Database Final Project Report: Denver Crime and RTD Stops

Paul Lintzen and Luke Dembosky

December 11, 2025

1 Introduction

Our project focuses on taking the crime records in Denver along with public transit data to explore some of the connections between the two. We were most interested in looking at the relative 'safety' of different stops and examining the abundance of different types of crimes. Our largest deliverable was an interactive bubble map that allows users to look at both the total count of crimes near a stop and how much of that crime is 'violent', using FBI's Uniform Crime Reporting Definition of a violent crime.

2 Component 1

2.1 Dataset Introduction

The datasets we chose were Denver Crime Dataset and Regional Transportation District (RTD) Bus Stops Dataset from Denver's Open Data Catalog. The crime dataset is updated daily but we downloaded it on November 25th, so our records range from January 2020 to November 2025. After cleaning this left us with 391,810 unique crime records and 8,386 bus stops. The crime dataset includes 20 columns by default and we kept 14 and added 3 of our own. The bus dataset includes 12 columns and we ended up keeping all of them and just adding 1. These datasets are both fully available to the public as part of Denver's Open Data Initiative with the only restrictions being that data is provided 'as is' for informational use only; the City of Denver is not liable and it is not for engineering purposes.

2.2 Requirements

Our dataset fulfills both the large dataset and 'mash up' dataset requirements. The crime dataset includes more than 390,000 entries and the bus dataset has more than 8,000 so they are definitely large enough. In addition, there is no obvious direct connection between crime and bus stops in Denver and we had to come up with a way to connect them using location data which was the only commonality between the two.

3 Component 2

We selected data science and analysis for our main application. Our main goal was to attempt to examine correlation betweens the amount and type of crimes and the location of RTD Bus Stops in Denver. The first step was getting the raw data into our database which we did using intermediate tables where we intially loaded all columns as TEXT and then converted to the appropriate type once they were clean. From there we needed a way to efficiently group crimes to their nearest bus stop. The datasets we used each contained a latitude and longitude location for each crime reported and each bus stop respectively. Using PostGIS we created new column in both datasets that encoded location and then created an indices for said columns to quickly compute and compare the closest distance for each item. This will be explored further in Section 4.1.

Once we had our tables fully set up, we started but creating a sunburst diagram to visualize the total makeup of our crime records see 4.4. We analyzed the amount and types of crimes by hour, as well as looking at potential correlations between the length of a bus route and the amount and type of crimes near it. After adding a new column that classifies each recorded offense as violent or non-violent, we then developed a query that would give us all the relevant info we wanted in a new csv file to then interpret and turn into a bubble map. These queries will be explored further in Section 4.2.

After extracting the data we needed from our tables using SQL, we created all visualizations in Python, using packages like Folium, Plotly, and Matplotlib to enable interactive exploration and effective communication of trends and patterns within the data.

4 Component 3

4.1 Performance Tuning

After getting our data fully cleaned and into our SQL tables, the next step was converting our coordinate information to a format that will process in a reasonable time. Since we had ≈390,000 entries in our crime dataset calculating the nearest bus stop for each record using raw latitude and longitude values would require computing millions of distances in a brute-force manner which would be very slow. Using the PostGIS extension we were able to use a spatially indexed nearest-neighbor join instead.

