Project

December 11, 2024

1 IST652 Project Deliverable 2

1.1 Phase 2: Project Progress

In this step you should have a road map of the steps you will be taking to complete your analysis. In addition, at this stage you should also complete the following - fine tune your research questions. - upload your dataset into jupyterhub and conduct some preliminary cleaning and transformation. - provide coding activities conducted so far. - have a better sense of team members responsibilities. - set a schedule to meet

Team Member:
Mengqi Li
Yunkai Yao
Tianyuan Sheng

1.1.1 Step 1: What is Your Idea and Research Questions, Again?

Please reiterate your project idea below (you can copy it from the project proposal if there were no changes).

`--== Double-click and put the title team members and brief description of your project below ==--`

Our project focuses on examining driver crash data from Montgomery County, Maryland, to find patterns and insights that might help make roads safer. In this project, we will analyze the "Crash Reporting - Drivers Data" dataset (https://catalog.data.gov/dataset/crash-reporting-drivers-data) from Montgomery County from the U.S. Government's open data portal (https://data.gov), which contains information about traffic collisions on both county and local roadways. Our focus will be on examining different aspects of driver behaviors, the causes of crashes, and the environmental conditions linked to these incidents. This dataset comes from the Maryland State Police's Automated Crash Reporting System (ACRS) and provides trustworthy information on every collision, as recorded by the Montgomery County Police and other local law enforcement agencies. Through examining this data, we aim to identify possible risk factors and offer recommendations based on the findings to help decrease the number of crashes.

We will look at driver demographics, vehicle types, weather conditions, and crash timings—important factors for understanding how traffic collisions happen. The insights from our findings will be really useful for policymakers, transportation agencies, and public safety organizations to create targeted interventions that tackle specific risk factors. We aim to use visualizations and statistical analyses to show important trends, outliers, and correlations in the dataset, which can help in creating effective traffic safety policies.

Preliminary Research Questions

1. How do environmental conditions impact traffic crashes?
   Objective: Examine the role of environmental factors like weather and road surface conditions in contributing to traffic crashes. Insights can inform strategies to improve road safety.

2. What is the influence of driver behavior on crash outcomes?
   Objective: Analyze how driver actions, such as distraction, speeding, and impairment, affect the frequency and severity of crashes. Findings can guide safety programs and policies.

3. What factors contribute to severe traffic crashes?
   Objective: Identify elements such as road conditions, vehicle characteristics, and driver actions that are associated with severe crashes. Results can inform targeted interventions to reduce severe incidents.

4. How do vehicle features relate to crash outcomes?
   Objective: Assess how attributes like vehicle make, model, year, and movement during a crash influence the extent of damage and outcomes. Insights can guide safety enhancements and risk assessments.

5. What patterns exist in traffic crashes over time?
   Objective: Analyze temporal trends in crash occurrences, including variations by time of day, week, or year, to identify high-risk periods and support resource allocation for safety measures.

6. Where are traffic crashes most likely to occur?
   Objective: Explore geographic patterns of crashes and their association with factors like road types and speed limits. Findings can help identify high-risk areas and prioritize safety improvements.

1.1.2 Step 2: Problem Analysis - Roadmap

What are the preliminary major steps you will be completing? Include the research question and steps taken to answer that question? Are there any unique functions you will be incorporating which we have not covered in the classroom? Describe below.

`--== Double-click and describe steps towards analysis of your project below ==--`

Data Loading and Exploration

- **Objective:** Understand the structure and content of the dataset and identify variables relevant to the research questions.
- **Actions:**
  1. Load the dataset.
  2. Inspect the structure of the dataset:
     – Check the shape (number of rows and columns).
     – Display column names and data types.
     – Preview the first few rows of the data.
  3. Identify key variables related to:
     – Environmental factors: weather conditions, road surface.
     – Driver behavior: speeding, distraction, impairment.
     – Vehicle attributes: make, model, year, and movement.

    - Crash outcomes: severity levels.
4. Check for missing values, inconsistent data, and data distribution.

Data Cleaning

- **Objective:** Handle missing and inconsistent data to prepare the dataset for analysis.
- **Actions:**
  1. Impute missing values:
     - For numerical variables: use mean or median.
     - For categorical variables: use mode or "Unknown."
  2. Remove irrelevant rows or columns with excessive missing data.
  3. Standardize and normalize variables:
     - Ensure consistent formats for date and time columns.
     - Normalize numerical variables where applicable for consistent scaling.
  4. Standardize categorical labels for uniformity.

Data Transformation

- **Objective:** Generate new features and preprocess data for analysis.
- **Actions:**
  1. Create new temporal features:
     - Extract time of day, day of the week, and month from timestamp columns.
  2. Categorize environmental conditions into simplified groups (e.g., "Rainy," "Clear").
  3. Encode categorical variables:
     - Use one-hot encoding for nominal variables (e.g., weather conditions).
     - Use label encoding for ordinal variables (e.g., crash severity levels).
  4. Create binary variables:
     - Flags for driver behavior, such as `Distraction_Flag` or `Speeding_Flag`.

Research Question-Specific Analysis

Q1: How do environmental conditions impact traffic crashes?

- **Steps:**
  1. Analyze the distribution of crashes under different weather and road surface conditions.
  2. Use frequency tables to observe the number of crashes for each condition.
  3. Create visualizations (e.g., bar charts or heatmaps) to highlight trends.
  4. Compare the severity of crashes (e.g., minor vs. severe) for specific environmental conditions to identify associations.

Q2: What is the influence of driver behavior on crash outcomes?

- **Steps:**
  1. Categorize driver behaviors such as distraction, impairment, and speeding.
  2. Create binary or grouped flags to identify crashes involving specific behaviors.
  3. Use frequency analysis to compare the occurrence of severe crashes across different driver behaviors.
  4. Visualize crash severity by driver behavior using stacked bar charts or heatmaps.

Q3: What factors contribute to severe traffic crashes?

- **Steps:**
  1. Examine factors such as road surface condition, weather, and driver behavior for their association with severe crashes.
  2. Use contingency tables to display the relationship between crash severity and each factor.
  3. Visualize these relationships using grouped bar charts or summary tables.
  4. Compare distributions of crash severity across different levels of categorical variables (e.g., "Wet Road" vs. "Dry Road").

Q4: How do vehicle features relate to crash outcomes?

- **Steps:**
  1. Focus on attributes like vehicle make, model, year, and condition during crashes.
  2. Analyze distributions of crash severity for specific vehicle features.
  3. Use visualizations (e.g., boxplots) to compare numerical attributes (e.g., vehicle year) across severity levels.
  4. Identify any notable trends (e.g., older vehicles are more prone to severe crashes).

Q5: What patterns exist in traffic crashes over time?

- **Steps:**
  1. Extract temporal features such as time of day, day of the week, and month from timestamp data.
  2. Generate frequency distributions to identify high-risk times (e.g., morning rush hour, weekends).
  3. Visualize temporal trends using histograms, line charts, or bar plots.
  4. Compare crash severity across different time intervals to identify high-risk periods.

Q6: Where are traffic crashes most likely to occur?

- **Steps:**
  1. Use geographic coordinates to plot crash locations on a map.
  2. Generate heatmaps or choropleth maps to visualize crash density.
  3. Summarize crash counts by geographic regions (e.g., intersections, highways).
  4. Highlight high-risk areas using spatial patterns or density analysis.

Visualization and Reporting

- **Objective:** Present findings effectively.
- **Actions:**
  1. Create visualizations for each research question:
     - Use bar charts, scatter plots, heatmaps, or geographic maps.
  2. Summarize insights for each research question.
  3. Document the final analysis in a clear and concise report or presentation.

Unique Functions or Techniques to Be Incorporated

1. **Geospatial Analysis:**
   - Use `geopandas` and `folium` for mapping crash locations and visualizing hotspots.

   - Perform spatial clustering to identify crash-prone zones.
2. **Feature Engineering:**
   - Create new variables, such as temporal features and binary behavior flags.
   - Group environmental conditions into simplified categories.

1.1.3 Step 3: Preliminary Code

Include coding that has been completed at this preliminary stage.

Read the data

[44]:
```
pip install geopandas
```

```
Collecting geopandas
  Downloading geopandas-1.0.1-py3-none-any.whl.metadata (2.2 kB)
Requirement already satisfied: numpy>=1.22 in /opt/conda/lib/python3.11/site-
packages (from geopandas) (1.26.3)
Collecting pyogrio>=0.7.2 (from geopandas)
  Downloading pyogrio-0.10.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (5.5
kB)
Requirement already satisfied: packaging in /opt/conda/lib/python3.11/site-
packages (from geopandas) (23.2)
Requirement already satisfied: pandas>=1.4.0 in /opt/conda/lib/python3.11/site-
packages (from geopandas) (2.1.4)
Collecting pyproj>=3.3.0 (from geopandas)
  Downloading
pyproj-3.7.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(31 kB)
Collecting shapely>=2.0.0 (from geopandas)
  Downloading shapely-2.0.6-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (7.0 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.11/site-packages (from pandas>=1.4.0->geopandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-
packages (from pandas>=1.4.0->geopandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.11/site-
packages (from pandas>=1.4.0->geopandas) (2023.4)
Requirement already satisfied: certifi in /opt/conda/lib/python3.11/site-
packages (from pyogrio>=0.7.2->geopandas) (2023.11.17)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-
packages (from python-dateutil>=2.8.2->pandas>=1.4.0->geopandas) (1.16.0)
Downloading geopandas-1.0.1-py3-none-any.whl (323 kB)
                         323.6/323.6 kB
14.2 MB/s eta 0:00:00
Downloading pyogrio-0.10.0-cp311-cp311-manylinux_2_28_x86_64.whl (24.1 MB)
                         24.1/24.1 MB
85.6 MB/s eta 0:00:00:00:0100:01
Downloading
pyproj-3.7.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (9.5 MB)
```

```
                         9.5/9.5 MB
98.7 MB/s eta 0:00:00:00:0100:01
Downloading
shapely-2.0.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.5
MB)
                         2.5/2.5 MB
102.5 MB/s eta 0:00:00
Installing collected packages: shapely, pyproj, pyogrio, geopandas
Successfully installed geopandas-1.0.1 pyogrio-0.10.0 pyproj-3.7.0 shapely-2.0.6
Note: you may need to restart the kernel to use updated packages.
```

[45]:
```
pip install contextily
```

```
Collecting contextily
  Downloading contextily-1.6.2-py3-none-any.whl.metadata (2.9 kB)
Collecting geopy (from contextily)
  Downloading geopy-2.4.1-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: matplotlib in
/home/jovyan/.local/lib/python3.11/site-packages (from contextily) (3.8.2)
Collecting mercantile (from contextily)
  Downloading mercantile-1.2.1-py3-none-any.whl.metadata (4.8 kB)
Requirement already satisfied: pillow in /opt/conda/lib/python3.11/site-packages
(from contextily) (10.2.0)
Collecting rasterio (from contextily)
  Downloading rasterio-1.4.3-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.1 kB)
Requirement already satisfied: requests in /opt/conda/lib/python3.11/site-
packages (from contextily) (2.31.0)
Collecting joblib (from contextily)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: xyzservices in
/home/jovyan/.local/lib/python3.11/site-packages (from contextily) (2023.10.1)
Collecting geographiclib<3,>=1.52 (from geopy->contextily)
  Downloading geographiclib-2.0-py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: contourpy>=1.0.1 in
/home/jovyan/.local/lib/python3.11/site-packages (from matplotlib->contextily)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in
/home/jovyan/.local/lib/python3.11/site-packages (from matplotlib->contextily)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/home/jovyan/.local/lib/python3.11/site-packages (from matplotlib->contextily)
(4.47.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/jovyan/.local/lib/python3.11/site-packages (from matplotlib->contextily)
(1.4.5)
Requirement already satisfied: numpy<2,>=1.21 in /opt/conda/lib/python3.11/site-
packages (from matplotlib->contextily) (1.26.3)
```

```
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.11/site-packages (from matplotlib->contextily) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/jovyan/.local/lib/python3.11/site-packages (from matplotlib->contextily)
(3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.11/site-packages (from matplotlib->contextily) (2.8.2)
Requirement already satisfied: click>=3.0 in /opt/conda/lib/python3.11/site-
packages (from mercantile->contextily) (8.1.7)
Collecting affine (from rasterio->contextily)
  Downloading affine-2.4.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: attrs in /opt/conda/lib/python3.11/site-packages
(from rasterio->contextily) (23.1.0)
Requirement already satisfied: certifi in /opt/conda/lib/python3.11/site-
packages (from rasterio->contextily) (2023.11.17)
Collecting cligj>=0.5 (from rasterio->contextily)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Collecting click-plugins (from rasterio->contextily)
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl.metadata (6.4 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in
/opt/conda/lib/python3.11/site-packages (from requests->contextily) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.11/site-
packages (from requests->contextily) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.11/site-packages (from requests->contextily) (2.1.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-
packages (from python-dateutil>=2.7->matplotlib->contextily) (1.16.0)
Downloading contextily-1.6.2-py3-none-any.whl (17 kB)
Downloading geopy-2.4.1-py3-none-any.whl (125 kB)
                         125.4/125.4 kB
6.5 MB/s eta 0:00:00
Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
                         301.8/301.8 kB
24.4 MB/s eta 0:00:00
Downloading mercantile-1.2.1-py3-none-any.whl (14 kB)
Downloading
rasterio-1.4.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (22.2
MB)
                         22.2/22.2 MB
92.7 MB/s eta 0:00:00:00:0100:01
Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Downloading geographiclib-2.0-py3-none-any.whl (40 kB)
                         40.3/40.3 kB
8.4 MB/s eta 0:00:00
Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Installing collected packages: mercantile, joblib, geographiclib, cligj, click-
plugins, affine, rasterio, geopy, contextily
```

```
Successfully installed affine-2.4.0 click-plugins-1.1.1 cligj-0.7.2
contextily-1.6.2 geographiclib-2.0 geopy-2.4.1 joblib-1.4.2 mercantile-1.2.1
rasterio-1.4.3
Note: you may need to restart the kernel to use updated packages.
```

[48]:
```
pip install scipy
```

```
Collecting scipy
  Downloading
scipy-1.14.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(60 kB)
                            60.8/60.8 kB
4.3 MB/s eta 0:00:00
Requirement already satisfied: numpy<2.3,>=1.23.5 in
/opt/conda/lib/python3.11/site-packages (from scipy) (1.26.3)
Downloading
scipy-1.14.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (41.2
MB)
                          41.2/41.2 MB
65.6 MB/s eta 0:00:00:00:0100:01
Installing collected packages: scipy
Successfully installed scipy-1.14.1
Note: you may need to restart the kernel to use updated packages.
```

[1]:
```python
# Step 3: Write code here. Add additional cells as necessary.
import pandas as pd

crash_data = pd.read_csv("Crash_Reporting_-_Drivers_Data.csv", low_memory=False)
```

[2]:
```python
# Display the first few rows of the dataset
crash_data.head()
```

[2]:
```
  Report Number Local Case Number                Agency Name  \
0    DM8479000T         210020119  Takoma Park Police Depart
1   MCP2970000R          15045937                 MONTGOMERY
2   MCP20160036         180040948   Montgomery County Police
3    EJ7879003C         230048975  Gaithersburg Police Depar
4   MCP2967004Y         230070277   Montgomery County Police

        ACRS Report Type         Crash Date/Time         Route Type  \
0  Property Damage Crash  05/27/2021 07:40:00 PM                NaN
1  Property Damage Crash  09/11/2015 01:29:00 PM                NaN
2  Property Damage Crash  08/17/2018 02:25:00 PM                NaN
3           Injury Crash  08/11/2023 06:00:00 PM                NaN
4  Property Damage Crash  12/06/2023 06:42:00 PM   Maryland (State)

         Road Name Cross-Street Name               Off-Road Description  \
```

```
0              NaN              NaN                     IN PARKING LOT
1              NaN              NaN    Parking Lot: \n2525 Ennalls Ave
2              NaN              NaN   PARKING LOT OF 16246 FREDERICK RD
3              NaN              NaN                     1 N SUMMIT DRIVE
4  CONNECTICUT AVE      BALTIMORE ST                                 NaN

  Municipality  … Vehicle Going Dir Speed Limit Driverless Vehicle  \
0          NaN  …               NaN         0.0                 No
1          NaN  …             South         5.0                 No
2          NaN  …              West        15.0                 No
3          NaN  …           Unknown        15.0                 No
4   KENSINGTON  …             South        35.0                 No

  Parked Vehicle Vehicle Year Vehicle Make Vehicle Model   Latitude  \
0            Yes       2017.0         HINO           TWK  38.987657
1             No       2012.0       TOYOTA            SU  39.039917
2             No       2015.0         MAZD            TK  38.743373
3             No       2018.0          RAM            TK  39.145873
4             No       2017.0         AUDI            A3  39.025170

   Longitude                     Location
0 -76.987545    (38.98765667, -76.987545)
1 -77.053649  (39.03991652, -77.05364898)
2 -77.546997    (38.743373, -77.54699707)
3 -77.191940  (39.14587303, -77.19194047)
4 -77.076333  (39.02517017, -77.07633333)

[5 rows x 39 columns]
```

