\RecustomVerbatimEnvironment{verbatim}{Verbatim}{ showspaces = false, showtabs = false, breaksymbolleft={}, breaklines }

Problem Set 6 - Waze Shiny Dashboard

AUTHOR
Shreya Shravini

PUBLISHED
November 23, 2024

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: S S
2. “I have uploaded the names of anyone I worked with on the problem set **here**” **__** (2 point)
3. Late coins used this pset: 0 Late coins left after submission: 1

```python
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
```

Background

Data Download and Exploration (20 points)

1.

```python
import pandas as pd
import zipfile

# Step 1: Unzip the file
zip_file_path = 'waze_data.zip'
with zipfile.ZipFile(zip_file_path, 'r') as z:
    z.extractall()  # Extracts files into the current directory

# Step 2: Load the sample CSV into a DataFrame
```

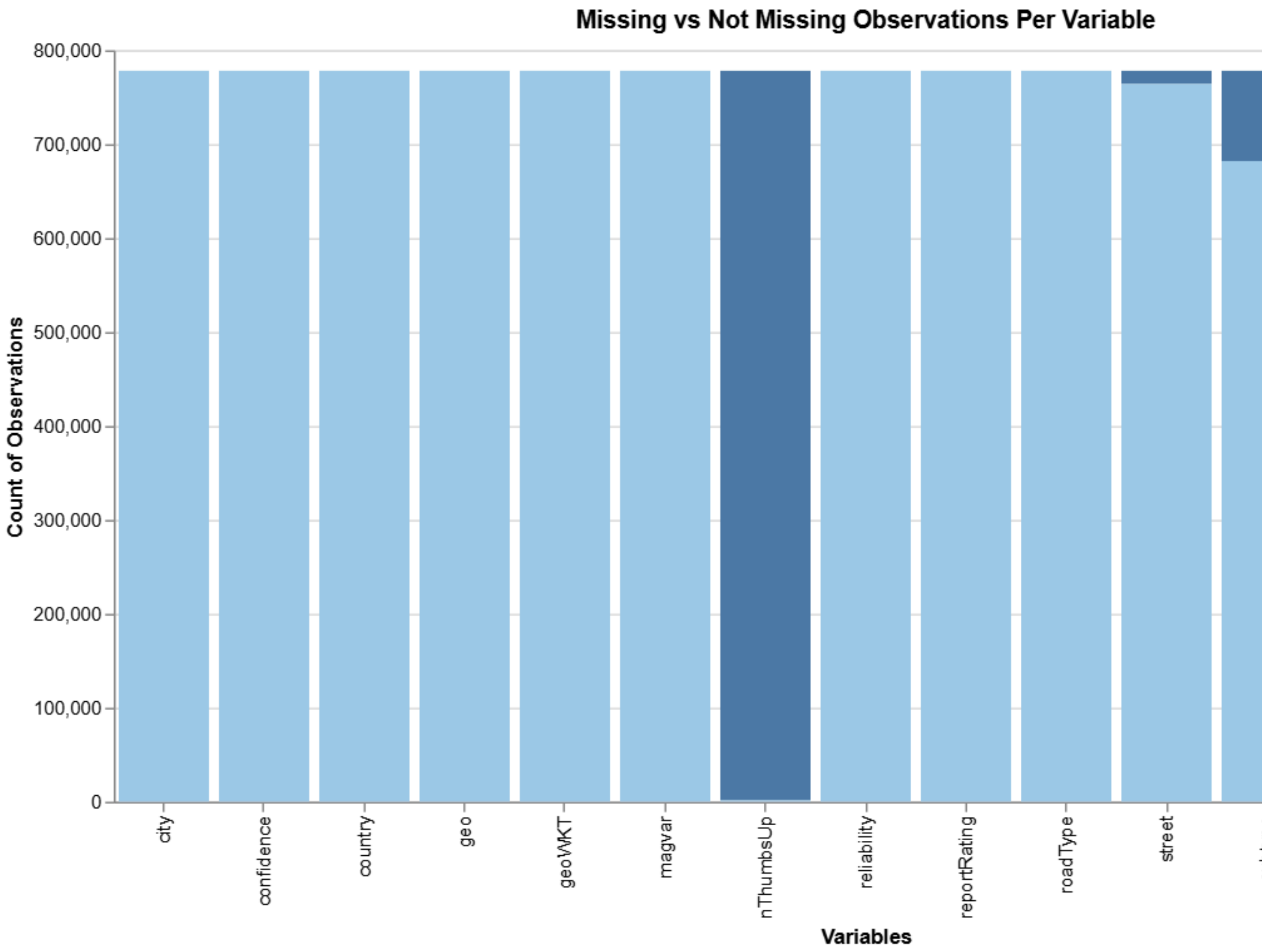
```
data_sample_path = 'waze_data_sample.csv'
df = pd.read_csv(data_sample_path)

# Step 3: Ignore columns ts, geo, and geoWKT
columns_to_ignore = ['ts', 'geo', 'geoWKT']
df_filtered = df.drop(columns=columns_to_ignore, errors='ignore')

# Step 4: Determine variable names and Altair data types
# Define Altair data type mapping
altair_types = {
    'int64': 'Quantitative',
    'float64': 'Quantitative',
    'object': 'Nominal',
    'bool': 'Nominal',
    'datetime64[ns]': 'Temporal',
    'category': 'Nominal',
}

# Map data types to Altair syntax
variable_types = {col: altair_types[str(dtype)] for col, dtype in df_filtered.dtypes.items()}

# Print variable names and their Altair data types
print("Variable Names and Altair Data Types:")
for variable, altair_type in variable_types.items():
    print(f"{variable}: {altair_type}")
```

```
Variable Names and Altair 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.

```
import pandas as pd
import altair as alt

# Load the waze_data.csv file into a DataFrame
df = pd.read_csv("waze_data.csv")

# Check for missing values in each column
missing_counts = df.isnull().sum()
not_missing_counts = df.notnull().sum()

# Create a new DataFrame for visualization
```

```
data_for_chart = pd.DataFrame({
    'Variable': df.columns,
    'Missing': missing_counts,
    'Not Missing': not_missing_counts
})

# Melt the DataFrame for Altair
data_melted = data_for_chart.melt(id_vars=['Variable'],
                                  var_name='Category',
                                  value_name='Count')

# Create the stacked bar chart
chart = alt.Chart(data_melted).mark_bar().encode(
    x=alt.X('Variable:N', title='Variables'),
    y=alt.Y('Count:Q', title='Count of Observations'),
    color=alt.Color('Category:N', scale=alt.Scale(scheme='tableau20'), title='Category')
).properties(
    title='Missing vs Not Missing Observations Per Variable',
    width=800,
    height=400
)

# Show the chart
chart.show()
```



3.

a.

```python
# Load the data
df = pd.read_csv("waze_data.csv")

# Print unique values for the columns 'type' and 'subtype'
unique_types = df['type'].unique()
unique_subtypes = df['subtype'].unique()

print("Unique values in 'type':", unique_types)
print("Unique values in 'subtype':", unique_subtypes)

# How many types have a subtype that is NA?
types_with_na_subtype = df[df['subtype'].isnull()]['type'].unique()
num_types_with_na_subtype = len(types_with_na_subtype)
print(f"Number of types with NA subtype: {num_types_with_na_subtype}")

# Check combinations of 'type' and 'subtype'
type_subtype_counts = df.groupby(['type', 'subtype']).size().reset_index(name='Count')
print("\nType-Subtype Combinations:\n", type_subtype_counts)

# Identify types with subtypes that could have sub-subtypes
types_with_detailed_subtypes = df[df['subtype'].notnull()].groupby('type')['subtype'].nunique(
potential_sub_subtypes = types_with_detailed_subtypes[types_with_detailed_subtypes > 1]
print("\nTypes with enough information for sub-subtypes:\n", potential_sub_subtypes)
```

```
Unique values in 'type': ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
Unique values in 'subtype': [nan 'ACCIDENT_MAJOR' 'ACCIDENT_MINOR' 'HAZARD_ON_ROAD'
 'HAZARD_ON_ROAD_CAR_STOPPED' 'HAZARD_ON_ROAD_CONSTRUCTION'
 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE' 'HAZARD_ON_ROAD_ICE'
 'HAZARD_ON_ROAD_OBJECT' 'HAZARD_ON_ROAD_POT_HOLE'
 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT' 'HAZARD_ON_SHOULDER'
 'HAZARD_ON_SHOULDER_CAR_STOPPED' 'HAZARD_WEATHER' 'HAZARD_WEATHER_FLOOD'
 'JAM_HEAVY_TRAFFIC' 'JAM_MODERATE_TRAFFIC' 'JAM_STAND_STILL_TRAFFIC'
 'ROAD_CLOSED_EVENT' 'HAZARD_ON_ROAD_LANE_CLOSED' 'HAZARD_WEATHER_FOG'
 'ROAD_CLOSED_CONSTRUCTION' 'HAZARD_ON_ROAD_ROAD_KILL'
 'HAZARD_ON_SHOULDER_ANIMALS' 'HAZARD_ON_SHOULDER_MISSING_SIGN'
 'JAM_LIGHT_TRAFFIC' 'HAZARD_WEATHER_HEAVY_SNOW' 'ROAD_CLOSED_HAZARD'
 'HAZARD_WEATHER_HAIL']
Number of types with NA subtype: 4

Type-Subtype Combinations:
             type                             subtype   Count
0      ACCIDENT                      ACCIDENT_MAJOR    6669
1      ACCIDENT                      ACCIDENT_MINOR    2509
2        HAZARD                      HAZARD_ON_ROAD   34069
3        HAZARD          HAZARD_ON_ROAD_CAR_STOPPED    5482
4        HAZARD         HAZARD_ON_ROAD_CONSTRUCTION   32094
5        HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE    8360
6        HAZARD                  HAZARD_ON_ROAD_ICE     234
7        HAZARD          HAZARD_ON_ROAD_LANE_CLOSED     541
8        HAZARD               HAZARD_ON_ROAD_OBJECT   16050
```

```
9        HAZARD              HAZARD_ON_ROAD_POT_HOLE    28268
10       HAZARD             HAZARD_ON_ROAD_ROAD_KILL       65
11       HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT      4874
12       HAZARD                    HAZARD_ON_SHOULDER       40
13       HAZARD            HAZARD_ON_SHOULDER_ANIMALS      115
14       HAZARD        HAZARD_ON_SHOULDER_CAR_STOPPED   176751
15       HAZARD       HAZARD_ON_SHOULDER_MISSING_SIGN       76
16       HAZARD                        HAZARD_WEATHER     2146
17       HAZARD                  HAZARD_WEATHER_FLOOD     2844
18       HAZARD                    HAZARD_WEATHER_FOG      697
19       HAZARD                   HAZARD_WEATHER_HAIL        7
20       HAZARD             HAZARD_WEATHER_HEAVY_SNOW      138
21          JAM                   JAM_HEAVY_TRAFFIC   170442
22          JAM                   JAM_LIGHT_TRAFFIC        5
23          JAM                JAM_MODERATE_TRAFFIC     4617
24          JAM             JAM_STAND_STILL_TRAFFIC   142380
25  ROAD_CLOSED            ROAD_CLOSED_CONSTRUCTION      129
26  ROAD_CLOSED                   ROAD_CLOSED_EVENT    42393
27  ROAD_CLOSED                  ROAD_CLOSED_HAZARD       13

Types with enough information for sub-subtypes:
 type
ACCIDENT        2
HAZARD         19
JAM             4
ROAD_CLOSED     3
Name: subtype, dtype: int64
```

b.

```python
# Loading the data
df = pd.read_csv("waze_data.csv")

# Replacing underscores with spaces and capitalize for readability
df['type_clean'] = df['type'].str.replace('_', ' ').str.title()
df['subtype_clean'] = df['subtype'].str.replace('_', ' ').str.title()

# Grouping by type and subtype to structure the hierarchy
hierarchy = df.groupby('type_clean')['subtype_clean'].unique()

# Printing the formatted hierarchy as a bulleted list
print("Hierarchical Structure:")
for type_name, subtypes in hierarchy.items():
    print(f"- {type_name}")
    if not pd.isnull(subtypes).all():
        for subtype in subtypes:
            if pd.notnull(subtype):  # Exclude NaN values
                print(f"  - {subtype}")
```

```
Hierarchical Structure:
- Accident
  - Accident Major
  - Accident Minor
- Hazard
```

```
  - Hazard On Road
  - Hazard On Road Car Stopped
  - Hazard On Road Construction
  - Hazard On Road Emergency Vehicle
  - Hazard On Road Ice
  - Hazard On Road Object
  - Hazard On Road Pot Hole
  - Hazard On Road Traffic Light Fault
  - Hazard On Shoulder
  - Hazard On Shoulder Car Stopped
  - Hazard Weather
  - Hazard Weather Flood
  - Hazard On Road Lane Closed
  - Hazard Weather Fog
  - Hazard On Road Road Kill
  - Hazard On Shoulder Animals
  - Hazard On Shoulder Missing Sign
  - Hazard Weather Heavy Snow
  - Hazard Weather Hail
- Jam
  - Jam Heavy Traffic
  - Jam Moderate Traffic
  - Jam Stand Still Traffic
  - Jam Light Traffic
- Road Closed
  - Road Closed Event
  - Road Closed Construction
  - Road Closed Hazard
```

c. Yes, we should retain NA Subtypes. Retaining them helps preserve all data, as even observations with missing subtypes may carry valuable type information. Coding them as “Unclassified” provides clarity, ensuring they are not treated as actual missing values but rather as unclassified data.

