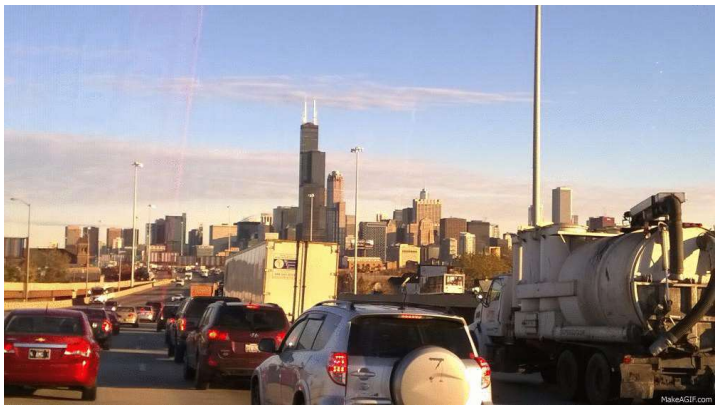


Chicago Road Safety Assessment: Predictive Modeling for Injury Severity

Authors: DSPT-05 Group 17

1. Mark Kuria
2. Rhoda Musyok
3. Mlati Ochieng'
4. Dianna Wangura
5. Caroline Mbugua
6. Eunita Nyengo

Business Understanding

This project aims to develop a predictive model for injury severity in car crashes within the jurisdiction of the City of Chicago. Sponsored by the Chicago City administration, this assessment serves as a strategic initiative to provide meaningful insights and guide resource allocation while optimizing safety measures, ultimately reducing fatalities resulting from traffic crashes

---

Data Understanding

The data source for this assessment is the Chicago Car Crashes data from the Chicago Data Portal.

Datasets Used:

1. Traffic Crashes - Crashes This dataset contains information about traffic crashes on city streets within the jurisdiction of the Chicago Police Department (CPD). Records are added when crash reports are finalized or amended in the electronic crash reporting system (E-Crash). Data includes parameters such as injuries, street and weather conditions, posted speed limits.
2. Traffic Crashes - People This dataset provides details about individuals involved in traffic crashes, including occupants of vehicles, pedestrians, cyclists and others. Injury reports are recorded for each person and the dataset can be linked to the Crash dataset using the "CRASH_RECORD_ID" field, maintaining appropriate relationships.

In [1]:
```
# import relevant libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from lightgbm import LGBMClassifier


import warnings
warnings.filterwarnings(action='ignore')
```

In [2]:
```
# Loading the datasets

df_crashes = pd.read_csv('Data\Traffic_Crashes_-_Crashes_20240208.csv', low_memory=False)
df_people = pd.read_csv('Data\Traffic_Crashes_-_People_20240208.csv', low_memory=False)
```

In [3]:
```
df_crashes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 804163 entries, 0 to 804162
Data columns (total 48 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   CRASH_RECORD_ID                 804163 non-null  object
 1   CRASH_DATE_EST_I                60229 non-null   object
 2   CRASH_DATE                      804163 non-null  object
 3   POSTED_SPEED_LIMIT              804163 non-null  int64
 4   TRAFFIC_CONTROL_DEVICE          804163 non-null  object
 5   DEVICE_CONDITION                804163 non-null  object
 6   WEATHER_CONDITION               804163 non-null  object
 7   LIGHTING_CONDITION              804163 non-null  object
 8   FIRST_CRASH_TYPE                804163 non-null  object
 9   TRAFFICWAY_TYPE                 804163 non-null  object
 10  LANE_CNT                        199007 non-null  float64
 11  ALIGNMENT                       804163 non-null  object
 12  ROADWAY_SURFACE_COND            804163 non-null  object
 13  ROAD_DEFECT                     804163 non-null  object
 14  REPORT_TYPE                     780500 non-null  object
 15  CRASH_TYPE                      804163 non-null  object
 16  INTERSECTION_RELATED_I          184348 non-null  object
 17  NOT_RIGHT_OF_WAY_I              37115 non-null   object
 18  HIT_AND_RUN_I                   251501 non-null  object
 19  DAMAGE                          804163 non-null  object
 20  DATE_POLICE_NOTIFIED            804163 non-null  object
 21  PRIM_CONTRIBUTORY_CAUSE         804163 non-null  object
 22  SEC_CONTRIBUTORY_CAUSE          804163 non-null  object
 23  STREET_NO                       804163 non-null  int64
 24  STREET_DIRECTION                804159 non-null  object
 25  STREET_NAME                     804162 non-null  object
 26  BEAT_OF_OCCURRENCE              804158 non-null  float64
 27  PHOTOS_TAKEN_I                  10504 non-null   object
 28  STATEMENTS_TAKEN_I              17803 non-null   object
 29  DOORING_I                       2474 non-null    object
 30  WORK_ZONE_I                     4628 non-null    object
 31  WORK_ZONE_TYPE                  3591 non-null    object
 32  WORKERS_PRESENT_I               1184 non-null    object
 33  NUM_UNITS                       804163 non-null  int64
 34  MOST_SEVERE_INJURY              802392 non-null  object
 35  INJURIES_TOTAL                  802404 non-null  float64
 36  INJURIES_FATAL                  802404 non-null  float64
 37  INJURIES_INCAPACITATING         802404 non-null  float64
 38  INJURIES_NON_INCAPACITATING     802404 non-null  float64
 39  INJURIES_REPORTED_NOT_EVIDENT   802404 non-null  float64
 40  INJURIES_NO_INDICATION          802404 non-null  float64
 41  INJURIES_UNKNOWN                802404 non-null  float64
 42  CRASH_HOUR                      804163 non-null  int64
 43  CRASH_DAY_OF_WEEK               804163 non-null  int64
 44  CRASH_MONTH                     804163 non-null  int64
 45  LATITUDE                        798674 non-null  float64
 46  LONGITUDE                       798674 non-null  float64
 47  LOCATION                        798674 non-null  object
dtypes: float64(11), int64(6), object(31)
memory usage: 294.5+ MB
```

In [4]:
```
df_people.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1765655 entries, 0 to 1765654
Data columns (total 29 columns):
 #   Column                 Dtype
---  ------                 -----
 0   PERSON_ID              object
 1   PERSON_TYPE            object
 2   CRASH_RECORD_ID        object
 3   VEHICLE_ID             float64
 4   CRASH_DATE             object
 5   SEAT_NO                float64
 6   CITY                   object
 7   STATE                  object
 8   ZIPCODE                object
 9   SEX                    object
 10  AGE                    float64
 11  DRIVERS_LICENSE_STATE  object
 12  DRIVERS_LICENSE_CLASS  object
 13  SAFETY_EQUIPMENT       object
 14  AIRBAG_DEPLOYED        object
 15  EJECTION               object
 16  INJURY_CLASSIFICATION  object
 17  HOSPITAL               object
 18  EMS_AGENCY             object
 19  EMS_RUN_NO             object
 20  DRIVER_ACTION          object
 21  DRIVER_VISION          object
 22  PHYSICAL_CONDITION     object
 23  PEDPEDAL_ACTION        object
 24  PEDPEDAL_VISIBILITY    object
 25  PEDPEDAL_LOCATION      object
 26  BAC_RESULT             object
 27  BAC_RESULT VALUE       float64
 28  CELL_PHONE_USE         object
dtypes: float64(4), object(25)
memory usage: 390.7+ MB
```

Comment: For the purpose of this project:

1. Traffic_Crashes_-_Crashes will be our main dataset
2. We focus on data recorded for the last three years - 2021, 2022 and 2023
3. We merge the INJURY_CLASSIFICATION column from Traffic_Crashes_-_People. This will be our target column

