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.

Model Performance - Evaluation Metrics

Random Forest Model:

Features: Weather, Speed Limit, Trafficway Type, Roadway Surface Condition

Hyperparameters:

- Number of Trees: 100
- Max Depth: 10

Performance Metrics:

- Accuracy: 86.55%
- Precision: 78.99%
- Recall: 86.55%
- F1 Score: 81.81%

Gradient Boosting Model:

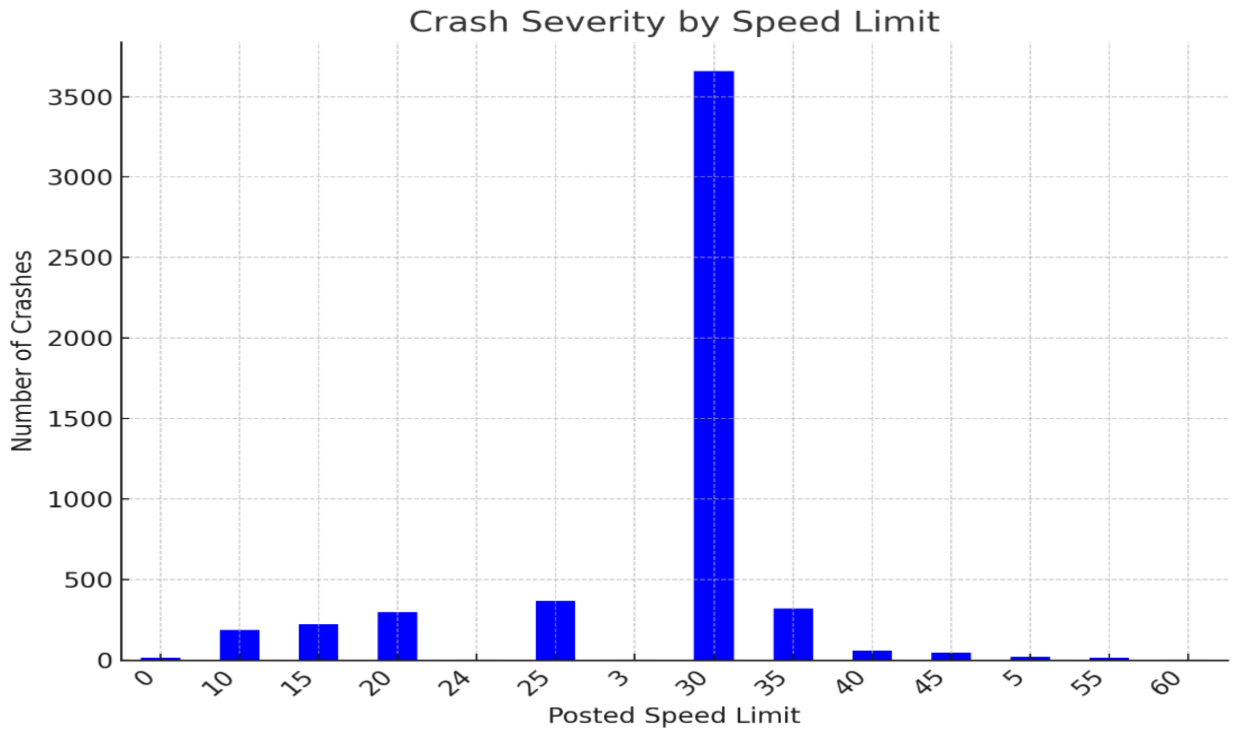
Features: Weather, Speed Limit, Trafficway Type, Roadway Surface Condition