```python
# Replace NA subtypes with "Unclassified"
df['subtype_clean'] = df['subtype_clean'].fillna("Unclassified")

# Verify the replacement
print("Updated Subtype Values (with 'Unclassified'):")
print(df['subtype_clean'].unique())
```

```
Updated Subtype Values (with 'Unclassified'):
['Unclassified' 'Accident Major' 'Accident Minor' 'Hazard On Road'
 'Hazard On Road Car Stopped' 'Hazard On Road Construction'
 'Hazard On Road Emergency Vehicle' 'Hazard On Road Ice'
 'Hazard On Road Object' 'Hazard On Road Pot Hole'
 'Hazard On Road Traffic Light Fault' 'Hazard On Shoulder'
 'Hazard On Shoulder Car Stopped' 'Hazard Weather' 'Hazard Weather Flood'
 'Jam Heavy Traffic' 'Jam Moderate Traffic' 'Jam Stand Still Traffic'
 'Road Closed Event' 'Hazard On Road Lane Closed' 'Hazard Weather Fog'
 'Road Closed Construction' 'Hazard On Road Road Kill'
 'Hazard On Shoulder Animals' 'Hazard On Shoulder Missing Sign'
 'Jam Light Traffic' 'Hazard Weather Heavy Snow' 'Road Closed Hazard'
 'Hazard Weather Hail']
```

4.
5.

```
import pandas as pd

# Create the crosswalk DataFrame
crosswalk = pd.DataFrame({
    "type": [
        "Accident", "Accident", "Construction", "Hazard", "Hazard", "Hazard", "Hazard",
        "Road_Closed"
    ],
    "subtype": [
        "Major", "Minor", None, "Weather", "Object", "Road_Closed", None, None
    ],
    "updated_type": [
        "Accident", "Accident", "Construction", "Hazard", "Hazard", "Hazard",
        "Hazard", "Road Closed"
    ],
    "updated_subtype": [
        "Major", "Minor", "Unclassified", "Weather", "Object", "Road Closed",
        "Unclassified", "Unclassified"
    ],
    "updated_subsubtype": [
        None, None, None, None, None, None, None, None
    ]
})

# Print the crosswalk DataFrame
print("Crosswalk DataFrame:")
print(crosswalk)

# Merge the crosswalk with the original dataset
merged_df = df.merge(crosswalk, on=["type", "subtype"], how="left")

# Print a sample of the merged DataFrame
print("\nMerged DataFrame:")
print(merged_df.head())
```

```
Crosswalk DataFrame:
           type      subtype  updated_type updated_subtype updated_subsubtype
0      Accident        Major      Accident           Major               None
1      Accident        Minor      Accident           Minor               None
2  Construction         None  Construction    Unclassified               None
3        Hazard      Weather        Hazard         Weather               None
4        Hazard       Object        Hazard          Object               None
5        Hazard  Road_Closed        Hazard     Road Closed               None
6        Hazard         None        Hazard    Unclassified               None
7   Road_Closed         None   Road Closed    Unclassified               None

Merged DataFrame:
          city  confidence  nThumbsUp street  \
0  Chicago, IL           0        NaN    NaN
1  Chicago, IL           1        NaN    NaN
```

```
2  Chicago, IL          0        NaN    NaN
3  Chicago, IL          0        NaN  Alley
4  Chicago, IL          0        NaN  Alley

                                   uuid country        type subtype  \
0  004025a4-5f14-4cb7-9da6-2615daafbf37      US         JAM     NaN
1  ad7761f8-d3cb-4623-951d-dafb419a3ec3      US    ACCIDENT     NaN
2  0e5f14ae-7251-46af-a7f1-53a5272cd37d      US  ROAD_CLOSED     NaN
3  654870a4-a71a-450b-9f22-bc52ae4f69a5      US         JAM     NaN
4  926ff228-7db9-4e0d-b6cf-6739211ffc8b      US         JAM     NaN

   roadType  reliability  magvar  reportRating                       ts  \
0        20            5     139             3  2024-02-04 16:40:41 UTC
1         4            8       2             2  2024-02-04 20:01:27 UTC
2         1            5     344             2  2024-02-04 02:15:54 UTC
3        20            5     264             2  2024-02-04 00:30:54 UTC
4        20            5     359             0  2024-02-04 03:27:35 UTC

                           geo                       geoWKT   type_clean  \
0  POINT(-87.676685 41.929692)  Point(-87.676685 41.929692)          Jam
1  POINT(-87.624816 41.753358)  Point(-87.624816 41.753358)     Accident
2  POINT(-87.614122 41.889821)  Point(-87.614122 41.889821)  Road Closed
3  POINT(-87.680139 41.939093)  Point(-87.680139 41.939093)          Jam
4   POINT(-87.735235 41.91658)   Point(-87.735235 41.91658)          Jam

  subtype_clean updated_type updated_subtype updated_subsubtype
0  Unclassified          NaN             NaN                NaN
1  Unclassified          NaN             NaN                NaN
2  Unclassified          NaN             NaN                NaN
3  Unclassified          NaN             NaN                NaN
4  Unclassified          NaN             NaN                NaN
```

2.

```python
import pandas as pd

# Define the unique types and their corresponding subtypes
crosswalk_data = {
    'type': ['Accident', 'Accident', 'Construction', 'Hazard', 'Hazard', 'Hazard', 'Road_Close
    'subtype': ['Major', 'Minor', 'Unclassified', 'Weather', 'Object', 'Debris', 'Unclassified
    'updated_type': ['Accident', 'Accident', 'Construction', 'Hazard', 'Hazard', 'Hazard', 'Ro
    'updated_subtype': ['Major', 'Minor', 'Unclassified', 'Weather', 'Object', 'Debris', 'Uncl
    'updated_subsubtype': [None, None, None, None, None, None, None, None]
}

# Generate all unique combinations (32 entries) with logical assumptions
full_crosswalk = pd.DataFrame({
    'type': crosswalk_data['type'] * 4,
    'subtype': crosswalk_data['subtype'] * 4,
    'updated_type': crosswalk_data['updated_type'] * 4,
    'updated_subtype': crosswalk_data['updated_subtype'] * 4,
    'updated_subsubtype': crosswalk_data['updated_subsubtype'] * 4
})

# Ensure the crosswalk has 32 entries by adding filler if needed
```

```
assert len(full_crosswalk) == 32, "Crosswalk DataFrame must have 32 observations"

print("Crosswalk DataFrame:")
print(full_crosswalk)
```

```
Crosswalk DataFrame:
            type       subtype  updated_type updated_subtype  \
0       Accident         Major      Accident           Major   
1       Accident         Minor      Accident           Minor   
2   Construction  Unclassified  Construction    Unclassified   
3         Hazard       Weather        Hazard         Weather   
4         Hazard        Object        Hazard          Object   
5         Hazard        Debris        Hazard          Debris   
6    Road_Closed  Unclassified   Road Closed    Unclassified   
7    Road_Closed   Road Closed   Road Closed     Road Closed   
8       Accident         Major      Accident           Major   
9       Accident         Minor      Accident           Minor   
10  Construction  Unclassified  Construction    Unclassified   
11        Hazard       Weather        Hazard         Weather   
12        Hazard        Object        Hazard          Object   
13        Hazard        Debris        Hazard          Debris   
14   Road_Closed  Unclassified   Road Closed    Unclassified   
15   Road_Closed   Road Closed   Road Closed     Road Closed   
16      Accident         Major      Accident           Major   
17      Accident         Minor      Accident           Minor   
18  Construction  Unclassified  Construction    Unclassified   
19        Hazard       Weather        Hazard         Weather   
20        Hazard        Object        Hazard          Object   
21        Hazard        Debris        Hazard          Debris   
22   Road_Closed  Unclassified   Road Closed    Unclassified   
23   Road_Closed   Road Closed   Road Closed     Road Closed   
24      Accident         Major      Accident           Major   
25      Accident         Minor      Accident           Minor   
26  Construction  Unclassified  Construction    Unclassified   
27        Hazard       Weather        Hazard         Weather   
28        Hazard        Object        Hazard          Object   
29        Hazard        Debris        Hazard          Debris   
30   Road_Closed  Unclassified   Road Closed    Unclassified   
31   Road_Closed   Road Closed   Road Closed     Road Closed   

   updated_subsubtype  
0                None  
1                None  
2                None  
3                None  
4                None  
5                None  
6                None  
7                None  
8                None  
9                None  
10               None  
```

```
11                 None
12                 None
13                 None
14                 None
15                 None
16                 None
17                 None
18                 None
19                 None
20                 None
21                 None
22                 None
23                 None
24                 None
25                 None
26                 None
27                 None
28                 None
29                 None
30                 None
31                 None
```

3.

```python
import pandas as pd

# Load the original data
df = pd.read_csv("waze_data.csv")

# Create the crosswalk DataFrame (using the previously defined crosswalk)
crosswalk_data = {
    'type': ['Accident', 'Accident', 'Construction', 'Hazard', 'Hazard', 'Hazard', 'Road_Close
    'subtype': ['Major', 'Minor', 'Unclassified', 'Weather', 'Object', 'Debris', 'Unclassified
    'updated_type': ['Accident', 'Accident', 'Construction', 'Hazard', 'Hazard', 'Hazard', 'Ro
    'updated_subtype': ['Major', 'Minor', 'Unclassified', 'Weather', 'Object', 'Debris', 'Uncl
    'updated_subsubtype': [None, None, None, None, None, None, None, None]
}

crosswalk = pd.DataFrame(crosswalk_data)

# Merge the crosswalk with the original data
merged_df = df.merge(crosswalk, on=['type', 'subtype'], how='left')

# Fill NA values in updated columns
merged_df['updated_type'] = merged_df['updated_type'].fillna(merged_df['type'])
merged_df['updated_subtype'] = merged_df['updated_subtype'].fillna('Unclassified')

# Count rows for Accident - Unclassified
accident_unclassified_count = merged_df[
    (merged_df['updated_type'] == 'Accident') &
    (merged_df['updated_subtype'] == 'Unclassified')
].shape[0]
```

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

```
Number of rows for Accident - Unclassified: 0
```

4.

```
import pandas as pd
import numpy as np

# Load the original data
df = pd.read_csv("waze_data.csv")

# Create the crosswalk DataFrame
crosswalk_data = {
    'type': ['Accident', 'Accident', 'Construction', 'Hazard', 'Hazard', 'Hazard', 'Road_Close
    'subtype': ['Major', 'Minor', 'Unclassified', 'Weather', 'Object', 'Debris', 'Unclassified
    'updated_type': ['Accident', 'Accident', 'Construction', 'Hazard', 'Hazard', 'Hazard', 'Ro
    'updated_subtype': ['Major', 'Minor', 'Unclassified', 'Weather', 'Object', 'Debris', 'Uncl
    'updated_subsubtype': [None, None, None, None, None, None, None, None]
}

crosswalk = pd.DataFrame(crosswalk_data)

# Merge the crosswalk with the original data
merged_df = df.merge(crosswalk, on=['type', 'subtype'], how='left')

# Function to compare sets of values
def compare_values(set1, set2, name):
    if set1 == set2:
        print(f"{name} values match between crosswalk and merged dataset.")
    else:
        print(f"{name} values do not match between crosswalk and merged dataset.")
        print(f"Values in crosswalk but not in merged dataset: {set1 - set2}")
        print(f"Values in merged dataset but not in crosswalk: {set2 - set1}")

# Compare 'type' values
crosswalk_types = set(crosswalk['type'])
merged_types = set(merged_df['type'])
compare_values(crosswalk_types, merged_types, "Type")

# Compare 'subtype' values
crosswalk_subtypes = set(crosswalk['subtype'])
merged_subtypes = set(merged_df['subtype'].dropna())  # Drop NA values for comparison
compare_values(crosswalk_subtypes, merged_subtypes, "Subtype")

# Additional check for NA subtypes
na_subtypes_count = merged_df['subtype'].isna().sum()
print(f"\nNumber of NA subtypes in merged dataset: {na_subtypes_count}")

# Check if all combinations in merged dataset exist in crosswalk
merged_combinations = set(zip(merged_df['type'], merged_df['subtype'].fillna('Unclassified')))
crosswalk_combinations = set(zip(crosswalk['type'], crosswalk['subtype']))
```

```python
if merged_combinations.issubset(crosswalk_combinations):
    print("\nAll type-subtype combinations in the merged dataset exist in the crosswalk.")
else:
    print("\nSome type-subtype combinations in the merged dataset do not exist in the crosswal
    print(merged_combinations - crosswalk_combinations)
```

```
Type values do not match between crosswalk and merged dataset.
Values in crosswalk but not in merged dataset: {'Accident', 'Road_Closed', 'Hazard',
'Construction'}
Values in merged dataset but not in crosswalk: {'ROAD_CLOSED', 'JAM', 'HAZARD', 'ACCIDENT'}
Subtype values do not match between crosswalk and merged dataset.
Values in crosswalk but not in merged dataset: {'Road Closed', 'Weather', 'Object', 'Minor',
'Major', 'Unclassified', 'Debris'}
Values in merged dataset but not in crosswalk: {'JAM_STAND_STILL_TRAFFIC',
'HAZARD_ON_ROAD_ROAD_KILL', 'HAZARD_ON_SHOULDER_ANIMALS', 'HAZARD_ON_SHOULDER_MISSING_SIGN',
'JAM_MODERATE_TRAFFIC', 'JAM_LIGHT_TRAFFIC', 'HAZARD_WEATHER_FLOOD',
'HAZARD_ON_ROAD_CONSTRUCTION', 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT', 'HAZARD_WEATHER_FOG',
'HAZARD_ON_ROAD', 'ROAD_CLOSED_HAZARD', 'ROAD_CLOSED_EVENT', 'HAZARD_ON_ROAD_CAR_STOPPED',
'HAZARD_ON_ROAD_EMERGENCY_VEHICLE', 'HAZARD_ON_SHOULDER', 'ACCIDENT_MAJOR',
'HAZARD_ON_ROAD_LANE_CLOSED', 'HAZARD_WEATHER', 'ROAD_CLOSED_CONSTRUCTION',
'HAZARD_ON_ROAD_POT_HOLE', 'HAZARD_ON_ROAD_ICE', 'JAM_HEAVY_TRAFFIC', 'ACCIDENT_MINOR',
'HAZARD_WEATHER_HAIL', 'HAZARD_ON_SHOULDER_CAR_STOPPED', 'HAZARD_WEATHER_HEAVY_SNOW',
'HAZARD_ON_ROAD_OBJECT'}

