CUNY-Specific Analytics

1. Which MTA bus routes are highly utilized by CUNY students?
2. How do violation rates compare among CUNY campuses?

This data specifically focuses on 2025 (Jan-Aug) as these questions place on emphasis on current campuses and students.

In [ ]:
```
import requests
import io
import numpy as np
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import seaborn as sns
from shapely.geometry import Point
from pathlib import Path
```

Data loading/cleaning - MTA Bus Ridership (currently using beginning of 2025 dataset)

In [ ]:
```
ridership_endpoint = "https://data.ny.gov/resource/gxb3-akrn.csv"

def fetch_ridership_for_routes(route_list, start_date="2025-01-01", end_date="
    ridership_frames = []
    chunk_size = 300

    for i in range(0, len(route_list), chunk_size):
        chunk = route_list[i:i+chunk_size]
        routes_str = ",".join([f"'{r}'" for r in chunk])

        params = {
            "$select": "bus_route, sum(ridership) as total_ridership",
            "$where": f"bus_route in ({routes_str}) AND transit_timestamp betw
            "$group": "bus_route",
            "$limit": 50000
        }

        # query API
        resp = requests.get(ridership_endpoint, params=params)
        resp.raise_for_status()

        # read response into pandas
        df_chunk = pd.read_csv(io.StringIO(resp.text))
        ridership_frames.append(df_chunk)

    return pd.concat(ridership_frames, ignore_index=True)
```

Data loading/cleaning - CUNY campuses

In [ ]:
```
url = "https://data.ny.gov/resource/irqs-74ez.csv"
campuses = pd.read_csv(url)
campuses.head()
```

In [ ]:
```
campuses.info()
```

In [ ]:
```
campuses[['campus', 'lat', 'long']]
```

In [ ]:
```
# borough mapping for all CUNY campuses
borough_map = {
    "Borough of Manhattan Community College": "Manhattan",
    "Bronx Community College": "Bronx",
    "Hostos Community College": "Bronx",
    "Kingsborough Community College": "Brooklyn",
    "LaGuardia Community College": "Queens",
    "Queensborough Community College": "Queens",
    "Guttman Community College": "Manhattan",
    "Medgar Evers College": "Brooklyn",
    "New York City College of Technology": "Brooklyn",
    "College of Staten Island": "Staten Island",
    "School of Labor and Urban Studies": "Manhattan",
    "School of Law": "Queens",
    "The Graduate School and University Center": "Manhattan",
    "School of Professional Studies": "Manhattan",
    "School of Public Health": "Manhattan",
    "School of Journalism": "Manhattan",
    "Macaulay Honors College": "Manhattan",
    "Baruch College": "Manhattan",
    "Brooklyn College": "Brooklyn",
    "The City College of New York": "Manhattan",
    "School of Medicine": "Manhattan",
    "Hunter College": "Manhattan",
    "John Jay College of Criminal Justice": "Manhattan",
    "Lehman College": "Bronx",
    "Queens College": "Queens",
    "York College": "Queens"
}

campuses["borough"] = campuses["campus"].map(borough_map)

# clean list of all campuses (will be used later to ensure 0 values show up)
all_campuses = campuses[["campus", "borough"]].rename(columns={"campus": "campu
```

In [ ]:
```
# conversion for gpd - mapping
campuses['geometry'] = campuses.apply(lambda row: Point(row['long'], row['lat'
campuses_gdf = gpd.GeoDataFrame(campuses, geometry='geometry', crs="EPSG:4326"
```

Data loading/cleaning - Bus stops/routes

In [ ]:
```
# GTFS dataset
GTFS_ROOT = Path("../raw_data")

feeds = {"bronx": "gtfs_bronx",
         "brooklyn": "gtfs_brooklyn",
         "manhattan": "gtfs_manhattan",
         "queens": "gtfs_queens",
         "staten_island": "gtfs_staten_island",
         "busco": "gtfs_busco"}

def load_concat(file_name):
    frames = []
```

```
    for key, sub in feeds.items():
        p = GTFS_ROOT / sub / file_name
        if p.exists():
            df = pd.read_csv(p)
            df["feed"] = key
            frames.append(df)
    if not frames:
        raise FileNotFoundError(f"Could not find {file_name} in any of the feed
    return pd.concat(frames, ignore_index=True)

stops = load_concat("stops.txt")
routes = load_concat("routes.txt")
trips = load_concat("trips.txt")
stop_times = load_concat("stop_times.txt")

print(stops.shape, routes.shape, trips.shape, stop_times.shape)
stops.head()
```

In [ ]:
```
# conversion for gpd - mapping
stops_gdf = gpd.GeoDataFrame(stops.assign(geometry=[Point(xy) for xy in zip(st
            stops["stop_lat"])]),
            geometry="geometry",
            crs="EPSG:4326")
```