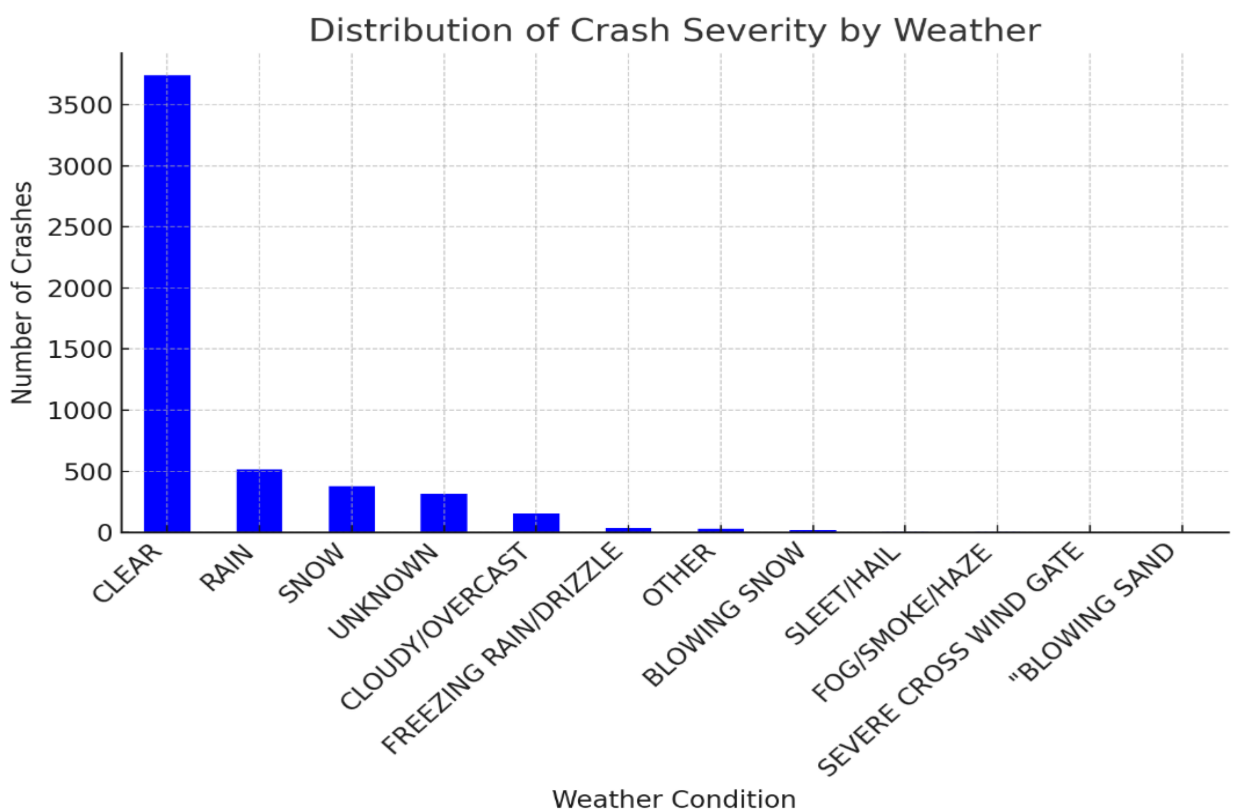
Hyperparameters:

- Learning Rate: 0.1
- Number of Estimators: 100

Performance Metrics:

- Accuracy: 87.13%
- Precision: 76.46%
- Recall: 87.13%
- F1 Score: 81.44%


Crash Severity by Speed Limit
Number of Crashes
3500
3000
2500
2000
1500
1000
500
0
0
10
15
20
24
25
3
30
35
40
45
5
55
60
Posted Speed Limit


Distribution of Crash Severity by Weather
Number of Crashes
3500
3000
2500
2000
1500
1000
500
0
CLEAR
RAIN
SNOW
UNKNOWN
CLOUDY/OVERCAST
FREEZING RAIN/DRIZZLE
OTHER
BLOWING SNOW
SLEET/HAIL
FOG/SMOKE/HAZE
SEVERE CROSS WIND GATE
"BLOWING SAND
Weather Condition

Findings and Recommendations:

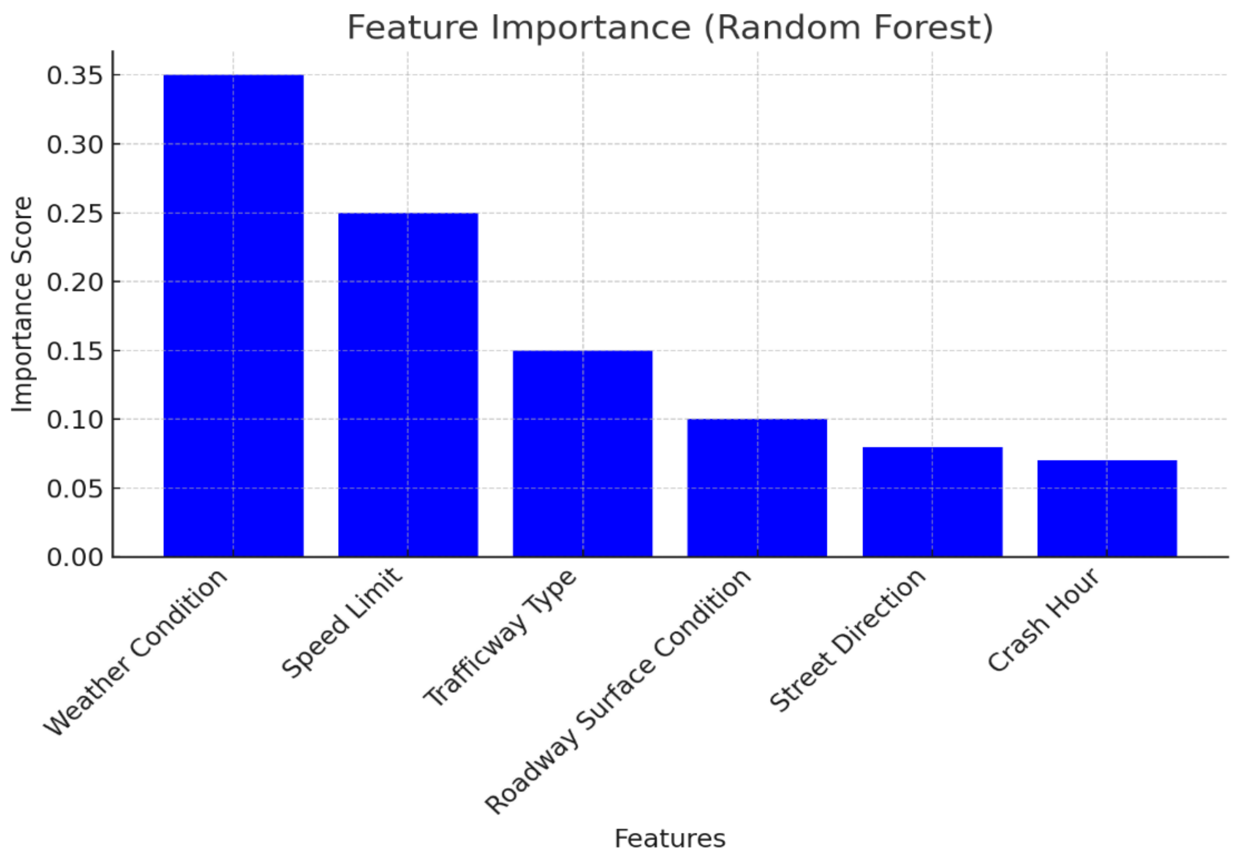
Findings:

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.



Modeling:

The modeling process will train and evaluate two machine learning models, Random Forest, and Gradient Boosting, to predict crash severity using key features like weather conditions, speed limits, and road characteristics. These models will show the accuracy, and F1-score to identify the best predictors of high-severity crashes in Chicago in 2022.

