```
from google.colab import drive

# Mount Google Drive to /content/drive
drive.mount('/content/drive')
```

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

```
file_path = "//content/drive/MyDrive/crash data set chicago /Traffic_Crashes_-_Crashes_20241115.csv "
```

```
#importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)
```

+ Code + Text

```
#loading the data
df = pd.read_csv("/content/drive/MyDrive/crash data set chicago /Traffic_Crashes_-_Crashes_20241115.csv")
```

```
# Checking the shape of the dataframe
print("Shape of DataFrame:", df.shape)
```

```
Shape of DataFrame: (892977, 48)
```

```
#printing first 5 rows
df.head()
```

	CRASH_RECORD_ID	CRASH_DATE_EST_I	CRASH_DATE	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CO
0	6c1659069e9c6285a650e70d6f9b574ed5f64c12888479...	NaN	08/18/2023 12:50:00 PM	15	OTHER	FUNCT PR(
1	5f54a59fcb087b12ae5b1acff96a3caf4f2d37e79f8db4...	NaN	07/29/2023 02:45:00 PM	30	TRAFFIC SIGNAL	FUNCT PR(
2	61fcb8c1eb522a6469b460e2134df3d15f82e81fd93e9c...	NaN	08/18/2023 05:58:00 PM	30	NO CONTROLS	NO CON
3	004cd14d0303a9163aad69a2d7f341b7da2a8572b2ab33...	NaN	11/26/2019 08:38:00 AM	25	NO CONTROLS	NO CON
4	a1d5f0ea90897745365a4cbb06cc60329a120d89753fac...	NaN	08/18/2023 10:45:00 AM	20	NO CONTROLS	NO CON

```
# Checking for missing values
missing_values = df.isna().sum()
total_missing = missing_values.sum()
```

```
print("Missing values in each column:")
print(missing_values)
```

```
Missing values in each column:
CRASH_RECORD_ID                     0
CRASH_DATE_EST_I               827002
CRASH_DATE                          0
POSTED_SPEED_LIMIT                  0
TRAFFIC_CONTROL_DEVICE              0
DEVICE_CONDITION                    0
WEATHER_CONDITION                   0
LIGHTING_CONDITION                  0
```

```
FIRST_CRASH_TYPE                     0
TRAFFICWAY_TYPE                      0
LANE_CNT                        693957
ALIGNMENT                            0
ROADWAY_SURFACE_COND                 0
ROAD_DEFECT                          0
REPORT_TYPE                      27660
CRASH_TYPE                           0
INTERSECTION_RELATED_I          687989
NOT_RIGHT_OF_WAY_I              852282
HIT_AND_RUN_I                   612878
DAMAGE                               0
DATE_POLICE_NOTIFIED                 0
PRIM_CONTRIBUTORY_CAUSE              0
SEC_CONTRIBUTORY_CAUSE               0
STREET_NO                            0
STREET_DIRECTION                     4
STREET_NAME                          1
BEAT_OF_OCCURRENCE                   5
PHOTOS_TAKEN_I                  880793
STATEMENTS_TAKEN_I              872438
DOORING_I                       890144
WORK_ZONE_I                     887973
WORK_ZONE_TYPE                  889110
WORKERS_PRESENT_I               891688
NUM_UNITS                            0
MOST_SEVERE_INJURY                1980
INJURIES_TOTAL                    1966
INJURIES_FATAL                    1966
INJURIES_INCAPACITATING           1966
INJURIES_NON_INCAPACITATING       1966
INJURIES_REPORTED_NOT_EVIDENT     1966
INJURIES_NO_INDICATION            1966
INJURIES_UNKNOWN                  1966
CRASH_HOUR                           0
CRASH_DAY_OF_WEEK                    0
CRASH_MONTH                          0
LATITUDE                          6424
LONGITUDE                         6424
LOCATION                          6424
dtype: int64
```

```
print("Total missing values:", total_missing)
```

```
Total missing values: 9048938
```

```
# Checking for total null values
total_null = df.isnull().sum().sum()
print("Total null values:", total_null)
```

```
Total null values: 9048938
```

```
# Checking value counts for CRASH_TYPE
print("Value counts for CRASH_TYPE:")
print(df['CRASH_TYPE'].value_counts())
```

```
Value counts for CRASH_TYPE:
CRASH_TYPE
NO INJURY / DRIVE AWAY              652729
INJURY AND / OR TOW DUE TO CRASH    240248
Name: count, dtype: int64
```

```
# Renaming CRASH_TYPE to SEVERE and mapping values
df.rename(columns={'CRASH_TYPE': 'SEVERE'}, inplace=True)
df['SEVERE'] = df['SEVERE'].map(lambda x: 1 if x == 'INJURY AND / OR TOW DUE TO CRASH' else 0)
```

```
# Checking value counts for SEVERE
print("Value counts for SEVERE:")
print(df['SEVERE'].value_counts())
```

```
Value counts for SEVERE:
SEVERE
0    652729
1    240248
Name: count, dtype: int64
```

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


# Extracting time features
df['HOUR'] = df['CRASH_DATE'].dt.hour
df['DAY_OF_WEEK'] = df['CRASH_DATE'].dt.dayofweek  # Monday=0, Sunday=6
df['MONTH'] = df['CRASH_DATE'].dt.month


# Displaying the first two rows of the updated DataFrame
print("First two rows of the DataFrame after date conversion:")
print(df.head(2))
```

```
First two rows of the DataFrame after date conversion:
                                CRASH_RECORD_ID CRASH_DATE_EST_I  \
0  6c1659069e9c6285a650e70d6f9b574ed5f64c12888479...              NaN
1  5f54a59fcb087b12ae5b1acff96a3caf4f2d37e79f8db4...              NaN

            CRASH_DATE  POSTED_SPEED_LIMIT TRAFFIC_CONTROL_DEVICE  \
0 2023-08-18 12:50:00                  15                 OTHER
1 2023-07-29 14:45:00                  30         TRAFFIC SIGNAL

       DEVICE_CONDITION WEATHER_CONDITION LIGHTING_CONDITION  \
0  FUNCTIONING PROPERLY             CLEAR           DAYLIGHT
1  FUNCTIONING PROPERLY             CLEAR           DAYLIGHT

       FIRST_CRASH_TYPE                  TRAFFICWAY_TYPE  LANE_CNT  \
0              REAR END                           OTHER       NaN
1  PARKED MOTOR VEHICLE  DIVIDED - W/MEDIAN (NOT RAISED)       NaN

            ALIGNMENT ROADWAY_SURFACE_COND ROAD_DEFECT REPORT_TYPE  SEVERE  \
0  STRAIGHT AND LEVEL                 DRY  NO DEFECTS    ON SCENE       1
1  STRAIGHT AND LEVEL                 DRY  NO DEFECTS    ON SCENE       0

  INTERSECTION_RELATED_I NOT_RIGHT_OF_WAY_I HIT_AND_RUN_I       DAMAGE  \
0                    NaN               NaN           NaN  OVER $1,500
1                    NaN               NaN             Y  OVER $1,500

     DATE_POLICE_NOTIFIED                PRIM_CONTRIBUTORY_CAUSE  \
0  08/18/2023 12:55:00 PM                  FOLLOWING TOO CLOSELY
1  07/29/2023 02:45:00 PM  FAILING TO REDUCE SPEED TO AVOID CRASH

                           SEC_CONTRIBUTORY_CAUSE  STREET_NO  \
0                  DISTRACTION - FROM INSIDE VEHICLE        700
1  OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELE...       2101

  STREET_DIRECTION  STREET_NAME  BEAT_OF_OCCURRENCE PHOTOS_TAKEN_I  \
0                W     OHARE ST               1654.0            NaN
1                S  ASHLAND AVE               1235.0            NaN

  STATEMENTS_TAKEN_I DOORING_I WORK_ZONE_I WORK_ZONE_TYPE WORKERS_PRESENT_I  \
0                NaN       NaN         NaN            NaN               NaN
1                NaN       NaN         NaN            NaN               NaN

   NUM_UNITS        MOST_SEVERE_INJURY  INJURIES_TOTAL  INJURIES_FATAL  \
0          2  NONINCAPACITATING INJURY             1.0             0.0
1          4   NO INDICATION OF INJURY             0.0             0.0

   INJURIES_INCAPACITATING  INJURIES_NON_INCAPACITATING  \
0                      0.0                          1.0
1                      0.0                          0.0

   INJURIES_REPORTED_NOT_EVIDENT  INJURIES_NO_INDICATION  INJURIES_UNKNOWN  \
0                            0.0                     1.0               0.0
1                            0.0                     1.0               0.0

   CRASH_HOUR  CRASH_DAY_OF_WEEK  CRASH_MONTH  LATITUDE  LONGITUDE  \
0          12                  6            8       NaN        NaN
1          14                  7            7  41.85412 -87.665902

                                LOCATION  HOUR  DAY_OF_WEEK  MONTH
```

```
# Finding value counts of specified columns using a for loop
columns = [
    'INTERSECTION_RELATED_I',
    'ALIGNMENT',
    'NOT_RIGHT_OF_WAY_I',
    'HIT_AND_RUN_I',
    'WORK_ZONE_I',
    'MOST_SEVERE_INJURY',
    'FIRST_CRASH_TYPE',
```

```
    'PRIM_CONTRIBUTORY_CAUSE'
]

for col in columns:
    print(f"Value counts for {col}:")
    print(df[col].value_counts())
    print('_____________________________________________________________________')
```

```
Value counts for INTERSECTION_RELATED_I:
INTERSECTION_RELATED_I
Y    195222
N      9766
Name: count, dtype: int64
_____________________________________________________________________
Value counts for ALIGNMENT:
ALIGNMENT
STRAIGHT AND LEVEL       871824
STRAIGHT ON GRADE         10909
CURVE, LEVEL               6299
STRAIGHT ON HILLCREST      2253
CURVE ON GRADE             1306
CURVE ON HILLCREST          386
Name: count, dtype: int64
_____________________________________________________________________
Value counts for NOT_RIGHT_OF_WAY_I:
NOT_RIGHT_OF_WAY_I
Y    36957
N     3738
Name: count, dtype: int64
_____________________________________________________________________
Value counts for HIT_AND_RUN_I:
HIT_AND_RUN_I
Y    268097
N     12002
Name: count, dtype: int64
_____________________________________________________________________
Value counts for WORK_ZONE_I:
WORK_ZONE_I
Y    3867
N    1137
Name: count, dtype: int64
_____________________________________________________________________
Value counts for MOST_SEVERE_INJURY:
MOST_SEVERE_INJURY
NO INDICATION OF INJURY     765630
NONINCAPACITATING INJURY     70455
REPORTED, NOT EVIDENT        38958
INCAPACITATING INJURY        14978
FATAL                          976
Name: count, dtype: int64
_____________________________________________________________________
Value counts for FIRST_CRASH_TYPE:
FIRST_CRASH_TYPE
PARKED MOTOR VEHICLE            206854
REAR END                        197598
SIDESWIPE SAME DIRECTION        137046
TURNING                         128279
ANGLE                            97066
FIXED OBJECT                     41509
PEDESTRIAN                       21100
PEDALCYCLIST                     14221
SIDESWIPE OPPOSITE DIRECTION     12421
REAR TO FRONT                     9109
OTHER OBJECT                      8911
HEAD ON                           7566
REAR TO SIDE                      5441
```

```
# Lambda functions to change values to binary
df['INTERSECTION_RELATED_I'] = df['INTERSECTION_RELATED_I'].map(lambda x: 1 if x == 'Y' else 0)
df['NOT_RIGHT_OF_WAY_I'] = df['NOT_RIGHT_OF_WAY_I'].map(lambda x: 1 if x == 'Y' else 0)
df['HIT_AND_RUN_I'] = df['HIT_AND_RUN_I'].map(lambda x: 1 if x == 'Y' else 0)


# Replacing null values with 0 and converting WORK_ZONE_I to binary
df['WORK_ZONE_I'] = df['WORK_ZONE_I'].fillna(0).map(lambda x: 1 if x == 'Y' else 0)


# Filling null values for specified columns
col = [
    'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITATING',
    'INJURIES_NON_INCAPACITATING', 'INJURIES_REPORTED_NOT_EVIDENT',
```

```
    'INJURIES_NO_INDICATION', 'LANE_CNT'
]
for i in col:
    df[i] = df[i].fillna(0)  # Direct assignment to avoid chained assignment warning


