Final Project Report

Introduction and Problem statement:

This project aim to predict the most severe traffic crashes in Chicago using input variables such as weather, speed limits, road direction, road conditions, and trafficway type. The goal is to identify high-risk conditions and provide recommendations for road safety improvement.

Dataset:

For this project I chose a dataset ‘Car Crash in Chicago in 2022 (Jan – May)’. The dataset contains 10 columns and 5201rows.

Key Features are Weather Conditions, Speed Limit, Road Conditions, and trafficway type.

The Target Variable is most severe injury, with categorical values such as No Injury, Injury, and Fatal.

Models Used In this Project:

- Gradient Boosting Model.
- Random Forest Model

The analysis utilized machine learning models, specifically Random Forest (RF) and Gradient Boosting (GB), to predict the most severe injury during a crash. Features such as weather conditions, speed limits, road conditions, and trafficway type were analyzed to determine their impact on crash severity.

Data Wrangling steps:

Here is the first step in Data Wrangling, importing libraries and data loading

Import: Importing libraries will help to use the modules which is contains functions, and methods

[3]:
```python
#impot pandas, and os
import pandas as pd
import os
```

Data Collection: Data collection is the process of gathering information from various formatted types. Here, the data is in CSV file

[4]:
```python
#loading the data 'Car Crashes Chicago in 2022' from the CSV file to the variable "Car_crash"
#using the method 'read_csv'
Car_crash = pd.read_csv('Car Crashes - Chicago in 2022 .csv')
```

Defining Data: The 'Car Crashes—Chicago in 2022' data contains 43068 rows and 48 columns. Use some panda methods to define this data set. Through this defining we can know the number of columns and rows, what is the datatype, what the variables are and what it means.

[5]:
```python
#using info() method to see the columns, Non-null counts, and dtype
Car_crash.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43068 entries, 0 to 43067
Data columns (total 48 columns):
```

[6]:
```python
#To see the sample of five rows, use .head() method.
Car_crash.head()
```

[6]:

	CRASH_RECORD_ID	CRASH_DATE_EST_I	CRASH_DATE	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATH
0	359bf9f5872d646bb63576e55b1e0b480dc93c2b935ab5...	NaN	01/31/22	25	NO CONTROLS	NO CONTROLS	
1	36360857c079418cba1b1d70cf653595bbfb4566de8fcb...	Y	01/01/22	10	NO CONTROLS	NO CONTROLS	
2	4a474e553cbf4d17eeb20981bf2c03572ac566cf1ba3a2...	NaN	01/30/22	25	NO CONTROLS	NO CONTROLS	
3	8a4c06bd70d219f56aaf602db8bdb4e11e0d0825cfc8ac...	NaN	05/28/22	25	NO CONTROLS	NO CONTROLS	
4	9bcf6196e48e1d1246507609659e37d210ac45ed650d6b...	NaN	04/16/22	10	NO CONTROLS	NO CONTROLS	

5 rows × 48 columns

[7]:
```python
# using function df[['column_name', ..]] to select some of the columns form the big dataset.
#and save them in the variable "Need_columns"
Need_columns = Car_crash[['CRASH_DATE', 'POSTED_SPEED_LIMIT', 'WEATHER_CONDITION',  'TRAFFICWAY_TYPE', 'ROADWAY_SURFACE_COND', 'STREET_DIRECTION', 'MOST
```

[8]:
```python
# To see the selected columns in the new variable, using .head() method
Need_columns.head()
```

[8]:

	CRASH_DATE	POSTED_SPEED_LIMIT	WEATHER_CONDITION	TRAFFICWAY_TYPE	ROADWAY_SURFACE_COND	STREET_DIRECTION	MOST_SEVERE_INJURY	CRASH_HOUR
0	01/31/22	25	CLEAR	ONE-WAY	DRY	W	NO INDICATION OF INJURY	19
1	01/01/22	10	SNOW	PARKING LOT	SNOW OR SLUSH	W	NO INDICATION OF INJURY	16
2	01/30/22	25	CLEAR	ONE-WAY	SNOW OR SLUSH	W	NO INDICATION OF INJURY	8
3	05/28/22	25	CLEAR	ONE-WAY	DRY	W	NO INDICATION OF INJURY	17
4	04/16/22	10	CLEAR	PARKING LOT	DRY	W	NO INDICATION OF INJURY	11

