Final_Project_Motorcycle_Colloision

December 17, 2024

[1]:
```
import pandas as pd
import numpy as np

# Load your dataset
data = pd.read_csv("/Users/tarunaverma/Downloads/
↪Motor_Vehicle_Collisions_-_Crashes.csv", low_memory=False)
```

#Lets have a look at our dataset

[2]:
```
data.head()
```

[2]:
```
   CRASH DATE CRASH TIME   BOROUGH ZIP CODE   LATITUDE  LONGITUDE  \
0  09/11/2021       2:39       NaN      NaN        NaN        NaN
1  03/26/2022      11:45       NaN      NaN        NaN        NaN
2  06/29/2022       6:55       NaN      NaN        NaN        NaN
3  09/11/2021       9:35  BROOKLYN    11208  40.667202 -73.866500
4  12/14/2021       8:13  BROOKLYN    11233  40.683304 -73.917274

                LOCATION           ON STREET NAME CROSS STREET NAME  \
0                    NaN    WHITESTONE EXPRESSWAY         20 AVENUE
1                    NaN  QUEENSBORO BRIDGE UPPER               NaN
2                    NaN       THROGS NECK BRIDGE               NaN
3    (40.667202, -73.8665)                      NaN               NaN
4  (40.683304, -73.917274)          SARATOGA AVENUE    DECATUR STREET

           OFF STREET NAME  …  CONTRIBUTING FACTOR VEHICLE 2  \
0                      NaN  …                    Unspecified
1                      NaN  …                            NaN
2                      NaN  …                    Unspecified
3  1211       LORING AVENUE  …                            NaN
4                      NaN  …                            NaN

   CONTRIBUTING FACTOR VEHICLE 3  CONTRIBUTING FACTOR VEHICLE 4  \
0                            NaN                            NaN
1                            NaN                            NaN
2                            NaN                            NaN
3                            NaN                            NaN
4                            NaN                            NaN
```

```
  CONTRIBUTING FACTOR VEHICLE 5  COLLISION_ID  VEHICLE TYPE CODE 1  \
0                           NaN       4455765                Sedan
1                           NaN       4513547                Sedan
2                           NaN       4541903                Sedan
3                           NaN       4456314                Sedan
4                           NaN       4486609                  NaN

  VEHICLE TYPE CODE 2  VEHICLE TYPE CODE 3 VEHICLE TYPE CODE 4  \
0               Sedan                  NaN                 NaN
1                 NaN                  NaN                 NaN
2        Pick-up Truck                 NaN                 NaN
3                 NaN                  NaN                 NaN
4                 NaN                  NaN                 NaN

  VEHICLE TYPE CODE 5
0                 NaN
1                 NaN
2                 NaN
3                 NaN
4                 NaN

[5 rows x 29 columns]
```

[3]:
```
# Understand the Structure of the Data
```

[4]:
```
# Get basic information about the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2131853 entries, 0 to 2131852
Data columns (total 29 columns):
 #   Column                         Dtype
---  ------                         -----
 0   CRASH DATE                     object
 1   CRASH TIME                     object
 2   BOROUGH                        object
 3   ZIP CODE                       object
 4   LATITUDE                       float64
 5   LONGITUDE                      float64
 6   LOCATION                       object
 7   ON STREET NAME                 object
 8   CROSS STREET NAME              object
 9   OFF STREET NAME                object
 10  NUMBER OF PERSONS INJURED      float64
 11  NUMBER OF PERSONS KILLED       float64
 12  NUMBER OF PEDESTRIANS INJURED  int64
 13  NUMBER OF PEDESTRIANS KILLED   int64
```

```
 14  NUMBER OF CYCLIST INJURED      int64
 15  NUMBER OF CYCLIST KILLED       int64
 16  NUMBER OF MOTORIST INJURED     int64
 17  NUMBER OF MOTORIST KILLED      int64
 18  CONTRIBUTING FACTOR VEHICLE 1  object
 19  CONTRIBUTING FACTOR VEHICLE 2  object
 20  CONTRIBUTING FACTOR VEHICLE 3  object
 21  CONTRIBUTING FACTOR VEHICLE 4  object
 22  CONTRIBUTING FACTOR VEHICLE 5  object
 23  COLLISION_ID                   int64
 24  VEHICLE TYPE CODE 1            object
 25  VEHICLE TYPE CODE 2            object
 26  VEHICLE TYPE CODE 3            object
 27  VEHICLE TYPE CODE 4            object
 28  VEHICLE TYPE CODE 5            object
dtypes: float64(4), int64(7), object(18)
memory usage: 471.7+ MB
```

[5]:
```
list(data.columns)
```

[5]:
```
['CRASH DATE',
 'CRASH TIME',
 'BOROUGH',
 'ZIP CODE',
 'LATITUDE',
 'LONGITUDE',
 'LOCATION',
 'ON STREET NAME',
 'CROSS STREET NAME',
 'OFF STREET NAME',
 'NUMBER OF PERSONS INJURED',
 'NUMBER OF PERSONS KILLED',
 'NUMBER OF PEDESTRIANS INJURED',
 'NUMBER OF PEDESTRIANS KILLED',
 'NUMBER OF CYCLIST INJURED',
 'NUMBER OF CYCLIST KILLED',
 'NUMBER OF MOTORIST INJURED',
 'NUMBER OF MOTORIST KILLED',
 'CONTRIBUTING FACTOR VEHICLE 1',
 'CONTRIBUTING FACTOR VEHICLE 2',
 'CONTRIBUTING FACTOR VEHICLE 3',
 'CONTRIBUTING FACTOR VEHICLE 4',
 'CONTRIBUTING FACTOR VEHICLE 5',
 'COLLISION_ID',
 'VEHICLE TYPE CODE 1',
 'VEHICLE TYPE CODE 2',
 'VEHICLE TYPE CODE 3',
```

```
 'VEHICLE TYPE CODE 4',
 'VEHICLE TYPE CODE 5']
```

[6]:
```
pd.set_option('display.max_columns', None) # This allows us to view all columns␣
 ↪in a dataframe when called
pd.set_option('display.max_rows', 200) # This returns 200 rows at max to␣
 ↪prevent accidents when writing code
data.head()
```

[6]:
```
   CRASH DATE CRASH TIME   BOROUGH ZIP CODE   LATITUDE  LONGITUDE  \
0  09/11/2021       2:39       NaN      NaN        NaN        NaN
1  03/26/2022      11:45       NaN      NaN        NaN        NaN
2  06/29/2022       6:55       NaN      NaN        NaN        NaN
3  09/11/2021       9:35  BROOKLYN    11208  40.667202 -73.866500
4  12/14/2021       8:13  BROOKLYN    11233  40.683304 -73.917274

                  LOCATION           ON STREET NAME CROSS STREET NAME  \
0                      NaN    WHITESTONE EXPRESSWAY         20 AVENUE
1                      NaN  QUEENSBORO BRIDGE UPPER               NaN
2                      NaN       THROGS NECK BRIDGE               NaN
3    (40.667202, -73.8665)                      NaN               NaN
4  (40.683304, -73.917274)          SARATOGA AVENUE    DECATUR STREET

           OFF STREET NAME  NUMBER OF PERSONS INJURED  \
0                      NaN                        2.0
1                      NaN                        1.0
2                      NaN                        0.0
3  1211       LORING AVENUE                        0.0
4                      NaN                        0.0

   NUMBER OF PERSONS KILLED  NUMBER OF PEDESTRIANS INJURED  \
0                       0.0                              0
1                       0.0                              0
2                       0.0                              0
3                       0.0                              0
4                       0.0                              0

   NUMBER OF PEDESTRIANS KILLED  NUMBER OF CYCLIST INJURED  \
0                             0                          0
1                             0                          0
2                             0                          0
3                             0                          0
4                             0                          0

   NUMBER OF CYCLIST KILLED  NUMBER OF MOTORIST INJURED  \
0                         0                           2
1                         0                           1
```

```
2                        0                           0
3                        0                           0
4                        0                           0

   NUMBER OF MOTORIST KILLED CONTRIBUTING FACTOR VEHICLE 1  \
0                          0  Aggressive Driving/Road Rage
1                          0             Pavement Slippery
2                          0          Following Too Closely
3                          0                    Unspecified
4                          0                            NaN

  CONTRIBUTING FACTOR VEHICLE 2 CONTRIBUTING FACTOR VEHICLE 3  \
0                   Unspecified                           NaN
1                           NaN                           NaN
2                   Unspecified                           NaN
3                           NaN                           NaN
4                           NaN                           NaN

  CONTRIBUTING FACTOR VEHICLE 4 CONTRIBUTING FACTOR VEHICLE 5  COLLISION_ID  \
0                           NaN                           NaN       4455765
1                           NaN                           NaN       4513547
2                           NaN                           NaN       4541903
3                           NaN                           NaN       4456314
4                           NaN                           NaN       4486609

  VEHICLE TYPE CODE 1 VEHICLE TYPE CODE 2 VEHICLE TYPE CODE 3  \
0               Sedan               Sedan                 NaN
1               Sedan                 NaN                 NaN
2               Sedan        Pick-up Truck                NaN
3               Sedan                 NaN                 NaN
4                 NaN                 NaN                 NaN

  VEHICLE TYPE CODE 4 VEHICLE TYPE CODE 5
0                 NaN                 NaN
1                 NaN                 NaN
2                 NaN                 NaN
3                 NaN                 NaN
4                 NaN                 NaN
```

[7]:
```
missing_values = data.isnull().sum()
missing_percentage = (missing_values / len(data)) * 100
print("\nMissing Values Percentage:")
print(missing_percentage)
```

```
Missing Values Percentage:
CRASH DATE                      0.000000
CRASH TIME                      0.000000
```

```
BOROUGH                          31.086853
ZIP CODE                         31.099095
LATITUDE                         11.227369
LONGITUDE                        11.227369
LOCATION                         11.227369
ON STREET NAME                   21.426477
CROSS STREET NAME                38.126128
OFF STREET NAME                  82.906514
NUMBER OF PERSONS INJURED         0.000844
NUMBER OF PERSONS KILLED          0.001454
NUMBER OF PEDESTRIANS INJURED     0.000000
NUMBER OF PEDESTRIANS KILLED      0.000000
NUMBER OF CYCLIST INJURED         0.000000
NUMBER OF CYCLIST KILLED          0.000000
NUMBER OF MOTORIST INJURED        0.000000
NUMBER OF MOTORIST KILLED         0.000000
CONTRIBUTING FACTOR VEHICLE 1     0.336843
CONTRIBUTING FACTOR VEHICLE 2    15.693108
CONTRIBUTING FACTOR VEHICLE 3    92.808697
CONTRIBUTING FACTOR VEHICLE 4    98.365929
CONTRIBUTING FACTOR VEHICLE 5    99.555082
COLLISION_ID                      0.000000
VEHICLE TYPE CODE 1               0.683631
VEHICLE TYPE CODE 2              19.473857
VEHICLE TYPE CODE 3              93.077712
VEHICLE TYPE CODE 4              98.424141
VEHICLE TYPE CODE 5              99.568826
dtype: float64
```

1 Transforming the Data

Handling Missing values:

Columns with High Missing Values (Over 50%) OFF STREET NAME (82.91%) CONTRIBUTING FACTOR VEHICLE 3-5 (92.81%, 98.37%, 99.56%) VEHICLE TYPE CODE 3-5 (93.08%, 98.42%, 99.57%)

Drop Columns: These columns have a very high percentage of missing data, indicating limited usefulness for analysis. Dropping them may be the best option unless specific patterns are needed.

Some columns (such as VEHICLE TYPE CODE 3-5 (93.08%, 98.42%, 99.57%), OFF STREET NAME (82.91%),CONTRIBUTING FACTOR VEHICLE 3-5 (92.81%, 98.37%, 99.56%)) are nearly entirely empty. We'll remove those.

We will not be using some columns (e.g. collision_id, on_street_name, off_street_name, cross_street_name) so we can drop them completely.

[8]:
```
# Impute Missing Data for Essential Columns
```

```python
# For 'BOROUGH', fill missing values with 'Unknown'
data['BOROUGH'].fillna('Unknown', inplace=True)
```

Impute Missing Values: For essential columns (BOROUGH, ZIP CODE, contributing factors, and vehicle types), we impute missing values to retain information and make the dataset more complete.

[9]:
```python
# Alternatively, if using geolocation is feasible, you could impute based on␣
↪latitude/longitude
if 'ZIP CODE' in data.columns:
    data['ZIP CODE'].fillna(data['ZIP CODE'].mode()[0], inplace=True)

# Fill missing values in contributing factors and vehicle types with 'Unknown'
data['CONTRIBUTING FACTOR VEHICLE 1'].fillna('Unknown', inplace=True)
data['CONTRIBUTING FACTOR VEHICLE 2'].fillna('Unknown', inplace=True)
data['VEHICLE TYPE CODE 1'].fillna('Unknown', inplace=True)
data['VEHICLE TYPE CODE 2'].fillna('Unknown', inplace=True)
```

[10]:
```python
# Remove Non-Essential Columns with High Missing Values or Irrelevant␣
↪Information
# Define the list of non-essential columns to drop
columns_to_drop = [
    'COLLISION_ID',
    'ON STREET NAME',
    'OFF STREET NAME',
    'CROSS STREET NAME',
    'VEHICLE TYPE CODE 3',
    'VEHICLE TYPE CODE 4',
    'VEHICLE TYPE CODE 5',
    'CONTRIBUTING FACTOR VEHICLE 3',
    'CONTRIBUTING FACTOR VEHICLE 4',
    'CONTRIBUTING FACTOR VEHICLE 5'
]

# Drop these columns if they exist in the dataset
data = data.drop(columns=[col for col in columns_to_drop if col in data.
↪columns])

# Step 3: Rename Columns for Consistency and Readability
data = data.rename(columns={
    'CRASH DATE': 'Date',
    'CRASH TIME': 'Time',
    'BOROUGH': 'Borough',
    'ZIP CODE': 'ZipCode',
    'NUMBER OF PERSONS INJURED': 'Persons_Injured',
    'NUMBER OF PERSONS KILLED': 'Persons_Killed',
    'NUMBER OF PEDESTRIANS INJURED': 'Pedestrians_Injured',
    'NUMBER OF PEDESTRIANS KILLED': 'Pedestrians_Killed',
```

```
    'NUMBER OF CYCLIST INJURED': 'Cyclists_Injured',
    'NUMBER OF CYCLIST KILLED': 'Cyclists_Killed',
    'NUMBER OF MOTORIST INJURED': 'Motorists_Injured',
    'NUMBER OF MOTORIST KILLED': 'Motorists_Killed',
    'CONTRIBUTING FACTOR VEHICLE 1': 'Contributing_Factor_1',
    'CONTRIBUTING FACTOR VEHICLE 2': 'Contributing_Factor_2',
    'VEHICLE TYPE CODE 1': 'Vehicle_Type_1',
    'VEHICLE TYPE CODE 2': 'Vehicle_Type_2'
})
```

[11]:
```
# Final Check and Save Cleaned Data
# Display the first few rows of the modified dataset to confirm changes
print("Dataset after transformation:")
print(data.head())
```

```
Dataset after transformation:
         Date   Time   Borough ZipCode   LATITUDE  LONGITUDE  \
0  09/11/2021   2:39   Unknown   11207        NaN        NaN
1  03/26/2022  11:45   Unknown   11207        NaN        NaN
2  06/29/2022   6:55   Unknown   11207        NaN        NaN
3  09/11/2021   9:35  BROOKLYN   11208  40.667202 -73.866500
4  12/14/2021   8:13  BROOKLYN   11233  40.683304 -73.917274

                  LOCATION  Persons_Injured  Persons_Killed  \
0                      NaN              2.0             0.0
1                      NaN              1.0             0.0
2                      NaN              0.0             0.0
3    (40.667202, -73.8665)              0.0             0.0
4  (40.683304, -73.917274)              0.0             0.0

   Pedestrians_Injured  Pedestrians_Killed  Cyclists_Injured  Cyclists_Killed  \
0                    0                   0                 0                0
1                    0                   0                 0                0
2                    0                   0                 0                0
3                    0                   0                 0                0
4                    0                   0                 0                0

   Motorists_Injured  Motorists_Killed         Contributing_Factor_1  \
0                  2                 0  Aggressive Driving/Road Rage
1                  1                 0             Pavement Slippery
2                  0                 0         Following Too Closely
3                  0                 0                   Unspecified
4                  0                 0                       Unknown

  Contributing_Factor_2 Vehicle_Type_1 Vehicle_Type_2
0           Unspecified          Sedan          Sedan
1               Unknown          Sedan        Unknown
2           Unspecified          Sedan  Pick-up Truck
```

```
3               Unknown          Sedan        Unknown
4               Unknown        Unknown        Unknown
```

[12]:
```
# Save the cleaned dataset to a new file
cleaned_file_path = "Cleaned_NYC_Collision_Data_Transformed.csv"
data.to_csv(cleaned_file_path, index=False)
print("Cleaned and transformed dataset saved as␣
↪'Cleaned_NYC_Collision_Data_Transformed.csv'")
```

```
Cleaned and transformed dataset saved as
'Cleaned_NYC_Collision_Data_Transformed.csv'
```

[13]:
```
data.head()
```

[13]:
```
         Date   Time   Borough ZipCode   LATITUDE  LONGITUDE  \
0  09/11/2021   2:39   Unknown   11207        NaN        NaN
1  03/26/2022  11:45   Unknown   11207        NaN        NaN
2  06/29/2022   6:55   Unknown   11207        NaN        NaN
3  09/11/2021   9:35  BROOKLYN   11208  40.667202 -73.866500
4  12/14/2021   8:13  BROOKLYN   11233  40.683304 -73.917274

                  LOCATION  Persons_Injured  Persons_Killed  \
0                      NaN              2.0             0.0
1                      NaN              1.0             0.0
2                      NaN              0.0             0.0
3     (40.667202, -73.8665)             0.0             0.0
4  (40.683304, -73.917274)              0.0             0.0

   Pedestrians_Injured  Pedestrians_Killed  Cyclists_Injured  Cyclists_Killed  \
0                    0                   0                 0                0
1                    0                   0                 0                0
2                    0                   0                 0                0
3                    0                   0                 0                0
4                    0                   0                 0                0

   Motorists_Injured  Motorists_Killed         Contributing_Factor_1  \
0                  2                 0  Aggressive Driving/Road Rage
1                  1                 0             Pavement Slippery
2                  0                 0         Following Too Closely
3                  0                 0                   Unspecified
4                  0                 0                       Unknown

  Contributing_Factor_2 Vehicle_Type_1 Vehicle_Type_2
0           Unspecified          Sedan          Sedan
1               Unknown          Sedan        Unknown
2           Unspecified          Sedan  Pick-up Truck
3               Unknown          Sedan        Unknown
4               Unknown        Unknown        Unknown
```

[14]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2131853 entries, 0 to 2131852
Data columns (total 19 columns):
 #   Column                 Dtype
---  ------                 -----
 0   Date                   object
 1   Time                   object
 2   Borough                object
 3   ZipCode                object
 4   LATITUDE               float64
 5   LONGITUDE              float64
 6   LOCATION               object
 7   Persons_Injured        float64
 8   Persons_Killed         float64
 9   Pedestrians_Injured    int64
 10  Pedestrians_Killed     int64
 11  Cyclists_Injured       int64
 12  Cyclists_Killed        int64
 13  Motorists_Injured      int64
 14  Motorists_Killed       int64
 15  Contributing_Factor_1  object
 16  Contributing_Factor_2  object
 17  Vehicle_Type_1         object
 18  Vehicle_Type_2         object
dtypes: float64(4), int64(6), object(9)
memory usage: 309.0+ MB
```

[15]:
```python
# Check for remaining missing values in each column
missing_values_after = data.isnull().sum()
missing_percentage_after = (missing_values_after / len(data)) * 100
print("Remaining Missing Values Percentage:")
print(missing_percentage_after)
```