[3]:
```python
# Display basic information about the dataset
crash_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 137564 entries, 0 to 137563
Data columns (total 39 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   Report Number                  137564 non-null  object
 1   Local Case Number              137564 non-null  object
 2   Agency Name                    137564 non-null  object
 3   ACRS Report Type               137564 non-null  object
 4   Crash Date/Time                137564 non-null  object
 5   Route Type                     123931 non-null  object
 6   Road Name                      124758 non-null  object
 7   Cross-Street Name              124747 non-null  object
 8   Off-Road Description           12804 non-null   object
 9   Municipality                   15236 non-null   object
```

```
 10  Related Non-Motorist           4367 non-null    object
 11  Collision Type                 137118 non-null  object
 12  Weather                        126847 non-null  object
 13  Surface Condition              121423 non-null  object
 14  Light                          136394 non-null  object
 15  Traffic Control                117158 non-null  object
 16  Driver Substance Abuse         112604 non-null  object
 17  Non-Motorist Substance Abuse   3468 non-null    object
 18  Person ID                      137563 non-null  object
 19  Driver At Fault                137563 non-null  object
 20  Injury Severity                137563 non-null  object
 21  Circumstance                   25072 non-null   object
 22  Driver Distracted By           137563 non-null  object
 23  Drivers License State          129539 non-null  object
 24  Vehicle ID                     137563 non-null  object
 25  Vehicle Damage Extent          137311 non-null  object
 26  Vehicle First Impact Location  137443 non-null  object
 27  Vehicle Body Type              135442 non-null  object
 28  Vehicle Movement               137259 non-null  object
 29  Vehicle Going Dir              135405 non-null  object
 30  Speed Limit                    137563 non-null  float64
 31  Driverless Vehicle             137563 non-null  object
 32  Parked Vehicle                 137563 non-null  object
 33  Vehicle Year                   137563 non-null  float64
 34  Vehicle Make                   137543 non-null  object
 35  Vehicle Model                  137509 non-null  object
 36  Latitude                       137563 non-null  float64
 37  Longitude                      137563 non-null  float64
 38  Location                       137563 non-null  object
dtypes: float64(4), object(35)
memory usage: 40.9+ MB
```

1.1.4 Dataset Overview

- The DataFrame contains **187,311 rows (entries)** and **39 columns** in total, representing various attributes of traffic crashes.

- Most of the columns have non-null entries, but several columns have significant missing values.
- Columns like `Municipality` (19,126 non-null), `Off-Road Description` (17,292 non-null), and `Related Non-Motorist` (6,009 non-null) have a substantial number of missing values.
- The `Vehicle Year` column has valid values but includes some erroneous entries, such as `0` and `9999`.

Data Types

- **Object**: Found in most columns, typically representing categorical or textual data (e.g., `Agency Name`, `Collision Type`, `Vehicle Make`).

- **Int64**: Used for numerical columns such as `Speed Limit` and `Vehicle Year`.
- **Float64**: Present in geographical data (`Latitude` and `Longitude`).
- **Datetime**: The `Crash Date/Time` column can be parsed for temporal analysis.

Key Features Categorical Variables - `ACRS Report Type`: Includes values like "Property Damage Crash," with "Property Damage Crash" being the most frequent type. - `Vehicle Make`: Contains 1,933 unique values, with "TOYOTA" as the most common vehicle make.

Numerical Variables - `Speed Limit`: Ranges from 0 to 75 mph, with a mean of 32.4 mph. This could indicate different crash locations (residential, highways). - `Vehicle Year`: Data spans a wide range but includes unrealistic entries (e.g., year `0`).

Location Data - The dataset includes latitude and longitude coordinates for geospatial analysis, with recurring crash hotspots.

Doing Data Cleaning and Preparation

[4]:
```python
# Check for missing values
print(crash_data.isnull().sum())
```

```
Report Number                      0
Local Case Number                  0
Agency Name                        0
ACRS Report Type                   0
Crash Date/Time                    0
Route Type                     13633
Road Name                      12806
Cross-Street Name              12817
Off-Road Description          124760
Municipality                  122328
Related Non-Motorist          133197
Collision Type                   446
Weather                        10717
Surface Condition              16141
Light                           1170
Traffic Control                20406
Driver Substance Abuse         24960
Non-Motorist Substance Abuse  134096
Person ID                          1
Driver At Fault                    1
Injury Severity                    1
Circumstance                  112492
Driver Distracted By               1
Drivers License State           8025
Vehicle ID                         1
Vehicle Damage Extent            253
Vehicle First Impact Location    121
Vehicle Body Type               2122
Vehicle Movement                 305
```

```
Vehicle Going Dir                    2159
Speed Limit                             1
Driverless Vehicle                      1
Parked Vehicle                          1
Vehicle Year                            1
Vehicle Make                           21
Vehicle Model                          55
Latitude                                1
Longitude                               1
Location                                1
dtype: int64
```

[5]:
```
# Counts the number of cells with the value 'UNKNOWN' in each column of the crash_data DataFrame
print(crash_data.isin(['UNKNOWN']).sum())
```

```
Report Number                         0
Local Case Number                     0
Agency Name                           0
ACRS Report Type                      0
Crash Date/Time                       0
Route Type                            0
Road Name                             0
Cross-Street Name                     0
Off-Road Description                 11
Municipality                          0
Related Non-Motorist                  0
Collision Type                      584
Weather                             551
Surface Condition                   399
Light                               557
Traffic Control                     233
Driver Substance Abuse             9639
Non-Motorist Substance Abuse        176
Person ID                             0
Driver At Fault                       0
Injury Severity                       0
Circumstance                          0
Driver Distracted By              26352
Drivers License State                 0
Vehicle ID                            0
Vehicle Damage Extent              5279
Vehicle First Impact Location      2493
Vehicle Body Type                   833
Vehicle Movement                   2239
Vehicle Going Dir                     0
Speed Limit                           0
Driverless Vehicle                    0
```

```
Parked Vehicle                    0
Vehicle Year                      0
Vehicle Make                   2960
Vehicle Model                  3053
Latitude                          0
Longitude                         0
Location                          0
dtype: int64
```

[6]:
```python
# Counts the number of cells with the value 'Unknown' in each column of the
 ↪crash_data DataFrame
print(crash_data.isin(['Unknown']).sum())
```

```
Report Number                   0
Local Case Number               0
Agency Name                     0
ACRS Report Type                0
Crash Date/Time                 0
Route Type                     14
Road Name                       0
Cross-Street Name               0
Off-Road Description            0
Municipality                    0
Related Non-Motorist            0
Collision Type                  0
Weather                         0
Surface Condition               0
Light                           0
Traffic Control                 0
Driver Substance Abuse          0
Non-Motorist Substance Abuse    0
Person ID                       0
Driver At Fault              3785
Injury Severity                 0
Circumstance                    0
Driver Distracted By            0
Drivers License State           0
Vehicle ID                      0
Vehicle Damage Extent           0
Vehicle First Impact Location   0
Vehicle Body Type               0
Vehicle Movement                0
Vehicle Going Dir            4053
Speed Limit                     0
Driverless Vehicle            583
Parked Vehicle                  0
Vehicle Year                    0
Vehicle Make                    0
```

```
Vehicle Model                    0
Latitude                         0
Longitude                        0
Location                         0
dtype: int64
```

[7]:

```
# Removes the irrelevant columns from the crash_data DataFrame
crash_data = crash_data.drop(['Municipality', 'Circumstance','Off-Road␣
↪Description','Related Non-Motorist','Non-Motorist Substance Abuse'], axis=1)
```

[8]:

```
# Convert the 'Crash Date/Time' column from string format to a datetime object␣
↪using a specific format.
# 'format' specifies the expected date and time structure (e.g., 'mm/dd/yyyy hh:
↪mm:ss AM/PM').
# 'errors="coerce"' ensures that invalid parsing will result in NaT (Not a␣
↪Time) instead of an error.
crash_data['Crash Date/Time'] = pd.to_datetime(crash_data['Crash Date/Time'],␣
↪format='%m/%d/%Y %I:%M:%S %p', errors='coerce')

# Format the datetime objects back into a standardized string format␣
↪('YYYY-MM-DD HH:MM:SS').
# This step ensures consistency in the representation of date and time data.
crash_data['Crash Date/Time'] = crash_data['Crash Date/Time'].dt.
↪strftime('%Y-%m-%d %H:%M:%S')

# Convert the formatted string representation of date and time back into␣
↪datetime objects.
# This ensures that the column is of datetime type, suitable for further␣
↪datetime operations.
crash_data['Crash Date/Time'] = pd.to_datetime(crash_data['Crash Date/Time'])

# Extract the year from the 'Crash Date/Time' column and create a new column␣
↪named 'Year'.
crash_data['Year'] = crash_data['Crash Date/Time'].dt.year

# Extract the month (1=January, 12=December) and create a new column named␣
↪'Month'.
crash_data['Month'] = crash_data['Crash Date/Time'].dt.month

# Extract the day of the month (1-31) and create a new column named 'Day'.
crash_data['Day'] = crash_data['Crash Date/Time'].dt.day

# Extract the hour of the day (0-23) and create a new column named 'Hour'.
crash_data['Hour'] = crash_data['Crash Date/Time'].dt.hour

# Extract the day of the week (0=Monday, 6=Sunday) and create a new column␣
↪named 'Weekday'.
```

```
crash_data['Weekday'] = crash_data['Crash Date/Time'].dt.weekday
```

[9]:
```
# Replaces all NaN (missing) values in the crash_data DataFrame with the string␣
↪"UNKNOWN".
crash_data_replaced = crash_data.fillna("UNKNOWN")
```

[10]:
```
# Replaces all occurrences of the string 'Unknown' with 'UNKNOWN' in the␣
↪crash_data_replaced DataFrame.
crash_data_filled = crash_data_replaced.replace('Unknown', 'UNKNOWN')
```

[11]:
```
import numpy as np

# Replaces all occurrences of the string 'UNKNOWN' with NaN (missing value) in␣
↪the crash_data_filled DataFrame in-place
crash_data_filled.replace('UNKNOWN', np.nan, inplace=True)

# Removes all rows containing NaN values from the crash_data_filled DataFrame
crash_data_filled = crash_data_filled.dropna()
```

[12]:
```
# Prints the number of missing (NaN) values in each column of the␣
↪crash_data_filled DataFrame
print(crash_data_filled.isnull().sum())
```

```
Report Number                    0
Local Case Number                0
Agency Name                      0
ACRS Report Type                 0
Crash Date/Time                  0
Route Type                       0
Road Name                        0
Cross-Street Name                0
Collision Type                   0
Weather                          0
Surface Condition                0
Light                            0
Traffic Control                  0
Driver Substance Abuse           0
Person ID                        0
Driver At Fault                  0
Injury Severity                  0
Driver Distracted By             0
Drivers License State            0
Vehicle ID                       0
Vehicle Damage Extent            0
Vehicle First Impact Location    0
Vehicle Body Type                0
Vehicle Movement                 0
```

```
Vehicle Going Dir              0
Speed Limit                    0
Driverless Vehicle             0
Parked Vehicle                 0
Vehicle Year                   0
Vehicle Make                   0
Vehicle Model                  0
Latitude                       0
Longitude                      0
Location                       0
Year                           0
Month                          0
Day                            0
Hour                           0
Weekday                        0
dtype: int64
```

[13]:

```python
# Save the cleaned data to a CSV file
crash_data_filled.to_csv('cleaned_crash_data.csv', index=False)
```

[14]:

```python
# Display the first few rows of the cleaned dataset
crash_data_filled.head()
```

[14]:

```
   Report Number Local Case Number                Agency Name  \
5    MCP3348000Z         230051804   Montgomery County Police
6    MCP302600BD         230046425   Montgomery County Police
8    MCP3372001V         230065250   Montgomery County Police
9    MCP3005007M         230060937   Montgomery County Police
10    EJ786600CN         230057666  Gaithersburg Police Depar

         ACRS Report Type      Crash Date/Time        Route Type     Road Name  \
5            Injury Crash 2023-08-28 11:09:00  Maryland (State)    NORBECK RD
6   Property Damage Crash 2023-07-27 12:30:00            County  GREENTREE RD
8   Property Damage Crash 2023-11-10 20:24:00  Maryland (State)   GEORGIA AVE
9   Property Damage Crash 2023-10-16 19:33:00  Maryland (State)   GEORGIA AVE
10  Property Damage Crash 2023-09-30 10:34:00      Municipality    PERRY PKWY

         Cross-Street Name           Collision Type Weather  … Vehicle Make  \
5                 DRURY RD        SAME DIR REAR END  CLOUDY  …     MERCEDES
6       OLD GEORGETOWN RD  STRAIGHT MOVEMENT ANGLE   CLEAR  …         HOND
8                   MAY ST        SAME DIR REAR END   CLEAR  …       TOYOTA
9               LINDELL ST        HEAD ON LEFT TURN   CLEAR  …        HONDA
10  ENT TO SHOPPING CENTER  STRAIGHT MOVEMENT ANGLE  CLOUDY  …         HOND

   Vehicle Model   Latitude  Longitude                   Location  Year  \
5          ML360  39.116462 -77.050530     (39.11646167, -77.05053)  2023
6          PILOT  39.000144 -77.109881  (39.00014446, -77.10988077)  2023
```

```
8          CAMRY  39.072460 -77.064860   (39.0724598, -77.06486034)  2023
9         ACCORD  39.054407 -77.050488  (39.05440667, -77.05048833)  2023
10        ACCORD  39.148779 -77.213439  (39.14877898, -77.21343947)  2023

   Month Day Hour Weekday
5      8  28   11       0
6      7  27   12       3
8     11  10   20       4
9     10  16   19       0
10     9  30   10       5

[5 rows x 39 columns]
```

[15]:
```python
# Display basic information about the dataset
crash_data_filled.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 65000 entries, 5 to 137557
Data columns (total 39 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Report Number                    65000 non-null  object
 1   Local Case Number                65000 non-null  object
 2   Agency Name                      65000 non-null  object
 3   ACRS Report Type                 65000 non-null  object
 4   Crash Date/Time                  65000 non-null  datetime64[ns]
 5   Route Type                       65000 non-null  object
 6   Road Name                        65000 non-null  object
 7   Cross-Street Name                65000 non-null  object
 8   Collision Type                   65000 non-null  object
 9   Weather                          65000 non-null  object
 10  Surface Condition                65000 non-null  object
 11  Light                            65000 non-null  object
 12  Traffic Control                  65000 non-null  object
 13  Driver Substance Abuse           65000 non-null  object
 14  Person ID                        65000 non-null  object
 15  Driver At Fault                  65000 non-null  object
 16  Injury Severity                  65000 non-null  object
 17  Driver Distracted By             65000 non-null  object
 18  Drivers License State            65000 non-null  object
 19  Vehicle ID                       65000 non-null  object
 20  Vehicle Damage Extent            65000 non-null  object
 21  Vehicle First Impact Location    65000 non-null  object
 22  Vehicle Body Type                65000 non-null  object
 23  Vehicle Movement                 65000 non-null  object
 24  Vehicle Going Dir                65000 non-null  object
 25  Speed Limit                      65000 non-null  float64
```

```
 26  Driverless Vehicle              65000 non-null  object
 27  Parked Vehicle                  65000 non-null  object
 28  Vehicle Year                    65000 non-null  float64
 29  Vehicle Make                    65000 non-null  object
 30  Vehicle Model                   65000 non-null  object
 31  Latitude                        65000 non-null  float64
 32  Longitude                       65000 non-null  float64
 33  Location                        65000 non-null  object
 34  Year                            65000 non-null  int32
 35  Month                           65000 non-null  int32
 36  Day                             65000 non-null  int32
 37  Hour                            65000 non-null  int32
 38  Weekday                         65000 non-null  int32
dtypes: datetime64[ns](1), float64(4), int32(5), object(29)
memory usage: 18.6+ MB
```

