ACE Intelligence System - Focused Analysis

In [ ]:
```python
# setting up our workshop
import pandas as pd
import geopandas as gpd
import os
import glob

# -- Part 1: Loading and Preparing the Core Violation Data --

# defining the path to our main dataset
# making sure our code knows where to find the files
DATA_DIR = '../data/'
RAW_DATA_PATH = os.path.join(DATA_DIR, 'raw', 'MTA_Bus_Automated_Camera_Enforc
GTFS_DIR = os.path.join(DATA_DIR, 'raw', 'gtfs')

print('loading the main violations dataset...')
# reading in the massive violations file
# we're telling pandas to treat the datetime columns as actual dates right awa
violations_df = pd.read_csv(
    RAW_DATA_PATH,
    parse_dates=['First Occurrence', 'Last Occurrence'],
    low_memory=False
)

# cleaning up the column names for easier access
# replacing spaces with underscores and making everything lowercase
violations_df.columns = violations_df.columns.str.lower().str.replace(' ', '_'

print(f'loaded {len(violations_df):,} violations.')
print('initial data preparation complete.')
```

```
loading the main violations dataset...
C:\Users\mabdulsamad_\AppData\Local\Temp\ipykernel_32868\1654665911.py:18: Use
rWarning: Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-expected, p
lease specify a format.
  violations_df = pd.read_csv(
C:\Users\mabdulsamad_\AppData\Local\Temp\ipykernel_32868\1654665911.py:18: Use
rWarning: Could not infer format, so each element will be parsed individually,
falling back to `dateutil`. To ensure parsing is consistent and as-expected, p
lease specify a format.
  violations_df = pd.read_csv(
loaded 3,778,568 violations.
initial data preparation complete.
```

In [5]:
```python
# -- Part 2: Consolidating All Bus Stop Locations from GTFS --

# getting a list of all the 'stops.txt' files from each borough's gtfs folder
all_stops_files = glob.glob(os.path.join(GTFS_DIR, '*', 'stops.txt'))

# creating a list to hold the data from each stops file
stops_list = []

print(f'found {len(all_stops_files)} gtfs stops files to process.')
```

```python
# looping through each file and reading its data into our list
for f in all_stops_files:
    stops_list.append(pd.read_csv(f))

# combining all the separate stops dataframes into one single, master datafram
all_stops_df = pd.concat(stops_list, ignore_index=True)

# removing any duplicate stops to keep our data clean
all_stops_df = all_stops_df.drop_duplicates(subset='stop_id')

# converting our pandas dataframe into a geodataframe
# this lets us perform powerful spatial operations, like finding stops near vi
# we're telling geopandas that the 'stop_lon' and 'stop_lat' columns represent
stops_gdf = gpd.GeoDataFrame(
    all_stops_df,
    geometry=gpd.points_from_xy(all_stops_df.stop_lon, all_stops_df.stop_lat),
    crs="EPSG:4326" # this is the standard coordinate system for lat/lon
)

print(f'consolidated {len(stops_gdf):,} unique bus stops across all boroughs.'
print('bus stop geo-dataframe created successfully.')

# displaying the first few rows of our prepared data to make sure everything l
print("\nViolations Data Head:")
display(violations_df.head(3))

print("\nAll Stops GeoDataFrame Head:")
display(stops_gdf.head(3))
```

```
found 6 gtfs stops files to process.
consolidated 13,486 unique bus stops across all boroughs.
bus stop geo-dataframe created successfully.

Violations Data Head:
```

	violation_id	vehicle_id	first_occurrence	last_occ
0	489749182	c5ae1411153b52556a1e648cc80d718aa519a4bdd189ab...	2025-08-20 23:12:08	202 0
1	489744714	df9044acf85cf55488aea4cd3ce1d0e17ef050551726b6...	2025-08-20 23:48:59	202 2
2	489743631	eb5a337966ba65f66ab1db8e169d2446a4fb429b0efc63...	2025-08-20 22:33:13	202 2

```
All Stops GeoDataFrame Head:
```

	stop_id	stop_name	stop_desc	stop_lat	stop_lon	zone_id	stop_url	location_type	p
0	100014	BEDFORD PK BLVD/GRAND CONCOURSE	NaN	40.872562	-73.888156	NaN	NaN	0.0	
1	100017	PAUL AV/W 205 ST	NaN	40.876836	-73.889710	NaN	NaN	0.0	
2	100018	PAUL AV/WEST MOSHOLU PKWY	NaN	40.880392	-73.886081	NaN	NaN	0.0	

