

Introduction to Data Mining

# SEVERITY PREDICTION - US TRAFFIC ACCIDENTS

---



**Submitted by:**

Sumaiya Chinoy - 18650

Samra Noman - 19645

Muhammad Mujtaba Khan - 18624

Muhammad Taha Waseem - 19743

---

---

## Contents

Problem Description .....	4
Data Description.....	4
Data Attributes .....	5
Traffic Attributes: .....	5
Address Attributes:.....	6
Weather Attributes: .....	6
Point-Of-Interest (POI) Attributes: .....	7
Period of Day Attributes: .....	7
Classification of Data Attributes .....	8
Importing Libraries .....	10
Reading and Loading Data on Google Colab .....	10
Missing Value of Data Attributes: .....	11
Class Proportion in Data Set .....	12
Count of Each Class: .....	12
Percentage of Each Class: .....	12
Visualization of Class Distribution: .....	13
Data Pre Processing .....	14
Dropping ineffectual features.....	14
Removing categorical features with same values .....	14

---

Removing features with 60% missing values .....	16
Dealing with chaotic categorical features .....	17
Checking 'Wind_Direction' column values: .....	17
Cleaning wind direction data: .....	17
Checking 'Weather_Condition' column values: .....	18
Cleaning wind direction data: .....	18
Dropping similar features .....	19
Breaking down 'Start_Time' attribute .....	19
Counting null values .....	20
Dropping rows with missing values .....	20
Dealing with null values in categorical columns .....	20
Dealing with missing values in numerical columns .....	21
End of preprocessing .....	22
Data modeling begins! .....	23
Random Forest Classifier: .....	24
Naive Bayes Classifier: .....	26
Gradient Boosted Trees Classifier: .....	28
Decision Trees Classifier: .....	30
Tree Ensemble Classifier: .....	32
Summary .....	34

---

---

Findings.....	35
Key insights: .....	35
Recommendations for Data collection: .....	36
Model expiration .....	37

## Problem Description

Traffic Accidents are very common and can be a huge hindrance to the incoming and outgoing traffic. There are approximately forty to fifty thousand road accidents every year in the US. If it can be identified how brutal an accident is, it can help to clear the affected area and keep the traffic moving. This way, it will open the traffic flow for the citizens as well as create simplicity for the accident dealing forces.

The problem is such that we need to identify how severe an accident is ranging from 1 to 4, with 1 being least severe and 4 being most severe. A least severe accident means that the delay on the traffic flow will be shorter while the most severe means longer delay on traffic. If we can correctly classify the severity of an accident, we will be able to use proper measures to deal with the different types of accident and save time.

To do so, the data mining technique we have chosen to implement is a ‘Multiclass Classification Problem’ as according to the given dataset we will determine which severity level will our accident be classified as. This implementation will help the law enforcing agencies as well as the traffic controlling agencies speed up their process of dealing with these accidents in an efficient manner without the entire traffic being put on hold.

## Data Description

The data contains countrywide accident info spanning over the 49 states of the United States. The data is constantly being gathered from February 2016, using several data providers, including different APIs that give streaming traffic data. The APIs were captured by different bodies such as US and state departments of transportation, law enforcement agencies, traffic cameras, and traffic sensors on the roads. As of now, the data contains 1.5 million accident records in this dataset collected from February 2016 to December 2020.

---

The data provided stated that each data source has its own metric of how 'severe' the number 4 is for e.g. (in the severity column). The ideal scenario would be to filter the data, source wise and then train the model accordingly

## Data Attributes

The data attributes are categorized into 5 types that includes Traffic Attributes, Address Attributes, Weather Attributes, POI Attributes and Period of Day Attributes.

### Traffic Attributes:

These attributes signify the impact of accidents on traffic flow.

Attribute	Description
ID	Unique Identifier of the accident record
Severity	Indicates the accident severity, ranked using number between 1 and 4, where 1 indicates the least impact on traffic (short delay) and 4 indicates a significant impact on traffic (longer delay)
Start_Time	Time at which accident was started in local time zone
End_Time	Time at which the accident ended in the local time zone. The accident is considered to end when its impact on traffic flow is diminished
Start_Lat	GPS coordinate latitude of the start point
Start_Lng	GPS coordinate longitude of the start point
End_Lat	GPS coordinate latitude of the end point
End_Lng	GPS coordinate longitude of the end point
Distance	Road length affected by the accident. The unit of measurement is in miles(mi).
Description	Description of the accident

---

### Address Attributes:

These attributes are the breakdown of address location where the accident took place.

Attribute	Description
Number	Street number in the address field
Street	Street name in the address field
Side	Relative Side of the street in address field. It can have two possible values Right or Left
City	City in the address field
Country	Country in the address field
State	State in the address field
ZipCode	ZipCode in the address field
Country	Country in the address field
Timezone	Timezone based on accident address location

### Weather Attributes:

These attributes refer to the weather related data recorded at the time of accident.

Attribute	Description
Airport Code	Nearest airport-based weather station from the accident location
Weather TimeStamp	Time-stamp of weather observation record in local time zone
Temperature	Temperature observed. Unit of measurement is in Farenheit(F)
Wind Chill	Wind Chill observed. Unit of measurement is in Farenheit(F)
Humidity	Humidity observed. Unit of measurement is in percentage(%)
Pressure	Air Pressure observed. Unit of measurement is in inches(in)
Visibility	Visibility observed. Unit of measurement is in miles(mi)
Wind Direction	Wind direction observed
Wind Speed	Wind Speed observed. Unit of measurement is in miles per hour(mph)
Precipitation	Precipitation if observed. Unit of measurement in inches(in)
Weather Condition	Weather condition observed during accident time

---

### Point-Of-Interest (POI) Attributes:

POI annotation tags are given in the form of following attributes to add details in accordance to the surroundings of the place where the accident took place.

Attributes	Description
Amenity	Indicates the presence of amenity (a particular place such as restaurant, library etc) in a nearby location
Bump	Indicates the presence of speed bump or hump to reduce speed in a nearby location
Crossing	Indicates the presence of road crossing for pedestrians or cyclists in a nearby location
Give Way	Indicates the presence of give-way (road sign indicating priority passing) in a nearby location
Junction	Indicates the presence of junction (highway ramp, exit or entrance) in a nearby location
No Exit	Indicates the presence of no-exit (road sign indicating no further possibility of traveling further) in a nearby location
Railway	Indicates the presence of railways in a nearby location
Roundabout	Indicates the presence of circular round junction in a nearby location
Station	Indicates the presence of any public transportation status in a nearby location
Stop	Indicates the presence of stop sign in a nearby location
Traffic Calming	Indicates the presence of any sign of slowing down traffic speed in a nearby location
Traffic Signal	Indicates the presence of traffic signal on intersections in a nearby location
Turning Loop	Indicates the presence of a widened area of a highway with a non-traversable island for turning around

### Period of Day Attributes:

These attributes include four different daylight systems and they are used to assign labels (day or night) to these systems. These systems are defined based on the position of the sun with respect to the horizon.

Attribute	Description
-----------	-------------

---

Sunrise Sunset	Period of day based on sunrise/sunset
Civil Twilight	Period of day based on civil twilight
Nautical Twilight	Period of day based on nautical twilight
Astronomical Twilight	Period of day based on astronomical twilight

## Classification of Data Attributes

### Categorical:

1. ID
2. Start\_Time
3. End\_Time
4. Description
5. Timezone
6. Severity
7. Street
8. City
9. County
10. State
11. ZipCode
12. Country
13. Timezone
14. Airport\_Code
15. Weather\_Timestamp
16. Wind\_Direction
17. Weather\_Condition

### Binary:

1. Amenity
2. Bump
3. Crossing
4. Give-way
5. Junction
6. No-exit



- 
7. Railway
  8. Roundabout
  9. Station
  10. Stop
  11. Traffic Calming
  12. Traffic Signal
  13. Turning Loop
  14. Sunrise/Sunset
  15. Civil Twilight
  16. Nautical Twilight
  17. Astronomical Twilight
  18. Side

**Numerical:**

1. Start\_Lat
2. Start\_Lng
3. End\_Lat
4. End\_Lng
5. Distance
6. Temperature
7. Wind\_Chill
8. Humidity
9. Pressure
10. Visibility
11. Wind\_Speed
12. Precipitation

---

## Importing Libraries

Several libraries were loaded on the google colab python notebook to be used for further processing.

```
import numpy as np
import pandas as pd
import json
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm
from datetime import datetime
import glob
import seaborn as sns
import re
import os
import io
```

## Reading and Loading Data on Google Colab

The data was loaded on google collab and saved in dataframe(df) to be accessed by all group members.

```
path = "/content/drive/MyDrive/IDM Dataset/data.csv"
df = pd.read_csv(path)
```

---

## Missing Value of Data Attributes:

We first showed the missing percentage of every column in data before data preprocessing.

```
# Showing missing percentage of every column in data before data preprocessing
missing = pd.DataFrame(df.isnull().sum()).reset_index()
missing.columns = ['Feature', 'Missing_Percent(%)']

missing['Missing_Percent(%)'] = missing['Missing_Percent(%)'].apply(lambda x: x / df.shape[0] * 100)
missing
```

The results were as shown below

	Feature	Missing_Percent(%)
0	ID	0.000000
1	Severity	0.000000
2	Start_Time	0.000000
3	End_Time	0.000000
4	Start_Lat	0.000000
5	Start_Lng	0.000000
6	End_Lat	0.000000
7	End_Lng	0.000000
8	Distance(mi)	0.000000
9	Description	0.000000
10	Number	69.000715
11	Street	0.000000
12	Side	0.000000
13	City	0.005475
14	County	0.000000
15	State	0.000000
16	Zipcode	0.061673
17	Country	0.000000
18	Timezone	0.151841
19	Airport_Code	0.280199
20	Weather_Timestamp	1.996222
21	Temperature(F)	2.838469
22	Wind_Chill(F)	29.637007
23	Humidity(%)	3.001786
24	Pressure(in)	2.392643
25	Visibility(mi)	2.916170
26	Wind_Direction	2.760965
27	Wind_Speed(mph)	8.499773
28	Precipitation(in)	33.675953
29	Weather_Condition	2.902714
30	Amenity	0.000000
31	Bump	0.000000
32	Crossing	0.000000
33	Give_Way	0.000000
34	Junction	0.000000
35	No_Exit	0.000000
36	Railway	0.000000
37	Roundabout	0.000000
38	Station	0.000000
39	Stop	0.000000
40	Traffic_Calming	0.000000
41	Traffic_Signal	0.000000
42	Turning_Loop	0.000000
43	Sunrise_Sunset	0.005475
44	Civil_Twilight	0.005475
45	Nautical_Twilight	0.005475
46	Astronomical_Twilight	0.005475

---

## Class Proportion in Data Set

In this classification problem, we are categorizing the accidents according to severity. The 'severity' is a categorical variable having 4 categories. The severity is mapped from number 1 to 4, where 1 indicates the least impact on traffic and 4 indicates the most impact on traffic.