# Handling missing values in MOST_SEVERE_INJURY
df['MOST_SEVERE_INJURY'] = df['MOST_SEVERE_INJURY'].fillna('Unknown')


drop_list = ['CRASH_DATE_EST_I', 'RD_NO','REPORT_TYPE','DATE_POLICE_NOTIFIED',
             'STREET_DIRECTION', 'BEAT_OF_OCCURRENCE','PHOTOS_TAKEN_I','STATEMENTS_TAKEN_I',
             'DOORING_I','WORK_ZONE_TYPE', 'WORKERS_PRESENT_I','LATITUDE',
             'LONGITUDE','CRASH_RECORD_ID','INJURIES_UNKNOWN','STREET_NO',
             'MOST_SEVERE_INJURY','SEC_CONTRIBUTORY_CAUSE','LOCATION','STREET_NAME']


existing_drop_list = [col for col in drop_list if col in df.columns]
df.drop(columns=existing_drop_list, inplace=True)


print("Shape of DataFrame after dropping columns:", df.shape)
df.info()
```

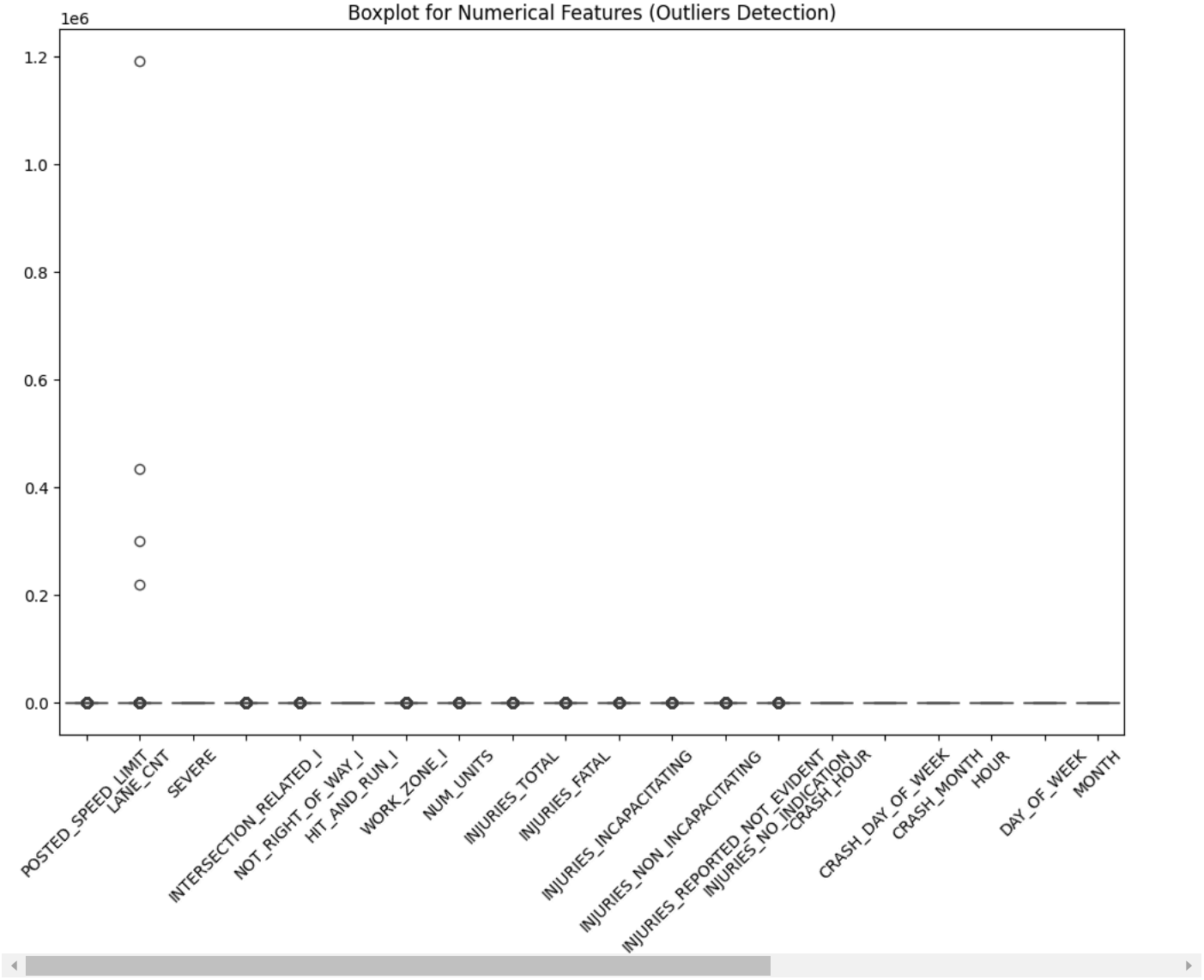
```
Shape of DataFrame after dropping columns: (892977, 32)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 892977 entries, 0 to 892976
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   CRASH_DATE                      892977 non-null  datetime64[ns]
 1   POSTED_SPEED_LIMIT              892977 non-null  int64
 2   TRAFFIC_CONTROL_DEVICE          892977 non-null  object
 3   DEVICE_CONDITION                892977 non-null  object
 4   WEATHER_CONDITION               892977 non-null  object
 5   LIGHTING_CONDITION              892977 non-null  object
 6   FIRST_CRASH_TYPE                892977 non-null  object
 7   TRAFFICWAY_TYPE                 892977 non-null  object
 8   LANE_CNT                        892977 non-null  float64
 9   ALIGNMENT                       892977 non-null  object
 10  ROADWAY_SURFACE_COND            892977 non-null  object
 11  ROAD_DEFECT                     892977 non-null  object
 12  SEVERE                          892977 non-null  int64
 13  INTERSECTION_RELATED_I          892977 non-null  int64
 14  NOT_RIGHT_OF_WAY_I              892977 non-null  int64
 15  HIT_AND_RUN_I                   892977 non-null  int64
 16  DAMAGE                          892977 non-null  object
 17  PRIM_CONTRIBUTORY_CAUSE         892977 non-null  object
 18  WORK_ZONE_I                     892977 non-null  int64
 19  NUM_UNITS                       892977 non-null  int64
 20  INJURIES_TOTAL                  892977 non-null  float64
 21  INJURIES_FATAL                  892977 non-null  float64
 22  INJURIES_INCAPACITATING         892977 non-null  float64
 23  INJURIES_NON_INCAPACITATING     892977 non-null  float64
 24  INJURIES_REPORTED_NOT_EVIDENT   892977 non-null  float64
 25  INJURIES_NO_INDICATION          892977 non-null  float64
 26  CRASH_HOUR                      892977 non-null  int64
 27  CRASH_DAY_OF_WEEK               892977 non-null  int64
 28  CRASH_MONTH                     892977 non-null  int64
 29  HOUR                            892977 non-null  int32
 30  DAY_OF_WEEK                     892977 non-null  int32
 31  MONTH                           892977 non-null  int32
dtypes: datetime64[ns](1), float64(7), int32(3), int64(10), object(11)
memory usage: 207.8+ MB
```

```
print(df['INJURIES_NO_INDICATION'].value_counts())
df['INJURIES_NO_INDICATION'].values[df['INJURIES_NO_INDICATION'] > 10] =10
print(df['INJURIES_NO_INDICATION'].value_counts())
```

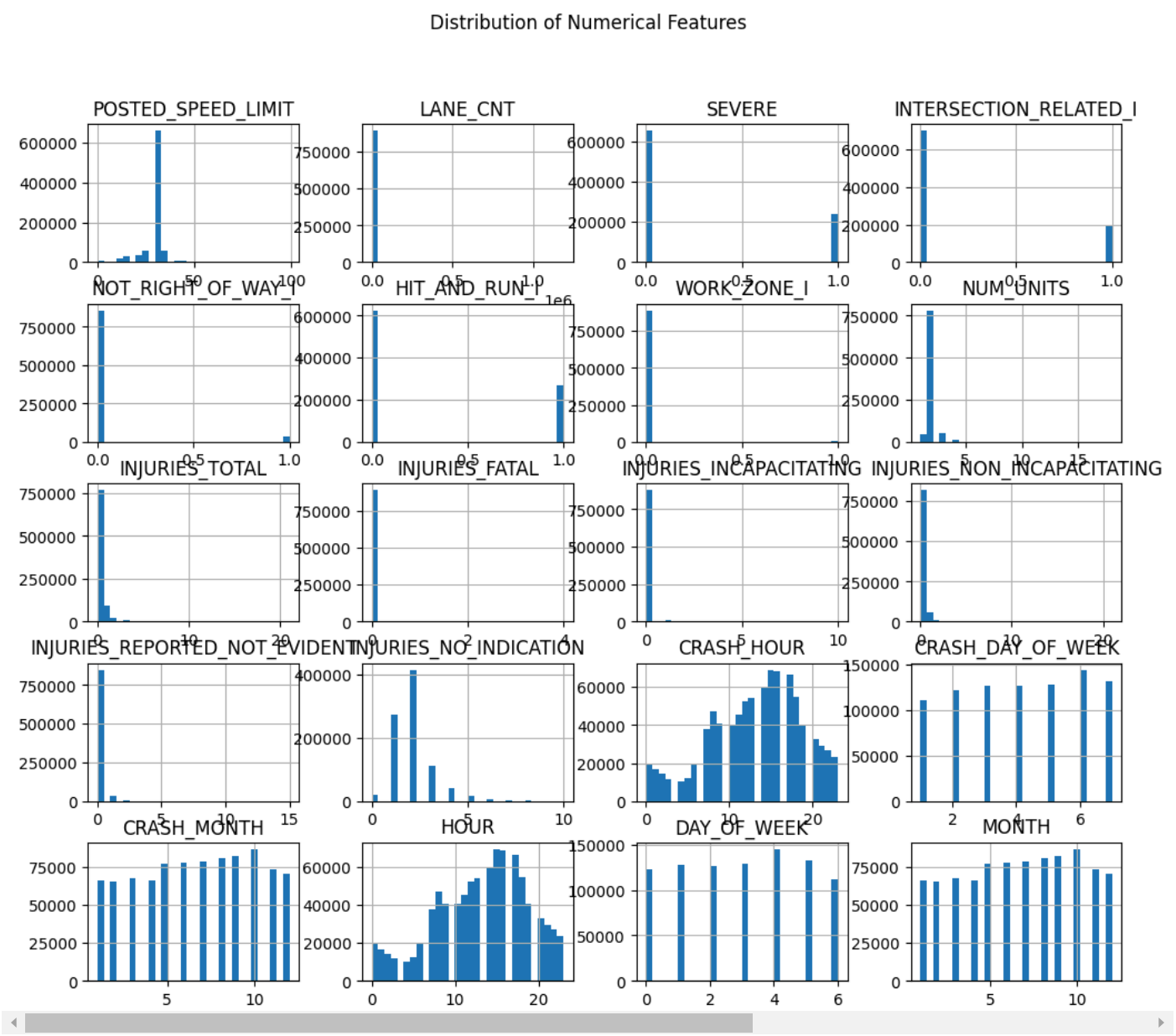
```
INJURIES_NO_INDICATION
2.0     414214
1.0     273234
3.0     112245
4.0      41927
0.0      21091
5.0      17610
6.0       7306
7.0       2845
8.0       1267
9.0        523
10.0       256
11.0       120
```

