Problem Set 6

Yunzhou Guo

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: YG
2. "I have uploaded the names of anyone I worked with on the problem set **here**" YG (2 point)
3. Late coins used this pset: 0 Late coins left after submission: 3
4. Before starting the problem set, make sure to read and agree to the terms of data usage for the Waze data here.
5. Knit your `ps6.qmd` as a pdf document and name it `ps6.pdf`.
6. Submit your `ps6.qmd`, `ps6.pdf`, `requirements.txt`, and all created folders (we will create three Shiny apps so you will have at least three additional folders) to the gradescope repo assignment (5 points).
7. Submit `ps6.pdf` and also link your Github repo via Gradescope (5 points)
8. Tag your submission in Gradescope. For the Code Style part (10 points) please tag the whole correspondingsection for the code style rubric.

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

IMPORTANT: For the App portion of the PS, in case you can not arrive to the expected functional dashboard we will need to take a look at your `app.py` file. You can use the following *code chunk template to "import" and print the content of that file. Please, don't forget to also tag the corresponding code chunk as part of your submission!*

Background

Data Download and Exploration (20 points)

1.

```
import zipfile
import pandas as pd

with zipfile.ZipFile('waze_data.zip', 'r') as zip_ref:
    zip_ref.extractall('waze_data')

df = pd.read_csv('waze_data/waze_data_sample.csv')

columns_to_check = [col for col in df.columns if col not in ['ts', 'geo',
↪  'geoWKT']]
column_types = {}

for col in columns_to_check:
    if pd.api.types.is_numeric_dtype(df[col]):
        column_types[col] = 'Quantitative'
    elif pd.api.types.is_datetime64_any_dtype(df[col]):
        column_types[col] = 'Temporal'
    else:
        column_types[col] = 'Nominal'

print("Variable Names and their Altair Data Types:")
for col, altair_type in column_types.items():
    print(f"{col}: {altair_type}")
```

```
Variable Names and their 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.

```python
import pandas as pd
import altair as alt

df = pd.read_csv('waze_data/waze_data.csv')

null_counts = df.isnull().sum()
non_null_counts = df.notnull().sum()

data = pd.DataFrame({
    'variable': null_counts.index,
    'null_count': null_counts.values,
    'non_null_count': non_null_counts.values
})

data_long = data.melt(id_vars='variable', value_vars=['null_count',
 ↪  'non_null_count'],
                      var_name='missing_status', value_name='count')

chart = alt.Chart(data_long).mark_bar().encode(
    x=alt.X('variable:N', title='Variable'),
    y=alt.Y('count:Q', title='Count of Observations'),
    color=alt.Color('missing_status:N', title='Missing Status',
                    scale=alt.Scale(domain=['null_count', 'non_null_count'],
                                    range=['red', 'green'])),
    tooltip=['variable', 'missing_status', 'count']
).properties(
    title="Null and Non-Null Counts for Each Variable in waze_data.csv"
)

chart.display()

variables_with_nulls = null_counts[null_counts > 0].index.tolist()
variable_highest_missing_share = null_counts.idxmax()
highest_missing_share_ratio = null_counts.max() / len(df)
```

```
print("Variables with NULL values:", variables_with_nulls)
print(f"Variable with the highest share of missing values:
 ↪  {variable_highest_missing_share} "
      f"({highest_missing_share_ratio:.2%} missing)")
```

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

```
Variables with NULL values: ['nThumbsUp', 'street', 'subtype']
Variable with the highest share of missing values: nThumbsUp (99.82% missing)
```

3.

```
import pandas as pd

df = pd.read_csv('waze_data/waze_data.csv')

unique_types = df['type'].unique()
unique_subtypes = df['subtype'].unique()

type_crosswalk = pd.DataFrame({
    'original_type': unique_types,
    'cleaned_type': [f'Cleaned_Type_{i+1}' for i in range(len(unique_types))]
     ↪
})

subtype_crosswalk = pd.DataFrame({
    'original_subtype': unique_subtypes,
    'cleaned_subtype': [f'Cleaned_Subtype_{i+1}' for i in
     ↪  range(len(unique_subtypes))]
})

df_cleaned = df.merge(type_crosswalk, how='left', left_on='type',
 ↪  right_on='original_type')
df_cleaned = df_cleaned.merge(subtype_crosswalk, how='left',
 ↪  left_on='subtype', right_on='original_subtype')

df_cleaned.drop(columns=['type', 'subtype', 'original_type',
 ↪  'original_subtype'], inplace=True)

print("Type Crosswalk Table:\n", type_crosswalk)
```

```python
print("Subtype Crosswalk Table:\n", subtype_crosswalk)
print("Cleaned DataFrame:\n", df_cleaned.head())
```

```
Type Crosswalk Table:
   original_type    cleaned_type
0           JAM  Cleaned_Type_1
1      ACCIDENT  Cleaned_Type_2
2   ROAD_CLOSED  Cleaned_Type_3
3        HAZARD  Cleaned_Type_4
Subtype Crosswalk Table:
                       original_subtype     cleaned_subtype
0                                   NaN   Cleaned_Subtype_1
1                        ACCIDENT_MAJOR   Cleaned_Subtype_2
2                        ACCIDENT_MINOR   Cleaned_Subtype_3
3                        HAZARD_ON_ROAD   Cleaned_Subtype_4
4            HAZARD_ON_ROAD_CAR_STOPPED   Cleaned_Subtype_5
5           HAZARD_ON_ROAD_CONSTRUCTION   Cleaned_Subtype_6
6      HAZARD_ON_ROAD_EMERGENCY_VEHICLE   Cleaned_Subtype_7
7                    HAZARD_ON_ROAD_ICE   Cleaned_Subtype_8
8                 HAZARD_ON_ROAD_OBJECT   Cleaned_Subtype_9
9               HAZARD_ON_ROAD_POT_HOLE  Cleaned_Subtype_10
10  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT  Cleaned_Subtype_11
11                   HAZARD_ON_SHOULDER  Cleaned_Subtype_12
12       HAZARD_ON_SHOULDER_CAR_STOPPED  Cleaned_Subtype_13
13                       HAZARD_WEATHER  Cleaned_Subtype_14
14                 HAZARD_WEATHER_FLOOD  Cleaned_Subtype_15
15                    JAM_HEAVY_TRAFFIC  Cleaned_Subtype_16
16                 JAM_MODERATE_TRAFFIC  Cleaned_Subtype_17
17              JAM_STAND_STILL_TRAFFIC  Cleaned_Subtype_18
18                    ROAD_CLOSED_EVENT  Cleaned_Subtype_19
19           HAZARD_ON_ROAD_LANE_CLOSED  Cleaned_Subtype_20
20                   HAZARD_WEATHER_FOG  Cleaned_Subtype_21
21             ROAD_CLOSED_CONSTRUCTION  Cleaned_Subtype_22
22             HAZARD_ON_ROAD_ROAD_KILL  Cleaned_Subtype_23
23           HAZARD_ON_SHOULDER_ANIMALS  Cleaned_Subtype_24
24      HAZARD_ON_SHOULDER_MISSING_SIGN  Cleaned_Subtype_25
25                    JAM_LIGHT_TRAFFIC  Cleaned_Subtype_26
26            HAZARD_WEATHER_HEAVY_SNOW  Cleaned_Subtype_27
27                   ROAD_CLOSED_HAZARD  Cleaned_Subtype_28
28                  HAZARD_WEATHER_HAIL  Cleaned_Subtype_29
Cleaned 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  roadType  reliability  \
0  004025a4-5f14-4cb7-9da6-2615daafbf37      US        20            5
1  ad7761f8-d3cb-4623-951d-dafb419a3ec3      US         4            8
2  0e5f14ae-7251-46af-a7f1-53a5272cd37d      US         1            5
3  654870a4-a71a-450b-9f22-bc52ae4f69a5      US        20            5
4  926ff228-7db9-4e0d-b6cf-6739211ffc8b      US        20            5

   magvar  reportRating                       ts                          geo
   \
0     139             3  2024-02-04 16:40:41 UTC  POINT(-87.676685 41.929692)
1       2             2  2024-02-04 20:01:27 UTC  POINT(-87.624816 41.753358)
2     344             2  2024-02-04 02:15:54 UTC  POINT(-87.614122 41.889821)
3     264             2  2024-02-04 00:30:54 UTC  POINT(-87.680139 41.939093)
4     359             0  2024-02-04 03:27:35 UTC   POINT(-87.735235 41.91658)

                        geoWKT    cleaned_type    cleaned_subtype
0  Point(-87.676685 41.929692)  Cleaned_Type_1  Cleaned_Subtype_1
1  Point(-87.624816 41.753358)  Cleaned_Type_2  Cleaned_Subtype_1
2  Point(-87.614122 41.889821)  Cleaned_Type_3  Cleaned_Subtype_1
3  Point(-87.680139 41.939093)  Cleaned_Type_1  Cleaned_Subtype_1
4   Point(-87.735235 41.91658)  Cleaned_Type_1  Cleaned_Subtype_1
```

4.

a.