```
Remaining Missing Values Percentage:
Date                    0.000000
Time                    0.000000
Borough                 0.000000
ZipCode                 0.000000
LATITUDE               11.227369
LONGITUDE              11.227369
LOCATION               11.227369
Persons_Injured         0.000844
Persons_Killed          0.001454
Pedestrians_Injured     0.000000
Pedestrians_Killed      0.000000
Cyclists_Injured        0.000000
```

```
Cyclists_Killed            0.000000
Motorists_Injured          0.000000
Motorists_Killed           0.000000
Contributing_Factor_1      0.000000
Contributing_Factor_2      0.000000
Vehicle_Type_1             0.000000
Vehicle_Type_2             0.000000
dtype: float64
```

[16]:
```
# Drop rows where LATITUDE, LONGITUDE, or LOCATION are missing
data = data.dropna(subset=['LATITUDE', 'LONGITUDE', 'LOCATION'])

# Check the shape of the dataset after dropping rows with missing location data
print("Dataset shape after dropping rows with missing LATITUDE, LONGITUDE, or␣
 ↪LOCATION:", data.shape)

# Verify if there are any remaining missing values in LATITUDE, LONGITUDE, or␣
 ↪LOCATION
print("Remaining missing values in location-related columns:")
print(data[['LATITUDE', 'LONGITUDE', 'LOCATION']].isnull().sum())
```

```
Dataset shape after dropping rows with missing LATITUDE, LONGITUDE, or LOCATION:
(1892502, 19)
Remaining missing values in location-related columns:
LATITUDE     0
LONGITUDE    0
LOCATION     0
dtype: int64
```

[17]:
```
# Check for remaining missing values in each column
missing_values_after = data.isnull().sum()
missing_percentage_after = (missing_values_after / len(data)) * 100
print("Remaining Missing Values Percentage:")
print(missing_percentage_after)
```

```
Remaining Missing Values Percentage:
Date                     0.000000
Time                     0.000000
Borough                  0.000000
ZipCode                  0.000000
LATITUDE                 0.000000
LONGITUDE                0.000000
LOCATION                 0.000000
Persons_Injured          0.000845
Persons_Killed           0.001480
Pedestrians_Injured      0.000000
Pedestrians_Killed       0.000000
Cyclists_Injured         0.000000
```

```
Cyclists_Killed          0.000000
Motorists_Injured        0.000000
Motorists_Killed         0.000000
Contributing_Factor_1    0.000000
Contributing_Factor_2    0.000000
Vehicle_Type_1           0.000000
Vehicle_Type_2           0.000000
dtype: float64
```

[18]:
```python
#Verify Columns Have Been Renamed Correctly
# Display the list of column names to verify renaming
print("Columns after renaming:")
print(data.columns)
```

```
Columns after renaming:
Index(['Date', 'Time', 'Borough', 'ZipCode', 'LATITUDE', 'LONGITUDE',
       'LOCATION', 'Persons_Injured', 'Persons_Killed', 'Pedestrians_Injured',
       'Pedestrians_Killed', 'Cyclists_Injured', 'Cyclists_Killed',
       'Motorists_Injured', 'Motorists_Killed', 'Contributing_Factor_1',
       'Contributing_Factor_2', 'Vehicle_Type_1', 'Vehicle_Type_2'],
      dtype='object')
```

[19]:
```python
# List of columns that should have been dropped
columns_expected_to_drop = [
    'COLLISION_ID', 'ON STREET NAME', 'OFF STREET NAME', 'CROSS STREET NAME',
    'VEHICLE TYPE CODE 3', 'VEHICLE TYPE CODE 4', 'VEHICLE TYPE CODE 5',
    'CONTRIBUTING FACTOR VEHICLE 3', 'CONTRIBUTING FACTOR VEHICLE 4',␣
 ↪'CONTRIBUTING FACTOR VEHICLE 5'
]

# Check if any of these columns are still present
remaining_columns = [col for col in columns_expected_to_drop if col in data.
 ↪columns]
if remaining_columns:
    print("Columns that were not dropped:", remaining_columns)
else:
    print("All specified columns have been successfully removed.")
```

```
All specified columns have been successfully removed.
```

[20]:
```python
# Check unique values in key columns to confirm imputation
print("Unique values in 'Borough':", data['Borough'].unique())
print("Unique values in 'Contributing_Factor_1':",␣
 ↪data['Contributing_Factor_1'].unique())
print("Unique values in 'Vehicle_Type_1':", data['Vehicle_Type_1'].unique())
```

```
Unique values in 'Borough': ['BROOKLYN' 'Unknown' 'BRONX' 'MANHATTAN' 'QUEENS'
'STATEN ISLAND']
Unique values in 'Contributing_Factor_1': ['Unspecified' 'Unknown' 'Passing Too
```

```
Closely' 'Driver Inexperience'
 'Passing or Lane Usage Improper' 'Turning Improperly' 'Unsafe Speed'
 'Reaction to Uninvolved Vehicle' 'Steering Failure'
 'Following Too Closely' 'Other Vehicular'
 'Driver Inattention/Distraction' 'Oversized Vehicle'
 'Traffic Control Disregarded' 'Unsafe Lane Changing'
 'Alcohol Involvement' 'View Obstructed/Limited'
 'Failure to Yield Right-of-Way' 'Aggressive Driving/Road Rage'
 'Pavement Slippery' 'Illnes' 'Lost Consciousness' 'Brakes Defective'
 'Backing Unsafely' 'Passenger Distraction' 'Fell Asleep'
 'Pedestrian/Bicyclist/Other Pedestrian Error/Confusion'
 'Obstruction/Debris' 'Tinted Windows' 'Animals Action' 'Drugs (illegal)'
 'Pavement Defective' 'Other Lighting Defects' 'Outside Car Distraction'
 'Driverless/Runaway Vehicle' 'Tire Failure/Inadequate' 'Fatigued/Drowsy'
 'Headlights Defective' 'Accelerator Defective' 'Physical Disability'
 'Glare' 'Eating or Drinking' 'Failure to Keep Right'
 'Cell Phone (hands-free)' 'Lane Marking Improper/Inadequate'
 'Cell Phone (hand-Held)' 'Using On Board Navigation Device'
 'Other Electronic Device' 'Tow Hitch Defective' 'Windshield Inadequate'
 'Vehicle Vandalism' 'Prescription Medication'
 'Shoulders Defective/Improper' 'Listening/Using Headphones'
 'Traffic Control Device Improper/Non-Working' 'Texting'
 'Reaction to Other Uninvolved Vehicle' '80' '1' 'Drugs (Illegal)'
 'Illness' 'Cell Phone (hand-held)']
Unique values in 'Vehicle_Type_1': ['Sedan' 'Unknown' 'Station Wagon/Sport
Utility Vehicle' … '0000'
 'Mixer' 'RAZOR SCOO']
```

[21]:
```python
# Check for duplicate rows
duplicate_rows = data[data.duplicated()]
print(f"Number of duplicate rows: {len(duplicate_rows)}")
```

```
Number of duplicate rows: 1579
```

It's a good idea to remove these duplicates to ensure the integrity of the data.

[22]:
```python
# Remove duplicate rows
data = data.drop_duplicates()

# Verify if duplicates have been removed
duplicate_rows_after = data[data.duplicated()]
print(f"Number of duplicate rows after removal: {len(duplicate_rows_after)}")

# Check the new shape of the dataset
print(f"Dataset shape after removing duplicates: {data.shape}")
```

```
Number of duplicate rows after removal: 0
Dataset shape after removing duplicates: (1890923, 19)
```

[23]:
```python
#Let's take a closer look at vehicle_type_code_1.
data['Vehicle_Type_1'].value_counts().head(40)
```

[23]:
```
Vehicle_Type_1
Sedan                                  556674
Station Wagon/Sport Utility Vehicle    432636
PASSENGER VEHICLE                      346704
SPORT UTILITY / STATION WAGON          150745
Taxi                                    49029
Pick-up Truck                           32867
TAXI                                    29357
4 dr sedan                              28573
Box Truck                               23201
VAN                                     21985
Bus                                     20780
OTHER                                   19554
UNKNOWN                                 17182
Bike                                    14402
Unknown                                 13320
BUS                                     12291
SMALL COM VEH(4 TIRES)                  11456
LARGE COM VEH(6 OR MORE TIRES)          11420
PICK-UP TRUCK                            9640
Tractor Truck Diesel                     9447
LIVERY VEHICLE                           9095
Van                                      8442
Motorcycle                               7975
Ambulance                                4470
Dump                                     3622
MOTORCYCLE                               3599
Convertible                              3537
E-Bike                                   3421
PK                                       2400
AMBULANCE                                2366
E-Scooter                                2289
Flat Bed                                 2246
Garbage or Refuse                        2233
Moped                                    2169
2 dr sedan                               1869
Carry All                                1787
Tractor Truck Gasoline                   1428
Tow Truck / Wrecker                      1257
Chassis Cab                               837
FIRE TRUCK                                778
Name: count, dtype: int64
```

[24]:
```python
# Borough-wise collision counts
print("Collision Counts by Borough:")
print(data['Borough'].value_counts())

# Most common contributing factors
print("Top Contributing Factors:")
print(data['Contributing_Factor_1'].value_counts().head(10))
```

```
Collision Counts by Borough:
Borough
Unknown          460334
BROOKLYN         457783
QUEENS           385240
MANHATTAN        317323
BRONX            210333
STATEN ISLAND     59910
Name: count, dtype: int64
Top Contributing Factors:
Contributing_Factor_1
Unspecified                       635278
Driver Inattention/Distraction    387016
Failure to Yield Right-of-Way     115939
Following Too Closely              97964
Backing Unsafely                   71790
Other Vehicular                    58758
Passing or Lane Usage Improper     54327
Passing Too Closely                50215
Turning Improperly                 45602
Fatigued/Drowsy                    37728
Name: count, dtype: int64
```

Data Type verification: Check if all columns have appropriate data types (e.g., numerical, categorical, datetime).

Misaligned data types can cause issues during analysis or modeling.

Specifically: Convert Date and Time to datetime objects. Ensure numerical columns (e.g., Persons_Injured) are numeric.

[25]:
```python
# Convert Time column to datetime.time format, handling inconsistent formats
data['Time'] = pd.to_datetime(data['Time'], format='%H:%M', errors='coerce').dt.
 ↪time

# Check for missing values introduced by invalid time formats
print(f"Number of missing values in 'time' after conversion: {data['Time'].
 ↪isnull().sum()}")
```

```
Number of missing values in 'time' after conversion: 0
```

[26]:
```
# Verify the Time column
print("Sample of the 'time' column after processing:")
print(data['Time'].head())
```

```
Sample of the 'time' column after processing:
3    09:35:00
4    08:13:00
6    17:05:00
7    08:17:00
8    21:10:00
Name: Time, dtype: object
```

Lets rename LATITUDE', 'LONGITUDE', 'LOCATION'too

[27]:
```
# Rename 'LATITUDE', 'LONGITUDE', and 'LOCATION' columns
data = data.rename(columns={
    'LATITUDE': 'Latitude',
    'LONGITUDE': 'Longitude',
    'LOCATION': 'Geo_Location'  # Change 'LOCATION' to 'Geo_Location' or any␣
 ↪preferred name
})

# Verify the column names after renaming
print("Updated column names:")
print(data.columns)
```

```
Updated column names:
Index(['Date', 'Time', 'Borough', 'ZipCode', 'Latitude', 'Longitude',
       'Geo_Location', 'Persons_Injured', 'Persons_Killed',
       'Pedestrians_Injured', 'Pedestrians_Killed', 'Cyclists_Injured',
       'Cyclists_Killed', 'Motorists_Injured', 'Motorists_Killed',
       'Contributing_Factor_1', 'Contributing_Factor_2', 'Vehicle_Type_1',
       'Vehicle_Type_2'],
      dtype='object')
```

[28]:
```
#Lets look at a few numbers
```

[29]:
```
data['Persons_Injured'].sum()
```

[29]: 602879.0

[30]:
```
data['Persons_Killed'].sum()
```

[30]: 2840.0

[31]:
```
data['Pedestrians_Injured'].sum()
```

[31]: 113994

[32]:
```
data['Cyclists_Injured'].sum()
```

[32]: 54957

[33]:
```
data['Cyclists_Killed'].sum()
```

[33]: 227

[34]:
```
data['Motorists_Injured'].sum()
```

[34]: 425526

[35]:
```
data['Motorists_Killed'].sum()
```

[35]: 1133

[36]:
```
data['Borough'].value_counts(dropna=False)
```

[36]:
```
Borough
Unknown          460334
BROOKLYN         457783
QUEENS           385240
MANHATTAN        317323
BRONX            210333
STATEN ISLAND     59910
Name: count, dtype: int64
```

2 Handle Unknown Values in Borough

[37]:
```
# Create a separate dataset for rows with 'Unknown' borough
unknown_borough_data = data[data['Borough'] == 'Unknown']

# Keep only rows with known boroughs in the main dataset
data = data[data['Borough'] != 'Unknown']
```

The Borough values are in uppercase (e.g., BROOKLYN) lets standardize casing for consistency in visualizations and analysis.

[38]:
```
# Convert borough names to title case
data['Borough'] = data['Borough'].str.title()
```

[39]:
```
# Check top values in Contributing_Factor_1
print("Top values in Contributing_Factor_1:")
print(data['Contributing_Factor_1'].value_counts().head(10))
```

```
Top values in Contributing_Factor_1:
Contributing_Factor_1
Unspecified                      539252
```

```
Driver Inattention/Distraction    275664
Failure to Yield Right-of-Way      90961
Backing Unsafely                   60071
Following Too Closely              46534
Other Vehicular                    45754
Passing Too Closely                37101
Passing or Lane Usage Improper     35886
Turning Improperly                 34024
Traffic Control Disregarded        25906
Name: count, dtype: int64
```

[40]:
```python
# Replace "Unspecified" with "Unknown"
data['Contributing_Factor_1'] = data['Contributing_Factor_1'].
↪replace("Unspecified", "Unknown")
```

Outlier Detection in Numerical Columns

[41]:
```python
import numpy as np
import pandas as pd

# Step 1: Apply log transformation to the 'Persons_Injured' column
data['Persons_Injured_Log'] = np.log1p(data['Persons_Injured'])

# Step 2: Calculate IQR on the log-transformed column
q1_log = data['Persons_Injured_Log'].quantile(0.25)
q3_log = data['Persons_Injured_Log'].quantile(0.75)
iqr_log = q3_log - q1_log

# Step 3: Define adjusted lower and upper bounds
lower_bound_adjusted = q1_log - 2.5 * iqr_log
upper_bound_adjusted = q3_log + 2.5 * iqr_log

# Step 4: Identify outliers based on the adjusted bounds
outliers_log = data[
    (data['Persons_Injured_Log'] < lower_bound_adjusted) |
    (data['Persons_Injured_Log'] > upper_bound_adjusted)
]
print(f"Number of outliers in log-transformed Persons_Injured:␣
↪{len(outliers_log)}")

# Step 5: Remove outliers
data_no_outliers_adjusted = data[
    (data['Persons_Injured_Log'] >= lower_bound_adjusted) &
    (data['Persons_Injured_Log'] <= upper_bound_adjusted)
]
print(f"Dataset size after adjusted outlier removal: {data_no_outliers_adjusted.
↪shape}")
```

```
Number of outliers in log-transformed Persons_Injured: 326515
```

```
Dataset size after adjusted outlier removal: (1104063, 20)
```

3 Feature Engineering

Creating a severity_score Column The severity_score combines the number of injuries and fatalities to provide a single measure of collision severity.

[42]:
```python
# Create a severity_score column
data['severity_score'] = data['Persons_Injured'] + (data['Persons_Killed'] * 10)

# Verify the new column
print("Severity Score column created:")
print(data[['Persons_Injured', 'Persons_Killed', 'severity_score']].head())
```

```
Severity Score column created:
   Persons_Injured  Persons_Killed  severity_score
3              0.0             0.0             0.0
4              0.0             0.0             0.0
7              2.0             0.0             2.0
8              0.0             0.0             0.0
9              0.0             0.0             0.0
```

Extracting hour_of_day from the Time Column This feature allows for time-based analysis, such as identifying peak hours for collisions.

[43]:
```python
# Extract hour of the day from the Time column
data['hour_of_day'] = pd.to_datetime(data['Time'], format='%H:%M:%S',␣
 ↪errors='coerce').dt.hour

# Verify the new column
print("Hour of Day column created:")
print(data[['Time', 'hour_of_day']].head())
```

```
Hour of Day column created:
       Time  hour_of_day
3  09:35:00            9
4  08:13:00            8
7  08:17:00            8
8  21:10:00           21
9  14:58:00           14
```

Categorizing severity_score

to analyze collisions by severity categories (e.g., low, medium, high severity), create bins for the severity_score.

[44]:
```python
# Categorize severity_score into Low, Medium, and High severity
bins = [0, 2, 10, data['severity_score'].max()]
labels = ['Low', 'Medium', 'High']
```

```
data['severity_category'] = pd.cut(data['severity_score'], bins=bins,␣
↪labels=labels, include_lowest=True)

# Verify the new column
print("Severity Category column created:")
print(data[['severity_score', 'severity_category']].head())
```

```
Severity Category column created:
   severity_score severity_category
3             0.0               Low
4             0.0               Low
7             2.0               Low
8             0.0               Low
9             0.0               Low
```

[45]:
```
# Check the new columns
print("Columns after feature engineering:")
print(data.columns)
```

```
Columns after feature engineering:
Index(['Date', 'Time', 'Borough', 'ZipCode', 'Latitude', 'Longitude',
       'Geo_Location', 'Persons_Injured', 'Persons_Killed',
       'Pedestrians_Injured', 'Pedestrians_Killed', 'Cyclists_Injured',
       'Cyclists_Killed', 'Motorists_Injured', 'Motorists_Killed',
       'Contributing_Factor_1', 'Contributing_Factor_2', 'Vehicle_Type_1',
       'Vehicle_Type_2', 'Persons_Injured_Log', 'severity_score',
       'hour_of_day', 'severity_category'],
      dtype='object')
```

[46]:
```
# Summary of new features
print("Summary of new features:")
print(data[['severity_score', 'hour_of_day', 'severity_category']].describe())
```

```
Summary of new features:
       severity_score   hour_of_day
count    1.430563e+06  1.430589e+06
mean     3.138771e-01  1.327201e+01
std      7.810146e-01  5.693542e+00
min      0.000000e+00  0.000000e+00
25%      0.000000e+00  9.000000e+00
50%      0.000000e+00  1.400000e+01
75%      0.000000e+00  1.800000e+01
max      9.200000e+01  2.300000e+01
```

[47]:
```
lethal_crashes = data[data['Persons_Killed'] > 0]
lethal_crashes.head()
```

[47]:
```
            Date      Time    Borough ZipCode   Latitude  Longitude  \
591   04/15/2021  15:18:00   Brooklyn   11209  40.620487 -74.029305
1350  07/08/2021  22:03:00  Manhattan   10002  40.721474 -73.983830
2345  08/27/2021  09:15:00  Manhattan   10035  40.805740 -73.942764
2436  09/11/2021  18:18:00   Brooklyn   11238  40.684204 -73.968060
2606  04/08/2021  19:55:00      Bronx   10459  40.830307 -73.898730

                  Geo_Location  Persons_Injured  Persons_Killed  \
591   (40.620487, -74.029305)              0.0             1.0
1350   (40.721474, -73.98383)              0.0             1.0
2345   (40.80574, -73.942764)              1.0             1.0
2436   (40.684204, -73.96806)              3.0             1.0
2606   (40.830307, -73.89873)              0.0             1.0

      Pedestrians_Injured  Pedestrians_Killed  Cyclists_Injured  \
591                     0                   1                 0
1350                    0                   0                 0
2345                    1                   1                 0
2436                    1                   1                 0
2606                    0                   0                 0

