gov.br/dataset/contagens-volumetricas-de-radares> acessado em 22 de novembro de 2022) onde os dados estão em arquivos de

extensão ZIP, contendo em cada arquivo vários arquivos em formato JSON, um para cada dia do respectivo mês.

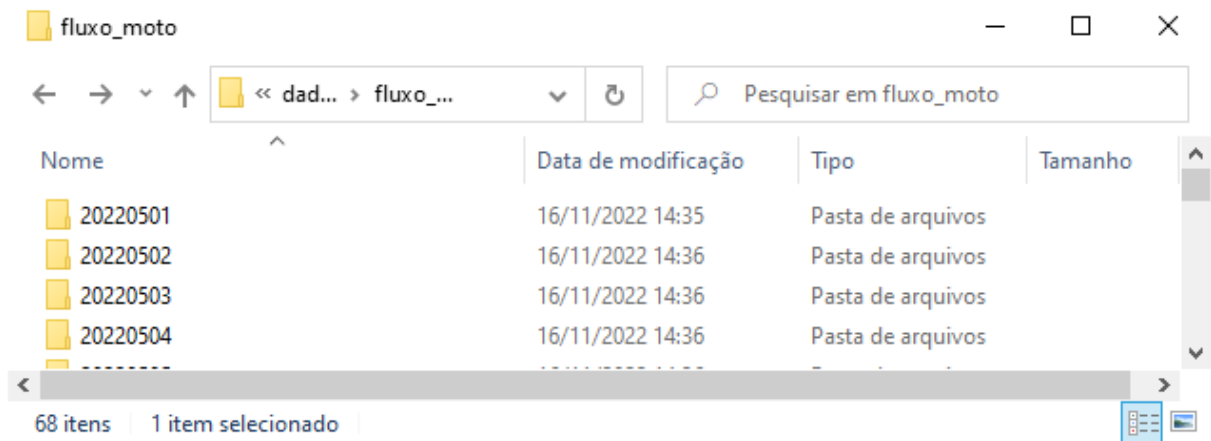
Para trabalhar na base da contagem volumétrica de radares na cidade de Belo Horizonte, foi necessário acessar o endereço onde estão disponíveis os dados e fazer o download dos meses escolhidos para o estudo proposto neste trabalho. Logo, foi encontrada a página exposta na Figura 1.

Figura 1 – Fonte de dados da base de contagem volumétrica de radares



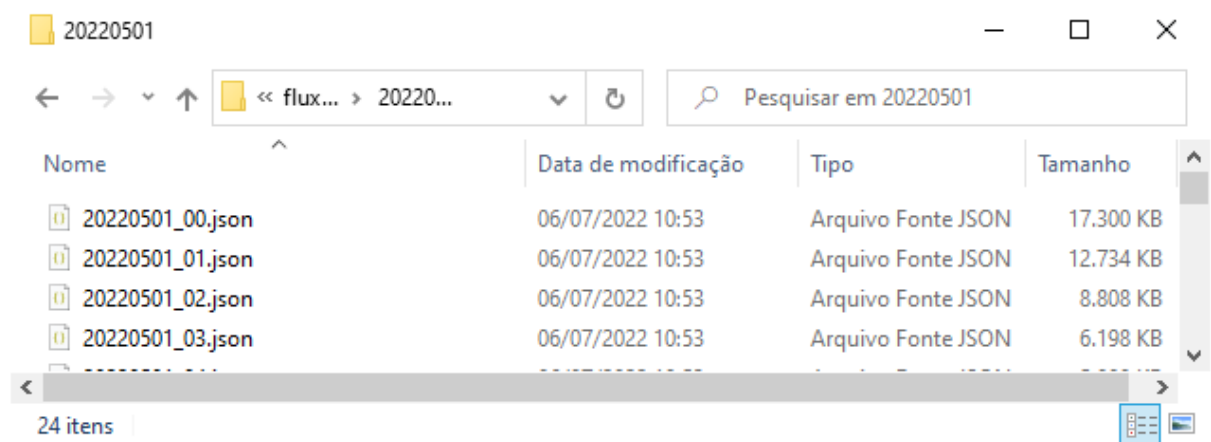
Como os meses selecionados foram maio e junho de 2022, os arquivos contendo os arquivos JSONs foram baixados e ao descompactar, percebeu-se que os dados estavam organizados por dia, como mostra a Figura 2.

Figura 2 – Pastas com os dados organizados por dia



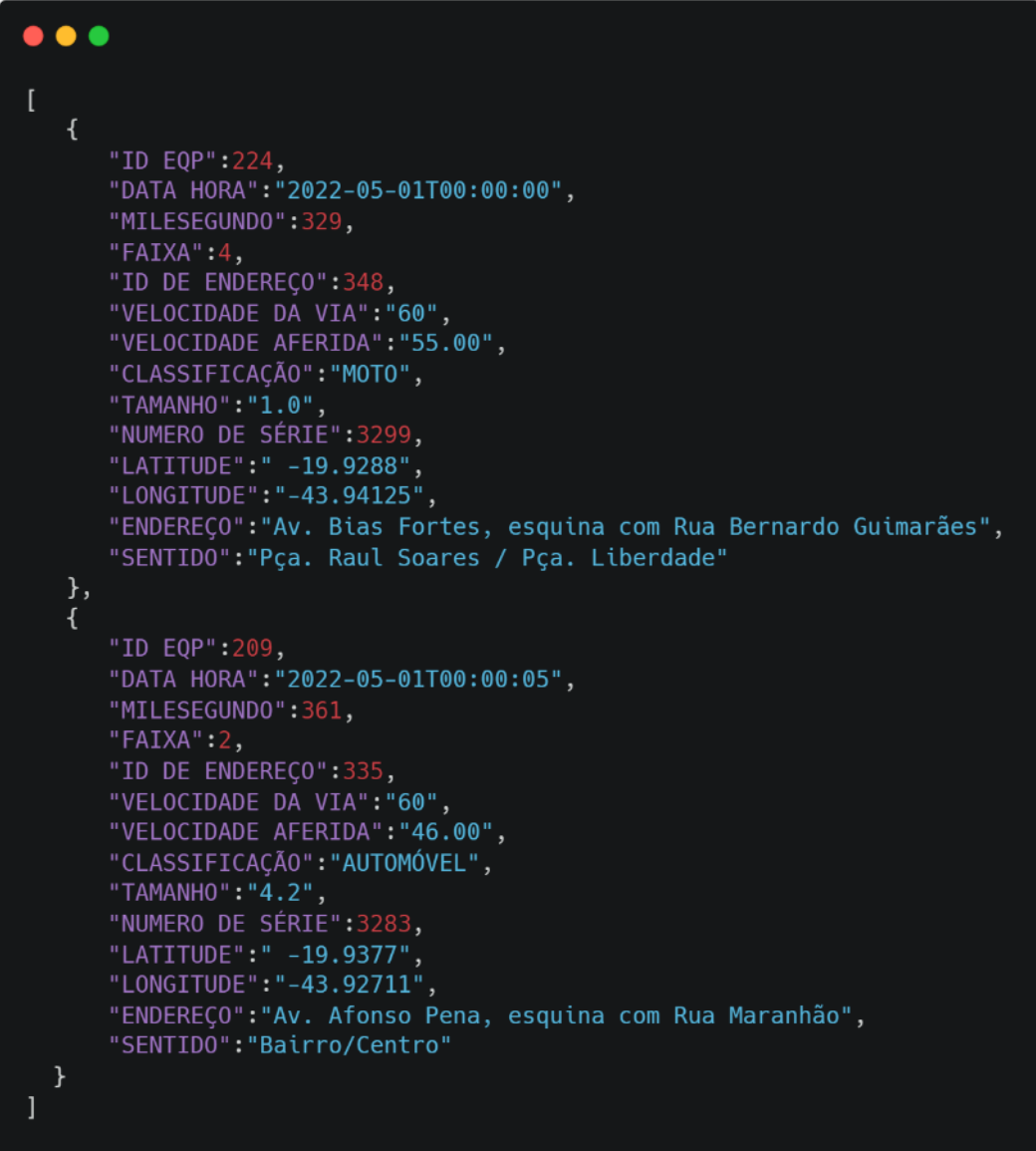
Ao acessar uma das pastas, relativa a um dia específico, observou-se um arquivo JSON para cada hora do respectivo dia, como mostra na Figura 3.

Figura 3 – Arquivos JSON contendo as informações de cada veículo que cruzou cada radar de transito dentro da cidade de Belo Horizonte



Por fim, para entender de que forma deve-se implementar o código em *Python* a fim de ler e converter esses dados para um *dataset* (*Pandas DataFrame*), logo foi aberto alguns dos arquivos e percebeu-se que cada arquivo JSON tem a estrutura de acordo com a apresentada na Figura 4.

Figura 4 – Estrutura do arquivo JSON



```
[
  {
    "ID EQP":224,
    "DATA HORA":"2022-05-01T00:00:00",
    "MILESEGUNDO":329,
    "FAIXA":4,
    "ID DE ENDEREÇO":348,
    "VELOCIDADE DA VIA":"60",
    "VELOCIDADE AFERIDA":"55.00",
    "CLASSIFICAÇÃO":"MOTO",
    "TAMANHO":"1.0",
    "NUMERO DE SÉRIE":3299,
    "LATITUDE":" -19.9288",
    "LONGITUDE":"-43.94125",
    "ENDEREÇO":"Av. Bias Fortes, esquina com Rua Bernardo Guimarães",
    "SENTIDO":"Pça. Raul Soares / Pça. Liberdade"
  },
  {
    "ID EQP":209,
    "DATA HORA":"2022-05-01T00:00:05",
    "MILESEGUNDO":361,
    "FAIXA":2,
    "ID DE ENDEREÇO":335,
    "VELOCIDADE DA VIA":"60",
    "VELOCIDADE AFERIDA":"46.00",
    "CLASSIFICAÇÃO":"AUTOMÓVEL",
    "TAMANHO":"4.2",
    "NUMERO DE SÉRIE":3283,
    "LATITUDE":" -19.9377",
    "LONGITUDE":"-43.92711",
    "ENDEREÇO":"Av. Afonso Pena, esquina com Rua Maranhão",
    "SENTIDO":"Bairro/Centro"
  }
]
```

Como pode ser visto na figura 4, o arquivo JSON é um grande *array* de objetos contendo os dados de cada registro de cruzamento de todos os tipos veículos e em todos os pontos de radares de transito.

Já sabendo como se organiza os dados de registros dos radares de transito, as pastas contendo o registro de cada dia referentes aos meses de maio e junho de 2022, foram colocadas em um único diretório e foi construído um código em *Python* para fazer uma leitura dinâmica de cada registro que posteriormente foram organizados em um *dataset*. O código em *Python* que leu todos os arquivos JSON no diretório citado e converteu em um *Padas DataFrame* está exposto na Figura 5.

Figura 5 – Código de leitura dos arquivos JSONs

```

mes_mai = '05'
dias_mai = ['01', '02', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31']
mes_junho = '06'
dias_junho = ['01', '02', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30']
horas_dia = ['00', '01', '02', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24']
df_final = pd.DataFrame(columns=['ID EQP', 'DATA_HORA', 'MILESEGUNDO', 'FAIXA', 'ID DE ENDEREÇO', 'VELOCIDADE DA VIA', 'VELOCIDADE AFERIDA'])
for dia in dias_mai:
    for hora in horas_dia:
        caminho_arquivo = f'dados/fluxo_moto/2022{mes_mai}/{dia}/2022{mes_mai}-{dia}-{hora}.json'
        with open(caminho_arquivo, 'r', encoding="utf8") as f:
            str_json = f.read()
            dicionario_dados = json.loads(str_json)
            df_lido = pd.json_normalize(dicionario_dados)
            df_lido = df_lido.loc[df_lido['CLASSIFICAÇÃO'] == 'MOTO']
            df_lido['LATITUDE'] = df_lido['LATITUDE'].str.replace(' ', '').astype(float)
            df_lido['LONGITUDE'] = df_lido['LONGITUDE'].str.replace(' ', '').astype(float)
            df_lido = df_lido.loc[df_lido['LATITUDE'] == -19.9377]
            df_lido = df_lido.loc[df_lido['LONGITUDE'] == -43.92711]
            df_final = pd.concat([df_final, df_lido])
for dia in dias_junho:
    for hora in horas_dia:
        caminho_arquivo = f'dados/fluxo_moto/2022{mes_junho}/{dia}/2022{mes_junho}-{dia}-{hora}.json'
        with open(caminho_arquivo, 'r', encoding="utf8") as f:
            str_json = f.read()
            dicionario_dados = json.loads(str_json)
            df_lido = pd.json_normalize(dicionario_dados)
            df_lido = df_lido.loc[df_lido['CLASSIFICAÇÃO'] == 'MOTO']
            df_lido['LATITUDE'] = df_lido['LATITUDE'].str.replace(' ', '').astype(float)
            df_lido['LONGITUDE'] = df_lido['LONGITUDE'].str.replace(' ', '').astype(float)
            df_lido = df_lido.loc[df_lido['LATITUDE'] == -19.9377]
            df_lido = df_lido.loc[df_lido['LONGITUDE'] == -43.92711]
            df_final = pd.concat([df_final, df_lido])
df_final.rename(columns={'ID EQP': 'ID_EQP', 'DATA_HORA': 'DATA_HORA', 'ID DE ENDEREÇO': 'ID_DE_ENDEREÇO', 'VELOCIDADE DA VIA': 'VELOCIDADE_DA_VIA', 'VELOCIDADE AFERIDA': 'VELOCIDADE_AFERIDA'})
df_final.to_csv("dados/fluxo_moto/base_fluxo_motos.csv", index=False)

```

O código da figura 5 começa criando *arrays* de dias e horas a para facilitar a construção do caminho de leitura de cada arquivo JSON, já que no total são 1464 arquivos para serem lidos. É feita uma leitura para cada mês com a finalidade de simplificar a complexidade do código, já que os meses de maio e junho possuem quantidade diferentes de dias.

Durante a leitura de cada mês, os dados são recuperados, filtrados por tipo de veículo e localização (selecionando apenas motocicletas no ponto de estudo de latitude -19.9377 e longitude -43.92711), que posteriormente são convertidos em um *dataset* local que depois é concatenado com um *dataset* de nome *df_final*, criado para acumular os dados lidos. Por fim o resultado, que pode ser visto na Figura 6, é salvo em um arquivo de extensão CSV, facilitando a posterior recuperação das informações.

Figura 6 – Dataset obtido da leitura dos arquivos JSONs da contagem volumétrica de radares

ID_EQP	DATA_HORA	MILESEGUNDO	FAIXA	ID_DE_ENDEREÇO	VELOCIDADE_DA_VIA	VELOCIDADE_AFERIDA	CLASSIFICACAO	TAMANHO	NUMERO_DE_SERIE	LATITUDE	LONGITUDE	ENDEREÇO	SENTIDO	
1958	209	2022-05-01T00:02:34	647	2	335	60	51.00	MOTO	1.0	3283	-19.9377	-43.92711	Av. Afonso Pena, esquina com Rua Maranhão	Bairro/Centro
3328	209	2022-05-01T00:04:04	631	3	335	60	30.00	MOTO	1.0	3283	-19.9377	-43.92711	Av. Afonso Pena, esquina com Rua Maranhão	Bairro/Centro
3566	209	2022-05-01T00:04:20	100	3	335	60	27.00	MOTO	1.0	3283	-19.9377	-43.92711	Av. Afonso Pena, esquina com Rua Maranhão	Bairro/Centro
4480	209	2022-05-01T00:05:24	272	3	335	60	37.00	MOTO	1.0	3283	-19.9377	-43.92711	Av. Afonso Pena, esquina com Rua Maranhão	Bairro/Centro

O significado de cada coluna do *dataset* da contagem volumétrica de radares da cidade de Belo Horizonte pode ser visto na Tabela 1.

Tabela 1 – Colunas do dataset da contagem volumétrica de radares

Nome da Coluna	Descrição	Tipo
ID_EQP	Chave de identificação única do equipamento (radar de trânsito)	Inteiro
DATA_HORA	Data e hora em que foi feito o registro do cruzamento de veículo pelo equipamento	Data e Hora
MILESEGUNDO	Milissegundo em que o registro foi feito	Inteiro
FAIXA	Faixa da avenida em que o registro foi feito	Inteiro
ID_DE_ENDERCO	Chave única que identifica o endereço	Inteiro
VELOCIDADE_DA_VIA	Velocidade máxima permitida na via de registro	Inteiro
VELOCIDADE_AFERIDA	Velocidade do veículo aferida durante o registro do cruzamento pelo equipamento	Decimal
CLASSIFICACAO	Classificação do tipo de veículo, podendo ser MOTO, CAMINHÃO / ÔNIBUS e AUTOMÓVEL	Texto
TAMANHO	Comprimento aproximado do veículo em metros	Decimal
NUMERO_DE_SERIE	Número de série o equipamento	Inteiro
LATITUDE	Latitude onde se localiza o equipamento	Decimal
LONGITUDE	Longitude onde se localiza o equipamento	Decimal
ENDERECO	Endereço onde se localiza o equipamento	Texto
SENTIDO	Sentido de fluxo do veículo registrado pelo equipamento	Texto

2.2. Base Tipos de Dia e Horário

A base de dados com os tipos de dias e horários tem como objetivo trazer as informações de quais dias são úteis, sábados, domingos, feriados e as respectivas horas em que ocorreu o registro de cruzamento de motocicleta. Por exemplo, se existem dois registros, onde um ocorreu no dia 22 de maio de 2022 às 14:23 e ou no dia 22 de maio de 2022 às 14:36, ambos registros são considerados ocorrências dentro das 14h do dia 22 de maio de 2022.

Essa base foi gerada majoritariamente por código, onde a única informação externa foi o dia do feriado de *Corpus Christi*, quanto a avaliação de dia útil ou final de semana, essa foi

feita utilizando a função “weekday” pertencente aos objetos de data e hora que foram gerados.

Para gerar a base de tipos e dia e horário, primeiramente foi criada a função da Figura 7 onde essa recebe um parâmetro de data e retorna o tipo de dia, podendo ser (“DIA_UTIL”, “SÁBADO”, “DOMINGO”, “FERIADO”). Logo após, utilizando a biblioteca *Pandas*, foi gerado um *dataset* contendo todos os dias e horários entre o dia 01 de maio de 2022 às 00:00h e 30 de junho de 2022 às 23:00, podendo ser visto na Figura 8.

Figura 7 – Função para avaliar o tipo de dia

```
def avalia_tipo_dia(data_atual):
    tipo_dia = "DIA_UTIL"
    if data_atual.weekday() == 5:
        tipo_dia = 'SABADO'
    elif data_atual.weekday() == 6:
        tipo_dia = 'DOMINGO'
    elif format(data_atual, "%Y-%m-%d") == '2022-06-16':
        tipo_dia = 'FERIADO'
    return tipo_dia
```

Figura 8 – Código para gerar o dataset de tipos de dia e horários

```
datas = pd.date_range(start = '2022-05-01 00:00:00', end = '2022-06-30 23:00:00', periods = 1464)
base_datas = pd.DataFrame(datas, columns=['DATA'])
base_datas['DATA_HORA_COMPARADOR'] = base_datas['DATA'].apply(lambda data_atual: format(data_atual, "%Y-%m-%dT%H:00:00"))
base_datas['DATA_COMPARADOR'] = base_datas['DATA'].apply(lambda data_atual: format(data_atual, "%Y-%m-%d"))
base_datas['TIPO_DIA'] = base_datas['DATA'].apply(lambda data_atual: avalia_tipo_dia(data_atual))
```

No código da figura 8, além de ter sido gerado o *dataset* com os dias e horários, foi também foi gerada a coluna “DATA_HORA_COMPARADOR” com a finalidade de auxiliar a união de todas as bases construídas ao longo da coleta dos dados.

Por fim, a coluna “TIPO_DIA” foi preenchida utilizando a função “weekday” a partir da coluna “DATA_ATUAL”. Ao final de todo o processo de confecção do *dataset* de tipos de dia e horários, Figura 9, o mesmo foi salvo em um arquivo CSV para facilitar a sua recuperação para o código posteriormente.

