TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY
ACADEMIC YEAR 2023/2024 Session 202301

BACS3013 DATA SCIENCE: Assignment Documentation

Title: Car Accident 2020

Programme and Tutorial Class: RDS1S3G2
Tutor Name: Dr. NOOR AIDA BINTI HUSAINI
Submission Date: 7-May-2023

Team Member:

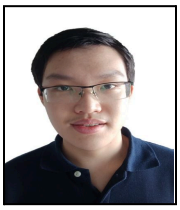
 **Student Name:** Murdi Starla Nathasa **Student ID:** 22WMR02835I **Module-in-Charge:** Logistic Regression	 **Student Name:** Ryan Kho Yuen Thian **Student ID:** 22WMR04097 **Module-in-Charge:** Random Forest
 **Student Name:** Thong Cheng How **Student ID:** 22WMR03154 **Module-in-Charge:** Decision Tree	 **Student Name:** Yong Zee Lin **Student ID:** 22WMR03770 **Module-in-Charge:** K Nearest Neighbour

TABLE OF CONTENT

A. BUSINESS UNDERSTANDING

1.1 Problem Background

YT Star is a consulting firm owned by 4 people, who are Ryan Kho, Starla, Cheng How and Zee Lin that is based in Brazil. It currently has 20 employees and is hired by the Brazilian government. Their mission is to investigate car accidents in Brazil. Car accidents are the major cause of unnatural deaths in the world. The Brazilian government works hard to make the people aware about the rules that must be adhered to when driving a vehicle on the road in order to reduce fatalities. According to a report by 360º CI agency, there were an estimate of 1.51 million road crashes from 2009 to 2019 (total of 79,085 reported road deaths). This was provided by Brazil's Federal Highway Police. Between 2009 and 2019, Brazil's road death rate was reduced by 26% (meaning that it fell short of the target placed by the World Health Organisation (WHO) of a 50% drop during this period of time). As many as 5,300 road deaths happened in Brazil in the year 2020. Apart from deaths, car accidents can also bring down a country's economy in several ways, such as property damage, workplace productivity loss, traffic congestion, medical costs etc.

To reduce the number of unnecessary fatalities on the road and the economic impact of car accidents on Brazil, YT Star will be using a dataset from the Detran (Department of Transit) database that records all the car accidents that occurred in Brazil in the year 2020. By using this dataset, YT Star will develop an analytics application that predicts the severity of car accidents and identifies ways to reduce the severity of car accidents.

1.2 Objectives/Aims

- To develop an analytics application that predicts the severity of car accidents, especially fatal accidents
- To identify the factors that lead to car accidents based on the dataset
- To support the Brazilian government's plan to make the roads a safer place for people to drive on
- To reduce the number of fatalities on the road and Brazil's economic burden
- To achieve at least 80% for Accuracy, Macro Average Precision, Macro Average Recall and Macro Average F-1 Score for Analytics application

1.3 Motivation

As stated in the objectives above, the motivation of this project is to make the roads in Brazil a safer place.

1.4 Timeline/Milestone

Week	Start Date	End Date	Activity
1	21/02/2023	27/02/2023	Selection of one of the candidate datasets. Obtain general understanding of the particular datasets. Determine whether supervised vs unsupervised, classification vs regression.
2	28/02/2023	06/03/2023	Planning and establishment of general milestones. Study what Data Science and Machine Learning is about.
3	07/03/2023	13/03/2023	Learn Python coding using Jupyter Notebook and learn how to use Pandas for pre-processing.
4	14/03/2023	20/03/2023	Continue tasks from Week 3
5	21/03/2023	27/03/2023	Learn about the different Classification ML algorithms.
6	28/03/2023	03/04/2023	Continue tasks from Week 5
7	04/04/2023	10/04/2023	Learn how to use the Scikit-Learn ML library and entire ML process
8	11/04/2023	17/04/2023	Perform Exploratory Data Analysis on selected dataset.
9	18/04/2023	24/04/2023	Perform Data Cleaning and Transformation on selected dataset and split the dataset into Training, Validation
10	25/04/2023	01/05/2023	Perform ML Modelling, Train models, Evaluate models, Select best model, and test the selected Best Model

11	**02/05/2023**	**07/05/2023**	**Deployment and Documentation**

1.5 Inventory of Resources

a) Personnel: Dr Noor Aida Binti Husaini, PHD

b) Data: https://www.kaggle.com/datasets/raphaelmarconato/detran-accidents-2020

c) Computing resources: 4 Laptops

d) Software: see below

Tools comparison	**Jupyter Notebook**	**Scikit-Learn**	**Python**
Type of license and open source license	3-clause BSD License and Open source licence	New BSD License	Python Software Foundation License
Year founded	Released in 2011 as IPython Notebook but in 2014, it was changed to Jupyter	2007	1991
Founding company	Fernando Pérez, Brian Granger, and Min Ragan-Kelley	David Cournapeau	Guido van Rossum
License Pricing	Free	Free for Development and Distribution	Free for Development and Distribution
Supported features	Notebook to run python scripts	Machine Learning Algorithms	General Programming Language
Common applications	Scientific Computing and Machine Learning	Predictive Analytics	Can be used in all types of applications but is especially popular in Data Science
Customer support	Online website, Forums	None. Only via forums like StarkOverflow.	Free forums and communities, commercial third parties

Limitations	Slows when a lot of code are executed. Does not support full IDE features e.g. code assist	Poor with Graph algorithms	Not as fast as programming languages such as Java and C++

Why the selected tools are suitable:

Jupyter is the main tool we use where we execute most of our code for data understanding, preparation, modelling, evaluation and conclusion. This is due to the fact that we are able to run live code therefore we can work in sections and make sure each section is a success so that we can proceed to the next step. Live code also enables us to get output of a certain section of the script without having to run everything manually from the top. It can also support code and text development and to do analysis for our dataset and find meaningful insights for our objective and aim for the dataset. Another reason is because jupyter is more user friendly as we are familiar with the commands and how to insert code and notice errors if they are encountered in the process of analyzing.

Python was chosen as the programming language because it is a general programming language (and therefore appeals to a wider audience) as opposed to R which was designed to work with statistical applications. Another candidate is Julia but it does not have as many learning resources as Python and R. Python, on the whole, has become a very popular programming language even outside of Data Science. This fact ensures that Python will be sustainable in the future. Scikit-Learn was chosen because it is the most popular Python ML library and it has many learning resources. More importantly, it is used in many production deployments. Another reason we choose Scikit-Learn is because it can have a more variety to do modelling and analyzing as it is considered one the most used libraries in machine learning.

1.6 System flowchart/activity diagram

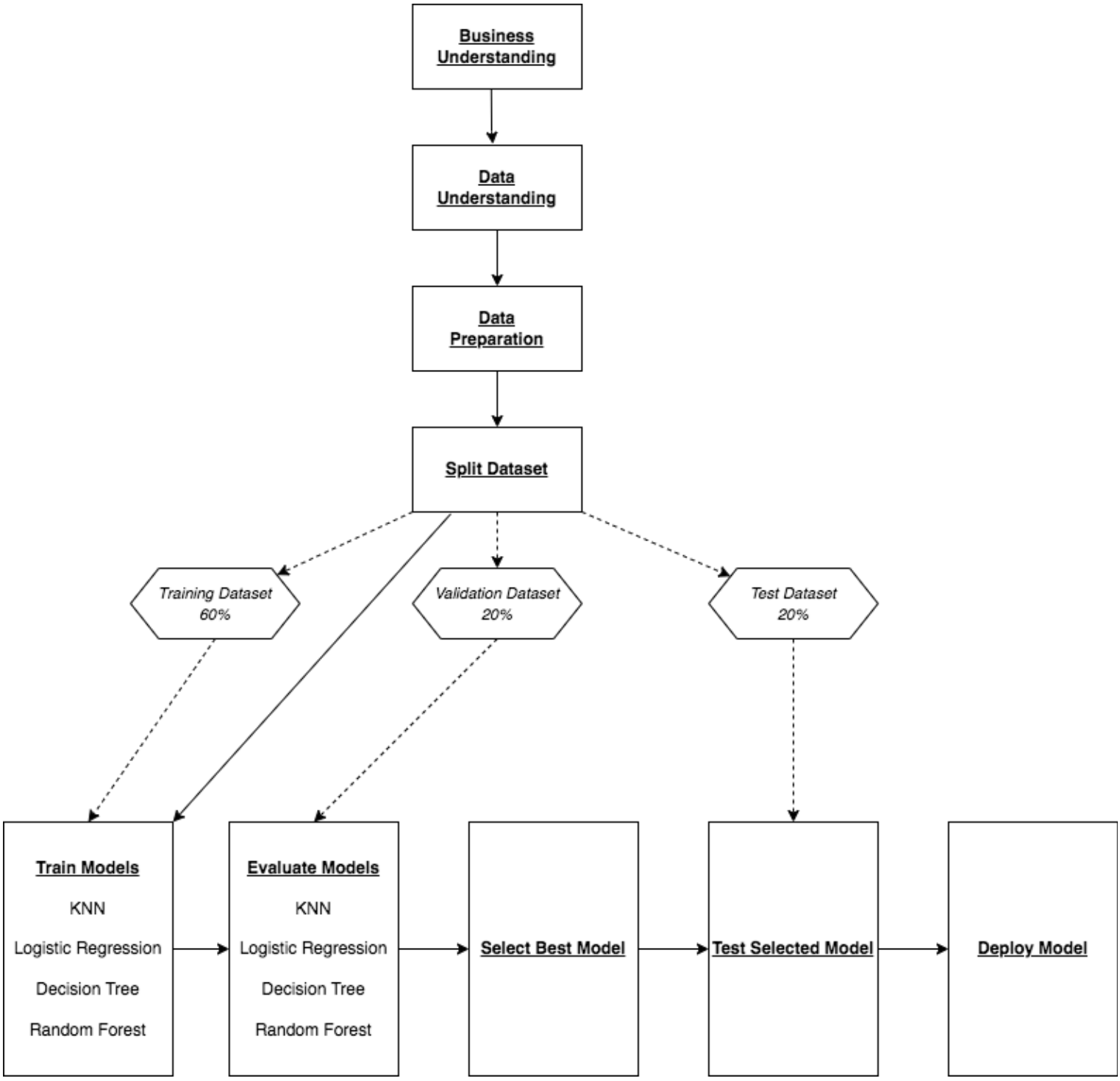


Figure 1: System flowchart/ activity diagram

1.7 Requirements, Assumptions and Constraints

- The deadline for this assignment is 7 May 2023
- We are required to produce good quality results that can be easily understood
- The Kaggle dataset is open-sourced, which means that it can be seen by the public. Therefore, we are free to use the data and there should not be any data security concerns and legal issues.
- We assume that most of the data recorded are correct and verified although some may be non-verifiable .
- We are only limited to 4 laptops and 2 kinds of software but we should be able to handle a decent size of data set for modelling.

1.8 Data Mining Success Criteria

To extract and find the most meaningful data for this dataset, we have different criteria such as accuracy of each model, recall score of each model, precision score of each model and the ROC curve of each model. Through these ways of criteria, we will be able to determine the best model for deployment and to determine whether our objectives and aims are met.

B. DATA UNDERSTANDING

2.1 Background of the Application

The application that we are developing is a Machine Learning Algorithm to predict the severity of Road Accidents. This is a supervised ML classification algorithm with Multiclass Prediction. The reason why the algorithm is in multiclass prediction and not in binomial is because not all

2.2 Analysis of selected tool with any other relevant tools

We have mentioned the tools that we will be using in section 1.5 Inventory of Resources.

2.3 Loading and viewing the dataset

This dataset was obtained from the Kaggle Website

```
In [1]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        import numpy as np
        import math
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [2]: df = pd.read_csv('si_env-2020.csv', sep=';', encoding='ISO-8859-1')#this csv file uses semi-colon as separator. Need to use
                                                                            #the specified encoding to solve UnicodeDecodeError

        df.head(50)
```

```
df.head(50)
```

Out[2]:

	num_boletim	data_hora_boletim	Nº_envolvido	condutor	cod_severidade	desc_severidade	sexo	cinto_seguranca	Embreagues	Idade	nascimento	cate
0	2020-014152383-001	20/03/2020 02:18	1	S	1	NAO FATAL	M	NÃO	NÃO	35	06/09/1984	
1	2020-014152383-001	20/03/2020 02:18	2	S	3	SEM FERIMENTOS	F	SIM	SIM	42	11/03/1978	
2	2020-014152383-001	20/03/2020 02:18	3	N	3	SEM FERIMENTOS	M	SIM	NÃO	32	20/11/1987	
3	2020-014158612-001	20/03/2020 05:39	1	N	1	NAO FATAL	F	SIM	NÃO	72	29/06/1947	
4	2020-014158612-001	20/03/2020 05:39	2	S	3	SEM FERIMENTOS	M	SIM	NÃO	55	13/03/1965	

Figure 2: Code to view the data in the dataset using jupyter notebook

2.3.1 Checking individual columns

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

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22643 entries, 0 to 22642
Data columns (total 18 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   num_boletim             22643 non-null  object
 1    data_hora_boletim      22643 non-null  object
 2    Nº_envolvido           22643 non-null  int64
 3    condutor               22643 non-null  object
 4    cod_severidade         22643 non-null  int64
 5    desc_severidade        22643 non-null  object
 6    sexo                   22643 non-null  object
 7    cinto_seguranca        22643 non-null  object
 8    Embreagues             22643 non-null  object
 9    Idade                  22643 non-null  int64
 10   nascimento             22643 non-null  object
 11   categoria_habilitacao  22643 non-null  object
 12   descricao_habilitacao  22643 non-null  object
 13   declaracao_obito       22643 non-null  int64
 14   cod_severidade_antiga  22643 non-null  int64
 15  especie_veiculo         22643 non-null  object
 16   pedestre               20489 non-null  object
 17   passageiro             20598 non-null  object
dtypes: int64(5), object(13)
memory usage: 3.1+ MB
```

Figure 3: Showing the column names, non-null value counts and the data type for each column in the dataset.

2.3.2 Description of dataset

The dataset we have chosen is called Car Accident 2020 that we obtained from Kaggle. This dataset contains details of car accidents that happened in the year 2020 in Brazil. It contains 18 columns, of the 18 columns, 13 are categorical variables, and we have 22643 accident records. Below are the data that is inside the dataset:

- *num_boletim* : Accident record number.
- *data_hora_boletim* : Date and time of registration.
- *nº_envolvido* : Number of people involved in the accident.
- *condutor* : Whether the driver was responsible for the accident.
- *desc_severidade* : Whether the accident was fatal or not.
- *sexo* : Person's sex.
- *cinto_seguranca* : Whether they were wearing a seat belt or not.

- *embreagues* : If the driver was under the influence of alcohol.
- *idade* : Age
- *nascimento* : Birth date.
- *categoria_habilitacao* : Enabling type.
- *descricao_habilitacao* : Description of Qualification.
- *declaracao_obito* : Whether there was a death or not.
- *cod_severidade_antiga* : severity code
- *especie_veiculo* : vehicle type
- *pedestre* : if there was a pedestrian involved.
- *passageiro* : whether there was a passenger or not.

As the column names may not be user-friendly, we have decided to rename the columns so that they are more understandable and English-like to allow everyone to understand what is the column names and what data contains inside.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22643 entries, 0 to 22642
Data columns (total 18 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   accidentRecordNum             22643 non-null  object
 1   reportedDateNTime             22643 non-null  object
 2   noPeopleInvolved              22643 non-null  int64
 3   driverResponsibleForAccident  22643 non-null  object
 4   severityCode                  22643 non-null  int64
 5   severityDesc                  22643 non-null  object
 6   sex                           22643 non-null  object
 7   woreSeatBelt                  22643 non-null  object
 8   driverUnderAlcohol            22643 non-null  object
 9   age                           22643 non-null  int64
 10  birthDate                     22643 non-null  object
 11  qualificationCat              22643 non-null  object
 12  qualificationDesc             22643 non-null  object
 13  involvedDeath                 22643 non-null  int64
 14  oldSeverityCode               22643 non-null  int64
 15  vehicleType                   22643 non-null  object
 16  involvedPedestrian            20489 non-null  object
 17  involvedPassenger             20598 non-null  object
dtypes: int64(5), object(13)
memory usage: 3.1+ MB
```

Figure 5: The Original Dataset with Renamed Columns

At figure 2, this shows the original dataset that has been renamed to english. In this table it shows that they are 18 columns in total and all the counts are not null. Inside the columns has over 20,000+ records for the data collected for this dataset.

1. accidentRecordNum: Accident record number
2. reportedDateNTime: Reported date and time
3. noPeopleInvolved: Number of people involved in the accident
4. driverResponsibleForAccident: Whether the driver was responsible for the accident
5. severityCode: Severity code
6. severityDesc: Whether the accident was fatal or not
7. sex: Person's sex
8. woreSeatBelt: Whether they were wearing a seat belt or not
9. driverUnderAlcohol: If the driver was under the influence of alcohol
10. age: Age
11. birthDate: Birth date
12. qualificationCat: Qualification Category
13. qualificationDesc: Description of Qualification
14. involvedDeath: Whether there was a death or not
15. oldSeverityCode: old severity code
16. vehicleType: vehicle type
17. involvedPedestrian: if there was a pedestrian involved
18. involvedPassenger: whether there was a passenger or not

Figure 6: Description of Each Column

Referring to figure 6, the columns have been explained to allow the reader to understand each column better .

2.4 Individual columns for better understanding

2.4.1 Finding missing values count in all columns

```
In [4]: df.isnull().sum()

Out[4]: num_boletim                 0
        data_hora_boletim           0
        Nº_envolvido                0
        condutor                    0
        cod_severidade              0
        desc_severidade             0
        sexo                        0
        cinto_seguranca             0
        Embreagues                  0
        Idade                       0
        nascimento                  0
        categoria_habilitacao       0
        descricao_habilitacao       0
        declaracao_obito            0
        cod_severidade_antiga       0
        especie_veiculo             0
        pedestre                 2154
        passageiro               2045
        dtype: int64
```

Figure 7: Checking for Null Values in All Columns

From figure 7, we can observe that the two columns, which are **pedestre** and **passageiro,** have missing values. For the other columns they seem to consist of no missing values, which is good, and can be examined later when we have reached data preparation. We will be cleaning the data, removing duplicate data, replacing values and more. This allows us to have a better understanding of the dataset so that by the time we reach the data preparation stage, we can be more familiar with the dataset and identify the best options to do with the dataset missing values and outliers.

2.4.2 Generating descriptive statistics for the dataset

```
In [5]: df.describe()
        #df.describe(include="all")
```

Out[5]:

	Nº_envolvido	cod_severidade	Idade	declaracao_obito	cod_severidade_antiga
count	22643.000000	22643.000000	22643.000000	22643.0	22643.0
mean	1.741465	1.801175	34.597624	0.0	0.0
std	0.918086	1.020479	15.906508	0.0	0.0
min	1.000000	0.000000	0.000000	0.0	0.0
25%	1.000000	1.000000	24.000000	0.0	0.0
50%	2.000000	1.000000	33.000000	0.0	0.0
75%	2.000000	3.000000	44.000000	0.0	0.0
max	18.000000	3.000000	91.000000	0.0	0.0

Figure 8: These general statistics help us to understand the dataset's distribution, shape, central tendency and dispersion.

2.4.3 Correlation Coefficient between all Columns

```
In [10]: df.corr()
```

Out[10]:

	noPeopleInvolved	severityCode	age	involvedDeath	oldSeverityCode
noPeopleInvolved	1.000000	-0.028281	-0.129599	NaN	NaN
severityCode	-0.028281	1.000000	0.285520	NaN	NaN
age	-0.129599	0.285520	1.000000	NaN	NaN
involvedDeath	NaN	NaN	NaN	NaN	NaN
oldSeverityCode	NaN	NaN	NaN	NaN	NaN

Figure 9: Correlation Coefficient of Some Columns

From the figure 9, we can observe that not all columns are included for the calculations of correlation coefficient. This is because the corr() method will generally exclude any NaN values. NaN is the short form of Not A Number. Therefore the method will also ignore any non-numeric columns or data type in the dataframe. The correlation coefficient having the value of NaN could

be due to the identical values found in one of the columns. Below is our conclusion for the dataset based on figure 9.

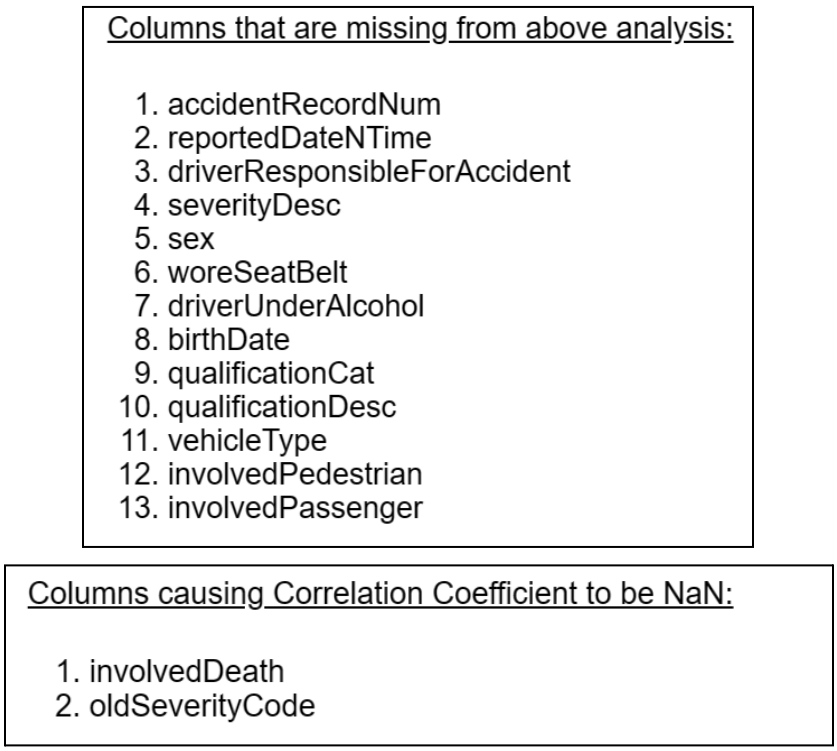


Figure 10: Columns missing from Correlation Coefficient calculations and Columns causing NaN values for Correlation Coefficient in Figure 9.

2.4.3 Category 1- Dependent Variables

There are 3 dependent variables, which are SeverityCode, SeverityDesc and oldSeverityCode. Since we have three different types of dependent variables, we first have to find the unique values for each of the three variables.

a) severityDesc

```
In [12]: print(df['severityDesc'].value_counts())

NAO FATAL          12544
SEM FERIMENTOS      9338
NAO INFORMADO        648
FATAL                113
Name: severityDesc, dtype: int64

In [13]: print(df['severityDesc'].unique())

['NAO FATAL      ' 'SEM FERIMENTOS ' 'NAO INFORMADO  ' 'FATAL          ']
```

Figure 11: finding unique values for severityDesc

From figure 11, there are 4 unique values in severityDesc which are NAO FATAL (not fatal), SEM FERIMENTOS (No injuries), NAO INFORMADO (no information) and FATAL (fatal). We can see that there are very few Fatal cases (113). This means that this dataset is very imbalanced. We will balance out the data in the later stages. This variable is nominal.

b) severityCode

```
In [14]: print(df['severityCode'].value_counts())

1    12544
3     9338
0      648
2      113
Name: severityCode, dtype: int64

In [15]: print(df['severityCode'].unique())

[1 3 0 2]
```

Figure 12: Finding Unique Values for severityCode

From figure 12, we can notice that there are also 4 unique values in severityCode which are 1, 3, 0 and 2. This variable is also nominal.

Comparing figures 11 and 12, we can see that they have the same number of unique values as well as the count for each unique value for severityDesc and severityCode. Therefore, to determine and confirm that these two columns are the same, we calculate the correlation coefficient between them..

Step 1: We first convert the severityDesc column from a categorical variable to a numerical variable by replacing its values with numbers.

```
In [16]: copydf = df.copy()

         copydf['severityDesc'] = copydf['severityDesc'].replace(['NAO FATAL      '], 1)
         copydf['severityDesc'] = copydf['severityDesc'].replace(['SEM FERIMENTOS '], 3)
         copydf['severityDesc'] = copydf['severityDesc'].replace(['NAO INFORMADO  '], 0)
         copydf['severityDesc'] = copydf['severityDesc'].replace(['FATAL          '], 2)
```

Step 2: We calculate the correlation coefficient between "severityCode" and "severityDesc"

```
In [17]: corrCoef = copydf['severityCode'].corr(copydf['severityDesc'])
         print("The correlation coefficient before rounding: ", corrCoef)
         print("The correlation coefficient after rounding: ", round(corrCoef, 2))

         The correlation coefficient before rounding:  1.0
         The correlation coefficient after rounding:  1.0
```

Figure 13: Steps to calculate the correlation coefficient between severityDesc and severityCode

From figure 13, we can observe that there is a perfect positive correlation between severityCode and severityDesc. But to be sure, we wrote additional code to check whether the columns match.

```
In [18]: dataFrame = np.where(copydf['severityCode'] == copydf['severityDesc'], 'True', 'False')
         matchingCount = 0;
         unmatchingCount = 0;
         for x in dataFrame:
           if x == 'True':
               matchingCount = matchingCount + 1
           else:
               unmatchingCount = unmatchingCount + 1

         print("Number of matching rows in both columns: ", matchingCount)
         print("Number of unmatching rows in both columns: ", unmatchingCount)

         Number of matching rows in both columns:  22643
         Number of unmatching rows in both columns:  0
```

Figure 14: Code to check for matching rows in severityDesc and severityCode columns

From the result gathered in figures 13 and figure 14, we can conclude that the severityCode and severityDesc columns are actually the same just that one is the code while the other is the

description. From the above figures, we can see that the 2 columns are the same because the correlation coefficient is 1,which is perfect, and there are no unmatching rows.

