

Problem Set 6 - Waze Shiny Dashboard

Sitong Guo

2024-11-26

1. **ps6**: Due Sat 23rd at 5:00PM Central. Worth 100 points (80 points from questions, 10 points for correct submission and 10 points for code style) + 10 extra credit.

We use (*) to indicate a problem that we think might be time consuming.

Steps to submit (10 points on PS6)

1. “This submission is my work alone and complies with the 30538 integrity policy.” Add your initials to indicate your agreement: ****__****
2. “I have uploaded the names of anyone I worked with on the problem set [here](#)” ****__**** (2 point)
3. Late coins used this pset: ****__**** Late coins left after submission: ****__****
4. Before starting the problem set, make sure to read and agree to the terms of data usage for the Waze data [here](#).
5. Knit your `ps6.qmd` as a pdf document and name it `ps6.pdf`.
6. Submit your `ps6.qmd`, `ps6.pdf`, `requirements.txt`, and all created folders (we will create three Shiny apps so you will have at least three additional folders) to the gradescope repo assignment (5 points).
7. Submit `ps6.pdf` and also link your Github repo via Gradescope (5 points)
8. Tag your submission in Gradescope. For the Code Style part (10 points) please tag the whole corresponding section for the code style rubric.

Notes: see the [Quarto documentation \(link\)](#) for directions on inserting images into your knitted document.

IMPORTANT: For the App portion of the PS, in case you can not arrive to the expected functional dashboard we will need to take a look at your `app.py` file. You can use the following

code chunk template to “import” and print the content of that file. Please, don’t forget to also tag the corresponding code chunk as part of your submission!

```
def print_file_contents(file_path):
    """Print contents of a file."""
    try:
        with open(file_path, 'r') as f:
            content = f.read()
            print("`python`")
            print(content)
            print("`")
    except FileNotFoundError:
        print("`python`")
        print(f"Error: File '{file_path}' not found")
        print("`")
    except Exception as e:
        print("`python`")
        print(f"Error reading file: {e}")
        print("`")

print_file_contents("./top_alerts_map_byhour/app.py") # Change accordingly
```

```
RendererRegistry.enable('png')
```

Background

Data Download and Exploration (20 points)

1.

```
import zipfile
```

```
zip_path =
↪ 'C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/student30538/problem_sets/ps6/waze_da
with zipfile.ZipFile(zip_path, 'r') as zip_ref:

↪ zip_ref.extractall('C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/student30538/probl

sample_data_path =
↪ 'C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/student30538/problem_sets/ps6/waze_da
```

```

waze_sample_df = pd.read_csv(sample_data_path)

columns_to_exclude = ['ts', 'geo', 'geoWKT']
altair_data_types = [(col,
                      'Quantitative' if
                        ↪ pd.api.types.is_numeric_dtype(waze_sample_df[col])
                        ↪ else 'Nominal')
                     for col in waze_sample_df.columns if col not in
                        ↪ columns_to_exclude]
print("Variable Names and Data Types:")
print(altair_data_types)

```

Variable Names and Data Types:

```

[('Unnamed: 0', 'Quantitative'), ('city', 'Nominal'), ('confidence',
'Quantitative'), ('nThumbsUp', 'Quantitative'), ('street', 'Nominal'),
('uuid', 'Nominal'), ('country', 'Nominal'), ('type', 'Nominal'), ('subtype',
'Nominal'), ('roadType', 'Quantitative'), ('reliability', 'Quantitative'),
('magvar', 'Quantitative'), ('reportRating', 'Quantitative')]

```

2.

```

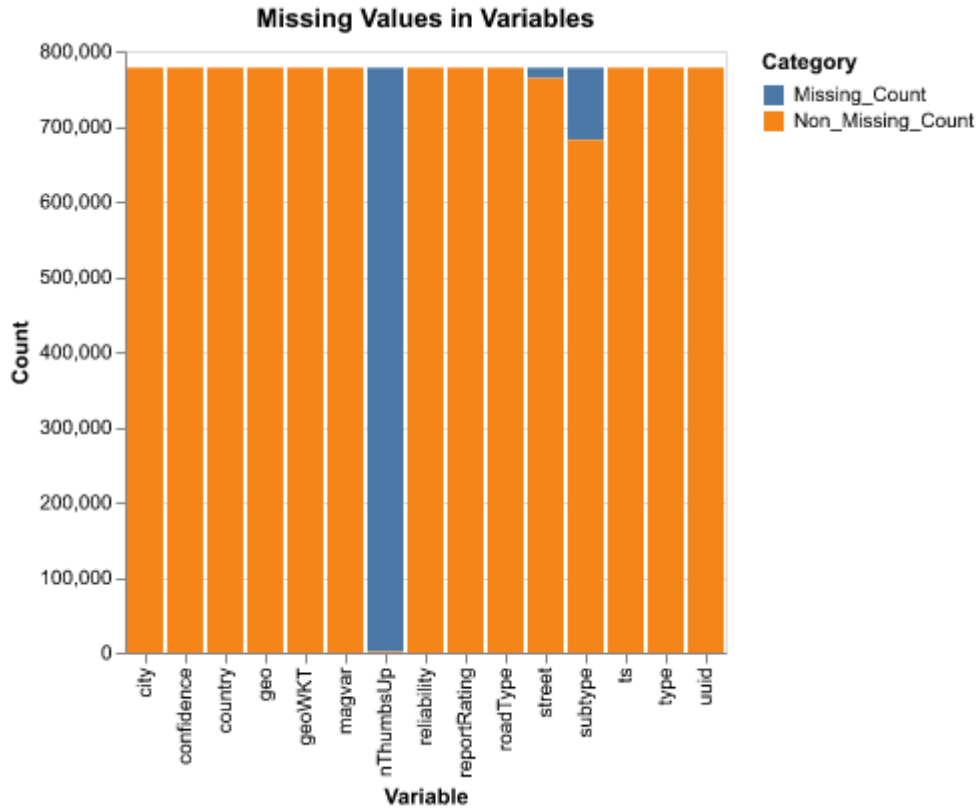
full_data_path =
↪ 'C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/student30538/problem_sets/ps6/waze_data.csv'
waze_full_df = pd.read_csv(full_data_path)

missing_data_summary = waze_full_df.isnull().sum().reset_index()
missing_data_summary.columns = ['Variable', 'Missing_Count']
missing_data_summary['Non_Missing_Count'] = len(waze_full_df) -
↪ missing_data_summary['Missing_Count']

stacked_data = pd.melt(missing_data_summary, id_vars=['Variable'],
                      value_vars=['Missing_Count', 'Non_Missing_Count'],
                      var_name='Category', value_name='Count')

chart = alt.Chart(stacked_data).mark_bar().encode(
    x='Variable:N',
    y='Count:Q',
    color='Category:N'
).properties(title='Missing Values in Variables')
chart

```



Variables with missing values are: nThumbsUp, street, subtype. Among which the most incomplete variable is nThumbsUp.

3. a,

```
unique_types = waze_full_df['type'].unique()
unique_subtypes = waze_full_df.groupby('type')['subtype'].unique()
print(f"Unique types: {unique_types}")
print(f"Unique subtypes: {unique_subtypes}")

# Identify types with NA subtype, check if there are sub-subs
na_subtypes = unique_subtypes.apply(lambda x: x[pd.isnull(x)])
na_subtypes_count = unique_subtypes.apply(lambda x: pd.isnull(x).any()).sum()
print(f"Number of types that have nan subtype:{na_subtypes_count}" )
```

```
Unique types: ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
Unique subtypes: type
ACCIDENT          [nan, ACCIDENT_MAJOR, ACCIDENT_MINOR]
HAZARD             [nan, HAZARD_ON_ROAD, HAZARD_ON_ROAD_CAR_STOPP...
```

```
JAM          [nan, JAM_HEAVY_TRAFFIC, JAM_MODERATE_TRAFFIC,...
ROAD_CLOSED  [nan, ROAD_CLOSED_EVENT, ROAD_CLOSED_CONSTRUCT...
Name: subtype, dtype: object
Number of types that have nan subtype:4
```

I can identify the types with sub-subs: HAZARD

b,

```
# Bulleted listed with the values at each layer given this hierarchy.
↪ Readable
hierarchy = {}
for t, subtypes in unique_subtypes.items():
    readable_type = t.replace("_", " ").title()
    readable_subtypes = [
        st.replace("_", " ").title().replace(readable_type, "").strip() if
↪ pd.notnull(st) else "Unclassified"
        for st in subtypes
    ]
    hierarchy[readable_type] = readable_subtypes
```

```
sub_subtype_hierarchy = {
    "Hazard": {
        "Unclassified": [],
        "On Road": ["Unclassified", "Car Stopped", "Construction", "Emergency
↪ Vehicle", "Ice", "Object", "Pot Hole", "Traffic Light Fault",
↪ "Lane Closed", "Road Kill"],
        "On Shoulder": ["Unclassified", "Car stopped", "Animals", "Missing
↪ Sign"],
        "Weather": ["Unclassified", "Flood", "Fog", "Heavy Snow", "Hail"]
    },
    "Accident": {
        "Unclassified": [],
        "Major": [],
        "Minor": [],
    },
    "Jam": {
        "Unclassified": [],
        "Heavy Traffic": [],
        "Moderate Traffic": [],
        "Stand Still Traffic": [],
        "Light Traffic": []
    },
}
```

```

    "Road Closed":{
        "Unclassified":[],
        "Event": [],
        "Construction": [],
        "Hazard": []
    }
}

print("Full Hierarchy with Sub-Subtypes:")
for t, subtypes in sub_subtype_hierarchy.items():
    print(f"- {t}")
    for subtype, sub_subtypes in subtypes.items():
        print(f"    - {subtype}")
        for sub_subtype in sub_subtypes:
            print(f"        - {sub_subtype}")

```

Full Hierarchy with Sub-Subtypes:

- Hazard
 - Unclassified
- On Road
 - Unclassified
 - Car Stopped
 - Construction
 - Emergency Vehicle
 - Ice
 - Object
 - Pot Hole
 - Traffic Light Fault
 - Lane Closed
 - Road Kill
- On Shoulder
 - Unclassified
 - Car stopped
 - Animals
 - Missing Sign
- Weather
 - Unclassified
 - Flood
 - Fog
 - Heavy Snow
 - Hail
- Accident

- Unclassified
- Major
- Minor
- Jam
 - Unclassified
 - Heavy Traffic
 - Moderate Traffic
 - Stand Still Traffic
 - Light Traffic
- Road Closed
 - Unclassified
 - Event
 - Construction
 - Hazard

c,

```
#Keep NA subtypes
```

Yes we should since that they do contain the information of those issues that are not readily classified yet.