Figura 9 – Dataset com dados de tipos de dia e horários

	DATA	DATA_HORA_COMPARADOR	DATA_COMPARADOR	TIPO_DIA
0	2022-05-01 00:00:00	2022-05-01T00:00:00	2022-05-01	DOMINGO
1	2022-05-01 01:00:00	2022-05-01T01:00:00	2022-05-01	DOMINGO
2	2022-05-01 02:00:00	2022-05-01T02:00:00	2022-05-01	DOMINGO
3	2022-05-01 03:00:00	2022-05-01T03:00:00	2022-05-01	DOMINGO
4	2022-05-01 04:00:00	2022-05-01T04:00:00	2022-05-01	DOMINGO
...
1459	2022-06-30 19:00:00	2022-06-30T19:00:00	2022-06-30	DIA_UTIL
1460	2022-06-30 20:00:00	2022-06-30T20:00:00	2022-06-30	DIA_UTIL
1461	2022-06-30 21:00:00	2022-06-30T21:00:00	2022-06-30	DIA_UTIL
1462	2022-06-30 22:00:00	2022-06-30T22:00:00	2022-06-30	DIA_UTIL
1463	2022-06-30 23:00:00	2022-06-30T23:00:00	2022-06-30	DIA_UTIL

Para descrever as colunas do *dataset* da figura 9, foi criada a Tabela 2 com os respectivos nomes das colunas, descrição e tipo.

Tabela 2 – Colunas do dataset da contagem volumétrica de radares

Nome da Coluna	Descrição	Tipo
DATA	Coluna com os registros de data e hora no formato data.	Data
DATA_HORA_COMPARADOR	Coluna com registro de data e hora em formato <i>string</i> padronizado para facilitar posterior união de bases	Texto
DATA_COMPARADOR	Coluna com registro de data em formato <i>string</i> padronizado para facilitar posterior união de bases	Texto
TIPO_DIA	Coluna com informação do tipo de dia, podendo ser “DIA_UTIL”, “SABADO”, “DOMINGO” e “FERIAQDO”	Texto

2.3. Base de Dados Climáticos

A base de dados climáticos foi obtida através da *API open meteo* (<https://open-meteo.com/> acessado em 22 de novembro de 2022) utilizando as informações de latitude, longitude, data e hora de um *dataset* gerado como base que contendo todos os dias e horários do intervalo observado. O código da Figura 10 mostra como o *dataset* foi criado:

Figura 10 – Código para gerar o dataset com dados de tipos de dia e horários

```

datas = pd.date_range(start = '2022-05-01 00:00:00', end = '2022-06-30 23:00:00', periods = 1464)
base_clima = pd.DataFrame(datas, columns=['DATA'])
base_clima['DATA_HORA_COMPARADOR'] = base_clima['DATA'].apply(lambda data_atual: format(data_atual, "%Y-%m-%dT%H:00:00"))
base_clima

```

De acordo com o código apresentado na figura 10, inicialmente foi gerado um *dataset* com todas as datas e horas do período observado e depois foi enriquecido com as colunas contendo os dados de temperatura e precipitação.

Para buscar as informações na *API open meteo*, foi criada a função “*retorna_dados_metereologicos*” que é mostrada na Figura 11:

Figura 11 – Função para busca dos dados climáticos na API open meteo

```
def retorna_dados_metereologicos(data_hora, latitude, longitude):
    dataDate = datetime.strptime(data_hora, '%Y-%m-%dT%H:%M:%S')
    dataStrRequest = dataDate.strftime('%Y-%m-%d')

    #chamada
    url = "https://archive-api.open-meteo.com/v1/era5?latitude="+str(latitude)+"&longitude="+str(longitude)+"&start_date="+str(dataDate.strftime('%Y-%m-%d'))
    response = requests.get(url)
    jsonObj = response.json()

    #pegando atributos
    dataStrComparador = dataDate.strftime('%Y-%m-%dT%H:00')
    indiceDataHora = jsonObj['hourly']['time'].index(dataStrComparador)

    temperature_2m = jsonObj['hourly']['temperature_2m'][indiceDataHora]
    precipitation = jsonObj['hourly']['precipitation'][indiceDataHora]

    return dict({'temperature_2m': temperature_2m, 'precipitation': precipitation})
```

A função da figura 11 recebe os dados de latitude, longitude data e hora e retorna um dicionário da linguagem *Python* contendo a temperatura em graus célsius à 2m do solo e a precipitação em milímetros de acordo com os dados informados inicialmente. Na Figura 12 pode ser visto um exemplo da função “*retorna_dados_metereologicos*” sendo utilizada:

Figura 12 – Função para busca dos dados climáticos na API open meteo

```
teste = retorna_dados_metereologicos("2022-05-01T00:00:00", -19.9377, -43.92711)
teste

{'temperature_2m': 15.8, 'precipitation': 0.0}
```

De posse da função “*retorna_dados_metereologicos*”, o dataset de dados climáticos foi enriquecido com as informações de temperatura e precipitação e por fim, salvo em um arquivo CSV para posterior leitura, o que pode ser verificado no código da Figura 13.

Figura 13 – Carregamento das informações climáticas

```
base_clima['TEMPERATURA'] = np.nan
base_clima['PRECIPITACAO'] = np.nan
for i in base_clima.index:
    dados_clima = retorna_dados_meteorologicos(base_clima['DATA_HORA_COMPARADOR'][i], -19.9377, -43.92711)
    base_clima.loc[i, 'TEMPERATURA'] = dados_clima['temperature_2m']
    base_clima.loc[i, 'PRECIPITACAO'] = dados_clima['precipitation']
base_clima.to_csv("dados/fluxo_moto/base_clima.csv", index=False)
```

Depois de executado o código da figura 13, o mesmo gerou o *dataset* “*base_clima*” que pode ser visto na Figura 14.

Figura 14 – Carregamento das informações climáticas

	DATA	DATA_HORA_COMPARADOR	TEMPERATURA	PRECIPITACAO
0	2022-05-01 00:00:00	2022-05-01T00:00:00	15.8	0.0
1	2022-05-01 01:00:00	2022-05-01T01:00:00	15.1	0.0
2	2022-05-01 02:00:00	2022-05-01T02:00:00	14.7	0.0
3	2022-05-01 03:00:00	2022-05-01T03:00:00	14.4	0.0
4	2022-05-01 04:00:00	2022-05-01T04:00:00	14.4	0.0
...
1459	2022-06-30 19:00:00	2022-06-30T19:00:00	23.2	0.0
1460	2022-06-30 20:00:00	2022-06-30T20:00:00	21.8	0.0
1461	2022-06-30 21:00:00	2022-06-30T21:00:00	17.3	0.0
1462	2022-06-30 22:00:00	2022-06-30T22:00:00	14.3	0.0
1463	2022-06-30 23:00:00	2022-06-30T23:00:00	13.5	0.0

1464 rows x 4 columns

A Tabela 3 contém as informações das colunas presentes nesse *dataset* apresentado na figura 14.

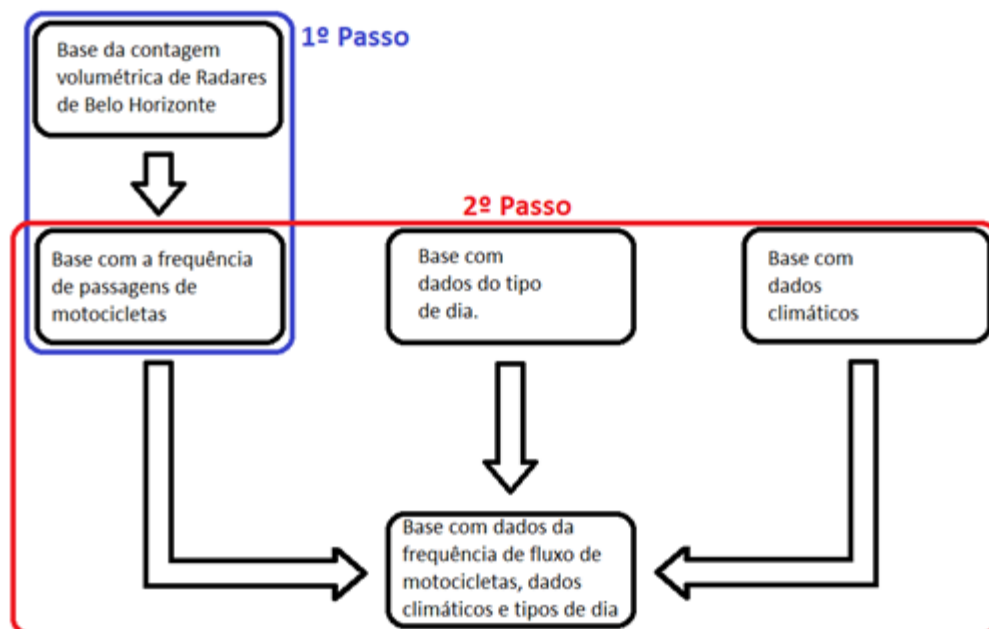
Tabela 3 – Colunas do dataset da contagem volumétrica de radares

Nome da Coluna	Descrição	Tipo
DATA	Coluna com os registros de data e hora no formato data.	Data
DATA_HORA_COMPARADOR	Coluna com registro de data e hora em formato <i>string</i> padronizado para facilitar posterior união de bases	Texto
TEMPERATURA	Temperatura à 2m do solo medidas em graus célsius.	Decimal
PRECIPITAÇÃO	Dado de precipitação em milímetros	Decimal

3. Processamento e Tratamento de Dados

O processamento dos dados foi feito em duas grandes etapas, a primeira foi a transformação da base de contagem volumétrica de radares da cidade de Belo Horizonte em uma base de frequência de cruzamentos de motocicletas no ponto de estudo por dia e horário. A segunda transformação foi a união dos dados de frequência, de dados climáticos e tipos de dia e horários, gerando a base final. A Figura 15 ilustra esse processo.

Figura 15 – Transformação e concatenação dos dados



3.1. Gerando Base de Frequência de Fluxo de Motocicletas por Hora

A base de frequência de cruzamento de motocicletas foi gerada a partir dos registros da contagem volumétrica dos radares, claro, olhando sempre para o mesmo radar da Av. Afonso Pena, esquina com a Rua Maranhão. A base de registro volumétrico foi previamente salva em um arquivo CSV após os dados serem convertidos de JSON para um *dataset* e então, neste momento, foi recuperada do arquivo e processada.

A contagem foi feita por hora, ou seja, contabilizou-se a quantidade de motocicletas que cruzaram o radar estudado na Av. Afonso Pena por hora e foi registrado esse dado na coluna “FREQUENCIA”. A estratégia para tal feito foi zerar as informações de minutos e segundos, deixando apenas as informações de hora. Isso foi feito utilizando a função “*data_hora_minuto*” exposta na Figura 16 e gerando uma nova coluna chamada “HORA_PASSAGEM_MOTO” a partir da coluna “DATA_HORA”.

Figura 16 – Função para zerar os dados de minutos e segundos

```
def data_hora_minuto(data_hora):
    dataDate = datetime.strptime(data_hora, '%Y-%m-%dT%H:%M:%S')
    dataStrRetorno = dataDate.strftime('%Y-%m-%dT%H:00:00')
    return dataStrRetorno
```

Para calcular a frequência simples de cruzamento de motocicletas por hora utilizou-se da função “*value_counts*” no *array* contido na coluna “HORA_PASSAGEM_MOTO” para contar quantos registro repetidos de uma mesma data e hora existem, chegando em uma tabela de frequência. O código utilizado para gerar os dados de frequência simples pode ser visto na Figura 17.

Figura 17 – Código para gerar os dados de frequência simples

```
frequencia_fluxo_motos_data = base_fluxo_motocicleta['HORA_PASSAGEM_MOTO'].value_counts()
```

Logo o código da figura 17 gerou um *Pandas Series*¹ que pode ser vista na Figura 18, onde os dados do *array* são as frequências simples e seus respectivos índices que são as datas e horas em que ocorreu a determinada frequência de passagem de motos no ponto de estudo.

¹ Pandas Series é uma estrutura da biblioteca Pandas que contém um array unidimensional e possuindo índices de referência para os dados.

Figura 18 – Pandas Series contendo os dados de frequência simples do cruzamento de motocicletas

```

2022-06-07T17:00:00    120
2022-06-06T17:00:00    116
2022-05-03T18:00:00    116
2022-06-10T17:00:00    115
2022-06-20T18:00:00    115
...
2022-06-02T04:00:00     1
2022-05-19T02:00:00     1
2022-05-10T03:00:00     1
2022-05-03T03:00:00     1
2022-05-11T04:00:00     1
Name: HORA_PASSAGEM_MOTO, Length: 1458, dtype: int64

```

Para acessar um dado do *Pandas Series* basta fazer como no código da Figura 19, onde neste caso está sendo acessado a frequência de cruzamento de motocicletas exatamente no dia 01 de maio de 2022 às 2h da manhã, logo retornando o valor 21, assim significando que na data e horário informados cruzaram 21 motocicletas no ponto da Av. Afonso Pena estudado.

Figura 19 – Acessando um único dado de frequência de cruzamento de motocicletas

```

frequencia_fluxo_motos_data['2022-05-01T02:00:00']
21

```

Após ter as frequências simples calculadas, um *dataset* base de nome “*base_frequencia_fluxo_moto*” foi criado já com os registros de data e hora, tanto no formato *date* (coluna “DATA”), quanto no formato texto (coluna “DATA_HORA_COMPARADOR”). Depois do *dataset* base gerado com as datas e horas, então criou-se a coluna “FREQUENCIA”. Esta coluna foi populada utilizando a coluna “DATA_HORA_COMPARADOR” como índice do *Pandas Series* de frequências simples, para os casos onde não eram encontrados dados de frequência, utilizou-se zero, pois o método “*value_counts*” conta apenas registros existentes, logo as datas e horas que não existem na base de contagem volumétrica significa que não houve passagem de motocicletas na frente do radar.

Todo o código para fazer a etapa do processamento dos dados descrita no parágrafo anterior pode ser visto na Figura 20.

Figura 20 – Código de criação do dataset base_frequencia_fluxo_moto

```

datas = pd.date_range(start = '2022-05-01 00:00:00', end = '2022-06-30 23:00:00', periods = 1464)
base_frequencia_fluxo_moto = pd.DataFrame(datas, columns=['DATA'])
base_frequencia_fluxo_moto['DATA_HORA_COMPARADOR'] = base_frequencia_fluxo_moto['DATA'].apply(lambda data_atual: format(data_atual, "%Y-%m-%dT%H:%M:%S"))
base_frequencia_fluxo_moto['HORA'] = base_frequencia_fluxo_moto['DATA'].apply(lambda data_atual: format(data_atual, "%H"))

base_frequencia_fluxo_moto['FREQUENCIA'] = np.nan

for i in base_frequencia_fluxo_moto.index:
    try:
        base_frequencia_fluxo_moto.loc[i, 'FREQUENCIA'] = frequencia_fluxo_motos_data[base_frequencia_fluxo_moto['DATA_HORA_COMPARADOR'].loc[i]]
    except:
        base_frequencia_fluxo_moto.loc[i, 'FREQUENCIA'] = 0

base_frequencia_fluxo_moto

```

Ao fim da execução do código da figura 19 o *dataset* pode ser visto na Figura 21, onde o campo “HORA” é apenas a informação de qual hora, independente de data, se refere o registro de frequência, o campo “FREQUENCIA” é a frequência simples do cruzamento de motocicletas no ponto de estudo.

Figura 21 – Dataset base_frequencia_fluxo_moto

	DATA	DATA_HORA_COMPARADOR	HORA	FREQUENCIA
0	2022-05-01 00:00:00	2022-05-01T00:00:00	00	37.0
1	2022-05-01 01:00:00	2022-05-01T01:00:00	01	30.0
2	2022-05-01 02:00:00	2022-05-01T02:00:00	02	21.0
3	2022-05-01 03:00:00	2022-05-01T03:00:00	03	13.0
4	2022-05-01 04:00:00	2022-05-01T04:00:00	04	10.0
...
1459	2022-06-30 19:00:00	2022-06-30T19:00:00	19	60.0
1460	2022-06-30 20:00:00	2022-06-30T20:00:00	20	47.0
1461	2022-06-30 21:00:00	2022-06-30T21:00:00	21	37.0
1462	2022-06-30 22:00:00	2022-06-30T22:00:00	22	40.0
1463	2022-06-30 23:00:00	2022-06-30T23:00:00	23	21.0

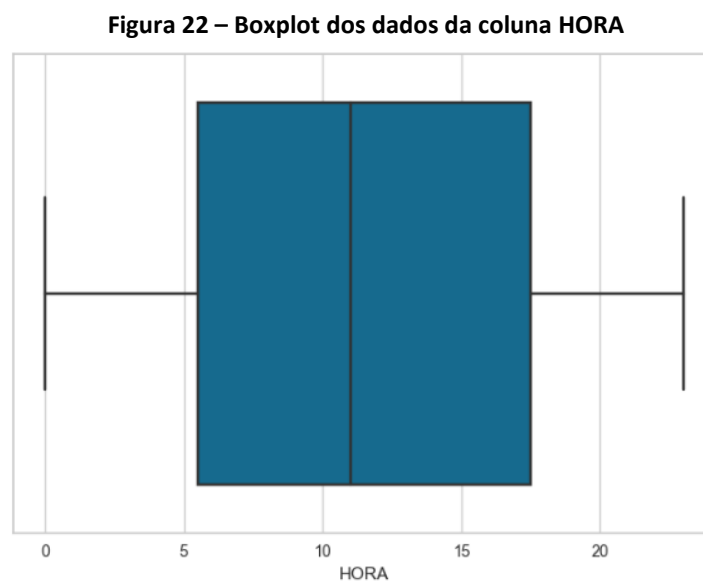
1464 rows x 4 columns

O *dataset* “base_frequencia_fluxo_moto” possui o total de 1464 registros e nenhum pode ser considerado registro duplicados, pois os dados de data e hora das colunas geradoras do *dataset* em questão são únicos, ou seja, não existe mais de uma linha para um mesmo registros de data e hora. Já o valor de 1464 registros se justifica pois o período de tempo de estudo possui exatamente 1464 horas.

3.1.1. Limpando e Transformando a Base de Frequência de Fluxo de Motocicletas

Como neste trabalho optou-se por utilizar algoritmos de *Machine Learning* de classificação, a frequência por ser um dado numérico e não uma classe teve de ser padronizada como uma classe. Então optou-se por criar a classe “ACIMA_MEDIA_FREQUENCIA”. Onde para as frequências acima da média aritmética foi dado o valor 1 e abaixo o valor 0.

O processo de limpeza e transformação dos dados dessa base se iniciou buscando *outliers* através de gráficos *boxplot* (diagramas de caixa). Primeiramente foi analisada a coluna “HORA” onde gerou-se o gráfico *boxplot* presente na Figura 22, podendo ser visto que não há *outliers*.



O mesmo foi feito para a coluna “FREQUENCIA”, lembrando que essa coluna contém os dados da classe alvo deste estudo.

Ao observar o *boxplot* na Figura 23 pode ser visto que estão presentes valores de frequência zero, sendo o limite inferior, valores de 116 como limite superior e um *outlier* com o valor de 120.

Para tratar o *outlier*, decidiu-se excluir o registro, já que de acordo com a Figura 23, existe apenas 1 registro como *outlier* de frequência e já que em um universo de 1464 registros, 1 registro representa apenas 0.07% de toda a amostra.

Após a exclusão do *outlier* percebe-se no gráfico *boxplot* da Figura 24 que não existem mais valores discrepantes quanto aos dados de frequência do cruzamento de motocicletas no ponto de estudo.

