Sistema de Análise de Tráfego Urbano

▼ Descrição do Problema

O tráfego urbano apresenta variações significativas entre diferentes regiões, com algumas áreas experimentando volumes muito maiores de veículos do que outras. Nesse contexto, o problema proposto busca desenvolver um sistema capaz de analisar e monitorar o fluxo de tráfego em diferentes locais utilizando técnicas avançadas de estruturas de dados, como tabelas hash. O sistema deve permitir o armazenamento eficiente de informações detalhadas sobre veículos, incluindo sua **placa**, **modelo** e **localização geográfica**, possibilitando consultas rápidas e precisas.

Além disso, o sistema deve fornecer funcionalidades para listar veículos em regiões específicas e identificar as áreas com maior concentração de tráfego, classificando-as em um ranking de congestionamento. Essa análise será fundamental para entender os padrões de movimentação e auxiliar no planejamento de soluções para otimizar o tráfego.

Para validar a eficiência e escalabilidade da solução, serão simulados cenários com grandes volumes de dados, variando de milhares a dezenas de milhares de registros. O desempenho do sistema será medido em termos de tempo de execução para operações de inserção, busca e listagem, e os resultados serão apresentados por meio de gráficos que ilustram o comportamento da solução em diferentes escalas. Esse projeto alia conceitos teóricos e práticos de estruturas de dados e serve como um estudo de caso aplicado ao monitoramento de tráfego urbano.

▼ Solução Implementada + Exemplos

▼ Estrutura de Dados

A solução implementada aborda desde a organização da estrutura de dados até a apresentação visual das informações de forma intuitiva, utilizando mapas. Com o crescente aumento no volume de tráfego e os problemas gerados por planejamentos inadequados, é evidente que, na maioria dos casos, o trânsito enfrenta situações de colapso. Assim, tornou-se essencial adotar uma estrutura de dados que não apenas seja capaz de armazenar grandes volumes de informações, mas que também permita manipulá-las de maneira inteligente, eficiente e eficaz.

Para atender a essa necessidade, utilizamos **Python** como a linguagem de desenvolvimento devido à sua versatilidade e robustez. Implementamos uma **tabela hash** dentro de uma classe, buscando maior organização e encapsulamento dos métodos associados. Essa estrutura foi projetada para garantir uma manipulação rápida e eficiente dos dados, com destaque para os seguintes métodos:

- `__init__`: inicializa a tabela hash, definindo os parâmetros necessários para seu funcionamento.
- `funcao_hash`: calcula um índice único a partir de uma entrada específica (no caso, uma placa de veículo).
- `inserir`: adiciona um novo veículo à tabela hash, associando a placa ao setor correspondente.
- `buscar`: localiza e retorna informações sobre um veículo a partir de sua placa.
- `excluir`: remove um veículo da tabela hash.
- `exibir`: apresenta os dados armazenados na tabela de forma organizada e compreensível.
- `contar_veiculos`: calcula a quantidade de veículos registrados em um setor específico, fornecendo uma visão detalhada do fluxo de tráfego em diferentes regiões.

Essa abordagem não só otimiza o armazenamento, mas também oferece uma base sólida para análises rápidas e precisas, contribuindo para a identificação de gargalos no tráfego e para um planejamento mais eficiente.

```
class TabelaHash:
    def __init__(self, tamanho):
        """Inicializa a tabela hash com um tamanho fixo e
        usa encadeamento para colisões."""
        self.tamanho = tamanho
        self.tabela = [[] for _ in range(tamanho)]  # Lista de listas para encadeamento

    def funcao_hash(self, chave):
        """
        Função hash que retorna o índice onde o item será armazenado.
        Usamos o hash built-in do Python e garantimos que o índice caia dentro
        do tamanho da tabela.
        """
        return hash(chave) % self.tamanho

    def inserir(self, placa, modelo, localizacao):
        """
        Insere um item na tabela hash. Em caso de colisão,
        o item é adicionado à lista de itens na mesma posição da tabela.
        A chave é a placa, o valor é o modelo e a localização é um valor separado.
        """
        indice = self.funcao_hash(placa)
        # Atualiza o valor se a placa já existir
        for item in self.tabela[indice]:
            if item[0] == placa:
                item[1] = modelo
                item[2] = localizacao
                return
        # Adiciona a chave (placa), o valor (modelo) e a localização se não houver colisão
        self.tabela[indice].append([placa, modelo, localizacao])

    def buscar(self, placa):
        """
        Busca por um item na tabela hash. Retorna o modelo
        e a localização associados à placa se encontrar.
        """
        indice = self.funcao_hash(placa)
        for item in self.tabela[indice]:
            if item[0] == placa:
                return item[1], item[2]  # Retorna o modelo e a localização
        print("Placa não encontrada")
        return None, None  # Retorna None se a placa não for encontrada

    def excluir(self, placa):
        """
        Remove um item da tabela hash.
        Retorna True se a placa for encontrada e removida.
        Caso contrário, retorna False.
        """
        indice = self.funcao_hash(placa)
        for i, item in enumerate(self.tabela[indice]):
            if item[0] == placa:
                del self.tabela[indice][i]
                return True
        return False  # Retorna False se a placa não for encontrada

    def exibir(self):
```

```
        """Exibe a tabela hash e suas colisões com modelo e localização."""
        for i, bucket in enumerate(self.tabela):
            print(f"Bucket {i}: {bucket}")

    def contar_veiculos(self, setor):
        """
        Conta quantos veículos estão no mesmo setor.
        """
        indice = self.funcao_hash(setor)
        return len(self.tabela[indice])
```