To do this we converted our latitude and longitude columns into geometry points with a defined spatial reference system (WGS 84, EPSG:4326). This allowed us to create spatial indexes on both the crime locations and bus stop locations. Using these indexes, we could perform a nearest-neighbor join, where PostGIS efficiently identifies the closest bus stop to each crime record without having to compute every possible distance which would be incredibly slow for such a large dataset. Essentially, PostGIS leverages these indexes to reduce the search space drastically, turning what would be a potentially infeasible computation into one that runs in a matter of seconds. This made it practical to associate each crime with its nearest bus stop and perform further aggregation and analysis for visualization. Included below is snippet of the code to convert our coordinates and add indices.

```sql
-- Enable PostGIS extension and create new geometry columns
CREATE EXTENSION IF NOT EXISTS postgis;
ALTER TABLE crime_data ADD COLUMN geom geometry(Point, 4326);
UPDATE crime_data
SET geom = ST_SetSRID(ST_MakePoint(longitude, latitude), 4326);

ALTER TABLE bus_data ADD COLUMN geom geometry(Point, 4326);
UPDATE bus_data
SET geom = ST_SetSRID(ST_MakePoint(longitude, latitude), 4326);

-- Create indexes to speed up join immensely
CREATE INDEX crimes_gix ON crime_data USING GIST (geom);
CREATE INDEX bus_stops_gix ON bus_data USING GIST (geom);
```

Testing the speedup from using the PostGIS geometry points over the raw coordinate data is infeasible as running an equivalent query using the raw coordinate data on our dataset would take hours if not days. However, we can actually test how much of an impact adding the indices makes using EXPLAIN ANALYZE. Without any indices our total time for running Query A was 118616 ms while with indices it was 1053 ms. This an insane speedup of about 113x. This is really impressive, especially considering it comes purely from adding indices. We haven't even accounted for the potential improvement from using a nearest neighbor join with our new geometry point objects. Included below is the full output of both explains:

Query A WITHOUT Indices:

```
Sort  (cost=222282263.73..222282263.74 rows=1 width=28) (actual time=118616.018..118616.034 rows=539 loops=1)
   Sort Key: (count(*)) DESC
   Sort Method: quicksort  Memory: 62kB
   ->  GroupAggregate  (cost=222281779.66..222282263.72 rows=1 width=28) (actual time=118611.980..118615.952 rows=539 loops=1)
         Group Key: bus_data.bsid, bus_data.lng, bus_data.lat
         Filter: (count(*) > 20)
         Rows Removed by Filter: 1978
         ->  Sort  (cost=222281779.66..222281876.47 rows=38724 width=20) (actual time=118611.971..118613.235 rows=39126 loops=1)
               Sort Key: bus_data.bsid, bus_data.lng, bus_data.lat
               Sort Method: quicksort  Memory: 3563kB
               ->  Nested Loop  (cost=5739.68..222278828.71 rows=38724 width=20) (actual time=5.325..118603.013 rows=39126 loops=1)
                     ->  Seq Scan on crime_data c  (cost=0.00..14589.10 rows=38724 width=32) (actual time=0.009..58.875 rows=39126 loops=1)
                           Filter: is_violent
                           Rows Removed by Filter: 352684
                     ->  Limit  (cost=5739.68..5739.68 rows=1 width=60) (actual time=3.029..3.029 rows=1 loops=39126)
                           ->  Sort  (cost=5739.68..5760.65 rows=8387 width=60) (actual time=3.029..3.029 rows=1 loops=39126)
                                 Sort Key: ((c.geom <-> bus_data.geom))
                                 Sort Method: top-N heapsort  Memory: 25kB
                                 ->  Seq Scan on bus_data  (cost=0.00..5697.74 rows=8387 width=60) (actual time=0.020..2.243 rows=8387 loops=39126)
Planning Time: 0.111 ms
Execution Time: 118616.191 ms
```

