

Relatório parcial de iniciação científica

Felipe Snitynski Camillo

July 11, 2024

1 Introdução

Este relatório descreve o processo de coleta e tratamento de dados para o projeto de análise de padrões no transporte público de Curitiba.

2 Parte 1: Coleta dos Dados

Os dados utilizados para a análise foram concedidos pela URBS (Secretaria de Urbanização de Curitiba) através de acesso ao sistema interno de dados, incluindo localização de pontos de ônibus, linhas, veículos, etc. Os dados são fornecidos no formato JSON e contêm informações em tempo real sobre a localização dos ônibus, que são essenciais para este projeto.

A coleta de dados foi realizada por meio de uma requisição a uma página web que retorna os dados no formato JSON. Em seguida, os dados foram armazenados em um *dataframe* no ambiente Python, com informações relacionadas às linhas de ônibus e às localizações dos respectivos pontos.

Ao todo, foram realizadas duas requisições chave para o projeto: 1) o acesso às informações em tempo real das linhas e, por conseguinte, dos veículos em operação, contendo informações como: código do ônibus em operação, código da linha em que o veículo está operando, coordenadas geográficas (latitude e longitude) do ônibus, situação (se está adiantado, no horário ou atrasado) e a informação do horário em que essas informações foram enviadas, no formato HH:MM. A maioria dos ônibus envia essas informações a cada minuto, embora alguns demorem um pouco mais, além disso temos a informação do sentido do ônibus, sendo IDA ou VOLTA. 2) os dados históricos referentes a localização dos pontos de ônibus de determinada linha, nessas informações temos: nome do ponto, sequencia (indo do ponto inicial até o final, no formato numérico 0 até n) e o sentido do ponto inicial ao final e final ao inicial

3 Parte 2: Processamento dos Dados

Após a coleta dos dados, houve a necessidade de criar funções para coletar e processar os dados em tempo real. Os dados fornecidos pela URBS, relacionados ao veículo, apesar de conterem a localização dos ônibus, não contêm informações sobre as paradas dos ônibus e as sequências de pontos, partes centrais para uma análise exploratória dos dados.

Para processar uma linha durante todo o percurso, as localizações dos ônibus foram atualizadas a cada um minuto. Portanto, para conseguir extrair as informações de um trajeto, foi necessário criar um algoritmo capaz de acessar múltiplas vezes a página web, minuto a minuto, e armazenar em um *dataframe* as informações dos ônibus e, em seguida, modelar em cima das informações. Para isso, foram criadas funções para realizar o processo. Dentre as funções temos:

```
def buscar_e_processar_dados():
```

```
def buscar_e_processar_dados(linha, df_paradas_linha_1, df_paradas_linha_2, df_paradas_linha_3):

    url_base = 'https://transporteservico.urbs.curitiba.pr.gov.br/getVeiculos.php?linha={}&c=821f0'.format(linha)
    try:
        response = requests.get(url_base)
        response.raise_for_status()

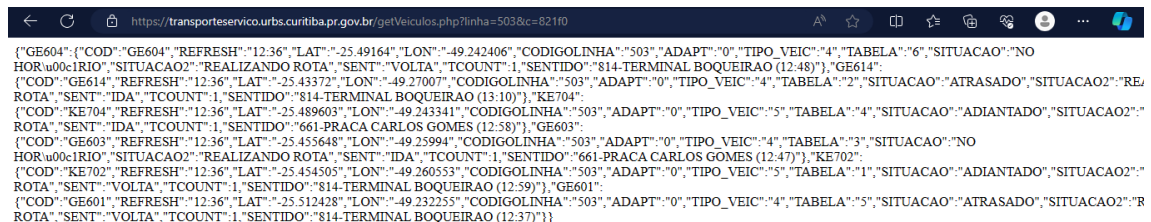
        timezone_sp = pytz.timezone('America/Sao_Paulo')
        hora_online = datetime.now(timezone_sp).strftime("%Y-%m-%d %H:%M:%S")
        dados_json = response.json()
        codigos_onibus = [{
            "COD": valor['COD'],
            "REFRESH": valor['REFRESH'],
            "LAT_IN_TIME": valor['LAT'],
            "LON_IN_TIME": valor['LON'],
            "CODIGOLINHA": valor['CODIGOLINHA'],
            "TABELA": valor['TABELA'],
            "SITUACAO": valor['SITUACAO'],
            "SITUACAO2": valor['SITUACAO2'],
            "SENT": valor['SENT'],
            "SENTIDO_IN_TIME": valor['SENTIDO']
        } for chave, valor in dados_json.items()]
        df_codigos_onibus = pd.DataFrame(codigos_onibus)

        #remover linhas que nao estao em operacao, mas aparecem no site
        indices_para_remover = df_codigos_onibus['SITUACAO2'] != 'REALIZANDO ROTA'].index
        # Removendo as linhas pelos indice

        df_codigos_onibus.drop(index=indices_para_remover, inplace=True)
        df_codigos_onibus.reset_index(drop=True, inplace=True)
```