▼ **Operações**

Neste bloco, agrupamos todas as funções responsáveis por realizar operações complementares, que são executadas fora da estrutura principal da tabela hash. Essas funções desempenham papéis essenciais para enriquecer e dinamizar a manipulação dos dados relacionados ao tráfego. A seguir, detalhamos cada uma delas:

- `gerar_placa`: cria automaticamente placas de veículos, seguindo o padrão brasileiro (três letras seguidas de quatro números). Essa função é útil para simulações de tráfego em larga escala, garantindo unicidade e conformidade.

```
# Gera uma placa de veículo aleatória
def gerar_placa():
    return ''.join(random.choices(string.ascii_uppercase, k=3)) + '-' + ''
    .join(random.choices(string.digits, k=4))
# Exemplo de retorno: 'KZL-7482'
```

- `gerar_modelo`: seleciona modelos de veículos com base em um arquivo de texto previamente gerado por uma Inteligência Artificial. Esse arquivo contém uma lista abrangente de modelos, garantindo maior realismo na simulação e variedade nos dados.

```
# Escolhe um modelo de carro com base nos modelos contidos no arquivo txt
def gerar_modelo():
    caminho_arquivo =
    '/content/drive/MyDrive/ED2 - PROJETO FINAL/Base de Dados/modelosCarros.txt'

    # Abrir o arquivo e ler as linhas
    with open(caminho_arquivo, 'r') as arquivo:
        modelos = [linha.strip() for linha in arquivo.readlines()]

    return random.choice(modelos)
# Exemplo de retorno: 'Volkswagen Polo'
```

- `gerar_localizacao_randomica`: gera localizações aleatórias utilizando dados reais extraídos de um arquivo CSV do IBGE (Instituto Brasileiro de Geografia e Estatística). Essa função permite que os veículos sejam distribuídos em setores específicos, simulando padrões reais de movimentação geográfica.

```
def gerar_localizacao_randomica():
    global ultima_localizacao  # Permite alterar a variável global

    caminho_arquivo =
    '/content/drive/MyDrive/ED2 - PROJETO FINAL/Base de Dados/bairrosGoiania.csv'

    # Verificar se o arquivo existe
    if not os.path.exists(caminho_arquivo):
        raise FileNotFoundError(f"Arquivo não encontrado: {caminho_arquivo}")
```

```
    # Abrir o arquivo CSV e ler os códigos de bairro
    with open(caminho_arquivo, newline='', encoding='utf-8') as arquivo_csv:
        leitor_csv = csv.DictReader(arquivo_csv)
        bairros = [linha['CD_GEOCODS'] for linha in leitor_csv]

    if not bairros:
        raise
        ValueError("A lista de bairros está vazia. Verifique o conteúdo do arquivo CSV.

    # Escolher uma nova localização diferente da atual
    nova_localizacao = ultima_localizacao
    while nova_localizacao == ultima_localizacao:
        nova_localizacao = random.choice(bairros)

    # Atualizar a última localização
    ultima_localizacao = nova_localizacao

    return nova_localizacao
```