```
12.0        77
13.0        45
14.0        41
15.0        23
16.0        22
17.0        18
20.0        13
19.0         9
29.0         7
21.0         7
26.0         6
37.0         6
22.0         6
27.0         5
30.0         5
18.0         5
24.0         5
25.0         4
31.0         4
36.0         4
42.0         3
28.0         3
33.0         2
34.0         2
45.0         2
32.0         2
40.0         2
35.0         1
23.0         1
41.0         1
39.0         1
50.0         1
43.0         1
48.0         1
38.0         1
61.0         1
46.0         1
49.0         1
Name: count, dtype: int64
INJURIES_NO_INDICATION
2.0     414214
1.0     273234
3.0     112245
4.0      41927
0.0      21091
```

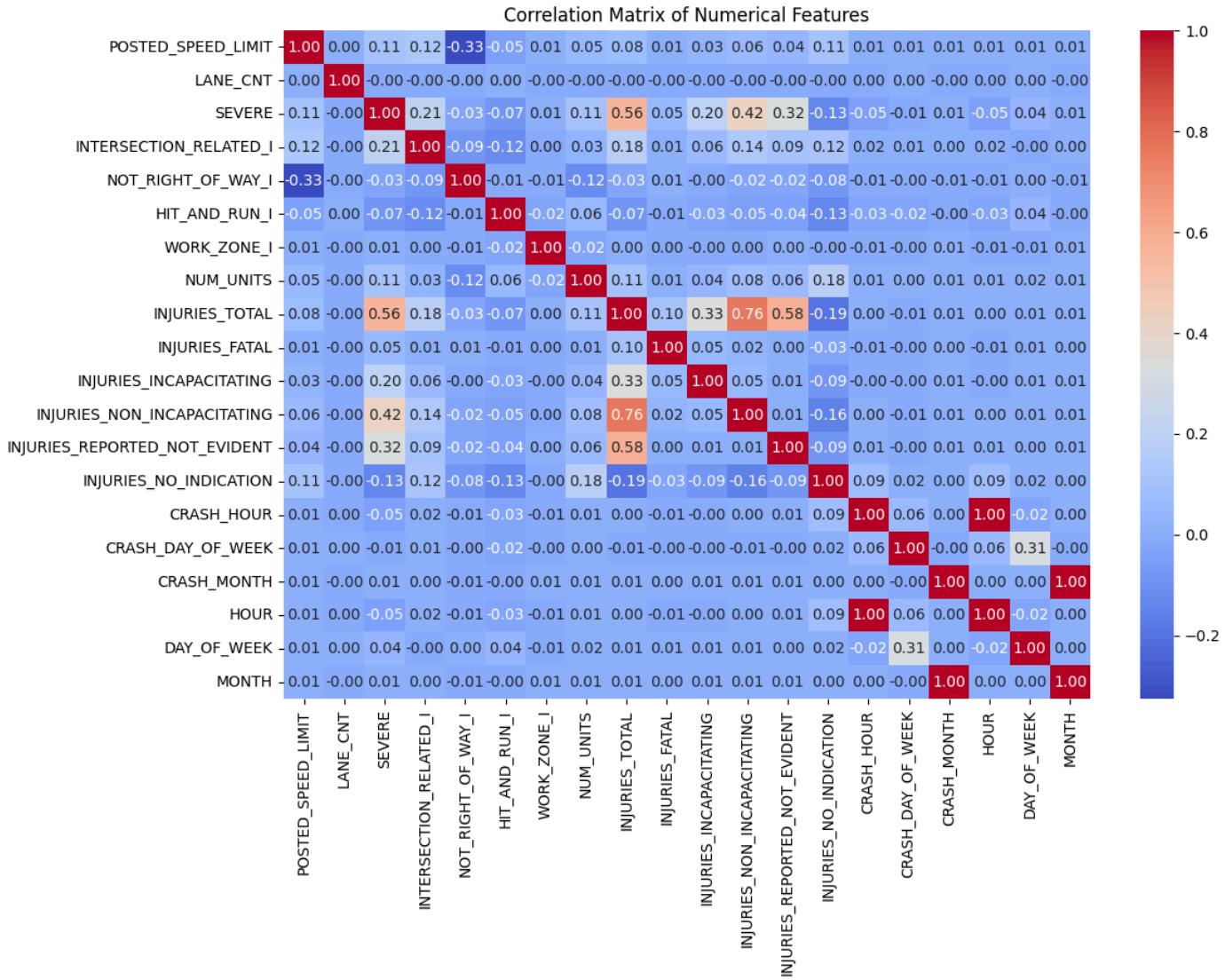
```
# Plotting boxplots for numerical features to detect outliers
numerical_columns = df.select_dtypes(include=[np.number]).columns
plt.figure(figsize=(12, 8))
sns.boxplot(data=df[numerical_columns])
plt.title('Boxplot for Numerical Features (Outliers Detection)')
plt.xticks(rotation=45)
plt.show()
```



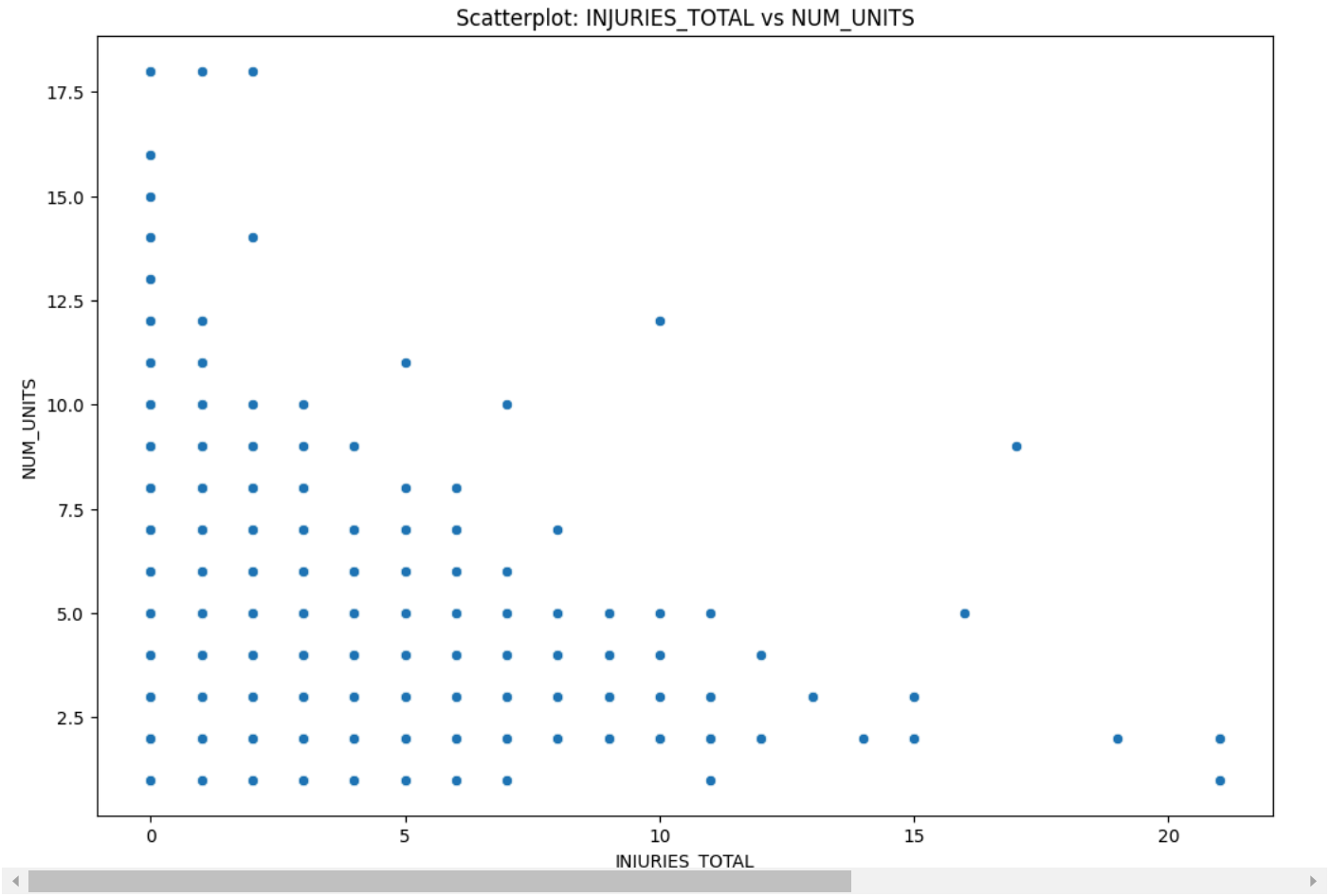

```
# Step 2: Univariate Analysis
# Histograms for numerical columns
df[numerical_columns].hist(figsize=(12, 10), bins=30)
plt.suptitle('Distribution of Numerical Features')
plt.show()
```



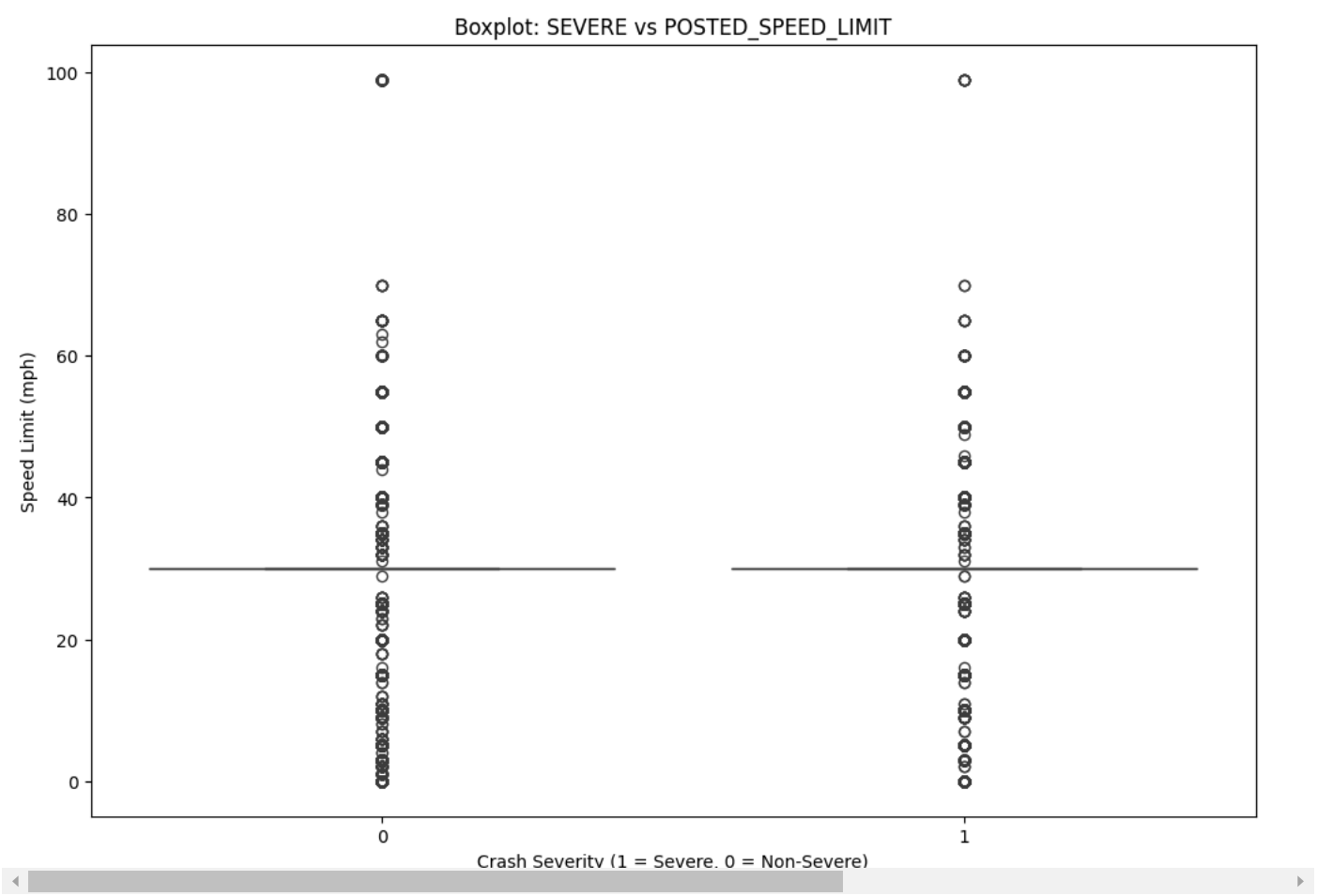

```
# Step 3: Bivariate Analysis#
# Correlation Matrix for numerical features
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 8))
corr_matrix = df[numerical_columns].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```




```
# Scatterplot between numerical features (Example: 'INJURIES_TOTAL' vs 'NUM_UNITS')
plt.figure(figsize=(12, 8))
sns.scatterplot(x='INJURIES_TOTAL', y='NUM_UNITS', data=df)
plt.title('Scatterplot: INJURIES_TOTAL vs NUM_UNITS')
plt.xlabel('INJURIES_TOTAL')
plt.ylabel('NUM_UNITS')
plt.show()
```

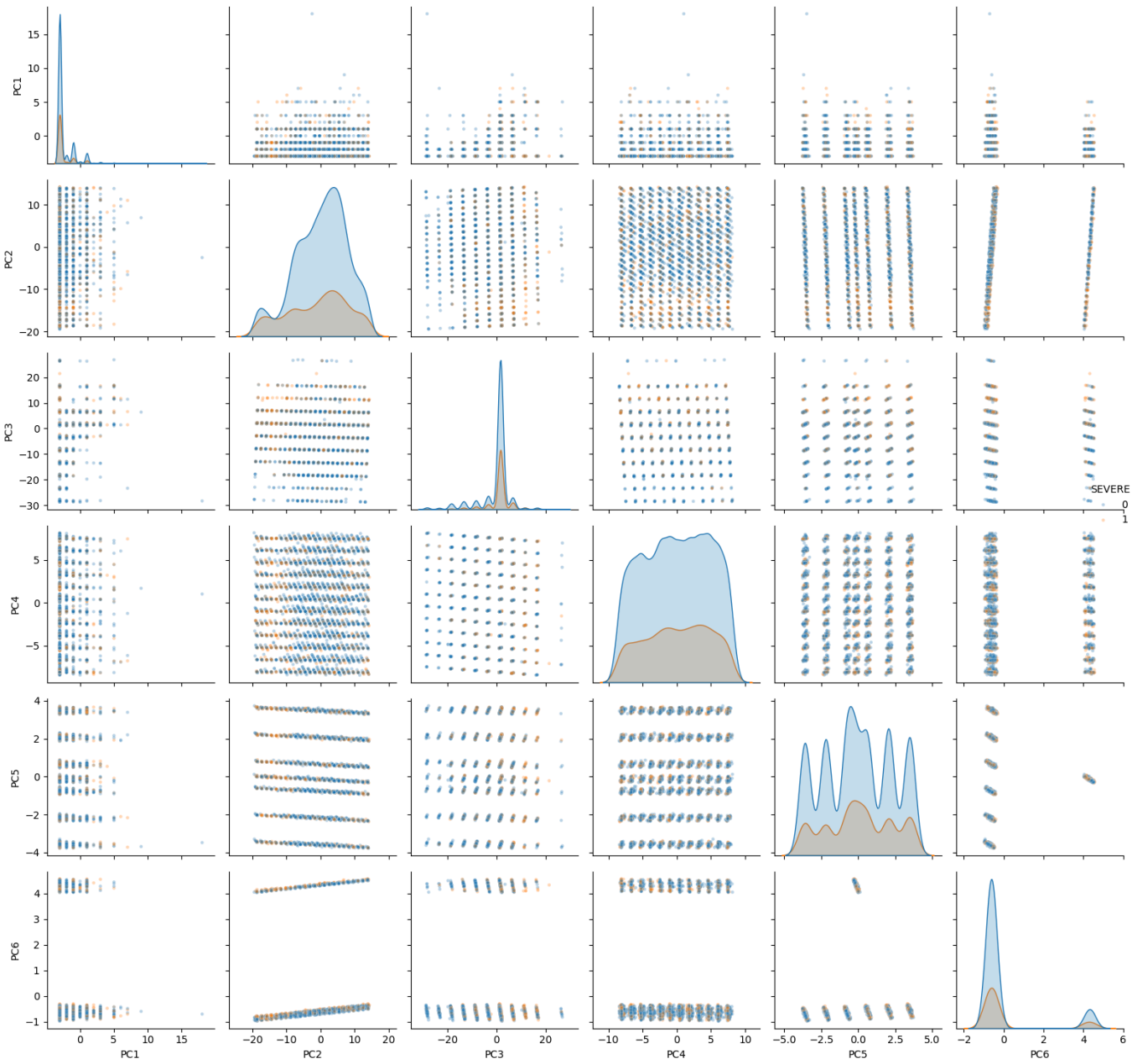



```
plt.figure(figsize=(12, 8))
sns.boxplot(x='SEVERE', y='POSTED_SPEED_LIMIT', data=df)
plt.title('Boxplot: SEVERE vs POSTED_SPEED_LIMIT')
plt.xlabel('Crash Severity (1 = Severe, 0 = Non-Severe)')
plt.ylabel('Speed Limit (mph)')
plt.show()
```



