GROUP PROJECT-1

Project Report: Data Analysis and Visualization with a focus on Static Visualizations and Statistical Analysis

Course: IE6600 Computation and Visualization

Spring Semester 2024

Group Number - 10

Sri Sai Prabhath Reddy Gudipalli (002207631)

Deepthi Umesha (002661626)

Ashwini Mahadevaswamy (002661627)

Yashwant Dontam (002844794)

Introduction

This project report presents a detailed analysis of a motor vehicle collisions dataset, focusing on identifying key patterns and trends contributing to vehicular accidents. Our comprehensive examination includes the investigation of the top factors leading to collisions, trends in crashes over time, the distribution of accidents across different boroughs, types of vehicles involved, and the variations in crash occurrences at different times of the day. We also explore the correlation between contributing factors and the involvement of various road users such as pedestrians, cyclists, and motorists, and identify high-risk areas by zip codes. Additionally, the report looks into seasonal and hourly variations in collision data. The goal of this analysis is to provide valuable insights for enhancing road safety measures and reducing the incidence of vehicle collisions.

Task: Data Acquisition and Inspection

```
import pandas as pd

dtype_dict = {
    'ZIP CODE': str
}

df = pd.read_csv('Motor_Vehicle_Collisions_dataset.csv', dtype=dtype_dict)
```

```
df.head()
```

	CRASH DATE	CRASH TIME	BOROUGH	ZIP CODE	LATITUDE	LONGITUDE	LOCATION	ON STREET NAME	CROSS STREET NAME	OFF STREET NAME	...	CONTRIBUTING FACTOR VEHICLE 2	CONTRIBUTING FACTOR VEHICLE 3	CONTRIBUTING FACTOR VEHICLE 4
0	09/11/2021	2:39	NaN	NaN	NaN	NaN	NaN	WHITESTONE EXPRESSWAY	20 AVENUE	NaN	...	Unspecified	NaN	NaN
1	03/26/2022	11:45	NaN	NaN	NaN	NaN	NaN	QUEENSBORO BRIDGE UPPER	NaN	NaN	...	NaN	NaN	NaN
2	06/29/2022	6:55	NaN	NaN	NaN	NaN	NaN	THROGS NECK BRIDGE	NaN	NaN	...	Unspecified	NaN	NaN
3	09/11/2021	9:35	BROOKLYN	11208	40.667202	-73.866500	(40.667202, -73.8665)	NaN	NaN	1211 LORING AVENUE	...	NaN	NaN	NaN
4	12/14/2021	8:13	BROOKLYN	11233	40.683304	-73.917274	(40.683304, -73.917274)	SARATOGA AVENUE	DECATUR STREET	NaN	...	NaN	NaN	NaN

5 rows × 29 columns

This code uses the pandas library to read a CSV file named 'Motor_Vehicle_Collisions_dataset.csv' into a DataFrame called 'df'. The 'dtype_dict' dictionary specifies the data type for the 'ZIP CODE' column as a string to ensure proper handling. The code then prints the first few rows of the DataFrame using the head() method, providing an initial glimpse of the dataset's structure and contents.

```
df.info()
print("\n")

rows, columns = df.shape
print(f"Number of rows: {rows}")
print(f"Number of columns: {columns}")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2061020 entries, 0 to 2061019
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: 456.0+ MB


Number of rows: 2061020
Number of columns: 29
```

The code uses the info() method on the DataFrame 'df' to display a concise summary of its information, including the data types of each column and the count of non-null values. After printing a newline for better readability, the code then retrieves the number of rows and columns in the DataFrame using the shape attribute and prints these values. This provides additional basic information about the dataset, such as its size and structure.

```
print(df.isnull().sum())
```

```
CRASH DATE                            0
CRASH TIME                            0
BOROUGH                          641181
ZIP CODE                         641429
LATITUDE                         232765
LONGITUDE                        232765
LOCATION                         232765
ON STREET NAME                   436626
CROSS STREET NAME                777123
OFF STREET NAME                 1716767
NUMBER OF PERSONS INJURED            18
NUMBER OF PERSONS KILLED             31
NUMBER OF PEDESTRIANS INJURED         0
NUMBER OF PEDESTRIANS KILLED          0
NUMBER OF CYCLIST INJURED             0
NUMBER OF CYCLIST KILLED              0
NUMBER OF MOTORIST INJURED            0
NUMBER OF MOTORIST KILLED             0
CONTRIBUTING FACTOR VEHICLE 1      6688
CONTRIBUTING FACTOR VEHICLE 2    318213
CONTRIBUTING FACTOR VEHICLE 3   1914038
CONTRIBUTING FACTOR VEHICLE 4   2027881
CONTRIBUTING FACTOR VEHICLE 5   2052056
COLLISION_ID                          0
VEHICLE TYPE CODE 1               13430
VEHICLE TYPE CODE 2              391685
VEHICLE TYPE CODE 3             1919315
VEHICLE TYPE CODE 4             2029019
VEHICLE TYPE CODE 5             2052328
dtype: int64
```

With the count of missing values in a motor vehicle collisions dataset. Key columns like 'BOROUGH', 'ZIP CODE', and 'LOCATION' have significant missing data, exceeding 200,000 entries each. 'CROSS STREET NAME' and 'OFF STREET NAME' are also notably incomplete. There are inconsistencies in the 'CONTRIBUTING FACTOR VEHICLE' and 'VEHICLE TYPE CODE' columns, with missing values increasing for subsequent entries. Basic columns such as 'NUMBER OF PERSONS INJURED' and 'NUMBER OF PERSONS KILLED' show minimal missing data. This information is essential for pre-analysis data cleaning.

```python
columns_to_drop = [
    'CONTRIBUTING FACTOR VEHICLE 2',
    'CONTRIBUTING FACTOR VEHICLE 3',
    'CONTRIBUTING FACTOR VEHICLE 4',
    'CONTRIBUTING FACTOR VEHICLE 5',
    'VEHICLE TYPE CODE 3',
    'VEHICLE TYPE CODE 4',
    'VEHICLE TYPE CODE 5',
    'OFF STREET NAME'
]

df.drop(columns=columns_to_drop, inplace=True)

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2061020 entries, 0 to 2061019
Data columns (total 21 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   NUMBER OF PERSONS INJURED      float64
 10  NUMBER OF PERSONS KILLED       float64
 11  NUMBER OF PEDESTRIANS INJURED  int64
 12  NUMBER OF PEDESTRIANS KILLED   int64
 13  NUMBER OF CYCLIST INJURED      int64
 14  NUMBER OF CYCLIST KILLED       int64
 15  NUMBER OF MOTORIST INJURED     int64
 16  NUMBER OF MOTORIST KILLED      int64
 17  CONTRIBUTING FACTOR VEHICLE 1  object
 18  COLLISION_ID                   int64
 19  VEHICLE TYPE CODE 1            object
 20  VEHICLE TYPE CODE 2            object
dtypes: float64(4), int64(7), object(10)
memory usage: 330.2+ MB
None
```