Data Preprocessing

In [5]:
```
# extract CRASH_YEAR from the CRASH_DATE merged_crashes['CRASH_YEAR']
df_crashes['CRASH_YEAR'] = pd.to_datetime(df_crashes['CRASH_DATE']).dt.year
```

In [6]:
```
# filtering the dataFrame to retain only the years 2021, 2022, and 2023 in the 'CRASH_YEAR' column
df_crashes = df_crashes[df_crashes['CRASH_YEAR'].isin([2021, 2022, 2023])]
```

In [7]:
```
df_crashes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 327869 entries, 0 to 804156
Data columns (total 49 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   CRASH_RECORD_ID                327869 non-null  object
 1   CRASH_DATE_EST_I               24763 non-null   object
 2   CRASH_DATE                     327869 non-null  object
 3   POSTED_SPEED_LIMIT             327869 non-null  int64
 4   TRAFFIC_CONTROL_DEVICE         327869 non-null  object
 5   DEVICE_CONDITION               327869 non-null  object
 6   WEATHER_CONDITION              327869 non-null  object
 7   LIGHTING_CONDITION             327869 non-null  object
 8   FIRST_CRASH_TYPE               327869 non-null  object
 9   TRAFFICWAY_TYPE                327869 non-null  object
 10  LANE_CNT                       41 non-null      float64
 11  ALIGNMENT                      327869 non-null  object
 12  ROADWAY_SURFACE_COND           327869 non-null  object
 13  ROAD_DEFECT                    327869 non-null  object
 14  REPORT_TYPE                    315922 non-null  object
```

In [8]:
```
# adding the INJURY_CLASSIFICATION column from df_people to df_crashes and renaming the new dataframe to merged_df
merged_df = pd.merge(df_crashes, df_people[['CRASH_RECORD_ID', 'INJURY_CLASSIFICATION']], on='CRASH_RECORD_ID', how='left')
```

In [9]:
```
merged_df.columns
```

Out[9]:
```
Index(['CRASH_RECORD_ID', 'CRASH_DATE_EST_I', 'CRASH_DATE',
       'POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION',
       'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE',
       'TRAFFICWAY_TYPE', 'LANE_CNT', 'ALIGNMENT', 'ROADWAY_SURFACE_COND',
       'ROAD_DEFECT', 'REPORT_TYPE', 'CRASH_TYPE', 'INTERSECTION_RELATED_I',
       'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN_I', 'DAMAGE', 'DATE_POLICE_NOTIFIED',
       'PRIM_CONTRIBUTORY_CAUSE', 'SEC_CONTRIBUTORY_CAUSE', 'STREET_NO',
       'STREET_DIRECTION', 'STREET_NAME', 'BEAT_OF_OCCURRENCE',
       'PHOTOS_TAKEN_I', 'STATEMENTS_TAKEN_I', 'DOORING_I', 'WORK_ZONE_I',
       'WORK_ZONE_TYPE', 'WORKERS_PRESENT_I', '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', 'CRASH_YEAR',
       'INJURY_CLASSIFICATION'],
      dtype='object')
```

In [10]:
```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 713502 entries, 0 to 713501
Data columns (total 50 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   CRASH_RECORD_ID                713502 non-null  object
 1   CRASH_DATE_EST_I               40427 non-null   object
 2   CRASH_DATE                     713502 non-null  object
 3   POSTED_SPEED_LIMIT             713502 non-null  int64
 4   TRAFFIC_CONTROL_DEVICE         713502 non-null  object
 5   DEVICE_CONDITION               713502 non-null  object
 6   WEATHER_CONDITION              713502 non-null  object
 7   LIGHTING_CONDITION             713502 non-null  object
 8   FIRST_CRASH_TYPE               713502 non-null  object
 9   TRAFFICWAY_TYPE                713502 non-null  object
 10  LANE_CNT                       98 non-null      float64
 11  ALIGNMENT                      713502 non-null  object
 12  ROADWAY_SURFACE_COND           713502 non-null  object
 13  ROAD_DEFECT                    713502 non-null  object
 14  REPORT_TYPE                    691811 non-null  object
```

Comment: From the output above, the entries in our dataset have increased. This could be because the CRASH_RECORD_ID column in df_people has duplicates. To review this, we check for duplicates

In [11]:
```
# Checking for duplicates in the CRASH_RECORD_ID column
duplicates = merged_df.duplicated(subset=['CRASH_RECORD_ID'], keep=False)

# Count the number of duplicates
num_duplicates = duplicates.sum()

if num_duplicates > 0:
    print(f'There are {num_duplicates} duplicates in the CRASH_RECORD_ID column.')
else:
    print('There are no duplicates in the CRASH_RECORD_ID column.')
```

```
There are 630115 duplicates in the CRASH_RECORD_ID column.
```

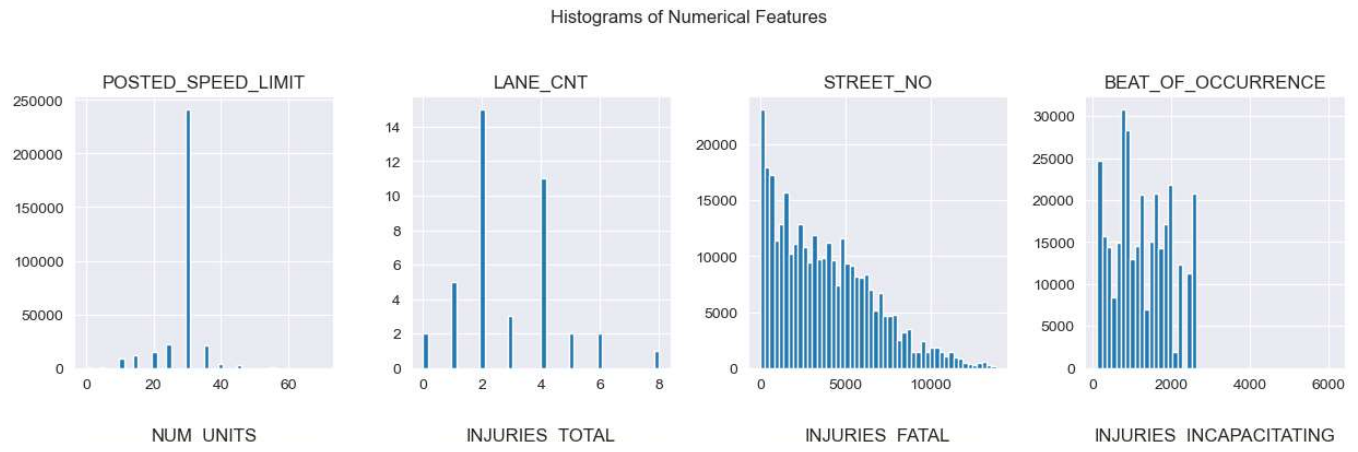
Comment: This could be as a result of having duplicate 'CRASH_RECORD_ID' entries in df_people For this project, we will drop these duplicates