Figure 1: 1ª parte da Função responsável pela coleta dos dados dos ônibus

Esse trecho da função `buscar_e_processar_dados()` é responsável por fazer a requisição ao site da URBS, o qual contém as informações em tempo real das linhas de ônibus. Os dados JSON estão no seguinte formato:



```
{
  "GE604": {
    "COD": "GE604",
    "REFRESH": "12:36",
    "LAT": "-25.49164",
    "LON": "-49.242406",
    "CODIGOLINHA": "503",
    "ADAPT": "0",
    "TIPO_VEIC": "4",
    "TABELA": "6",
    "SITUACAO": "NO",
    "SITUACAO2": "REALIZANDO ROTA",
    "SENT": "VOLTA",
    "TCOUNT": "1",
    "SENTIDO": "814-TERMINAL BOQUEIRAO (12:48)",
    "GE614": {
      "COD": "GE614",
      "REFRESH": "12:36",
      "LAT": "-25.43372",
      "LON": "-49.27007",
      "CODIGOLINHA": "503",
      "ADAPT": "0",
      "TIPO_VEIC": "4",
      "TABELA": "2",
      "SITUACAO": "ATRASADO",
      "SITUACAO2": "REALIZANDO ROTA",
      "SENT": "IDA",
      "TCOUNT": "1",
      "SENTIDO": "814-TERMINAL BOQUEIRAO (13:10)",
      "KE704": {
        "COD": "KE704",
        "REFRESH": "12:36",
        "LAT": "-25.489603",
        "LON": "-49.243341",
        "CODIGOLINHA": "503",
        "ADAPT": "0",
        "TIPO_VEIC": "5",
        "TABELA": "4",
        "SITUACAO": "ADIANATADO",
        "SITUACAO2": "REALIZANDO ROTA",
        "SENT": "IDA",
        "TCOUNT": "1",
        "SENTIDO": "661-PRACA CARLOS GOMES (12:58)",
        "GE603": {
          "COD": "GE603",
          "REFRESH": "12:36",
          "LAT": "-25.455648",
          "LON": "-49.25994",
          "CODIGOLINHA": "503",
          "ADAPT": "0",
          "TIPO_VEIC": "4",
          "TABELA": "3",
          "SITUACAO": "NO",
          "SITUACAO2": "REALIZANDO ROTA",
          "SENT": "IDA",
          "TCOUNT": "1",
          "SENTIDO": "661-PRACA CARLOS GOMES (12:47)",
          "KE702": {
            "COD": "KE702",
            "REFRESH": "12:36",
            "LAT": "-25.454505",
            "LON": "-49.260553",
            "CODIGOLINHA": "503",
            "ADAPT": "0",
            "TIPO_VEIC": "5",
            "TABELA": "1",
            "SITUACAO": "ADIANATADO",
            "SITUACAO2": "REALIZANDO ROTA",
            "SENT": "VOLTA",
            "TCOUNT": "1",
            "SENTIDO": "814-TERMINAL BOQUEIRAO (12:59)",
            "GE601": {
              "COD": "GE601",
              "REFRESH": "12:36",
              "LAT": "-25.512428",
              "LON": "-49.232255",
              "CODIGOLINHA": "503",
              "ADAPT": "0",
              "TIPO_VEIC": "4",
              "TABELA": "5",
              "SITUACAO": "ATRASADO",
              "SITUACAO2": "REALIZANDO ROTA",
              "SENT": "VOLTA",
              "TCOUNT": "1",
              "SENTIDO": "814-TERMINAL BOQUEIRAO (12:37)"
            }
          }
        }
      }
    }
  }
}
```

Figure 2: Disposição dos dados fornecidos pela URBS

Após o armazenamento no dataframe `df_codigos_onibus`, são realizadas as seguintes sequências de chamadas de funções, dentro da função `buscar_e_processar_dados()`:

```
if(df_paradas_linha_1['COD'].iloc[0] == linha):
    df_sentido_pontos = df_paradas_linha_1.copy()

elif(df_paradas_linha_2['COD'].iloc[0] == linha):
    df_sentido_pontos = df_paradas_linha_2.copy()

elif(df_paradas_linha_3['COD'].iloc[0] == linha):
    df_sentido_pontos = df_paradas_linha_3.copy()
else:
    logging.info("ERRO, DATFRAME DA LINHA NÃO INSERIDO")

df_codigos_onibus['Hora'] = hora_online
mapeamento = dict(zip(df_sentido_pontos['Nome'], df_sentido_pontos['SEQ']))

# Encontrar parada mais proxima vetorizado
df_codigos_onibus['Parada_Mais_Proxima'] = df_codigos_onibus.apply(
    lambda row: encontrar_parada_mais_proxima(row['LAT_IN_TIME'], row['LON_IN_TIME'], df_sentido_pontos),
    axis=1
)

df_codigos_onibus['Distancia_ate_ponto'] = calcular_distancia_ponto(df_codigos_onibus, df_sentido_pontos)
df_codigos_onibus['SEQ'] = df_codigos_onibus['Parada_Mais_Proxima'].apply(lambda x: mapeamento.get(x))
df_codigos_onibus_max = retorna_seq_max(df_codigos_onibus, df_sentido_pontos)

return df_codigos_onibus_max

except requests.RequestException as e:
    logging.error("Falha ao acessar a pagina: %s", e)
    return None
except Exception as e:
    logging.error("Erro ao processar dados: %s", e)
    return None
```