4. a, b:

```
# Define the primary subtypes manually
primary_subtypes = {
    "Hazard": ["On Road", "On Shoulder", "Weather", "Unclassified"],
    "Accident": ["Major", "Minor", "Unclassified"],
    "Jam": ["Light Traffic", "Moderate Traffic", "Heavy Traffic", "Stand
    ↪ Still Traffic", "Unclassified"],
    "Road Closed": ["Event", "Construction", "Hazard", "Unclassified"]
}

def clean_type_and_subtypes(row):
    # Clean the type
    readable_type = row['type'].replace("_", " ").title()

    # Clean the subtype
    if pd.notnull(row['subtype']):
        readable_subtype = row['subtype'].replace("_", "
        ↪ ").title().replace(readable_type, "").strip() # Clean the subtype
    else:
```

```

        readable_subtype = "Unclassified"

# Initialize subsubtype as None
readable_subsubtype = None

# Check if the cleaned subtype is a sub
if readable_subtype in primary_subtypes.get(readable_type, []):
    readable_subsubtype = "Unclassified"
else:
    readable_subsubtype = readable_subtype

# For those under hazard, assign the proper Subtype in the dict!
for primary in primary_subtypes.get(readable_type, []):
    if primary in readable_subtype:
        readable_subtype = primary
        # Drop the subtype string in the subsub
        readable_subsubtype = readable_subsubtype.replace(primary,
↪ "").strip()
        break

    return pd.Series([readable_type, readable_subtype, readable_subsubtype])

crosswalk_df = waze_full_df[['type', 'subtype']].drop_duplicates()
crosswalk_df[['updated_type', 'updated_subtype', 'updated_subsubtype']] =
↪ crosswalk_df.apply(clean_type_and_subtypes, axis=1)

crosswalk_df

```

	type	subtype	updated_type	updated_subtype
0	JAM	NaN	Jam	Unclassified
1	ACCIDENT	NaN	Accident	Unclassified
2	ROAD_CLOSED	NaN	Road Closed	Unclassified
26	HAZARD	NaN	Hazard	Unclassified
122	ACCIDENT	ACCIDENT_MAJOR	Accident	Major
131	ACCIDENT	ACCIDENT_MINOR	Accident	Minor
148	HAZARD	HAZARD_ON_ROAD	Hazard	On Road
190	HAZARD	HAZARD_ON_ROAD_CAR_STOPPED	Hazard	On Road
240	HAZARD	HAZARD_ON_ROAD_CONSTRUCTION	Hazard	On Road
276	HAZARD	HAZARD_ON_ROAD_EMERGENCY_VEHICLE	Hazard	On Road
302	HAZARD	HAZARD_ON_ROAD_ICE	Hazard	On Road
303	HAZARD	HAZARD_ON_ROAD_OBJECT	Hazard	On Road

	type	subtype	updated_type	updated_subtype
355	HAZARD	HAZARD_ON_ROAD_POT_HOLE	Hazard	On Road
478	HAZARD	HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT	Hazard	On Road
483	HAZARD	HAZARD_ON_SHOULDER	Hazard	On Shoulder
485	HAZARD	HAZARD_ON_SHOULDER_CAR_STOPPED	Hazard	On Shoulder
854	HAZARD	HAZARD_WEATHER	Hazard	Weather
857	HAZARD	HAZARD_WEATHER_FLOOD	Hazard	Weather
858	JAM	JAM_HEAVY_TRAFFIC	Jam	Heavy Traffic
1122	JAM	JAM_MODERATE_TRAFFIC	Jam	Moderate Traffic
1184	JAM	JAM_STAND_STILL_TRAFFIC	Jam	Stand Still
1335	ROAD_CLOSED	ROAD_CLOSED_EVENT	Road Closed	Event
1905	HAZARD	HAZARD_ON_ROAD_LANE_CLOSED	Hazard	On Road
5557	HAZARD	HAZARD_WEATHER_FOG	Hazard	Weather
7331	ROAD_CLOSED	ROAD_CLOSED_CONSTRUCTION	Road Closed	Construction
21443	HAZARD	HAZARD_ON_ROAD_ROAD_KILL	Hazard	On Road
21447	HAZARD	HAZARD_ON_SHOULDER_ANIMALS	Hazard	On Shoulder
21940	HAZARD	HAZARD_ON_SHOULDER_MISSING_SIGN	Hazard	On Shoulder
38546	JAM	JAM_LIGHT_TRAFFIC	Jam	Light Traffic
44216	HAZARD	HAZARD_WEATHER_HEAVY_SNOW	Hazard	Weather
54556	ROAD_CLOSED	ROAD_CLOSED_HAZARD	Road Closed	Hazard
229005	HAZARD	HAZARD_WEATHER_HAIL	Hazard	Weather

c,

```
merged_data = pd.merge(waze_full_df, crosswalk_df, on=['type', 'subtype'],
                        how='left')

# Count rows for accident unclassified
accident_unclassified_count = merged_data[
    (merged_data['updated_type'] == 'Accident') &
    (merged_data['updated_subtype'] == 'Unclassified')
].shape[0]

print(f"Number of Accident-Unclassified: {accident_unclassified_count}")
```

Number of Accident-Unclassified: 24359

d,

```
# Extract unique combinations of type-subtype from both data
crosswalk_combinations = crosswalk_df[['type', 'subtype']].drop_duplicates()
```

```
merged_combinations = merged_data[['type', 'subtype']].drop_duplicates()

# Compare unique combination
are_combinations_equal = crosswalk_combinations.equals(merged_combinations)

print(f"Do they have same values in type-subtype?
↪ '{are_combinations_equal}'")
```

Do they have same values in type-subtype? 'True'

App #1: Top Location by Alert Type Dashboard (30 points)

1.

- a. Prompt to ChatGPT: Write a regex pattern to extract latitude and longitude from a string formatted like this: "POINT (longitude latitude)".

Response: The regex pattern is:

python: `r"POINT ((-?\d+\.\d+) (-?\d+\.\d+))"` Explanation: POINT matches the literal string. (and) escape parentheses. -? matches an optional negative sign. \d+ matches one or more digits. \. matches a literal dot. Captures the first number as longitude and the second as latitude.

```
import re
```

```
# Delete that space after POINT, no spaces between point and (
# Extract latitude and longitude
merged_data['latitude'] =
↪ merged_data['geo'].str.extract(r"POINT\((-?\d+\.\d+) (-?\d+\.\d+)\)",
↪ expand=True)[1].astype(float)
merged_data['longitude'] =
↪ merged_data['geo'].str.extract(r"POINT\((-?\d+\.\d+) (-?\d+\.\d+)\)",
↪ expand=True)[0].astype(float)
merged_data['longitude']
```

```
0      -87.676685
1      -87.624816
2      -87.614122
3      -87.680139
4      -87.735235
      ...
```

```

778089    -87.615862
778090    -87.615882
778091    -87.645584
778092    -87.598843
778093    -87.598843
Name: longitude, Length: 778094, dtype: float64

```

b.

```

# Create bins with step size 0.01
merged_data['latitude'] = (merged_data['latitude'] // 0.01 * 0.01).round(2)
merged_data['longitude'] = (merged_data['longitude'] // 0.01 * 0.01).round(2)

# most frequent bin combination
most_frequent_bin = merged_data.groupby(['latitude',
    ↪ 'longitude']).size().idxmax()
print(f"The most frequent bin is: {most_frequent_bin}")

```

The most frequent bin is: (41.96, -87.75)

c.

```

# Aggregate data
aggregated_df = (
    merged_data.groupby(['latitude', 'longitude', 'updated_type',
    ↪ 'updated_subtype'])
    .size()
    .reset_index(name='alert count')
)

output_path =
    ↪ "C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/PS6_SitongGuo/top_alerts_map/top_aler
aggregated_df.to_csv(output_path, index=False)

# Check aggregation level and rows
print(f"Aggregation level: latitude, longitude, type, subtype. \nNumber of
    ↪ rows in the DataFrame: {aggregated_df.shape[0]}")

```

Aggregation level: latitude, longitude, type, subtype.
 Number of rows in the DataFrame: 6764

2.