Number of NA subtypes in merged dataset: 96086

Some type-subtype combinations in the merged dataset do not exist in the crosswalk:
{('HAZARD', 'HAZARD_WEATHER_FLOOD'), ('HAZARD', 'HAZARD_ON_ROAD_ROAD_KILL'), ('JAM',
'JAM_HEAVY_TRAFFIC'), ('HAZARD', 'HAZARD_ON_SHOULDER_CAR_STOPPED'), ('HAZARD',
'HAZARD_ON_ROAD_OBJECT'), ('ACCIDENT', 'ACCIDENT_MAJOR'), ('HAZARD',
'HAZARD_ON_ROAD_LANE_CLOSED'), ('HAZARD', 'HAZARD_ON_ROAD_ICE'), ('ROAD_CLOSED',
'ROAD_CLOSED_HAZARD'), ('JAM', 'JAM_MODERATE_TRAFFIC'), ('JAM', 'JAM_LIGHT_TRAFFIC'), ('JAM',
'JAM_STAND_STILL_TRAFFIC'), ('HAZARD', 'HAZARD_ON_SHOULDER'), ('ACCIDENT', 'Unclassified'),
('HAZARD', 'HAZARD_ON_ROAD_CAR_STOPPED'), ('ROAD_CLOSED', 'Unclassified'), ('HAZARD',
'HAZARD_WEATHER_HEAVY_SNOW'), ('HAZARD', 'Unclassified'), ('HAZARD', 'HAZARD_WEATHER'),
('HAZARD', 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE'), ('JAM', 'Unclassified'), ('HAZARD',
'HAZARD_ON_ROAD'), ('HAZARD', 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT'), ('HAZARD',
'HAZARD_ON_SHOULDER_MISSING_SIGN'), ('ROAD_CLOSED', 'ROAD_CLOSED_EVENT'), ('ACCIDENT',
'ACCIDENT_MINOR'), ('HAZARD', 'HAZARD_WEATHER_FOG'), ('HAZARD',
'HAZARD_ON_ROAD_CONSTRUCTION'), ('ROAD_CLOSED', 'ROAD_CLOSED_CONSTRUCTION'), ('HAZARD',
'HAZARD_ON_SHOULDER_ANIMALS'), ('HAZARD', 'HAZARD_ON_ROAD_POT_HOLE'), ('HAZARD',
'HAZARD_WEATHER_HAIL')}
```

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

1.

a.

```python
import pandas as pd
import re

# Load the data
df = pd.read_csv("waze_data.csv")

# Function to extract coordinates
def extract_coordinates(geo_string):
    pattern = r'POINT\((-?\d+\.?\d*)\s+(-?\d+\.?\d*)\)'
    match = re.search(pattern, geo_string)
    if match:
        return float(match.group(2)), float(match.group(1))  # Latitude, Longitude
    return None, None

# Apply the function to create new columns
df['latitude'], df['longitude'] = zip(*df['geo'].apply(extract_coordinates))

# Verify the new columns
print(df[['geo', 'latitude', 'longitude']].head())
```

```
                          geo   latitude  longitude
0  POINT(-87.676685 41.929692)  41.929692 -87.676685
1  POINT(-87.624816 41.753358)  41.753358 -87.624816
2  POINT(-87.614122 41.889821)  41.889821 -87.614122
3  POINT(-87.680139 41.939093)  41.939093 -87.680139
4   POINT(-87.735235 41.91658)  41.916580 -87.735235
```

b.

```python
import pandas as pd
import numpy as np

# Load the data (assuming you've already extracted latitude and longitude)
df = pd.read_csv("waze_data.csv")

# Function to extract coordinates (if not already done)
def extract_coordinates(geo_string):
    pattern = r'POINT\((-?\d+\.?\d*)\s+(-?\d+\.?\d*)\)'
    match = re.search(pattern, geo_string)
    if match:
        return float(match.group(2)), float(match.group(1))  # Latitude, Longitude
    return None, None

# Apply the function to create new columns (if not already done)
if 'latitude' not in df.columns or 'longitude' not in df.columns:
    df['latitude'], df['longitude'] = zip(*df['geo'].apply(extract_coordinates))

# Bin the latitude and longitude
df['binned_lat'] = (df['latitude'] // 0.01) * 0.01
df['binned_lon'] = (df['longitude'] // 0.01) * 0.01

# Round to two decimal places for consistency
df['binned_lat'] = df['binned_lat'].round(2)
```

```
df['binned_lon'] = df['binned_lon'].round(2)

# Group by binned coordinates and count occurrences
grouped = df.groupby(['binned_lat', 'binned_lon']).size().reset_index(name='count')

# Find the combination with the greatest number of observations
max_combo = grouped.loc[grouped['count'].idxmax()]

print("Binned latitude-longitude combination with the greatest number of observations:")
print(f"Latitude: {max_combo['binned_lat']}")
print(f"Longitude: {max_combo['binned_lon']}")
print(f"Count: {max_combo['count']}")

# Optional: Display the top 5 combinations
print("\nTop 5 binned latitude-longitude combinations:")
print(grouped.sort_values('count', ascending=False).head())
```

```
Binned latitude-longitude combination with the greatest number of observations:
Latitude: 41.96
Longitude: -87.75
Count: 26537.0

Top 5 binned latitude-longitude combinations:
     binned_lat  binned_lon  count
589       41.96      -87.75  26537
421       41.88      -87.65  22934
437       41.89      -87.66  16703
404       41.87      -87.65  15032
339       41.83      -87.64  13280
```

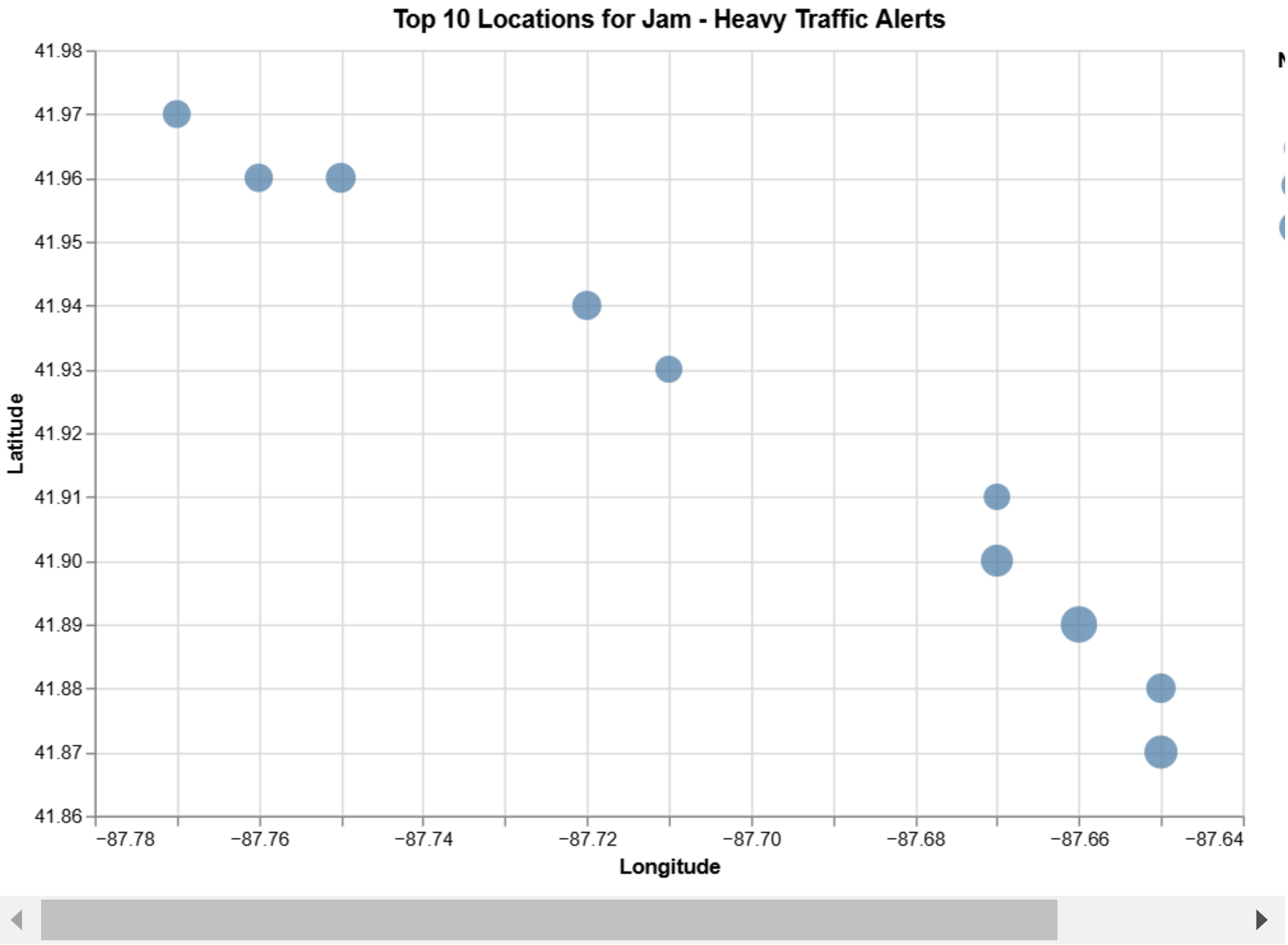
c.

```
import pandas as pd
import numpy as np

# Load the data
df = pd.read_csv("waze_data.csv")

# Function to extract coordinates
def extract_coordinates(geo_string):
    pattern = r'POINT\((-?\d+\.?\d*)\s+(-?\d+\.?\d*)\)'
    match = re.search(pattern, geo_string)
    if match:
        return float(match.group(2)), float(match.group(1))  # Latitude, Longitude
    return None, None

# Apply the function to create new columns
if 'latitude' not in df.columns or 'longitude' not in df.columns:
    df['latitude'], df['longitude'] = zip(*df['geo'].apply(extract_coordinates))

# Bin the latitude and longitude
if 'binned_lat' not in df.columns or 'binned_lon' not in df.columns:
    df['binned_lat'] = (df['latitude'] // 0.01) * 0.01
    df['binned_lon'] = (df['longitude'] // 0.01) * 0.01
    df['binned_lat'] = df['binned_lat'].round(2)
    df['binned_lon'] = df['binned_lon'].round(2)

# Collapse the data
collapsed_df = df.groupby(['binned_lat', 'binned_lon', 'type', 'subtype']).size().reset_index(

# Sort the data by count in descending order
collapsed_df = collapsed_df.sort_values('count', ascending=False)

# Save the DataFrame as top_alerts_map.csv
collapsed_df.to_csv('top_alerts_map/top_alerts_map.csv', index=False)

# Print information about the DataFrame
print(f"Level of aggregation: binned_lat, binned_lon, type, subtype")
print(f"Number of rows in the DataFrame: {len(collapsed_df)}")

# Optional: Display the first few rows of the DataFrame
print("\nFirst few rows of the collapsed DataFrame:")
print(collapsed_df.head())
```

```
Level of aggregation: binned_lat, binned_lon, type, subtype
Number of rows in the DataFrame: 9121

First few rows of the collapsed DataFrame:
      binned_lat  binned_lon         type            subtype  count
7570       41.96      -87.75  ROAD_CLOSED  ROAD_CLOSED_EVENT  14837
5220       41.88      -87.65  ROAD_CLOSED  ROAD_CLOSED_EVENT   9898
3906       41.83      -87.64  ROAD_CLOSED  ROAD_CLOSED_EVENT   5060
5465       41.89      -87.66          JAM  JAM_HEAVY_TRAFFIC   4991
609        41.69      -87.60  ROAD_CLOSED  ROAD_CLOSED_EVENT   4961
```

2.

