```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
```

```
# Load the dataset
crashes_data = pd.read_csv('/content/drive/MyDrive/CTA/Traffic_Crashes.csv')
```

```
# Displaying the first few rows of the dataset to understand its structure
crashes_data.head()
```

| | CRASH_RECORD_ID | CRASH_DATE_EST_I | CRASH_DATE | POSTED |
|---|---|---|---|---|
| **0** | 23a79931ef555d54118f64dc9be2cf2dbf59636ce253f7... | NaN | 09/05/2023 07:05:00 PM | |
| **1** | 2675c13fd0f474d730a5b780968b3cafc7c12d7adb661f... | NaN | 09/22/2023 06:45:00 PM | |
| **2** | 5f54a59fcb087b12ae5b1acff96a3caf4f2d37e79f8db4... | NaN | 07/29/2023 02:45:00 PM | |
| **3** | 7ebf015016f83d09b321afd671a836d6b148330535d5df... | NaN | 08/09/2023 11:00:00 PM | |
| **4** | 6c1659069e9c6285a650e70d6f9b574ed5f64c12888479... | NaN | 08/18/2023 12:50:00 PM | |

5 rows × 48 columns

```
#There are 48 columns and 786386 rows
crashes_data.shape
```

```
(786386, 48)
```

```
# Checking for missing values and the data type of each column
missing_values = crashes_data.isnull().sum()
data_types = crashes_data.dtypes

# Combining the information into a single DataFrame for easier analysis
data_overview = pd.DataFrame({'Missing Values': missing_values, 'Data Type': data_types})
data_overview.sort_values(by='Missing Values', ascending=False)
```

| | | |
|---|---|---|
| **INJURIES_TOTAL** | 1726 | float64 |
| **INJURIES_FATAL** | 1726 | float64 |
| **INJURIES_INCAPACITATING** | 1726 | float64 |
| **INJURIES_NON_INCAPACITATING** | 1726 | float64 |
| **INJURIES_REPORTED_NOT_EVIDENT** | 1726 | float64 |
| **INJURIES_NO_INDICATION** | 1726 | float64 |
| **INJURIES_UNKNOWN** | 1726 | float64 |
| **BEAT_OF_OCCURRENCE** | 5 | float64 |
| **STREET_DIRECTION** | 4 | object |
| **STREET_NAME** | 1 | object |
| **FIRST_CRASH_TYPE** | 0 | object |
| **CRASH_DATE** | 0 | object |
| **POSTED_SPEED_LIMIT** | 0 | int64 |
| **CRASH_MONTH** | 0 | int64 |
| **CRASH_DAY_OF_WEEK** | 0 | int64 |
| **CRASH_HOUR** | 0 | int64 |
| **TRAFFIC_CONTROL_DEVICE** | 0 | object |
| **DEVICE_CONDITION** | 0 | object |
| **WEATHER_CONDITION** | 0 | object |
| **LIGHTING_CONDITION** | 0 | object |
| **TRAFFICWAY_TYPE** | 0 | object |
| **STREET_NO** | 0 | int64 |
| **ALIGNMENT** | 0 | object |
| **ROADWAY_SURFACE_COND** | 0 | object |

```
# Step 2: Identify Missing Values
missing_values = crashes_data.isnull().sum()

# Creating a DataFrame to display the number of missing values and the percentage of total v
missing_values_df = pd.DataFrame({'Missing Values': missing_values,
                                  'Percentage': (missing_values / len(crashes_data)) * 100})

# Sorting the DataFrame to show columns with the highest percentage of missing values first
missing_values_df.sort_values(by='Percentage', ascending=False)
```

| | | |
|---|---|---|
| **INJURIES_TOTAL** | 1726 | 0.219485 |
| **INJURIES_FATAL** | 1726 | 0.219485 |
| **INJURIES_INCAPACITATING** | 1726 | 0.219485 |
| **INJURIES_NON_INCAPACITATING** | 1726 | 0.219485 |
| **INJURIES_REPORTED_NOT_EVIDENT** | 1726 | 0.219485 |
| **INJURIES_NO_INDICATION** | 1726 | 0.219485 |
| **INJURIES_UNKNOWN** | 1726 | 0.219485 |
| **BEAT_OF_OCCURRENCE** | 5 | 0.000636 |
| **STREET_DIRECTION** | 4 | 0.000509 |
| **STREET_NAME** | 1 | 0.000127 |
| **FIRST_CRASH_TYPE** | 0 | 0.000000 |
| **CRASH_DATE** | 0 | 0.000000 |
| **POSTED_SPEED_LIMIT** | 0 | 0.000000 |
| **CRASH_MONTH** | 0 | 0.000000 |
| **CRASH_DAY_OF_WEEK** | 0 | 0.000000 |
| **CRASH_HOUR** | 0 | 0.000000 |
| **TRAFFIC_CONTROL_DEVICE** | 0 | 0.000000 |
| **DEVICE_CONDITION** | 0 | 0.000000 |
| **WEATHER_CONDITION** | 0 | 0.000000 |

```
# Dropping columns with a high percentage of missing values
columns_to_drop = missing_values[missing_values > 0.5 * len(crashes_data)].index
crashes_data_cleaned = crashes_data.drop(columns=columns_to_drop)
```

| | | |
|---|---|---|
| **STREET_NO** | 0 | 0.000000 |

```
#crashes_data_cleaned.to_csv('Traffic_Crashes_Cleaned.csv')
```

| | | |
|---|---|---|
| **ROADWAY_SURFACE_COND** | 0 | 0.000000 |

```
# Converting 'CRASH_DATE' to datetime format
crashes_data_cleaned['CRASH_DATE'] = pd.to_datetime(crashes_data_cleaned['CRASH_DATE'])
```