Severity Code	Severity Description
0	NAO INFORMADO
1	NAO FATAL
2	FATAL
3	SEM FERIMENTOS

Figure 15: Relationship between Severity Code and Severity Description

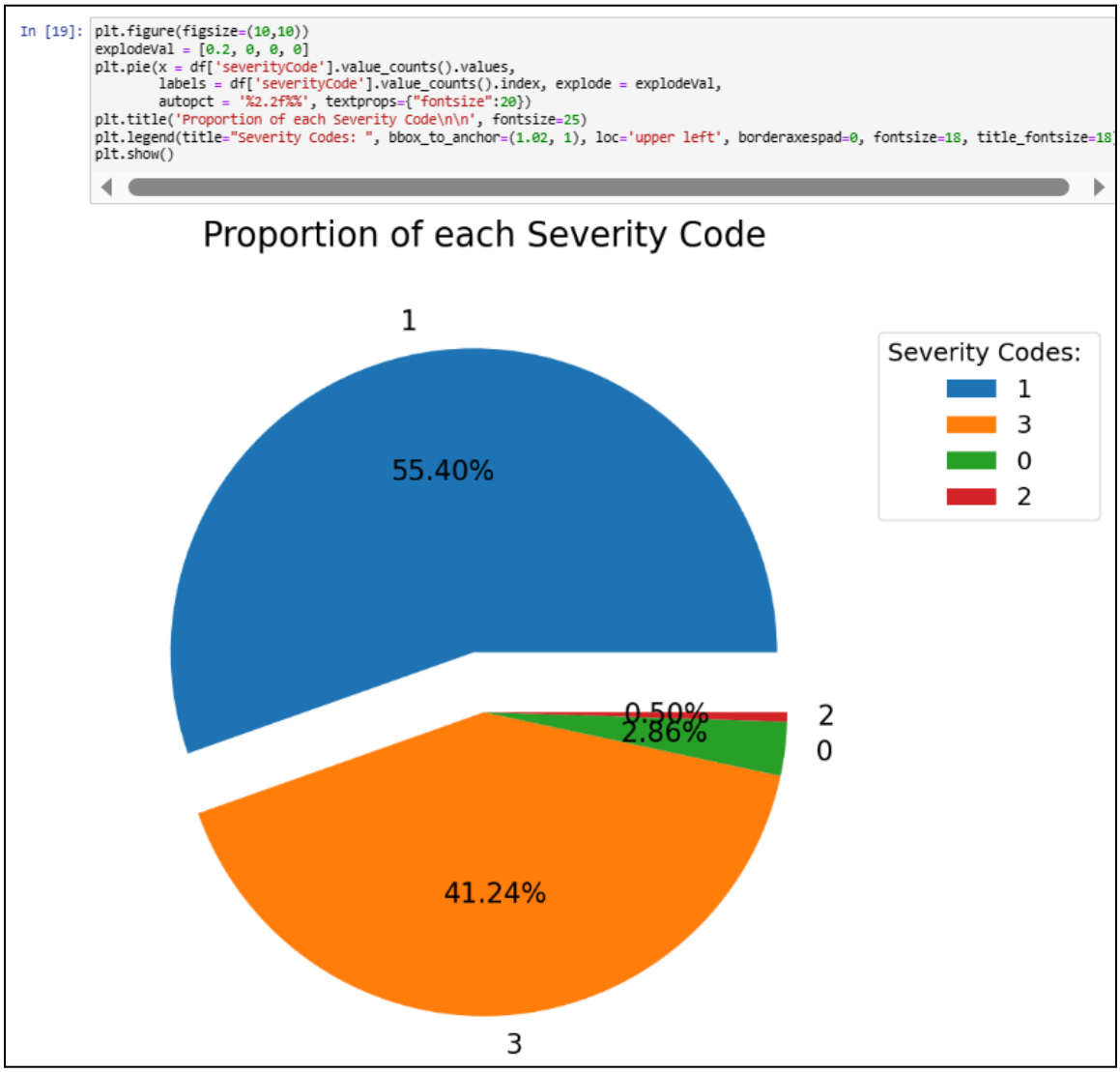


Figure 16: Pie Chart Showing Proportion of each severityCode

From figure 16, we can conclude that the majority of the accidents recorded are classified as "Not Fatal" / Severity Code 1, followed by "No injuries" / Severity Code 3, "No information" / Severity Code 0 and "Fatal" / Severity Code 2. We can also conclude that the dataset is very imbalanced because there are very few fatal cases (0.5% only). We will do something about it in Modelling.

c) oldSeverityCode

```
In [21]: print(df['oldSeverityCode'].value_counts())

0    22643
Name: oldSeverityCode, dtype: int64
```

Figure 17: The Unique Value for oldSeverityCode

From figure 17, the only unique value is 0. In the Data Preparation Stage, we will be removing this column as the constant value, 0, is not going to affect our prediction or teach us about the target variables.

```
In [22]: corrCoef = copydf['severityCode'].corr(copydf['oldSeverityCode'])
         print("Severity Code and Old Severity Code")
         print("-----------------------------------")
         print("The correlation coefficient before rounding: ", corrCoef)
         print("The correlation coefficient after rounding: ", round(corrCoef, 2), '\n')

         corrCoef = copydf['severityDesc'].corr(copydf['oldSeverityCode'])
         print("Severity Description and Old Severity Code")
         print("-----------------------------------")
         print("The correlation coefficient before rounding: ", corrCoef)
         print("The correlation coefficient after rounding: ", round(corrCoef, 2))

Severity Code and Old Severity Code
-----------------------------------
The correlation coefficient before rounding:  nan
The correlation coefficient after rounding:  nan

Severity Description and Old Severity Code
-----------------------------------
The correlation coefficient before rounding:  nan
The correlation coefficient after rounding:  nan
```

Figure 18: Calculating Correlation Coefficient between Severity Code and Old Severity Code and Severity Description

The value of NaN for the correlation coefficients means that there are no correlations between severity code and old severity code & between severity description and old severity code. This is because of the identical values found in the old severity code column.

Below, we change the severityCode values so that the column becomes ordinal.

Severity Code	Severity Description
0	NAO INFORMADO
1	SEM FERIMENTOS
2	NAO FATAL
3	FATAL

In [24]:
```
forCC['severityCode'] = forCC['severityCode'].replace([1], 12)
forCC['severityCode'] = forCC['severityCode'].replace([3], 11)
forCC['severityCode'] = forCC['severityCode'].replace([0], 10)
forCC['severityCode'] = forCC['severityCode'].replace([2], 13)

forCC['severityCode'] = forCC['severityCode'].replace([12], 2)
forCC['severityCode'] = forCC['severityCode'].replace([11], 1)
forCC['severityCode'] = forCC['severityCode'].replace([10], 0)
forCC['severityCode'] = forCC['severityCode'].replace([13], 3)
```

2.4.4 Category 2- Independent variables

a) accidentRecordNum

In [28]:
```
print(df['accidentRecordNum'].value_counts())
```

```
2020-014233487-001    17
2020-060896323-001    10
2020-059740386-001     9
2020-051471680-001     8
2020-002262683-001     8
                      ..
2020-020710349-001     1
2020-012789344-001     1
2020-050587901-001     1
2020-023929653-001     1
2020-004191432-001     1
Name: accidentRecordNum, Length: 10624, dtype: int64
```

In [29]:
```
print(df['accidentRecordNum'].unique())
```

```
['2020-014152383-001' '2020-014158612-001' '2020-014161105-001' ...
 '2020-060013548-001' '2020-060019495-001' '2020-060029999-001']
```

It is a categorical variable, which has a lot of unique values. If there are 2 or more parties involved in an accident, then the 2 samples will have the same accident record number. Since it's just accident record number, we will not convert it into a numerical variable.

b) reportedDateNTime

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

```
20/03/2020 16:07    17
19/12/2020 08:38    10
27/06/2020 21:30     9
12/12/2020 13:21     9
27/10/2020 23:46     8
                    ..
02/02/2020 18:00     1
14/05/2020 17:42     1
21/02/2020 19:03     1
03/02/2020 16:26     1
11/02/2020 14:00     1
Name: reportedDateNTime, Length: 10456, dtype: int64
```

```
print(df['reportedDateNTime'].unique())
```

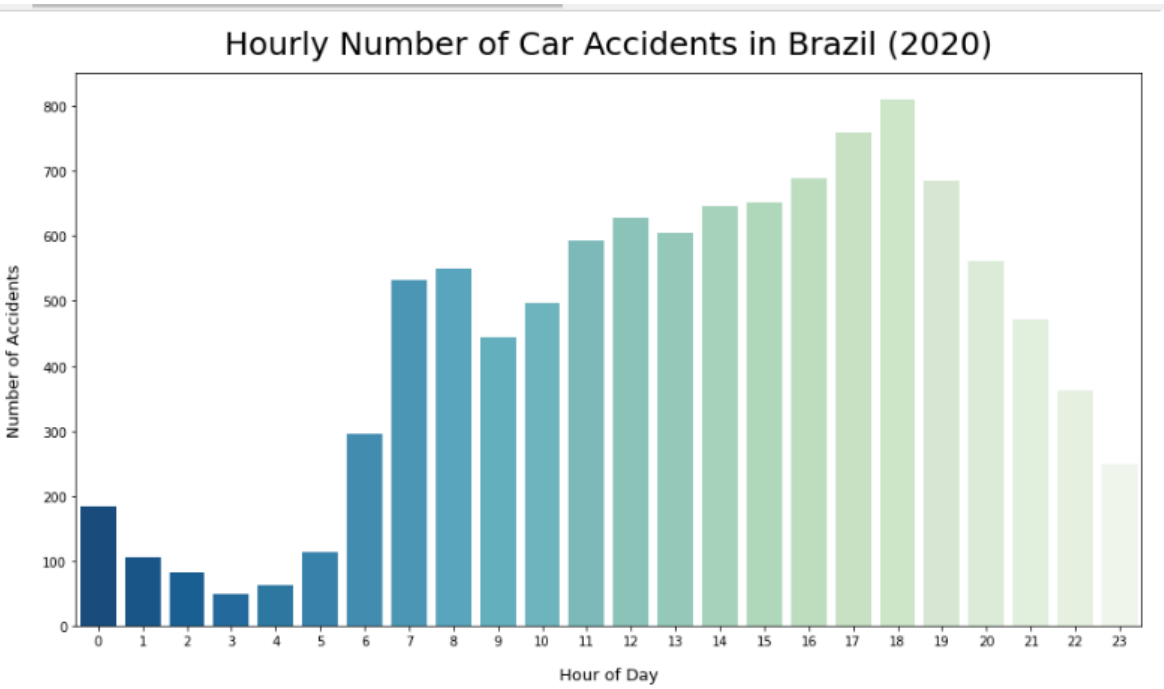
```
['20/03/2020 02:18' '20/03/2020 05:39' '20/03/2020 06:48' ...
 '14/12/2020 07:55' '14/12/2020 10:15' '13/12/2020 16:20']
```

We can see that there are several unique values for Reported Date and Time. The format for the column's value is date followed by time. We will now convert the data type of the reportedDateNTime column from string to datetime so that we can use datetime functions to perform further analysis on the column.

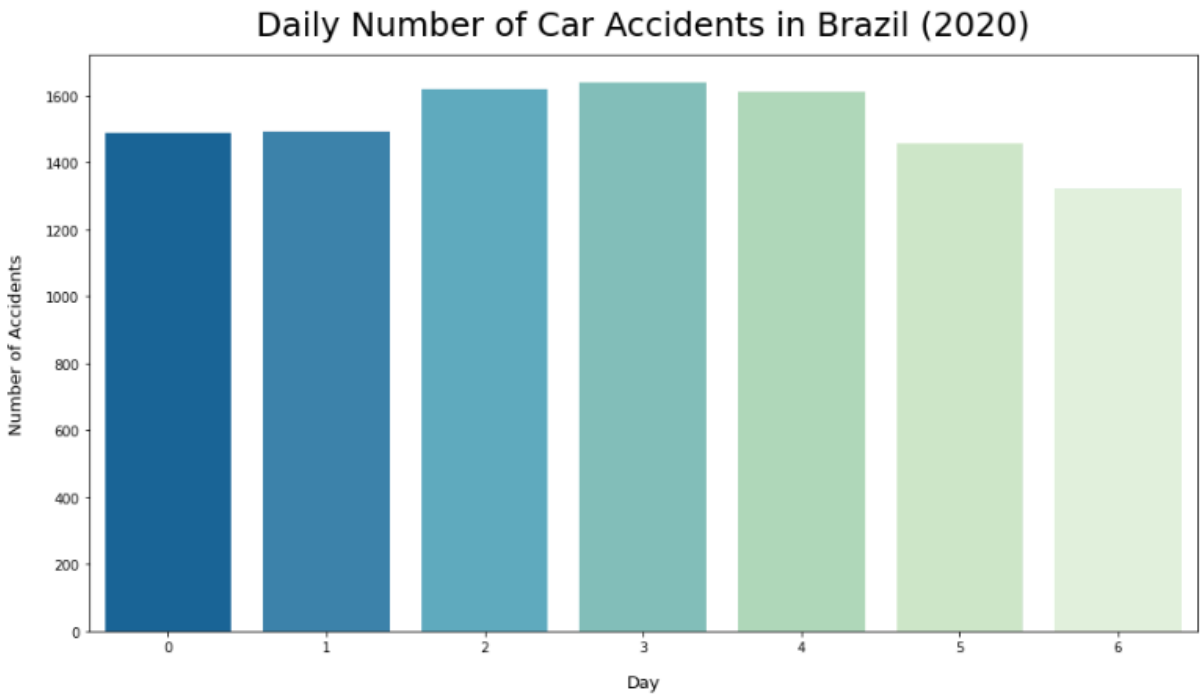
```
from datetime import datetime

forCC['reportedDateNTime']= pd.to_datetime(forCC['reportedDateNTime'])

forCC.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22643 entries, 0 to 22642
Data columns (total 18 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   accidentRecordNum           22643 non-null  object
 1   reportedDateNTime           22643 non-null  datetime64[ns]
```

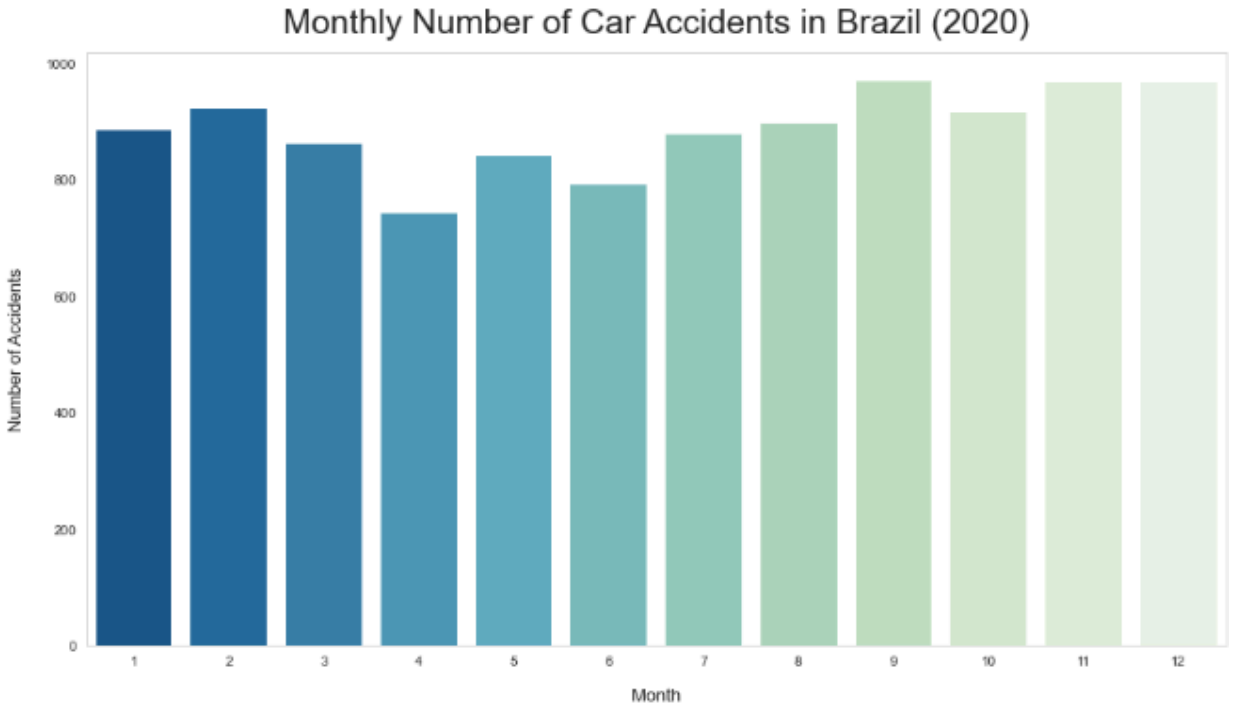
Figure 19: Result of Changing the Data Type of this Column



For the bar plot above, we grouped the records in the dataset by the "Hour" column and then counted the number of unique "accidentRecordNum" in each group. The x axis represents the Hour of Day while the y axis represents the number of car accidents. We can see that the highest peak for the number of accidents is at the 18th hour of day (6pm to 7pm). This means that car accidents occur very often at that time. The number of accidents also peaked at other hours of day, which are the 8th (8am to 9am) and 0th (12am to 1am). The lowest number of car accidents occurred at the 3rd hour of day (3am to 4am).



The highest number of Vehicle Accidents can be found on Thursday while the lowest number of Vehicle Accidents happen on Sunday.



The highest number of accidents occurred on September while the lowest number of accidents occurred on April.

Below, we create 4 additional columns, which are reportedHour, reportedDayOfMonth, reportedDayOfWeek and reportedMonth. The reportedDateNTime by itself is not useful for Machine Learning because each value from this column is a specific date and time. However, we can extract components from the reportedDateNTime, such as the hour of the day, day of the month, day of the week and month, which may be more useful in finding relationship with the severityCode. **This is called Feature Engineering.**

```
from datetime import datetime

forCC = forCC.assign(reportedHour = forCC['reportedDateNTime'])
forCC = forCC.assign(reportedDayOfMonth = forCC['reportedDateNTime'])
forCC = forCC.assign(reportedDayOfWeek = forCC['reportedDateNTime'])
forCC = forCC.assign(reportedMonth = forCC['reportedDateNTime'])

forCC['reportedHour'] = pd.to_datetime(forCC["reportedHour"]).dt.hour
forCC['reportedDayOfMonth'] = pd.to_datetime(forCC["reportedDayOfMonth"]).dt.day
forCC['reportedDayOfWeek'] = pd.to_datetime(forCC["reportedDayOfWeek"]).dt.dayofweek
forCC['reportedMonth'] = pd.to_datetime(forCC["reportedMonth"]).dt.month

forCC
```

Figure 20 Creation of 4 New Columns

c) noPeopleInvolved

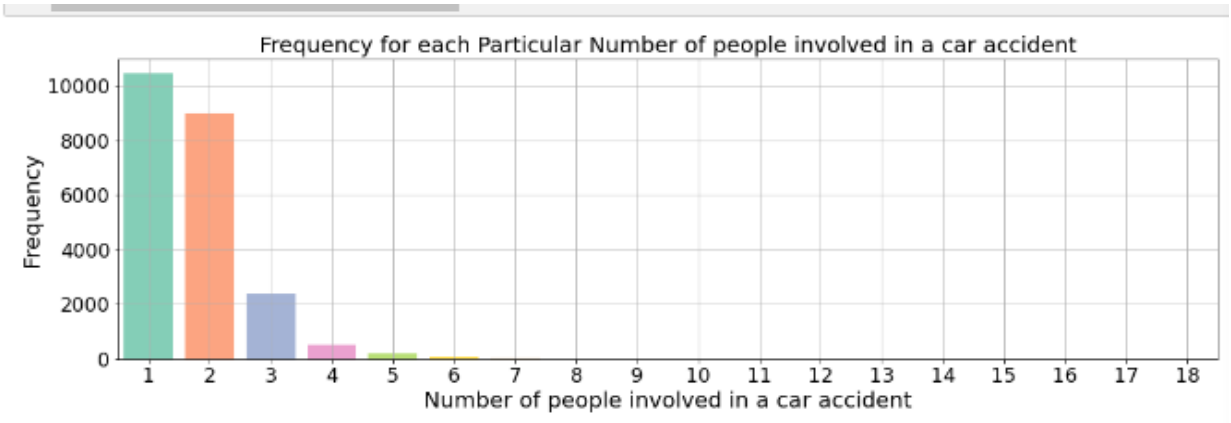
```
print(df['noPeopleInvolved'].value_counts())
```

```
1     10453
2      8997
3      2363
4       532
5       176
6        69
7        23
8        11
9         6
12        3
11        2
10        2
17        1
15        1
18        1
14        1
13        1
16        1
Name: noPeopleInvolved, dtype: int64
```

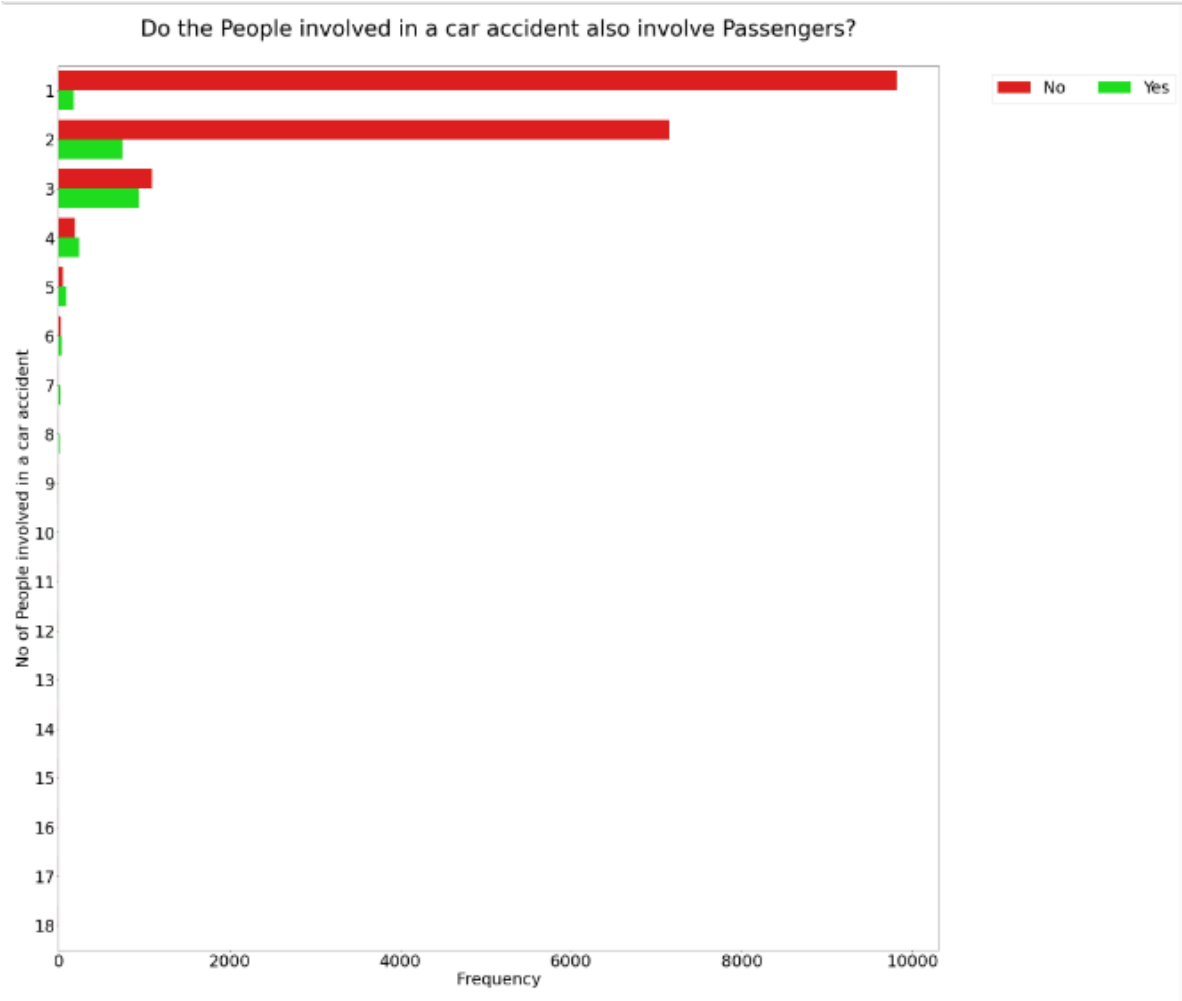
```
print(df['noPeopleInvolved'].unique())
```

```
[ 1  2  3  4  5  6  7  8  9 11 12 13 14 15 16 17 18 10]
```

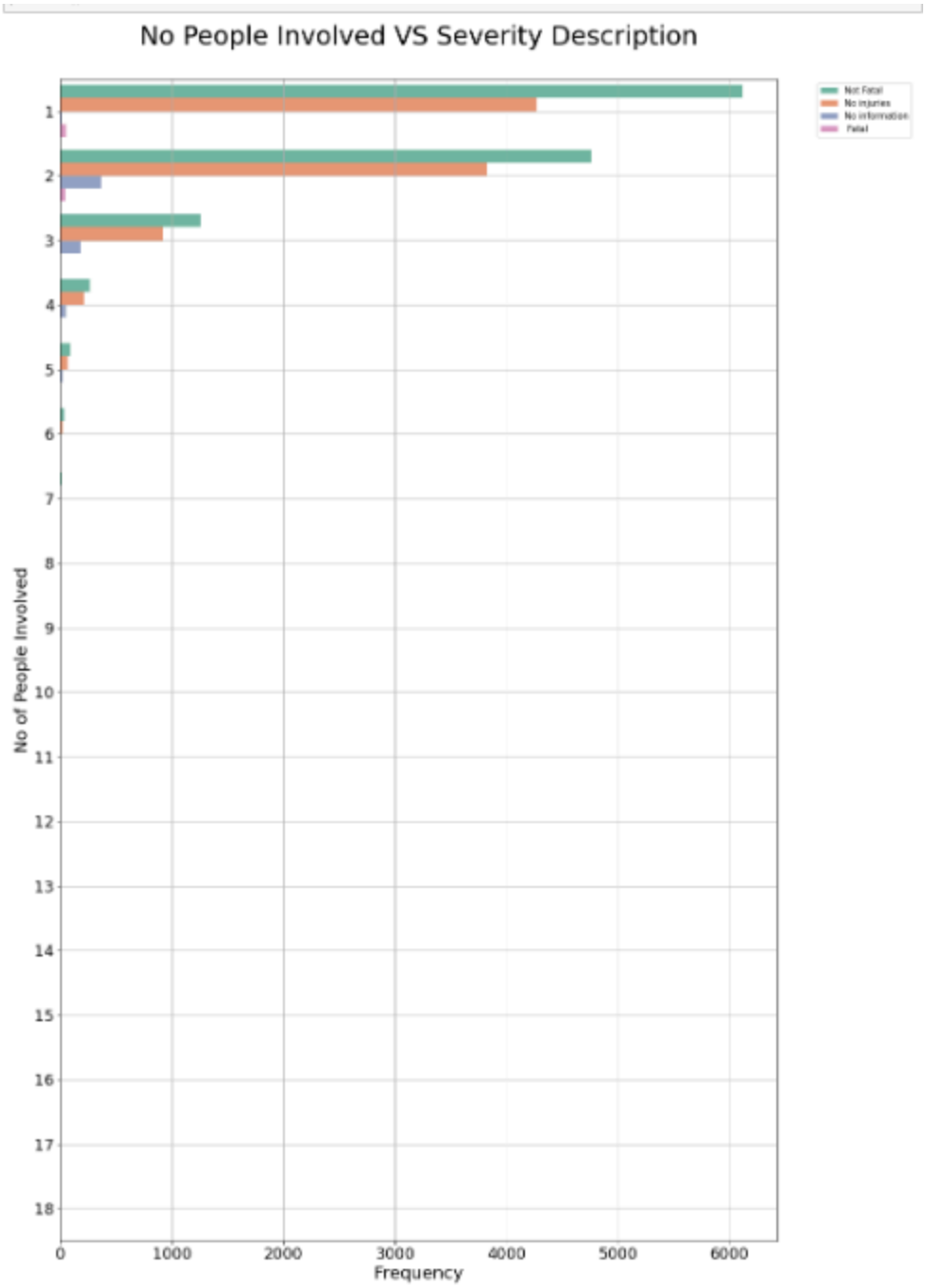
It is a discrete variable. There are several unique values, ranging from 1 to 16.



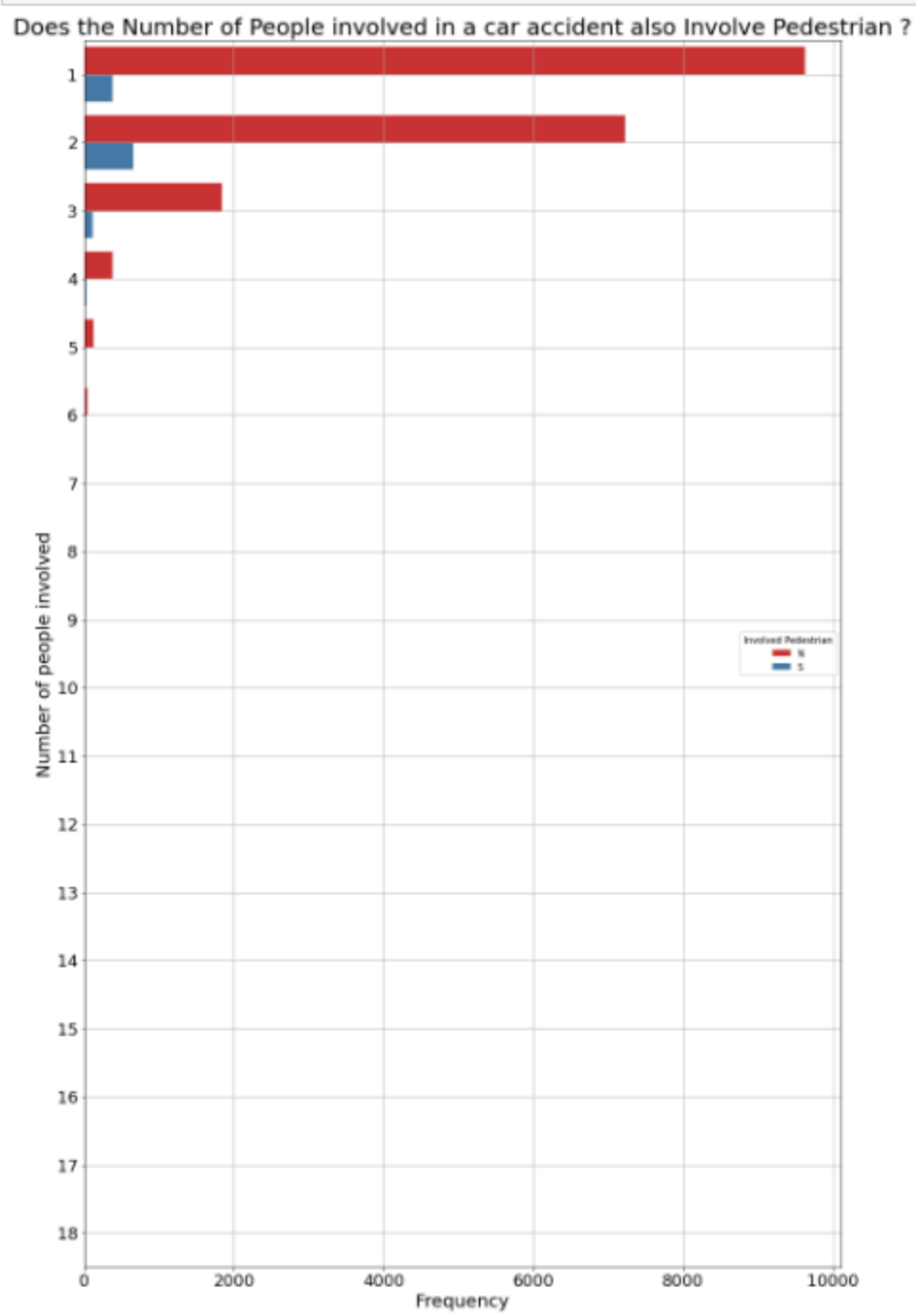
Findings: From the chart above, we can identify that most of the car accidents involved only one individual. As the number of people involved in a car accident increases, its likelihood decreases.