- `contar_veiculos_por_regiao`: realiza a contagem de veículos por região, permitindo a análise de densidade de tráfego em setores distintos. Essa funcionalidade é crucial para identificar áreas com maior congestionamento e para priorizar intervenções.

```
def contar_veiculos_por_regiao(tabela_hash):
    """
    Conta quantos veículos existem em cada região com base na localização
    armazenada na tabela hash e retorna uma lista com as contagens.

    Parâmetros:
    tabela_hash (TabelaHash): Instância da classe TabelaHash.

    Retorno:
    list: Lista de tuplas com as regiões e suas respectivas contagens, ordenada por
    número de veículos (decrescente).
    """
    contador_regioes = Counter()

    # Percorre todos os buckets na tabela hash
    for bucket in tabela_hash.tabela:
        for item in bucket:  # Cada item é uma lista [placa, modelo, localizacao]
            _, _, localizacao = item
            contador_regioes[localizacao] += 1  # Incrementa o contador para a região

    # Ordenar regiões por número de veículos (decrescente)
    regioes_ordenadas = contador_regioes.most_common()

    # Retornar a lista de regiões ordenadas
    return regioes_ordenadas
```

- `adicionar_nomes_ao_vetor`: uma função auxiliar que insere nomes ao ranking dos setores mais congestionados. Essa lista é gerada com base nos dados de tráfego e auxilia na criação de relatórios e visualizações, destacando as áreas críticas de forma clara e ordenada.

```
def adicionar_nomes_ao_vetor(vetor, arquivo_csv):
    """
    Adiciona os nomes associados aos códigos do vetor
    usando os dados do arquivo CSV.
```

```
    :param vetor: Lista de tuplas (código, valor)
    :param arquivo_csv: Caminho para o arquivo CSV contendo as colunas
    'CD_GEOCODS' e 'NM_SUBDIST'
    :return: Lista de tuplas (código, nome, valor)
    """
    # Carrega o CSV
    cod_para_nome = {}
    with open(arquivo_csv, mode='r', encoding='utf-8') as file:
        reader = csv.DictReader(file)  # Usar DictReader para mapear colunas por nome
        for linha in reader:
            cod = linha['CD_GEOCODS']
            nome = linha['NM_SUBDIST']
            cod_para_nome[cod] = nome

    # Adiciona os nomes ao vetor
    vetor_atualizado = []
    for codigo, valor in vetor:
        nome = cod_para_nome.get(codigo, "Nome não encontrado")  # Busca o nome pelo có
        vetor_atualizado.append((codigo, nome, valor))

    return vetor_atualizado
```

Considerações Importantes

1. **Integração com Dados Reais**: O uso de fontes reais, como arquivos CSV do IBGE e listas geradas por IA, assegura maior fidelidade nas simulações. Isso torna o sistema uma ferramenta valiosa para estudos de mobilidade urbana e planejamento viário.
2. **Escalabilidade**: As funções foram projetadas para lidar com grandes volumes de dados, permitindo sua aplicação em cenários que vão desde estudos locais até análises em nível estadual ou nacional.
3. **Flexibilidade e Modularidade**: Cada função é independente e modular, facilitando atualizações, adaptações e expansões futuras, caso novos requisitos sejam identificados.
4. **Visualização e Relatórios**: Os dados gerados por essas funções podem ser integrados a ferramentas de visualização, como mapas interativos ou gráficos, proporcionando insights mais acessíveis e intuitivos para planejadores urbanos e gestores de tráfego.

Essas funções não apenas complementam a tabela hash, mas também agregam valor ao sistema, criando uma solução robusta e abrangente para a análise e gestão de tráfego.

▼ Avalição de Desempenho

Este bloco desempenhou um papel crucial na avaliação da qualidade e eficiência das operações relacionadas à manipulação dos dados. Foi desenvolvida uma função específica para calcular o tempo de execução de cada operação, considerando diferentes tamanhos de entrada (n).