Figure 3: 2ª parte da função `buscar_e_processar_dados()`

Para fazermos a análise exploratória precisamos de mais variáveis, sendo elas: a próxima parada do ônibus, dadas suas coordenadas em tempo real, a distância (em metros) até o ponto mais próximo (podendo ou não ser o próximo ponto), a sequência de pontos da linha do ônibus, do primeiro ponto até o último, e a sequência final (último ponto das linhas, exemplo de sequencia de 1 á 30 pontos, seq max = 30). Problemas relacionados a essas variáveis:

- ****Parada de ônibus e distância****: Quando fiz o algoritmo, os principais problemas encontrados foram: tenho dois sentidos para uma linha, de A para B e de B para A. Como calcular a distância entre o ônibus e os pontos?

- **Resolução****: Dado o dataframe contendo as informações sobre os pontos, selecionamos apenas 1 sentido para servir de base, tive muita dificuldade em trabalhar com os sentidos, pois o nome das variáveis são diferentes entre si, por exemplo, um ônibus que está indo sentido Terminal Santa Cândida, fica registrado com o sentido em tempo real como terminal santa candida, mas no dataframe contendo as informações dos pontos, temos TERM STA CANDIDA, não sendo possível trabalharmos de maneira direta com essas duas informações, como estamos trabalhando apenas com linhas expressas, ou seja, que possuem os mesmos pontos, mas com sequencias diferentes, optei por escolher apenas um sentido. Assim, foi possível calcular corretamente a distância até o ponto mais próximo, a cada minuto o algoritmo calcula a distancia ate o ponto mais próximo, podendo ter varias linhas com o mesmo ponto mais próximo, mas com distancias diferentes, para isso A função `manter_menor_distancia` agrupa os ônibus baseados no seu número de identificação e retorna o menor deles, baseado na distância, atribuindo assim o horário em que foram enviadas as coordenadas à menor distância possível até o ponto.

A execução do código para a captura é feita através de um loop, o qual verifica se

está o horário atual está no intervalo 5h30 - 23h30. O trabalho é feito em paralelismo, utilizando threads, para agilizar o processamento

```
linhas = [203, 503, 303]
df_paradas_linha_1 = retorna_df_pontos_linha(203)
df_paradas_linha_2 = retorna_df_pontos_linha(503)
df_paradas_linha_3 = retorna_df_pontos_linha(303)
df_concatenado = pd.DataFrame()
def processar_linha(linha, result_list):
    try:
        df_result = buscar_e_processar_dados(linha, df_paradas_linha_1, df_paradas_linha_2, df_paradas_linha_3)
        if df_result is not None and not df_result.empty:
            result_list.append(df_result)
    except Exception as e:
        logging.error(f"Erro ao processar linha {linha}: {e}")
def dentro_do_horario():
    agora = datetime.now().time()
    inicio = datetime.strptime("05:30", "%H:%M").time()
    fim = datetime.strptime("23:30", "%H:%M").time()
    return inicio <= agora <= fim
# Loop principal
while True:
    if dentro_do_horario():
        result_list = []
        threads = [threading.Thread(target=processar_linha, args=(linha, result_list)) for linha in linhas]
        for thread in threads:
            thread.start()
        for thread in threads:
            thread.join()
        if result_list:
            for df_result in result_list:
                df_concatenado = pd.concat([df_concatenado, df_result], ignore_index=True)
            df_concatenado = df_concatenado.drop_duplicates(['COD', 'REFRESH']).reset_index(drop=True)
            df_concatenado = manter_menor_distancia(df_concatenado)
            df_concatenado = retorna_sentido(df_concatenado)
            df_concatenado = mandar_bd(df_concatenado, df_paradas_linha_1, df_paradas_linha_2, df_paradas_linha_3)
        # Espera 60 segundos antes da próxima iteração
        time.sleep(60)
```

Figure 4: algoritmo de processamento de linhas

4 Conclusão

A definir até dia 01/08/2024 * Estamos utilizando um processo de clueterização para análise exploratória