Query A WITH Indices:

```
Sort  (cost=54141.01..54141.02 rows=1 width=28) (actual time=1052.823..1052.839 rows=540 loops=1)
   Sort Key: (count(*)) DESC
   Sort Method: quicksort  Memory: 62kB
   ->  HashAggregate  (cost=54140.99..54141.00 rows=1 width=28) (actual time=1052.597..1052.766 rows=540 loops=1)
         Group Key: bus_data.bsid, bus_data.lng, bus_data.lat
         Filter: (count(*) > 20)
         Batches: 1  Memory Usage: 649kB
         Rows Removed by Filter: 1978
         ->  Nested Loop  (cost=0.15..53753.75 rows=38724 width=20) (actual time=0.115..1045.210 rows=39126 loops=1)
               ->  Seq Scan on crime_data c  (cost=0.00..14589.10 rows=38724 width=32) (actual time=0.017..54.705 rows=39126 loops=1)
                     Filter: is_violent
                     Rows Removed by Filter: 352684
               ->  Limit  (cost=0.15..0.99 rows=1 width=60) (actual time=0.025..0.025 rows=1 loops=39126)
                     ->  Index Scan using bus_stops_gix on bus_data  (cost=0.15..7056.80 rows=8387 width=60) (actual time=0.025..0.025 rows=1 loops=39126)
                           Order By: (geom <-> c.geom)
Planning Time: 0.167 ms
Execution Time: 1052.877 ms
```

4.2 Interesting or Complex Queries

Note: Query A was used for our performance analysis and to compile the data for our initial bubble chart in Figure 1. We later made further modifications leading to Query A-v2 which was used to create the csv for our bubble chart in Figure 2. They are very similar but just for full transparency the performance numbers were with the original Query A.

4.2.1 Query A-v2

```
\copy (
WITH nearest AS (
    SELECT
        c.offense_type_id,
        c.is_violent,
        b.bsid AS stop_id,
        b.lng,
        b.lat,
        b.routes,
        b.stopname,
        b.postal
    FROM crime_data c
    JOIN LATERAL (
        SELECT bsid, lng, lat, routes, stopname, postal, geom
        FROM bus_data
        ORDER BY c.geom <-> geom
        LIMIT 1
    ) b ON TRUE
)
SELECT
    stop_id,
    lng,
    lat,
    routes,
    stopname,
    postal,
    COUNT(*) AS total_crimes,
    COUNT(*) FILTER (WHERE is_violent = TRUE) AS violent_crimes
FROM nearest
GROUP BY stop_id, lng, lat, routes, stopname, postal
HAVING COUNT(*) > 20
ORDER BY violent_crimes DESC
) TO 'crime_bus_counts_v2.csv' CSV HEADER;
```

Explanation:

This query first constructs a new CTE called nearest, which takes our fields of interest from both the crime and bus data tables and joins them into one based on location. The Lateral Join is Postgres specific join that allows a subquery to reference columns from the outer query. In our case this was necessary for our subquery to be able to access c.geom which is what enabled calculating the closest bus stop to each crime. To actually calculate the closest bus stop we use the PostGIS distance operator '<->' along with LIMIT

1 which is much faster than manually calculating all the distances. The ON TRUE after our JOIN is to satisfy the normal JOIN syntax since we already have our join condition baked in with the lateral join on location. So, the result of this CTE is a row for every crime along with the attributes of its nearest bus stop.

We then aggregate the information from this CTE to count total crimes at each stop along with violent crimes. We had previously added a column to our crime data table that labels each crime as violent or not based on the offense category. We then group by all the columns that we aren't aggregating and filter out any stops with less than 20 total crimes. Finally, we order the data by violent crime count which doesn't matter for our visualization but helps when inspecting the raw numerical data. The return of this query then gets written to a csv file which we can later parse in Python.

4.2.2 Query B:

```
WITH
crime_nearest AS (
    SELECT
        c.*,
        b.bsid
    FROM crime_data c
    JOIN LATERAL (
        SELECT bsid
        FROM bus_data
        ORDER BY c.geom <-> bus_data.geom
        LIMIT 1
    ) b ON TRUE
),
split_routes AS (
    SELECT
        bd.bsid,
        trim(route) AS route,
        bd.distance
    FROM bus_data bd,
    LATERAL regexp_split_to_table(bd.routes, '\s*,\s*') AS route
),
route_lengths AS (
    SELECT
        route,
        MAX(distance) AS route_length
    FROM split_routes
    GROUP BY route
),
route_crime AS (
    SELECT
        sr.route,
        COUNT(*) AS crime_count
    FROM crime_nearest c
    JOIN split_routes sr ON sr.bsid = c.bsid
    GROUP BY sr.route
```

```
)
SELECT
    rl.route,
    rl.route_length,
    rc.crime_count
FROM route_lengths rl
JOIN route_crime rc ON rc.route = rl.route
ORDER BY rl.route_length;
```