Figura 22 – Boxplot dos dados da coluna FREQUENCIA

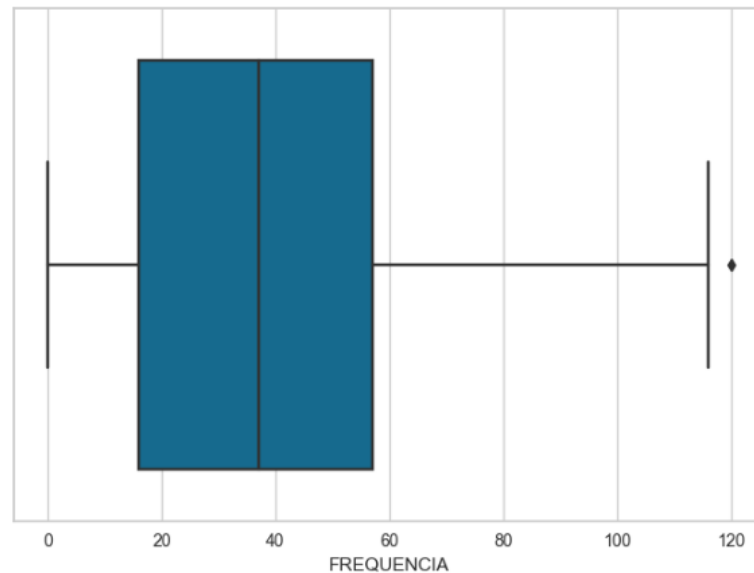
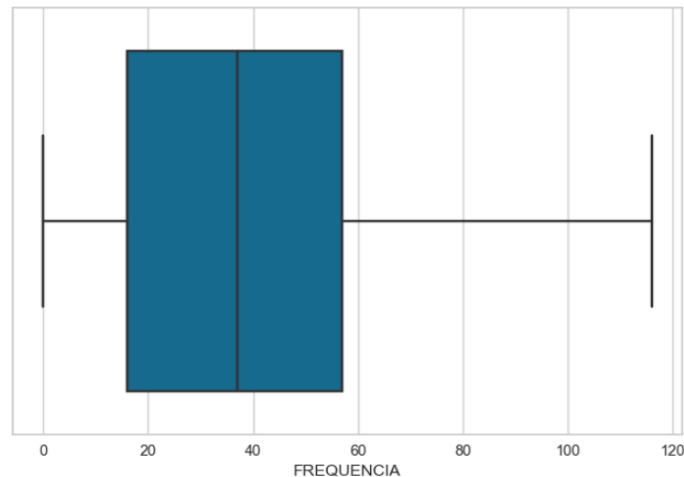


Figura 23 – Visualização do outlier de frequência

```
base_frequencia_fluxo_moto.loc[base_frequencia_fluxo_moto['FREQUENCIA'] > 119]
```

	DATA	DATA_HORA_COMPARADOR	HORA	FREQUENCIA
905	2022-06-07 17:00:00	2022-06-07T17:00:00	17	120.0

Figura 24 – Boxplot dos dados da coluna FREQUENCIA sem outliers



Ao utilizar o código da Figura 25 pode ser visto algumas informações da coluna “FREQUENCIA” do *dataset* de frequência de cruzamento de motocicletas por hora no ponto de estudo.

Figura 25 – Boxplot dos dados da coluna FREQUENCIA sem outliers

```
base_frequencia_fluxo_moto['FREQUENCIA'].describe()
count      1463.000000
mean        39.707450
std         27.439559
min          0.000000
25%         16.000000
50%         37.000000
75%         57.000000
max        116.000000
Name: FREQUENCIA, dtype: float64
```

Os dados presentes na figura 25 representam exatamente o que está expresso no *boxplot* da figura 24. A média geral das frequências de cruzamentos de motocicletas por hora no ponto estudado ficou em 39.7 cruzamentos por hora aproximadamente, o limite inferior em 0 e o superior em 116, o primeiro quartil em 16 e o terceiro quartil em 57 cruzamentos por hora.

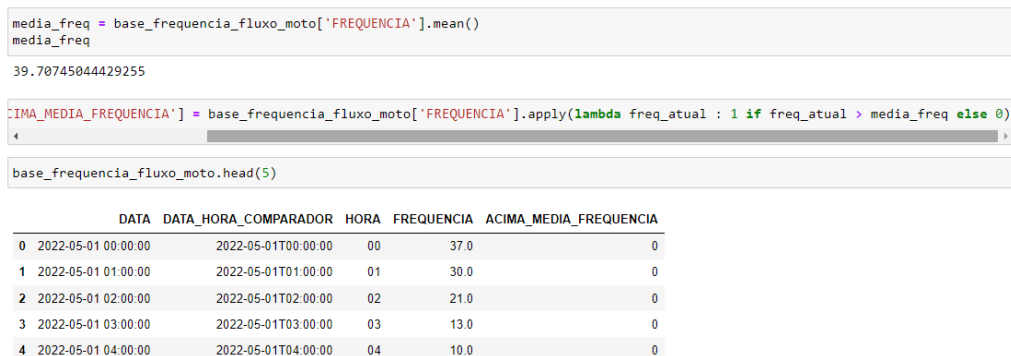
Como já mencionado, serão utilizados algoritmos de *Machine Learning* de classificação para estudar o quanto alguns fatores externos influenciam num alto fluxo ou baixo fluxo de

motocicletas no ponto de estudo, foi necessário transformar o dado numérico discreto da frequência simples de cruzamentos de motocicletas por hora em um dado categórico. Resolveu-se criar uma coluna de nome “ACIMA_MEDIA_FREQUENCIA” que indica se o registro se encontra acima ou abaixo da média das frequências registradas.

Para construir tal coluna com os dados, utilizou-se da codificação da Figura 26, onde primeiro foi calculada a média da frequência, depois iterou-se sob a coluna “FREQUENCIA” aplicando uma função que retorna 0 caso o valor da frequência seja menor que a média previamente calculada e 1 caso o valor seja maior que a média. Após esse processo, pode-se observar o *dataset* com a nova coluna de dados.

Por fim, esse *dataset* foi salvo em um arquivo de extensão CSV, de nome “base_frequencia_moto.csv” com a intenção de facilitar a recuperação dessas informações posteriormente.

Figura 26 – Criação da coluna ACIMA_MEDIA_FREQUENCIA



3.2. Mesclando as Bases de Frequência de Fluxo, de Clima e de Dias

Nesta etapa do trabalho será feito o último passo indicado na figura 15, onde os *datasets* de frequência, de dados climáticos e de dados de tipo de dia serão unidos. Para tal tarefa foi tomado o cuidado de criar a coluna “DATA_HORA_COMPARADOR” nos três *datasets*, assim facilitando a junção dos dados. Essa coluna contém as informações de data e hora e como essas informações não se repetem em nenhum *dataset* por serem inseridos de maneira controlada via código, essa coluna pode ser vista como uma espécie de chave primária e estrangeira, correlacionando os três *datasets* de forma bem simples.

Para realizar a tarefa de unir as três bases de dados, essas foram recuperadas dos arquivos CSV previamente salvos, mostrado na Figura 27.

Figura 27 – Recuperação dos três datasets dos respectivos arquivos

```
base = ""
base_tipo_dia = pd.read_csv("dados/fluxo_moto/base_datos.csv")
base_tipo_dia.head(3)
```

	DATA	DATA_HORA_COMPARADOR	DATA_COMPARADOR	TIPO_DIA
0	2022-05-01 00:00:00	2022-05-01T00:00:00	2022-05-01	DOMINGO
1	2022-05-01 01:00:00	2022-05-01T01:00:00	2022-05-01	DOMINGO
2	2022-05-01 02:00:00	2022-05-01T02:00:00	2022-05-01	DOMINGO

```
base_clima = pd.read_csv("dados/fluxo_moto/base_clima.csv")
base_clima.head(3)
```

	DATA	DATA_HORA_COMPARADOR	TEMPERATURA	PRECIPITACAO
0	2022-05-01 00:00:00	2022-05-01T00:00:00	15.8	0.0
1	2022-05-01 01:00:00	2022-05-01T01:00:00	15.1	0.0
2	2022-05-01 02:00:00	2022-05-01T02:00:00	14.7	0.0

```
base_frequencia_fluxo_moto = pd.read_csv("dados/fluxo_moto/base_frequencia_moto.csv")
```

```
base_frequencia_fluxo_moto.head(3)
```

	DATA	DATA_HORA_COMPARADOR	HORA	TIPO_DIA	PRECIPITACAO	TEMPERATURA	FREQUENCIA	ACIMA_MEDIA_FREQUENCIA
0	2022-05-01 00:00:00	2022-05-01T00:00:00	0	DOMINGO	0.0	15.8	37.0	0
1	2022-05-01 01:00:00	2022-05-01T01:00:00	1	DOMINGO	0.0	15.1	30.0	0
2	2022-05-01 02:00:00	2022-05-01T02:00:00	2	DOMINGO	0.0	14.7	21.0	0

Para unir os três *datasets*, o primeiro passo foi atribuir o *dataset* de fluxos de motocicletas à uma variável chamada “*base*” e depois as colunas com os dados de temperatura, precipitação e tipo de dia foram mapeadas para o *dataset* “*base*”, como mostra na Figura 28.

Figura 28 – Código de união dos três datasets

```
base = base_frequencia_fluxo_moto
```

```
base.insert(loc = 3, column = 'TEMPERATURA', value = base['DATA_HORA_COMPARADOR'].map(lambda x : base_clima.loc[base_clima.DATA_HORA_COMPARADOR == x].iloc[0]['TEMPERATURA']))
```

```
base.insert(loc = 3, column = 'PRECIPITACAO', value = base['DATA_HORA_COMPARADOR'].map(lambda x : base_clima.loc[base_clima.DATA_HORA_COMPARADOR == x].iloc[0]['PRECIPITACAO']))
```

```
base.insert(loc = 3, column = 'TIPO_DIA', value = base['DATA_HORA_COMPARADOR'].map(lambda x : base_tipo_dia.loc[base_tipo_dia.DATA_HORA_COMPARADOR == x].iloc[0]['TIPO_DIA']))
```

```
base = base.drop(['DATA_HORA_COMPARADOR', 'DATA', 'FREQUENCIA'], axis=1)
```

Ao final da execução do código da figura 28 gerou-se o *dataset* exibido conforme a Figura 29.

Figura 29 – Dataset resultante com todos os dados do estudo

	HORA	TIPO_DIA	PRECIPITACAO	TEMPERATURA	ACIMA_MEDIA_FREQUENCIA
0	0	DOMINGO	0.0	15.8	0
1	1	DOMINGO	0.0	15.1	0
2	2	DOMINGO	0.0	14.7	0
3	3	DOMINGO	0.0	14.4	0
4	4	DOMINGO	0.0	14.4	0
...
1458	19	DIA_UTIL	0.0	23.2	1
1459	20	DIA_UTIL	0.0	21.8	1
1460	21	DIA_UTIL	0.0	17.3	0
1461	22	DIA_UTIL	0.0	14.3	1
1462	23	DIA_UTIL	0.0	13.5	0

1463 rows x 5 columns

Ao final do processo de unificação dos dados, obteve-se um *dataset* com 1463 registros, onde nenhum dado nulo foi encontrado, como mostra a Figura 30, porém como mostra a Figura 31, observou-se que existem 197 registros duplicados, onde optou-se por remover esses registros, já que não existem mais referência de datas neles e os mesmos podem tornar os modelos de *Machine Learning* mais propensos ao fenômeno de *overfitting*.

Figura 30 – Dados nulos no dataset resultante

```
base.isnull().sum()
HORA                0
TIPO_DIA            0
PRECIPITACAO        0
TEMPERATURA         0
ACIMA_MEDIA_FREQUENCIA 0
dtype: int64
```

Figura 31 – Dados duplicados no dataset resultante

```
base.duplicated().sum()
```

197

Após apagar os registros duplicados no *dataset* resultante, foi gerado o *dataset* final mostrado na Figura 32, onde o mesmo possui 1266 registros e 5 colunas de dados, sendo elas: “HORA”, “TIPO_DIA”, “PRECIPITACAO”, “TEMPERATURA” e “ACIMA_MEDIA_FREQUENCIA”. Por fim esse *dataset* foi salvo em um arquivo CSV de nome “*base_fluxo_motos_tratada.csv*” para facilitar posterior recuperação.

Figura 32 – Dataset final

	HORA	TIPO_DIA	PRECIPITACAO	TEMPERATURA	ACIMA_MEDIA_FREQUENCIA
0	0	DOMINGO	0.0	15.8	0
1	1	DOMINGO	0.0	15.1	0
2	2	DOMINGO	0.0	14.7	0
3	3	DOMINGO	0.0	14.4	0
4	4	DOMINGO	0.0	14.4	0
...
1452	13	DIA_UTIL	0.0	18.7	1
1454	15	DIA_UTIL	0.0	22.6	1
1456	17	DIA_UTIL	0.0	23.6	1
1458	19	DIA_UTIL	0.0	23.2	1
1460	21	DIA_UTIL	0.0	17.3	0

1266 rows × 5 columns

A partir do *dataset* da figura 32 foi feita a exploração dos dados e análises dos mesmos a fim de extrair informações e entender como esses dados correlacionam-se entre si, assim sendo possível perceber como influenciam na ocorrência de um fluxo de motocicletas acima ou abaixo da média no ponto de estudo.

4. Análise e Exploração dos Dados

Nesta seção primeiramente procurou-se analisar as colunas individualmente a fim de entender os dados disponíveis em cada coluna do *dataset* bem como os seus balanceamentos. Por fim, foram feitas algumas correlações entre as colunas procurando possíveis padrões de influência de duas ou mais colunas na classe alvo “ACIMA_MEDIA_FREQUENCIA”.

4.1. Visualização e Análise dos Dados por Coluna

Esta é uma análise mais simples com relação a que será apresentada na próxima seção, pois analisar as colunas de um *dataset* separadamente não possibilita a obtenção de muitos insights quanto à análise da correlação entre as colunas e a classe alvo, porém essa análise se faz necessária para conhecer a base que está sendo estudada e se há alguma anomalia que possa atrapalhar os estudos posteriormente.

Cada coluna do *dataset* presente na figura 32 foi exibida em forma de gráficos a fim de mostrar de maneira resumida o perfil de cada campo da base, dessa forma é possível entender a dimensão das informações e se os dados estão desbalanceados ou balanceados, podendo avaliar assim uma possível influência negativa como por exemplo um *overfitting* nos algoritmos de *Machine Learning*.

Para desenhar os gráficos foram construídas duas funções: “*plota_countplot*”, Figura 33, e “*plota_histplot*”, Figura 34. Essas funções têm como finalidade formatar alguns dos gráficos que serão exibidos, não sendo utilizadas em todas as construções gráficas posteriores.

Figura 33 – Função *plota_countplot*

```
def plota_countplot(base_principal, variavel, titulo_x, titulo_y, titulo_do_grafico):
    plt.clf()
    grafico = sns.countplot(x=base_principal[variavel])
    grafico.set_ylabel('Quantidade de Ocorrências');
    grafico.set_title(f'{titulo_do_grafico}\n');
    plt.show()
```

Figura 34 – Função *plota_histplot*

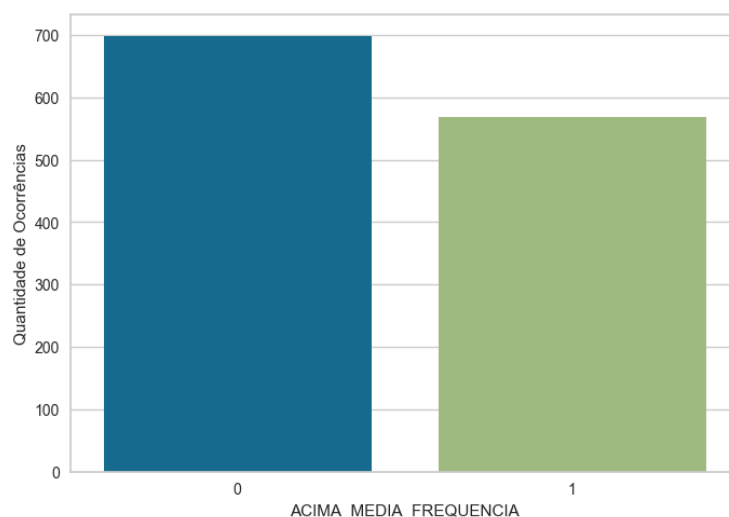
```
def plota_histplot(base_principal, variavel, titulo_x, titulo_y, titulo_do_grafico):
    plt.clf()
    grafico = sns.histplot(x=base_principal[variavel])
    grafico.set_ylabel('Quantidade de Ocorrências');
    grafico.set_title(f'{titulo_do_grafico}\n');
    plt.show()
```

4.1.1. Coluna ACIMA_MEDIA_FREQUENCIA

Como o objetivo deste trabalho é analisar o que influencia na frequência de cruzamentos de motocicletas no ponto de estudo na cidade de Belo Horizonte, a primeira coluna analisada foi a “ACIMA_MEDIA_FREQUENCIA”. Essa é uma variável categórica nominal e representa se houve uma frequência de cruzamento de motocicletas maior ou menor que a média de cruzamentos por hora no ponto estudado.

Esta é a variável alvo do estudo e de acordo com o gráfico mostrado na Figura 35, está levemente desbalanceada, contando com 750 registros abaixo da média e 603 acima da média.

Figura 34 – Ocorrências da variável ACIMA_MEDIA_FREQUENCIA

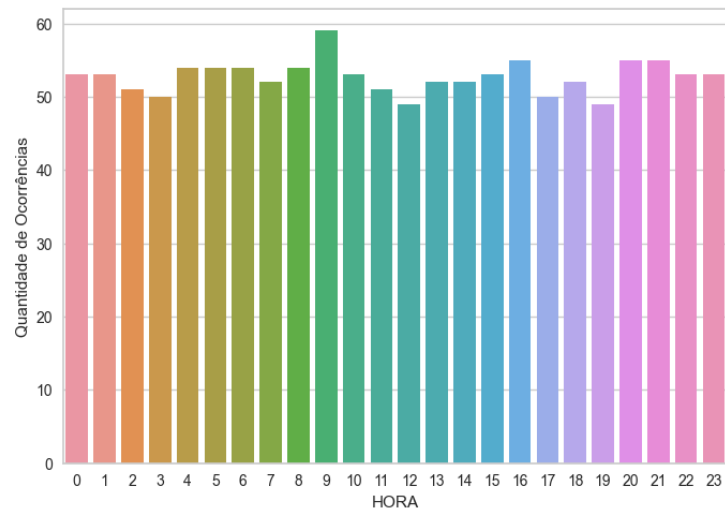


4.1.2. A Coluna HORA

A coluna HORA representa uma variável numérica ordinal que é exatamente o valor inteiro da hora a qual se refere o registro de cruzamento de motocicletas em frente ao radar estudado. Os dados dessa coluna foram mostrados em um gráfico do tipo *countplot*, Figura 35, e percebe-se uma certa homogeneidade quando à quantidade de registros em cada hora.

Apesar dos dados relativos à variável hora estarem bem equilibrados quanto ao balanceamento, os registros não apresentam a mesma quantidade para cada hora pois na etapa de limpeza dos dados foram eliminados os registros duplicados.

Figura 35 – Ocorrências da variável HORA



4.1.3. A Coluna TEMPERATURA

A coluna TEMPERATURA é uma variável numérica contínua e representa a temperatura na hora em que foi feita a medição da frequência de cruzamentos de motocicletas no ponto estudado. Para a análise da variável temperatura, construiu-se tanto um histograma, Figura 36, quanto um diagrama de caixas (*boxplot*), Figura 37, percebeu-se que não existem *outliers*.

Quanto à variável de temperatura, a mesma não se apresenta balanceada pois é uma variável contínua e que varia ao longo das horas de um dia.