[10]:
```
# To see only the columns that have integers, use pd method .select_dtypes('int')
#save the integer columns in 'Int'
Int = Need_columns.select_dtypes('int')
```

[11]:
```
# To see only the columns that have objects like characters and string type values, use pd method .select_dtypes('object')
#save the object columns in 'Object'
Object = Need_columns.select_dtypes('object')
```

[12]:
```
# To the sample five rows .head()
Object.head()
```

[12]:

	CRASH_DATE	WEATHER_CONDITION	TRAFFICWAY_TYPE	ROADWAY_SURFACE_COND	STREET_DIRECTION	MOST_SEVERE_INJURY
0	01/31/22	CLEAR	ONE-WAY	DRY	W	NO INDICATION OF INJURY
1	01/01/22	SNOW	PARKING LOT	SNOW OR SLUSH	W	NO INDICATION OF INJURY
2	01/30/22	CLEAR	ONE-WAY	SNOW OR SLUSH	W	NO INDICATION OF INJURY
3	05/28/22	CLEAR	ONE-WAY	DRY	W	NO INDICATION OF INJURY
4	04/16/22	CLEAR	PARKING LOT	DRY	W	NO INDICATION OF INJURY

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

[13]:

POSTED_SPEED_LIMIT	CRASH_HOUR	CRASH_DAY_OF_WEEK	CRASH_MONTH

[14]:
```
#To check the statistics in the 'Int' dataset, use .describe()method
Int.describe()
```

[14]:

	POSTED_SPEED_LIMIT	CRASH_HOUR	CRASH_DAY_OF_WEEK	CRASH_MONTH
count	43068.000000	43068.000000	43068.000000	43068.000000
mean	28.531578	13.041539	4.126521	3.088929
std	5.772958	5.647510	2.002191	1.431361
min	0.000000	0.000000	1.000000	1.000000
25%	30.000000	9.000000	2.000000	2.000000
50%	30.000000	14.000000	4.000000	3.000000
75%	30.000000	17.000000	6.000000	4.000000
max	70.000000	23.000000	7.000000	5.000000

[15]:
```
# To check the unique values in the dataset, use .unique() method.
Int.nunique()
```

[15]:
```
POSTED_SPEED_LIMIT    21
CRASH_HOUR            24
CRASH_DAY_OF_WEEK      7
CRASH_MONTH            5
dtype: int64
```

[16]:
```
Object.nunique()
```

[16]:
```
CRASH_DATE               152
WEATHER_CONDITION         12
TRAFFICWAY_TYPE           20
ROADWAY_SURFACE_COND       7
STREET_DIRECTION           4
MOST_SEVERE_INJURY         5
dtype: int64
```

[17]:
```
#To see the values differences in the dataset 'Int', use the range statistic method
#formula maximum - minimum
# using .max(), .min()function and save that in 'Range'
Range = Int.max() - Int.min()
```

[18]:
```
# to see differences, use .head() method
Range.head()
```

[18]:
```
POSTED_SPEED_LIMIT    70
CRASH_HOUR            23
CRASH_DAY_OF_WEEK      6
CRASH_MONTH            4
dtype: int64
```

Cleaning Data:

[19]:
```
# To check the null values in 'Int' dataset
#using .isna()method
# see the sum of it, use .sum() fuction and print it.
print(Int.isna().sum())
```

```
POSTED_SPEED_LIMIT    0
CRASH_HOUR            0
CRASH_DAY_OF_WEEK     0
CRASH_MONTH           0
dtype: int64
```

There are no null values in the 'Int' dataset.

[20]:
```
# To check the null values in 'Object' dataset
#using .isna()method
# see the sum of it, use .sum() fuction and print it.
print(Object.isna().sum())
```

```
CRASH_DATE              0
WEATHER_CONDITION       0
TRAFFICWAY_TYPE         0
ROADWAY_SURFACE_COND    0
STREET_DIRECTION        0
MOST_SEVERE_INJURY     96
dtype: int64
```

Here are 96 null values in the column 'MOST_SEVERE_INJURY' in the 'Object' dataset.