[16]:
```python
# Display descriptive statistics for numerical columns
crash_data_filled.describe()
```

[16]:
```
                     Crash Date/Time   Speed Limit  Vehicle Year  \
count                          65000  65000.000000  65000.000000
mean   2019-04-24 18:13:21.692307456     35.275692   2010.516569
min              2015-01-01 00:30:00      0.000000      0.000000
25%              2016-12-10 11:26:00     30.000000   2006.000000
50%              2019-01-22 08:29:00     35.000000   2012.000000
75%              2021-09-25 19:46:00     40.000000   2016.000000
max              2023-12-31 21:19:00     75.000000   9999.000000
std                              NaN      7.783583     57.562331

           Latitude     Longitude          Year         Month           Day  \
count  65000.000000  65000.000000  65000.000000  65000.000000  65000.000000
mean      39.084622    -77.114513   2018.789877      6.777985     15.686215
min       38.353495    -77.651753   2015.000000      1.000000      1.000000
25%       39.025473    -77.188308   2016.000000      4.000000      8.000000
50%       39.075917    -77.109294   2019.000000      7.000000     16.000000
75%       39.140523    -77.043739   2021.000000     10.000000     23.000000
max       39.988369    -76.322565   2023.000000     12.000000     31.000000
std        0.072178      0.093017      2.647969      3.485675      8.780185

               Hour       Weekday
count  65000.000000  65000.000000
mean      13.250785      2.771677
min        0.000000      0.000000
25%        9.000000      1.000000
50%       14.000000      3.000000
75%       17.000000      4.000000
max       23.000000      6.000000
```

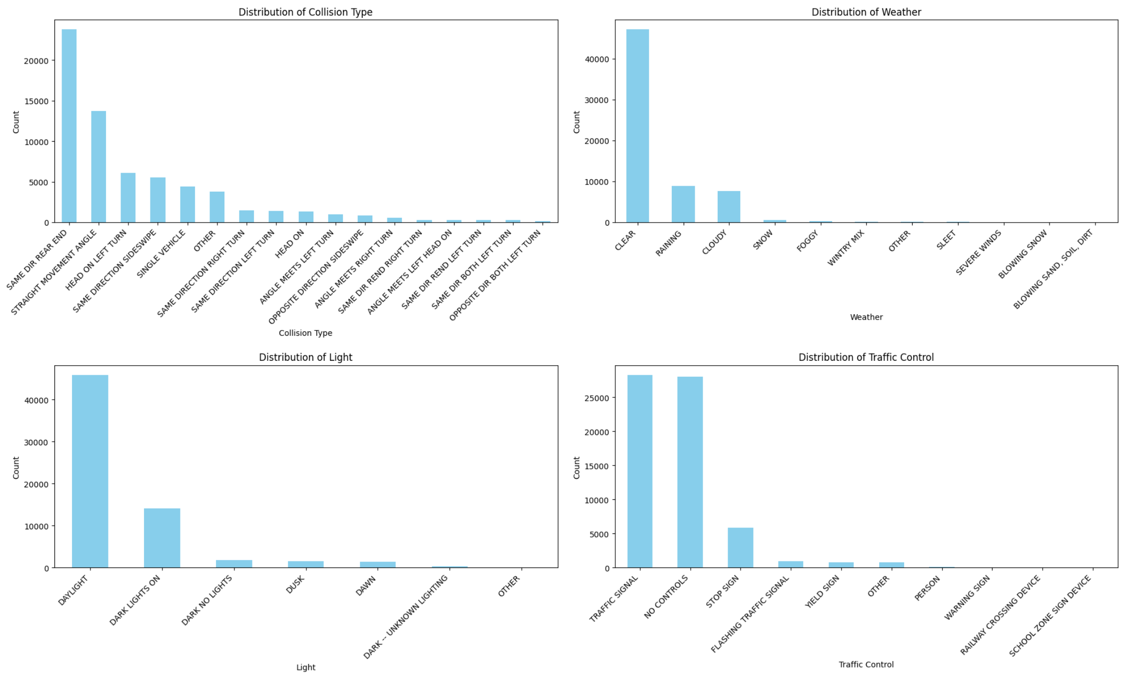
```
std        5.087246      1.887400
```

[17]:
```
# Handel outliers in 'Vehicle Year' column

import datetime

# Define valid range for vehicle year
current_year = datetime.datetime.now().year
valid_range = (1900, current_year)

# Remove rows where 'Vehicle Year' is outside the valid range
crash_data_filled = crash_data_filled[
    (crash_data_filled['Vehicle Year'] >= valid_range[0]) &
    (crash_data_filled['Vehicle Year'] <= valid_range[1])
]
```

Basic EDA:

[18]:
```
# Plot distributions of key features
import matplotlib.pyplot as plt
import seaborn as sns
```

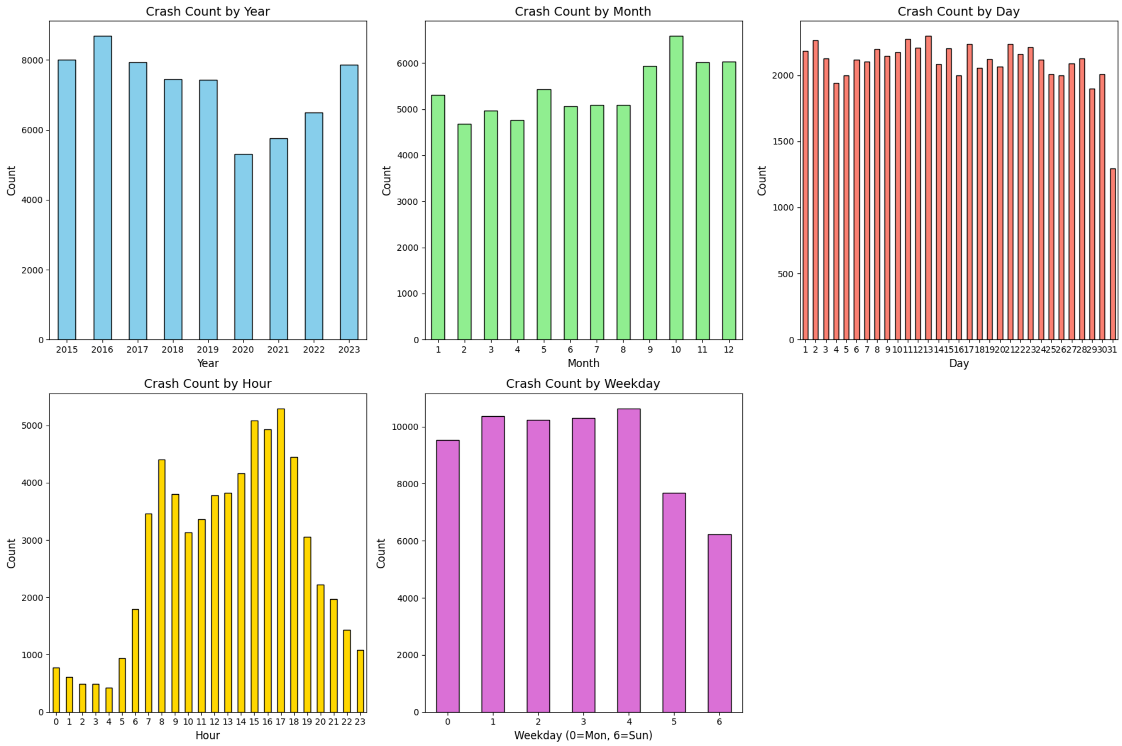
[19]:
```
# Categorical feature distribution: Collision Type, Weather, Light, Traffic␣
↪Control

# Improved layout for categorical feature distribution plots
plt.figure(figsize=(20, 12))

# Categorical features to plot
categorical_features = ['Collision Type', 'Weather', 'Light', 'Traffic Control']

for i, feature in enumerate(categorical_features, 1):
    plt.subplot(2, 2, i)  # Adjust rows and columns for more space
    crash_data_filled[feature].value_counts().plot(kind='bar', color='skyblue')
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better␣
↪readability

# Adjust layout spacing
plt.tight_layout(h_pad=2, w_pad=2)
plt.show()
```



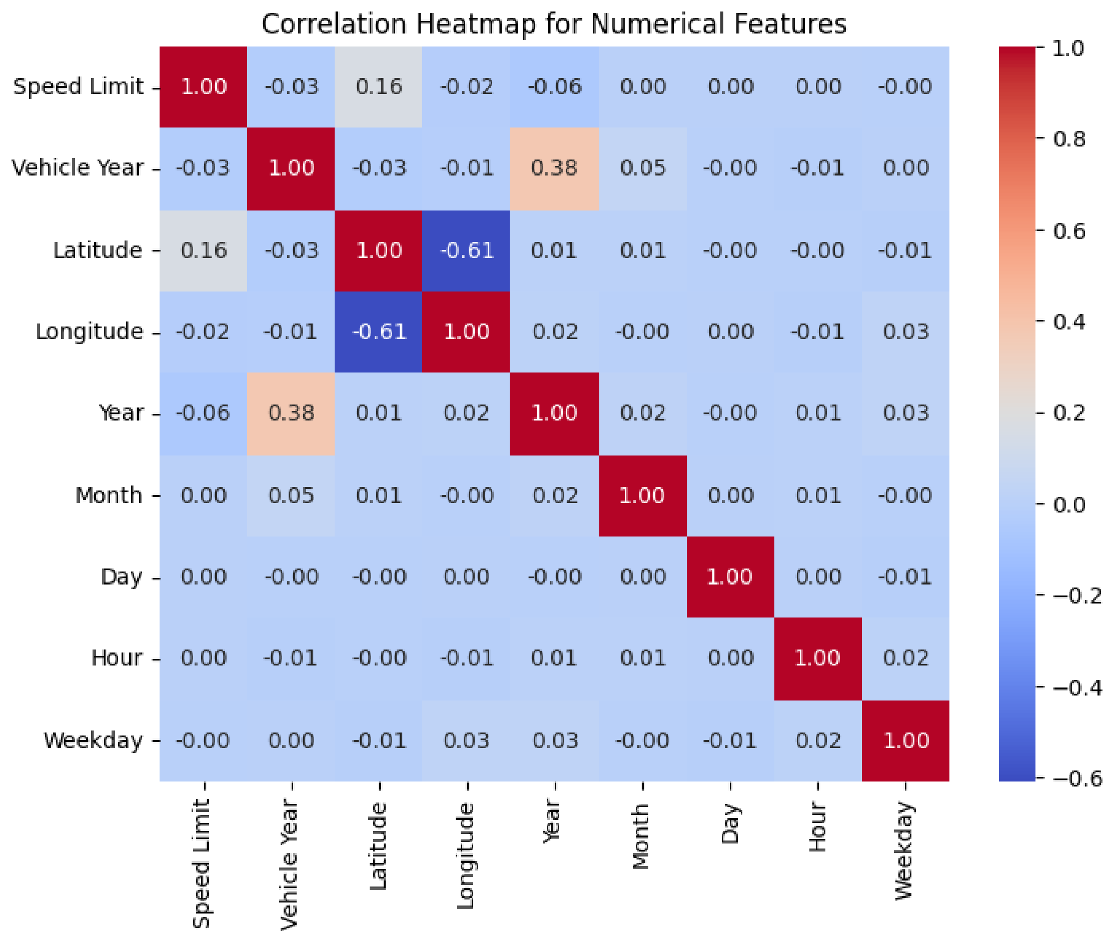
[20]:

```python
# Distribution of temporal features: Year, Month, Day, Hour, Weekday

plt.figure(figsize=(18, 12))

# Crash Count by Year
plt.subplot(2, 3, 1)
crash_data_filled['Year'].value_counts().sort_index().plot(kind='bar',
    color='skyblue', edgecolor='black')
plt.title('Crash Count by Year', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=0)

# Crash Count by Month
plt.subplot(2, 3, 2)
crash_data_filled['Month'].value_counts().sort_index().plot(kind='bar',
    color='lightgreen', edgecolor='black')
plt.title('Crash Count by Month', fontsize=14)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=0)

# Crash Count by Day
plt.subplot(2, 3, 3)
```

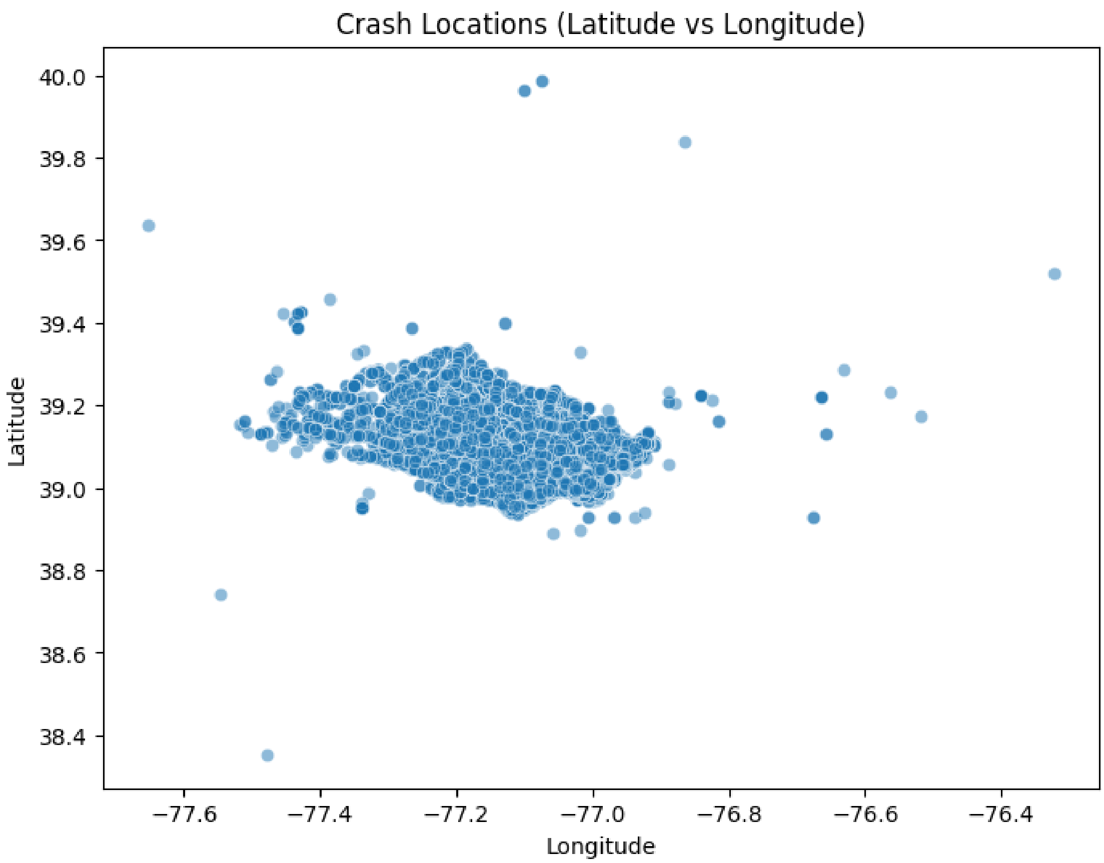
```python
crash_data_filled['Day'].value_counts().sort_index().plot(kind='bar',␣
↪color='salmon', edgecolor='black')
plt.title('Crash Count by Day', fontsize=14)
plt.xlabel('Day', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=0)

# Crash Count by Hour
plt.subplot(2, 3, 4)
crash_data_filled['Hour'].value_counts().sort_index().plot(kind='bar',␣
↪color='gold', edgecolor='black')
plt.title('Crash Count by Hour', fontsize=14)
plt.xlabel('Hour', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=0)

# Crash Count by Weekday
plt.subplot(2, 3, 5)
crash_data_filled['Weekday'].value_counts().sort_index().plot(kind='bar',␣
↪color='orchid', edgecolor='black')
plt.title('Crash Count by Weekday', fontsize=14)
plt.xlabel('Weekday (0=Mon, 6=Sun)', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=0)

plt.tight_layout()
plt.show()
```