Here columns with many missing values or deemed non-essential have been dropped from a motor vehicle collisions DataFrame to streamline the data for analysis. The resulting DataFrame has 21 columns, retaining essential information such as crash date, time, location, and injury counts. This cleaned DataFrame includes over 2 million entries and uses about 330 MB of memory.

```python
# Calculate the mean values for 'NUMBER OF PERSONS INJURED' and 'NUMBER OF PERSONS KILLED'
mean_persons_injured = df['NUMBER OF PERSONS INJURED'].mean()
mean_persons_killed = df['NUMBER OF PERSONS KILLED'].mean()

# Replace missing values with mean values
df['NUMBER OF PERSONS INJURED'].fillna(mean_persons_injured, inplace=True)
df['NUMBER OF PERSONS KILLED'].fillna(mean_persons_killed, inplace=True)
```

```python
print(df.isnull().sum())
```

```
CRASH DATE                          0
CRASH TIME                          0
BOROUGH                        641181
ZIP CODE                       641429
LATITUDE                       232765
LONGITUDE                      232765
LOCATION                       232765
ON STREET NAME                 436626
CROSS STREET NAME              777123
NUMBER OF PERSONS INJURED           0
NUMBER OF PERSONS KILLED            0
NUMBER OF PEDESTRIANS INJURED       0
NUMBER OF PEDESTRIANS KILLED        0
NUMBER OF CYCLIST INJURED           0
NUMBER OF CYCLIST KILLED            0
NUMBER OF MOTORIST INJURED          0
NUMBER OF MOTORIST KILLED           0
CONTRIBUTING FACTOR VEHICLE 1    6688
COLLISION_ID                        0
VEHICLE TYPE CODE 1             13430
VEHICLE TYPE CODE 2            391685
dtype: int64
```

The image shows Python code where missing data in the 'NUMBER OF PERSONS INJURED' and 'NUMBER OF PERSONS KILLED' columns of a dataset are filled with the mean values of these columns. The subsequent null value check reveals that these two columns now have zero missing values, though significant gaps remain in other columns like 'BOROUGH' and 'CROSS STREET NAME'. This imputation enhances the dataset's robustness for analysis.

```
df['CONTRIBUTING FACTOR VEHICLE 1'].fillna('Unknown', inplace=True)
```

```
print(df.isnull().sum())
```

```
CRASH DATE                          0
CRASH TIME                          0
BOROUGH                        641181
ZIP CODE                       641429
LATITUDE                       232765
LONGITUDE                      232765
LOCATION                       232765
ON STREET NAME                 436626
CROSS STREET NAME              777123
NUMBER OF PERSONS INJURED           0
NUMBER OF PERSONS KILLED            0
NUMBER OF PEDESTRIANS INJURED       0
NUMBER OF PEDESTRIANS KILLED        0
NUMBER OF CYCLIST INJURED           0
NUMBER OF CYCLIST KILLED            0
NUMBER OF MOTORIST INJURED          0
NUMBER OF MOTORIST KILLED           0
CONTRIBUTING FACTOR VEHICLE 1       0
COLLISION_ID                        0
VEHICLE TYPE CODE 1             13430
VEHICLE TYPE CODE 2            391685
dtype: int64
```

This Python code used to handle missing values in the 'CONTRIBUTING FACTOR VEHICLE 1' column by filling them with the string 'Unknown'. After this operation, the null value summary shows zero missing values in this column, indicating that all missing data points have been addressed. Other columns, however, like 'BOROUGH', 'ZIP CODE', and 'CROSS STREET NAME', still have a large number of missing values. The code snippet is part of the data cleaning process to prepare the dataset for further analysis.

```python
columns_to_replace = {
    'BOROUGH': 'Unknown',
    'LATITUDE': 'Unknown',
    'LONGITUDE': 'Unknown',
    'LOCATION': 'Unknown',
    'ZIP CODE': 'Unknown',
    'ON STREET NAME': 'Unknown',
    'CROSS STREET NAME': 'Unknown',
    'VEHICLE TYPE CODE 1': 'Unknown',
    'VEHICLE TYPE CODE 2': 'Unknown'
}

for column, replacement in columns_to_replace.items():
    df[column].fillna(replacement, inplace=True)
```

```python
print(df.isnull().sum())
```

```
CRASH DATE                      0
CRASH TIME                      0
BOROUGH                         0
ZIP CODE                        0
LATITUDE                        0
LONGITUDE                       0
LOCATION                        0
ON STREET NAME                  0
CROSS STREET NAME               0
NUMBER OF PERSONS INJURED       0
NUMBER OF PERSONS KILLED        0
NUMBER OF PEDESTRIANS INJURED   0
NUMBER OF PEDESTRIANS KILLED    0
NUMBER OF CYCLIST INJURED       0
NUMBER OF CYCLIST KILLED        0
NUMBER OF MOTORIST INJURED      0
NUMBER OF MOTORIST KILLED       0
CONTRIBUTING FACTOR VEHICLE 1   0
COLLISION_ID                    0
VEHICLE TYPE CODE 1             0
VEHICLE TYPE CODE 2             0
dtype: int64
```

This Python code block where a dictionary named `columns_to_replace` is used to fill missing values in several columns of a DataFrame with the string 'Unknown'. After this mass imputation, a null check reveals no missing values across multiple columns, indicating a successful cleanup. This method standardizes the handling of missing data, ensuring no null entries are present in these columns, which is a crucial step in data preprocessing for reliable analysis.

```
filtered_df = df[(df['LATITUDE'] != 'Unknown') & (df['LONGITUDE'] != 'Unknown') & (df['LOCATION'] != 'Unknown')]

print("Shape before dropping:", df.shape)
print("\n")
print("Shape after dropping:", filtered_df.shape)
```

```
Shape before dropping: (2061020, 21)


Shape after dropping: (1828255, 21)
```

```
print(df.isnull().sum())
```

```
CRASH DATE                      0
CRASH TIME                      0
BOROUGH                         0
ZIP CODE                        0
LATITUDE                        0
LONGITUDE                       0
LOCATION                        0
ON STREET NAME                  0
CROSS STREET NAME               0
NUMBER OF PERSONS INJURED       0
NUMBER OF PERSONS KILLED        0
NUMBER OF PEDESTRIANS INJURED   0
NUMBER OF PEDESTRIANS KILLED    0
NUMBER OF CYCLIST INJURED       0
NUMBER OF CYCLIST KILLED        0
NUMBER OF MOTORIST INJURED      0
NUMBER OF MOTORIST KILLED       0
CONTRIBUTING FACTOR VEHICLE 1   0
COLLISION_ID                    0
VEHICLE TYPE CODE 1             0
VEHICLE TYPE CODE 2             0
dtype: int64
```