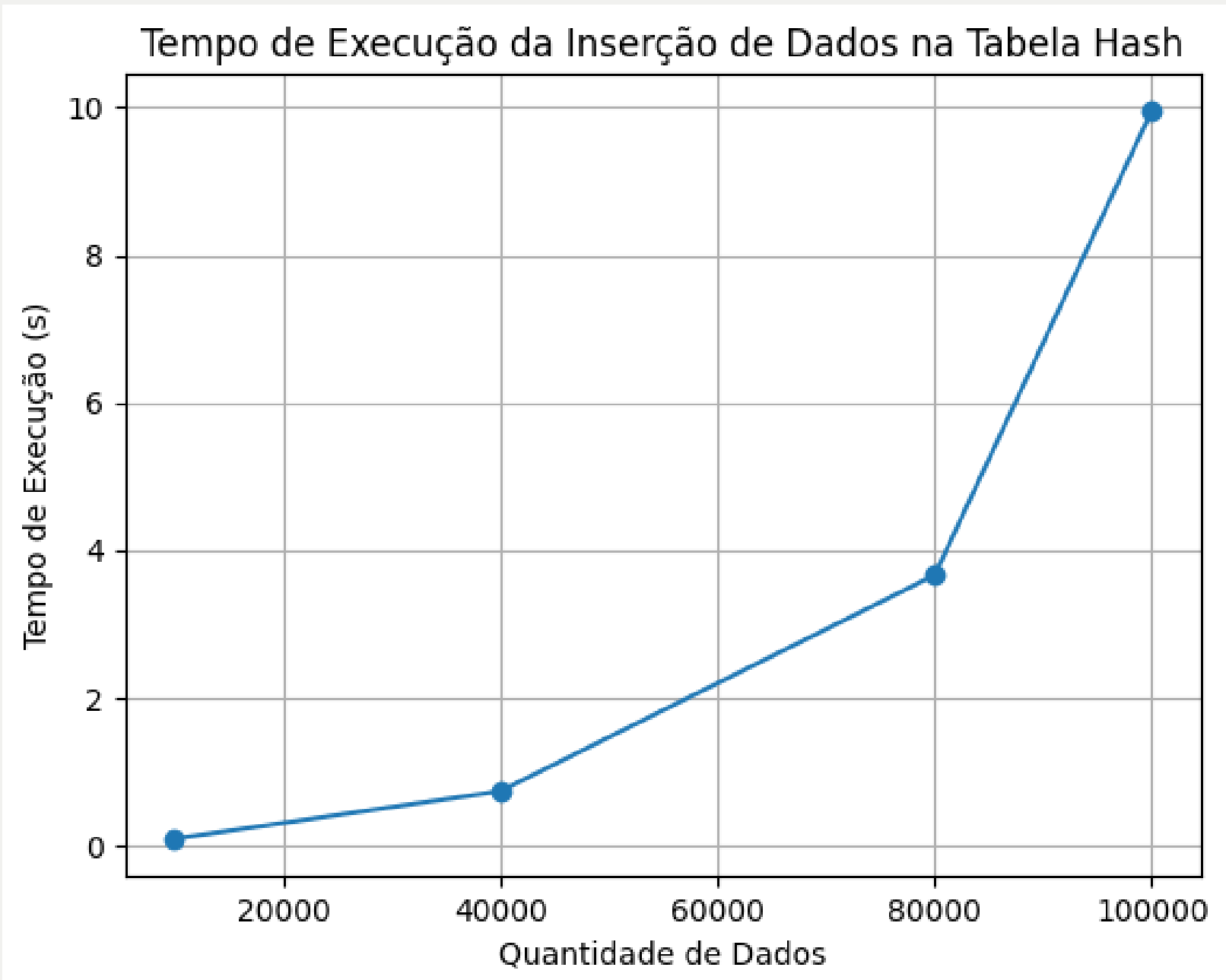
```
# Analisa o tempo de execução de uma determinada função
def avaliar_tempo_execucao(func, *args, **kwargs):
    """
    Função para avaliar o tempo de execução de uma operação (função).

    Parâmetros:
    func (callable): A função a ser executada.
    *args: Argumentos posicionais para passar para a função.
    **kwargs: Argumentos nomeados para passar para a função.

    Retorna:
```

```
    Resultado da execução da função e o tempo gasto.
    """
    # Marcar o tempo inicial
    tempo_inicial = time.time()

    # Executar a função com os argumentos fornecidos
    resultado = func(*args, **kwargs)

    # Marcar o tempo final
    tempo_final = time.time()

    # Calcular o tempo de execução
    tempo_execucao = tempo_final - tempo_inicial

    # Retornar o resultado e o tempo de execução
    return resultado, tempo_execucao
```

Com essa função, realizamos testes sistemáticos nos métodos implementados na tabela hash, utilizando como entrada conjuntos de dados gerados aleatoriamente por meio das funções auxiliares de geração. Essas simulações abrangeram diferentes escalas de dados, variando de 10.000 a 100.000 registros, o que nos permitiu analisar o comportamento do sistema em cenários que simulam situações reais de alta demanda.

```
# Gera 10000 dados aleatórios e armazena em uma lista
dados_10000_aleatorios = []
for _ in range(10000):
    placa = gerar_placa()
    modelo = gerar_modelo()
    localizacao = gerar_localizacao()
    dados_10000_aleatorios.append((placa, modelo, localizacao))
```

Os resultados foram extremamente satisfatórios, demonstrando que a tabela hash foi uma escolha acertada para as manipulações exigidas. A estrutura mostrou-se não apenas eficiente em termos de tempo de execução, mas também altamente escalável, suportando grandes volumes de dados sem degradação significativa no desempenho.