Figura 36 – Ocorrências da variável TEMPERATURA

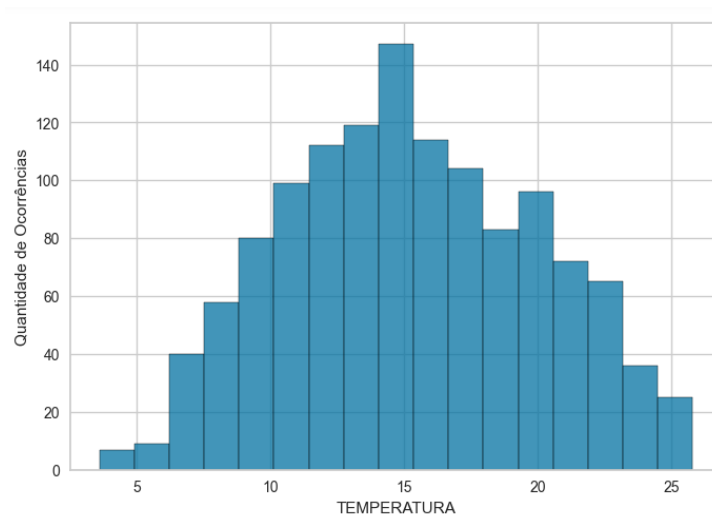
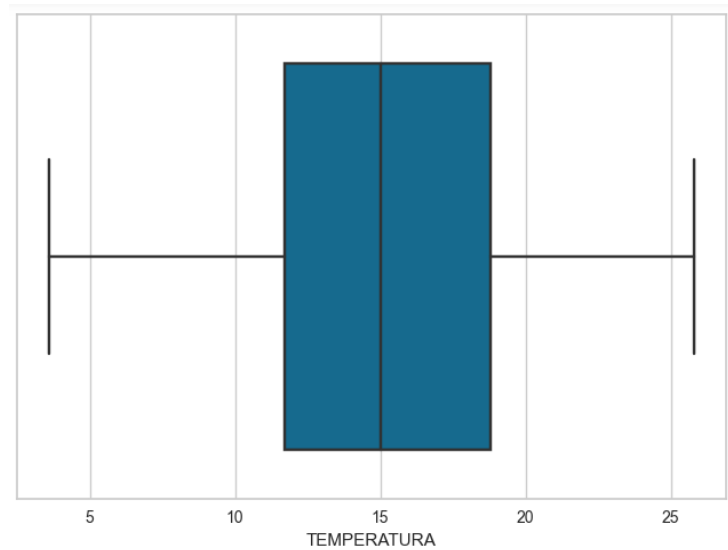
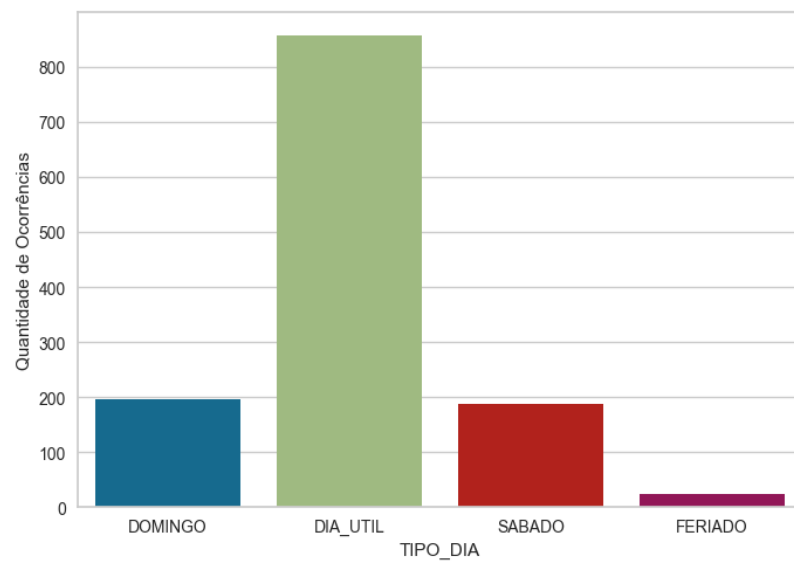


Figura 37 – Boxplot da variável TEMPERATURA

4.1.4. A Coluna TIPO_DIA

A coluna TIPO_DIA é uma variável categórica nominal e classifica o dia da observação em dia útil, sábado, domingo ou feriado. De acordo com o gráfico de contagem de registros mostrado na Figura 38, pode-se perceber que a grande maioria dos registros foram durante dias úteis, o que já se espera pois foram observados todos os dias durante os meses de maio e junho de 2022.

Figura 38 – Ocorrências da variável TIPO_DIA

4.2. Exploração dos Dados Correlacionando Colunas

Para encontrar informações relacionadas ao tema do estudo nos dados da base apresentada na figura 32, se faz necessário correlacionar suas colunas e perceber como uma variável se comporta com a variação da outra, logo essa seção se dedica a correlacionar as colunas da base de dados desse estudo e obter pistas da influência das variáveis preditoras na variável alvo.

4.2.1. Quantidade de Ocorrências da Variável ACIMA_MEDIA_FREQUENCIA por TIPO_DIA

A primeira correlação feita foi da variável predecessora TIPO_DIA e a variável alvo ACIMA_MEDIA_FREQUENCIA, onde o objetivo foi explicitar a influência do tipo de dia na intensidade de fluxo de motocicletas no ponto estudado. Com a observação do gráfico construído abaixo percebe-se que apenas em dias úteis ocorrem frequências de fluxos acima da média maiores que as ocorrências de frequências abaixo da média.

Para construir um gráfico onde fosse possível correlacionar a quantidade de ocorrência de cada classe da variável alvo com relação ao tipo de dia, foi desenvolvido o código da Figura 39 que por sua vez gerou o gráfico da Figura 40.

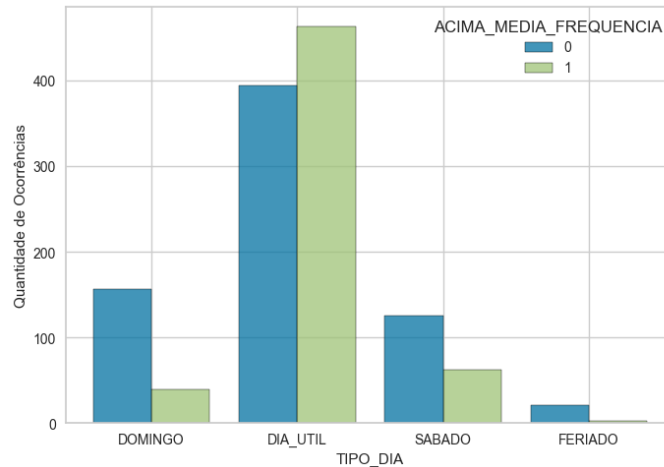
Figura 39 – Código para correlacionar a classe alvo com a variável tipo de dia

```
plt.clf()
grafico = sns.histplot(data=base, x="TIPO_DIA", hue="ACIMA_MEDIA_FREQUENCIA", multiple="dodge", shrink=0.8)
grafico.set_title(f'Ocorrência de fluxos acima e abaixo da média por tipo de dia de registro\n');
grafico.set_ylabel('Quantidade de Ocorrências');
plt.show()
```

No gráfico da figura 40 percebe-se que apenas em dias úteis ocorrem frequências de fluxos acima da média maiores que as ocorrências de frequências abaixo da média enquanto nos finais de semanas e feriados ocorre o oposto, onde ao longo dos dias ocorrem mais frequências abaixo da média que acima. Esse comportamento pode ser atribuído às atividades comerciais e o funcionamento de diversos sistemas na cidade, onde são mais intensos em dias úteis.

Como de se esperar, a quantidade de ocorrências de frequências de cruzamento de motocicletas acima da média nos dias de sábado são maiores que nos dias de domingo, já que nos sábados tendem a funcionar algumas atividades em meio período.

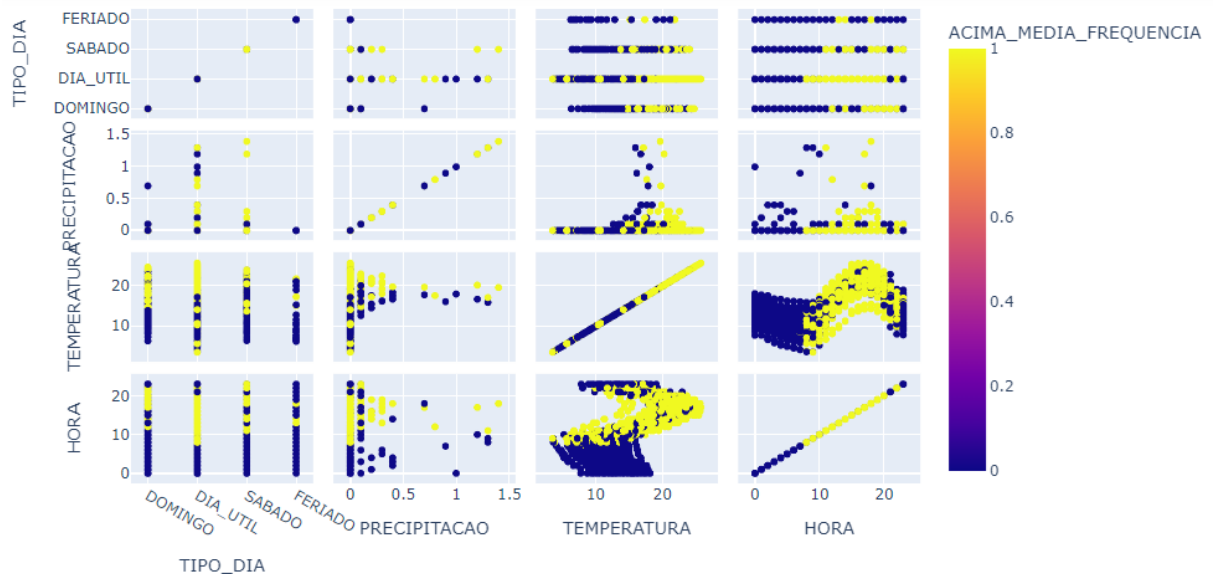
Figura 40 – Gráfico de correlação da classe alvo com a variável tipo de dia



4.2.2. Correlacionando Todas as Colunas com a Classe Alvo

Para observar em apenas um gráfico como as variáveis preditoras influenciam na variável alvo (ACIMA_MEDIA_FREQUENCIA), foi construído um gráfico do tipo "scatter_matrix" que pode ser visto na Figura 41.

Figura 41 – Correlação de todas as colunas com a classe alvo



No gráfico da figura 41 é possível observar que algumas variáveis influenciam mais no fato da frequência de cruzamento de motocicletas ficarem acima ou abaixo da média. O Tipo de dia quando comparado com a hora gera uma visualização gráfica bastante heterogênea quanto à variável alvo, é possível perceber tipos de dias e horários em que as frequências de cruzamentos de motocicletas ficam acima e abaixo da média. Por outro lado, os dados climáticos não se mostram fontes de gráficos com bons contrastes, principalmente quando analisada somente a precipitação.

Como as variáveis preditoras se originam das bases de dados climáticos e tipos de dia (que tem relação com horários), optou-se por fazer a mesma análise para suas respectivas variáveis, colocando-as em dois grupos, um de dados de horário e de tipo de dia e em outro grupo os dados climáticos.

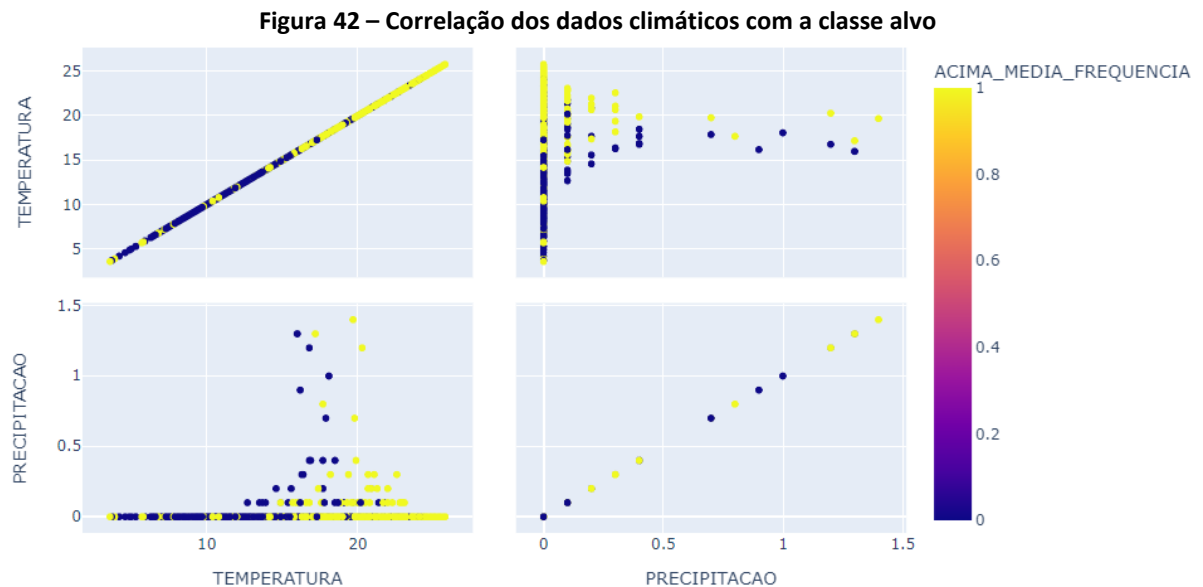
4.2.3. Exploração dos Dados Climáticos

Para analisar apenas como os dados climáticos influenciam na variável alvo, um gráfico do tipo "*scatter_matrix*" foi construído, Figura 42, onde pode-se observar que a variável de temperatura apresenta uma certa tendência a influenciar no fluxo de motocicletas, pois pode ser visto no gráfico um bom contraste entre as amostras acima da média e abaixo da média de frequência, portanto, mostrando que para temperaturas mais altas as frequências ficam acima da média e para temperaturas mais baixas, abaixo da média.

Os dados de precipitação por sua vez não mostram uma boa heterogeneidade quanto à variável alvo, mostrando que tanto para precipitações altas e baixas temos fluxos de motocicletas altos e baixos, o que leva a crer que independente da precipitação a maioria dos motociclistas não deixam de utilizar suas motocicletas.

Diante destas informações, pode-se observar que os dados de precipitação não tendem a influenciar de forma significativa na frequência de cruzamentos de motocicletas no ponto de estudo. Como os dados temperatura que tende a ser um fator menos influenciador no desconforto de um condutor de motocicletas consegue ainda ser melhor que a precipitação? Isso se deve ao fato da temperatura poder carregar as informações do horário, onde as madrugadas que tendem a ter menos fluxo de motocicletas também tendem a ser

mais frias e os dias por volta do horário comercial, mais quentes. Assim se mostrando uma variável com certa influência na frequência de cruzamento de motocicletas quanto a sua média.



4.2.4. Exploração dos Dados do Tipo de Dia e Horário

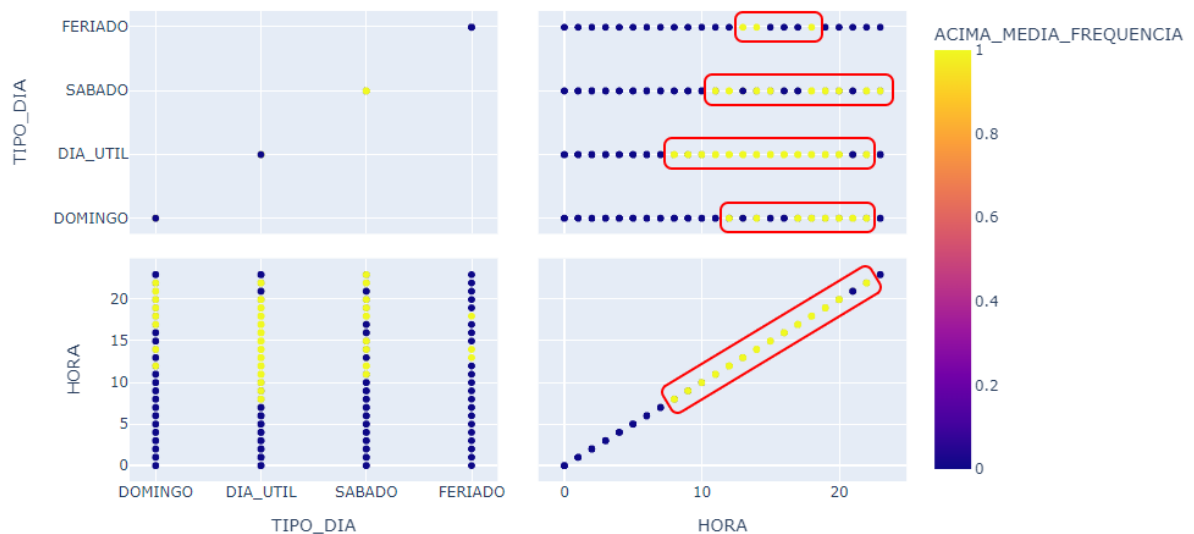
De acordo com o gráfico "*scatter_matrix*", presente na Figura 43, que foi construído para os dados de tipo de dia e hora, pode-se concluir que essas variáveis conseguem descrever de forma mais precisa a frequência de cruzamento de motocicletas no ponto estudado, já que ao confrontar as duas variáveis pode-se observar um alto contraste entre as frequências acima da média (em amarelo) e abaixo da média (em azul).

Desse gráfico consegue-se observar que em dias úteis as frequências de cruzamento de motocicletas tendem ser maiores que a média, isso no intervalo que vai das 9h até às 22h, enquanto nos finais de semana e no feriado a tendência geral é ser menor.

Outra observação interessante é que o fluxo de motocicletas no sábado tem predominância acima da média em horários mais variados durante o dia que nos domingos, onde tende a ser mais alta nos horários do final do dia. Isso pode ser explicado pelas atividades normais da cidade que apesar de menor que nos dias úteis, acontecem também aos sábados.

Outro ponto interessante é que a variável hora quando cruzada com ela mesmo apresenta um alto contraste, podendo-se observar horários de frequências acima da média em um período bem definido do dia, das 9h até às 22h. Esse fato sugere que o horário deve ser o maior influenciador de frequências de cruzamentos de motocicletas acima ou abaixo da média.

Figura 43 – Correlação dos dados de tipo de dia e horário



4.3. Exploração dos Dados com Enfoque na Classe Alvo

Como a variável de hora apresentou-se na análise anterior a mais significativa para a influenciar se existirá um fluxo de motocicletas acima ou abaixo da média, tanto os dados de tipo de dia quanto dados climáticos foram exibidos e analisados com relação à variável de hora em gráficos de calor.

4.3.1. Analisando Dados de Tipo de Dia e Horário

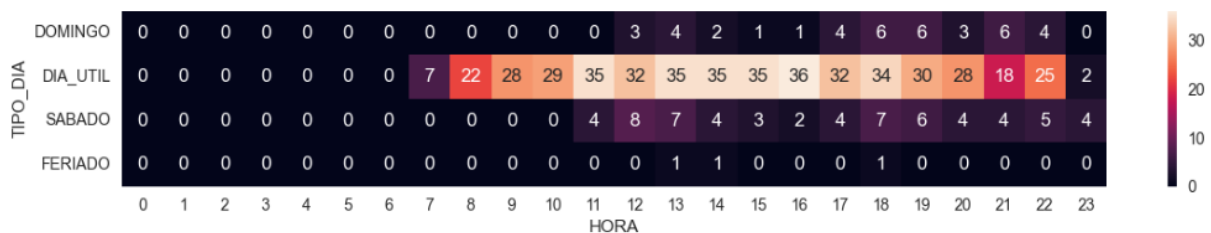
Como um gráfico de calor, não pode ser construído utilizando os dados como se encontram atualmente na base de dados final, foi construída uma função, exposta na Figura 44, para receber a base e transformar os dados de interesse em uma matriz de hora por tipo de dia onde as variações de cores se dão pela variável alvo, ou seja, a variável alvo é o contraste do gráfico.

Após a construção dessa função, a mesma foi executada onde foram passados alguns parâmetros e gerou-se o mapa de calor apresentado na Figura 45.

Figura 44 – Função para gerar o mapa de calor de hora por tipo de dia

```
def plota_heatmap_1(rotulo_y, rotulo_x, rotulo_preenchimento, foco_preenchimento, largura = 5, altura = 5, titulo = ''):
    plt.clf()
    df = base.loc[:, [rotulo_y, rotulo_x, rotulo_preenchimento]]
    colunas = df[rotulo_x].unique()
    indices = df[rotulo_y].unique()
    mapa = pd.DataFrame(columns=colunas, index = indices)
    for i in mapa.columns:
        for j in mapa.index:
            mapa[i][j] = pd.to_numeric(df.loc[df[rotulo_y]==j].loc[df[rotulo_x]==i].loc[df[rotulo_preenchimento]].shape[0])
    mapa[i] = pd.to_numeric(mapa[i], errors = 'coerce')
    plt.figure(figsize = (largura, altura))
    grafico = sns.heatmap(mapa, annot=True, linewidths=0)
    grafico.set_title(f'{titulo}\n')
    grafico.set_ylabel(rotulo_y);
    grafico.set_xlabel(rotulo_x);
    plt.show()
plota_heatmap_1('TIPO_DIA', 'HORA', 'ACIMA_MEDIA_FREQUENCIA', 1, 14, 2, 'Mapa de Calor Indicando de Fluxo Acima da Média')
```

Figura 45 – Mapa de calor de hora por tipo de dia



Na análise dos dados de tipo de dia e horário, assim como na análise dos gráficos "scatter_matrix" pode-se perceber que a maior ocorrência de altas frequências cruzamento de motocicletas ocorrem em dias úteis, onde inicia-se sua intensificação às 10 horas da manhã, chegando até às 22h, como mencionado, provavelmente se dá pelas atividades normais de transporte de pessoas e trabalhadores que utilizam motocicletas em seus trabalhos (atividades comerciais por exemplo).