In [6]:

```
# -- Part 3: Spatially Connecting Violations to Bus Stops --

# creating a geodataframe for the violations, just like we did for the stops
# this will allow us to compare the locations of violations and stops
print('converting violations to a geo-dataframe...')
violations_gdf = gpd.GeoDataFrame(
    violations_df,
    geometry=gpd.points_from_xy(violations_df.violation_longitude, violations_
    crs="EPSG:4326"
)

# performing the spatial join
# this is the magic step. for each violation, it finds the single nearest bus s
# this might take a few minutes to run because it's a heavy computation.
print('performing the nearest-neighbor spatial join... this may take a moment.
violations_with_stops_gdf = gpd.sjoin_nearest(violations_gdf, stops_gdf, how='

# dropping unnecessary columns to keep our dataframe clean and efficient
# the 'index_right' column is a leftover from the join that we don't need
violations_with_stops_gdf = violations_with_stops_gdf.drop(columns=['index_rig

# saving our enriched data to a parquet file
# parquet is a much faster and more efficient file format than csv.
# saving our progress now means we won't have to re-run the slow spatial join
PROCESSED_DATA_PATH = os.path.join(DATA_DIR, 'processed', 'violations_with_sto
violations_with_stops_gdf.to_parquet(PROCESSED_DATA_PATH)

print(f'spatial join complete. enriched data saved to: {PROCESSED_DATA_PATH}')
print(f'we now have {len(violations_with_stops_gdf):,} violations, each linked

# showing the result of our work
# notice the new columns from the stops data ('stop_id', 'stop_name', etc.) ar
print("\nEnriched Violations GeoDataFrame Head:")
display(violations_with_stops_gdf.head(3))
```

```
converting violations to a geo-dataframe...
performing the nearest-neighbor spatial join... this may take a moment.
```

```
c:\Users\mabdulsamad_\anaconda3\Lib\site-packages\geopandas\array.py:408: User
Warning: Geometry is in a geographic CRS. Results from 'sjoin_nearest' are lik
ely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projecte
d CRS before this operation.

  warnings.warn(
```

```
spatial join complete. enriched data saved to: ../data/processed\violations_wi
th_stops.parquet
we now have 3,779,110 violations, each linked to its closest bus stop.

Enriched Violations GeoDataFrame Head:
```

	violation_id	vehicle_id	first_occurrence	last_occ
0	489749182	c5ae1411153b52556a1e648cc80d718aa519a4bdd189ab...	2025-08-20 23:12:08	202 0
1	489744714	df9044acf85cf55488aea4cd3ce1d0e17ef050551726b6...	2025-08-20 23:48:59	202 2
2	489743631	eb5a337966ba65f66ab1db8e169d2446a4fb429b0efc63...	2025-08-20 22:33:13	202 2

In [8]:

```python
# -- Part 4: Isolating Chronic "Exempt" Offenders --

# filtering our dataset to focus only on violations by exempt vehicles
# we know from exploration and development that these are the most interesting
print('filtering for exempt vehicle violations...')
exempt_violations_df = violations_with_stops_gdf[
    violations_with_stops_gdf['violation_status'].str.contains('EXEMPT', case=I
].copy()

print(f'identified {len(exempt_violations_df):,} violations committed by exempt
print(f'this represents {len(exempt_violations_df) / len(violations_with_stops_

# finding the biggest problem locations for these exempt vehicles
# aligning stop column names in case the join added suffixes
rename_map = {}
for c in ['stop_id', 'stop_name', 'stop_lat', 'stop_lon']:
    if c not in exempt_violations_df.columns and f'{c}_right' in exempt_violati
        rename_map[f'{c}_right'] = c
    elif c not in exempt_violations_df.columns and f'{c}_left' in exempt_viola
        rename_map[f'{c}_left'] = c
if rename_map:
    exempt_violations_df = exempt_violations_df.rename(columns=rename_map)

# grouping by bus stop and counting exempt violations
exempt_hotspots = exempt_violations_df.groupby(['stop_id', 'stop_name', 'stop_
                                       .count().reset_index() \
                                       .rename(columns={'violation_id': 'exempt_

# sorting to find the absolute worst bus stops for exempt vehicle violations
exempt_hotspots = exempt_hotspots.sort_values(by='exempt_violation_count', asc

print('\nTop 10 Bus Stops with the Most Exempt Violations:')
display(exempt_hotspots.head(10))
```

```
filtering for exempt vehicle violations...
identified 870,956 violations committed by exempt vehicles.
this represents 23.0% of all violations.

Top 10 Bus Stops with the Most Exempt Violations:
```