Import - Libraries: will help to use the modules that contain functions, and methods

In [2]:
```
import os
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, Strat
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassif
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score, precision
import numpy as np
```

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

In [3]:
```
#loading dataset in a new dataframe 'Car_Crash'
Car_Crash = pd.read_csv('Crash_analyzed.csv')
```

In [4]:
```
#To see the sample of five rows, use .head() method.
Car_Crash.head()
```

Out[4]:

	CRASH_DATE	POSTED_SPEED_LIMIT	WEATHER_CONDITION	TRAFFICWAY_TY
0	2022-01-31	25	CLEAR	ONE-W
1	2022-01-01	10	SNOW	PARKING L
2	2022-01-30	25	CLEAR	ONE-W
3	2022-05-28	25	CLEAR	ONE-W
4	2022-04-16	10	CLEAR	PARKING L

In [5]:
```
#To see columns names and dtype in Car_Crash
Car_Crash.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5201 entries, 0 to 5200
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   CRASH_DATE           5201 non-null   object
 1   POSTED_SPEED_LIMIT   5201 non-null   int64
 2   WEATHER_CONDITION    5201 non-null   object
 3   TRAFFICWAY_TYPE      5201 non-null   object
 4   ROADWAY_SURFACE_COND 5201 non-null   object
 5   STREET_DIRECTION     5201 non-null   object
 6   MOST_SEVERE_INJURY   5201 non-null   object
 7   CRASH_HOUR           5201 non-null   int64
 8   CRASH_DAY_OF_WEEK    5201 non-null   int64
 9   CRASH_MONTH          5201 non-null   int64
dtypes: int64(4), object(6)
memory usage: 406.5+ KB
```

Deleting columns using .drop() method in python

In [6]:
```
# Drop unnecessary columns
Crash_data = Car_Crash.drop(columns=['CRASH_DATE'])
```

In [7]:
```
#To see number of rows and columns in Crash_data
Crash_data.shape
```

Out[7]:
```
(5201, 9)
```

Before split the dataset, encode the categorical features from the dataset

In [8]:
```
# Encode categorical features
label_encoders = {}
for column in ['WEATHER_CONDITION', 'TRAFFICWAY_TYPE', 'ROADWAY_SURFACE_COND
    le = LabelEncoder()
    Crash_data[column] = le.fit_transform(Crash_data[column])
    label_encoders[column] = le
```

In [9]:
```
# Split dataset into features and target
X = Crash_data.drop(columns=['MOST_SEVERE_INJURY'])
y = Crash_data['MOST_SEVERE_INJURY']
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5201 entries, 0 to 5200
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   POSTED_SPEED_LIMIT   5201 non-null   int64
 1   WEATHER_CONDITION    5201 non-null   int32
 2   TRAFFICWAY_TYPE      5201 non-null   int32
 3   ROADWAY_SURFACE_COND 5201 non-null   int32
 4   STREET_DIRECTION     5201 non-null   int32
 5   CRASH_HOUR           5201 non-null   int64
 6   CRASH_DAY_OF_WEEK    5201 non-null   int64
 7   CRASH_MONTH          5201 non-null   int64
dtypes: int32(4), int64(4)
memory usage: 243.9 KB
```

In [ ]:

In [11]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

Cross-validation: A technique to assess model performance by splitting the data into multiple training and testing subsets.

In [12]:
```
# Cross-validation setup
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

In [13]:
```
# Random Forest Model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

Out[13]:
```
▼ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [15]:
```
# Cross-validation for Random Forest
rf_cv_scores = cross_val_score(rf_model, X_train, y_train, cv=cv, scoring='a
print("Random Forest Cross-Validation Accuracy Scores:", rf_cv_scores)
```

```
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\model_selection\_split.p
y:700: UserWarning: The least populated class in y has only 3 members, which
is less than n_splits=5.
  warnings.warn(
Random Forest Cross-Validation Accuracy Scores: [0.86658654 0.85576923 0.865
38462 0.86177885 0.86418269]
```

In [16]:
```
#To check mean value for the CV Accuracy
print("Mean CV Accuracy:", np.mean(rf_cv_scores))
```

