ENCI 619.4 - Data Mining for Urban Infrastructure Applications

Winter 2025 – Final Project Term Paper

Analyzing Traffic Incidents and Volume Around Bus-Only Lanes in Calgary Using Data Mining Techniques

Bilal Dawood [a]

Department of Civil Engineering, University of Calgary

Abstract:

This study employs data mining techniques to analyze traffic incidents around bus-only lanes in Calgary to optimize their implementation and enforcement. By integrating transit route data, traffic incidents, traffic volumes, bus lane locations, and GTFS transit data, the research establishes a comprehensive analytical framework for transit priority infrastructure. Two metrics were developed – a Priority Metric quantifying implementation need and a Flow Impedance Metric measuring traffic disruption. Using K-means clustering, the study identified four distinct types of transit corridors with different needs: medium-priority routes with moderate flow impedance (30 routes, 6.7% with bus lanes), low-priority routes with smooth flow (102 routes, 3.9% with bus lanes), high-priority routes with high ridership (16 routes, all with bus lanes), and critical routes with extreme conditions (2 routes, both with bus lanes). Based on these clusters, the research provides route-specific recommendations for bus lane implementation and tailored enforcement strategies. The analysis validates that existing bus lanes in Calgary are well-placed on high-priority corridors while identifying additional routes that would benefit from similar infrastructure. This data-driven approach offers transit planners an evidence-based framework for optimizing bus priority measures, potentially improving service reliability for thousands of transit users.

Introduction:

Public transportation plays a crucial role in urban mobility, providing an essential service that reduces congestion, decreases environmental impact, and improves accessibility. However, the effectiveness of transit systems often depends on their ability to navigate through general traffic efficiently. Bus priority measures, particularly dedicated bus lanes, represent a significant infrastructure investment aimed at enhancing transit service quality and reliability by separating buses from general traffic flow.

Bus-only lanes grant transit vehicles dedicated space within existing road infrastructure, allowing them to bypass congestion and maintain schedules more consistently. As Levinson et al. [1] notes, the level of service provided by buses is directly linked to the type of infrastructure they operate on, with fully separated lanes offering service quality comparable to rail-based systems. In Calgary, as in many North American cities, bus-only lanes have been implemented on select corridors to improve transit performance.

However, the effectiveness of bus lanes is often compromised by unauthorized use, such as private vehicles driving in bus lanes or using them for parking. Studies by Kepaptsoglou et al. [2] and Gavanas et al. [3] found that violations can reduce bus speeds by up to 10% in high-traffic corridors and significantly impact schedule adherence. This creates a management challenge: transit authorities need to balance the costs of implementing and enforcing bus lanes against the potential benefits.

The challenge is further complicated by limited resources. Transit agencies cannot implement bus lanes on every route, nor can they provide intensive enforcement everywhere. This necessitates a strategic approach to both implementation and enforcement. Transit agencies must identify which routes would benefit most from bus lanes and what type of enforcement would be most effective in different contexts.

Previous research has primarily focused on either the physical design of bus lanes [4] or enforcement strategies [5] in isolation. There has been limited work on developing a comprehensive, data-driven framework that integrates transit operations data, traffic conditions, and enforcement considerations to optimize both the placement and protection of bus priority infrastructure.

This study addresses this gap by employing data mining techniques to analyze traffic incidents around bus-only lanes in Calgary. By integrating multiple datasets and applying clustering analysis, we aim to identify patterns that can inform both implementation and enforcement decisions. The research contributes to the field by developing a methodological framework that transit agencies can use to optimize their bus priority networks, potentially improving service for thousands of riders while making more efficient use of limited resources.

Methodology

This study employed a data-driven methodology with 5 components:

- Data collection and preprocessing and integration
- Early stage analysis of data
- Feature engineering
- Unsupervised learning (K-means clustering analysis)
- Geographic visualization
- Recommendation development

This study combines spatial analysis techniques to understand the geographic distribution of incidents and traffic volumes around bus lanes and transit routes.

Data Collection, Preprocessing and Integration:

This study utilized five distinct datasets to conduct a comprehensive analysis of traffic patterns and incidents around bus-only lanes in Calgary. These datasets, sourced from the City of Calgary Open Data portal and Calgary Transit, provide detailed information on transit routes, traffic incidents, traffic volume, and the locations of bus priority infrastructure. A summary of each dataset, including its key variables and data types, is presented in Table 1. Please refer to the appendix for more details on the datasets and their content.

Table 1: Description of utilized dataset

Dataset	Description	Key Variables
Calgary Transit Routes	Geocoded Calgary Transit route information obtained from the City of Calgary Open Data portal. Contains information on various modes of transportation, including LRT, BRT, and regular bus routes. The MULTILINESTRING column represents the geographic location of the route as a sequence of latitude and longitude pairs.	ROUTE_CATEGORY, ROUTE_SHORT_NAME, ROUTE_LONG_NAME, CREATE_DT_UTC, MOD_DT_UTC, GLOBALID, MULTILINESTRING
Calgary Transit GTFS Data (CT-GTFS)	Schedule and service information in the standard General Transit Feed Specification format. Contains information on route sequences, stop arrival and departure times for different routes, times of operation (days of the week and times of day)	trip_id, route_id, stop_id, arrival_time, route_short_name service_id
Calgary Traffic Incidents	An archive of traffic incidents within Calgary updated every 10 minutes, collected using the Advanced Traveler Information System (ATIS). Contains traffic disruptions affecting traffic flow, such as traffic signal issues, hazardous road conditions, stalled vehicles, and unverified traffic collisions. The data is based off unrecorded imagery where traffic cameras are present.	INCIDENT INFO, DESCRIPTION, START_DT, MODIFIED_DT, QUADRANT, Longitude, Latitude, Count, route_id, Point

Dataset	Description	Key Variables
2023 Traffic Volume Calgary	The average number of vehicles in 24 hours, adjusted for seasonal variation to represent the average weekday traffic volume in Calgary. Information is provided by the City of Calgary. The multilinestring column represents the geographic location of the traffic volume collection point as a sequence of latitude and longitude pairs.	YEAR, Collection, Section, Section Name, Volume, Shape Length, multilinestring
Bus Reserved Lanes	Intersections and lengths of bus priority infrastructure within the City of Calgary, provided directly from Calgary Transit. Contains information on the starting and ending intersections of bus-only lanes, along with the direction of traffic flow, lane category, lane type, commission date, operational status, and length of the lane in meters.	ID Number, From, To, Quadrant, Direction, Lane Category, Lane Type, Commission Date, Status, Length m

Data Preprocessing:

Data preprocessing was conducted to transform the raw data into a suitable format for analysis. This involved data cleaning, filtering, and integration of the four datasets. The following steps were performed for each dataset:

Calgary Transit Routes: The Calgary Transit Routes dataset initially contained various transportation modes, including LRT and school routes, which were extraneous to the scope of this analysis. To refine the dataset, routes categorized as 'School' and 'LRT' were removed. This was achieved by manually removing the values in a text editor due to issues with character overflow through Excel.

Calgary Traffic Incidents: The Calgary Traffic Incidents dataset was preprocessed to focus on incidents directly affecting traffic flow. This involved filtering the data based on keywords in the incident descriptions and when the incidents took place (post 2023). For example, the inclusion criteria was applied to descriptions mentioning blocked lanes or vehicle incidents. Exclusion criteria included incidents related to construction, weather, or equipment maintenance, as these factors are often predictable and will give transit planners time to adjust routes beforehand. The filtering process, implemented using a Python script, reduced the dataset from 53,376 to 15,041 relevant incidents with 1779 unique incident descriptions. Part of the Python code used is shown below,

```
def filter_csv(input_file, output_file):
    # Keywords to include - any of these should include the row
    include_keywords = [
        'vehicle', 'vehicles', 'car', 'cars', 'truck', 'trucks',
        'blocked lane', 'blocked lanes', 'lane blocked', 'lanes blocked',
        'blocking', 'blocked', 'traffic', 'collision', 'crash', 'accident', 'breakdown', 'stalled'
    ]
    # Keywords to exclude - any of these should exclude the row
```

```
exclude_keywords = [
    'construction', 'weather', 'equipment', 'maintenance', 'road work', 'roadwork', 'repair', 'rain', 'snow',
     'ice', 'storm', 'hazardous', 'traffic light', 'traffic signal', 'railway'  ]
    …
```

2023 Traffic Volume Calgary: This dataset required cleaning to remove null values. The key variables extracted for analysis were traffic volume and multilinestring coordinates. The cleaned dataset was converted to GeoJSON format for mapping and visualization purposes

Bus Reserved Lanes: The Bus Reserved Lanes dataset required external data collection to supplement the provided information. The dataset was initially filtered to include only 'reserved bus lanes.' Due to incomplete information on the exact start and end points of bus lanes, latitude and longitude coordinates for the 'From' and 'To' intersections were collected using Google Earth. This process involved manual inspection of the data, and geo-coordinates could not be determined for all bus lanes due to data limitations.

