Problem Set 6 - Waze Shiny Dashboard

Summer Negahdar

2024-11-23

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: SN
2. “I have uploaded the names of anyone I worked with on the problem set
3. Late coins used this pset: 00 Late coins left after submission: 00

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!*

````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.

```
Waze= pd.read_csv('/Users/samarnegahdar/Desktop/untitled
 ↪  folder/student30538/problem_sets/ps6/waze_data/waze_data.csv')

Waze_df= pd.DataFrame(Waze)

##defining data types in altair syntax system

def map_to_altair_type(dtype):
    if pd.api.types.is_numeric_dtype(dtype):
        return "Q"
    elif pd.api.types.is_datetime64_any_dtype(dtype):
        return "T"
    elif pd.api.types.is_bool_dtype(dtype):
        return "N"
    elif pd.api.types.is_categorical_dtype(dtype):
        return "O"
    else:
        return "N"

##making a subset to ignor the three columns
Q1_subset = Waze_df.drop(columns=['geo', 'ts', 'geoWKT'])
##Assigning data types based on Altair syntax
altair_types_report = pd.DataFrame({
    "Column Name": Q1_subset.columns,
    "Altair Data Type": [map_to_altair_type(Q1_subset[col]) for col in
     ↪  Q1_subset.columns]
})
```

```
print(altair_types_report)
```

```
     Column Name Altair Data Type
0           city                N
1     confidence                Q
2      nThumbsUp                Q
3         street                N
4           uuid                N
5        country                N
6           type                N
7        subtype                N
8       roadType                Q
9    reliability                Q
10        magvar                Q
11  reportRating                Q
```

2.

```
##summing up NAs
missing_counts = Waze_df.isnull().sum()
not_missing_counts = Waze_df.notnull().sum()

missing_summary = pd.DataFrame({
    'Variable': Waze_df.columns,
    'Missing': missing_counts,
    'Not Missing': not_missing_counts
})

# Melt the DataFrame to make it suitable for Altair
melted_data = missing_summary.melt(
    id_vars='Variable',
    value_vars=['Missing', 'Not Missing'],
    var_name='Status',
    value_name='Count'
)

# Step 3: Plot the stacked bar chart
Q2_stack_chart = alt.Chart(melted_data).mark_bar().encode(
    x=alt.X('Variable:N', title='Variables',axis=alt.Axis(labelAngle=45)),
    y=alt.Y('Count:Q', title='Number of Observations'),
    color=alt.Color('Status:N', title='Status',
↪   scale=alt.Scale(domain=['Missing', 'Not Missing'], range=['red',
↪   'green'])),
    tooltip=['Variable', 'Status', 'Count']
).properties(
    title='NA vs non-NA Observations by Variable',
    width=800,
    height=400
)

Q2_stack_chart.display()

# Step 4: Analyze variables with missing values
# Find variables with missing values
variables_with_missing = missing_summary[missing_summary['Missing'] > 0]

# Find the variable with the highest share of missing values
missing_summary['Missing Share'] = missing_summary['Missing'] /
 ↪  (missing_summary['Missing'] + missing_summary['Not Missing'])
variable_highest_missing = missing_summary.loc[missing_summary['Missing
 ↪  Share'].idxmax()]

print("Variables with missing values:")
print(variables_with_missing)

print("\nVariable with the highest share of missing values:")
print(variable_highest_missing)
```

```
alt.Chart(...)

Variables with missing values:
            Variable  Missing  Not Missing
nThumbsUp  nThumbsUp   776723         1371
street        street    14073       764021
subtype      subtype    96086       682008

Variable with the highest share of missing values:
Variable         nThumbsUp
Missing             776723
Not Missing           1371
Missing Share     0.998238
Name: nThumbsUp, dtype: object
```

3.

a.

Print the unique values for the columns type and subtype. How many types have a subtype that is NA?

```python
unique_types = Waze_df['type'].unique()
unique_subtypes = Waze_df['subtype'].unique()

print(f"Unique 'type' values: {unique_types}")
print(f"Unique 'subtype' values: {unique_subtypes}")

##we need to group variables based on Type column and find those whose
 ↪  subtype is NA

missing_subtype_counts =
 ↪  Waze_df[Waze_df['subtype'].isna()].groupby('type').size()

print("Count of types with NA subtypes:")
print(missing_subtype_counts)

# Total number of types with at least one NA subtype
types_with_na_subtype = missing_subtype_counts.index.nunique()
print(f"Number of types with at least one NA subtype:
 ↪  {types_with_na_subtype}")
```

```
Unique 'type' values: ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
Unique 'subtype' values: [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']
Count of types with NA subtypes:
type
ACCIDENT       24359
```

```
HAZARD          3212
JAM            55041
ROAD_CLOSED    13474
dtype: int64
Number of types with at least one NA subtype: 4
```

b.

Printing all the unique combos of type and subtype, we have 24 unique combos.

```
unique_combinations = Waze_df[['type',
 ↪  'subtype']].drop_duplicates().reset_index().drop(columns= 'index')
print(unique_combinations)

detailed_types = unique_combinations.groupby('type')['subtype'].nunique()
print("Number of unique subtypes per type:")
print(detailed_types)
# to see which ones can have a sub-subtype, it means that there sohuld be
 ↪  more than one type and subtype combo for a specific subtype.
```

```
           type                             subtype
0           JAM                                 NaN
1      ACCIDENT                                 NaN
2   ROAD_CLOSED                                 NaN
3        HAZARD                                 NaN
4      ACCIDENT                      ACCIDENT_MAJOR
5      ACCIDENT                      ACCIDENT_MINOR
6        HAZARD                      HAZARD_ON_ROAD
7        HAZARD          HAZARD_ON_ROAD_CAR_STOPPED
8        HAZARD         HAZARD_ON_ROAD_CONSTRUCTION
9        HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE
10       HAZARD                  HAZARD_ON_ROAD_ICE
11       HAZARD               HAZARD_ON_ROAD_OBJECT
12       HAZARD             HAZARD_ON_ROAD_POT_HOLE
13       HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT
14       HAZARD                  HAZARD_ON_SHOULDER
15       HAZARD      HAZARD_ON_SHOULDER_CAR_STOPPED
16       HAZARD                      HAZARD_WEATHER
17       HAZARD                HAZARD_WEATHER_FLOOD
18          JAM                   JAM_HEAVY_TRAFFIC
19          JAM                JAM_MODERATE_TRAFFIC
20          JAM             JAM_STAND_STILL_TRAFFIC
```