| | | |
|---|---|---|
| **ROAD_DEFECT** | 0 | 0.000000 |

```
# Displaying the results of the cleaning
summary = {
    "Original Columns": len(crashes_data.columns),
    "Cleaned Columns": len(crashes_data_cleaned.columns),
    "Dropped Columns": len(columns_to_drop),
    "Dropped Columns Names": columns_to_drop,
    "Columns with Missing Values": len(missing_values[missing_values > 0]),
    "Sample Data After Cleaning": crashes_data_cleaned.head()
}

summary
```

```
 'Cleaned Columns': 37,
 'Dropped Columns': 11,
 'Dropped Columns Names': Index(['CRASH_DATE_EST_I', 'LANE_CNT',
'INTERSECTION_RELATED_I',
       'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN_I', 'PHOTOS_TAKEN_I',
       'STATEMENTS_TAKEN_I', 'DOORING_I', 'WORK_ZONE_I', 'WORK_ZONE_TYPE',
       'WORKERS_PRESENT_I'],
      dtype='object'),
 'Columns with Missing Values': 26,
 'Sample Data After Cleaning':                                    CRASH_RECORD_ID
CRASH_DATE  \
 0  23a79931ef555d54118f64dc9be2cf2dbf59636ce253f7... 2023-09-05 19:05:00
 1  2675c13fd0f474d730a5b780968b3cafc7c12d7adb661f... 2023-09-22 18:45:00
 2  5f54a59fcb087b12ae5b1acff96a3caf4f2d37e79f8db4... 2023-07-29 14:45:00
 3  7ebf015016f83d09b321afd671a836d6b148330535d5df... 2023-08-09 23:00:00
 4  6c1659069e9c6285a650e70d6f9b574ed5f64c12888479... 2023-08-18 12:50:00

    POSTED_SPEED_LIMIT TRAFFIC_CONTROL_DEVICE      DEVICE_CONDITION  \
 0                  30         TRAFFIC SIGNAL  FUNCTIONING PROPERLY
 1                  50            NO CONTROLS           NO CONTROLS
 2                  30         TRAFFIC SIGNAL  FUNCTIONING PROPERLY
 3                  30            NO CONTROLS           NO CONTROLS
 4                  15                  OTHER  FUNCTIONING PROPERLY

   WEATHER_CONDITION      LIGHTING_CONDITION          FIRST_CRASH_TYPE  \
 0             CLEAR                    DUSK                     ANGLE
 1             CLEAR  DARKNESS, LIGHTED ROAD                 REAR END
 2             CLEAR                DAYLIGHT      PARKED MOTOR VEHICLE
 3             CLEAR  DARKNESS, LIGHTED ROAD  SIDESWIPE SAME DIRECTION
 4             CLEAR                DAYLIGHT                 REAR END

                    TRAFFICWAY_TYPE           ALIGNMENT  ...  \
 0              FIVE POINT, OR MORE  STRAIGHT AND LEVEL  ...
 1       DIVIDED - W/MEDIAN BARRIER  STRAIGHT AND LEVEL  ...
 2  DIVIDED - W/MEDIAN (NOT RAISED)  STRAIGHT AND LEVEL  ...
 3                     NOT DIVIDED  STRAIGHT AND LEVEL  ...
 4                           OTHER  STRAIGHT AND LEVEL  ...

   INJURIES_NON_INCAPACITATING INJURIES_REPORTED_NOT_EVIDENT  \
 0                         2.0                          0.0
 1                         0.0                          0.0
 2                         0.0                          0.0
```

```
   INJURIES_NO_INDICATION INJURIES_UNKNOWN CRASH_HOUR CRASH_DAY_OF_WEEK  \
0                     2.0              0.0         19                  3
1                     2.0              0.0         18                  6
2                     1.0              0.0         14                  7
3                     2.0              0.0         23                  4
4                     1.0              0.0         12                  6

   CRASH_MONTH  LATITUDE  LONGITUDE                                LOCATION
0            9       NaN        NaN                                     NaN
1            9       NaN        NaN                                     NaN
2            7  41.85412 -87.665902  POINT (-87.665902342962 41.854120262952)
3            8       NaN        NaN                                     NaN
4            8       NaN        NaN                                     NaN
```

```
crashes_data_cleaned.columns
```

```
Index(['CRASH_RECORD_ID', 'CRASH_DATE', 'POSTED_SPEED_LIMIT',
       'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION', 'WEATHER_CONDITION',
       'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE', 'TRAFFICWAY_TYPE',
       'ALIGNMENT', 'ROADWAY_SURFACE_COND', 'ROAD_DEFECT', 'REPORT_TYPE',
       'CRASH_TYPE', 'DAMAGE', 'DATE_POLICE_NOTIFIED',
       'PRIM_CONTRIBUTORY_CAUSE', 'SEC_CONTRIBUTORY_CAUSE', 'STREET_NO',
       'STREET_DIRECTION', 'STREET_NAME', 'BEAT_OF_OCCURRENCE', 'NUM_UNITS',
       'MOST_SEVERE_INJURY', 'INJURIES_TOTAL', 'INJURIES_FATAL',
       'INJURIES_INCAPACITATING', 'INJURIES_NON_INCAPACITATING',
       'INJURIES_REPORTED_NOT_EVIDENT', 'INJURIES_NO_INDICATION',
       'INJURIES_UNKNOWN', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
       'LATITUDE', 'LONGITUDE', 'LOCATION'],
      dtype='object')
```

```
#Dimensions of cleaned Traffic Crashes - 37 columns and 786386 rows
crashes_data_cleaned.shape
```

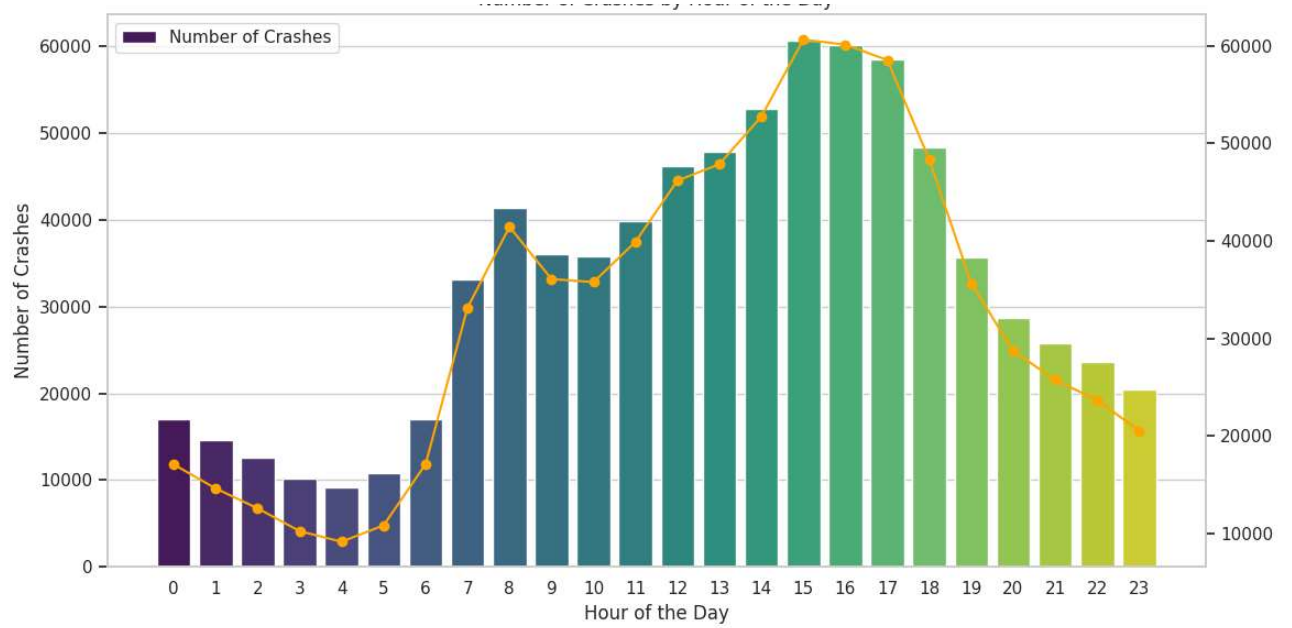
```
(786386, 37)
```

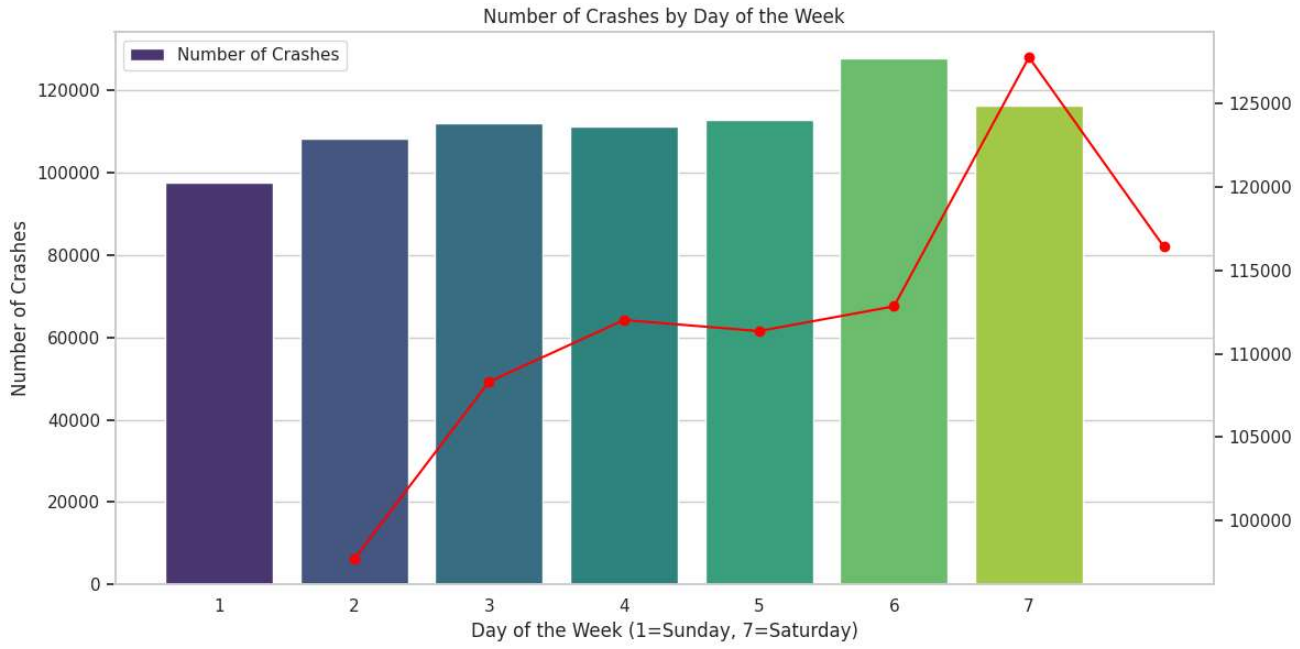
```
# Rechecking the missing values in the cleaned dataset
cleaned_missing_values = crashes_data_cleaned.isnull().sum()
cleaned_missing_values[cleaned_missing_values > 0].sort_values(ascending=False)
```