Lookup for which routes serve which stops

In [ ]:
```
# cleaning columns presented
stop_times_small = stop_times[["trip_id", "stop_id"]]
trips_small = trips[["trip_id", "route_id"]]

stops_to_route = (stop_times_small.merge(trips_small, on="trip_id", how="inner"
stops_to_route.head()
```

Routes Near CUNY Campuses - Dynamic buffer based on campus size

In [ ]:
```
# dynamic buffer sizes (in meters) determined by general campus size
buffer_dict = {
    "Borough of Manhattan Community College": 400,
    "Bronx Community College": 600,
    "Hostos Community College": 400,
    "Kingsborough Community College": 800,
    "LaGuardia Community College": 500,
    "Queensborough Community College": 700,
    "Guttman Community College": 300,
    "Medgar Evers College": 500,
    "New York City College of Technology": 400,
    "College of Staten Island": 1000,
    "School of Labor and Urban Studies": 300,
    "School of Law": 400,
    "The Graduate School and University Center": 300,
    "School of Professional Studies": 300,
    "School of Public Health": 300,
    "School of Journalism": 300,
    "Macaulay Honors College": 300,
    "Baruch College": 400,
    "Brooklyn College": 700,
    "The City College of New York": 700,
    "School of Medicine": 400,
```

```
    "Hunter College": 400,
    "John Jay College of Criminal Justice": 400,
    "Lehman College": 700,
    "Queens College": 800,
    "York College": 600
}

# conversion to meters (for buffer)
campuses_meters = campuses_gdf.to_crs(3857)
stops_meters = stops_gdf.to_crs(3857)

# if campus is not in dictionary, fall back to 500m
campus_buffers = campuses_meters.copy()
campus_buffers["geometry"] = campuses_meters.apply(
    lambda row: row["geometry"].buffer(buffer_dict.get(row["campus"], 500)),
    axis=1
)

# checking which stops fall inside each campus buffer
stops_near_campus = gpd.sjoin(stops_meters, campus_buffers[["campus", "geometry
stops_near_campus = stops_near_campus.rename(columns={"campus":"campus_name"})

# conversion back to latitude/longitude
stops_near_campus = stops_near_campus.to_crs(4326)
```

In [ ]:
```
# mapping stops -> routes then aggregating routes per campus
routes_near_campus = (stops_near_campus[["stop_id", "campus_name"]]
                      .merge(stops_to_route, on="stop_id", how="left")
                      .dropna(subset=["route_id"])
                      .merge(routes[["route_id", "route_short_name", "route_lo
                      .drop_duplicates(subset=["campus_name", "route_id"])
                      )

# show top routes per campus
top_routes_by_campus = (stops_near_campus[["stop_id", "campus_name"]]
                        .merge(stops_to_route, on="stop_id", how="left")
                        .dropna(subset=["route_id"])
                        .groupby(["campus_name", "route_id"])
                        .size().reset_index(name="nearby_stop_count")
                        .sort_values(["campus_name", "nearby_stop_count"], as
                        )

top_routes_by_campus.head()
```

Data loading/cleaning - MTA Bus ACE Violations

In [ ]:
```
file_path = "../raw_data/mta_ace_violations.csv"

# loading mta ace dataset in chunks (showing these columns listed)
use_columns = [
    "Violation ID", "First Occurrence", "Violation Type", "Bus Route ID", "Sto
    ]

violations_iterator = pd.read_csv(file_path, usecols=use_columns, chunksize=10

# putting into one dataframe with the listed columns
violations = pd.concat(violations_iterator, ignore_index=True)

violations = violations.rename(columns={
```

```
    "Stop ID": "stop_id",
    "Stop Name": "stop_name",
    "Bus Route ID": "route_id"
})

# time based filters
violations["First Occurrence"] = pd.to_datetime(violations["First Occurrence"]
                                format="%m/%d/%Y %I:%M:%S %p",
                                errors="coerce")
violations["year"] = violations["First Occurrence"].dt.year
violations["month"] = violations["First Occurrence"].dt.month_name()
violations["weekday"] = violations["First Occurrence"].dt.day_name()
violations_all_years = violations.copy()
```

In [ ]:
```
# FOR UNDERSTANDING: why are certain campuses outputting zero violations?
campus_list = all_campuses["campus_name"].unique()

for campus in campus_list:
    stops_c = stops_near_campus[stops_near_campus["campus_name"] == campus]["s
    v_all = violations_all_years[violations_all_years["stop_id"].isin(stops_c)]
    v_2025 = violations[violations["stop_id"].isin(stops_c)]

    print(f"{campus}: {len(stops_c)} stops linked, {v_all.shape[0]} total viol
```