```
21  ROAD_CLOSED                   ROAD_CLOSED_EVENT
22       HAZARD          HAZARD_ON_ROAD_LANE_CLOSED
23       HAZARD                  HAZARD_WEATHER_FOG
24  ROAD_CLOSED            ROAD_CLOSED_CONSTRUCTION
25       HAZARD            HAZARD_ON_ROAD_ROAD_KILL
26       HAZARD          HAZARD_ON_SHOULDER_ANIMALS
27       HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN
28          JAM                   JAM_LIGHT_TRAFFIC
29       HAZARD           HAZARD_WEATHER_HEAVY_SNOW
30  ROAD_CLOSED                  ROAD_CLOSED_HAZARD
31       HAZARD                 HAZARD_WEATHER_HAIL
Number of unique subtypes per type:
type
ACCIDENT        2
HAZARD         19
JAM             4
ROAD_CLOSED     3
Name: subtype, dtype: int64
```

```python
# I want to see how many different variations of subtype for each type I have
unique_subtypes_per_type = Waze_df.groupby('type')['subtype'].unique()

# Display the unique subtypes for each type
for type_value, subtypes in unique_subtypes_per_type.items():
    print(f"Type: {type_value}")
    print(f"Unique Subtypes: {list(subtypes)}")
    print("-" * 50)
```

```
Type: ACCIDENT
Unique Subtypes: [nan, 'ACCIDENT_MAJOR', 'ACCIDENT_MINOR']
--------------------------------------------------
Type: HAZARD
Unique Subtypes: [nan, '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']
--------------------------------------------------
```

```
Type: JAM
Unique Subtypes: [nan, 'JAM_HEAVY_TRAFFIC', 'JAM_MODERATE_TRAFFIC',
'JAM_STAND_STILL_TRAFFIC', 'JAM_LIGHT_TRAFFIC']
--------------------------------------------------
Type: ROAD_CLOSED
Unique Subtypes: [nan, 'ROAD_CLOSED_EVENT', 'ROAD_CLOSED_CONSTRUCTION',
'ROAD_CLOSED_HAZARD']
--------------------------------------------------
```

c.

Accident(t)> Major Minor Nan

Road_closed(t)> Event Nan

Hazard(t)> shoulder road: construction, car, emergency, traffic, pothole, object, lane_closure, road kill weather: snow, fog, flood

Jam(t)> heavy standstill moderate we can agree that HAZARD has enough subtypes with information that can have a new sub-subtype, also, JAM can have sub-subtype if we change the type to "traffic"

d.

```python
# I want to see how many NAs I have in the subtype column
na_subtype_count = Waze_df['subtype'].isna().sum()
ratio_of_subtype= na_subtype_count/ len(Waze_df)

print(f"The number of NA subtypes is: {na_subtype_count}")
print(ratio_of_subtype)
```

```
The number of NA subtypes is: 96086
0.12348893578410834
```

there are 96k subtype and which is almost 1/8th(12%) of our variables and I dont think removing them is smart. that is why I will change all of it to "unclassified"

```python
Waze_df['subtype'] = Waze_df['subtype'].fillna('Unclassified')
```

4.

a.

```
# Step 1: Extract unique combinations of type and subtype from Waze_df
crosswalk_df = Waze_df[['type',
 ↪   'subtype']].drop_duplicates().reset_index(drop=True)
```

b.

```
import pandas as pd

# Define updated hierarchy functions for subtype and sub-subtype
def assign_updated_subtype(row):
    """
    Categorize HAZARD subtypes into 'Road', 'Shoulder', or 'Weather'.
    Handle JAM and other types accordingly.
    """
    if row['type'] == 'HAZARD':
        if 'HAZARD_ON_SHOULDER' in str(row['subtype']).upper():
            return 'Shoulder'
        elif 'HAZARD_ON_ROAD' in str(row['subtype']).upper():
            return 'Road'
        elif 'HAZARD_WEATHER' in str(row['subtype']).upper():
            return 'Weather'
        else:
            return 'Unclassified'
    elif row['type'] == 'ACCIDENT':
        if 'ACCIDENT_MAJOR' in str(row['subtype']).upper():
            return 'Major'
        elif 'ACCIDENT_MINOR' in str(row['subtype']).upper():
            return 'Minor'
        else:
            return 'Unclassified'
    elif row['type'] == 'ROAD_CLOSED':
        if 'EVENT' in str(row['subtype']).upper():
            return 'Event'
        else:
            return 'Unclassified'
    elif row['type'] == 'JAM':
        return 'Traffic'
    else:
        return 'Unclassified'