Adicionalmente, a utilização do **Google Colab** como ambiente de desenvolvimento e teste foi um diferencial significativo. O processamento em nuvem proporcionou versatilidade e agilidade, permitindo que as análises fossem realizadas de forma rápida e sem limitações de recursos locais. Isso foi fundamental para identificar gargalos, ajustar a solução conforme necessário e garantir a entrega de um sistema robusto e adequado ao problema proposto.

Em resumo, a combinação da tabela hash com ferramentas modernas como o Colab e uma abordagem orientada a testes consolidou uma solução eficiente, escalável e otimizada para a manipulação e análise de grandes volumes de dados.

▼ Vizulização em Mapa

A fim de adicionar um elemento dinâmico e interativo à visualização, integrarmos um mapa geográfico real do município de Goiânia. Como é amplamente reconhecido, o Python oferece uma vasta gama de bibliotecas poderosas para análise de dados, e aproveitamos esse recurso para aprimorar a experiência visual e tornar a análise mais intuitiva e acessível.

Utilizando a malha de setores censitários disponibilizada pelo IBGE, obtivemos o arquivo shapefile de Goiás. Com isso, filtramos as divisões territoriais do município de Goiânia e extraímos as informações de cada setor. Com esses dados, conseguimos gerar uma visualização precisa e detalhada das regiões da cidade, utilizando bibliotecas como **GeoPandas** e **Matplotlib**.

```
# Estado de Goiás dividido em setores censitários
# corrigir o caminho do arquivo no google drive
setores =
gp.read_file(
'/content/drive/MyDrive/ED2 - PROJETO FINAL/Setores: GO/52SEE250GC_SIR.shp'
)
```

```
# Selecionando a cidade de Goiânia
goiania = setores.query('NM_MUNICIP == "GOIÂNIA"')
mapa_goiania = goiania.query('TIPO == "URBANO"')
```

A função `plot_mapa_goiania` foi desenvolvida para facilitar a criação desse mapa dinâmico. Ela requer apenas dois parâmetros para gerar a visualização completa: o primeiro parâmetro é um vetor de dados que especifica os valores das regiões, e o segundo parâmetro é utilizado para configurar a formatação do mapa. Com essa função, conseguimos gerar um mapa de calor, onde as regiões são coloridas de acordo com os dados fornecidos, permitindo uma visualização clara e eficiente das áreas de maior ou menor intensidade, conforme o vetor passado.

```
# Função para criar um mapa com base em um GeoDataFrame e um vetor
def plot_mapa_goiania(df, vetor, coluna_cod='CD_GEOCODS', cmap='plasma_r'):
    """
    Plota o mapa de Goiânia com coloração baseada
    em um vetor de valores e adiciona uma legenda
    com os 10 setores mais congestionados.

    :param df: GeoDataFrame contendo o mapa de Goiânia
    :param vetor: Lista de tuplas (código, nome, valor)
    :param coluna_cod: Nome da coluna no GeoDataFrame que
    corresponde aos códigos no vetor
    :param cmap: Mapa de cores para a coloração,
    invertido para mais escuro no maior valor
    """
    # Criar um dicionário de mapeamento: código -> valor
    mapa_valores = {codigo: valor for codigo, _, valor in vetor}

    # Adicionar a coluna de valores ao GeoDataFrame
    df['Valor'] = df[coluna_cod].map(mapa_valores)

    # Ordenar os 10 setores mais congestionados (maiores valores)
    top_10_setores = sorted(vetor, key=lambda x: x[2], reverse=True)[:10]
    top_10_setores = [(codigo, nome) for codigo, nome, _ in top_10_setores]

    # Criar o mapa
    fig, ax = plt.subplots(figsize=(15, 10))

    # Plotar o mapa com coloração baseada na coluna 'Valor'
    df.plot(
        column='Valor',  # Coluna para basear a coloração
        cmap=cmap,       # Escolher o mapa de cores invertido
        legend=True,     # Exibir legenda
        legend_kwds={'label': "Intensidade"},  # Rótulo da legenda
        ax=ax            # Usar o eixo criado
    )

    # Configurar título e remover eixos
    ax.set_title("Mapa de Goiânia - Regiões Urbanas", fontsize=20, fontweight='bold')
    ax.set_axis_off()

    # Adicionar a legenda com os 10 setores mais congestionados
    # Posição da legenda na parte inferior do gráfico (relativa ao gráfico)
    legend_pos = (0.02, 0.02)
    legend_text = "\n".join([f"{nome}" for _, nome in top_10_setores])
    ax.text(legend_pos[0], legend_pos[1], f"Top 10 Setores mais Congestionados:\n{legend_t
```

```
        transform=ax.transAxes, fontsize=12, ha='left', va='top',
        bbox=dict(facecolor='white', alpha=0.7, edgecolor='black', boxstyle="round,pad=1")

    # Exibir o gráfico
    plt.show()
```