In [12]:
```
# Dropping these duplicates in the CRASH_RECORD_ID column
merged_df.drop_duplicates(subset=['CRASH_RECORD_ID'], keep='first', inplace=True)
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 327869 entries, 0 to 713499
Data columns (total 50 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   CRASH_RECORD_ID                 327869 non-null  object
 1   CRASH_DATE_EST_I                24763 non-null   object
 2   CRASH_DATE                      327869 non-null  object
 3   POSTED_SPEED_LIMIT              327869 non-null  int64
 4   TRAFFIC_CONTROL_DEVICE          327869 non-null  object
 5   DEVICE_CONDITION                327869 non-null  object
 6   WEATHER_CONDITION               327869 non-null  object
 7   LIGHTING_CONDITION              327869 non-null  object
 8   FIRST_CRASH_TYPE                327869 non-null  object
 9   TRAFFICWAY_TYPE                 327869 non-null  object
 10  LANE_CNT                        41 non-null      float64
 11  ALIGNMENT                       327869 non-null  object
 12  ROADWAY_SURFACE_COND            327869 non-null  object
 13  ROAD_DEFECT                     327869 non-null  object
 14  REPORT_TYPE                     315922 non-null  object
```

Exploratory Data Analysis

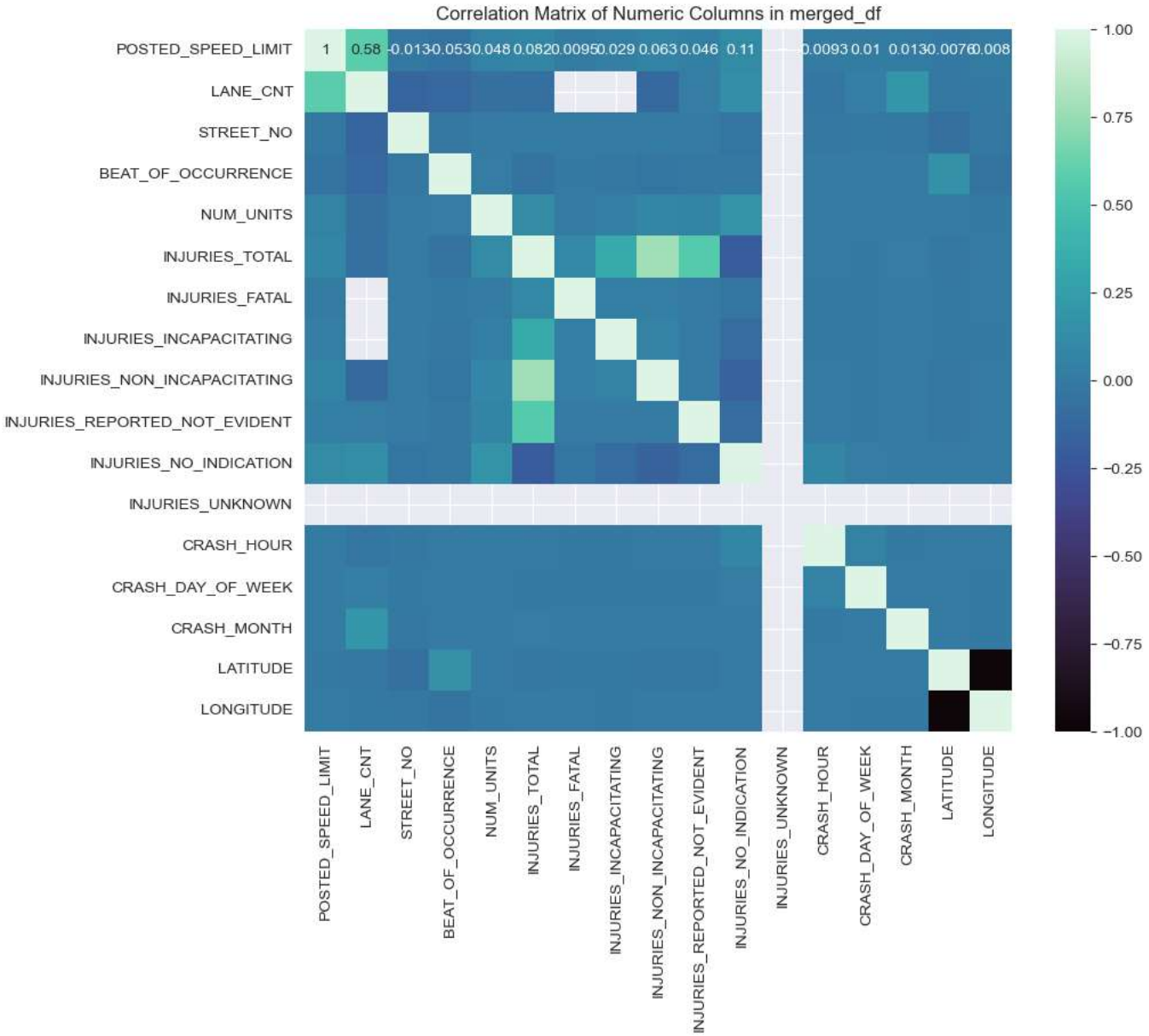
In [13]:
```
# Using Histograms to check the distribution of the numerical features
merged_df.hist(figsize=(15, 20), bins=50)
plt.suptitle('Histograms of Numerical Features', y=0.92)
plt.show()
```



In [ ]:

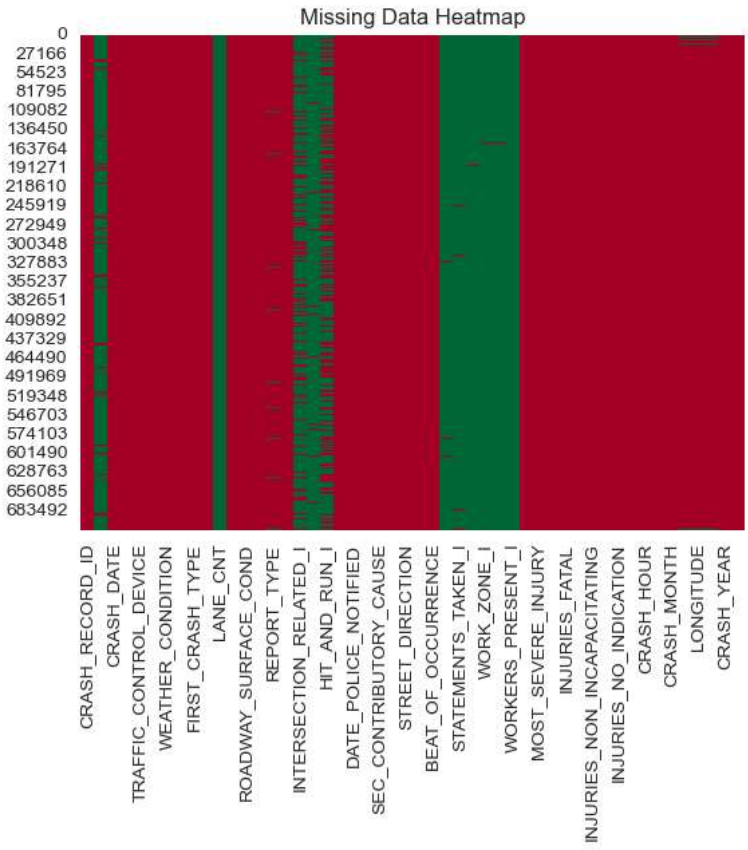
In [17]:

```
# Select numeric columns
numeric_df = merged_df.select_dtypes(include=['float64', 'int64'])

# Compute the correlation matrix
corr_matrix = numeric_df.corr()

# Generate a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, vmin=-1.0, cmap='mako')
plt.title('Correlation Matrix of Numeric Columns in merged_df')
plt.show()
```



Missing Values

In [18]:
```
# Creating a heatmap of missing values
sns.heatmap(merged_df.isnull(), cbar=False, cmap='RdYlGn')
plt.title('Missing Data Heatmap')
plt.show()
```



In [19]:
```
# getting the percentage of missing values for each column
null_percentage = merged_df.isna().mean()*100
null_percentage_sorted = null_percentage.sort_values(ascending=False)
null_percentage_sorted
```