def assign_updated_subsubtype(row):
    """
    Assign sub-subcategories for 'Road', 'Shoulder', and 'Weather' in HAZARD,
 ↪
    and for JAM subtypes.
    """
    if row['type'] == 'HAZARD' and row['updated_subtype'] == 'Road':
        if 'CONSTRUCTION' in str(row['subtype']).upper():
            return 'Construction'
        elif 'CAR_STOPPED' in str(row['subtype']).upper():
            return 'Car Stopped'
        elif 'EMERGENCY' in str(row['subtype']).upper():
            return 'Emergency Vehicle'
        elif 'TRAFFIC_LIGHT' in str(row['subtype']).upper():
            return 'Traffic Light Fault'
        elif 'POT_HOLE' in str(row['subtype']).upper():
            return 'Pothole'
        elif 'OBJECT' in str(row['subtype']).upper():
            return 'Object'
        elif 'LANE_CLOSED' in str(row['subtype']).upper():
            return 'Lane Closed'
        elif 'ROAD_KILL' in str(row['subtype']).upper():
            return 'Road Kill'
        else:
            return 'Unclassified'
    elif row['type'] == 'HAZARD' and row['updated_subtype'] == 'Weather':
        if 'SNOW' in str(row['subtype']).upper():
            return 'Snow'
        elif 'FOG' in str(row['subtype']).upper():
            return 'Fog'
        elif 'FLOOD' in str(row['subtype']).upper():
            return 'Flood'
        else:
            return 'Unclassified'
    elif row['type'] == 'HAZARD' and row['updated_subtype'] == 'Shoulder':
        if 'CAR_STOPPED' in str(row['subtype']).upper():
            return 'Car Stopped'
        else:
            return 'Unclassified'
    elif row['type'] == 'JAM':
        if 'HEAVY_TRAFFIC' in str(row['subtype']).upper():
            return 'Heavy'
        elif 'MODERATE_TRAFFIC' in str(row['subtype']).upper():
            return 'Moderate'
        elif 'STAND_STILL_TRAFFIC' in str(row['subtype']).upper():
            return 'Standstill'
        elif 'LIGHT_TRAFFIC' in str(row['subtype']).upper():
            return 'Light'
        else:
            return 'Unclassified'
    elif row['type'] == 'ACCIDENT':
        return row['updated_subtype']  # Keep Major/Minor as the sub-subtype
    elif row['type'] == 'ROAD_CLOSED':
        return row['updated_subtype']  # Keep Event as the sub-subtype
    return 'Unclassified'

# Assign updated_type to crosswalk
crosswalk_df['updated_type'] = crosswalk_df['type'].str.capitalize()

# Apply updated_subtype logic to crosswalk
crosswalk_df['updated_subtype'] = crosswalk_df.apply(assign_updated_subtype,
 ↪  axis=1)

# Apply updated_subsubtype logic to crosswalk
crosswalk_df['updated_subsubtype'] =
 ↪  crosswalk_df.apply(assign_updated_subsubtype, axis=1)

# Verify the updated crosswalk
print("Updated Crosswalk Table:")
print(crosswalk_df)

# Save the crosswalk table for reference
crosswalk_df.to_csv("crosswalk_table.csv", index=False)
print("Crosswalk table saved as 'crosswalk_table.csv'")
```

```
Updated Crosswalk Table:
           type                             subtype updated_type  \
0           JAM                        Unclassified          Jam
1      ACCIDENT                        Unclassified     Accident
2   ROAD_CLOSED                        Unclassified  Road_closed
3        HAZARD                        Unclassified       Hazard
```

```
4      ACCIDENT                      ACCIDENT_MAJOR     Accident
5      ACCIDENT                      ACCIDENT_MINOR     Accident
6        HAZARD                      HAZARD_ON_ROAD       Hazard
7        HAZARD          HAZARD_ON_ROAD_CAR_STOPPED       Hazard
8        HAZARD         HAZARD_ON_ROAD_CONSTRUCTION       Hazard
9        HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE       Hazard
10       HAZARD                  HAZARD_ON_ROAD_ICE       Hazard
11       HAZARD               HAZARD_ON_ROAD_OBJECT       Hazard
12       HAZARD             HAZARD_ON_ROAD_POT_HOLE       Hazard
13       HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT       Hazard
14       HAZARD                  HAZARD_ON_SHOULDER       Hazard
15       HAZARD      HAZARD_ON_SHOULDER_CAR_STOPPED       Hazard
16       HAZARD                      HAZARD_WEATHER       Hazard
17       HAZARD                HAZARD_WEATHER_FLOOD       Hazard
18          JAM                   JAM_HEAVY_TRAFFIC          Jam
19          JAM                JAM_MODERATE_TRAFFIC          Jam
20          JAM             JAM_STAND_STILL_TRAFFIC          Jam
21  ROAD_CLOSED                   ROAD_CLOSED_EVENT  Road_closed
22       HAZARD          HAZARD_ON_ROAD_LANE_CLOSED       Hazard
23       HAZARD                  HAZARD_WEATHER_FOG       Hazard
24  ROAD_CLOSED            ROAD_CLOSED_CONSTRUCTION  Road_closed
25       HAZARD            HAZARD_ON_ROAD_ROAD_KILL       Hazard
26       HAZARD          HAZARD_ON_SHOULDER_ANIMALS       Hazard
27       HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN       Hazard
28          JAM                   JAM_LIGHT_TRAFFIC          Jam
29       HAZARD           HAZARD_WEATHER_HEAVY_SNOW       Hazard
30  ROAD_CLOSED                  ROAD_CLOSED_HAZARD  Road_closed
31       HAZARD                 HAZARD_WEATHER_HAIL       Hazard

   updated_subtype   updated_subsubtype
0          Traffic         Unclassified
1     Unclassified         Unclassified
2     Unclassified         Unclassified
3     Unclassified         Unclassified
4            Major                Major
5            Minor                Minor
6             Road         Unclassified
7             Road          Car Stopped
8             Road         Construction
9             Road    Emergency Vehicle
10            Road         Unclassified
11            Road               Object
12            Road              Pothole
```

```
13            Road  Traffic Light Fault
14        Shoulder         Unclassified
15        Shoulder          Car Stopped
16         Weather         Unclassified
17         Weather                Flood
18         Traffic                Heavy
19         Traffic             Moderate
20         Traffic           Standstill
21           Event                Event
22            Road          Lane Closed
23         Weather                  Fog
24    Unclassified         Unclassified
25            Road            Road Kill
26        Shoulder         Unclassified
27        Shoulder         Unclassified
28         Traffic                Light
29         Weather                 Snow
30    Unclassified         Unclassified
31         Weather         Unclassified
Crosswalk table saved as 'crosswalk_table.csv'
```

c.

```
# Merge the crosswalk with the original data
Waze_merged_df = Waze_df.merge(
    crosswalk_df,
    on=['type', 'subtype'],
    how='left'
)

# Check for rows where type is 'Accident' and subtype is 'Unclassified'
accident_unclassified_count = Waze_merged_df[
    (Waze_merged_df['type'] == 'ACCIDENT') &
    (Waze_merged_df['subtype'] == 'Unclassified')
].shape[0]

# Display the count
print(f"Number of rows for Accident - Unclassified:
 ↪  {accident_unclassified_count}")
```

```
Number of rows for Accident - Unclassified: 24359
```

d.