```python
df = pd.read_csv('waze_data/waze_data.csv')

unique_types = df['type'].unique()
unique_subtypes = df['subtype'].unique()

type_crosswalk = pd.DataFrame({
    'type': unique_types,
    'updated_type': [f'Cleaned_Type_{i+1}' for i in range(len(unique_types))]
})
```

```
subtype_crosswalk = pd.DataFrame({
    'subtype': unique_subtypes,
    'updated_subtype': [f'Cleaned_Subtype_{i+1}' for i in
     ↪  range(len(unique_subtypes))]
})

crosswalk_data = []

for _, row in df[['type', 'subtype']].drop_duplicates().iterrows():
    updated_type = f"Cleaned_{row['type']}"
    updated_subtype = f"Cleaned_{row['subtype']}" if
 ↪  pd.notnull(row['subtype']) else 'Unclassified'
    updated_subsubtype = None

    crosswalk_data.append({
        'type': row['type'],
        'subtype': row['subtype'],
        'updated_type': updated_type,
        'updated_subtype': updated_subtype,
        'updated_subsubtype': updated_subsubtype
    })

crosswalk_df = pd.DataFrame(crosswalk_data)

df_cleaned = df.merge(crosswalk_df, on=['type', 'subtype'], how='left')

df_cleaned.drop(columns=['type', 'subtype'], inplace=True)

print("Crosswalk DataFrame:\n", crosswalk_df)
print("Data with Updated Hierarchy:\n", df_cleaned.head())
```

```
Crosswalk DataFrame:
             type                             subtype         updated_type  \
0            JAM                                 NaN          Cleaned_JAM
1       ACCIDENT                                 NaN     Cleaned_ACCIDENT
2    ROAD_CLOSED                                 NaN  Cleaned_ROAD_CLOSED
3         HAZARD                                 NaN       Cleaned_HAZARD
4       ACCIDENT                      ACCIDENT_MAJOR     Cleaned_ACCIDENT
5       ACCIDENT                      ACCIDENT_MINOR     Cleaned_ACCIDENT
6         HAZARD                      HAZARD_ON_ROAD       Cleaned_HAZARD
7         HAZARD         HAZARD_ON_ROAD_CAR_STOPPED       Cleaned_HAZARD
8         HAZARD        HAZARD_ON_ROAD_CONSTRUCTION       Cleaned_HAZARD
```

```
9        HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE       Cleaned_HAZARD
10       HAZARD                  HAZARD_ON_ROAD_ICE       Cleaned_HAZARD
11       HAZARD               HAZARD_ON_ROAD_OBJECT       Cleaned_HAZARD
12       HAZARD             HAZARD_ON_ROAD_POT_HOLE       Cleaned_HAZARD
13       HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT       Cleaned_HAZARD
14       HAZARD                  HAZARD_ON_SHOULDER       Cleaned_HAZARD
15       HAZARD      HAZARD_ON_SHOULDER_CAR_STOPPED       Cleaned_HAZARD
16       HAZARD                      HAZARD_WEATHER       Cleaned_HAZARD
17       HAZARD                HAZARD_WEATHER_FLOOD       Cleaned_HAZARD
18          JAM                   JAM_HEAVY_TRAFFIC          Cleaned_JAM
19          JAM                JAM_MODERATE_TRAFFIC          Cleaned_JAM
20          JAM             JAM_STAND_STILL_TRAFFIC          Cleaned_JAM
21  ROAD_CLOSED                   ROAD_CLOSED_EVENT  Cleaned_ROAD_CLOSED
22       HAZARD          HAZARD_ON_ROAD_LANE_CLOSED       Cleaned_HAZARD
23       HAZARD                  HAZARD_WEATHER_FOG       Cleaned_HAZARD
24  ROAD_CLOSED            ROAD_CLOSED_CONSTRUCTION  Cleaned_ROAD_CLOSED
25       HAZARD            HAZARD_ON_ROAD_ROAD_KILL       Cleaned_HAZARD
26       HAZARD          HAZARD_ON_SHOULDER_ANIMALS       Cleaned_HAZARD
27       HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN       Cleaned_HAZARD
28          JAM                   JAM_LIGHT_TRAFFIC          Cleaned_JAM
29       HAZARD           HAZARD_WEATHER_HEAVY_SNOW       Cleaned_HAZARD
30  ROAD_CLOSED                  ROAD_CLOSED_HAZARD  Cleaned_ROAD_CLOSED
31       HAZARD                 HAZARD_WEATHER_HAIL       Cleaned_HAZARD

                               updated_subtype updated_subsubtype
0                                 Unclassified               None
1                                 Unclassified               None
2                                 Unclassified               None
3                                 Unclassified               None
4                       Cleaned_ACCIDENT_MAJOR               None
5                       Cleaned_ACCIDENT_MINOR               None
6                       Cleaned_HAZARD_ON_ROAD               None
7           Cleaned_HAZARD_ON_ROAD_CAR_STOPPED               None
8          Cleaned_HAZARD_ON_ROAD_CONSTRUCTION               None
9     Cleaned_HAZARD_ON_ROAD_EMERGENCY_VEHICLE               None
10                 Cleaned_HAZARD_ON_ROAD_ICE               None
11              Cleaned_HAZARD_ON_ROAD_OBJECT               None
12            Cleaned_HAZARD_ON_ROAD_POT_HOLE               None
13 Cleaned_HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT               None
14                 Cleaned_HAZARD_ON_SHOULDER               None
15     Cleaned_HAZARD_ON_SHOULDER_CAR_STOPPED               None
16                     Cleaned_HAZARD_WEATHER               None
17               Cleaned_HAZARD_WEATHER_FLOOD               None
```

```
18                   Cleaned_JAM_HEAVY_TRAFFIC               None
19                Cleaned_JAM_MODERATE_TRAFFIC               None
20             Cleaned_JAM_STAND_STILL_TRAFFIC               None
21                   Cleaned_ROAD_CLOSED_EVENT               None
22          Cleaned_HAZARD_ON_ROAD_LANE_CLOSED               None
23                  Cleaned_HAZARD_WEATHER_FOG               None
24            Cleaned_ROAD_CLOSED_CONSTRUCTION               None
25            Cleaned_HAZARD_ON_ROAD_ROAD_KILL               None
26          Cleaned_HAZARD_ON_SHOULDER_ANIMALS               None
27     Cleaned_HAZARD_ON_SHOULDER_MISSING_SIGN               None
28                   Cleaned_JAM_LIGHT_TRAFFIC               None
29           Cleaned_HAZARD_WEATHER_HEAVY_SNOW               None
30                  Cleaned_ROAD_CLOSED_HAZARD               None
31                 Cleaned_HAZARD_WEATHER_HAIL               None
Data with Updated Hierarchy:
           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  roadType  reliability  \
0  004025a4-5f14-4cb7-9da6-2615daafbf37      US        20            5
1  ad7761f8-d3cb-4623-951d-dafb419a3ec3      US         4            8
2  0e5f14ae-7251-46af-a7f1-53a5272cd37d      US         1            5
3  654870a4-a71a-450b-9f22-bc52ae4f69a5      US        20            5
4  926ff228-7db9-4e0d-b6cf-6739211ffc8b      US        20            5

   magvar  reportRating                       ts                          geo
   \
0     139            3  2024-02-04 16:40:41 UTC  POINT(-87.676685 41.929692)
1       2            2  2024-02-04 20:01:27 UTC  POINT(-87.624816 41.753358)
2     344            2  2024-02-04 02:15:54 UTC  POINT(-87.614122 41.889821)
3     264            2  2024-02-04 00:30:54 UTC  POINT(-87.680139 41.939093)
4     359            0  2024-02-04 03:27:35 UTC   POINT(-87.735235 41.91658)

                        geoWKT         updated_type updated_subtype  \
0  Point(-87.676685 41.929692)          Cleaned_JAM    Unclassified
1  Point(-87.624816 41.753358)     Cleaned_ACCIDENT    Unclassified
2  Point(-87.614122 41.889821)  Cleaned_ROAD_CLOSED    Unclassified
3  Point(-87.680139 41.939093)          Cleaned_JAM    Unclassified
4   Point(-87.735235 41.91658)          Cleaned_JAM    Unclassified
```

```
  updated_subsubtype
0               None
1               None
2               None
3               None
4               None
```

b.