[21]:
```
# Correlation heatmap for numerical features
numerical_features = crash_data_filled.select_dtypes(include='number').columns
plt.figure(figsize=(8, 6))
sns.heatmap(crash_data_filled[numerical_features].corr(), annot=True,␣
↪cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap for Numerical Features')
plt.show()
```



[22]:
```
# Preview some geographical data
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Longitude', y='Latitude', data=crash_data_filled, alpha=0.5)
plt.title('Crash Locations (Latitude vs Longitude)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



1.1.5 Overview of EDA Visualizations

Distributions **Collision Type**:
The most common collision type is **"Same Direction Rear End"**, indicating rear-end collisions are a major issue.
Weather Conditions:
Crashes predominantly occur in **clear weather**, showing that visibility alone may not prevent crashes.
Lighting Conditions:
Most crashes happen during **daylight**, likely due to higher traffic volumes during the day.
Traffic Control:
A significant number of crashes occur at **traffic signals** or under **no controls**, highlighting areas for potential traffic management improvements. #### *Temporal Trends*
Yearly Trends:
Crashes are distributed across years, with some variation that may reflect policy or traffic volume changes.
Monthly and Daily Trends:
Crashes are fairly consistent across months and days, though specific months might show slight peaks.
Hourly Trends:

Clear peaks during morning and evening **rush hours**, aligning with traffic density during commuting times.
Weekly Trends:
Crashes show variations across weekdays, with potential reductions during weekends. #### *Correlation Heatmap*
The correlation matrix highlights relationships between numerical variables:
Speed Limit shows a mild positive correlation with latitude.
Vehicle Year is weakly correlated with temporal features, indicating newer vehicles in recent years.
Geographical Data
The scatterplot of **latitude** and **longitude** reveals clustering of crash locations, likely around high-traffic or urban areas. These clusters can be further analyzed for hotspot identification.

1. How do environmental conditions impact traffic crashes?
Objective: Examine the role of environmental factors like weather and road surface conditions in contributing to traffic crashes. Insights can inform strategies to improve road safety.

[23]:
```
# Get the unique values in the 'Weather' column and print them
# This step is intended to understand all possible weather conditions present␣
↪in the dataset, which helps in further data processing and analysis
weather_unique_values = crash_data_filled['Weather'].unique()
print("Unique values in 'Weather':")
print(weather_unique_values)

# Get the unique values in the 'Surface Condition' column and print them
# This step is intended to understand all possible surface conditions in the␣
↪dataset, helping to determine if data merging or standardization is needed
surface_condition_unique_values = crash_data_filled['Surface Condition'].
↪unique()
print("\nUnique values in 'Surface Condition':")
print(surface_condition_unique_values)
```

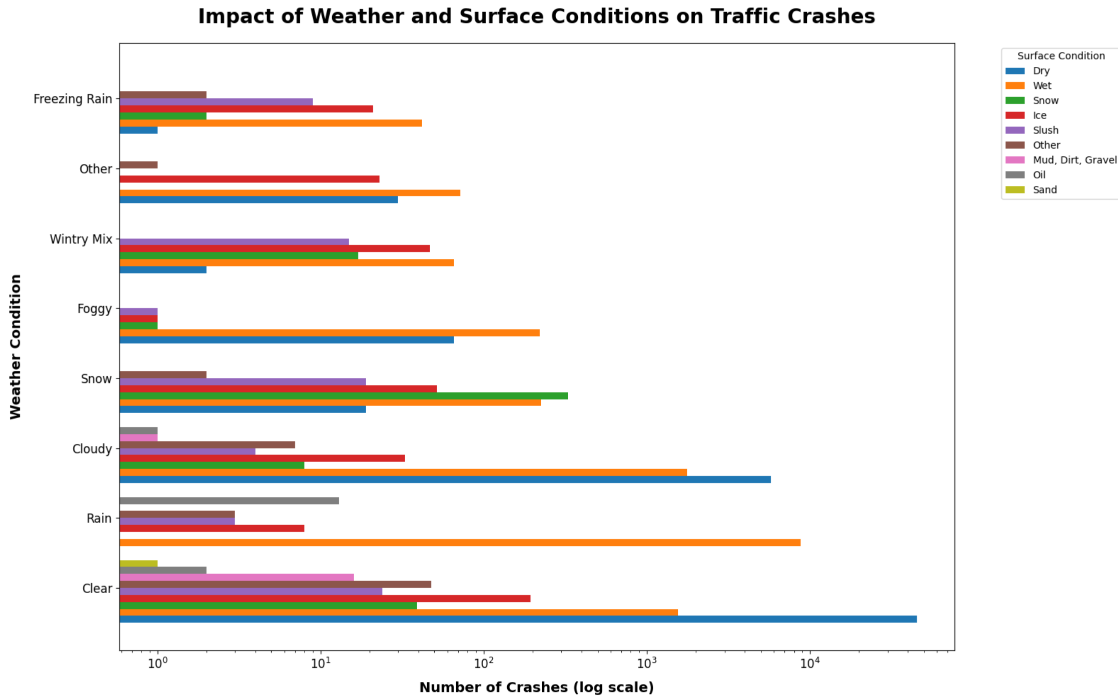
```
Unique values in 'Weather':
['CLOUDY' 'CLEAR' 'RAINING' 'SNOW' 'FOGGY' 'WINTRY MIX' 'OTHER' 'SLEET'
 'BLOWING SNOW' 'SEVERE WINDS' 'BLOWING SAND, SOIL, DIRT']

Unique values in 'Surface Condition':
['DRY' 'ICE' 'WET' 'WATER(STANDING/MOVING)' 'SNOW' 'MUD, DIRT, GRAVEL'
 'OTHER' 'SLUSH' 'OIL' 'SAND']
```

[24]:
```
# Merge similar categories in Weather and Surface Condition
crash_data_filled['Weather'] = crash_data_filled['Weather'].replace({
    'CLOUDY': 'Cloudy',
    'CLEAR': 'Clear',
    'RAINING': 'Rain',
    'SNOW': 'Snow',
    'FOGGY': 'Foggy',
    'WINTRY MIX': 'Wintry Mix',
    'BLOWING SNOW': 'Snow',
```

```
    'SEVERE WINDS': 'Windy',
    'SEVERE CROSSWINDS': 'Windy',
    'BLOWING SAND, SOIL, DIRT': 'Windy',
    'Fog, Smog, Smoke': 'Foggy',
    'Freezing Rain Or Freezing Drizzle': 'Freezing Rain',
    'SLEET': 'Freezing Rain',
    'OTHER': 'Other'
})

crash_data_filled['Surface Condition'] = crash_data_filled['Surface Condition'].
 ↪replace({
    'DRY': 'Dry',
    'ICE': 'Ice',
    'WET': 'Wet',
    'WATER(STANDING/MOVING)': 'Wet',
    'WATER (standing, moving)': 'Wet',
    'MUD, DIRT, GRAVEL': 'Mud, Dirt, Gravel',
    'SLUSH': 'Slush',
    'OIL': 'Oil',
    'SAND': 'Sand',
    'SNOW': 'Snow',
    'OTHER': 'Other'
})
```

[25]:
```
import matplotlib.pyplot as plt
import numpy as np

# Group data by Weather and Surface Condition
grouped_data = crash_data_filled.groupby(['Weather', 'Surface Condition']).
 ↪size().unstack(fill_value=0)

# Sort data by total crashes per weather condition
grouped_data['Total_Crashes'] = grouped_data.sum(axis=1)
grouped_data = grouped_data.sort_values('Total_Crashes', ascending=False).
 ↪drop(columns='Total_Crashes')

# Top 8 weather conditions
top_weather_conditions = grouped_data.head(8)
sorted_conditions = top_weather_conditions.sum().sort_values(ascending=False).
 ↪index
top_weather_conditions = top_weather_conditions[sorted_conditions]

# Plotting a horizontal bar chart with a log scale on the x-axis
fig, ax = plt.subplots(figsize=(16, 10))

# Setting y values and width for bars
y = np.arange(len(top_weather_conditions.index))
```

```python
width = 0.1  # Width of each bar

# Plotting each Surface Condition as a separate bar within each group
for i, surface_condition in enumerate(top_weather_conditions.columns):
    ax.barh(y + i * width, top_weather_conditions[surface_condition], width, label=surface_condition)

# Set x-axis to logarithmic scale
ax.set_xscale('log')

# Customize the plot
ax.set_title('Impact of Weather and Surface Conditions on Traffic Crashes', fontsize=20, pad=20, weight='bold')
ax.set_xlabel('Number of Crashes (log scale)', fontsize=14, labelpad=10, weight='bold')
ax.set_ylabel('Weather Condition', fontsize=14, labelpad=10, weight='bold')
ax.set_yticks(y + width * (len(top_weather_conditions.columns) / 2))
ax.set_yticklabels(top_weather_conditions.index, fontsize=12)
ax.tick_params(axis='x', labelsize=12)
ax.tick_params(axis='y', labelsize=12)

# Add a legend to differentiate between surface conditions
ax.legend(title='Surface Condition', bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=10)

# Adjust layout for better display
plt.tight_layout()

# Show plot
plt.show()
```



Observations: For this analysis, we aimed to understand how environmental conditions, such as weather and surface conditions, contribute to traffic crashes. The objective was to investigate the relationship between traffic crashes and these factors to identify critical conditions that significantly impact road safety. To achieve this, we pre-processed the `Weather` and `Surface Condition` columns by unifying similar categories (e.g., "RAINING" and "Rain" were merged into "Rain"). Crashes were then grouped by `Weather` and `Surface Condition`, with the crash counts for each combination calculated. The top eight most frequent weather conditions were included in the final analysis, and a horizontal bar chart with a logarithmic scale on the x-axis was created to visually compare the impact of different weather and surface condition combinations.

The results reveal that clear weather consistently accounts for the highest number of crashes across multiple surface conditions, especially on dry and wet surfaces, which may be due to high traffic volumes during clear weather. Rain significantly increases crash counts on wet surfaces, as expected, due to reduced visibility and slippery roads. Snowy conditions contribute to a high number of crashes on snow-covered surfaces, highlighting the challenges of maintaining traction. Similarly, wintry mix and freezing rain result in a substantial number of crashes, particularly on surfaces like ice and slush, emphasizing their hazardous nature. Cloudy weather contributes to a moderate number of crashes across various surface conditions, potentially due to reduced visibility and cautious driving, while other conditions like foggy and windy weather have lower crash counts but still pose significant risks in certain scenarios.

This analysis underscores that wet and icy road surfaces pose the highest risk across most weather conditions. Effective countermeasures, such as improved road maintenance, public alerts during hazardous weather, and better infrastructure for drainage and snow removal, can help mitigate these risks. Public awareness campaigns about the dangers of specific weather and surface combinations can also improve driver behavior and reduce crashes. Overall, this study highlights the critical role of environmental factors in road safety and offers actionable insights for traffic management and crash prevention.

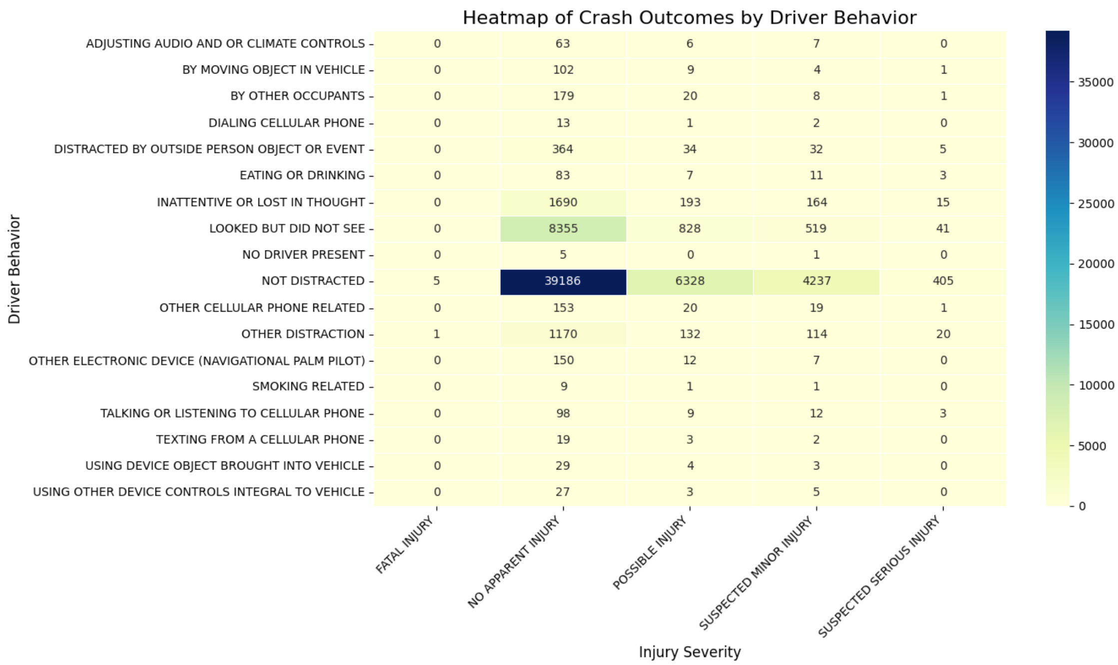
2. What is the influence of driver behavior on crash outcomes?
Objective: Analyze how driver actions, such as distraction, speeding, and impairment, affect the frequency and severity of crashes. Findings can guide safety programs and policies.

[26]:
```
# Check unique values in 'Driver Distracted By'
driver_distracted_unique_values = crash_data_filled['Driver Distracted By'].
 ↪unique()
print("Unique values in 'Driver Distracted By':")
print(driver_distracted_unique_values)

# Check unique values in 'Injury Severity'
injury_severity_unique_values = crash_data_filled['Injury Severity'].unique()
print("\nUnique values in 'Injury Severity':")
print(injury_severity_unique_values)
```

```
Unique values in 'Driver Distracted By':
['NOT DISTRACTED' 'LOOKED BUT DID NOT SEE'
 'TALKING OR LISTENING TO CELLULAR PHONE' 'OTHER DISTRACTION'
 'EATING OR DRINKING' 'INATTENTIVE OR LOST IN THOUGHT'
 'BY MOVING OBJECT IN VEHICLE' 'ADJUSTING AUDIO AND OR CLIMATE CONTROLS'
 'OTHER ELECTRONIC DEVICE (NAVIGATIONAL PALM PILOT)'
 'DISTRACTED BY OUTSIDE PERSON OBJECT OR EVENT'
 'OTHER CELLULAR PHONE RELATED' 'BY OTHER OCCUPANTS'
 'USING OTHER DEVICE CONTROLS INTEGRAL TO VEHICLE'
 'USING DEVICE OBJECT BROUGHT INTO VEHICLE' 'DIALING CELLULAR PHONE'
 'TEXTING FROM A CELLULAR PHONE' 'NO DRIVER PRESENT' 'SMOKING RELATED']

Unique values in 'Injury Severity':
['NO APPARENT INJURY' 'SUSPECTED MINOR INJURY' 'POSSIBLE INJURY'
 'SUSPECTED SERIOUS INJURY' 'FATAL INJURY']
```

[27]:
```
import seaborn as sns

# Create a crosstabulation table, grouping by 'Driver Distracted By' and␣
 ↪'Injury Severity'
heatmap_data = pd.crosstab(crash_data_filled['Driver Distracted By'],␣
 ↪crash_data_filled['Injury Severity'])

# Draw a heat map
plt.figure(figsize=(14, 8))
sns.heatmap(
    heatmap_data,
    annot=True,
    fmt='d',
```

```
    cmap='YlGnBu',
    linewidths=.5
)
plt.title('Heatmap of Crash Outcomes by Driver Behavior', fontsize=16)
plt.xlabel('Injury Severity', fontsize=12)
plt.ylabel('Driver Behavior', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Observations: For this analysis, we examined how driver behavior influences crash outcomes, focusing on the relationship between specific driver actions and the severity of injuries resulting from crashes. The objective was to identify behaviors most strongly associated with crashes of varying severity, providing insights to guide safety programs and policies.