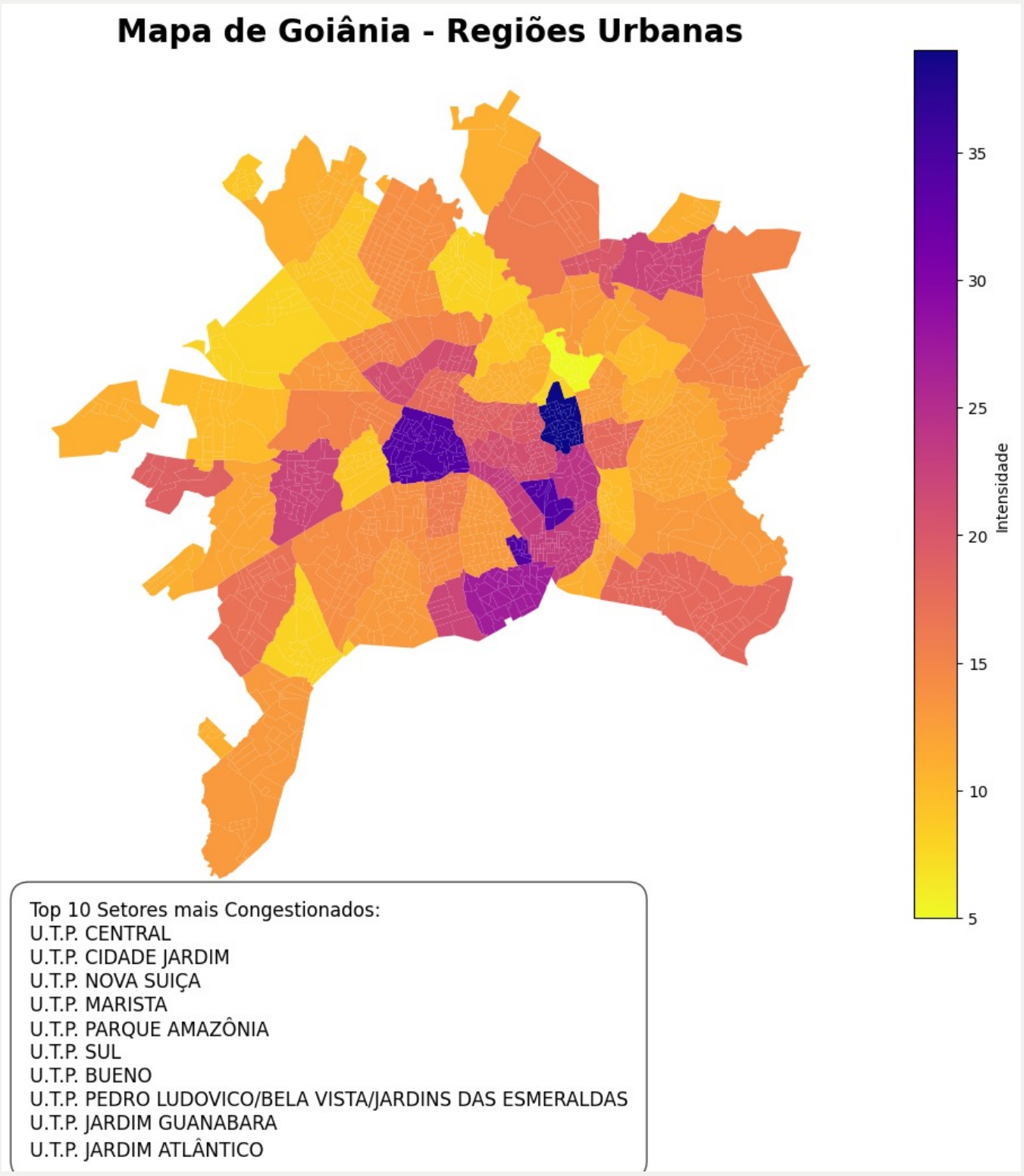
Essa abordagem não só enriqueceu a análise de dados, como também trouxe um elemento visual poderoso para identificar padrões geográficos e analisar o comportamento das variáveis em diferentes setores do município. O uso de um mapa interativo e dinâmico proporciona uma compreensão mais intuitiva dos dados, facilitando a identificação de zonas críticas ou com alto tráfego, além de contribuir significativamente para o planejamento urbano e a tomada de decisões.