```
# Step 2: Check consistency between 'type' and 'subtype' in crosswalk_df and
 ↪  merged_df
crosswalk_types = crosswalk_df[['type', 'subtype']].drop_duplicates()
merged_types = Waze_merged_df[['type', 'subtype']].drop_duplicates()

# Step 3: Verify that all combinations in crosswalk are in merged_df
missing_in_merged = crosswalk_types.merge(
    merged_types,
    on=['type', 'subtype'],
    how='left',
    indicator=True
).query("_merge == 'left_only'")

# Print results
if missing_in_merged.empty:
    print("All type and subtype combinations in the crosswalk are present in
     ↪  the merged dataset.")
else:
    print("The following type and subtype combinations in the crosswalk are
     ↪  missing in the merged dataset:")
    print(missing_in_merged[['type', 'subtype']])
```

```
All type and subtype combinations in the crosswalk are present in the merged
dataset.
```

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

1.

a.

```
import re

def extract_coordinates(geo_string):
    if pd.notna(geo_string):  # Ensure the string is not NaN
        match = re.search(r"POINT\(([-\d.]+) ([-\d.]+)\)", geo_string)
        if match:
            return float(match.group(1)), float(match.group(2))
    return None, None

# Apply the function to extract latitude and longitude
Waze_merged_df["longitude"], Waze_merged_df["latitude"] =
 ↪  zip(*Waze_merged_df["geo"].apply(extract_coordinates))

# Display the updated DataFrame
print(Waze_merged_df.head(5))
```

```
          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  Unclassified
1  ad7761f8-d3cb-4623-951d-dafb419a3ec3      US     ACCIDENT  Unclassified
2  0e5f14ae-7251-46af-a7f1-53a5272cd37d      US  ROAD_CLOSED  Unclassified
3  654870a4-a71a-450b-9f22-bc52ae4f69a5      US          JAM  Unclassified
4  926ff228-7db9-4e0d-b6cf-6739211ffc8b      US          JAM  Unclassified

   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 updated_type  \
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

  updated_subtype updated_subsubtype  longitude   latitude
0         Traffic       Unclassified -87.676685  41.929692
1    Unclassified       Unclassified -87.624816  41.753358
2    Unclassified       Unclassified -87.614122  41.889821
3         Traffic       Unclassified -87.680139  41.939093
```

```
4         Traffic       Unclassified -87.735235  41.916580
```

b.

```python
# Ensure latitude and longitude are numeric
Waze_merged_df['latitude'] = pd.to_numeric(Waze_merged_df['latitude'],
 ↪  errors='coerce')
Waze_merged_df['longitude'] = pd.to_numeric(Waze_merged_df['longitude'],
 ↪  errors='coerce')

# Bin latitude and longitude with a step size of 0.01
Waze_merged_df['binned_latitude'] = (Waze_merged_df['latitude'] // 0.01 *
 ↪  0.01).round(2)
Waze_merged_df['binned_longitude'] = (Waze_merged_df['longitude'] // 0.01 *
 ↪  0.01).round(2)

# Combine the binned latitude and longitude into a single column for unique
 ↪  combinations
Waze_merged_df['binned_lat_lon'] =
 ↪  list(zip(Waze_merged_df['binned_latitude'],
 ↪  Waze_merged_df['binned_longitude']))

# Count occurrences of each binned latitude-longitude combination
binned_counts = Waze_merged_df['binned_lat_lon'].value_counts().reset_index()
binned_counts.columns = ['binned_lat_lon', 'count']

# Find the combination with the greatest number of observations
most_common_bin = binned_counts.iloc[0]
print(f"Most frequent binned latitude-longitude:
 ↪  {most_common_bin['binned_lat_lon']} with {most_common_bin['count']}
 ↪  observations")

# Create the 'type_subtype' column by combining 'updated_type' and
 ↪  'updated_subtype'
Waze_merged_df['type_subtype'] = Waze_merged_df['updated_type'] + " - " +
 ↪  Waze_merged_df['updated_subtype']

##now I want to save Waze_merged_df as a csv so I can use it later for my
 ↪  shiny app!!
Waze_merged_df.to_csv('/Users/samarnegahdar/Desktop/untitled
 ↪  folder/student30538/problem_sets/ps6/Waze_merged_data.csv', index=False)
```

```
Most frequent binned latitude-longitude: (41.96, -87.75) with 26537
observations
```

c.

```
# Filter data for the chosen type and subtype

Q2_1c = Waze_merged_df[
    (Waze_merged_df['type'] == 'HAZARD') &
    (Waze_merged_df['updated_subtype'] == 'Weather')
]

# Aggregate data at the binned latitude-longitude level
top_alerts_map = (
    Q2_1c.groupby('binned_lat_lon')
    .size()
    .reset_index(name='alert_count')
    .sort_values(by='alert_count', ascending=False)
    .head(10)  # Top 10 bins
)

print(f"Level of Aggregation: Binned latitude-longitude")
print(f"Number of Rows: {len(top_alerts_map)}")
print(top_alerts_map)
```

```
Level of Aggregation: Binned latitude-longitude
Number of Rows: 10
      binned_lat_lon  alert_count
6    (41.65, -87.59)          176
13    (41.66, -87.6)          113
304  (41.85, -87.65)           97
370  (41.89, -87.62)           82
387   (41.9, -87.63)           78
352  (41.88, -87.65)           78
253  (41.82, -87.72)           78
269   (41.83, -87.7)           72
289  (41.84, -87.65)           70
398  (41.91, -87.67)           70
```

2.

a.