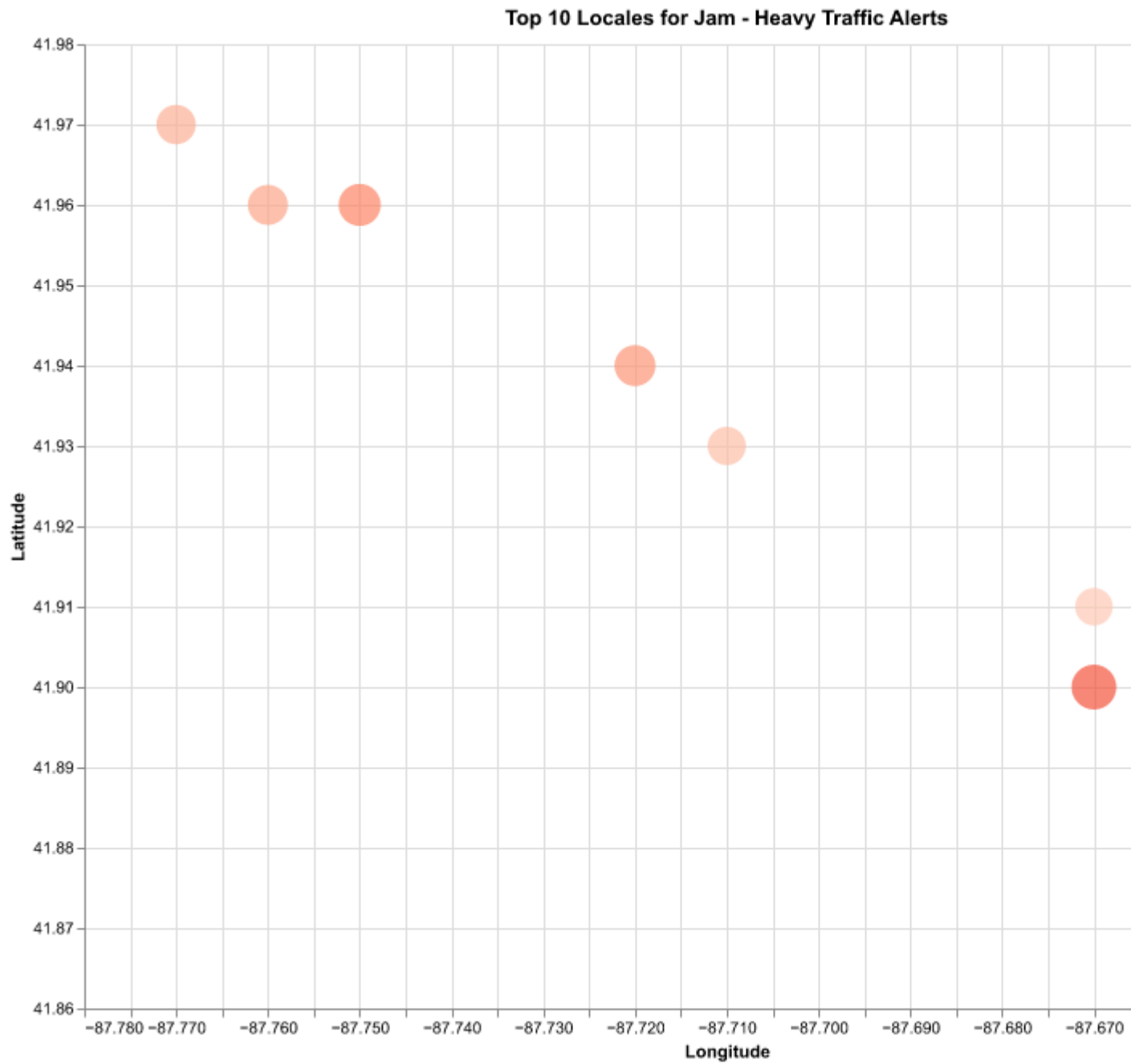
```

# Filter for Jam Heavy Traffic
filtered_df = (
    aggregated_df[
        (aggregated_df['updated_type'] == 'Jam') &
        (aggregated_df['updated_subtype'] == 'Heavy Traffic')
    ]
    .nlargest(10, 'alert count')
)

# Define axis ranges for better presentation!
latitude_min, latitude_max = filtered_df['latitude'].min() - 0.01,
    ↪ filtered_df['latitude'].max() + 0.01
longitude_min, longitude_max = filtered_df['longitude'].min() - 0.01,
    ↪ filtered_df['longitude'].max() + 0.01

chart = alt.Chart(filtered_df).mark_circle().encode(
    x=alt.X('longitude:Q', title='Longitude',
    ↪ scale=alt.Scale(domain=[longitude_min, longitude_max])),
    y=alt.Y('latitude:Q', title='Latitude',
    ↪ scale=alt.Scale(domain=[latitude_min, latitude_max])),
    size=alt.Size('alert count:Q', title='Alert Count',
    ↪ scale=alt.Scale(range=[50, 1000])),
    color=alt.Color('alert count:Q', scale=alt.Scale(scheme='reds')),
    tooltip=['latitude:Q', 'longitude:Q', 'alert count:Q']
).properties(
    title='Top 10 Locales for Jam - Heavy Traffic Alerts',
    width=800,
    height=600
)
chart

```



3.

a.

```

import requests
# Download using python
url =
    ↪ "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"
file_path =
    ↪ "C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/PS6_SitongGuo/top_alerts_map/chicago-1

response = requests.get(url)
if response.status_code == 200:
    with open(file_path, "wb") as f:
        f.write(response.content)
        print("successful!")
else:
    print(f"Failed:{response.status_code}")

```

successful!

b.

```

import json
# MODIFY ACCORDINGLY

with open(file_path) as f:
    chicago_geojson = json.load(f)

geo_data = alt.Data(values=chicago_geojson["features"])

```

4.

```

# Create map layer
map_layer = alt.Chart(geo_data).mark_geoshape(
    fill="lightgray",
    stroke="white",
    strokeWidth=0.5
).properties(
    width=800,
    height=600
).project(
    type="equirectangular" # Projection type
)

```

```

# scatter plot layer
scatter_layer = alt.Chart(filtered_df).mark_circle().encode(

    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size('alert count:Q', scale=alt.Scale(range=[50, 500])),
    color=alt.Color('alert count:Q', scale=alt.Scale(scheme='reds')),
    tooltip=['latitude', 'longitude', 'alert count']
).properties(
    title="Top 10 Jam - Heavy Traffic Alerts in Chicago"
)

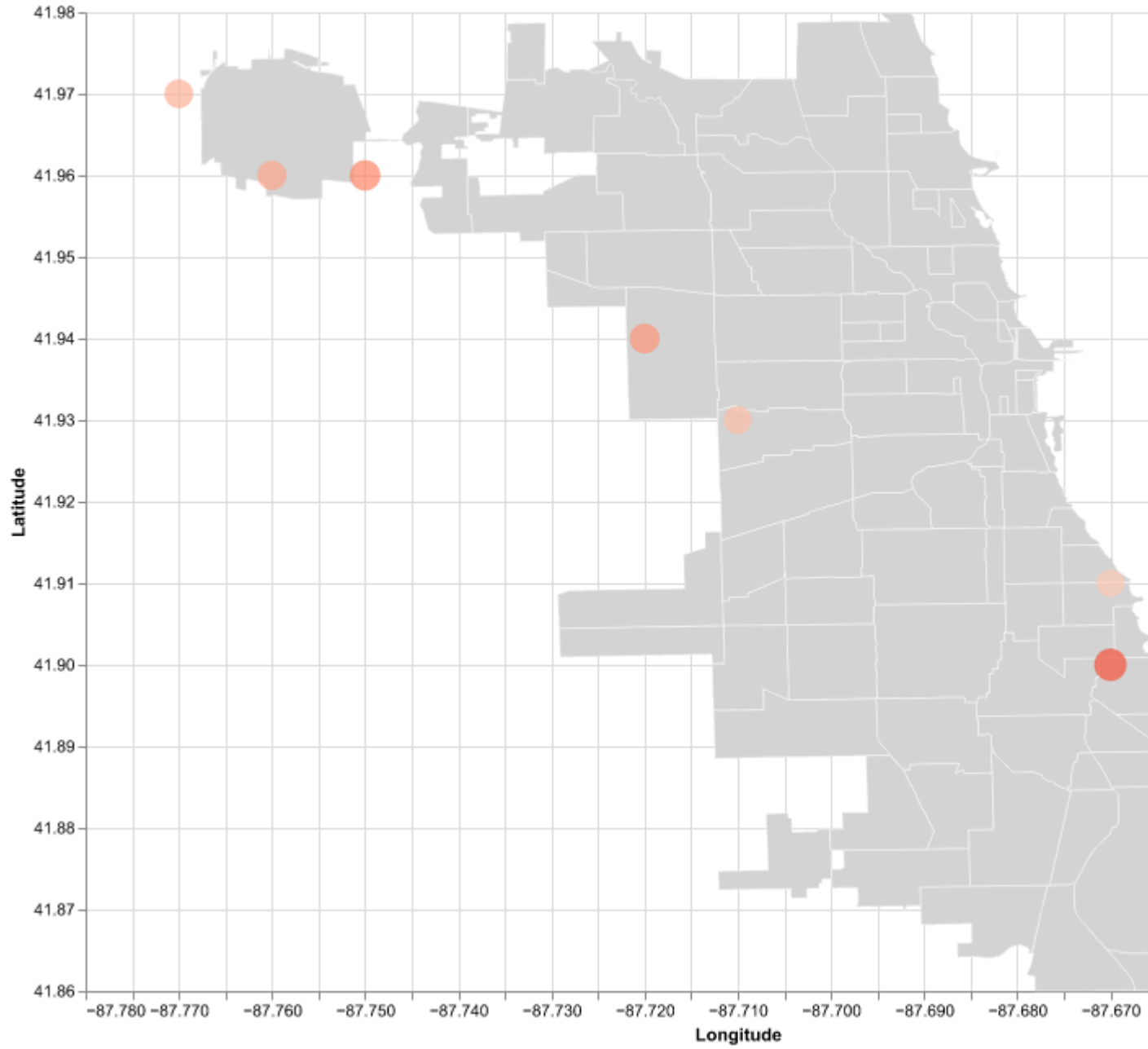
# Set axis domains for alignment
lat_min, lat_max = filtered_df['latitude'].min() - 0.01,
↳ filtered_df['latitude'].max() + 0.01
lon_min, lon_max = filtered_df['longitude'].min() - 0.01,
↳ filtered_df['longitude'].max() + 0.01

# Combine the layers!!
combined_plot = (map_layer + scatter_layer).configure_view(
    stroke=None
    # Remove the default borders around the map
).properties(
    width=800,
    height=600
).configure_title(
    fontSize=16,
    anchor="start"
).encode(
    x=alt.X('longitude:Q', scale=alt.Scale(domain=[lon_min, lon_max])),
    ↳ title="Longitude"),
    y=alt.Y('latitude:Q', scale=alt.Scale(domain=[lat_min, lat_max])),
    ↳ title="Latitude")
)

combined_plot

```

Top 10 Jam - Heavy Traffic Alerts in Chicago



5.

a.