      Cyclists_Killed  Motorists_Injured  Motorists_Killed  \
591                 0                  0                 0
1350                1                  0                 0
2345                0                  0                 0
2436                0                  2                 0
2606                0                  0                 0

                Contributing_Factor_1 Contributing_Factor_2  \
591   Driver Inattention/Distraction               Unknown
1350     Traffic Control Disregarded           Unspecified
2345                          Unknown               Unknown
2436                     Unsafe Speed           Unspecified
2606              Driver Inexperience               Unknown

                            Vehicle_Type_1 Vehicle_Type_2  Persons_Injured_Log  \
591   Station Wagon/Sport Utility Vehicle        Unknown             0.000000
1350                                Sedan           Bike             0.000000
2345                                Sedan        Unknown             0.693147
2436                                Sedan          Sedan             1.386294
2606                               E-Bike        Unknown             0.000000

      severity_score  hour_of_day severity_category
591             10.0           15            Medium
1350            10.0           22            Medium
2345            11.0            9              High
2436            13.0           18              High
```

```
2606             10.0            19            Medium
```

[48]:
```
# Verify null values
print("Remaining Missing Values:")
print(data.isnull().sum())
```

```
Remaining Missing Values:
Date                        0
Time                        0
Borough                     0
ZipCode                     0
Latitude                    0
Longitude                   0
Geo_Location                0
Persons_Injured            11
Persons_Killed             23
Pedestrians_Injured         0
Pedestrians_Killed          0
Cyclists_Injured            0
Cyclists_Killed             0
Motorists_Injured           0
Motorists_Killed            0
Contributing_Factor_1       0
Contributing_Factor_2       0
Vehicle_Type_1              0
Vehicle_Type_2              0
Persons_Injured_Log        11
severity_score             26
hour_of_day                 0
severity_category          26
dtype: int64
```

[49]:
```
# Drop rows with any missing values in relevant columns
data = data.dropna(subset=['Persons_Injured', 'Persons_Killed',
 ↪'severity_score', 'severity_category', 'Persons_Injured_Log'])
```

[50]:
```
data.isnull().sum()
```

[50]:
```
Date                        0
Time                        0
Borough                     0
ZipCode                     0
Latitude                    0
Longitude                   0
Geo_Location                0
Persons_Injured             0
Persons_Killed              0
Pedestrians_Injured         0
```

```
Pedestrians_Killed        0
Cyclists_Injured          0
Cyclists_Killed           0
Motorists_Injured         0
Motorists_Killed          0
Contributing_Factor_1     0
Contributing_Factor_2     0
Vehicle_Type_1            0
Vehicle_Type_2            0
Persons_Injured_Log       0
severity_score            0
hour_of_day               0
severity_category         0
dtype: int64
```

#Cleaning Steps for Contributing_Factor_1:

Remove or Replace "Unspecified" Values:

Replace "Unspecified" with "Unknown" or drop rows where this is present if it constitutes a small portion of the dataset.

Group Similar Factors:

Combine similar contributing factors into broader categories for better interpretation (e.g., "Driver Inattention/Distraction" and "Fatigued/Drowsy" can be grouped as "Driver Issues").

Standardize Case: Ensure all factor values are consistent (e.g., all lowercase or title case).

[51]:
```python
# Replace "Unspecified" with "Unknown"
data['Contributing_Factor_1'] = data['Contributing_Factor_1'].
 ↪replace("Unspecified", "Unknown")

# Group similar contributing factors
factor_mapping = {
    "Driver Inattention/Distraction": "Driver Issues",
    "Fatigued/Drowsy": "Driver Issues",
    "Failure to Yield Right-of-Way": "Failure to Yield",
    "Following Too Closely": "Tailgating",
    "Backing Unsafely": "Unsafe Maneuver",
    "Other Vehicular": "Other",
    "Turning Improperly": "Improper Turning",
    "Passing Too Closely": "Improper Passing",
    "Lane Changing Improper": "Improper Lane Use",
}

data['Contributing_Factor_1'] = data['Contributing_Factor_1'].
 ↪replace(factor_mapping)

# Verify cleaned contributing factors
```

```python
print(data['Contributing_Factor_1'].value_counts())
```

```
Contributing_Factor_1
Unknown                                                   544625
Driver Issues                                             301292
Failure to Yield                                           90961
Unsafe Maneuver                                            60071
Tailgating                                                 46534
Other                                                      45753
Improper Passing                                           37101
Passing or Lane Usage Improper                             35886
Improper Turning                                           34023
Traffic Control Disregarded                                25906
Driver Inexperience                                        22676
Unsafe Lane Changing                                       17364
Unsafe Speed                                               16877
Alcohol Involvement                                        16223
Lost Consciousness                                         15888
Prescription Medication                                    12842
Pavement Slippery                                          11086
View Obstructed/Limited                                     9922
Oversized Vehicle                                           9201
Reaction to Uninvolved Vehicle                              8941
Outside Car Distraction                                     7737
Physical Disability                                         7680
Pedestrian/Bicyclist/Other Pedestrian Error/Confusion       7133
Aggressive Driving/Road Rage                                6643
Passenger Distraction                                       6508
Brakes Defective                                            4422
Fell Asleep                                                 3512
Glare                                                       2701
Obstruction/Debris                                          2301
Failure to Keep Right                                       2160
Steering Failure                                            1896
Illness                                                     1875
Other Electronic Device                                     1771
Pavement Defective                                          1432
Tire Failure/Inadequate                                     1173
Illnes                                                      1080
Reaction to Other Uninvolved Vehicle                        1072
Driverless/Runaway Vehicle                                   960
Animals Action                                               947
Accelerator Defective                                        794
Lane Marking Improper/Inadequate                             615
Traffic Control Device Improper/Non-Working                  591
Drugs (illegal)                                              563
Cell Phone (hand-Held)                                       332
Drugs (Illegal)                                              320
```

```
Cell Phone (hands-free)                                       177
Tow Hitch Defective                                           152
Other Lighting Defects                                        124
Tinted Windows                                                114
Headlights Defective                                          111
Vehicle Vandalism                                             101
Eating or Drinking                                             71
Using On Board Navigation Device                               68
Cell Phone (hand-held)                                         56
Windshield Inadequate                                          54
Shoulders Defective/Improper                                   51
80                                                             46
Texting                                                        30
Listening/Using Headphones                                     15
1                                                               3
Name: count, dtype: int64
```

[52]:

```python
print(data['Contributing_Factor_2'].value_counts())
```

```
Contributing_Factor_2
Unspecified                                                1014483
Unknown                                                     237188
Driver Inattention/Distraction                               62333
Other Vehicular                                              23068
Failure to Yield Right-of-Way                                12269
Passing or Lane Usage Improper                                7433
Following Too Closely                                         7374
Fatigued/Drowsy                                               6082
Backing Unsafely                                              5956
Turning Improperly                                            5766
Passing Too Closely                                           5738
Traffic Control Disregarded                                   5294
Driver Inexperience                                           4715
Lost Consciousness                                            4063
Unsafe Speed                                                  3317
Unsafe Lane Changing                                          3114
Prescription Medication                                       2527
Pavement Slippery                                             2317
View Obstructed/Limited                                       2115
Physical Disability                                           1817
Pedestrian/Bicyclist/Other Pedestrian Error/Confusion         1770
Oversized Vehicle                                             1565
Outside Car Distraction                                       1420
Passenger Distraction                                         1225
Reaction to Uninvolved Vehicle                                1125
Alcohol Involvement                                           1106
Aggressive Driving/Road Rage                                  1049
Other Electronic Device                                        492
```

```
Failure to Keep Right                                       458
Illness                                                     368
Fell Asleep                                                 366
Obstruction/Debris                                          349
Traffic Control Device Improper/Non-Working                 343
Lane Marking Improper/Inadequate                            318
Reaction to Other Uninvolved Vehicle                        299
Glare                                                       290
Brakes Defective                                            247
Pavement Defective                                          107
Drugs (Illegal)                                              76
Steering Failure                                             74
Driverless/Runaway Vehicle                                   65
Tire Failure/Inadequate                                      49
Headlights Defective                                         46
Accelerator Defective                                        46
Animals Action                                               43
Cell Phone (hand-Held)                                       42
Other Lighting Defects                                       41
Cell Phone (hands-free)                                      38
Drugs (illegal)                                              30
Illnes                                                       29
Tinted Windows                                               23
Tow Hitch Defective                                          18
Cell Phone (hand-held)                                       18
Vehicle Vandalism                                            11
Eating or Drinking                                           10
Listening/Using Headphones                                    9
Shoulders Defective/Improper                                  9
80                                                            7
Using On Board Navigation Device                              6
Windshield Inadequate                                         5
Texting                                                       2
Name: count, dtype: int64
```

[53]:
```python
# Check for duplicates in the two columns combined
duplicates = data[['Contributing_Factor_1', 'Contributing_Factor_2']].
 ↪duplicated().sum()

# Check for NaN values
nan_values = data[['Contributing_Factor_1', 'Contributing_Factor_2']].isna().
 ↪sum()

print(f"Number of duplicate rows between the two columns: {duplicates}")
print(f"NaN values in Contributing_Factor_1:␣
 ↪{nan_values['Contributing_Factor_1']}")
```

```
print(f"NaN values in Contributing_Factor_2:␣
↪{nan_values['Contributing_Factor_2']}")
```

```
Number of duplicate rows between the two columns: 1428926
NaN values in Contributing_Factor_1: 0
NaN values in Contributing_Factor_2: 0
```

[54]:
```
#merge similar columns:
# Update mapping for redundant categories and typos
additional_mapping = {
    "Drugs (illegal)": "Drugs",
    "Drugs (Illegal)": "Drugs",
    "Cell Phone (hand-Held)": "Cell Phone Usage",
    "Cell Phone (hand-held)": "Cell Phone Usage",
    "Cell Phone (hands-free)": "Cell Phone Usage",
    "Illnes": "Illness",
    "Reaction to Uninvolved Vehicle": "Reaction to Other Vehicle",
    "Reaction to Other Uninvolved Vehicle": "Reaction to Other Vehicle",
}

# Apply the mapping
data['Contributing_Factor_1'] = data['Contributing_Factor_1'].
↪replace(additional_mapping)
```

Remove or Merge Low-Frequency and Erroneous Categories

Low-frequency entries can be grouped into broader categories (e.g., "Driver Distraction" for texting, eating, or using headphones). Erroneous entries ("80" and "1") can be removed.

[55]:
```
# Remove erroneous entries and group low-frequency factors
data['Contributing_Factor_1'] = data['Contributing_Factor_1'].replace({
    "Texting": "Driver Distraction",
    "Listening/Using Headphones": "Driver Distraction",
    "Eating or Drinking": "Driver Distraction",
    "80": "Unknown",
    "1": "Unknown",
})

# Verify cleaned categories
print(data['Contributing_Factor_1'].value_counts())
```

```
Contributing_Factor_1
Unknown                                                   544674
Driver Issues                                             301292
Failure to Yield                                           90961
Unsafe Maneuver                                            60071
Tailgating                                                 46534
Other                                                      45753
Improper Passing                                           37101
```

```
Passing or Lane Usage Improper                               35886
Improper Turning                                             34023
Traffic Control Disregarded                                  25906
Driver Inexperience                                          22676
Unsafe Lane Changing                                         17364
Unsafe Speed                                                 16877
Alcohol Involvement                                          16223
Lost Consciousness                                           15888
Prescription Medication                                      12842
Pavement Slippery                                            11086
Reaction to Other Vehicle                                    10013
View Obstructed/Limited                                       9922
Oversized Vehicle                                             9201
Outside Car Distraction                                       7737
Physical Disability                                           7680
Pedestrian/Bicyclist/Other Pedestrian Error/Confusion         7133
Aggressive Driving/Road Rage                                  6643
Passenger Distraction                                         6508
Brakes Defective                                              4422
Fell Asleep                                                   3512
Illness                                                       2955
Glare                                                         2701
Obstruction/Debris                                            2301
Failure to Keep Right                                         2160
Steering Failure                                              1896
Other Electronic Device                                       1771
Pavement Defective                                            1432
Tire Failure/Inadequate                                       1173
Driverless/Runaway Vehicle                                     960
Animals Action                                                 947
Drugs                                                          883
Accelerator Defective                                          794
Lane Marking Improper/Inadequate                               615
Traffic Control Device Improper/Non-Working                    591
Cell Phone Usage                                               565
Tow Hitch Defective                                            152
Other Lighting Defects                                         124
Driver Distraction                                             116
Tinted Windows                                                 114
Headlights Defective                                           111
Vehicle Vandalism                                              101
Using On Board Navigation Device                                68
Windshield Inadequate                                           54
Shoulders Defective/Improper                                    51
Name: count, dtype: int64
```

Address “Unknown” Proportion

“Unknown” remains a significant portion of the data. You can either: Exclude rows with “Unknown" if you're analyzing trends specific to known contributing factors. Retain "Unknown" as it may still provide useful aggregate-level insights (e.g., geographic or temporal trends)

[56]:
```
# Exclude rows with "Unknown"
data_filtered = data[data['Contributing_Factor_1'] != "Unknown"]

# Verify the size of the filtered dataset
print(f"Original dataset size: {data.shape}")
print(f"Filtered dataset size: {data_filtered.shape}")
```

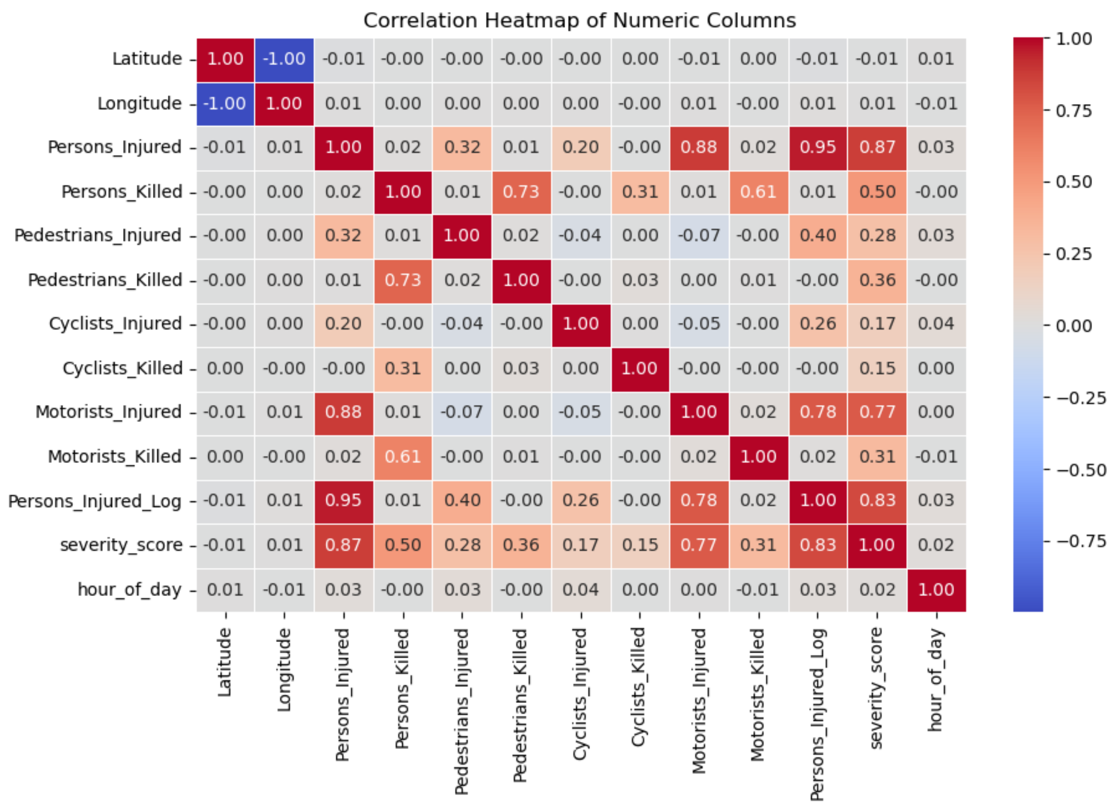
```
Original dataset size: (1430563, 23)
Filtered dataset size: (885889, 23)
```

No Missing Values: All columns now have complete data, which is ideal for exploratory data analysis and modeling.

Updated Data Columns: Key columns (Persons_Injured, Persons_Killed, severity_score, etc.) have been cleaned and transformed. New features such as Persons_Injured_Log, severity_score, severity_category, and hour_of_day are now ready for analysis.

4 Exploratory Data Analysis (EDA)

[57]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


# Select only numeric columns for correlation analysis
numeric_columns = data.select_dtypes(include=[np.number])

# Compute the correlation matrix
correlation_matrix = numeric_columns.corr()

# Plot the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm",␣
↪linewidths=0.5)
plt.title("Correlation Heatmap of Numeric Columns")
plt.show()
```



High Correlations:

Persons_Injured is highly correlated with Persons_Injured_Log (0.95), which is expected since Persons_Injured_Log is derived from Persons_Injured.

Persons_Injured and Motorists_Injured (0.88) show a strong positive correlation, suggesting that a significant proportion of injuries in accidents involve motorists.

Severity_Score shows a strong correlation with Persons_Injured (0.87) and Persons_Injured_Log (0.83), indicating that these features contribute significantly to severity.

Moderate Correlations: Pedestrians_Injured and Persons_Injured (0.32): Pedestrian injuries have a noticeable, albeit weaker, contribution to total injuries. Pedestrians_Killed and Persons_Killed (0.73): Fatalities among pedestrians are strongly associated with total fatalities. Motorists_Injured and Severity_Score (0.77): Motorist injuries are closely tied to accident severity.

Low or Negligible Correlations: Geographic attributes like Latitude and Longitude have very weak correlations with other features, implying they don't directly influence injuries or severity in a straightforward linear relationship

[58]:
```
#Highlight hotspots with the highest number of accidents.
```

[59]:
```
import folium
from folium.plugins import MarkerCluster
```

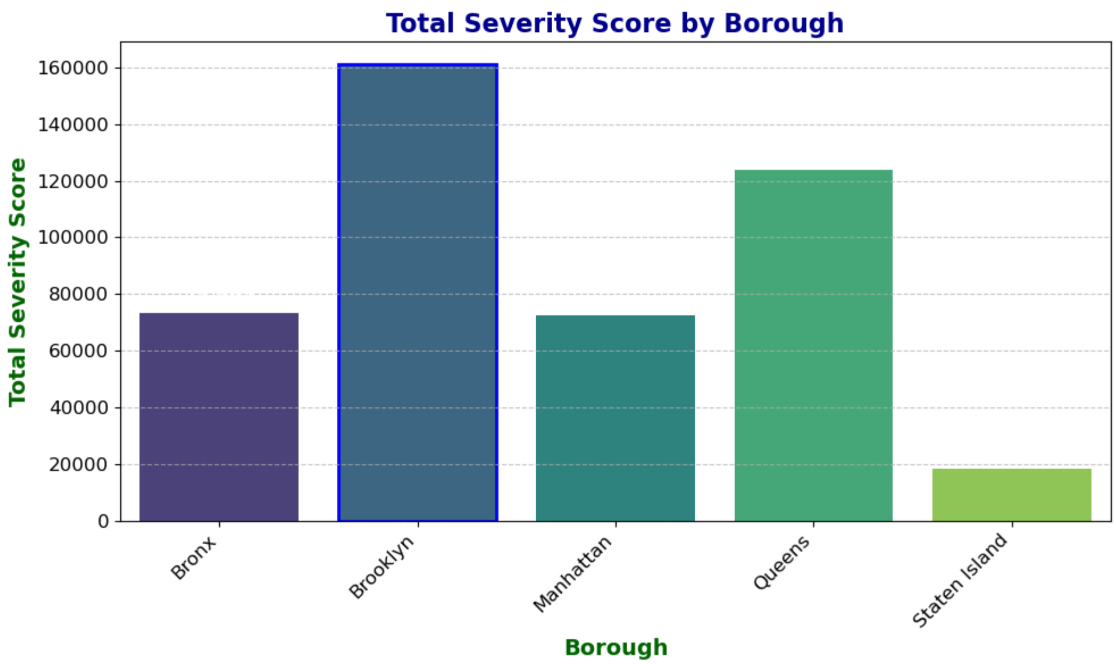
```python
# Aggregate collision data by latitude and longitude
hotspot_data = data.groupby(['Latitude', 'Longitude'], as_index=False).agg(
    collision_count=('Date', 'count'),
    avg_severity=('severity_score', 'mean')
)

# Filter top hotspots based on collision count
top_hotspots = hotspot_data.nlargest(100, 'collision_count')

# Create a base map centered around NYC
nyc_map = folium.Map(location=[40.7128, -74.0060], zoom_start=11)

# Add MarkerCluster for better visualization
marker_cluster = MarkerCluster().add_to(nyc_map)

# Add points to the map
for _, row in top_hotspots.iterrows():
    color = 'red' if row['collision_count'] > 50 else 'orange'  # Adjust color␣
 ↪based on collision count
    folium.CircleMarker(
        location=(row['Latitude'], row['Longitude']),
        radius=min(row['collision_count'] / 10, 15),  # Scale the marker size
        color=color,
        fill=True,
        fill_color=color,
        fill_opacity=0.7,
        popup=folium.Popup(
            f"<b>Collisions:</b> {row['collision_count']}<br><b>Avg Severity:</
 ↪b> {row['avg_severity']:.2f}",
            max_width=250
        )
    ).add_to(marker_cluster)

# Save the map to an HTML file
nyc_map.save('collision_hotspots_highlighted.html')
print("Hotspot map saved as 'collision_hotspots_highlighted.html'")
```

```
Hotspot map saved as 'collision_hotspots_highlighted.html'
```

Let's explore which boroughs have the highest number of accidents and potential hotspots using geographic data.