Now we have cleaned the dataset

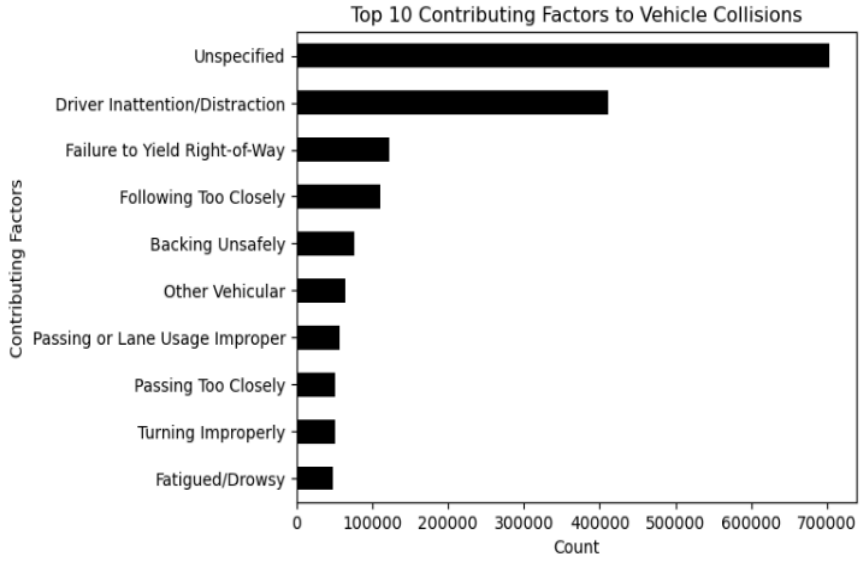
The python code above filters out records from a DataFrame where 'LATITUDE', 'LONGITUDE', and 'LOCATION' are labeled 'Unknown'. This operation reduces the DataFrame size from 2,061,020 to 1,828,255 records, retaining only entries with known location data. The subsequent null value check confirms that there are no missing values in the modified DataFrame. This filtering process is part of data cleaning to ensure analyses are based on complete and accurate location data.

Task- Exploratory Data Analysis (EDA) with a Focus on Static Visualization

Visualizing the top 10 contributing factors to vehicle collisions

First, visualizing the top 10 contributing factors to vehicle collisions.

```
top_factors = df['CONTRIBUTING FACTOR VEHICLE 1'].value_counts().nlargest(10)
top_factors.plot(kind='barh', color='black')
plt.xlabel('Count')
plt.ylabel('Contributing Factors')
plt.title('Top 10 Contributing Factors to Vehicle Collisions')
plt.gca().invert_yaxis()
plt.show()
```

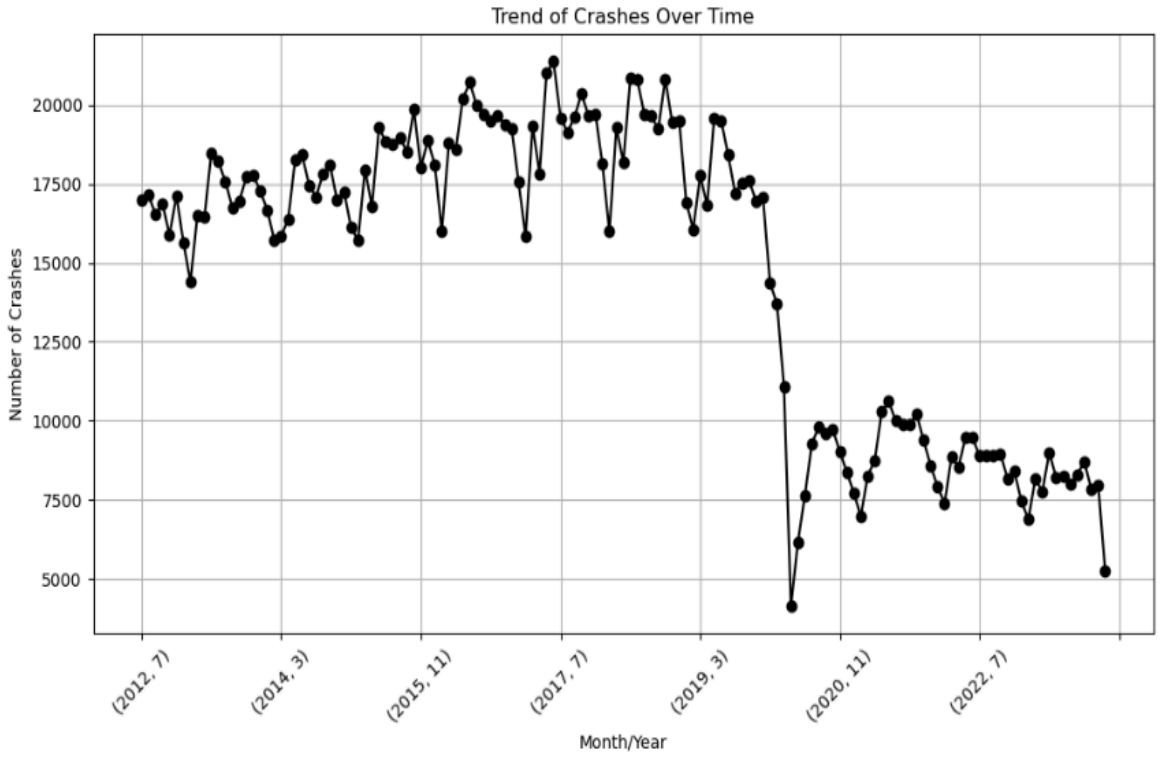


The bar chart was created with Python's matplotlib library, visualizing the top 10 contributing factors to vehicle collisions. The factors are listed on the y-axis in descending order of count, which is represented on the x-axis. The most common contributing factor is labeled as 'Unspecified', suggesting a high number of reports did not have this information filled. The next significant factors include 'Driver Inattention/Distraction', 'Failure to Yield Right-of-Way', and 'Following Too Closely', among others. The chart's y-axis is inverted, placing the factor with the highest count at the top for easier readability. The visual representation provides a clear and immediate understanding of the main reasons behind vehicle collisions as recorded in the dataset.