Top Alerts Map

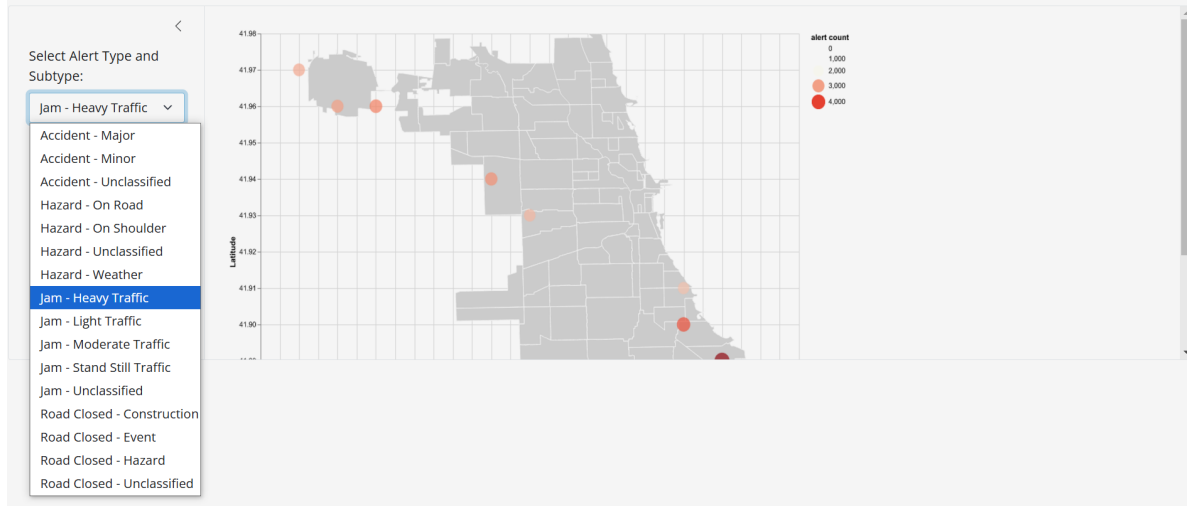


Figure 1: 2-5-1

There are 16 combinations in the menu.

b.

Top Alerts Map

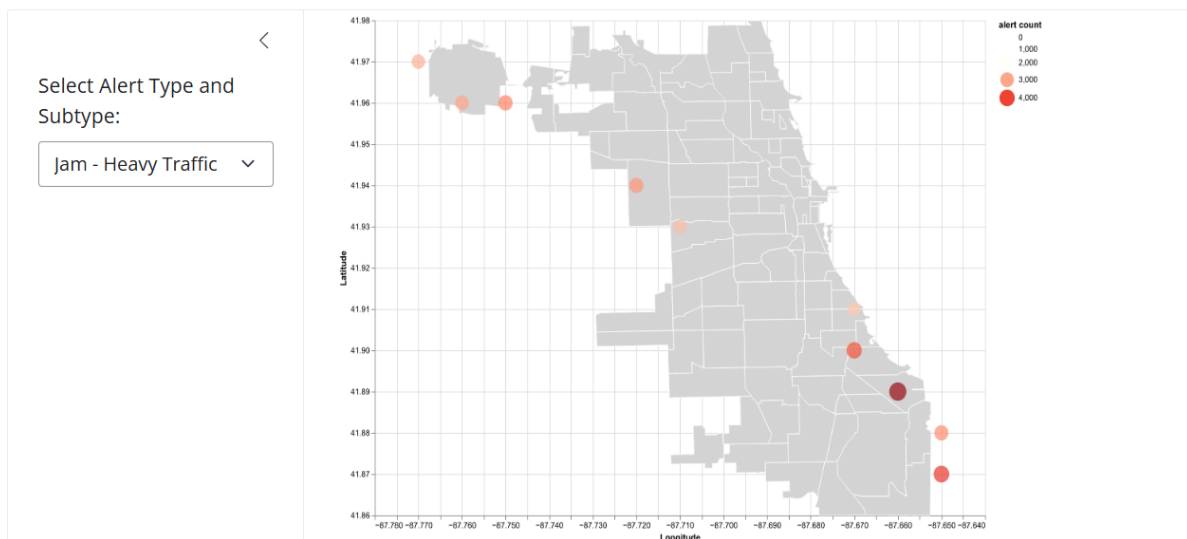


Figure 2: 2-5-2

c.

Top Alerts Map

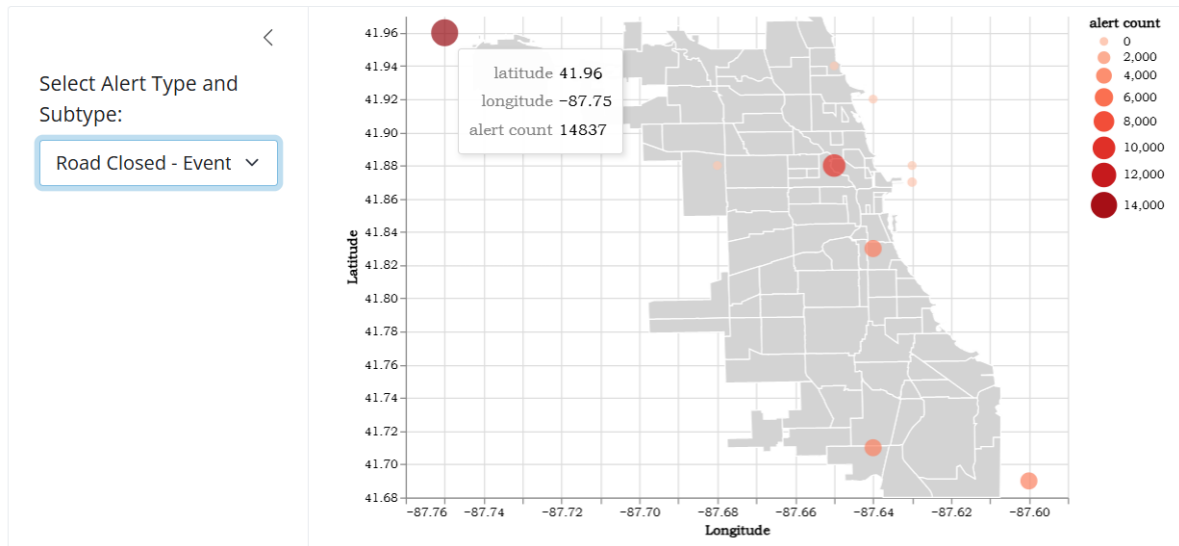


Figure 3: 2-5-3

```
filtered_df1 = (
    aggregated_df[
        (aggregated_df['updated_type'] == 'Road Closed') &
        (aggregated_df['updated_subtype'] == 'Event')
    ]
    .nlargest(10, 'alert count')
)

scatter_layer = alt.Chart(filtered_df1).mark_circle().encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size('alert count:Q', scale=alt.Scale(range=[50, 500])),
    color=alt.Color('alert count:Q', scale=alt.Scale(scheme='reds')),
    tooltip=['latitude', 'longitude', 'alert count']
).properties(
    title="Top 10 Road Closure - Event Alerts in Chicago"
)

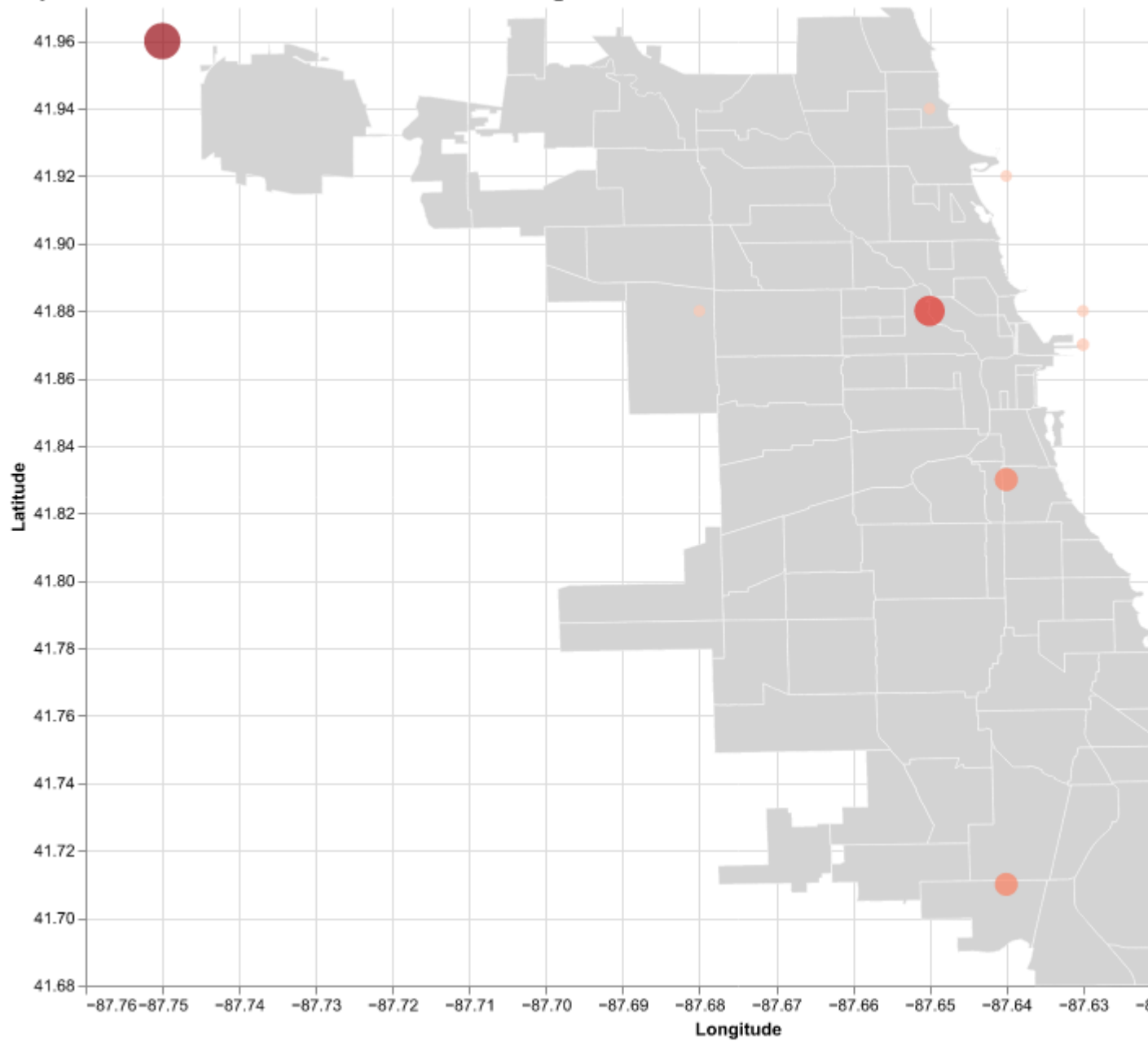
lat_min, lat_max = filtered_df1['latitude'].min() - 0.01,
    filtered_df1['latitude'].max() + 0.01
lon_min, lon_max = filtered_df1['longitude'].min() - 0.01,
    filtered_df1['longitude'].max() + 0.01
combined_plot1 = (map_layer + scatter_layer).configure_view(
```

```

    stroke=None
    # Remove the default borders around the map
).properties(
    width=800,
    height=600
).configure_title(
    fontSize=16,
    anchor="start"
).encode(
    x=alt.X('longitude:Q', scale=alt.Scale(domain=[lon_min, lon_max]),
    ↪ title="Longitude"),
    y=alt.Y('latitude:Q', scale=alt.Scale(domain=[lat_min, lat_max]),
    ↪ title="Latitude")
)
combined_plot1