```python
unique_combinations = df[['type', 'subtype']].drop_duplicates()

type_mapping = {
    "ACCIDENT": "Accident",
    "CONSTRUCTION": "Construction",
}

subtype_mapping = {
    "ACCIDENT_MAJOR": ("Accident", "Major"),
    "ACCIDENT_MINOR": ("Accident", "Minor"),
}

crosswalk_data = []

for _, row in unique_combinations.iterrows():
    updated_type = type_mapping.get(row['type'], f"Cleaned_{row['type']}")
    if pd.isna(row['subtype']):
        updated_subtype = "Unclassified"
        updated_subsubtype = None
    else:
        updated_subtype, updated_subsubtype = subtype_mapping.get(
            row['subtype'], (f"Cleaned_{row['subtype']}", None)
        )

    crosswalk_data.append({
        'type': row['type'],
        'subtype': row['subtype'],
        'updated_type': updated_type,
        'updated_subtype': updated_subtype,
        'updated_subsubtype': updated_subsubtype
    })

crosswalk_df = pd.DataFrame(crosswalk_data)
```

```python
print("Crosswalk DataFrame:\n", crosswalk_df)
print("Number of unique combinations in crosswalk:", crosswalk_df.shape[0])

df_cleaned = df.merge(crosswalk_df, on=['type', 'subtype'], how='left')

df_cleaned.drop(columns=['type', 'subtype'], inplace=True)

print("Data with Updated Hierarchy:\n", df_cleaned.head())
```

```
Crosswalk DataFrame:
            type                             subtype         updated_type  \
0           JAM                                 NaN          Cleaned_JAM
1      ACCIDENT                                 NaN             Accident
2   ROAD_CLOSED                                 NaN  Cleaned_ROAD_CLOSED
3        HAZARD                                 NaN       Cleaned_HAZARD
4      ACCIDENT                      ACCIDENT_MAJOR             Accident
5      ACCIDENT                      ACCIDENT_MINOR             Accident
6        HAZARD                      HAZARD_ON_ROAD       Cleaned_HAZARD
7        HAZARD          HAZARD_ON_ROAD_CAR_STOPPED       Cleaned_HAZARD
8        HAZARD         HAZARD_ON_ROAD_CONSTRUCTION       Cleaned_HAZARD
9        HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE       Cleaned_HAZARD
10       HAZARD                  HAZARD_ON_ROAD_ICE       Cleaned_HAZARD
11       HAZARD               HAZARD_ON_ROAD_OBJECT       Cleaned_HAZARD
12       HAZARD             HAZARD_ON_ROAD_POT_HOLE       Cleaned_HAZARD
13       HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT       Cleaned_HAZARD
14       HAZARD                  HAZARD_ON_SHOULDER       Cleaned_HAZARD
15       HAZARD      HAZARD_ON_SHOULDER_CAR_STOPPED       Cleaned_HAZARD
16       HAZARD                      HAZARD_WEATHER       Cleaned_HAZARD
17       HAZARD                HAZARD_WEATHER_FLOOD       Cleaned_HAZARD
18          JAM                   JAM_HEAVY_TRAFFIC          Cleaned_JAM
19          JAM                JAM_MODERATE_TRAFFIC          Cleaned_JAM
20          JAM             JAM_STAND_STILL_TRAFFIC          Cleaned_JAM
21  ROAD_CLOSED                   ROAD_CLOSED_EVENT  Cleaned_ROAD_CLOSED
22       HAZARD          HAZARD_ON_ROAD_LANE_CLOSED       Cleaned_HAZARD
23       HAZARD                  HAZARD_WEATHER_FOG       Cleaned_HAZARD
24  ROAD_CLOSED            ROAD_CLOSED_CONSTRUCTION  Cleaned_ROAD_CLOSED
25       HAZARD            HAZARD_ON_ROAD_ROAD_KILL       Cleaned_HAZARD
26       HAZARD          HAZARD_ON_SHOULDER_ANIMALS       Cleaned_HAZARD
27       HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN       Cleaned_HAZARD
28          JAM                   JAM_LIGHT_TRAFFIC          Cleaned_JAM
29       HAZARD           HAZARD_WEATHER_HEAVY_SNOW       Cleaned_HAZARD
```

```
30  ROAD_CLOSED                  ROAD_CLOSED_HAZARD  Cleaned_ROAD_CLOSED
31       HAZARD                 HAZARD_WEATHER_HAIL       Cleaned_HAZARD

                               updated_subtype updated_subsubtype
0                                 Unclassified               None
1                                 Unclassified               None
2                                 Unclassified               None
3                                 Unclassified               None
4                                     Accident              Major
5                                     Accident              Minor
6                       Cleaned_HAZARD_ON_ROAD               None
7           Cleaned_HAZARD_ON_ROAD_CAR_STOPPED               None
8          Cleaned_HAZARD_ON_ROAD_CONSTRUCTION               None
9      Cleaned_HAZARD_ON_ROAD_EMERGENCY_VEHICLE               None
10                  Cleaned_HAZARD_ON_ROAD_ICE               None
11               Cleaned_HAZARD_ON_ROAD_OBJECT               None
12             Cleaned_HAZARD_ON_ROAD_POT_HOLE               None
13  Cleaned_HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT               None
14                  Cleaned_HAZARD_ON_SHOULDER               None
15      Cleaned_HAZARD_ON_SHOULDER_CAR_STOPPED               None
16                      Cleaned_HAZARD_WEATHER               None
17                Cleaned_HAZARD_WEATHER_FLOOD               None
18                   Cleaned_JAM_HEAVY_TRAFFIC               None
19                Cleaned_JAM_MODERATE_TRAFFIC               None
20             Cleaned_JAM_STAND_STILL_TRAFFIC               None
21                   Cleaned_ROAD_CLOSED_EVENT               None
22          Cleaned_HAZARD_ON_ROAD_LANE_CLOSED               None
23                  Cleaned_HAZARD_WEATHER_FOG               None
24            Cleaned_ROAD_CLOSED_CONSTRUCTION               None
25            Cleaned_HAZARD_ON_ROAD_ROAD_KILL               None
26          Cleaned_HAZARD_ON_SHOULDER_ANIMALS               None
27     Cleaned_HAZARD_ON_SHOULDER_MISSING_SIGN               None
28                   Cleaned_JAM_LIGHT_TRAFFIC               None
29           Cleaned_HAZARD_WEATHER_HEAVY_SNOW               None
30                  Cleaned_ROAD_CLOSED_HAZARD               None
31                 Cleaned_HAZARD_WEATHER_HAIL               None
Number of unique combinations in crosswalk: 32
Data with Updated Hierarchy:
           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  roadType  reliability  \
0  004025a4-5f14-4cb7-9da6-2615daafbf37      US        20            5
1  ad7761f8-d3cb-4623-951d-dafb419a3ec3      US         4            8
2  0e5f14ae-7251-46af-a7f1-53a5272cd37d      US         1            5
3  654870a4-a71a-450b-9f22-bc52ae4f69a5      US        20            5
4  926ff228-7db9-4e0d-b6cf-6739211ffc8b      US        20            5

   magvar  reportRating                       ts                          geo
   \
0     139             3  2024-02-04 16:40:41 UTC  POINT(-87.676685 41.929692)
1       2             2  2024-02-04 20:01:27 UTC  POINT(-87.624816 41.753358)
2     344             2  2024-02-04 02:15:54 UTC  POINT(-87.614122 41.889821)
3     264             2  2024-02-04 00:30:54 UTC  POINT(-87.680139 41.939093)
4     359             0  2024-02-04 03:27:35 UTC   POINT(-87.735235 41.91658)

                        geoWKT         updated_type updated_subtype  \
0  Point(-87.676685 41.929692)          Cleaned_JAM    Unclassified
1  Point(-87.624816 41.753358)             Accident    Unclassified
2  Point(-87.614122 41.889821)  Cleaned_ROAD_CLOSED    Unclassified
3  Point(-87.680139 41.939093)          Cleaned_JAM    Unclassified
4   Point(-87.735235 41.91658)          Cleaned_JAM    Unclassified

  updated_subsubtype
0               None
1               None
2               None
3               None
4               None
```

c.