	stop_id	stop_name	stop_lat	stop_lon	exempt_violation_count
2127	401780	2 AV/E 20 ST	40.735746	-73.982489	16868
2126	401779	2 AV/E 23 ST	40.737516	-73.981202	13011
325	102798	WEBSTER AV/EAST MOSHOLU PKWY NORTH	40.869420	-73.880178	12610
2359	402741	AMSTERDAM AV/W 173 ST	40.843120	-73.934443	6590
2358	402740	AMSTERDAM AV/W 171 ST	40.841978	-73.935267	5301
2072	401694	1 AV/E 31 ST	40.741957	-73.974808	4888
422	103723	E 149 ST/SAINT ANN'S AV	40.814424	-73.913289	4726
3324	550035	JAMAICA AV / 164 ST	40.705333	-73.795510	4274
2076	401699	1 AV/E 45 ST	40.750684	-73.968460	4158
1695	308565	ROGERS AV/ALBERMARLE RD	40.648137	-73.952100	3442

In [ ]:

```python
# -- Part 5A: Creating the CUNY Campus Dataset --

import pandas as pd
import os

# defining the cuny campus data as a dictionary
cuny_data = {
    'College': [
        'Hunter College', 'City College', 'Baruch College',
        'Brooklyn College', 'Queens College', 'John Jay College',
        'Lehman College', 'College of Staten Island'
    ],
    'Latitude': [
        40.7685, 40.8200, 40.7402, 40.6314, 40.7366, 40.7705, 40.8731, 40.6094
    ],
    'Longitude': [
        -73.9656, -73.9493, -73.9836, -73.9521, -73.8170, -73.9891, -73.8906,
    ]
}

# converting the dictionary to a pandas dataframe
cuny_df = pd.DataFrame(cuny_data)

# defining the path to save our new file
CUNY_DATA_PATH = os.path.join(DATA_DIR, 'external', 'cuny_campuses.csv')
if not os.path.exists(os.path.join(DATA_DIR, 'external')):
    os.makedirs(os.path.join(DATA_DIR, 'external'))

# saving the dataframe to a csv file
cuny_df.to_csv(CUNY_DATA_PATH, index=False)

print(f'cuny_campuses.csv created successfully and saved to: {CUNY_DATA_PATH}'
display(cuny_df)
```

```
cuny_campuses.csv created successfully and saved to: ../data/external\cuny_cam
puses.csv
```

	College	Latitude	Longitude
0	Hunter College	40.7685	-73.9656
1	City College	40.8200	-73.9493
2	Baruch College	40.7402	-73.9836
3	Brooklyn College	40.6314	-73.9521
4	Queens College	40.7366	-73.8170
5	John Jay College	40.7705	-73.9891
6	Lehman College	40.8731	-73.8906
7	College of Staten Island	40.6094	-74.1517