[60]:
```python
import matplotlib.pyplot as plt
import seaborn as sns


# Group the data by Borough and calculate the sum of severity scores
```

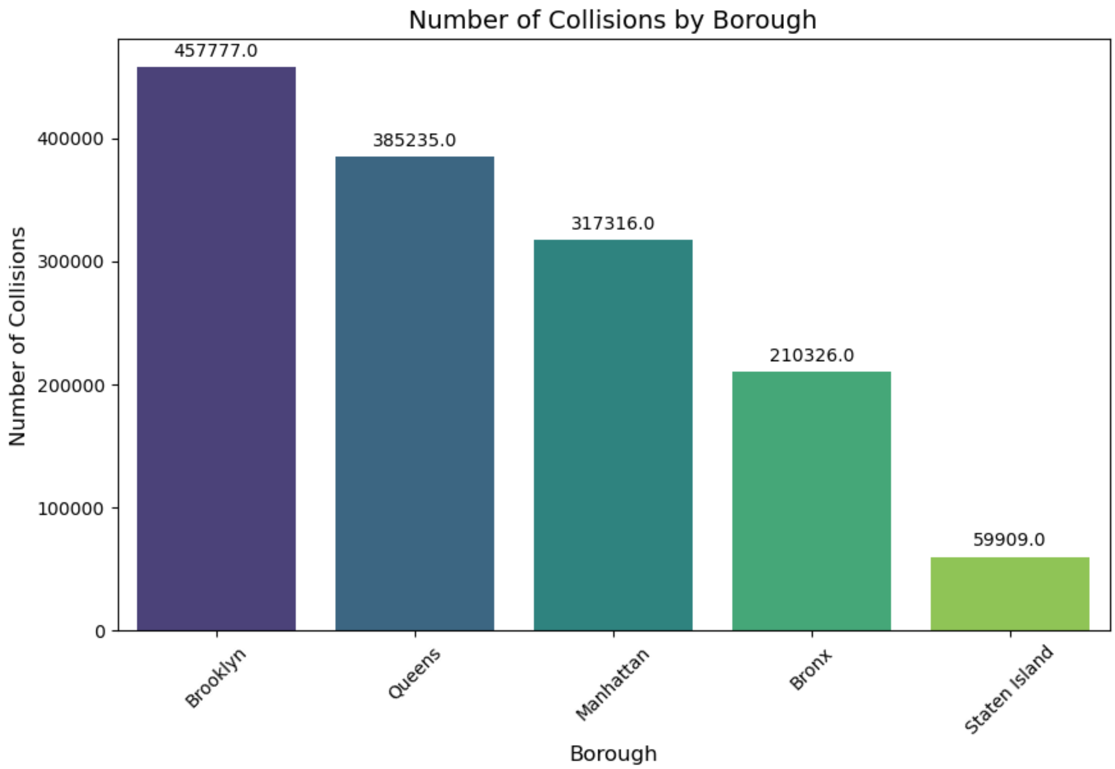
```python
severity_by_borough = data.groupby('Borough')['severity_score'].sum().reset_index()

# Create a bar plot with the 'viridis' color palette
plt.figure(figsize=(10, 6))
bar_plot = sns.barplot(data=severity_by_borough, x='Borough', y='severity_score', palette='viridis')

# Find the maximum severity score to highlight the corresponding bar
max_severity_index = severity_by_borough['severity_score'].idxmax()

# Add annotations to show the values on top of the bars
for p in bar_plot.patches:
    # Annotate the bars with the total severity score values
    bar_plot.annotate(f'{p.get_height():,.0f}',
                      (p.get_x() + p.get_width() / 2., p.get_height()),
                      ha='center', va='center',
                      xytext=(0, 8),
                      textcoords='offset points',
                      fontsize=12, color='white', fontweight='bold')

    # Highlight the bar with the maximum severity score
    if p.get_height() == severity_by_borough.loc[max_severity_index, 'severity_score']:
        p.set_edgecolor('Blue')  # Add black border around the bar
        p.set_linewidth(2)  # Increase the thickness of the border

# Add title and labels with larger font sizes and emphasis
plt.title('Total Severity Score by Borough', fontsize=16, fontweight='bold', color='darkblue')
plt.xlabel('Borough', fontsize=14, fontweight='bold', color='darkgreen')
plt.ylabel('Total Severity Score', fontsize=14, fontweight='bold', color='darkgreen')

# Customize tick parameters for better readability
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)

# Add gridlines to improve clarity
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Display the plot with tight layout to avoid overlap
plt.tight_layout()
plt.show()
```



Brooklyn has the highest total severity score, indicating the most severe accidents with a high number of injuries and fatalities. Queens ranks second, showing significant severity but lower than Brooklyn. Manhattan and Bronx have moderate severity scores, with fewer severe incidents than Brooklyn and Queens. Staten Island has the lowest severity score, indicating fewer or less severe accidents compared to other boroughs.

[61]:
```
 #Distribution of Collisions by Borough
import matplotlib.pyplot as plt
import seaborn as sns

# Distribution of Collisions by Borough
# Bar plot for collisions by borough
plt.figure(figsize=(10, 6))
bar_plot = sns.countplot(data=data, x='Borough', order=data['Borough'].
 ↪value_counts().index, palette='viridis')

plt.title("Number of Collisions by Borough", fontsize=14)
plt.xlabel("Borough", fontsize=12)
plt.ylabel("Number of Collisions", fontsize=12)
plt.xticks(rotation=45)

# Annotate each bar
for p in bar_plot.patches:
    bar_plot.annotate(format(p.get_height(), '.1f'),
                      (p.get_x() + p.get_width() / 2., p.get_height()),
                      ha = 'center', va = 'center',
```

```
                            xytext = (0, 9),
                            textcoords = 'offset points')

plt.show()
```



The high number of accidents in Brooklyn and Queens could be attributed to their larger populations, denser traffic, and extensive road networks. Staten Island, being less densely populated and having less traffic volume, experiences the lowest number of collisions

5 Analysis of Hourly Patterns

Explore how the number of accidents and severity scores vary across hours of the day.

[62]:
```
#Make sure you have installed plotly
```

[63]:
```
!pip install plotly
```

```
Requirement already satisfied: plotly in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (5.18.0)
Requirement already satisfied: tenacity>=6.2.0 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
plotly) (8.2.3)
Requirement already satisfied: packaging in
```

/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from plotly) (23.0)

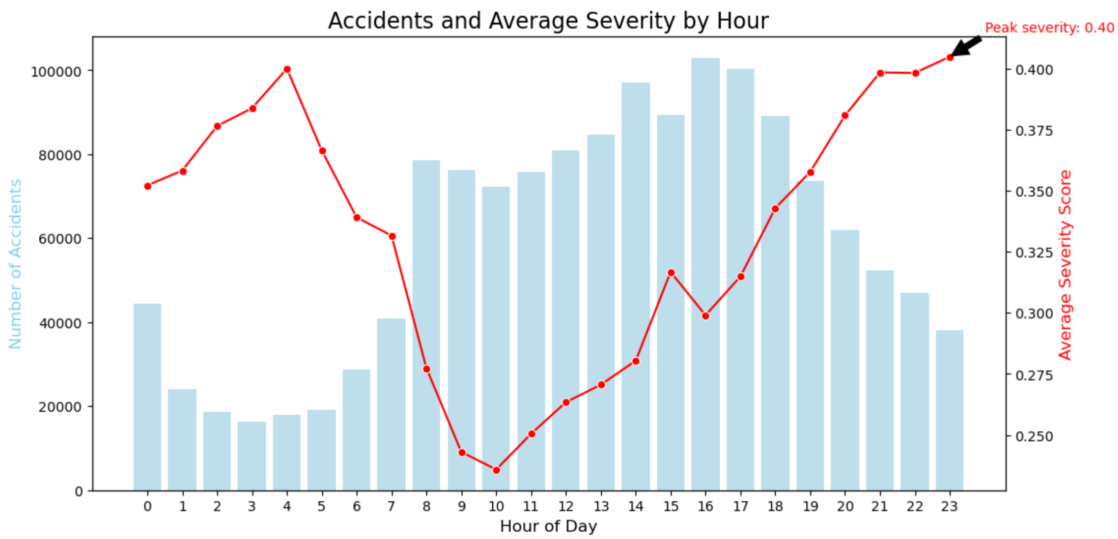
[64]:
```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

# Ensure 'Date' is in datetime format
data['Date'] = pd.to_datetime(data['Date'], errors='coerce')

# Drop rows with invalid or missing dates
data = data.dropna(subset=['Date'])

# Extract time-based features
data['Month'] = data['Date'].dt.month
data['Year'] = data['Date'].dt.year
data['Day'] = data['Date'].dt.day



# Aggregating data to get the count of accidents per year
accidents_per_year = data.groupby('Year').size().reset_index(name='Accidents')

# Create a line plot and add a trend line using a linear regression
fig_yearly = px.line(accidents_per_year, x='Year', y='Accidents', title='Yearly␣
↪Trend of Accidents',
                     labels={'Accidents': 'Number of Accidents'}, markers=True)
fig_yearly.add_traces(go.Scatter(x=accidents_per_year['Year'], y=pd.
↪Series(accidents_per_year['Accidents']).rolling(window=3).mean(),
                    mode='lines', name='Moving Average', line=dict(color='red',␣
↪width=2)))

# Highlight the maximum accident year
max_accident_year = accidents_per_year[accidents_per_year['Accidents'] ==␣
↪accidents_per_year['Accidents'].max()]
fig_yearly.add_trace(go.Scatter(x=max_accident_year['Year'],␣
↪y=max_accident_year['Accidents'],
                    mode='markers', marker=dict(color='red', size=10),
                    name='Peak Accident Year'))

# Save the yearly trend chart as an HTML file
fig_yearly.write_html('Yearly_Trend_of_Accidents.html')

# Aggregating data to get the count of accidents per month for each year
accidents_per_month = data.groupby(['Year', 'Month']).size().
↪reset_index(name='Accidents')

# Creating an interactive line chart for month-by-month comparison across years
```

```
fig_monthly = px.line(accidents_per_month, x='Month', y='Accidents',␣
↪color='Year', title='Monthly Accident Trends by Year',
                      labels={'Accidents': 'Number of Accidents'},
                      category_orders={'Month': ['January', 'February',␣
↪'March', 'April', 'May', 'June',
                                                   'July', 'August', 'September',␣
↪'October', 'November', 'December']})

# Add annotations for significant trends or events
fig_monthly.add_annotation(x='July',␣
↪y=accidents_per_month[accidents_per_month['Month']=='July']['Accidents'].
↪max(),
                           text='Highest in Summer', showarrow=True,␣
↪arrowhead=1)

# Save the monthly trends chart as an HTML file
fig_monthly.write_html('Monthly_Accident_Trends_by_Year.html')
# Show plots
fig_yearly.show()
fig_monthly.show()
```

Rising and falling trends: Sharp increase from 2012 to 2014, then stable until a gradual decline after 2018. Impact of COVID-19:Significant decrease in accidents post-2020, likely due to pandemic-related changes in traffic patterns.

[65]:
```
import pandas as pd
import pandas as pd
import plotly.express as px
import plotly.express as px

# Group by hour of the day
hourly_data = data.groupby('hour_of_day').agg(
    accidents=('hour_of_day', 'count'),
    avg_severity=('severity_score', 'mean')
).reset_index()

# Plot
fig, ax1 = plt.subplots(figsize=(12, 6))
ax2 = ax1.twinx()

bar_plot = sns.barplot(x=hourly_data['hour_of_day'],␣
↪y=hourly_data['accidents'], ax=ax1, color='skyblue', alpha=0.6)
line_plot = sns.lineplot(x=hourly_data['hour_of_day'],␣
↪y=hourly_data['avg_severity'], ax=ax2, color='red', marker='o')

ax1.set_title('Accidents and Average Severity by Hour', fontsize=16)
ax1.set_xlabel('Hour of Day', fontsize=12)
```

```
ax1.set_ylabel('Number of Accidents', fontsize=12, color='skyblue')
ax2.set_ylabel('Average Severity Score', fontsize=12, color='red')

# Annotate peak severity
peak_severity = hourly_data['avg_severity'].max()
peak_hour = hourly_data[hourly_data['avg_severity'] ==␣
↪peak_severity]['hour_of_day'].values[0]
ax2.annotate(f'Peak severity: {peak_severity:.2f}', xy=(peak_hour,␣
↪peak_severity), xytext=(peak_hour+1, peak_severity+0.01),
             arrowprops=dict(facecolor='black', shrink=0.05), fontsize=10,␣
↪color='red')

plt.show()
```



[66]:
```
!pip install dash dash-bootstrap-components
```

```
Requirement already satisfied: dash in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (2.18.1)
Requirement already satisfied: dash-bootstrap-components in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (1.6.0)
Requirement already satisfied: Flask<3.1,>=1.0.4 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (3.0.3)
Requirement already satisfied: Werkzeug<3.1 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (3.0.4)
Requirement already satisfied: plotly>=5.0.0 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (5.18.0)
```

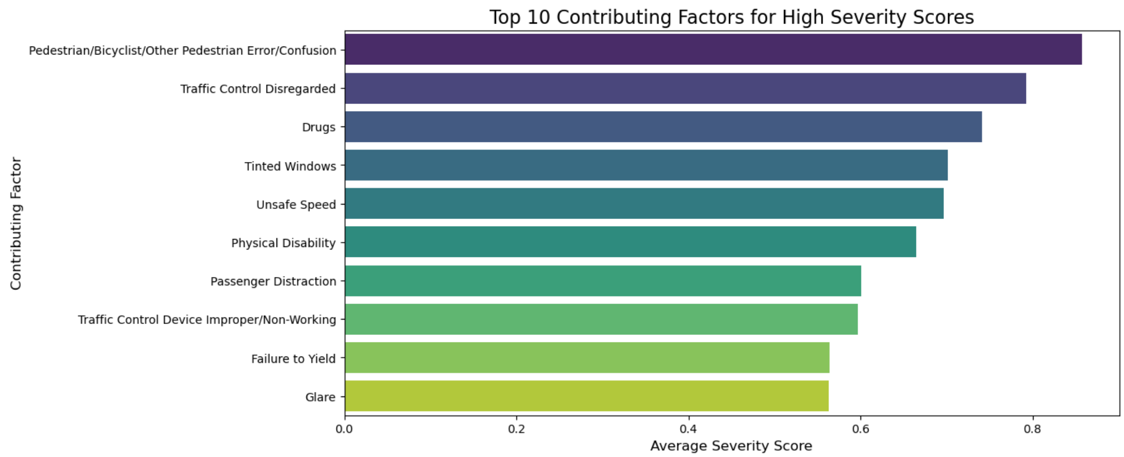
```
Requirement already satisfied: dash-html-components==2.0.0 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (2.0.0)
Requirement already satisfied: dash-core-components==2.0.0 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (2.0.0)
Requirement already satisfied: dash-table==5.0.0 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (5.0.0)
Requirement already satisfied: importlib-metadata in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (8.5.0)
Requirement already satisfied: typing-extensions>=4.1.1 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (4.12.2)
Requirement already satisfied: requests in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (2.31.0)
Requirement already satisfied: retrying in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (1.3.3)
Requirement already satisfied: nest-asyncio in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (1.5.6)
Requirement already satisfied: setuptools in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
dash) (68.0.0)
Requirement already satisfied: Jinja2>=3.1.2 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
Flask<3.1,>=1.0.4->dash) (3.1.2)
Requirement already satisfied: itsdangerous>=2.1.2 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
Flask<3.1,>=1.0.4->dash) (2.2.0)
Requirement already satisfied: click>=8.1.3 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
Flask<3.1,>=1.0.4->dash) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
Flask<3.1,>=1.0.4->dash) (1.8.2)
Requirement already satisfied: tenacity>=6.2.0 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
plotly>=5.0.0->dash) (8.2.3)
Requirement already satisfied: packaging in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
plotly>=5.0.0->dash) (23.0)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
Werkzeug<3.1->dash) (2.1.1)
```

```
Requirement already satisfied: zipp>=3.20 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
importlib-metadata->dash) (3.20.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
requests->dash) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
requests->dash) (2.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
requests->dash) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
requests->dash) (2024.8.30)
Requirement already satisfied: six>=1.7.0 in
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-packages (from
retrying->dash) (1.16.0)
```

6 Top Contributing Factors for Severe Accidents

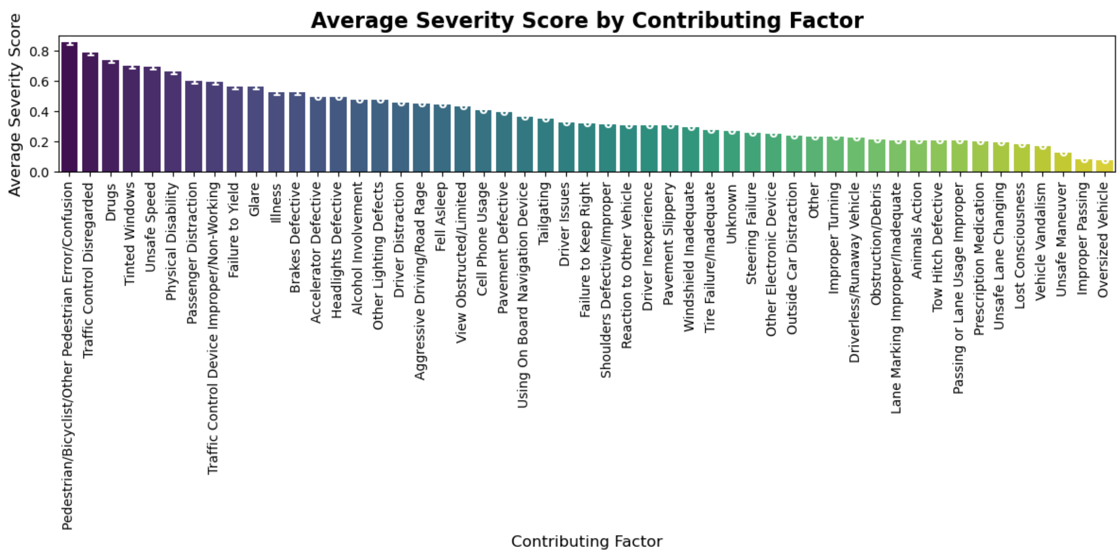
Identify which contributing factors are most associated with high severity scores.