```
Mean CV Accuracy: 0.8627403846153847
```

Prediction Evaluation for Random Forest Model: Measures the accuracy, precision, recall, and F1-score of the Random Forest model.

In [17]:
```python
# Predictions and evaluation for Random Forest
y_pred_rf = rf_model.predict(X_test)
print("\nRandom Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00        15
           2       0.88      0.99      0.93       910
           3       0.20      0.02      0.04        83
           4       0.17      0.03      0.05        32

    accuracy                           0.87      1041
   macro avg       0.25      0.21      0.21      1041
weighted avg       0.79      0.87      0.82      1041
```

```
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [18]:
```python
#Accuracy Score
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Accuracy: 0.8655139289145053
```

In [19]:
```python
#Precision Score
print("Precision:", precision_score(y_test, y_pred_rf, average='weighted'))
```

```
Precision: 0.789918844828565
```

```
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.
0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [20]:
```python
#Recall Score
print("Recall:", recall_score(y_test, y_pred_rf, average='weighted'))
```

```
Recall: 0.8655139289145053
```

In [21]:
```python
# F1 Score
print("F1 Score:", f1_score(y_test, y_pred_rf, average='weighted'))
```

```
F1 Score: 0.8180924274063786
```

Gradient Boosting Model:

After evaluating the performance of the Random Forest model, we proceed with another Machine Learning Model which is the Gradient Boosting model. The Gradient Boosting will correct errors iteratively for improved predictive accuracy. By using the second model for the same Car Crash Chicago in the 2022 dataset, we can compare the prediction (Most Severe Injuries).

In [22]:

```
# Gradient Boosting Model
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)
```

Out[22]:

```
▼ GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

Cross-validation: Here, we are going to improve the model's performance and stability using Cross-Validation technique for the Gradient Boosting model.

In [23]:

```
# Cross-validation for Gradient Boosting
gb_cv_scores = cross_val_score(gb_model, X_train, y_train, cv=cv, scoring='a
print("\nGradient Boosting Cross-Validation Accuracy Scores:", gb_cv_scores)
```

```
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\model_selection\_split.p
y:700: UserWarning: The least populated class in y has only 3 members, which
is less than n_splits=5.
  warnings.warn(
Gradient Boosting Cross-Validation Accuracy Scores: [0.87139423 0.86778846
0.87139423 0.87019231 0.87019231]
```

In [24]:

```
#To check the mean value for the CV accuracy
print("Mean CV Accuracy:", np.mean(gb_cv_scores))
```

```
Mean CV Accuracy: 0.8701923076923077
```

Prediction Evaluation for Gradient Boosting Model: Measures the accuracy, precision, recall, and F1-score of the Gradient Boosting model.

In [26]:

```
# Predictions and evaluation for Gradient Boosting
y_pred_gb = gb_model.predict(X_test)
print("\nGradient Boosting Classification Report:")
print(classification_report(y_test, y_pred_gb))
```

```
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00        15
           2       0.87      1.00      0.93       910
           3       0.00      0.00      0.00        83
           4       0.00      0.00      0.00        32

    accuracy                           0.87      1041
   macro avg       0.17      0.20      0.19      1041
weighted avg       0.76      0.87      0.81      1041
```

```
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [27]:

```
#Accuracy Score
print("Accuracy:", accuracy_score(y_test, y_pred_gb))
```

```
Accuracy: 0.8712776176753122
```

In [28]:

```
#Precision Score
print("Precision:", precision_score(y_test, y_pred_gb, average='weighted'))
```

```
Precision: 0.7645734157035043
```

```
C:\Users\mercy\anaconda3\Lib\site-packages\sklearn\metrics\_classification.p
y:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.
0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [29]:

```
#Recall Score
print("Recall:", recall_score(y_test, y_pred_gb, average='weighted'))
```

```
Recall: 0.8712776176753122
```

In [30]:

```
#F1 Score
print("F1 Score:", f1_score(y_test, y_pred_gb, average='weighted'))
```

```
F1 Score: 0.8144454361423052
```

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.

In [ ]:

In [ ]:

In [ ]:

Loading [MathJax]/extensions/Safe.js

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.