Out[19]:
```
LANE_CNT                        99.987495
WORKERS_PRESENT_I               99.864275
DOORING_I                       99.715435
WORK_ZONE_TYPE                  99.651080
WORK_ZONE_I                     99.534265
PHOTOS_TAKEN_I                  98.634516
STATEMENTS_TAKEN_I              97.507236
NOT_RIGHT_OF_WAY_I              95.476852
CRASH_DATE_EST_I                92.447288
INTERSECTION_RELATED_I          76.601631
HIT_AND_RUN_I                   65.706425
REPORT_TYPE                      3.643833
LOCATION                         0.861930
LONGITUDE                        0.861930
LATITUDE                         0.861930
INJURY_CLASSIFICATION            0.253150
MOST_SEVERE_INJURY               0.244915
INJURIES_NON_INCAPACITATING      0.244610
INJURIES_REPORTED_NOT_EVIDENT    0.244610
INJURIES_NO_INDICATION           0.244610
```

In [20]:
```
# drop columns with missing values above 30%
null_columns = ['CRASH_DATE_EST_I', 'LANE_CNT', 'REPORT_TYPE', 'INTERSECTION_RELATED_I', 'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN_I', 'PHOTOS_
merged_df = merged_df.drop(null_columns, axis = 1)
```

In [21]:
```
# review dataframe after dropping null values
merged_df.isna().mean()*100
```

Out[21]:
```
CRASH_RECORD_ID                0.000000
CRASH_DATE                     0.000000
POSTED_SPEED_LIMIT             0.000000
TRAFFIC_CONTROL_DEVICE         0.000000
DEVICE_CONDITION               0.000000
WEATHER_CONDITION              0.000000
LIGHTING_CONDITION             0.000000
FIRST_CRASH_TYPE               0.000000
TRAFFICWAY_TYPE                0.000000
ALIGNMENT                      0.000000
ROADWAY_SURFACE_COND           0.000000
ROAD_DEFECT                    0.000000
CRASH_TYPE                     0.000000
DAMAGE                         0.000000
DATE_POLICE_NOTIFIED           0.000000
PRIM_CONTRIBUTORY_CAUSE        0.000000
SEC_CONTRIBUTORY_CAUSE         0.000000
STREET_NO                      0.000000
STREET_DIRECTION               0.000305
```

In [22]:
```
# Calculating the count of null values per column after dropping
null_counts = merged_df.isnull().sum()

print("Count of null values per column:")
print(null_counts)
```

```
Count of null values per column:
CRASH_RECORD_ID                0
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
ALIGNMENT                      0
ROADWAY_SURFACE_COND           0
ROAD_DEFECT                    0
CRASH_TYPE                     0
DAMAGE                         0
DATE_POLICE_NOTIFIED           0
PRIM_CONTRIBUTORY_CAUSE        0
SEC_CONTRIBUTORY_CAUSE         0
STREET_NO                      0
```

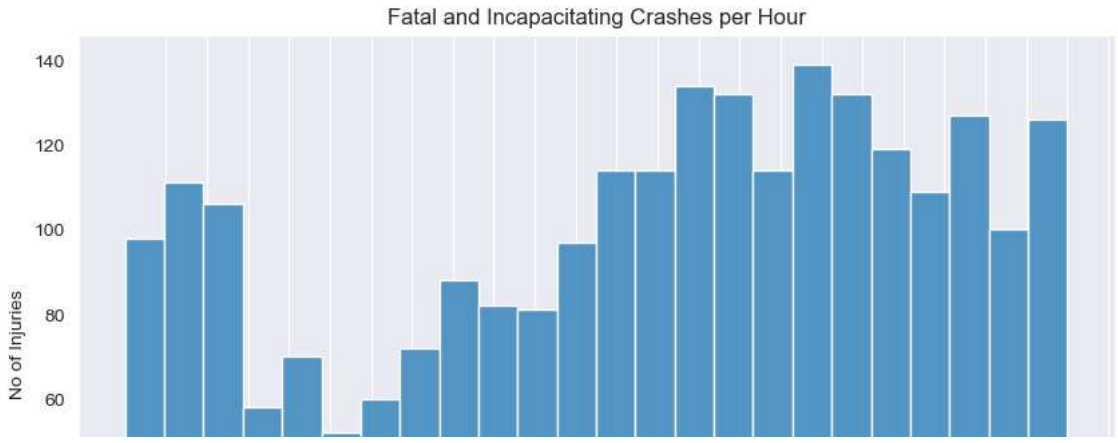
In [23]:
```
# drop the rows with null values as they are not significant
merged_df = merged_df.dropna(axis=0).reset_index(drop=True)
```

In [24]:
```
# Calculating the total number of missing values in the entire dataframe
total_null_values = merged_df.isna().sum().sum()

print("Total number of null values in the DataFrame:", total_null_values)
```

```
Total number of null values in the DataFrame: 0
```

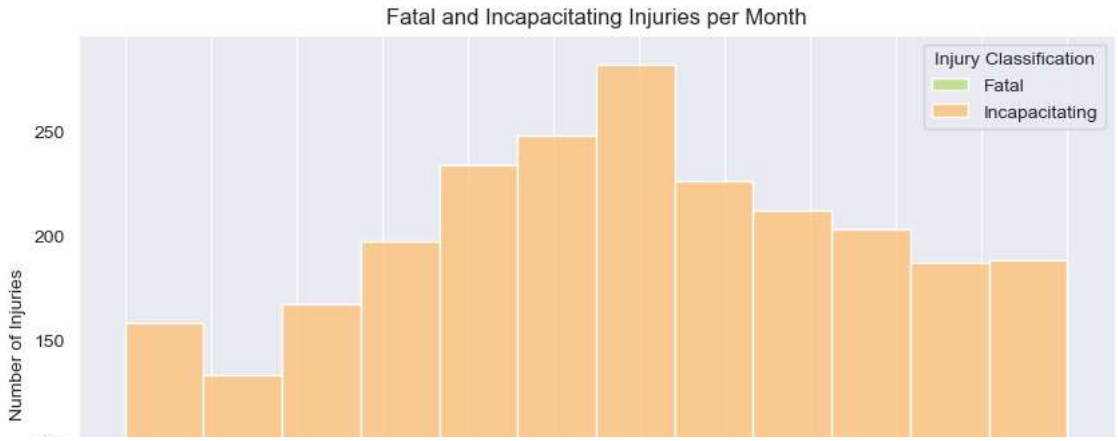
Number of Fatal & Incapacitating Crashes per Hour

In [25]:
```
# Filtering merged_df to include both fatal and incapacitating injuries
fatal_incap_injuries = merged_df[merged_df['INJURY_CLASSIFICATION'].isin(['FATAL', 'INCAPACITATING INJURY'])]

# Plot histogram
plt.figure(figsize=(10, 6))
sns.histplot(data=fatal_incap_injuries, x='CRASH_HOUR', bins=24, kde=False, palette='RdYlGn')
plt.xlabel('Hour of Crash')
plt.ylabel('No of Injuries')
plt.title('Fatal and Incapacitating Crashes per Hour')
plt.xticks(range(1, 25))
plt.grid(axis='y')
plt.show()
```