Plotting the trend of crashes over time (month/year)

Plotting the trend of crashes over time (month/year)

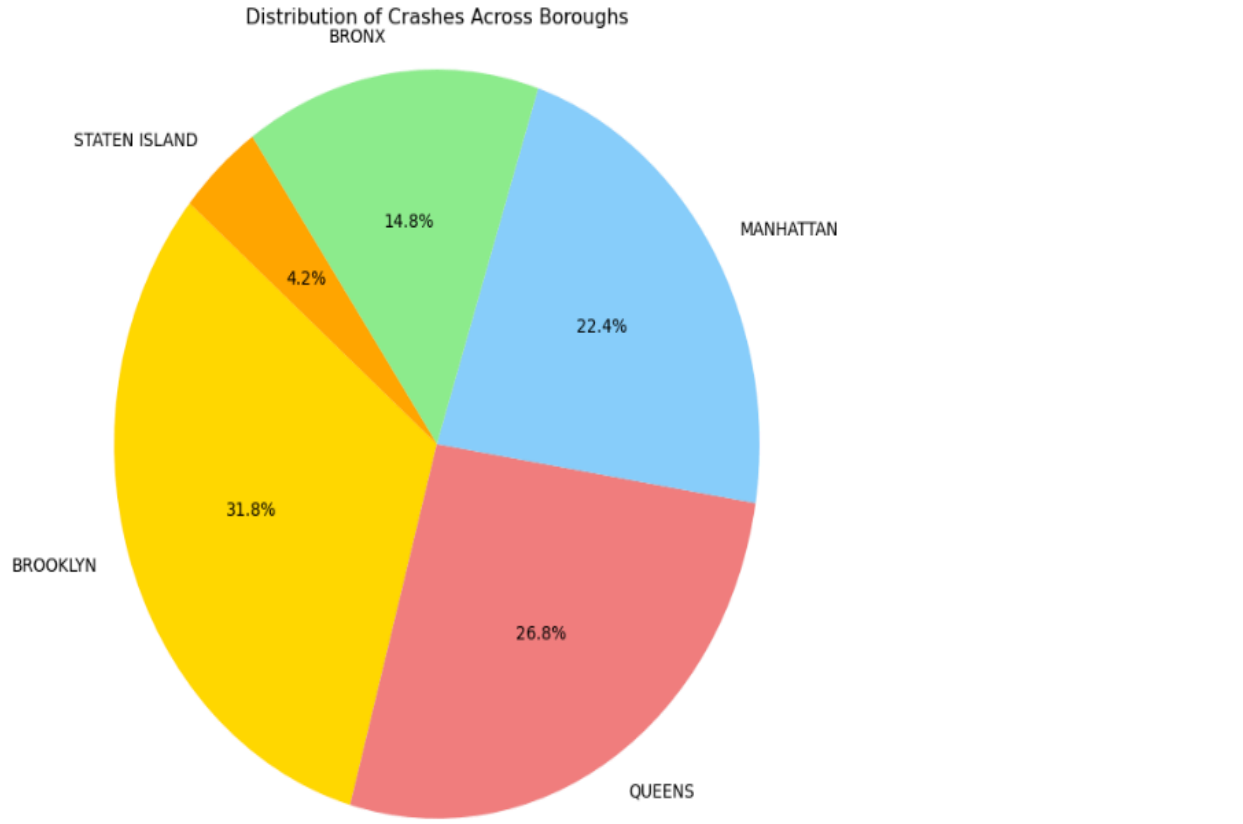
```
df['CRASH DATE'] = pd.to_datetime(df['CRASH DATE'])

df['Month'] = df['CRASH DATE'].dt.month
df['Year'] = df['CRASH DATE'].dt.year

crash_trend = df.groupby(['Year', 'Month']).size()
crash_trend.plot(figsize=(12, 6), marker='o', color='black')
plt.xlabel('Month/Year')
plt.ylabel('Number of Crashes')
plt.title('Trend of Crashes Over Time')
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



The above image illustrates a line graph showing the trend of vehicle crashes over time, plotted on a monthly/yearly basis. The x-axis represents time, with each point denoting a specific month and year, while the y-axis indicates the number of crashes. The data points are connected by lines, with individual occurrences marked by 'o' symbols. There's a noticeable fluctuation in the number of crashes, with some peaks and troughs, suggesting variability in crash frequency over time. The graph includes a grid to enhance readability, and the x-axis tick labels are rotated 45 degrees to fit the graph and prevent overlap. This visualization is designed to help identify patterns or trends in crash occurrences across different times.

Visualizing the distribution of crashes across boroughs

```
filtered_df = df[df['BOROUGH'] != 'Unknown']

filtered_borough_counts = filtered_df['BOROUGH'].value_counts()

plt.figure(figsize=(8, 8))
plt.pie(filtered_borough_counts, labels=filtered_borough_counts.index, autopct='%1.1f%%', startangle=140, colors=['gold', 'lightcoral', 'lightskyblue',
plt.axis('equal')
plt.title('Distribution of Crashes Across Boroughs')
plt.show()
```

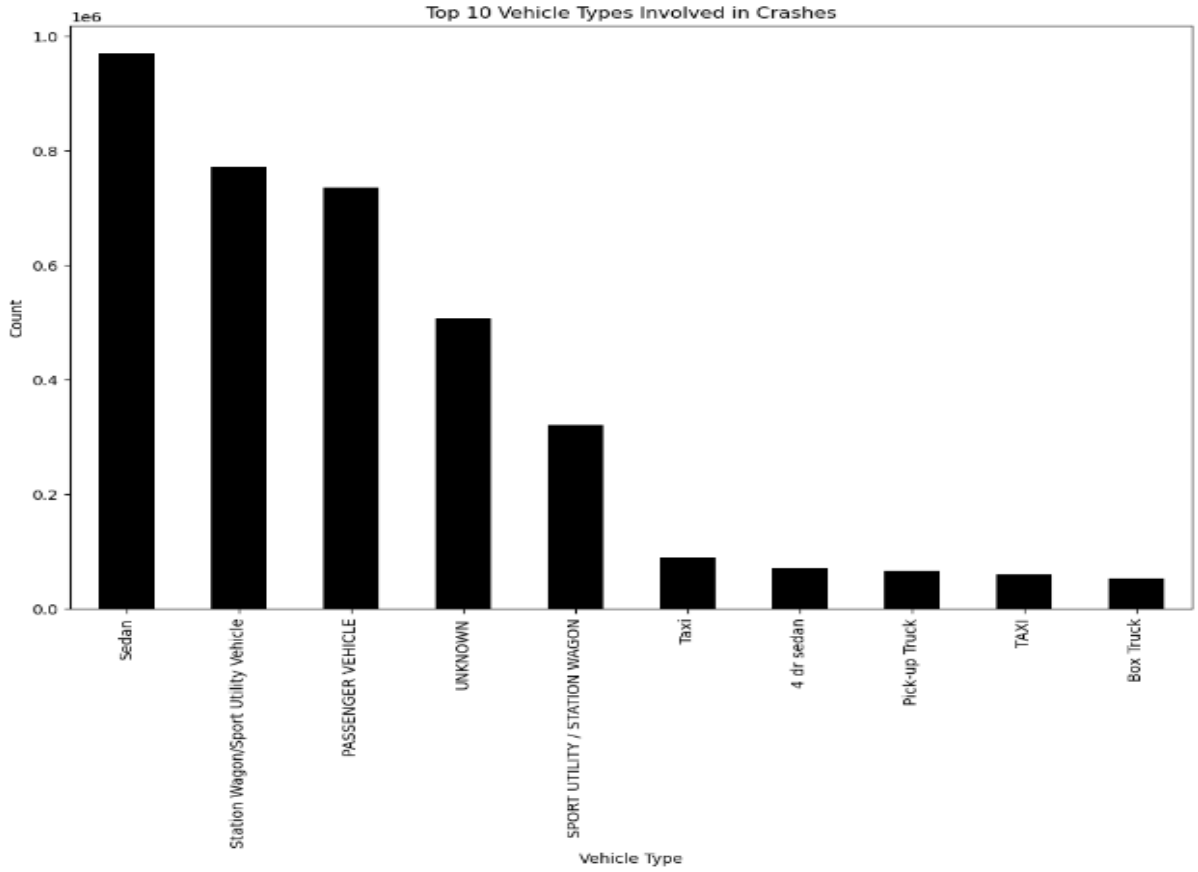


The pie chart represents the distribution of vehicle crashes across different boroughs. The chart is color-coded to distinguish between boroughs, and percentages are provided to indicate the proportion of crashes in each area. Brooklyn has the largest share with 31.8%, followed by Queens with 26.8%, Manhattan with 22.4%, the Bronx with 14.8%, and Staten Island with the smallest share at 4.2%. This visualization is useful for understanding the relative frequency of crashes in different parts of the city, which can inform resource allocation for traffic safety and control measures.

Visualizing the types of vehicles involved in crashes