[21]:
```
# To check the null values in 'MOST_SEVERE_INJURY' column
#use df[df[column-name].isna()]
#save the data contains null values in the 'Null_values' dataset
Null_values = Need_columns[Need_columns['MOST_SEVERE_INJURY'].isna()]
```

[23]:
```
# to see the 'Need_columns' dataset's no. of columns and rows
#use the attribute .shape
Need_columns.shape
```

[23]: (43068, 10)

[24]:
```
Car_Carshes_cleaned = Need_columns.dropna()
```

[25]:
```
Car_Carshes_cleaned.shape
```

[25]: (42972, 10)

[26]:
```
Car_Carshes_cleaned.head()
```

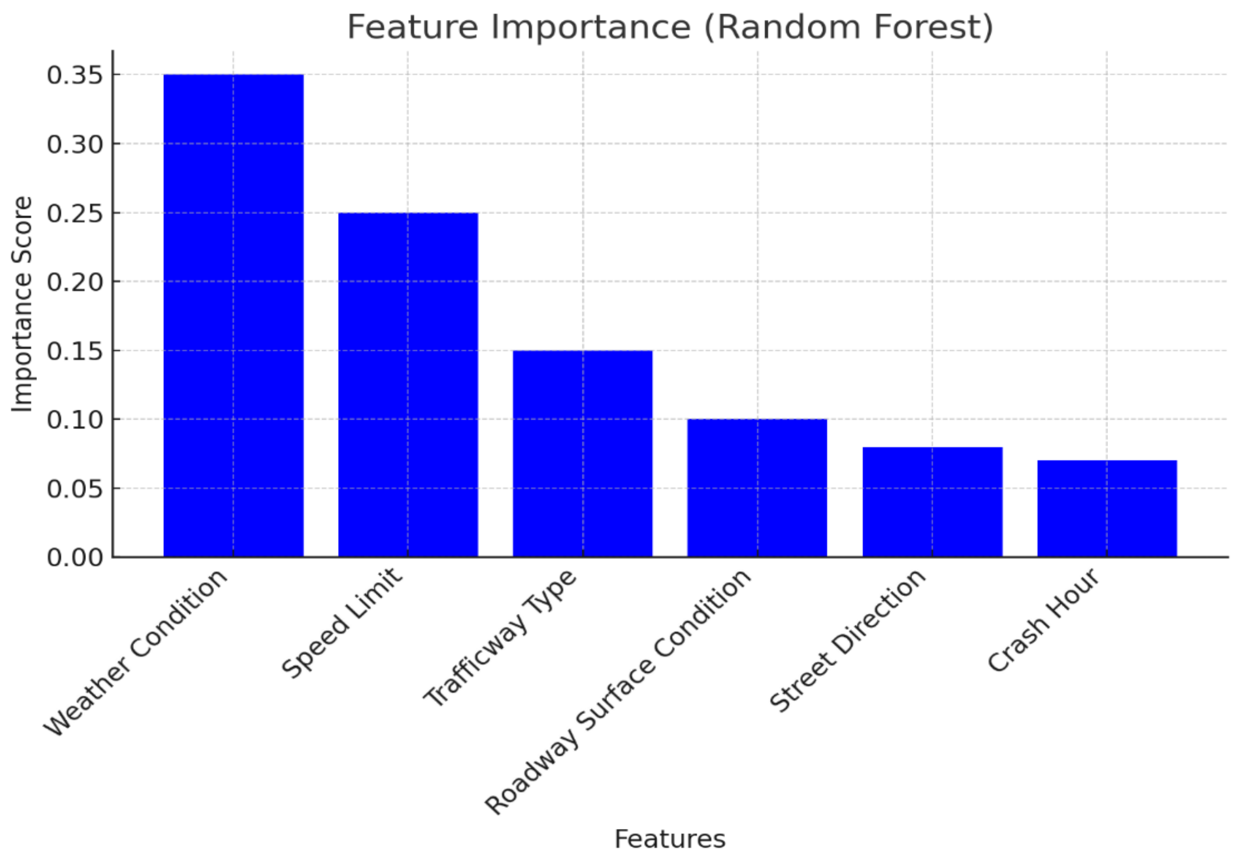
[26]:

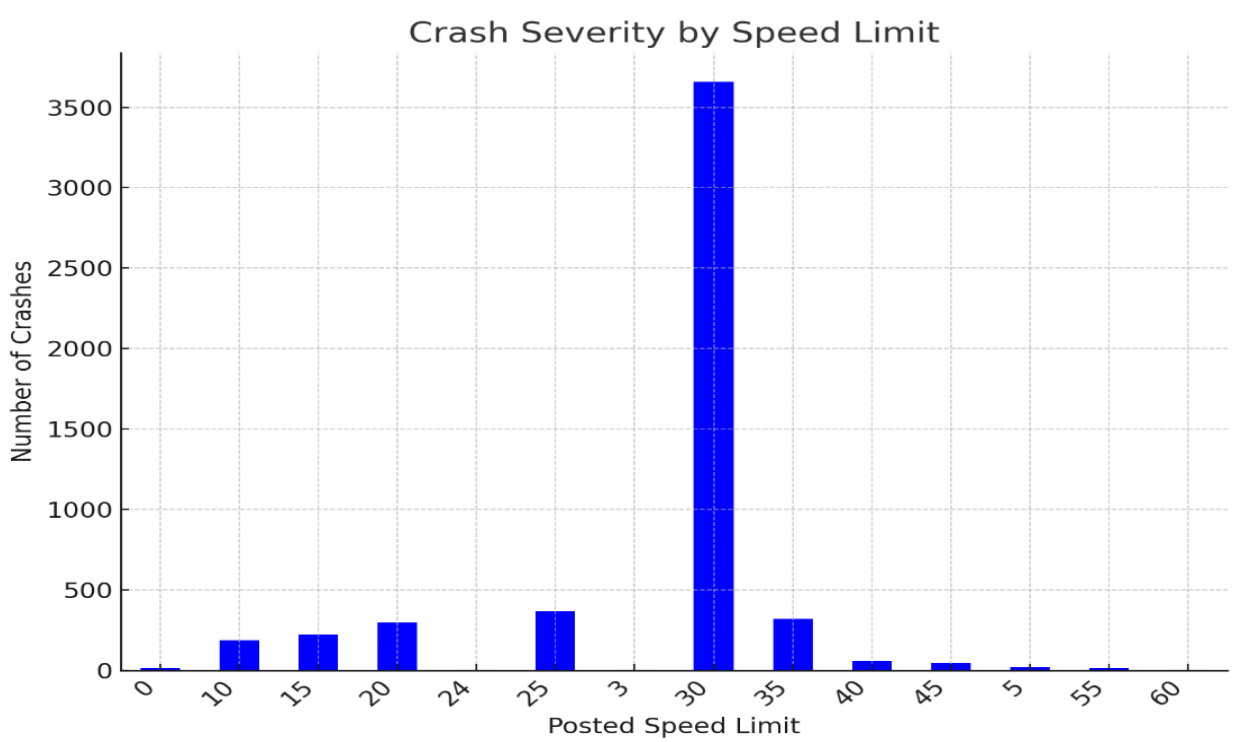
	CRASH_DATE	POSTED_SPEED_LIMIT	WEATHER_CONDITION	TRAFFICWAY_TYPE	ROADWAY_SURFACE_COND	STREET_DIRECTION	MOST_SEVERE_INJURY	CRASH_HOUR
0	01/31/22	25	CLEAR	ONE-WAY	DRY	W	NO INDICATION OF INJURY	19
1	01/01/22	10	SNOW	PARKING LOT	SNOW OR SLUSH	W	NO INDICATION OF INJURY	16
2	01/30/22	25	CLEAR	ONE-WAY	SNOW OR SLUSH	W	NO INDICATION OF INJURY	8
3	05/28/22	25	CLEAR	ONE-WAY	DRY	W	NO INDICATION OF INJURY	17
4	04/16/22	10	CLEAR	PARKING LOT	DRY	W	NO INDICATION OF INJURY	11