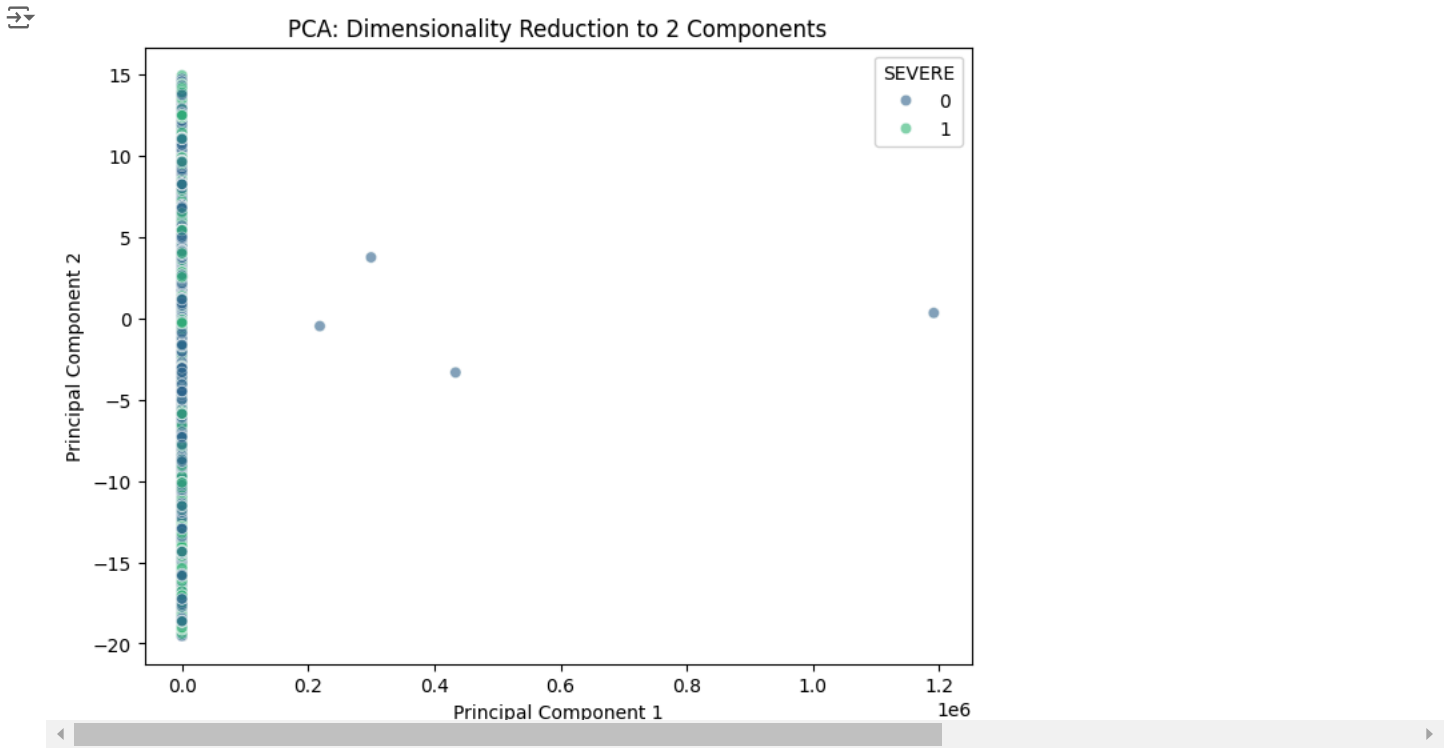

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Step 4: Multivariate Analysis


# 1. Select a subset of features or use dimensionality reduction
if len(numerical_columns) > 6:  # If there are too many features, use PCA
    pca = PCA(n_components=6)
    pca_result = pca.fit_transform(df[numerical_columns])
    pca_df = pd.DataFrame(data=pca_result, columns=[f'PC{i+1}' for i in range(6)])
    pca_df['SEVERE'] = df['SEVERE']
    plot_df = pca_df
else:
    plot_df = df[numerical_columns + ['SEVERE']].copy()

# 2. Sample the data if it's too large
if len(plot_df) > 10000:
    plot_df = plot_df.sample(n=10000, random_state=42)

# 3. Create the pairplot with optimized settings

g = sns.pairplot(plot_df, hue='SEVERE', plot_kws={'alpha': 0.3, 's': 10}, diag_kws={'warn_singular': False})

plt.suptitle('Pairplot: Relationships between Numerical Features', y=1.02)
plt.tight_layout()
plt.show()
```

Pairplot: Relationships between Numerical Features



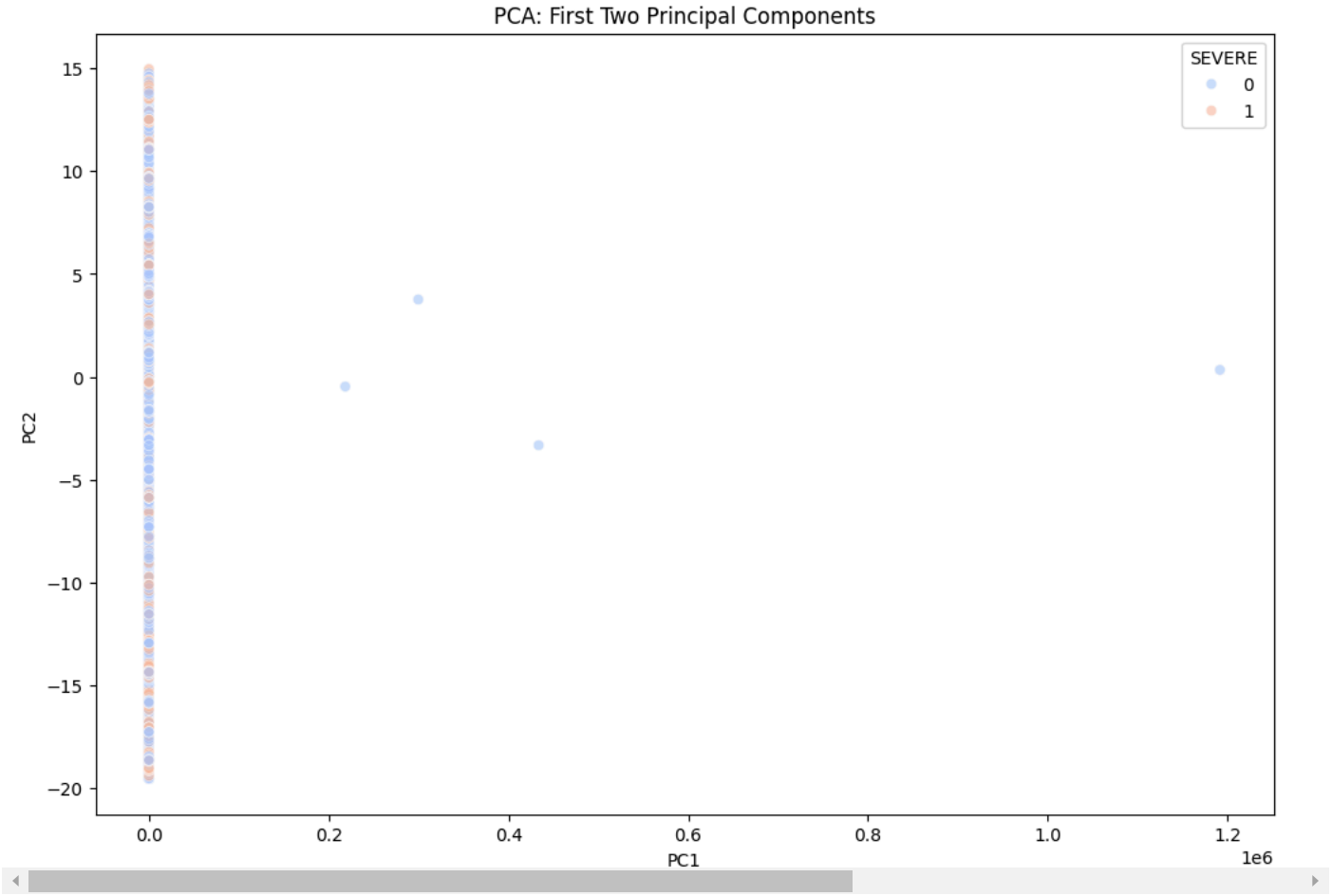

```
# PCA (Principal Component Analysis) for dimensionality reduction
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

```
# Perform PCA to reduce dimensions to 2 components
pca = PCA(n_components=2)
principal_components = pca.fit_transform(df[numerical_columns])

# Create a DataFrame with the PCA results
df_pca = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
df_pca['SEVERE'] = df['SEVERE']  # Add the target variable for visualization

# Visualize the two principal components
plt.figure(figsize=(8, 6))
sns.scatterplot(
    x='PC1',
    y='PC2',
    hue='SEVERE',
    data=df_pca,
    palette='viridis',
    alpha=0.6
)
plt.title('PCA: Dimensionality Reduction to 2 Components')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='SEVERE')
plt.show()
```



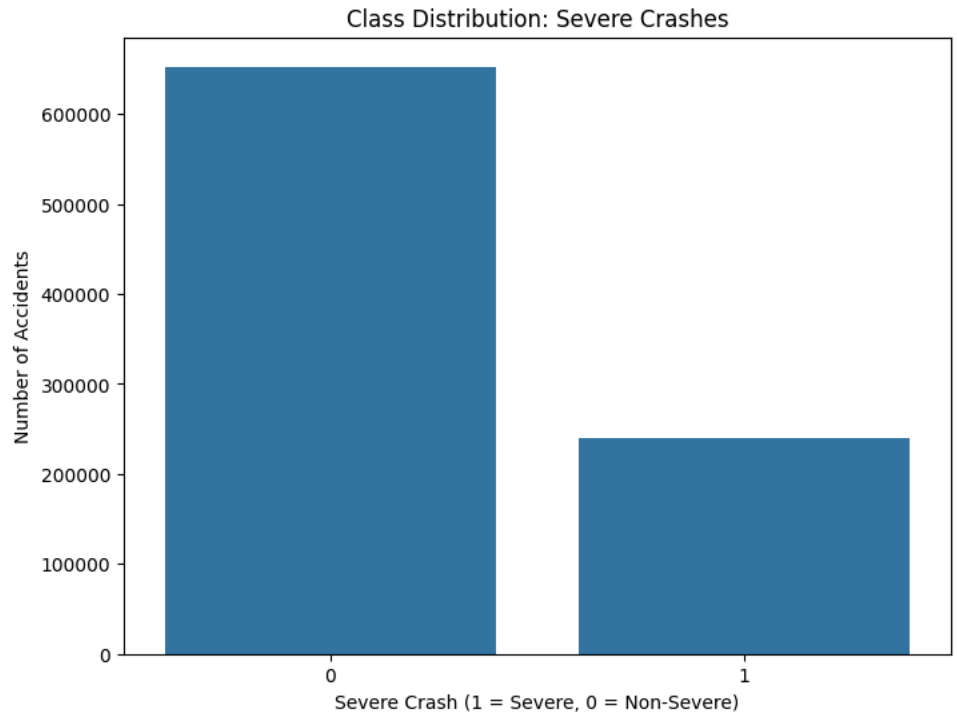

```
# PCA (Principal Component Analysis) for dimensionality reduction
pca = PCA(n_components=2)
principal_components = pca.fit_transform(df[numerical_columns])
df_pca = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])


# Adding target variable to PCA results for visualization
df_pca['SEVERE'] = df['SEVERE']


# Plotting the first two principal components
plt.figure(figsize=(12, 8))
# Plotting the first two principal components
plt.figure(figsize=(12, 8))
sns.scatterplot(x='PC1', y='PC2', hue='SEVERE', data=df_pca, palette='coolwarm', alpha=0.6)
plt.title('PCA: First Two Principal Components')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```




```
  # Step 5: Class Imbalance Check (for Classification Tasks)# ------------------------------------------------
# Checking for class imbalance in the target variable 'SEVERE'
severe_counts = df['SEVERE'].value_counts()
print('Class Distribution for Severe Crashes (Imbalance Check):')
print(severe_counts)
# Plotting the class distribution
plt.figure(figsize=(8, 6))
sns.countplot(x='SEVERE', data=df)
plt.title('Class Distribution: Severe Crashes')
plt.xlabel('Severe Crash (1 = Severe, 0 = Non-Severe)')
plt.ylabel('Number of Accidents')
plt.show()
# Step 6: Summary Statistics# ------------------------------------------------
# Summary statistics for numerical columns
print("Summary Statistics for Numerical Features:")
print(df[numerical_columns].describe())
```

```
Class Distribution for Severe Crashes (Imbalance Check):
SEVERE
0    652729
1    240248
Name: count, dtype: int64
```




```
Summary Statistics for Numerical Features:
       POSTED_SPEED_LIMIT      LANE_CNT         SEVERE  \
count       892977.000000  8.929770e+05  892977.000000
mean            28.421275  2.970784e+00       0.269042
std              6.111311  1.398123e+03       0.443462
min              0.000000  0.000000e+00       0.000000
25%             30.000000  0.000000e+00       0.000000
50%             30.000000  0.000000e+00       0.000000
75%             30.000000  0.000000e+00       1.000000
max             99.000000  1.191625e+06       1.000000

       INTERSECTION_RELATED_I  NOT_RIGHT_OF_WAY_I  HIT_AND_RUN_I  \
count           892977.000000       892977.000000  892977.000000
mean                0.218619            0.041386       0.300228
std                 0.413310            0.199182       0.458357
min                 0.000000            0.000000       0.000000
25%                 0.000000            0.000000       0.000000
50%                 0.000000            0.000000       0.000000
75%                 0.000000            0.000000       1.000000
max                 1.000000            1.000000       1.000000
```