```
# Using 'updated_type' and 'updated_subsubtype' for filtering
jam_heavy_df = Waze_merged_df[
    (Waze_merged_df['updated_type'].str.contains('jam', case=False,
 ↪  na=False)) &  # Case insensitive filtering
    (Waze_merged_df['updated_subsubtype'].str.contains('heavy', case=False,
 ↪  na=False))  # Case insensitive filtering
]

# Check if any data exists after filtering
print(jam_heavy_df.shape)  # Check number of rows after filtering
print(jam_heavy_df[['updated_type', 'updated_subsubtype',
 ↪  'binned_lat_lon']].head())  # Check the first few rows

# Step 2: Aggregate the data to find the number of alerts for each
 ↪  binned_lat_lon
top_jam_heavy = (
    jam_heavy_df.groupby('binned_lat_lon')
    .size()
    .reset_index(name='alert_count')
    .sort_values(by='alert_count', ascending=False)
    .head(10)
)

# Step 3: Handle possible string representations of tuples or lists
top_jam_heavy['binned_lat_lon'] =
 ↪  top_jam_heavy['binned_lat_lon'].apply(lambda x: ast.literal_eval(x) if
 ↪  isinstance(x, str) else x)

# Step 4: Split the binned_lat_lon into latitude and longitude columns
top_jam_heavy[['latitude', 'longitude']] = pd.DataFrame(
    top_jam_heavy['binned_lat_lon'].tolist(),
    index=top_jam_heavy.index
)

# Check the result
print(top_jam_heavy.head())

# Step 4: Create the scatter plot using Altair
scatter_plot_Jam = alt.Chart(top_jam_heavy).mark_circle().encode(
    x=alt.X('longitude:Q', title='Longitude', scale=alt.Scale(domain=[-87.8,
 ↪  -87.4])),
    y=alt.Y('latitude:Q', title='Latitude', scale=alt.Scale(domain=[41.8,
 ↪  42.0])),
    size=alt.Size('alert_count:Q', title='Number of Alerts',
 ↪  legend=alt.Legend(title="Alert Count")),
    tooltip=['latitude:Q', 'longitude:Q', 'alert_count:Q']
).properties(
    title='Top 10 Locations with Most Jam - Heavy Traffic Alerts',
    width=600,
    height=400
).project(type="identity", reflectY=True)  # Apply the same Mercator
 ↪  projection to the scatter plot

scatter_plot_Jam.display()
```

```
(170442, 24)
    updated_type updated_subsubtype   binned_lat_lon
858          Jam              Heavy  (41.89, -87.66)
859          Jam              Heavy  (41.89, -87.66)
860          Jam              Heavy  (41.89, -87.66)
861          Jam              Heavy  (41.92, -87.69)
862          Jam              Heavy  (41.89, -87.62)
      binned_lat_lon  alert_count  latitude  longitude
382  (41.89, -87.66)         4991     41.89     -87.66
349  (41.87, -87.65)         4121     41.87     -87.65
401   (41.9, -87.67)         3845     41.90     -87.67
518  (41.96, -87.75)         3360     41.96     -87.75
366  (41.88, -87.65)         3267     41.88     -87.65
```

```
alt.Chart(...)
```

3.

```
import requests

# Send a GET request to the URL
response =
 ↪  requests.get("https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&forma
```

b.

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

# Load the GeoJSON data
geojson_file = '/Users/samarnegahdar/Desktop/untitled
 ↪  folder/student30538/problem_sets/ps6/chicago_neighborhoods.geojson'
with open(geojson_file) as f:
    chicago_geojson = json.load(f)

# Convert GeoJSON to Altair's data format
geo_data = alt.Data(values=chicago_geojson["features"])
```

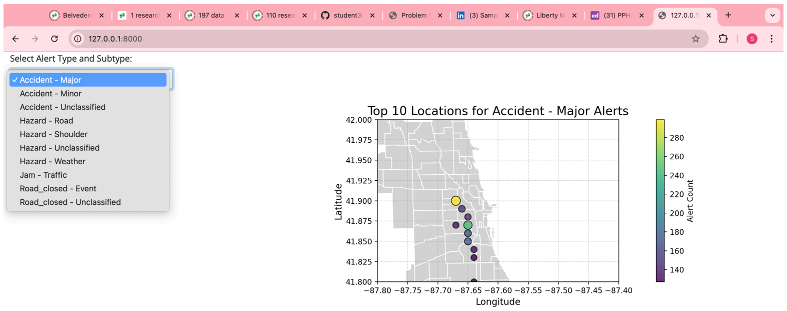
4.

```
# Define latitude and longitude ranges (domain)
lat_range = [41.8, 42.0]
lon_range = [-87.8, -87.4]

# Base map using GeoJSON with specific latitude and longitude ranges
base_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',  # Fill color for the map
    stroke='white'     # Border color for neighborhoods
).project(
    type='mercator',
    scale=100000,  # Adjust scale to zoom in to the specific region
    center=[-87.58, 41.9],  # Center the map on Chicago's approximate
 ↪  coordinates
).properties(
    width=600,
    height=400
).encode(
    longitude='longitude:Q',
    latitude='latitude:Q'
)

# Adjust the base map's latitude and longitude range (domain)
base_map = base_map.encode(
    x=alt.X('longitude:Q', scale=alt.Scale(domain=lon_range)),  # Longitude
 ↪  domain
    y=alt.Y('latitude:Q', scale=alt.Scale(domain=lat_range))    # Latitude
 ↪  domain
)

# Display the base map
base_map.show()

# Layer the scatter plot on top of the base map
layered_chart_Jam = base_map + scatter_plot_Jam

# Display the final layered chart
layered_chart_Jam
```

```
alt.Chart(...)
```

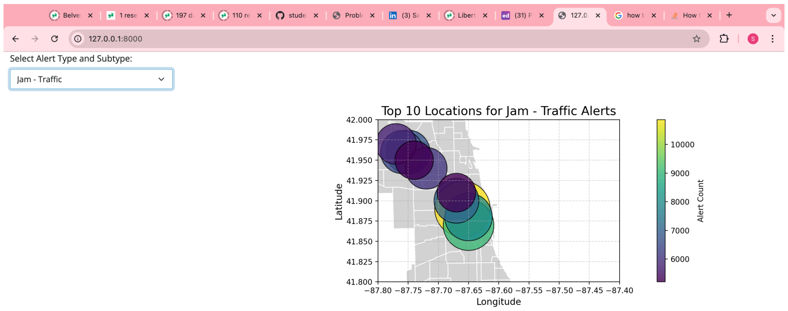
```
alt.LayerChart(...)
```

5.



a. there are 10 options for the dropdown menu

b. I just wanted to mention that since you asked for dropdown menu to be a combo of type and subtype, we cannot define “heavy traffic” as in type and subtype there is only