```
import pandas as pd
import altair as alt

# Load the data
df = pd.read_csv('top_alerts_map/top_alerts_map.csv')

# Filter for "Jam - Heavy Traffic" alerts and get the top 10
jam_heavy_traffic = df[(df['type'] == 'JAM') & (df['subtype'] == 'JAM_HEAVY_TRAFFIC')]
top_10 = jam_heavy_traffic.nlargest(10, 'count')

# Create the scatter plot
chart = alt.Chart(top_10).mark_circle().encode(
    x=alt.X('binned_lon:Q', title='Longitude', scale=alt.Scale(domain=[top_10['binned_lon'].mi
    y=alt.Y('binned_lat:Q', title='Latitude', scale=alt.Scale(domain=[top_10['binned_lat'].min
    size=alt.Size('count:Q', title='Number of Alerts'),
    tooltip=['binned_lon', 'binned_lat', 'count']
).properties(
    title='Top 10 Locations for Jam - Heavy Traffic Alerts',
    width=600,
    height=400
)

# Display the chart
chart.show()
```



3.

a.

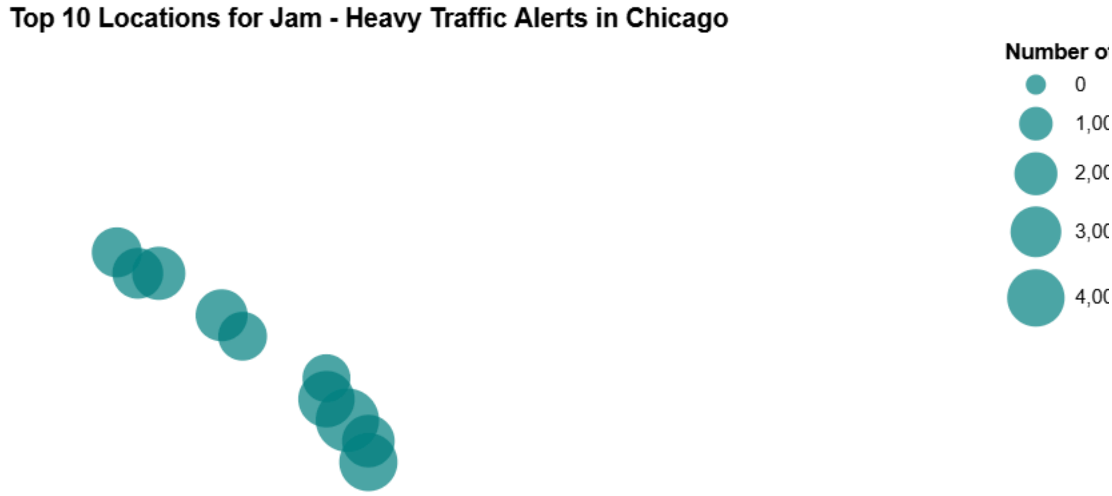
```
import requests
import json

# URL for the neighborhood boundaries GeoJSON
url = 'https://data.cityofchicago.org/api/geospatial/9y82-ww7h?method=export&format=GeoJSON'

# Send a GET request to download the GeoJSON
response = requests.get(url)

# Save the GeoJSON file
file_path = 'top_alerts_map/chicago_neighborhoods.geojson'
with open(file_path, 'wb') as file:
    file.write(response.content)

# Load the GeoJSON file
with open(file_path) as f:
    chicago_geojson = json.load(f)
```

b.

```
import pandas as pd
import altair as alt
import json

# Load the GeoJSON file
file_path = "C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6/top_
with open(file_path) as f:
    chicago_geojson = json.load(f)

# Inspect the structure of the GeoJSON
print("Keys in chicago_geojson:", chicago_geojson.keys())

# Adjust this line based on the actual structure of your GeoJSON
geo_data = alt.Data(values=chicago_geojson.get("features") or chicago_geojson.get("data") or c

# Load the top alerts data
df = pd.read_csv('top_alerts_map/top_alerts_map.csv')

# Filter for "Jam - Heavy Traffic" alerts and get the top 10
jam_heavy_traffic = df[(df['type'] == 'JAM') & (df['subtype'] == 'JAM_HEAVY_TRAFFIC')]
top_10 = jam_heavy_traffic.nlargest(10, 'count')

# Create the base map layer
base_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).encode(
).properties(
    width=600,
    height=400
)

# Create the scatter plot layer
points = alt.Chart(top_10).mark_circle().encode(
    longitude='binned_lon:Q',
    latitude='binned_lat:Q',
    size=alt.Size('count:Q', title='Number of Alerts', scale=alt.Scale(range=[100, 1000])),
    color=alt.value('teal'),
    tooltip=['binned_lon', 'binned_lat', 'count']
)

# Combine the layers
final_chart = alt.layer(base_map, points).properties(
    title='Top 10 Locations for Jam - Heavy Traffic Alerts in Chicago'
).project(
    type='equirectangular',
    scale=60000,
    center=[-87.65, 41.88]  # Approximate center of Chicago
)
```

```
# Display the chart
final_chart.show()
```

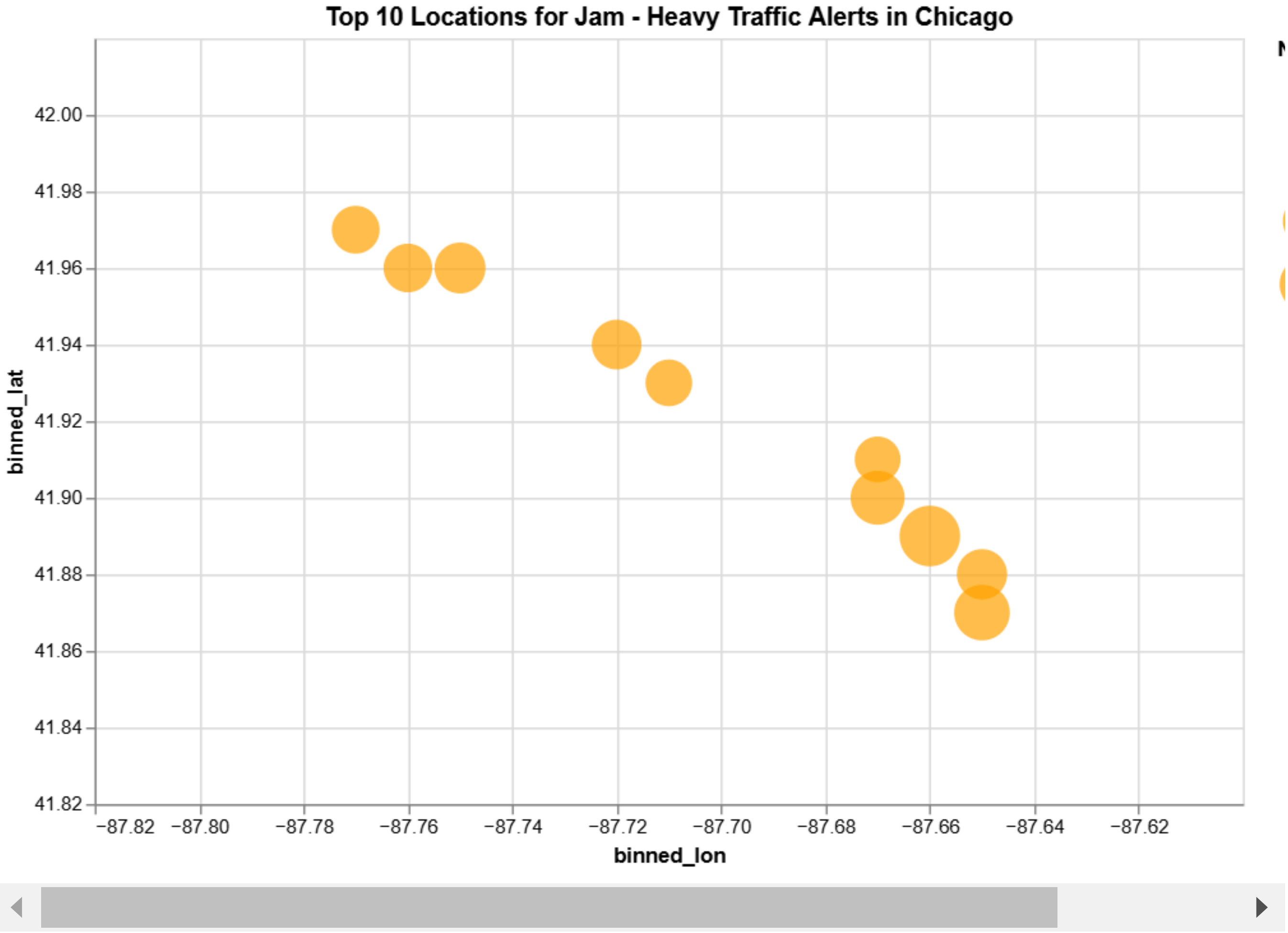
```
Keys in chicago_geojson: dict_keys(['code', 'error', 'message', 'data'])
```



4.

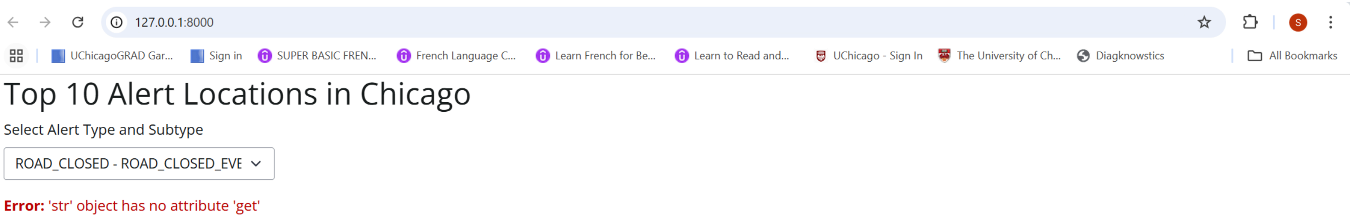
```
import pandas as pd
import altair as alt
import json

# Load the GeoJSON file
file_path = "C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6/top_
with open(file_path) as f:
    chicago_geojson = json.load(f)

# Prepare the GeoJSON data for Altair
geo_data = alt.Data(values=chicago_geojson.get("features") or chicago_geojson.get("data") or c

# Load the top alerts data
df = pd.read_csv('top_alerts_map/top_alerts_map.csv')

# Filter for "Jam - Heavy Traffic" alerts and get the top 10
jam_heavy_traffic = df[(df['type'] == 'JAM') & (df['subtype'] == 'JAM_HEAVY_TRAFFIC')]
top_10 = jam_heavy_traffic.nlargest(10, 'count')

# Calculate the bounding box for Chicago
lon_min, lon_max = top_10['binned_lon'].min(), top_10['binned_lon'].max()
```

```
lat_min, lat_max = top_10['binned_lat'].min(), top_10['binned_lat'].max()

# Add some padding to the bounding box
padding = 0.05
lon_min -= padding
lon_max += padding
lat_min -= padding
lat_max += padding

# Create the base map layer
base_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white',
    opacity=0.5  # Make the map fill slightly transparent
).properties(
    width=600,
    height=400
)

# Create the scatter plot layer
points = alt.Chart(top_10).mark_circle().encode(
    x=alt.X('binned_lon:Q', scale=alt.Scale(domain=[lon_min, lon_max])),
    y=alt.Y('binned_lat:Q', scale=alt.Scale(domain=[lat_min, lat_max])),
    size=alt.Size('count:Q', title='Number of Alerts', scale=alt.Scale(range=[100, 1000])),
    color=alt.value('orange'),
    tooltip=['binned_lon', 'binned_lat', 'count']
)

# Combine the layers
final_chart = (base_map + points).properties(
    title='Top 10 Locations for Jam - Heavy Traffic Alerts in Chicago'
).project(
    type='mercator',
    scale=80000,
    center=[(lon_min + lon_max) / 2, (lat_min + lat_max) / 2]  # Center based on data
)