To achieve this, we utilized the `Driver Distracted By` and `Injury Severity` columns in the dataset. We first standardized the unique categories within these columns to ensure consistency. A crosstabulation was created to group the data by driver behavior and injury severity, allowing us to analyze the frequency of crashes associated with each behavior across different severity levels. A heatmap visualization was then generated to provide a clear and interpretable representation of the data.

The heatmap reveals several key insights. The most frequent driver behavior recorded was "Not Distracted," accounting for the majority of crashes across all injury severities, with over 39,000 crashes involving no apparent injury. This finding suggests that crashes are often caused by factors other than distraction, such as road conditions or other external influences. Among distracted behaviors, “Looked But Did Not See” is the most prevalent, with significant crash counts across all injury categories, particularly those with no apparent injury. Other distracted behaviors, such as “Distracted By Outside Person, Object, or Event,” “Inattentive or Lost in Thought,” and “Eating or Drinking,” also contribute to crash frequencies but at lower rates.

The results indicate that distracted behaviors generally result in fewer crashes with severe injuries compared to those with minor or no apparent injuries. For example, behaviors like “Dialing Cellular Phone” or “Texting from a Cellular Phone” are associated with low crash counts but have the potential to cause serious injuries. This suggests the importance of addressing distraction-related behaviors through targeted interventions, such as public awareness campaigns, stricter enforcement of distracted driving laws, and the implementation of technology to mitigate distractions.

Overall, the analysis highlights the significant impact of driver behavior on crash outcomes. While the majority of crashes occur when drivers are not distracted, distracted behaviors remain a critical area for intervention, especially given their potential to cause severe injuries. These findings can inform policy decisions and the development of safety programs to reduce crashes and improve road safety.

3. What factors contribute to severe traffic crashes?
   Objective: Identify elements such as road conditions, vehicle characteristics, and driver actions that are associated with severe crashes. Results can inform targeted interventions to reduce severe incidents.

[28]:
```
# fiter data when there is fatal injury or serious injury
crash_data = pd.read_csv("Crash_Reporting_-_Drivers_Data.csv",low_memory=False)
crash_data.head()
```

[28]:
```
  Report Number Local Case Number                  Agency Name  \
0    DM8479000T         210020119  Takoma Park Police Depart
1   MCP2970000R          15045937                 MONTGOMERY
2   MCP20160036         180040948   Montgomery County Police
3    EJ7879003C         230048975  Gaithersburg Police Depar
4   MCP2967004Y         230070277   Montgomery County Police

        ACRS Report Type         Crash Date/Time        Route Type  \
0  Property Damage Crash  05/27/2021 07:40:00 PM               NaN
1  Property Damage Crash  09/11/2015 01:29:00 PM               NaN
2  Property Damage Crash  08/17/2018 02:25:00 PM               NaN
3           Injury Crash  08/11/2023 06:00:00 PM               NaN
4  Property Damage Crash  12/06/2023 06:42:00 PM  Maryland (State)

         Road Name Cross-Street Name               Off-Road Description  \
0              NaN               NaN                     IN PARKING LOT
1              NaN               NaN    Parking Lot: \n2525 Ennalls Ave
2              NaN               NaN  PARKING LOT OF 16246 FREDERICK RD
3              NaN               NaN                   1 N SUMMIT DRIVE
4  CONNECTICUT AVE      BALTIMORE ST                                NaN

  Municipality  … Vehicle Going Dir Speed Limit Driverless Vehicle  \
```

```
0          NaN  …            NaN         0.0                No
1          NaN  …          South         5.0                No
2          NaN  …           West        15.0                No
3          NaN  …        Unknown        15.0                No
4   KENSINGTON  …          South        35.0                No

  Parked Vehicle Vehicle Year Vehicle Make Vehicle Model   Latitude  \
0            Yes       2017.0         HINO           TWK  38.987657
1             No       2012.0       TOYOTA            SU  39.039917
2             No       2015.0         MAZD            TK  38.743373
3             No       2018.0          RAM            TK  39.145873
4             No       2017.0         AUDI            A3  39.025170

   Longitude                     Location
0 -76.987545    (38.98765667, -76.987545)
1 -77.053649  (39.03991652, -77.05364898)
2 -77.546997    (38.743373, -77.54699707)
3 -77.191940  (39.14587303, -77.19194047)
4 -77.076333  (39.02517017, -77.07633333)

[5 rows x 39 columns]
```

[29]:

```python
# Values to filter
values_to_keep = ['SUSPECTED SERIOUS INJURY', 'FATAL INJURY']

# Filter rows where the 'Injury Severity' column has specific values
df = crash_data
injury_df = df[df['Injury Severity'].isin(values_to_keep)]

print(injury_df.head(3))
```

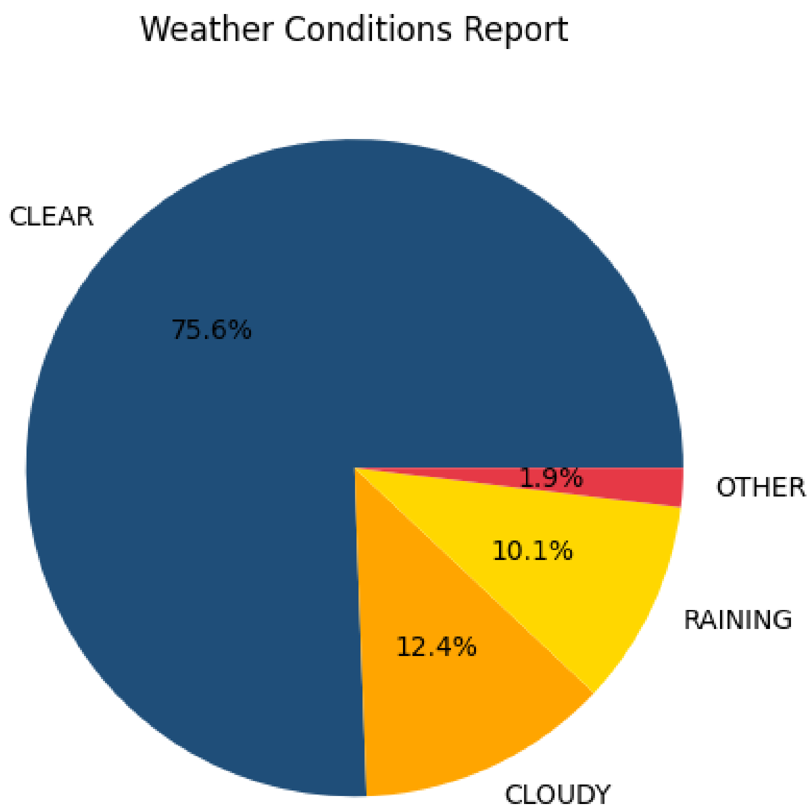
```
    Report Number Local Case Number                Agency Name  \
84     MCP291400F6         230059993   Montgomery County Police
148     DD55830089         230054878  Rockville Police Departme
261    MCP12270021         230063063   Montgomery County Police

    ACRS Report Type          Crash Date/Time        Route Type  \
84      Injury Crash  10/12/2023 03:40:00 AM        US (State)
148     Injury Crash  09/14/2023 04:22:00 PM  Maryland (State)
261      Fatal Crash  10/28/2023 12:15:00 PM              Ramp

                          Road Name Cross-Street Name Off-Road Description  \
84                     COLESVILLE RD      FRANKLIN AVE                  NaN
148                    HUNGERFORD DR   ENT TO BUSINESS                  NaN
261   RAMP 8 FR US 29 SB TO DUSTIN RD         DUSTIN RD                  NaN

    Municipality  … Vehicle Going Dir Speed Limit Driverless Vehicle  \
```

```
84           NaN  …             North        35.0                 No
148          NaN  …             South        40.0                 No
261          NaN  …             North        30.0                 No

    Parked Vehicle Vehicle Year Vehicle Make Vehicle Model   Latitude  \
84              No       2022.0        CHEVY        CAMARO  39.009453
148             No       2022.0        HONDA      GOLDWING  39.089287
261             No       2009.0      HYUNDAI       ELANTRA  39.122468

     Longitude                     Location
84  -77.017724  (39.00945337, -77.01772396)
148 -77.151436    (39.08928733, -77.151436)
261 -76.926338  (39.12246766, -76.92633791)

[3 rows x 39 columns]
```

[30]:

```
# create a pie char to show the most related weather factor of a car crash
from matplotlib import colormaps
print(injury_df.value_counts(injury_df["Weather"]))
slices = [1108, 182, 148, 28]
labels = ['CLEAR', 'CLOUDY', 'RAINING', 'OTHER']
# Generate colors from a colormap
cmap = colormaps['summer']  # Choose a colormap
colors_weather = ['#1F4E79', '#FFA500', '#FFD700', '#E63946']
plt.pie(slices, labels = labels, autopct = '%1.1f%%', colors = colors_weather)
plt.title("Weather Conditions Report")
plt.tight_layout()
plt.show()
```

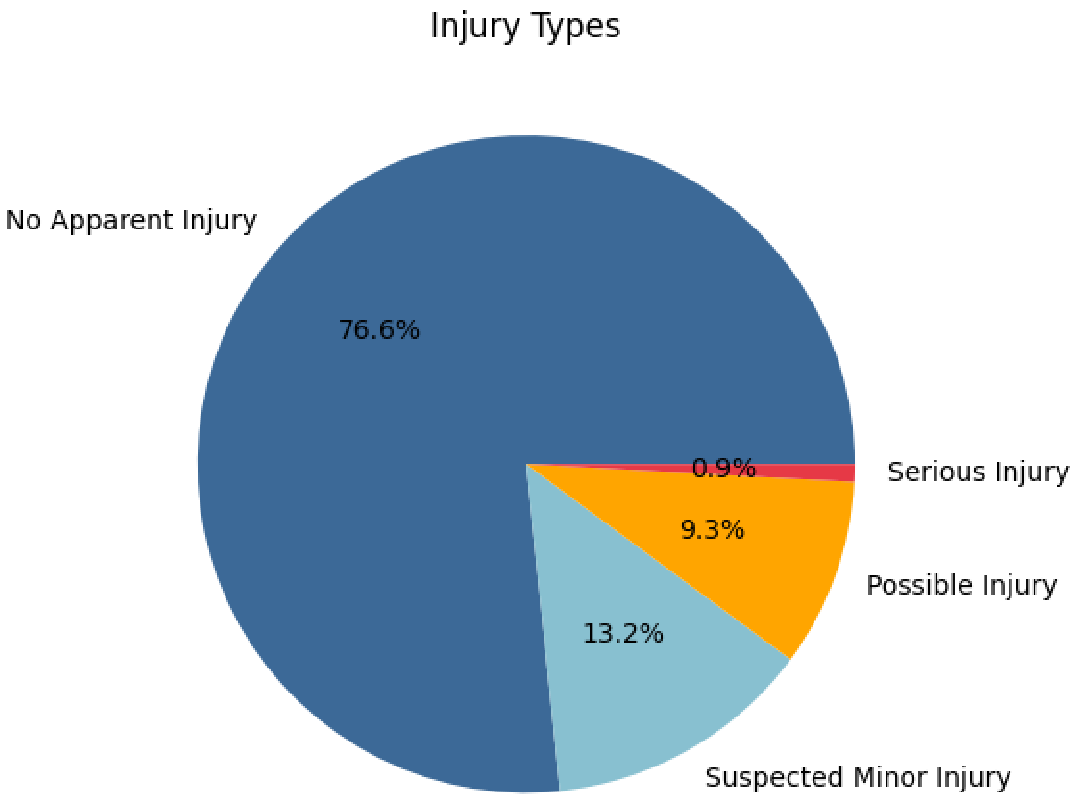
```
Weather
CLEAR           878
CLOUDY          153
RAINING         121
FOGGY             6
WINTRY MIX        5
SNOW              3
UNKNOWN           3
BLOWING SNOW      1
OTHER             1
SLEET             1
Name: count, dtype: int64
```



[31]:
```
# how is the contribution of sever injury to whole
crash_data.dropna()
print(crash_data.value_counts(crash_data["Injury Severity"]))
slices = [152952, 26424, 18493, 1717]
labels = ['No Apparent Injury', 'Suspected Minor Injury', 'Possible Injury ',␣
↪'Serious Injury']
# Generate colors from a colormap
cmap = colormaps['cool']  # Choose a colormap
colors_injury = ['#3C6997', '#88C0D0', '#FFA500', '#E63946']
plt.pie(slices, labels = labels, autopct = '%1.1f%%', colors = colors_injury)
plt.title("Injury Types")
plt.tight_layout()
plt.show()
```

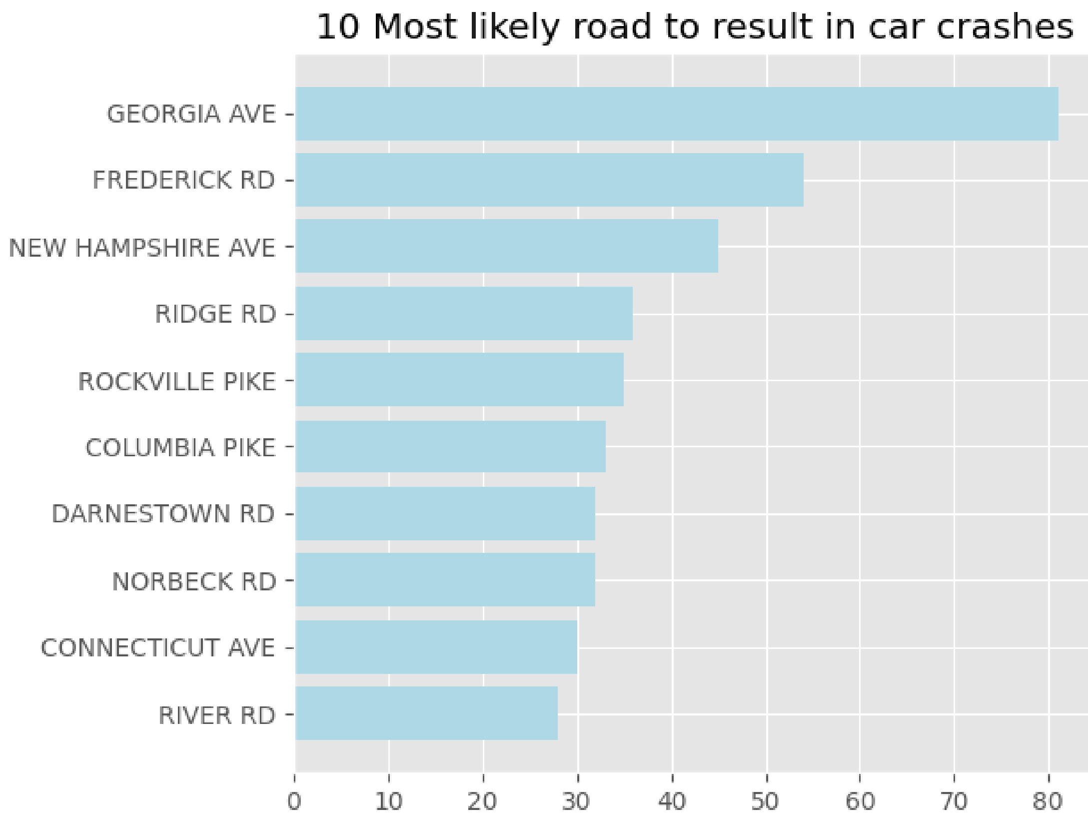
```
Injury Severity
NO APPARENT INJURY          112888
POSSIBLE INJURY              13965
SUSPECTED MINOR INJURY        9467
SUSPECTED SERIOUS INJURY      1126
```

```
FATAL INJURY                         117
Name: count, dtype: int64
```