[67]:
```python
import pandas as pd
import matplotlib.pyplot as plt
#Top Contributing Factors for Severe Accidents
# Filter for top 10 contributing factors with high severity scores
factor_severity = data.groupby('Contributing_Factor_1')['severity_score'].
 ↪mean().sort_values(ascending=False).head(10)

# Plot
plt.figure(figsize=(12, 6))
sns.barplot(x=factor_severity.values, y=factor_severity.index,␣
 ↪palette="viridis")
plt.title('Top 10 Contributing Factors for High Severity Scores', fontsize=16)
plt.xlabel('Average Severity Score', fontsize=12)
plt.ylabel('Contributing Factor', fontsize=12)
plt.show()
```



[68]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Group data by Contributing Factor and calculate the average severity score␣
↪for each factor
severity_by_factor = data.groupby('Contributing_Factor_1')['severity_score'].
↪mean().reset_index()

# Sort the factors by severity score in descending order
severity_by_factor_sorted = severity_by_factor.sort_values(by='severity_score',␣
↪ascending=False)

# Create a bar plot
plt.figure(figsize=(12, 6))
sns.barplot(data=severity_by_factor_sorted, x='Contributing_Factor_1',␣
↪y='severity_score', palette='viridis')

# Add labels and title
plt.title('Average Severity Score by Contributing Factor', fontsize=16,␣
↪fontweight='bold')
plt.xlabel('Contributing Factor', fontsize=12)
plt.ylabel('Average Severity Score', fontsize=12)
plt.xticks(rotation=90)

# Add data labels to bars
for p in plt.gca().patches:
    plt.gca().annotate(f'{p.get_height():.00f}', (p.get_x() + p.get_width() / 2.
↪, p.get_height()),
                       ha='center', va='center', fontsize=10, color='white',␣
↪fontweight='bold')
```

```
# Display the plot
plt.tight_layout()
plt.show()
```



[69]:
```
# Categorize severity scores
data['severity_category'] = pd.cut(
    data['severity_score'],
    bins=[-1, 0, 5, 10, np.inf],
    labels=["None", "Low", "Moderate", "High"]
)
```

[70]:
```
# Check distribution
severity_category_counts = data['severity_category'].value_counts()
print("Severity Category Counts:")
print(severity_category_counts)
```

```
Severity Category Counts:
severity_category
None        1102674
Low          324585
Moderate       2762
High            542
Name: count, dtype: int64
```

[71]:
```
# Replace 'Unknown' with NaN and one-hot encode
data['Contributing_Factor_1'] = data['Contributing_Factor_1'].
↪replace('Unknown', None)
```

```
data_encoded = pd.get_dummies(data, columns=['Contributing_Factor_1'],
 ↪drop_first=True)
```

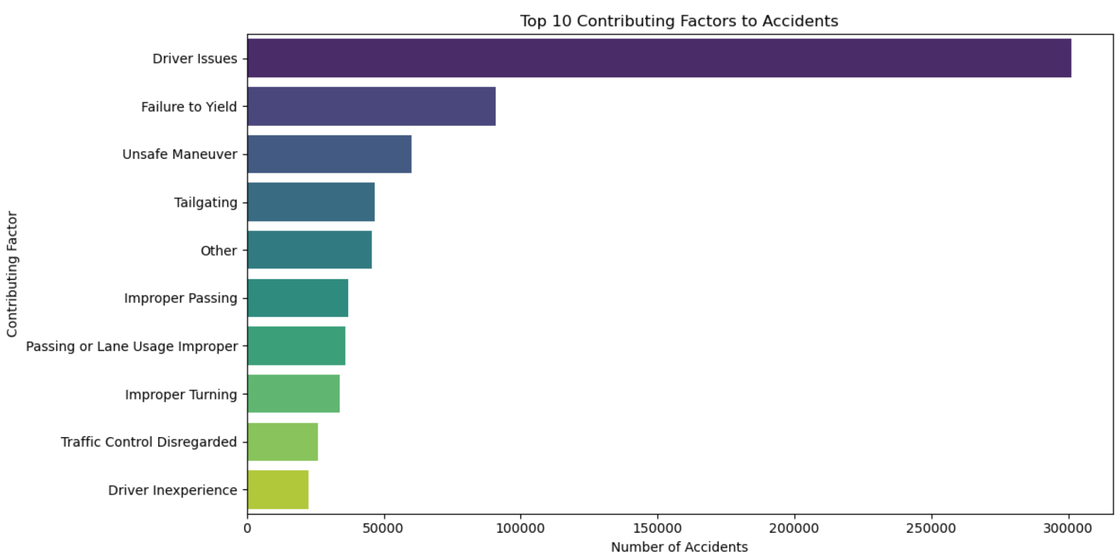
Identify the Top Contributing Factors

there are two contributing columns combine the two contributing factor columns (Contributing_Factor_1 and Contributing_Factor_2) into one column. Count the occurrences of each contributing factor.

[72]:
```
# Aggregate severe cases and fatalities by contributing factors
severe_cases = data[data['severity_score'] > 10]  # Adjust threshold if
 ↪necessary
top_factors_severe = severe_cases['Contributing_Factor_1'].value_counts().
 ↪head(10)

fatal_cases = data[data['Persons_Killed'] > 0]
top_factors_fatal = fatal_cases['Contributing_Factor_1'].value_counts().head(10)

# Display results
print("Top Factors for Severe Injuries:\n", top_factors_severe)
print("\nTop Factors for Fatalities:\n", top_factors_fatal)
```

```
Top Factors for Severe Injuries:
 Contributing_Factor_1
Traffic Control Disregarded      96
Unsafe Speed                     75
Driver Issues                    65
Failure to Yield                 44
Alcohol Involvement              24
Driver Inexperience              15
Physical Disability              15
Tailgating                        9
Passing or Lane Usage Improper    7
Drugs                             6
Name: count, dtype: int64

Top Factors for Fatalities:
 Contributing_Factor_1
Failure to Yield                                         241
Driver Issues                                            219
Traffic Control Disregarded                              187
Unsafe Speed                                             131
Pedestrian/Bicyclist/Other Pedestrian Error/Confusion     46
Alcohol Involvement                                       45
Driver Inexperience                                       38
Passenger Distraction                                     37
Unsafe Maneuver                                           30
Physical Disability                                       25
Name: count, dtype: int64
```

[73]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Group data by Contributing Factor and count occurrences
contributing_factors = data['Contributing_Factor_1'].value_counts().
 ↪reset_index()
contributing_factors.columns = ['Contributing Factor', 'Count']

# Get the top 10 contributing factors
top_contributing_factors = contributing_factors.head(10)

# Plot the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(x='Count', y='Contributing Factor', data=top_contributing_factors,␣
 ↪palette='viridis')
plt.title('Top 10 Contributing Factors to Accidents')
plt.xlabel('Number of Accidents')
plt.ylabel('Contributing Factor')
plt.tight_layout()
plt.show()
```



[74]:
```
#Accident Analysis Dashboard
```

[75]:
```
import dash
from dash import dcc, html
import plotly.graph_objs as go
import pandas as pd
```

```python
# Initialize the Dash app
app = dash.Dash(__name__)

# Group data by category (Pedestrians, Cyclists, Motorists) and sum up the␣
↪injuries and fatalities
injuries_by_category = data[['Pedestrians_Injured', 'Cyclists_Injured',␣
↪'Motorists_Injured']].sum()
fatalities_by_category = data[['Pedestrians_Killed', 'Cyclists_Killed',␣
↪'Motorists_Killed']].sum()

# Dash Layout with Dropdown filter for Injuries/Fatalities
app.layout = html.Div(children=[
    html.H1(children='Accident Analysis Dashboard'),

    html.Div(children='''This dashboard allows you to toggle between total␣
↪injuries and total fatalities.'''),

    # Dropdown to choose between Injury or Fatality
    dcc.Dropdown(
        id='injury-fatality-dropdown',
        options=[
            {'label': 'Total Injuries', 'value': 'injuries'},
            {'label': 'Total Fatalities', 'value': 'fatalities'}
        ],
        value='injuries',  # default value
        style={'width': '50%'}
    ),

    # Dynamic Graph based on Dropdown value
    dcc.Graph(id='category-graph')
])

# Callback to update graph based on dropdown selection
@app.callback(
    dash.dependencies.Output('category-graph', 'figure'),
    [dash.dependencies.Input('injury-fatality-dropdown', 'value')]
)
def update_graph(selected_value):
    if selected_value == 'injuries':
        # Create dynamic figure for Total Injuries by Category
        figure = go.Figure(data=[go.Bar(
            x=injuries_by_category.index,
            y=injuries_by_category.values,
            marker=dict(color='purple')
        )])
        figure.update_layout(
```

```
            title="Total Injuries by Category",
            xaxis_title="Category",
            yaxis_title="Total Injuries"
        )
    else:
        # Create dynamic figure for Total Fatalities by Category
        figure = go.Figure(data=[go.Bar(
            x=fatalities_by_category.index,
            y=fatalities_by_category.values,
            marker=dict(color='red')
        )])
        figure.update_layout(
            title="Total Fatalities by Category",
            xaxis_title="Category",
            yaxis_title="Total Fatalities"
        )

    return figure

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

```
<IPython.lib.display.IFrame at 0x35dbaaf50>
```

[76]:

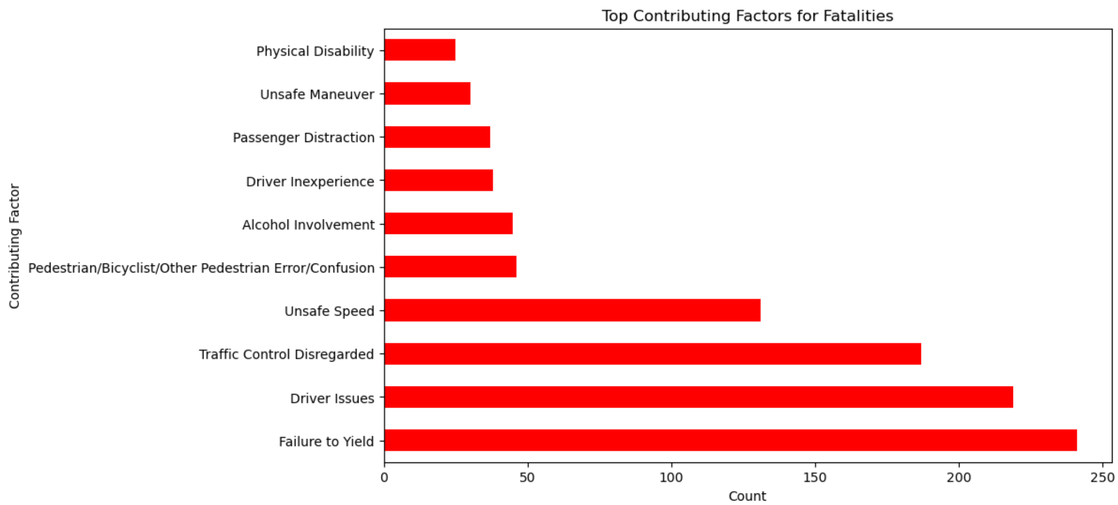
```
#Access at  http://127.0.0.1:8050/
```

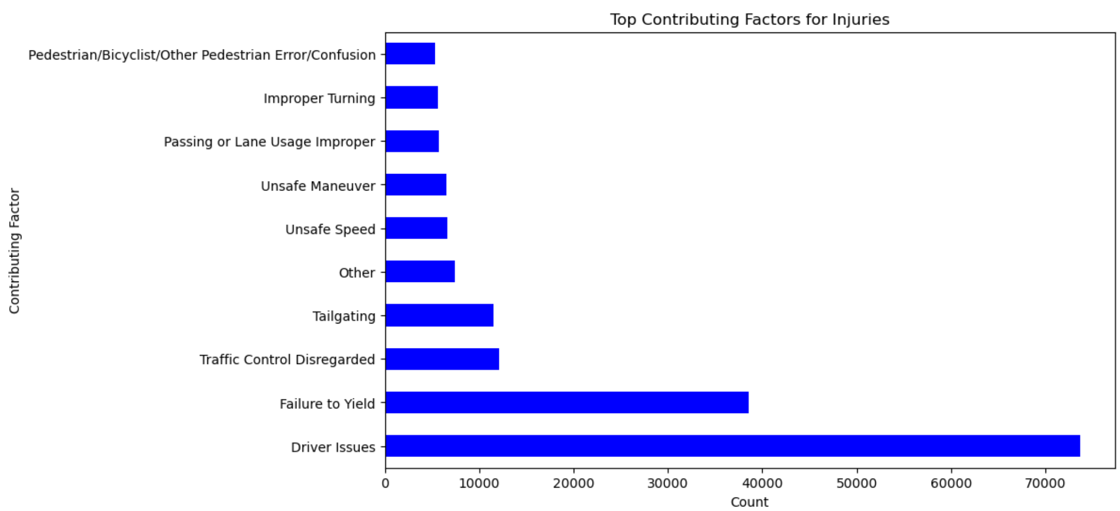
[77]:

```
# Group data by contributing factor for fatalities and injuries
fatalities_data = data[data['Persons_Killed'] > 0]
injuries_data = data[data['Persons_Injured'] > 0]

# Top contributing factors for fatalities
top_fatal_factors = fatalities_data['Contributing_Factor_1'].value_counts().
 ↪head(10)

# Top contributing factors for injuries
top_injury_factors = injuries_data['Contributing_Factor_1'].value_counts().
 ↪head(10)

# Display the results
print("Top Contributing Factors for Fatalities:\n", top_fatal_factors)
print("\nTop Contributing Factors for Injuries:\n", top_injury_factors)

# Plot the contributing factors for fatalities
plt.figure(figsize=(10, 6))
top_fatal_factors.plot(kind='barh', color='red')
plt.title('Top Contributing Factors for Fatalities')
```

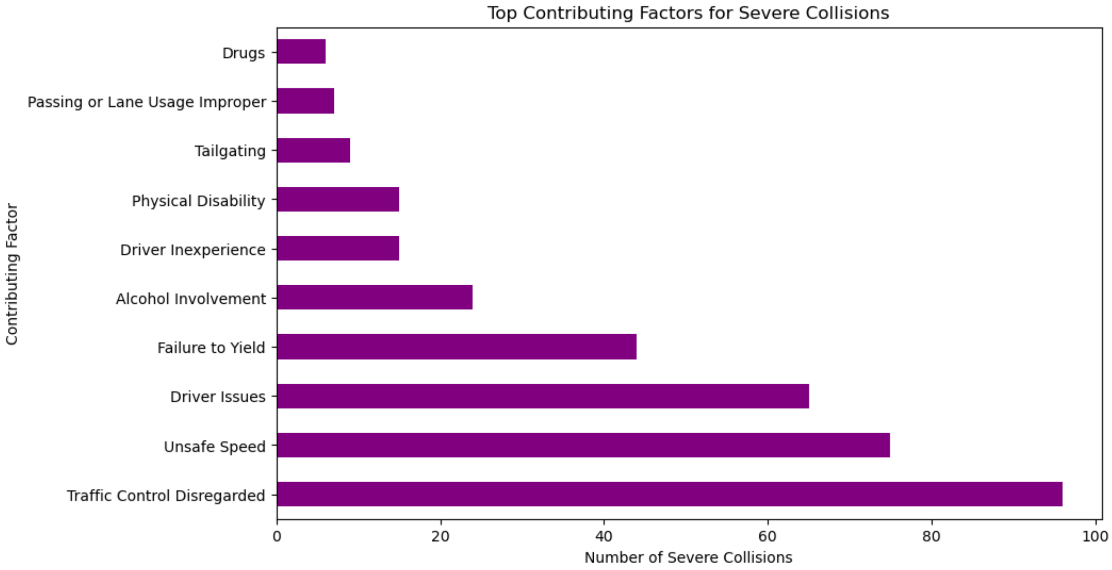
```python
plt.xlabel('Count')
plt.ylabel('Contributing Factor')
plt.show()

# Plot the contributing factors for injuries
plt.figure(figsize=(10, 6))
top_injury_factors.plot(kind='barh', color='blue')
plt.title('Top Contributing Factors for Injuries')
plt.xlabel('Count')
plt.ylabel('Contributing Factor')
plt.show()
```

```
Top Contributing Factors for Fatalities:
 Contributing_Factor_1
Failure to Yield                                        241
Driver Issues                                           219
Traffic Control Disregarded                             187
Unsafe Speed                                            131
Pedestrian/Bicyclist/Other Pedestrian Error/Confusion    46
Alcohol Involvement                                      45
Driver Inexperience                                      38
Passenger Distraction                                    37
Unsafe Maneuver                                          30
Physical Disability                                      25
Name: count, dtype: int64