CT-GTFS: This dataset presented unique preprocessing challenges as identifiers for different subsets of data were spread across multiple text files. Each text file was first manually inspected and useful column names were extracted, along with IDs that would help match different parameters. Below are a series of steps followed to utilize the CT-GTFS dataset.

1. **Time format handling**: GTFS uses a special time format which extends beyond 24 hours.
2. **Service Calendar Processing**: The calendar.txt file was processed to separate and identify weekday service routes. Routes only operating during the weekend were discarded at this stage.
3. **Trip Frequency calculation:** service frequency was calculated by counting trips per route on a typical weekday. This was further analyzed by matching trips with their times of operation, to separate peak-hour trips (7-9am and 4-6pm) from regular trips.
4. **Headway Calculation:** Peak and non-peak headways were calculated by analyzing the time between consecutive trips (consistent route_id and stop_id, consecutive trip_id)
5. **Ridership Proxy Development**: Since actual ridership data was not available, a proxy was developed based on trip frequency and route importance, with routes operating during peak-hours weighted more heavily than regular routes

Data Integration:

Due to the diverse datasets utilized in this study, and differences in their structures and identifiers, a series of steps took place to develop a unified dataset.

1. **Spatial Reference Standardization**: Before integration, all spatial datasets were converted to be following a common coordinate reference system (EPSG:4326).
2. **Route-Incident Spatial Integration:** In order to asses the traffic incidents along individual transit routes, a 10 meter zone was created around each route geometry, and incidents within this zone were considered to affect the route.
3. **Route-Traffic Volume Integration:** Traffic volume was associated with routes by identifying road segments that overlapped with or ran parallel to transit routes.

4. **Bus Lane Assignment:** The number of bus-only lanes per route was determined by comparing route geometries with the collected locations of bus only lanes.
5. **GTFS Metric Integrations:** The transit operational metrics derived from GTFS data (headway, number of trips per day, peak hour operations) were joined to the spatial dataset using the route_short_name identifier.

By following this approach for data preprocessing and integration, the final dataset contained 150 transit routes with 10 key variables representing different factors for analysis. Each route had complete information on its operational characteristics, traffic conditions, incident exposure, and existing bus-lane infrastructure. This integrated dataset provided the foundation for the subsequent feature engineering, clustering analysis, and recommendation development.

Feature Engineering:

Another aspect of this study involved the development of two metrics to quantify different transit characteristics and ensure normalized values so results are not skewed in clustering.

1. **Priority Metric**: Quantifies how much a route needs a bus lane based on:
    a. Transit demand component (40%): Combines estimated riders and number of trips during peak hour.
    b. Traffic Impedence components (40%): Combines traffic volume and traffic incidents.
    c. Existing Infrastructure Penalty (20%): Reduces priority for busses that already have bus only lanes.
2. **Flow Impedence:** Measures the degree of traffic disruption experienced on a route.
    a. Traffic volume component (40%)
    b. Traffic Incident component (30%)
    c. Incident-to-Volume ratio (30%) – to capture areas which have unusually high amounts of incidents compared to their traffic volume.

Clustering Analysis

After completing the feature engineering phase was completed, and the dataset was verified to be in working condition, unsupervised learning was employed to find hidden patterns. Specifically, K-means clustering was utilized in Python to identify the grouping of transit routes with similar characteristics. The model was trained on the following features:

- Number of bus only lanes on route
- Priority Metric
- Flow Impedence Metric

Prior to clustering, all features were standardized using StandardScaler from the scikit-learn library, ensuring fair comparison of features of different scales.

The optimal number of clusters was determined using a sillhoutte score analysis, which measures how well clusters fit within their assigned clusters. Based on these methods, four clusters were identified as the optimal solution, providing a balance between detail and interpretability.

The K-means algorithm was then applied with k=4 and random_state=42 to ensure reproducibility. The resulting cluster assignments were added to the dataset for analysis and visualization.

Geographic Visualization

To understand and visualize the spatial distribution of clusters and priority areas, interactive maps were created using Folium. These visualizations showed:

1. Transit routes colored by cluster assignment
2. Priority routes for potential bus lane implementation

These maps allowed for the identification of spatial patterns to understand the distribution of the routes and provided an effective way to visualize results.

Please refer to the appendix B for more information about the scripts for the methodology.

Results and Analysis

Analysis Phase 1: Individually Understanding the Databases

Exploratory Data Analysis (EDA) was conducted to gain initial insights into the characteristics of the datasets and identify potential relationships between variables. The analysis was performed using Python libraries such as GeoPandas, Matplotlib, NumPy, Pandas, Shapely, and Folium.

In the visualization below, Calgary Traffic is mapped against transit routes. 326 traffic segments and 151 transit routes were loaded and visualized using the geojson file datasets above. These visualizations also help in confirming the final phase of my analysis.

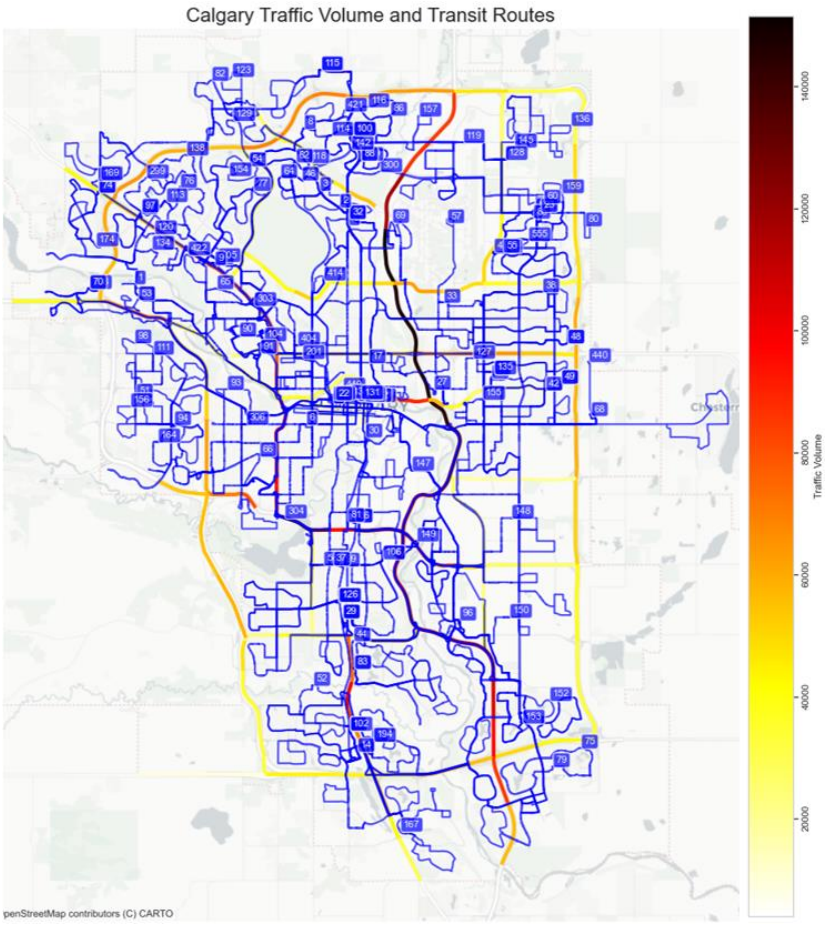


Figure 1: Calgary Traffic Mapped Against Traffic Routes

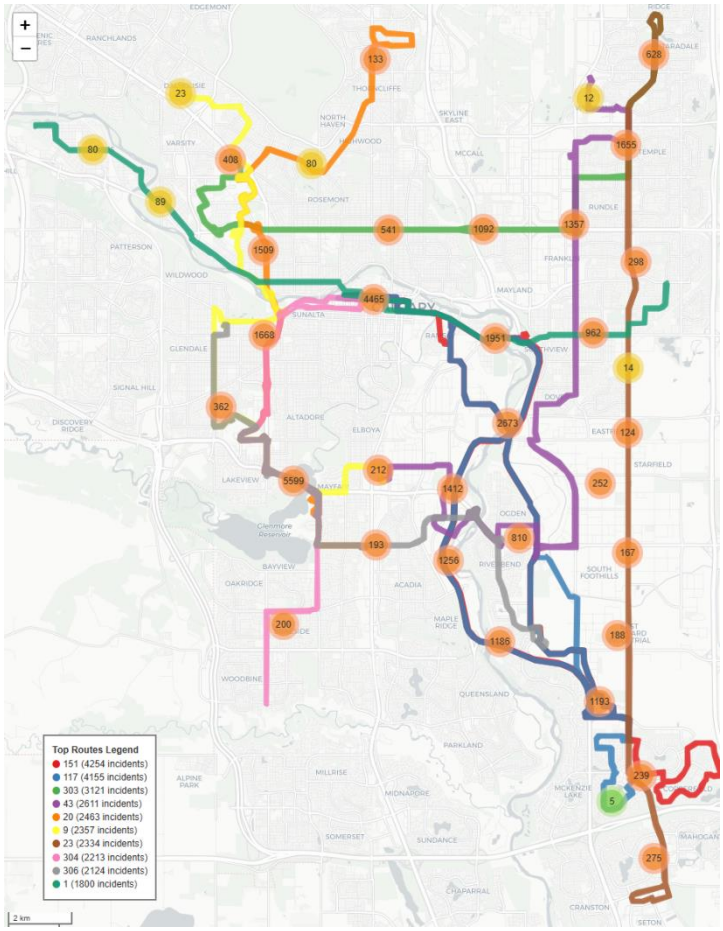


Figure 2: Traffic Incidents by Route Visualization

Manual observation of Figure 1 (left) indicates that routes 106, 151, 825, 845, 36, 81, 306, 66, and 9 experience higher traffic density compared to other routes.