[32]:
```
# create a vertical bar char to show the most related route type factor of a␣
↪car crash
road_name  = injury_df['Road Name']
from collections import Counter
road_counter = Counter()
for response in road_name.dropna():
    road_counter.update(response.split(';'))
Road=[]
Count=[]
for item in road_counter.most_common(10):
    Road.append(item[0])
    Count.append(item[1])
Road.reverse()
Count.reverse()
plt.style.use('ggplot')
plt.barh(Road, Count, color = 'lightblue')
```

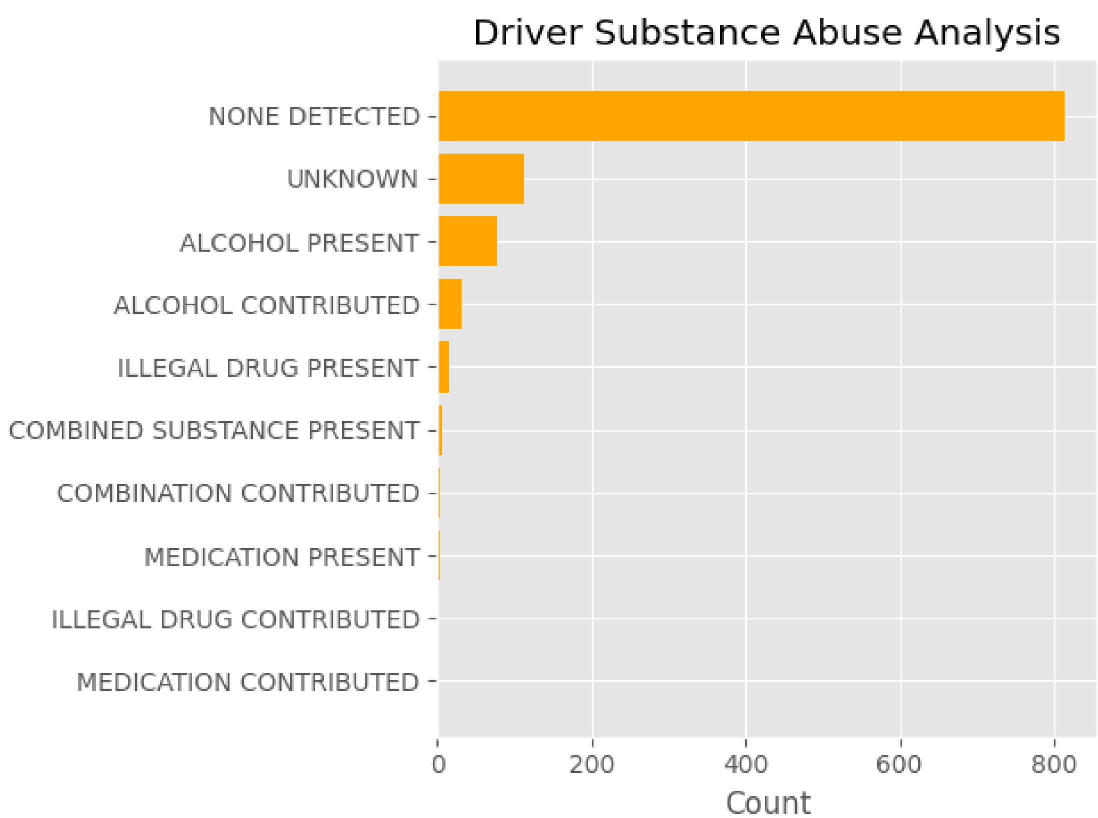
```
plt.title('10 Most likely road to result in car crashes')
plt.tight_layout()
plt.show()
```



[33]:
```
# Driver Substance Abuse Analysis
abuse = injury_df["Driver Substance Abuse"]
print(abuse.value_counts())
abuse_counter = Counter()
# Split responses and update the Counter
for response in abuse.dropna():  # Skip NaN values
    abuse_counter.update(response.split(';'))  # Split by ';'

# Extract the top 10 most common substances
abuse_types = []
counts = []
for item in abuse_counter.most_common(10):
    abuse_types.append(item[0])
    counts.append(item[1])

# Reverse the lists for a better horizontal bar chart
abuse_types.reverse()
```

```
counts.reverse()

# Plot the horizontal bar chart
plt.style.use('ggplot')
plt.barh(abuse_types, counts, color='orange')
plt.xlabel("Count")
plt.title("Driver Substance Abuse Analysis")
plt.tight_layout()
plt.show()
```

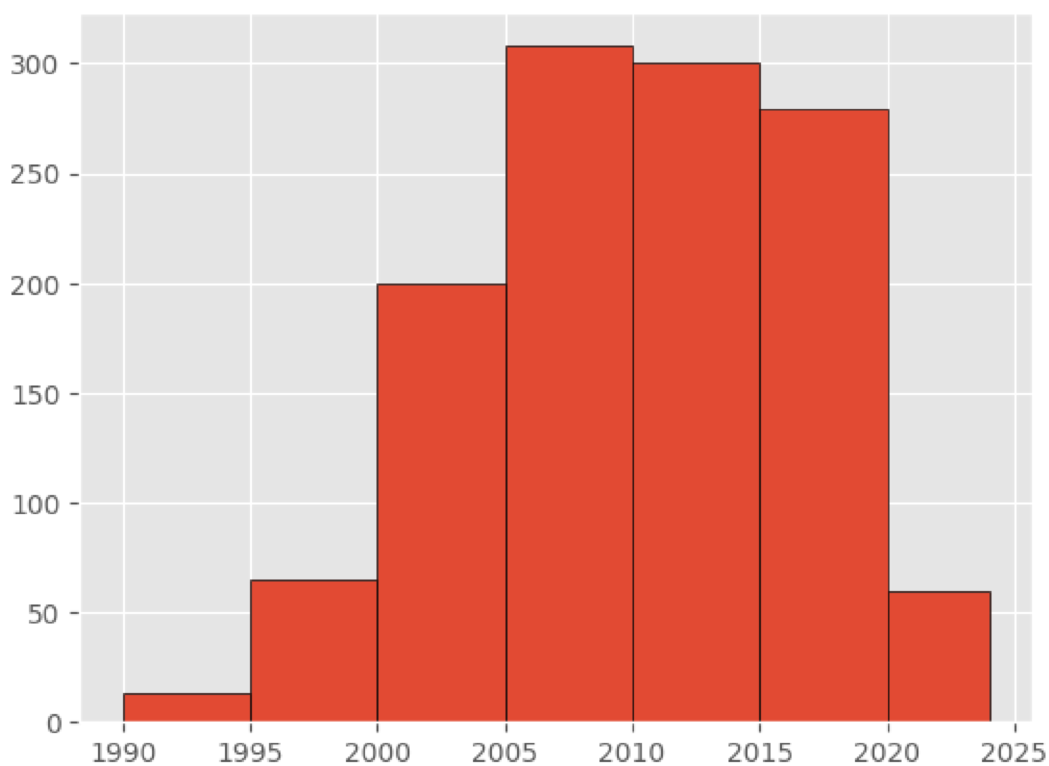
```
Driver Substance Abuse
NONE DETECTED                815
UNKNOWN                      113
ALCOHOL PRESENT               77
ALCOHOL CONTRIBUTED           31
ILLEGAL DRUG PRESENT          14
COMBINED SUBSTANCE PRESENT     7
COMBINATION CONTRIBUTED        4
MEDICATION PRESENT             3
ILLEGAL DRUG CONTRIBUTED       2
MEDICATION CONTRIBUTED         1
OTHER                          1
Name: count, dtype: int64
```



Observations: From the analysis above, we can conclude that serious car crashes including fatal crashes are 0.09% of the overall car crashes. And most drivers in car crashes are likely to have no apparent injury or minor injury. That suggests drivers in this area are pretty safe. And from the drug abuse analysis plot, we can conclude that most of our drivers are in good shape without being addicticted to drugs. Although, there is a small portion of people have alcohol before driving, which is definitely a strong evidence of causing fatal crashes. From the analysis above, it seems there is no specific road where there is going to be a car crashes. But Goergia Ave has the most possible that there could be a serious car crash compared with other raod.

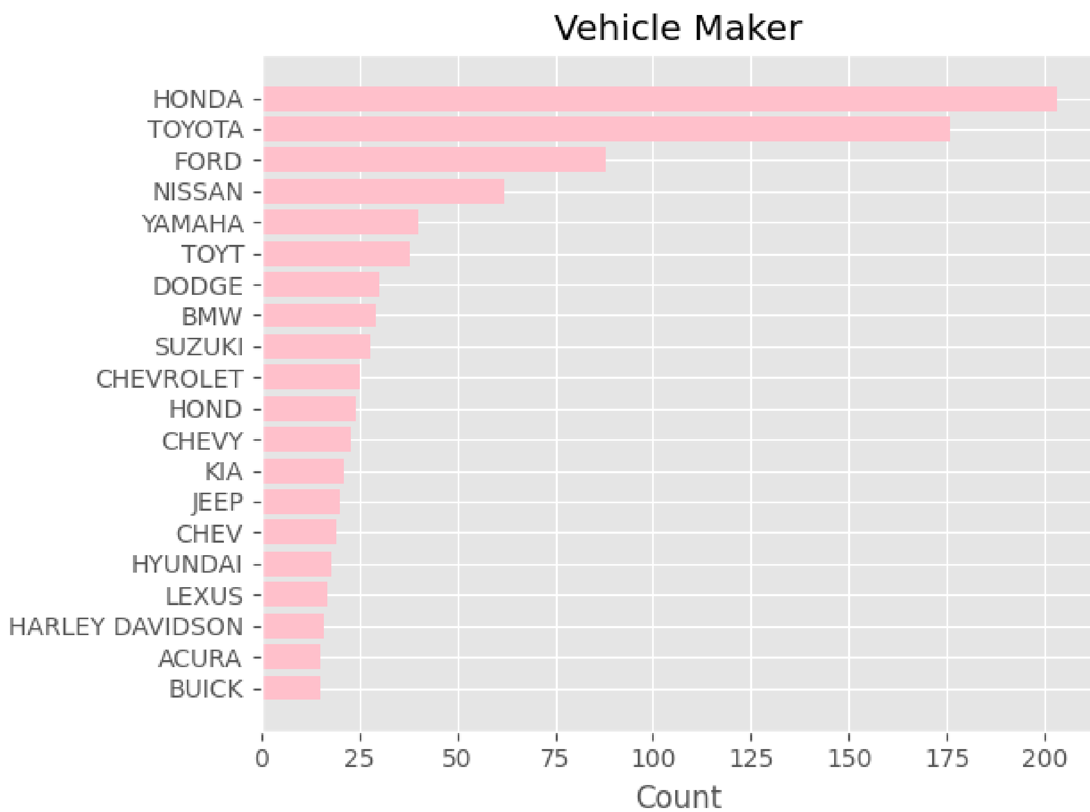
4. How do vehicle features relate to crash outcomes?
   Objective: Assess how attributes like vehicle make, model, year, and movement during a crash influence the extent of damage and outcomes. Insights can guide safety enhancements and risk assessments.

[34]:
```
# Vehicle Year
plt.style.use('ggplot')
vehicle_year = injury_df["Vehicle Year"]
bins = [1990, 1995, 2000, 2005, 2010, 2015, 2020, 2024]
plt.hist(vehicle_year, bins, edgecolor = 'black')
plt.show()
```



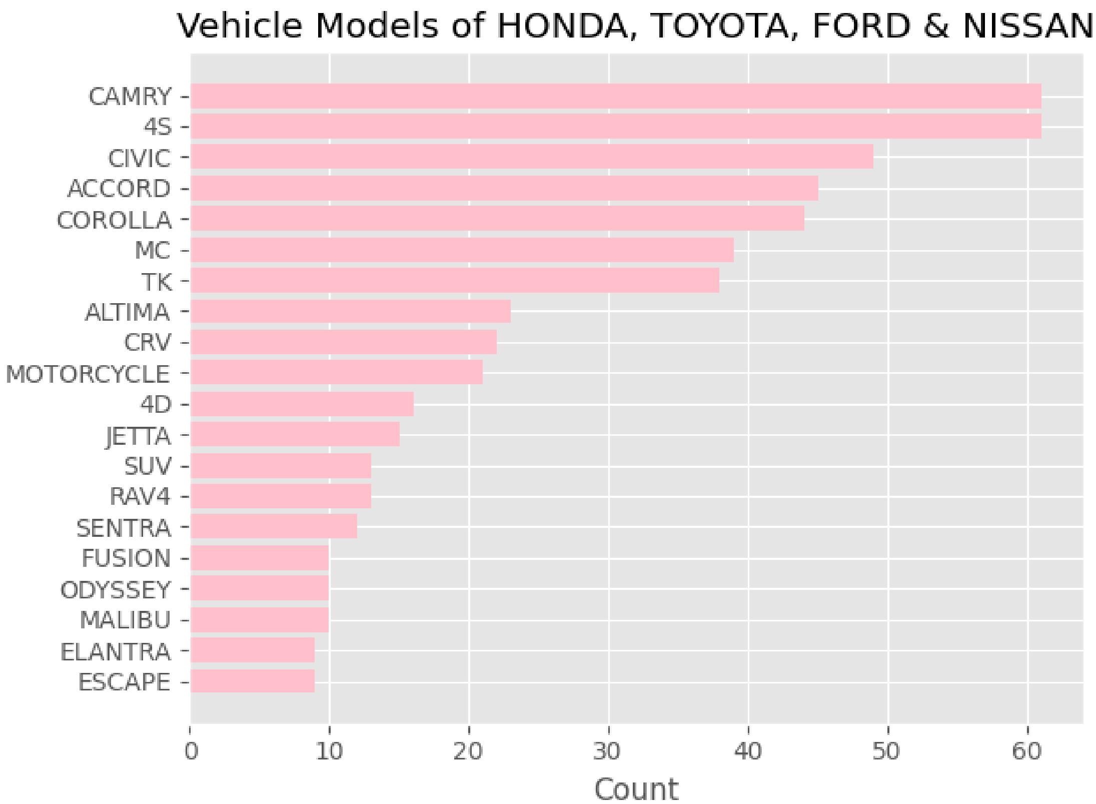
[35]:

```python
# vehicle maker
vehicle_maker = injury_df["Vehicle Make"]
vehicle_maker_counter = Counter()
# Split responses and update the Counter
for response in vehicle_maker.dropna():  # Skip NaN values
    vehicle_maker_counter.update(response.split(';'))  # Split by ';'

# Extract the top 10 most common substances
vehicle_maker_types = []
counts = []
for item in vehicle_maker_counter.most_common(20):
    vehicle_maker_types.append(item[0])
    counts.append(item[1])

# Reverse the lists for a better horizontal bar chart
vehicle_maker_types.reverse()
counts.reverse()

# Plot the horizontal bar chart
plt.style.use('ggplot')
plt.barh(vehicle_maker_types, counts, color='pink')
plt.xlabel("Count")
plt.title("Vehicle Maker")
plt.tight_layout()
plt.show()
```



[36]:
```python
# Vehicle Model
vehicle_model = injury_df["Vehicle Model"]
vehicle_model_counter = Counter()
# Split responses and update the Counter
for response in vehicle_model.dropna():  # Skip NaN values
    vehicle_model_counter.update(response.split(';'))  # Split by ';'

# Extract the top 10 most common substances
vehicle_model_types = []
counts = []
for item in vehicle_model_counter.most_common(20):
    vehicle_model_types.append(item[0])
    counts.append(item[1])

# Reverse the lists for a better horizontal bar chart
vehicle_model_types.reverse()
counts.reverse()

# Plot the horizontal bar chart
plt.style.use('ggplot')
plt.barh(vehicle_model_types, counts, color='pink')
```

```
plt.xlabel("Count")
plt.title("Vehicle Models of HONDA, TOYOTA, FORD & NISSAN")
plt.tight_layout()
plt.show()
```