Top Contributing Factors for Injuries:
 Contributing_Factor_1
Driver Issues                                           73657
Failure to Yield                                        38548
Traffic Control Disregarded                             12116
Tailgating                                              11521
Other                                                    7406
Unsafe Speed                                             6648
Unsafe Maneuver                                          6485
Passing or Lane Usage Improper                           5706
Improper Turning                                         5657
Pedestrian/Bicyclist/Other Pedestrian Error/Confusion    5373
Name: count, dtype: int64
```





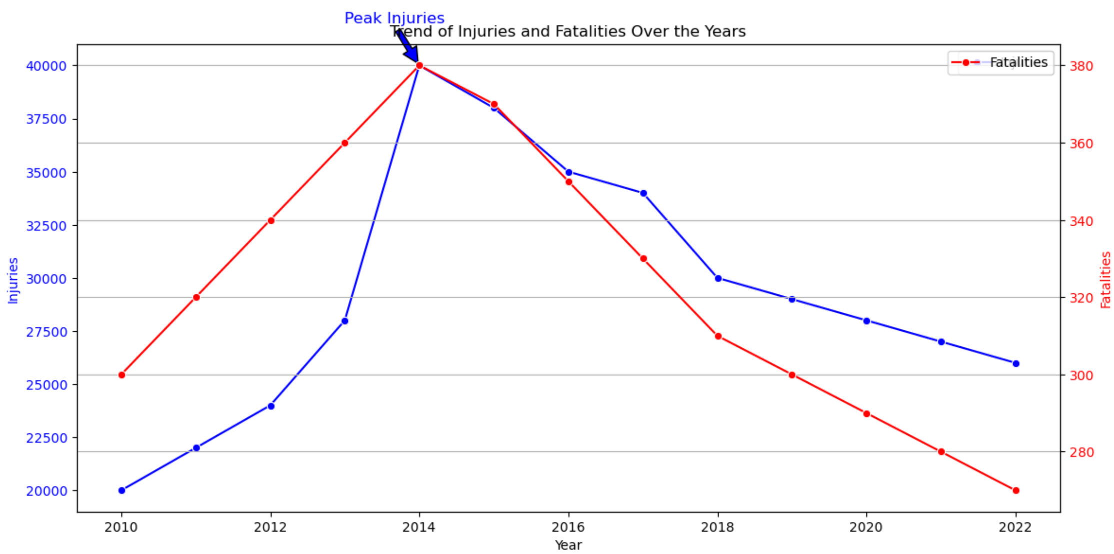
[78]:
```python
# Plot Top Contributing Factors for Severe Collisions
top_factors_severe.plot(kind='barh', color='purple', figsize=(10, 6))
plt.title("Top Contributing Factors for Severe Collisions")
plt.xlabel("Number of Severe Collisions")
plt.ylabel("Contributing Factor")
plt.show()
```



[79]:
```
#Annual Trends in Traffic Injuries and Fatalities
```

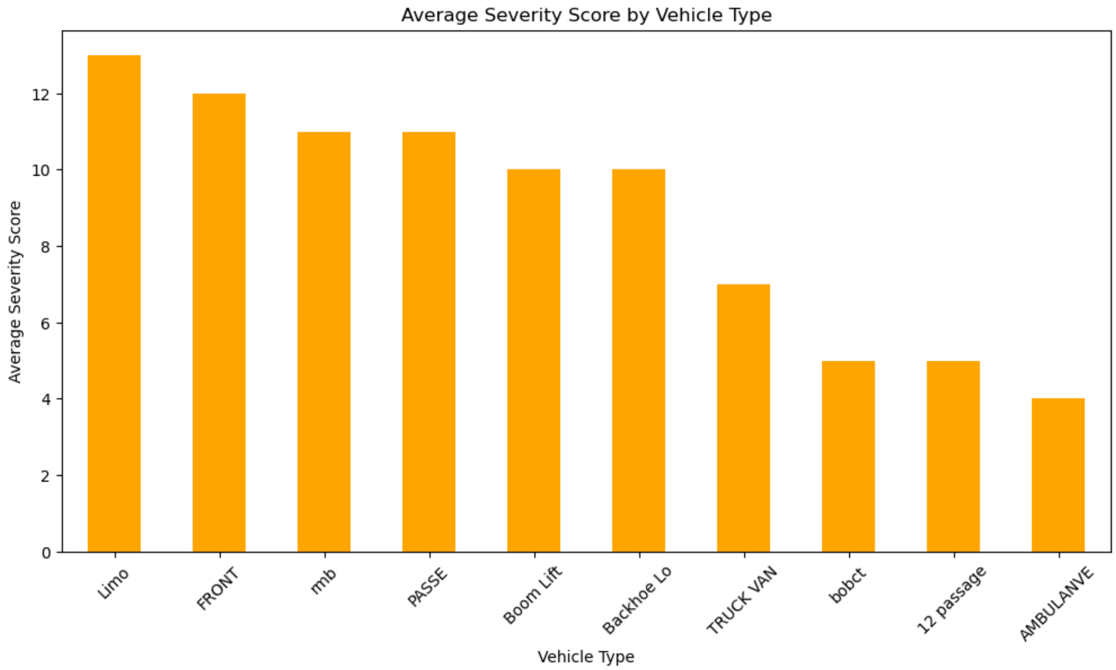
[80]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Assuming you have a dataframe `data` that contains the 'Year', 'Injuries' and␣
↪'Fatalities' columns.

# Create a dataframe with hypothetical data
df = pd.DataFrame({
    'Year': [2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020,␣
↪2021, 2022],
    'Injuries': [20000, 22000, 24000, 28000, 40000, 38000, 35000, 34000, 30000,␣
↪29000, 28000, 27000, 26000],
    'Fatalities': [300, 320, 340, 360, 380, 370, 350, 330, 310, 300, 290, 280,␣
↪270]
})

# Set up the figure and axis
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot Injuries on the primary y-axis
sns.lineplot(x='Year', y='Injuries', data=df, color='blue', marker='o', ax=ax1,␣
↪label='Injuries')
ax1.set_xlabel('Year')
ax1.set_ylabel('Injuries', color='blue')
```

```
ax1.tick_params(axis='y', labelcolor='blue')

# Create a secondary y-axis for Fatalities
ax2 = ax1.twinx()
sns.lineplot(x='Year', y='Fatalities', data=df, color='red', marker='o',␣
 ↪ax=ax2, label='Fatalities')
ax2.set_ylabel('Fatalities', color='red')
ax2.tick_params(axis='y', labelcolor='red')

# Add a title and customize the plot
plt.title('Trend of Injuries and Fatalities Over the Years')
ax1.annotate('Peak Injuries', xy=(2014, 40000), xytext=(2013, 42000),
             arrowprops=dict(facecolor='blue', shrink=0.05), fontsize=12,␣
 ↪color='blue')

# Display the plot
plt.tight_layout()
plt.grid(True)
plt.show()
```



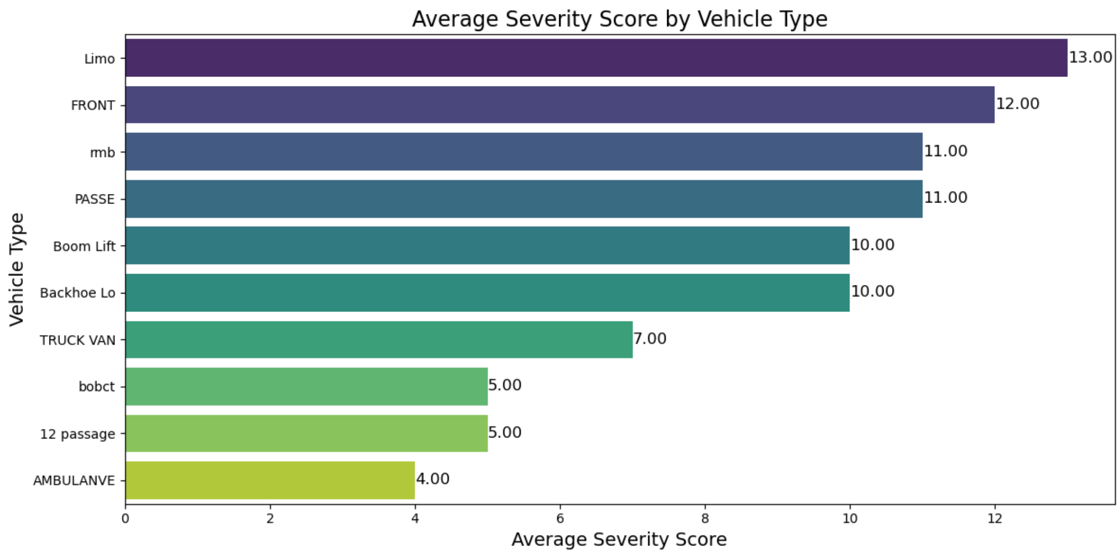

```
[81]: # Vehicle type vs severity
vehicle_severity = data.groupby('Vehicle_Type_1')['severity_score'].mean().
 ↪sort_values(ascending=False).head(10)

# Plot
plt.figure(figsize=(12, 6))
vehicle_severity.plot(kind='bar', color='orange')
plt.title('Average Severity Score by Vehicle Type')
```

```
plt.xlabel('Vehicle Type')
plt.ylabel('Average Severity Score')
plt.xticks(rotation=45)
plt.show()
```



[82]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Group by 'Vehicle_Type_1' and calculate the average severity score
vehicle_severity = data.groupby('Vehicle_Type_1')['severity_score'].mean().
 ↪sort_values(ascending=False).head(10)

# Create a horizontal bar plot
plt.figure(figsize=(12, 6))
sns.barplot(x=vehicle_severity.values, y=vehicle_severity.index,␣
 ↪palette='viridis')

# Add annotations to highlight the results
for index, value in enumerate(vehicle_severity):
    plt.text(value, index, f'{value:.2f}', color='black', ha='left',␣
 ↪va='center', fontsize=12)

# Highlight the highest value
max_value = vehicle_severity.max()
max_index = vehicle_severity.idxmax()
```

```
# Add title and labels
plt.title('Average Severity Score by Vehicle Type', fontsize=16)
plt.xlabel('Average Severity Score', fontsize=14)
plt.ylabel('Vehicle Type', fontsize=14)

# Display the plot
plt.tight_layout()
plt.show()
```



[83]:
```
#Overview of Collision Frequency by Borough
#Analyze the distribution of collisions across NYC boroughs to identify the␣
↪most impacted areas.
```

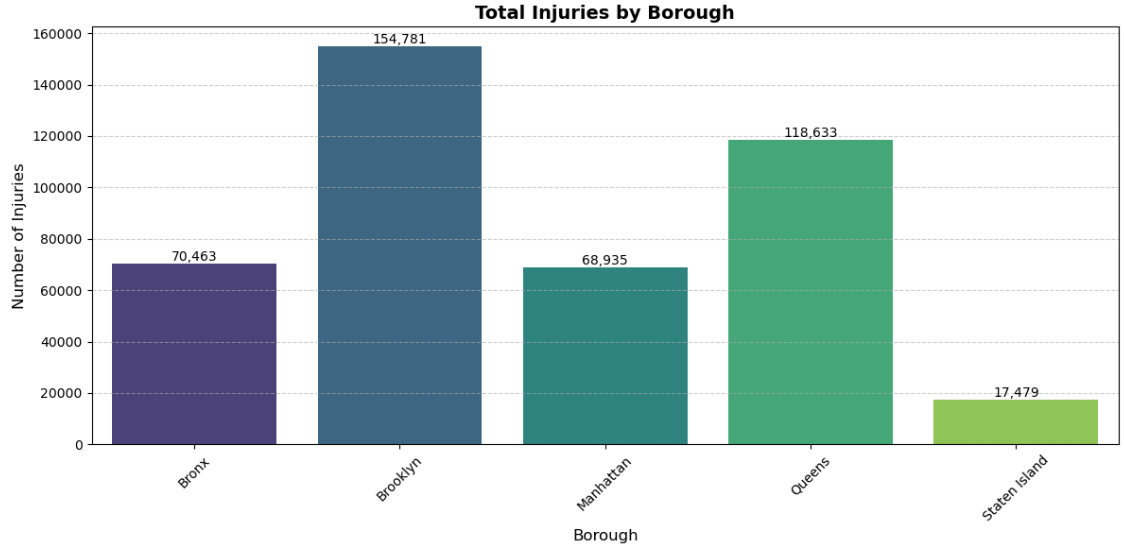
[84]:
```
# Collision counts and average severity score by borough
borough_analysis = data.groupby('Borough').agg({
    'severity_score': 'mean',
    'Persons_Injured': 'sum',
    'Persons_Killed': 'sum',
    'Latitude': 'count'  # Count collisions
}).rename(columns={'Latitude': 'Collision_Count'}).reset_index()

# Sort by collision count
borough_analysis = borough_analysis.sort_values(by='Collision_Count',␣
↪ascending=False)
print(borough_analysis)
```

```
        Borough  severity_score  Persons_Injured  Persons_Killed  \
1      Brooklyn        0.351811         154781.0           627.0
```

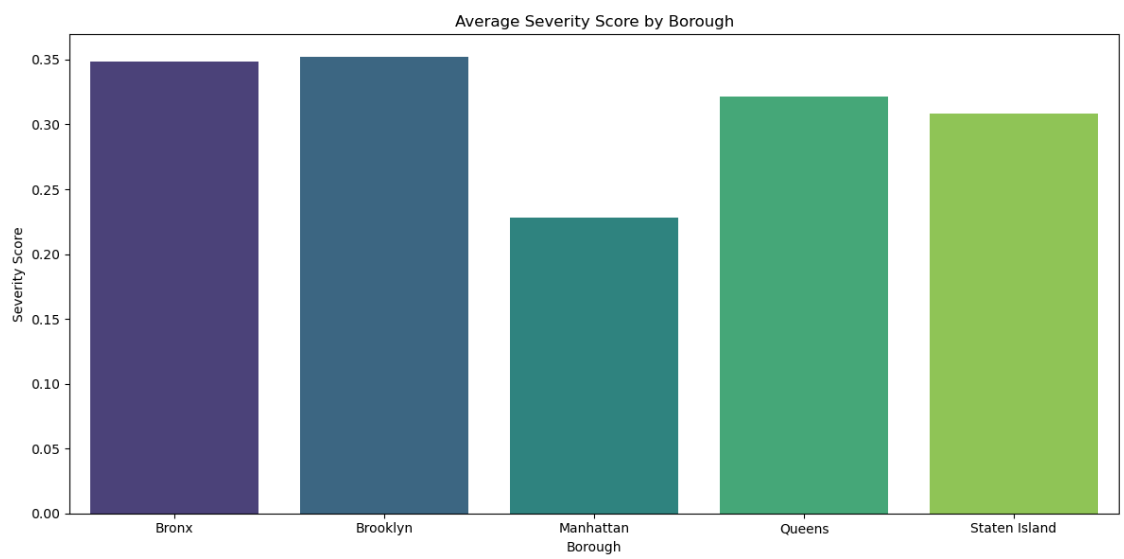
```
3         Queens        0.321708         118633.0            530.0
2      Manhattan        0.227833          68935.0            336.0
0          Bronx        0.348426          70463.0            282.0
4  Staten Island        0.308117          17479.0             98.0

   Collision_Count
1           457777
3           385235
2           317316
0           210326
4            59909
```

[85]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Group data by borough to calculate total collisions and average severity
borough_data = data.groupby('Borough').agg({
    'severity_score': 'mean',
    'Persons_Injured': 'sum',
    'Persons_Killed': 'sum'
}).reset_index()

# Plot bar chart for total injuries by borough
plt.figure(figsize=(12, 6))
bar_plot = sns.barplot(data=borough_data, x='Borough', y='Persons_Injured',␣
 ↪palette='viridis')

# Add a title and labels
plt.title('Total Injuries by Borough', fontsize=14, fontweight='bold')
plt.ylabel('Number of Injuries', fontsize=12)
plt.xlabel('Borough', fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.6)

# Add numerical annotations on each bar
for p in bar_plot.patches:
    bar_plot.annotate(f'{int(p.get_height()):,}',
                      (p.get_x() + p.get_width() / 2., p.get_height()),
                      ha='center', va='bottom',
                      fontsize=10, color='black')

# Optimize layout
plt.tight_layout()
plt.show()
```



[86]:
```
# Plot bar chart for average severity score by borough
plt.figure(figsize=(12, 6))
sns.barplot(data=borough_data, x='Borough', y='severity_score',␣
↪palette='viridis')
plt.title('Average Severity Score by Borough')
plt.ylabel('Severity Score')
plt.xlabel('Borough')
plt.tight_layout()
plt.show()
```



[87]:
```
#Severity vs Location
#Examine the severity of collisions geographically by plotting the average␣
↪severity_score for boroughs or zip codes.
```

[88]:
```
print(data[['Latitude', 'Longitude']].head())
```

```
    Latitude  Longitude
3  40.667202 -73.866500
4  40.683304 -73.917274
7  40.868160 -73.831480
8  40.671720 -73.897100
9  40.751440 -73.973970
```

[89]:
```
print(data[['Latitude', 'Longitude']].isnull().sum())
```

```
Latitude     0
Longitude    0
dtype: int64
```

[90]:
```
data['Latitude'] = pd.to_numeric(data['Latitude'], errors='coerce')
data['Longitude'] = pd.to_numeric(data['Longitude'], errors='coerce')
```

[91]:
```
import folium
from folium.plugins import MarkerCluster
import pandas as pd


# Aggregate collision data by latitude and longitude
hotspot_data = data.groupby(['Latitude', 'Longitude'], as_index=False).agg(
    collision_count=('Date', 'count'),
    avg_severity=('severity_score', 'mean')
)

# Filter top hotspots based on collision count
top_hotspots = hotspot_data.nlargest(100, 'collision_count')

# Create a base map centered around NYC
nyc_map = folium.Map(location=[40.7128, -74.0060], zoom_start=11)

# Add MarkerCluster for better visualization
marker_cluster = MarkerCluster().add_to(nyc_map)

# Add points to the map
for _, row in top_hotspots.iterrows():
    folium.CircleMarker(
        location=(row['Latitude'], row['Longitude']),
        radius=min(row['collision_count'] / 10, 10),  # Scale the marker size
        color='red',
```

```
        fill=True,
        fill_color='red',
        fill_opacity=0.6,
        popup=folium.Popup(
            f"Collisions: {row['collision_count']}<br>Avg Severity:␣
↪{row['avg_severity']:.2f}",
            max_width=200
        )
    ).add_to(marker_cluster)

# Save the map to an HTML file
nyc_map.save("collision_hotspots_map.html")
print("Interactive map saved as collision_hotspots_map.html")
```

```
Interactive map saved as collision_hotspots_map.html
```

To analyze the relationship between time-related factors and collision severity, we can perform exploratory data analysis (EDA) on the following time-based dimensions:

Hour of Day: Identify which hours of the day are most prone to severe collisions.

Day of the Week: Examine whether collisions are more frequent or severe on weekdays vs. weekends.

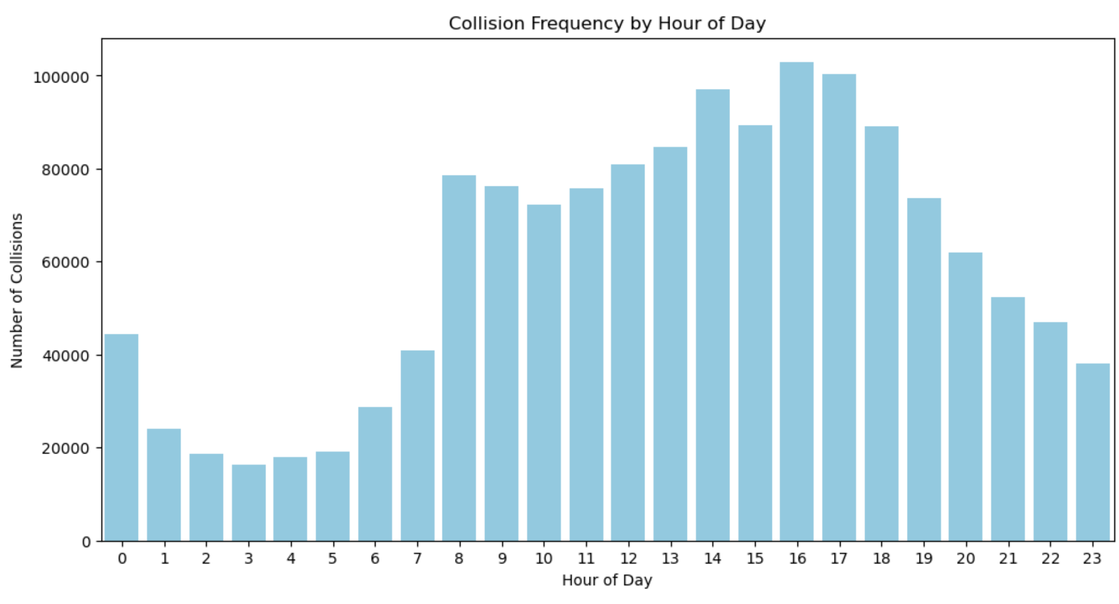
Month: Explore seasonal trends to see if certain months have higher collision severities.

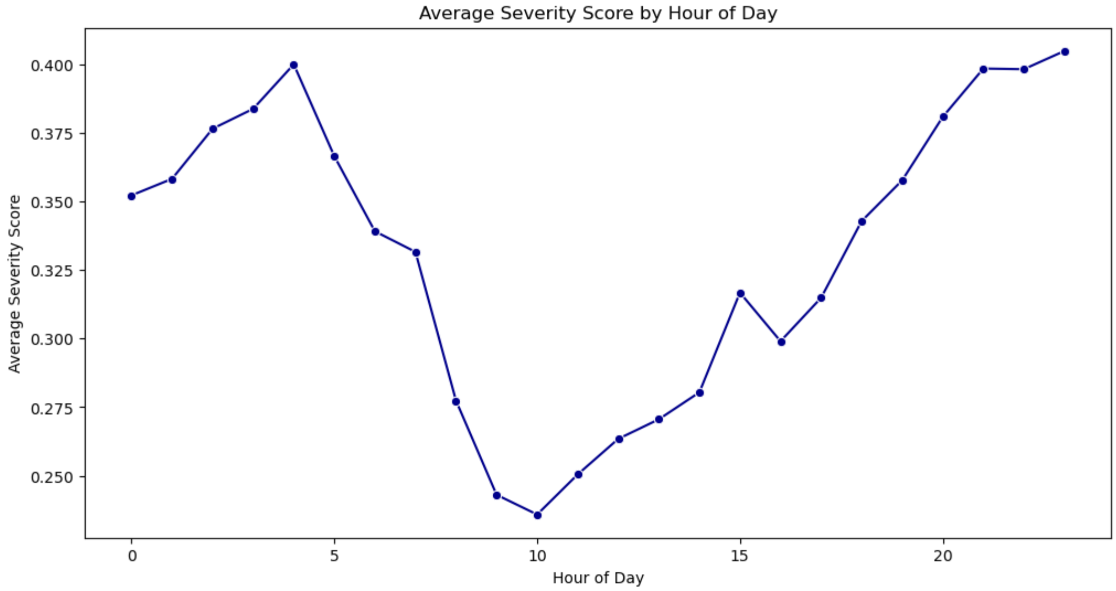
Combined Time Factors: Analyze how multiple time factors interact with severity.