The analysis of Figure 2 (right) shows that routes traveling along Deerfoot trail SE (such as 151, 117 and 1) witness the most traffic incidents, alongside the routes travelling alongside routes travelling on 36 Street NE (303 and 43). These ten routes accounted for a total of 33,311 recorded incidents.

The next phase of the analysis involved integrating the traffic incident, traffic volume and Calgary Transit routes datasets to create a new dataset incorporating traffic volume, bus-only lane presence, and traffic incident data for each route. Two scripts were developed to identify traffic volume and incidents along transit routes, and to identify bus-only lanes for each route. Geospatial data was used to find common corridors. With this new dataset, a more rigorous analysis is done to find direct relationship between and statistics for the routes. At the end it was determined out of 151 routes, 24 utilize bus only lanes.

A comparison of routes with and without bus-only lanes revealed that routes with bus-only lanes tend to pass through higher traffic density areas and experience more traffic incidents (Figures 3 and 4)

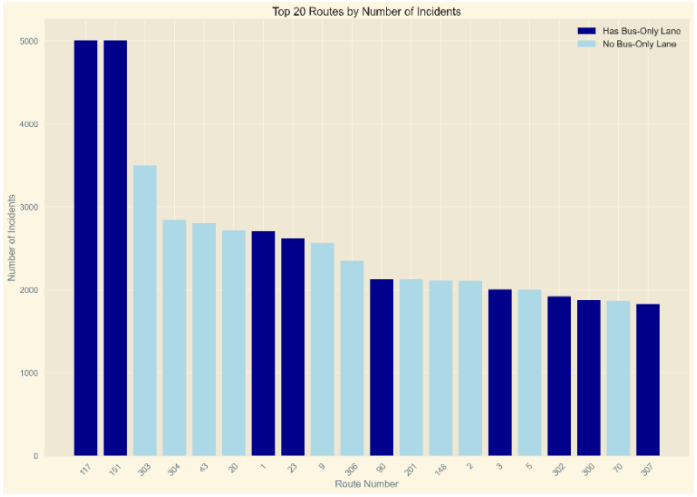


Figure 3: Top 20 Routes by Traffic Incidents

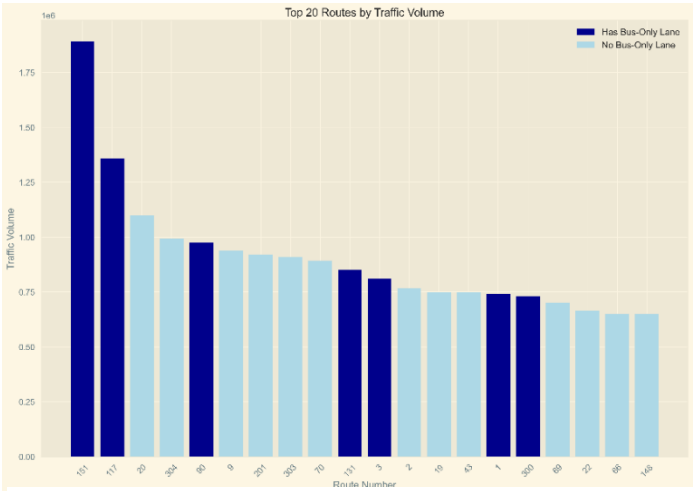


Figure 4: Top 20 Routes by Traffic Volume

The two figures above show the top 20 routes that experience the most traffic incidents and which routes travel through the most volume of traffic. In both cases, the routes that utilize bus only lanes are the ones to travel the most traffic dense areas and would experience the most traffic incidents. By looking at the figures above, it should also be noted that routes with bus only lanes make only between 35% – 45% of the charts above.

A temporal analysis was also conducted to examine the distribution of traffic incidents over time (Figure 5)

The analysis indicated that the highest number of incidents occurred between November and February and during the afternoon hours (12:00 PM to 6:00 PM)

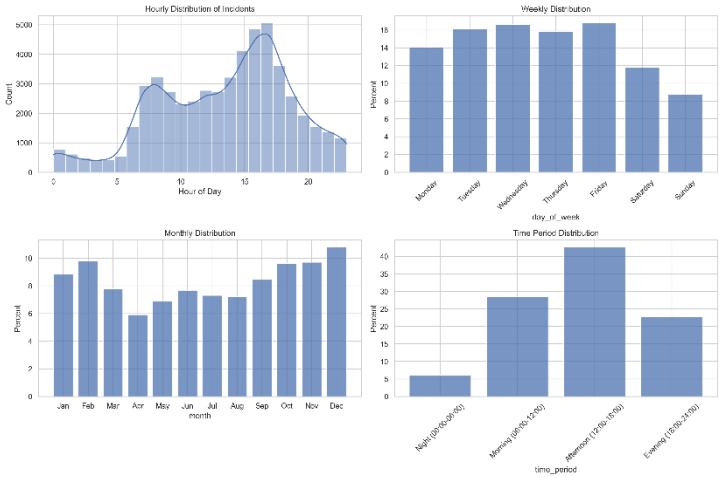


Figure 5: Temporal Distribution of Traffic Violation Dataset

Analysis Phase 2: Clustering Analysis

The K-means clustering analysis revealed four clusters of transit routes in Calgary, each with different characteristics relevant to bus lane implementation and enforcement. The table below summarizes the key statistics for each cluster.

Cluster	Count	Bus Lanes (%)	Daily Trips	Traffic Volume	Incidents	Estimated Riders	Priority	Flow Impedance
0	30	6.7%	379	658,774	1,808	11,964	41.9	25.8
1	102	3.9%	207	111,386	440	6,784	29.1	7.5
2	16	100.0%	275	462,841	1,636	8,916	31.6	22.3
3	2	100.0%	10	1,624,880	5,009	1,884	53.8	64.3

Cluster 0: Medium-High Priority Routes with Moderate Flow Impedance (30 routes) These routes have relatively high priority scores (41.9) and moderate flow impedance (25.8). They experience substantial traffic volumes and incidents, with high ridership. Only 6.7% currently have bus lanes, suggesting they are prime candidates for new implementations.

Cluster 1: Low Priority Routes with Smooth Flow (102 routes) The largest cluster consists of routes with low priority scores (29.1) and very low flow impedance (7.5). These routes experience relatively smooth operations with lower traffic volumes, fewer incidents, and moderate ridership. Only 3.9% have bus lanes, which aligns with their operational characteristics.

Cluster 2: High Priority Routes with High Ridership (16 routes) All routes in this cluster already have bus lanes (100%). They have high priority scores (31.6) with moderate flow impedance (22.3) and notably high ridership (8,916). This suggests that Calgary Transit has effectively identified high-ridership corridors for bus lane implementation. Performance of bus lanes can be enhanced in these areas by utilizing police force or camera-based violation detection to deter violators.

Cluster 3: Critical Routes with Extreme Conditions (2 routes) This small cluster consists of routes with very high priority scores (53.8) and extreme flow impedance (64.3). These routes face extraordinary traffic volumes and incidents, and all have existing bus lanes. The reliability of these routes can be enhanced by adding extra bus lanes and ensuring violations are minimized. This could be by implementing physical barriers to separate bus only lanes or utilizing the police force.

A Priority vs Flow Impedance relationship was then plotted to understand the relationship between clusters and how they relate to the two metrics created (priority metric and flow impedence). Figure 6 shows the distribution of routes according to these metrics, with colors indicating cluster membership.