The reason why we want to plot this chart is to determine the relationship between the Number of People Involved In a Car Accident and Whether Passengers were involved in the car accident. As shown in the chart, we can analyze that there is also a likelihood that the passenger is also involved in the car accident. In cases where only 1 person is involved in the car accident, majority of those cases did not involve passengers. But for cases where 2 or 3 people were involved in the accident, the likelihood of a passenger(s) being involved in the car accident is more likely. In cases where 4 people are involved in an accident, the majority of those cases involved passengers. This shows that the more people are involved in an accident, the higher the chance that the accident involved a passenger.



The reason why we want to plot this chart is to determine the relationship between the number of people involved and the severity description. As shown on the chart, most of the cases are not fatal. But there are a few cases that are fatal, which are accidents involving 1 to 2 people.



The reason why we want to plot this chart is to determine whether the presence of a pedestrian in a car accident is related to the number of people involved in a car accident. As shown in the chart, majority of the car accidents did not involve pedestrians. However some of the cases where 1 to 4 people are involved in a car accident involved pedestrians.

d) driverResponsibleForAccident

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

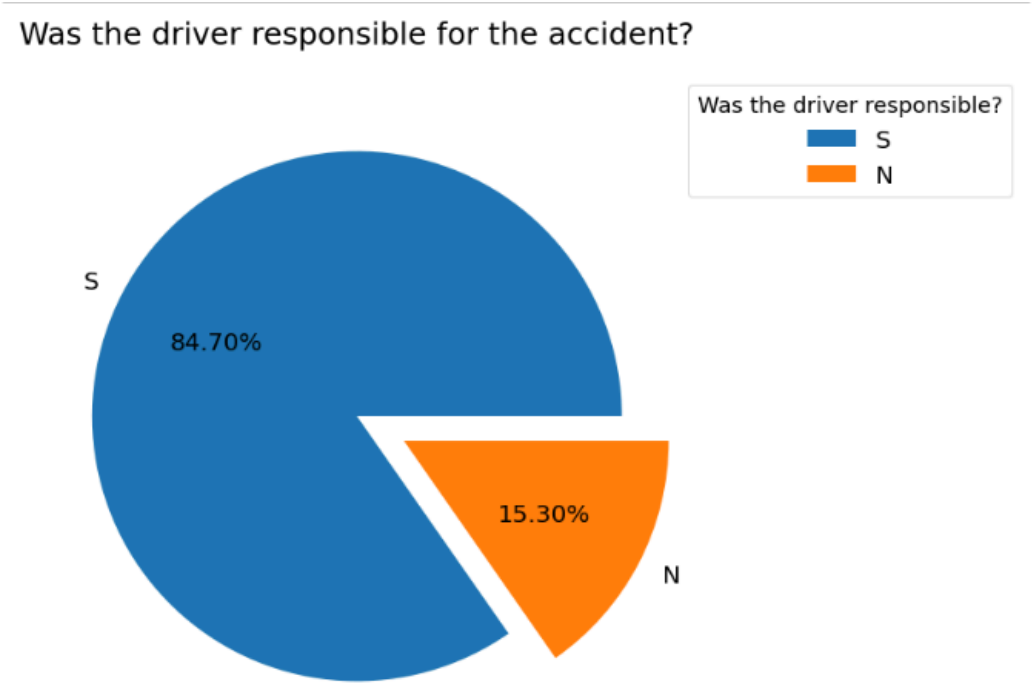
```
S    19178
N     3465
Name: driverResponsibleForAccident, dtype: int64
```

The unique values for this column are 'S' (Yes) and 'N' (No).

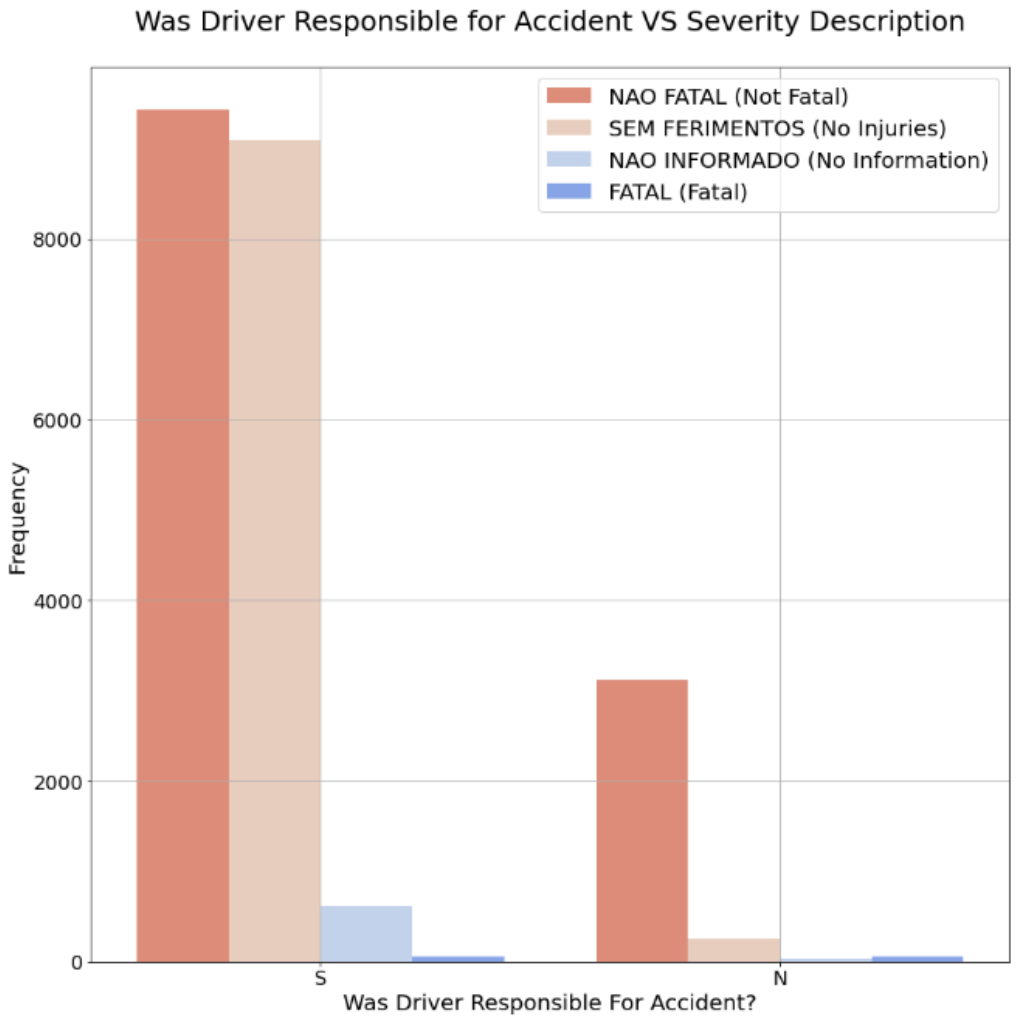
```
print(df['driverResponsibleForAccident'].unique())
```

```
['S' 'N']
```

This is a binary column. This feature has 2 unique values: S represents "Yes" while N represents "No".



From the pie chart, we can conclude that in the majority of the accidents, the driver was responsible for the accidents.



From the count plot, most of the "Not fatal", "No injuries", "No information" and "Fatal" accidents came from cases where the driver was responsible for the accident.

Below, we convert the values in this column to numerical so that we can calculate correlation coefficient and possibly for modelling.

```
In [50]: forCC['driverResponsibleForAccident'] = forCC['driverResponsibleForAccident'].replace(['N'], 0)

In [51]: forCC['driverResponsibleForAccident'] = forCC['driverResponsibleForAccident'].replace(['S'], 1)

In [52]: print(forCC['driverResponsibleForAccident'].value_counts())

1    19178
0     3465
Name: driverResponsibleForAccident, dtype: int64
```

Figure 21: Conversion into a Numerical Column

e) Age

The Unique Values for this feature:

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

0     1278
24     775
23     768
22     713
25     694
      ...
86       4
89       3
87       3
90       2
91       1
Name: age, Length: 92, dtype: int64
```

This is a discrete column. The maximum and minimum ages can be found below:

```
print("Maximum age: ", df['age'].max())
print("Minimum age: ", df['age'].min())

Maximum age:  91
Minimum age:  0
```

We will implement binning to categorise the age values. Binning is a technique that groups data into bins. It basically replaces values contained within a small interval with one representative value for that interval. Binning enhances accuracy in predictive models sometimes. We will bin every person's age into age groups based on the following intervals (converting numeric to categorical data):

1. 0 - 10 years old
2. 11 - 20 years old
3. 21 - 30 years old
4. 31 - 40 years old
5. 41 - 50 years old
6. 51 - 60 years old
7. 61 - 70 years old
8. 71 - 80 years old
9. 81 - 90 years old
10. 91 - 100 years old

Result of binning without label

```
bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
forCC = forCC.assign(ageCategory=forCC.age)
forCC.ageCategory = pd.cut(x = forCC['age'], bins = bins, include_lowest = True)
forCC.ageCategory.value_counts()
```

```
56]: (20.0, 30.0]      6743
     (30.0, 40.0]      5522
     (40.0, 50.0]      3805
     (50.0, 60.0]      2247
     (10.0, 20.0]      1531
     (-0.001, 10.0]    1430
     (60.0, 70.0]       975
     (70.0, 80.0]       326
     (80.0, 90.0]        63
     (90.0, 100.0]        1
     Name: ageCategory, dtype: int64
```

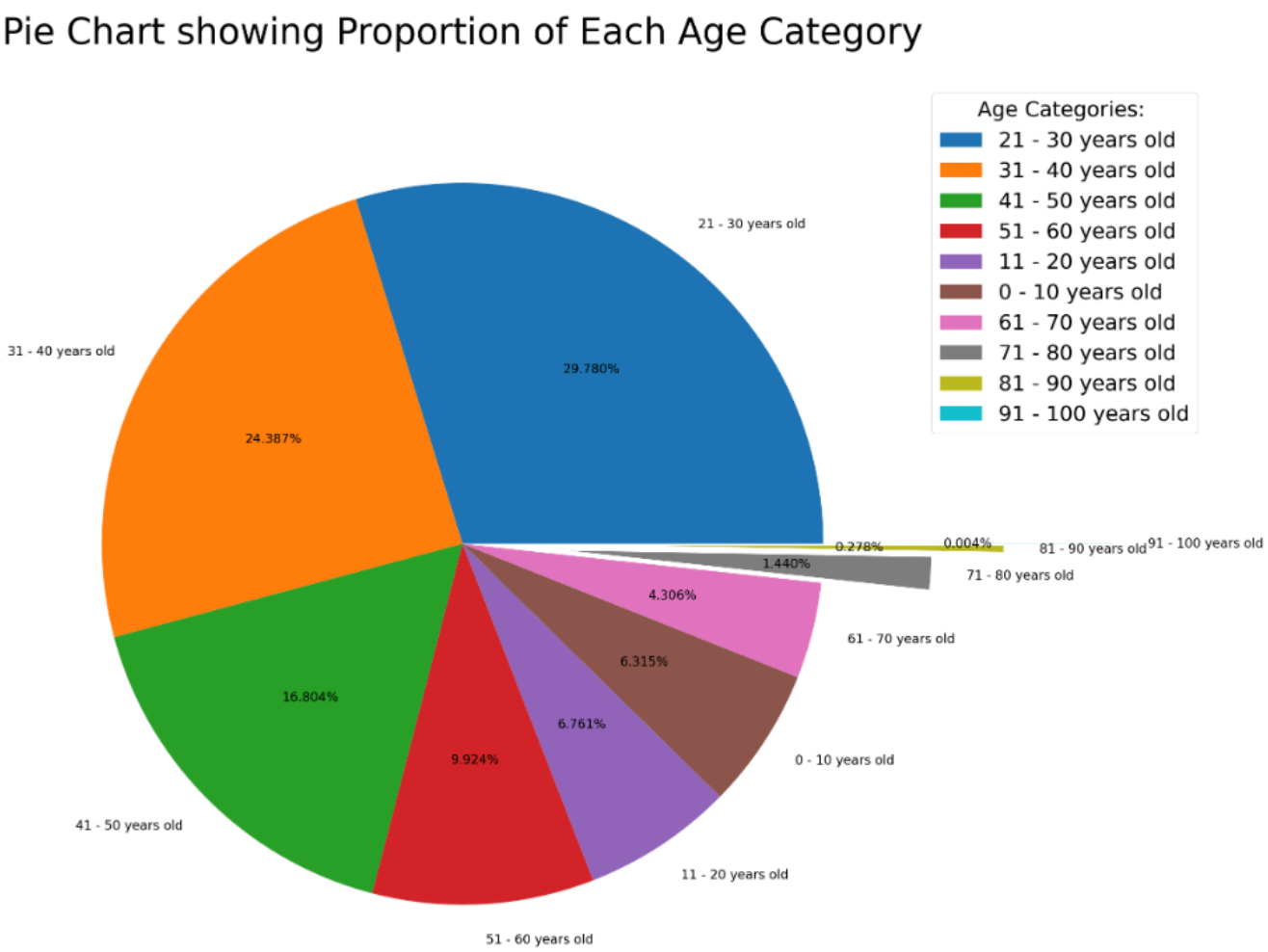
Result of binning with label

```
bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
labels = ['0 - 10 years old', '11 - 20 years old', '21 - 30 years old', '31 - 40 years old', '41 - 50 years old', '51 - 60 ye
forCC = forCC.assign(ageCategory=forCC.age)
forCC.ageCategory = pd.cut(x = forCC['age'], bins = bins, labels = labels, include_lowest = True)
forCC.ageCategory.value_counts()
```

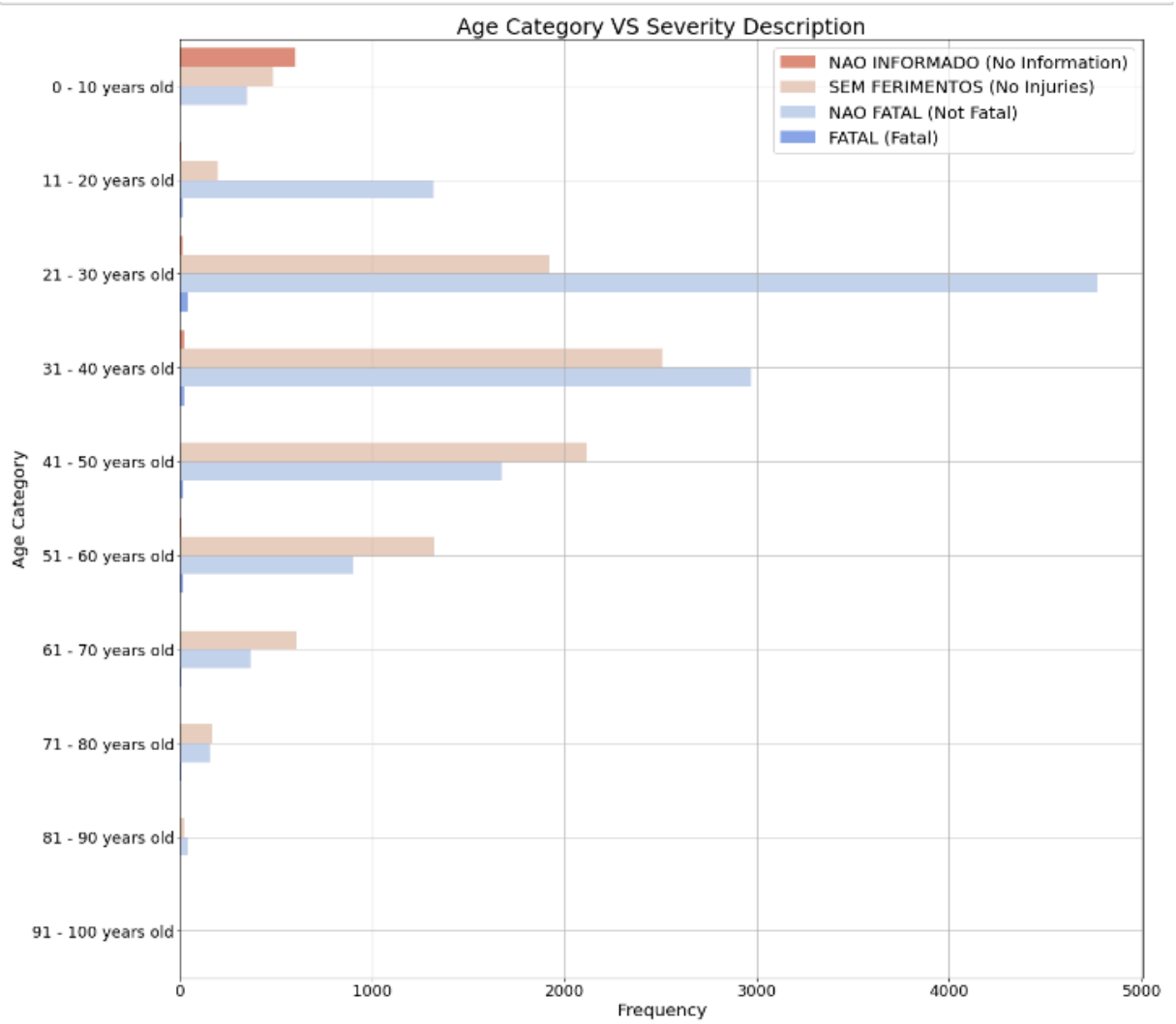
```
7]: 21 - 30 years old     6743
    31 - 40 years old     5522
    41 - 50 years old     3805
    51 - 60 years old     2247
    11 - 20 years old     1531
    0 - 10 years old      1430
    61 - 70 years old      975
    71 - 80 years old      326
    81 - 90 years old       63
    91 - 100 years old       1
    Name: ageCategory, dtype: int64
```

Figure 22: Creation of ageCategory Column

A new column called ageCategory has been added to the dataset dataframe.



From the pie chart, we can see that the majority of car accidents involved people from the 21 - 30 years old age group (29.780%). However, only 0.004% of the accidents involved people aged between 91 and 100 years old inclusive.



From the countplot, we can see that the majority of the "Not Fatal" and "Fatal" accidents came from the 21 - 30 years old age group. However, the majority of the "No Information" accidents came from the 0 - 10 years old age group while the majority of the "No Injuries" accidents came from the 31 - 40 years old age group.

f) sex

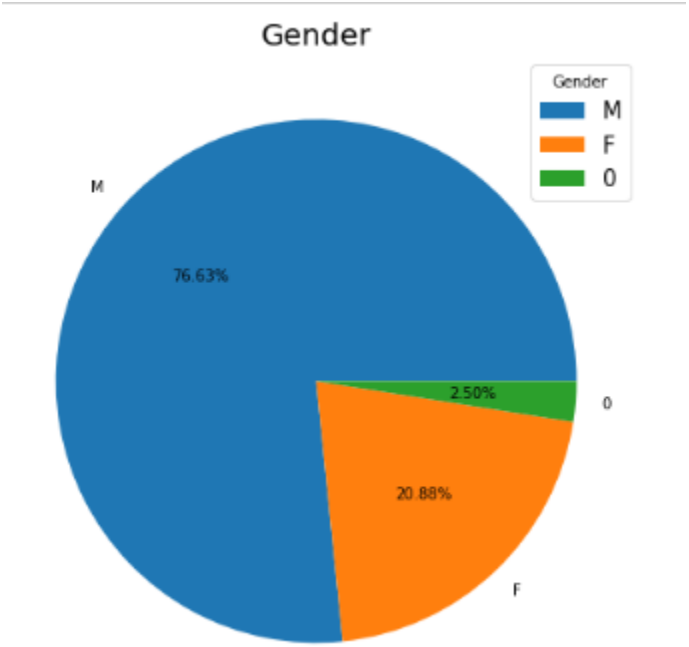
```
print(df['sex'].value_counts())
```

```
M    17351
F     4727
0      565
Name: sex, dtype: int64
```

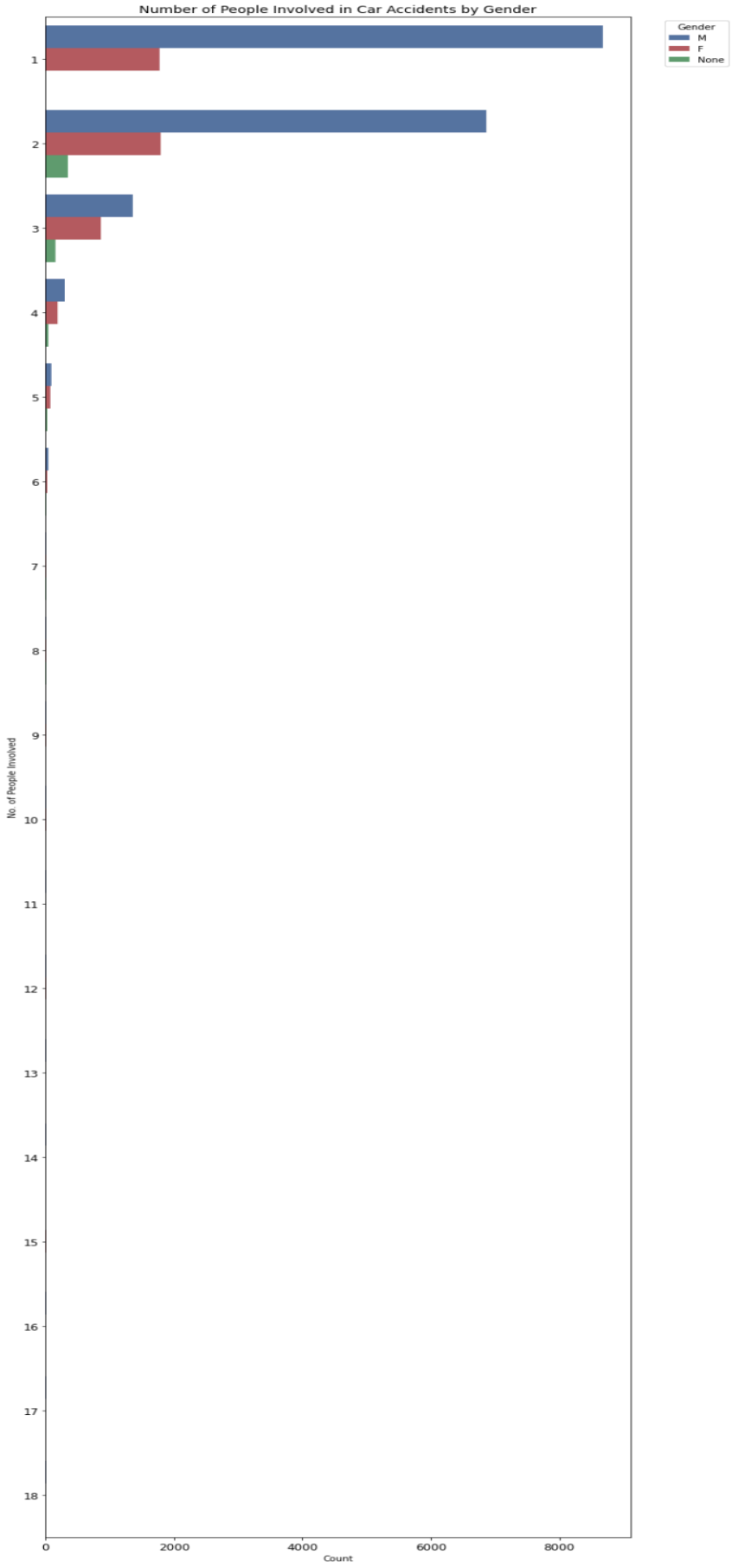
```
print(df['sex'].unique())
```