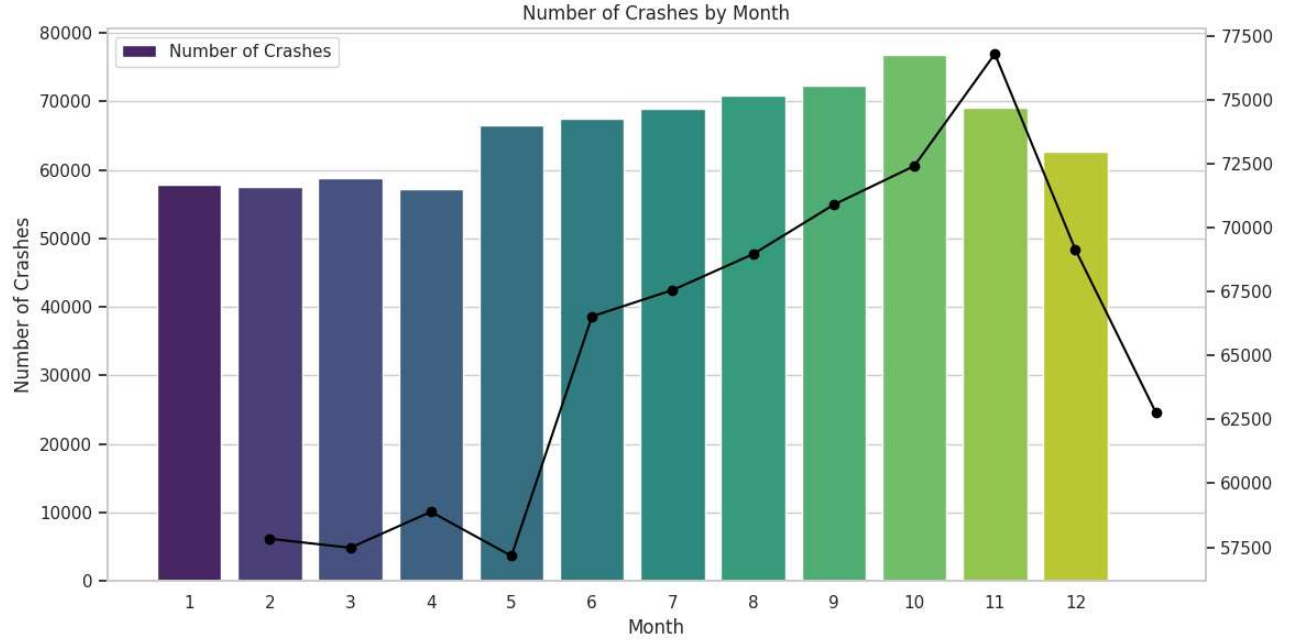
```
REPORT_TYPE                    22904
LATITUDE                        5309
LONGITUDE                       5309
LOCATION                        5309
MOST_SEVERE_INJURY              1737
INJURIES_TOTAL                  1726
INJURIES_FATAL                  1726
INJURIES_INCAPACITATING         1726
INJURIES_NON_INCAPACITATING     1726
INJURIES_REPORTED_NOT_EVIDENT   1726
INJURIES_NO_INDICATION          1726
INJURIES_UNKNOWN                1726
BEAT_OF_OCCURRENCE                 5
```

```
STREET_DIRECTION    4
STREET_NAME         1
dtype: int64
```

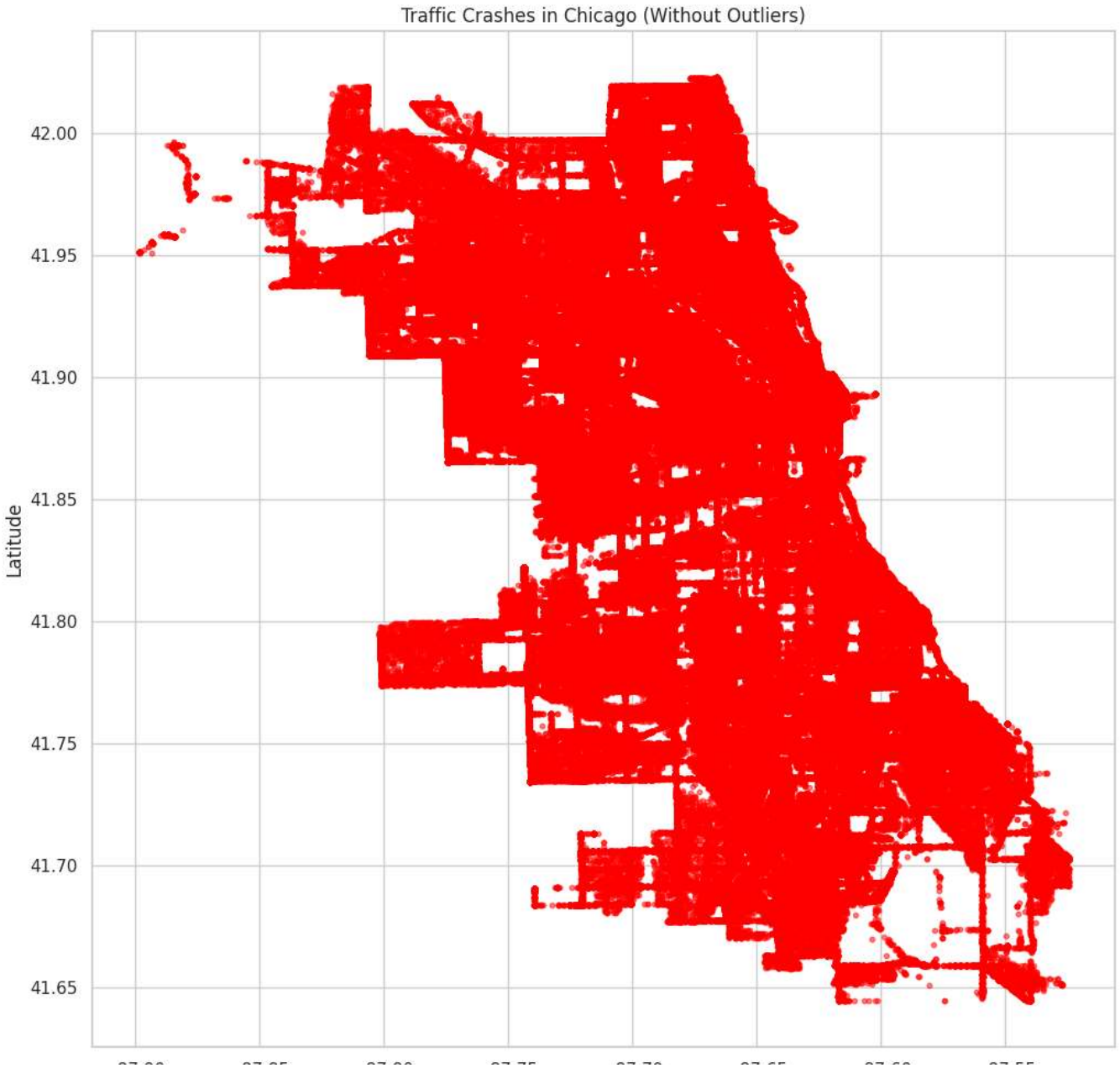
```
import matplotlib.pyplot as plt
import seaborn as sns

# Setting the aesthetic style of the plots
sns.set(style="whitegrid")

# Analysis of when and where crashes are most likely

# 1. Temporal Analysis: Analyzing CRASH_HOUR, CRASH_DAY_OF_WEEK, and CRASH_MONTH
hourly_crashes = crashes_data_cleaned['CRASH_HOUR'].value_counts().sort_index()
weekday_crashes = crashes_data_cleaned['CRASH_DAY_OF_WEEK'].value_counts().sort_index()
monthly_crashes = crashes_data_cleaned['CRASH_MONTH'].value_counts().sort_index()

# Plotting the temporal data
fig, axes = plt.subplots(3, 1, figsize=(12, 18))

# Hourly crashes
hourly_plot = sns.barplot(x=hourly_crashes.index, y=hourly_crashes.values, ax=axes[0], palet
axes[0].set_title('Number of Crashes by Hour of the Day')
axes[0].set_xlabel('Hour of the Day')
axes[0].set_ylabel('Number of Crashes')

# Line plot for hourly trend
hourly_trend = hourly_crashes.plot(kind='line', marker='o', ax=axes[0], secondary_y=True, co
axes[0].legend([hourly_plot.patches[0], hourly_trend], ['Number of Crashes', 'Hourly Trend']

# Weekday crashes
weekday_plot = sns.barplot(x=weekday_crashes.index, y=weekday_crashes.values, ax=axes[1], pa
axes[1].set_title('Number of Crashes by Day of the Week')
axes[1].set_xlabel('Day of the Week (1=Sunday, 7=Saturday)')
axes[1].set_ylabel('Number of Crashes')

# Line plot for weekday trend
weekday_trend = weekday_crashes.plot(kind='line', marker='o', ax=axes[1], secondary_y=True,
axes[1].legend([weekday_plot.patches[0], weekday_trend], ['Number of Crashes', 'Weekday Trer

# Monthly crashes
monthly_plot = sns.barplot(x=monthly_crashes.index, y=monthly_crashes.values, ax=axes[2], pa
axes[2].set_title('Number of Crashes by Month')
axes[2].set_xlabel('Month')
axes[2].set_ylabel('Number of Crashes')

# Line plot for monthly trend
monthly_trend = monthly_crashes.plot(kind='line', marker='o', ax=axes[2], secondary_y=True,
axes[2].legend([monthly_plot.patches[0], monthly_trend], ['Number of Crashes', 'Monthly Trer

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


Number of Crashes
Number of Crashes
Hour of the Day


Number of Crashes by Day of the Week
Number of Crashes
Number of Crashes
Day of the Week (1=Sunday, 7=Saturday)

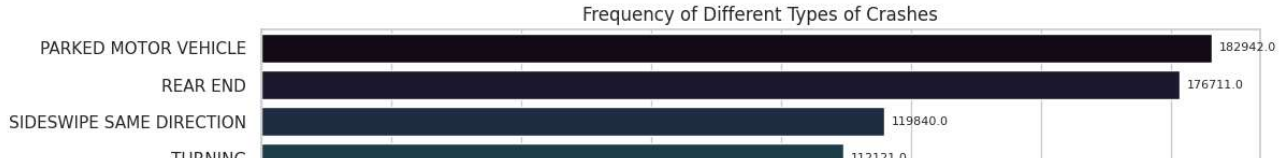

Number of Crashes by Month
Number of Crashes
Number of Crashes
Month