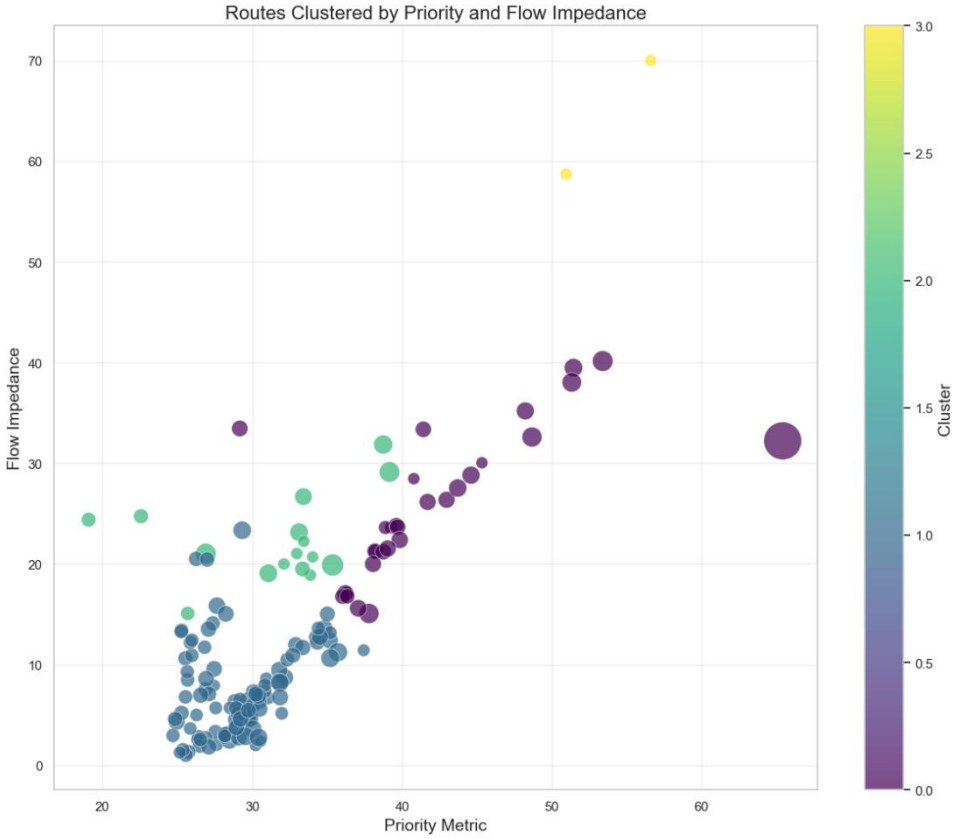


Figure 6: Clustering Result Plot

This visualization demonstrates how the four clusters occupy different regions in the priority-flow space:

- Cluster 1 routes (blue) concentrate in the lower left, representing low priority and low flow impedance
- Cluster 0 routes (purple) spread across the middle left to upper middle regions, showing medium to high priority and moderate to high flow impedance
- Cluster 2 routes (green) occupy the middle region, with moderate to high values on both axes
- Cluster 3 routes (yellow) stand out in the upper right corner with extremely high values on both metric

Finally, a geographical visualization was prepared to spatially understand the relationship between the clusters.

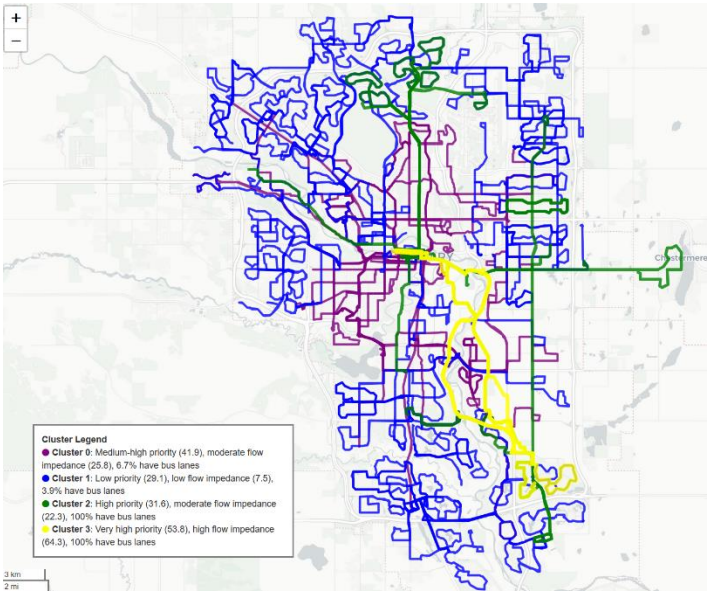
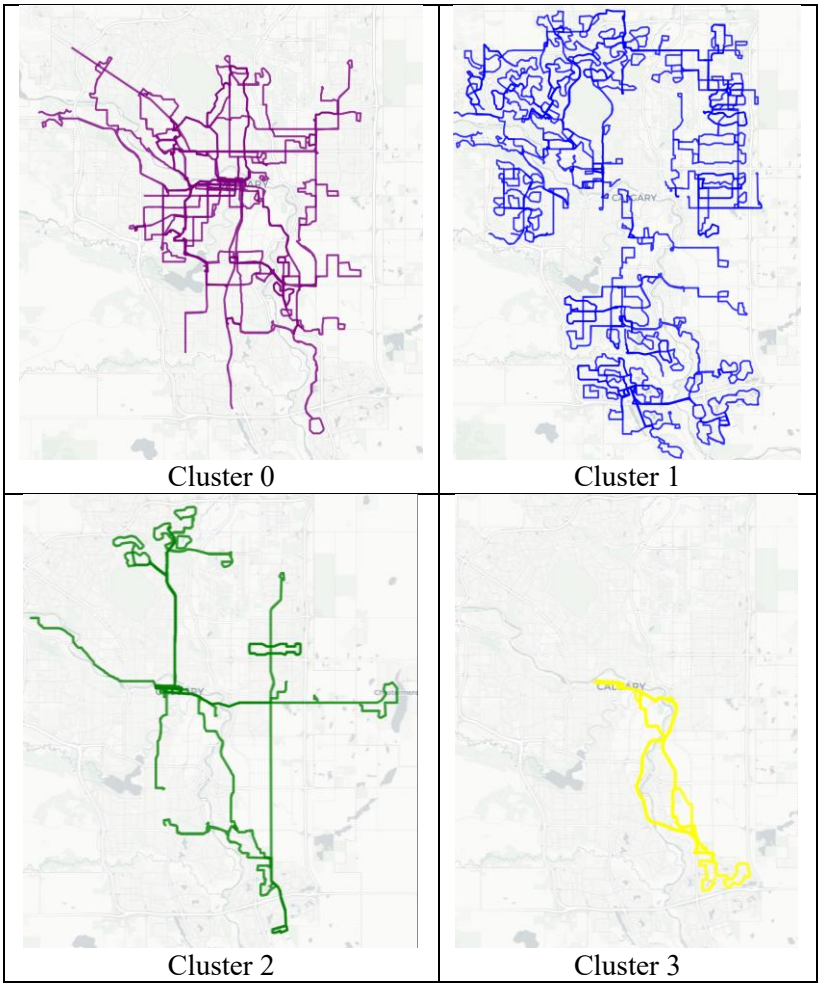


Figure 7: Spatial Distribution of Clusters (All)



The spatial visualization of the routes showed that:

- Cluster 0 routes (medium-high priority, moderate flow) tend to follow major arterial roads and connect important activity centers (downtown Calgary)
- Cluster 1 routes (low priority, low flow) covers most residential areas
- Cluster 2 routes (high priority, all with bus lanes) primarily follow major corridors through the city center or crosses from one quadrant of the city to an opposite quadrant.
- Cluster 3 routes (critical priority, extreme conditions) follow extremely high-volume corridors that connects the south of the city to downtown Calgary.

Discussion

Interpretation of Cluster Findings:

The clustering analysis revealed distinct patterns in Calgary's transit network that have important implications for bus lane implementation and enforcement. The fact that 100% of routes in Clusters 2 and 3 already have bus lanes suggests that Calgary Transit has effectively identified critical corridors for bus priority infrastructure. These include high-ridership routes (Cluster 2) and routes facing extreme traffic conditions (Cluster 3). If performance on these routes is still considered not reliable, transit planners may look into implementing more bus only lanes or adding enforcement on these corridors.

The large number of routes in Cluster 1 (102 routes, 68% of all routes) with low priority scores and flow impedance indicates that most of Calgary's transit network operates in conditions where bus lanes may not be necessary. This finding suggests the city would benefit from targeted implementation rather than system-wide deployment.