Já no sábado percebe-se um aumento da frequência das 11h até às 14h, após esse horário costuma diminuir a intensidade das atividades comerciais. Observou-se também ocorrência de frequências acima da média das 17 às 22h.

O dia de domingo mostra um comportamento parecido com o sábado, porém no horário do almoço a frequência se mostra menor que no sábado, provavelmente influenciado por menos atividade comercial neste horário no domingo que no sábado.

Nos finais de semana também é percebido que tanto no sábado quando no domingo há um aumento de frequência mais para o fim do dia, variando de 17h às 22h. Como são

horários em que as atividades comerciais são baixas, talvez pode-se atribuir esse aumento de frequência à utilização das motocicletas para transporte pessoal.

O Feriado diferente de todos os outros dias apresenta baixo fluxo de motocicletas, apenas registrando fluxo mais alto às 14h e às 18h.

4.3.2. Analisando Dados de Clima

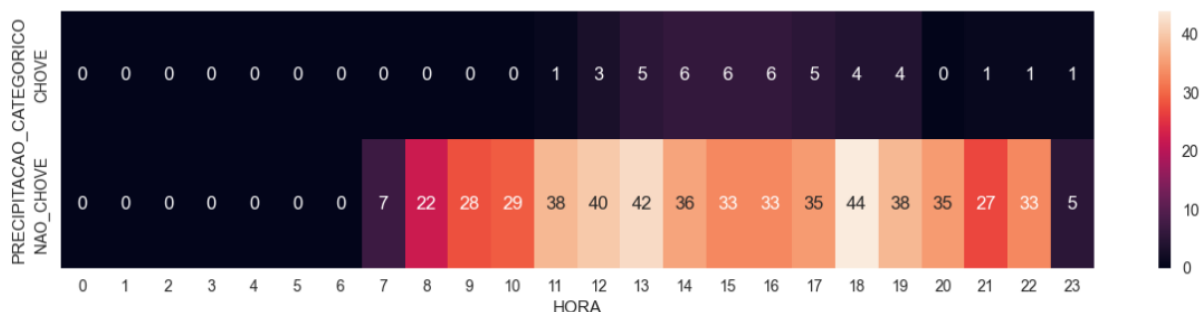
Assim como na seção 4.3.1 mostrou-se a função criada para plotar o mapa de calor para as variáveis lá abordadas, aqui também houve a necessidade de criar uma função para construir o mapa de calor dos dados de precipitação por hora e temperatura por hora.

A função da figura 46 foi criada para construir o mapa de calor dos dados de precipitação por hora, que está exposto na Figura 47.

Figura 46 – Função para gerar o mapa de calor da precipitação por hora

```
def plota_heatmap_2(rotulo_y, rotulo_x, rotulo_preenchimento, foco_preenchimento, largura = 5, altura = 5, titulo = ''):
    plt.clf()
    base[rotulo_x] = base[rotulo_x].apply(lambda x : round(int(x)))
    base['PRECIPITACAO_CATEGORICO'] = base[rotulo_y].apply(lambda x : "NAO_CHOVE" if x == 0.0 else "CHOVE")
    rotulo_y = 'PRECIPITACAO_CATEGORICO'
    df = base.loc[:, [rotulo_y, rotulo_x, rotulo_preenchimento]]
    colunas = sorted(df[rotulo_x].unique())
    indices = sorted(df[rotulo_y].unique())
    mapa = pd.DataFrame(columns=colunas, index=indices)
    for i in mapa.columns:
        for j in mapa.index:
            mapa[i][j] = pd.to_numeric(df.loc[df[rotulo_y]==j].loc[df[rotulo_x]==i].loc[df[rotulo_preenchimento] == foco_preenchimento].shape[0])
    mapa[i] = pd.to_numeric(mapa[i], errors = 'coerce')
    plt.figure(figsize = (largura, altura))
    grafico = sns.heatmap(mapa, annot=True, linewidths=0)
    grafico.set_title(f'{titulo}\n')
    grafico.set_ylabel(rotulo_y);
    grafico.set_xlabel(rotulo_x);
    plt.show()
plota_heatmap_2('PRECIPITACAO', 'HORA', 'ACIMA_MEDIA_FREQUENCIA', 1, 15, 3, 'Mapa de Calor Indicando Registro de Frequência de Fluxo Acima da Média')
```

Figura 47 – Mapa de calor da precipitação por hora



Para a análise dos dados de chuva, como existem poucos registros de precipitações, criou-se uma variável categórica nominal de nome 'PRECIPITACAO_CATEGORICO', que foi preenchida com a informação 'CHOVE' e 'NAO_CHOVE'. Isso foi feito com a finalidade de

Analisando a temperatura com relação ao horário percebe-se no gráfico da figura 49 a temperatura se correlacionando com o horário, já que as noites são mais frias que os dias.

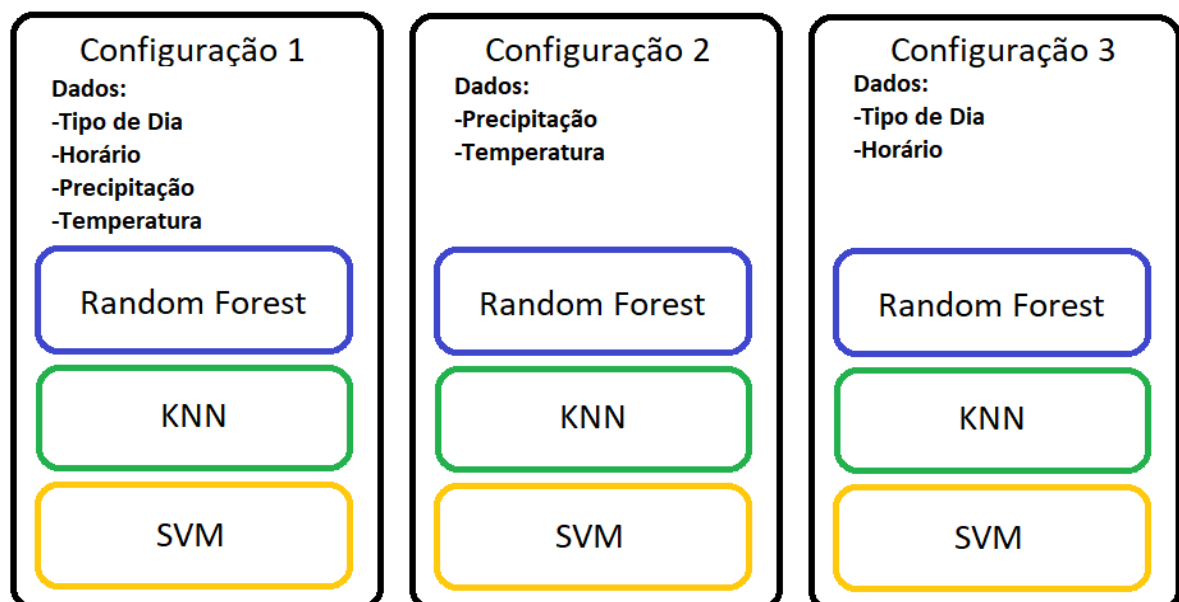
As informações das frequências de cruzamento de motocicletas acima da média variam de forma coordenada com a temperatura, onde para temperaturas mais altas tem-se mais presença de frequências mais altas e nas temperaturas mais baixas menos presença de altas frequências.

Visto o gráfico da figura 49 e os *insights* que a análise do mesmo trouxe, logo percebe-se que a informação de hora está de alguma forma embutida nos dados de temperatura.

5. Criação de Modelos de Machine Learning

Como a análise dos dados foi de certa forma separada em dados de tipos de dias e horários e dados climáticos, na etapa da criação do modelo de *Machine Learning* não foi diferente. Foram construídos e avaliados três modelos diferentes que por sua vez foram utilizados em três algoritmos. A Figura 50 mostra um esquema das configurações dos dados e os algoritmos utilizados.

Figura 50 – Esquema de configuração de dados e algoritmos de Machine Learning



A configurações de dados ficaram da seguinte forma:

- Configuração 1 - Dados de frequência de cruzamento de motocicletas, dados climáticos (temperatura e precipitação) e dados de tipo de dia e horário;
- Configuração 2 - Dados de frequência de cruzamento de motocicletas e dados climáticos (temperatura e precipitação);
- Configuração 3 - Dados de frequência de cruzamento de motocicletas e dados de tipo de dia e horário.

Dessa forma, se tornou possível avaliar através do desempenho dos modelos quais conjuntos de dados influenciam mais ou menos em altas ou baixas frequências de cruzamento de motocicletas no ponto estudado.

Em todas as 3 configurações, foram utilizadas estratégias de otimização (*model tuning*) e pôr fim a validação cruzada. Para a otimização dos parâmetros de configurações dos algoritmos, utilizou-se a *GridSearchCV* do pacote *sklearn* e para a validação cruzada utilizou-se o *kfold* do pacote *sklearn*. Para a validação cruzada, optou-se por fazer 30 vezes o teste de validação cruzada com o *kfold* e depois calculou-se uma média dos resultados para cada algoritmo. No final da execução de todos os testes, os resultados foram apresentados em uma tabela e em gráficos para melhorar a capacidade de interpretação e análise.

5.1. Construindo Modelo de Machine Learning com Dados de Fluxo de Motocicletas, Dados Climáticos, Tipo de Dia e Horário (Configuração 1)

Na seção 3.1 após ter finalizado os tratamentos, enriquecimentos e limpeza da base a mesma foi salva em um arquivo de nome “*base_fluxo_motos_tratada*” com a extensão .CSV para facilitar a sua posterior recuperação.

A base foi lida do arquivo onde apresenta o formato exposto na Figura 32. Como os algoritmos de *Machine Learning* escolhidos trabalham com dados numéricos, a coluna “TIPO_DIA” por conter um dado categórico, deve de ser convertida em numérico, para isso utilizou-se a função “*get_dummies*” da biblioteca *pandas*, que resultou na base exibida na Figura 51. Neste processo, basicamente os dados categóricos da coluna “TIPO_DIA”, foram convertidos em 4 colunas com dados numéricos, podendo ser 0 ou 1.

Com a base transformada, chegou o momento de separar a base em variáveis preditoras ou independentes e variável resposta ou dependente (também chamada de variável alvo). Essa tarefa foi feita pelo código da Figura 52, onde da base original retirou-se a coluna “ACIMA_MEDIA_FREQUENCIA” (variável resposta) gerando assim uma base que foi chamada de “X”, contendo somente as variáveis independentes. Depois atribuiu-se a coluna da variável resposta a uma base chamada de “y”. Lembrando que o parâmetro “axis” ao receber o valor 1 indica que está sendo excluída uma coluna.

Figura 51 – Base de dados com a coluna de dados categóricos convertida em numéricos

	HORA	PRECIPITACAO	TEMPERATURA	ACIMA_MEDIA_FREQUENCIA	TIPO_DIA_DIA_UTIL	TIPO_DIA_DOMINGO	TIPO_DIA_FERIADO	TIPO_DIA_SABADO
0	0	0.0	15.8	0	0	1	0	0
1	1	0.0	15.1	0	0	1	0	0
2	2	0.0	14.7	0	0	1	0	0
3	3	0.0	14.4	0	0	1	0	0
4	4	0.0	14.4	0	0	1	0	0
...
1261	13	0.0	18.7	1	1	0	0	0
1262	15	0.0	22.6	1	1	0	0	0
1263	17	0.0	23.6	1	1	0	0	0
1264	19	0.0	23.2	1	1	0	0	0
1265	21	0.0	17.3	0	1	0	0	0

1266 rows x 8 columns

Figura 52 – Separação dos dados em variáveis preditoras e variável resposta

```
X = base_ml.drop('ACIMA_MEDIA_FREQUENCIA', axis=1)
y = base_ml['ACIMA_MEDIA_FREQUENCIA']
```

5.1.1. Balanceado a Base (Oversampling)

A variável alvo “ACIMA_MEDIA_FREQUENCIA” encontra-se levemente desbalanceada, contando com 698 registros onde a frequência de cruzamento de motocicletas no ponto de estudo foram abaixo da média e 568 acima da média. Ao utilizar a estratégia de *oversampling* (SMOTE) foram geradas amostras sintéticas da variável alvo minoritária, o que no final balanceou a base, contando com 698 registros de cada classe.

Ao visualizar o gráfico do tipo *countplot* mostrado na figura 43 é possível ver a variável resposta “ACIMA_MEDIA_FREQUENCIA” desbalanceada. Após a execução do código exibido na Figura 53, onde utilizou-se a função “SMOTE” do pacote “imblearn”, foi possível balancear os dados.

Figura 53 – Código para o balanceamento da base

```
smote = SMOTE(sampling_strategy='minority')
X, y = smote.fit_resample(X, y)
```

5.1.2. Escalonando os Dados

Por conter dados com proporções diferentes, existe a possibilidade dos algoritmos considerarem valores maiores como mais influentes na resposta que valores menores. Por isso, optou-se por escalonar os dados, trazendo todos eles para a mesma proporção. Para realizar o escalonamento dos dados, utilizou-se a função “*fit_transform*” da classe *MinMaxScaler* do pacote *sklearn*. O código da Figura 54 exibe tal processamento.

Figura 54 – Código para o escalonamento dos dados

```
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
X
array([[0.          , 0.          , 0.54954955, ..., 1.          , 0.          ,
        0.          ],
       [0.04347826, 0.          , 0.51801802, ..., 1.          , 0.          ,
        0.          ],
       [0.08695652, 0.          , 0.5          , ..., 1.          , 0.          ,
        0.          ],
       ...,
       [0.39130435, 0.          , 0.30267785, ..., 0.          , 0.          ,
        0.          ],
       [0.73913043, 0.          , 0.7537327 , ..., 0.          , 0.          ,
        0.          ],
       [0.69565217, 0.          , 0.77209989, ..., 0.          , 0.          ,
        0.          ]])
```

Ao final deste processo, obteve-se o modelo com a configuração 1 para a realização dos treinamentos e avaliações dos algoritmos de *Machine Learning* selecionados (Random Forest, KNN e SVM).

5.1.3. Executando o Majority Learner para Ter um Nível de Acurácia de Referência

Com o objetivo de se obter um limite mínimo aceitável de desempenho dos algoritmos, foi feito um treinamento utilizando o algoritmo *majority learner* na base desbalanceada, fazendo o aprendizado pela maioria.

Utilizando a métrica de avaliação de acurácia, obteve-se 55% de taxa de acertos, indicando assim que qualquer outro algoritmo que ficar abaixo dessa taxa de acertos é melhor classificar a variável alvo como a de maior ocorrência.

Figura 55 – Código para avaliar o desempenho do algoritmo majority learner no modelo

```
base_majority = Orange.data.Table('dados/fluxo_moto/base_fluxo_motos_tratada_regras.csv')

base_majority.domain

[FERIADO, DOMINGO, SABADO, DIA_UTIL, PRECIPITACAO, TEMPERATURA, HORA | ACIMA_MEDIA_FREQUENCIA]

majority = Orange.classification.MajorityLearner()

previsoes = Orange.evaluation.testing.TestOnTestData(base_majority, base_majority, [majority])

Orange.evaluation.CA(previsoes)

array([0.55432373])
```

5.1.4. Tuning dos Parâmetros

O *tuning* de parâmetros tem como objetivo avaliar qual configuração de parâmetros utilizada no algoritmo de *Machine Learning* tem melhor desempenho. Para realizar o *tuning*, foi utilizada a classe *GridSearchCV* do pacote *sklearn* do *Python*.

5.1.4.1. Tuning de Parâmetros Para o Algoritmo Random Forest

Para a aplicação do algoritmo *random forest* foi utilizada a classe *RandomForestClassifier* do pacote *sklearn* do *Python* onde os parâmetros alterados para encontrar a melhor configuração foram: *criterion*, *n_estimator*, *min_samples_split* e *min_samples_leaf*.

A fim de explicar o significado de cada parâmetro utilizado no *tuning* do algoritmo *random forest* foi criada a Tabela 4.

Tabela 4 – Descrição dos parâmetros utilizados para o algoritmo random forest

Parâmetro	Descrição	Melhor Parâmetro
criterion	Função que mede a qualidade da divisão dos dados	gini
n_estimator	Número de árvores de decisão que serão utilizadas no algoritmo	36
min_samples_split	Número mínimo de amostras para dividir um nó interno	5
min_samples_leaf	Número mínimo de amostras necessárias em um nó folha	6

Ao utilizar o código presente na Figura 56, foi possível realizar o *tunning* de parâmetros para o algoritmo *random forest* onde com os parâmetros apontados pelo *tunning* é possível obter uma acurácia de 87%.

Figura 56 – Código para o tuning do algoritmo random forest

```
parametros = {'criterion': ['gini', 'entropy'],
              'n_estimators': range(35, 45),
              'min_samples_split': range(2, 10),
              'min_samples_leaf': range(2, 10)}

grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)

{'criterion': 'gini', 'min_samples_leaf': 6, 'min_samples_split': 5, 'n_estimators': 36}
0.8717665130568356
```

5.1.4.2. Tuning de Parâmetros Para o Algoritmo KNN

O algoritmo KNN (K-nearest neighbors ou K vizinhos mais próximos) foi utilizado neste trabalho através da classe *KNeighborsClassifier* do pacote *sklearn*, onde os parâmetros utilizados foram: *n_neighbors*, *p*, *algorithm* e *metric*. Esses parâmetros estão descritos na Tabela 5, apontando também os melhores valores apontados no *tunning*.

Tabela 5 – Descrição dos parâmetros utilizados para o algoritmo KNN

Parâmetro	Descrição	Melhor Parâmetro
n_neighbors	Número de vizinhos usados por padrão	14
p	Parâmetro de potências para a métrica Minkowski, quando 1 é equivalente a usar a distância de manhattan quando 2 é equivalente à distância euclidiana	1
algorithm	Algoritmo utilizado para calcular o vizinho mais próximo	auto
metric	Métrica utilizada para o cálculo da distância	cityblock

Na Figura 57 está o código utilizado para realizar o *tunning* dos parâmetros do algoritmo KNN, onde o código aponta os melhores parâmetros seguido da respectiva acurácia de aproximadamente 85%.

Figura 57 – Código para o tuning do algoritmo KNN

```

parametros = {'n_neighbors': range(10, 15),
              'p': [1, 2],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'metric': ['cityblock', 'minkowski', 'euclidean']}
grid_search = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)

{'algorithm': 'auto', 'metric': 'cityblock', 'n_neighbors': 14, 'p': 1}
0.8531182795698925

```

5.1.4.3. Tuning de Parâmetros Para o Algoritmo SVM

Com a finalidade de aplicar o algoritmo SVM (Support Vector Machine) utilizou-se a classe SVC (Support Vector Classification) do pacote *sklearn* da linguagem *Python*.

Como parâmetros foram utilizados *tol*, *random_state*, *C* e *kernel*, especificados e apontados os melhores valores na Tabela 5, sendo possível ver também os melhores parâmetros para o algoritmo SVM.

Tabela 6 – Descrição dos parâmetros utilizados para o algoritmo SVM

Parâmetro	Descrição	Melhor Parâmetro
tol	Tolerância do critério de parada	0.1
Random_state	Controla a geração de números pseudoaleatórios para embaralhar os dados para estimativas de probabilidade	1
C	Parâmetro de regularização	3.0
kernel	Especifica qual kernel que será utilizado no algoritmo	rbf

Segundo o processo de *tuning* mostrado na Figura 58, ao utilizar os parâmetros apontados é possível obter uma acurácia de aproximadamente 86%.

Figura 58 – Código para o tuning do algoritmo SVM

```

parametros = {'tol': [0.1, 0.01, 0.001, 0.0001, 0.00001],
              'random_state': [1,2,3,4,5,6],
              'C': [2.5,3.0,3.5,4.0,4.5,5.0],
              'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}

grid_search = GridSearchCV(estimator=SVC(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)

{'C': 3.0, 'kernel': 'rbf', 'random_state': 1, 'tol': 0.1}
0.8602995391705068

```

5.1.5. Avaliando o Modelo com Cross-validation

Para realizar a validação cruzada dos algoritmos aplicados ao modelo, utilizou-se a classe *KFold* do pacote *sklearn* do *Python*. Onde optou-se por utilizar um número de divisões igual a 10 de maneira randômica.

O processo de execução da validação cruzada foi repetido por 30 vezes onde o resultado de cada processo foi guardado em um *array* que por fim foi transformado em um *Pandas DataFrame* a fim de facilitar o cálculo dos seus parâmetros estatísticos.