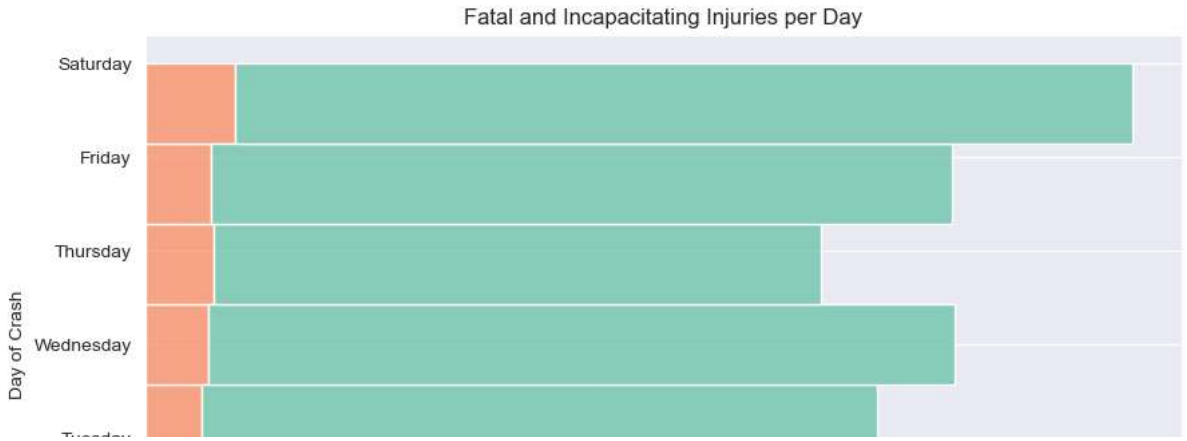
Fatal and Incapacitating Injuries per Month

In [26]:
```
# Filtering merged_df to include both fatal and incapacitating injuries
fatal_incap_injuries = merged_df[merged_df['INJURY_CLASSIFICATION'].isin(['FATAL', 'INCAPACITATING INJURY'])]

# Plot histogram
plt.figure(figsize=(10, 6))
sns.histplot(data=fatal_incap_injuries, x='CRASH_MONTH', hue='INJURY_CLASSIFICATION', bins=12, multiple='stack', palette='RdYlGn')
plt.xlabel('Month of Crash')
plt.ylabel('Number of Injuries')
plt.title('Fatal and Incapacitating Injuries per Month')
plt.xticks(range(1, 13))
plt.legend(title='Injury Classification', labels=['Fatal', 'Incapacitating'], loc='upper right')
plt.grid(axis='y')
plt.show()
```



Fatal and Incapacitating Injuries per Day

In [27]:
```
# Define custom day labels
day_labels = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']

# Filtering merged_df to include both fatal and incapacitating injuries
fatal_incap_injuries = merged_df[merged_df['INJURY_CLASSIFICATION'].isin(['FATAL', 'INCAPACITATING INJURY'])]

plt.figure(figsize=(10, 6))
sns.histplot(data=fatal_incap_injuries, y='CRASH_DAY_OF_WEEK', hue='INJURY_CLASSIFICATION', bins=7, multiple='stack', palette='Set2')
plt.ylabel('Day of Crash')
plt.xlabel('Number of Injuries')
plt.title('Fatal and Incapacitating Injuries per Day')
plt.yticks(range(1, 8), labels=day_labels)  # Specify custom day labels
plt.legend(title='Injury Classification', labels=['Fatal', 'Incapacitating'], loc='lower right')
plt.grid(axis='x')
plt.show()
```



Columns not needed

In [28]:
```
merged_df.columns
```

Out[28]:
```
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', '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',
       'CRASH_YEAR', 'INJURY_CLASSIFICATION'],
      dtype='object')
```

Comment: Some of the columns hold administrative data that will not be needed. These inculde 'CRASH_RECORD_ID', 'CRASH_DATE', 'DATE_POLICE_NOTIFIED', 'BEAT_OF_OCCURRENCE', 'SEC_CONTRIBUTORY_CAUSE', 'STREET_NO', 'STREET_DIRECTION', 'MOST_SEVERE_INJURY', 'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITATING', 'INJURIES_NON_INCAPACITATING', 'INJURIES_REPORTED_NOT_EVIDENT', 'INJURIES_NO_INDICATION', 'INJURIES_UNKNOWN'

In [29]:
```
# dropping columns that are not needed
unneeded_columns = ['CRASH_RECORD_ID', 'CRASH_DATE', 'DATE_POLICE_NOTIFIED', 'BEAT_OF_OCCURRENCE', 'SEC_CONTRIBUTORY_CAUSE', 'STREET_NO
merged_df = merged_df.drop(unneeded_columns, axis = 1)
```

In [30]:
```
# generate a dictionary of columns with categorical data and have values as the count of unique values
{column: len(merged_df[column].unique()) for column in merged_df.columns if merged_df.dtypes[column] == 'object'}
```

Out[30]:
```
{'TRAFFIC_CONTROL_DEVICE': 18,
 'DEVICE_CONDITION': 8,
 'WEATHER_CONDITION': 12,
 'LIGHTING_CONDITION': 6,
 'FIRST_CRASH_TYPE': 18,
 'TRAFFICWAY_TYPE': 20,
 'ALIGNMENT': 6,
 'ROADWAY_SURFACE_COND': 7,
 'ROAD_DEFECT': 7,
 'CRASH_TYPE': 2,
 'DAMAGE': 3,
 'PRIM_CONTRIBUTORY_CAUSE': 38,
 'STREET_NAME': 1453,
 'LOCATION': 161409,
 'INJURY_CLASSIFICATION': 5}
```

In [31]:
```
# drop the columns with very high unique values and not necessary
high_count_columns = ['LOCATION', 'STREET_NAME']
merged_df = merged_df.drop(high_count_columns, axis =1)
```

In [32]:
```
{column: len(merged_df[column].unique()) for column in merged_df.columns if merged_df.dtypes[column] == 'object'}
```

Out[32]:
```
{'TRAFFIC_CONTROL_DEVICE': 18,
 'DEVICE_CONDITION': 8,
 'WEATHER_CONDITION': 12,
 'LIGHTING_CONDITION': 6,
 'FIRST_CRASH_TYPE': 18,
 'TRAFFICWAY_TYPE': 20,
 'ALIGNMENT': 6,
 'ROADWAY_SURFACE_COND': 7,
 'ROAD_DEFECT': 7,
 'CRASH_TYPE': 2,
 'DAMAGE': 3,
 'PRIM_CONTRIBUTORY_CAUSE': 38,
 'INJURY_CLASSIFICATION': 5}
```

In [33]:
```
# Iterating through each column and displaying value counts
for column in merged_df.columns:
    print(f"Value counts for column '{column}':")
    print(merged_df[column].value_counts())
    print()
```

```
Value counts for column 'POSTED_SPEED_LIMIT':
POSTED_SPEED_LIMIT
30    239499
25     21487
35     20701
20     14507
15     11598
10      8507
40      3300
45      2393
5        983
0        650
55       270
50       109
3         74
39        30
24        23
60        18
2          9
```

In [34]:
```
merged_df
```

Out[34]:

	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGHTING_CONDITION	FIRST_CRASH_TYPE	TRAFFICWAY_
0	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	DAYLIGHT	PARKED MOTOR VEHICLE	DIV W/MEDIAN RA
1	30	NO CONTROLS	NO CONTROLS	CLEAR	DAYLIGHT	PEDALCYCLIST	NOT DI
2	10	NO CONTROLS	NO CONTROLS	UNKNOWN	UNKNOWN	ANGLE	PARKIN
3	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	DARKNESS	SIDESWIPE OPPOSITE DIRECTION	NOT DI
4	15	NO CONTROLS	NO CONTROLS	CLEAR	DAYLIGHT	REAR TO SIDE	PARKIN
...	...	...	...	...	...	...	
						PARKED MOTOR	