Following are the four categories of severity:

1. Severity = 1
2. Severity = 2
3. Severity = 3
4. Severity = 4

Now we analyze the dataset by checking how many records of every specific category is present in the dataset.

### Count of Each Class:

```
print("Severity 1: ",(df["Severity"]==1).sum())
print("Severity 2: ",(df["Severity"]==2).sum())
print("Severity 3: ",(df["Severity"]==3).sum())
print("Severity 4: ",(df["Severity"]==4).sum())
```

```
Severity 1: 28178
Severity 2: 1212382
Severity 3: 161052
Severity 4: 114452
```

### Percentage of Each Class:

```
total = len(df.index)-1
print("Severity 1: ",((df["Severity"]==1).sum()/total)*100)
print("Severity 2: ",((df["Severity"]==2).sum()/total)*100)
print("Severity 3: ",((df["Severity"]==3).sum()/total)*100)
print("Severity 4: ",((df["Severity"]==4).sum()/total)*100)
```

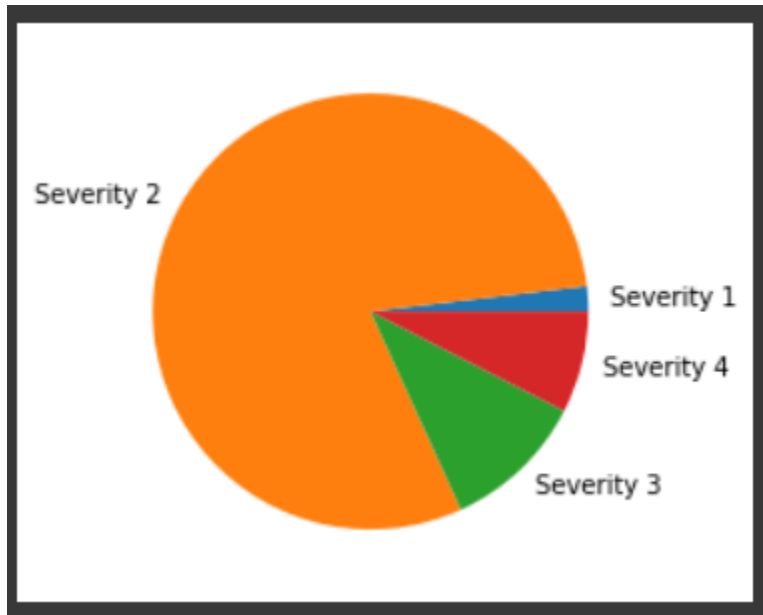
```
Severity 1: 1.8586298854335208
Severity 2: 79.96910418630361
Severity 3: 10.623041390760147
Severity 4: 7.549290497822319
```

---

## Visualization of Class Distribution:

```
y = np.array([(df["Severity"]==1).sum()/total*100, (df["Severity"]==2).sum()/total*100, (df["Severity"]==3).sum()/total*100, (df["Severity"]==4).sum()/total*100])
mylabels = ["Severity 1", "Severity 2", "Severity 3", "Severity 4"]

plt.pie(y, labels = mylabels)
plt.show()
```



Thus, this tells us that this is an imbalance classification problem and thus there are two groups of metrics that will be useful for predicting model performance as they focus on one class are sensitivity-specificity(ROC Curve) and precision-recall .

---

## Data Pre Processing

### Dropping ineffectual features

Firstly, we drop columns that don't contribute towards accuracy. 'ID' is dropped because it isn't a predictor at all and 'Description' was dropped because it has the same features as POI (Point of interest) column in the dataset and so 'Description' is also just another redundant feature. 'Distance(mi)', 'End\_Time', 'End\_Lat', 'End\_Lng' are dropped because the exact values of the features can't be known before the accident and can only be known once the accident has occurred.

Hence we drop the following features:

1. 'ID'
2. 'Description'
3. 'Distance(mi)'
4. 'End\_Time'
5. 'End\_Lat'
6. 'End\_Lng'

```
[ ] # Removing columns which are not required
df = df.drop(['ID','Description','Distance(mi)', 'End_Time', 'End_Lat', 'End_Lng'], axis=1)
```

Originally there were 47 columns, out of which 6 were removed and the dataframe shape was:

```
df.shape
(1516064, 41)
```

### Removing categorical features with same values

We dropped some of the categorical features that had the same values throughout the data. A for loop was made to iterate over each value of all columns, and if the count of unique values in a particular column was equal to 1 that meant there was a single value throughout the column and that column was redundant for the model.

```
# Removing categorical features who have a same values throughout the data
cat_features = ['Side', 'Country', 'Timezone', 'Amenity', 'Bump', 'Crossing',
                'Give_Way', 'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station',
                'Stop', 'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop', 'Sunrise_Sunset',
                'Civil_Twilight', 'Nautical_Twilight', 'Astronomical_Twilight']

for i in cat_features:
    count = df[i].unique().size
    if (count == 1):
        print('Feature :', i)
        df = df.drop(i, axis=1)
```

After iterating over all these columns listed below, we found out that there were two columns, 'Country' and 'Turning\_Loop' which have the same value throughout and so 'Country' and 'Turning\_Loop' were dropped out from the dataframe.

Columns iterated: 'Side', 'Country', 'Time Zone', 'Amenity', 'Bump', 'Crossing', 'Give\_Way', 'Junction', 'No\_Exit', 'Railway', 'Roundabout', 'Station', 'Stop', 'Traffic\_Calming', 'Traffic\_Signal', 'Turning\_Loop', 'Sunrise\_Sunset', 'Civil\_Twilight', 'Nautical\_Twilight', 'Astronomical\_Twilight'.

Columns dropped:

1. 'Country'
2. 'Turning\_Loop'

And so we have 39 columns remaining in the dataframe.

```
# Removing categorical features who have a same values throughout the data
cat_features = ['Side', 'Country', 'Timezone', 'Amenity', 'Bump', 'Crossing',
                'Give_Way', 'Junction', 'No_Exit', 'Railway', 'Roundabout', 'Station',
                'Stop', 'Traffic_Calming', 'Traffic_Signal', 'Turning_Loop', 'Sunrise_Sunset',
                'Civil_Twilight', 'Nautical_Twilight', 'Astronomical_Twilight']

for i in cat_features:
    count = df[i].unique().size
    if (count == 1):
        print('Feature :', i)
        df = df.drop(i, axis=1)

df.shape

Feature : Country
Feature : Turning_Loop
(1516064, 39)
```

---

## Removing features with 60% missing values

There are missing values in the dataset. So to analyze columns with missing values and remove null/missing values from the dataset, we made a new dataframe called 'missing', added two columns 'Feature' and 'Missing\_Percent(%)' that represent the feature along with the percentage of missing values the feature has.

```
# Removing feature which has more than 60 % missing value
missing = pd.DataFrame(df.isnull().sum()).reset_index()
missing.columns = ['Feature', 'Missing_Percent(%)']

missing['Missing_Percent(%)'] = missing['Missing_Percent(%)'].apply(lambda x: x / df.shape[0] * 100)

for i in missing.iterrows():
    if (i[1]['Missing_Percent(%)'] > 60 ):
        print(i[1]['Feature'], ': ', i[1]['Missing_Percent(%)'] , ' %')
        df = df.drop(i[1]['Feature'], axis=1)
```

The output of this code snippet shows us that the 'Number' feature had 69% missing values. 'Number' feature was dropped and so the dataframe has 38 features remaining now.

```
# Removing feature which has more than 60 % missing value
missing = pd.DataFrame(df.isnull().sum()).reset_index()
missing.columns = ['Feature', 'Missing_Percent(%)']

missing['Missing_Percent(%)'] = missing['Missing_Percent(%)'].apply(lambda x: x / df.shape[0] * 100)

for i in missing.iterrows():
    if (i[1]['Missing_Percent(%)'] > 60 ):
        print(i[1]['Feature'], ': ', i[1]['Missing_Percent(%)'] , ' %')
        df = df.drop(i[1]['Feature'], axis=1)

df.shape

Number : 69.00071500939275 %
(1516064, 38)
```



---

## Dealing with chaotic categorical features

### Checking 'Wind\_Direction' column values:

```
# Checking wind direction data
print("Wind Direction: ", df['Wind_Direction'].unique())

Wind Direction:  ['SW' 'Calm' 'WSW' 'WNW' 'West' 'NNW' 'South' 'W' 'NW' 'North' 'SSE' 'SSW'
'ESE' 'SE' nan 'East' 'Variable' 'NNE' 'NE' 'ENE' 'CALM' 'S' 'VAR' 'N'
'E']
```

'Wind\_Direction' and 'Weather\_Condition' columns have a plethora of values. However, if we map values of 'Wind\_Direction' and 'Weather\_Condition' to some limited values, that will make the data simpler and easier to understand and interpret.