```

Top 10 Road Closure - Event Alerts in Chicago



From the plot, the most common area (latitude, longitude) is (41.96, -87.75) .

d. Question: which approximate area is more likely for minor traffic accidents to occur?

Top Alerts Map

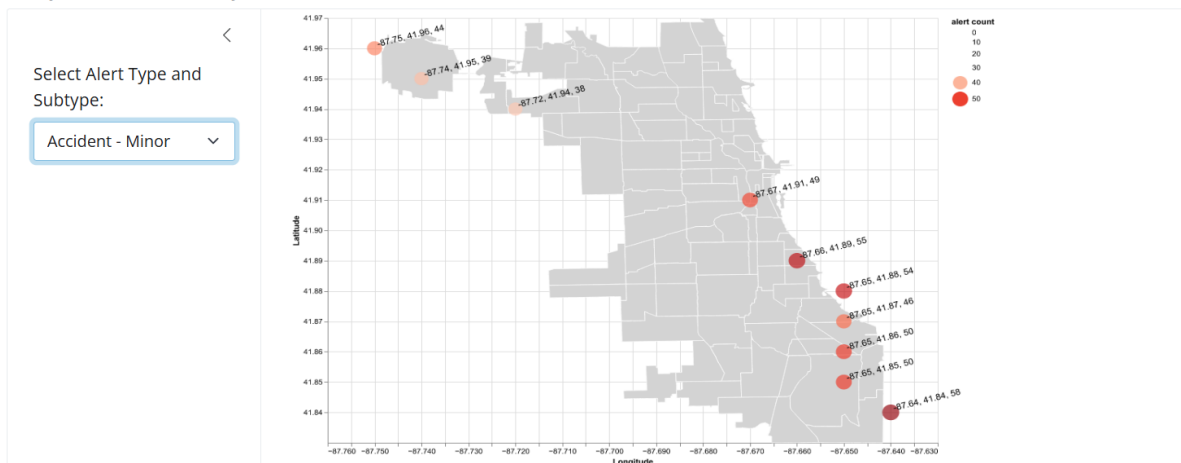


Figure 4: 2-5-4

The area is the southeast area of Chicago, judging from the geographical distribution.

- e. Add the subsubtype column to provide more granular analysis on the hazard subtypes. Or, add the roadtype or the ts columns to examine insights on relationship between alerts and road types or the timing.

The code in App1:

```
def print_file_contents(file_path):
    """Print contents of a file."""
    try:
        with open(file_path, 'r') as f:
            content = f.read()
            print("`python")
            print(content)
            print("`")
    except FileNotFoundError:
        print("`python")
        print(f"Error: File '{file_path}' not found")
        print("`")
    except Exception as e:
        print("`python")
        print(f"Error reading file: {e}")
        print("`")
```

```
print_file_contents("C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/PS6_SitongGuo/top_alerts_map/
↪ # Change accordingly
```

```
```python
from shiny import App, ui, render
from shinywidgets import render_altair, output_widget
import pandas as pd
import altair as alt
import json
import tempfile

Load data
file_path =
"C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/PS6_SitongGuo/top_alerts_map/top_alerts_map/
aggregated_df = pd.read_csv(file_path)

Define the UI
app_ui = ui.page_fluid(
 ui.panel_title("Top Alerts Map"),
 ui.layout_sidebar(
 ui.sidebar(
 ui.input_select(
 "alert_choice",
 "Select Alert Type and Subtype:",
 sorted([
 f"{row['updated_type']} - {row['updated_subtype']}"
 for _, row in aggregated_df.iterrows()
]),
 selected="Jam - Heavy Traffic",
)
),
 output_widget("map_plot")
),
)

def server(input, output, session):
 @output
 @render_altair
 def map_plot():
 # Filter data for the selected type and subtype
 selected_data = (
```

```

 aggregated_df[
 (aggregated_df["updated_type"] + " - " +
 aggregated_df["updated_subtype"])
 == input.alert_choice()
]
 .nlargest(10, "alert count")
 .reset_index(drop=True)
)

Scatter Plot
scatter = (
 alt.Chart(selected_data)
 .mark_circle(size=200)
 .encode(
 longitude='longitude:Q',
 latitude='latitude:Q',
 size=alt.Size('alert count:Q', scale=alt.Scale(range=[50,
500])),
 color=alt.Color('alert count:Q',
scale=alt.Scale(scheme='reds')),
 tooltip=['latitude', 'longitude', 'alert count']
)
)

Load and prepare the Chicago map,
chicago_geojson_path =
"C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/PS6_SitongGuo/top_alerts_map/chicago_geojson.json"
with open(chicago_geojson_path) as f:
 chicago_geojson = json.load(f)

geo_data = alt.Data(values=chicago_geojson["features"])

chicago_map = (
 alt.Chart(geo_data).mark_geoshape(fill="lightgray",
stroke="white")
 .encode()
 .properties(width=800, height=600)
 .project(type="equirectangular")
)

Combine the map and scatter plot
Set axis domains for alignment

```

```

lat_min, lat_max = selected_data['latitude'].min() - 0.01,
selected_data['latitude'].max() + 0.01
lon_min, lon_max = selected_data['longitude'].min() - 0.01,
selected_data['longitude'].max() + 0.01

Combine the layers!!
combined_plot = (chicago_map + scatter).configure_view(
 stroke=None
 # Remove the default borders around the map
).properties(
 width=550,
 height=400
).configure_title(
 fontSize=16,
 anchor="start"
).encode(
 x=alt.X('longitude:Q', scale=alt.Scale(domain=[lon_min,
lon_max])), title="Longitude"),
 y=alt.Y('latitude:Q', scale=alt.Scale(domain=[lat_min, lat_max])),
 title="Latitude")
)
return combined_plot

app = App(app_ui, server)

...

```

## App #2: Top Location by Alert Type and Hour Dashboard (20 points)

1.
  - a. No. The ts column contains highly specified granular data (to minute and second). Collapsing by ts would create numerous unique rows for every specific timestamp as each moment is unique. This would not aggregate data meaningfully and will even increase computational complexity.
  - b.

```

hour_data = merged_data
hour_data['ts'] = pd.to_datetime(hour_data['ts'])

```



```
hour_data['hour'] = hour_data['ts'].dt.strftime('%H:00')
```

```
Collapse
collapsed_data = (
 hour_data.groupby(['hour', 'updated_type', 'updated_subtype',
 ↪ 'longitude', 'latitude'])
 .size()
 .reset_index(name='alert_count')
)
```

```
top 10 locations
top_10_data = (
 collapsed_data
 .sort_values(['hour', 'updated_type', 'updated_subtype', 'alert_count'],
 ↪ ascending=[True, True, True, False])
 .groupby(['hour', 'updated_type', 'updated_subtype'])
 .head(10)
)
```

```
top_10_data.to_csv("C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/PS6_SitongGuo/top_alerts.csv",
 ↪ index=False)
```

```
print(f"Number of rows in the collapsed dataset: {len(top_10_data)}")
```

Number of rows in the collapsed dataset: 3201

c.