```
['M' 'F' '0']
```

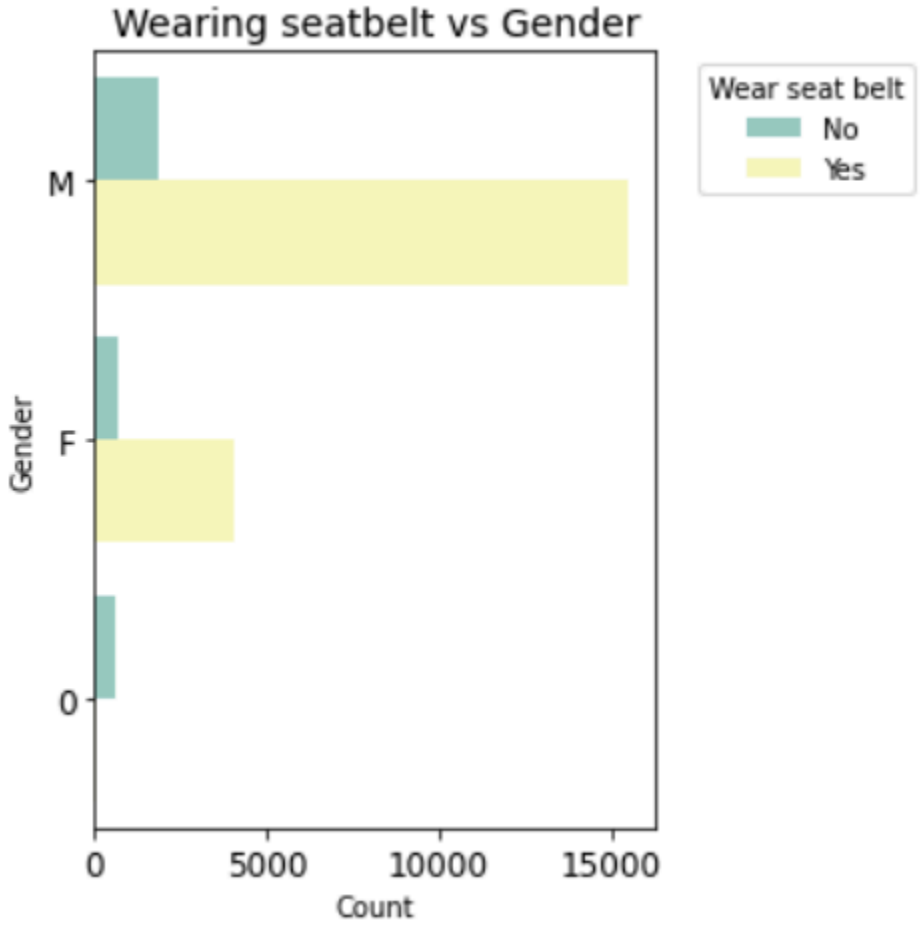
The unique values for this column are 'M', 'F' and '0'. Although one may think that this is a binary variable, it turns out that it is nominal. However, the reason '0' is there could be because it represents either no information or a null value.



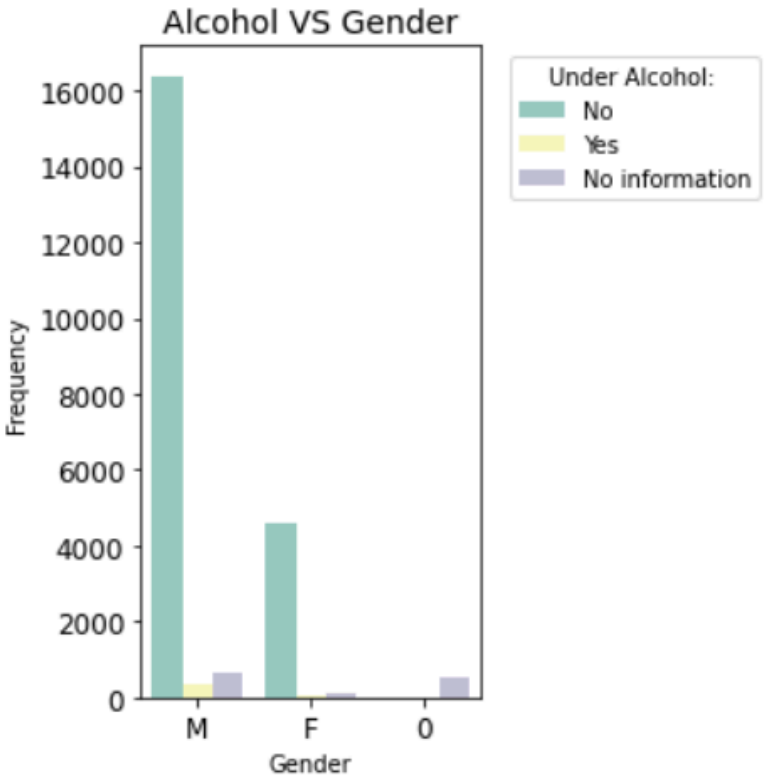
As shown above most of the accidents involved males.


Number of People Involved in Car Accidents by Gender
Gender
M
F
None
No. of People Involved
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
0
2000
4000
6000
8000
Count

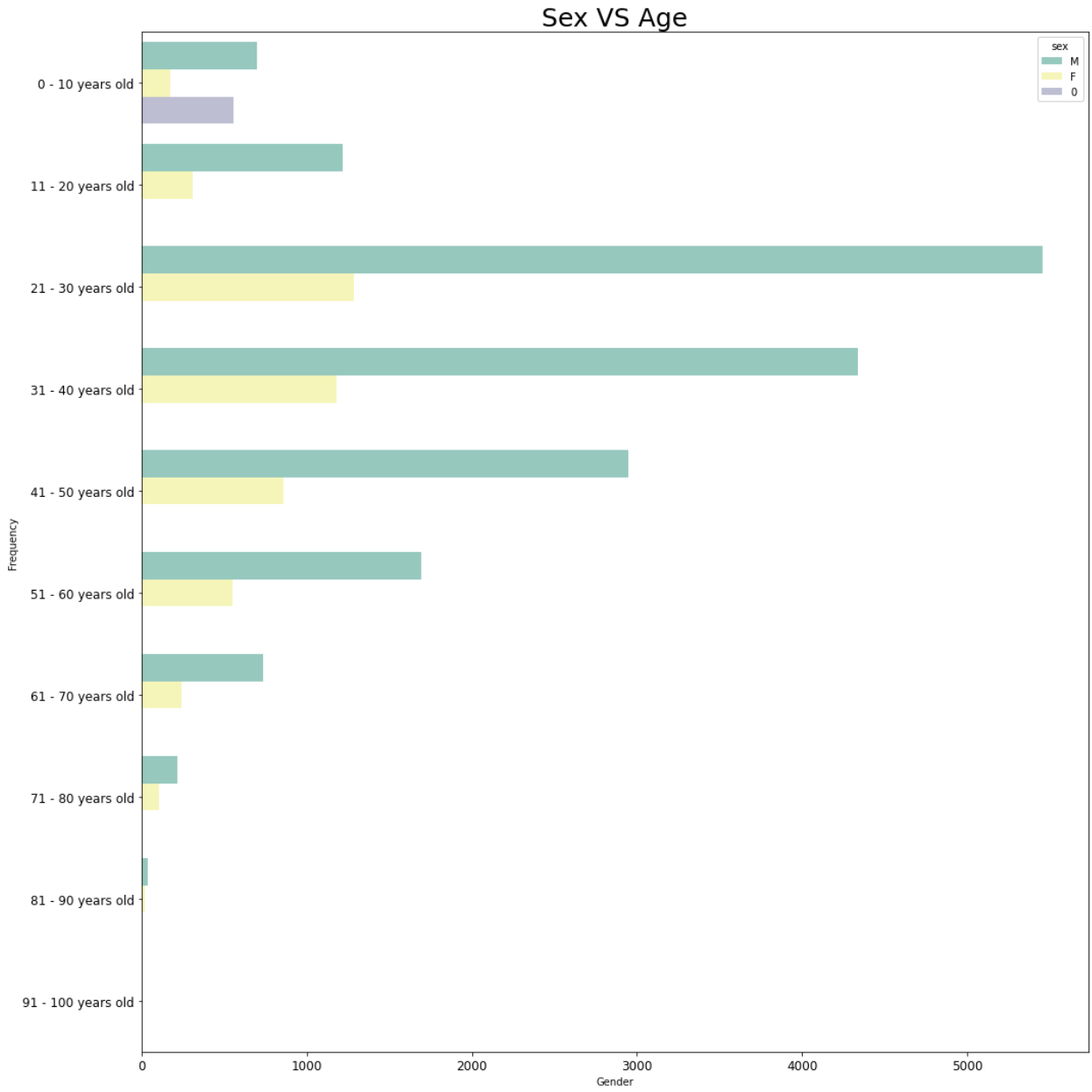
The reason why we want to plot the chart on the previous page is to determine whether gender has an influence on the number of people involved in the car accident. As shown in the chart, the majority of the the car accidents involved male. Females are also involved but not as many as the males. According to the chart above, for cases where 1 to 6 people are involved in a car accident, most of the people involved are male. For cases where only 1 person is involved, the majority of the car accidents involved males.



The reason why we want to plot this chart is to see the relationship between gender and whether seat belt was worn. Most of the males and females had a seat belt on before the car accident, but some of the males and females did not wear any seat belts. This shows that Males are more likely to wear a seat belt compared to females.



The reason why we want to plot this chart is to determine whether there is a link between gender and the influence of alcohol in a car accident. As shown in the chart, majority of males and females were not under the influence of alcohol. But, a few of the males and females were under the influence of alcohol.



In each age category, the majority of the people were males.
Below, we make the column numeric for calculating correlation coefficients and possibly modelling.

```
forCC['sex'] = forCC['sex'].replace(['0'], 2)
```

```
forCC['sex'] = forCC['sex'].replace(['M'], 1)
```

```
forCC['sex'] = forCC['sex'].replace(['F'], 0)
```

f) woreSeatBelt

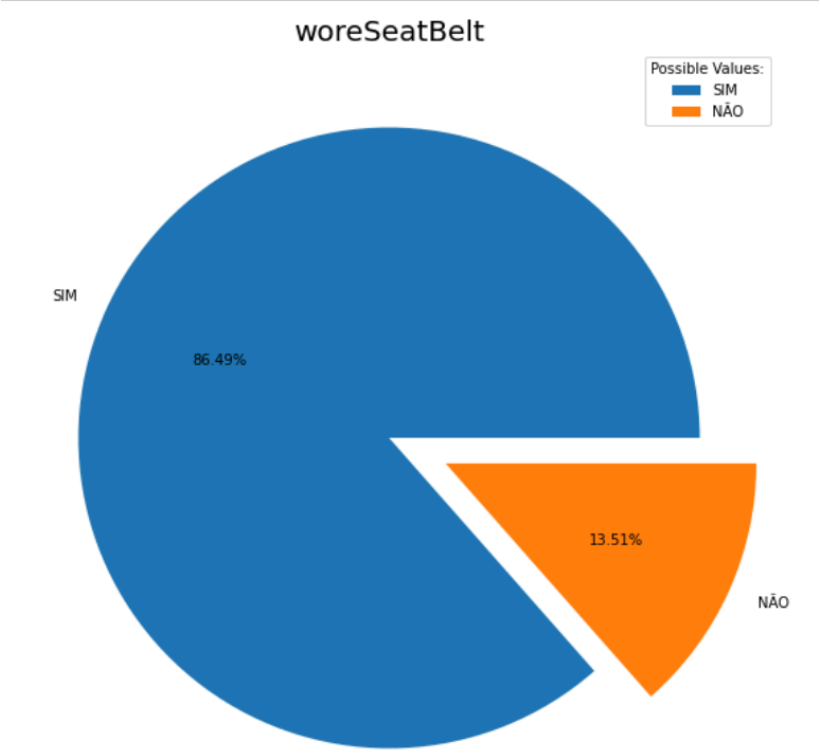
```
print(df['woreSeatBelt'].value_counts())
```

```
SIM    19583
NÃO     3060
Name: woreSeatBelt, dtype: int64
```

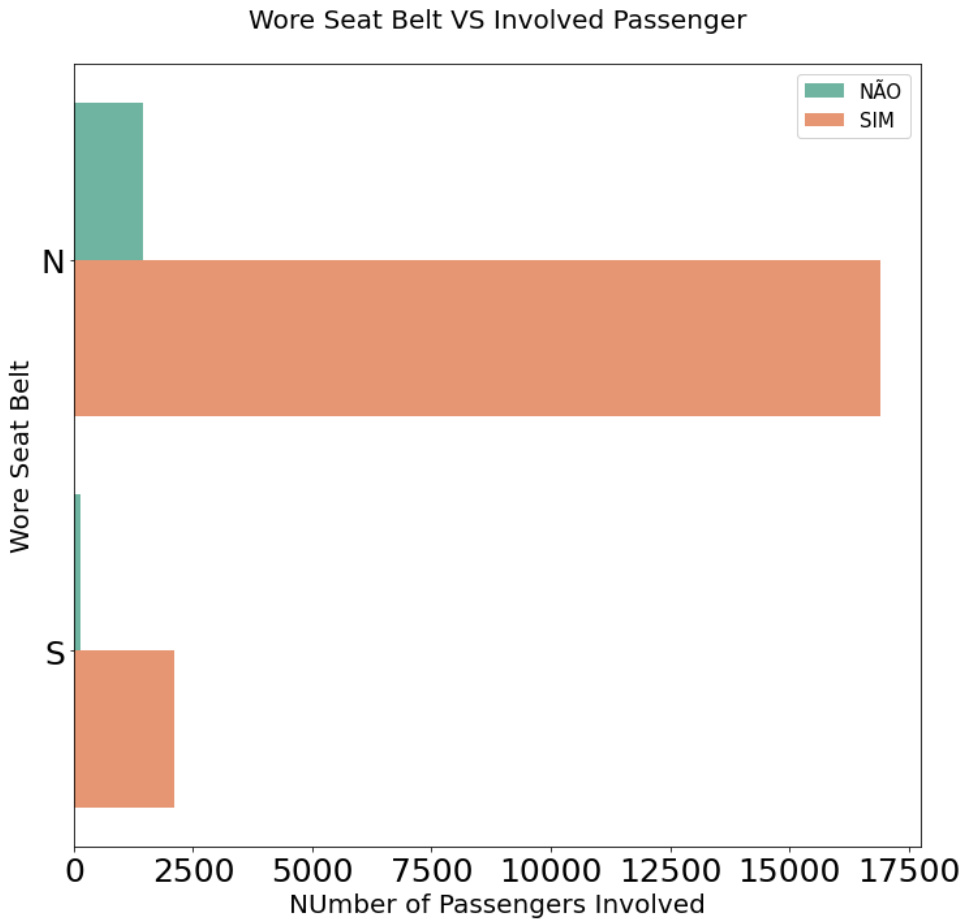
```
print(df['woreSeatBelt'].unique())
```

```
['NÃO' 'SIM']
```

It is binary. The unique values are 'SIM' (Yes) and 'NÃO' (No).



In most of the car accidents, the majority of the drivers wore seat belt.



In both cases where seat belt was worn or was not worn, most of those cases involved passengers.

Below, we make the column numerical so that we can calculate correlation coefficients and possibly for modelling.

```
forCC['woreSeatBelt'] = forCC['woreSeatBelt'].replace(['NÃO'], 0)
```

```
forCC['woreSeatBelt'] = forCC['woreSeatBelt'].replace(['SIM'], 1)
```

g) driverUnderAlcohol

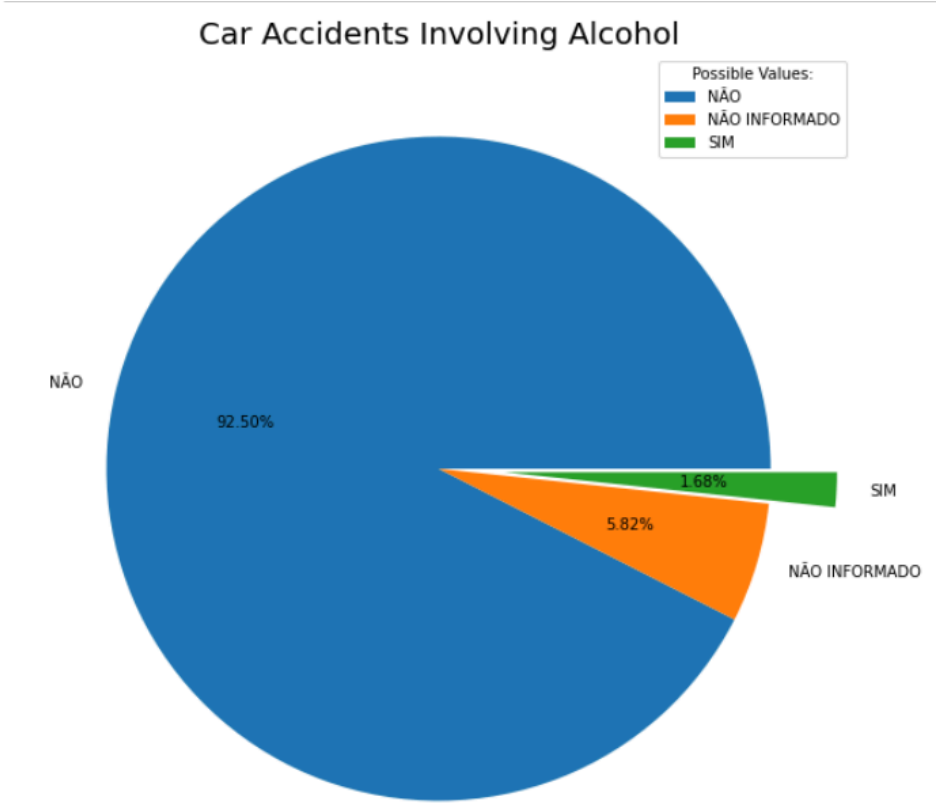
```
print(df['driverUnderAlcohol'].value_counts())
```

```
NÃO              20944
NÃO INFORMADO     1318
SIM                381
Name: driverUnderAlcohol, dtype: int64
```

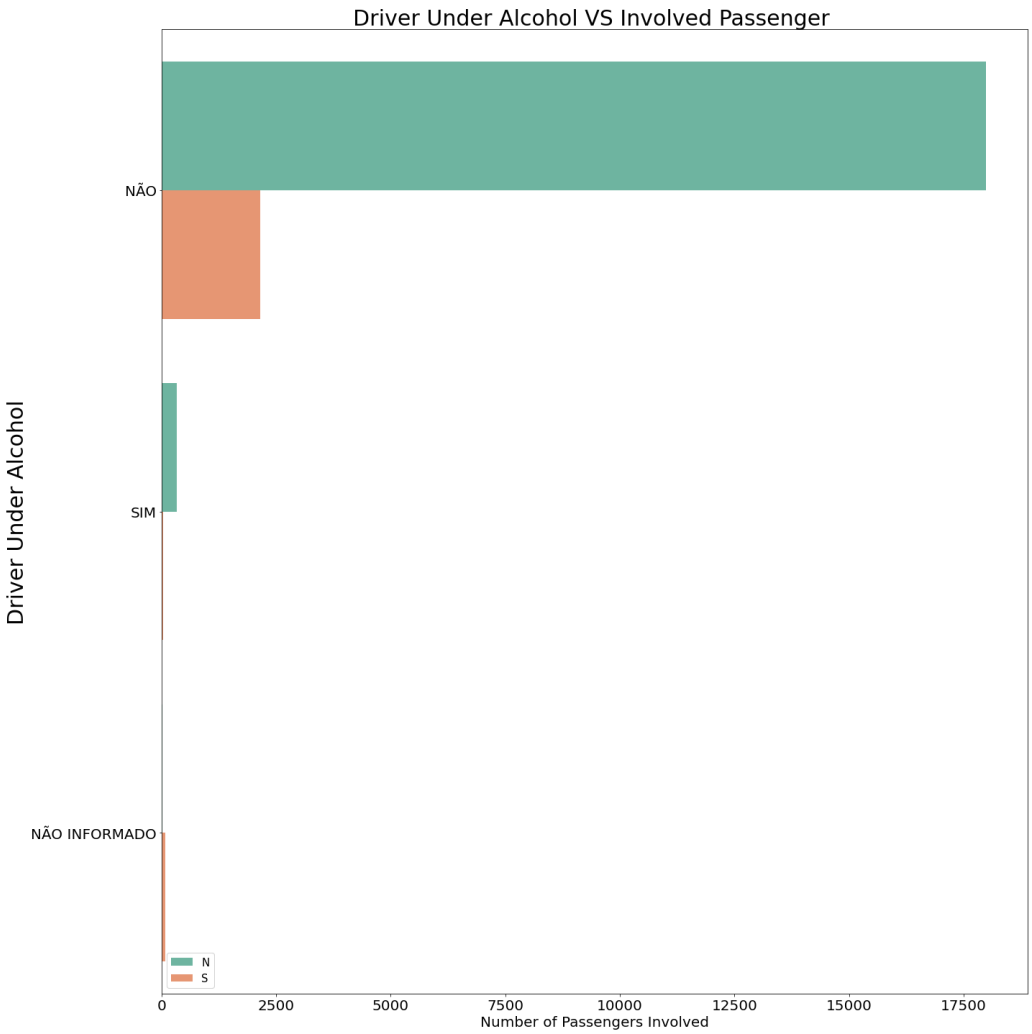
```
print(df['driverUnderAlcohol'].unique())
```

```
['NÃO' 'SIM' 'NÃO INFORMADO']
```

The possible values are 'NÃO', 'SIM' and 'NÃO INFORMADO'.



The majority of the accidents did not involve drivers who were under alcohol, followed by no information and drivers who were under alcohol.



In cases where the driver was under alcohol or not under alcohol, most of those cases did not involve passengers.

Below, we make the column numerical so that we can calculate correlation coefficients and possibly for modelling.

```
forCC['driverUnderAlcohol'] = forCC['driverUnderAlcohol'].replace(['NÃO'], 0)
```

```
forCC['driverUnderAlcohol'] = forCC['driverUnderAlcohol'].replace(['SIM'], 1)
```

```
forCC['driverUnderAlcohol'] = forCC['driverUnderAlcohol'].replace(['NÃO INFORMADO'], 2)
```

h) birthDate

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

```
00/00/0000    1504
29/12/1997       9
19/02/1993       9
23/04/1993       8
07/10/1986       8
              ...
16/06/1971       1
03/05/1974       1
04/07/1959       1
26/02/1958       1
22/10/1980       1
Name: birthDate, Length: 12042, dtype: int64
```

```
print(df['birthDate'].unique())
```

```
['06/09/1984' '11/03/1978' '20/11/1987' ... '17/03/1957' '12/07/2001'
 '07/09/1988']
```

It follows the format 'DD/MM/YYYY'. There is an abnormal value in the column, which is 00/00/0000. Therefore, we try to run additional queries to compare birthDate with age.

```
In [91]: dfResult1 = df.loc[(df['birthDate'] == '00/00/0000') & (df['age'] == 0)]

         dfResult1
```

Out[91]:

n	reportedDateNTime	noPeopleInvolved	driverResponsibleForAccident	severityCode	severityDesc	sex	woreSeatBelt	driverUnderAlcohol	age	birthDate
1	16/04/2020 07:11	3	S	3	SEM FERIMENTOS	M	NÃO	NÃO INFORMADO	0	00/00/0000
1	08/04/2020 19:00	2	S	1	NAO FATAL	M	NÃO	NÃO INFORMADO	0	00/00/0000
1	16/04/2020 16:33	2	S	1	NAO FATAL	M	NÃO	NÃO INFORMADO	0	00/00/0000
1	12/04/2020 02:00	2	S	0	NAO INFORMADO	0	NÃO	NÃO INFORMADO	0	00/00/0000
1	16/04/2020 19:00	2	N	1	NAO FATAL	M	NÃO	NÃO INFORMADO	0	00/00/0000
...	...	...	...	...	...	...	...	...	...	...
1	11/12/2020 20:00	2	S	3	SEM FERIMENTOS	M	NÃO	NÃO INFORMADO	0	00/00/0000
1	13/12/2020 19:16	3	S	3	SEM FERIMENTOS	0	NÃO	NÃO INFORMADO	0	00/00/0000
1	13/12/2020 21:45	2	S	1	NAO FATAL	M	NÃO	NÃO INFORMADO	0	00/00/0000
1	13/12/2020 21:43	6	S	0	NAO INFORMADO	M	NÃO	NÃO INFORMADO	0	00/00/0000
1	11/12/2020 22:30	2	S	0	NAO INFORMADO	0	NÃO	NÃO INFORMADO	0	00/00/0000

1276 rows x 18 columns (Figure showing records where birthDate is 00/00/0000 and age is equal to 0.)

```
dfResult2 = df.loc[(df['birthDate'] == '00/00/0000') & (df['age'] != 0)]

dfResult2
```

]:

im	reportedDateNTime	noPeopleInvolved	driverResponsibleForAccident	severityCode	severityDesc	sex	woreSeatBelt	driverUnderAlcohol	age	birthDate
01	19/03/2020 20:15	2	N	1	NAO FATAL	M	NÃO	NÃO	31	00/00/0000
01	16/04/2020 08:15	3	S	3	SEM FERIMENTOS	M	SIM	NÃO	55	00/00/0000
01	07/03/2020 02:19	2	N	2	FATAL	M	NÃO	NÃO	60	00/00/0000
01	18/04/2020 16:30	2	N	1	NAO FATAL	F	NÃO	NÃO	15	00/00/0000
01	19/04/2020 20:51	2	N	1	NAO FATAL	M	NÃO	NÃO	37	00/00/0000
...	...	...	...	...	...	...	...	...	...	...
01	24/11/2020 18:48	2	N	1	NAO FATAL	M	NÃO	NÃO	55	00/00/0000
01	12/12/2020 12:01	3	N	1	NAO FATAL	0	SIM	NÃO	35	00/00/0000
01	28/12/2020 18:20	2	S	1	NAO FATAL	M	NÃO	NÃO	28	00/00/0000
01	13/12/2020 07:17	3	N	1	NAO FATAL	M	NÃO	SIM	24	00/00/0000
01	13/12/2020 16:20	2	S	1	NAO FATAL	M	SIM	NÃO	26	00/00/0000

228 rows × 18 columns (Figure showing records where birthDate is 00/00/0000 but age is not 0.)

```
dfResult3 = df.loc[(df['birthDate'] != '00/00/0000') & (df['age'] == 0)]

dfResult3
```

]:

eForAccident	severityCode	severityDesc	sex	woreSeatBelt	driverUnderAlcohol	age	birthDate	qualificationCat	qualificationDesc	involvedDeath	oldSeve
N	1	NAO FATAL	F	NÃO	NÃO	0	22/10/2019		...	0	
N	3	SEM FERIMENTOS	M	SIM	NÃO	0	05/12/2019		...	0	

2 rows x 18 columns (Figure showing records where age is 0 but birthDate is not 00/00/0000)

From these queries, we can conclude that there were birth date and age entry errors. A possible reason for both birthDate to be 00/00/0000 and age to be 0 could be because those values were not available. In the Data Preparation stage, we wil deal with this issue.

i) qualificationCat

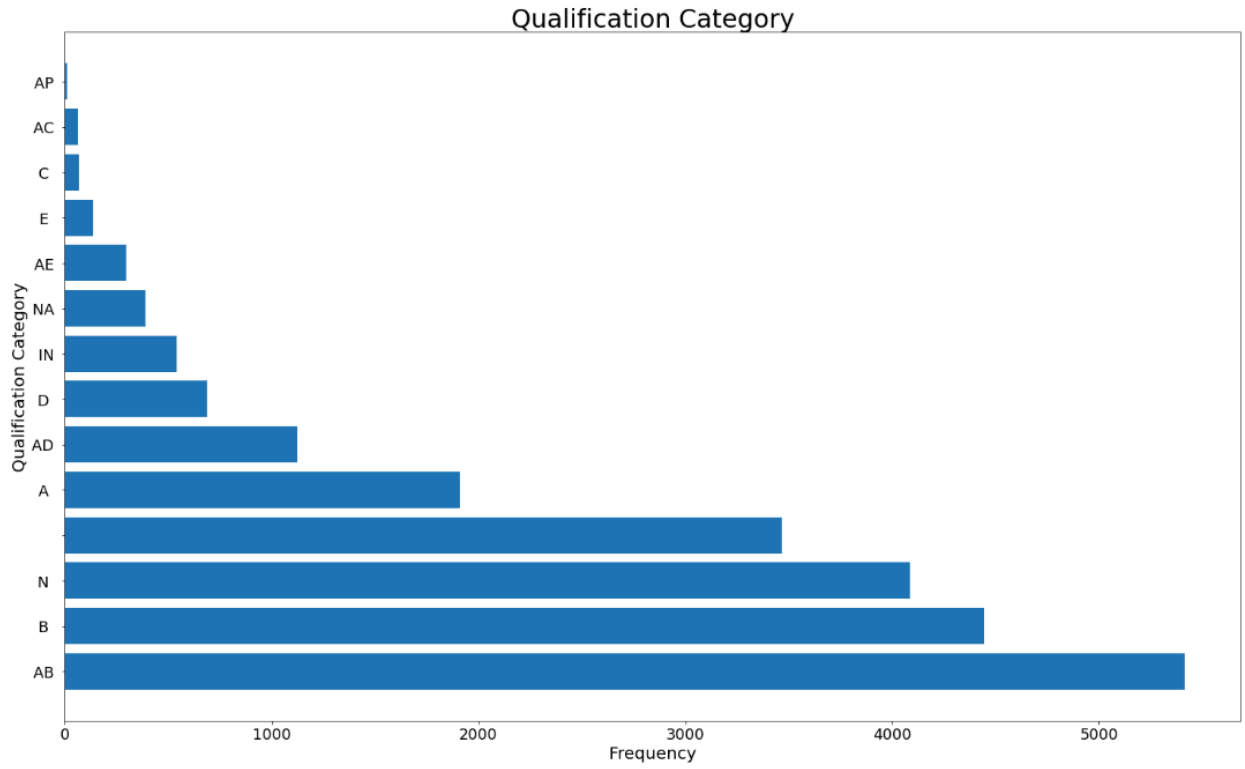
```
print(df['qualificationCat'].value_counts())
```

```
AB      5413
B       4445
N       4084
        3467
A       1908
AD      1123
D        688
IN       541
NA       391
AE       298
E        138
C         70
AC        66
AP        11
Name: qualificationCat, dtype: int64
```