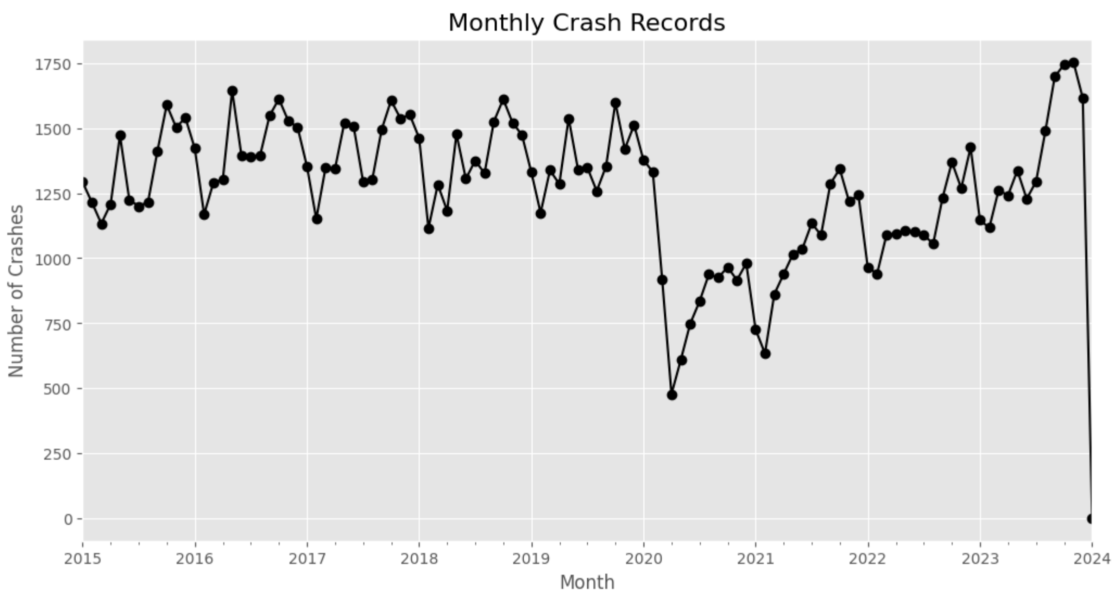


**Observations: From the analysis above, most likely vehicle types which could cause a crash are 4s, camry, civic, accord etc. And their makers are HONDA, TOYOTA, FORD and NISSAN.

[37]:
```
# there is a trend or sensonality in the data?
import pandas as pd
import matplotlib.pyplot as plt
# Convert 'Crash Date' to datetime
crash_data['Crash Date/Time'] = pd.to_datetime(crash_data['Crash Date/Time'])
# Extract year and month from 'Crash Date'
crash_data['Year-Month'] = crash_data['Crash Date/Time'].dt.to_period('M')

# Group by 'Year-Month' and count the number of crashes
monthly_crash_counts = crash_data.groupby('Year-Month').size()

# Plot the time series
plt.style.use('ggplot')
```

```
plt.figure(figsize=(12, 6))
monthly_crash_counts.plot(kind='line', marker='o', linestyle='-', color='k')
plt.title('Monthly Crash Records', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Number of Crashes', fontsize=12)
plt.grid(True)
plt.show()
```

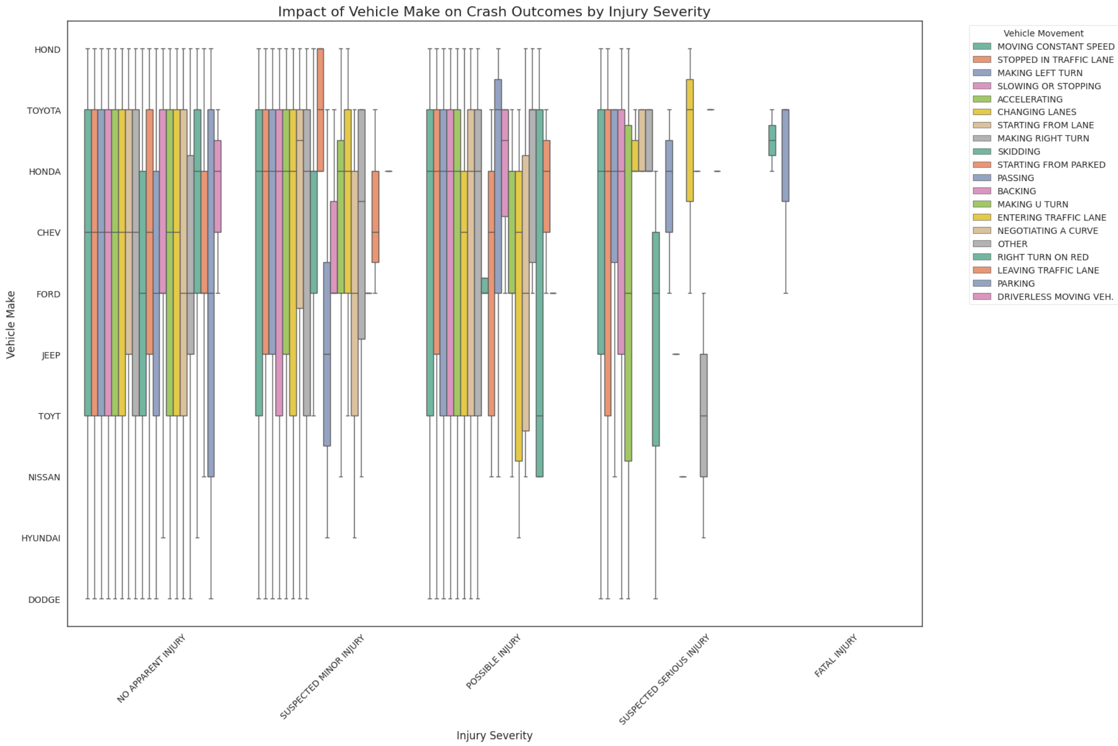
```
/tmp/ipykernel_329/648373190.py:5: UserWarning: Could not infer format, so each
element will be parsed individually, falling back to `dateutil`. To ensure
parsing is consistent and as-expected, please specify a format.
  crash_data['Crash Date/Time'] = pd.to_datetime(crash_data['Crash Date/Time'])
```



****Observations:** From the time series plot, we can conclude that there is an obvious seasonalitiy in the plot. During the pandemic, the number of car crashes happened in this region declined and trend for today is still below the average car crashed bafore pandemic. We can see there is a significant rise of car crashes in the first quarter every year. More measures to prevent car accidents need to be taken especially in Q1.

[51]:
```
# Set plot style to white background
sns.set_style("white")

# Filter related fields
vehicle_data = crash_data_filled[['Vehicle Make', 'Injury Severity', 'Vehicle␣
↪Movement']].dropna()

# Show the top 10 manufacturers
```

```
top_10_makes = vehicle_data['Vehicle Make'].value_counts().head(10).index
vehicle_data_filtered = vehicle_data[vehicle_data['Vehicle Make'].isin(top_10_makes)]

# Sort by severity
vehicle_data_filtered.loc[:, 'Injury Severity'] = pd.Categorical(
    vehicle_data_filtered['Injury Severity'],
    categories=['NO APPARENT INJURY', 'POSSIBLE INJURY', 'SUSPECTED MINOR INJURY', 'SUSPECTED SERIOUS INJURY', 'FATAL INJURY'],
    ordered=True
)

# Create drawing
plt.figure(figsize=(18, 12))
sns.boxplot(
    data=vehicle_data_filtered,
    x='Injury Severity',
    y='Vehicle Make',
    hue='Vehicle Movement',
    palette='Set2',
    showfliers=False  # Remove outliers
)

# Add title and tag
plt.title('Impact of Vehicle Make on Crash Outcomes by Injury Severity', fontsize=16)
plt.xlabel('Injury Severity', fontsize=12)
plt.ylabel('Vehicle Make', fontsize=12)
plt.xticks(rotation=45)
plt.legend(title='Vehicle Movement', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show
plt.tight_layout()
plt.show()
```



Observations:

Vehicle Manufacturer Variations: TOYOTA, FORD and HONDA are prominent across all injury severity levels, reflecting their widespread representation in the crash data. These manufacturers show consistent distributions for lower severity injuries, such as 'NO APPARENT INJURY' and 'POSSIBLE INJURY', while higher severity levels, such as 'SUSPECTED SERIOUS INJURY' and 'FATAL INJURY', show greater variability.

Vehicle Movement Patterns: Certain vehicle movements, such as "MAKING LEFT TURN" and "ACCELERATING," are strongly associated with severe outcomes, including higher injury levels. Conversely, movements like "SLOWING OR STOPPING" and "MOVING CONSTANT SPEED" are linked to safer outcomes, highlighting the role of movement type in crash severity.

Impact of Rare Vehicle Actions: Uncommon actions like "MAKING U TURN" or "BACKING" sometimes contribute to severe outcomes. These actions, although less frequent, warrant further investigation to understand the contexts in which they increase crash severity.

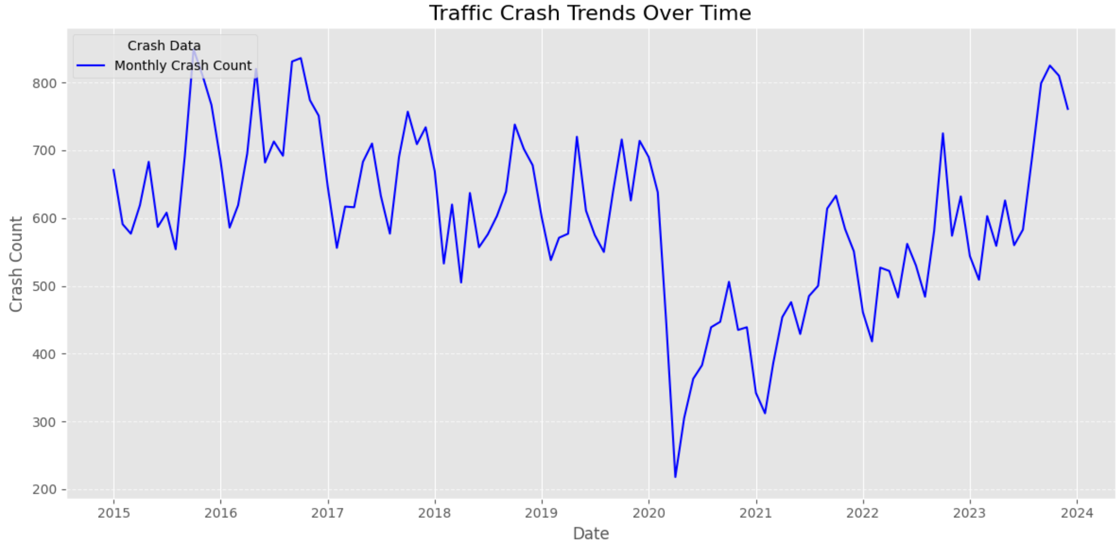
Manufacturer-specific Trends: Brands such as HYUNDAI and DODGE show a concentration in high-severity crashes, potentially tied to specific vehicle characteristics or associated driving patterns.

Injury Severity Insights: Lower severity injuries dominate across all manufacturers and categories. Severe injuries such as "FATAL INJURY" occur less frequently but show greater variability, suggesting a complex interplay between vehicle characteristics and crash outcomes.

5. What patterns exist in traffic crashes over time?
   Objective: Analyze temporal trends in crash occurrences, including variations by time of

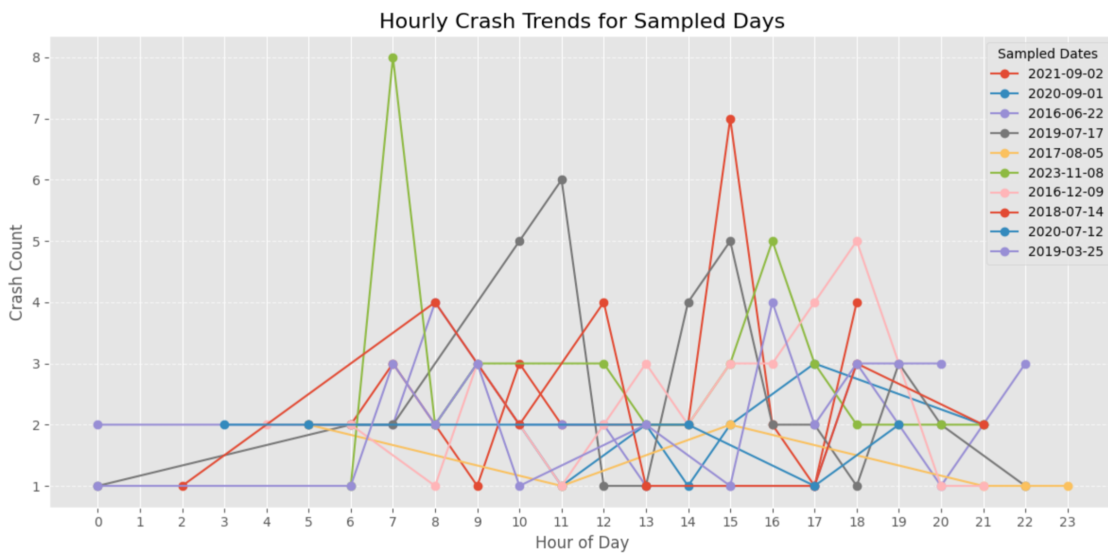
day, week, or year, to identify high-risk periods and support resource allocation for safety measures.

[39]:
```
# Ensure the Date column is in datetime format
crash_data_filled['Crash Date/Time'] = pd.to_datetime(crash_data_filled['Crash␣
↪Date/Time'])

# Aggregate data by date (e.g., monthly or weekly counts of crashes)
crash_data_filled['YearMonth'] = crash_data_filled['Crash Date/Time'].dt.
↪to_period('M')  # Monthly aggregation
monthly_crash_counts = crash_data_filled.groupby('YearMonth').size().
↪reset_index(name='Crash_Count')

# Convert YearMonth back to datetime for plotting
monthly_crash_counts['YearMonth'] = monthly_crash_counts['YearMonth'].dt.
↪to_timestamp()

# Plot the temporal trends
plt.figure(figsize=(12, 6))
plt.plot(monthly_crash_counts['YearMonth'],␣
↪monthly_crash_counts['Crash_Count'], label='Monthly Crash Count',␣
↪color='blue')

# Customize the plot
plt.title('Traffic Crash Trends Over Time', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Crash Count', fontsize=12)
plt.legend(title='Crash Data', loc='upper left', fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Show the plot
plt.show()
```



[40]:
```
import random

# Ensure the Crash Date/Time column is in datetime format
crash_data_filled['Crash Date/Time'] = pd.to_datetime(crash_data_filled['Crash␣
↪Date/Time'])

# Extract the date and hour for grouping
crash_data_filled['Date'] = crash_data_filled['Crash Date/Time'].dt.date
crash_data_filled['Hour'] = crash_data_filled['Crash Date/Time'].dt.hour

# Randomly sample a few dates
unique_dates = crash_data_filled['Date'].unique()
sampled_dates = random.sample(list(unique_dates), 10)  # Adjust the number of␣
↪sampled days here
sampled_data = crash_data_filled[crash_data_filled['Date'].isin(sampled_dates)]

# Group by date and hour to count crashes
hourly_crash_counts = sampled_data.groupby(['Date', 'Hour']).size().
↪reset_index(name='Crash_Count')

# Plot the trends
plt.figure(figsize=(12, 6))
for date in sampled_dates:
    day_data = hourly_crash_counts[hourly_crash_counts['Date'] == date]
    plt.plot(
        day_data['Hour'],
        day_data['Crash_Count'],
        marker='o',
```

```python
        label=str(date)
    )

# Customize the plot
plt.title('Hourly Crash Trends for Sampled Days', fontsize=16)
plt.xlabel('Hour of Day', fontsize=12)
plt.ylabel('Crash Count', fontsize=12)
plt.xticks(range(0, 24))  # Show all hours
plt.legend(title='Sampled Dates', loc='upper right', fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Show the plot
plt.show()
```



Observations:

The occurrence of traffic accidents exhibits a stable fluctuating trend over time, with a notable decline in 2020 potentially attributable to the reduction in traffic during the pandemic. Following this decline, the number of traffic accidents demonstrates a recovery and growth trend after 2021, which may be attributed to the resumption of normal traffic activity. These changes suggest that traffic accidents frequently align with broader social patterns, such as shifts in travel patterns or seasonal traffic.