5.1.5.1. Cross-validation do Algoritmo Random Forest

Para realizar o *cross-validation* do algoritmo *random-forest* foi construído o código mostrado na Figura 59. Na sequência foi gerado um *dataset* com os dados estatísticos dos resultados, mostrado na Figura 60.

Na figura 60 é possível ver que a média da acurácia é de aproximadamente 85%, podendo considerar assim que para o modelo da configuração de dados 1 utilizando o algoritmo *random-forest*, obtém-se em média uma acurácia de 85% aproximadamente.

Figura 59 – Código de cross-validation do algoritmo random forest para o modelo de configuração 1

```
resultados_random_forest = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    random_forest = RandomForestClassifier(n_estimators=36, criterion='gini', min_samples_leaf= 6, min_samples_split= 5, random_s
    pontos = cross_val_score(random_forest, X, y, cv = kfold, scoring='accuracy')
    resultados_random_forest.append(pontos.mean())
```

Figura 60 – Parâmetros estatísticos da acurácia oriunda do cross-validation do algoritmo random forest para o modelo de configuração 1

RESULTADOS_RANDOM_FOREST	
count	30.000000
mean	0.869041
std	0.003462
min	0.863165
25%	0.866928
50%	0.868546
75%	0.871028
max	0.876048

5.1.5.2. Cross-validation do Algoritmo KNN

Da mesma forma que foi desenvolvido o código para realizar o *cross-validation* do modelo submetido ao algoritmo *random forest*, foi feito para o algoritmo *KNN*. O código da Figura 61 mostra o *cross-validation* sendo feito para o modelo de configuração 1 submetido ao algoritmo *KNN*.

Figura 61 – Código de cross-validation do algoritmo KNN para o modelo de configuração 1

```
resultados_knn = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    knn = KNeighborsClassifier(algorithm='auto', n_neighbors=14, metric='cityblock', p = 1)
    pontos = cross_val_score(knn, X, y, cv = kfold, scoring='accuracy')
    resultados_knn.append(pontos.mean())
```

Por fim, o resultado da avaliação foi colocado em um *Pandas DataFrame*, presente na Figura 62, onde é possível ver alguns parâmetros estatísticos avaliados durante o *cross-validation*.

Figura 62 – Parâmetros estatísticos da acurácia oriunda do cross-validation do algoritmo KNN para o modelo de configuração 1

RESULTADOS_KNN	
count	30.000000
mean	0.868331
std	0.003355
min	0.859527
25%	0.865690
50%	0.868564
75%	0.870319
max	0.874666

5.1.5.3. Cross-validation do Algoritmo SVM

Assim como foi feita a avaliação dos algoritmos *random forest* e *KNN*, também foi feita para o algoritmo *SVM*. A codificação mostrada na Figura 63 é responsável pela geração dos resultados estatísticos apresentados o *Pandas Dataframe* da Figura 64.

Figura 63 – Código de cross-validation do algoritmo SVM para o modelo de configuração 1

```
resultados_svm = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    svm = SVC(kernel='rbf', random_state=1, C = 3.0, tol = 0.01)
    pontos = cross_val_score(svm, X, y, cv = kfold, scoring='accuracy')
    resultados_svm.append(pontos.mean())
```

Figura 64 – Parâmetros estatísticos da acurácia oriunda do cross-validation do algoritmo SVM para o modelo de configuração 1

RESULTADOS_SVM	
count	30.000000
mean	0.867332
std	0.002398
min	0.862477
25%	0.865445
50%	0.867518
75%	0.868959
max	0.871115

5.1.6. Agrupando tabelas de resultados

Após ser feito o *cross-validation* dos resultados dos três algoritmos utilizados para o modelo de configuração 1, os resultados foram agrupados em um único *dataframe* com o objetivo de facilitar a avaliação dos treinamentos. Para unir esses *dataframes*, foi utilizado o código da Figura 65 que gerou o *dataframe* exibido na Figura 66.

Figura 65 – Código para agrupar os resultados do cross-validation em um único dataframe

```
resultados = pd.DataFrame({'RANDOM_FOREST': resultados_random_forest, 'KNN': resultados_knn, 'SVM': resultados_svm})
resultados.describe()
```

Figura 66 – Dataframe contendo os resultados do cross-validation para os dados da configuração 1

	RANDOM_FOREST	KNN	SVM
count	30.000000	30.000000	30.000000
mean	0.869041	0.868331	0.867332
std	0.003462	0.003355	0.002398
min	0.863165	0.859527	0.862477
25%	0.866928	0.865690	0.865445
50%	0.868546	0.868564	0.867518
75%	0.871028	0.870319	0.868959
max	0.876048	0.874666	0.871115

5.2. Construindo Modelos de Machine Learning com Dados de Fluxo de Motocicletas e Dados Climáticos (Configuração 2)

Como este trabalho propôs inicialmente construir três modelos de *Machine-Learning* e compara-los a fim de perceber quais dados influenciam na frequência de cruzamento de motocicletas em um ponto da cidade de Belo Horizonte. Todos os passos feitos na seção 5.1 para chegar no resultado da acurácia do modelo construído para os três algoritmos a partir da configuração 1 de dados, também foram necessários fazer para a configuração 2. Esses passos são respectivamente:

- Ler a base de dados;
- Balancear com *oversampling*;
- Converter a variável categórica TIPO_DIA em colunas numéricas;
- Escalonar os dados;
- Fazer o *tunning* dos parâmetros para os três algoritmos utilizados;
- Avaliar os três algoritmos com *cross-validation*;
- Exibir o resultado em um único *dataframe*.

Para realizar todo o processo descrito acima de uma forma menos explicativa, como feito na seção 5.1, foi construindo um código, Figura 67, que executou todos os passos para a configuração 2 dos dados.

Figura 67 – Código para avaliar a configuração 2

```

#Recupera e trata colunas da base
base_ml = pd.read_csv('dados/fluxo_moto/base_fluxo_motos_tratada.csv')
base_ml = base_ml.drop(['TIPO_DIA', 'HORA'], axis=1)

#Separa a classe alvo
X = base_ml.drop('ACIMA_MEDIA_FREQUENCIA', axis=1)
y = base_ml['ACIMA_MEDIA_FREQUENCIA']

#Escalona os dados
scaler = StandardScaler()
X = scaler.fit_transform(X)

#Faz o tuning dos parametros
# -> Random Forest
parametros = {'criterion': ['gini', 'entropy'],
              'n_estimators': range(38, 41),
              'min_samples_split': range(6, 10),
              'min_samples_leaf': range(6, 10)}
grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print("=====\n Tuning Random Forest\n=====")
print(melhores_parametros)
print(melhor_resultado)
# -> Random KNN
parametros = {'n_neighbors': [35, 40, 45],
              'p': [1, 2],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'metric': ['cityblock', 'minkowski', 'euclidean']}
grid_search = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print("=====\n Tuning KNN\n=====")
print(melhores_parametros)
print(melhor_resultado)
# -> Random SVM
parametros = {'tol': [0.1, 0.01, 0.001, 0.0001, 0.00001],
              'random_state': [1, 2, 3, 4, 5, 6],
              'C': [0.5, 1.0, 1.5, 2.0],
              'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}
grid_search = GridSearchCV(estimator=SVC(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print("=====\n Tuning SVM\n=====")
print(melhores_parametros)
print(melhor_resultado)

# Avaliacao dos algoritmos com cross-validation
resultados_random_forest = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    random_forest = RandomForestClassifier(n_estimators=38, criterion='entropy', min_samples_leaf=6, min_samples_split=7, random_state=i)
    pontos = cross_val_score(random_forest, X, y, cv=kfold, scoring='accuracy')
    resultados_random_forest.append(pontos.mean())

resultados_knn = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    knn = KNeighborsClassifier(algorithm='ball_tree', n_neighbors=40, metric='cityblock', p=1)
    pontos = cross_val_score(knn, X, y, cv=kfold, scoring='accuracy')
    resultados_knn.append(pontos.mean())

resultados_svm = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    svm = SVC(kernel='rbf', random_state=1, C=1.5, tol=0.1)
    pontos = cross_val_score(svm, X, y, cv=kfold, scoring='accuracy')
    resultados_svm.append(pontos.mean())
resultados_dados_climaticos = pd.DataFrame({'RANDOM_FOREST': resultados_random_forest, 'KNN': resultados_knn, 'SVM': resultados_svm})
resultados_dados_climaticos.describe()

```

O código da Figura 67 gerou o *dataframe* com os resultados do *cross-validation* do modelo construído a partir da configuração 2 dos dados, que pode ser visto na Figura 68.

Figura 68 – Dataframe contendo os resultados do cross-validation para os dados da configuração 2

	RANDOM_FOREST	KNN	SVM
count	30.000000	30.000000	30.000000
mean	0.712039	0.715240	0.722995
std	0.004573	0.003634	0.002298
min	0.702237	0.710049	0.718023
25%	0.709332	0.712105	0.721899
50%	0.710930	0.714867	0.723169
75%	0.716209	0.717787	0.724358
max	0.718841	0.721910	0.727503

5.3. Construindo Modelos de Machine Learning com Dados de Fluxo de Motocicletas e Dados de Tipo de Dia e Hora (Configuração 3)

Não diferente do treinamento e avaliação das configurações 1 e 2, o mesmo foi feito para a configuração 3 dos dados. O código construído foi bastante semelhante ao apresentado na figura 67, porém no lugar de excluir os dados de tipo de dia e hora (logo na segunda linha de código) foi excluído os dados de precipitação e temperatura, assim restando apenas as colunas “TIPO_DIA” e “HORA”, sendo possível fazer novamente todo o processo para a configuração 3 dos dados. Valendo lembrar os parâmetros de cada algoritmo foram utilizados de acordo com o *tunning* realizado para cada configuração de dados.

A Figura 69 mostra o resultado obtido do treinamento e *cross-validation* do modelo de configuração 3 para cada um dos três algoritmos utilizados neste trabalho.

Figura 69 – Dataframe contendo os resultados do cross-validation para os dados da configuração 3

	RANDOM_FOREST	KNN	SVM
count	30.000000	30.000000	30.000000
mean	0.858604	0.861141	0.846917
std	0.004165	0.003209	0.001874
min	0.850662	0.853056	0.842745
25%	0.856241	0.859389	0.845346
50%	0.858561	0.860602	0.846772
75%	0.860794	0.863150	0.848381
max	0.866504	0.866529	0.850762

5.4. Matriz de Confusão

Como a configuração 1 dos dados apresentou a melhor acurácia média para os três algoritmos utilizados para o modelo de *Machine Learning*, resolveu-se construir a matriz de confusão para os resultados de acertos e erros dos três algoritmos.

As Figuras 70, 71 e 72 mostram a matriz de confusão para o algoritmo *Random Forest*, KNN e SVM respectivamente.

Figura 70 – Matriz de confusão para o modelo de configuração 1 submetido ao algoritmo random forest

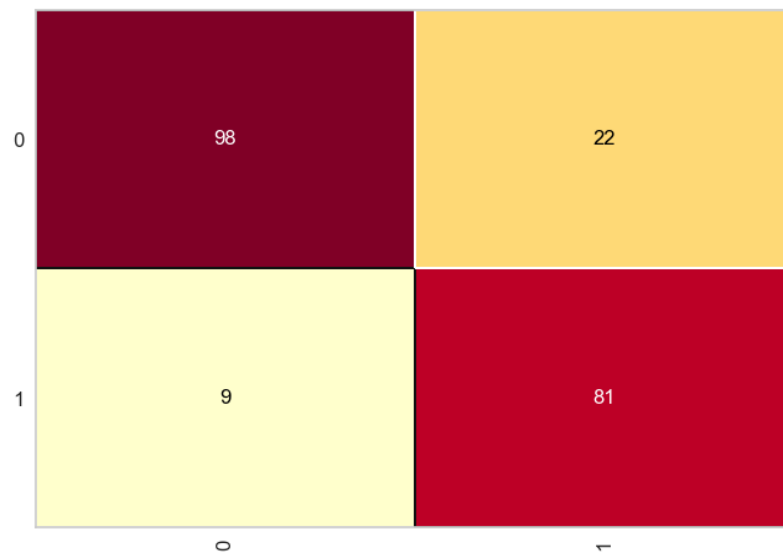


Figura 71 – Matriz de confusão para o modelo de configuração 1 submetido ao algoritmo KNN

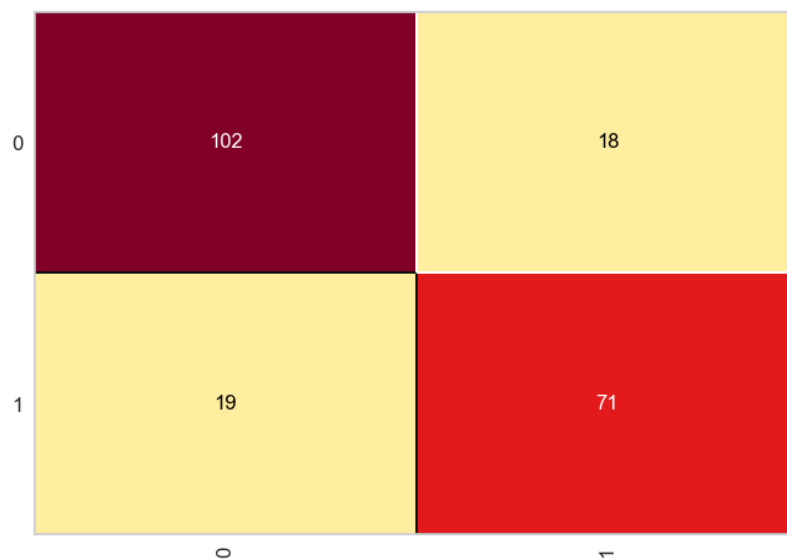
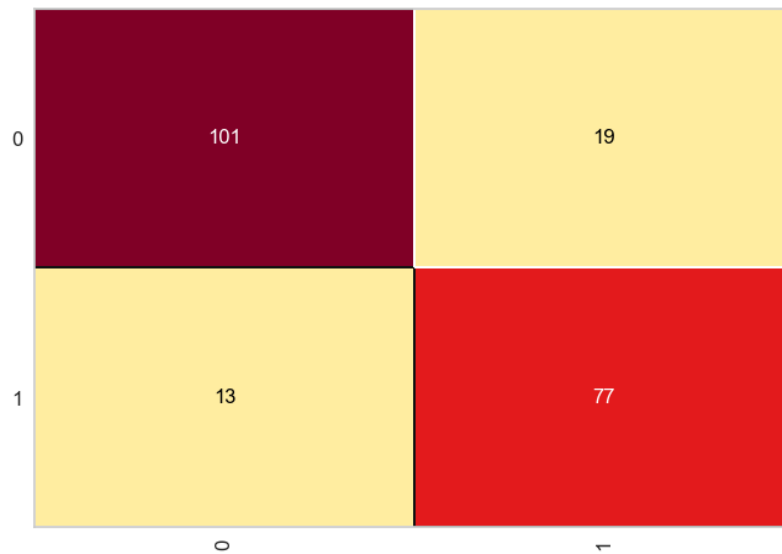


Figura 72 – Matriz de confusão para o modelo de configuração 1 submetido ao algoritmo SVM



6. Interpretação e Comunicação dos Resultados

Durante a análise e exploração dos dados, seção 4, pode ser observado que a base de dados utilizada para o estudo estava levemente desbalanceada quanto a classe alvo, podendo ser visto na Figura 34, porém, durante a construção dos modelos de *Machine Learning* esse dado foi balanceado com a técnica de *oversampling*. Outro dado desbalanceado foi a quantidade de amostras quanto ao tipo de dia, que predominou majoritariamente os dias úteis, o que já era de se esperar, pois durante os dois meses de observação, a maioria dos dias foram úteis.

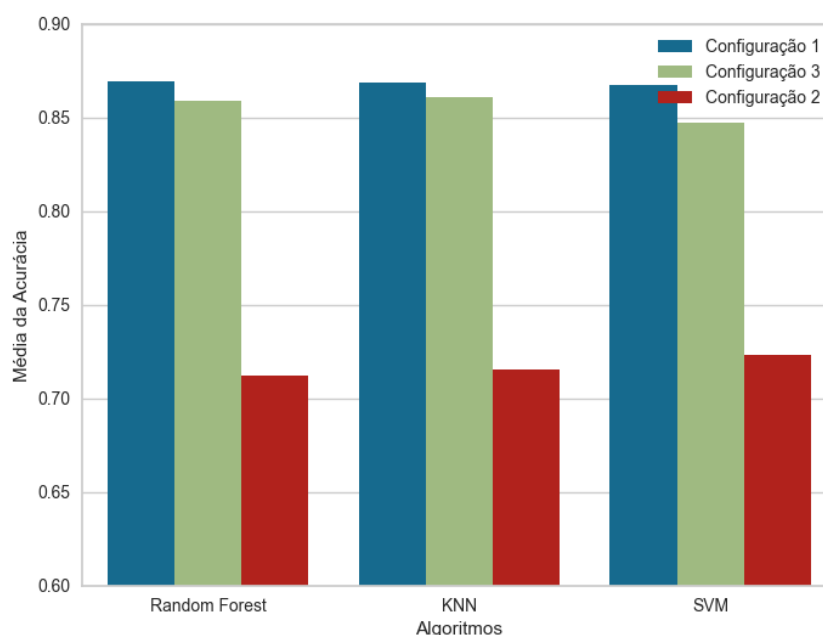
A fim de entender como os conjuntos de dados influenciam em uma frequência acima ou abaixo da média, foram traçados três gráficos, presentes nas figuras 41, 42 e 43, sendo respectivamente o relacionamento de todas as variáveis por todas as variáveis do *dataset* original, o relacionamento das variáveis climáticas (precipitação e temperatura) e por fim o relacionamento dos dados de tipo de dia e horário. O que pôde ser percebido nesse processo é que as variáveis climáticas sozinhas são mais confusas para prever se o cruzamento de motocicletas no ponto de estudo está acima ou abaixo da média, porém ainda possuem informações sobre a variável alvo. Já na última figura, onde é possível ver o relacionamento das variáveis do tipo de dia e horário, percebe-se que essas possuem informações bem claras de suas influências na variável alvo, levando ao entendimento que podem contribuir mais na previsão da frequência de cruzamento de motocicletas no ponto estudado.

Para fazer outra análise nos dados e perceber padrões, além dos gráficos das figuras 41, 42 e 43, foram construídos mapas de calor para as variáveis climáticas, de horário e tipo de dia, esses gráficos podem ser vistos nas Figuras 45, 47 e 49. A figura 45 mostra o tipo de

dia por horário e como contraste do mapa de calor a quantidade de ocorrências de frequências acima da média. Nesse gráfico percebe-se claramente a influência do horário em ocorrências de frequências acima da média, sendo mais uma contribuição para a conclusão que o tipo de dia e horário são bons previsores de altas ocorrências de frequências acima da média. Nas figuras 47 e 49 estão os mapas de calor onde o contraste também é a quantidade de ocorrências de frequências acima da média e as variáveis confrontadas são as climáticas. Na figura 47 o gráfico de calor não leva em conta o volume de precipitação e sim somente se houve ou não precipitação, e independente se houve ou não precipitação, percebe-se o aumento e diminuição da frequência de cruzamentos nos mesmos horários, claro, existem mais amostras de dias que não chovem que dias que chovem, fazendo as ocorrências para os dias sem chuvas serem maiores. Quando analisado a temperatura em função do horário no gráfico de calor da figura 49, percebe-se um contraste bem visível quanto à variável alvo, mas pode-se interpretar que a temperatura, de alguma maneira carrega a informação contida no horário, já que durante o dia tem-se temperaturas maiores e nas madrugadas menores.

Durante a análise e exploração dos dados pode-se perceber que os dados de horário de tipo de dia são os que mais influenciam em frequências altas ou baixas de cruzamento de motocicletas, portanto foram propostos 3 modelos de *Machine Learning* com diferentes configurações de dados, a fim de verificar qual dos três modelos tem melhor desempenho, entendendo assim quais conjuntos de dados (presentes em cada modelo) influencia mais ou menos em ocorrências de frequências acima ou abaixo da média. A acurácia de cada modelo submetido a cada algoritmo de *Machine Learning* foi avaliada utilizando a técnica de *cross validation* e exibidas nos *dataframes* das figuras 66, 68 e 69. Todos esses dados de acurácias foram compilados e mostrados no gráfico da Figura 73.