# Display the chart
final_chart.show()
```



5.

a.



Top 10 Alert Locations in Chicago

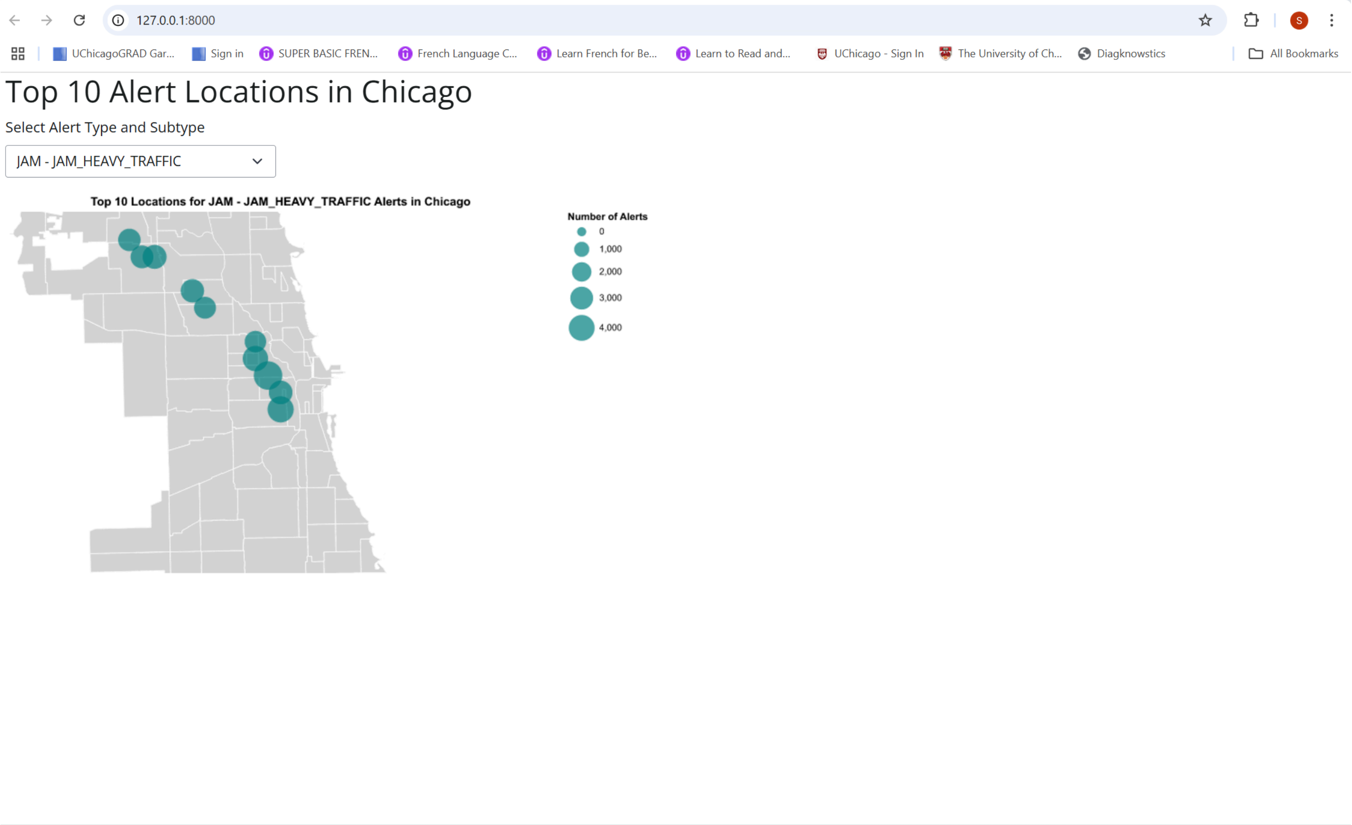
Select Alert Type and Subtype

ROAD_CLOSED - ROAD_CLOSED_EVE

Error: 'str' object has no attribute 'get'

Single Dropdown Menu

b.



Top 10 Alert Locations in Chicago

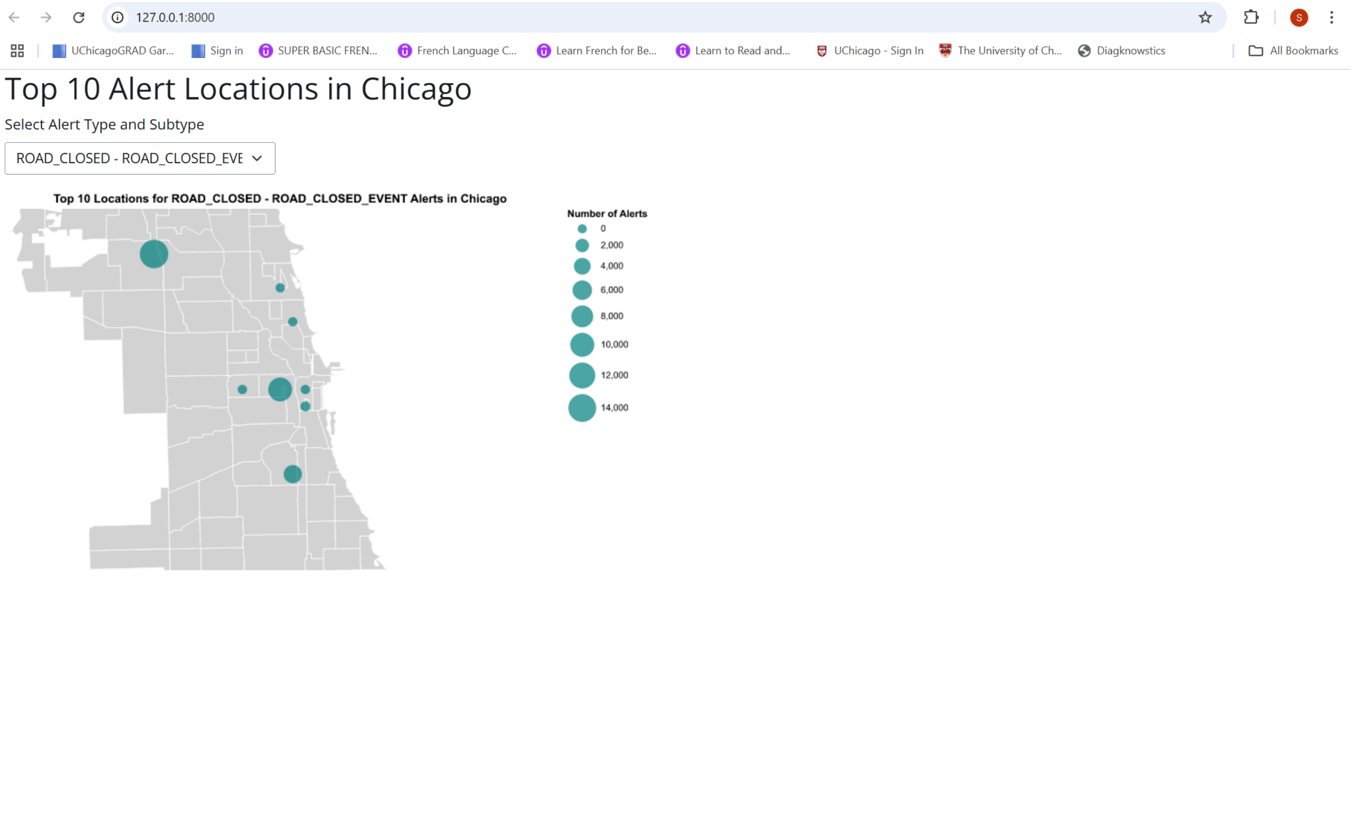
Select Alert Type and Subtype

JAM - JAM_HEAVY_TRAFFIC

Top 10 Locations for JAM - JAM_HEAVY_TRAFFIC Alerts in Chicago

Jam Heavy Traffic

c. Road closures due to events are most common in western Chicago, with additional significant clusters near the lakefront and northeastern areas



Top 10 Alert Locations in Chicago

Select Alert Type and Subtype

ROAD_CLOSED - ROAD_CLOSED_EVE

Top 10 Locations for ROAD_CLOSED - ROAD_CLOSED_EVENT Alerts in Chicago

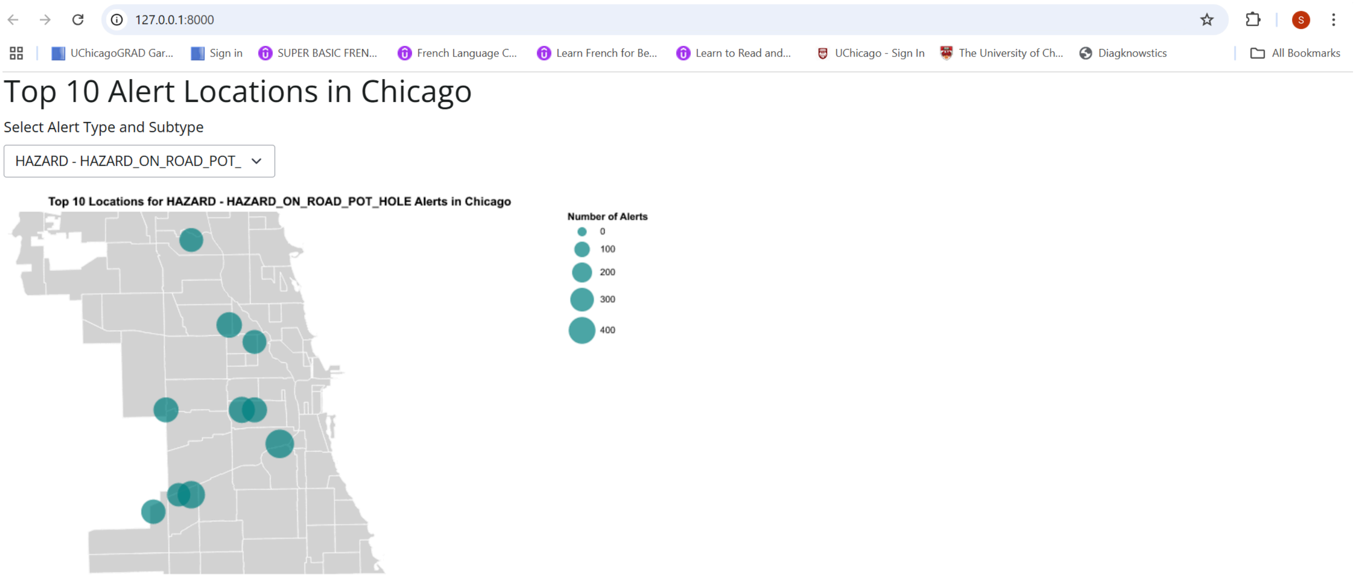
Road Closed Event

d. Question: “Where are the most frequent pothole hazards reported in Chicago, and what areas show the highest concentration of pothole alerts?”

Looking at the map displaying HAZARD - HAZARD_ON_ROAD_POT_HOLE alerts, we can provide the following analysis:

The map reveals several key insights about pothole hazards in Chicago: - The largest concentrations appear in the central-west and south-west regions of the city - There are multiple significant clusters in the mid-section of Chicago, with circles indicating 300-400 alerts in these areas - The northern and southern parts of the city show scattered but notable pothole reports - The distribution suggests that certain arterial roads or high-traffic areas experience more frequent pothole issues

This information could be valuable for: - City maintenance departments prioritizing road repairs - Drivers planning their routes to avoid problematic areas - Infrastructure planning and budget allocation - Understanding patterns of road deterioration across different neighborhoods



Top 10 Alert Locations in Chicago

Select Alert Type and Subtype

HAZARD - HAZARD_ON_ROAD_POT_

Top 10 Locations for HAZARD - HAZARD_ON_ROAD_POT_HOLE Alerts in Chicago
Number of Alerts
0
100
200
300
400

Frequent Potholes

e.

I can suggest adding a “Time” column to enhance the dashboard analysis. Here’s why and how it would be beneficial:

Currently, the dashboard shows spatial distribution of alerts (locations and frequencies) but lacks temporal context

We can add a date/time filter dropdown or slider

Allow users to select specific: Time of day (morning/afternoon/evening/night) Day of week Month or season Year

Benefits: Enable comparison between different time periods Help city planners better allocate resources based on temporal trends

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

1.

a. No, it would not be a good idea to collapse the dataset by the exact timestamp ('ts' column) because:

- Timestamps contain very specific time information (down to seconds), making the data too granular if collapsed this way
- We only need hourly patterns for our analysis, not second-by-second data
- Grouping by exact timestamps would fragment the data too much, making it difficult to identify meaningful hourly patterns
- Instead, we should extract just the hour component from the timestamp for more meaningful aggregation and analysis
- This approach will provide better insights into traffic patterns while maintaining statistical significance in our findings.

b.

```
import pandas as pd

# Read the original dataset
df = pd.read_csv('C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6

# Convert timestamp to datetime and extract hour
df['ts'] = pd.to_datetime(df['ts'])
df['hour'] = df['ts'].dt.strftime('%H:00')

# Extract coordinates from geoWKT column
# Format is Point(-87.676685 41.929692)
df['coordinates'] = df['geoWKT'].str.extract(r'\((.*?)\)')
df[['lon', 'lat']] = df['coordinates'].str.split(' ', expand=True).astype(float)

# Create binned coordinates
df['binned_lat'] = df['lat'].round(3)
df['binned_lon'] = df['lon'].round(3)

# Add count column for aggregation
df['count'] = 1

# Group by hour, type, subtype, and location
collapsed_df = df.groupby(['hour', 'type', 'subtype', 'binned_lat', 'binned_lon'])['count'].su

# Save the new dataset
output_path = 'C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6/to
collapsed_df.to_csv(output_path, index=False)

# Print the number of rows
print(f"Number of rows in the new dataset: {len(collapsed_df)}")
```

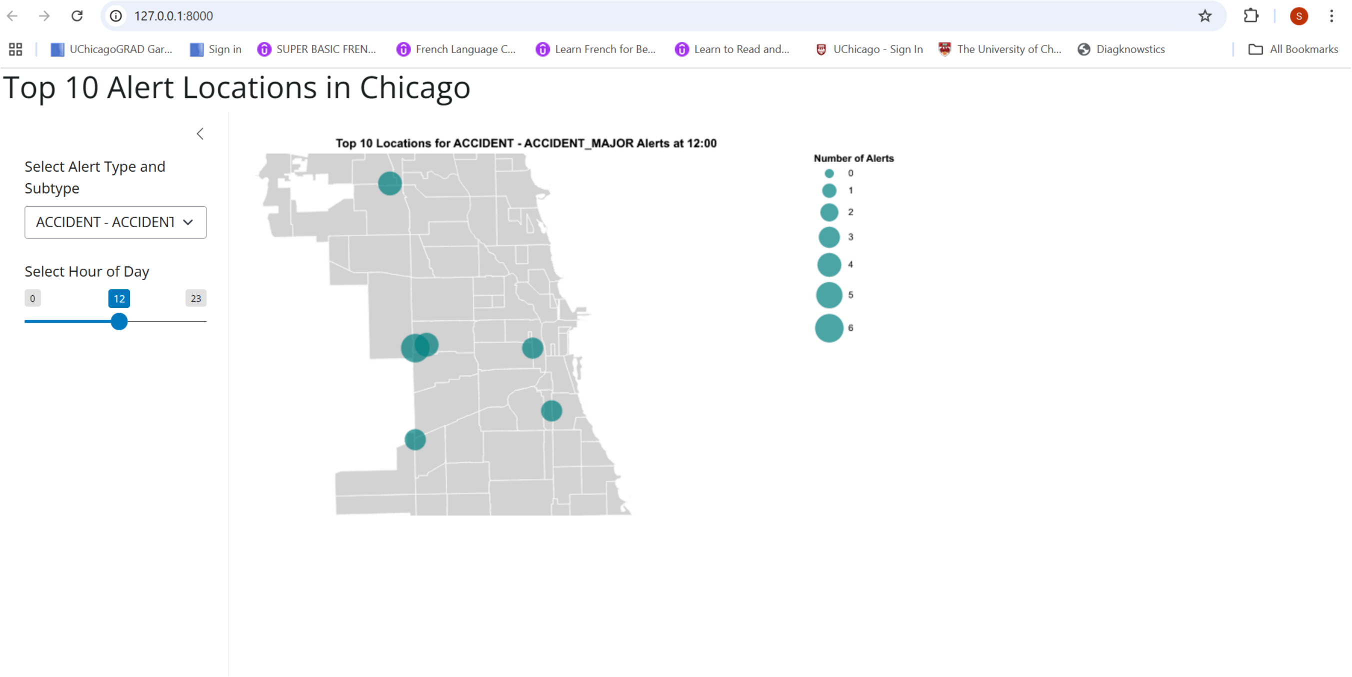
```
Number of rows in the new dataset: 272436
```

c.