```
# crashes_data_cleaned.to_excel("Traffic_Crashes_Cleaned_Excel.xlsx")


import matplotlib.pyplot as plt

# Assuming your dataset is named 'crashes_data_cleaned'

# Define latitude and longitude bounds for Chicago
latitude_bounds = (41.6, 42.1)
longitude_bounds = (-87.9, -87.5)

# Filter the data based on the defined bounds
filtered_crashes = crashes_data_cleaned[
    (crashes_data_cleaned['LATITUDE'] >= latitude_bounds[0]) & (crashes_data_cleaned['LATITU
    (crashes_data_cleaned['LONGITUDE'] >= longitude_bounds[0]) & (crashes_data_cleaned['LONG
]

# Create a scatter plot of the filtered crash locations
plt.figure(figsize=(12, 12))
plt.scatter(filtered_crashes['LONGITUDE'], filtered_crashes['LATITUDE'], color='red', s=10,

# Customize the plot
plt.title('Traffic Crashes in Chicago (Without Outliers)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)
plt.show()
```




## What types of crashes are most likely?

```
# Analyzing the frequency of each crash type
crash_type_counts = crashes_data_cleaned['FIRST_CRASH_TYPE'].value_counts()

# Plotting the frequency of each crash type
plt.figure(figsize=(12, 8))
ax = sns.barplot(x=crash_type_counts.values, y=crash_type_counts.index, palette="cubehelix")
plt.title('Frequency of Different Types of Crashes')
plt.xlabel('Number of Crashes')
plt.ylabel('Type of Crash')

# Annotating each bar with its value
for p in ax.patches:
    ax.annotate(f'{p.get_width()}', (p.get_width(), p.get_y() + p.get_height() / 2.),
                ha='left', va='center', xytext=(5, 0), textcoords='offset points', fontsize=

plt.show()
```



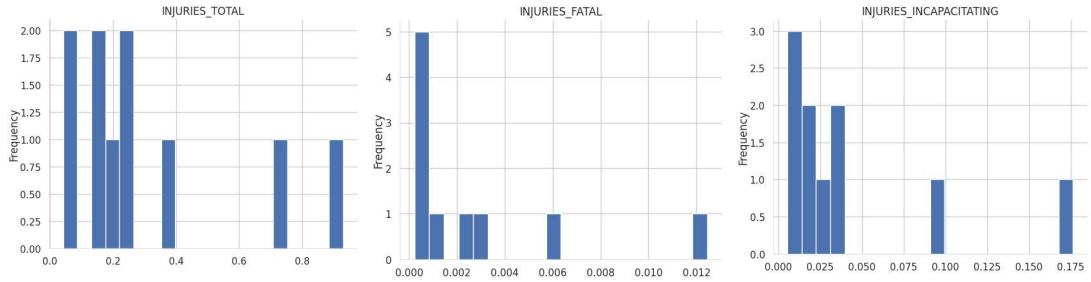

```
# Identifying the most common crash types
most_common_crash_types = crash_type_counts.head(5).index.tolist()

# Filtering the dataset for these crash types
filtered_data = crashes_data_cleaned[crashes_data_cleaned['FIRST_CRASH_TYPE'].isin(most_comm

# Analyzing characteristics for each of the most common crash types
# Looking at factors like Weather Condition, Lighting Condition, Traffic Control Device, Spe
crash_type_characteristics = filtered_data.groupby('FIRST_CRASH_TYPE')[
    ['WEATHER_CONDITION', 'LIGHTING_CONDITION', 'TRAFFIC_CONTROL_DEVICE', 'POSTED_SPEED_LIMI
].apply(lambda x: x.mode().iloc[0]).reset_index()

# Displaying the characteristics
crash_type_characteristics
```

| | FIRST_CRASH_TYPE | WEATHER_CONDITION | LIGHTING_CONDITION | TRAFFIC_CONTROL_DEVICE | POST |
|---|---|---|---|---|---|
| 0 | ANGLE | CLEAR | DAYLIGHT | NO CONTROLS | |
| 1 | PARKED MOTOR VEHICLE | CLEAR | DAYLIGHT | NO CONTROLS | |
| 2 | REAR END | CLEAR | DAYLIGHT | TRAFFIC SIGNAL | |
| 3 | SIDESWIPE SAME DIRECTION | CLEAR | DAYLIGHT | NO CONTROLS | |

```
# Analyzing the severity of crashes by type
severity_columns = ['INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITATING']
severity_by_crash_type = filtered_data.groupby('FIRST_CRASH_TYPE')[severity_columns].mean()