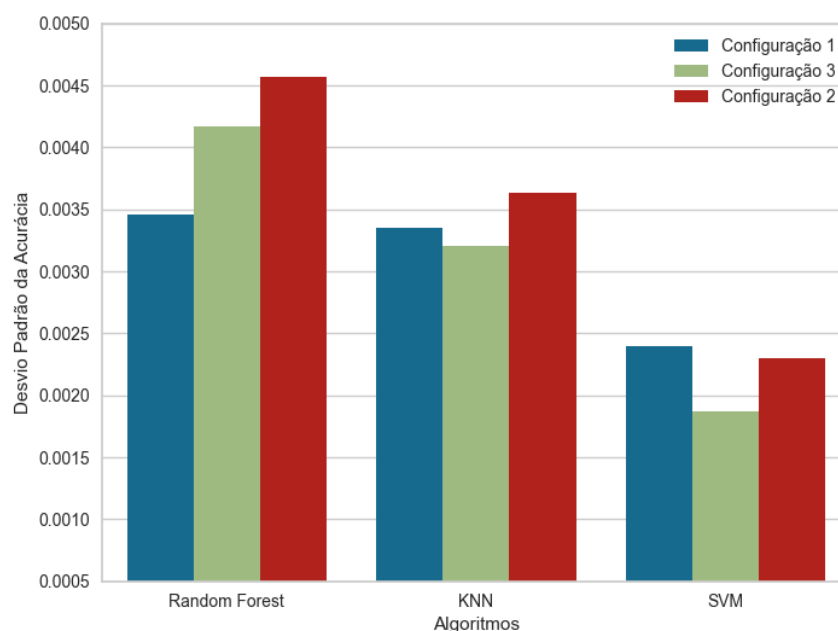
Figura 73 – Média da Acurácia por Algoritmo e Configuração de Dados do Modelo



Na figura 73 consegue-se perceber que a configuração de dados que teve o pior desempenho foi a configuração 2, ou seja, os dados climáticos sozinhos são os que menos conseguem ser utilizados para prever frequências altas ou baixas de cruzamento de motocicletas no ponto de estudo. Já a configuração 1 e 3 foram as que tiveram melhores desempenhos, valendo ressaltar que o fato de conter os dados de clima no modelo de classificação de intensidade de cruzamentos de motocicletas se faz pouco relevante para as previsões. Mas o que explica os dados de clima ainda se mostrarem melhor que a acurácia do *majority learner* de 55.4%? Provavelmente as informações de temperatura carregam em si as informações do horário, já que foi possível visualizar graficamente que a temperatura varia conforme o horário, assim, dependendo da temperatura, pode-se estimar se está em um horário de pico de fluxo ou não. Logo pode-se tirar como conhecimento dessa análise que o clima, ao contrário do que se esperava, tem baixa influência na previsão da variável alvo, enquanto os tipos de dias e horários são os fatores que mais influenciam em uma alta ou baixa frequência de cruzamento de motocicletas no ponto estudado. Percebe-se também que o algoritmo *Random Forest* tem melhor desempenho para a configuração 1, contando com 86.9% de acurácia, enquanto o algoritmo KNN tem melhor desempenho na configuração 3, contando com 86.1% de acurácia. Por fim a configuração 2 foi a que teve o pior resultado, apesar de ficar acima dos 55,4% do *majority learner*, não passou dos 73% de acurácia.

Da mesma maneira que foi feita uma compilação gráfica para a média das acurácias, também se construiu uma compilação para os desvios padrão. Lembrando que no mundo real a base de treinamento evolui constantemente, ao analisar a compilação do desvio padrão é possível perceber o quão estável é um modelo quando submetido aos algoritmos de *Machine Learning* em uma situação real. Para tal análise, foi gerado o gráfico da Figura 74.

Figura 74 – Desvio Padrão da Acurácia por Algoritmo e Configuração de Dados do Modelo



De acordo com a figura 74, percebe-se que o algoritmo que menos varia sua acurácia quando submetido à várias configurações de bases de teste e treinamento é o SVM, porém este algoritmo foi o que apresentou menor acurácia média também. Quanto às configurações de dados, percebe-se que a configuração 1 torna os algoritmos com acurácias mais estáveis, levando a crer que os dados climáticos apesar de não contribuir muito com a acurácia, contribui tornando os modelos mais estáveis.

Além de tirar informações das médias e desvios padrão das acurácias de cada modelo e algoritmo, foram construídas matrizes de confusão para cada algoritmo escolhido utilizando o modelo de configuração de dados 1, pois foi o que se mostrou o melhor modelo. Essas matrizes de confusão foram expostas nas figuras 70, 71 e 72.

Na figura 70, onde está apresenta a matriz de confusão para o modelo *Random Forest*, estão presentes 98 verdadeiros negativos, 81 verdadeiros positivos, 22 falsos positivos e 9 falsos negativos. Isso significa que 98 amostras abaixo da média de frequência foram classificadas corretamente, 81 amostras acima da média foram classificadas corretamente, 22 amostras que estavam abaixo da média foram classificadas como acima da média e 9 amostras que estavam acima da média foram classificadas como abaixo da média. Pensando no problema, é pior classificar uma amostra acima da média como abaixo que o contrário, já que uma quantidade de cruzamentos de motocicletas acima da média é mais problemática para o trânsito que uma quantidade abaixo da média. Em outras palavras, deve-se optar pelo algoritmo que tem menor quantidade de falsos negativos.

A figura 71 mostra que o algoritmo KNN tende a classificar mais amostras acima da média como abaixo da média que o contrário, tendo mais falsos positivos que falsos negativos, logo apresenta-se como um algoritmo que pode prever erradamente mais um trânsito bom enquanto o mesmo está ruim. Já na figura 72 mostra o algoritmo SVM classificando mais falsos negativos que falso positivos, assim como o *Random Forest*.

Perante as matrizes de confusão, o algoritmo *Random Forest* e SVM se mostraram melhores. Já quanto à acurácia o algoritmo *Random Forest* se mostrou melhor que os demais quando aplicado no modelo de *Machine Learning* com configuração 1 de dados. Assim percebe-se como melhor opção a utilização da configuração 1 de dados com o algoritmo *Random Forest* para a previsão de ocorrência de frequências acima ou abaixo da média.

7. Conclusão

Dos estudos apresentados foi possível concluir que ao contrário do que se imaginava, condições climáticas de temperatura e precipitação pouco influenciam ou nada influenciam em aumentar ou diminuir fluxos de motocicletas no ponto estudado da Av. Afonso Pena na esquina com a rua Maranhão em Belo Horizonte. Enquanto o tipo de dia (se é um dia útil ou

um final de semana ou feriado) e também o horário são as variáveis que mais contribuem para prever se a frequência de cruzamento de motocicletas no ponto de estudo vai ser acima ou abaixo da média. Isso ficou explícito tanto com a análise gráfica durante a exploração dos dados quanto com o desempenho dos algoritmos de *Machine Learning* quando submetido a diferentes modelos.

No estudo também pode-se concluir que o melhor algoritmo para avaliar se a frequência de cruzamento de motocicletas no ponto estudado vai ser acima ou abaixo da média foi o *Random Forest*, apresentando menor erro quando prevê uma frequência maior que a média enquanto na verdade é menor e também apresentou a maior acurácia para o modelo de configuração 1.

8. Links

Link para o vídeo: https://www.youtube.com/watch?v=iOb88eH_Pl0

Link para o repositório: https://github.com/jrdutra/tcc_ciencia_de_dados_puc_minas

APÊNDICE

Programação/Scripts

```

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import requests
import json
from datetime import date, datetime
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, classification_report
import Orange
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
from scipy import stats
from sklearn.model_selection import cross_val_score, KFold
from yellowbrick.classifier import ConfusionMatrix

mes_maio = '05'
dias_maio = ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15',
'16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31']
mes_junho = '06'
dias_junho = ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15',
'16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30']
horas_dia = ['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14',
'15', '16', '17', '18', '19', '20', '21', '22', '23']

df_final = pd.DataFrame(columns=['ID EQP', 'DATA HORA', 'MILESEGUNDO', 'FAIXA', 'ID DE ENDEREÇO', 'VE-
LOCIDADE DA VIA', 'VELOCIDADE AFERIDA', 'CLASSIFICAÇÃO', 'TAMANHO', 'NUMERO DE SÉRIE', 'LATITUDE',
'LONGITUDE', 'ENDEREÇO', 'SENTIDO'])
for dia in dias_maio:
    for hora in horas_dia:
        caminho_arquivo = f'dados/fluxo_moto/2022{mes_maio}{dia}/2022{mes_maio}{dia}_{hora}.json'
        with open(caminho_arquivo, 'r', encoding="utf8") as f:
            str_json = f.read()
            dicionario_dados = json.loads(str_json)
            df_lido = pd.json_normalize(dicionario_dados)
            df_lido = df_lido.loc[df_lido['CLASSIFICAÇÃO'] == 'MOTO']
            df_lido['LATITUDE'] = df_lido['LATITUDE'].str.replace(' ', '').astype(float)
            df_lido['LONGITUDE'] = df_lido['LONGITUDE'].str.replace(' ', '').astype(float)
            df_lido = df_lido.loc[df_lido['LATITUDE'] == -19.9377]
            df_lido = df_lido.loc[df_lido['LONGITUDE'] == -43.92711]
            df_final = pd.concat([df_final, df_lido])
for dia in dias_junho:
    for hora in horas_dia:
        caminho_arquivo = f'dados/fluxo_moto/2022{mes_junho}{dia}/2022{mes_junho}{dia}_{hora}.json'
        with open(caminho_arquivo, 'r', encoding="utf8") as f:
            str_json = f.read()
            dicionario_dados = json.loads(str_json)
            df_lido = pd.json_normalize(dicionario_dados)
            df_lido = df_lido.loc[df_lido['CLASSIFICAÇÃO'] == 'MOTO']
            df_lido['LATITUDE'] = df_lido['LATITUDE'].str.replace(' ', '').astype(float)
            df_lido['LONGITUDE'] = df_lido['LONGITUDE'].str.replace(' ', '').astype(float)
            df_lido = df_lido.loc[df_lido['LATITUDE'] == -19.9377]
            df_lido = df_lido.loc[df_lido['LONGITUDE'] == -43.92711]
            df_final = pd.concat([df_final, df_lido])
df_final.rename(columns={'ID EQP': 'ID_EQP', 'DATA HORA': 'DATA_HORA', 'ID DE ENDEREÇO': 'ID_DE_ENDERE-
ÇO', 'VELOCIDADE DA VIA': 'VELOCIDADE_DA_VIA', 'VELOCIDADE AFERIDA': 'VELOCIDADE_AFERIDA', 'CLASSIFICA-
ÇÃO': 'CLASSIFICACAO', 'NUMERO DE SÉRIE': 'NUMERO_DE_SERIE', 'ENDEREÇO': 'ENDERECO'}, inplace = True)
df_final.to_csv("dados/fluxo_moto/base_fluxo_motos.csv", index=False)

```

```

df_final.head(5)

def avalia_tipo_dia(data_atual):
    tipo_dia = "DIA_UTIL"
    if data_atual.weekday() == 5:
        tipo_dia = 'SABADO'
    elif data_atual.weekday() == 6:
        tipo_dia = 'DOMINGO'
    elif format(data_atual, "%Y-%m-%d") == '2022-06-16':
        tipo_dia = 'FERIADO'
    return tipo_dia

datas = pd.date_range(start='2022-05-01 00:00:00', end='2022-06-30 23:00:00', periods = 1464)
base_dados = pd.DataFrame(datas, columns=['DATA'])
base_dados['DATA_HORA_COMPARADOR'] = base_dados['DATA'].apply(lambda data_atual: format(data_atual, "%Y-%m-%dT%H:00:00"))
base_dados['DATA_COMPARADOR'] = base_dados['DATA'].apply(lambda data_atual: format(data_atual, "%Y-%m-%d"))
base_dados['TIPO_DIA'] = base_dados['DATA'].apply(lambda data_atual: avalia_tipo_dia(data_atual))

base_dados.to_csv("dados/fluxo_moto/base_dados.csv", index=False)
base_dados

datas = pd.date_range(start='2022-05-01 00:00:00', end='2022-06-30 23:00:00', periods = 1464)
base_clima = pd.DataFrame(datas, columns=['DATA'])
base_clima['DATA_HORA_COMPARADOR'] = base_clima['DATA'].apply(lambda data_atual: format(data_atual, "%Y-%m-%dT%H:00:00"))
base_clima

def retorna_dados_metereologicos(data_hora, latitude, longitude):
    dataDate = datetime.strptime(data_hora, '%Y-%m-%dT%H:%M:%S')
    dataStrRequest = dataDate.strftime('%Y-%m-%d')

    #chamada
    url = "https://archive-api.open-meteo.com/v1/era5?latitude="+str(latitude)+"&longitude="+str(longitude)+"&start_date="+str(dataStrRequest)+"&end_date="+str(dataStrRequest)+"&hourly=temperature_2m,precipitation"
    response = requests.get(url)
    jsonObj = response.json()

    #pegando atributos
    dataStrComparador = dataDate.strftime('%Y-%m-%dT%H:00')
    indiceDataHora = jsonObj['hourly']['time'].index(dataStrComparador)

    temperature_2m = jsonObj['hourly']['temperature_2m'][indiceDataHora]
    precipitation = jsonObj['hourly']['precipitation'][indiceDataHora]

    return dict({'temperature_2m': temperature_2m, 'precipitation': precipitation})

teste = retorna_dados_metereologicos("2022-05-01T00:00:00", -19.9377, -43.92711)
teste

base_clima['TEMPERATURA'] = np.nan
base_clima['PRECIPITACAO'] = np.nan
for i in base_clima.index:
    dados_clima = retorna_dados_metereologicos(base_clima['DATA_HORA_COMPARADOR'][i], -19.9377, -43.92711)
    base_clima.loc[i, 'TEMPERATURA'] = dados_clima['temperature_2m']
    base_clima.loc[i, 'PRECIPITACAO'] = dados_clima['precipitation']
base_clima.to_csv("dados/fluxo_moto/base_clima.csv", index=False)

base_clima

base_fluxo_motocicleta = pd.read_csv('dados/fluxo_moto/base_fluxo_motos.csv')

base_fluxo_motocicleta.head(3)

def data_hora_minuto(data_hora):
    dataDate = datetime.strptime(data_hora, '%Y-%m-%dT%H:%M:%S')
    dataStrRetorno = dataDate.strftime('%Y-%m-%dT%H:00:00')
    return dataStrRetorno

```

```

base_fluxo_motocicleta['HORA_PASSAGEM_MOTO'] = base_fluxo_motocicleta['DATA_HORA'].apply(lambda
data_hora_atual: data_hora_minuto(data_hora_atual))

base_fluxo_motocicleta.head(3)

frequencia_fluxo_motos_data = base_fluxo_motocicleta['HORA_PASSAGEM_MOTO'].value_counts()
frequencia_fluxo_motos_data

type(frequencia_fluxo_motos_data)

frequencia_fluxo_motos_data['2022-05-01T02:00:00']

def recupera_frequencia(indice, frequencia_fluxo_motos_data):
    freq = 0
    try:
        freq = frequencia_fluxo_motos_data[indice]
    except:
        freq = 0
    return freq

datas = pd.date_range(start='2022-05-01 00:00:00', end='2022-06-30 23:00:00', periods = 1464)
base_frequencia_fluxo_moto = pd.DataFrame(datas, columns=['DATA'])
base_frequencia_fluxo_moto['DATA_HORA_COMPARADOR'] = base_frequencia_fluxo_moto['DATA'].apply(lambda
data_atual: format(data_atual, "%Y-%m-%dT%H:00:00"))
base_frequencia_fluxo_moto['HORA'] = base_frequencia_fluxo_moto['DATA'].apply(lambda data_atual: for-
mat(data_atual, "%H"))

base_frequencia_fluxo_moto['FREQUENCIA'] = np.nan

for i in base_frequencia_fluxo_moto.index:
    try:
        base_frequencia_fluxo_moto.loc[i, 'FREQUENCIA'] = frequencia_fluxo_motos_data[base_frequen-
cia_fluxo_moto['DATA_HORA_COMPARADOR'][i]]
    except:
        base_frequencia_fluxo_moto.loc[i, 'FREQUENCIA'] = 0

base_frequencia_fluxo_moto

sns.boxplot(x=base_frequencia_fluxo_moto['HORA'].apply(lambda x : int(x)))

sns.boxplot(x=base_frequencia_fluxo_moto['FREQUENCIA'])

base_frequencia_fluxo_moto.loc[base_frequencia_fluxo_moto['FREQUENCIA'] > 119]

base_frequencia_fluxo_moto.drop(base_frequencia_fluxo_moto[base_frequencia_fluxo_moto['FREQUENCIA'] >
119].index, axis=0, inplace=True)

sns.boxplot(x=base_frequencia_fluxo_moto['FREQUENCIA'])

base_frequencia_fluxo_moto['FREQUENCIA'].describe()

sns.countplot(x = base_frequencia_fluxo_moto['FREQUENCIA'])

media_freq = base_frequencia_fluxo_moto['FREQUENCIA'].mean()
media_freq

base_frequencia_fluxo_moto['ACIMA_MEDIA_FREQUENCIA'] = base_frequencia_fluxo_moto['FREQUENCIA'].ap-
ply(lambda freq_atual : 1 if freq_atual > media_freq else 0)

base_frequencia_fluxo_moto.head(5)

base_frequencia_fluxo_moto.to_csv("dados/fluxo_moto/base_frequencia_moto.csv", index=False)

base = ""
base_tipo_dia = pd.read_csv("dados/fluxo_moto/base_datas.csv")
base_tipo_dia.head(3)

base_clima = pd.read_csv("dados/fluxo_moto/base_clima.csv")
base_clima.head(3)

base_frequencia_fluxo_moto = pd.read_csv("dados/fluxo_moto/base_frequencia_moto.csv")

base = base_frequencia_fluxo_moto

```

```

base.insert(loc = 3, column = 'TEMPERATURA', value = base['DATA_HORA_COMPARADOR'].map(lambda x :
base_clima.loc[base_clima.DATA_HORA_COMPARADOR == x].iloc[0]['TEMPERATURA']))

base.insert(loc = 3, column = 'PRECIPITACAO', value = base['DATA_HORA_COMPARADOR'].map(lambda x :
base_clima.loc[base_clima.DATA_HORA_COMPARADOR == x].iloc[0]['PRECIPITACAO']))

base.insert(loc = 3, column = 'TIPO_DIA', value = base['DATA_HORA_COMPARADOR'].map(lambda x :
base_tipo_dia.loc[base_tipo_dia.DATA_HORA_COMPARADOR == x].iloc[0]['TIPO_DIA']))

base = base.drop(['DATA_HORA_COMPARADOR', 'DATA', 'FREQUENCIA'], axis=1)

base.columns[base.isna().any()].tolist()

base

base.isnull().sum()

base.duplicated().sum()

base = base.drop_duplicates()

base

base.to_csv("dados/fluxo_moto/base_fluxo_motos_tratada.csv", index=False)

def plota_countplot(base_principal, variavel, titulo_x, titulo_y, titulo_do_grafico):
    plt.clf()
    grafico = sns.countplot(x=base_principal[variavel])
    grafico.set_ylabel('Quantidade de Ocorrências');
    grafico.set_title(f'{titulo_do_grafico}\n');
    plt.show()

def plota_histplot(base_principal, variavel, titulo_x, titulo_y, titulo_do_grafico):
    plt.clf()
    grafico = sns.histplot(x=base_principal[variavel])
    grafico.set_ylabel('Quantidade de Ocorrências');
    grafico.set_title(f'{titulo_do_grafico}\n');
    plt.show()

base = pd.read_csv("dados/fluxo_moto/base_fluxo_motos_tratada.csv")
base

base.isna().sum()

base.isnull().sum()

np.unique(base['ACIMA_MEDIA_FREQUENCIA'], return_counts = True)

plota_countplot(base, 'ACIMA_MEDIA_FREQUENCIA', 'Acima da média da frequência', 'Quantidade de Ocorrências', 'Visualização dos registros que estão acima e abaixo da média da frequência')

plota_countplot(base, 'HORA', 'Hora', 'Quantidade de Ocorrências', 'Visualização da quantidade de amostras por hora')

plota_histplot(base, 'TEMPERATURA', 'Temperatura durante a hora de registros', 'Quantidade de Ocorrências', 'Visualização dos registros de temperatura')

sns.boxplot(x=base['TEMPERATURA'])

plota_countplot(base, 'TIPO_DIA', 'Tipo do dia do registro', 'Quantidade de Ocorrências', 'Visualização da quantidade de tipos de dia da ocorrência')

plt.clf()
grafico = sns.histplot(data=base, x="TIPO_DIA", hue="ACIMA_MEDIA_FREQUENCIA", multiple="dodge", shrink=0.8)
grafico.set_title(f'Ocorrência de fluxos acima e abaixo da média por tipo de dia de registro\n');
grafico.set_ylabel('Quantidade de Ocorrências');
plt.show()

grafico = px.scatter_matrix(base, dimensions=['TIPO_DIA', 'PRECIPITACAO', 'TEMPERATURA', 'HORA'], color = 'ACIMA_MEDIA_FREQUENCIA')
grafico.show()