### Cleaning wind direction data:

```
# Cleaning the wind direction data to simple words
df.loc[df['Wind_Direction']=='Calm', 'Wind_Direction'] = 'CALM'
df.loc[(df['Wind_Direction']=='West')|(df['Wind_Direction']=='WSW')|(df['Wind_Direction']=='WNW'), 'Wind_Direction'] = 'W'
df.loc[(df['Wind_Direction']=='South')|(df['Wind_Direction']=='SSW')|(df['Wind_Direction']=='SSE'), 'Wind_Direction'] = 'S'
df.loc[(df['Wind_Direction']=='North')|(df['Wind_Direction']=='NNW')|(df['Wind_Direction']=='NNE'), 'Wind_Direction'] = 'N'
df.loc[(df['Wind_Direction']=='East')|(df['Wind_Direction']=='ESE')|(df['Wind_Direction']=='ENE'), 'Wind_Direction'] = 'E'
df.loc[df['Wind_Direction']=='Variable', 'Wind_Direction'] = 'VAR'
print("Wind Direction after cleaning: ", df['Wind_Direction'].unique())

Wind Direction after cleaning:  ['SW' 'CALM' 'W' 'N' 'S' 'NW' 'E' 'SE' nan 'VAR' 'NE']
```

The above code snippet maps 'WSW', 'WNW' and 'West' to 'W' since they all represent wind direction towards the West, 'SSW', 'SSE' and 'South' to 'S' since they all represent wind direction towards the South, 'NNW', 'NNE' and 'North' to 'N' since they all represent wind direction towards the North, 'ESE', 'ENE' and 'East' to 'E' since they all represent wind direction towards the East. It also maps 'Calm' to 'CALM' and 'Variable' to 'VAR' for consistency and improved readability.

---

## Checking 'Weather\_Condition' column values:

```
# Checking weather condition data
print("Weather Condition: ", df['Weather_Condition'].unique())

Weather Condition: ['Light Rain' 'Overcast' 'Mostly Cloudy' 'Snow' 'Light Snow' 'Cloudy' nan
'Scattered Clouds' 'Clear' 'Partly Cloudy' 'Light Freezing Drizzle'
'Light Drizzle' 'Haze' 'Rain' 'Heavy Rain' 'Fair' 'Drizzle' 'Fog'
'Thunderstorms and Rain' 'Patches of Fog' 'Light Thunderstorms and Rain'
'Mist' 'Rain Showers' 'Light Rain Showers' 'Heavy Drizzle' 'Smoke'
'Light Freezing Fog' 'Light Freezing Rain' 'Blowing Snow'
'Heavy Thunderstorms and Rain' 'Heavy Snow' 'Snow Grains' 'Squalls'
'Light Fog' 'Shallow Fog' 'Thunderstorm' 'Light Ice Pellets' 'Thunder'
'Thunder in the Vicinity' 'Fair / Windy' 'Light Rain with Thunder'
'Heavy Thunderstorms and Snow' 'Light Snow Showers' 'Cloudy / Windy'
'Ice Pellets' 'N/A Precipitation' 'Light Thunderstorms and Snow'
'T-Storm' 'Rain / Windy' 'Wintry Mix' 'Partly Cloudy / Windy'
'Heavy T-Storm' 'Sand' 'Light Rain / Windy' 'Widespread Dust'
'Mostly Cloudy / Windy' 'Blowing Dust / Windy' 'Blowing Dust'
'Volcanic Ash' 'Freezing Rain / Windy' 'Small Hail' 'Wintry Mix / Windy'
'Light Snow / Windy' 'Heavy Ice Pellets' 'Heavy Snow / Windy'
'Heavy Rain / Windy' 'Heavy T-Storm / Windy' 'Fog / Windy' 'Dust Whirls'
'Showers in the Vicinity' 'Funnel Cloud' 'Haze / Windy'
'Light Rain Shower' 'Smoke / Windy' 'Light Drizzle / Windy'
'Snow / Windy' 'Partial Fog' 'Light Freezing Rain / Windy' 'Sleet'
'Blowing Snow / Windy' 'Snow and Sleet' 'Snow and Sleet / Windy'
'Squalls / Windy' 'Light Sleet / Windy' 'Freezing Drizzle'
'Freezing Rain' 'Thunder / Windy' 'Drizzle and Fog' 'Light Sleet'
'Thunder / Wintry Mix / Windy' 'Mist / Windy' 'Light Snow and Sleet'
'Rain Shower' 'Sleet / Windy' 'T-Storm / Windy'
'Light Snow and Sleet / Windy' 'Patches of Fog / Windy' 'Drizzle / Windy'
'Sand / Dust Whirls Nearby' 'Light Rain Shower / Windy'
'Heavy Rain Shower' 'Thunder and Hail' 'Drifting Snow'
'Light Snow Shower' 'Sand / Dust Whirlwinds' 'Heavy Blowing Snow' 'Hail'
'Heavy Freezing Drizzle' 'Low Drifting Snow' 'Light Blowing Snow']
```

## Cleaning wind direction data:

```
# Cleaning the weather condition data to simple words

df.loc[(df['Weather_Condition'].str.contains("Clear")==True)|(df['Weather_Condition'].str.contains("Fair")==True), 'Weather_Condition'] = 'Clear'
df.loc[(df['Weather_Condition'].str.contains("Cloud")==True)|(df['Weather_Condition'].str.contains("Overcast")==True), 'Weather_Condition'] = 'Cloud'
df.loc[(df['Weather_Condition'].str.contains("Fog")==True)|(df['Weather_Condition'].str.contains("Haze")==True)|(df['Weather_Condition'].str.contains("Mist")==True), 'Weather_Condition'] = 'Fog'
df.loc[(df['Weather_Condition'].str.contains("Ice")==True)|(df['Weather_Condition'].str.contains("Snow")==True)|(df['Weather_Condition'].str.contains("Hail")==True)|(df['Weather_Condition'].str.contains("Sleet")==True), 'Weather_Condition'] = 'Snow'
df.loc[(df['Weather_Condition'].str.contains("storm")==True)|(df['Weather_Condition'].str.contains("Thunder")==True)|(df['Weather_Condition'].str.contains("Tornado")==True)|(df['Weather_Condition'].str.contains("T-Storm")==True)|(df['Weather_Condition'].str.contains("Squalls")==True), 'Weather_Condition'] = 'Thunderstorm'
df.loc[(df['Weather_Condition'].str.contains("Rain")==True)|(df['Weather_Condition'].str.contains("Showers")==True)|(df['Weather_Condition'].str.contains("Drizzle")==True), 'Weather_Condition'] = 'Rain'
df.loc[(df['Weather_Condition'].str.contains("Sand")==True)|(df['Weather_Condition'].str.contains("Dust")==True), 'Weather_Condition'] = 'Dust'

print("Weather Condition after cleaning: ", df['Weather_Condition'].unique())

Weather Condition after cleaning: ['Rain' 'Cloud' 'Snow' nan 'Clear' 'Fog' 'Thunderstorm' 'Smoke'
'N/A Precipitation' 'Wintry Mix' 'Dust' 'Volcanic Ash'
'Wintry Mix / Windy' 'Smoke / Windy']
```

The above code snippet maps 'Weather\_Condition' values to a simpler and lesser number of values. The 'Weather\_Condition' variable has short sentences that can be mapped to simpler words.

If a sentence has 'Clear' and/or 'Fair', that sentence will be mapped to 'Clear'.

---

If a sentence has 'Overcast' and/or 'Cloud', that sentence will be mapped to 'Cloud'.

If a sentence has 'Haze' and/or 'Mist' and/or 'Fog', that sentence will be mapped to 'Fog'.

If a sentence has 'Ice' and/or 'Snow' and/or 'Hail' and/or 'Sleet', that sentence will be mapped to 'Snow'.

If a sentence has 'storm' and/or 'Thunder' and/or 'Tornado' and/or 'T-Storm' and/or 'Squalls' that sentence will be mapped to 'Thunderstorm'.

If a sentence has 'Rain' and/or 'Showers' and/or 'Drizzle', that sentence will be mapped to 'Rain'.

If a sentence has 'Sand' and/or 'Dust', that sentence will be mapped to 'Dust'.