```
drop_list = ['CRASH_DATE_EST_I', 'RD_NO','REPORT_TYPE','DATE_POLICE_NOTIFIED',
             'STREET_DIRECTION', 'BEAT_OF_OCCURRENCE','PHOTOS_TAKEN_I','STATEMENTS_TAKEN_I',
             'DOORING_I','WORK_ZONE_TYPE', 'WORKERS_PRESENT_I','LATITUDE',
             'LONGITUDE','CRASH_RECORD_ID','INJURIES_UNKNOWN','STREET_NO',
             'MOST_SEVERE_INJURY','SEC_CONTRIBUTORY_CAUSE','LOCATION','STREET_NAME']
```

```
50%         0.000000       2.000000        0.000000        0.000000
```

```
existing_drop_list = [col for col in drop_list if col in df.columns]
df.drop(columns=existing_drop_list, inplace=True)
```

```
count           892977.000000               892977.000000
```

```
print("Shape of DataFrame after dropping columns:", df.shape)
df.info()
```

```
Shape of DataFrame after dropping columns: (892977, 32)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 892977 entries, 0 to 892976
Data columns (total 32 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   CRASH_DATE                    892977 non-null  datetime64[ns]
 1   POSTED_SPEED_LIMIT            892977 non-null  int64
 2   TRAFFIC_CONTROL_DEVICE        892977 non-null  object
 3   DEVICE_CONDITION              892977 non-null  object
```

```
 4   WEATHER_CONDITION             892977 non-null  object
 5   LIGHTING_CONDITION            892977 non-null  object
 6   FIRST_CRASH_TYPE              892977 non-null  object
 7   TRAFFICWAY_TYPE               892977 non-null  object
 8   LANE_CNT                      892977 non-null  float64
 9   ALIGNMENT                     892977 non-null  object
 10  ROADWAY_SURFACE_COND          892977 non-null  object
 11  ROAD_DEFECT                   892977 non-null  object
 12  SEVERE                        892977 non-null  int64
 13  INTERSECTION_RELATED_I        892977 non-null  int64
 14  NOT_RIGHT_OF_WAY_I            892977 non-null  int64
 15  HIT_AND_RUN_I                 892977 non-null  int64
 16  DAMAGE                        892977 non-null  object
 17  PRIM_CONTRIBUTORY_CAUSE       892977 non-null  object
 18  WORK_ZONE_I                   892977 non-null  int64
 19  NUM_UNITS                     892977 non-null  int64
 20  INJURIES_TOTAL                892977 non-null  float64
 21  INJURIES_FATAL                892977 non-null  float64
 22  INJURIES_INCAPACITATING       892977 non-null  float64
 23  INJURIES_NON_INCAPACITATING   892977 non-null  float64
 24  INJURIES_REPORTED_NOT_EVIDENT 892977 non-null  float64
 25  INJURIES_NO_INDICATION        892977 non-null  float64
 26  CRASH_HOUR                    892977 non-null  int64
 27  CRASH_DAY_OF_WEEK             892977 non-null  int64
 28  CRASH_MONTH                   892977 non-null  int64
 29  HOUR                          892977 non-null  int32
 30  DAY_OF_WEEK                   892977 non-null  int32
 31  MONTH                         892977 non-null  int32
dtypes: datetime64[ns](1), float64(7), int32(3), int64(10), object(11)
memory usage: 207.8+ MB
```

```
print(df['INJURIES_NO_INDICATION'].value_counts())
df['INJURIES_NO_INDICATION'].values[df['INJURIES_NO_INDICATION'] > 10] =10
print(df['INJURIES_NO_INDICATION'].value_counts())
```

```
INJURIES_NO_INDICATION
2.0     414214
1.0     273234
3.0     112245
4.0      41927
0.0      21091
5.0      17610
6.0       7306
7.0       2845
8.0       1267
10.0       715
9.0        523
Name: count, dtype: int64
INJURIES_NO_INDICATION
2.0     414214
1.0     273234
3.0     112245
4.0      41927
0.0      21091
5.0      17610
6.0       7306
7.0       2845
8.0       1267
10.0       715
9.0        523
Name: count, dtype: int64
```

```
print(df['NUM_UNITS'].value_counts())
df['NUM_UNITS'].values[df['NUM_UNITS'] > 7] = 7
print(df['NUM_UNITS'].value_counts())
```

```
NUM_UNITS
2     781888
3      49261
1      48917
4       9518
5       2324
6        664
7        223
8         99
9         40
10        19
11         8
18         5
12         5
14         2
```

```
16         2
13         1
15         1
Name: count, dtype: int64
NUM_UNITS
2    781888
3     49261
1     48917
4      9518
5      2324
6       664
7       405
Name: count, dtype: int64
```

```
print(df['INJURIES_TOTAL'].value_counts())
df['INJURIES_TOTAL'].values[df['INJURIES_TOTAL'] > 5] = 5
print(df['INJURIES_TOTAL'].value_counts())
```

```
INJURIES_TOTAL
0.0     767610
1.0      94222
2.0      21062
3.0       6406
4.0       2281
5.0        815
6.0        323
7.0        130
8.0         52
9.0         27
10.0        16
11.0         9
15.0         7
12.0         6
21.0         4
13.0         3
17.0         1
14.0         1
19.0         1
16.0         1
Name: count, dtype: int64
INJURIES_TOTAL
0.0    767610
1.0     94222
2.0     21062
3.0      6406
4.0      2281
5.0      1396
Name: count, dtype: int64
```

```
print(df['WEATHER_CONDITION'].value_counts())
```

```
WEATHER_CONDITION
CLEAR                        702900
RAIN                          77492
UNKNOWN                       50779
SNOW                          28402
CLOUDY/OVERCAST               25915
OTHER                          2759
FREEZING RAIN/DRIZZLE          1759
FOG/SMOKE/HAZE                 1350
SLEET/HAIL                     1014
BLOWING SNOW                    444
SEVERE CROSS WIND GATE          156
BLOWING SAND, SOIL, DIRT          7
Name: count, dtype: int64
```

```
df['WEATHER_CONDITION'] = df['WEATHER_CONDITION'].replace({'UNKNOWN':'UNKNOWN/OTHER','OTHER':'UNKNOWN/OTHER',
                                                           'FREEZING RAIN/DRIZZLE': 'SLEET/HAIL',
                                                           'BLOWING SNOW': 'SNOW',
                                                            'SEVERE CROSS WIND GATE': 'CLOUDY/OVERCAST',
                                                           'BLOWING SAND, SOIL, DIRT': 'UNKNOWN/OTHER'})
print(df['WEATHER_CONDITION'].value_counts())
```

```
WEATHER_CONDITION
CLEAR              702900
RAIN                77492
UNKNOWN/OTHER       53545
SNOW                28846
```

```
CLOUDY/OVERCAST     26071
SLEET/HAIL           2773
FOG/SMOKE/HAZE       1350
Name: count, dtype: int64
```

```
print(df['ROADWAY_SURFACE_COND'].value_counts())
```

```
ROADWAY_SURFACE_COND
DRY                661176
WET                116579
UNKNOWN             78850
SNOW OR SLUSH       28131
ICE                  5661
OTHER                2259
SAND, MUD, DIRT       321
Name: count, dtype: int64
```

```
df['ROADWAY_SURFACE_COND'] =df['ROADWAY_SURFACE_COND'].replace({'UNKNOWN':'UNKNOWN/OTHER',
                                                                'OTHER':'UNKNOWN/OTHER',
                                                              'SAND, MUD, DIRT': 'UNKNOWN/OTHER',
                                                                'ICE': 'SNOW OR SLUSH'})
print(df['ROADWAY_SURFACE_COND'].value_counts())
```

```
ROADWAY_SURFACE_COND
DRY              661176
WET              116579
UNKNOWN/OTHER     81430
SNOW OR SLUSH     33792
Name: count, dtype: int64
```

```
print(df['TRAFFIC_CONTROL_DEVICE'].value_counts())
```

```
TRAFFIC_CONTROL_DEVICE
NO CONTROLS                 505786
TRAFFIC SIGNAL              247517
STOP SIGN/FLASHER            88466
UNKNOWN                      37760
OTHER                         6039
YIELD                         1351
LANE USE MARKING              1226
OTHER REG. SIGN               1075
OTHER WARNING SIGN             710
PEDESTRIAN CROSSING SIGN       624
RAILROAD CROSSING GATE         578
FLASHING CONTROL SIGNAL        367
DELINEATORS                    351
SCHOOL ZONE                    350
POLICE/FLAGMAN                 306
OTHER RAILROAD CROSSING        191
RR CROSSING SIGN               190
NO PASSING                      56
BICYCLE CROSSING SIGN           34
Name: count, dtype: int64
```

```
df['TRAFFIC_CONTROL_DEVICE']=df['TRAFFIC_CONTROL_DEVICE'].replace(
                            {'UNKNOWN':'UNKNOWN/OTHER',
                            'OTHER':'UNKNOWN/OTHER',
                            'FLASHING CONTROL SIGNAL':'STOP SIGN/FALSHER',
                              'OTHER RAILROAD CROSSING':'RAILROAD CROSSING GATE'})
print(df['TRAFFIC_CONTROL_DEVICE'].value_counts())
```

```
TRAFFIC_CONTROL_DEVICE
NO CONTROLS                 505786
TRAFFIC SIGNAL              247517
STOP SIGN/FLASHER            88466
UNKNOWN/OTHER                43799
YIELD                         1351
LANE USE MARKING              1226
OTHER REG. SIGN               1075
RAILROAD CROSSING GATE         769
OTHER WARNING SIGN             710
PEDESTRIAN CROSSING SIGN       624
STOP SIGN/FALSHER              367
DELINEATORS                    351
SCHOOL ZONE                    350
POLICE/FLAGMAN                 306
RR CROSSING SIGN               190
NO PASSING                      56
BICYCLE CROSSING SIGN           34
```

```
Name: count, dtype: int64
```

```
print(df['ROAD_DEFECT'].value_counts())
```

```
ROAD_DEFECT
NO DEFECTS           712018
UNKNOWN              163881
RUT, HOLES             6337
OTHER                  4841
WORN SURFACE           3701
SHOULDER DEFECT        1543
DEBRIS ON ROADWAY       656
Name: count, dtype: int64
```

```
df['ROAD_DEFECT'] = df['ROAD_DEFECT'].replace({'UNKNOWN':'UNKNOWN/OTHER',
                            'OTHER':'UNKNOWN/OTHER'})
```

```
print(df['ROAD_DEFECT'].value_counts())
```

```
ROAD_DEFECT
NO DEFECTS           712018
UNKNOWN/OTHER        168722
RUT, HOLES             6337
WORN SURFACE           3701
SHOULDER DEFECT        1543
DEBRIS ON ROADWAY       656
Name: count, dtype: int64
```

```
print(df['TRAFFICWAY_TYPE'].value_counts())
```

```
TRAFFICWAY_TYPE
NOT DIVIDED                         384844
DIVIDED - W/MEDIAN (NOT RAISED)     141251
ONE-WAY                             113120
FOUR WAY                             61468
PARKING LOT                          60487
DIVIDED - W/MEDIAN BARRIER           50543
OTHER                                24180
ALLEY                                14682
T-INTERSECTION                       12195
UNKNOWN                              10506
CENTER TURN LANE                      6325
DRIVEWAY                              2872
RAMP                                  2800
UNKNOWN INTERSECTION TYPE             2731
FIVE POINT, OR MORE                   1354
Y-INTERSECTION                        1330
TRAFFIC ROUTE                         1129
NOT REPORTED                           671
ROUNDABOUT                             304
L-INTERSECTION                         185
Name: count, dtype: int64
```

```
df['TRAFFICWAY_TYPE'] = df['TRAFFICWAY_TYPE'].replace({'T-INTERSECTION':'INTERSECTION', 'UNKNOWN INTERSECTION TYPE':'INTERSECTION',
                                                       'Y-INTERSECTION':'INTERSECTION','L-INTERSECTION':'INTERSECTION',
                                                        'FIVE POINT, OR MORE': 'INTERSECTION', 'FOUR WAY':'INTERSECTION',
                                                        'ROUNDABOUT': 'INTERSECTION', 'OTHER': 'UNKNOWN/OTHER',
                                                       'UNKNOWN':'UNKNOWN/OTHER','NOT REPORTED':'UNKNOWN/OTHER',
                                                       'TRAFFIC ROUTE':'UNKNOWN/OTHER'})
```