▼ **Simulação de Tráfego**

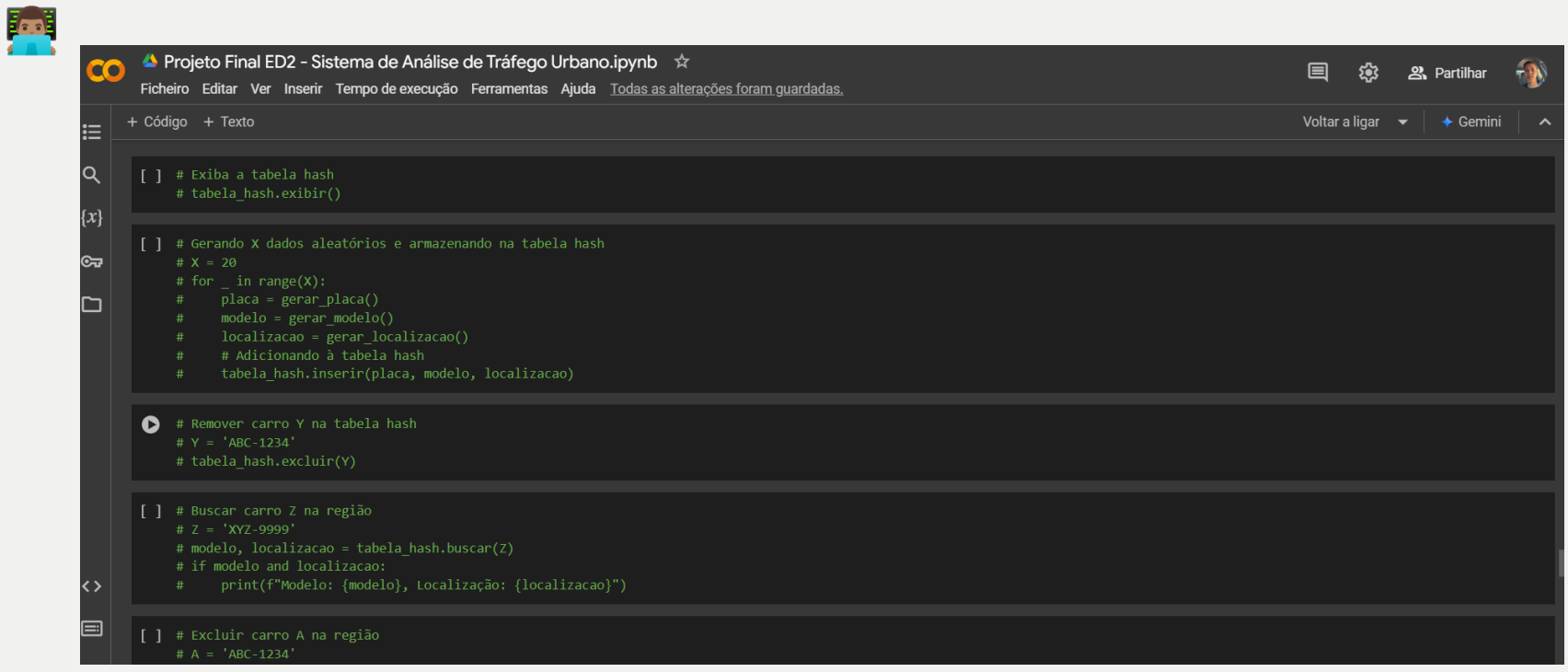
Para simular o tráfego, utilizamos um arquivo **CSV** contendo dados de 1.000 veículos. Através de uma função aleatória, alteramos a localização desses veículos, simulando sua movimentação nas diversas regiões do município. Esse processo permitiu criar uma dinâmica realista de circulação de veículos, o que, por sua vez, possibilitou a visualização das áreas mais congestionadas da cidade, com base nos dados gerados.

```
# Atualiza a localização
atualizar_localizacoes_aleatorias_tabela(tabela_hash_goiania, quantidade=100)

# Gerar vetor ordenado de ranking por Bairro
quantidade_veiculos = contar_veiculos_por_regiao(tabela_hash_goiania)
arquivo_csv = '/content/drive/MyDrive/ED2 - PROJETO FINAL/Base de Dados/bairrosGoiania.csv
ranking_transito_setor = adicionar_nomes_ao_vetor(quantidade_veiculos, arquivo_csv)

# Chamar a função para plotar o mapa
plot_mapa_goiania(mapa_goiania, ranking_transito_setor)
```

Com as funções implementadas no bloco de operações, criamos um vetor ordenado com os setores mais congestionados, facilitando a manipulação dos dados e a geração de uma visualização precisa. Isso garantiu que o mapa fosse plotado com qualidade, destacando claramente as regiões com maior fluxo de veículos. Essa organização também proporcionou facilidade de interpretação, permitindo identificar rapidamente as áreas críticas em termos de tráfego.