## Dropping similar features

Since the 'Weather\_Timestamp' is almost the same as 'Start\_Time', we just keep 'Start\_Time' and drop 'Weather\_Timestamp'. We have 37 columns remaining now.

```
# Dropping weather timestamp as it is very similar to start time
df = df.drop(['Weather_Timestamp'], axis=1)
df.shape

(1516064, 37)
```

## Breaking down 'Start\_Time' attribute

Next we divide the 'Start\_Time' to 'Year', 'Month', 'Weekday', 'Day' (in a year), 'Hour', and 'Minute' (in a day). We have 42 columns now.

```
# Dividing start time into year, month, day, hour, minute in data set

df['Start_Time'] = pd.to_datetime(df['Start_Time'], format='%Y/%m/%d %H:%M:%S')
df['Year'] = df['Start_Time'].dt.year
df['Month'] = df['Start_Time'].dt.month
df['Day'] = df['Start_Time'].dt.day
df['Hour'] = df['Start_Time'].dt.hour
df['Minute'] = (df['Hour']*60.0) + df["Start_Time"].dt.minute
```

---

## Counting null values

The following code snippet was executed to see the count of all the null values in every feature to decide how to preprocess the data.

```
[ ] # Null values
    df.isnull().sum()

    for i in df:
        if (df[i].isnull().sum() > 0):
            print(i,":",df[i].isnull().sum())
```

```
City : 83
Zipcode : 935
Timezone : 2302
Airport_Code : 4248
Temperature(F) : 43033
Wind_Chill(F) : 449316
Humidity(%) : 45509
Pressure(in) : 36274
Visibility(mi) : 44211
Wind_Direction : 41858
Wind_Speed(mph) : 128862
Precipitation(in) : 510549
Weather_Condition : 44007
Sunrise_Sunset : 83
Civil_Twilight : 83
Nautical_Twilight : 83
Astronomical_Twilight : 83
```

## Dropping rows with missing values

Some of the rows have very low missing values. In fact, if we sum all the missing values from these columns they will form a very small portion of the whole dataset. Hence since the impact these values have on the accuracy is negligible, we drop those rows instead of filling the missing values by average or most occurring value.

```
# Dropping those column data who has very low amount of missing values
df = df.dropna(subset=[['City','Zipcode','Timezone','Airport_Code','Sunrise_Sunset',
                        'Civil_Twilight','Nautical_Twilight','Astronomical_Twilight']])
```

## Dealing with null values in categorical columns

To fill in missing values of 'Weather\_Condition' and 'Wind\_Direction' (both of them are categorical features), we applied a groupby function and grouped the data on 'Airport\_Code' and 'Month' and filled

---

in the missing values of 'Weather\_Condition' and 'Wind\_Direction' with the most common values of 'Airport\_Code' and 'Month'.

However if some rows still had missing values, we dropped them as the portion of rows having missing values was very small and did not contribute in improving the accuracy.

```
# Filling missing values of categorical columns

# Grouping data by 'Airport_Code' and 'Month' then filling missing value with most common value
weather_cat = ['Wind_Direction'] + ['Weather_Condition']
print("The number of remaining missing values that could not be replaced hence dropped: ")
for i in weather_cat:
    df[i] = df.groupby(['Airport_Code', 'Month'])[i].apply(lambda x: x.fillna(Counter(x).most_common()[0][0]) if all(x.isnull())==False else x)
    print(i + " : " + df[i].isnull().sum().astype(str))

# Remaining null amount of data is very small hence we drop that data
df = df.dropna(subset=weather_cat)
print(weather_cat)

The number of remaining missing values that could not be replaced hence dropped:
Wind_Direction : 21357
Weather_Condition : 22697
['Wind_Direction', 'Weather_Condition']
```

## Dealing with missing values in numerical columns

To fill in missing values of 'Temperature(F)', 'Wind\_Chill(F)', 'Humidity(%)', 'Pressure(in)', 'Visibility(mi)', 'Wind\_Speed(mph)', (all of them are numerical features), we applied a groupby function and grouped the data on 'Airport\_Code' and 'Month' and filled in the missing values of all the aforementioned numerical columns with the mean of 'Airport\_Code' and 'Month'.

However if some rows still had missing values, we dropped them as the portion of rows having missing values was very small and did not contribute to improving the accuracy.

```
# Filling missing values of numerical columns

# Grouping data by 'Airport_Code' and 'Month' then filling missing value with mean value
Weather_data=['Temperature(F)', 'Wind_Chill(F)', 'Humidity(%)', 'Pressure(in)', 'Visibility(mi)', 'Wind_Speed(mph)']
print("The number of remaining missing values that could not be replaced hence dropped: ")
for i in Weather_data:
    df[i] = df.groupby(['Airport_Code', 'Month'])[i].apply(lambda x: x.fillna(x.mean()))
    print( i + " : " + df[i].isnull().sum().astype(str))

# Remaining null amount of data is very small hence we drop that data
df = df.dropna(subset=Weather_data)

The number of remaining missing values that could not be replaced hence dropped:
Temperature(F) : 41
Wind_Chill(F) : 11088
Humidity(%) : 54
Pressure(in) : 23
Visibility(mi) : 145
Wind_Speed(mph) : 509
```

---

Finally, we will fill in the missing values in ‘Precipitation(in)’ feature by taking out the mean of all non-missing values of ‘Precipitation(in)’. We don't use the groupby function for ‘Precipitation(in)’ as filling missing values after using groupby of some features may not fill in every missing value and in ‘Precipitation(in)’ there were a lot of missing values remaining after applying groupby.

```
# Filling precipitation missing value with mean value of whole data
# Not using grouping for this column because after grouping there are still many values missing
df['Precipitation(in)'] = df['Precipitation(in)'].fillna(df['Precipitation(in)'].mean())
```

## End of preprocessing

This marks the end of preprocessing! At this stage our data is free from missing values and redundant data. We have 42 columns remaining now.

```
# Our data preprocessing is complete, No missing value rows left in our data either they have been replaced with values or dropped
print("Total Number of missing values in data: ",df.isnull().sum().sum())
print("Our dataset after preprocessing :", df.shape)
```

```
Total Number of missing values in data: 0
Our dataset after preprocessing : (1473539, 42)
```

---

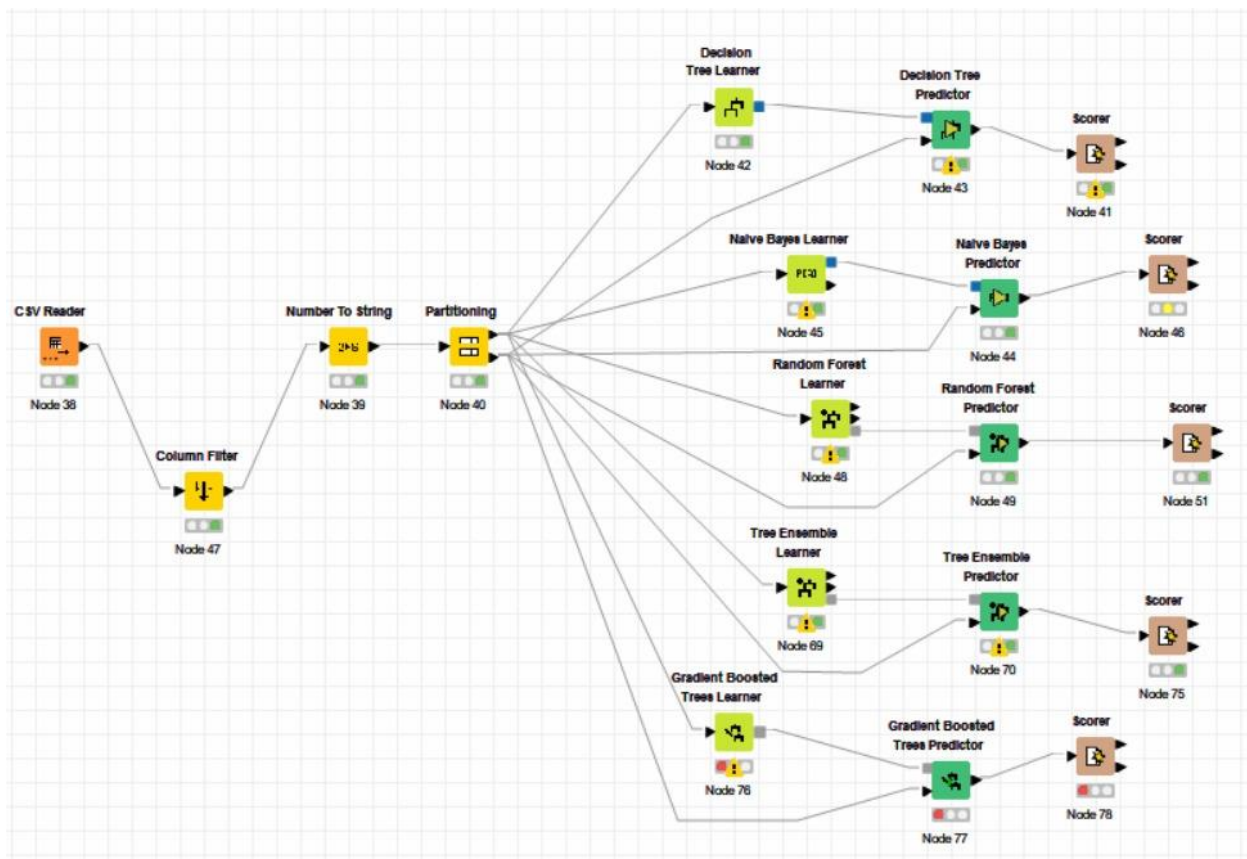
## Data modeling begins!

For Data modeling we switched to KNIME Analytics Platform. KNIME workflow can be easy to understand and is explainable which gives an advantage when it comes to explaining to clients how the data is modeled and how new records are predicted.

We read the preprocessed data in a csv file into KNIME and then applied certain models to the dataset.

We tried 5 different algorithms and tuned their hyperparameters to get the best model that provides the highest accuracy. The 5 algorithms that we tried were:

1. Random Forest
2. Naive Bayes
3. Gradient Boosted trees
4. Ensemble
5. Decision trees



---

Note that for all of the above classifiers, the validation technique used was **holdout sampling**, with the data partitioned into **train and test** into ratios of **70 and 30 respectively**.