Target Column

In [35]:
```
# target column
merged_df['INJURY_CLASSIFICATION'].value_counts()
```

Out[35]:
```
INJURY_CLASSIFICATION
NO INDICATION OF INJURY     305572
NONINCAPACITATING INJURY     11420
REPORTED, NOT EVIDENT         4793
INCAPACITATING INJURY         2227
FATAL                          208
Name: count, dtype: int64
```

In [36]:
```
# reducing the number of unique variables within this column
#define the replacements
replacements = {
    'NO INDICATION OF INJURY': 'NO_INJURY',
    'NONINCAPACITATING INJURY': 'NONINCAPACITATING',
    'REPORTED, NOT EVIDENT': 'NONINCAPACITATING'
}

# Replace the values
merged_df['INJURY_CLASSIFICATION'] = merged_df['INJURY_CLASSIFICATION'].replace(replacements)

# Verify the updated value counts
print(merged_df['INJURY_CLASSIFICATION'].value_counts())
```

```
INJURY_CLASSIFICATION
NO_INJURY                305572
NONINCAPACITATING         16213
INCAPACITATING INJURY      2227
FATAL                       208
Name: count, dtype: int64
```

Encoding

In [37]:
```
def onehot_encode(df, columns, prefixes):
    df = df.copy()
    for column, prefix in zip(columns, prefixes):
        dummies = pd.get_dummies(df[column], prefix=prefix)
        df = pd.concat([df, dummies], axis=1)
        df = df.drop(column, axis=1)
    return df
```

In [38]:
```
{column: len(merged_df[column].unique()) for column in merged_df.columns if merged_df.dtypes[column] == 'object'}
```

Out[38]:
```
{'TRAFFIC_CONTROL_DEVICE': 18,
 'DEVICE_CONDITION': 8,
 'WEATHER_CONDITION': 12,
 'LIGHTING_CONDITION': 6,
 'FIRST_CRASH_TYPE': 18,
 'TRAFFICWAY_TYPE': 20,
 'ALIGNMENT': 6,
 'ROADWAY_SURFACE_COND': 7,
 'ROAD_DEFECT': 7,
 'CRASH_TYPE': 2,
 'DAMAGE': 3,
 'PRIM_CONTRIBUTORY_CAUSE': 38,
 'INJURY_CLASSIFICATION': 4}
```

In [39]:
```
merged_df = onehot_encode(merged_df, columns = ['TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION', 'WEATHER_CONDITION', 'LIGHTING_CONDITION'
```

In [40]:
```
merged_df
```

Out[40]:

	POSTED_SPEED_LIMIT	CRASH_TYPE	NUM_UNITS	CRASH_HOUR	CRASH_DAY_OF_WEEK	CRASH_MONTH	LATITUDE	LONGITUDE	CRASH_YEAR	INJURY_CL
0	30	NO INJURY / DRIVE AWAY	4	14	7	7	41.854120	-87.665902	2023	
1	30	INJURY AND / OR TOW DUE TO CRASH	2	17	6	8	41.942976	-87.761883	2023	NONIN
2	10	NO INJURY / DRIVE AWAY	2	14	7	7	41.809781	-87.594213	2023	
3	30	NO INJURY / DRIVE AWAY	2	0	7	7	41.899225	-87.696642	2023	
4	15	NO INJURY / DRIVE AWAY	2	12	4	9	41.744152	-87.585945	2023	
...	...	...	...	...	...	...	...	...	...	
324215	30	NO INJURY / DRIVE AWAY	2	2	3	3	41.785636	-87.614314	2022	
324216	30	INJURY AND / OR TOW DUE TO CRASH	1	22	6	11	41.876044	-87.700459	2021	
324217	30	INJURY AND / OR TOW DUE TO CRASH	2	21	6	6	41.756217	-87.641596	2023	INCAPACI
324218	30	NO INJURY / DRIVE AWAY	2	12	2	7	41.857531	-87.644929	2023	
324219	15	NO INJURY / DRIVE AWAY	2	17	4	5	41.945084	-87.667035	2023	

324220 rows × 153 columns

In [41]:
```
# Identifying boolean columns
boolean_columns = merged_df.select_dtypes(include=bool).columns

# Converting the boolean columns to binary (1/0)
merged_df[boolean_columns] = merged_df[boolean_columns].astype(int)
```

In [42]:
```
merged_df
```

Out[42]:

	POSTED_SPEED_LIMIT	CRASH_TYPE	NUM_UNITS	CRASH_HOUR	CRASH_DAY_OF_WEEK	CRASH_MONTH	LATITUDE	LONGITUDE	CRASH_YEAR	INJURY_CL
0	30	NO INJURY / DRIVE AWAY	4	14	7	7	41.854120	-87.665902	2023	
1	30	INJURY AND / OR TOW DUE TO CRASH	2	17	6	8	41.942976	-87.761883	2023	NONIN
2	10	NO INJURY / DRIVE AWAY	2	14	7	7	41.809781	-87.594213	2023	
3	30	NO INJURY / DRIVE AWAY	2	0	7	7	41.899225	-87.696642	2023	
4	15	NO INJURY / DRIVE AWAY	2	12	4	9	41.744152	-87.585945	2023	
...	...	...	...	...	...	...	...	...	...	
324215	30	NO INJURY / DRIVE AWAY	2	2	3	3	41.785636	-87.614314	2022	
324216	30	INJURY AND / OR TOW DUE TO CRASH	1	22	6	11	41.876044	-87.700459	2021	
324217	30	INJURY AND / OR TOW DUE TO CRASH	2	21	6	6	41.756217	-87.641596	2023	INCAPACI
324218	30	NO INJURY / DRIVE AWAY	2	12	2	7	41.857531	-87.644929	2023	
324219	15	NO INJURY / DRIVE AWAY	2	17	4	5	41.945084	-87.667035	2023	

324220 rows × 153 columns

In [43]:
```
# Defining the mapping dictionary for CRASH_TYPE column
mapping = {'NO INJURY / DRIVE AWAY': 0, 'INJURY AND / OR TOW DUE TO CRASH': 1}

# Replace values in the CRASH_TYPE column
merged_df['CRASH_TYPE'] = merged_df['CRASH_TYPE'].replace(mapping)
```

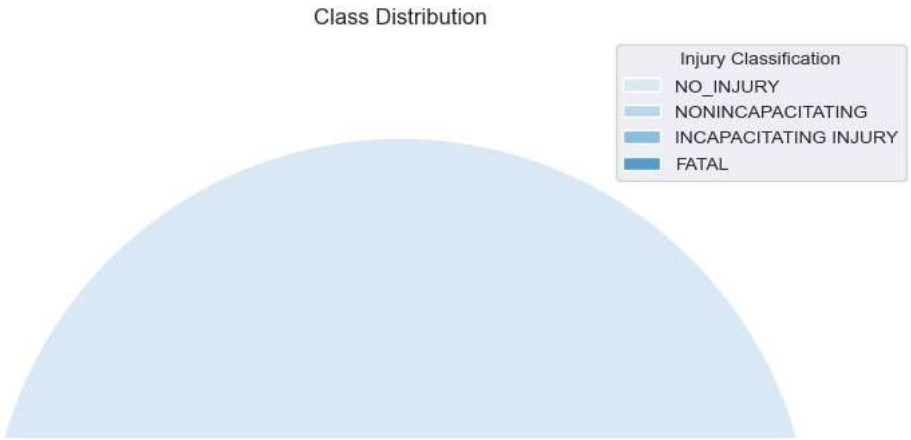
In [44]:
```
merged_df
```

Out[44]:

	POSTED_SPEED_LIMIT	CRASH_TYPE	NUM_UNITS	CRASH_HOUR	CRASH_DAY_OF_WEEK	CRASH_MONTH	LATITUDE	LONGITUDE	CRASH_YEAR	INJURY
0	30	0	4	14	7	7	41.854120	-87.665902	2023	
1	30	1	2	17	6	8	41.942976	-87.761883	2023	NO
2	10	0	2	14	7	7	41.809781	-87.594213	2023	
3	30	0	2	0	7	7	41.899225	-87.696642	2023	
4	15	0	2	12	4	9	41.744152	-87.585945	2023	
...	...	...	...	...	...	...	...	...	...	...
324215	30	0	2	2	3	3	41.785636	-87.614314	2022	

Handling Class Imbalance & Training

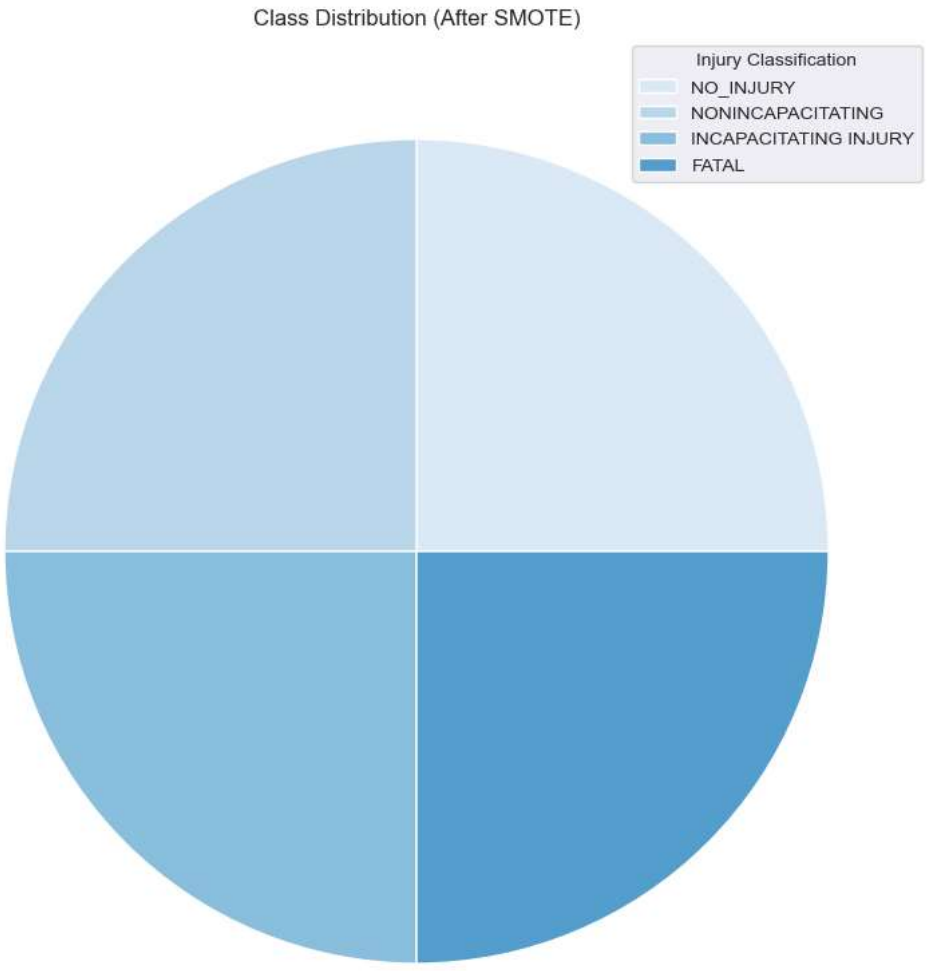
In [45]:
```
# Get the value counts for the INJURY_CLASSIFICATION column
class_counts = merged_df['INJURY_CLASSIFICATION'].value_counts()

# Define labels
labels = ["NO_INJURY", "NONINCAPACITATING", "INCAPACITATING INJURY", "FATAL"]

# Create a color palette
subtle_palette = sns.color_palette("Blues")

# Create the pie chart with the chosen palette
plt.figure(figsize=(10, 10))
plt.pie(class_counts, colors=subtle_palette)

# Add legend
plt.legend(title="Injury Classification", loc="best", labels=labels)

# Add title
plt.title("Class Distribution")

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



In [47]:
```
# Establishing predictor and result variables
# Splitting merged_df into X and y
def preprocess_data(merged_df):
    y = merged_df['INJURY_CLASSIFICATION']
    X = merged_df.drop('INJURY_CLASSIFICATION', axis=1)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=True, random_state=1)

    # Scale X
    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train = pd.DataFrame(scaler.transform(X_train), index=X_train.index, columns=X_train.columns)
    X_test = pd.DataFrame(scaler.transform(X_test), index=X_test.index, columns=X_test.columns)

    return X_train, X_test, y_train, y_test

# Call the preprocess_data function with merged_df as input
X_train, X_test, y_train, y_test = preprocess_data(merged_df)
```

In [49]:

```
from imblearn.over_sampling import SMOTE

y = merged_df['INJURY_CLASSIFICATION']
X = merged_df.drop('INJURY_CLASSIFICATION', axis=1)

# Instantiate SMOTE
smote = SMOTE(random_state=42)

# Resample the data
X_resampled, y_resampled = smote.fit_resample(X, y)

# Get the value counts for the resampled target variable
class_counts_resampled = y_resampled.value_counts()

# Create a color palette
subtle_palette = sns.color_palette("Blues")

# Create the pie chart with the chosen palette
plt.figure(figsize=(10, 10))
plt.pie(class_counts_resampled, colors=subtle_palette)

# Add legend
plt.legend(title="Injury Classification", loc="best", labels=labels)

# Add title
plt.title("Class Distribution (After SMOTE)")

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



In [50]:
```
X_train
```

Out[50]:

	POSTED_SPEED_LIMIT	CRASH_TYPE	NUM_UNITS	CRASH_HOUR	CRASH_DAY_OF_WEEK	CRASH_MONTH	LATITUDE	LONGITUDE	CRASH_YEAR	TCD_BICY CROSS S
314343	0.253353	-0.649649	-0.088387	-0.208123	-1.564707	-0.788451	0.379326	0.006923	1.215304	-0.005
129715	0.253353	1.539292	-0.088387	0.497366	-0.561560	1.293777	0.280175	-0.138339	-0.007272	-0.005
96739	0.253353	-0.649649	-0.088387	-0.208123	-0.059987	1.293777	-0.070236	-0.030803	-0.007272	-0.005
239906	-0.636290	-0.649649	-0.088387	0.673739	-1.564707	0.401393	-0.347839	-0.052994	-0.007272	-0.005
190662	0.253353	1.539292	-0.088387	-0.560867	-0.059987	-1.680835	-0.101584	0.042695	-1.229848	-0.005
...	...	...	...	...	...	...	...	...	...	
117583	-1.525933	-0.649649	-0.088387	0.320994	1.444734	-0.490990	-0.513901	0.020855	-0.007272	-0.005
73349	0.253353	-0.649649	-2.215853	-0.913612	1.444734	-0.193529	-0.226165	-0.068587	-0.007272	-0.005
312201	0.253353	-0.649649	4.166545	0.850111	-0.561560	0.401393	0.301608	-0.242336	-1.229848	-0.005
267336	0.253353	1.539292	-0.088387	1.379228	-1.564707	0.401393	-0.048150	-0.030715	-0.007272	-0.005
128037	-1.525933	1.539292	-0.088387	0.320994	-0.561560	-0.490990	-0.355330	0.070402	-0.007272	-0.005