```
df['VEHICLE TYPE CODE 1'] = df['VEHICLE TYPE CODE 1'].replace('Unknown', 'UNKNOWN')
df['VEHICLE TYPE CODE 2'] = df['VEHICLE TYPE CODE 2'].replace('Unknown', 'UNKNOWN')
```

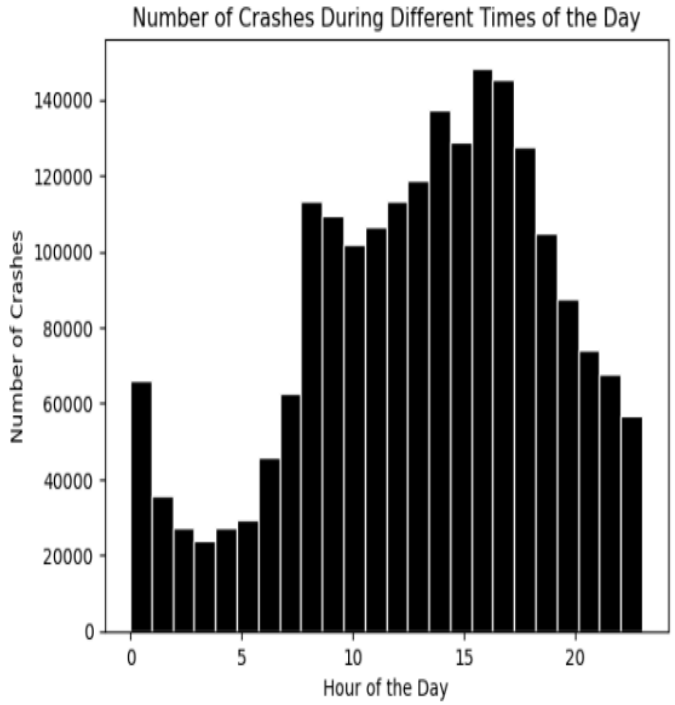
```
vehicle_counts = pd.concat([df['VEHICLE TYPE CODE 1'], df['VEHICLE TYPE CODE 2']]).value_counts().nlargest(10)

plt.figure(figsize=(12, 8))

vehicle_counts.plot(kind='bar', color='black')
plt.xlabel('Vehicle Type')
plt.ylabel('Count')
plt.title('Top 10 Vehicle Types Involved in Crashes')
plt.xticks(rotation=90)
plt.show()
```



The image depicts a bar chart illustrating the top 10 vehicle types involved in crashes. The 'sedan' is the most common, followed by 'station wagon/sport utility vehicle'. Other types listed include 'PASSENGER VEHICLE', 'SPORT UTILITY / STATION WAGON', and a category for unknown vehicle types. The counts are plotted on a logarithmic scale, indicating a broad range in the number of crashes involving different vehicle types. The chart provides a clear visualization to identify which types of vehicles are most frequently involved in collisions, which is critical information for traffic safety analysis and vehicle design safety improvements. The x-axis labels are rotated for readability.

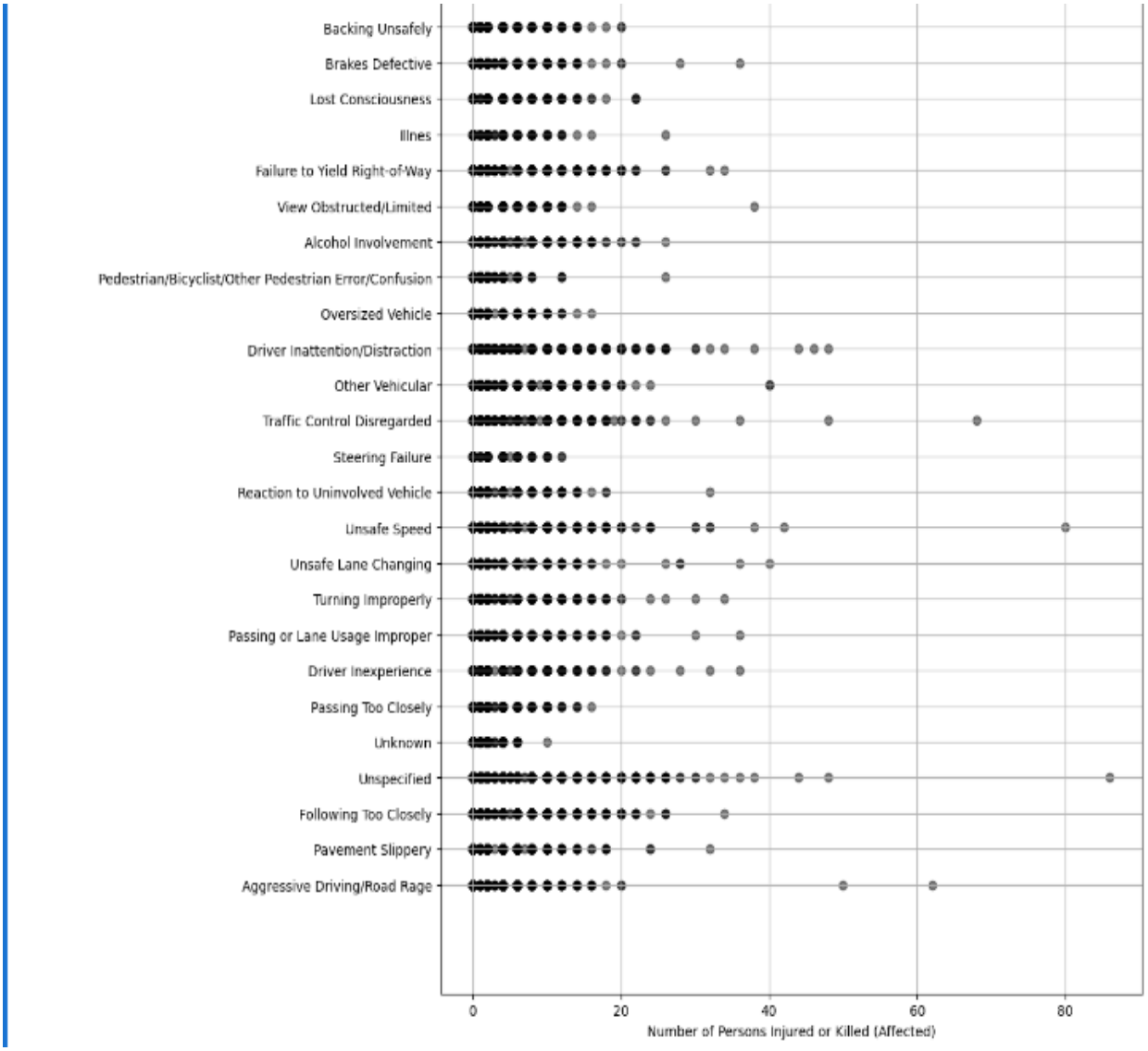
Plotting the number of crashes during different times of the day