## Machine Description

Processor: Intel(R) Core(TM) i7-8550U CPU @ 1.80Ghz (64 bit Operating System)

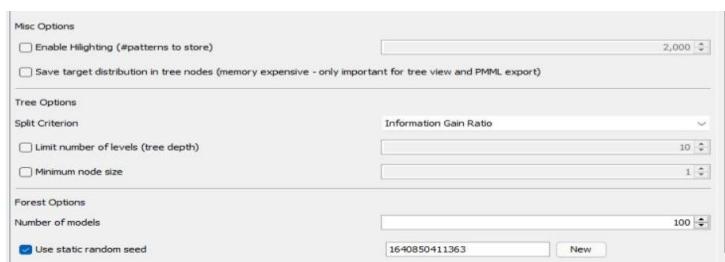
Installed RAM: 16 GB (15.8 GB usable)

Assigned to Knime: 14 GB

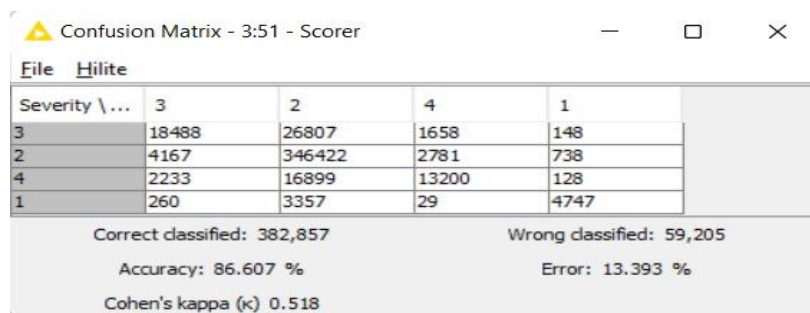
## Random Forest Classifier:

Random Forest Classifier was run with 100 models and gave **86.6% accuracy**. When models were increased to more than 100, the model did not run as KNIME prompted us with an out of memory error. However memory errors are dependent on the machines individuals use and this model was run on a machine having 16GB RAM where 12GB of RAM was allocated only to KNIME. Random Forests with 100 models took 46 minutes. All of its area under curve value for severity was above 0.9 which is one of best amongst all 5 models.

The split criteria for Random Forest was Information Gain Ratio with unlimited tree depth.



Here's a confusion matrix for the Random Forest classifier with 100 models.

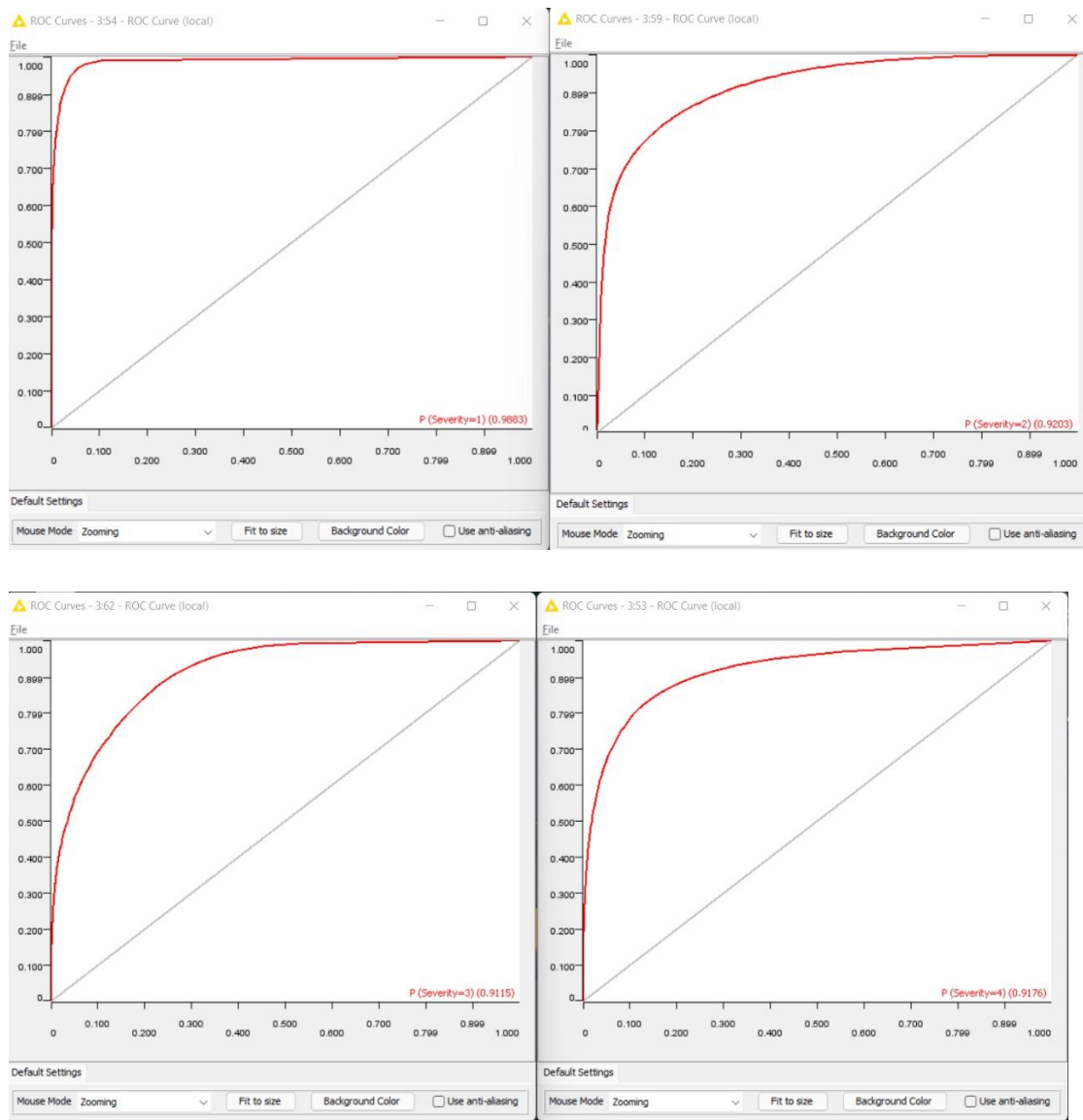


Severity \ ...	3	2	4	1
3	18488	26807	1658	148
2	4167	346422	2781	738
4	2233	16899	13200	128
1	260	3357	29	4747

Correct classified: 382,857      Wrong classified: 59,205  
Accuracy: 86.607 %      Error: 13.393 %  
Cohen's kappa ( $\kappa$ ) 0.518



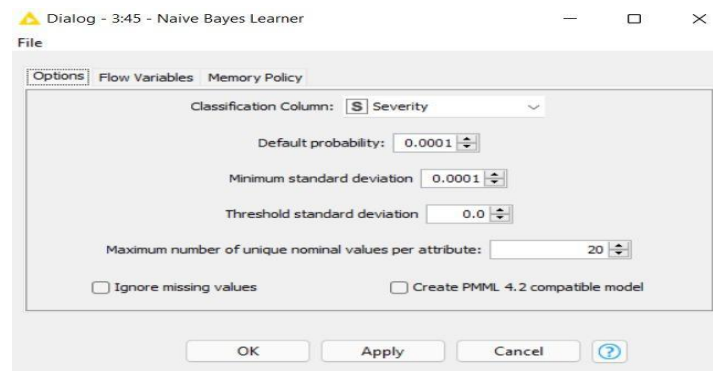
Following are the images of ROC curves for severity 1-4 for Random Forest Classifier. Since it's a multi class classification problem with 4 different classes, there will be 4 different ROC curves for each class.



	Severity 1	Severity 2	Severity 3	Severity 4
<b>Random Forest</b>	0.9883	0.9203	0.9115	0.9176

## Naive Bayes Classifier:

Naive Bayes Classifier predicted severity for new records with 56% accuracy which is quite low compared to other models. However, even with 1.4 million records it was very fast in predicting new records. This model took almost 30-40 seconds to execute. Though its accuracy was very low its area under curve value for all severity was greater than 0.6 .