```
df_cleaned = df.merge(crosswalk_df, on=['type', 'subtype'], how='left')

accident_unclassified_count = df_cleaned[
    (df_cleaned['updated_type'] == 'Accident') &
    (df_cleaned['updated_subtype'] == 'Unclassified')
].shape[0]

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

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

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

1.

a.

```python
import pandas as pd
import re

df = pd.read_csv('waze_data/waze_data.csv')

df[['longitude', 'latitude']] = df['geo'].str.extract(r'POINT \(([-\d.]+)
↪   ([-\d.]+)\)')
df['latitude'] = pd.to_numeric(df['latitude'], errors='coerce')
df['longitude'] = pd.to_numeric(df['longitude'], errors='coerce')

print("Number of missing latitude values:", df['latitude'].isna().sum())
print("Number of missing longitude values:", df['longitude'].isna().sum())

df = df.dropna(subset=['latitude', 'longitude'])
```

```
Number of missing latitude values: 778094
Number of missing longitude values: 778094
```

b.

```python
import pandas as pd
import re

df = pd.read_csv('waze_data/waze_data.csv')

print("First few rows of geo column:")
print(df['geo'].head())

df[['longitude', 'latitude']] = df['geo'].str.extract(r'POINT \(([-\d.]+)
↪   ([-\d.]+)\)')

print(f"Number of missing latitude values: {df['latitude'].isna().sum()}")
print(f"Number of missing longitude values: {df['longitude'].isna().sum()}")
```

```python
df['latitude'] = pd.to_numeric(df['latitude'], errors='coerce')
df['longitude'] = pd.to_numeric(df['longitude'], errors='coerce')

df = df.dropna(subset=['latitude', 'longitude'])

df['binned_latitude'] = df['latitude'].round(2)
df['binned_longitude'] = df['longitude'].round(2)

df['binned_coordinates'] = list(zip(df['binned_latitude'],
 ↪  df['binned_longitude']))

print(f"Number of missing binned coordinates:
 ↪  {df['binned_coordinates'].isna().sum()}")
print("First few binned coordinates:")
print(df[['binned_latitude', 'binned_longitude',
 ↪  'binned_coordinates']].head())

df = df.dropna(subset=['binned_coordinates'])

binned_counts = df['binned_coordinates'].value_counts()

if not binned_counts.empty:
    most_common_bin = binned_counts.idxmax()
    most_common_bin_count = binned_counts.max()
    print(f"The binned latitude-longitude combination with the greatest
     ↪  number of observations is: {most_common_bin}")
    print(f"Number of observations in this bin: {most_common_bin_count}")
else:
    print("No binned coordinates found.")

top_alerts_df =
 ↪  df.groupby(['binned_coordinates']).size().reset_index(name='alert_count')

top_alerts_df.to_csv('top_alerts_map/top_alerts_map.csv', index=False)

print(f"Number of rows in the top_alerts_map DataFrame:
 ↪  {top_alerts_df.shape[0]}")
```

```
First few rows of geo column:
0    POINT(-87.676685 41.929692)
1    POINT(-87.624816 41.753358)
```

```
2    POINT(-87.614122 41.889821)
3    POINT(-87.680139 41.939093)
4     POINT(-87.735235 41.91658)
Name: geo, dtype: object
Number of missing latitude values: 778094
Number of missing longitude values: 778094
Number of missing binned coordinates: 0
First few binned coordinates:
Empty DataFrame
Columns: [binned_latitude, binned_longitude, binned_coordinates]
Index: []
No binned coordinates found.
Number of rows in the top_alerts_map DataFrame: 0
```

c.

```
chosen_type = 'Accident'
chosen_subtype = 'Unclassified'

filtered_df = df[(df['type'] == chosen_type) & (df['subtype'] ==
 ↪  chosen_subtype)]

aggregated_df =
 ↪  filtered_df.groupby(['binned_coordinates']).size().reset_index(name='alert_count')

top_10_alerts = aggregated_df.sort_values(by='alert_count',
 ↪  ascending=False).head(10)

top_10_alerts.to_csv('top_alerts_map/top_alerts_map.csv', index=False)

print("Level of aggregation: Data is aggregated by binned coordinates
 ↪  (latitude and longitude).")
print(f"Number of rows in the final DataFrame (Top 10 binned coordinates):
 ↪  {top_10_alerts.shape[0]}")
```

```
Level of aggregation: Data is aggregated by binned coordinates (latitude and
longitude).
Number of rows in the final DataFrame (Top 10 binned coordinates): 0
```

2.

```python
import pandas as pd

df = pd.read_csv('waze_data/waze_data.csv')

print("\nUnique values in the 'type' column:")
print(df['type'].unique())

print("\nFirst few rows of the 'geo' column:")
print(df['geo'].head())

print("\nMissing values in 'type' and 'geo' columns:")
print(df[['type', 'geo']].isnull().sum())
```

```
Unique values in the 'type' column:
['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']

First few rows of the 'geo' column:
0    POINT(-87.676685 41.929692)
1    POINT(-87.624816 41.753358)
2    POINT(-87.614122 41.889821)
3    POINT(-87.680139 41.939093)
4     POINT(-87.735235 41.91658)
Name: geo, dtype: object

Missing values in 'type' and 'geo' columns:
type    0
geo     0
dtype: int64
```

```python
valid_geo_df = df.dropna(subset=['geo'])

valid_geo_df[['longitude', 'latitude']] =
 ↪  valid_geo_df['geo'].str.extract(r'POINT\((-?\d+\.\d+) (-?\d+\.\d+)\)')

print("\nNumber of missing latitude values:",
 ↪  valid_geo_df['latitude'].isna().sum())
print("Number of missing longitude values:",
 ↪  valid_geo_df['longitude'].isna().sum())

print("\nFirst few rows after extracting coordinates:")
print(valid_geo_df[['geo', 'longitude', 'latitude']].head())
```

```
Number of missing latitude values: 0
Number of missing longitude values: 0

First few rows after extracting coordinates:
                           geo   longitude   latitude
0  POINT(-87.676685 41.929692)  -87.676685  41.929692
1  POINT(-87.624816 41.753358)  -87.624816  41.753358
2  POINT(-87.614122 41.889821)  -87.614122  41.889821
3  POINT(-87.680139 41.939093)  -87.680139  41.939093
4   POINT(-87.735235 41.91658)  -87.735235   41.91658
```

```
valid_geo_df['binned_latitude'] = valid_geo_df['latitude'].round(2)
valid_geo_df['binned_longitude'] = valid_geo_df['longitude'].round(2)

valid_geo_df['binned_coordinates'] =
 ↪  list(zip(valid_geo_df['binned_latitude'],
 ↪  valid_geo_df['binned_longitude']))

print("\nUnique binned coordinates:")
print(valid_geo_df['binned_coordinates'].unique())
```

```
Unique binned coordinates:
[('41.929692', '-87.676685') ('41.753358', '-87.624816')
 ('41.889821', '-87.614122') ... ('41.954212', '-87.645009')
 ('41.887432', '-87.615862') ('41.887442', '-87.615882')]
```

```
aggregated_df = valid_geo_df.groupby(['binned_coordinates', 'type',
 ↪  'subtype']).size().reset_index(name='alert_count')

jam_heavy_traffic_df = aggregated_df[aggregated_df['type'] == 'JAM']

top_10_alerts = jam_heavy_traffic_df.sort_values(by='alert_count',
 ↪  ascending=False).head(10)

print("\nTop 10 'Jam - Heavy Traffic' alerts:")
print(top_10_alerts[['binned_coordinates', 'alert_count']].head())
```

```
Top 10 'Jam - Heavy Traffic' alerts:
             binned_coordinates  alert_count
294185  (41.880559, -87.645263)           11
333095  (41.893597, -87.656027)            8
294369   (41.88061, -87.645262)            7
451512  (41.941924, -87.702779)            7
563688  (41.981468, -87.782524)            7
```

```
top_10_alerts[['binned_latitude', 'binned_longitude']] =
 ↪  pd.DataFrame(top_10_alerts['binned_coordinates'].to_list(),
 ↪  index=top_10_alerts.index)

top_10_alerts['binned_latitude'] =
 ↪  top_10_alerts['binned_latitude'].astype(float)
top_10_alerts['binned_longitude'] =
 ↪  top_10_alerts['binned_longitude'].astype(float)

print("\nTop 10 Alerts after splitting coordinates:")
print(top_10_alerts[['binned_coordinates', 'binned_latitude',
 ↪  'binned_longitude', 'alert_count']])
```

```
Top 10 Alerts after splitting coordinates:
             binned_coordinates  binned_latitude  binned_longitude  \
294185  (41.880559, -87.645263)        41.880559        -87.645263
333095  (41.893597, -87.656027)        41.893597        -87.656027
294369   (41.88061, -87.645262)        41.880610        -87.645262
451512  (41.941924, -87.702779)        41.941924        -87.702779
563688  (41.981468, -87.782524)        41.981468        -87.782524
215035  (41.867025, -87.619029)        41.867025        -87.619029
404766  (41.924847, -87.683472)        41.924847        -87.683472
404767  (41.924847, -87.683472)        41.924847        -87.683472
571823  (41.982313, -87.792593)        41.982313        -87.792593
590969   (41.985212, -87.66218)        41.985212        -87.662180

        alert_count
294185           11
333095            8
294369            7
451512            7
563688            7
```

```
215035             6
404766             6
404767             6
571823             6
590969             6
```

```python
import altair as alt

chart = alt.Chart(top_10_alerts).mark_circle(size=200).encode(
    x=alt.X('binned_longitude:Q',
 ↪  scale=alt.Scale(domain=[top_10_alerts['binned_longitude'].min(),
 ↪  top_10_alerts['binned_longitude'].max()]), axis=alt.Axis(format=".5f")),
    y=alt.Y('binned_latitude:Q',
 ↪  scale=alt.Scale(domain=[top_10_alerts['binned_latitude'].min(),
 ↪  top_10_alerts['binned_latitude'].max()]), axis=alt.Axis(format=".5f")),
    size='alert_count:Q',
    tooltip=['binned_coordinates', 'alert_count']
).properties(
    title='Top 10 "Jam - Heavy Traffic" Alerts'
)

chart.show()
```