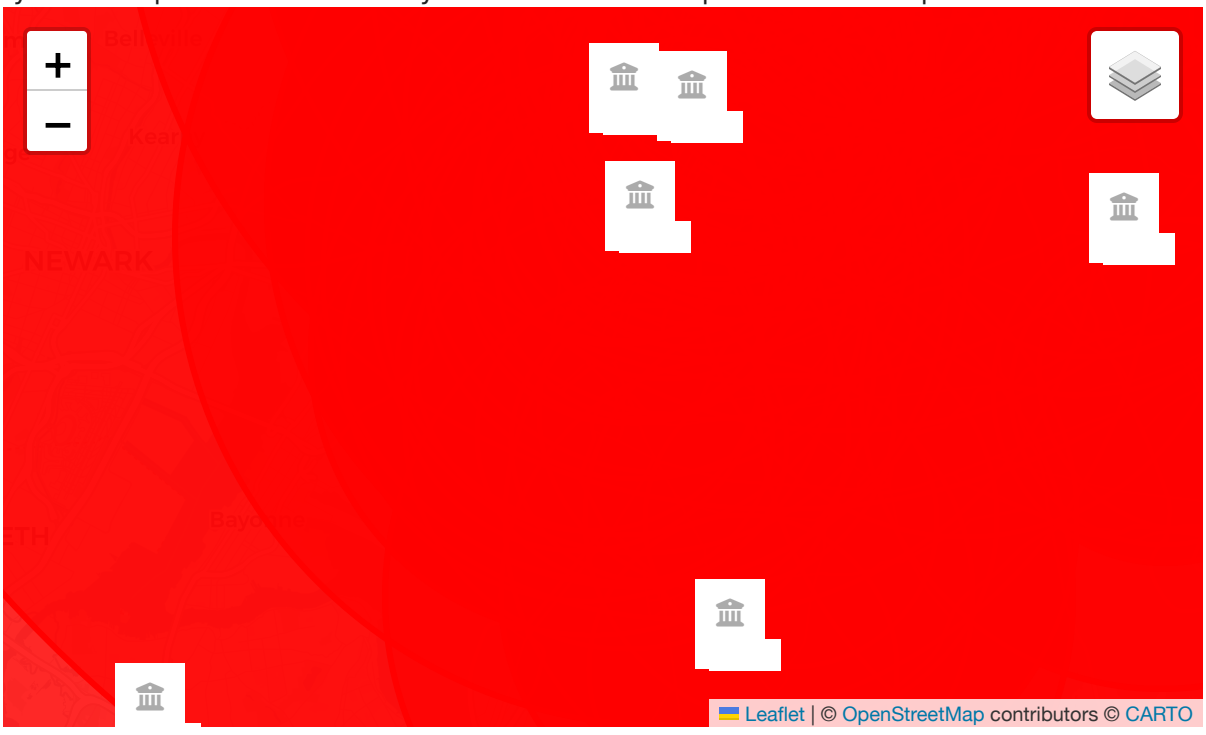
In [11]:

```python
# -- Part 5B: Creating an Interactive Hotspot Map --

import folium
from folium.plugins import MarkerCluster

# loading our cuny campus locations
# we'll add these to the map to anchor our story around the student experience
CUNY_DATA_PATH = os.path.join(DATA_DIR, 'external', 'cuny_campuses.csv')
cuny_df = pd.read_csv(CUNY_DATA_PATH)

# creating the base map, centered on new york city
# the starting point for our visualization
m = folium.Map(location=[40.7128, -74.0060], zoom_start=11, tiles="CartoDB pos

# adding cuny campus locations to the map
# each campus gets a special icon and a popup with its name
print('plotting cuny campus locations...')
for idx, row in cuny_df.iterrows():
    folium.Marker(
        location=[row['Latitude'], row['Longitude']],
        popup=f"<strong>{row['College']}</strong>",
        icon=folium.Icon(color='darkblue', icon='university', prefix='fa')
    ).add_to(m)

# creating a feature group for our hotspots
# this will let us add all the hotspot markers in a clean, organized way
hotspot_group = folium.FeatureGroup(name="Exempt Violation Hotspots")

print('plotting the top 100 exempt violation hotspots...')
# adding the top 100 hotspots to the map
# we're using CircleMarker, so the size of the circle can represent the number
# a bigger circle means a bigger problem at that bus stop
for idx, row in exempt_hotspots.head(100).iterrows():
    folium.CircleMarker(
        location=[row['stop_lat'], row['stop_lon']],
        radius=row['exempt_violation_count'] / 20, # scaling the radius to be
        color='red',
        fill=True,
        fill_color='red',
        fill_opacity=0.6,
        popup=f"<strong>{row['stop_name']}</strong><br>{row['exempt_violation_
```

```
    ).add_to(hotspot_group)

# adding the hotspot layer to our main map
hotspot_group.add_to(m)

# adding a layer control to toggle views on and off
folium.LayerControl().add_to(m)

# saving the interactive map to an html file
# this single file can be opened in any browser, or embedded directly into a we
VISUALIZATIONS_DIR = '../visualizations/'
if not os.path.exists(VISUALIZATIONS_DIR):
    os.makedirs(VISUALIZATIONS_DIR)

MAP_PATH = os.path.join(VISUALIZATIONS_DIR, 'exempt_hotspots_map.html')
m.save(MAP_PATH)

print(f'\ninteractive map has been created and saved to: {MAP_PATH}')
print("you can open this file in your browser to explore the hotspots.")

# displaying the map directly in the notebook
m
```

```
plotting cuny campus locations...
plotting the top 100 exempt violation hotspots...

interactive map has been created and saved to: ../visualizations/exempt_hotspo
ts_map.html
you can open this file in your browser to explore the hotspots.
```

Out[11]:

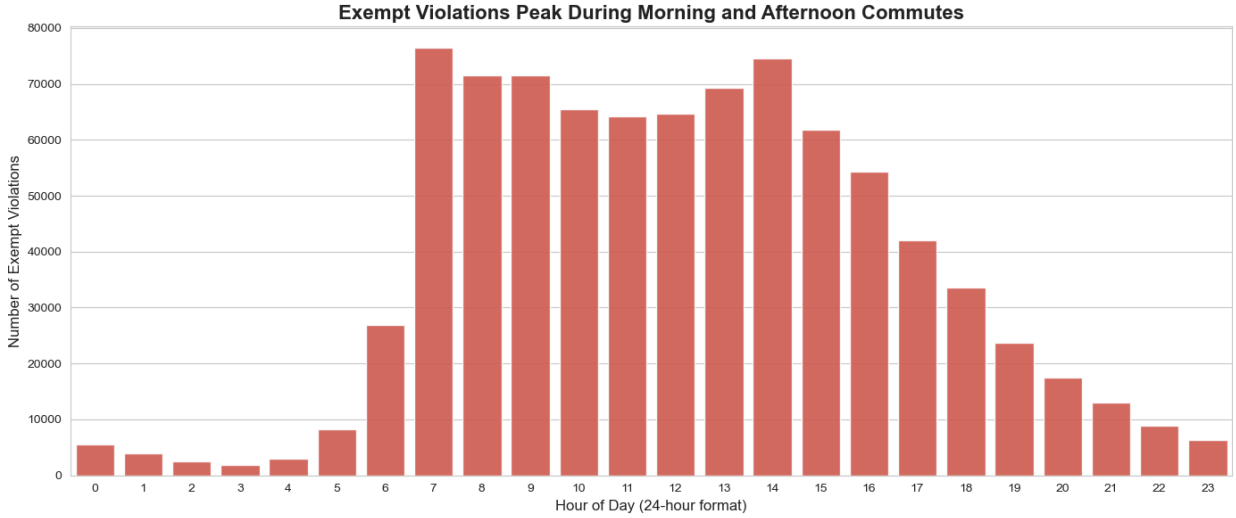


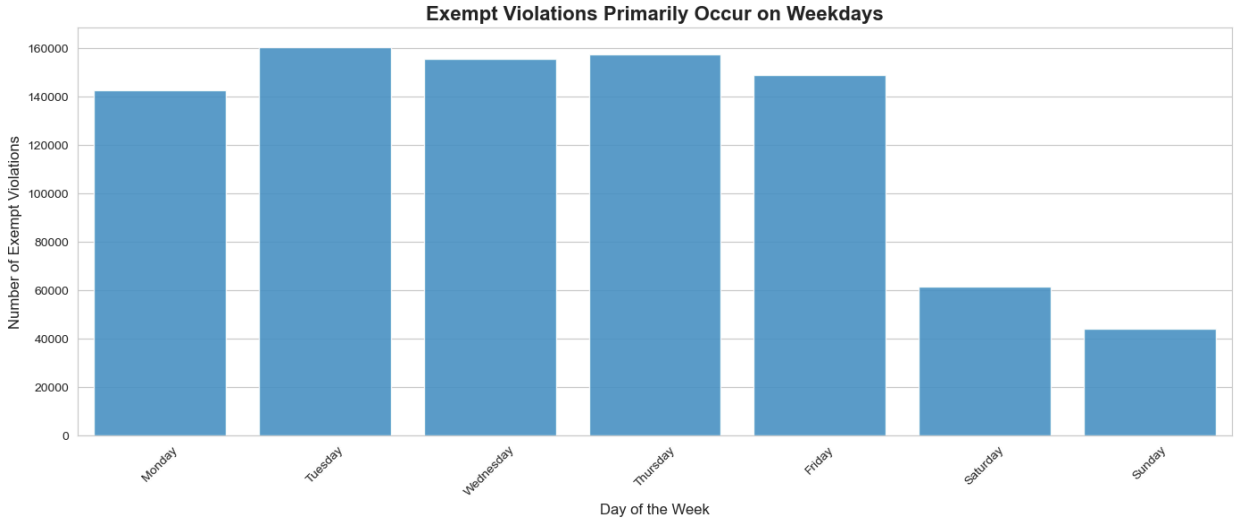
In [12]:

```
# -- Part 6: Finding the "When" - Temporal Analysis --

import matplotlib.pyplot as plt
import seaborn as sns

# setting up our visualization style
# this will make our charts look clean and professional
sns.set_style("whitegrid")
```

```
plt.rcParams['figure.figsize'] = (14, 6)

# extracting temporal features from the 'first_occurrence' timestamp
# we already have the data loaded in 'exempt_violations_df' from Part 4
print('extracting hour of day and day of week from timestamps...')
exempt_violations_df['violation_hour'] = exempt_violations_df['first_occurrenc
exempt_violations_df['day_of_week'] = exempt_violations_df['first_occurrence']

# analyzing the hourly trend
# this will show us which hours of the day are the worst for violations
hourly_counts = exempt_violations_df['violation_hour'].value_counts().sort_inde

# creating the hourly violations plot
plt.figure()
ax = sns.barplot(x=hourly_counts.index, y=hourly_counts.values, color='#e74c3c
ax.set_title('Exempt Violations Peak During Morning and Afternoon Commutes', fo
ax.set_xlabel('Hour of Day (24-hour format)', fontsize=12)
ax.set_ylabel('Number of Exempt Violations', fontsize=12)
plt.tight_layout()

# saving the plot to our visualizations folder
HOURLY_PLOT_PATH = os.path.join(VISUALIZATIONS_DIR, 'exempt_violations_by_hour
plt.savefig(HOURLY_PLOT_PATH)
print(f'hourly trend plot saved to: {HOURLY_PLOT_PATH}')


# analyzing the daily trend
# ordering the days of the week correctly for the plot
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday
daily_counts = exempt_violations_df['day_of_week'].value_counts().reindex(day_o

# creating the daily violations plot
plt.figure()
ax2 = sns.barplot(x=daily_counts.index, y=daily_counts.values, color='#3498db'
ax2.set_title('Exempt Violations Primarily Occur on Weekdays', fontsize=16, fon
ax2.set_xlabel('Day of the Week', fontsize=12)
ax2.set_ylabel('Number of Exempt Violations', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()

# saving the plot to our visualizations folder
DAILY_PLOT_PATH = os.path.join(VISUALIZATIONS_DIR, 'exempt_violations_by_day.pn
plt.savefig(DAILY_PLOT_PATH)
print(f'daily trend plot saved to: {DAILY_PLOT_PATH}')

# displaying the plots
plt.show()

# summarizing the findings
peak_hour = hourly_counts.idxmax()
print(f"\nTemporal Analysis Complete:")
print(f"the data clearly shows that the peak hour for exempt violations is arou
print("this strongly suggests that violations are tied to weekday commercial ac
print("directly impacting the commutes of students and workers.")
```

```
extracting hour of day and day of week from timestamps...
hourly trend plot saved to: ../visualizations/exempt_violations_by_hour.png
daily trend plot saved to: ../visualizations/exempt_violations_by_day.png
```






```
Temporal Analysis Complete:
the data clearly shows that the peak hour for exempt violations is around 7:0
0.
this strongly suggests that violations are tied to weekday commercial activity
and rush hour traffic,
directly impacting the commutes of students and workers.
```