```
df['CRASH TIME'] = pd.to_datetime(df['CRASH TIME'], format='%H:%M')

df['Hour'] = df['CRASH TIME'].dt.hour

plt.hist(df['Hour'], bins=24, color='black', edgecolor='white')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Crashes')
plt.title('Number of Crashes During Different Times of the Day')
plt.show()
```



The image displays a histogram visualizing the number of vehicle crashes during different times of the day. The x-axis represents the hours of the day, and the y-axis shows the count of crashes. The data indicates a pattern with the number of crashes increasing during certain hours. There is a notable peak during the late afternoon hours, which might correspond with rush hour traffic. This pattern suggests that the frequency of crashes is correlated with traffic density and times of day, information that can be crucial for traffic management and accident prevention strategies. The black bars with white edges make the individual hour counts distinct for analysis.

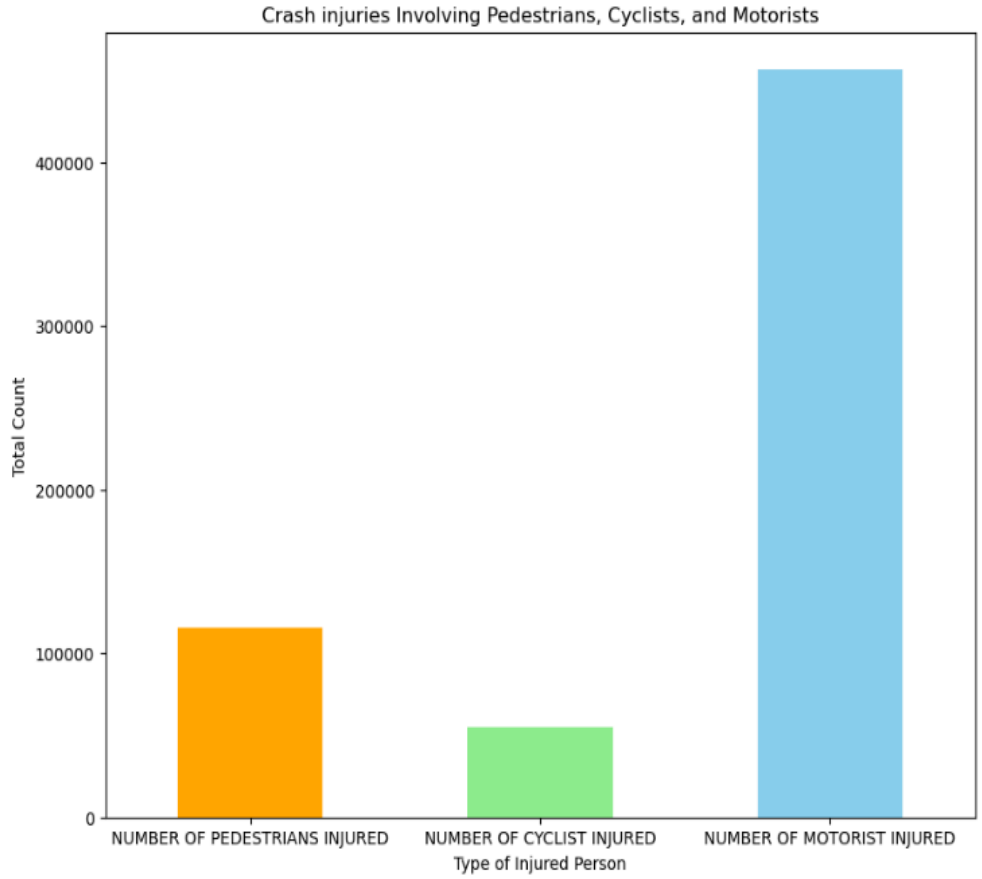
Scatter plot of the correlation between persons involved and the contributing factors



The above images depict scatter plots that correlate the number of persons injured or killed in vehicle collisions with various contributing factors. The distribution of dots illustrates the frequency of each factor in relation to the number of affected individuals. Factors like 'Driver Inattention/Distraction' and 'Failure to Yield Right-of-Way' show a higher prevalence and a broad range of affected person counts, suggesting these factors are often involved in incidents with greater numbers of injuries or fatalities. These visualizations can help identify which contributing factors to collisions are most commonly associated with higher injury and fatality rates.

Plotting the number of crashes injuring pedestrians vs. cyclists vs. motorists

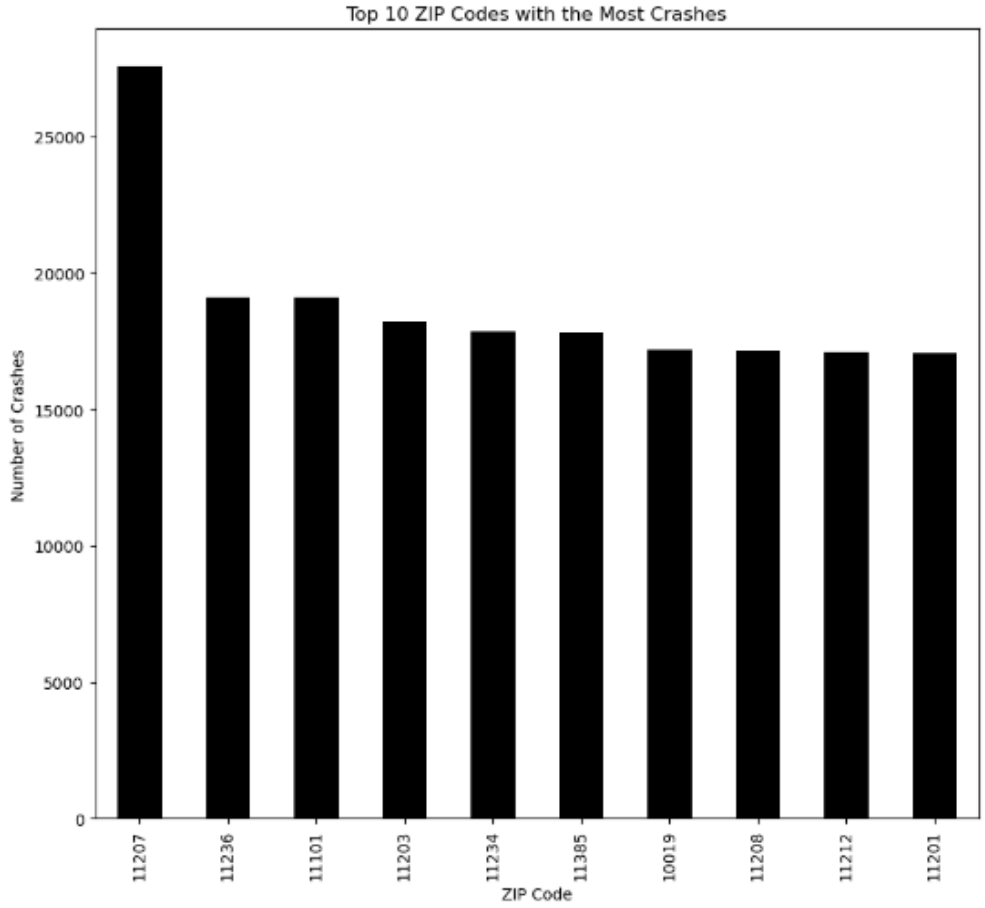
```
crash_injury_counts = df[['NUMBER OF PEDESTRIANS INJURED', 'NUMBER OF CYCLIST INJURED', 'NUMBER OF MOTORIST INJURED']].sum()