226954 rows × 152 columns

In [51]:
```
y_train
```

Out[51]:
```
314343            NO_INJURY
129715    NONINCAPACITATING
96739             NO_INJURY
239906            NO_INJURY
190662    NONINCAPACITATING
                   ...
117583            NO_INJURY
73349             NO_INJURY
312201            NO_INJURY
267336            NO_INJURY
128037            NO_INJURY
Name: INJURY_CLASSIFICATION, Length: 226954, dtype: object
```

Modelling & Evaluation

Model I: Logistic Regression

In [52]:
```
# Instantiate Logistic Regression classifier
log_reg_classifier = LogisticRegression()

# Fit the classifier
log_reg_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred_log_reg = log_reg_classifier.predict(X_test)
```

In [53]:
```
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score

def print_metrics(labels, preds):
    precision = precision_score(labels, preds, average='weighted') * 100
    recall = recall_score(labels, preds, average='weighted') * 100
    accuracy = accuracy_score(labels, preds) * 100
    f1 = f1_score(labels, preds, average='weighted') * 100

    print("Precision Score: {:.2f}%".format(precision))
    print("Recall Score: {:.2f}%".format(recall))
    print("Accuracy Score: {:.2f}%".format(accuracy))
    print("F1 Score: {:.2f}%".format(f1))

print("\nMetrics for Logistic Regression:")
print_metrics(y_test, y_pred_log_reg)
```

```
Metrics for Logistic Regression:
Precision Score: 91.15%
Recall Score: 94.31%
Accuracy Score: 94.31%
F1 Score: 91.69%
```

In [54]:
```
unique_test_classes = np.unique(y_test)
unique_pred_classes = np.unique(y_pred_log_reg)

print("Unique classes in y_test:", unique_test_classes)
print("Unique classes in y_pred_log_reg:", unique_pred_classes)
```

```
Unique classes in y_test: ['FATAL' 'INCAPACITATING INJURY' 'NONINCAPACITATING' 'NO_INJURY']
Unique classes in y_pred_log_reg: ['FATAL' 'INCAPACITATING INJURY' 'NONINCAPACITATING' 'NO_INJURY']
```

Model II: K-Nearest Neighbors

In [55]:
```
# Instantiate KNeighborsClassifier
knn_classifier = KNeighborsClassifier()

# Fit the classifier
knn_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred_knn = knn_classifier.predict(X_test)

# Printing metrics
print("\nMetrics for KNN:")
print_metrics(y_test, y_pred_knn)
```

```
Metrics for KNN:
Precision Score: 90.46%
Recall Score: 93.57%
Accuracy Score: 93.57%
F1 Score: 91.72%
```

Model III: Decision Tree

In [56]:
```
# Instantiate DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier()

# Fit the classifier
dt_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred_dt = dt_classifier.predict(X_test)

# Printing metrics
print("\nMetrics for Decision Tree:")
print_metrics(y_test, y_pred_dt)
```

```
Metrics for Decision Tree:
Precision Score: 91.00%
Recall Score: 90.57%
Accuracy Score: 90.57%
F1 Score: 90.78%
```

Model IV: Random Forest

In [57]:
```
# Instantiate RandomForestClassifier
rf_classifier = RandomForestClassifier()

# Fit the classifier
rf_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = rf_classifier.predict(X_test)

# Print metrics for Random Forest
print("\nMetrics for RandomForestClassifier:")
print_metrics(y_test, y_pred_rf)
```

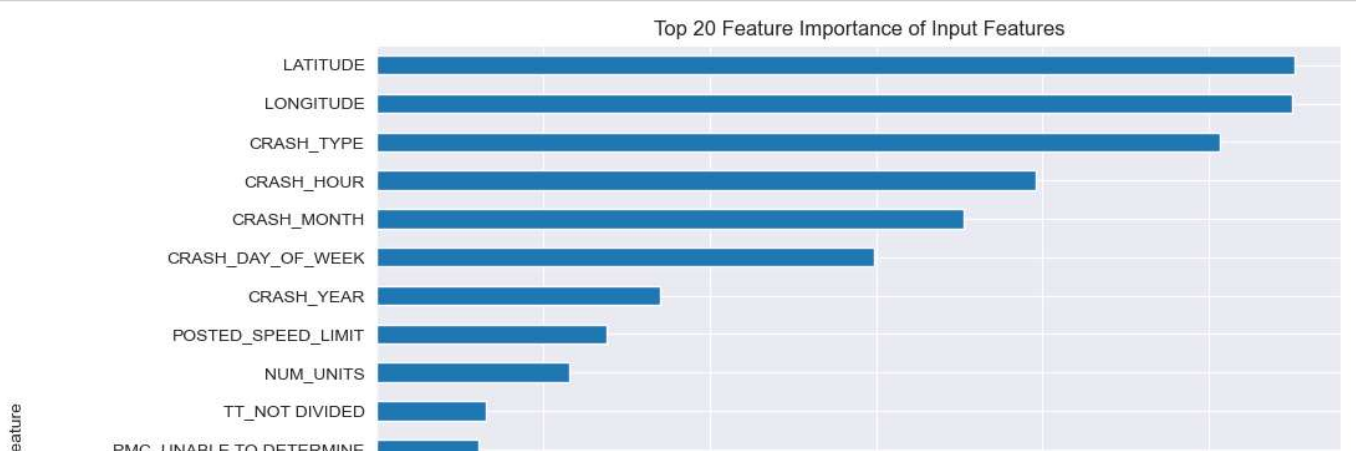
```
Metrics for RandomForestClassifier:
Precision Score: 91.04%
Recall Score: 94.29%
Accuracy Score: 94.29%
F1 Score: 91.67%
```

Interpretation

- All the models performed relartively well, with scores above 90% across the four models
- Logistic Regression and Random Forest models performed better than the other two
- We settle on Random Forest because it is better in:
  - Dealing with complex, non-linear relationships in our data
  - Handling large dataset. The datasets provided by Chicago City are large.

Feature Importance

In [58]:
```
# checking for feature importance
importance = pd.Series(rf_classifier.feature_importances_, index=X_train.columns)
top_20_importance = importance.nlargest(20)

# Plot the horizontal bar plot
plt.figure(figsize=(10, 8))
top_20_importance.sort_values().plot(kind='barh')
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Top 20 Feature Importance of Input Features')
plt.show()
```



In [ ]:

Top 5 contributors to our model's prediction:

---

1. Latitude
2. Longitude
3. Crash_hour
4. Crash_type
5. Crash_month

These features have a stronger influence on the model's decision-making process as compared to the others.

They align with key factors known to influence road safety outcomes, including geographical location, time of day, month and type of crash. Incorporating these features into our model provides valuable insights into the underlying patterns and risk factors associated with road crashes, enabling City of Chicago administration to develop targeted interventions and strategies for reducing fatalities arising from road crashes.

Recommendations ¶

1. **Strategic Location-Based Safety Measures:** Focus on high-risk areas identified by latitude and longitude coordinates to implement targeted safety interventions, such as increased police presence and improved road infrastructure.
2. **Time-Specific Safety Initiatives:** Utilize crash hour and month data to deploy time-specific safety measures, like heightened law enforcement during peak accident hours or months, to reduce severe injuries.
3. **Customized Educational Campaigns:** Tailor educational campaigns based on crash types to specific demographics or communities, empowering residents with knowledge to prevent accidents and minimize injury severity.