# Displaying the severity analysis
severity_by_crash_type
```

| FIRST_CRASH_TYPE | INJURIES_TOTAL | INJURIES_FATAL | INJURIES_INCAPACITATING |
|---|---|---|---|
| ANGLE | 0.358022 | 0.001268 | 0.032912 |
| FIXED OBJECT | 0.223867 | 0.006291 | 0.038680 |
| OTHER OBJECT | 0.155309 | 0.002641 | 0.027866 |
| PARKED MOTOR VEHICLE | 0.042378 | 0.000549 | 0.005535 |
| PEDALCYCLIST | 0.713511 | 0.002752 | 0.095329 |
| PEDESTRIAN | 0.929764 | 0.012450 | 0.176390 |
| REAR END | 0.183370 | 0.000232 | 0.011273 |
| SIDESWIPE OPPOSITE DIRECTION | 0.159860 | 0.000630 | 0.015743 |
| SIDESWIPE SAME DIRECTION | 0.071389 | 0.000242 | 0.005866 |
| TURNING | 0.242863 | 0.000669 | 0.020790 |

**Distributions**




**2-d distributions**

```
# Preparing data for correlation analysis
# Converting categorical variables into dummy variables for correlation analysis
categorical_columns = ['WEATHER_CONDITION', 'LIGHTING_CONDITION', 'TRAFFIC_CONTROL_DEVICE']
dummies = pd.get_dummies(filtered_data[categorical_columns], drop_first=True)
correlation_data = pd.concat([filtered_data[severity_columns + ['FIRST_CRASH_TYPE']], dummie
```

```
correlation_data.head()
```

| | INJURIES_TOTAL | INJURIES_FATAL | INJURIES_INCAPACITATING | FIRST_CRASH_TYPE | WEATHER_CO |
|---|---|---|---|---|---|
| **0** | 3.0 | 0.0 | 1.0 | ANGLE | |
| **1** | 0.0 | 0.0 | 0.0 | REAR END | |
| **2** | 0.0 | 0.0 | 0.0 | PARKED MOTOR VEHICLE | |

```
# Calculating correlations
correlation_matrix = correlation_data.corr()
```

```
<ipython-input-27-14bf4516b634>:2: FutureWarning: The default value of numeric_only in D
  correlation_matrix = correlation_data.corr()
```

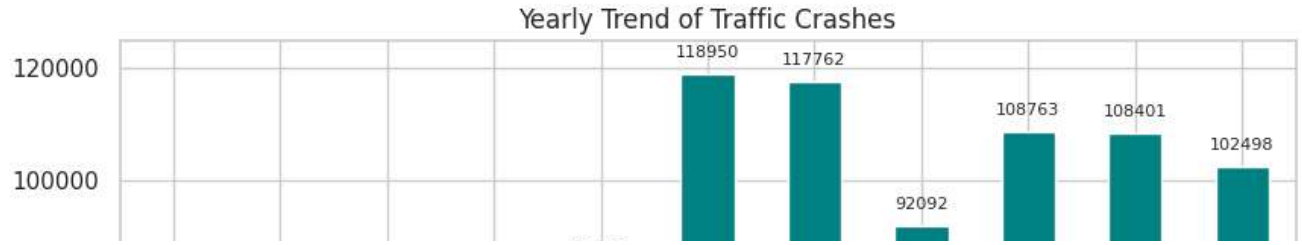
```
# Isolating correlations of different factors with the severity columns
severity_correlations = correlation_matrix[severity_columns].drop(severity_columns + ['FIRST

# Displaying the top 5 correlated factors for each severity type
top_correlated_factors = pd.DataFrame({
    "Total Injuries": severity_correlations['INJURIES_TOTAL'].sort_values(ascending=False).h
    "Fatal Injuries": severity_correlations['INJURIES_FATAL'].sort_values(ascending=False).h
    "Incapacitating Injuries": severity_correlations['INJURIES_INCAPACITATING'].sort_values(
})

top_correlated_factors
```

| | Total Injuries | Fatal Injuries | Incapacitating Injuries |
|---|---|---|---|
| **LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD** | 0.055574 | 0.022365 | 0.030263 |
| **TRAFFIC_CONTROL_DEVICE_PEDESTRIAN CROSSING SIGN** | 0.017409 | 0.003561 | 0.012163 |
| **TRAFFIC_CONTROL_DEVICE_RAILROAD CROSSING GATE** | NaN | 0.003599 | NaN |
| **TRAFFIC_CONTROL_DEVICE_SCHOOL ZONE** | NaN | 0.004973 | NaN |
| **TRAFFIC_CONTROL_DEVICE_STOP SIGN/FLASHER** | 0.065791 | NaN | 0.016498 |
| **TRAFFIC_CONTROL_DEVICE_TRAFFIC SIGNAL** | 0.091177 | 0.003722 | 0.026475 |

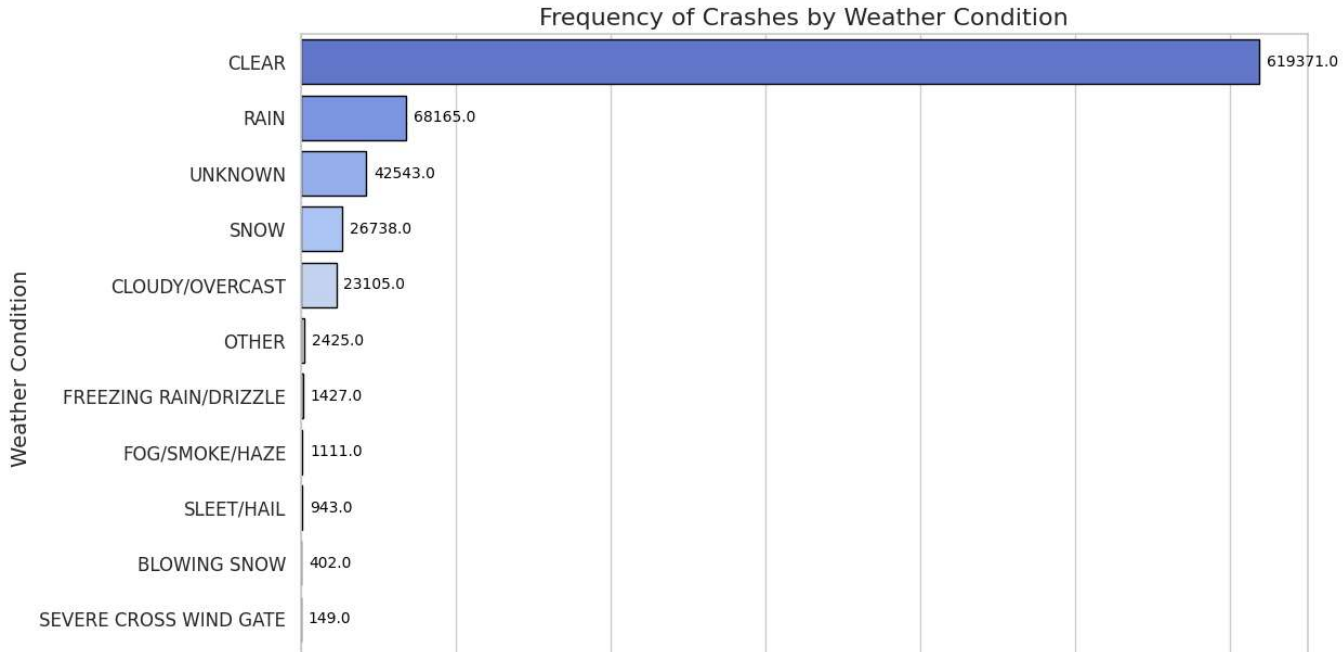
```
# Location-specific analysis
# Grouping data by location and aggregating counts and average severity
location_data = crashes_data_cleaned.groupby(['LATITUDE', 'LONGITUDE']).agg(
    Number_of_Crashes=pd.NamedAgg(column="CRASH_RECORD_ID", aggfunc="count"),
    Avg_Total_Injuries=pd.NamedAgg(column="INJURIES_TOTAL", aggfunc="mean"),
    Avg_Fatal_Injuries=pd.NamedAgg(column="INJURIES_FATAL", aggfunc="mean")
).reset_index()