```
print(df.TRAFFICWAY_TYPE.value_counts())
```

```
TRAFFICWAY_TYPE
NOT DIVIDED                         384844
DIVIDED - W/MEDIAN (NOT RAISED)     141251
ONE-WAY                             113120
INTERSECTION                         79567
PARKING LOT                          60487
DIVIDED - W/MEDIAN BARRIER           50543
UNKNOWN/OTHER                        36486
ALLEY                                14682
CENTER TURN LANE                      6325
DRIVEWAY                              2872
RAMP                                  2800
Name: count, dtype: int64
```

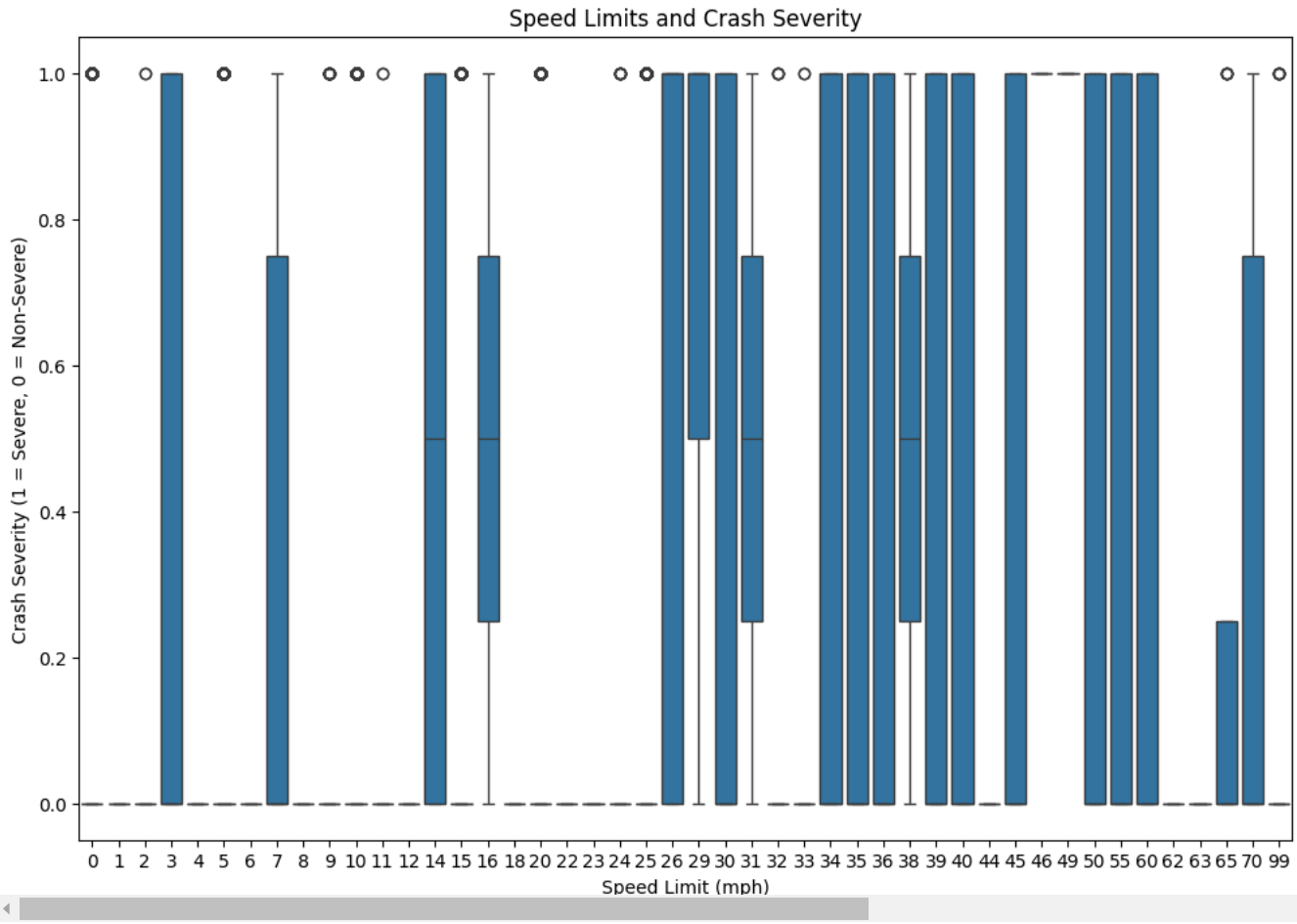
```
month_bins = [1,4,7,10,13]
label=('Winter','Spring','Summer','Fall')
month_binned = pd.cut(df['CRASH_MONTH'], month_bins, labels= label)
month_binned= month_binned.cat.as_unordered()
df['SEASON']= month_binned
df['SEASON'].value_counts()
```

SEASON	count
Summer	249700
Spring	233532
Winter	199511
Fall	144167

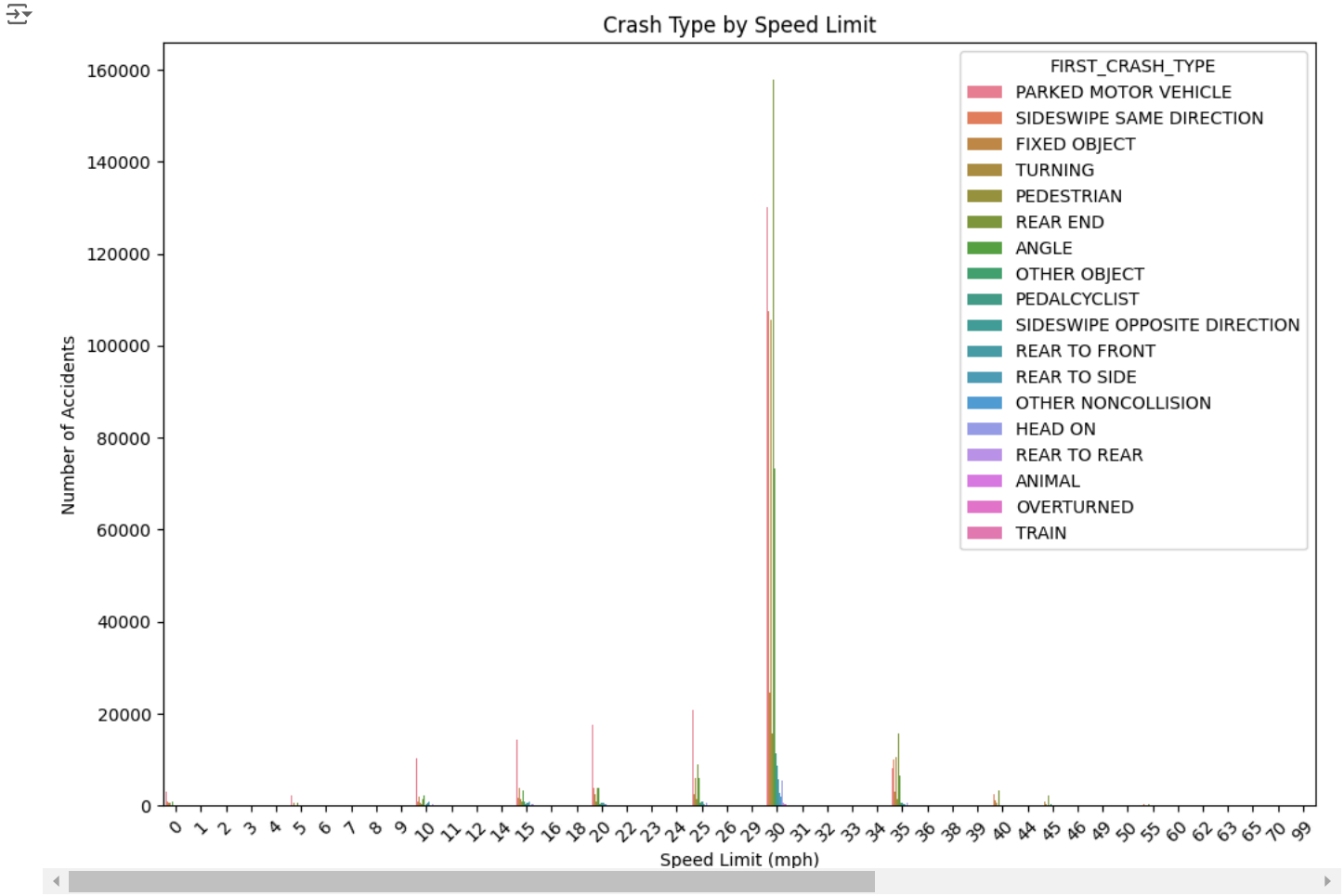
dtype: int64

```
# Check if 'SPEED_LIMIT' and 'FIRST_CRASH_TYPE' columns are available
if 'SPEED_LIMIT' in df.columns and 'FIRST_CRASH_TYPE' in df.columns:
    print(df['SPEED_LIMIT'].value_counts())
    print(df['FIRST_CRASH_TYPE'].value_counts())
```

```
# Boxplot for posted_speed limit vs CRASH_TYPE
plt.figure(figsize=(12, 8))
# Corrected column name: remove extra spaces
sns.boxplot(x='POSTED_SPEED_LIMIT', y='SEVERE', data=df)
plt.title('Speed Limits and Crash Severity')
plt.xlabel('Speed Limit (mph)')
plt.ylabel('Crash Severity (1 = Severe, 0 = Non-Severe)')
plt.show()
```



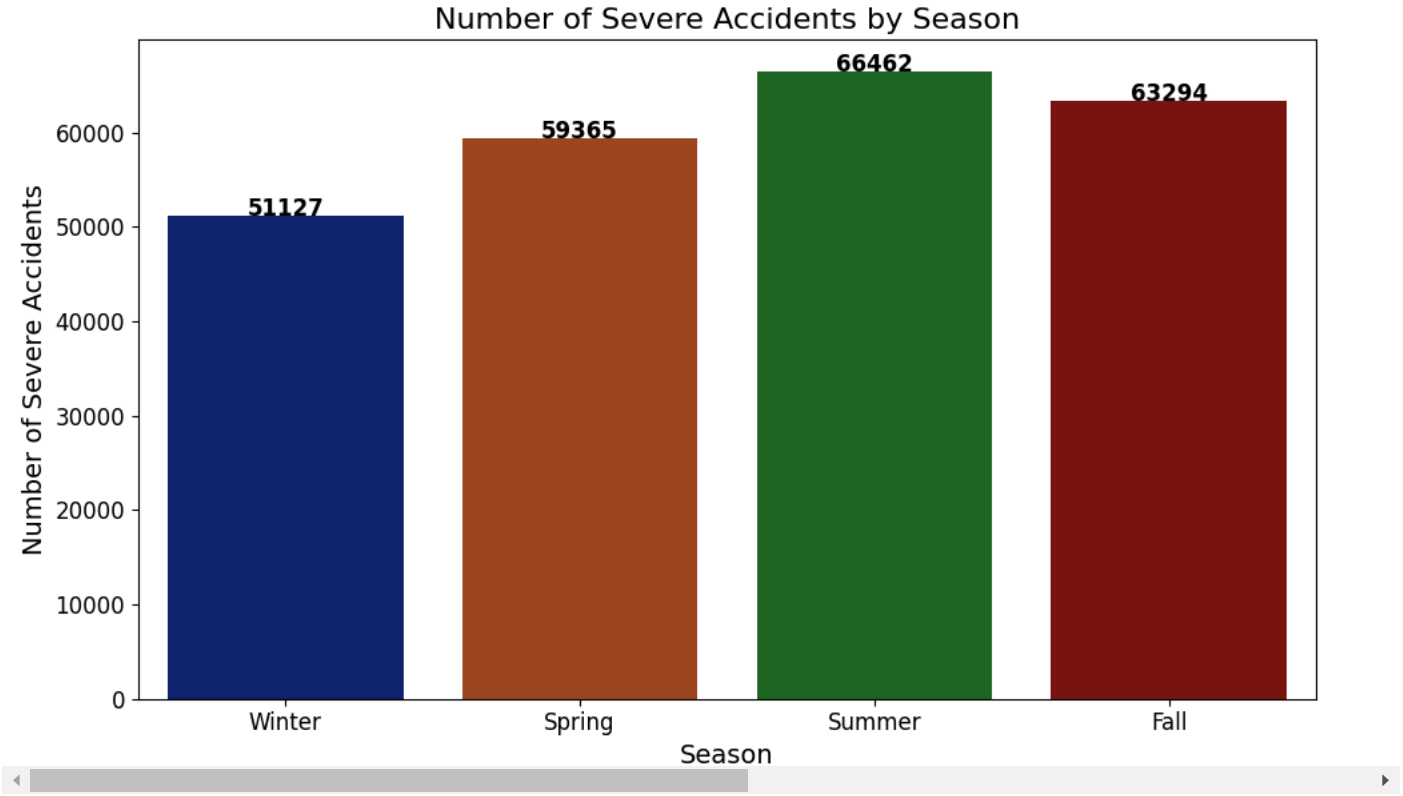
```
 # Countplot for SPEED_LIMIT vs CRASH_TYPE
plt.figure(figsize=(12, 8))
sns.countplot(x='POSTED_SPEED_LIMIT', hue='FIRST_CRASH_TYPE', data=df)
plt.title('Crash Type by Speed Limit')
plt.xlabel('Speed Limit (mph)')
plt.ylabel('Number of Accidents')
plt.xticks(rotation=45)
plt.show()
```




```
# Bin months into seasons
month_bins = [1, 4, 7, 10, 13]  # Define bins for seasons
labels = ['Winter', 'Spring', 'Summer', 'Fall']  # Define season labels
month_binned = pd.cut(df['CRASH_MONTH'], bins=month_bins, labels=labels, right=False)
df['SEASON'] = month_binned  # Assign the season to a new column

# Filter data for severe accidents
filtered_df = df[df['SEVERE'] == 1]  # Include only severe accidents

# Count the number of accidents in each season
season_counts = filtered_df['SEASON'].value_counts()

# Sort the seasons in order
season_counts = season_counts.reindex(labels)

# Plot the number of accidents by season
plt.figure(figsize=(10, 6))
barplot = sns.barplot(x=season_counts.index, y=season_counts.values, palette='dark')

# Add labels on top of the bars
for i, value in enumerate(season_counts.values):
    plt.text(i, value + 2, str(value), ha='center', fontsize=12, fontweight='bold')  # Adjust `value + 2` for padding