```
print(df['qualificationCat'].unique())
```

```
['B  ' 'N  ' '   ' 'AE ' 'AB ' 'C  ' 'D  ' 'A  ' 'IN ' 'AD ' 'NA ' 'AC '
 'E  ' 'AP ']
```

This is a nominal variable has 13 unique values, excluding the blank space. This column seems to correspond to the qualificationDesc column on the next page.

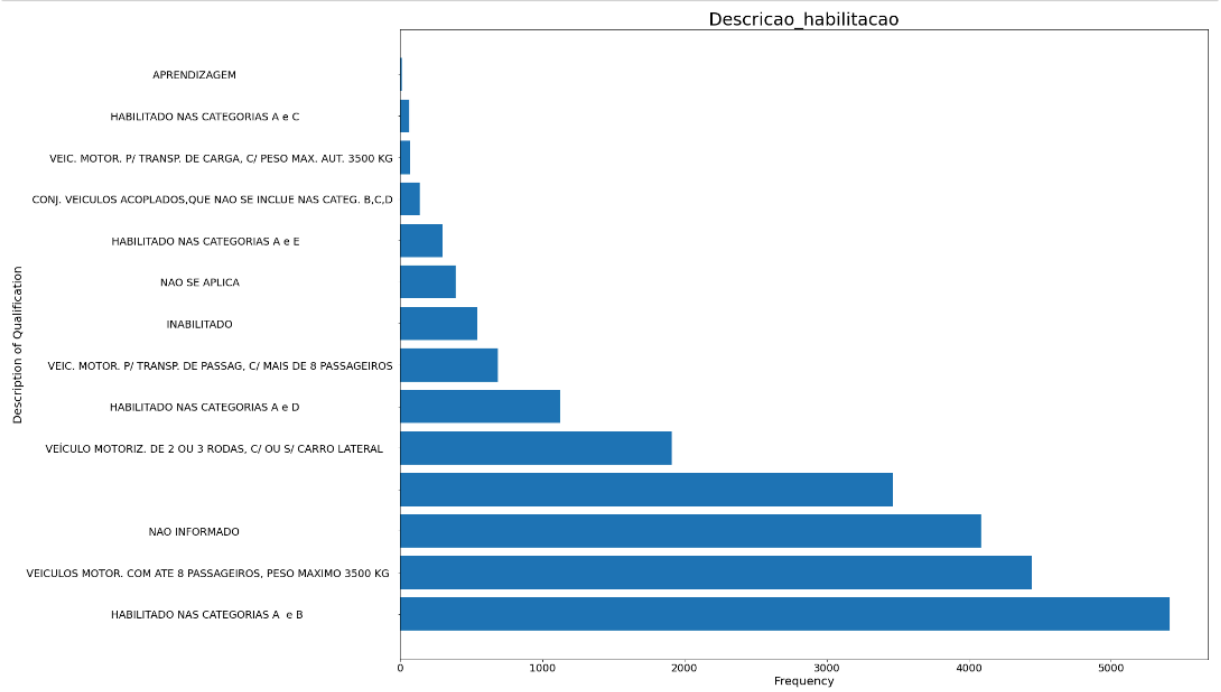


j) qualificationDesc

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

```
HABILITADO NAS CATEGORIAS A  e B                              5413
VEICULOS MOTOR. COM ATE 8 PASSAGEIROS, PESO MAXIMO 3500 KG    4445
NAO INFORMADO                                                 4084
                                                              3467
VEÍCULO MOTORIZ. DE 2 OU 3 RODAS, C/ OU S/ CARRO LATERAL      1908
HABILITADO NAS CATEGORIAS A e D                               1123
VEIC. MOTOR. P/ TRANSP. DE PASSAG, C/ MAIS DE 8 PASSAGEIROS    688
INABILITADO                                                    541
NAO SE APLICA                                                  391
HABILITADO NAS CATEGORIAS A e E                                298
CONJ. VEICULOS ACOPLADOS,QUE NAO SE INCLUE NAS CATEG. B,C,D    138
VEIC. MOTOR. P/ TRANSP. DE CARGA, C/ PESO MAX. AUT. 3500 KG     70
HABILITADO NAS CATEGORIAS A e C                                 66
APRENDIZAGEM                                                    11
Name: qualificationDesc, dtype: int64
```

This is a nominal variable that seems to be the description of the qualification categories on the previous page.



k) vehicleType

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

```
AUTOMOVEL                           9769
MOTOCICLETA                         9102
                                    1207
ONIBUS                               849
BICICLETA                            474
CAMINHONETE                          402
CAMINHAO                             382
CAMIONETA                            141
MOTONETA                              92
NAO INFORMADO                         57
CAMINHAO-TRATOR                       56
MICROONIBUS                           43
CICLOMOTOR                            19
KOMBI                                 14
REBOQUE E SEMI-REBOQUE                 9
CARRO DE MAO                           8
CARROCA                                7
BONDE                                  4
TRICICLO                               3
TRATOR DE RODAS                        3
PATINETE                               1
TRATOR MISTO                           1
Name: vehicleType, dtype: int64
```

This is a nominal variable that has 20 unique values, excluding 'NAO INFORMADO' and the blank space. Those 2 values are the abnormal values.

Below are the translations for the vehicle types:

AUTOMOVEL	- Car
MOTOCICLETA	- Motorcycle
ONIBUS	- Bus
BICICLETA	- Bicycle
CAMINHONETE	- Pickup Truck
CAMINHAO	- Truck
CAMIONETA	- Truck
MOTONETA	- Motor Scooter
NAO INFORMADO	- No Information
CAMINHAO-TRATOR	- Tractor Truck
MICROONIBUS	- Micro Bus
CICLOMOTOR	- Cyclomotor
KOMBI	- Kombi
REBOQUE E SEMI-REBOQUE	- TRAILER AND SEMI-TRAILER
CARRO DE MAO	- Hand Car
CARROCA	- Cart
BONDE	- Tram
TRATOR DE RODAS	- Wheel Tractor
TRICICLO	- Tricycle
PATINETE	- Scooter
TRATOR MISTO	- Mixed Tractor

To calculate correlation coefficient, all the values must be numerical so we perform the following, including on the ‘NAO INFORMADO’ and blank spaces.

```
In [130]: forCC['vehicleType'] = forCC['vehicleType'].replace(['AUTOMOVEL          '], 0)
In [131]: forCC['vehicleType'] = forCC['vehicleType'].replace(['MOTOCICLETA       '], 1)
In [132]: forCC['vehicleType'] = forCC['vehicleType'].replace(['                  '], 2)
In [133]: forCC['vehicleType'] = forCC['vehicleType'].replace(['ONIBUS            '], 3)
In [134]: forCC['vehicleType'] = forCC['vehicleType'].replace(['BICICLETA         '], 4)
In [135]: forCC['vehicleType'] = forCC['vehicleType'].replace(['CAMINHONETE       '], 5)
In [136]: forCC['vehicleType'] = forCC['vehicleType'].replace(['CAMINHAO          '], 6)
In [137]: forCC['vehicleType'] = forCC['vehicleType'].replace(['CAMIONETA         '], 7)
In [138]: forCC['vehicleType'] = forCC['vehicleType'].replace(['MOTONETA          '], 8)
In [139]: forCC['vehicleType'] = forCC['vehicleType'].replace(['NAO INFORMADO     '], 9)
In [140]: forCC['vehicleType'] = forCC['vehicleType'].replace(['CAMINHAO-TRATOR   '], 10)
In [141]: forCC['vehicleType'] = forCC['vehicleType'].replace(['MICROONIBUS       '], 11)
In [142]: forCC['vehicleType'] = forCC['vehicleType'].replace(['CICLOMOTOR        '], 12)
In [143]: forCC['vehicleType'] = forCC['vehicleType'].replace(['KOMBI             '], 13)
In [144]: forCC['vehicleType'] = forCC['vehicleType'].replace(['REBOQUE E SEMI-REBOQUE'], 14)
In [145]: forCC['vehicleType'] = forCC['vehicleType'].replace(['CARRO DE MAO      '], 15)
In [146]: forCC['vehicleType'] = forCC['vehicleType'].replace(['CARROCA           '], 16)
In [147]: forCC['vehicleType'] = forCC['vehicleType'].replace(['BONDE             '], 17)
In [148]: forCC['vehicleType'] = forCC['vehicleType'].replace(['TRATOR DE RODAS   '], 18)
In [149]: forCC['vehicleType'] = forCC['vehicleType'].replace(['TRICICLO          '], 19)
In [150]: forCC['vehicleType'] = forCC['vehicleType'].replace(['PATINETE          '], 20)
In [151]: forCC['vehicleType'] = forCC['vehicleType'].replace(['TRATOR MISTO      '], 21)
```

l) involvedPedestrian

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

```
N    19284
S     1205
Name: involvedPedestrian, dtype: int64
```

```
print(df['involvedPedestrian'].unique())
```

```
['N' 'S' nan]
```

This is a binary variable that can be either 'N' or 'S'. However, there are 2154 missing values in this column. In the Data Preparation stage, we will deal with the missing values.
To calculate correlation coefficient, all the values must be numerical and no value can be null. That is why we temporarily replaced the null values with 2.

```
forCC['involvedPedestrian'] = forCC['involvedPedestrian'].replace(['N'], 0)
```

```
forCC['involvedPedestrian'] = forCC['involvedPedestrian'].replace(['S'], 1)
```

```
forCC['involvedPedestrian'].fillna(2, inplace = True)
```

m) involvedPassenger

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

```
N    18345
S     2253
Name: involvedPassenger, dtype: int64
```

```
print(df['involvedPassenger'].unique())
```

```
['N' 'S' nan]
```

This is a binary variable that can be either 'N' or 'S'. However, there are 2045 missing values in this column. In the Data Preparation stage, we will deal with the missing values.
To calculate correlation coefficient, all the values must be numerical and no value can be null. That is why we temporarily replaced the null values with 2.

```
forCC['involvedPassenger'] = forCC['involvedPassenger'].replace(['N'], 0)
```

```
forCC['involvedPassenger'] = forCC['involvedPassenger'].replace(['S'], 1)
```

```
forCC['involvedPassenger'].fillna(2, inplace = True)
```

n) involvedDeath

```
print(forCC['involvedDeath'].value_counts())
```

```
0    22643
Name: involvedDeath, dtype: int64
```

```
print(df['involvedDeath'].unique())
```

```
[0]
```

This column has only 1 unique value, which is 0. This column will not be meaningful for our machine learning algorithms and its correlation coefficient with other columns and itself is NaN so we will be removing this column.

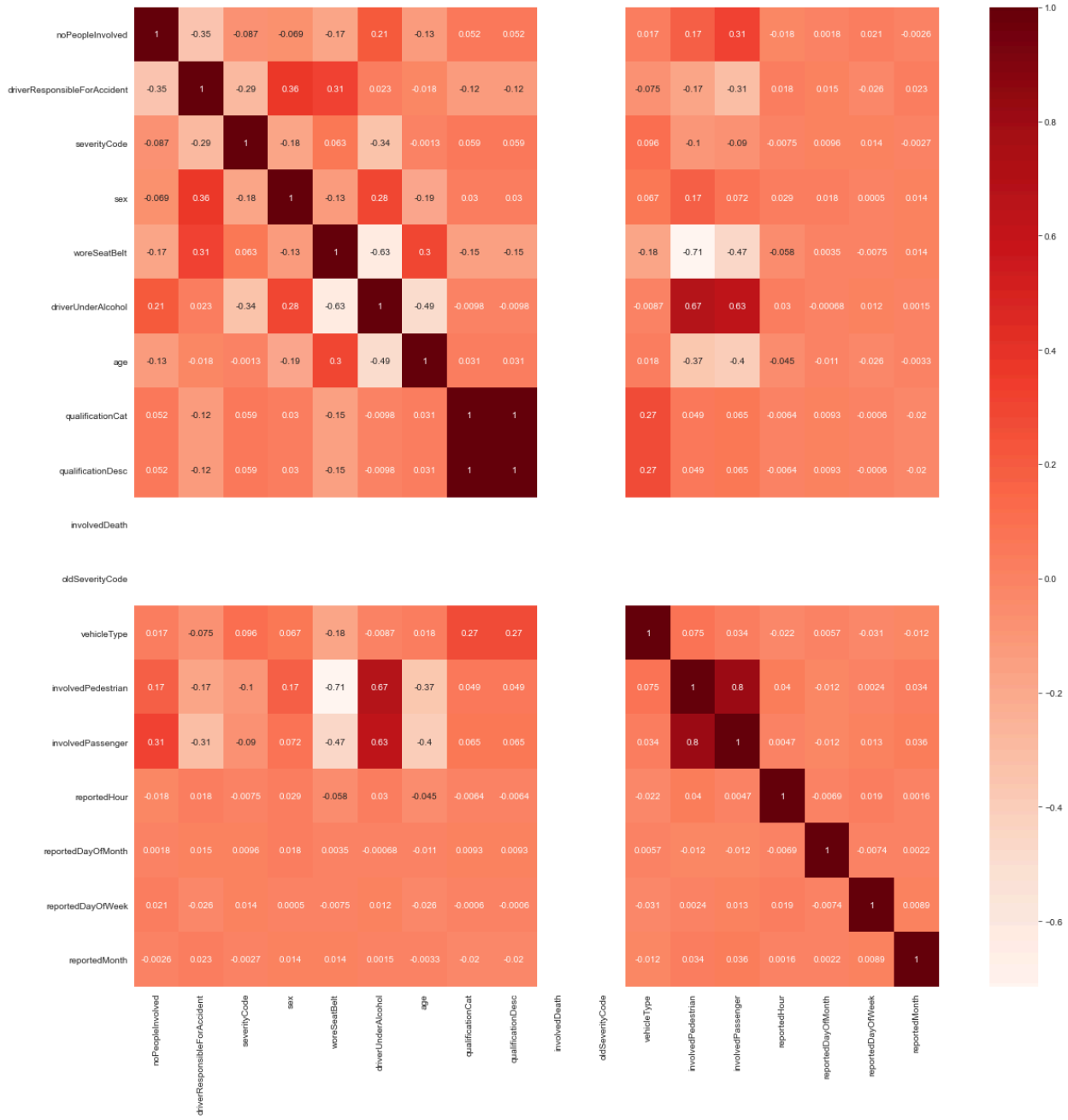


Figure 23: Heatmap Showing Correlation Coefficients between Most of the Columns. We will be using this for Data Preparation later on

C. DATA PREPARATION

In this section, we implemented several techniques, which are:

1. Removing Rows with Abnormal Value for a Particular Important Column
2. Removing Redundant Columns
3. Replace Missing or Invalid Values with Mean, Median or Mode
4. Outlier Detection
5. Feature Selection

(For your information, we balanced the dataset by upsampling at the Modelling Section)

3.1 Remove Unnecessary Rows

3.1.1 Severity Description (severityDesc) & Severity Code (severityCode)

Since the severity code column with value 0 with description *NAO INFORMADO* which means 'No Information' is not useful, we decided to remove it, reducing the number of rows from 22642 to 21995. The content in the severity code is shown in figure below:

```
In [175]: forDP_filtered1 = forDP[forDP['severityCode'] == 0].index
          forDP.drop(forDP_filtered1 , inplace=True)

In [176]: print(forDP['severityCode'].value_counts())

          2    12544
          1     9338
          3      113
          Name: severityCode, dtype: int64
```

3.2.Drop Redundant Columns

3.2.1 Severity Description (severityDesc)

Based on the previous section, severity code and severity description are similar where they share the same purpose. Therefore, the severity description column will be dropped using the code below.

```
In [179]: forDP = forDP.drop(['severityDesc'], axis=1)
```

Below is a snippet of the final result after the column is dropped.

```
In [181]: forDP.head(15)
```

Out[181]:

	accidentRecordNum	reportedDateNTime	noPeopleInvolved	driverResponsibleForAccident	severityCode	sex	woreSeatBelt
0	2020-014152383-001	2020-03-20 02:18:00	1	1	2	1	0
1	2020-014152383-001	2020-03-20 02:18:00	2	1	1	0	1
2	2020-014152383-001	2020-03-20 02:18:00	3	0	1	1	1
3	2020-014158612-001	2020-03-20 05:39:00	1	0	2	0	1
4	2020-014158612-001	2020-03-20 05:39:00	2	1	1	1	1
5	2020-014161105-001	2020-03-20 06:48:00	1	1	1	1	1

3.2.2 Old Severity Code (oldSeverityCode)

Old severity code column is removed because it only has one unique value which is 0. We have no use for this so it is dropped using the code below.

```
In [183]: forDP = forDP.drop(['oldSeverityCode'], axis=1)
```

3.2.3 Involved Death (involvedDeath)

Involved death column is removed because it only has one unique value which is 0. We have no use of this so it is dropped using the code below.

```
In [186]: forDP = forDP.drop(['involvedDeath'], axis=1)
```

3.2.4 Accident Record Number (accidentRecordNum)

Accident Record Number column contains too many unique values, which is not usable for machine learning, so it is dropped using the code below.

```
In [189]: forDP = forDP.drop(['accidentRecordNum'], axis=1)
```

3.2.5 Qualification Description (qualificationDesc)

Relationship between qualificationCat and qualificationDesc before being converted into numerical variables is shown below

QualificationCat	QualificationDesc
AB (0)	HABILITADO NAS CATEGORIAS A e B
B (1)	VEICULOS MOTOR. COM ATE 8 PASSAGEIROS, PESO MAXIMO 3500 KG
N (2)	NAO INFORMADO
A (4)	VEÍCULO MOTORIZ. DE 2 OU 3 RODAS, C/ OU S/ CARRO LATERAL
AD (5)	HABILITADO NAS CATEGORIAS A e D
D (6)	VEIC. MOTOR. P/ TRANSP. DE PASSAG, C/ MAIS DE 8 PASSAGEIROS
IN (7)	INABILITADO
NA (8)	NAO SE APLICA
AE (9)	HABILITADO NAS CATEGORIAS A e E
E (10)	CONJ. VEICULOS ACOPLADOS,QUE NAO SE INCLUE NAS CATEG. B,C,D
C (11)	VEIC. MOTOR. P/ TRANSP. DE CARGA, C/ PESO MAX. AUT. 3500 KG
AC (12)	HABILITADO NAS CATEGORIAS A e C
AP (13)	APRENDIZAGEM

Both qualificationCat and qualificationDesc are actually the same thing except that one is in the form of a code while the other is the description based on the code. Therefore, we will drop the qualificationDesc column using the code below.

```
In [193]: forDP = forDP.drop(['qualificationDesc'], axis=1)
```

3.2.6 Reported Date and Time (reportedDateAndTime)

In the data understanding section, we created four columns from the reportedDateNTime column. They are reportedHour, reportedDayOfMonth, reportedDayOfWeek and reportedMonth. The reason for doing that is so that we can extract general features, such as hour, month and day of the week. Since we have extracted those components, it is no longer necessary to keep the reportedDateNTime column. So, we will drop the reportedDateNTime column.

```
In [195]: forDP = forDP.drop(['reportedDateNTime'], axis=1)
```

3.3 Replace Missing or Invalid Values

3.3.1 Age

One of the values in Age is 0, which is illogical. To fix this, a comparison is made between age and birthdate column to determine the right action to be taken to deal with the 0 age value.

```
In [201]: forDPResult1 = forDP.loc[(forDP['birthDate'] == '00/00/0000') & (forDP['age'] == 0)]

          forDPResult1
```

Out[201]:

	noPeopleInvolved	driverResponsibleForAccident	severityCode	sex	woreSeatBelt	driverUnderAlcohol	age	birthDate
80	3	1	1	1	0	2	0	00/00/0000
98	2	1	2	1	0	2	0	00/00/0000
100	2	1	2	1	0	2	0	00/00/0000
135	2	0	2	1	0	2	0	00/00/0000
154	3	0	2	0	1	0	0	00/00/0000
...	...	...	...	...	...	...	...	...
22356	3	1	1	2	0	2	0	00/00/0000
22434	3	1	2	1	0	2	0	00/00/0000
22562	2	1	1	1	0	2	0	00/00/0000
22611	3	1	1	2	0	2	0	00/00/0000
22617	2	1	2	1	0	2	0	00/00/0000

From this, we obtained records with birthdate of 00/00/0000 and age of 0. However, there are still data with age as 0 but not having birthdate of 00/00/0000 as shown below.

```
In [202]: forDPResult2 = forDP.loc[(forDP['birthDate'] != '00/00/0000') & (forDP['age'] == 0)]

          forDPResult2
```

Out[202]:

	noPeopleInvolved	driverResponsibleForAccident	severityCode	sex	woreSeatBelt	driverUnderAlcohol	age	birthDate
8614	3	0	2	0	0	0	0	22/10/2019
11902	6	0	1	1	1	0	0	05/12/2019

Since 0 is the outlier data, it needs to be replaced in order to keep the accuracy of the models. The solution is to replace all the 0s with the mean (not including those rows with age as 0) which is calculated in the figure below.

```
In [205]: ageMean = forDP.loc[forDP['age'] > 0, 'age'].mean()
          print(math.floor(ageMean))

          36
```

```
In [206]: forDP['age'] = forDP['age'].replace([0], math.floor(ageMean))
```

```
In [207]: forDP[forDP['age'] == math.floor(ageMean)]
```

Out[207]:

	noPeopleInvolved	driverResponsibleForAccident	severityCode	sex	woreSeatBelt	driverUnderAlcohol	age	birthDate
24	2	1	1	1	1	0	36	16/03/1984
36	5	1	1	1	1	0	36	20/11/1983
40	9	0	3	1	1	0	36	03/08/1983
80	3	1	1	1	0	2	36	00/00/0000
98	2	1	2	1	0	2	36	00/00/0000
...	...	...	...	...	...	...	...	...
22568	2	1	1	0	1	0	36	19/04/1984
22597	2	1	2	1	1	0	36	28/09/1984
22611	3	1	1	2	0	2	36	00/00/0000

Shown above, the age 0 is successfully replaced with 36. Next, we drop the birthdate column as it serves the same purpose as the age column as shown below.

```
In [208]: forDP = forDP.drop(['birthDate'], axis=1)
```

The ageCategory column we created in Data Understanding might not align with the data corrections that we have done to the age column. So, we will drop it and then perform binning on age again because there can be several unique values for age. Therefore, a new column called ageCategory will be created again.

```
In [210]: bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
          labels = ['0 - 10 years old', '11 - 20 years old', '21 - 30 years old', '31 - 40 years old', '41 - 50 years old',
                    '51 - 60 years old', '61 - 70 years old', '71 - 80 years old', '81 - 90 years old', '91 - 100 years old']
          forDP = forDP.assign(ageCategory=forDP.age)
          forDP.ageCategory = pd.cut(x = forDP['age'], bins = bins, labels = labels, include_lowest = True)
          forDP.ageCategory.value_counts()

Out[210]: 21 - 30 years old     6730
          31 - 40 years old     6184
          41 - 50 years old     3801
          51 - 60 years old     2240
          11 - 20 years old     1525
          61 - 70 years old      975
          71 - 80 years old      326
          0 - 10 years old       150
          81 - 90 years old       63
          91 - 100 years old       1
          Name: ageCategory, dtype: int64
```

Since we have a new column that categorises age, it is no longer necessary to retain the age column so we will drop it.

```
In [211]: forDP = forDP.drop(['age'], axis=1)
```

```
In [212]: forDP.head(15)
```

rerUnderAlcohol	qualificationCat	vehicleType	involvedPedestrian	involvedPassenger	reportedHour	reportedDayOfMonth	reportedDayOfWeek	reportedMonth	ageCategory
0	1	4	0.0	0.0	2	20	4	3	31 - 40 years old
1	2	0	0.0	0.0	2	20	4	3	41 - 50 years old
0	3	0	0.0	1.0	2	20	4	3	31 - 40 years old
0	3	3	0.0	1.0	5	20	4	3	71 - 80 years old
0	2	3	0.0	0.0	5	20	4	3	51 - 60 years old
0	9	3	0.0	0.0	6	20	4	3	21 - 30 years old
0	0	0	0.0	0.0	6	20	4	3	31 - 40 years old

3.3.2 Sex

From the data understanding section, 1 represents 'M', 0 represents 'F' and 2 represents the literal value '0'.Since gender can be either male or female only, we will have to replace the sex values for those records, where the sex value represents the literal value '0'. So, we will replace the sex values with the value 2 with the mode, which is the most frequently occurring value. Below, we find the mode.

```
In [215]: x = forDP['sex'].mode()[0]
          print("Mode for Sex: ", x)

          Mode for Sex:  1
```

Male is the most frequently occurring sex category. So we will replace the records with the sex value 2 with the mode, which is 1.

```
In [216]: forDP['sex'] = forDP['sex'].replace([2], x)
```

3.3.3 Involved Pedestrian (involvedPedestrian)

From the data understanding section, 1 represents 'S', 0 represents 'N' and 2 represents a null value. Since the value of involvedPedestrian can either represent 'N' (No) or 'S' (Yes), we will have to replace the involvedPedestrian values for those records, where the involvedPedestrian value represents a null value, which is 2. So, we will replace the involvedPedestrian values with the value 2 with the mode, which is the most frequently occurring value. Below, we find the mode and replace those involvedPedestrian values with the value 2 with the mode.

```
In [218]: x = forDP['involvedPedestrian'].mode()[0]
          print("Mode for involvedPedestrian: ", x)

          Mode for involvedPedestrian:  0.0

In [219]: forDP['involvedPedestrian'] = forDP['involvedPedestrian'].replace([2], x)
```

3.3.4 Involved Passenger (involvedPassenger)

```
In [220]: print(forDP['involvedPassenger'].value_counts())

          0.0    18318
          1.0     2225
          2.0     1452
          Name: involvedPassenger, dtype: int64
```

From the data understanding section, 1 represents ‘S’ that means Yes, 0 represents ‘N’ meaning No and 2 represents a null value. Since involvedPassenger can only be either ‘S’ (Yes) or ‘N’ (No), we have to replace the value 2 with the mode, which indicates the most frequently occurring value.

```
x = forDP['involvedPassenger'].mode()[0]
print("Mode for involvedPassenger: ", x)

Mode for involvedPassenger:  0.0

forDP['involvedPassenger'] = forDP['involvedPassenger'].replace([2], x)
```

The figure above shows the mode and the code for replacing the value 2 with the mode.

3.3.5 Qualification Category (qualificationCat)

As mentioned in data understanding, 2 represents N, which means "No information", while 3 represents a blank value. These 2 values do not provide any information about the qualificationCat feature. Therefore, we decided to replace it with the mode. Below, we find the mode and replace qualificationCat values, which are either 2 or 3, with the mode..

```
x = forDP['qualificationCat'].mode()[0]
print("Mode for qualificationCat: ", x)
```

```
Mode for qualificationCat:  0
```

```
forDP['qualificationCat'] = forDP['qualificationCat'].replace([2], x)
forDP['qualificationCat'] = forDP['qualificationCat'].replace([3], x)
```

```
print(forDP['qualificationCat'].value_counts())
```

```
0     12345
1      4439
4      1900
5      1118
6       688
7       537
8       388
9       297
10      136
11       70
12       66
13       11
Name: qualificationCat, dtype: int64
```

3.3.6 Vehicle type

As mentioned before in the data understanding, we can observe that the value 2 represents blank values and 9 represents no information recorded in the column data. Therefore to fix this problem we have to do how we did for the qualification category. So, we find the mode and replace the 2s and 9s with the mode. .

```
x = forDP['vehicleType'].mode()[0]
print("Mode for vehicleType: ", x)
```

```
Mode for vehicleType:  0
```

```
forDP['vehicleType'] = forDP['vehicleType'].replace([2], x)
forDP['vehicleType'] = forDP['vehicleType'].replace([9], x)
```

```
print(forDP['vehicleType'].value_counts())
```

```
0     10619
1      8937
3       831
4       468
5       392
6       356
7       136
8        92
10       55
11       43
12       19
13       14
14        8
15        8
16        7
18        3
17        3
19        2
21        1
20        1
Name: vehicleType, dtype: int64
```

From the figure above, we can observe that the mode is zero.

3.4 Check for outliers

3.4.1 noPeopleInvolved

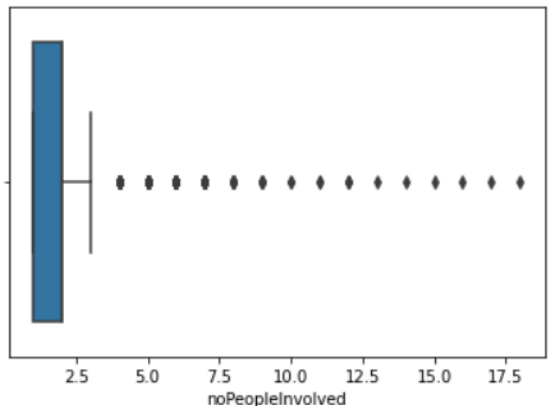
```
In [233]: forDP.describe()['noPeopleInvolved']

Out[233]: count    21995.000000
          mean         1.717208
          std          0.907393
          min          1.000000
          25%          1.000000
          50%          2.000000
          75%          2.000000
          max         18.000000
          Name: noPeopleInvolved, dtype: float64
```

From the result above, we can observe that the minimum value is 1 and mean is 1.717208. However the maximum value is 18, which is significantly larger than the minimum. Therefore we can say that this column contains outliers.

Therefore, we can plot a box plot to find the outliers for noPeopleInvolved.