```
heavy_traffic_data = top_10_data[
 (top_10_data['updated_type'] == 'Jam') &
 (top_10_data['updated_subtype'] == 'Heavy Traffic')
]
```

```
Select three times of day
times = ['08:00', '12:00', '18:00']
```

```
Create a plot for each time
plots = []
for time in times:
 # specific hour
```

```

data_for_time = heavy_traffic_data[heavy_traffic_data['hour'] == time]

scatter plot for top 10 locations
scatter_layer = alt.Chart(data_for_time).mark_circle().encode(
 longitude='longitude:Q',
 latitude='latitude:Q',
 size=alt.Size('alert_count:Q', scale=alt.Scale(range=[50, 500])),
 color=alt.Color('alert_count:Q', scale=alt.Scale(scheme='reds')),
 tooltip=['latitude', 'longitude', 'alert_count']
).properties(
 title = f"Top 10 Jam - Heavy Traffic Alerts by {time} in Chicago"
)

map_layer = map_layer

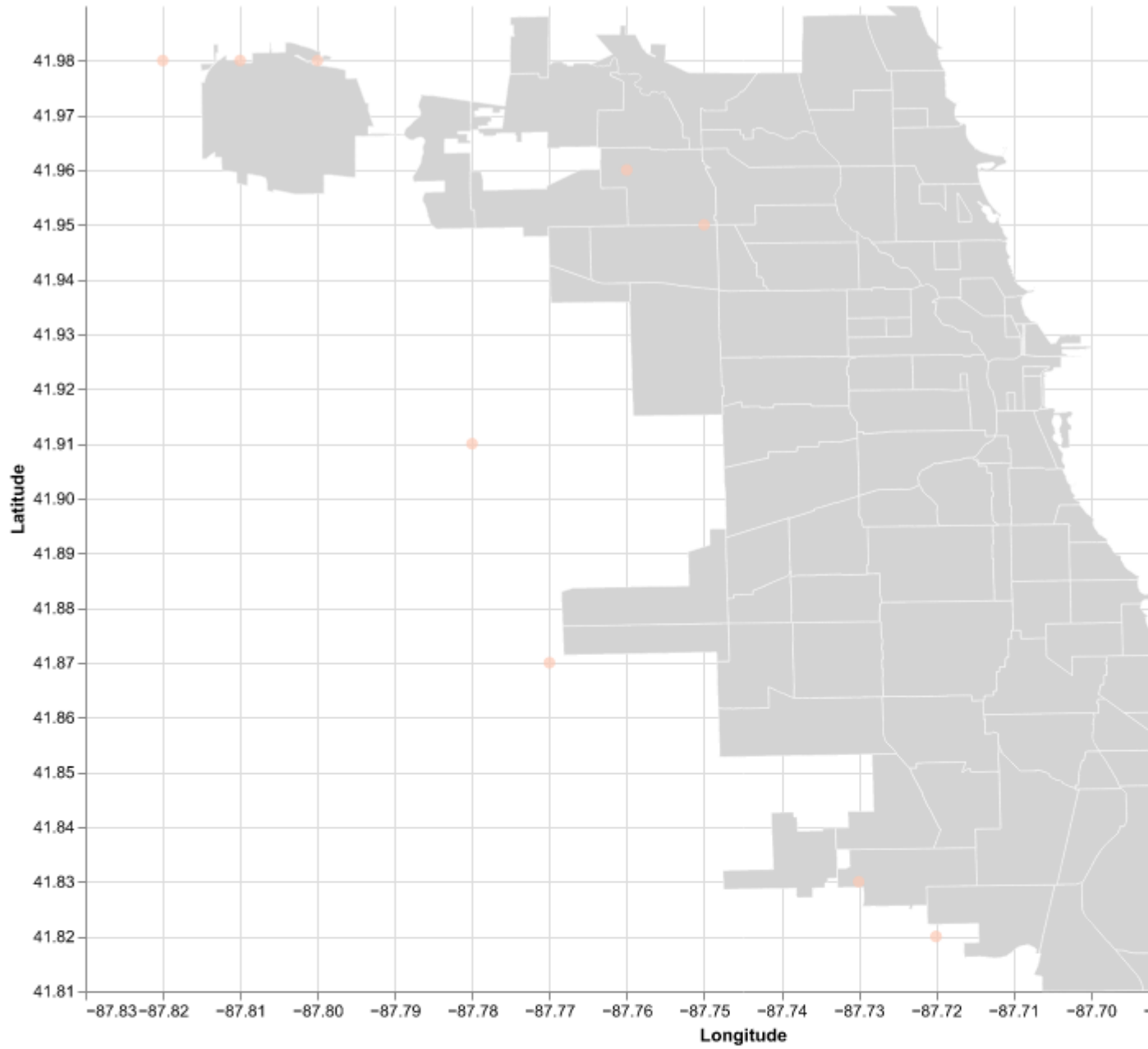
Set axis domains for alignment
lat_min, lat_max = data_for_time['latitude'].min() - 0.01,
↪ data_for_time['latitude'].max() + 0.01
lon_min, lon_max = data_for_time['longitude'].min() - 0.01,
↪ data_for_time['longitude'].max() + 0.01

Combine the layers!!
plot = (map_layer + scatter_layer).encode(
 x=alt.X('longitude:Q', scale=alt.Scale(domain=[lon_min, lon_max])),
↪ title="Longitude"),
 y=alt.Y('latitude:Q', scale=alt.Scale(domain=[lat_min, lat_max])),
↪ title="Latitude")
).properties(
 width=800,
 height=600
)
plots.append(plot)