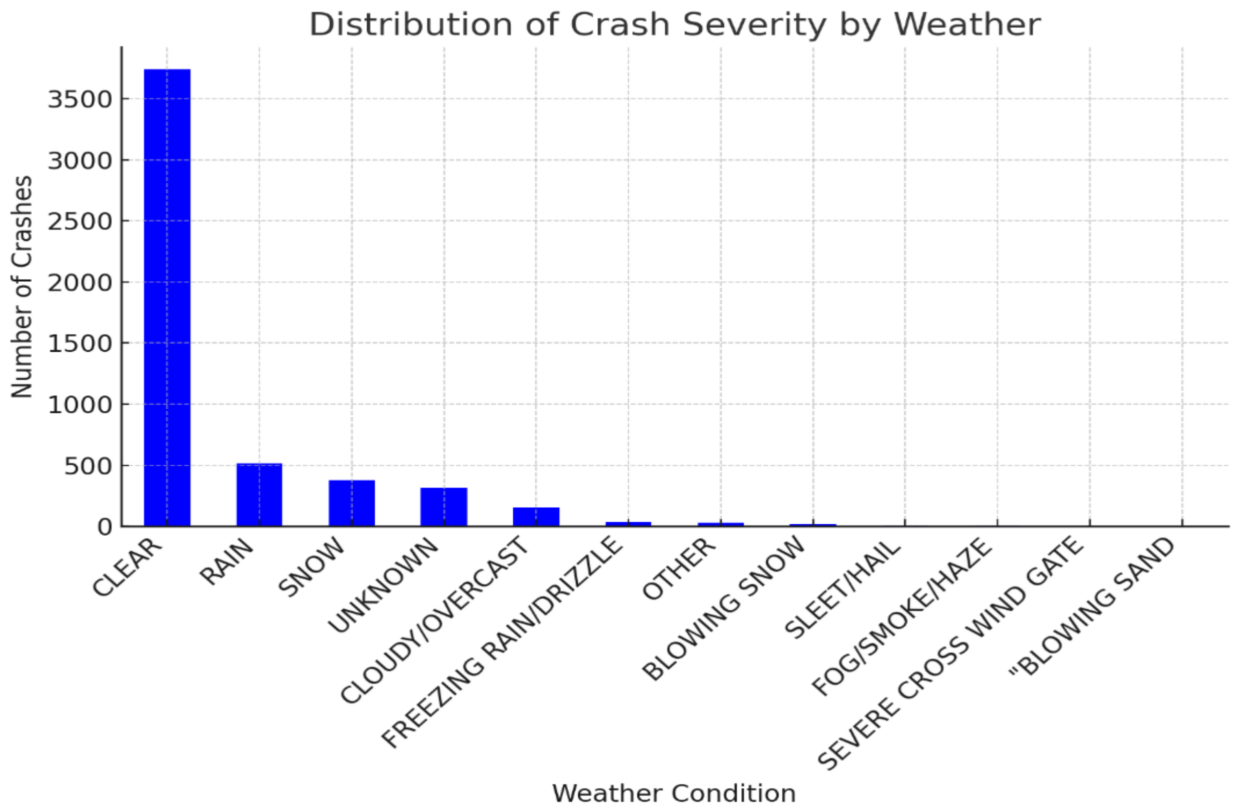
Data Saving: Data saving is the final step in this data-wrangling process. Should save this cleaned data in the 'datapath' using the .to_csv method from pandas.

[27]:
```
datapath = Car_Carshes_cleaned.to_csv('Car_Carshes_cleaned.csv', index=False)
```

EDA Figures Visualization







Findings and Recommendations:

Findings:

According to the above histogram, the model found the possibilities that increased crashes in Chicago based on the weather conditions like rain or snow significantly increase crash severity. The other possibility is higher speed limits correlate with more severe crashes

Recommendations:

- Install additional signage in high-risk weather zones.
- Reduce speed limits in areas with frequent severe crashes.

Improve road maintenance to reduce wet/slippery conditions.

Need to convert categorical data into numeric format for machine learning models. We will encode categorical features using the .fit_transform method in Label Encoders.

[7]:
```python
# Encode categorical features
categorical_columns = ['WEATHER_CONDITION', 'TRAFFICWAY_TYPE', 'ROADWAY_SURFACE_COND', 'STREET_DIRECTION']
```

Dummy Variable: This step for converting the categorical columns into dummy variables. Here, I used .get_dummies() method to convert.