Explanation:

The purpose of this query is to find the amount of crime on each individual route that a bus travels. The bus dataset includes routes that each bus at that stop travels, so everything can be found from our original data. The first CTE uses the created location columns, called geom, to find the closest bus stop to each crime. The second one splits the route column if needed, since certain rows have comma separated lists of traveled routes. The third one uses the distance column in the bus data to find the longest individual route. The fourth one then counts the amount of crime that is closest to a bus stop that travels on each individual route. Finally, we select and join everything from our CTEs, then order by length for easier graphing.

4.3 Statistical and Machine Learning Analysis

Example SQL statements:

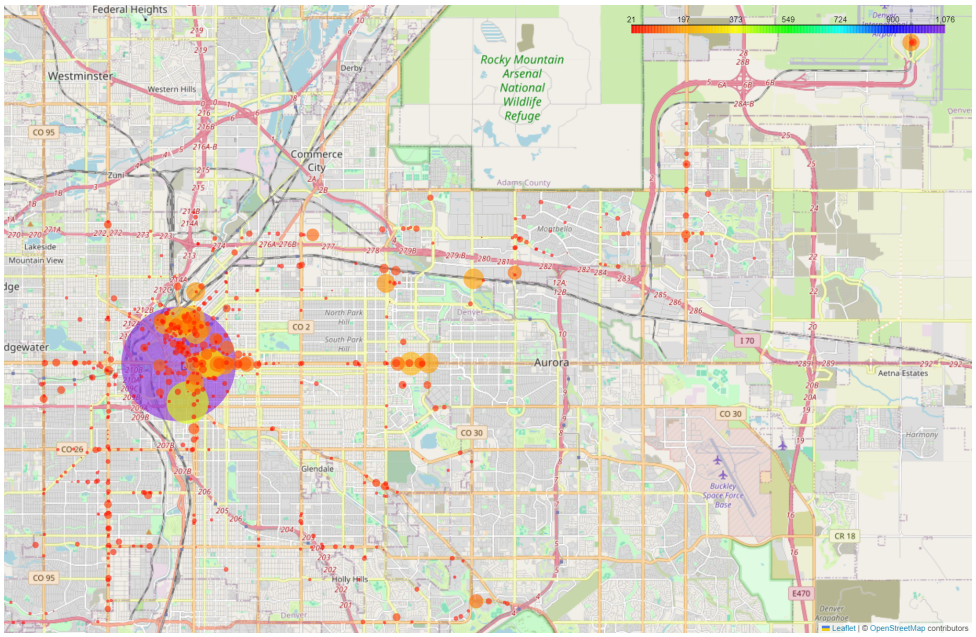

Figure 1: Visualization of violent crime frequency by bus stop

Explanation:

This bubble chart visualizes the violent crimes at each stop. We filtered out any stops with less than 20 violent crimes to reduce clutter. The circle size and color are both based on total violent crimes close to that stop. The lowest stops had just 20 violent crimes while the highest had around 1000. This is of course heavily influences by location and general traffic in the area. The largest purple bubble clearly visible in our graph was actually the

bus stop right outside the Denver Justice Center, Courthouse and Sheriff's department. This helps explain why there are so many more violent crimes recorded specifically here. Other interesting takeaways include that the downtown area just generally has more crime at stops which makes sense as there is generally much more foot traffic and potential for confrontation and just more police presense, both of which will inflate this number.