In [ ]:
```
# 2025 vs all years

violations_all_years = violations.copy()

# assign campus_name to all violations
violations_all_years = violations_all_years.merge(
    stops_near_campus[["stop_id", "campus_name"]],
    on="stop_id", how="inner"
)

# total violations across all years
total_by_campus = (
    violations_all_years.groupby("campus_name")
    .agg(total_violations=("Violation ID", "count"))
)

# total violations in 2025
total_2025 = (
    violations_all_years[violations_all_years["year"] == 2025]
    .groupby("campus_name")
    .agg(violations_2025=("Violation ID", "count"))
)

# merge them
violations_all_vs_2025 = (
    total_by_campus.merge(total_2025, on="campus_name", how="left")
    .fillna({"violations_2025": 0})
    .reset_index()
    .sort_values("total_violations", ascending=False)
)

print(violations_all_vs_2025.head(15))
```

Conclusion: Since there are stops linked to those colleges with 0 violations in all years, we can conclude that they simply do not exist in the ACE dataset. This enforces the fact that ACE is currently deployed on limited routes and is not citywide, therefore campuses near routes that are not covered will naturally output 0.

In [ ]:
```
# keep only violations from 2025 (to match ridership timeframe)
violations = violations[violations["year"] == 2025]
```

In [ ]:
```
# checking if time frame is actually restricted to jan-aug 2025
print("Violations timeframe:", violations["First Occurrence"].min(), "to", vio

print(violations.shape)
print(violations.head())
```

In [ ]:
```
# building tidy fact table for event-level violations

violations_fact = (
    violations  # already filtered to 2025 in your notebook
    .merge(stops_near_campus[["stop_id", "campus_name"]], on="stop_id", how="i
    .merge(all_campuses, on="campus_name", how="left")  # adds borough
    .rename(columns={"Violation Type": "violation_type"})
    [[
        "Violation ID",
        "First Occurrence",
        "campus_name",
        "borough",
        "route_id",
        "violation_type",
        "year",
        "month",
        "weekday"
    ]]
)

# yyyy-mm
violations_fact["year_month"] = violations_fact["First Occurrence"].dt.to_peri

# make month categorical (for ordering)
month_order = ["January","February","March","April","May","June","July","Augus
violations_fact["month"] = pd.Categorical(
    violations_fact["month"], categories=month_order, ordered=True
)
```

In [ ]:
```
# build routes_per_campus with ridership
routes_per_campus = (
    stops_near_campus[["campus_name", "stop_id"]]
    .merge(stops_to_route, on="stop_id", how="left")
    .dropna(subset=["route_id"])
    .groupby(["campus_name", "route_id"])
    .size()
    .reset_index(name="stop_count")
)

routes_per_campus = (
    all_campuses.merge(routes_per_campus, on="campus_name", how="left")
    .fillna({"stop_count": 0, "route_id": "None"})
)
```

```
routes_per_campus["stop_count"] = routes_per_campus["stop_count"].astype(int)

# add ridership
unique_routes = routes_per_campus[routes_per_campus["route_id"] != "None"]["rou
ridership_data = fetch_ridership_for_routes(unique_routes)
ridership_data["bus_route"] = ridership_data["bus_route"].astype(str)
ridership_data["total_ridership"] = ridership_data["total_ridership"].astype(f

routes_per_campus = routes_per_campus.merge(
    ridership_data,
    left_on="route_id",
    right_on="bus_route",
    how="left"
).drop(columns=["bus_route"])

routes_per_campus["total_ridership"] = routes_per_campus["total_ridership"].fi
```