# Identifying high-risk areas: locations with high numbers of crashes or high average severi
high_risk_locations = location_data.sort_values(by='Number_of_Crashes', ascending=False).hea

# Displaying high-risk locations
high_risk_locations
```

| | LATITUDE | LONGITUDE | Number_of_Crashes | Avg_Total_Injuries | Avg_Fatal_Injuries |
|---|---|---|---|---|---|
| **268479** | 41.976201 | -87.905309 | 1251 | 0.109062 | 0.001604 |
| **183306** | 41.900959 | -87.619928 | 721 | 0.197222 | 0.004167 |
| **82432** | 41.791420 | -87.580148 | 557 | 0.254937 | 0.000000 |
| **42666** | 41.751461 | -87.585972 | 543 | 0.272560 | 0.003683 |
| **22041** | 41.722257 | -87.585276 | 427 | 0.185012 | 0.000000 |
| **45483** | 41.754660 | -87.741385 | 367 | 0.093151 | 0.000000 |
| **154014** | 41.880856 | -87.617636 | 319 | 0.238245 | 0.000000 |
| **80803** | 41.789329 | -87.741646 | 308 | 0.022801 | 0.000000 |
| **183083** | 41.900753 | -87.624235 | 289 | 0.186851 | 0.000000 |
| **178069** | 41.896805 | -87.617027 | 286 | 0.230769 | 0.000000 |

```
# Extracting the year from the crash date and analyzing yearly trends
crashes_data_cleaned['Year'] = crashes_data_cleaned['CRASH_DATE'].dt.year

# Yearly trend of crashes
yearly_trends = crashes_data_cleaned['Year'].value_counts().sort_index()

# Plotting the yearly trend of crashes
plt.figure(figsize=(10, 6))
ax = yearly_trends.plot(kind='bar', color='teal')
plt.title('Yearly Trend of Traffic Crashes')
plt.xlabel('Year')
plt.ylabel('Number of Crashes')
plt.xticks(rotation=45)

# Annotating each bar with its value
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 10), textcoords='offset points', fontsi

plt.show()

# Checking the span of years in the dataset
yearly_trends.index.tolist()
```



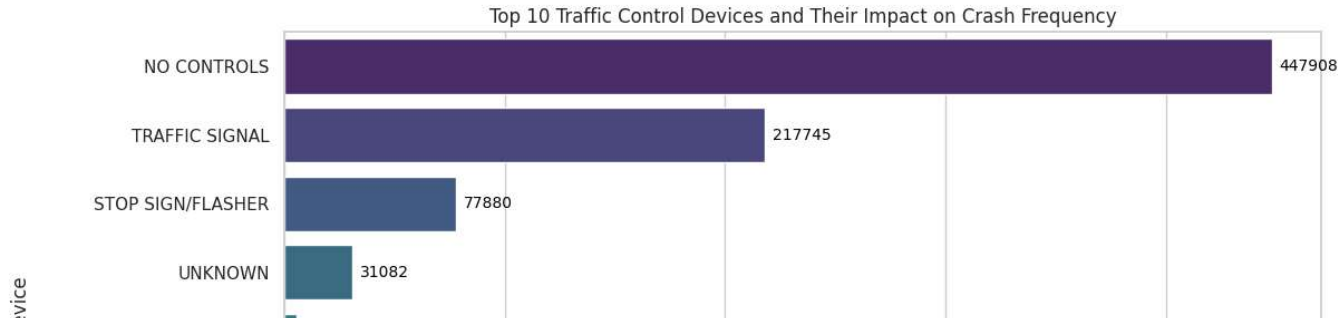

```
import seaborn as sns

# Frequency of crashes under different weather conditions
weather_condition_counts = crashes_data_cleaned['WEATHER_CONDITION'].value_counts()

# Plotting the frequency of crashes under various weather conditions with improved aesthetic
plt.figure(figsize=(12, 8))
ax = sns.barplot(x=weather_condition_counts.values, y=weather_condition_counts.index, palett

plt.title('Frequency of Crashes by Weather Condition', fontsize=16)
plt.xlabel('Number of Crashes', fontsize=14)
plt.ylabel('Weather Condition', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Annotating each bar with its value
for p in ax.patches:
    ax.annotate(f'{p.get_width()}', (p.get_width(), p.get_y() + p.get_height() / 2.),
                ha='left', va='center', xytext=(5, 0), textcoords='offset points', fontsize=

plt.show()
```




```
# Analyzing the impact of traffic control devices on crash frequency and severity
traffic_control_analysis = crashes_data_cleaned.groupby('TRAFFIC_CONTROL_DEVICE').agg(
    Number_of_Crashes=pd.NamedAgg(column="CRASH_RECORD_ID", aggfunc="count"),
    Avg_Total_Injuries=pd.NamedAgg(column="INJURIES_TOTAL", aggfunc="mean"),
    Avg_Fatal_Injuries=pd.NamedAgg(column="INJURIES_FATAL", aggfunc="mean")
).sort_values(by='Number_of_Crashes', ascending=False).head(10)

# Displaying the analysis
traffic_control_analysis
```

| TRAFFIC_CONTROL_DEVICE | Number_of_Crashes | Avg_Total_Injuries | Avg_Fatal_Injuries |
|---|---|---|---|
| **NO CONTROLS** | 447908 | 0.134067 | 0.001174 |
| **TRAFFIC SIGNAL** | 217745 | 0.270836 | 0.001351 |
| **STOP SIGN/FLASHER** | 77880 | 0.297621 | 0.000887 |
| **UNKNOWN** | 31082 | 0.115324 | 0.000774 |
| **OTHER** | 5331 | 0.184948 | 0.001881 |
| **LANE USE MARKING** | 1226 | 0.222404 | 0.000000 |
| **YIELD** | 1158 | 0.324114 | 0.001729 |
| **OTHER REG. SIGN** | 855 | 0.192488 | 0.000000 |
| **OTHER WARNING SIGN** | 646 | 0.192248 | 0.003101 |
| **RAILROAD CROSSING GATE** | 514 | 0.236328 | 0.007812 |

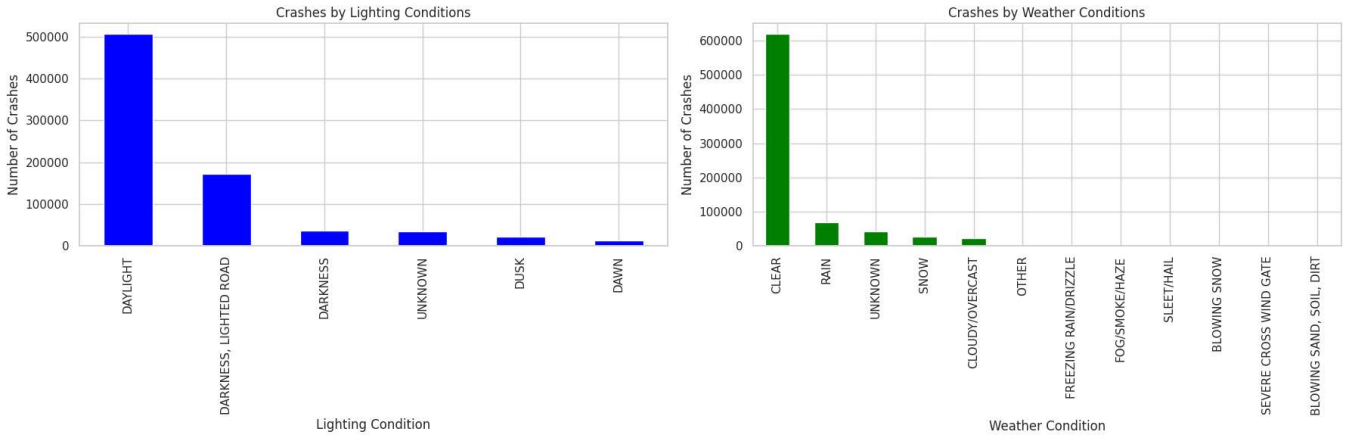
```
# Analyzing the impact of traffic control devices on crash frequency and severity
traffic_control_analysis = crashes_data_cleaned.groupby('TRAFFIC_CONTROL_DEVICE').agg(
    Number_of_Crashes=pd.NamedAgg(column="CRASH_RECORD_ID", aggfunc="count"),
    Avg_Total_Injuries=pd.NamedAgg(column="INJURIES_TOTAL", aggfunc="mean"),
    Avg_Fatal_Injuries=pd.NamedAgg(column="INJURIES_FATAL", aggfunc="mean")
).sort_values(by='Number_of_Crashes', ascending=False).head(10)