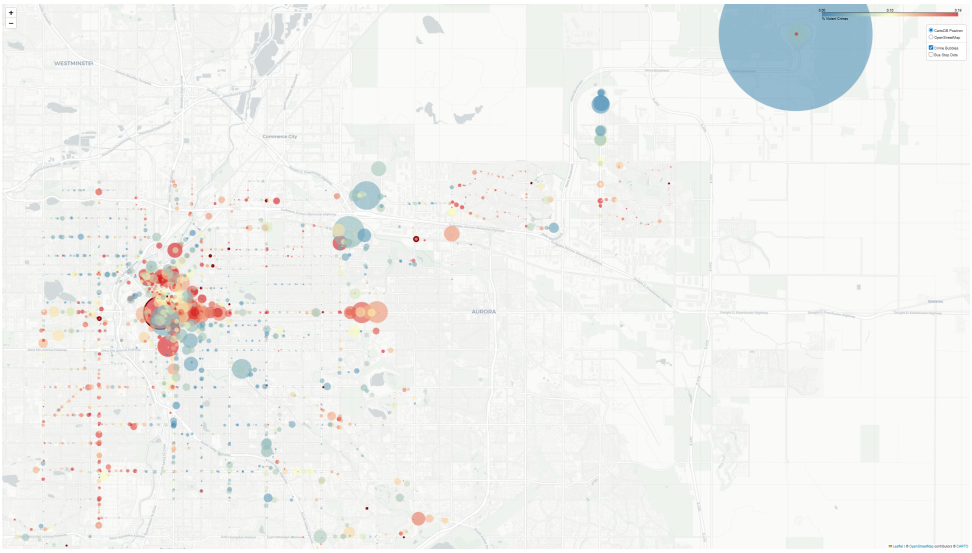

Figure 2: Visualization violent crime percentage by stop

Explanation:

This visualization was a refinement of our previous bubble chart. The full visualization can be accessed here. This one actually takes into account total crime counts versus violent crimes. The size of the circle at each stop is based on the total crime count while the color is based on what percentage of that crime is violent. Our color was based on a diverging scale where we set a baseline of our mean violent crime percent across all stops which was about 10%. Stops are colored more red if they are significantly higher than this and more blue if they are significantly below. Additionally a dark red outline is placed around stops with a violent crime percentage above 33%, to highlight potentially more dangerous stops. The bubble chart is interactive and allows users to toggle between a more dull map to make the bubbles stand out more and a more detailed map that includes location names to help explain why some stops may have higher percentages than others. Additionally, users can zoom in or out and click bubbles to actually look at the numbers and see the bus stop names.

Some interesting takeways here were that Denver International Airport (DIA) has significantly higher total crime than any other stops but only a very low percentage of violent crime. There are two main factors that we think explain this. First, being an aiport it is naturally much higher traffic than other areas and has a much higher law enforcement presence so we would expect more crimes to be recorded. Additionally, it is very isolated from other stops which means that even if crimes occurred in the areas surrounding the airport, all of them would be treated as crimes at the airport with how we are assigning our crimes. Another interesting note is that the courthouse stop discussed in the previous bubble map again stands out with a violent crime rate of 53.4%, much higher than the average.