# Add labels and title
plt.title('Number of Severe Accidents by Season', fontsize=16)
plt.xlabel('Season', fontsize=14)
plt.ylabel('Number of Severe Accidents', fontsize=14)
```

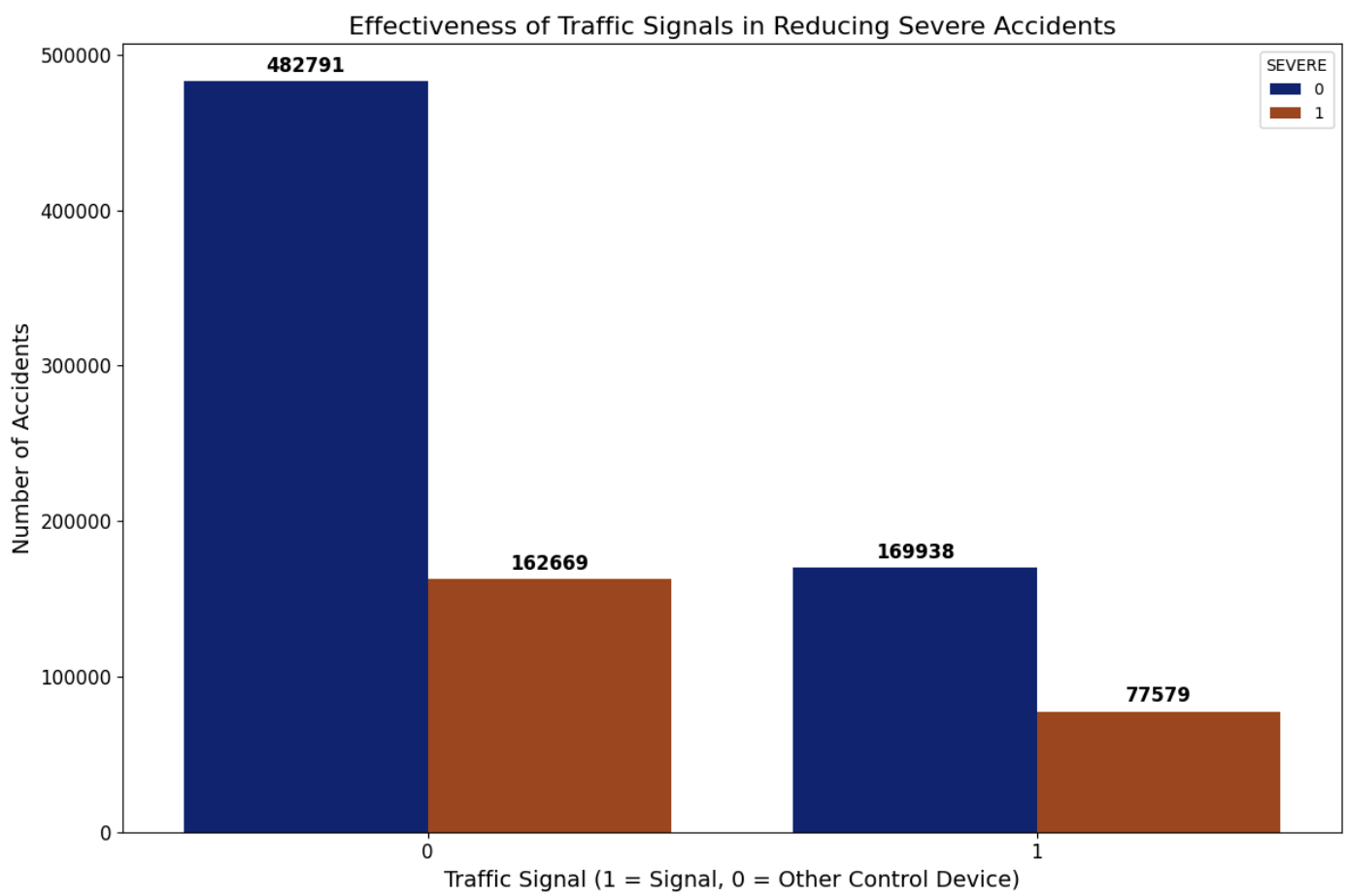
```
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

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

```
<ipython-input-60-2aab1fc7eb6d>:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  barplot = sns.barplot(x=season_counts.index, y=season_counts.values, palette='dark')
```




```
# Checking for the presence of traffic control devices
if 'TRAFFIC_CONTROL_DEVICE' in df.columns:
    # Compare the effectiveness of traffic signals
    df['IS_TRAFFIC_SIGNAL'] = df['TRAFFIC_CONTROL_DEVICE'].apply(lambda x: 1 if 'SIGNAL' in str(x).upper() else 0)
```

```
# Plot the impact of traffic signals on accident severity with a dark color palette
plt.figure(figsize=(12, 8))
countplot = sns.countplot(x='IS_TRAFFIC_SIGNAL', hue='SEVERE', data=df, palette='dark')  # Dark color palette

# Add labels on top of the bars
for container in countplot.containers:
    countplot.bar_label(container, label_type='edge', fontsize=12, fontweight='bold', padding=3)

# Add title and axis labels
plt.title('Effectiveness of Traffic Signals in Reducing Severe Accidents', fontsize=16)
plt.xlabel('Traffic Signal (1 = Signal, 0 = Other Control Device)', fontsize=14)
plt.ylabel('Number of Accidents', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

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

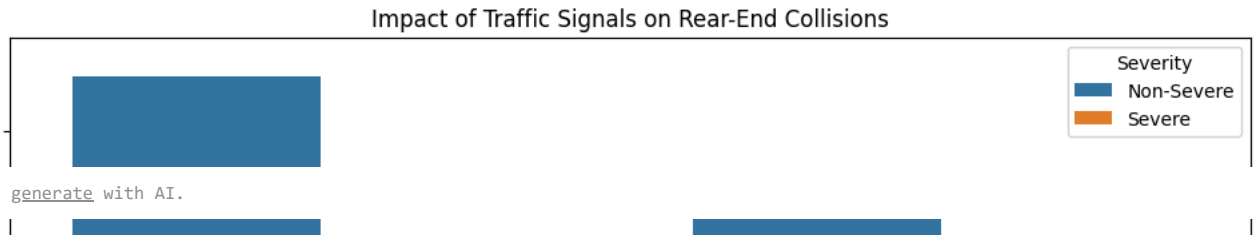



```
#
rear_end_collisions = df[df['FIRST_CRASH_TYPE'] == 'REAR END ']


rear_end_count = df[df['FIRST_CRASH_TYPE'] == 'REAR END'].shape[0]

print(f"Number of rear-end collisions in the dataset: {rear_end_count}")
```

```
Number of rear-end collisions in the dataset: 197598
```

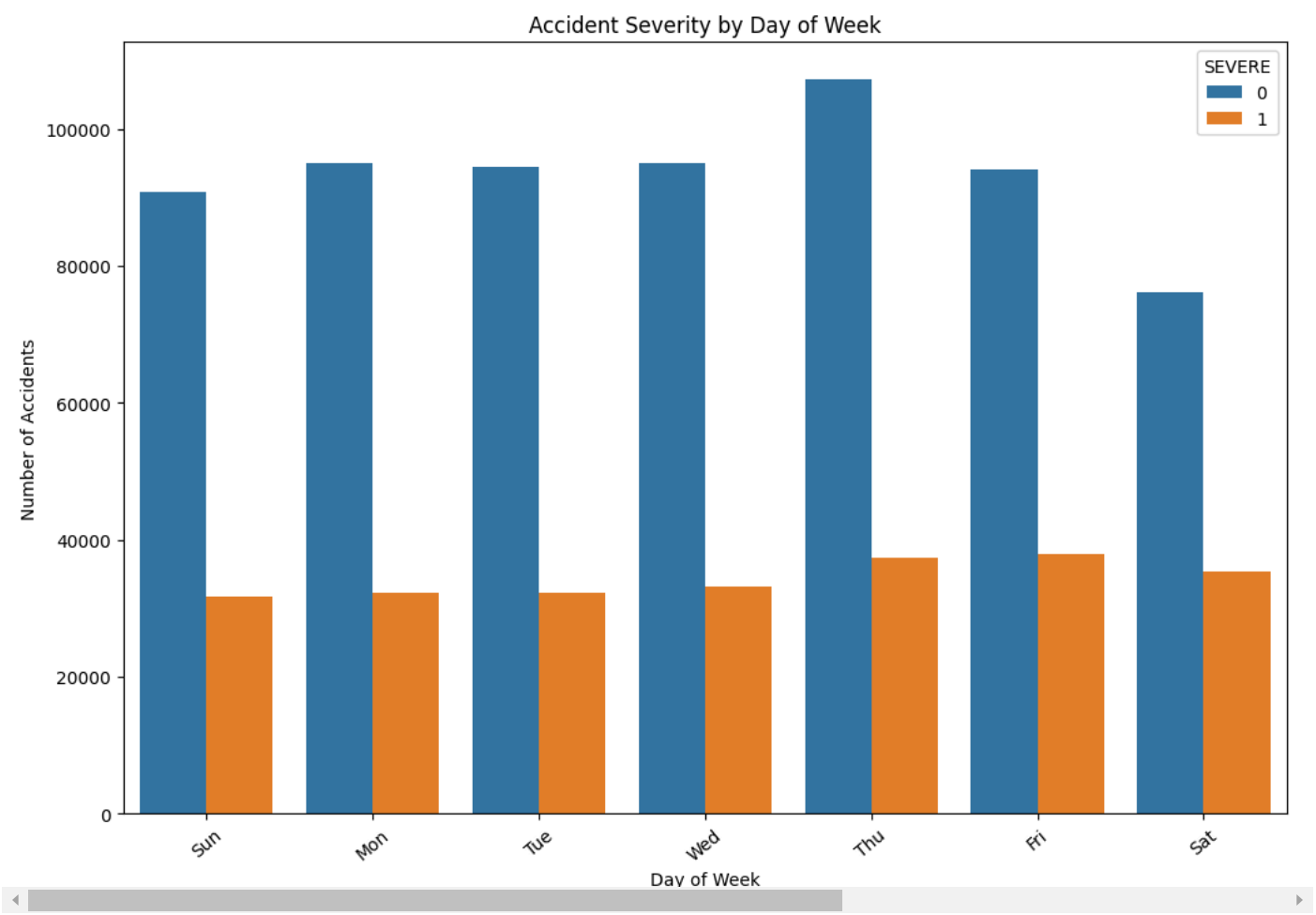
```
# Filter the dataset for rear-end collisions
rear_end_collisions = df[df['FIRST_CRASH_TYPE'] == 'REAR END']

# Display the number of rear-end collisions based on the presence of traffic signals
print("Distribution of rear-end collisions by traffic control device:")
signal_distribution = rear_end_collisions['IS_TRAFFIC_SIGNAL'].value_counts()
print(signal_distribution)

# Plotting the impact of traffic signals on rear-end collisions severity
plt.figure(figsize=(12, 8))
sns.countplot(x='IS_TRAFFIC_SIGNAL', hue='SEVERE', data=rear_end_collisions)
plt.title('Impact of Traffic Signals on Rear-End Collisions')
plt.xlabel('Traffic Signal (1 = Signal, 0 = Other Control Device)')
plt.ylabel('Number of Rear-End Collisions')
plt.legend(title='Severity', labels=['Non-Severe', 'Severe'])
plt.show()
```