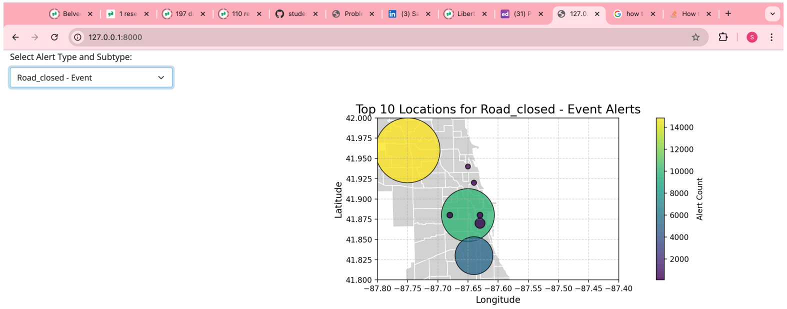


Select Alert Type and Subtype:

Jam - Traffic

jam-traffic!!!

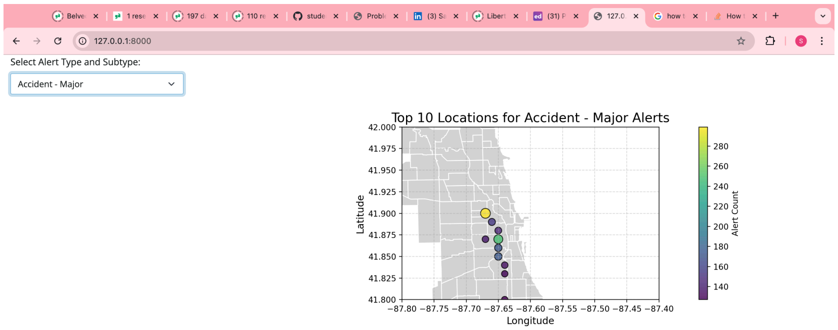
c. mosly in bucktown and chinatown and somewhere up north-west (that I exactly don't know what it is called)



Select Alert Type and Subtype:

Road_closed - Event

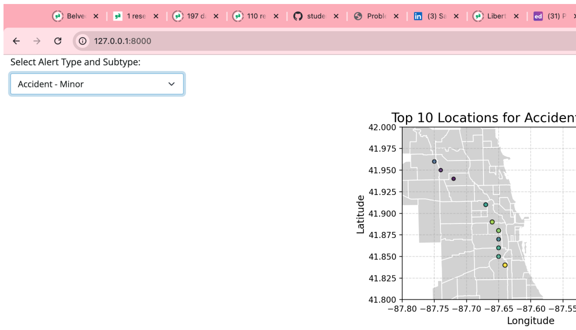
d. are there more major accidents in downtown or minor accidents? there are mor major accidents than minor accidents in downtown (which is interesting since there is a speed limit inside the city vs. on highways/freeways)



127.0.0.1:8000

Select Alert Type and Subtype:

Accident - Major



127.0.0.1:8000

Select Alert Type and Subtype:

Accident - Minor

e. I would say the most crucial thing to add is time frame, either annual or daily (like which times of the year which events happen or which times of the day certain alerts are more frequent)

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

1.

a. I would say using the raw data from ts is not smart as it will give us too many bins based on each day. what we want to to have an accumulated set of data for each hour across time (days,weeks,...) so its better to limit the categories if we want the user to choose a specific time.

b.

```python
import pandas as pd
import os

# Ensure the 'ts' column is in datetime format
Waze_merged_df['ts'] = pd.to_datetime(Waze_merged_df['ts'], errors='coerce')

# Create a new 'hour' column by extracting the hour from the 'ts' column
Waze_merged_df['hour'] =
↪   Waze_merged_df['ts'].dt.hour.astype(str).str.zfill(2) + ":00"
# We will now collapse the dataset, aggregating by hour and binned
↪   latitude-longitude to get the count of alerts
```

```
collapsed_df = (
    Waze_merged_df.groupby(['hour', 'binned_lat_lon'])
    .size()
    .reset_index(name='alert_count')
)

# Sort the dataset by hour and alert count (descending)
collapsed_df = collapsed_df.sort_values(by=['hour', 'alert_count'],
 ↪  ascending=[True, False])

# Get the top 10 locations per hour
top_alerts_by_hour = collapsed_df.groupby('hour').head(10)

# Check how many rows this dataset has
print(f"The collapsed dataset has {top_alerts_by_hour.shape[0]} rows.")

# Save the collapsed dataset as 'top_alerts_map_byhour.csv'
output_file = os.path.join('/Users/samarnegahdar/Desktop/untitled
 ↪  folder/student30538/problem_sets/ps6/top_alerts_map_byhour',
 ↪  'top_alerts_map_byhour.csv')
top_alerts_by_hour.to_csv(output_file, index=False)

print(f"Collapsed dataset saved as {output_file}")
```

```
The collapsed dataset has 240 rows.
Collapsed dataset saved as /Users/samarnegahdar/Desktop/untitled
folder/student30538/problem_sets/ps6/top_alerts_map_byhour/top_alerts_map_byhour.csv
```