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

3.

a.

```python
import requests

geojson_url =
 ↪  'https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON'

response = requests.get(geojson_url)

if response.status_code == 200:
    with open('./top_alerts_map/chicago-boundaries.geojson', 'wb') as f:
        f.write(response.content)
    print("GeoJSON file downloaded successfully.")
else:
    print(f"Failed to download GeoJSON. Status code: {response.status_code}")
```

```
GeoJSON file downloaded successfully.
```

b.

```
import json
import altair as alt

file_path = './top_alerts_map/chicago-boundaries.geojson'

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

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

map_chart = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray', stroke='black'
).properties(
    title='Chicago Neighborhood Boundaries'
)

map_chart.show()
```

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

```
scatter_plot = alt.Chart(top_10_alerts).mark_circle(size=200).encode(
    x=alt.X('binned_longitude:Q',
 ↪  scale=alt.Scale(domain=[top_10_alerts['binned_longitude'].min(),
 ↪  top_10_alerts['binned_longitude'].max()]), axis=alt.Axis(format=".5f")),
    y=alt.Y('binned_latitude:Q',
 ↪  scale=alt.Scale(domain=[top_10_alerts['binned_latitude'].min(),
 ↪  top_10_alerts['binned_latitude'].max()]), axis=alt.Axis(format=".5f")),
    size='alert_count:Q',
    tooltip=['binned_coordinates', 'alert_count']
)

final_chart = map_chart + scatter_plot

final_chart.show()
```

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

4.

```
import altair as alt
import json

file_path = './top_alerts_map/chicago-boundaries.geojson'
with open(file_path) as f:
    chicago_geojson = json.load(f)

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

map_chart = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray', stroke='black'
).properties(
    title='Chicago Neighborhood Boundaries',
    width=600,
    height=400
)
```

```
scatter_plot = alt.Chart(top_10_alerts).mark_circle(size=200).encode(
    x=alt.X('binned_longitude:Q',
 ↪  scale=alt.Scale(domain=[top_10_alerts['binned_longitude'].min(),
 ↪  top_10_alerts['binned_longitude'].max()]), axis=alt.Axis(format=".5f")),
    y=alt.Y('binned_latitude:Q',
 ↪  scale=alt.Scale(domain=[top_10_alerts['binned_latitude'].min(),
 ↪  top_10_alerts['binned_latitude'].max()]), axis=alt.Axis(format=".5f")),
    size='alert_count:Q',
    tooltip=['binned_coordinates', 'alert_count']
)
```

```
lat_min, lat_max = 41.6, 42.1
lon_min, lon_max = -87.9, -87.5

map_chart = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray', stroke='black'
).properties(
    title='Chicago Neighborhood Boundaries',
    width=600,
    height=400
).project(
    type='identity'
)
```

```
scatter_plot = alt.Chart(top_10_alerts).mark_circle(size=200).encode(
    x=alt.X('binned_longitude:Q', scale=alt.Scale(domain=[lon_min, lon_max]),
 ↪  axis=alt.Axis(format=".5f")),
    y=alt.Y('binned_latitude:Q', scale=alt.Scale(domain=[lat_min, lat_max]),
 ↪  axis=alt.Axis(format=".5f")),
    size='alert_count:Q',
    tooltip=['binned_coordinates', 'alert_count']
)
```

```
final_chart = map_chart + scatter_plot

final_chart = final_chart.configure_view(
    strokeWidth=0,
    fill='transparent'
)

final_chart.show()
```

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

5.

a. import os import dash from dash import dcc, html import geopandas as gpd import pandas as pd import plotly.express as px from dash.dependencies import Input, Output import re

Ensure file paths are correct

geojson_path = os.path.abspath(‘top_alerts_map/chicago-boundaries.geojson’) csv_path = os.path.abspath(‘waze_data/waze_data.csv’)

Validate file paths

if not os.path.exists(geojson_path): raise FileNotFoundError(f”GeoJSON file not found at {geojson_path}“) if not os.path.exists(csv_path): raise FileNotFoundError(f”CSV file not found at {csv_path}“)

Load data

geo_df = gpd.read_file(geojson_path) waze_df = pd.read_csv(csv_path)

Function to extract coordinates

def extract_coordinates(geo_str): if pd.notnull(geo_str): match = re.match(r’((.*?), (.*?))’, geo_str) if match: return float(match.group(1)), float(match.group(2)) return None, None

Apply coordinate extraction

waze_df[[‘latitude’, ‘longitude’]] = waze_df[‘geo’].apply(lambda x: pd.Series(extract_coordinates(x)))

Create a crosswalk for type and subtype

unique_combinations = waze_df[[‘type’, ‘subtype’]].drop_duplicates() crosswalk_data = [] for *, row in unique_combinations.iterrows(): updated_type = f”Cleaned*{row[‘type’]}” updated_subtype = row[‘subtype’] if pd.notnull(row[‘subtype’]) else ‘Unclassified’ crosswalk_data.append({ ‘type’: row[‘type’], ‘subtype’: row[‘subtype’], ‘updated_type’: updated_type, ‘updated_subtype’: updated_subtype })

crosswalk_df = pd.DataFrame(crosswalk_data) df_cleaned = waze_df.merge(crosswalk_df, on=[‘type’, ‘subtype’], how=‘left’)

Ensure consistent CRS for GeoJSON and points

geo_df = geo_df.to_crs(“EPSG:4326”) points = gpd.GeoDataFrame(df_cleaned, geometry=gpd.points_from_xy(df_cleaned.longitude, df_cleaned.latitude), crs=“EPSG:4326”)

Spatial join between points and geo boundaries

merged_df = gpd.sjoin(points, geo_df, how=“left”, predicate=“within”)

Add alert_count column if not present

if ‘alert_count’ not in merged_df.columns: merged_df[‘alert_count’] = 1

Combine updated type and subtype

merged_df[‘type_subtype’] = merged_df[‘updated_type’] + ’ - ’ + merged_df[‘updated_subtype’] combinations = merged_df[‘type_subtype’].dropna().unique()

Dash App

app = dash.Dash(**name**) app.layout = html.Div([html.H1(“Alert Data Visualization”), dcc.Dropdown(id=‘alert-dropdown’, options=[{‘label’: comb, ‘value’: comb} for comb in combinations], value=combinations[0] if combinations.size > 0 else None, style={‘width’: ‘50%’}), dcc.Graph(id=‘alert-plot’)])