```
: sns.boxplot(forDP['noPeopleInvolved'])
```

```
: <AxesSubplot:xlabel='noPeopleInvolved'>
```



We can see that there are several outliers, which are more than 3.

```
In [235]: def outliers_iqr(ys):
              quartile_1, median, quartile_3 = np.percentile(ys, [25, 50, 75])
              print("Quartile 1: ", quartile_1)
              print("Quartile 3: ", quartile_3)
              print("Median: ", median)
              iqr = quartile_3 - quartile_1
              lower_bound = quartile_1 - (iqr * 1.5)
              upper_bound = quartile_3 + (iqr * 1.5)
              print("Lower bound: ", lower_bound)
              print("Upper bound: ", upper_bound)
              return np.where((ys > upper_bound) | (ys < lower_bound))
```

The function above computes the 1st quartile, median and 3rd quartile and returns the indexes of the records that contain outlier values for noPeopleInvolved.

```
In [236]: pos = outliers_iqr(forDP['noPeopleInvolved'])
          pos
```

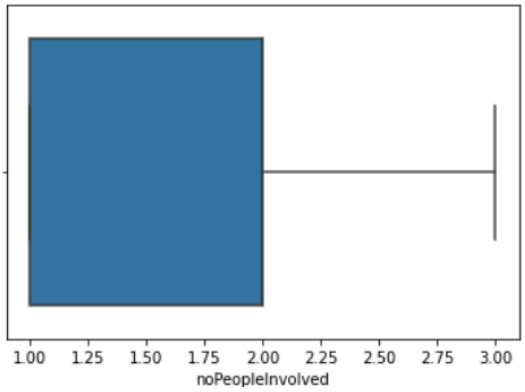
```
Quartile 1:  1.0
Quartile 3:  2.0
Median:  2.0
Lower bound:  -0.5
Upper bound:  3.5
```

The 1st quartile is 1. The 3rd quartile and median are 2. If the noPeopleInvolved value is less than the lower bound or greater than the upper bound, the value is considered as an outlier.

```
In [237]: forDP.loc[forDP["noPeopleInvolved"] > 3.5, "noPeopleInvolved"] = 2
          forDP.loc[forDP["noPeopleInvolved"] < -0.5, "noPeopleInvolved"] = 2
```

We will be replacing the outlier values with the median, which has the value of 2.

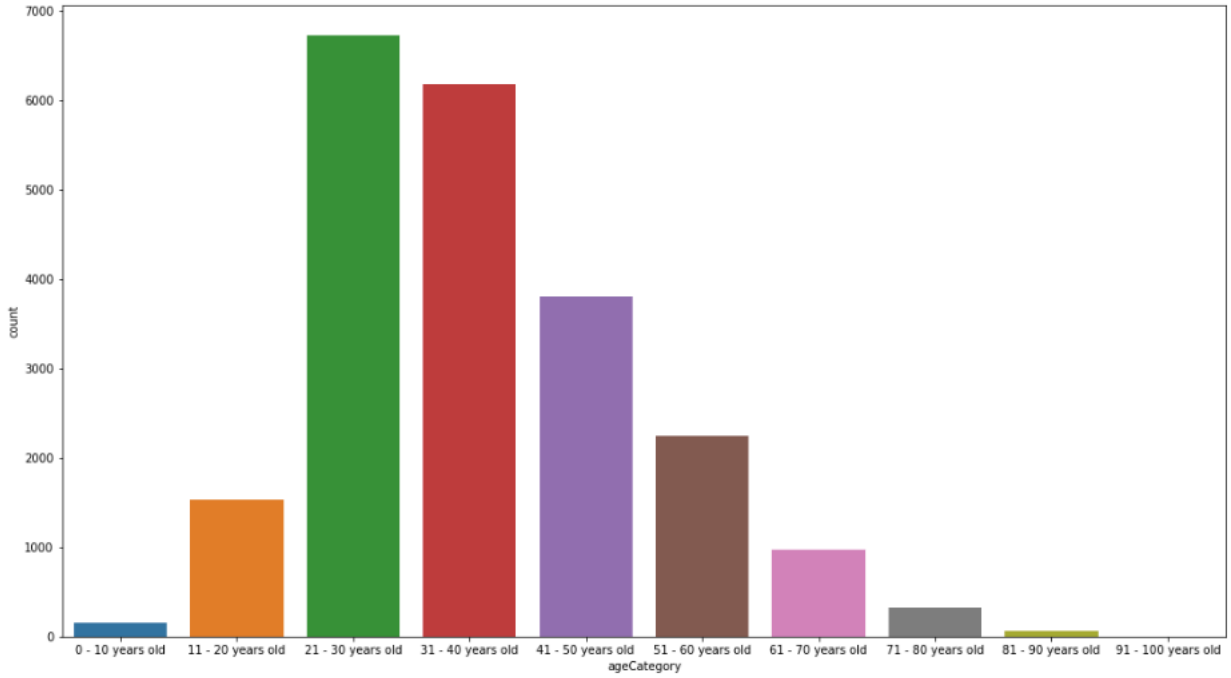
```
In [238]: sns.boxplot(forDP['noPeopleInvolved'])
Out[238]: <AxesSubplot:xlabel='noPeopleInvolved'>
```



This image shows now that noPeopleInvolved has no more outliers because we have replaced the outliers with the value of 2.

3.4.2 ageCategory

Here, we investigate for any outliers in the ageCategory column by using a countplot, which is similar to a histogram and bar plot.



The distribution for the ageCategory column almost has a normal distribution. This is because of the binning that we applied to the age column earlier. Binning is considered as transforming the data and data transformation can actually eliminate outliers. Therefore, we will not be making any changes to the ageCategory column.

As the data type of the values in the ageCategory column is currently categorical, we have to create a new column that is the numerical version of the ageCategory column. Below is our plan for assigning the number to each age category.

Here, we create a new column called ageCategoryNum and populate it.

```
forDP.loc[forDP['ageCategory'] == '0 - 10 years old', 'ageCategoryNum'] = 0
forDP.loc[forDP['ageCategory'] == '11 - 20 years old', 'ageCategoryNum'] = 1
forDP.loc[forDP['ageCategory'] == '21 - 30 years old', 'ageCategoryNum'] = 2
forDP.loc[forDP['ageCategory'] == '31 - 40 years old', 'ageCategoryNum'] = 3
forDP.loc[forDP['ageCategory'] == '41 - 50 years old', 'ageCategoryNum'] = 4
forDP.loc[forDP['ageCategory'] == '51 - 60 years old', 'ageCategoryNum'] = 5
forDP.loc[forDP['ageCategory'] == '61 - 70 years old', 'ageCategoryNum'] = 6
forDP.loc[forDP['ageCategory'] == '71 - 80 years old', 'ageCategoryNum'] = 7
forDP.loc[forDP['ageCategory'] == '81 - 90 years old', 'ageCategoryNum'] = 8
forDP.loc[forDP['ageCategory'] == '91 - 100 years old', 'ageCategoryNum'] = 9
```

Below, we ensure that the data type of the values in ageCategoryNum is numerical.

```
forDP = forDP.astype({'ageCategoryNum':'int'})
```

```
print(forDP['ageCategoryNum'].value_counts())
```

```
2    6730
3    6184
4    3801
5    2240
1    1525
6     975
7     326
0     150
8      63
9       1
Name: ageCategoryNum, dtype: int64
```

Since we already have ageCategoryNum column, it is no longer necessary to retain ageCategory column.

```
forDP = forDP.drop(['ageCategory'], axis=1)
```

Below, we convert the data types of the columns involvedPedestrian and involvedPassenger into integer because integers should be simpler to process.

```
forDP = forDP.astype({'involvedPedestrian':'int'})
forDP = forDP.astype({'involvedPassenger':'int'})
```

3.5 Feature Selection

	noPeopleInvolved	driverResponsibleForAccident	severityCode	sex	woreSeatBelt	driverUnderAlcohol	qualificationCat
noPeopleInvolved	1.000000	-0.336717	-0.022043	-0.141637	-0.128200	0.150285	-0.133045
driverResponsibleForAccident	-0.336717	1.000000	-0.307678	0.374263	0.373684	-0.023534	0.263815
severityCode	-0.022043	-0.307678	1.000000	-0.049808	-0.153507	-0.079491	-0.051898
sex	-0.141637	0.374263	-0.049808	1.000000	0.042692	0.044150	0.170246
woreSeatBelt	-0.128200	0.373684	-0.153507	0.042692	1.000000	-0.513505	0.028742
driverUnderAlcohol	0.150285	-0.023534	-0.079491	0.044150	-0.513505	1.000000	-0.086960
qualificationCat	-0.133045	0.263815	-0.051898	0.170246	0.028742	-0.086960	1.000000
vehicleType	-0.019466	0.065893	0.064712	0.102785	-0.036564	-0.023731	0.258490
involvedPedestrian	0.053078	-0.558775	0.213111	-0.112472	-0.664085	0.044685	-0.147413
involvedPassenger	0.365598	-0.780346	0.208668	-0.366113	0.053069	-0.005139	-0.205867
reportedHour	-0.014613	0.016746	-0.000505	0.025952	-0.056440	0.025560	-0.010602
reportedDayOfMonth	0.005824	0.015963	-0.000976	0.018858	-0.006020	0.017504	0.012374
reportedDayOfWeek	0.005907	-0.027506	0.011144	0.003302	-0.012377	0.022961	-0.015407
reportedMonth	-0.000543	0.023950	-0.005699	0.015127	0.012942	0.007065	-0.016611
ageCategoryNum	-0.009660	0.006576	-0.253421	-0.045450	-0.018628	-0.007062	0.034440

vehicleType	involvedPedestrian	involvedPassenger	reportedHour	reportedDayOfMonth	reportedDayOfWeek	reportedMonth	ageCategoryNum
-0.019466	0.053078	0.365598	-0.014613	0.005824	0.005907	-0.000543	-0.009660
0.065893	-0.558775	-0.780346	0.016746	0.015963	-0.027506	0.023950	0.006576
0.064712	0.213111	0.208668	-0.000505	-0.000976	0.011144	-0.005699	-0.253421
0.102785	-0.112472	-0.366113	0.025952	0.018858	0.003302	0.015127	-0.045450
-0.036564	-0.664085	0.053069	-0.056440	-0.006020	-0.012377	0.012942	-0.018628
-0.023731	0.044685	-0.005139	0.025560	0.017504	0.022961	0.007065	-0.007062
0.258490	-0.147413	-0.205867	-0.010602	0.012374	-0.015407	-0.016611	0.034440
1.000000	-0.132668	0.020877	-0.027718	0.005953	-0.029395	-0.007633	-0.012550
-0.132668	1.000000	-0.080588	0.040851	-0.014089	0.005040	-0.020442	0.086980
0.020877	-0.080588	1.000000	-0.050468	-0.008888	0.028814	-0.014157	-0.073566
-0.027718	0.040851	-0.050468	1.000000	-0.007330	0.018788	0.001806	-0.036820
0.005953	-0.014089	-0.008888	-0.007330	1.000000	-0.008663	0.002548	-0.017406
-0.029395	0.005040	0.028814	0.018788	-0.008663	1.000000	0.008425	-0.030147
-0.007633	-0.020442	-0.014157	0.001806	0.002548	0.008425	1.000000	-0.001095
-0.012550	0.086980	-0.073566	-0.036820	-0.017406	-0.030147	-0.001095	1.000000

Figure Showing Correlation Coefficients Between All Columns

There is a high negative correlation between involvedPassenger and driverResponsibleForAccident (-0.780346), which will increase the complexity of the algorithmns that we will be using later on. We decided to delete one of them, which is involvedPassenger. This is because we think that the driverResponsibleForAccident is a more important feature.

We also decided to drop the reportedDayOfMonth because it does not have much correlation with the severityCode. Although the reportedHour also has a very low negative correlation with the severityCode, we think that it has more impact than the reportedDayOfMonth.

```
forDP = forDP.drop(['involvedPassenger'], axis=1)
forDP = forDP.drop(['reportedDayOfMonth'], axis=1)
```

D. MODELLING

1.1 A Brief Introduction of what we have done

As there are 4 members in our assignment group, each person has written the codes for 1 machine learning algorithm/model. The 4 main models/algorithms we have developed are Random Forest, Decision Tree, Logistic Regression and K Nearest Neighbors. Earlier on, we mentioned that the dataset is very imbalanced since there are very few Fatal cases. Therefore, for each model, we will compare its performance metrics before and after upsampling to check for any improvements.

1.2 Performance Metrics (How we are going to compare each model)

Before going into the results produced by the various algorithms, we first define the performance metrics which we will use as the results and why we chose them.

Definitions:

TP = True Positive
TN = True Negative
FP = False Positive
FN = False Negative

Accuracy

This is the ratio of correct predictions and the total number of predictions. Its formula is:
(TP + TN) / (TP + TN + FP + FN)
We will not be using this metric because our dataset is extremely imbalanced and this metric is unsuitable for imbalanced datasets. We use a simple example to explain this. Let's say we have a dataset of 10 samples where only 1 sample belong to the "Fatal" class and the remaining 9 samples belong to either the "No Injuries" or "Not Fatal" classes. Let's say we have a faulty classifier which gives only "No Injuries" or "Not Fatal" predictions. Let's designate the "Fatal" class as the Positive class and the "No Injuries" and "Not Fatal" classes collectively as the Negative class. When we input our 10 samples into the faulty classifier, it will predict all 10 records as Negative, including the single Positive one. Therefore, we will have 9 TN and 1 FN. Therefore, Accuracy = 9/10 which interprets as 90% accurate. So, even though our classifier is defective, it still has an Accuracy score of 90%, which is highly inappropriate. In the dataset that we are using, the Fatal accident class represents only about 0.5% of the total number of samples/accidents in the dataset. Accuracy is a reliable metric only if the dataset is class-balanced, that is each class of the dataset has more or less the same number of records.

Precision

This is the ratio of True Positives and Total Positives (i.e. the cases which have been classified as Positive, whether or not they are actually so).
Formula: TP / (TP + FP)

Recall
This is the ratio of True Positives and all the really Positive cases (whether or not they are classified as so).
Formula: TP / (TP + FN)

F1-Score
For our original imbalanced dataset, we will be focussing on this metric to assess the performance of our models. Its formula is:
(2 x Precision x Recall) / (Precision + Recall)
As you can see, F1-Score gives equal weight to both Precision and Recall. Since it considers both the precision and recall ability of the model, it provides a good assessment of model performance and it also works when the dataset is imbalanced. When the above formula is substituted with the formulas for Precision and Recall, it becomes:
TP / (TP + 0.5 x (FP + FN))
From this version of the formula, F1-Score may not be so affected by a large number of negative samples. The F1-Score will decrease if there are many false predictions. For the 10-samples example, described in the Accuracy section above, the F1-Score will have a score of 0 (as well as the Precision and Recall).
For our dataset, we have three target classes: 1 (No Injuries), 2 (Non-Fatal) and 3 (Fatal). Each of them will have its own F1-Score (this is shown on the Classification Report). It is more convenient to have a single F1-Score which represents the overall performance which covers all 3 classes. On the Classification Report, there are 3 Average F1-Scores: Micro, Macro and Weighted. The Weighted Average is not suitable for our imbalanced dataset because when calculating this average, the majority class will have a larger impact than the minority class because of the much larger number of samples in the majority class. When the Micro Average is worked out, it actually ends up to be the Accuracy so we don't want this. On the other hand, the Macro Average treats all classes as equal as it is simply the arithmetic mean of the F1-Scores of all the classes. We will be using the Macro Average in our evaluation.

Multi-Class Confusion Matrix
A Confusion Matrix is a comparison between Actual and Predicted values. Our target variable, Severity Code, has 3 outcomes or classes: No Injuries (1), Not Fatal (2), Fatal (3). As such, the Confusion Matrix for our dataset is a 3x3 matrix. For a multi-class confusion matrix, since there are no positive and negative classes, in order to find TP, TN, FP and FN, we have to find TP, TN, FP and FN for each class. By converting the confusion matrix into a one-vs-all type matrix, the metrics Precision and Recall can be derived, and subsequently the F1-Score.

1.3 How we split the training, validation and testing datasets

Before we can perform any modelling, we must split the dataset into train, test and validation sets. We wrote the code below so that we can split matrices or arrays into test and train subsets. 60% of the records will be allocated to training, 20% will be allocated to validation and 20% will be allocated to testing.

1.3.1 Code for Dataset that has not been Upsampled:

```
inputVar = modelData.drop(["severityCode"], axis=1)
outputVar = modelData["severityCode"]

trainFeatures, testvalidationFeatures, trainLabels, testvalidationLabels =
train_test_split(inputVar, outputVar, test_size = 0.4, random_state=42)

validationFeatures, testFeatures, validationLabels, testLabels =
train_test_split(testvalidationFeatures, testvalidationLabels, test_size=0.5,
random_state=42)
```

1.3.2 Code for Dataset that has been Upsampled:

```
from sklearn.utils import resample
upDF = modelData.copy()
fatal = upDF[upDF["severityCode"] == 3]
nonFatal  = upDF[upDF["severityCode"] == 2]
noinjuries = upDF[upDF["severityCode"] == 1]
print("Shape of Fatal cases Before Upsampling:", fatal.shape)
print("Shape of Not Fatal cases Before Upsampling:", nonFatal.shape)
print("Shape of No Injuries cases Before Upsampling:", noinjuries.shape)
```

```
Shape of Fatal cases Before Upsampling: (113, 13)
Shape of Not Fatal cases Before Upsampling: (12544, 13)
Shape of No Injuries cases Before Upsampling: (9338, 13)
```

```
fatal_upsample = resample(fatal,
             replace=True,
             n_samples=len(nonFatal),
             random_state=42)

print(fatal_upsample.shape)
```

```
(12544, 13)
```

```
noinjuries_upsample = resample(noinjuries,
             replace=True,
             n_samples=len(nonFatal),
             random_state=42)

print(noinjuries_upsample.shape)
```

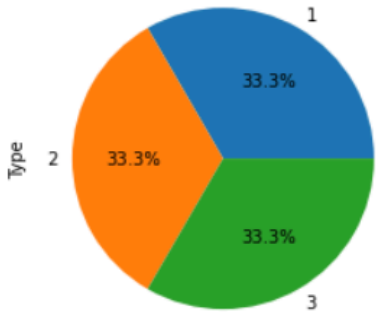
```
(12544, 13)
```

We first split up the dataset into separate dataframes based on their severity codes. After that, we have to upsample the fatal and noinjuries dataframes so that they have the same number of records as the notFatal dataframe. Next, the upsampled dataframes are concatenated with the notFatal frame.

```
data_upsampled = pd.concat([nonFatal, fatal_upsample, noinjuries_upsample])

print(data_upsampled["severityCode"].value_counts())

data_upsampled.groupby('severityCode').size().plot(kind='pie',
                                        y = "severityCode",
                                        label = "Type",
                                        autopct='%1.1f%%')
```

```
3    12544
2    12544
1    12544
Name: severityCode, dtype: int64
```

```
]: <AxesSubplot:ylabel='Type'>
```



The pie chart shown above proves that the new dataset has an even number of records for no injuries, not fatal and fatal cases.
Below, we start to split the dataset into training, validation and testing datasets.

```
modelDatav2 = data_upsampled.copy()
inputVar2 = modelDatav2.drop(["severityCode"], axis=1)
outputVar2 = modelDatav2["severityCode"]

trainFeaturesv2, testvalidationFeaturesv2, trainLabelsv2,
testvalidationLabelsv2 = train_test_split(inputVar2, outputVar2,
test_size = 0.4, random_state=42, shuffle=True)

validationFeaturesv2, testFeaturesv2, validationLabelsv2,
testLabelsv2 = train_test_split(testvalidationFeaturesv2,
testvalidationLabelsv2, test_size=0.5, random_state=42, shuffle=True)
```

1.4 Random Forest (Ryan Kho Yuen Thian)

1.4.1 What is Random Forest

Random Forest is a bagging ensemble algorithm which can be used for both classification and regression problems. It is an ensemble of decision trees which it builds from data samples. Each decision tree in the forest provides its own prediction. Voting is used to determine the best prediction for classification and for regression, the predictions are averaged. This tree bagging technique is called bootstrap aggregation. A Random Forest model offers a more superior solution than a single decision tree model because it reduces overfitting. A single decision tree by itself is more prone to overfitting (if it is allowed to grow without control) but several decision trees taken together will overfit less. The reasoning behind this is, if one decision tree makes an error, other decision trees in the random forest may be able to makeup for the mistake. On average, the ensemble solution is superior to that of a single model as it decreases variance and bias in prediction, and therefore increases accuracy.

This is how a Random Forest works. Given an existing dataset with X features, random samples known as bootstrap samples are selected from the dataset, with replacement. A decision tree is built from each sample. The number of trees to be built is determined by a hyper-parameter (n_estimators) of the random forest algorithm. At the start of the decision tree construction process, only a subset N features is randomly selected from the total X features. This N is determined by the max_features hyper-parameter. From these N features, the feature that results in the highest information gain will be chosen to split the node. The remaining construction of the decision tree will be performed just as it is done for a normal decision tree. By considering only a random subset of the features, bias is further reduced. The predictions from all the individual decision trees are then aggregated and voting is conducted on them (since our road accidents problem involves classification, we will only consider voting). The prediction that has the most votes will become the final prediction.

Hyperparameter Tuning

In order to improve the performance of this algorithm and to handle this imbalanced dataset, we have decided tune some of the hyperparameters of this algorithm which we think will have the biggest impact. Note that later when we perform upsampling to make our dataset balanced, we will no longer be using the hyperparameter values below.

a) n_estimators:

This determines the number of decision trees to be used to build the random forest. The more the number of trees, the algorithm's performance will increase until it reach a point where adding more trees will not impact the algorithm's performance. The default is 100. Since more trees imply more computational costs, we cannot choose a very large number. We decided to go with 150 which is higher than the default value but yet not so large.

b) criterion:
This is the function which is used to measure the quality of splits in a decision tree. For classification, there are only 2 choices: “gini" (default) or “entropy”. We decided to go with entropy because we are more familiar with it.

c) max_features:
This is the maximum number of features that can be used in splitting a node. This is a subset of the complete set of features. For classification, the recommendation is to use “sqrt” which is the square root of the total number of features.

d) max_depth
This is the maximum depth of the decision tree. The default value is None which means the tree will keep splitting until purity is achieved. A lower value will prevent overfitting but if the value is too low, this will lead to underfitting. A larger value will result in longer computational time. We decided to go with 10 which is neither too high nor low.

e) min_samples_split
This is the minimum number of samples required to further split a node. The default value is 2, which we feel is too low. We think a higher value like 6 is better because splitting will stop earlier and therefore avoid overfitting.

f) class_weight
This parameter is particularly important for our imbalanced dataset because algorithms tend to be bias towards the majority class. The default value for this parameter is None. This parameter allows weights to be assigned to each target variable class when calculating the impurity score during the splitting of nodes. Alternatively, this parameter can be assigned “balanced” which will adjust the weights inversely proportional to the frequencies of the target variable classes, so as to bolster the minority classes. Another alternative parameter value is “balanced_subsample” which does the same thing as “balanced” except that the weights are calculated for the bootstrap sample corresponding to each decision tree. We decided to go with “balanced_subsample” since we are using bootstrap samples and each decision tree is actually built from the bootstrap sample, not from the entire dataset. Each bootstrap sample will have its own class distribution.

1.4.2 Result of Random Forest Before Upsampling

To determine which values of the hyperparameters would produce the Random Forest model with the best performance, I created 3 Random Forest models with different values for their hyperparameters.

a) Modelling with Default Hyperparameters

```
[[1595  220    3]
 [ 326 2231    1]
 [   0   23    0]]
              precision    recall  f1-score   support

           1       0.83      0.88      0.85      1818
           2       0.90      0.87      0.89      2558
           3       0.00      0.00      0.00        23

    accuracy                           0.87      4399
   macro avg       0.58      0.58      0.58      4399
weighted avg       0.87      0.87      0.87      4399
```

Although the precision, recall and f1-score values for labels 1 and 2 are acceptable, the values of those performance metrics for label 3 are 0. This could be because when TP = 0, both the recall and precision become 0. Because of that , the f1-score cannot be calculated since division by 0 is impossible. These will actually categorise the classifier as useless. Therefore, we try to find the best hyperparameters for the Random Forest Model using GridSearchCV below.

b) Modelling with Tuned Hyperparameters

```
from sklearn.model_selection import GridSearchCV
paramGrid = {
    'n_estimators': [200, 250, 300, 350, 400],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [2, 4, 6, 8, 10],
    'class_weight': ['balanced','balanced_subsample'],
    'min_samples_split': [4,6,8],
}
gridSearch = GridSearchCV(RandomForestClassifier(),
                          param_grid=paramGrid)
gridSearch.fit(trainFeatures, trainLabels)
print(gridSearch.best_estimator_)
```

```
RandomForestClassifier(class_weight='balanced_subsample', max_depth=10,
                       max_features='log2', min_samples_split=6,
                       n_estimators=200)
```

From the above analysis, the best values for class_weight, max_depth, max_features, min_samples_split and n_estimators are 'balanced_subsample', 10, 'log2', 6 and 200.