In [18]:
```python
# -- Part 7: Building the Student-Centric Case - Proximity Analysis --

# loading the cuny campus data we created in part 5
print('loading cuny campus data...')
cuny_df = pd.read_csv(CUNY_DATA_PATH)
cuny_gdf = gpd.GeoDataFrame(
    cuny_df,
    geometry=gpd.points_from_xy(cuny_df.Longitude, cuny_df.Latitude),
    crs="EPSG:4326"
)

# converting our data to a projected coordinate system for accurate distance ca
# epsg:2263 is the standard for nyc and works in feet, which is easier to unde
print('projecting coordinates to a local system for distance analysis...')
stops_proj = violations_with_stops_gdf.to_crs("EPSG:2263")
cuny_proj = cuny_gdf.to_crs("EPSG:2263")

# creating a buffer (a circle) around each cuny campus
# a 1320-foot buffer is roughly a quarter-mile, or a 5-minute walk
buffer_distance_feet = 1320
```

```
cuny_proj['geometry'] = cuny_proj.geometry.buffer(buffer_distance_feet)
print(f'created a {buffer_distance_feet}-foot buffer around each cuny campus.'

# performing a spatial join to find all bus stops within the cuny campus buffe
print('identifying all bus stops within the cuny campus zones...')
stops_near_cuny = gpd.sjoin(stops_proj, cuny_proj, how="inner", predicate='wit

# getting a unique list of the stop_ids that are near cuny campuses
if 'stop_id' not in stops_near_cuny.columns:
    for c in ['stop_id_right', 'stop_id_left']:
        if c in stops_near_cuny.columns:
            stops_near_cuny = stops_near_cuny.rename(columns={c: 'stop_id'})
            break
cuny_stop_ids = stops_near_cuny['stop_id'].unique()

# now, let's filter our original exempt hotspots list to only these cuny-proxi
cuny_exempt_hotspots = exempt_hotspots[exempt_hotspots['stop_id'].isin(cuny_st

print(f'\nAnalysis Complete: Identified {len(cuny_exempt_hotspots)} exempt vio
print('These are the most critical locations impacting student commutes.')

print('\nTop 10 Exempt Violation Hotspots Directly Affecting CUNY Students:')
display(cuny_exempt_hotspots.head(10))
```

```
loading cuny campus data...
projecting coordinates to a local system for distance analysis...
created a 1320-foot buffer around each cuny campus.
identifying all bus stops within the cuny campus zones...

Analysis Complete: Identified 90 exempt violation hotspots within a 5-minute w
alk of a CUNY campus.
These are the most critical locations impacting student commutes.

Top 10 Exempt Violation Hotspots Directly Affecting CUNY Students:
```