# Plotting crash frequency for the top 10 traffic control devices
plt.figure(figsize=(12, 8))
ax = sns.barplot(x='Number_of_Crashes', y=traffic_control_analysis.index, data=traffic_contr

# Adding values at the end of each bar
for p in ax.patches:
    ax.annotate(f'{p.get_width():.0f}', (p.get_width(), p.get_y() + p.get_height() / 2.),
                ha='left', va='center', xytext=(5, 0), textcoords='offset points', fontsize=

plt.title('Top 10 Traffic Control Devices and Their Impact on Crash Frequency')
plt.xlabel('Number of Crashes')
plt.ylabel('Traffic Control Device')
plt.show()
```



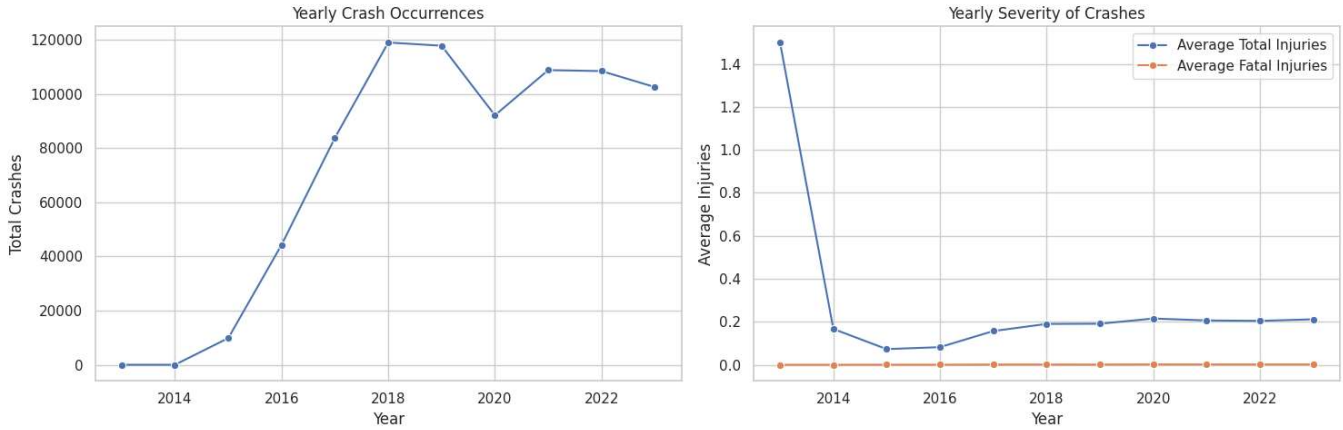

```
# Analyzing the impact of road features on crash frequency and severity
road_feature_analysis = crashes_data_cleaned.groupby('TRAFFICWAY_TYPE').agg(
    Number_of_Crashes=pd.NamedAgg(column="CRASH_RECORD_ID", aggfunc="count"),
    Avg_Total_Injuries=pd.NamedAgg(column="INJURIES_TOTAL", aggfunc="mean"),
    Avg_Fatal_Injuries=pd.NamedAgg(column="INJURIES_FATAL", aggfunc="mean")
).sort_values(by='Number_of_Crashes', ascending=False).head(10)

# Displaying the analysis for trafficway type
road_feature_analysis
```

| TRAFFICWAY_TYPE | Number_of_Crashes | Avg_Total_Injuries | Avg_Fatal_Injuries |
|---|---|---|---|
| NOT DIVIDED | 342724 | 0.179275 | 0.001011 |
| DIVIDED - W/MEDIAN (NOT RAISED) | 126562 | 0.217052 | 0.001440 |
| ONE-WAY | 100670 | 0.101622 | 0.000937 |
| PARKING LOT | 53694 | 0.044344 | 0.000187 |
| FOUR WAY | 47469 | 0.437929 | 0.001981 |
| DIVIDED - W/MEDIAN BARRIER | 45066 | 0.264928 | 0.002646 |
| OTHER | 21542 | 0.169800 | 0.001260 |
| ALLEY | 13013 | 0.100224 | 0.000850 |
| T-INTERSECTION | 9632 | 0.378288 | 0.002288 |

```
# import folium
# from folium import plugins
# from IPython.display import display, HTML

# # Filter out rows with missing coordinates
# crashes_data_cleaned1 = crashes_data_cleaned.dropna(subset=['LATITUDE', 'LONGITUDE'])

# # Creating a base map centered around Chicago
# chicago_map = folium.Map(location=[41.8781, -87.6298], zoom_start=10)

# # Adding markers for each crash location
# for index, row in crashes_data_cleaned1.iterrows():
#     folium.Marker([row['LATITUDE'], row['LONGITUDE']], popup=f"Crash Type: {row['TRAFFICWA

# # Save the map as an HTML string
# html_map = chicago_map._repr_html_()

# # Display the map in Colab using an iframe
# display(HTML(html_map))



# Analyzing crashes by lighting conditions
lighting_conditions_counts = crashes_data_cleaned['LIGHTING_CONDITION'].value_counts()

# Analyzing crashes by weather conditions
weather_conditions_counts = crashes_data_cleaned['WEATHER_CONDITION'].value_counts()

# Plotting the environmental factors
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(18, 6))

# Lighting conditions plot
lighting_conditions_counts.plot(kind='bar', ax=axes[0], color='blue')
axes[0].set_title('Crashes by Lighting Conditions')
axes[0].set_ylabel('Number of Crashes')
axes[0].set_xlabel('Lighting Condition')

# Weather conditions plot
weather_conditions_counts.plot(kind='bar', ax=axes[1], color='green')
axes[1].set_title('Crashes by Weather Conditions')
axes[1].set_ylabel('Number of Crashes')
axes[1].set_xlabel('Weather Condition')