Além disso, dentro do notebook, deixamos funções prontas para que o usuário possa, de maneira dinâmica e intuitiva, inserir, remover ou consultar placas de veículos. Isso confere ainda mais interatividade ao sistema, tornando-o flexível para diferentes cenários de teste e permitindo simulações personalizadas conforme as necessidades do usuário.



Acreditamos que os resultados alcançados foram bastante próximos da realidade, tornando a ferramenta altamente eficaz para a análise do tráfego urbano. Essa solução não apenas permite a visualização de congestionamentos, mas também oferece insights valiosos para possíveis intervenções no tráfego e no planejamento urbano, tornando-a uma ferramenta essencial para a investigação e solução de problemas relacionados ao trânsito.

▼ Prompts de IA Utilizados + Materiais

- Leitura de Arquivos

```
# Lendo um arquivo .txt
with open('arquivo.txt', 'r') as file:
    # Lê todo o conteúdo
    conteudo = file.read()
    print(conteudo)

# Lendo linha por linha
with open('arquivo.txt', 'r') as file:
    for linha in file:
        print(linha.strip())  # .strip() remove espaços em branco extras

--------------------------------------------------------------------------------
import csv

# Lendo um arquivo CSV
with open('arquivo.csv', mode='r', newline='', encoding='utf-8') as file:
    reader = csv.reader(file)
    for linha in reader:
        print(linha)  # Cada linha será uma lista de valores

--------------------------------------------------------------------------------
import pandas as pd

# Lendo um arquivo CSV com pandas
df = pd.read_csv('arquivo.csv')
print(df)
```

- Manipulação de Dados

```
from collections import Counter
```

```
# Vetor (lista) de exemplo
vetor = [1, 2, 3, 2, 1, 1, 4]

# Contando todas as ocorrências
contador = Counter(vetor)
print(contador)  # Saída: Counter({1: 3, 2: 2, 3: 1, 4: 1})

-------------------------------------------------------------------------------------
cod_para_nome = {}
    with open(arquivo_csv, mode='r', encoding='utf-8') as file:
    # Usar DictReader para mapear colunas por nome
        reader = csv.DictReader(file)
        for linha in reader:
            cod = linha['CD_GEOCODS']
            nome = linha['NM_SUBDIST']
            cod_para_nome[cod] = nome
-------------------------------------------------------------------------------------
 for _ in range(10000):
     placa = gerar_placa()
     modelo = gerar_modelo()
     localizacao = gerar_localizacao()
     dados_10000_aleatorios.append((placa, modelo, localizacao))
-------------------------------------------------------------------------------------
 # Plotar o gráfico
 plt.plot(quantidades, tempos_execucao, marker='o')
 plt.xlabel('Quantidade de Dados')
 plt.ylabel('Tempo de Execução (s)')
 plt.title('Tempo de Execução da Inserção de Dados na Tabela Hash')
 plt.grid(True)
 plt.show()

-------------------------------------------------------------------------------------
 # Ordenar os 10 setores mais congestionados (maiores valores)
top_10_setores = sorted(vetor, key=lambda x: x[2], reverse=True)[:10]
top_10_setores = [(codigo, nome) for codigo, nome, _ in top_10_setores]
-------------------------------------------------------------------------------------
# Adicionar a legenda com os 10 setores mais congestionados
legend_pos = (0.02, 0.02)  # Posição da legenda na parte inferior do gráfico (relativa ao
legend_text = "\n".join([f"{nome}" for _, nome in top_10_setores])
ax.text(legend_pos[0], legend_pos[1],
    f"Top 10 Setores mais Congestionados:\n{legend_text}",
    transform=ax.transAxes, fontsize=12, ha='left', va='top',
    bbox=dict(facecolor='white',
    alpha=0.7, edgecolor='black',
    boxstyle="round,pad=1")
)
```

- Geração de comentários intuitivos no código por inteiro
- Ajustes de nomenclatura e escrita
- Identação e correção de falhas
- Organização e coesão de texto nessa documentação
- Mapa: https://www.youtube.com/watch?v=EZ2Y1_GW0Ew
- Materiais de estudos utilizados durante o semestre

`print("Project finished, now I need COFFEE!")` ☕