	stop_id	stop_name	stop_lat	stop_lon	exempt_violation_count
2127	401780	2 AV/E 20 ST	40.735746	-73.982489	16868
2126	401779	2 AV/E 23 ST	40.737516	-73.981202	13011
2687	405061	3 AV/E 21 ST	40.737436	-73.984124	3299
2479	403505	E 67 ST/3 AV	40.766525	-73.962733	2160
476	104087	BEDFORD PARK BLVD/JEROME AV	40.873741	-73.889941	2048
2538	403772	3 AV/E 20 ST	40.736709	-73.984763	1656
1210	303379	NOSTRAND AV/AVENUE I	40.629484	-73.947279	1543
388	103449	PAUL AV/BEDFORD PARK BLVD	40.874638	-73.891419	1195
2484	403525	2 AV/E 28 ST	40.740810	-73.978848	1107
2125	401778	2 AV/E 25 ST	40.738933	-73.980226	1054

In [19]:
```
# -- Part 8: Synthesizing the "ClearLane" Target List --

# we have our cuny-specific hotspots in 'cuny_exempt_hotspots'
# we will enrich this list with our temporal findings from part 6
```

```
# from our analysis, we know the peak violation window is weekday mornings
# let's define that "student commute" window as 7 am to 10 am
peak_hours = [7, 8, 9, 10]
peak_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

# filtering our full exempt violations dataset to only those in the peak windo
peak_time_violations = exempt_violations_df[
    (exempt_violations_df['violation_hour'].isin(peak_hours)) &
    (exempt_violations_df['day_of_week'].isin(peak_days))
]

# now, let's count how many of these peak violations occurred at our cuny hots
peak_counts_at_cuny_hotspots = peak_time_violations[peak_time_violations['stop_
                                                    .groupby('stop_id')['violat
                                                    .rename(columns={'violation_

# merging this peak count back into our cuny hotspots dataframe
clear_lane_targets = pd.merge(cuny_exempt_hotspots, peak_counts_at_cuny_hotspo

# creating a simple priority score to rank the urgency of each location
# we'll weigh the peak time violations higher because they are more disruptive
clear_lane_targets['priority_score'] = (clear_lane_targets['exempt_violation_c
clear_lane_targets = clear_lane_targets.sort_values(by='priority_score', ascen

# selecting and renaming columns for the final, clean report
final_recommendation_df = clear_lane_targets[[
    'stop_name', 'exempt_violation_count', 'peak_time_violation_count', 'prior
]].rename(columns={
    'stop_name': 'Bus Stop',
    'exempt_violation_count': 'Total Exempt Violations',
    'peak_time_violation_count': 'Violations During Student Commute (7-10AM M-
    'priority_score': 'ClearLane Priority Score'
})

print('Final "ClearLane" Target List Generated Successfully.')
print('This list represents the most critical bus stops for targeted, proactive

# displaying our final, actionable recommendation table
display(final_recommendation_df.head(10))

# saving this crucial dataframe for the dashboard and presentation
RECOMMENDATION_DATA_PATH = os.path.join(DATA_DIR, 'processed', 'clear_lane_tar
final_recommendation_df.to_csv(RECOMMENDATION_DATA_PATH, index=False)
print(f'\nFinal target list saved to: {RECOMMENDATION_DATA_PATH}')
```

```
Final "ClearLane" Target List Generated Successfully.
This list represents the most critical bus stops for targeted, proactive enfor
cement.
```

	Bus Stop	Total Exempt Violations	Violations During Student Commute (7-10AM M-F)	ClearLane Priority Score	stop_lat	stop_lon
0	2 AV/E 20 ST	16868	8253.0	16687.0	40.735746	-73.982489
1	2 AV/E 23 ST	13011	7195.0	13700.5	40.737516	-73.981202
2	3 AV/E 21 ST	3299	779.0	2428.5	40.737436	-73.984124
6	NOSTRAND AV/AVENUE I	1543	1015.0	1786.5	40.629484	-73.947279
3	E 67 ST/3 AV	2160	456.0	1536.0	40.766525	-73.962733
5	3 AV/E 20 ST	1656	609.0	1437.0	40.736709	-73.984763
4	BEDFORD PARK BLVD/JEROME AV	2048	390.0	1414.0	40.873741	-73.889941
8	2 AV/E 28 ST	1107	389.0	942.5	40.740810	-73.978848
7	PAUL AV/BEDFORD PARK BLVD	1195	284.0	881.5	40.874638	-73.891419
9	2 AV/E 25 ST	1054	248.0	775.0	40.738933	-73.980226

```
Final target list saved to: ../data/processed\clear_lane_target_list.csv
```