plt.figure(figsize=(10,8))
crash_injury_counts.plot(kind='bar', color=['orange', 'lightgreen', 'skyblue'])
plt.xlabel('Type of Injured Person')
plt.ylabel('Total Count')
plt.title('Crash injuries Involving Pedestrians, Cyclists, and Motorists')
plt.xticks(rotation=0)
plt.show()
```



The above image displays a bar chart that compares the total count of injuries among pedestrians, cyclists, and motorists resulting from vehicle collisions. The 'NUMBER OF PEDESTRIAN INJURED' is represented by an orange bar, 'NUMBER OF CYCLIST INJURED' by a light green bar, and 'NUMBER OF MOTORIST INJURED' by a sky blue bar. The chart shows that the number of motorists injured is substantially higher than the number of pedestrians or cyclists injured, depicted by the prominent sky blue bar. This visualization helps in understanding the distribution of injuries among different road users and could be used to inform road safety measures.

Plotting the Top 10 zipcodes with most crashes

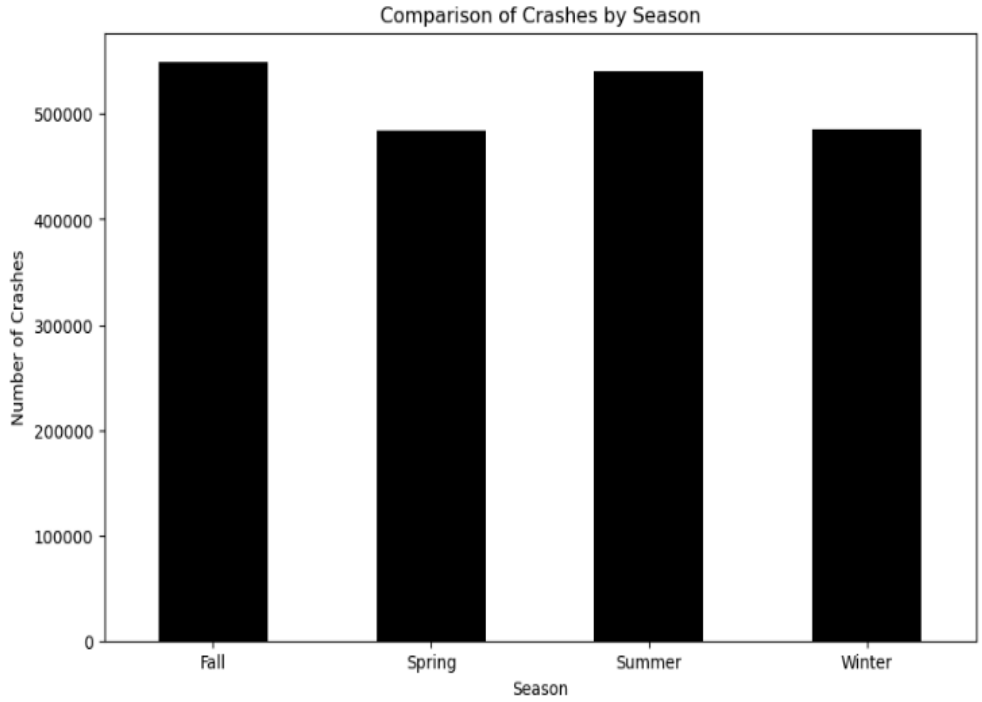
```
filtered_df = df[df['ZIP CODE'] != 'Unknown']

zip_code_counts = filtered_df['ZIP CODE'].value_counts().nlargest(10)