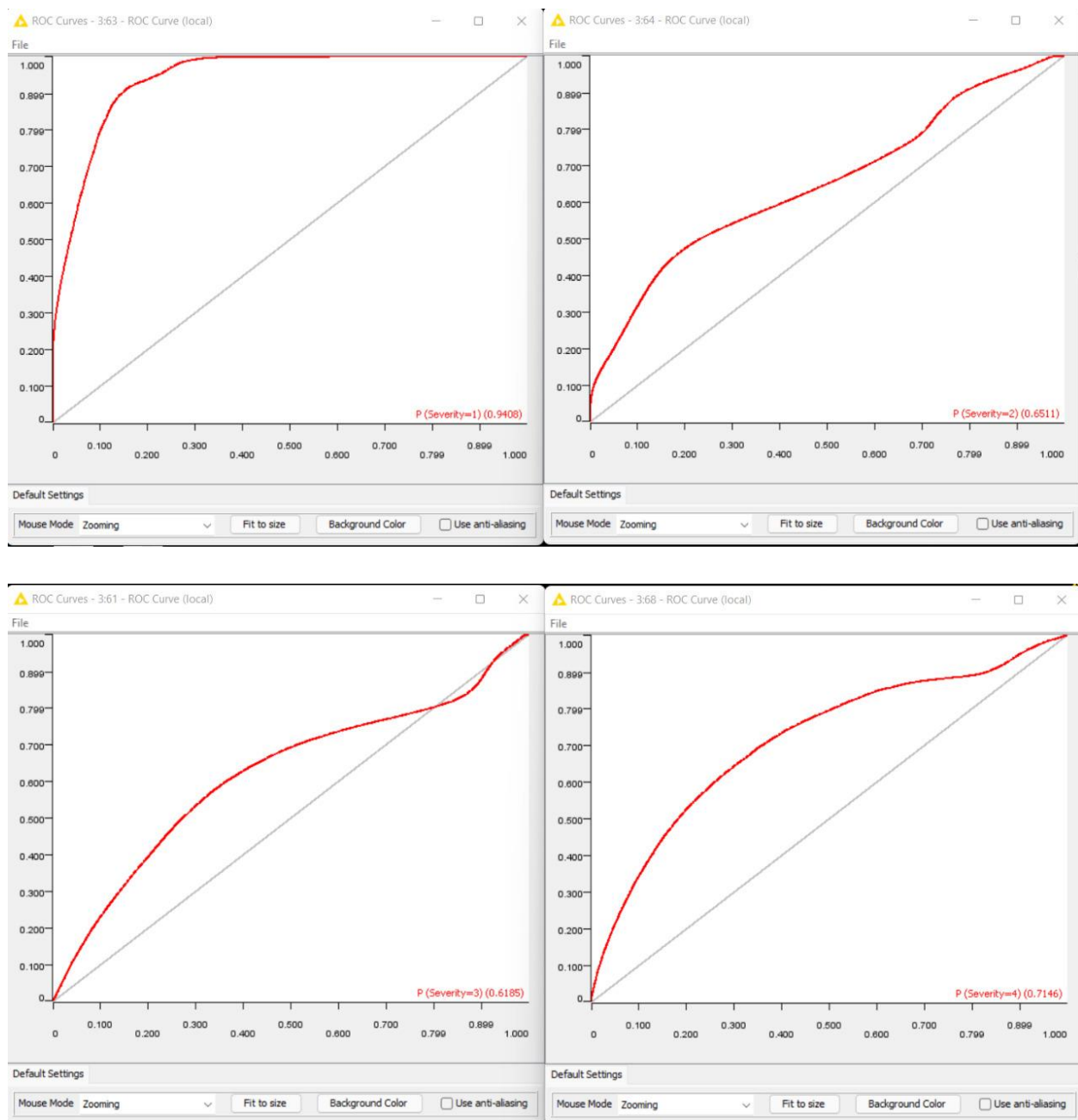
The confusion matrix for Naive Bayes is as follows.

The screenshot shows the 'Confusion Matrix - Scorer' window. It displays a confusion matrix for the 'Severity' variable with four categories: 3, 2, 4, and 1. The matrix shows the counts of correct and incorrect classifications. Below the matrix, it provides summary statistics: Correct classified: 249,386, Wrong classified: 192,676, Accuracy: 56.414 %, Error: 43.586 %, and Cohen's kappa ( $\kappa$ ) 0.114.

Severity \ ...	3	2	4	1
3	7404	23637	3785	12275
2	23337	228182	14165	88424
4	2791	19059	5711	4899
1	48	250	6	8089

Correct classified: 249,386      Wrong classified: 192,676  
Accuracy: 56.414 %      Error: 43.586 %  
Cohen's kappa ( $\kappa$ ) 0.114

Following are the images of ROC curves for severity 1-4 for Naive Bayes Classifier. Since it's a multi class classification problem with 4 different classes, there will be 4 different ROC curves for each class.




	Severity 1	Severity 2	Severity 3	Severity 4
<b>Naive Bayes</b>	0.9408	0.6511	0.6185	0.7146

## Gradient Boosted Trees Classifier:

After Naive Bayes, we worked with Gradient Boosted Trees for modeling and the Gradient Boosted Trees were better than Naive Bayes in predicting new records. The accuracy that Gradient Boosted Trees provide for in predicting severity for new records is 83.579% with 100 models. With 250 models the accuracy increased to 84% which shows that increasing 150 models didn't really increase the accuracy a lot. Running 100 models took a time of almost 15 minutes. Its accuracy is closer to ensemble and random forest but its area under curve for severity 2,3,4 is below 0.9.

Following is the confusion matrix for Gradient Boosted Trees with 100 models.


Confusion Matrix - 0:6 - Scorer
—
□
✕

File Hilite

Severity \ ...	3	2	4	1	
3	13125	31657	2152	288	
2	4344	345948	3156	585	
4	2430	22393	7398	205	
1	237	5134	8	3002	

Correct classified: 369,473

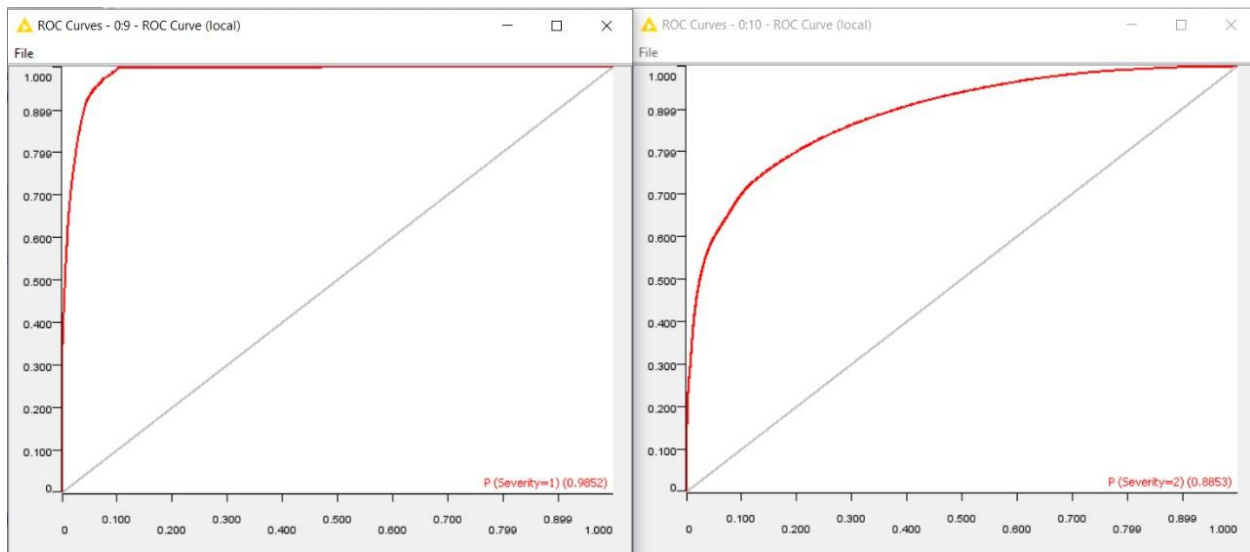
Wrong classified: 72,589

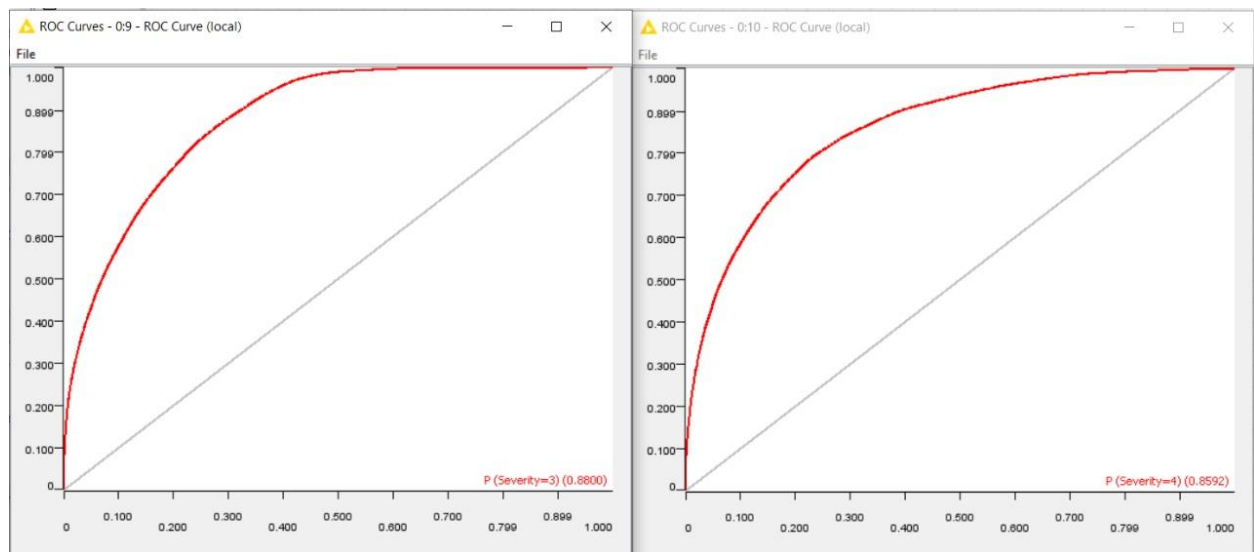
Accuracy: 83.579 %

Error: 16.421 %

Cohen's kappa ( $\kappa$ ) 0.366

Following are the images of ROC curves for severity 1-4 for Gradient Boosted Tree Classifier. Since it's a multi class classification problem with 4 different classes, there will be 4 different ROC curves for each class.