```
[[1648  166    4]
 [ 310 2223   25]
 [   0   20    3]]
              precision    recall  f1-score   support

           1       0.84      0.91      0.87      1818
           2       0.92      0.87      0.90      2558
           3       0.09      0.13      0.11        23

    accuracy                           0.88      4399
   macro avg       0.62      0.64      0.63      4399
weighted avg       0.88      0.88      0.88      4399
```

Even though we used the best parameter values suggested by GridSearchCV for the RandomForest model, the values of the precision, recall and f1-score are still low. Therefore, we will enter our own values for the model in order to improve the results of precision, recall and f1-score for label 3.

c) Modelling with Customised Values for Hyperparameters

```
randomF_model = RandomForestClassifier(random_state=42,
n_estimators=300,criterion="entropy",max_features="sqrt",max_dep
th=8,min_samples_split=4,class_weight="balanced_subsample")
```

```
[[1650  166    2]
 [ 309 2196   53]
 [   0   17    6]]
              precision    recall  f1-score   support

           1       0.84      0.91      0.87      1818
           2       0.92      0.86      0.89      2558
           3       0.10      0.26      0.14        23

    accuracy                           0.88      4399
   macro avg       0.62      0.68      0.64      4399
weighted avg       0.89      0.88      0.88      4399
```

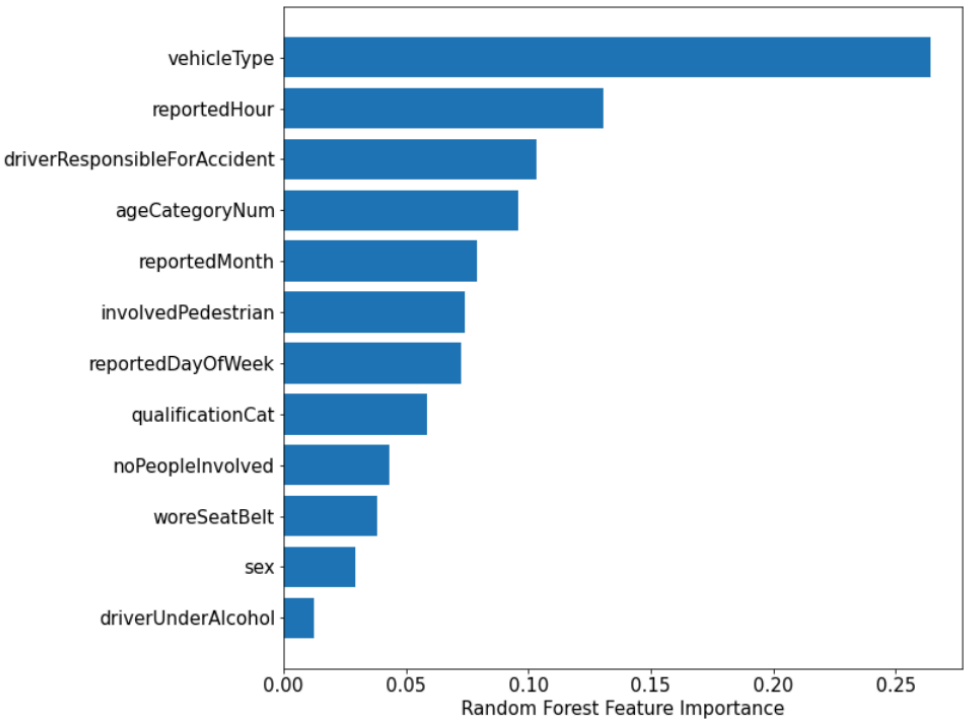
The values of the precision, recall and f1-score have improved for label 3. Therefore, we will be using the values 300, "sqrt", 8, 4 and "balanced_subsample" for the n_estimators, max_features, max_depth, min_samples_split and class_weight parameters respectively.

We can see that the macro average for precision improved from 0.58 to 0.62, the macro average for recall improved from 0.58 to 0.68 and the macro average for f1-score improved from 0.58 to 0.64. As for the accuracy, it improved from 0.87 to 0.88.

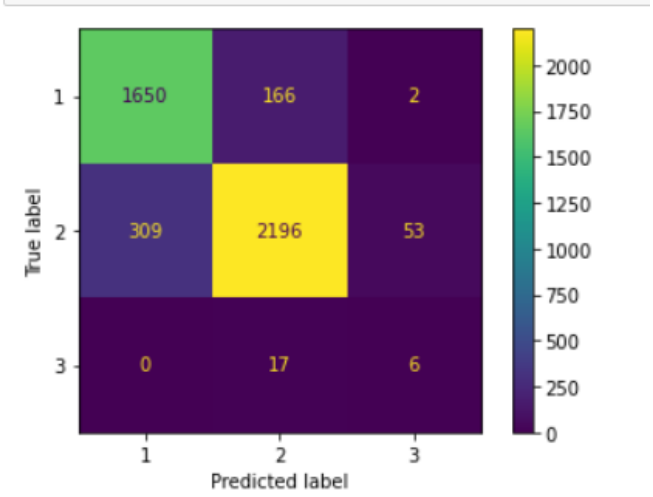
Feature Importance from Random Forest Model

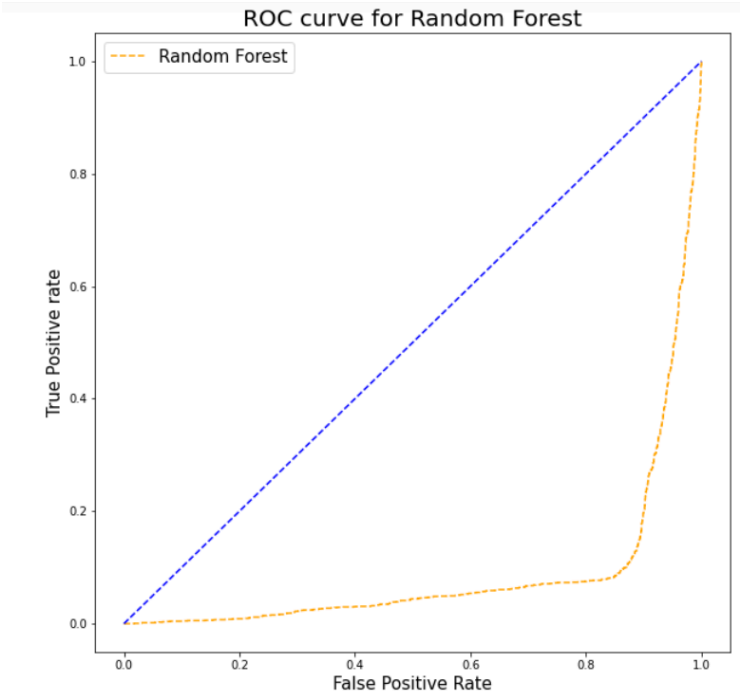
Feature importance indicates the features that are the most predictive of the target variable. It also aids us in comprehending which features are the most essential to our models and which features we can ignore safely.

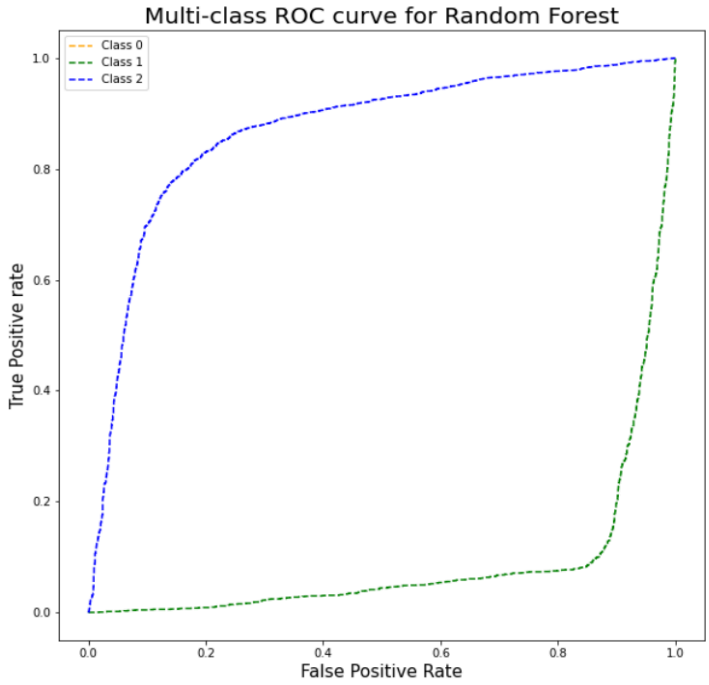
```
 1) vehicleType                  0.263912
 2) reportedHour                 0.130456
 3) driverResponsibleForAccident 0.103364
 4) ageCategoryNum               0.095641
 5) reportedMonth                0.078944
 6) involvedPedestrian           0.073737
 7) reportedDayOfWeek            0.072486
 8) qualificationCat             0.058380
 9) noPeopleInvolved             0.042964
10) woreSeatBelt                 0.038384
11) sex                          0.029400
12) driverUnderAlcohol           0.012332
```



We can see that the most important feature is vehicleType while driverUnderAlcohol is the least important feature. On the next page, we show the confusion matrix as well as 2 ROC graphs for the model.


1650
166
2
309
2196
53
0
17
6
True label
Predicted label
1
2
3
2000
1750
1500
1250
1000
750
500
250
0


ROC curve for Random Forest
Random Forest
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Multi-class ROC curve for Random Forest
Class 0
Class 1
Class 2
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

1.4.3 Result of Random Forest After Upsampling

a) Modelling with Default Hyperparameters

```
[[2378  129    2]
 [ 337 2200   11]
 [   0    0 2469]]
              precision    recall  f1-score   support

           1       0.88      0.95      0.91      2509
           2       0.94      0.86      0.90      2548
           3       0.99      1.00      1.00      2469

    accuracy                           0.94      7526
   macro avg       0.94      0.94      0.94      7526
weighted avg       0.94      0.94      0.94      7526
```

After upsampling, the results for the macro average of precision, recall and f1-score improved a lot, obtaining 0.94, 0.94 and 0.94 respectively. The accuracy is 0.94 as well. Below, we use GridSearchCV to try to find the best values for the hyperparameters.

b) Modelling with Tuned Hyperparameters

```
from sklearn.model_selection import GridSearchCV
paramGrid = {
    'n_estimators': [200, 250, 300, 350, 400],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [2, 4, 6, 8, 10],
    'class_weight': ['balanced','balanced_subsample'],
    'min_samples_split': [4,6,8],
}
gridSearch = GridSearchCV(RandomForestClassifier(),
                          param_grid=paramGrid)
gridSearch.fit(trainFeaturesv2, trainLabelsv2)
print(gridSearch.best_estimator_)
```

```
RandomForestClassifier(class_weight='balanced_subsample', max_depth=10,
                       max_features='sqrt', min_samples_split=8,
                       n_estimators=300)
```

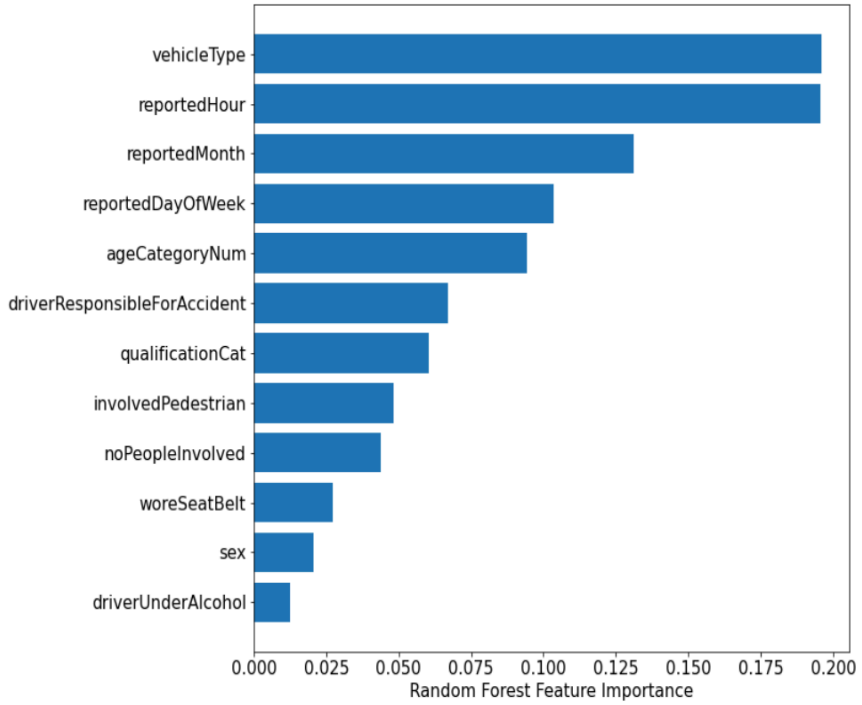
From the above analysis, the best values for class_weight, max_depth, max_features, min_samples_split and n_estimators are 'balanced_subsample', 10, 'sqrt', 8 and 300.

```
[[2270  230    9]
 [ 328 2112  108]
 [  28  275 2166]]
              precision    recall  f1-score   support

           1       0.86      0.90      0.88      2509
           2       0.81      0.83      0.82      2548
           3       0.95      0.88      0.91      2469

    accuracy                           0.87      7526
   macro avg       0.87      0.87      0.87      7526
weighted avg       0.87      0.87      0.87      7526
```

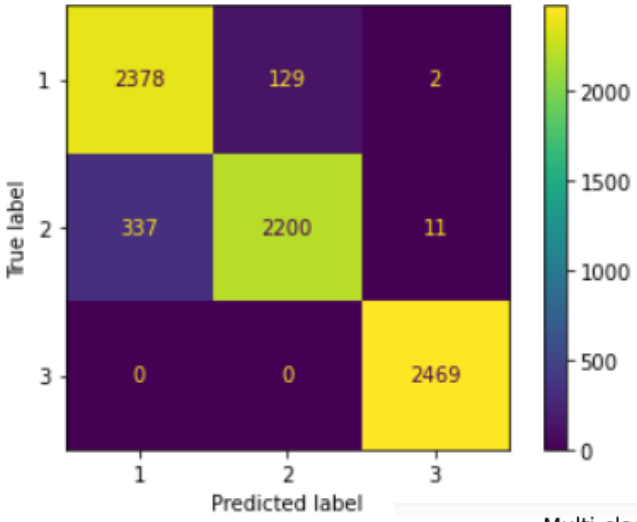
For the model with tuned hyperparameters, its performance is good but worse than the model with default hyperparameters. It obtained 0.87 for macro average precision, macro average recall, macro average f1-score and accuracy. As the results for the Random Forest model with default parameters is performing reasonably well, we will not be creating another Random Forest model with customised values for the hyperparameters.

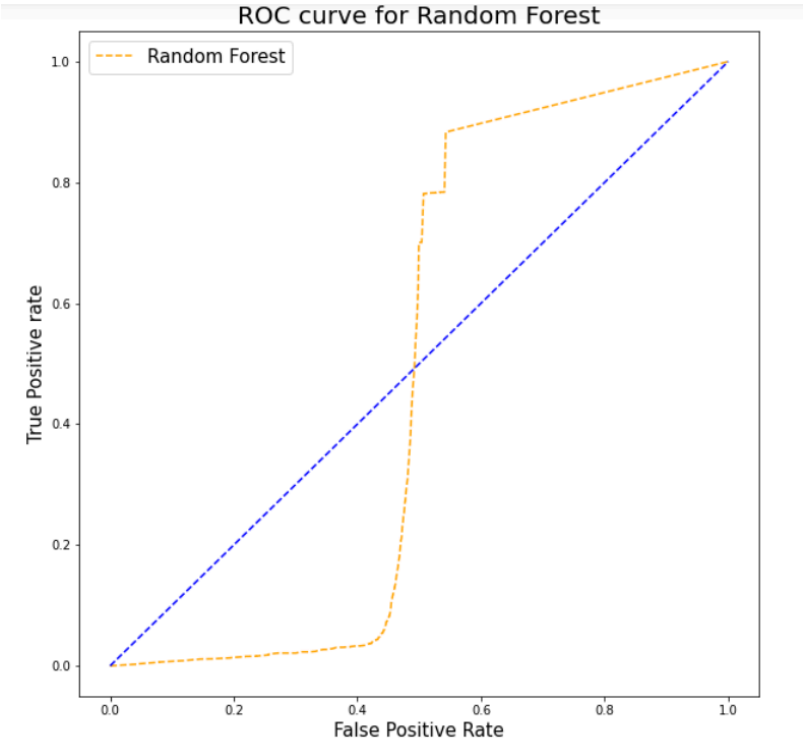
Feature Importance from Random Forest Model

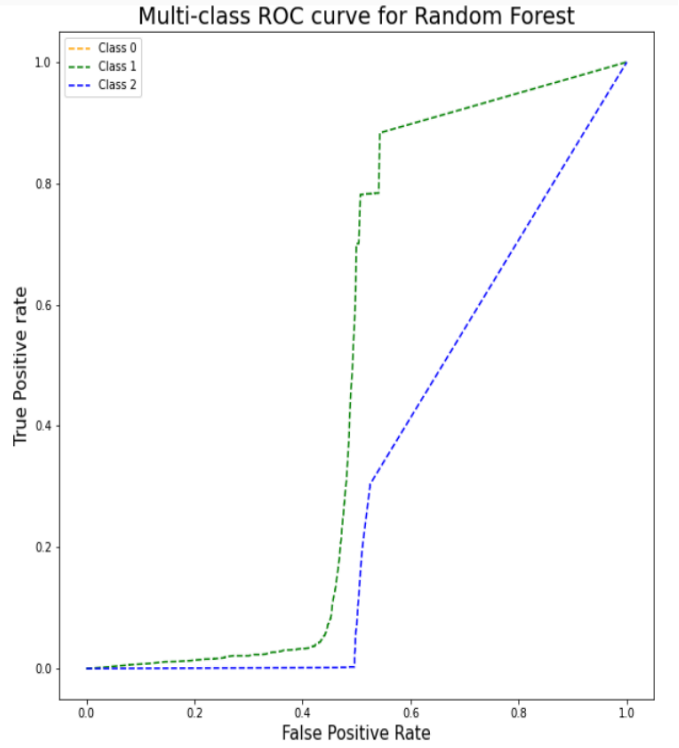
```
 1) vehicleType                 0.195799
 2) reportedHour                0.195491
 3) reportedMonth               0.131172
 4) reportedDayOfWeek           0.103686
 5) ageCategoryNum              0.094471
 6) driverResponsibleForAccident 0.066896
 7) qualificationCat            0.060303
 8) involvedPedestrian          0.048171
 9) noPeopleInvolved            0.043725
10) woreSeatBelt                0.027120
11) sex                         0.020755
12) driverUnderAlcohol          0.012413
```



vehicleType is the most important feature while driverUnderAlcohol is the least important feature. Take note that reportedHour almost has the same importance as vehicleType. On the next page, we show the confusion matrix and the 2 ROC graphs.


2378
129
2
337
2200
11
0
0
2469
True label
Predicted label
1
2
3
2000
1500
1000
500
0


ROC curve for Random Forest
Random Forest
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Multi-class ROC curve for Random Forest
Class 0
Class 1
Class 2
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

1.5 Decision Tree (Thong Cheng How)

1.5.1 What is Decision Tree

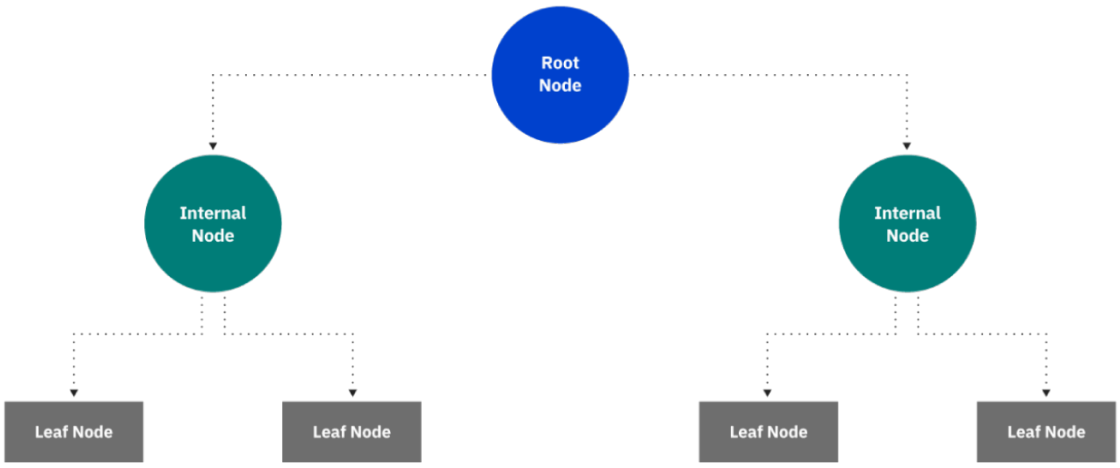


Figure 1

Decision tree is one of the types of supervised learning algorithm that is a non-parameterized model. The meaning of non-parameterized means that there is no parameter needed to be entered to run the model successfully. It only needs the training and testing set to run the model. Therefore the training the testing data set must be in numerical form for it to run successfully. The decision tree is utilized for both classification and regression tasks. Decision tree learning contains different methods of acquiring data and using the data to conduct a wide variety of searches to identify the most optimal split of points within a tree. The process of splitting is repeated multiple times from top to bottom until all nodes inside the tree are classified under its own specific class labels. It also depends on the dataset whether all data are classified homogeneously. It depends on the complexity of the dataset and the complexity of the decision tree. The simpler the model of the dataset it will be easier for the decision tree model to identify the leaf nodes as it contains lesser attributes and data. A decision tree can also be added and can be grown in size, there if finding the output for the leaf nodes, it may take longer time because the data will usually result in too little or too little data falling into the data labels. This can also result in overfitting.

As shown above for figure 1, this image is taken from the IBM website. In order to explain a decision tree clearly, the one in the blue circle until the four dark grey square boxes are the process for the decision tree model to work. The decision tree first starts from a root node. A root node is the beginning of the decision tree which will not consist of any incoming roots, the outgoing roots from the root nodes will be then split into different roots that lead to the next node. This node will then be called the internal node / decision node. Through this node, it will conduct a type of homogenous subnet, which will be determined by the internal node and the leaf node to have different possible outcomes for the model. After the internal node, it will then send outgoing roots to the next node. This node is considered the final stage for the model which is

called the leaf nodes / terminal nodes. The leaf nodes represent all possible outcomes for the specific dataset. However please note that different dataset can lead to different outcomes and results for the result to be outputted and different machines will have different performances so the result would vary between different machines. In decision tree there is also many different types for example Hunt's algorithm which was developed in 1960 which is to help human learning in psychology, ID3 stands for Iterative Dichotomiser 3, C4.5 is the later iteration of ID3 stands for Iterative Dichotomiser 3 and CART is the abbreviation of classification and regression trees. (IBM 2022)

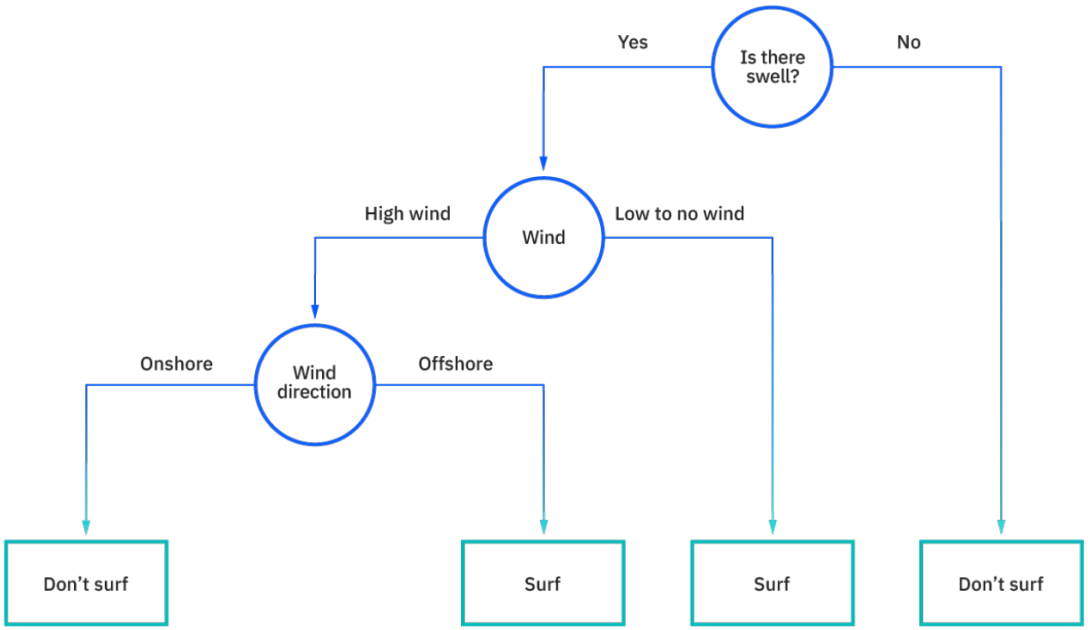


Figure 2

As shown in figure 2, this diagram is to demonstrate how a decision making structure is made. This diagram is a simpler representation of how a flowchart structure is created to help different individuals or organizations to understand the process of decision making diagrams. To make it understandable this figure is a similar kind of decision tree function but in a flowchart decision graph

There are multiple ways to select the best nodes. But commonly there are two commonly used methods which are the information gain for gini impurity and entropy. These two methods act as a popular splitting technique for decision tree models. These methods will help to evaluate the quality of each test condition and how effective it will be classified into a class for that specific model. Another part is that, we need to consider the max_depth for the dataset too, because this will determine whether the dataset will be overfitted or under fitted, this can be done through calculating the columns that have been prepared in the data preparation. To determine whether it

will be underfit or overfit can be determined by the result where the result will be slightly weird and uncommon.

1.5.2 Result of Decision Tree Before Upsampling

In [265]:
```
#Runnning the model using validation first to test the model accuracy

# 1. choose model class
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import decomposition
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

scl = StandardScaler()
dcm = decomposition.PCA()
dtc= DecisionTreeClassifier()
pipe = Pipeline(steps=[('scl',scl),('dcm', dcm), ('dtc',dtc)])
#pipe = Pipeline(steps=[('dcm', dcm), ('dtc',dtc)])

n_components = list(range(1,trainFeatures.shape[1]+1,1))

criterion = ['gini', 'entropy']
max_depth = [2,4,6,8,10,12,14]

parameters = dict(dcm__n_components=n_components, dtc__criterion= criterion, dtc__max_depth=max_depth)

clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(trainFeatures, trainLabels)
```

```
print('In order to find the hyperparameters for decision tree, we first have to indentify the criterioin and the max_depth to pr
print();
print('Best criterion to use: ', clf_GS.best_estimator_.get_params()['dtc__criterion'])
print('Best max_depth suitable for the dataset: ', clf_GS.best_estimator_.get_params()['dtc__max_depth'])
print('Best number of components: ', clf_GS.best_estimator_.get_params()['dcm__n_components'])
print();
print();
print(clf_GS.best_estimator_.get_params()['dtc'])
```

```
In order to find the hyperparameters for decision tree, we first have to indentify the criterioin and the max_depth to proceed

Best criterion to use:  entropy
Best max_depth suitable for the dataset:  12
Best number of components:  12
```

In this code, it shows that the best criterion and max_depth for the model are entropy and 12 respectively. These are obtained via GridSearchCV. If the value for max_depth is too high, it will lead to overfitting, which will greatly affect the results. The number of components shows the reduction of the dimension done by the leaf nodes to find the precise information for the output. Since we have to use the training, testing and validation data set for our dataset, we have to use standard scalar, which is used to equally split the data for the leaf nodes to be split together. This is because without the standard scalar, the data will be hard for the leaf nodes to execute and manage to find the best nodes for the model. Another reason I import the decomposition function is to lower down the dimensional space of the dataset because if we do not lower it down the decision tree model will have a harder time to find and will take time to produce meaningful outcomes.. The reason we use decomposition is also because the input data is not centered proportionally, so including this function it will help the nodes to find the result

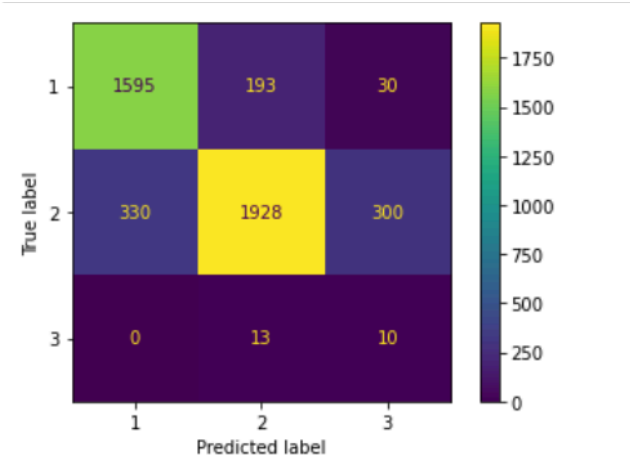
faster than traditionally waiting for the nodes to find the outcome. Therefore can cause the dataset to produce not so good outcomes because the data is not sampled properly but after that we will show another result which the data is balanced and will get even more precise results The decision tree needs criterion and max_depth to accurately analyse the dataset to get optimal results.

After finding the criterion, max_depth and component, we calculate the precision, recall, f1-score and accuracy to evaluate this model's performance.

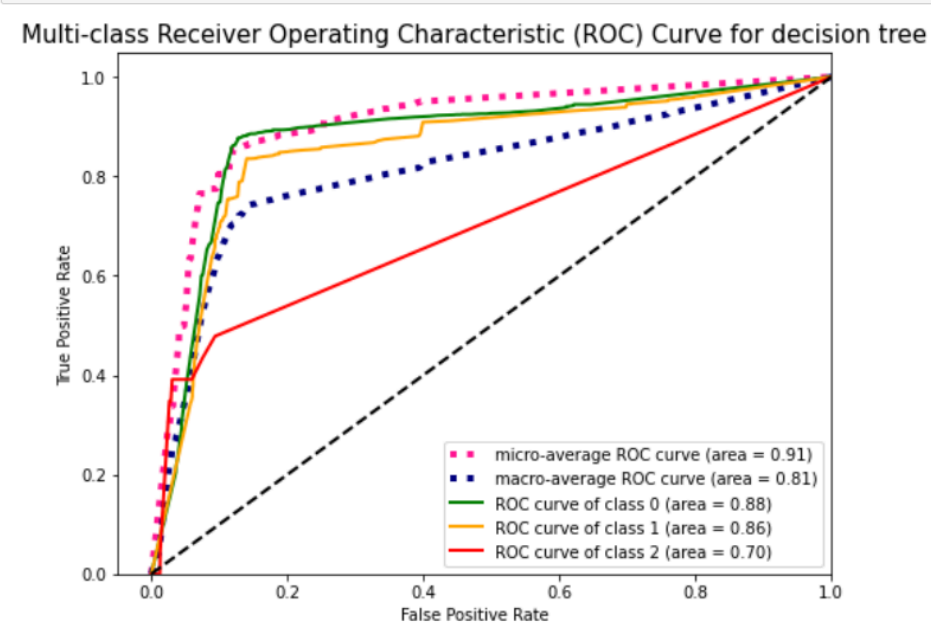
In [267]:
```
# 2. instantiate model
dtc = DecisionTreeClassifier(criterion='entropy', max_depth=12, class_weight="balanced")

# 3. fit model to data, for training purpose, training dataset
dtc.fit(trainFeatures, trainLabels)

# 4. predict on new data, testing data
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.metrics import confusion_matrix
val_dtc, predval_dtc = validationLabels, dtc.predict(validationFeatures)
dtc_cfm=confusion_matrix(validationLabels, dtc.predict(validationFeatures))
print(dtc_cfm)
print(classification_report(val_dtc, predval_dtc))
```