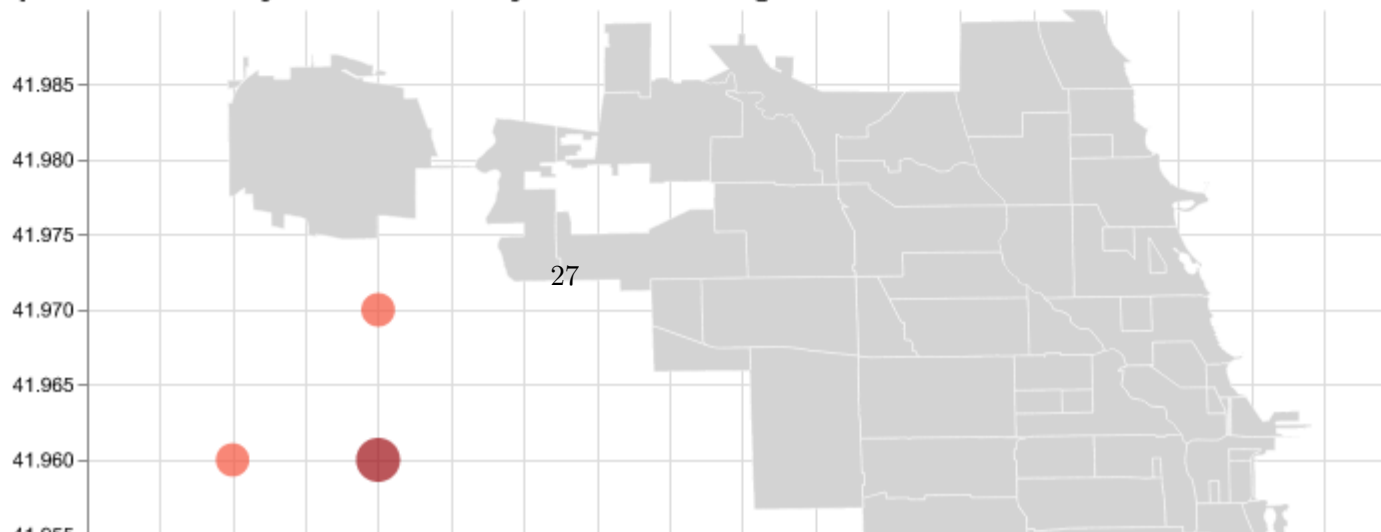
Concatenate the plots vertically
final_plot = alt.vconcat(*plots).configure_view(
 stroke=None
).configure_title(
 fontSize=16,
 anchor="start"
)
final_plot

```

**Top 10 Jam - Heavy Traffic Alerts by 08:00 in Chicago**



**Top 10 Jam - Heavy Traffic Alerts by 12:00 in Chicago**



2.

## Top Alerts Map by Hour

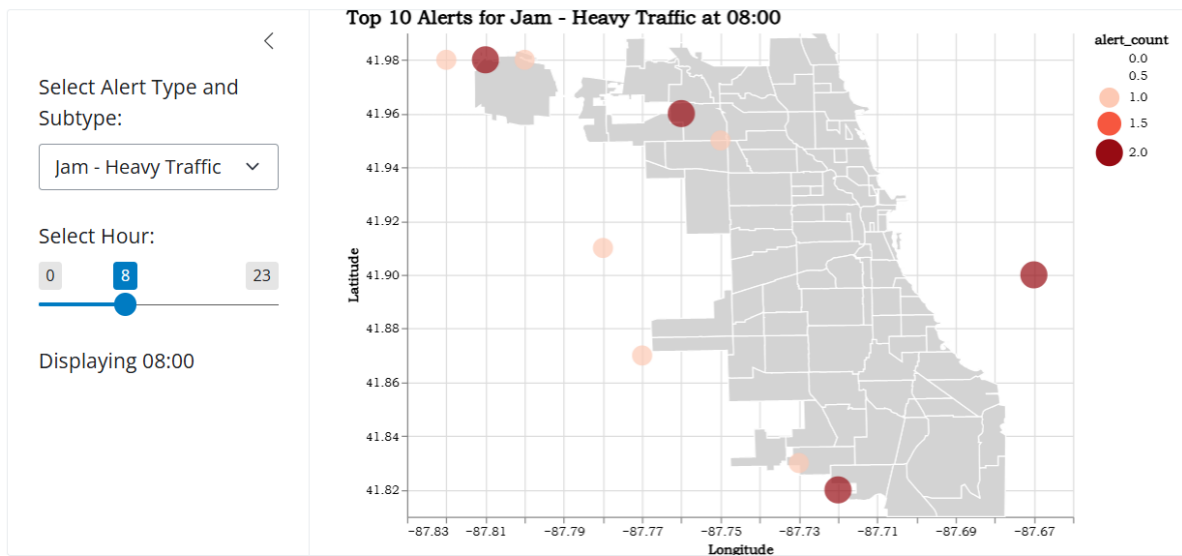
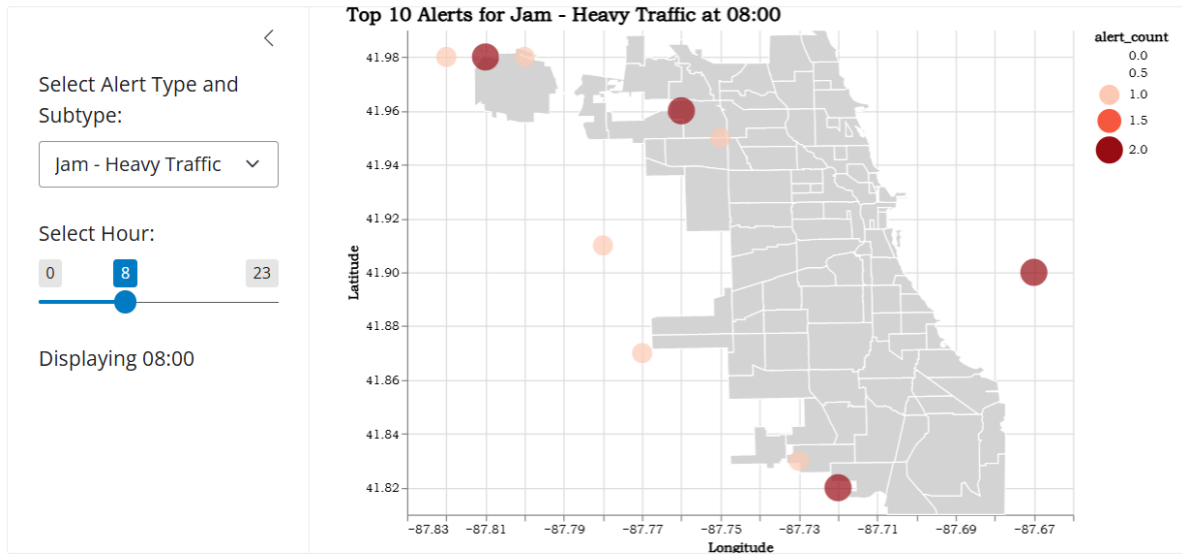


Figure 5: 3-2-1

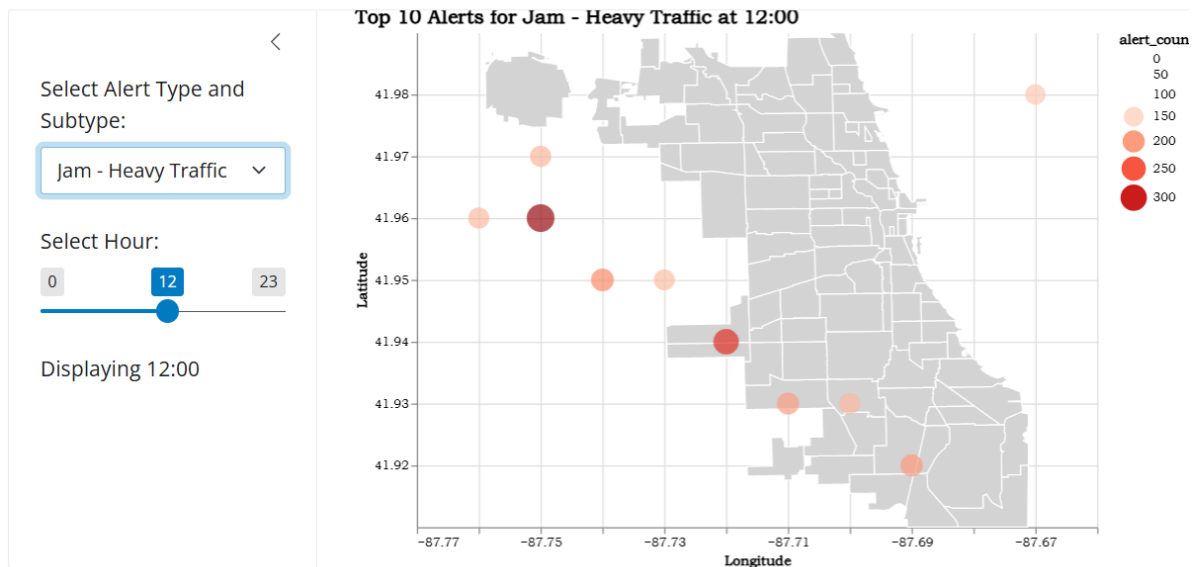
a.

b.

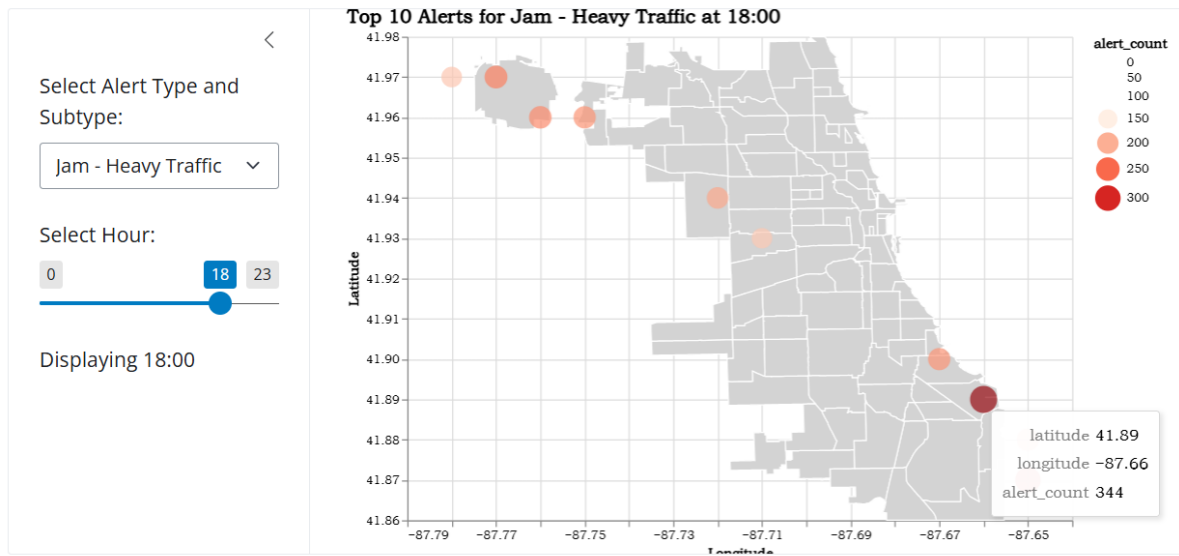
## Top Alerts Map by Hour



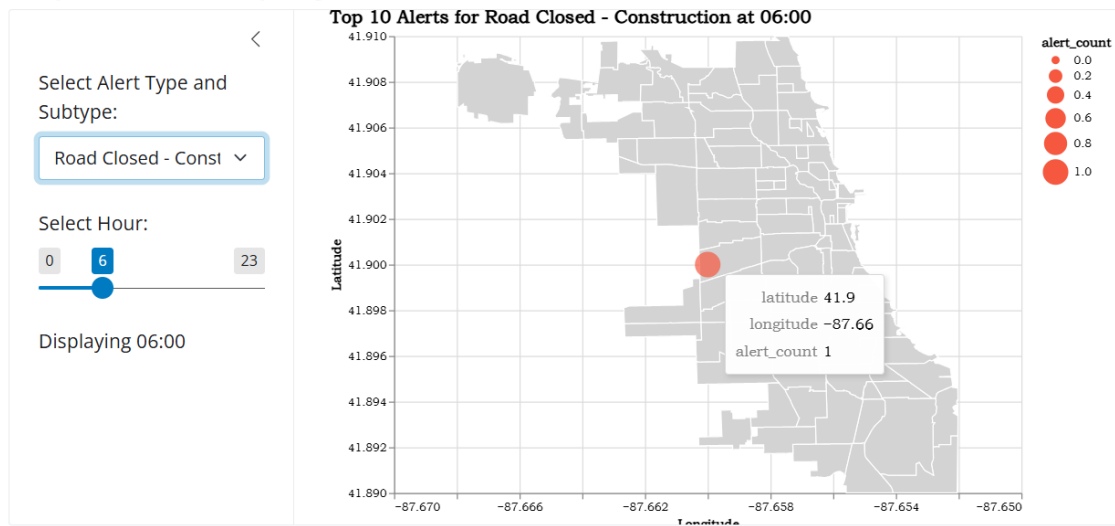
## Top Alerts Map by Hour



## Top Alerts Map by Hour

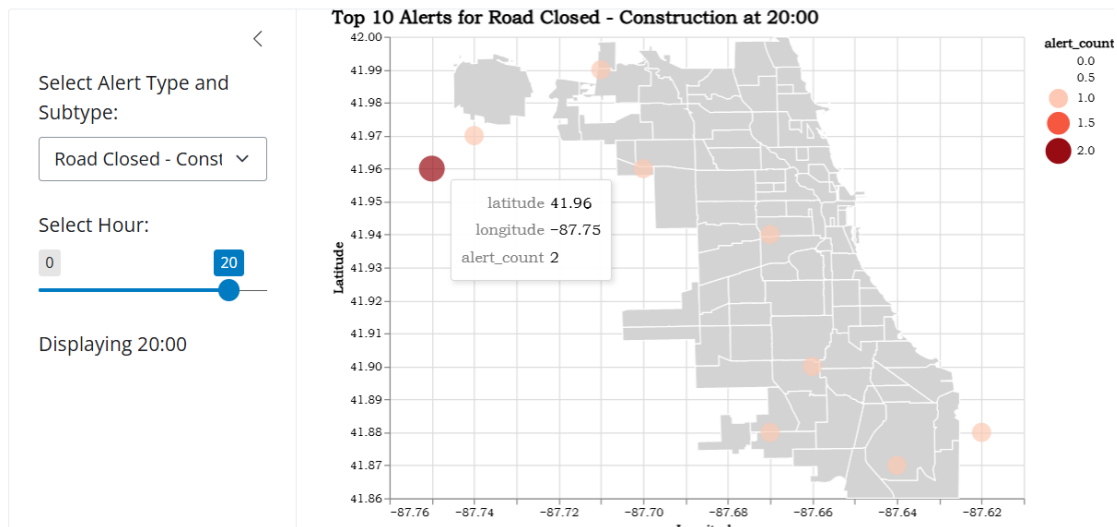


## Top Alerts Map by Hour



C.

## Top Alerts Map by Hour



From the plots showing 6am and 8 pm, we can say that is the night hours seeing more constructions on road.

The code for app2:

```
def print_file_contents(file_path):
 """Print contents of a file."""
 try:
 with open(file_path, 'r') as f:
 content = f.read()
 print("`python`")
 print(content)
 print("`")
 except FileNotFoundError:
 print("`python`")
 print(f"Error: File '{file_path}' not found")
 print("`")
 except Exception as e:
 print("`python`")
 print(f"Error reading file: {e}")
 print("`")

print_file_contents("C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/PS6_SitongGuo/top_alerts.txt")
Change accordingly
```

`python`