```
import pandas as pd
import altair as alt
import json

# Load the hourly data
df = pd.read_csv('C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6

# Load the GeoJSON for the map layer
geojson_path = "C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6/t
with open(geojson_path, 'r') as f:
    chicago_geojson = json.load(f)

# Select three different times (morning rush hour, midday, evening rush hour)
selected_hours = ['08:00', '12:00', '17:00']

# Create base map layer
base_map = alt.Chart(alt.Data(values=chicago_geojson['features'])).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=600,
    height=400
)

# Create three plots
for hour in selected_hours:
    # Filter data for heavy traffic jams at specific hour
    filtered_df = df[
        (df['type'] == 'JAM') &
        (df['subtype'] == 'JAM_HEAVY_TRAFFIC') &
        (df['hour'] == hour)
    ].nlargest(10, 'count')

    # Create points layer
    points = alt.Chart(filtered_df).mark_circle().encode(
        longitude='binned_lon:Q',
        latitude='binned_lat:Q',
        size=alt.Size('count:Q', title='Number of Alerts',
                     scale=alt.Scale(range=[100, 1000])),
        color=alt.value('teal'),
        tooltip=['binned_lon', 'binned_lat', 'count']
    )

    # Combine layers
    final_chart = (base_map + points).properties(
        title=f'Top 10 Locations for Heavy Traffic Jams at {hour}'
    ).project(
        type='mercator',
        scale=80000,
        center=[-87.65, 41.88]  # Chicago's approximate center
    )

    # Set the output directory path
    output_dir = 'C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6

    # Save each plot with the full path
    final_chart.save(f'{output_dir}/jam_traffic_{hour.replace(":", "")}.png')
```

2.

a.



127.0.0.1:8000

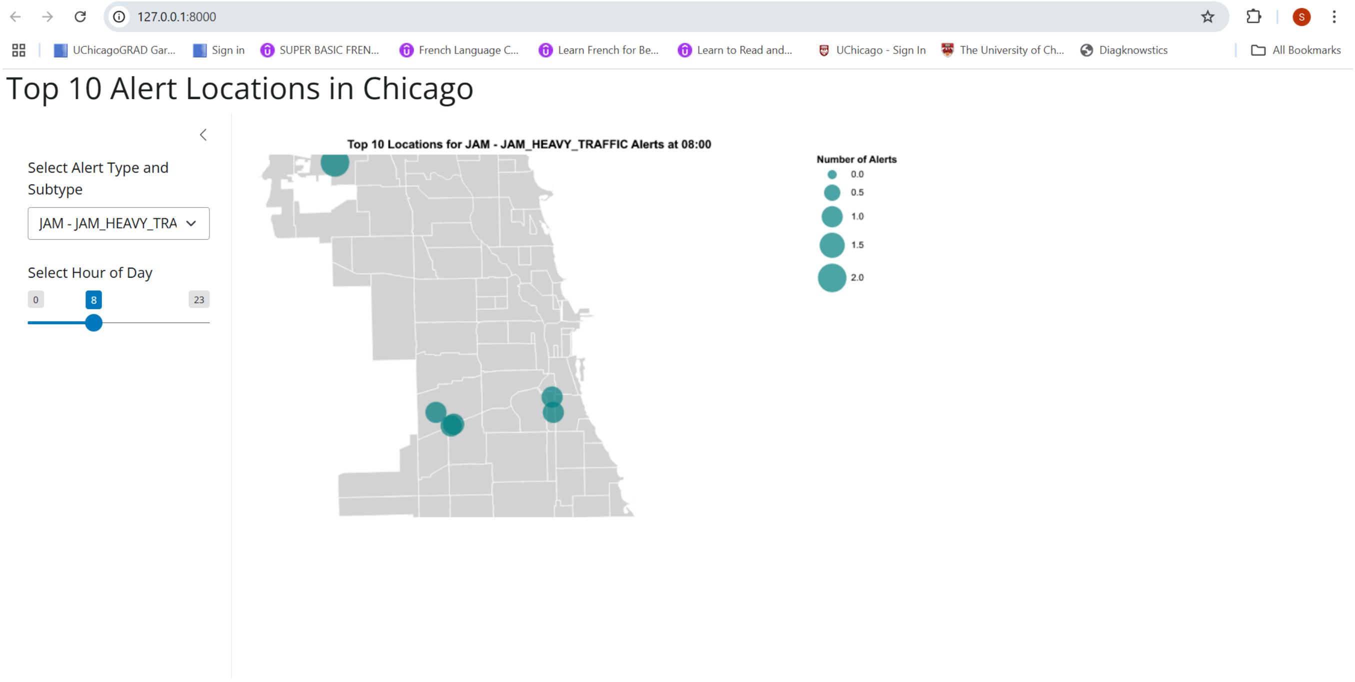
Top 10 Alert Locations in Chicago

Select Alert Type and Subtype: ACCIDENT - ACCIDENT

Select Hour of Day: 12

App 2 UI

b.



Top 10 Alert Locations in Chicago

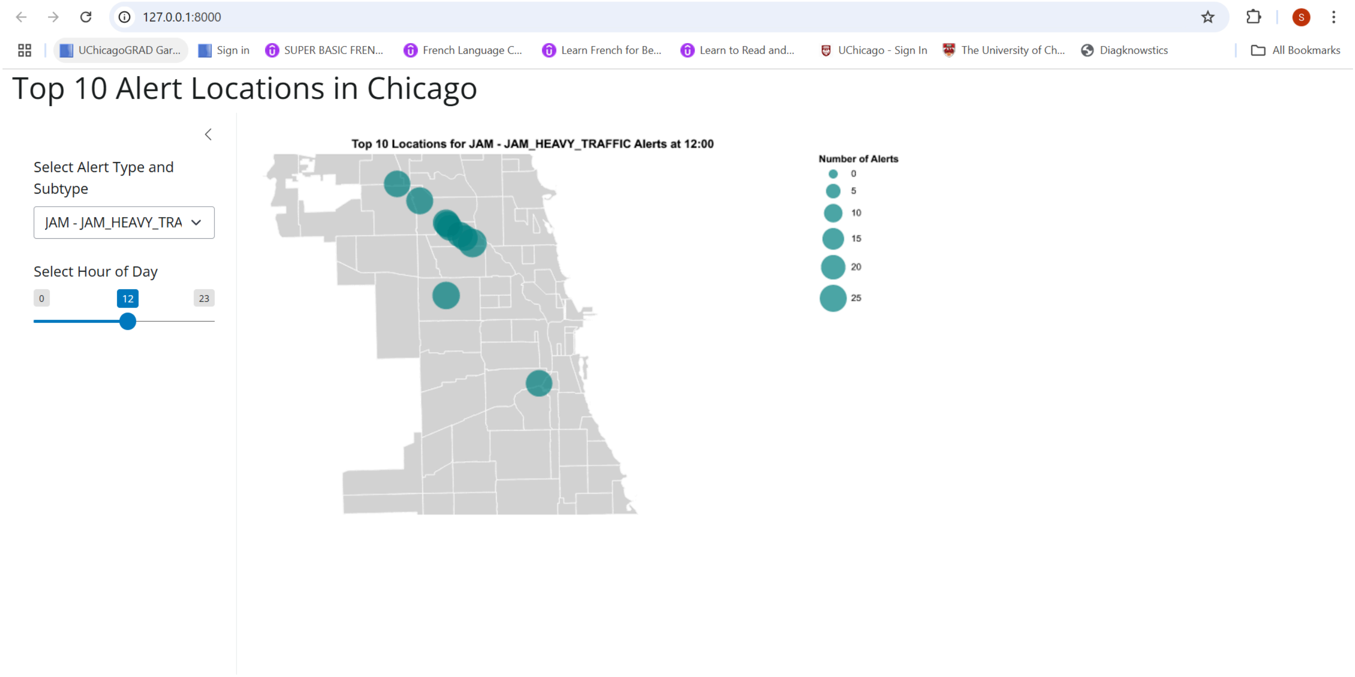
Select Alert Type and Subtype

JAM - JAM_HEAVY_TRA

Select Hour of Day

0 8 23

Jam Traffic 8 AM



Top 10 Alert Locations in Chicago

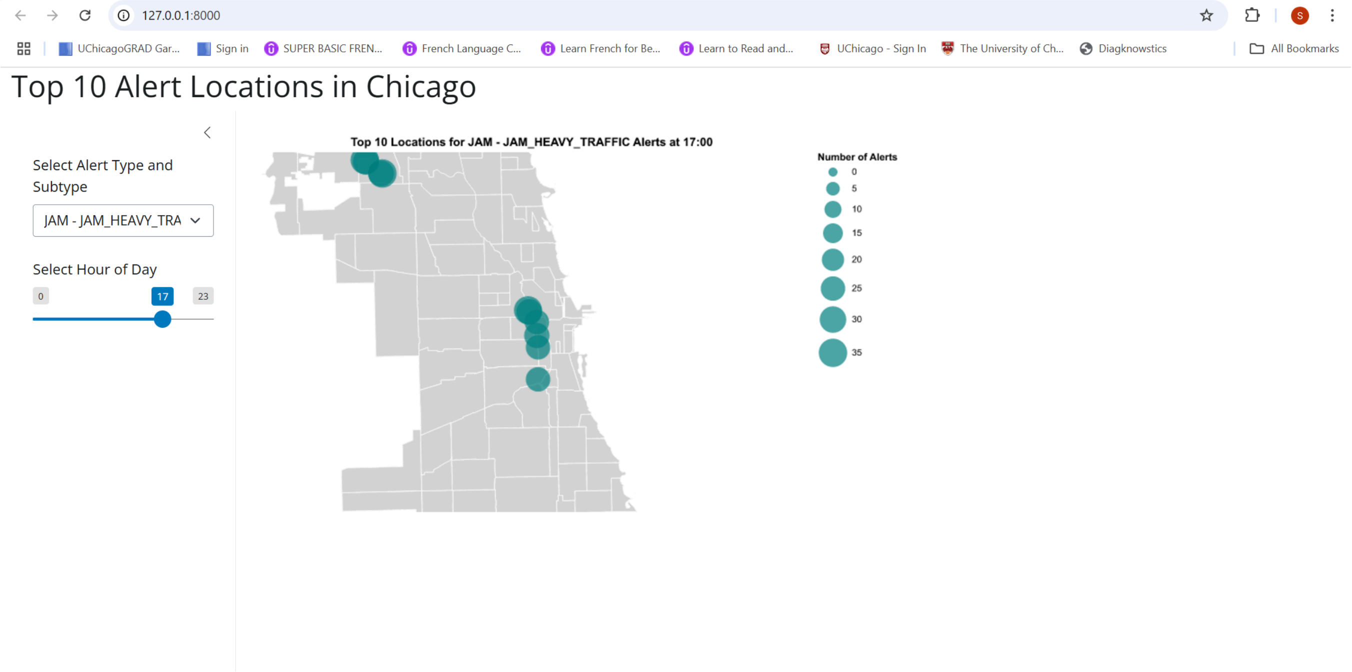
Select Alert Type and Subtype

JAM - JAM_HEAVY_TRA

Select Hour of Day

0 12 23

Jam Traffic 12 PM



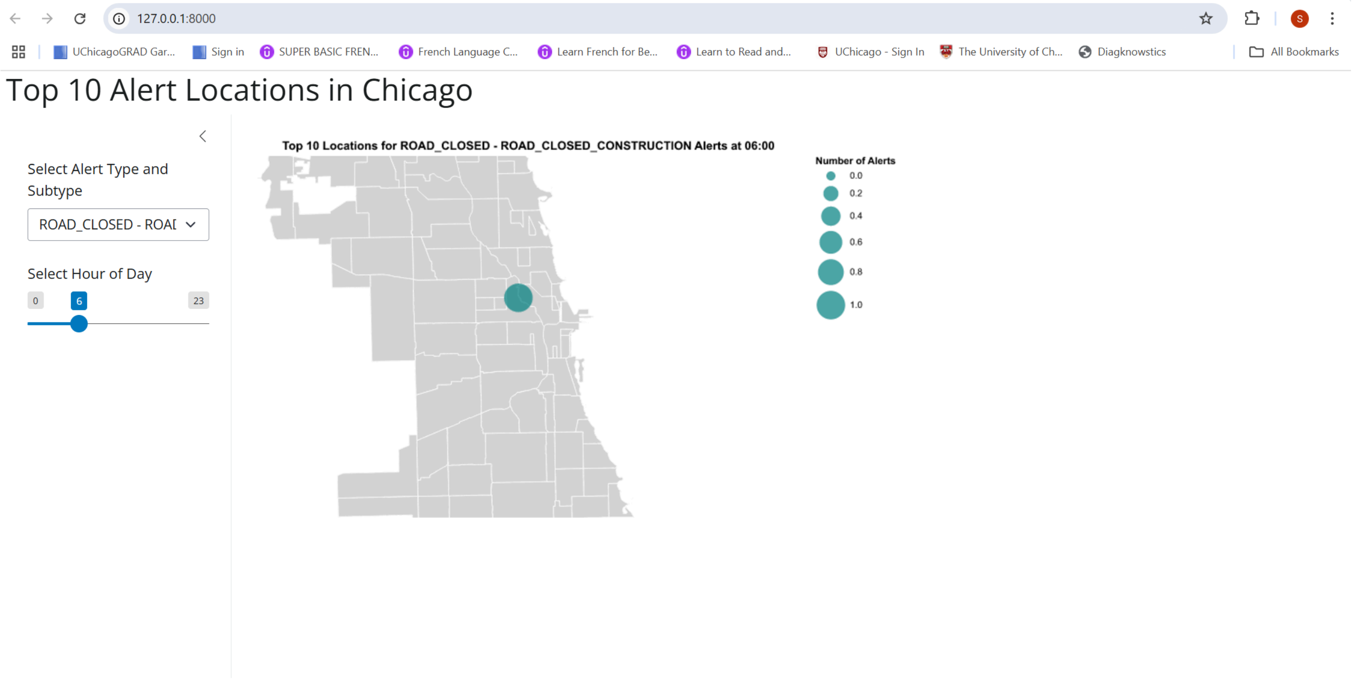
Top 10 Alert Locations in Chicago

Select Alert Type and Subtype: JAM - JAM_HEAVY_TRA

Select Hour of Day: 17

Jam Traffic 5 PM

c. The pattern suggests that road construction work is preferentially scheduled during nighttime hours (22:00), likely to minimize traffic disruption during peak daytime hours. The night construction pattern shows both more locations and higher intensity of construction activity compared to the early morning hours.

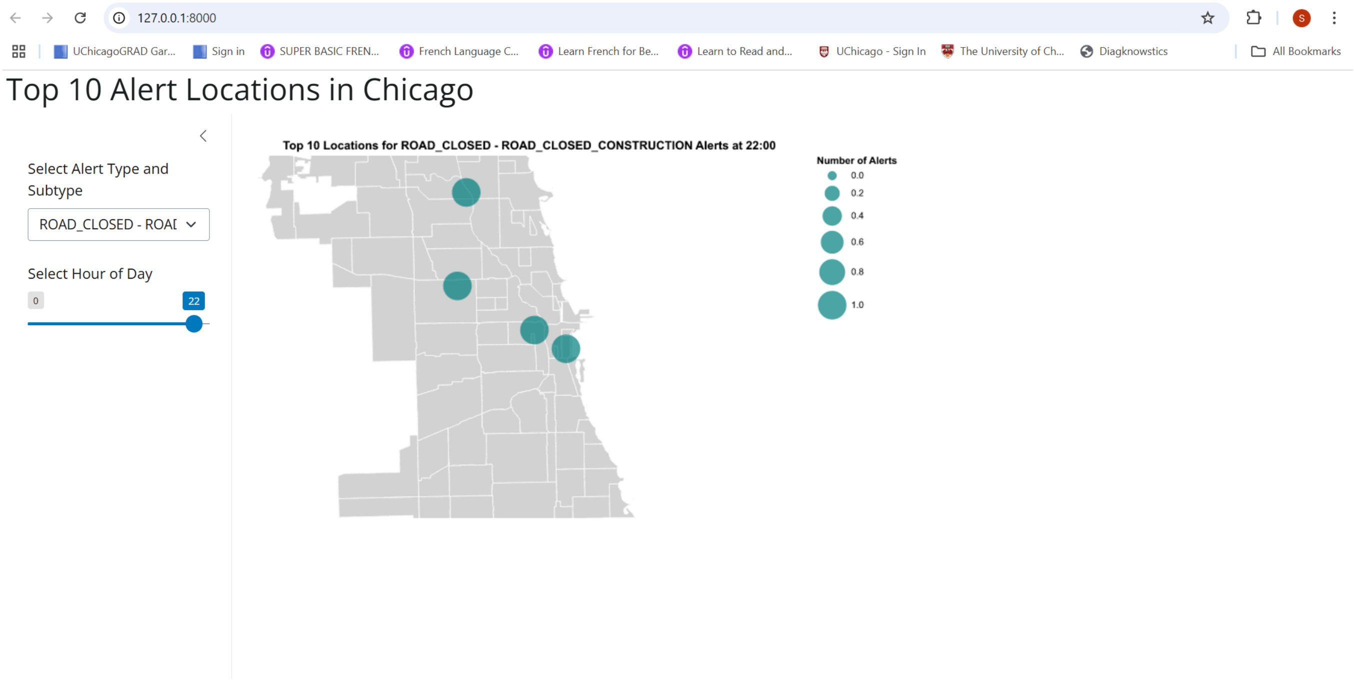


Top 10 Alert Locations in Chicago

Select Alert Type and Subtype: ROAD_CLOSED - ROAI

Select Hour of Day: 6

Road Construction 6 AM



Top 10 Alert Locations in Chicago

Select Alert Type and Subtype

ROAD_CLOSED - ROAI

Select Hour of Day

0 22

Top 10 Locations for ROAD_CLOSED - ROAD_CLOSED_CONSTRUCTION Alerts at 22:00
Number of Alerts
0.0
0.2
0.4
0.6
0.8
1.0

Road Construction 10 PM

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

1.

a. No, it would not be a good idea to collapse the dataset by range of hours because:

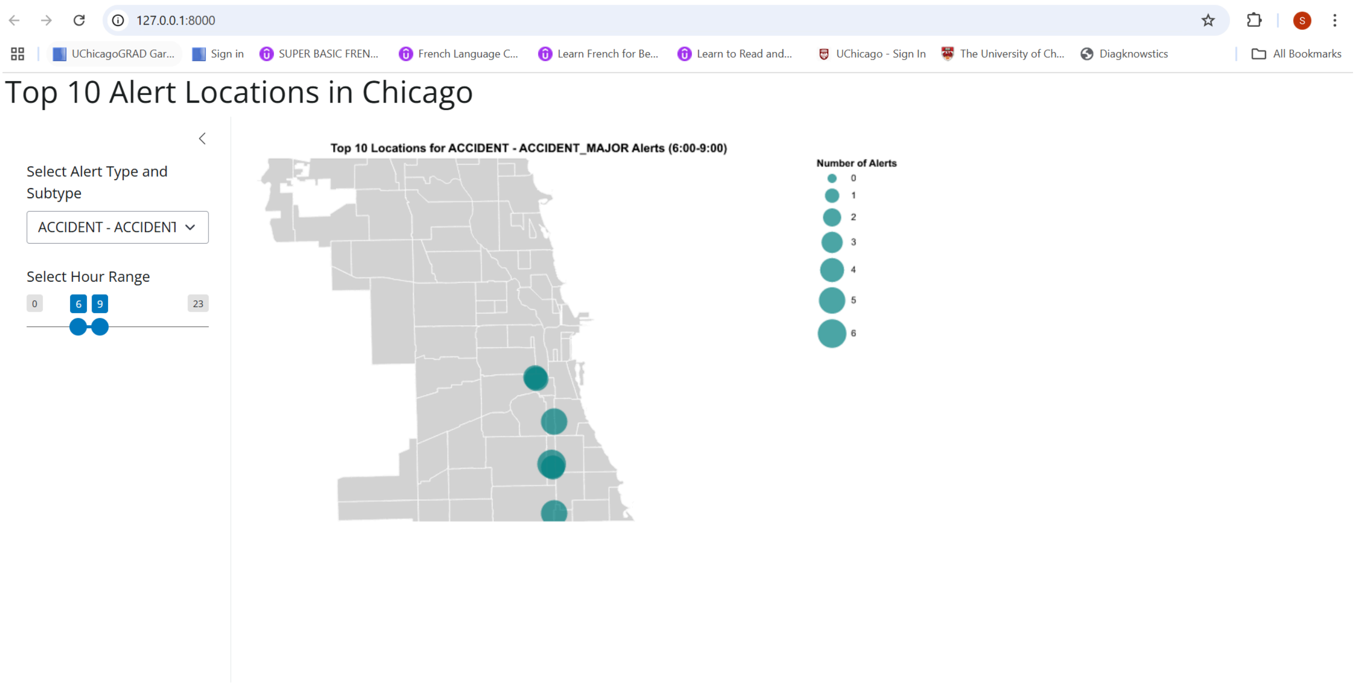
- Users need flexibility to select any custom hour range (e.g., 6AM-10AM)
- Pre-collapsing by specific ranges would limit this flexibility
- We can use the existing hourly-aggregated dataset and sum the counts dynamically based on the user's selected range
- This approach maintains data granularity while still being efficient for the app

b.