```
Distribution of rear-end collisions by traffic control device:
IS_TRAFFIC_SIGNAL
0    108229
1     89369
Name: count, dtype: int64
```

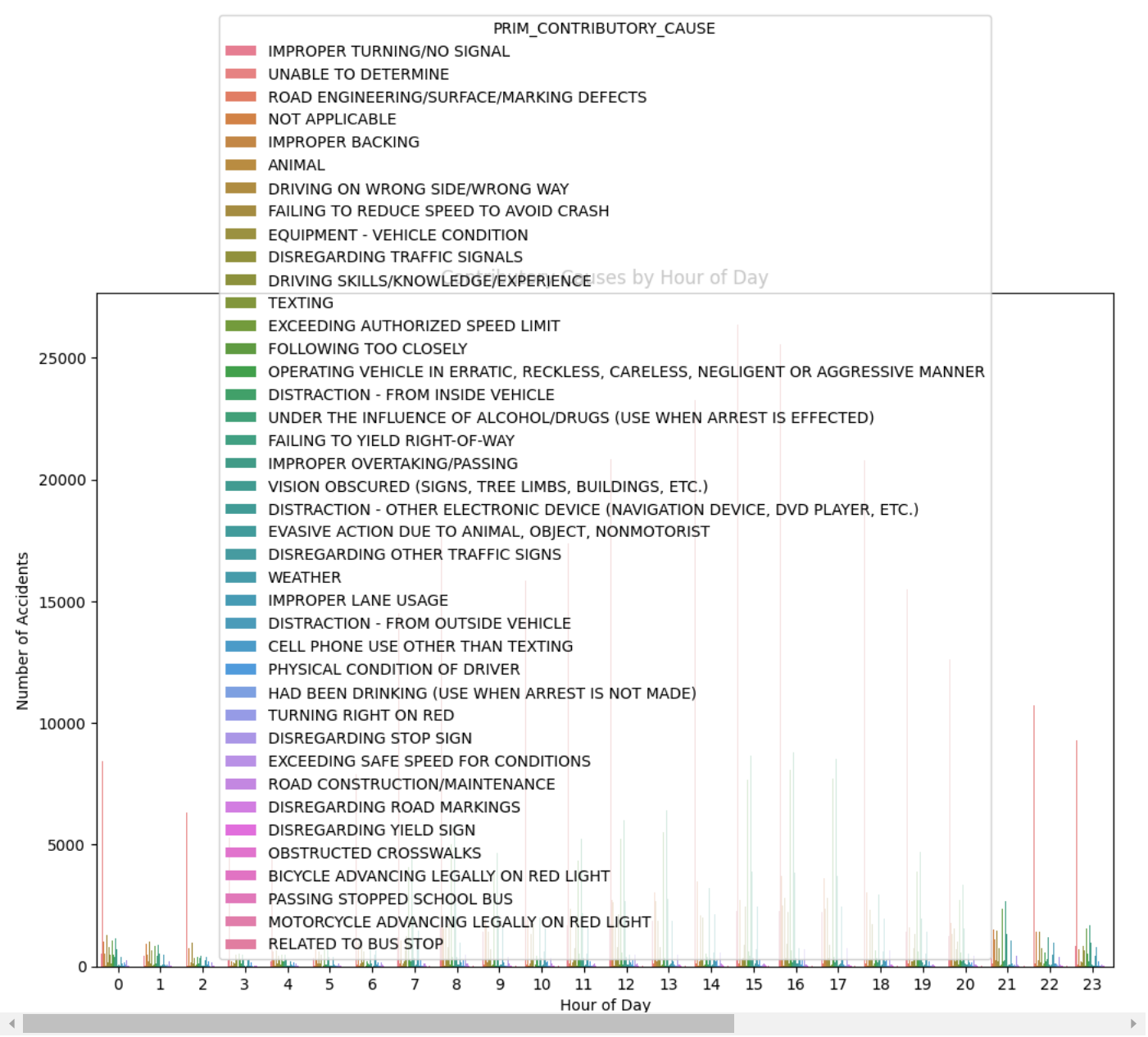


Start coding or generate with AI.

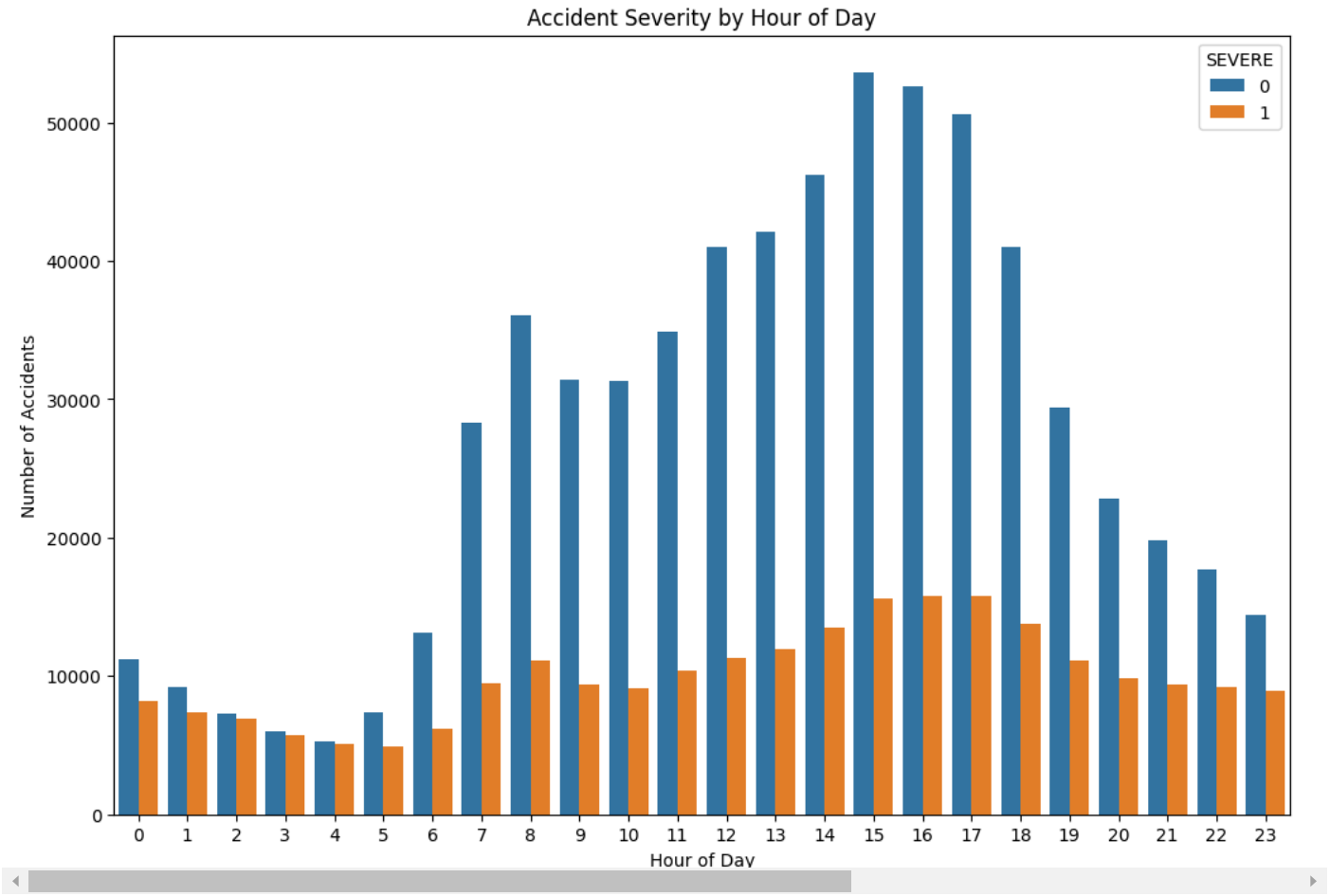
```
# Visualize crash severity by day of the week
plt.figure(figsize=(12, 8))
sns.countplot(x='DAY_OF_WEEK', hue='SEVERE', data=df)
plt.title('Accident Severity by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Number of Accidents')
plt.xticks(np.arange(7), ('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'), rotation=40)
plt.show()
```



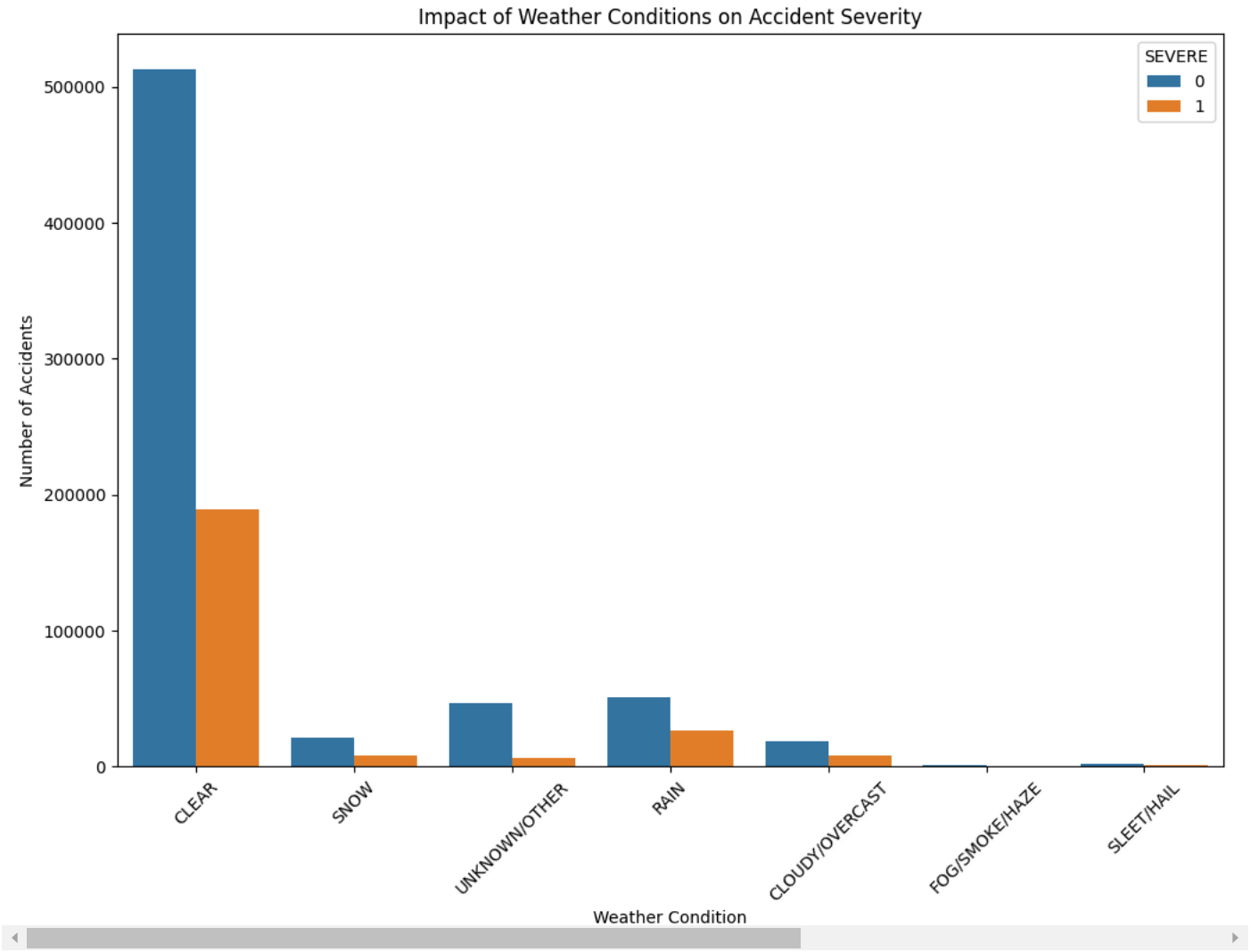

```
# Visualize contributory causes by time of day
plt.figure(figsize=(12, 8))
sns.countplot(x='HOUR', hue='PRIM_CONTRIBUTORY_CAUSE', data=df)
plt.title('Contributory Causes by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Accidents')
plt.show()
```



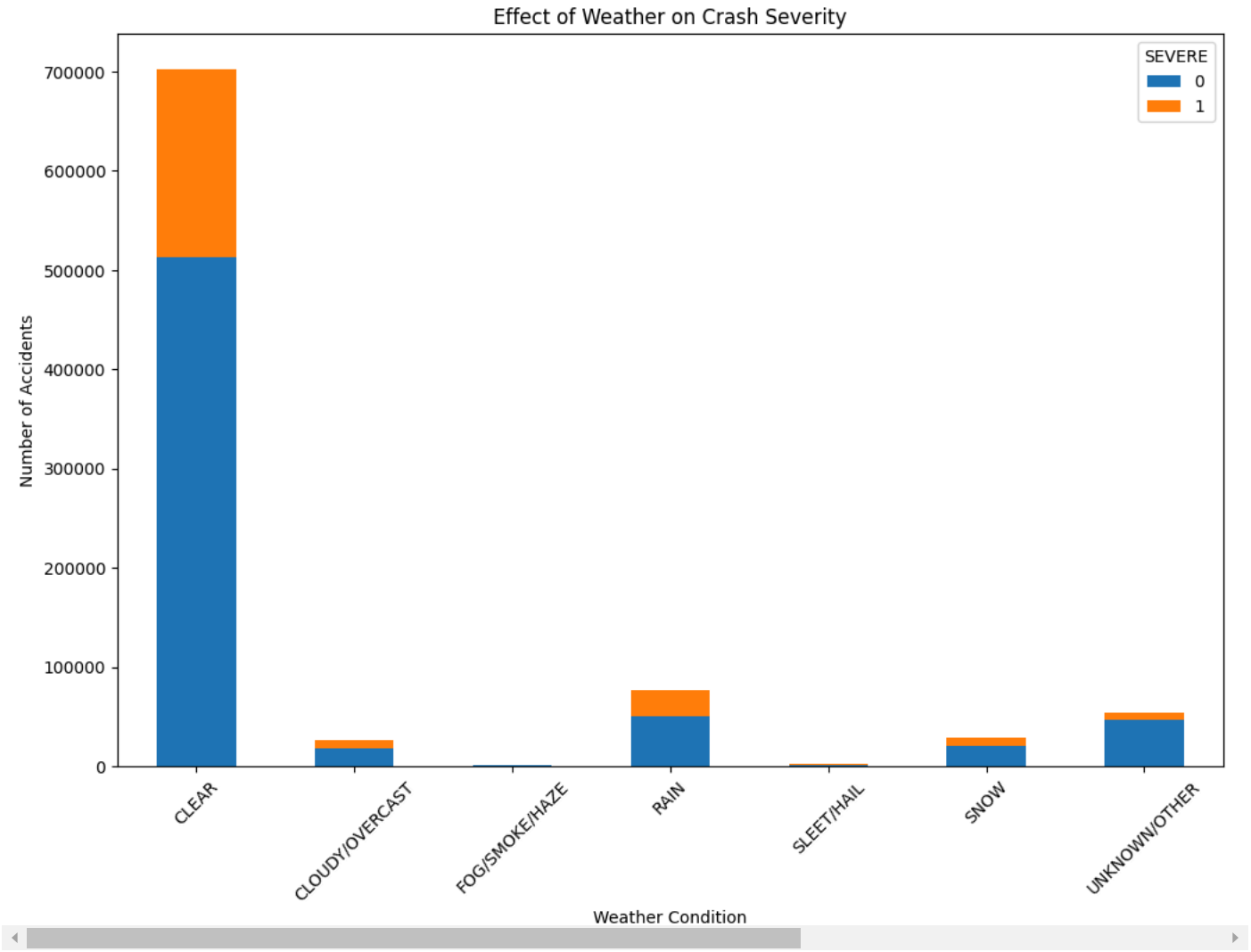

```
# Severity by Hour of Day
plt.figure(figsize=(12, 8))
sns.countplot(x='HOUR', hue='SEVERE', data=df)
plt.title('Accident Severity by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Accidents')
plt.show()
```




```
# Weather condition vs crash severity
plt.figure(figsize=(12, 8))
sns.countplot(x='WEATHER_CONDITION', hue='SEVERE', data=df)
plt.title('Impact of Weather Conditions on Accident Severity')
plt.xlabel('Weather Condition')
plt.ylabel('Number of Accidents')
plt.xticks(rotation=45)
plt.show()
```




```
# Compare frequency of accidents by weather condition
weather_severity = df.groupby(['WEATHER_CONDITION', 'SEVERE']).size().unstack().fillna(0)
weather_severity.plot(kind='bar', stacked=True, figsize=(12, 8))
plt.title('Effect of Weather on Crash Severity')
plt.xlabel('Weather Condition')
plt.ylabel('Number of Accidents')
plt.xticks(rotation=45)
plt.show()
```




```
plt.figure(figsize=(10,8))
sns.countplot(x= "CRASH_DAY_OF_WEEK" , data=df,
              hue="SEVERE")
plt.title('Accidents by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Number of Accidents')
plt.xticks(np.arange(7), ('Sun','Mon','Tue','Wed','Thr','Fri','Sat'),
           rotation=40)
```