```

from shiny import App, ui, render
import pandas as pd
from shinywidgets import render_altair, output_widget
import altair as alt
import tempfile
import json

data_path =
"C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/PS6_SitongGuo/top_alerts_map_byhour/top_a
data = pd.read_csv(data_path)
data['type_subtype'] = data['updated_type'] + ' - ' + data['updated_subtype']

type_subtype_choices = sorted([
 f"{row['updated_type']} - {row['updated_subtype']}"
 for _, row in data[['updated_type',
 'updated_subtype']].drop_duplicates().iterrows()
])

UI
app_ui = ui.page_fluid(
 ui.panel_title("Top Alerts Map by Hour"),
 ui.layout_sidebar(
 ui.sidebar(
 ui.input_select(
 "alert_choice",
 "Select Alert Type and Subtype:",
 type_subtype_choices,
 selected=type_subtype_choices[0], # Default selected value
),
 ui.input_slider(
 "hour_slider",
 "Select Hour:",
 min=0,
 max=23,
 value=12, # Default value (12:00 PM)
 step=1,
),
 ui.output_text("hour_label") # To display the selected hour
),
 output_widget("hourly_plot")
),
)

```



```

server
def server(input, output, session):
 @output
 @render.text
 def hour_label():
 return f"Displaying {input.hour_slider():02}:00"

 @output
 @render.altair
 def hourly_plot():
 selected_data = data[
 (data["type_subtype"] == input.alert_choice()) &
 (data["hour"] == f"{input.hour_slider():02}:00")
]

 scatter_layer = alt.Chart(selected_data).mark_circle().encode(
 longitude="longitude:Q",
 latitude="latitude:Q",
 size=alt.Size("alert_count:Q", scale=alt.Scale(range=[50, 500])),
 color=alt.Color("alert_count:Q", scale=alt.Scale(scheme="reds")),
 tooltip=["latitude", "longitude", "alert_count"]
).properties(
 width=800,
 height=600,
 title=f"Top 10 Alerts for {input.alert_choice()} at {input.hour_slider():02}:00"
)

 # Load and prepare the Chicago map,
 chicago_geojson_path =
 "C:/Users/RedthinkerDantler/Documents/GitHub/DPPP2/PS6_SitongGuo/top_alerts_map/chicago_geojson.json"
 with open(chicago_geojson_path) as f:
 chicago_geojson = json.load(f)
 geo_data = alt.Data(values=chicago_geojson["features"])
 chicago_map = (
 alt.Chart(geo_data).mark_geoshape(fill="lightgray",
 stroke="white")
 .encode()
 .properties(width=800, height=600)
 .project(type="equirectangular")
)

```

```

lat_min, lat_max = selected_data['latitude'].min() - 0.01,
selected_data['latitude'].max() + 0.01
lon_min, lon_max = selected_data['longitude'].min() - 0.01,
selected_data['longitude'].max() + 0.01
Combine the layers!!
combined_plot = (chicago_map + scatter_layer).configure_view(
 stroke=None
).properties(
 width=550,
 height=400
).configure_title(
 fontSize=16,
 anchor="start"
).encode(
 x=alt.X('longitude:Q', scale=alt.Scale(domain=[lon_min,
lon_max])), title="Longitude"),
 y=alt.Y('latitude:Q', scale=alt.Scale(domain=[lat_min, lat_max])),
 title="Latitude")
)
return combined_plot

app = App(app_ui, server)
...

```

### App #3: Top Location by Alert Type and Hour Dashboard (20 points)

1.
  - a. Yes, this way improves efficiency in processing data, also maintain certain granularity in time window analyses. Dividing a day into major components characterize the pattern of traffic situations while avoid going into overly details and confusing the user.
  - b.

```

ranged_data = collapsed_data
ranged_data['hour'] = pd.to_datetime(ranged_data['hour'], format='%H:%M')

sixnine_data = ranged_data[ranged_data['hour'].dt.hour.between(6, 9)]

```

```

sixnine_data = sixnine_data[
 (sixnine_data["updated_type"] == "Jam") &
 (sixnine_data["updated_subtype"] == "Heavy Traffic")
]

sixnine_data = sixnine_data.groupby(['latitude', 'longitude'],
 ↪ as_index=False)['alert_count'].sum()

top_10_location3 = sixnine_data.nlargest(10, 'alert_count')

```

```

scatter_plot3 = alt.Chart(top_10_location3).mark_circle().encode(
 longitude='longitude:Q',
 latitude='latitude:Q',
 size=alt.Size('alert_count:Q', scale=alt.Scale(range=[50, 500])),
 color=alt.Color('alert_count:Q', scale=alt.Scale(scheme='reds')),
 tooltip=['latitude', 'longitude', 'alert_count']
).properties(
 width=800,
 height=600,
 title="Top 10 Jam - Heavy Traffic Alerts Between 6 AM and 9 AM"
)

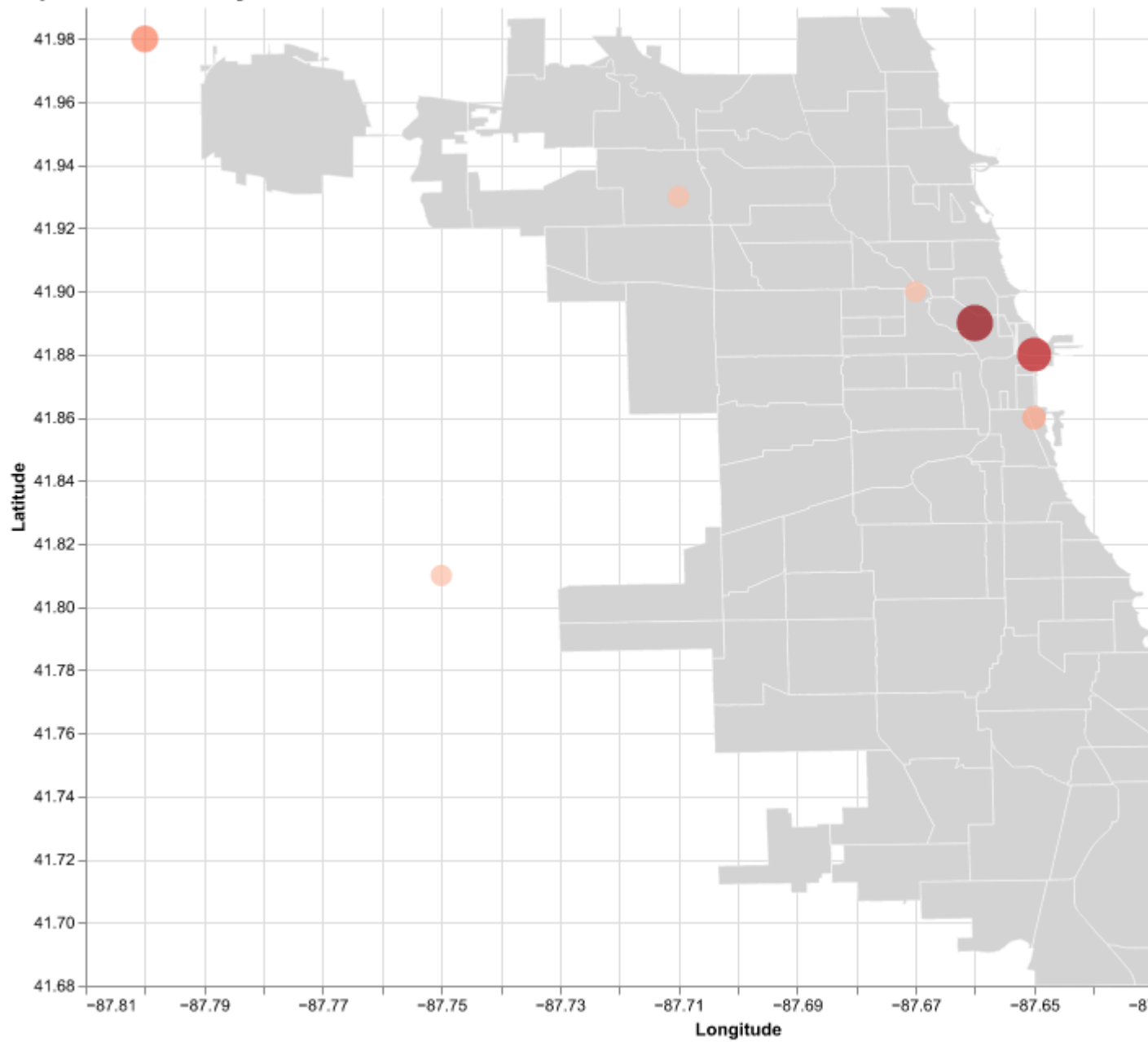
Set axis domains for alignment
lat_min, lat_max = top_10_location3['latitude'].min() - 0.01,
 ↪ top_10_location3['latitude'].max() + 0.01
lon_min, lon_max = top_10_location3['longitude'].min() - 0.01,
 ↪ top_10_location3['longitude'].max() + 0.01

Combine the layers!!
rangeplot = (map_layer + scatter_plot3).configure_view(
 stroke=None
).configure_title(
 fontSize=16,
 anchor="start"
).encode(
 x=alt.X('longitude:Q', scale=alt.Scale(domain=[lon_min, lon_max])),
 ↪ title="Longitude",
 y=alt.Y('latitude:Q', scale=alt.Scale(domain=[lat_min, lat_max])),
 ↪ title="Latitude")
).properties(
 width=800,

```

```
height=600
)
rangeplot.show()
```

### Top 10 Jam - Heavy Traffic Alerts Between 6 AM and 9 AM



2.
  - a.
  - b.
3.
  - a.
  - b.
  - c.
- d. 1, Add a “Time Period” Categorization (Morning vs Afternoon). Categorize data based on the time of day and color the points. This is by extracting the hour and classifying them into morning and afternoon: Morning 6AM - 12PM, Afternoon: 1PM - 6PM. Add a column to tell that for each case. 2, Add a Legend for Size and Time Period. 3, Use `alt.shape()` to assign different marks for the two categories.