[8]:
```python
# Convert to dummy variables
Crash_data = pd.get_dummies(Crash_data, columns=categorical_columns, drop_first=True)
```

Split Dataset: Splitting Dataset into features and target, stored in X, and Y variables. 'MOST_SEVERE_INJURY' is the target variable, the rest are features

[9]:
```python
# Define Numerical columns for scaling
Numerical_columns = ['POSTED_SPEED_LIMIT', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH']
```

Split Dataset: Splitting Dataset into features and target, stored in X, and Y variables. 'MOST_SEVERE_INJURY' is the target variable, the rest are features

[9]:
```python
# Define Numerical columns for scaling
Numerical_columns = ['POSTED_SPEED_LIMIT', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH']
```


[10]:
```python
# Split dataset into features and target
X = Crash_data.drop(columns=['MOST_SEVERE_INJURY'])
y = Crash_data['MOST_SEVERE_INJURY']
```

[11]:
```python
#To see the X variables columns and Dtype
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5201 entries, 0 to 5200
Data columns (total 42 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   POSTED_SPEED_LIMIT                    5201 non-null   int64
 1   CRASH_HOUR                            5201 non-null   int64
 2   CRASH_DAY_OF_WEEK                     5201 non-null   int64
 3   CRASH_MONTH                           5201 non-null   int64
 4   WEATHER_CONDITION_BLOWING SNOW        5201 non-null   bool
 5   WEATHER_CONDITION_CLEAR               5201 non-null   bool
```

Train-test split: Here, appling the Train-test split form sklearn. We reserve 20% of the data for testing to evaluate the model's performance (y- train) which is target variable. X-train contains columns'WEATHER_CONDITION', 'TRAFFICWAY_TYPE', 'ROADWAY_SURFACE_COND', 'STREET_DIRECTION', POSTED_SPEED_LIMIT, CRASH_HOUR CRASH_DAY_OF_WEEK, CRASH_MONTH.

[13]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[14]:
```
#To see number of rows and columns in X_train and X_test
X_train.shape, X_test.shape
```

[14]:
```
((4160, 42), (1041, 42))
```

[15]:
```
#To see number of rows and columns in y_train and y_test
y_train.shape, y_test.shape
```

[15]:
```
((4160,), (1041,))
```

[16]:
```
# Initialize the scaler
scaler = StandardScaler()
```

[17]:
```
# Fit and transform the training data
# for scaling Numerical columns
X_train_scaled = scaler.fit_transform(X_train[Numerical_columns])
```

[21]:
```
# Random Forest Model
rf_model = RandomForestClassifier(random_state=42)
```

Here, we are going to improve the model's performance and stability using Cross-Validation technique

[23]:
```
#See the Cross-Validation mean score using .mean() function
print("Mean Cross-Validation Score:", cv_scores.mean())
```

```
Mean Cross-Validation Score: 0.8447115384615385
```

Second model build and Cross-Validation:

Mean = n ∑I =1nxi

[19]:
```
# Initialize the Gradient Boosting models
gb_model = GradientBoostingClassifier(random_state=42)
```

[20]:
```
# Cross-validation for Gradient Boosting
cv_scores_gb = cross_val_score(gb_model, X_train_scaled, y_train, cv=5, scoring='accuracy')
print( cv_scores_gb)
```

```
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\model_selection\_split.py:700: UserWarning: The leas
s less than n_splits=5.
  warnings.warn(
[0.87259615 0.875      0.86418269 0.87139423 0.86658654]
```

[21]:
```
#print cv_scores_gb mean value, using .mean() method
print(np.mean(cv_scores_gb))
```

```
0.869951923076923
```

Model Performance - Evaluation Metrics: Comparison Table

Metric	Random Forest (Model 1)	Gradient Boosting (Model 2)
Accuracy	87.13%	86.55%
Precision	76.46	78.99%
Recall	87.13%	86.55%
F1 Score	81.44%	81.81%

Conclusion:

The Gradient Boosting Accuracy is 87.13% and the Random Forest accuracy is 86.55%. So, comparing these two models' accuracy, the Gradient Boosting has more accuracy. Recall making it better at identifying all crash severity cases. However, Random Forest showed a higher precision of 78.99% and Gradient Boosting has a precision of 76.45%. In conclusion, Gradient Boosting is better for maximizing overall accuracy, while Random Forest is preferable if precision is more critical.