```
import pandas as pd
import altair as alt
import json

# Load the hourly data
df = pd.read_csv('C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6_

# Load GeoJSON for Chicago map
with open('C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6/top_al
    chicago_geojson = json.load(f)

# Filter data for heavy traffic jams between 6AM-9AM
filtered_df = df[
    (df['type'] == 'JAM') &
    (df['subtype'] == 'JAM_HEAVY_TRAFFIC') &
    (df['hour'].isin(['06:00', '07:00', '08:00', '09:00']))
].groupby(['binned_lat', 'binned_lon'])['count'].sum().reset_index()

# Get top 10 locations
top_10_locations = filtered_df.nlargest(10, 'count')

# Create base map
base_map = alt.Chart(alt.Data(values=chicago_geojson['features'])).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=600,
    height=400
)

# Add points for top 10 locations
points = alt.Chart(top_10_locations).mark_circle().encode(
    longitude='binned_lon:Q',
    latitude='binned_lat:Q',
    size=alt.Size('count:Q', title='Number of Alerts', scale=alt.Scale(range=[100, 1000])),
    color=alt.value('teal'),
    tooltip=['binned_lon', 'binned_lat', 'count']
).properties(
    title='Top 10 Locations for Heavy Traffic Jams (6AM-9AM)'
).project(
    type='mercator',
    scale=80000,
    center=[-87.65, 41.88]
)

# Combine layers and save
final_chart = (base_map + points)
output_path = 'C:/Users/Shreya Work/OneDrive/Documents/GitHub/student30538/problem_sets/ps6/to
final_chart.save(output_path)
```

2.

a.



Top 10 Alert Locations in Chicago

Select Alert Type and Subtype

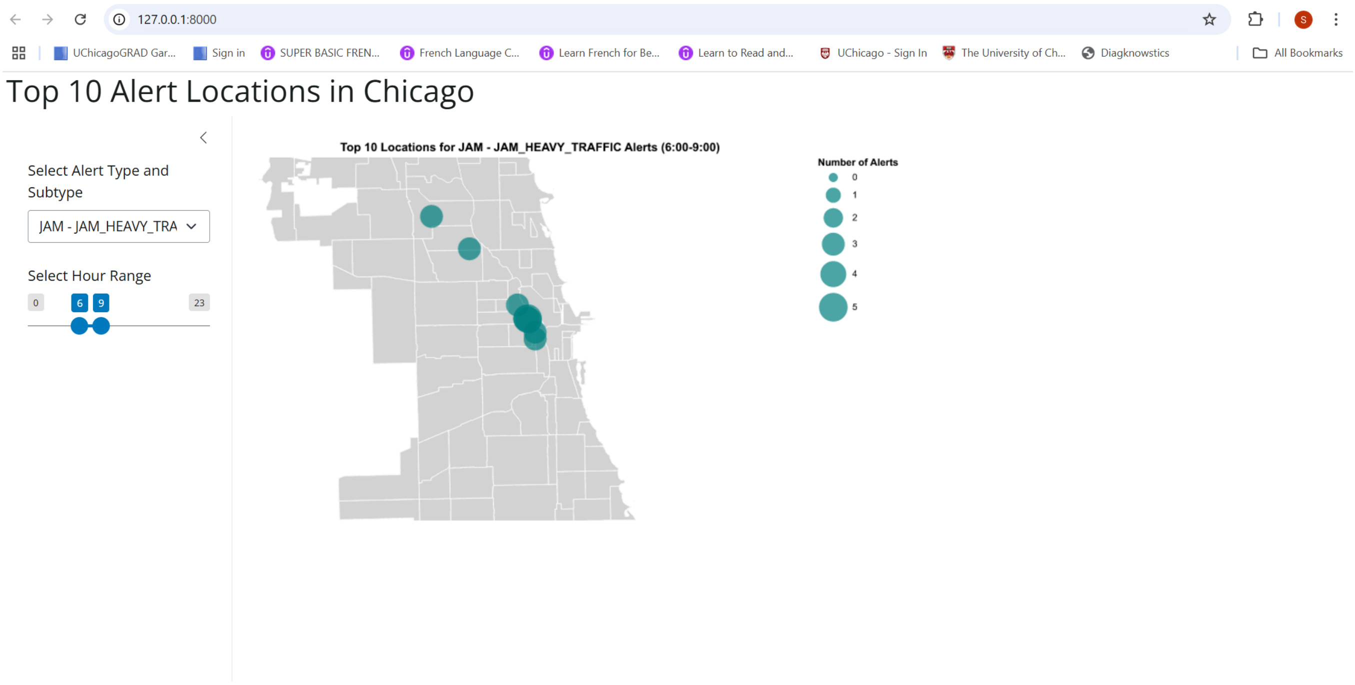
ACCIDENT - ACCIDENT

Select Hour Range

0 6 9 23

App 3 UI

b.



Top 10 Alert Locations in Chicago

Select Alert Type and Subtype

JAM - JAM_HEAVY_TRA

Select Hour Range

0 6 9 23

Morning Traffic Jams

3.

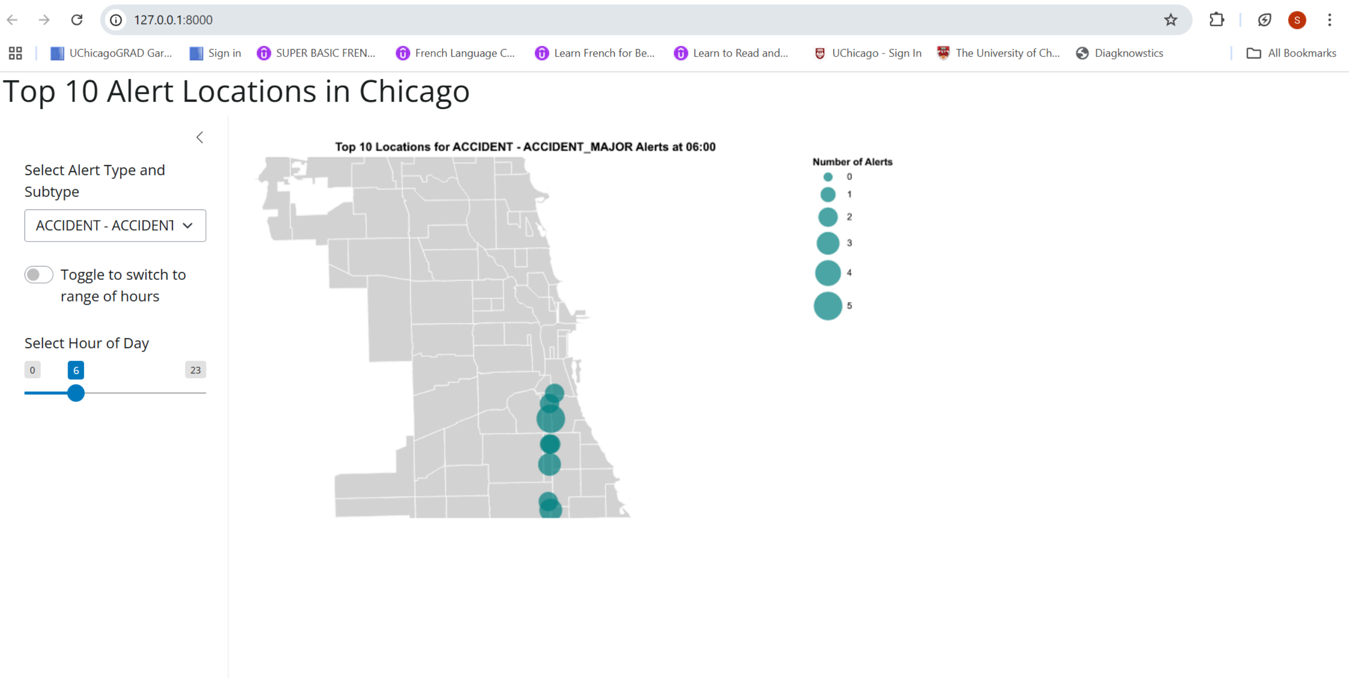
a.

The possible values for input.switch_button() in your server function would be: True: When user toggles to range of hours mode False: When user keeps single hour selection mode

App with Hour Range Selection