The most interesting finding from a planning perspective is the identification of 30 routes in Cluster 0 with medium-high priority scores but only 6.7% bus lane coverage. These routes present the greatest opportunity for expanding bus priority infrastructure to improve transit service. The specific characteristics of these routes (high ridership, substantial traffic volumes, and significant incident counts) suggest they would benefit considerably from dedicated bus lanes.

Limitations and Future Work:

The analysis faced several data limitations that should be considered when interpreting the results:

1. Estimated Ridership: The study relied on estimated ridership derived from GTFS data rather than actual passenger counts. Automated Passenger Counter (APC) data would offer more precise ridership figures.
2. Temporal Matching: There was no temporal matching between incidents and bus operations, making it difficult to determine if certain incidents affected bus trips. Future work could match trip timing information with incident time frames to better understand these interactions.
3. Lack of data on Bus Only Lane violations: Through general traffic incidents, it can not be determined if busses were affected by traffic incidents or volumes. Future studies could utilize bus GPS data or information on bus lane violations to determined if certain corridors are affecting the use of bus only lanes, while also utilizing length of bus only lanes as opposed to a count.

Conclusion:

This study developed a data-driven framework for bus lane implementation and enforcement decisions in Calgary. By integrating multiple datasets and applying clustering analysis, four clusters were identified for different types of transit corridors, each with different needs for bus priority infrastructure and enforcement.

The analysis validated that existing bus lanes in Calgary are generally well-aligned with high-priority and high-impedance routes. All routes in Clusters 2 and 3, which have the highest need based on ridership or traffic conditions, already have bus lanes. This suggests that past implementation decisions have been sound.

The research identified a significant opportunity for expansion of bus priority infrastructure on Cluster 0 routes, which have medium-high priority scores but only 6.7% bus lane coverage.

The study also developed tailored enforcement recommendations based on the characteristics of each cluster:

- Cluster 0: Target enforcement during peak hours at key congestion points
- Cluster 1: Light, intermittent enforcement with a focus on educational campaigns
- Cluster 2: Consistent enforcement to protect high-ridership corridors
- Cluster 3: Intensive enforcement with both cameras and physical barriers

These cluster-specific strategies provide a framework for allocating enforcement resources efficiently across the transit network.

The methodological approach developed in this study (Priority Metric and Flow Impedance Metric) present a quantitative basis for decision-making about bus priority infrastructure.

By providing a comprehensive, data-driven framework for both implementation and enforcement decisions, this research contributes to the optimization of transit priority measures. The application of these findings could lead to improved transit service reliability, reduced delays, and enhanced safety for both transit users and general traffic.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

While preparing this Term Project report, I utilized ChatGPT to assist with organizing content, correcting grammatical errors, enhancing literature flow, expanding on analysis and assisting with development of code. After using ChatGPT, I thoroughly reviewed and edited the content as necessary and took full responsibility for the content of the Term Project report.

References:

[1] H. S. Levinson, S. Zimmerman, J. Clinger, and J. Gast, “Bus Rapid Transit: Synthesis of Case Studies,” Transp. Res. Rec. J. Transp. Res. Board, vol. 1841, no. 1, pp. 1–11, Jan. 2003, doi: 10.3141/1841-01.

[2] K. Kepaptsoglou, D. Pyrialakou, C. Milioti, M. G. Karlafatis, and D. Tsamboulas, “Bus lane violations: an exploration of causes,” Eur. Transp. Trasp. Eur., no. 48, pp. 87–98, 2011.

[3] N. Gavanas, A. Tsakalidis, A. Aggelakakis, and M. Pitsiava-Latinopoulou, “Assessment of bus lane violations in relation to road infrastructure, traffic and land-use features:: The case of Thessaloniki, Greece,” 2013.

[4] L. T. Truong, M. Sarvi, and G. Currie, “Exploring Multiplier Effects Generated by Bus Lane Combinations,” Transp. Res. Rec. J. Transp. Res. Board, vol. 2533, no. 1, pp. 68–77, Jan. 2015, doi: 10.3141/2533-08.

[5] A. Agrawal, T. Goldman, and N. Hannaford, “Shared-Use Bus Priority Lanes on City Streets: Approaches to Access and Enforcement,” J. Public Transp., vol. 16, no. 4, pp. 25–41, Dec. 2013, doi: 10.5038/2375-0901.16.4.2.

Appendix A: Additional Dataset Information

Dataset 1 - Calgary Transit Routes:

Column Name	Description	Sample Inputs
ROUTE_CATEGORY	This column contains information on what mode of transportation the route in this category is.	LRT: Light Rail Transit BRT: Bus Rapid Transit (routes designed to transport passengers over long distances quickly, with few stops.) Regular: Typical Calgary Transit routes Special: routes providing specialized service that do not fall into the previous categories School: Routes designed to serve students attending school
ROUTE_SHORT_NAME	The number identifying the route.	296 items, ranging with values from 1 to 896
ROUTE_LONG_NAME	The full name of the route	Example: Brentwood Station/Temple Dashmesh Centre
CREATE_DT_UTC		2025/03/04
MOD_DT_UTC		2025/03/04
GLOBALID	Row Unique identifier in text format	a07b15bd-7aa8-4664-a058-fae7a7cb7771
MULTILINESTRING	Represents a set of paths on the Earth as sequences of WGS84 Latitude and Longitude pairs. The location is encoded as a GeoJSON "multilinestring".	MULTILINESTRING ((-113.944827 51.092629, -113.943324 51.092629, -113.942686 51.092628), …

Dataset 2: Calgary Traffic Incidents:

Column Name	Description	Sample Inputs	Data Type
INCIDENT INFO	Where did it occur	6 Avenue and 4 Street SW	Text
DESCRIPTION	Description of incident, status and impact	Stalled vehicle. Partially blocking the right lane	Text
START_DT	When was the incident recorded	2022-06-21 7:31	Floating Timestamp
MODIFIED_DT	Last update of incident	2022-06-21 7:33	Floating Timestamp
QUADRANT	Which quadrant of the City did it occur	NE, NW, SW, SE	Text
Longitude		-114.027	Number
Latitude		51.067485129276236	Number
Count	No information	Only Input: 1	Text
id	Incident ID	2022-06-21T07:31:4051.067485129276236-114.02668672232672	Text
Point	Represents a location on the Earth as a WGS84 Latitude and Longitude.	POINT (-114.02668672232672 51.067485129276236)	Point

Dataset 3: 2023 Traffic Volume Calgary

This dataset contains the average number of vehicles in 24 hours. The volume is adjusted for seasonal variation to represent the average weekday traffic volume in Calgary. This information is provided by the city of Calgary

Column Name	**Description**	**Sample Inputs**	**Data Type**
YEAR	Year data was collected	2023	Text
Collection	What type of infrastructure is the collection device on/at	Intersection/Perm/Prov/Segment	Text
Section	Is the collection site within the city or pronincial section	City Section/Provincial Section	Text
Section Name	Where is the volume being recorded	10STW5, 11AVS10	Text
Volume	Recorded traffic volume	4000 - 151520	Number
Shape Length		shape_length	Number
multilinestring	Represents a set of paths on the Earth as sequences of WGS84 Latitude and Longitude pairs. The location is encoded as a GeoJSON “multilinestring”.	MULTILINESTRING ((-114.05833821143568 51.04225127516467, -114.06079540749437 51.04232099143237))	MultiLine

Dataset 4: Bus Reserved Lanes:

This dataset was provided directly from Calgary Transit and contains the intersections and lengths of bus priority infrastructure within the City of Calgary.