```

```

grafico = px.scatter_matrix(base, dimensions=['TEMPERATURA', 'PRECIPITACAO'], color = 'ACIMA_MEDIA_FRE-
QUENCIA')
grafico.show()

grafico = px.scatter_matrix(base, dimensions=['TIPO_DIA', 'HORA'], color = 'ACIMA_MEDIA_FREQUENCIA')
grafico.show()

def plota_heatmap_1(rotulo_y, rotulo_x, rotulo_preenchimento, foco_preenchimento, largura = 5, altura =
5, titulo = ''):
    plt.clf()
    df = base.loc[:, [rotulo_y, rotulo_x, rotulo_preenchimento]]
    colunas = df[rotulo_x].unique()
    indices = df[rotulo_y].unique()
    mapa = pd.DataFrame(columns=colunas, index = indices)
    for i in mapa.columns:
        for j in mapa.index:
            mapa[i][j] = pd.to_numeric(df.loc[df[rotulo_y]==j].loc[df[rotulo_x]==i].loc[df[rotulo_pre-
enchimento] == foco_preenchimento].shape[0])
            mapa[i] = pd.to_numeric(mapa[i], errors = 'coerce')
    plt.figure(figsize = (largura, altura))
    grafico = sns.heatmap(mapa, annot=True, linewidths=0)
    grafico.set_title(f'{titulo}\n')
    grafico.set_ylabel(rotulo_y);
    grafico.set_xlabel(rotulo_x);
    plt.show()
plota_heatmap_1('TIPO_DIA', 'HORA', 'ACIMA_MEDIA_FREQUENCIA', 1, 14, 2, 'Mapa de Calor Indicando Regis-
tro de Frequência de Fluxo Acima da Média')

def plota_heatmap_2(rotulo_y, rotulo_x, rotulo_preenchimento, foco_preenchimento, largura = 5, altura =
5, titulo = ''):
    plt.clf()
    base[rotulo_x] = base[rotulo_x].apply(lambda x : round(int(x)))
    base['PRECIPITACAO_CATEGORICO'] = base[rotulo_y].apply(lambda x : "NAO_CHOVE" if x == 0.0 else
"CHOVE")
    rotulo_y = 'PRECIPITACAO_CATEGORICO'
    df = base.loc[:, [rotulo_y, rotulo_x, rotulo_preenchimento]]
    colunas = sorted(df[rotulo_x].unique())
    indices = sorted(df[rotulo_y].unique())
    mapa = pd.DataFrame(columns=colunas, index = indices)
    for i in mapa.columns:
        for j in mapa.index:
            mapa[i][j] = pd.to_numeric(df.loc[df[rotulo_y]==j].loc[df[rotulo_x]==i].loc[df[rotulo_pre-
enchimento] == foco_preenchimento].shape[0])
            mapa[i] = pd.to_numeric(mapa[i], errors = 'coerce')
    plt.figure(figsize = (largura, altura))
    grafico = sns.heatmap(mapa, annot=True, linewidths=0)
    grafico.set_title(f'{titulo}\n')
    grafico.set_ylabel(rotulo_y);
    grafico.set_xlabel(rotulo_x);
    plt.show()
plota_heatmap_2('PRECIPITACAO', 'HORA', 'ACIMA_MEDIA_FREQUENCIA', 1, 15, 3, 'Mapa de Calor Indicando
Registro de Frequência de Fluxo Acima da Média')

def plota_heatmap_3(rotulo_y, rotulo_x, rotulo_preenchimento, foco_preenchimento, largura = 5, altura =
5, titulo = ''):
    plt.clf()
    base[rotulo_x] = base[rotulo_x].apply(lambda x : round(x))
    base[rotulo_y] = base[rotulo_y].apply(lambda y : round(y))
    df = base.loc[:, [rotulo_y, rotulo_x, rotulo_preenchimento]]
    colunas = sorted(df[rotulo_x].unique())
    indices = sorted(df[rotulo_y].unique(), reverse=True)
    mapa = pd.DataFrame(columns=colunas, index = indices)
    for i in mapa.columns:
        for j in mapa.index:
            mapa[i][j] = pd.to_numeric(df.loc[df[rotulo_y]==j].loc[df[rotulo_x]==i].loc[df[rotulo_pre-
enchimento] == foco_preenchimento].shape[0])
            mapa[i] = pd.to_numeric(mapa[i], errors = 'coerce')
    plt.figure(figsize = (largura, altura))
    grafico = sns.heatmap(mapa, annot=True, linewidths=0)
    grafico.set_title(f'{titulo}\n')
    grafico.set_ylabel(rotulo_y);
    grafico.set_xlabel(rotulo_x);
    plt.show()

```



```

plota_heatmap_3('TEMPERATURA', 'HORA', 'ACIMA_MEDIA_FREQUENCIA', 1, 9, 6, 'Mapa de Calor Indicando Registro de Frequência de Fluxo de Motocicletas Acima da Média')

base_ml = pd.read_csv('dados/fluxo_moto/base_fluxo_motos_tratada.csv')
base_ml

base_ml = pd.get_dummies(base_ml)
base_ml

X = base_ml.drop('ACIMA_MEDIA_FREQUENCIA', axis=1)
y = base_ml['ACIMA_MEDIA_FREQUENCIA']

plt.clf()
grafico = sns.countplot(x=y)
grafico.set_ylabel('Quantidade de Ocorrências');
grafico.set_title(f'Visualização dos registros que estão acima e abaixo da média da frequência\n');
plt.show()

np.unique(y, return_counts = True)

smote = SMOTE(sampling_strategy='minority')
X, y = smote.fit_resample(X, y)

plt.clf()
grafico = sns.countplot(x=y)
grafico.set_ylabel('Quantidade de Ocorrências');
grafico.set_title(f'Visualização dos registros que estão acima e abaixo da média da frequência\n');
plt.show()

np.unique(y, return_counts = True)

scaler = MinMaxScaler()
X = scaler.fit_transform(X)
X

base_majority = Orange.data.Table('dados/fluxo_moto/base_fluxo_motos_tratada_regras.csv')

base_majority.domain

majority = Orange.classification.MajorityLearner()

previsoes = Orange.evaluation.testing.TestOnTestData(base_majority, base_majority, [majority])

Orange.evaluation.CA(previsoes)

parametros = {'criterion': ['gini', 'entropy'],
              'n_estimators': range(35, 45),
              'min_samples_split': range(2, 10),
              'min_samples_leaf': range(2, 10)}

grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)

parametros = {'n_neighbors': range(10, 15),
              'p': [1, 2],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'metric': ['cityblock', 'minkowski', 'euclidean']}

grid_search = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)

parametros = {'tol': [0.1, 0.01, 0.001, 0.0001, 0.00001],
              'random_state': [1, 2, 3, 4, 5, 6],
              'C': [2.5, 3.0, 3.5, 4.0, 4.5, 5.0],
              'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}

grid_search = GridSearchCV(estimator=SVC(), param_grid=parametros)

```

```

grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)

resultados_random_forest = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    random_forest = RandomForestClassifier(n_estimators=36, criterion='gini', min_samples_leaf= 6,
min_samples_split= 5, random_state = 0)
    pontos = cross_val_score(random_forest, X, y, cv = kfold, scoring='accuracy')
    resultados_random_forest.append(pontos.mean())

df_resultados_random_forest = pd.DataFrame(resultados_random_forest, columns=['RESULTADOS_RANDOM_FO-
REST'])
df_resultados_random_forest.describe()

resultados_knn = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    knn = KNeighborsClassifier(algorithm='auto', n_neighbors=14, metric='cityblock', p = 1)
    pontos = cross_val_score(knn, X, y, cv = kfold, scoring='accuracy')
    resultados_knn.append(pontos.mean())

df_resultados_knn = pd.DataFrame(resultados_knn, columns=['RESULTADOS_KNN'])
df_resultados_knn.describe()

resultados_svm = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    svm = SVC(kernel='rbf', random_state=1, C = 3.0, tol = 0.01)
    pontos = cross_val_score(svm, X, y, cv = kfold, scoring='accuracy')
    resultados_svm.append(pontos.mean())

df_resultados_svm = pd.DataFrame(resultados_svm, columns=['RESULTADOS_SVM'])
df_resultados_svm.describe()

resultados = pd.DataFrame({'RANDOM_FOREST': resultados_random_forest, 'KNN': resultados_knn, 'SVM': re-
sultados_svm})
resultados.describe()

(resultados.std() / resultados.mean())*100

#Recupera e trata colunas da base
base_ml = pd.read_csv('dados/fluxo_moto/base_fluxo_motos_tratada.csv')
base_ml = base_ml.drop(['TIPO_DIA', 'HORA'], axis=1)

#Separa a classe alvo
X = base_ml.drop('ACIMA_MEDIA_FREQUENCIA', axis=1)
y = base_ml['ACIMA_MEDIA_FREQUENCIA']

#Escalona os dados
scaler = StandardScaler()
X = scaler.fit_transform(X)

#Faz o tuning dos parametros
# -> Random Forest
parametros = {'criterion': ['gini', 'entropy'],
              'n_estimators': range(38, 41),
              'min_samples_split': range(6, 10),
              'min_samples_leaf': range(6, 10)}
grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print("=====\n Tuning Random Forest\n=====")
print(melhores_parametros)
print(melhor_resultado)
# -> Random KNN
parametros = {'n_neighbors': [35,40, 45],
              'p': [1, 2],
              'algorithm':['auto', 'ball_tree', 'kd_tree', 'brute'],
              'metric': ['cityblock', 'minkowski', 'euclidean']}

```

```

grid_search = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print("=====\n      Tuning KNN\n=====")
print(melhores_parametros)
print(melhor_resultado)
# -> Random SVM
parametros = {'tol': [0.1, 0.01, 0.001, 0.0001, 0.00001],
              'random_state': [1,2,3,4,5,6],
              'C': [0.5, 1.0, 1.5, 2.0],
              'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}
grid_search = GridSearchCV(estimator=SVC(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print("=====\n      Tuning SVM\n=====")
print(melhores_parametros)
print(melhor_resultado)

# Avaliacao dos algoritmos com cross-validation
resultados_random_forest = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    random_forest = RandomForestClassifier(n_estimators=38, criterion='entropy', min_samples_leaf=6,
min_samples_split=7, random_state = 0)
    pontos = cross_val_score(random_forest, X, y, cv = kfold, scoring='accuracy')
    resultados_random_forest.append(pontos.mean())

resultados_knn = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    knn = KNeighborsClassifier(algorithm='ball_tree', n_neighbors=40, metric='cityblock', p = 1)
    pontos = cross_val_score(knn, X, y, cv = kfold, scoring='accuracy')
    resultados_knn.append(pontos.mean())

resultados_svm = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    svm = SVC(kernel='rbf', random_state=1, C = 1.5, tol = 0.1)
    pontos = cross_val_score(svm, X, y, cv = kfold, scoring='accuracy')
    resultados_svm.append(pontos.mean())
resultados_dados_climaticos = pd.DataFrame({'RANDOM_FOREST': resultados_random_forest, 'KNN': resultados_knn, 'SVM': resultados_svm})
resultados_dados_climaticos.describe()

#Recupera e trata colunas da base
base_ml = pd.read_csv('dados/fluxo_moto/base_fluxo_motos_tratada.csv')
base_ml = base_ml.drop(['PRECIPITACAO', 'TEMPERATURA'], axis=1)
base_ml = pd.get_dummies(base_ml)

#Separa a classe alvo
X = base_ml.drop('ACIMA_MEDIA_FREQUENCIA', axis=1)
y = base_ml['ACIMA_MEDIA_FREQUENCIA']

#Escalona os dados
scaler = StandardScaler()
X = scaler.fit_transform(X)

#Faz o tuning dos parametros
# -> Random Forest
parametros = {'criterion': ['gini', 'entropy'],
              'n_estimators': range(42, 45),
              'min_samples_split': range(6, 10),
              'min_samples_leaf': range(4, 8)}
grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print("=====\n Tuning Random Forest\n=====")
print(melhores_parametros)
print(melhor_resultado)
# -> Random KNN
parametros = {'n_neighbors': [4,6,8,10,12,14,15,16],

```

```

        'p': [1, 2],
        'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
        'metric': ['cityblock', 'minkowski', 'euclidean']}
grid_search = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print("=====\n      Tuning KNN\n=====")
print(melhores_parametros)
print(melhor_resultado)
# -> Random SVM
parametros = {'tol': [0.1, 0.01, 0.001, 0.0001, 0.00001],
              'random_state': [1,2],
              'C': [0.5, 1.0, 1.5, 2.0],
              'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}
grid_search = GridSearchCV(estimator=SVC(), param_grid=parametros)
grid_search.fit(X, y)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print("=====\n      Tuning SVM\n=====")
print(melhores_parametros)
print(melhor_resultado)

# Avaliacao dos algoritmos com cross-validation
resultados_random_forest = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    random_forest = RandomForestClassifier(n_estimators=42, criterion='entropy', min_samples_leaf=5,
min_samples_split=7, random_state = 0)
    pontos = cross_val_score(random_forest, X, y, cv = kfold, scoring='accuracy')
    resultados_random_forest.append(pontos.mean())

resultados_knn = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    knn = KNeighborsClassifier(algorithm='brute', n_neighbors=15, metric='minkowski', p = 2)
    pontos = cross_val_score(knn, X, y, cv = kfold, scoring='accuracy')
    resultados_knn.append(pontos.mean())

resultados_svm = []
for i in range(30):
    kfold = KFold(n_splits=10, shuffle=True, random_state=i)
    svm = SVC(kernel='rbf', random_state=1, C = 1.0, tol = 0.1)
    pontos = cross_val_score(svm, X, y, cv = kfold, scoring='accuracy')
    resultados_svm.append(pontos.mean())
resultados_dados_dia_hora = pd.DataFrame({'RANDOM_FOREST': resultados_random_forest, 'KNN': resulta-
dos_knn, 'SVM': resultados_svm})
resultados_dados_dia_hora.describe()

print("=====")
print("Dados Fluxo + Dados Clima + Dados de Tipos de Dias e Horários")
print("=====")
print(resultados.describe())
print(".....")
print(" ")
print("=====")
print("Dados Fluxo + Dados Clima")
print("=====")
print(resultados_dados_climaticos.describe())
print(".....")
print(" ")
print("=====")
print("Dados Fluxo + Dados de Tipos de Dias e Horários")
print("=====")
print(resultados_dados_dia_hora.describe())

dados_medias = {'ALGORITMO': ['Random Forest', 'Random Forest', 'Random Forest', 'KNN', 'KNN', 'KNN',
'SVM', 'SVM', 'SVM'],
                'CONFIGURACAO': ['Configuração 1', 'Configuração 3', 'Configuração 2', 'Configuração
1', 'Configuração 3', 'Configuração 2', 'Configuração 1', 'Configuração 3', 'Configuração 2'],
                'MEDIA': [resultados.describe()['RANDOM_FOREST']['mean'], resultados_dados_dia_hora.des-
cribe()['RANDOM_FOREST']['mean'], resultados_dados_climaticos.describe()['RANDOM_FOREST']['mean'],
resultados_dados_dia_hora.des-
cribe()['KNN']['mean'], resultados_dados_climaticos.describe()['KNN']['mean'],
resultados_dados_dia_hora.des-

```

```

        resultados.describe()['SVM']['mean'], resultados_dados_dia_hora.des-
cribe()['SVM']['mean'], resultados_dados_climaticos.describe()['SVM']['mean']}]
df_medias = pd.DataFrame(dados_medias)

plt.clf()
plt.figure(figsize = (8, 6))
grafico = sns.barplot(data=df_medias, x="ALGORITMO", y="MEDIA", hue="CONFIGURACAO")
grafico.set_title(f'Média da Acurácia por Algoritmo e Configuração de Dados do Modelo\n');
grafico.set_ylabel('Média da Acurácia');
grafico.set_xlabel('Algoritmos');
plt.legend(loc='upper right')
plt.ylim(0.6, 0.9)
plt.show()

dados_desvio_padrao = {'ALGORITMO': ['Random Forest', 'Random Forest', 'Random Forest', 'KNN', 'KNN',
'KNN', 'SVM', 'SVM', 'SVM'],
                        'CONFIGURACAO': ['Configuração 1', 'Configuração 3', 'Configuração 2', 'Configuração
1', 'Configuração 3', 'Configuração 2', 'Configuração 1', 'Configuração 3', 'Configuração 2'],
                        'DESVIO_PADRAO': [resultados.describe()['RANDOM_FOREST']['std'], resultados_da-
dos_dia_hora.describe()['RANDOM_FOREST']['std'], resultados_dados_climaticos.describe()['RANDOM_FO-
REST']['std'],
                        resultados.describe()['KNN']['std'], resultados_dados_dia_hora.des-
cribe()['KNN']['std'], resultados_dados_climaticos.describe()['KNN']['std'],
                        resultados.describe()['SVM']['std'], resultados_dados_dia_hora.des-
cribe()['SVM']['std'], resultados_dados_climaticos.describe()['SVM']['std']]}
df_desvio_padrao = pd.DataFrame(dados_desvio_padrao)

plt.clf()
plt.figure(figsize = (8, 6))
grafico = sns.barplot(data=df_desvio_padrao, x="ALGORITMO", y="DESVIO_PADRAO", hue="CONFIGURACAO")
grafico.set_title(f'Desvio Padrão da Acurácia por Algoritmo e Configuração de Dados do Modelo\n');
grafico.set_ylabel('Desvio Padrão da Acurácia');
grafico.set_xlabel('Algoritmos');
plt.legend(loc='upper right')
plt.ylim(0.0005, 0.005)
plt.show()

base_ml = pd.read_csv('dados/fluxo_moto/base_fluxo_motos_tratada.csv')
base_ml = pd.get_dummies(base_ml)
X = base_ml.drop('ACIMA_MEDIA_FREQUENCIA', axis=1)
y = base_ml['ACIMA_MEDIA_FREQUENCIA']
X, y = smote.fit_resample(X, y)
X = scaler.fit_transform(X)
X_treinamento, X_teste, y_treinamento, y_teste = train_test_split(X, y, test_size = 0.15, random_state
= 0)
X_treinamento.shape, y_treinamento.shape, X_teste.shape, y_teste.shape, X.shape, y.shape

random_forest = RandomForestClassifier(n_estimators=37, criterion='entropy', min_samples_leaf=7,
min_samples_split=9, random_state=0)
random_forest.fit(X_treinamento, y_treinamento)

previsoes = random_forest.predict(X_teste)

accuracy_score(y_teste, previsoes)

cm = ConfusionMatrix(random_forest)
cm.fit(X_treinamento, y_treinamento)
cm.score(X_teste, y_teste)

print(classification_report(y_teste, previsoes))

knn = KNeighborsClassifier(algorithm='auto', n_neighbors=20, metric='cityblock', p = 1)
knn.fit(X_treinamento, y_treinamento)

previsoes = knn.predict(X_teste)

accuracy_score(y_teste, previsoes)

cm = ConfusionMatrix(knn)
cm.fit(X_treinamento, y_treinamento)
cm.score(X_teste, y_teste)

print(classification_report(y_teste, previsoes))

```

```
svm = SVC(kernel='rbf', random_state=1, C =2.5, tol=0.01)
svm.fit(X_treinamento, y_treinamento)

previsoes = svm.predict(X_teste)

accuracy_score(y_teste, previsoes)

cm = ConfusionMatrix(svm)
cm.fit(X_treinamento, y_treinamento)
cm.score(X_teste, y_teste)

print(classification_report(y_teste, previsoes))
```