Creating CSV files

In [ ]:

```
# 1. Monthly violations per campus (using yyyy-mm)
monthly_violations_per_campus = (
    violations_fact
    .groupby(["campus_name", "year_month"], observed=True)
    .size()
    .reset_index(name="violations")
)

# ensure all (campus, year_month) pairs exist
all_pairs_months = pd.MultiIndex.from_product(
    [all_campuses["campus_name"].unique(), violations_fact["year_month"].unique
    names=["campus_name", "year_month"]
).to_frame(index=False)

monthly_violations_per_campus = (
    all_pairs_months
    .merge(monthly_violations_per_campus, on=["campus_name", "year_month"], how
    .fillna({"violations": 0})
    .astype({"violations": int})
)

monthly_violations_per_campus.to_csv("../insights/CUNY_Insights/monthly_violat


# 2. Violations by type per campus
violations_by_type_per_campus = (
    violations_fact
    .groupby(["campus_name", "violation_type"], observed=True)
    .size()
    .reset_index(name="violations")
)

# fill zeros for all campus/type pairs
all_types = violations_fact["violation_type"].dropna().unique()
all_pairs_types = pd.MultiIndex.from_product(
    [all_campuses["campus_name"].unique(), all_types],
    names=["campus_name", "violation_type"]
).to_frame(index=False)
```

```
violations_by_type_per_campus = (
    all_pairs_types
    .merge(violations_by_type_per_campus, on=["campus_name","violation_type"],
    .fillna({"violations": 0})
    .astype({"violations": int})
)

violations_by_type_per_campus.to_csv("../insights/CUNY_Insights/violations_by_


# 3. Routes per campus (with ridership)
routes_fact = routes_per_campus[[
    "campus_name", "borough", "route_id", "stop_count", "total_ridership"
]].copy()

routes_fact.to_csv("../insights/CUNY_Insights/routes_per_campus_tidy_2025.csv"


# 4. Campus-level totals (summary view)
campus_summary = (
    violations_fact
    .groupby("campus_name", observed=True)
    .agg(total_violations=("Violation ID", "count"))
    .reset_index()
)

campus_ridership = (
    routes_fact.groupby("campus_name", observed=True)
    .agg(total_ridership=("total_ridership","sum"))
    .reset_index()
)

campus_summary = (
    campus_summary
    .merge(campus_ridership, on="campus_name", how="outer")
    .fillna({"total_violations": 0, "total_ridership": 0})
    .astype({"total_violations": int, "total_ridership": int})
)

campus_summary.to_csv("../insights/CUNY_Insights/campus_summary_2025.csv", ind


# 5. Monthly trend (all campuses combined)
violations_month_trend = (
    violations_fact.groupby("month", observed=True)
    .size()
    .reset_index(name="total_violations")
)

# 5. Monthly trend (all campuses combined, yyyy-mm)
violations_month_trend = (
    violations_fact.groupby("year_month", observed=True)
    .size()
    .reset_index(name="total_violations")
    .sort_values("year_month")
)

violations_month_trend.to_csv("../insights/CUNY_Insights/violations_monthly_tr
```

```
# ============================================================
print("Dashboard CSVs exported to ../insights/CUNY_Insights")
```

Visualizations/Insights

In [ ]:

```
# 1. Top routes by ridership per campus
campus_ridership = (
    routes_fact.groupby("campus_name", observed=True)
    .agg(total_ridership=("total_ridership","sum"))
    .reset_index()
    .sort_values("total_ridership", ascending=False)
)

plt.figure(figsize=(12, 8))
sns.barplot(
    data=campus_ridership,
    x="total_ridership",
    y="campus_name",
    hue="campus_name",
    dodge=False,
    palette="viridis",
    legend=False
)
plt.title("Total Bus Ridership (Jan–Aug 2025) by CUNY Campus")
plt.xlabel("Total Ridership")
plt.ylabel("Campus")
plt.tight_layout()
plt.show()


# 2. Total violations per campus
campus_violations = (
    violations_fact.groupby("campus_name", observed=True)
    .agg(total_violations=("Violation ID", "count"))
    .reset_index()
    .sort_values("total_violations", ascending=False)
)

plt.figure(figsize=(12, 8))
sns.barplot(
    data=campus_violations,
    x="total_violations",
    y="campus_name",
    hue="campus_name",
    dodge=False,
    palette="magma",
    legend=False
)
plt.title("Total ACE Violations (Jan–Aug 2025) by CUNY Campus")
plt.xlabel("Total Violations")
plt.ylabel("Campus")
plt.tight_layout()
plt.show()


# 3. Violation types across all campuses
violation_types = (
    violations_fact.groupby("violation_type", observed=True)
    .size()
```

```
    .reset_index(name="total_violations")
    .sort_values("total_violations", ascending=False)
)

plt.figure(figsize=(10, 6))
sns.barplot(
    data=violation_types,
    x="total_violations",
    y="violation_type",
    hue="violation_type",
    dodge=False,
    palette="Set2",
    legend=False
)
plt.title("ACE Violation Types Across CUNY Campuses (Jan–Aug 2025)")
plt.xlabel("Total Violations")
plt.ylabel("Violation Type")
plt.tight_layout()
plt.show()


# 4. Time trend: violations by month (yyyy-mm)
violations_month = (
    violations_fact.groupby("year_month", observed=True)
    .size()
    .reset_index(name="total_violations")
    .sort_values("year_month")
)

plt.figure(figsize=(10, 6))
sns.lineplot(
    data=violations_month,
    x="year_month",
    y="total_violations",
    marker="o"
)
plt.title("Monthly Trend of ACE Violations Near CUNY Campuses (Jan–Aug 2025)")
plt.xlabel("Month (yyyy-mm)")
plt.ylabel("Total Violations")
plt.tight_layout()
plt.show()
```

note: august is an artifical drop (dataset is incomplete)

Notes

- Probably should normalize the data for each campus (e.g. if a school is bigger and has more students, it'll most likely show more violations)

Recommendations

- Use polygon footprint to have better spatial accuracy --> campus boundaries will be more precise therefore insights will be better

Answers

- Q1 is answered by routes_fact + ridership plots

- Q2 is answered by violations_fact aggregates + violation plots