Column Name	**Description**	**Sample Inputs**
ID Number		3-23
From	Starting intersection of bus only lane	Edmonton TR
To	Intersection Bus only lane ends at	Mcknight BV
Quadrant	Quadrant of city the lane is in	NE
Direction	Which direction does traffic flow on this lane	SB
Lane Category		Reserved Lane for Buses Bus Shoulder Lane HOV Lane Transitway
Lane Type	Defines extra conditions to lane (if any)	Bus Only Lane (along far side stop) Part Time NB HOV Lane 15:30-18:00 Mon-Fri Part Time Reserved Bus Lane. Mon - Fri: 03:30PM-06:00 PM
Comission Date	When it was implemented	5/1/2015
Status	Current operational status of infrastructure	Current
Length m	length	29-4019m

Appendix B: Python Scripts for Methods Section

Convert csv data files to geojson:

```
# Load CSV file
df = pd.read_csv("Data Preprocessing\Traffic_Volumes_for_2023_20250309.csv")

# Convert MULTILINESTRING (WKT) to geometry
df["geometry"] = df["multilinestring"].apply(loads)

# Create GeoDataFrame
gdf = gpd.GeoDataFrame(df, geometry="geometry", crs="EPSG:4326")

# Save as GeoJSON
gdf.to_file("Traffic-Volume.geojson", driver="GeoJSON")
```

Analysis Phase 2 – Displaying top 20 routes by volume and incidents:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os

# Load the data
file_path = './analysis-phase-2/zTransit_route_analysis.csv'

# Data cleaning and preparation
df['Traffic_Volume'] = pd.to_numeric(df['Traffic_Volume'], errors='coerce')
df['Incidents'] = pd.to_numeric(df['Incidents'], errors='coerce')
df['Number_of_Bus-Only-Lanes'] = pd.to_numeric(df['Number_of_Bus-Only-Lanes'],
errors='coerce')

# Fill any NaN values with 0 (for routes with missing data)
df.fillna(0, inplace=True)

# Create a binary column for routes with bus-only lanes
df['Has_Bus_Lane'] = df['Number_of_Bus-Only-Lanes'] > 0

# Calculate incidents per 100,000 traffic volume (safety metric)
df['Incidents_per_100k'] = df.apply(
    lambda x: (x['Incidents'] / x['Traffic_Volume'] * 100000) if
x['Traffic_Volume'] > 0 else 0,
    axis=1
```

```
)

# Function to set up the plot style
def setup_plot_style():
    plt.style.use('seaborn-v0_8-dark-palette')
    plt.rcParams['figure.figsize'] = (12, 8)
    plt.rcParams['font.size'] = 12
    plt.rcParams['axes.titlesize'] = 16
    plt.rcParams['axes.labelsize'] = 14
    plt.rcParams['xtick.labelsize'] = 12
    plt.rcParams['ytick.labelsize'] = 12

# Create a directory for saving plots
if not os.path.exists('transit_analysis_plots'):
    os.makedirs('transit_analysis_plots')

# Common legend elements
from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='darkblue', label='Has Bus-Only Lane'),
    Patch(facecolor='lightblue', label='No Bus-Only Lane')
]

# VISUALIZATION 5: Top 20 routes by traffic volume
setup_plot_style()
plt.figure(figsize=(14, 10))

# Create a copy of df to avoid warnings about modifying data
top_by_traffic = df.sort_values('Traffic_Volume',
ascending=False).head(20).copy()

# Create the plot manually to ensure correct bar colors
fig, ax = plt.subplots(figsize=(14, 10))
x = np.arange(len(top_by_traffic))
width = 0.8

# Create bars with colors based on Has_Bus_Lane
bars = []
for i, (_, row) in enumerate(top_by_traffic.iterrows()):
    color = 'darkblue' if row['Has_Bus_Lane'] else 'lightblue'
    bar = ax.bar(i, row['Traffic_Volume'], width, color=color)
    bars.append(bar)

# Set x-axis labels
ax.set_xticks(x)
```

```
ax.set_xticklabels(top_by_traffic['Route_Short_Name'], rotation=45)

# Add titles and labels
ax.set_title('Top 20 Routes by Traffic Volume')
ax.set_xlabel('Route Number')
ax.set_ylabel('Traffic Volume')
ax.legend(handles=legend_elements)

plt.tight_layout()
plt.show()
# plt.savefig('transit_analysis_plots/top_routes_by_traffic.png', dpi=300,
bbox_inches='tight')
plt.close()

# VISUALIZATION 6: Top 20 routes by incidents
setup_plot_style()
plt.figure(figsize=(14, 10))

# Create a copy of df to avoid warnings about modifying data
top_by_incidents = df.sort_values('Incidents', ascending=False).head(20).copy()

# Create the plot manually to ensure correct bar colors
fig, ax = plt.subplots(figsize=(14, 10))
x = np.arange(len(top_by_incidents))
width = 0.8

# Create bars with colors based on Has_Bus_Lane
bars = []
for i, (_, row) in enumerate(top_by_incidents.iterrows()):
    color = 'darkblue' if row['Has_Bus_Lane'] else 'lightblue'
    bar = ax.bar(i, row['Incidents'], width, color=color)
    bars.append(bar)

# Set x-axis labels
ax.set_xticks(x)
ax.set_xticklabels(top_by_incidents['Route_Short_Name'], rotation=45)

# Add titles and labels
ax.set_title('Top 20 Routes by Number of Incidents')
ax.set_xlabel('Route Number')
ax.set_ylabel('Number of Incidents')
ax.legend(handles=legend_elements)

plt.tight_layout()
plt.show()
```

```
# plt.savefig('transit_analysis_plots/top_routes_by_incidents.png', dpi=300,
bbox_inches='tight')
plt.close()

# VISUALIZATION 7: Top 20 routes by incident rate
setup_plot_style()
plt.figure(figsize=(14, 10))

# Filter out routes with zero traffic volume to avoid division by zero issues
df_nonzero_traffic = df[df['Traffic_Volume'] > 0].copy()
top_by_incident_rate = df_nonzero_traffic.sort_values('Incidents_per_100k',
ascending=False).head(20).copy()

# Create the plot manually to ensure correct bar colors
fig, ax = plt.subplots(figsize=(14, 10))
x = np.arange(len(top_by_incident_rate))
width = 0.8

# Create bars with colors based on Has_Bus_Lane
bars = []
for i, (_, row) in enumerate(top_by_incident_rate.iterrows()):
    color = 'darkblue' if row['Has_Bus_Lane'] else 'lightblue'
    bar = ax.bar(i, row['Incidents_per_100k'], width, color=color)
    bars.append(bar)

# Set x-axis labels
ax.set_xticks(x)
ax.set_xticklabels(top_by_incident_rate['Route_Short_Name'], rotation=45)

# Add titles and labels
ax.set_title('Top 20 Routes by Incident Rate (Incidents per 100k Traffic
Volume)')
ax.set_xlabel('Route Number')
ax.set_ylabel('Incidents per 100k Traffic Volume')
ax.legend(handles=legend_elements)

plt.tight_layout()
plt.show()
# plt.savefig('transit_analysis_plots/top_routes_by_incident_rate.png', dpi=300,
bbox_inches='tight')
plt.close()

# Original pie chart visualization (keeping for reference)
setup_plot_style()
plt.figure(figsize=(10, 10))
```

```
bus_lane_counts = df['Number_of_Bus-Only-Lanes'].value_counts().sort_index()
plt.pie(bus_lane_counts, labels=[f'{i} Bus Lane(s): {count} routes' for i, count
in enumerate(bus_lane_counts)],
        autopct='%1.1f%%', startangle=90, colors=sns.color_palette('Blues',
len(bus_lane_counts)))
plt.title('Distribution of Routes by Number of Bus-Only Lanes')
plt.savefig('transit_analysis_plots/bus_lane_distribution.png', dpi=300,
bbox_inches='tight')
plt.close()

# ANALYSIS FOR RECOMMENDATIONS: Finding routes that would benefit most from bus-
only lanes
# Normalize values for fair comparison
df['Traffic_Volume_Norm'] = df['Traffic_Volume'] / df['Traffic_Volume'].max()
df['Incidents_Norm'] = df['Incidents'] / df['Incidents'].max()
df['Incidents_Rate_Norm'] = df['Incidents_per_100k'] /
df['Incidents_per_100k'].max()

# Create recommendation score (only for routes without bus lanes)
df_no_bus_lanes = df[df['Number_of_Bus-Only-Lanes'] == 0].copy()
df_no_bus_lanes['Recommendation_Score'] = (
    df_no_bus_lanes['Traffic_Volume_Norm'] * 0.4 +
    df_no_bus_lanes['Incidents_Norm'] * 0.3 +
    df_no_bus_lanes['Incidents_Rate_Norm'] * 0.3
)

# VISUALIZATION 9: Top 15 routes recommended for bus-only lanes
setup_plot_style()
plt.figure(figsize=(16, 12))
top_recommendations = df_no_bus_lanes.sort_values('Recommendation_Score',
ascending=False).head(15).copy()

# Create a horizontal bar chart manually
fig, ax = plt.subplots(figsize=(16, 12))
y = np.arange(len(top_recommendations))
width = 0.8

# Create bars with gradient colors
bars = []
for i, (_, row) in enumerate(top_recommendations.iterrows()):
    color = plt.cm.Blues(0.2 + row['Recommendation_Score'] * 1.1)
    bar = ax.barh(i, row['Recommendation_Score'], width, color=color)
    bars.append(bar)

# Set y-axis labels
```

```
ax.set_yticks(y)
ax.set_yticklabels(top_recommendations['Route_Short_Name'])

# Add titles and labels
ax.set_title('Top 15 Routes Recommended for Bus-Only Lanes')
ax.set_xlabel('Recommendation Score')
ax.set_ylabel('Route Number')

# Add route names as annotations with better formatting
for i, route in enumerate(top_recommendations.itertuples()):
    # Add route name after the route number
    ax.text(0.01, i, f"{route.Route_Long_Name}",
            ha='left', va='center', color='black', fontsize=11)