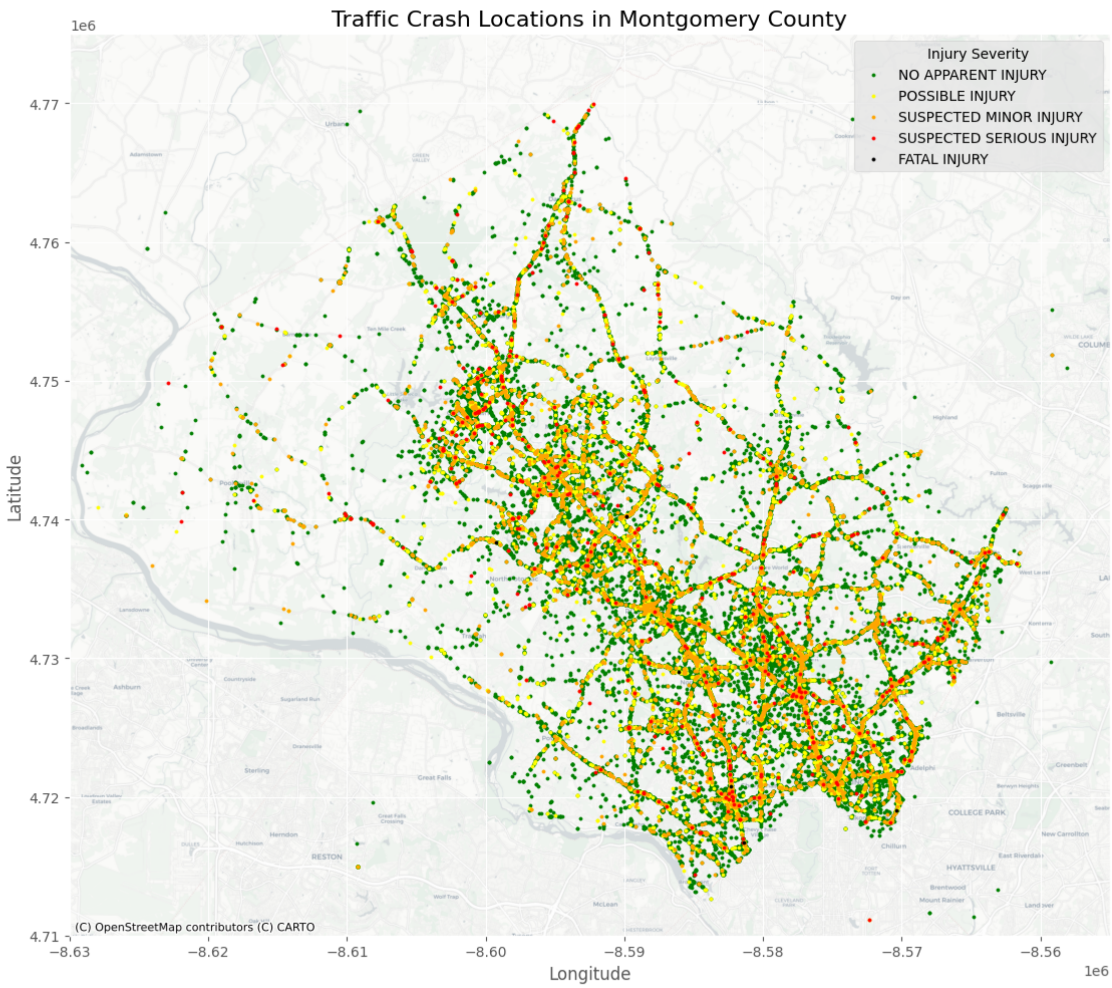
The hourly trends for the sample day demonstrate notable fluctuations in crash rates throughout the day on a random sample day. Peaks in the number of accidents occur in the early morning (between 7 a.m. and 9 a.m.) and late afternoon (between 4 p.m. and 6 p.m.), coinciding with typical rush hour periods. On certain days, there are unique spikes that may be caused by specific events or conditions, such as weather or other events. The number of accidents is lower at night (between 12 a.m. and 6 a.m.), possibly due to reduced traffic.

The combined analysis reveals discernible temporal patterns in crash occurrence, monthly trends reflecting broader societal changes, and daily trends associated with typical traffic flow patterns. This information underscores the necessity for targeted traffic safety measures during periods of elevated risk, such as rush hour and specific months of the year.

6. Where are traffic crashes most likely to occur?
   Objective: Explore geographic patterns of crashes and their association with factors like road types and speed limits. Findings can help identify high-risk areas and prioritize safety improvements.

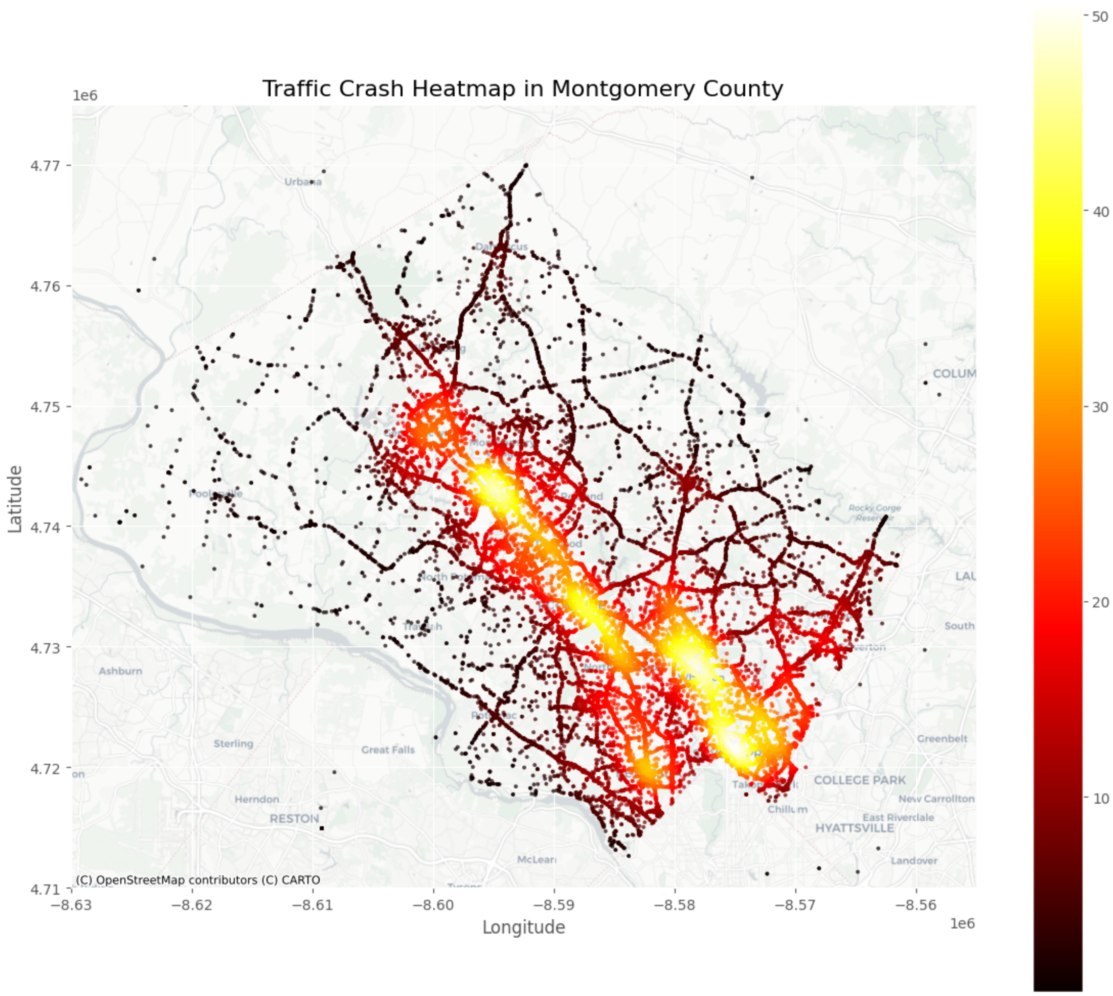
[46]:
```python
import geopandas as gpd
import contextily as ctx

# Ensure 'Latitude' and 'Longitude' columns exist in crash_data_filled
geo_data = crash_data_filled[['Latitude', 'Longitude', 'Speed Limit', 'Injury␣
 ↪Severity']].dropna()

# Convert to GeoDataFrame
gdf = gpd.GeoDataFrame(
    geo_data,
    geometry=gpd.points_from_xy(geo_data['Longitude'], geo_data['Latitude']),
    crs="EPSG:4326"  # WGS84 coordinate reference system
)

# Reproject GeoDataFrame to Web Mercator for overlaying basemap
gdf = gdf.to_crs(epsg=3857)

# Define Montgomery County bounding box (in Web Mercator)
montgomery_county_bounds = {
    "xmin": -8630000,  # Minimum X (Longitude)
    "xmax": -8555000,  # Maximum X (Longitude)
    "ymin": 4710000,   # Minimum Y (Latitude)
    "ymax": 4775000    # Maximum Y (Latitude)
}

# Create figure and axis
fig, ax = plt.subplots(figsize=(12, 10))

# Plot crash data points with a color-coded injury severity
severity_colors = {
    'NO APPARENT INJURY': 'green',
    'POSSIBLE INJURY': 'yellow',
    'SUSPECTED MINOR INJURY': 'orange',
    'SUSPECTED SERIOUS INJURY': 'red',
    'FATAL INJURY': 'black'
}
for severity, color in severity_colors.items():
    gdf[gdf['Injury Severity'] == severity].plot(
```

```
        ax=ax, color=color, markersize=5, label=severity
    )

# Set the map's focus to Montgomery County
ax.set_xlim(montgomery_county_bounds["xmin"], montgomery_county_bounds["xmax"])
ax.set_ylim(montgomery_county_bounds["ymin"], montgomery_county_bounds["ymax"])

# Add basemap
ctx.add_basemap(ax, source=ctx.providers.CartoDB.Positron, zoom=12)

# Customize plot appearance
ax.set_title("Traffic Crash Locations in Montgomery County", fontsize=16)
ax.set_xlabel("Longitude", fontsize=12)
ax.set_ylabel("Latitude", fontsize=12)
ax.legend(title="Injury Severity", loc='upper right', fontsize=10)
ax.set_axis_on()

# Adjust layout
plt.tight_layout()

# Save the figure
plt.savefig("traffic_crash_montgomery_county.png", dpi=300)

# Show plot
plt.show()
```



[49]:
```
from scipy.stats import gaussian_kde

# Filter valid latitude and longitude
geo_data = crash_data_filled[['Latitude', 'Longitude']].dropna()

# Convert to arrays for density calculation
latitudes = geo_data['Latitude'].values
longitudes = geo_data['Longitude'].values

# Calculate point density using Gaussian KDE
xy = np.vstack([longitudes, latitudes])
density = gaussian_kde(xy)(xy)

# Create a GeoDataFrame for plotting
geo_data['Density'] = density
gdf = gpd.GeoDataFrame(
```

```
    geo_data,
    geometry=gpd.points_from_xy(geo_data['Longitude'], geo_data['Latitude']),
    crs="EPSG:4326"
)

# Reproject GeoDataFrame to Web Mercator
gdf = gdf.to_crs(epsg=3857)

# Define Montgomery County bounding box (adjusted for the fixed region)
montgomery_county_bounds = {
    "xmin": -8630000,  # Minimum X (Longitude)
    "xmax": -8555000,  # Maximum X (Longitude)
    "ymin": 4710000,   # Minimum Y (Latitude)
    "ymax": 4775000    # Maximum Y (Latitude)
}

# Create the plot
fig, ax = plt.subplots(figsize=(12, 10))

# Scatter plot of density
gdf.plot(
    ax=ax,
    column='Density',
    cmap='hot',  # Heatmap color scheme
    markersize=5,
    alpha=0.7,
    legend=True
)

# Set the map's focus to Montgomery County's bounding box
ax.set_xlim(montgomery_county_bounds["xmin"], montgomery_county_bounds["xmax"])
ax.set_ylim(montgomery_county_bounds["ymin"], montgomery_county_bounds["ymax"])

# Add basemap for background using contextily
ctx.add_basemap(ax, source=ctx.providers.CartoDB.Positron, crs=gdf.crs.
 ↪to_string())

# Add title and labels
ax.set_title('Traffic Crash Heatmap in Montgomery County', fontsize=16)
ax.set_xlabel('Longitude', fontsize=12)
ax.set_ylabel('Latitude', fontsize=12)

# Save and display the plot
plt.tight_layout()
plt.savefig("traffic_crash_heatmap.png", dpi=300)
plt.show()
```



Observations:

The heat map illustrates that traffic accidents in Montgomery County are concentrated along major roadways, particularly highways and primary routes, which experience high traffic volumes and higher speed limits. These factors may contribute to an elevated frequency of accidents. The most intensely colored areas on the heat map indicate locations with a high concentration of accidents, such as the central and southern regions situated along major transportation routes. These areas exhibit a higher collision density, which may be attributed to factors such as intersections, high-speed zones, or regions with intricate traffic patterns.

The relatively low number of collisions on secondary roads and local streets suggests that the accident rate is significantly influenced by the type of road and the corresponding traffic flow. The findings underscore the necessity for targeted safety enhancements in high-risk areas, including enhanced traffic management, improved road infrastructure, and stricter speed limits in hotspot regions, with the aim of reducing the frequency of accidents.

1.1.6 Step 4: Meeting Schedule

We are all busy. You and your team should be working towards this project on a weekly basis. Share your proposed schedule below.

`--== Double-click and describe your project schedule below ==--`

Our project group started on **October 25, 2024**, and we are working towards the **poster day on December 6, 2024**. To ensure consistent progress, we will have a weekly meeting every Friday evening.

Weekly Meeting Details: - **Day**: Every Friday - **Time**: 8:00 PM - 9:00 PM - **Platform**: Zoom - **Agenda**: Discuss current progress, address any challenges, and assign tasks for the upcoming week. Each member will provide an update on their responsibilities.

Meeting Schedule Overview:

Week 1: October 25, 2024 - The project is initially launched. - Team members are assigned tasks and the next steps are planned.

Week 2: November 1, 2024 - Review data collection and preprocessing plans. - Assign responsibilities for data set preparation. - Identify questions to be explored. Write a project proposal.

Week 3: November 8, 2024 - Discuss the progress of data cleaning and preprocessing and identify any problems. - Finalize data cleaning strategy (e.g., handling missing values, outliers, and inconsistencies). - Perform exploratory data analysis after cleaning the data.

Week 4: November 15, 2024 - Present initial EDA findings, including descriptive statistics and preliminary visualizations. - Identify key patterns and potential features for deeper analysis.
- Analysis first two questions. - Write project progress reports

Week 5: November 22, 2024 - Analyze the remaining problems discuss findings related to severe crash factors, temporal trends, and geographic patterns.
- Continue refining visualizations and statistical analyses.
- Review and integrate all insights into a coherent narrative for the poster.
- Assign tasks for finalizing the poster content.

Week 6: November 29, 2024 - Review and finalize all visualizations, analyses, and text for the poster.
- Ensure the poster addresses all research questions and is ready for submission.
- Plan for a dry-run presentation to simulate poster day interactions.

Week 7: December 6, 2024 (Poster Day) - Conduct a final review of the poster and accompanying presentation materials.
- Prepare for audience questions and feedback.
- Ensure all deliverables are submitted on time.