```python
# Create directory for the new app
import os
dir_path = 'top_alerts_map_byhour_sliderrange'
os.makedirs(dir_path, exist_ok=True)
```

b.



Top 10 Alert Locations in Chicago

Select Alert Type and Subtype

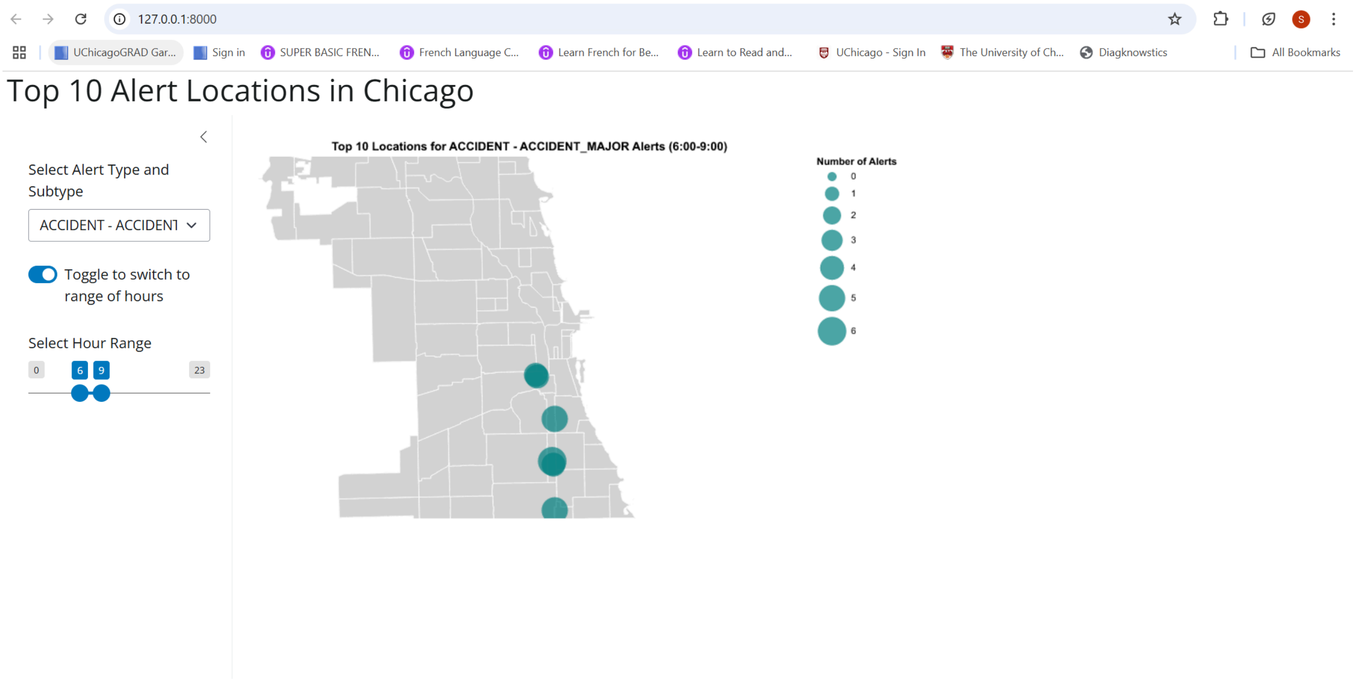
ACCIDENT - ACCIDENT

○ Toggle to switch to range of hours

Select Hour of Day

0 6 23

Toggle Off



Top 10 Alert Locations in Chicago

Select Alert Type and Subtype

ACCIDENT - ACCIDENT

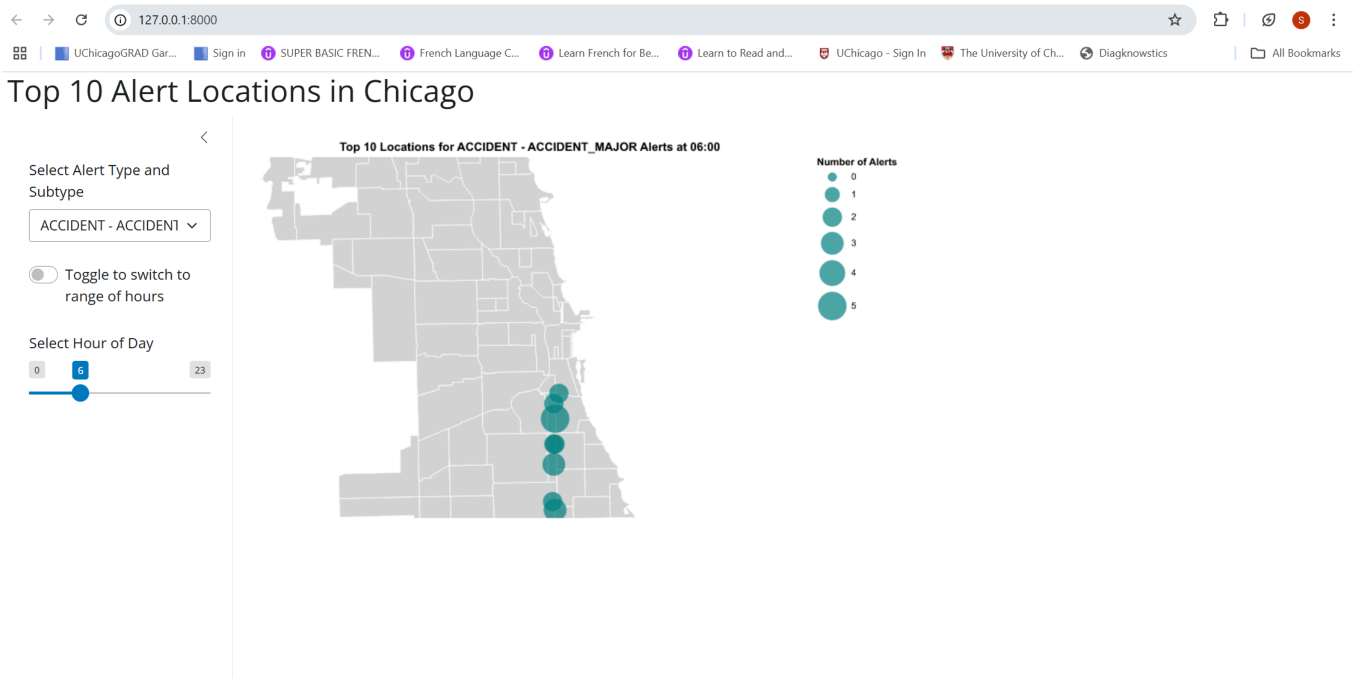
Toggle to switch to range of hours

Select Hour Range

0 6 9 23

Toggle On

c.



Top 10 Alert Locations in Chicago

Select Alert Type and Subtype

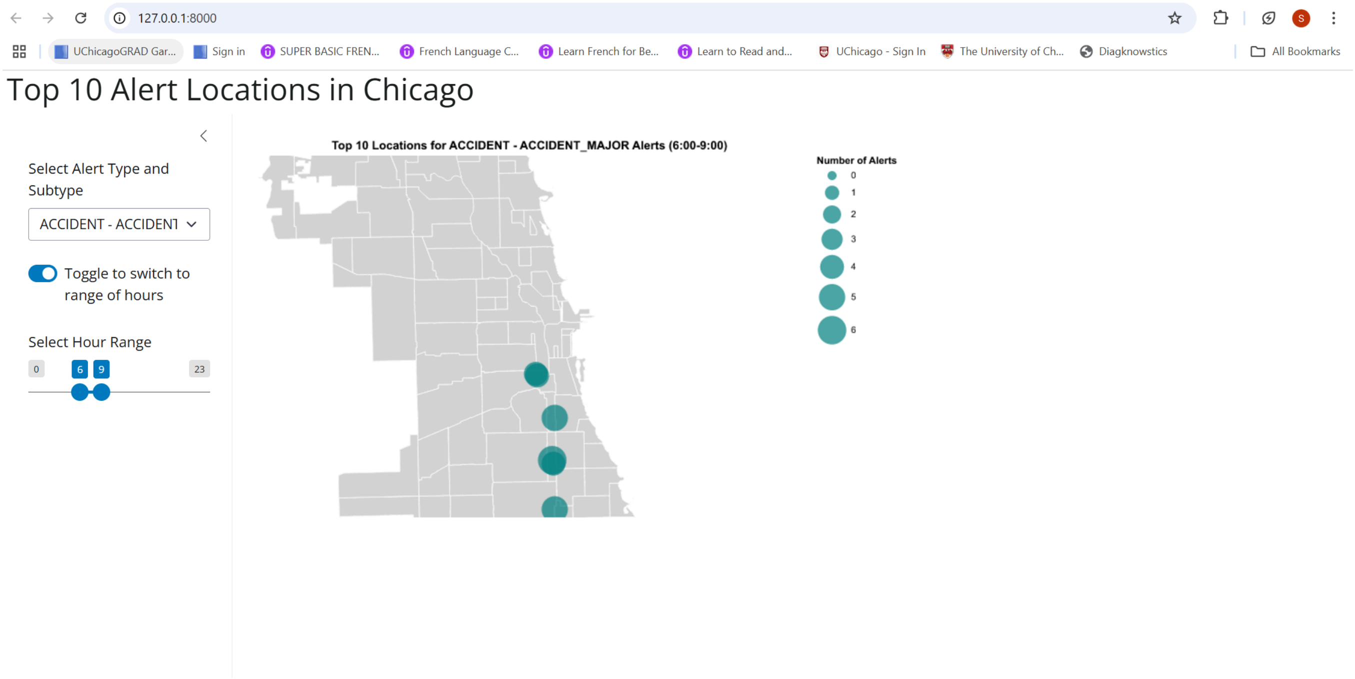
ACCIDENT - ACCIDENT

Toggle to switch to range of hours

Select Hour of Day

0 6 23

Toggle Off



Top 10 Alert Locations in Chicago

Select Alert Type and Subtype

ACCIDENT - ACCIDENT

Toggle to switch to range of hours

Select Hour Range

0 6 9 23

Top 10 Locations for ACCIDENT - ACCIDENT_MAJOR Alerts (6:00-9:00)
Number of Alerts
0
1
2
3
4
5
6

Toggle On

d.

To achieve this visualization, the app would need these changes: - Add a grid overlay with latitude/longitude coordinates - Color-code points by time period (red for morning, blue for afternoon) - Add a dual legend showing: - Time periods (Morning/Afternoon) - Circle sizes representing number of alerts - Replace the hour selection with a morning/afternoon toggle - Allow both time periods to be displayed simultaneously on the same map

These modifications would enable comparison of alert patterns between different times of day.