c.

```
selected_hours= ['11:00', '14:00', '22:00']
##the jam-heavy df does not have hour column so I will add it:
jam_heavy_df['hour']= Waze_merged_df['hour']
# Define a function to create the plot for a given hour
def create_hourly_plot(hour):
    # Filter data for the selected hour
    hour_data = jam_heavy_df[jam_heavy_df['hour'] == hour]

    # Aggregate the data to find the number of alerts for each binned_lat_lon
    top_alerts = (
        hour_data.groupby('binned_lat_lon')
        .size()
        .reset_index(name='alert_count')
        .sort_values(by='alert_count', ascending=False)
        .head(10)
    )

    # Split the 'binned_lat_lon' into latitude and longitude
    top_alerts[['latitude', 'longitude']] = pd.DataFrame(
        top_alerts['binned_lat_lon'].tolist(),
        index=top_alerts.index
    )

    # Create the scatter plot
    scatter_plot_hourly = alt.Chart(top_alerts).mark_circle().encode(
        x=alt.X('longitude:Q', title='Longitude',
 ↪  scale=alt.Scale(domain=[-87.8, -87.4])),
        y=alt.Y('latitude:Q', title='Latitude', scale=alt.Scale(domain=[41.8,
 ↪  42.0])),
        size=alt.Size('alert_count:Q', title='Number of Alerts',
 ↪  legend=alt.Legend(title="Alert Count")),
        tooltip=['latitude:Q', 'longitude:Q', 'alert_count:Q']
    ).properties(
        title=f'Top 10 Locations for Jam - Heavy Traffic Alerts at {hour}',
        width=600,
        height=400
    ).project(type="identity", reflectY=True)



    # Layer the scatter plot on top of the base map
    layered_chart_3C = base_map + scatter_plot_hourly

    return layered_chart_3C

# Generate and display the plots for the selected hours
for hour in selected_hours:
    plot = create_hourly_plot(hour)
    plot.display()
```

```
/var/folders/j5/rv933w1173s068kbzq0kp2xh0000gn/T/ipykernel_50401/91981137.py:3:
SettingWithCopyWarning:
```

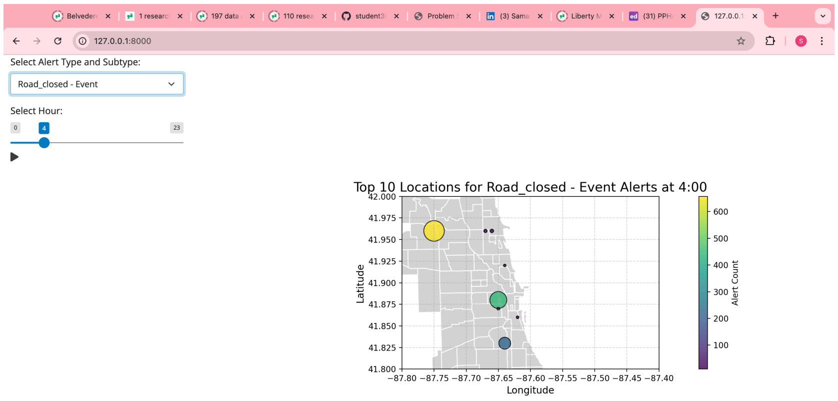
```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
```

```
alt.LayerChart(...)
```

```
alt.LayerChart(...)
```

```
alt.LayerChart(...)
```

2.



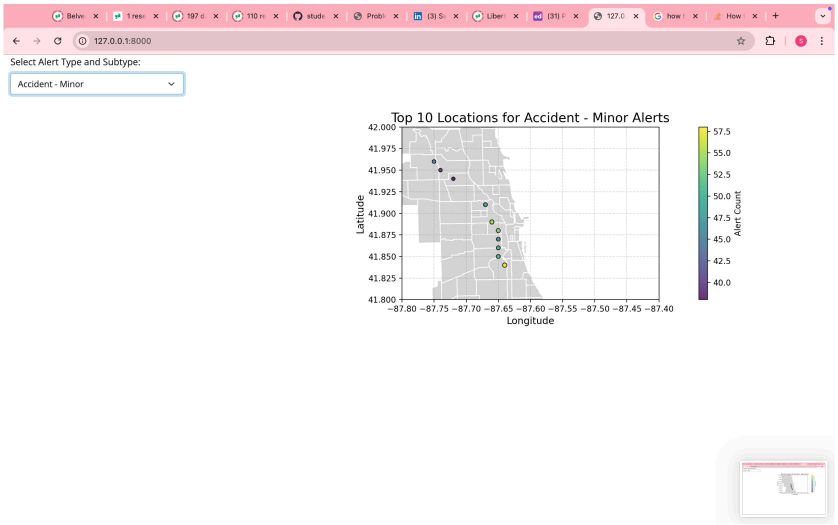
127.0.0.1:8000

Select Alert Type and Subtype: Road_closed - Event

Select Hour: 4

Figure 1: Hourly alerts

a.

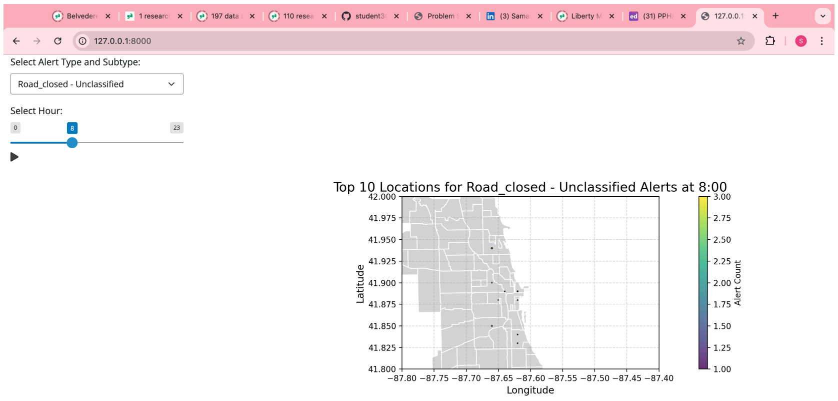

Select Alert Type and Subtype:

Accident - Minor

Figure 2: Minor Accident alerts

b.

c. I would like to remind again that you asked the wrong combination (if we wanted to see road closed- hazard-construction we needed subcategory and sub-subcategory) but now is too late for me to go back and your instructions were wrong so although I know I should look at the sub-category and sub-sub-category I will look at road-close- unclassified for morning and night.