plt.figure(figsize=(10, 9))
zip_code_counts.plot(kind='bar', color='black')
plt.xlabel('ZIP Code')
plt.ylabel('Number of Crashes')
plt.title('Top 10 ZIP Codes with the Most Crashes')
plt.xticks(rotation=90)
plt.show()
```



The above image features a bar chart displaying the top 10 ZIP codes with the highest number of vehicle crashes. The bars are colored in black, uniformly representing each ZIP code along the x-axis, with the y-axis indicating the number of crashes. The first ZIP code on the left has the highest count of crashes, significantly more than the others, with a steady decline in crash numbers observed across the subsequent ZIP codes. The chart effectively highlights areas with higher occurrences of collisions, which could be critical for targeted safety interventions and resource allocation. The x-axis labels, which are ZIP codes, are rotated to ensure they are legible.

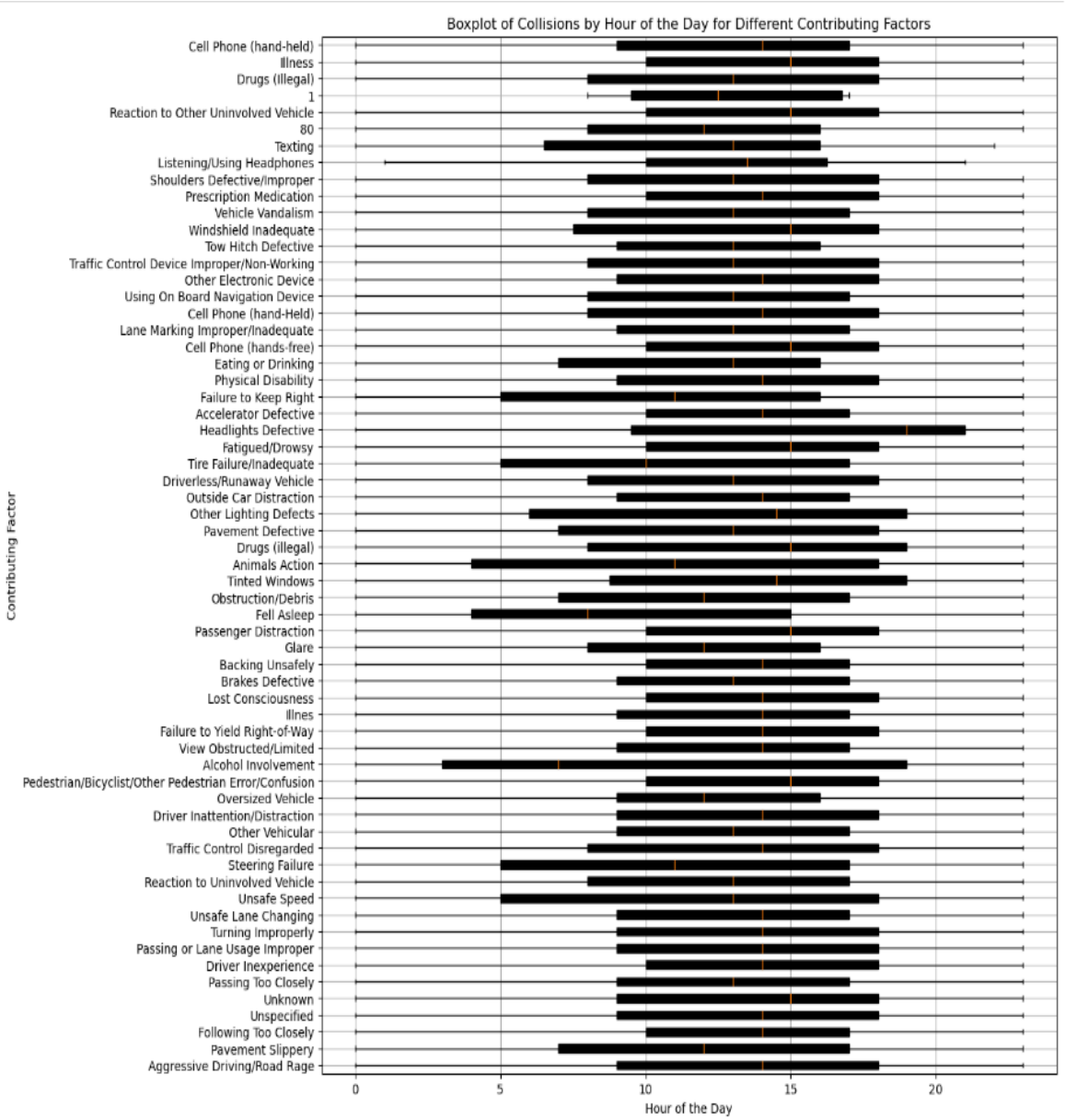
Sorting and Plotting the motor vehicle crashes by Season

```
df['Month'] = df['CRASH DATE'].dt.month

seasons = {1: 'Winter', 2: 'Winter', 3: 'Spring', 4: 'Spring', 5: 'Spring', 6: 'Summer', 7: 'Summer', 8: 'Summer', 9: 'Fall', 10: 'Fall', 11: 'Fall', 12
#we have sorted seasons based on data from google, may differ for different geographical locations
df['Season'] = df['Month'].map(seasons)

plt.figure(figsize=(10, 6))
df['Season'].value_counts().sort_index().plot(kind='bar', color='black')
plt.xlabel('Season')
plt.ylabel('Number of Crashes')
plt.title('Comparison of Crashes by Season')
plt.xticks(rotation=0)
plt.show()
```



The above image displays a bar chart that compares the number of vehicle crashes across different seasons: Fall, Spring, Summer, and Winter. The seasons are arranged on the x-axis, while the y-axis shows the total count of crashes. The bars, colored in black, indicate that the number of crashes in Fall is the highest among the seasons, followed closely by Spring and Summer, with Winter having the least. This seasonal trend could be influenced by various factors such as weather conditions, daylight hours, and driving behavior. The graph provides a visual representation of how crash frequencies vary with seasons, which can inform seasonal safety campaigns and driving advisories.

Boxplot of Collisions by Hour of the Day for Different Contributing Factors



The above boxplot illustrates the distribution of vehicle crashes by hour across various contributing factors. Each boxplot displays the central tendency and spread of crash times for a specific factor, indicating peak hours and outliers. This visualization helps to identify patterns in crash occurrences related to these factors, which can inform traffic safety strategies and interventions.

Conclusion

The analysis of vehicle collision data highlighted key factors contributing to accidents, with inattention and failure to yield being most common, suggesting targeted areas for safety improvements. Temporal trends pointed to rush hours as peak times for crashes, while spatial distribution identified high-risk boroughs and ZIP codes. The data showed motorists as the most frequent victims, followed by pedestrians and cyclists, indicating the need for varied safety measures. Seasonal trends were also observed, with fall having the highest number of crashes. The study's findings advocate for multifaceted, data-informed road safety strategies to effectively reduce traffic collisions.