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




```
# Comparative analysis - Yearly trends in crash occurrences and severity
yearly_crash_data = crashes_data_cleaned.groupby('Year').agg(
    Total_Crashes=pd.NamedAgg(column='CRASH_RECORD_ID', aggfunc='count'),
    Avg_Total_Injuries=pd.NamedAgg(column='INJURIES_TOTAL', aggfunc='mean'),
    Avg_Fatal_Injuries=pd.NamedAgg(column='INJURIES_FATAL', aggfunc='mean')
).reset_index()

# Plotting the yearly trends in crash occurrences and severity
plt.figure(figsize=(15, 5))

# Yearly Crash Occurrences
plt.subplot(1, 2, 1)
sns.lineplot(x='Year', y='Total_Crashes', data=yearly_crash_data, marker='o')
plt.title('Yearly Crash Occurrences')
plt.xlabel('Year')
plt.ylabel('Total Crashes')

# Yearly Severity of Crashes
plt.subplot(1, 2, 2)
sns.lineplot(x='Year', y='Avg_Total_Injuries', data=yearly_crash_data, marker='o', label='Av
sns.lineplot(x='Year', y='Avg_Fatal_Injuries', data=yearly_crash_data, marker='o', label='Av
plt.title('Yearly Severity of Crashes')
plt.xlabel('Year')
plt.ylabel('Average Injuries')
plt.legend()

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

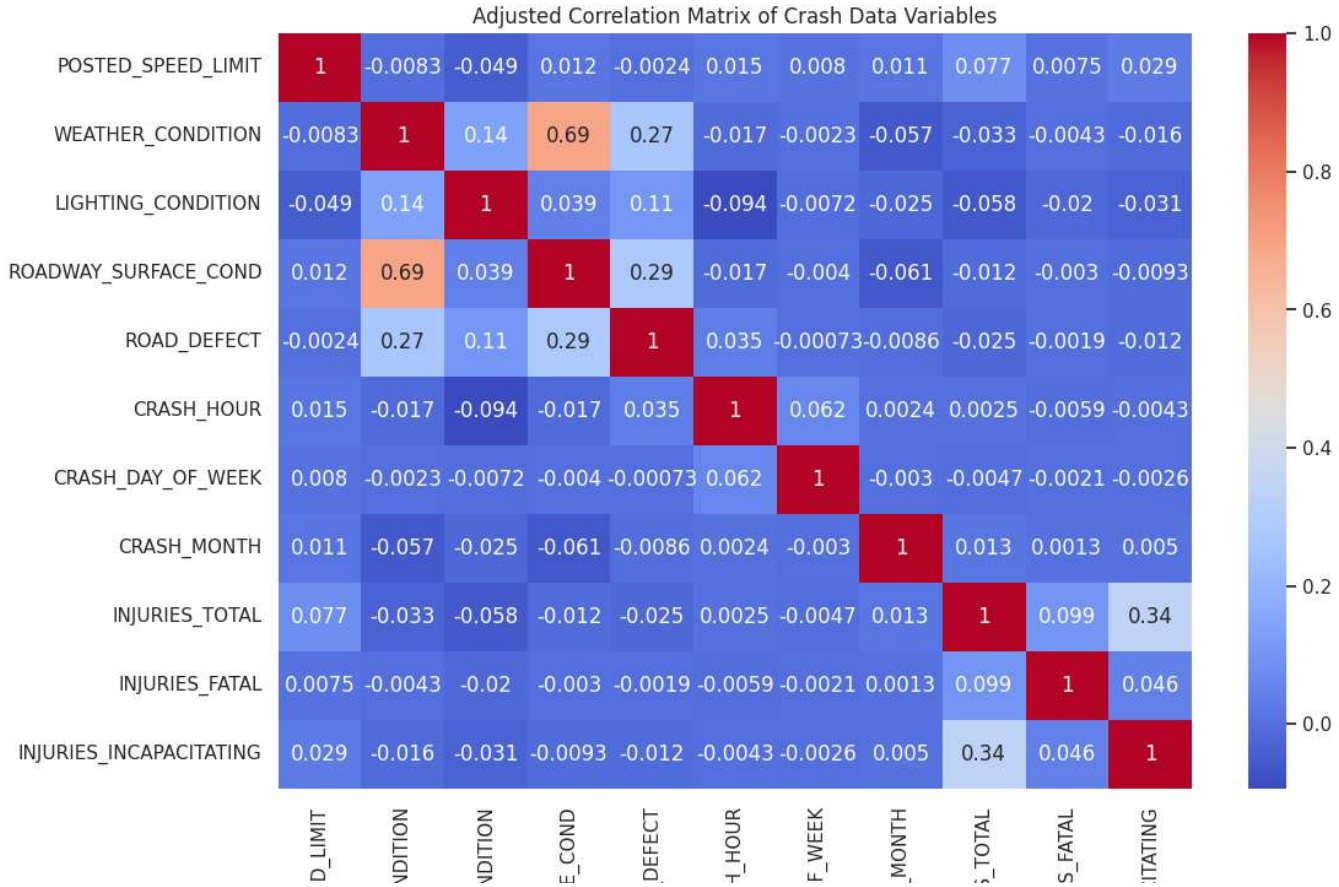



```
# Descriptive cluster analysis - Categorizing crashes based on key characteristics
# Creating categories for time of day
crashes_data_cleaned['TimeOfDay'] = pd.cut(crashes_data_cleaned['CRASH_HOUR'],
                                bins=[0, 6, 12, 18, 24],
                                labels=['Night', 'Morning', 'Afternoon', 'Evening'],
                                right=False)

# Grouping data by categories to identify patterns
cluster_analysis = crashes_data_cleaned.groupby(['TimeOfDay', 'WEATHER_CONDITION', 'TRAFFICW

# Displaying the top 10 categories with the most crashes
top_clusters = cluster_analysis.sort_values(by='Count', ascending=False).head(10)
top_clusters
```

| TimeOfDay | WEATHER_CONDITION | TRAFFICWAY_TYPE | Count |
|---|---|---|---|

```
# Adjusting the list of variables for correlation analysis
correlation_cols_adjusted = [
    'POSTED_SPEED_LIMIT', 'WEATHER_CONDITION', 'LIGHTING_CONDITION',
    'ROADWAY_SURFACE_COND', 'ROAD_DEFECT', 'CRASH_HOUR',
    'CRASH_DAY_OF_WEEK', 'CRASH_MONTH', 'INJURIES_TOTAL',
    'INJURIES_FATAL', 'INJURIES_INCAPACITATING'
]

# Removing 'TRAFFIC_CONTROL_DEVICE' from categorical columns
categorical_cols_adjusted = ['WEATHER_CONDITION', 'LIGHTING_CONDITION', 'ROADWAY_SURFACE_CON

# Selecting and converting relevant columns
correlation_data_adjusted = crashes_data_cleaned[correlation_cols_adjusted]
correlation_data_adjusted[categorical_cols_adjusted] = correlation_data_adjusted[categorical

# Recalculating the correlation matrix
correlation_matrix_adjusted = correlation_data_adjusted.corr()

# Plotting the adjusted correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix_adjusted, annot=True, cmap='coolwarm')
plt.title('Adjusted Correlation Matrix of Crash Data Variables')
plt.show()
```

```
<ipython-input-39-9b2fa4b9d703>:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  correlation_data_adjusted[categorical_cols_adjusted] = correlation_data_adjusted[categ
```




```
# Basic statistics for numerical variables
basic_statistics = crashes_data_cleaned.describe()

# Displaying basic statistics
basic_statistics[['POSTED_SPEED_LIMIT', 'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPA
```

```
POSTED_SPEED_LIMIT  INJURIES_TOTAL  INJURIES_FATAL  INJURIES_INCAPACITATING
```

```
# Distribution of crashes by key categories
category_cols = ['FIRST_CRASH_TYPE', 'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'ROADWAY_SUR

# Calculating the frequency of each category
category_distributions = {col: crashes_data_cleaned[col].value_counts() for col in category_

# Displaying the distribution for each category
category_distributions['FIRST_CRASH_TYPE'].head(10), category_distributions['WEATHER_CONDITI
```

```
(PARKED MOTOR VEHICLE            182942
 REAR END                        176711
 SIDESWIPE SAME DIRECTION        119840
 TURNING                         112121
 ANGLE                            85202
 FIXED OBJECT                     36913
 PEDESTRIAN                       18153
 PEDALCYCLIST                     11990
 SIDESWIPE OPPOSITE DIRECTION     11116
 OTHER OBJECT                      7802
 Name: FIRST_CRASH_TYPE, dtype: int64,
 CLEAR                    619371
 RAIN                      68165
 UNKNOWN                   42543
 SNOW                      26738
 CLOUDY/OVERCAST           23105
 OTHER                      2425
 FREEZING RAIN/DRIZZLE      1427
 FOG/SMOKE/HAZE             1111
 SLEET/HAIL                  943
 BLOWING SNOW                402
 Name: WEATHER_CONDITION, dtype: int64,
 DAYLIGHT                  505920
 DARKNESS, LIGHTED ROAD    172158
 DARKNESS                   37209
 UNKNOWN                    35366
 DUSK                       22636
 DAWN                       13097
 Name: LIGHTING_CONDITION, dtype: int64,
 DRY                583615
 WET                103555
 UNKNOWN             65112
 SNOW OR SLUSH       26573
 ICE                  5269
 OTHER                1966
 SAND, MUD, DIRT       296
 Name: ROADWAY_SURFACE_COND, dtype: int64)
```

```
#!jupyter nbconvert --to html /content/drive/MyDrive/Colab\ Notebooks/Traffic_Crashes.ipynb
```