@app.callback(Output(‘alert-plot’, ‘figure’), Input(‘alert-dropdown’, ‘value’)) def update_plot(selected_combination): if not selected_combination: return px.scatter(title=“No data available.”)

```
filtered_data = merged_df[merged_df['type_subtype'] == selected_combination]

# Filter for valid latitudes and longitudes
filtered_data = filtered_data.dropna(subset=['latitude', 'longitude'])
filtered_data = filtered_data[filtered_data['latitude'].between(-90, 90)]
filtered_data = filtered_data[filtered_data['longitude'].between(-180, 180)]

fig = px.scatter(
    filtered_data,
    x='longitude',
    y='latitude',
    size='alert_count',
    title=f'Alerts for {selected_combination}',
    labels={'latitude': 'Latitude', 'longitude': 'Longitude', 'alert_count':
    'Alert Count'}
)

fig.update_layout(
    title=f'Alerts for {selected_combination}',
    geo=dict(
        scope='usa',
        projection_type='albers usa',
        showland=True,
        landcolor='rgb(255, 255, 255)',
        subunitwidth=1,
        countrywidth=1
    ),
    margin={'r': 0, 't': 40, 'l': 0, 'b': 0}
)

return fig
```

Run the server on a custom port

if **name** == '**main**': app.run_server(debug=True, port=8060)

b. import dash from dash import dcc, html import geopandas as gpd import pandas as pd import plotly.express as px import plotly.graph_objects as go from dash.dependencies import Input, Output import re

geo_df = gpd.read_file('top_alerts_map/chicago-boundaries.geojson') waze_df = pd.read_csv('waze_data/waze_data.csv')

def extract_coordinates(geo_str): if pd.notnull(geo_str): match = re.match(r'((.*?), (.*?))', geo_str) if match: return float(match.group(1)), float(match.group(2)) return None, None

waze_df[['latitude', 'longitude']] = waze_df['geo'].apply(lambda x: pd.Series(extract_coordinates(x)))

unique_combinations = waze_df[['type', 'subtype']].drop_duplicates() crosswalk_data = [] for *, row in unique_combinations.iterrows(): updated_type = f"Cleaned*{row['type']}" updated_subtype = row['subtype'] if pd.notnull(row['subtype']) else 'Unclassified' crosswalk_data.append({ 'type': row['type'], 'subtype': row['subtype'], 'updated_type': updated_type, 'updated_subtype': updated_subtype })

crosswalk_df = pd.DataFrame(crosswalk_data) df_cleaned = waze_df.merge(crosswalk_df, on=['type', 'subtype'], how='left')

geo_df = geo_df.to_crs("EPSG:4326") points = gpd.GeoDataFrame(df_cleaned, geometry=gpd.points_from_xy(df_cleaned.longitude, df_cleaned.latitude), crs="EPSG:4326")

merged_df = gpd.sjoin(points, geo_df, how="left", predicate="within")

if 'alert_count' not in merged_df.columns: merged_df['alert_count'] = 1

merged_df['type_subtype'] = merged_df['updated_type'] + ' - ' + merged_df['updated_subtype'] combinations = merged_df[‘type_subtype’].dropna().unique()

app = dash.Dash(**name**)

app.layout = html.Div([html.H1(“Alert Data Visualization”),

```
dcc.Dropdown(
    id='alert-dropdown',
    options=[{'label': comb, 'value': comb} for comb in combinations],
    value=combinations[0],
    style={'width': '50%'}
),


dcc.Graph(id='alert-map'),

dcc.Graph(id='alert-plot')
```

])

@app.callback([Output(‘alert-map’, ‘figure’), Output(‘alert-plot’, ‘figure’)], Input(‘alert-dropdown’, ‘value’)) def update_plot(selected_combination):

```
filtered_data = merged_df[merged_df['type_subtype'] == selected_combination]

filtered_data = filtered_data.dropna(subset=['latitude', 'longitude'])
filtered_data = filtered_data[filtered_data['latitude'].between(-90, 90)]
filtered_data = filtered_data[filtered_data['longitude'].between(-180, 180)]

scatter_fig = px.scatter(
    filtered_data,
    x='longitude',
    y='latitude',
    size='alert_count',
    title=f'Alerts for {selected_combination}',
    labels={'latitude': 'Latitude', 'longitude': 'Longitude', 'alert_count':
    'Alert Count'}
)

region_alert_count = filtered_data.groupby('geometry').agg({'alert_count':
'sum'}).reset_index()
geojson = geo_df.copy()
```

```
geojson['alert_count'] = geojson.apply(lambda row:
region_alert_count.loc[region_alert_count['geometry'] == row['geometry'],
'alert_count'].values[0] if not
region_alert_count.loc[region_alert_count['geometry'] == row['geometry'],
'alert_count'].empty else 0, axis=1)

map_fig = go.Figure(go.Choroplethmapbox(
    geojson=geojson.geometry.__geo_interface__,
    locations=geojson.index,
    z=geojson['alert_count'],
    colorscale="Viridis",
    colorbar_title="Alert Count",
))

map_fig.update_layout(
    mapbox_style="carto-positron",
    mapbox_zoom=10,
    mapbox_center={"lat": 41.8781, "lon": -87.6298},
    title=f"Alert Density for {selected_combination}"
)

return map_fig, scatter_fig
```

if **name** == ‘**main**’: app.run_server(debug=True)

c. import dash from dash import dcc, html import geopandas as gpd import pandas as pd import plotly.graph_objects as go from dash.dependencies import Input, Output import re

geojson_path = ‘top_alerts_map/chicago-boundaries.geojson’ csv_path = ‘waze_data/waze_data.csv’

geo_df = gpd.read_file(geojson_path) waze_df = pd.read_csv(csv_path)

def extract_coordinates(geo_str): if pd.notnull(geo_str): match = re.match(r’((.?), (.?))’, geo_str) if match: return float(match.group(1)), float(match.group(2)) return None, None

waze_df[[‘latitude’, ‘longitude’]] = waze_df[‘geo’].apply(lambda x: pd.Series(extract_coordinates(x)))

unique_combinations = waze_df[[‘type’, ‘subtype’]].drop_duplicates() crosswalk_data = [{ ‘type’: row[‘type’], ‘subtype’: row[‘subtype’], ‘updated_type’: f”Cleaned_{row[‘type’]}“, ‘updated_subtype’: row[‘subtype’] if pd.notnull(row[‘subtype’]) else ‘Unclassified’ } for _, row in unique_combinations.iterrows()]

crosswalk_df = pd.DataFrame(crosswalk_data) df_cleaned = waze_df.merge(crosswalk_df, on=[‘type’, ‘subtype’], how=‘left’) geo_df = geo_df.to_crs(“EPSG:4326”) points = gpd.GeoDataFrame(df_cleaned, geometry=gpd.points_from_xy(df_cleaned.longitude, df_cleaned.latitude), crs=“EPSG:4326”)

merged_df = gpd.sjoin(points, geo_df, how=“left”, predicate=“within”) merged_df[‘alert_count’] = 1 merged_df[‘type_subtype’] = merged_df[‘updated_type’] + ’ - ’ + merged_df[‘updated_subtype’]

app = dash.Dash(**name**)

app.layout = html.Div([html.H1(“Alert Data Visualization”), dcc.Dropdown(id=‘alert-dropdown’, options=[{‘label’: comb, ‘value’: comb} for comb in merged_df[‘type_subtype’].unique()], value=merged_df[‘type_subtype’].unique()[0], style={‘width’: ‘50%’}), dcc.Graph(id=‘alert-map’), dcc.Graph(id=‘alert-plot’)])