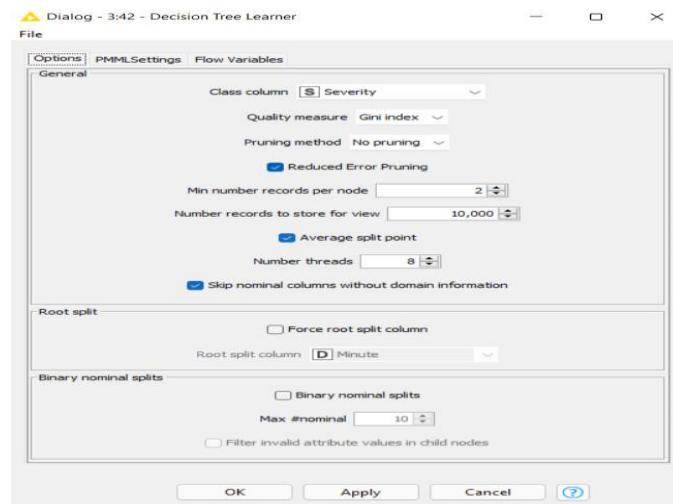
	Severity 1	Severity 2	Severity 3	Severity 4
<b>Gradient Boosted Trees</b>	0.9852	0.8853	0.8800	0.8592

## Decision Trees Classifier:


The Decision Tree Classifier predicted severity for new records with accuracy 82%. The evaluation metric used for this model was **Gini Index**.

Shown below are the hyperparameters for the decision tree model that predicted severity for new records with 82% accuracy. This model took less than 2 minutes to execute. Its area under the curve value for all severity is below 0.9 even naive bayes had severity 1 value greater than 0.9 but for severity 2,3,4 decision trees are better than naive bayes.


There was no pruning in the model and the minimum records per node were set to 2.



The confusion matrix for decision tree model is as follows.


Confusion Matrix - 3:41 - Scorer

File
Hilite


There were missing values in the reference or in the prediction class column..

Severity \ ...	3	2	4	1	
3	22681	19689	3990	444	
2	19923	321447	9725	2254	
4	5051	12208	14760	230	
1	638	2447	248	5016	

Correct classified: 363,904

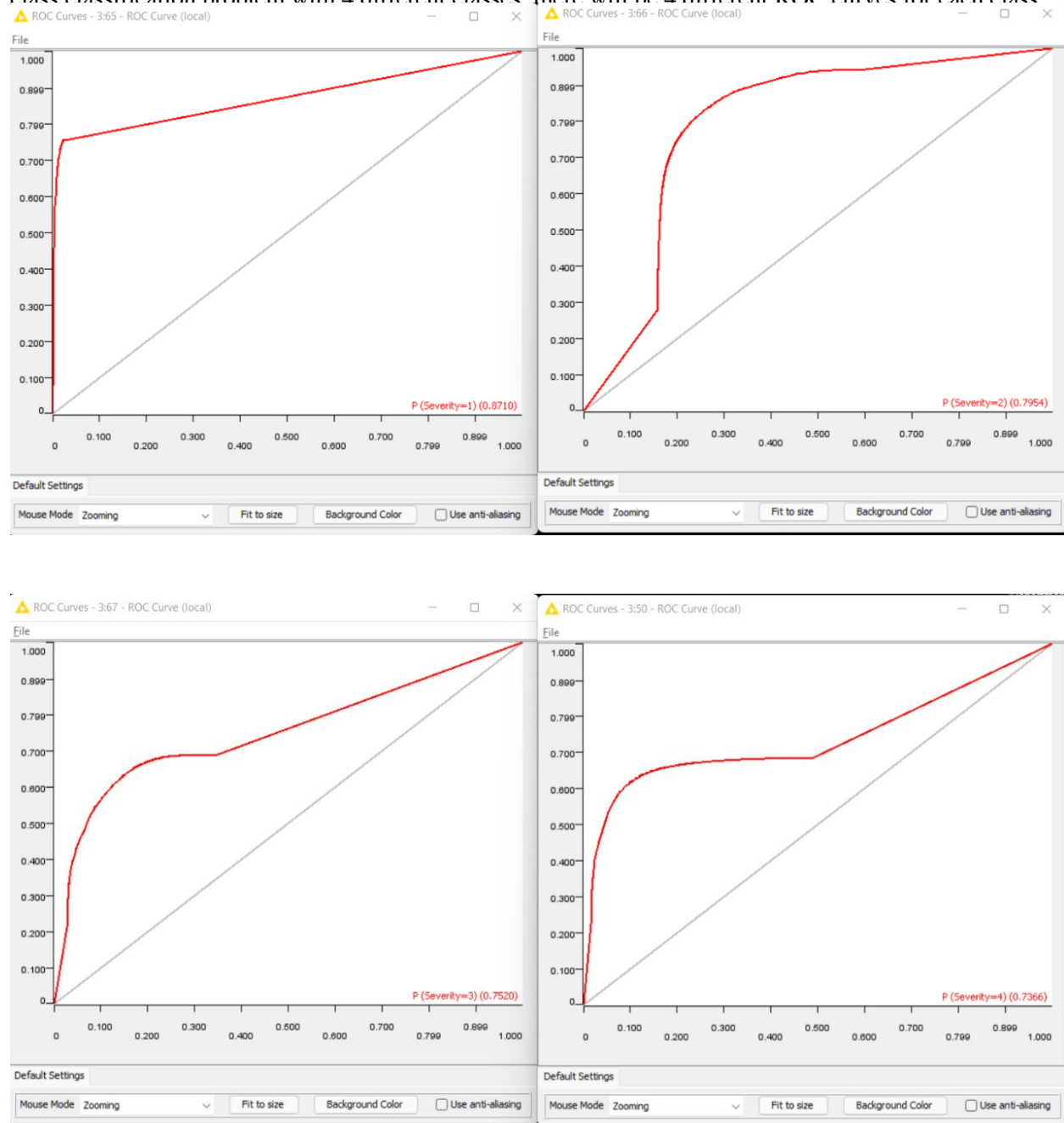
Wrong classified: 76,847

Accuracy: 82.565 %

Error: 17.435 %

Cohen's kappa ( $\kappa$ ) 0.481

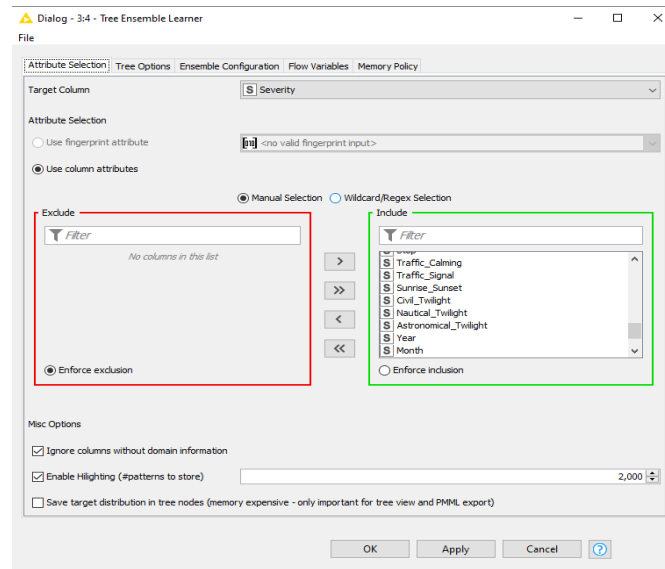
Following are the images of ROC curves for severity 1-4 for Decision Tree Classifier. Since it's a multi class classification problem with 4 different classes there will be 4 different ROC curves for each class




	Severity 1	Severity 2	Severity 3	Severity 4
<b>Decision Trees</b>	0.8710	0.7954	0.7520	0.7336

## Tree Ensemble Classifier:

Tree Ensemble Classifier predicted severity for new records with an accuracy of 86% (with and without enabling highlighting). Ensemble classifier took around 34 minutes to execute. All of its area under curve value for severity was above 0.9 which is one of the best amongst all 5 models.



The confusion matrix for scorer is as follows.


Confusion Matrix - 3:75 - Scorer
—
□
✕

File
Hilite

Severity \ ...	3	2	4	1
3	18559	26767	1632	143
2	4197	346384	2777	750
4	2222	16849	13279	110
1	252	3396	36	4709