    # Add metrics at the end of each bar
    ax.text((route.Recommendation_Score-0.01), i,
            f"Traffic: {route.Traffic_Volume:,.0f} | Incidents: {route.Incidents}
| Rate: {route.Incidents_per_100k:.1f}",
            ha='right', va='center', fontsize=11)

plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
# plt.savefig('transit_analysis_plots/top_recommendations.png', dpi=300,
bbox_inches='tight')
plt.close()

# Generate a detailed recommendation report
top_recommendations_df = top_recommendations[['Route_Short_Name',
'Route_Long_Name',
                                              'Traffic_Volume', 'Incidents',
                                              'Incidents_per_100k',
'Recommendation_Score']].copy()

# Format and save the recommendation report as CSV
top_recommendations_df.to_csv('transit_analysis_plots/bus_lane_recommendations.cs
v', index=False)

# Print some summary statistics
print("\n=== TRANSIT ROUTE ANALYSIS SUMMARY ===")
print(f"Total routes analyzed: {len(df)}")
print(f"Routes with bus-only lanes: {len(df[df['Has_Bus_Lane']])}")
print(f"Routes without bus-only lanes: {len(df[~df['Has_Bus_Lane']])}")
print("\nAverage metrics for routes WITH bus-only lanes:")
print(f"- Traffic Volume: {df[df['Has_Bus_Lane']]['Traffic_Volume'].mean():.1f}")
```

```
print(f"- Incidents: {df[df['Has_Bus_Lane']]['Incidents'].mean():.1f}")
print(f"- Incidents per 100k Traffic:
{df[df['Has_Bus_Lane']]['Incidents_per_100k'].mean():.2f}")
print("\nAverage metrics for routes WITHOUT bus-only lanes:")
print(f"- Traffic Volume:
{df[~df['Has_Bus_Lane']]['Traffic_Volume'].mean():.1f}")
print(f"- Incidents: {df[~df['Has_Bus_Lane']]['Incidents'].mean():.1f}")
print(f"- Incidents per 100k Traffic:
{df[~df['Has_Bus_Lane']]['Incidents_per_100k'].mean():.2f}")
print("\nTop 5 routes recommended for new bus-only lanes:")
for i, route in enumerate(top_recommendations.head().itertuples(), 1):
    print(f"{i}. Route {route.Route_Short_Name} ({route.Route_Long_Name})")
    print(f"   Traffic Volume: {route.Traffic_Volume:.0f}")
    print(f"   Incidents: {route.Incidents}")
    print(f"   Incidents per 100k: {route.Incidents_per_100k:.2f}")
    print(f"   Recommendation Score: {route.Recommendation_Score:.3f}")
# print("\nAnalysis complete! All visualizations saved to the
'transit_analysis_plots'")
print("\n Analysis compleete")
```

Calculating Peak Trips and Peak hour headway per route:

```
import pandas as pd
import numpy as np


# Step 1: Load the necessary GTFS data
def load_gtfs_data():
    # Load the required GTFS files
    stop_times_df = pd.read_csv('CT_GTFS/stop_times.txt')
    trips_df = pd.read_csv('CT_GTFS/trips.txt')
    routes_df = pd.read_csv('CT_GTFS/routes.txt')

    return stop_times_df, trips_df, routes_df

# Step 2: Process time format correctly
def process_time_format(stop_times_df):
    # Convert GTFS time format (HH:MM:SS) to minutes since midnight
    def time_to_minutes(time_str):
        try:
            h, m, s = map(int, time_str.split(':'))
            return h * 60 + m
        except:
            return np.nan
```

```
    # Apply the conversion
    stop_times_df['arrival_minutes'] =
stop_times_df['arrival_time'].apply(time_to_minutes)

    # Display sample after time conversion
    print("\nSample after time conversion:")
    print(stop_times_df[['trip_id', 'arrival_time', 'arrival_minutes']].head(5))

    return stop_times_df

# Step 3: Calculate headways following your Excel approach
def calculate_headways(stop_times_df, trips_df, routes_df):
    # Join stop_times with trips to get route information

    # trips_with_service = pd.merge(trips_df, calendar_df, on='service_id')
    # weekday_trips = trips_with_service[trips_with_service['monday'] == 1]

    stop_times_with_routes = pd.merge(
        stop_times_df,
        trips_df[['trip_id', 'route_id', 'direction_id']],
        on='trip_id',
        how='left'
    )

    # Display sample after joining with trips
    print("\nSample after joining with trips:")
    print(stop_times_with_routes[['trip_id', 'route_id', 'direction_id',
'arrival_minutes']].head(5))

    # Join with routes to get route short names
    stop_times_with_routes = pd.merge(
        stop_times_with_routes,
        routes_df[['route_id', 'route_short_name']],
        on='route_id',
        how='left'
    )

    # Display sample after joining with routes
    print("\nSample after joining with routes:")
    print(stop_times_with_routes[['trip_id', 'route_id', 'route_short_name',
'direction_id', 'arrival_minutes']].head(5))

    # Filter for first stop in each trip's sequence (stop_sequence == 1)
    first_stops = stop_times_with_routes[stop_times_with_routes['stop_sequence']
== 1].copy()
```

```
    # Display first stops sample
    print("\nSample of first stops (stop_sequence == 1):")
    print(first_stops[['trip_id', 'route_id', 'route_short_name', 'direction_id',
'arrival_minutes']].head(5))
    print(f"\nNumber of first stops: {len(first_stops)}")

    # Define peak periods: Morning peak (7-9 AM) = 420-540 minutes
    morning_peak_start, morning_peak_end, afternoon_peak_start,
afternoon_peak_end = 7*60, 9*60, 16*60, 18*60

    # Create peak/off-peak indicators
    first_stops['is_peak'] = (
        (first_stops['arrival_minutes'] >= morning_peak_start) &
        (first_stops['arrival_minutes'] < morning_peak_end) |
        (first_stops['arrival_minutes'] >= afternoon_peak_start) &
        (first_stops['arrival_minutes'] < afternoon_peak_end)
    )

    # Display sample with peak indicators
    print("\nSample with peak indicators:")
    print(first_stops[['trip_id', 'route_short_name', 'direction_id',
'arrival_minutes', 'is_peak']].head(10))

    # Show count of peak vs non-peak trips
    peak_count = first_stops['is_peak'].sum()
    print(f"\nTotal trips: {len(first_stops)}")
    print(f"Peak trips (7-9 AM): {peak_count}")
    print(f"Non-peak trips: {len(first_stops) - peak_count}")

    # Process each route and direction
    results = []

    # Get unique route_id, direction_id combinations
    route_directions = first_stops[['route_id', 'route_short_name',
'direction_id']].drop_duplicates()
    print(f"\nUnique route-direction combinations: {len(route_directions)}")

    # Process a sample route to demonstrate the headway calculation
    sample_route = route_directions.iloc[0]
    sample_route_id = sample_route['route_id']
    sample_route_name = sample_route['route_short_name']
    sample_direction = sample_route['direction_id']
```

```
    print(f"\nDemonstrating headway calculation for Route {sample_route_name},
Direction {sample_direction}:")

    # Filter for this route and direction
    sample_route_data = first_stops[
        (first_stops['route_id'] == sample_route_id) &
        (first_stops['direction_id'] == sample_direction)
    ].sort_values('arrival_minutes')

    print("\nSample route data (sorted by arrival_minutes):")
    print(sample_route_data.head(10))

    # Calculate headways for sample route
    sample_route_data['next_arrival'] =
sample_route_data['arrival_minutes'].shift(-1)
    sample_route_data['headway'] = sample_route_data['next_arrival'] -
sample_route_data['arrival_minutes']

    print("\nSample route data with headway calculations:")
    print(sample_route_data[['trip_id', 'arrival_minutes', 'next_arrival',
'headway', 'is_peak']].head(10))

    # Calculate peak and non-peak headways for sample route
    peak_sample = sample_route_data[sample_route_data['is_peak']]
    non_peak_sample = sample_route_data[~sample_route_data['is_peak']]

    valid_peak_headways = peak_sample['headway'].dropna()
    valid_peak_headways = valid_peak_headways[(valid_peak_headways > 0) &
(valid_peak_headways < 120)]

    valid_non_peak_headways = non_peak_sample['headway'].dropna()
    valid_non_peak_headways = valid_non_peak_headways[(valid_non_peak_headways >
0) & (valid_non_peak_headways < 120)]

    print(f"\nPeak trips for sample route: {len(peak_sample)}")
    print(f"Valid peak headway measurements: {len(valid_peak_headways)}")
    if len(valid_peak_headways) > 0:
        print(f"Average peak headway: {valid_peak_headways.mean():.2f} minutes")

    print(f"\nNon-peak trips for sample route: {len(non_peak_sample)}")
    print(f"Valid non-peak headway measurements: {len(valid_non_peak_headways)}")
    if len(valid_non_peak_headways) > 0:
        print(f"Average non-peak headway: {valid_non_peak_headways.mean():.2f}
minutes")
```