@app.callback([Output(‘alert-map’, ‘figure’), Output(‘alert-plot’, ‘figure’)], Input(‘alert-dropdown’, ‘value’)) def update_plot(selected_combination): filtered_data = merged_df[merged_df[‘type_sub
== selected_combination]

```
# Create Scattermapbox for alert locations
scatter_fig = go.Figure(go.Scattermapbox(
    lat=filtered_data['latitude'],
    lon=filtered_data['longitude'],
    mode='markers',
    marker=dict(
        size=filtered_data['alert_count'],
        color='rgba(255, 0, 0, 0.6)',
        opacity=0.6
    ),
    text=filtered_data['type_subtype']
))

scatter_fig.update_layout(
    mapbox_style="carto-positron",
    mapbox_zoom=10,
    mapbox_center={"lat": 41.8781, "lon": -87.6298},
    title="Alert Locations"
)

# Create Choroplethmapbox for alert density
region_alert_count =
filtered_data.groupby(filtered_data.geometry).agg({'alert_count':
'sum'}).reset_index()
region_alert_count = gpd.GeoDataFrame(region_alert_count,
geometry='geometry', crs="EPSG:4326")
geo_df.set_crs("EPSG:4326", allow_override=True, inplace=True)
```

```
merged_geo_df = gpd.sjoin(geo_df, region_alert_count, how="left",
predicate="intersects")
merged_geo_df['alert_count'] = merged_geo_df['alert_count'].fillna(0)

map_fig = go.Figure(go.Choroplethmapbox(
    geojson=merged_geo_df.geometry.__geo_interface__,
    locations=merged_geo_df.index,
    z=merged_geo_df['alert_count'],
    colorscale="Viridis",
    colorbar_title="Alert Count"
))

map_fig.update_layout(
    mapbox_style="carto-positron",
    mapbox_zoom=10,
    mapbox_center={"lat": 41.8781, "lon": -87.6298},
    title="Alert Density for Selected Type-Subtype"
)

return map_fig, scatter_fig
```

if **name** == ‘**main**’: app.run_server(debug=True, port=8060)

d.

```
filtered_data = merged_df[
    (merged_df['type'] == 'Traffic') &
    (merged_df['subtype'].isin(['Accident', 'Congestion']))  # Adjust
    subtypes as needed
]
```

The highest number of traffic-related alerts in Chicago are concentrated around downtown and major intersections like State Street and Lake Shore Drive, which shows frequent accidents and congestion, as indicated by the red markers on the map

e.

```
merged_df['timestamp'] = pd.to_datetime(merged_df['timestamp'])

merged_df['hour_of_day'] = merged_df['timestamp'].dt.hour
merged_df['day_of_week'] = merged_df['timestamp'].dt.day_name()
```

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

1.

a. The ts (timestamp) column represents the date and time of each alert.
Collapsing the dataset by this column could make sense depending on the
analysis you want to perform:

When it would be a good idea: If you're interested in analyzing alerts on a
daily or hourly basis, collapsing the dataset by ts would simplify it and
allow you to group and aggregate alerts by time. This could be useful if
you're analyzing trends over specific time periods, such as the number of
alerts per day, or understanding how alerts change during different hours of
the day.

When it might not be a good idea: If the goal is to retain the granular
detail of each individual alert, such as its exact timestamp, location, and
type, then collapsing by ts could lose important information. In this case,
you may prefer to retain the individual timestamps and use grouping or
filtering during the analysis.


b.

::: {.cell execution_count=24}
``` {.python .cell-code}
import os
import pandas as pd
import re

waze_df = pd.read_csv('waze_data/waze_data.csv')

print(waze_df.columns)

def extract_coordinates(geo_str):
    if pd.notnull(geo_str):
        match = re.match(r'\((.*?), (.*?)\)', geo_str)
        if match:
            return float(match.group(1)), float(match.group(2))
    return None, None

waze_df[['latitude', 'longitude']] = waze_df['geo'].apply(lambda x:
pd.Series(extract_coordinates(x)))
~~~

```
waze_df['ts'] = pd.to_datetime(waze_df['ts'])

waze_df['hour'] = waze_df['ts'].dt.floor('H')

collapsed_df = waze_df.groupby(['hour', 'latitude', 'longitude', 'type',
'subtype']).size().reset_index(name='alert_count')

output_dir = 'top_alerts_map_byhour'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

collapsed_df.to_csv(os.path.join(output_dir, 'top_alerts_map_byhour.csv'),
index=False)

print(f"The collapsed dataset has {collapsed_df.shape[0]} rows.")
```

```
Index(['city', 'confidence', 'nThumbsUp', 'street', 'uuid', 'country',
'type',
       'subtype', 'roadType', 'reliability', 'magvar', 'reportRating', 'ts',
       'geo', 'geoWKT'],
      dtype='object')
The collapsed dataset has 0 rows.
```

```
C:\Users\Yunzh\AppData\Local\Temp\ipykernel_3220\883916095.py:20:
FutureWarning:

'H' is deprecated and will be removed in a future version, please use 'h'
instead.
```

:::

c.

import pandas as pd

waze_df = pd.read_csv(‘waze_data/waze_data.csv’)

print(waze_df.columns)

collapsed_df = waze_df.groupby([‘hour’, ‘latitude’, ‘longitude’, ‘type’, ‘subtype’]).size().reset_index(name=‘ale

d.

import pandas as pd import plotly.express as px

waze_df = pd.read_csv(‘waze_data/waze_data.csv’)

print(waze_df.columns) # This will print all column names to help identify the correct ones

waze_df[‘ts’] = pd.to_datetime(waze_df[‘ts’])

waze_df[‘hour’] = waze_df[‘ts’].dt.floor(‘H’) # ‘floor’ rounds down to the hour

collapsed_df = waze_df.groupby([‘hour’, ‘latitude’, ‘longitude’, ‘type’, ‘subtype’]).size().reset_index(name=‘ale

heavy_traffic_df = collapsed_df[collapsed_df[‘type’] == ‘Jam - Heavy Traffic’]

hour = ‘2024-11-01 08:00:00’ # Example hour, change this as needed hour_df = heavy_traffic_df[heavy_traffic_df[‘hour’] == hour]

top_10_df = hour_df.nlargest(10, ‘alert_count’)

fig = px.scatter_mapbox(top_10_df, lat=‘latitude’, lon=‘longitude’, size=‘alert_count’, color=‘alert_count’, color_continuous_scale=‘Viridis’, title=f”Top 10 Locations for ‘Jam - Heavy Traffic’ at {hour}“, mapbox_style=”carto-positron”)

fig.update_layout(mapbox_center={“lat”: 41.8781, “lon”: -87.6298}, # Chicago lat/lon mapbox_zoom=10) fig.show()

2.

a. import dash from dash import dcc, html import pandas as pd

app = dash.Dash(**name**)

app.layout = html.Div([

```
dcc.Dropdown(
    id='alert-dropdown',
    options=[
        {'label': 'Jam - Heavy Traffic', 'value': 'Jam - Heavy Traffic'},

    ],
    value='Jam - Heavy Traffic',
    multi=False
),

dcc.RangeSlider(
    id='hour-slider',
    min=0,
    max=23,
    step=1,
    marks={i: f'{i} AM' if i < 12 else f'{i - 12} PM' for i in range(0, 24)},
    value=[6, 9]
),


dcc.Graph(id='alert-plot')
```

])

if **name** == '**main**': app.run_server(debug=True)

b. import plotly.express as px

@app.callback(dash.dependencies.Output('alert-plot', 'figure'), [dash.dependencies.Input('alert-dropdown', 'value'), dash.dependencies.Input('hour-slider', 'value')]) def update_plot(selected_alert, hour_range):

```
filtered_df = collapsed_df[(collapsed_df['type'] == selected_alert) &
                           (collapsed_df['hour'] >= hour_range[0]) &
                           (collapsed_df['hour'] <= hour_range[1])]

top_10_df = filtered_df.groupby(['latitude',
'longitude']).size().reset_index(name='alert_count')
top_10_df = top_10_df.nlargest(10, 'alert_count')

fig = px.scatter_geo(
    top_10_df,
    lat='latitude',
    lon='longitude',
    size='alert_count',
    title=f'Top 10 Locations for {selected_alert} between {hour_range[0]} AM
    and {hour_range[1]} AM',
    projection="natural earth"
)
return fig
```

c. If night hours have more alerts near known construction zones, the construction happens more at night.