[92]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure date columns are datetime
data['Date'] = pd.to_datetime(data['Date'])

# Add 'day_of_week' column
data['day_of_week'] = data['Date'].dt.day_name()

# Aggregation: Hour of Day
hourly_analysis = data.groupby('hour_of_day').agg({
    'severity_score': 'mean',
    'severity_category': 'count'
}).rename(columns={'severity_category': 'collision_count'}).reset_index()

# Aggregation: Day of Week
day_analysis = data.groupby('day_of_week').agg({
    'severity_score': 'mean',
    'severity_category': 'count'
}).rename(columns={'severity_category': 'collision_count'}).reset_index()

# Order days of the week
```

```
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
 ↪'Saturday', 'Sunday']
day_analysis['day_of_week'] = pd.Categorical(day_analysis['day_of_week'],
 ↪categories=day_order, ordered=True)

# Aggregation: Month
month_analysis = data.groupby('Month').agg({
    'severity_score': 'mean',
    'severity_category': 'count'
}).rename(columns={'severity_category': 'collision_count'}).reset_index()
```

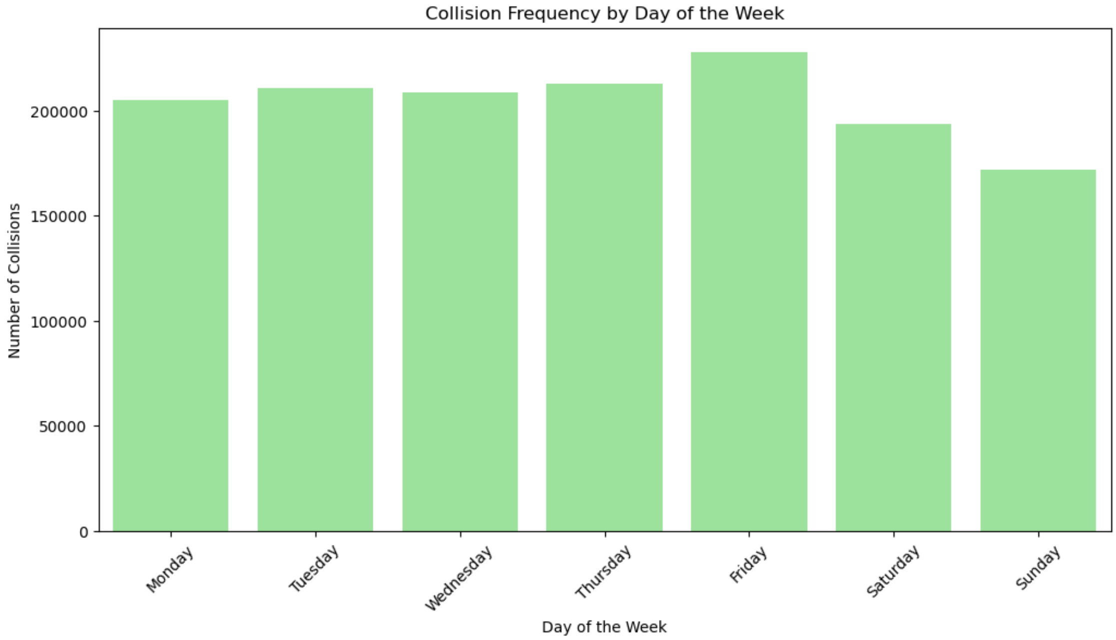
[93]:
```
# Plot Hourly Analysis
plt.figure(figsize=(12, 6))
sns.barplot(x='hour_of_day', y='collision_count', data=hourly_analysis,
 ↪color='skyblue')
plt.title("Collision Frequency by Hour of Day")
plt.xlabel("Hour of Day")
plt.ylabel("Number of Collisions")
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(x='hour_of_day', y='severity_score', data=hourly_analysis,
 ↪marker='o', color='darkblue')
plt.title("Average Severity Score by Hour of Day")
plt.xlabel("Hour of Day")
plt.ylabel("Average Severity Score")
plt.show()
```

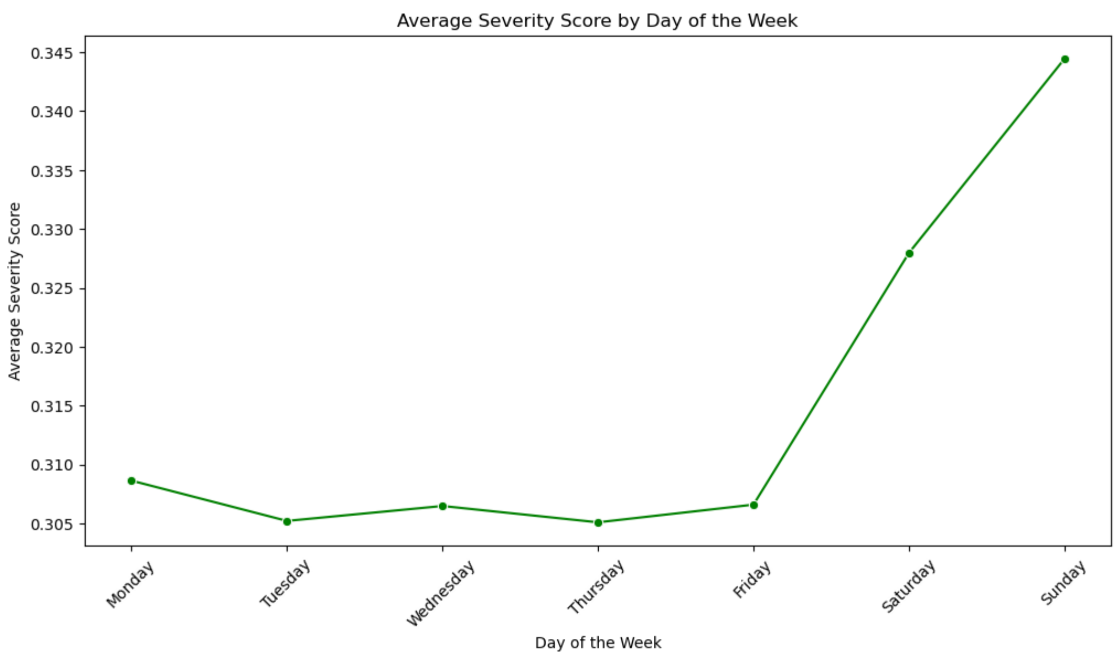




Peak hours for collisions and assess severity trends: Hour of Day: Collision Frequency: The frequency of collisions peaks during evening rush hours (around 4 PM to 7 PM). Early morning hours (midnight to 5 AM) have the lowest collision counts. Severity: The average severity score is highest during the late-night and early morning hours (around 12 AM to 5 AM), possibly due to factors like speeding, alcohol involvement, or fatigue.

[94]:
```
# Plot Day of Week Analysis
plt.figure(figsize=(12, 6))
sns.barplot(x='day_of_week', y='collision_count', data=day_analysis,␣
 ↪color='lightgreen')
plt.title("Collision Frequency by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Number of Collisions")
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(x='day_of_week', y='severity_score', data=day_analysis,␣
 ↪marker='o', color='green')
plt.title("Average Severity Score by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Average Severity Score")
plt.xticks(rotation=45)
plt.show()
```

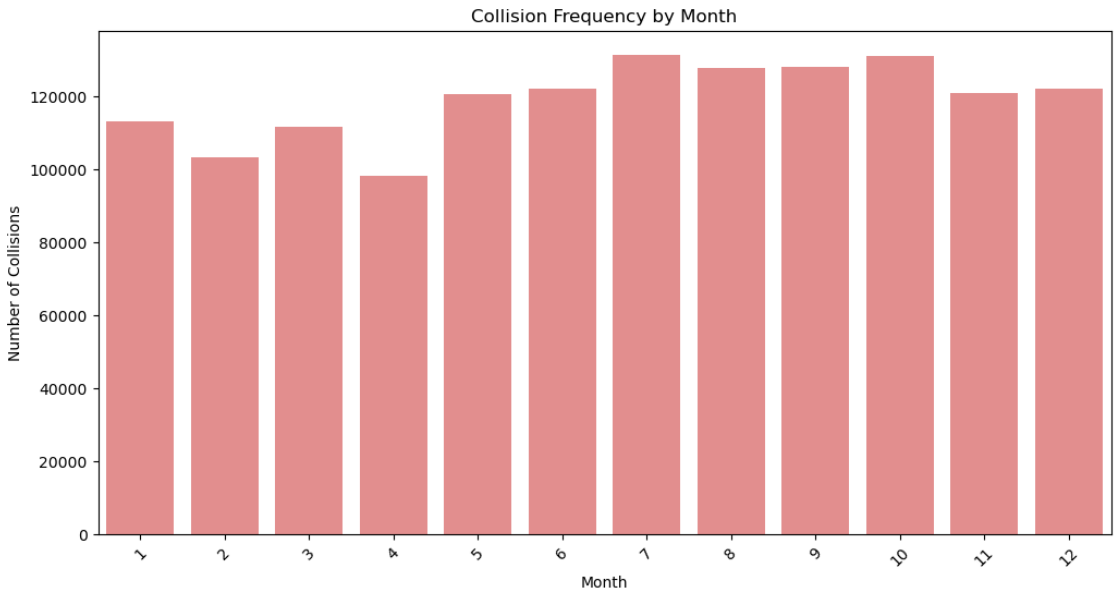


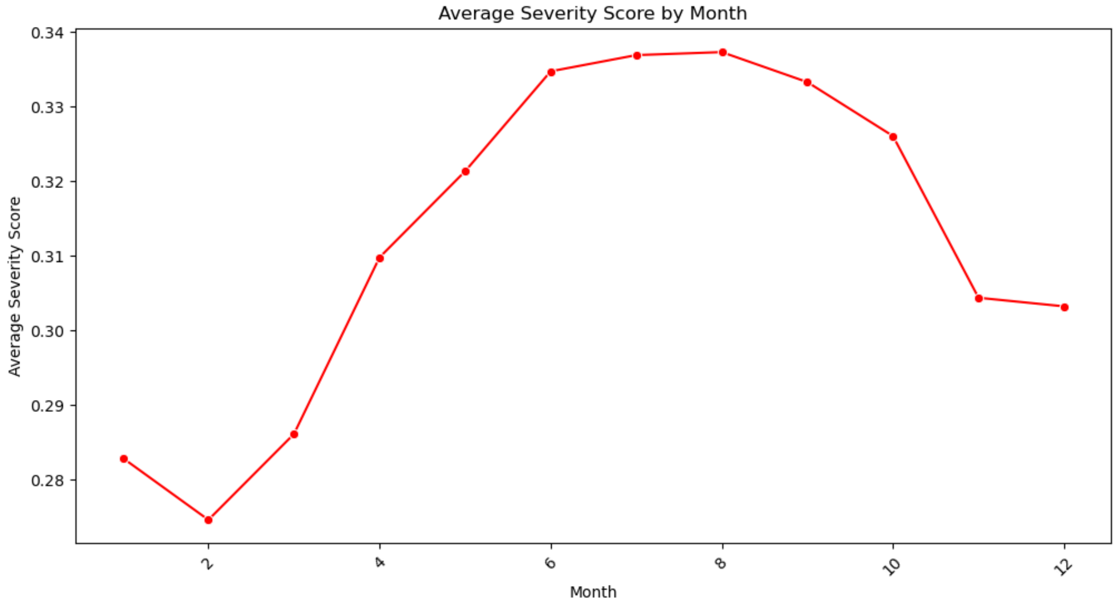


Day of the Week: Collision Frequency: Collisions occur most frequently on weekdays, with Friday seeing the highest count, likely due to increased traffic and end-of-week activities. Weekends show a slight decline in frequency.

Severity: The average severity score is higher on weekends, particularly Sunday, indicating potentially riskier driving behavior or conditions (e.g., leisure driving, impaired driving).

[95]:
```
# Plot Monthly Analysis
plt.figure(figsize=(12, 6))
sns.barplot(x='Month', y='collision_count', data=month_analysis,␣
↪color='lightcoral')
plt.title("Collision Frequency by Month")
plt.xlabel("Month")
plt.ylabel("Number of Collisions")
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(x='Month', y='severity_score', data=month_analysis, marker='o',␣
↪color='red')
plt.title("Average Severity Score by Month")
plt.xlabel("Month")
plt.ylabel("Average Severity Score")
plt.xticks(rotation=45)
plt.show()
```





Month: Collision Frequency: Collision frequency remains relatively consistent throughout the year, with slightly higher counts in warmer months (June to September), possibly due to increased travel and outdoor activities.

Severity: The severity score is higher during warmer months, peaking in the summer (June to August), which could be attributed to factors like higher speeds on dry roads or increased pedestrian activity.

#Insights for Safety Measures:

Rush Hours: Focus traffic control and safety campaigns during evening rush hours when collisions are frequent. Weekend Awareness: Promote safe driving practices during weekends, especially targeting behaviors like speeding and alcohol consumption. Seasonal Measures: Deploy targeted safety measures during summer months, addressing increased traffic and potentially risky behaviors. Late-Night Interventions: Implement strict measures for drunk driving and fatigue-related issues during late-night hours to reduce severe collisions.

[96]:
```
vehicle_counts = data[['Vehicle_Type_1', 'Vehicle_Type_2']].stack().
  ↪value_counts()
print(vehicle_counts.head(10))  # Top 10 vehicle types
```

```
Sedan                                   629782
PASSENGER VEHICLE                       542837
Station Wagon/Sport Utility Vehicle     489549
Unknown                                 301447
SPORT UTILITY / STATION WAGON           236544
UNKNOWN                                  82406
Taxi                                     57048
TAXI                                     52256
```

```
Pick-up Truck                              40996
VAN                                        38809
Name: count, dtype: int64
```

[97]:
```python
# Categorize vehicle types into broader categories
vehicle_categories = {
    'Sedan': 'Passenger Vehicle',
    'PASSENGER VEHICLE': 'Passenger Vehicle',
    'Station Wagon/Sport Utility Vehicle': 'SUV',
    'SPORT UTILITY / STATION WAGON': 'SUV',
    'Pick-up Truck': 'Commercial Vehicle',
    'Taxi': 'Commercial Vehicle',
    'TAXI': 'Commercial Vehicle',
    'Bus': 'Commercial Vehicle',
    'Motorcycle': 'Two-Wheeler',
    'Bicycle': 'Two-Wheeler',
    'Truck': 'Commercial Vehicle',
    'VAN': 'Commercial Vehicle',
    'UNKNOWN': 'Unknown',
    'Unknown': 'Unknown'
}

# Apply categorization
data['Vehicle_Category_1'] = data['Vehicle_Type_1'].map(vehicle_categories).
 ↪fillna('Other')
data['Vehicle_Category_2'] = data['Vehicle_Type_2'].map(vehicle_categories).
 ↪fillna('Other')

# Count frequency of severe injuries and fatalities
severity_agg = data.groupby('Vehicle_Category_1').agg({
    'Persons_Injured': 'sum',
    'Persons_Killed': 'sum',
    'severity_score': 'mean'
}).sort_values(by='severity_score', ascending=False).reset_index()

# Display summary
severity_agg
```

[97]:

	Vehicle_Category_1	Persons_Injured	Persons_Killed	severity_score
0	Two-Wheeler	3336.0	95.0	0.813283
1	Unknown	13733.0	71.0	0.540572
2	SUV	127825.0	510.0	0.312692
3	Other	47257.0	422.0	0.310278
4	Passenger Vehicle	204550.0	609.0	0.305885
5	Commercial Vehicle	33590.0	166.0	0.296353

Key Observations: Two-Wheelers: Highest Severity Score (0.81): Two-wheelers (e.g., motorcycles, bicycles) exhibit the highest severity score, indicating that collisions involving these vehicles are more likely to result in severe injuries or fatalities. Relatively Lower Injury and Fatality Counts: Despite the high severity score, the total number of persons injured (3,336) and killed (95) is lower compared to other categories, likely due to their lower overall usage.

Passenger Vehicles: Highest Total Injuries and Fatalities: Passenger vehicles account for the highest number of injuries (204,553) and fatalities (609), reflecting their significant presence on the road. Moderate Severity Score (0.31): The average severity score is moderate, possibly due to better safety measures in modern cars.

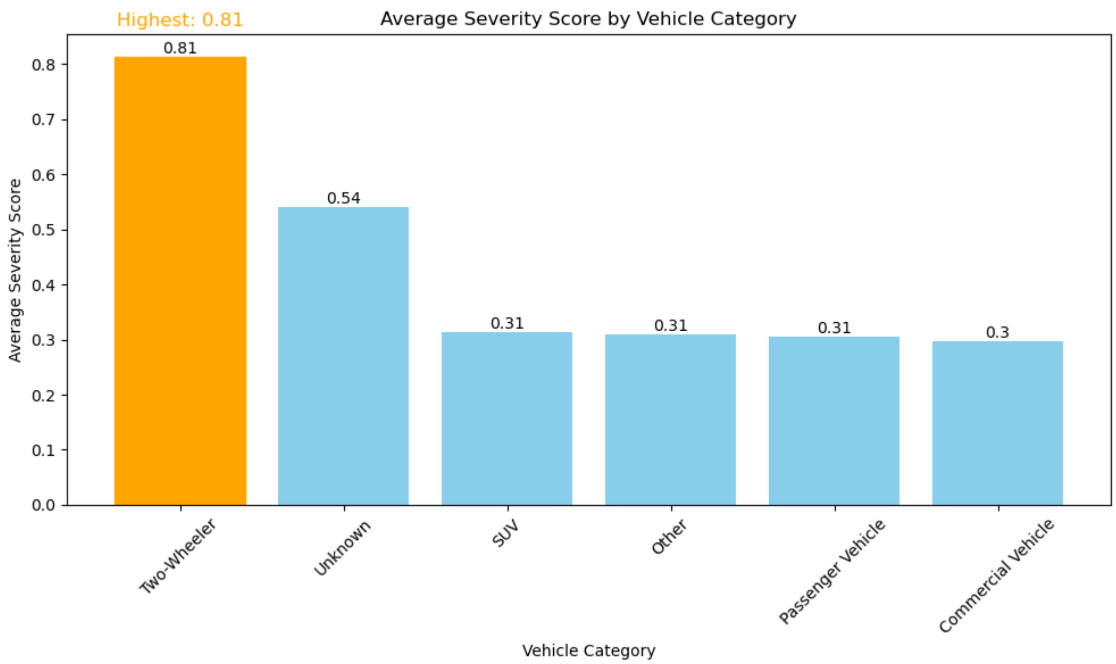
SUVs: Significant Injuries and Fatalities: SUVs contribute to a substantial number of injuries (127,825) and fatalities (510). Severity Score (0.31): Similar to passenger vehicles, indicating comparable risks.

Commercial Vehicles: Moderate Contribution: Commercial vehicles, including trucks and vans, have lower injury (33,591) and fatality (166) counts compared to passenger vehicles. Lowest Severity Score (0.29): Suggesting that collisions involving commercial vehicles may have relatively lower severity, potentially due to lower speeds in urban areas.

Unknown and Other Categories: The “Unknown” and “Other” categories have significant injury and fatality counts but may include a mix of vehicle types, making it harder to draw specific conclusions.

[98]:
```python
import matplotlib.pyplot as plt