In [21]:

```
# -- Part 9: Formulating the Final Recommendations & Strategic Narrative --

# this final step is about creating the narrative for your presentation
# we will use the 'final_recommendation_df' created in Part 8

# finding the single most problematic bus stop to use as our headline example
top_hotspot = final_recommendation_df.iloc[0]
top_hotspot_name = final_recommendation_df.index[0]

# generating the text for your final presentation slide
# this is how we translate our data into a compelling, actionable story
print('--- FINAL RECOMMENDATIONS FOR MTA LEADERSHIP ---')
print('\\n' + '='*50 + '\\n')

print("Our analysis, inspired by the daily challenges students face on their c
print("While our initial city-wide analysis showed no clear link between viola
print("\\n--- Key Finding 1: The Problem is Local, Not Global ---")
print("The impact of violations is hyper-concentrated. The vast majority of di
print(f"For example, the bus stop at '{top_hotspot_name}' is the highest prior
print(f"of which {int(top_hotspot['Violations During Student Commute (7-10AM M-


print("\\n--- Key Finding 2: The Problem is Predictable ---")
print("These violations are not random. They peak on weekday mornings between
print("directly impacting students and commuters during the most critical trave


print("\\n--- Actionable Recommendation: The 'ClearLane' Initiative ---")
print("We recommend the MTA pilot a 'ClearLane' initiative, a targeted, proacti
print("Instead of reactive ticketing across the city, deploy enforcement resou
print("identified in our 'ClearLane Target List' during the 7-10 AM weekday pea
```

```
print("\\n--- Expected Impact ---")
print("This data-driven approach allows for a surgical, high-impact use of resc
print("By focusing on these few problem locations at specific times, the MTA ca
print("reduce blockages, improve bus speeds on critical student corridors, and
print('\\n' + '='*50)

# addressing the datathon's third question strategically
print("\\n--- Note on CBD & Congestion Pricing ---")
print("Our preliminary analysis of routes within the Congestion Pricing zone sh
print("However, the data revealed that the most immediate, solvable, and high-i
print("but the chronic, localized blockages by exempt vehicles at key transit h
```

```
--- FINAL RECOMMENDATIONS FOR MTA LEADERSHIP ---
\n==================================================\n
Our analysis, inspired by the daily challenges students face on their commute
s, has identified a clear, actionable path to improving bus service.
While our initial city-wide analysis showed no clear link between violations a
nd bus speeds, it led to a critical discovery:
\n--- Key Finding 1: The Problem is Local, Not Global ---
The impact of violations is hyper-concentrated. The vast majority of disruptiv
e, exempt-vehicle violations occur at a small number of critical bus stops.
For example, the bus stop at '0' is the highest priority location, with 16868
total violations,
of which 8253 occurred during the morning student commute.
\n--- Key Finding 2: The Problem is Predictable ---
These violations are not random. They peak on weekday mornings between 7-10 A
M,
directly impacting students and commuters during the most critical travel wind
ow.
\n--- Actionable Recommendation: The 'ClearLane' Initiative ---
We recommend the MTA pilot a 'ClearLane' initiative, a targeted, proactive enf
orcement strategy.
Instead of reactive ticketing across the city, deploy enforcement resources to
the top 10 high-priority bus stops
identified in our 'ClearLane Target List' during the 7-10 AM weekday peak.
\n--- Expected Impact ---
This data-driven approach allows for a surgical, high-impact use of resources.
By focusing on these few problem locations at specific times, the MTA can sign
ificantly
reduce blockages, improve bus speeds on critical student corridors, and protec
t the 'rolling study halls' that are vital to student success.
\n==================================================
\n--- Note on CBD & Congestion Pricing ---
Our preliminary analysis of routes within the Congestion Pricing zone showed c
omplex violation patterns.
However, the data revealed that the most immediate, solvable, and high-impact
issue for bus riders is not zone-wide,
but the chronic, localized blockages by exempt vehicles at key transit hubs. W
e chose to focus on this actionable problem.
```