```
    # Now process all routes and directions
    for _, row in route_directions.iterrows():
        route_id = row['route_id']
        route_short_name = row['route_short_name']
        direction_id = row['direction_id']

        # Filter for this route and direction
        route_data = first_stops[
            (first_stops['route_id'] == route_id) &
            (first_stops['direction_id'] == direction_id)
        ].sort_values('arrival_minutes')

        # Count peak trips
        peak_trips = route_data[route_data['is_peak']].shape[0]

        # Calculate peak headways (if there are enough trips)
        peak_data = route_data[route_data['is_peak']].copy()

        if len(peak_data) > 1:
            # Calculate headways using the next arrival approach
            peak_data['next_arrival'] = peak_data['arrival_minutes'].shift(-1)
            peak_data['headway'] = peak_data['next_arrival'] -
peak_data['arrival_minutes']

            # Filter out missing values and unreasonable headways
            valid_peak_headways = peak_data['headway'].dropna()
            valid_peak_headways = valid_peak_headways[(valid_peak_headways > 0) &
(valid_peak_headways < 120)]

            if len(valid_peak_headways) > 0:
                avg_peak_headway = valid_peak_headways.mean()
            else:
                # Fallback to the original formula if we can't calculate direct
headways
                avg_peak_headway = 120 / peak_trips if peak_trips > 0 else np.nan
        else:
            # Fallback to the original formula if we can't calculate direct
headways
            avg_peak_headway = 120 / peak_trips if peak_trips > 0 else np.nan

        # Calculate non-peak headways
        non_peak_data = route_data[~route_data['is_peak']].copy()

        if len(non_peak_data) > 1:
            # Calculate headways using the next arrival approach
```

```
            non_peak_data['next_arrival'] =
non_peak_data['arrival_minutes'].shift(-1)
            non_peak_data['headway'] = non_peak_data['next_arrival'] -
non_peak_data['arrival_minutes']

            # Filter out missing values and unreasonable headways
            valid_non_peak_headways = non_peak_data['headway'].dropna()
            valid_non_peak_headways =
valid_non_peak_headways[(valid_non_peak_headways > 0) & (valid_non_peak_headways
< 120)]

            if len(valid_non_peak_headways) > 0:
                avg_non_peak_headway = valid_non_peak_headways.mean()
            else:
                avg_non_peak_headway = np.nan
        else:
            avg_non_peak_headway = np.nan

        # Store results
        results.append({
            'route_id': route_id,
            'route_short_name': route_short_name,
            'direction_id': direction_id,
            'peak_trips': peak_trips,
            'peak_headway_minutes': avg_peak_headway,
            'non_peak_headway_minutes': avg_non_peak_headway
        })

    # Convert to DataFrame
    results_df = pd.DataFrame(results)

    # Display sample of detailed results
    print("\nSample of detailed results (by route and direction):")
    print(results_df.head(10))

    # Calculate statistics on the detailed results
    print("\nStatistics for peak_headway_minutes:")
    print(results_df['peak_headway_minutes'].describe())

    # For backward compatibility, create a version with just route_id and
required columns
    peak_trips_per_route = results_df.groupby('route_id').agg({
        'peak_trips': 'sum',  # Sum trips across both directions
        'peak_headway_minutes': 'mean'  # Average headways across both directions
    }).reset_index()
```

```
    # Display the backward-compatible format
    print("\nSample of backward-compatible format:")
    print(peak_trips_per_route.head(10))

    return results_df, peak_trips_per_route



# Load data
stop_times_df, trips_df, routes_df = load_gtfs_data()

# Process time format
stop_times_df = process_time_format(stop_times_df)

# Calculate headways
detailed_results, peak_trips_per_route = calculate_headways(stop_times_df,
trips_df, routes_df)

print("\nFINAL OUTPUTS:")
print("\nDetailed results by route and direction:")
print(detailed_results.head())

print("\nPeak trips per route (compatible format for your existing code):")
print(peak_trips_per_route.head())
```

Proxy Ridership Estimation:

```
daily_trips = trips_df.groupby('route_id').size().reset_index(name='daily_trips')

# Merge with peak trips information from detailed_results
# We'll sum peak_trips across directions for each route
peak_trips_summary =
detailed_results.groupby('route_id')['peak_trips'].sum().reset_index()

# Calculate non-peak trips as daily_trips - peak_trips
trips_summary = pd.merge(daily_trips, peak_trips_summary, on='route_id',
how='left')
trips_summary['non_peak_trips'] = trips_summary['daily_trips'] -
trips_summary['peak_trips']

# Count stops per route
stop_times_with_routes = pd.merge(
    stop_times_df,
    trips_df[['trip_id', 'route_id']],
    on='trip_id'
)
stops_per_route =
stop_times_with_routes.groupby('route_id')['stop_id'].nunique().reset_index(name=
'stop_count')

ridership_proxy = pd.merge(trips_summary, stops_per_route, on='route_id',
how='left')
ridership_proxy = pd.merge(ridership_proxy, first_last_times[['service_hours']],
on='route_id', how='left')

# Add route information for easier joining later
routes_df = pd.read_csv('CT_GTFS/routes.txt')
ridership_proxy = pd.merge(
    ridership_proxy,
    routes_df[['route_id', 'route_short_name', 'route_long_name']],
    on='route_id',
    how='left'
)

# 4. Calculate  ridership proxy with separate weights for peak and non-peak trips
# This gives more weight to peak trips since they often represent higher
ridership
ridership_proxy['estimated_riders'] = (
    ridership_proxy['peak_trips'] * 0.5 +           # Higher weight for peak
trips
```

```
    ridership_proxy['non_peak_trips'] * 0.2 +       # Lower weight for non-peak
trips
    ridership_proxy['stop_count'] * 0.2 +           # Same weight for stop
coverage
    ridership_proxy['service_hours'] * 0.1          # Same weight for service
span
) * 100  # Scale factor
```

Priority Score and Flow Impedence:

```
# 1. Priority Metric - need for bus lane implementation
df['priority_metric'] = (
    # Transit demand component (40%)
    (0.4 * (
        (df['estimated_riders'] / df['estimated_riders'].max()) * 0.6 +
        (df['peak_trips'] / df['daily_trips']) * 0.4
    )) +

    # Traffic impedance component (40%)
    (0.4 * (
        (df['Traffic_Volume'] / df['Traffic_Volume'].max()) * 0.5 +
        (df['Incidents'] / df['Incidents'].max()) * 0.5
    )) +

    # Existing infrastructure penalty (20%)
    # Routes with more bus lanes get lower priority
    (0.2 * (1 - (df['Number_of_Bus-Only-Lanes'] / (df['Number_of_Bus-Only-
Lanes'].max() + 1))))
) * 100

# Check for any NaN or infinite values in priority_metric
if df['priority_metric'].isna().any() or np.isinf(df['priority_metric']).any():
    print("\nDetected NaN or infinite values in priority_metric. Fixing...")
    df['priority_metric'] = df['priority_metric'].replace([np.inf, -np.inf],
np.nan)
    df['priority_metric'] =
df['priority_metric'].fillna(df['priority_metric'].median())

# 2. Flow Impedance Metric - traffic flow disruption
# Calculate incidents per traffic volume (higher ratio = worse flow)
# Avoid division by zero by adding a small constant
df['incident_to_volume_ratio'] = df['Incidents'] / (df['Traffic_Volume'] + 1e-10)

# Calculate volume per incident (higher ratio = better flow)
df['volume_to_incident_ratio'] = df['Traffic_Volume'] / (df['Incidents'] + 1)

# Normalize these ratios
max_inc_ratio = df['incident_to_volume_ratio'].max()
max_vol_ratio = df['volume_to_incident_ratio'].max()

df['normalized_incident_ratio'] = df['incident_to_volume_ratio'] / max_inc_ratio
if max_inc_ratio > 0 else 0
```

```
df['normalized_volume_ratio'] = df['volume_to_incident_ratio'] / max_vol_ratio if
max_vol_ratio > 0 else 0

# Flow impedance metric (higher = worse flow)
df['flow_impedance'] = (
    # Base component from traffic volume
    (df['Traffic_Volume'] / df['Traffic_Volume'].max()) * 0.4 +

    # Incident component
    (df['Incidents'] / df['Incidents'].max()) * 0.3 +

    # Interactive component - incidents per volume
    (df['normalized_incident_ratio']) * 0.3
) * 100
```