Correct classified: 382,931

Wrong classified: 59,131

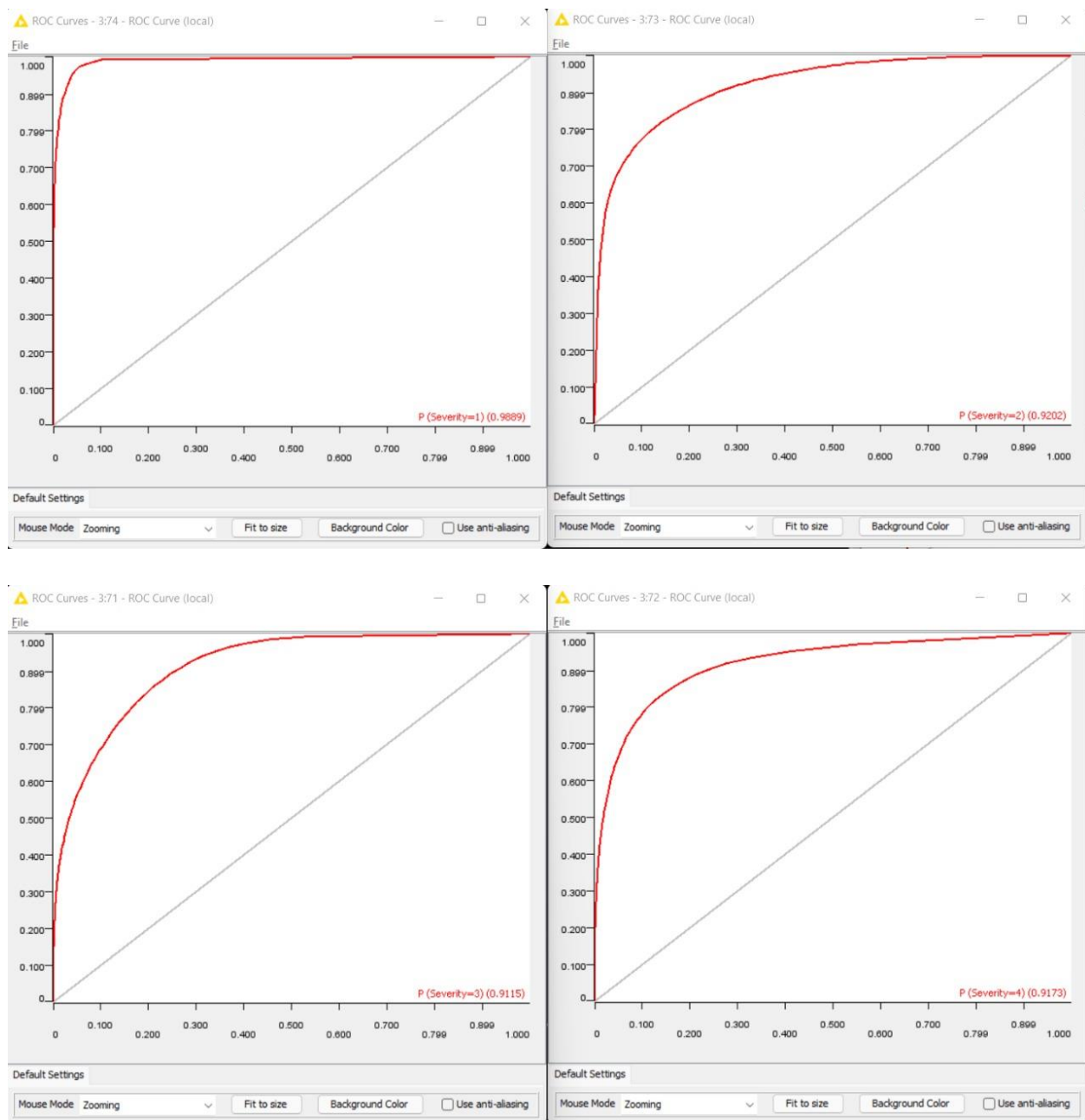
Accuracy: 86.624 %

Error: 13.376 %

Cohen's kappa ( $\kappa$ ) 0.519



Following are the images of ROC curves for severity 1-4 for Tree Ensemble Classifier. Since it's a multi class classification problem with 4 different classes, there will be 4 different ROC curves for each class.



	Severity 1	Severity 2	Severity 3	Severity 4
<b>Tree Ensemble</b>	0.9889	0.9202	0.9115	0.9173

---

## Summary

To summarize the modeling phase, 5 different classifiers with hyperparameter tuning were used to predict severity for this dataset of approximately 1.4 million records.

	Severity 1	Severity 2	Severity 3	Severity 4	Average
<b>Random Forest</b>	0.9883	0.9203	0.9115	0.9176	0.9344
<b>Naive Bayes</b>	0.9408	0.6511	0.6185	0.7146	0.7313
<b>Gradient Boosted Trees</b>	0.9852	0.8853	0.8800	0.8592	0.9024
<b>Decision Trees</b>	0.8710	0.7954	0.7520	0.7336	0.7880
<b>Tree Ensemble</b>	0.9889	0.9202	0.9115	0.9173	0.9345

As it is an imbalance class problem to choose the best model we will not be looking at accuracy rather than area under curve value for all four severity values.

Random Forest and Tree Ensemble classifier turned out to be the best classifier although it was computationally expensive and time taking whereas Naive Bayes showed to be a poor classifier for this dataset but predicted records in almost no time.

Decision Trees and Gradient Boosted Trees all were slightly low in performance in terms of area under curve severity values but were faster when it came to predicting as compared to Random Forest and ensemble classifier.

However, Tree Ensemble took less computing time as compared to Random Forest Classifier but was almost equally accurate when predicting severity for new records. Since we are looking for a balance between accuracy and computing time, Ensemble could be a good choice as a classifier for this dataset.

---

## Findings

### Key insights:

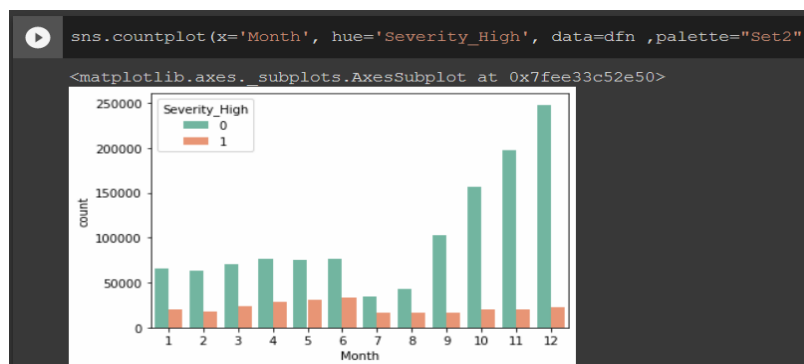
1. Most accidents were reported in Los Angeles (both City & County)

```
[ ] df['City'].describe() [13] df['County'].describe()

count      1515981      count      1516064
unique       10657      unique        1671
top      Los Angeles      top      Los Angeles
freq       39984      freq       138819
Name: City, dtype: object      Name: County, dtype: object
```

2. High Severity accidents were reported the most in the month of June followed by May.

Note: High severity is described as an accident either being 3 or 4 in the severity index and It is denoted by 1. Low severity is described as an accident either being 1 or 2 in the severity index and It is denoted by 0.



3. Most accidents were reported when the visibility was greater than 7 miles and less than 14 miles.

```
df['Visibility(mi)'].value_counts(bins=20)
```

Visibility Bin (mi)	Count
(7.0, 14.0]	1247984
(-0.14100000000000001, 7.0]	217972
(14.0, 21.0]	2394
(28.0, 35.0]	1217
(49.0, 56.0]	813
(35.0, 42.0]	803
(21.0, 28.0]	217
(42.0, 49.0]	140
(63.0, 70.0]	92
(77.0, 84.0]	85
(56.0, 63.0]	80
(70.0, 77.0]	35
(98.0, 105.0]	7
(84.0, 91.0]	5
(119.0, 126.0]	5
(105.0, 112.0]	2
(126.0, 133.0]	1
(133.0, 140.0]	1
(91.0, 98.0]	0
(112.0, 119.0]	0

Name: Visibility(mi), dtype: int64

---

## Limitations of the Study

Our study has numerous limitations that stem from various perspectives including data worked on, hardware used, and the global context of Covid-19.

1. The data provided stated that each data source has its own metric of how 'severe' the number 4 is for e.g. (in the severity column). The ideal scenario would be to filter the data, source wise and then train the model accordingly but the source column has been removed from the latest updated version of the dataset.
2. Due to the enormous size of the dataset, the assigned RAM (12 GB) provided to KNIME was not enough for Random Forest to run on more than about 100 models. Details of the machine used are present in the introduction of model building.
3. The dataset had a lot of missing values due to which several rows were deleted which amount to a deletion of 2.8 % records. And rest were filled with either mean values for numerical features and most common value for categorical features which might have reduced the accuracy of the model.
4. While working on Python, classifiers required each feature to be converted to numerical thus leading to an increased processing time as the unique values for some features were in the thousands.
5. Our study does not take into account the changing traffic patterns that resulted from the emergence of Covid-19 at the start of 2020 and throughout.

## Recommendations for Data collection:

1. Standardize attributes for Wind Direction & Weather Condition

The data had numerous words depicting the same meaning thus had to be thoroughly cleaned before processing. For e.g. very cloudy & mostly cloudy. Same applies for Wind Direction. Based on data from all the sources, we recommend the following standard to be used for both features:

WIND DIRECTION & WEATHER UNIQUE VALUES HERE

Weather Condition : '**Rain**', '**Cloud**', '**Snow**', '**Clear**', '**Fog**', '**Thunderstorm**', '**Smoke**', '**Precipitation**', '**Wintry Mix**', '**Dust**', '**Volcanic Ash**'

Wind Direction: '**SW**' (South West), '**CALM**', '**W**' (West), '**N**' (North), '**S**' (South), '**NW**' (North West), '**E**' (East), '**SE**' (South East), '**VAR**'(Variable), '**NE**' (North East)

---

2. Count the average cars passing by at start time.

Right at the beginning of the accident, if the measure of the number of cars present at that moment is recorded, then the effect on severity of traffic flow can be gauged more precisely. This can be done using cameras and motion sensors on the road to detect the number of cars. This will help us to detect if there are more vehicles at that time so at that instance the probability of traffic severity being higher will be more as compared to less vehicles.

3. Less missing values in the future.

For instance, precipitation could have been an important feature in determining the severity but it had 33.67% missing values which had to be replaced by the mean of other values. Also, the address attribute 'number' had to be dropped as its missing values was 69%. It could have had a positive impact in predicting the severity.

## **Model expiration**

1. Due to advancements in technology, a shift towards electric vehicles is being observed. Artificial Intelligence is very strong in avoiding roadblocks ahead of them hence the chances of having a high severity traffic is very low. Hence in a few years time when the proportion of self driving cars increase in the United States then the model's traffic severity model will expire.

2. GPS rerouting is becoming more common nowadays and it is expected that in months it will be more efficient and accurate. When an accident occurs GPS relocates the route on map and guides them to an alternative route for their destination hence the severity of traffic after the accident will be low as compared to before. Hence when this technology expands further and GPS covers all routes on map the model will expire.

3. No turning loop present in all records. In data preprocessing we observed that every accident recorded had no turning loop hence we removed that feature while training the model. If new accidents for which prediction has to be made contain a turning loop, for that turning loop will not have any contribution in prediction. Hence if a high frequency of new data comes for prediction with a turning loop then also the model expires and we need to create a new model for prediction.

.