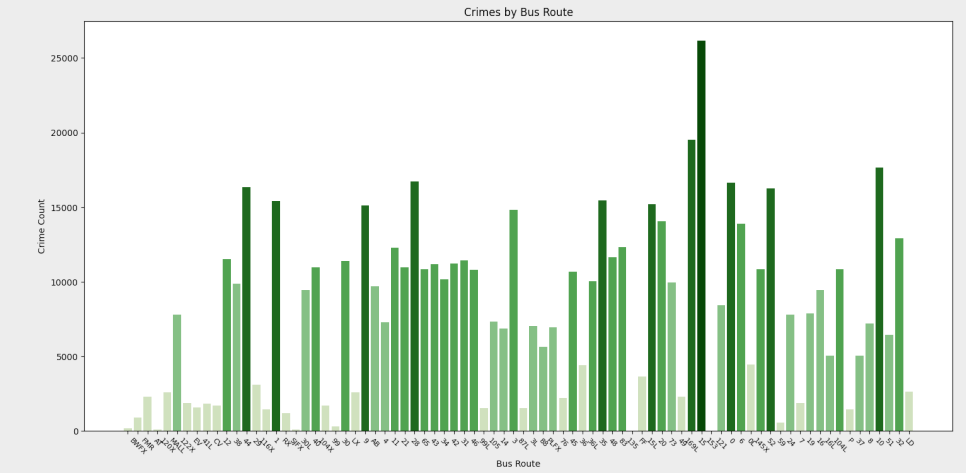

Figure 3: Visualization of Amount of Crimes per Bus Route ordered by length of Route

Explanation:

This table was created based on the output from query B. The x axis is route names according to the bus data table, ordered from shortest to longest. The crime counts are measured by the bar length and divided into 5 categories, indicated by the shade of green, based on the amount of crime in that route. This chart was created using the matplotlib package in Python. Overall, there does not seem to be any correlation between crime count and route length, however some notable routes are 15, the largest crime count, which is Colfax, a street that already has a solid reputation for being generally high in crime and dangerous. In general, based on this data, the more rural a road is, the less crime it has, but that is likely due to population rather than length.

4.4 Data Visualization

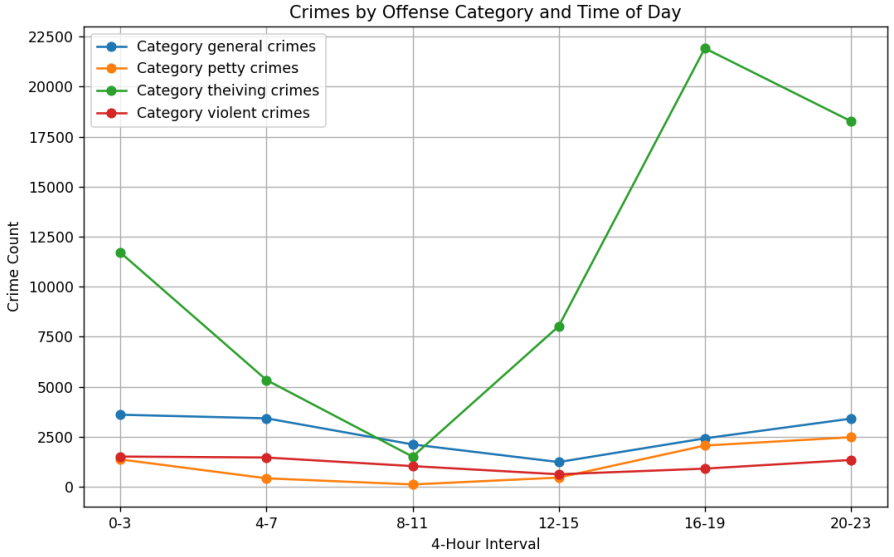

Figure 4: Visualization of Crime Types by Day

Explanation:

This table was created via a query that parsed through the reported date category of our crimes dataset to find the time, then classified it as one of our four categories that we defined, based on the thirteen offense categories from the table. We grouped them based on time of day to find the most common. In general, thievery crimes were by far the most common, with automotive thieving specifically being the highest, and only at late afternoon/early nightfall. Beyond that, the rest of the crime types were mostly consistently low, compared to thieving, across all hours of the day.

Figure 5: Visualization of total crime types and amounts

Explanation:

This chart is a sunburst diagram that breaks down all the records in our crimes data table. It was made in python using plotly. It first classifies every entry as violent or non-violent, based on the FBI's UCR definition of violent crime. The ring outside of that are the same small categories we created for our time visualization that further condense the 13 categories listed in the original data. The next ring, is all unique categories in the original crime dataset. Finally, the outermost ring is all unique types of crime in our database. This provides us a great way to visualize the total breakdown of increasingly specific information about the crime. We can see that a vast majority of the crime is non violent. Additionally, thieving makes up the majority of that non-violent crime. And we can see that the the largest type of crime in our thieving category is theft of parts and items from motor vehicles, closely followed by theft of the vehicles itself.