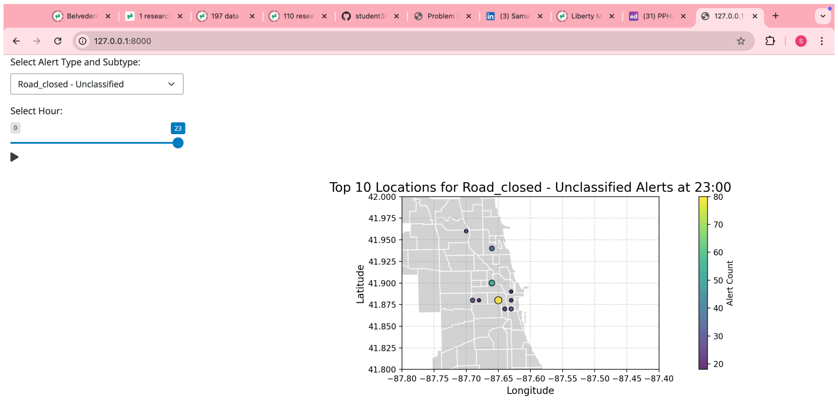

Select Alert Type and Subtype:

Road_closed - Unclassified

Select Hour:

0 8 23

Figure 3: road closed morning



Select Alert Type and Subtype:

Road_closed - Unclassified

Select Hour:

0

23

if you take a look at the maps, you will see how most constructions take place at night

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

1.

a. I don't think it is a good idea to do so since the range needs to be flexible while collapsing the date takes away the flexibility. still I think it would not be easy on shiny if we leave the range super open so maybe we can do a combination? where the hours will still be whole numbers but they range is defined within those hours.

b.

```
# Ensure the 'hour' column is in the correct format
jam_heavy_df['hour'] = pd.to_datetime(jam_heavy_df['hour'], errors='coerce').dt.strftime('%H:%M')

# Filter for the time range 6AM-9AM and for a specific type-subtype combination (e.g., 'Jam - Heavy Traffic')
filtered_df = jam_heavy_df[jam_heavy_df['hour'].isin(['06:00', '07:00', '08:00', '09:00'])]
filtered_df = filtered_df[filtered_df['type_subtype'] == 'Jam - Traffic']  # Filter for the specific type-subtype

# If 'alert_count' does not exist, create it (assuming we're counting occurrences of 'binned_lat_lon')
```

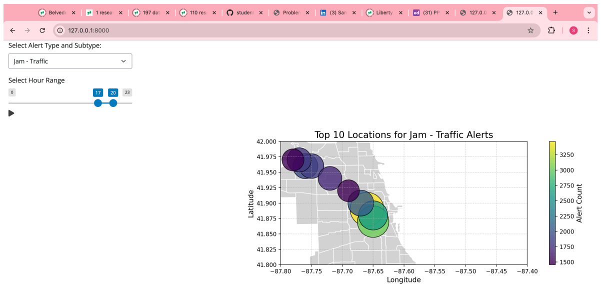
```
if 'alert_count' not in filtered_df.columns:
    filtered_df['alert_count'] =
 ↪  filtered_df.groupby('binned_lat_lon')['binned_lat_lon'].transform('count')

# Aggregate alert counts by type_subtype and binned_lat_lon
top_locations = (
    filtered_df.groupby(['type_subtype', 'binned_lat_lon'])['alert_count']
    .sum()
    .reset_index()
    .sort_values(by='alert_count', ascending=False)
    .head(10)
)

# Check if 'binned_lat_lon' contains valid tuples (latitude, longitude)
top_locations = top_locations[top_locations['binned_lat_lon'].apply(lambda x:
 ↪  isinstance(x, tuple) and len(x) == 2)]

# If no valid locations, print a warning and exit
if top_locations.empty:
    print("Warning: No valid locations found for the selected type_subtype.")
else:
    # Split 'binned_lat_lon' into latitude and longitude columns
    top_locations[['latitude', 'longitude']] = pd.DataFrame(
        top_locations['binned_lat_lon'].tolist(),
        index=top_locations.index
    )

# Define latitude and longitude ranges for the map (for Chicago)
lat_range = [41.8, 42.0]
lon_range = [-87.8, -87.4]

# Create the base map using Altair with specified latitude and longitude
 ↪  ranges
base_map_hourly_jam = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',  # Fill color for the map
    stroke='white'     # Border color for neighborhoods
).project(
    type='mercator',
    scale=70000,  # Adjust scale to zoom in to the specific region # Center
 ↪  the map on Chicago's approximate coordinates
).properties(
    width=600,
    height=400
).encode(
    longitude='longitude:Q',
    latitude='latitude:Q'
)

# Adjust the base map's latitude and longitude range (domain)
base_map_hourly_jam = base_map_hourly_jam.encode(
    x=alt.X('longitude:Q', scale=alt.Scale(domain=lon_range)),  # Longitude
 ↪  domain
    y=alt.Y('latitude:Q', scale=alt.Scale(domain=lat_range))    # Latitude
 ↪  domain
)

# Create the scatter plot for the top 10 locations
scatter_plot = alt.Chart(top_locations).mark_circle().encode(
    x=alt.X('longitude:Q', title='Longitude',
 ↪  scale=alt.Scale(domain=lon_range)),  # Longitude domain
    y=alt.Y('latitude:Q', title='Latitude',
 ↪  scale=alt.Scale(domain=lat_range)),  # Latitude domain
    size=alt.Size('alert_count:Q', title='Alert Count',
 ↪  legend=alt.Legend(title="Alert Count")),
    color=alt.Color('alert_count:Q', title='Alert Count',
 ↪  scale=alt.Scale(scheme='viridis')),
    tooltip=['latitude:Q', 'longitude:Q', 'alert_count:Q']
).properties(
    title='Top 10 Locations for Jam - Heavy Traffic (6AM-9AM)',
    width=600,
    height=400
)

# Create the layered map by adding the scatter plot to the base map
layered_chart_hourly_jam = base_map + scatter_plot

# Show the layered map
layered_chart_hourly_jam.display()
```

```
/var/folders/j5/rv933w1173s068kbzq0kp2xh0000gn/T/ipykernel_50401/375493366.py:2:
UserWarning:
```

```
Could not infer format, so each element will be parsed individually, falling
back to `dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.

/var/folders/j5/rv933w1173s068kbzq0kp2xh0000gn/T/ipykernel_50401/375493366.py:2:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu

alt.LayerChart(...)
```

2.

a.



127.0.0.1:8000

Select Alert Type and Subtype:

Jam - Traffic

Select Hour Range

0 17 20 23

Figure 4: Traffic jam with an hiyrly range

b. I used the same settings for part a so did not include the same photo twice

3. I cannot get the third part to work before everything falls into pieces.

a.
b.
c.
d.