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

1.

a. Collapsing the dataset by a range of hours (e.g., 6 AM - 9 AM) might not be the best approach because:

Loss of granularity: If you collapse the data by range of hours, you lose the ability to distinguish alerts for individual hours within the range. For instance, if there is a sharp peak at 6 AM that isn't present at 9 AM, collapsing this into a range would obscure this trend. Flexibility in the app: Since the goal is to allow users to explore specific hour ranges interactively, it would be better to keep the data granular (by hour) so that the app can filter and display the relevant subset dynamically based on the user's selected range. Thus, it is better to keep the data collapsed by individual hours and filter it in real-time based on the selected range in the Shiny app.

b. import pandas as pd import plotly.express as px

print(waze_df.columns) # Check the column names

waze_df['ts'] = pd.to_datetime(waze_df['ts'])

waze_df['hour'] = waze_df['ts'].dt.hour

heavy_traffic_df = waze_df[(waze_df['type'] == 'Jam - Heavy Traffic') & (waze_df['hour'] >= 6) & (waze_df['hour'] <= 9)]

print(heavy_traffic_df.columns) # Check column names in the filtered DataFrame

collapsed_df = heavy_traffic_df.groupby(['hour', 'latitude', 'longitude', 'type', 'subtype']).size().reset_index(name='alert_count')

top_10_df = collapsed_df.nlargest(10, 'alert_count')

fig = px.scatter_mapbox(top_10_df, lat='latitude', lon='longitude', size='alert_count', color='alert_count', color_continuous_scale='Viridis', title="Top 10 Locations for 'Jam - Heavy Traffic' between 6AM and 9AM", mapbox_style="carto-positron")

fig.update_layout(mapbox_center={"lat": 41.8781, "lon": -87.6298}, # Chicago lat/lon mapbox_zoom=10)

fig.show()

1.

a. import dash import dash_core_components as dcc import dash_html_components as html from dash.dependencies import Input, Output

app = dash.Dash(**name**)

app.layout = html.Div([html.H1('Traffic Alerts Analysis'),

```
dcc.Dropdown(
    id='alert-type-dropdown',
    options=[
        {'label': 'Jam - Heavy Traffic', 'value': 'Jam - Heavy Traffic'}
    ],
    value='Jam - Heavy Traffic',
    multi=False
),

dcc.RangeSlider(
    id='hour-range-slider',
    min=0,
    max=23,
    step=1,
    marks={i: f'{i} AM' if i < 12 else f'{i - 12} PM' for i in range(24)},
    value=[6, 9]
),

dcc.Graph(id='alert-plot')
```

])

if **name** == ‘**main**’: app.run_server(debug=True)

b. import pandas as pd import plotly.express as px data = { ‘hour’: [6, 7, 8, 9, 10, 11], ‘latitude’: [41.8781, 41.8790, 41.8800, 41.8810, 41.8820, 41.8830], ‘longitude’: [-87.6298, -87.6300, -87.6310, -87.6320, -87.6330, -87.6340], ‘type’: [‘Jam - Heavy Traffic’]*6, ‘subtype’: [‘Construction’]*6, ‘alert_count’: [100, 150, 200, 250, 300, 350] } df = pd.DataFrame(data)

app = dash.Dash(**name**)

app.layout = html.Div([html.H1(‘Traffic Alerts Analysis’),

```
dcc.Dropdown(
    id='alert-type-dropdown',
    options=[
        {'label': 'Jam - Heavy Traffic', 'value': 'Jam - Heavy Traffic'}
    ],
    value='Jam - Heavy Traffic',
    multi=False
),

dcc.RangeSlider(
    id='hour-range-slider',
    min=0,
    max=23,
    step=1,
    marks={i: f'{i} AM' if i < 12 else f'{i - 12} PM' for i in range(24)},
    value=[6, 9]
),

dcc.Graph(id='alert-plot')
```

])

@app.callback(Output(‘alert-plot’, ‘figure’), [Input(‘alert-type-dropdown’, ‘value’), Input(‘hour-range-slider’, ‘value’)]) def update_plot(alert_type, hour_range):

```
filtered_df = df[(df['hour'] >= hour_range[0]) & (df['hour'] <=
hour_range[1])]

fig = px.scatter(filtered_df, x='longitude', y='latitude',
                 color='alert_count', size='alert_count',
                 hover_name='subtype', title=f"Traffic Alerts
                 ({alert_type})")

return fig
```

if **name** == ‘**main**’: app.run_server(debug=True)

3.

a. import dash import dash_core_components as dcc import dash_html_components as html from dash.dependencies import Input, Output

app = dash.Dash(**name**)

app.layout = html.Div([html.H1(‘Traffic Alerts Analysis’),

```
dcc.Dropdown(
    id='alert-type-dropdown',
    options=[
        {'label': 'Jam - Heavy Traffic', 'value': 'Jam - Heavy Traffic'}
    ],
    value='Jam - Heavy Traffic',
    multi=False
),

dcc.RadioItems(
    id='hour-toggle-switch',
    options=[
        {'label': 'Single Hour', 'value': 'single'},
        {'label': 'Range of Hours', 'value': 'range'}
    ],
    value='range',
    labelStyle={'display': 'inline-block'}
),

html.Div([
    dcc.Slider(
        id='hour-slider',
        min=0,
        max=23,
        step=1,
        marks={i: f'{i} AM' if i < 12 else f'{i - 12} PM' for i in
        range(24)},
        value=6
    )
], id='single-hour-slider', style={'display': 'none'}),

html.Div([
    dcc.RangeSlider(
        id='hour-range-slider',
        min=0,
        max=23,
        step=1,
        marks={i: f'{i} AM' if i < 12 else f'{i - 12} PM' for i in
        range(24)},
        value=[6, 9]
    )
], id='hour-range-slider-container', style={'display': 'block'}),


dcc.Graph(id='alert-plot')
```

])

@app.callback([Output('single-hour-slider', 'style'), Output('hour-range-slider-container', 'style')], [Input('hour-toggle-switch', 'value')]) def toggle_slider(value): if value == 'range':
return {'display': 'none'}, {'display': 'block'} else:
return {'display': 'block'}, {'display': 'none'}

if **name** == '**main**': app.run_server(debug=True)

b. @app.callback([Output(‘single-hour-slider’, ‘style’), Output(‘hour-range-slider-container’, ‘style’)], [Input(‘hour-toggle-switch’, ‘value’)]) def toggle_slider(value): if value: return {‘display’: ‘none’}, {‘display’: ‘block’} else: return {‘display’: ‘block’}, {‘display’: ‘none’}

c. import dash import dash_core_components as dcc import dash_html_components as html from dash.dependencies import Input, Output

app = dash.Dash(**name**)

app.layout = html.Div([html.H1(‘Traffic Alerts Analysis’),

```
dcc.Dropdown(
    id='alert-type-dropdown',
    options=[
        {'label': 'Jam - Heavy Traffic', 'value': 'Jam - Heavy Traffic'}
    ],
    value='Jam - Heavy Traffic',
    multi=False
),

dcc.RadioItems(
    id='hour-toggle-switch',
    options=[
        {'label': 'Single Hour', 'value': 'single'},
        {'label': 'Range of Hours', 'value': 'range'}
    ],
    value='range',
    labelStyle={'display': 'inline-block'}
),

html.Div([
    dcc.Slider(
        id='hour-slider',
        min=0,
        max=23,
        step=1,
        marks={i: f'{i} AM' if i < 12 else f'{i - 12} PM' for i in
        range(24)},
        value=6
    )
], id='single-hour-slider', style={'display': 'none'}),

html.Div([
    dcc.RangeSlider(
        id='hour-range-slider',
        min=0,
        max=23,
        step=1,
        marks={i: f'{i} AM' if i < 12 else f'{i - 12} PM' for i in
        range(24)},
        value=[6, 9]
    )
], id='hour-range-slider-container', style={'display': 'block'}),

dcc.Graph(id='alert-plot')
```

])

@app.callback([Output(‘single-hour-slider’, ‘style’), Output(‘hour-range-slider-container’, ‘style’)], [Input(‘hour-toggle-switch’, ‘value’)]) def toggle_slider(value): if value == ‘range’: return {‘display’: ‘none’}, {‘display’: ‘block’} else:
return {‘display’: ‘block’}, {‘display’: ‘none’}

if **name** == ‘**main**’: app.run_server(debug=True)

d. To achieve a plot similar to the one you shared, you might need to implement the following changes in the app or code generating the visualization:

Geospatial Data Integration: Ensure that you have access to a shapefile or boundary data for the map (in this case, the Chicago area) to provide the base map.

Visualization Layer Updates: Use a scatter plot overlaid on a base map, where each point represents an alert. Add concentric circles to reflect the number of alerts in specific locations.

Time Period Differentiation: Use distinct colors for different time periods (e.g., red for “Morning” and blue for “Afternoon”). Add a legend to indicate the corresponding time periods.

Point Size Mapping: Map the size of each point or circle to the number of alerts to visually represent alert density in specific locations. Use a logarithmic scale for point sizes if the range of values is large.

Custom Labels and Legends: Include a clear legend for the number of alerts (circle sizes) and time periods (colors). Label axes with appropriate longitude and latitude.

Enhance Map Clarity: Simplify the base map by including only the necessary boundaries, omitting excess details that may clutter the visualization.

Coordinate System: Ensure the data uses the correct coordinate reference system to align with the map background.