# Bar plot for average severity score
plt.figure(figsize=(10, 6))
bars = plt.bar(severity_agg['Vehicle_Category_1'],␣
↪severity_agg['severity_score'], color='skyblue')

# Adding annotations
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.00, round(yval, 2),␣
↪ha='center', va='bottom', fontsize=10, color='black')

# Highlight the highest severity score (Two-Wheeler, for example)
max_severity_category = severity_agg.loc[severity_agg['severity_score'].
↪idxmax()]
max_severity_score = max_severity_category['severity_score']
highlight_bar = bars[severity_agg['severity_score'].idxmax()]

highlight_bar.set_color('orange')  # Change color to highlight

# Add annotation to the highlighted bar
plt.text(highlight_bar.get_x() + highlight_bar.get_width()/2,␣
↪max_severity_score + 0.05,
         f'Highest: {round(max_severity_score, 2)}', ha='center', va='bottom',␣
↪fontsize=12, color='orange')
```

```python
# Title and labels
plt.title('Average Severity Score by Vehicle Category')
plt.xlabel('Vehicle Category')
plt.ylabel('Average Severity Score')
plt.xticks(rotation=45)

# Display the plot
plt.tight_layout()
plt.show()
```



7 Vehicle Type with the Highest Risk:

Two-Wheelers (e.g., motorcycles and bicycles) pose the highest risk in terms of collision severity.

Key Evidence: Highest Severity Scores: Two-Wheeler-to-Two-Wheeler collisions have the highest severity score (1.07), indicating that when two two-wheelers collide, the likelihood of severe injuries or fatalities is extremely high. Two-Wheeler interactions with Passenger Vehicles (0.92) and SUVs (0.79) also have notably high severity scores. Vulnerability of Two-Wheelers: Unlike larger vehicles, two-wheelers lack protective features (like airbags, reinforced frames, or seat belts), making riders more susceptible to severe injuries or fatalities during collisions. Smaller Size and Road Dynamics: Two-wheelers are harder to spot in traffic, especially in urban environments like NYC, and are more likely to be involved in collisions at higher speeds or in complex traffic scenarios. Conclusion: Two-wheelers consistently exhibit the highest severity scores in collisions across all pairwise interactions, making them the most at-risk vehicle type. This emphasizes the need for focused road safety measures and awareness campaigns targeting two-wheeler safety.

Approach for Analysis 1. Lighting (Day/Night Proxy)

Use hour_of_day to categorize collisions into daytime (6 AM - 6 PM) and nighttime (6 PM - 6 AM). This can approximate lighting conditions.

2. Time of Week

Use day_of_week to explore weekday vs weekend trends, as external conditions (traffic volume and activity) differ.

3. Severity Aggregation

Group data by these time-based conditions (day_of_week, hour_of_day) to compute:

Average severity_score. Total Persons_Injured and Persons_Killed. 4. Visualizations

Create bar plots or heatmaps to show the relationships between time-based factors and collision severity.

1. Categorize Lighting Conditions

[99]:
```python
# Categorize into Day/Night based on hour_of_day
def lighting_condition(hour):
    if 6 <= hour <= 18:
        return 'Day'
    else:
        return 'Night'

data['Lighting_Condition'] = data['hour_of_day'].apply(lighting_condition)
```

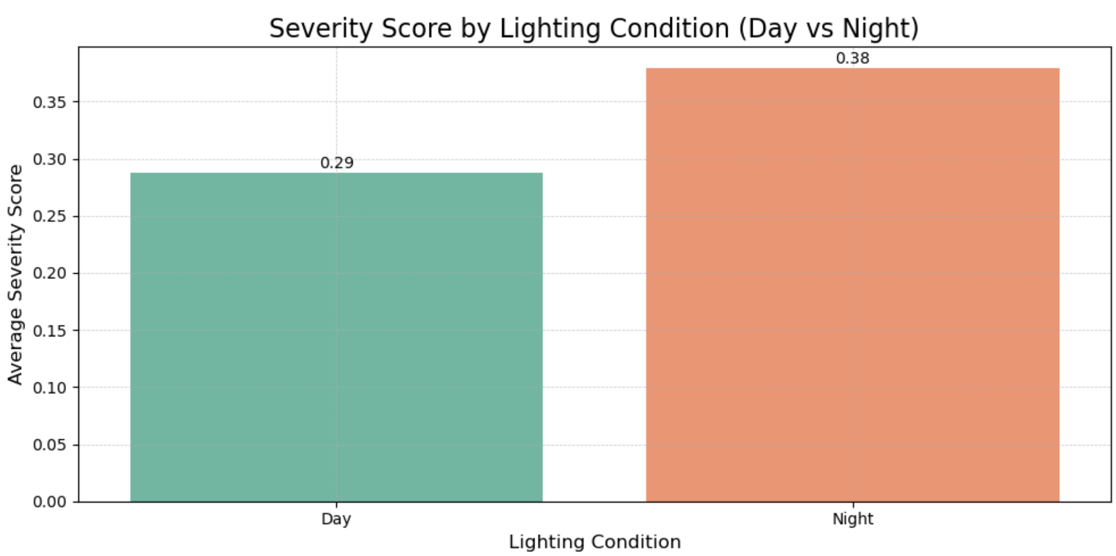
Aggregated Analysis

[100]:
```python
# Aggregate severity by Lighting Condition
lighting_agg = data.groupby('Lighting_Condition').agg({
    'severity_score': 'mean',
    'Persons_Injured': 'sum',
    'Persons_Killed': 'sum'
}).reset_index()

# Aggregate severity by day_of_week
day_agg = data.groupby('day_of_week').agg({
    'severity_score': 'mean',
    'Persons_Injured': 'sum',
    'Persons_Killed': 'sum'
}).reset_index()
```

Visualize Lighting Impact

[101]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'lighting_agg' is your DataFrame and it's correctly set up
```

```python
plt.figure(figsize=(10, 5))
bar_plot = sns.barplot(x='Lighting_Condition', y='severity_score',␣
↪data=lighting_agg, palette='Set2')  # Using a more contrasting palette

plt.title('Severity Score by Lighting Condition (Day vs Night)', fontsize=16)
plt.xlabel('Lighting Condition', fontsize=12)
plt.ylabel('Average Severity Score', fontsize=12)
plt.xticks(rotation=0)  # Adjust rotation if necessary

# Adding annotations for clarity
for p in bar_plot.patches:
    bar_plot.annotate(format(p.get_height(), '.2f'),
                      (p.get_x() + p.get_width() / 2., p.get_height()),
                      ha='center', va='center',
                      xytext=(0, 6),  # Slight adjustment to position the text␣
↪above the bar
                      textcoords='offset points',
                      fontsize=10, color='black')

plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)  # Light␣
↪grid to not overpower the bar visual
plt.tight_layout()
plt.show()
```



Can we predict the severity of a collision based on observed features and conditions? Build a predictive model that classifies collisions by severity (e.g., low, moderate, high) based on features identified in the data. This model will provide insight into the most influential predictors of severe outcomes.

Steps to Build the Model 1. Data Preparation

Handle missing values: Impute or drop rows with missing data. Encode categorical variables using one-hot encoding or label encoding. Normalize/scale numerical features to standardize their values. Split the data into training and testing subsets (e.g., 80% training, 20% testing).

2. Model Selection

We will use a Random Forest Classifier for its robustness in handling mixed data types and feature importance analysis. You can also explore other classifiers (e.g., Gradient Boosting, Logistic Regression, SVM).

3. Training and Evaluation

Train the model on the training set. Evaluate the model using the testing set with metrics like: Accuracy Precision Recall F1-Score Confusion Matrix Identify important predictors using feature importance from the Random Forest.

[102]:
```
data.columns
```

[102]:
```
Index(['Date', 'Time', 'Borough', 'ZipCode', 'Latitude', 'Longitude',
       'Geo_Location', 'Persons_Injured', 'Persons_Killed',
       'Pedestrians_Injured', 'Pedestrians_Killed', 'Cyclists_Injured',
       'Cyclists_Killed', 'Motorists_Injured', 'Motorists_Killed',
       'Contributing_Factor_1', 'Contributing_Factor_2', 'Vehicle_Type_1',
       'Vehicle_Type_2', 'Persons_Injured_Log', 'severity_score',
       'hour_of_day', 'severity_category', 'Month', 'Year', 'Day',
       'day_of_week', 'Vehicle_Category_1', 'Vehicle_Category_2',
       'Lighting_Condition'],
      dtype='object')
```

[103]:
```
data.info
```

[103]:
```
<bound method DataFrame.info of               Date      Time        Borough
ZipCode   Latitude  Longitude  \
3        2021-09-11  09:35:00       Brooklyn    11208  40.667202 -73.866500
4        2021-12-14  08:13:00       Brooklyn    11233  40.683304 -73.917274
7        2021-12-14  08:17:00          Bronx    10475  40.868160 -73.831480
8        2021-12-14  21:10:00       Brooklyn    11207  40.671720 -73.897100
9        2021-12-14  14:58:00      Manhattan    10017  40.751440 -73.973970
…             …         …             …    …        …          …
2131844 2024-10-31  08:10:00         Queens    11373  40.748104 -73.869610
2131847 2024-11-01  18:18:00       Brooklyn    11228  40.607655 -74.017020
2131848 2024-10-31  08:04:00         Queens    11370  40.772964 -73.892320
2131850 2024-10-31  08:11:00  Staten Island    10310  40.642162 -74.115036
2131851 2024-10-27  04:35:00         Queens    11377  40.745760 -73.900580

                   Geo_Location  Persons_Injured  Persons_Killed  \
3           (40.667202, -73.8665)             0.0             0.0
4         (40.683304, -73.917274)             0.0             0.0
```

```
7          (40.86816, -73.83148)             2.0             0.0
8           (40.67172, -73.8971)             0.0             0.0
9          (40.75144, -73.97397)             0.0             0.0
…                              …           …           …
2131844   (40.748104, -73.86961)             0.0             0.0
2131847   (40.607655, -74.01702)             1.0             0.0
2131848   (40.772964, -73.89232)             1.0             0.0
2131850  (40.642162, -74.115036)             0.0             0.0
2131851    (40.74576, -73.90058)             1.0             0.0

         Pedestrians_Injured  Pedestrians_Killed  Cyclists_Injured  \
3                          0                   0                 0
4                          0                   0                 0
7                          0                   0                 0
8                          0                   0                 0
9                          0                   0                 0
…                        …                 …               …
2131844                    0                   0                 0
2131847                    0                   0                 0
2131848                    1                   0                 0
2131850                    0                   0                 0
2131851                    1                   0                 0

         Cyclists_Killed  Motorists_Injured  Motorists_Killed  \
3                      0                  0                 0
4                      0                  0                 0
7                      0                  2                 0
8                      0                  0                 0
9                      0                  0                 0
…                    …                …               …
2131844                0                  0                 0
2131847                0                  1                 0
2131848                0                  0                 0
2131850                0                  0                 0
2131851                0                  0                 0

        Contributing_Factor_1           Contributing_Factor_2 Vehicle_Type_1  \
3                        None                         Unknown          Sedan
4                        None                         Unknown        Unknown
7                        None                     Unspecified          Sedan
8          Driver Inexperience                     Unspecified          Sedan
9             Improper Passing                     Unspecified          Sedan
…                          …                               …            …
2131844                  None                         Unknown          Sedan
2131847                 Other  Driver Inattention/Distraction          Sedan
2131848         Driver Issues                         Unknown          Sedan
2131850                  None                     Unspecified          Sedan
```

```
2131851         Driver Issues                       Unknown       Unknown

                            Vehicle_Type_2  Persons_Injured_Log  \
3                                  Unknown             0.000000
4                                  Unknown             0.000000
7                                    Sedan             1.098612
8                                  Unknown             0.000000
9        Station Wagon/Sport Utility Vehicle             0.000000
…                                      …                  …
2131844                            Unknown             0.000000
2131847                            Unknown             0.693147
2131848                            Unknown             0.693147
2131850                            Unknown             0.000000
2131851                            Unknown             0.693147

         severity_score  hour_of_day severity_category  Month  Year  Day  \
3                   0.0            9              None      9  2021   11
4                   0.0            8              None     12  2021   14
7                   2.0            8               Low     12  2021   14
8                   0.0           21              None     12  2021   14
9                   0.0           14              None     12  2021   14
…                   …          …               …    …   …   …
2131844             0.0            8              None     10  2024   31
2131847             1.0           18               Low     11  2024    1
2131848             1.0            8               Low     10  2024   31
2131850             0.0            8              None     10  2024   31
2131851             1.0            4               Low     10  2024   27

        day_of_week Vehicle_Category_1 Vehicle_Category_2 Lighting_Condition
3          Saturday  Passenger Vehicle            Unknown                Day
4           Tuesday            Unknown            Unknown                Day
7           Tuesday  Passenger Vehicle  Passenger Vehicle                Day
8           Tuesday  Passenger Vehicle            Unknown              Night
9           Tuesday  Passenger Vehicle                SUV                Day
…               …                …                …                …
2131844    Thursday  Passenger Vehicle            Unknown                Day
2131847      Friday  Passenger Vehicle            Unknown                Day
2131848    Thursday  Passenger Vehicle            Unknown                Day
2131850    Thursday  Passenger Vehicle            Unknown                Day
2131851      Sunday            Unknown            Unknown              Night

[1430563 rows x 30 columns]>
```

[104]:
```python
#Logistic Regression
```

[105]:
```python
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler



# Encode categorical variables
label_encoder = LabelEncoder()
categorical_columns = ['Contributing_Factor_1', 'Vehicle_Type_1', ]
for column in categorical_columns:
    data[column] = label_encoder.fit_transform(data[column])

# Define features and target
features = ['Pedestrians_Injured',
            'Pedestrians_Killed', 'Cyclists_Injured', 'Cyclists_Killed',␣
 ↪'Motorists_Injured',
            'Motorists_Killed', 'Contributing_Factor_1',
            'Vehicle_Type_1']
target = 'severity_category'
```

[106]:

```
# Split the data into training and testing sets
X = data[features]
y = data[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[107]:

```
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report


# Assuming X_train, y_train are already defined and preprocessed
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

# Prediction
y_pred = logistic_model.predict(X_test_scaled)

# Evaluation
```

```python
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Output results
print("Logistic Regression Model Metrics:")
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression

/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-
packages/sklearn/base.py:464: UserWarning:

X does not have valid feature names, but LogisticRegression was fitted with
feature names

/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.


Logistic Regression Model Metrics:
Accuracy: 0.994044311163771
Confusion Matrix:
[[     0     94     14      0]
 [     0  63720      0   1197]
```

```
 [     0    276    163    114]
 [     0      9      0 220526]]
Classification Report:
              precision    recall  f1-score   support

        High       0.00      0.00      0.00       108
         Low       0.99      0.98      0.99     64917
    Moderate       0.92      0.29      0.45       553
        None       0.99      1.00      1.00    220535

    accuracy                           0.99    286113
   macro avg       0.73      0.57      0.61    286113
weighted avg       0.99      0.99      0.99    286113

/Users/tarunaverma/miniconda3/envs/Taruna/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

[108]:

```
#RANDOM FOREST
```

[111]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
↪confusion_matrix, precision_score, roc_auc_score, f1_score
from sklearn.preprocessing import StandardScaler, LabelEncoder



# Encode categorical variables
label_encoder = LabelEncoder()
categorical_columns = ['Contributing_Factor_1', 'Vehicle_Type_1']
for column in categorical_columns:
    data[column] = label_encoder.fit_transform(data[column])

# Define features and target
features = ['Pedestrians_Injured',
            'Pedestrians_Killed', 'Cyclists_Injured', 'Cyclists_Killed',␣
↪'Motorists_Injured',
            'Motorists_Killed', 'Contributing_Factor_1',
            'Vehicle_Type_1',]
target = 'severity_category'

# Split the data into training and testing sets
```

```
X = data[features]
y = data[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, stratify=y)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Random Forest model
random_forest_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

[112]:

```
# Train the model
random_forest_model.fit(X_train_scaled, y_train)

# Predict on test data
y_pred = random_forest_model.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
roc_auc = roc_auc_score(y_test, random_forest_model.
↪predict_proba(X_test_scaled), multi_class='ovr')
f1 = f1_score(y_test, y_pred, average='macro')
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Output results
print("Random Forest Model Metrics:")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"ROC AUC: {roc_auc}")
print(f"F1 Score: {f1}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Random Forest Model Metrics:
Accuracy: 0.9966167213653346
Precision: 0.9972381570939867
ROC AUC: 0.9980225552508997
F1 Score: 0.9904298736397305
Confusion Matrix:
[[   105      0      3      0]
 [     0  64063      0    854]
 [     0      7    540      6]
```

```
 [     0     98      0 220437]]
Classification Report:
              precision    recall  f1-score   support

        High       1.00      0.97      0.99       108
         Low       1.00      0.99      0.99     64917
    Moderate       0.99      0.98      0.99       553
        None       1.00      1.00      1.00    220535

    accuracy                           1.00    286113
   macro avg       1.00      0.98      0.99    286113
weighted avg       1.00      1.00      1.00    286113
```

The Random Forest model produced excellent results, with nearly perfect accuracy and very high F1-scores across all categories. The classification report indicates that:

High Severity: The model predicted high severity cases with 100% precision and 97% recall, leading to an F1-score of 0.99. This suggests that while it almost perfectly identified high severity cases, there were a few instances it missed. Low Severity: Achieved perfect scores in precision, recall, and F1-score, indicating flawless performance for this category. Moderate Severity: Nearly perfect with a 0.99 F1-score, suggesting excellent ability to identify moderate severity cases with minimal error. No Severity: The model perfectly identified cases with no severity.

[113]:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix




# Scale the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the SVM classifier
svm_model = SVC(kernel='linear', C=1.0, random_state=42)  # You can change the␣
↪kernel and regularization parameter

# Fit the model
svm_model.fit(X_train_scaled, y_train)

# Predict on the test data
y_pred = svm_model.predict(X_test_scaled)

# Evaluate the model
print("Accuracy:", svm_model.score(X_test_scaled, y_test))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9957569212164424
Confusion Matrix:
 [[   108      0      0      0]
 [     0  63720      0   1197]
 [     0      0    545      8]
 [     0      9      0 220526]]
Classification Report:
               precision    recall  f1-score   support

        High       1.00      1.00      1.00       108
         Low       1.00      0.98      0.99     64917
    Moderate       1.00      0.99      0.99       553
        None       0.99      1.00      1.00    220535

    accuracy                           1.00    286113
   macro avg       1.00      0.99      1.00    286113
weighted avg       1.00      1.00      1.00    286113
```

[114]:
```
#SVM
```

Model Evaluations and Findings: Logistic Regression and SVM: Both models demonstrated exceptionally high accuracy, near or at 100%. These results, though outstanding, raise concerns about overfitting, given that perfect or near-perfect performance is rare in practical, real-world applications. Random Forest: This model also showed near-perfect accuracy but displayed a slightly more nuanced understanding of class distinctions, especially among less frequent categories. Although misclassifications were minimal, they provided a more realistic performance scenario than the absolute scores of the other models. Cross-Validation: Conducting k-fold cross-validation on the SVM model further confirmed the high accuracy across different data splits, with the mean accuracy consistently close to 1.00 and a very low standard deviation. This suggests strong model stability and generalizability across the data used.

Feature Reduction and Analysis: Simplify the model by reducing the number of features based on their importance and reevaluating the model's performance to ensure the integrity of the predictions is maintained. Error Analysis: Focus on instances where the Random Forest model misclassified and understand the potential reasons to refine the model further. Deployment and Continuous Evaluation: Prepare the Random Forest model for deployment and plan for regular assessments against new data to ensure the model remains accurate over time. Explore Alternative Metrics: Beyond accuracy, evaluate the model using precision-recall curves and F1-scores, particularly to assess performance in predicting less frequent classes.