5 Conclusion

In summary, we learned there seemed to be some correlation between crime count and violent crime percentage by bus stop location. But it's likely not the bus stop itself that determines this and usually outside factors such as surrounding buildings (courts, airports, police stations, etc) can help explain this. While there is no direct correlation, there are stops that would be considered safer than others when traveling. For example, for our second bubble map visualization stops that are outlined in red especially may be areas to avoid as they have much higher rates of violent crime than surrounding areas. However, it is important to keep in mind the limitations with our project as a whole. For example, areas with less bus stops, may seem to have inflated crime counts as they will have many more associated crimes than in areas with more stops. Additionally, higher traffic areas will naturally have more crime but that doesn't necessarily mean they are less safe. This project highlighted various ways of data visualization, including a bubble map with folium, a sunburst chart with plotly, and various graphs with matplotlib, using visualization to communicating information that would be very hard to interpret from the raw tables. Overall, this was a super interesting exploration of combinign datasets that may at first appear unrelated to extract new information and was a great opportunity to learn how to work with lots of new ways of visualizing data.o

6 Appendix

Bubble Map Code:

```
import folium
import pandas as pd
from branca.colormap import LinearColormap

# load from SQL output
df = pd.read_csv("crime_bus_counts_v2.csv")

# Add % violent column
df["pct_violent"] = df["violent_crimes"] / df["total_crimes"]

mean_v = df["pct_violent"].mean()

colormap = LinearColormap(
    ["#2c7bb6", "#ffffbf", "#d7191c"],
    vmin=0,
    vmax=mean_v * 2
)

# Feature groups for toggle
bubble_layer = folium.FeatureGroup(name="Crime Bubbles",
   show=True)
dot_layer    = folium.FeatureGroup(name="Bus Stop Dots",
   show=False)

m = folium.Map(
    location=[df.lat.mean(), df.lng.mean()],
```

```
    zoom_start=12,
    tiles=None
)

# CartoDB Positron
folium.TileLayer(
    "CartoDB Positron",
    name="CartoDB Positron",
    control=True,    # appears in LayerControl
    show=True        # this makes it the visible default
).add_to(m)

# OpenStreetMap
folium.TileLayer(
    "OpenStreetMap",
    name="OpenStreetMap",
    control=True,
    show=False       # hidden initially
).add_to(m)

for _, row in df.iterrows():
    popup_html = f"""
    <b>{row.stopname}</b><br>
    Stop ID: {row.stop_id}<br>
    Routes: {row.routes}<br>
    Postal Code: {row.postal}<br><br>

    <b>Total Crimes:</b> {row.total_crimes}<br>
    <b>Violent Crimes:</b> {row.violent_crimes}<br>
    <b>% Violent:</b> {row.pct_violent:.1%}
    """

    is_extreme = row.pct_violent > 0.33

    folium.Circle(
        location=[row.lat, row.lng],
        radius=row.total_crimes * 0.35, # size = total crime
           volume
        fill=True,
        fill_color=colormap(row.pct_violent), # color = % violent
        fill_opacity=0.6,
        stroke=is_extreme,
        color="#800000" if is_extreme else None,
        weight=4 if is_extreme else 0,
        popup=folium.Popup(popup_html, max_width=300)
    ).add_to(bubble_layer)

    # Dot-only marker
    folium.CircleMarker(
        location=[row.lat, row.lng],
        radius=4,
```

```
        fill=True,
        fill_color="black",
        fill_opacity=0.8,
        stroke=False,
        popup=popup_html
    ).add_to(dot_layer)

# Add both layers
bubble_layer.add_to(m)
dot_layer.add_to(m)

colormap.caption = "% Violent Crimes"
colormap.add_to(m)
folium.LayerControl(collapsed=False).add_to(m)

m.save("crime_bubble_map_v2.html")
```