```
[[1595  193   30]
 [ 330 1928  300]
 [   0   13   10]]
              precision    recall  f1-score   support

           1       0.83      0.88      0.85      1818
           2       0.90      0.75      0.82      2558
           3       0.03      0.43      0.06        23

    accuracy                           0.80      4399
   macro avg       0.59      0.69      0.58      4399
weighted avg       0.87      0.80      0.83      4399
```

Although the results for weighted averages are good (above 80%), we will be focusing on macro averages because macro average treats all target classes equally. The result shows that the macro average for precision, recall and f1-score are 0.59, 0.69 and 0.58 respectively. However, the accuracy is good (0.8). Below is the confusion matrix for this model.



The confusion matrix summarises the performance of the Decision Tree model. Below is the ROC curve for this model.



This is the image for the roc curve generated using the model (decision tree) for before unsample dataset

1.5.3 Result of Decision Tree After Upsampling

In [290]:
```
#Runnning the model using validation first to test the model accuracy

# 1. choose model class
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import decomposition
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

scl = StandardScaler()
dcm = decomposition.PCA()
dtc= DecisionTreeClassifier()
pipe = Pipeline(steps=[('scl',scl),('dcm', dcm), ('dtc',dtc)])
#pipe = Pipeline(steps=[('dcm', dcm), ('dtc',dtc)])

n_components = list(range(1,trainFeaturesv2.shape[1]+1,1))

criterion = ['gini', 'entropy']
max_depth = [2,4,6,8,10,12,14]

parameters = dict(dcm__n_components=n_components, dtc__criterion= criterion, dtc__max_depth=max_depth)

clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(trainFeaturesv2, trainLabelsv2)

print('In order to find the hyperparameters for decision tree, we first have to indentify the criterioin and the max_depth to prc
print();
print('Best criterion to use: ', clf_GS.best_estimator_.get_params()['dtc__criterion'])
print('Best max_depth suitable for the dataset: ', clf_GS.best_estimator_.get_params()['dtc__max_depth'])
print('Best number of components: ', clf_GS.best_estimator_.get_params()['dcm__n_components'])
print();
```

```
print();
print(clf_GS.best_estimator_.get_params()['dtc'])
```

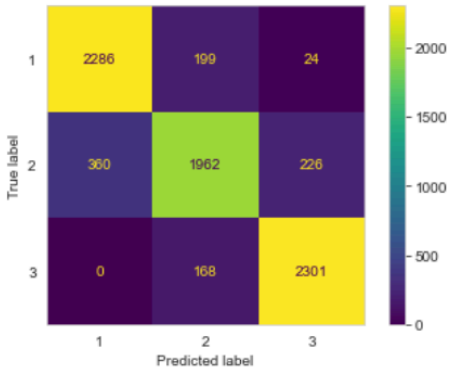
```
In order to find the hyperparameters for decision tree, we first have to indentify the criterioin and the max_depth to proceed

Best criterion to use:  gini
Best max_depth suitable for the dataset:  14
Best number of components:  12
```

In this code, it shows that the best criterion and max_depth for the model are gini and 14 respectively. These are obtained via GridSearchCV. This is data is after unsampling and the result are above.

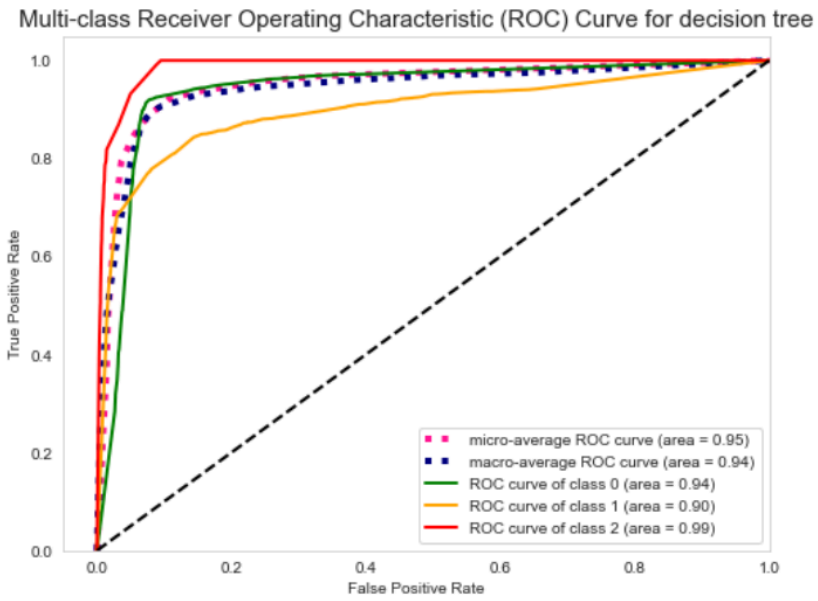
```
DecisionTreeClassifier(max_depth=14)
```

```
# 2. instantiate model
dtcv2 = DecisionTreeClassifier(criterion='entropy', max_depth=14)

# 3. fit model to data, for training purpose, training dataset
dtcv2.fit(trainFeaturesv2, trainLabelsv2)

# 4. predict on new data, testing data
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.metrics import confusion_matrix
val_dtcv2, predval_dtcv2 = validationLabelsv2, dtcv2.predict(validationFeaturesv2)
dtc_cfm=confusion_matrix(validationLabelsv2, dtcv2.predict(validationFeaturesv2))
print(dtc_cfm)
print(classification_report(val_dtcv2, predval_dtcv2))
conf = confusion_matrix(validationLabelsv2, predval_dtcv2, labels=dtcv2.classes_)

color = 'white'

disp = ConfusionMatrixDisplay(confusion_matrix=conf, display_labels=dtcv2.classes_)

disp.plot()

plt.show()
```

```
[[2286  199   24]
 [ 360 1962  226]
 [   0  168 2301]]
              precision    recall  f1-score   support

           1       0.86      0.91      0.89      2509
           2       0.84      0.77      0.80      2548
           3       0.90      0.93      0.92      2469

    accuracy                           0.87      7526
   macro avg       0.87      0.87      0.87      7526
weighted avg       0.87      0.87      0.87      7526
```



The model definitely performed better after upsampling based on its new accuracy and macro average for precision, recall, f1-score. The result shows that the macro average for precision, recall and f1-score are 0.87, 0.87 and 0.87 respectively, which is good. The accuracy is also good (0.87). The confusion matrix for this model is shown above while the ROC curve is shown below.



This is the ROC curve for the model (decision tree) after unsampling the data.

1.6 Logistic Regression (Murdi Starla Nathasa)

1.6.1 What is Logistic Regression

Logistic Regression is a supervised learning algorithm that analyses and describes patterns and relationships between data from a labelled dataset in both classification, where logistic regression is used for predicting the possible outcomes, and regression, where logistic regression is used to predict numerical values. Not to be mistaken with Linear Regression which predicts continuous outcomes, Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, True or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. This is a mathematical logistic function known as the sigmoid function . The main usage of Logistic regression in this situation is to predict the probability of something specific happening. In this case, our dependent variable is severityCode, in which we use logistic regression in order to find its connection to other variables such as age, sex, seat belt worn, and so on.

There are three types of Logistic Regression:

- Binary logistic regression is a logistic regression with two possible outcomes, for example a yes or no or true and false of whether or not a car accident includes death.
- Multinomial logistic regression is a logistic regression with multiple outcomes, example is to predict whether a car accident is caused by one specific reason or multiple such as lack of sleep, intoxicated driver or machine malfunction.
- Ordinal logistic regression is a logistic regression where outcome is ordered, example: the different severity of a car accident, is there injuries, deaths, or no one is harmed.

Since this is a multi-classification problem, Ordinal logistic regression is used to describe data and the relationship between one dependent variable and one or more independent variables as the main goal is to compare the relations between the severity of the car accident compared to other independent variables. The independent variables in this case can be nominal, ordinal, or of interval type.

1.6.2 Result of Logistic Regression Before Upsampling

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn import svm
hyperPara = {
    'penalty': ['l1', 'l2', 'elasticnet', None],
    'C': [100, 10, 1.0, 0.1, 0.01],
    'solver' : ['newton-cg', 'lbfgs', 'liblinear'],
}
# 1. choose model class
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.metrics import confusion_matrix, classification_report

gridSearch = GridSearchCV(LogisticRegression(), param_grid=hyperPara)
gridSearch.fit(trainFeatures, trainLabels)
print(gridSearch.best_params_)
```

```
{'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
```

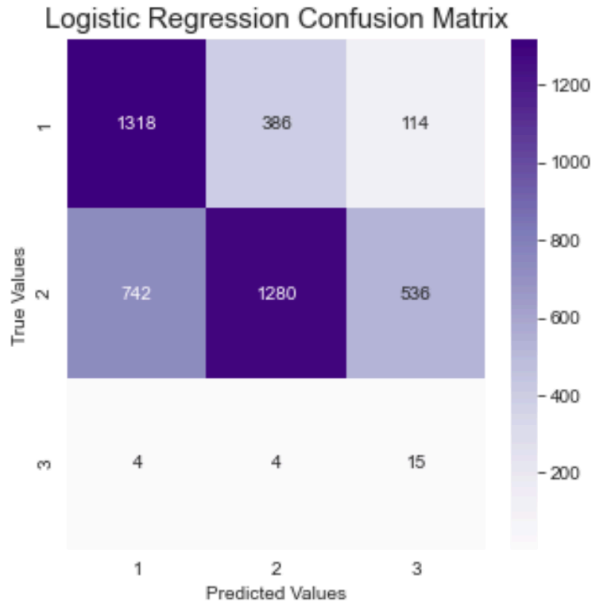
Shown in the figure above, hyperparameter tuning is used to find the best values for its hyperparameters. We executed GridSearchCV from Sklearn in order to find the most optimal values. The best values for the hyperparameters are {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}.

Explanation on parameters:

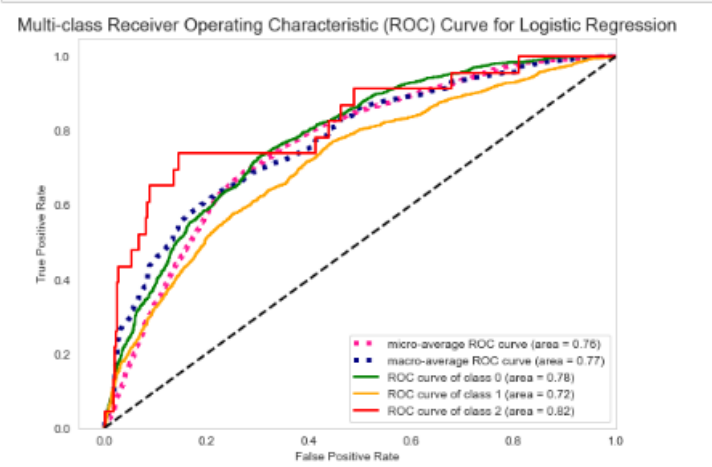
1. C = 0.1
   This parameter is the inverse of regularisation strength that has to be positive float numbers. The smaller the number, the stronger regularisation is, so in this case it is very strong since the value is only 0.1.
2. Penalty = 12
   Stand for L2 or Ridge regulation which adds a penalty term to the loss function that is proportional to the square of the magnitude of the coefficients. It will reduce the coefficients towards zero, but never set them exactly as zero.
3. Solver = liblinear
   A small and fast algorithm for logistic regression that optimises and utilises well for small datasets.

```
Confusion matrix :
 [[1318  386  114]
 [ 742 1280  536]
 [   4    4   15]]

Classification report :
```

	precision	recall	f1-score	support
1	0.64	0.72	0.68	1818
2	0.77	0.50	0.61	2558
3	0.02	0.65	0.04	23
accuracy			0.59	4399
macro avg	0.48	0.63	0.44	4399
weighted avg	0.71	0.59	0.63	4399



The result shows that the macro average for precision, recall and f1-score are 0.48, 0.63 and 0.44 respectively. This shows that the logistic regression model did not perform well. The accuracy for this model is not good (0.59). The confusion matrix for this model is shown above. The ROC curve is shown below.



1.6.3 Result of Logistic Regression After Upsampling

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn import svm
hyperPara = {
    'penalty': ['l1', 'l2', 'elasticnet', None],
    'C': [100, 10, 1.0, 0.1, 0.01],
    'solver' : ['newton-cg', 'lbfgs', 'liblinear'],
}
# 1. choose model class
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.metrics import confusion_matrix, classification_report

gridSearch = GridSearchCV(LogisticRegression(), param_grid=hyperPara)
gridSearch.fit(trainFeaturesv2, trainLabelsv2)
print(gridSearch.best_params_)
```

```
{'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
```

Shown in the figure above, hyperparameter tuning is used to find the best values for its hyperparameters. We executed GridSearchCV from Sklearn in order to find the most optimal values. The best values for the hyperparameters: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}

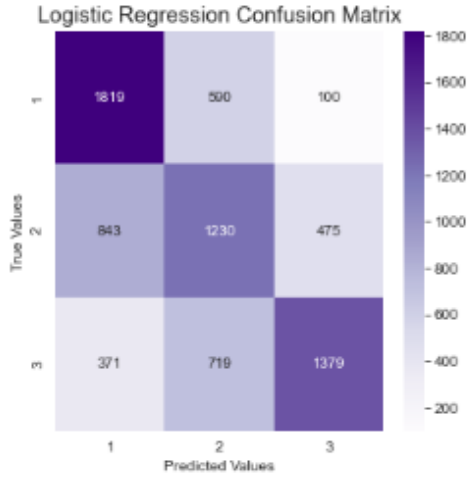
Explanation on parameters:

1. C = 0.1
   This parameter is the inverse of regularisation strength that has to be positive float numbers. The smaller the number, the stronger regularisation is, so in this case it is very strong since the value is only 0.1.
2. Penalty = 12
   Stand for L2 or Ridge regulation which adds a penalty term to the loss function that is proportional to the square of the magnitude of the coefficients. It will reduce the coefficients towards zero, but never set them exactly as zero.
3. Solver = liblinear
   A small and fast algorithm for logistic regression that optimises and utilises well for small datasets.

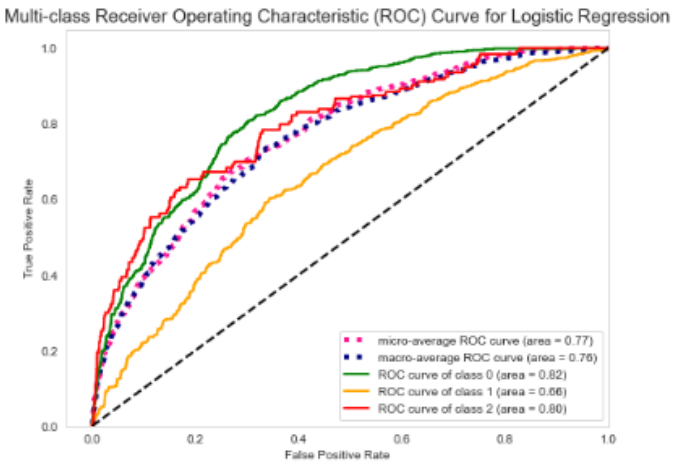
```
Confusion matrix :
 [[1819  590  100]
 [ 843 1230  475]
 [ 371  719 1379]]
```

```
Classification report :
```

	precision	recall	f1-score	support
1	0.60	0.72	0.66	2509
2	0.48	0.48	0.48	2548
3	0.71	0.56	0.62	2469
accuracy			0.59	7526
macro avg	0.60	0.59	0.59	7526
weighted avg	0.60	0.59	0.59	7526

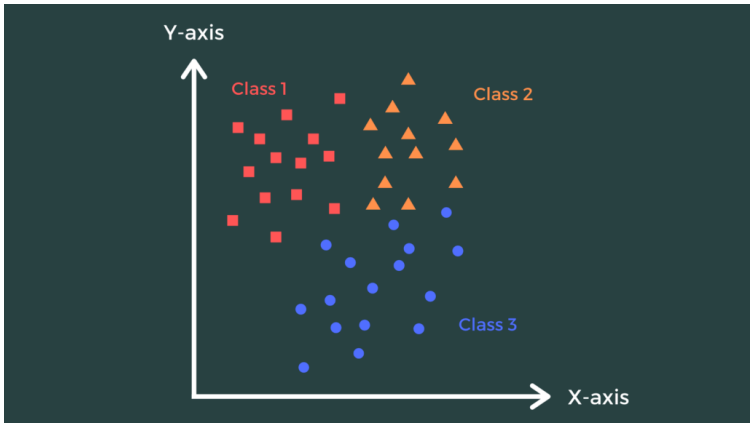


The result shows that the macro average for precision, recall and f1-score are 0.60, 0.59 and 0.59 respectively. The macro average precision and f1-score for this model improved after upsampling but its macro average recall worsened. As for its accuracy, it is the same as before the dataset was upsampled (0.59). The confusion matrix for this model is shown above. The ROC curve is shown below.


Multi-class Receiver Operating Characteristic (ROC) Curve for Logistic Regression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
micro-average ROC curve (area = 0.77)
macro-average ROC curve (area = 0.76)
ROC curve of class 0 (area = 0.82)
ROC curve of class 1 (area = 0.66)
ROC curve of class 2 (area = 0.80)

1.7 K Nearest Neighbour (Yong Zee Lin)

1.7.1 What is K Nearest Neighbour



K-nearest neighbours (KNN) is a popular, simple machine learning algorithm used for both classification and regression tasks. The basic principle behind KNN is that similar data points tend to belong to the same class or have similar output values.

In KNN, a new data point is classified or predicted based on the class or value of its k nearest neighbours in the feature space. To find the nearest neighbours, KNN measures the distance between the new data point and all other data points in the feature space. The most common distance measures used are Euclidean distance, Manhattan distance, or cosine similarity. Once the distances are calculated, the KNN algorithm selects the k closest neighbours to the new data point.

For classification, the KNN algorithm assigns the class that is most common among the k nearest neighbours to the new data point. For example, if most of the k nearest neighbours are classified as "apple," then the KNN algorithm would classify the new data point as "apple" as well. For regression, the KNN algorithm takes the average of the output values of the k nearest neighbours to predict the value of the new data point.

The value of k is usually chosen by the user, and it can affect the performance of the KNN algorithm. A small value of k will result in more local classification or regression decisions, while a larger value of k will result in more global decisions. However, a larger k can also increase the chance of misclassifying or overfitting the data.

KNN is a simple and intuitive algorithm, but it can be computationally expensive for large datasets or high-dimensional feature spaces. Additionally, the choice of distance measure and the value of k can significantly impact the accuracy of the algorithm. Nonetheless, KNN is still widely used as a baseline algorithm for comparison and as a component in more complex models.

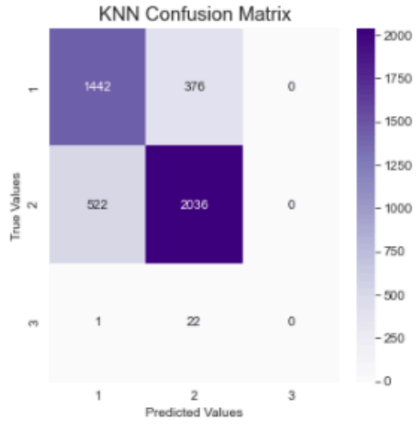
1.7.2 Result of K Nearest Neighbour Before Upsampling

```
numRange = list(range(1, 31))
paramGrid = dict(n_neighbors=numRange)

gridSearch = GridSearchCV(KNeighborsClassifier(), param_grid=paramGrid)
gridSearch.fit(trainFeatures, trainLabels)
print(gridSearch.best_estimator_)
```

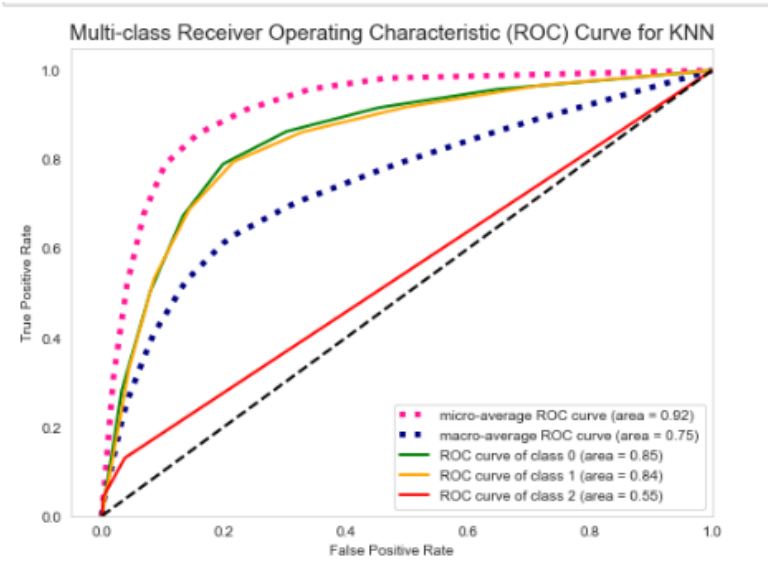
```
KNeighborsClassifier(n_neighbors=9)
```

Using GridSearchCV, we found that the best value for the n_neighbors is 9.

```
[[1442  376    0]
 [ 522 2036    0]
 [   1   22    0]]
              precision    recall  f1-score   support

           1       0.73      0.79      0.76      1818
           2       0.84      0.80      0.82      2558
           3       0.00      0.00      0.00        23

    accuracy                           0.79      4399
   macro avg       0.52      0.53      0.53      4399
weighted avg       0.79      0.79      0.79      4399
```



The result shows that the macro average for precision, recall and f1-score are 0.52, 0.53 and 0.53 respectively, which are considered as OK. As for its accuracy, it is 0.79, which is not bad. The confusion matrix for this model is shown above. The ROC curve is shown below.

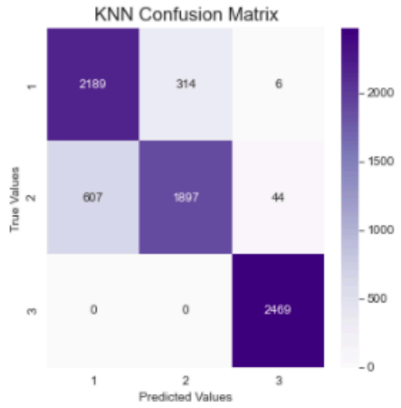


1.7.3 Result of K Nearest Neighbour After Upsampling

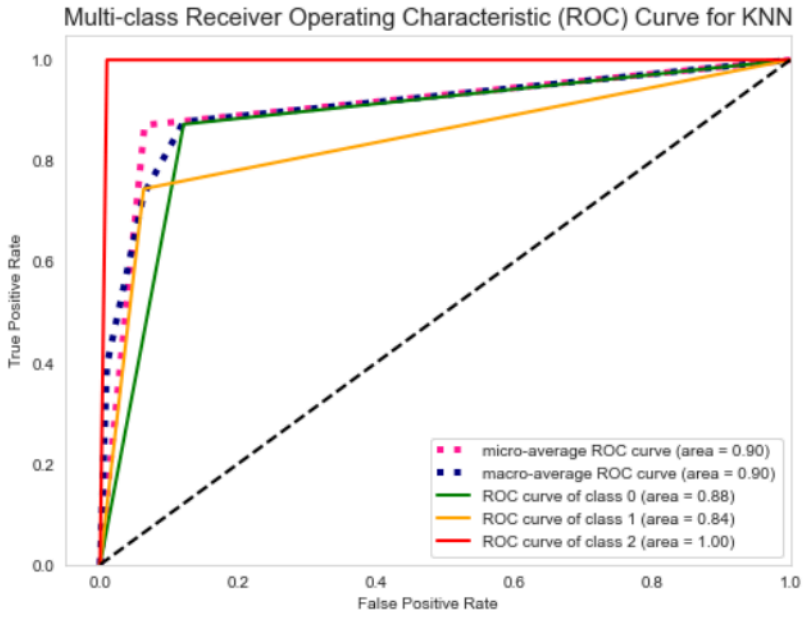
```
numRange = list(range(1, 31))
paramGrid = dict(n_neighbors=numRange)

gridSearch = GridSearchCV(KNeighborsClassifier(), param_grid=paramGrid)
gridSearch.fit(trainFeaturesv2, trainLabelsv2)
print(gridSearch.best_estimator_)
```

```
KNeighborsClassifier(n_neighbors=1)
```

Above, we try to find the best value for the n_neighbors hyperparameter of the K Nearest Neighbour model. Using GridSearchCV, the best value is 1.



The result shows that the macro average for precision, recall and f1-score are 0.87, 0.87 and 0.87 respectively, which are good. As for its accuracy, it is 0.87. These performance metrics tell us that the KNN model performed better after the upsampling of the dataset. The confusion matrix for this model is shown above. The ROC curve is shown below.



E. EVALUATION

1.1 Achievements

a) Results for all 4 Models BEFORE Upsampling (Imbalanced Dataset)

	Random Forest	Decision Tree	Logistic Regression	K Nearest Neighbours
Accuracy	87.57%	80.31%	59.40%	79.06%
Macro Avg Precision	62.12%	58.72%	47.59%	52.34%
Macro Avg Recall	67.56%	68.86%	62.58%	52.97%
Macro Avg F1-Score	63.54%	57.64%	44.27%	52.60%

Figure A.1: Showing Tabular Results of Performance Metrics for all 4 Models Before Upsampling

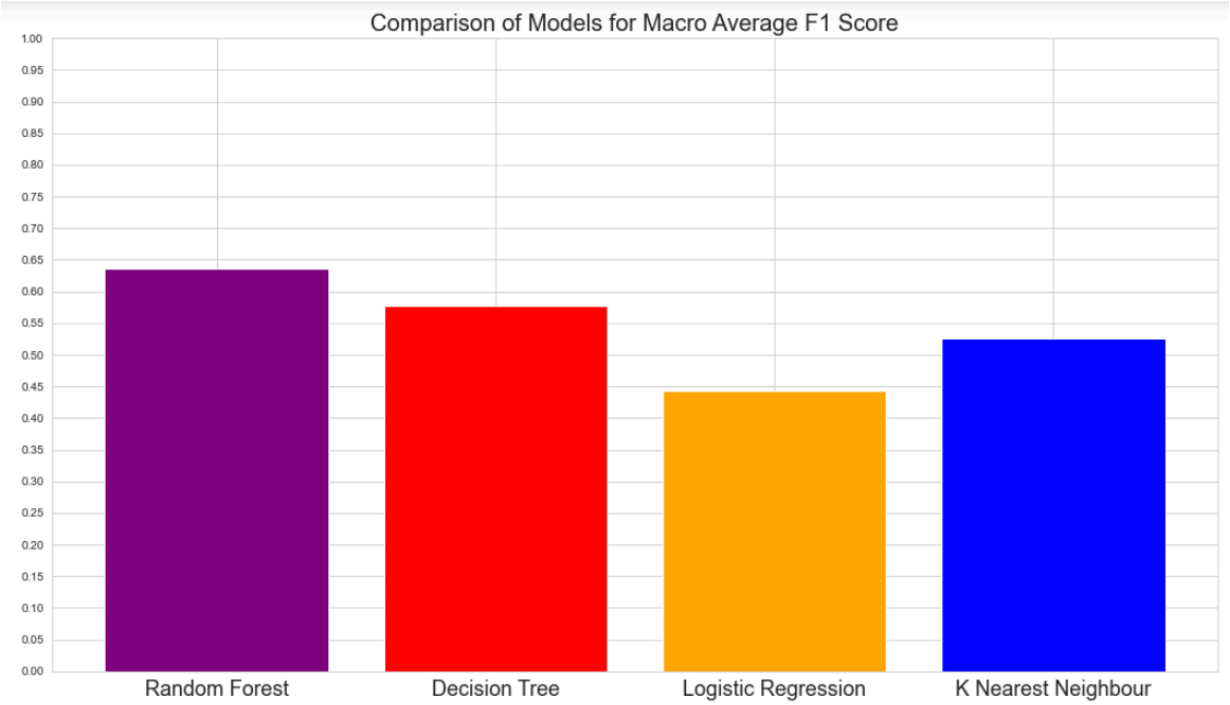


Figure A.2: Macro Average F1 Score of Models

Random Forest had the highest Macro Average F1 Score while Logistic Regression had the lowest.

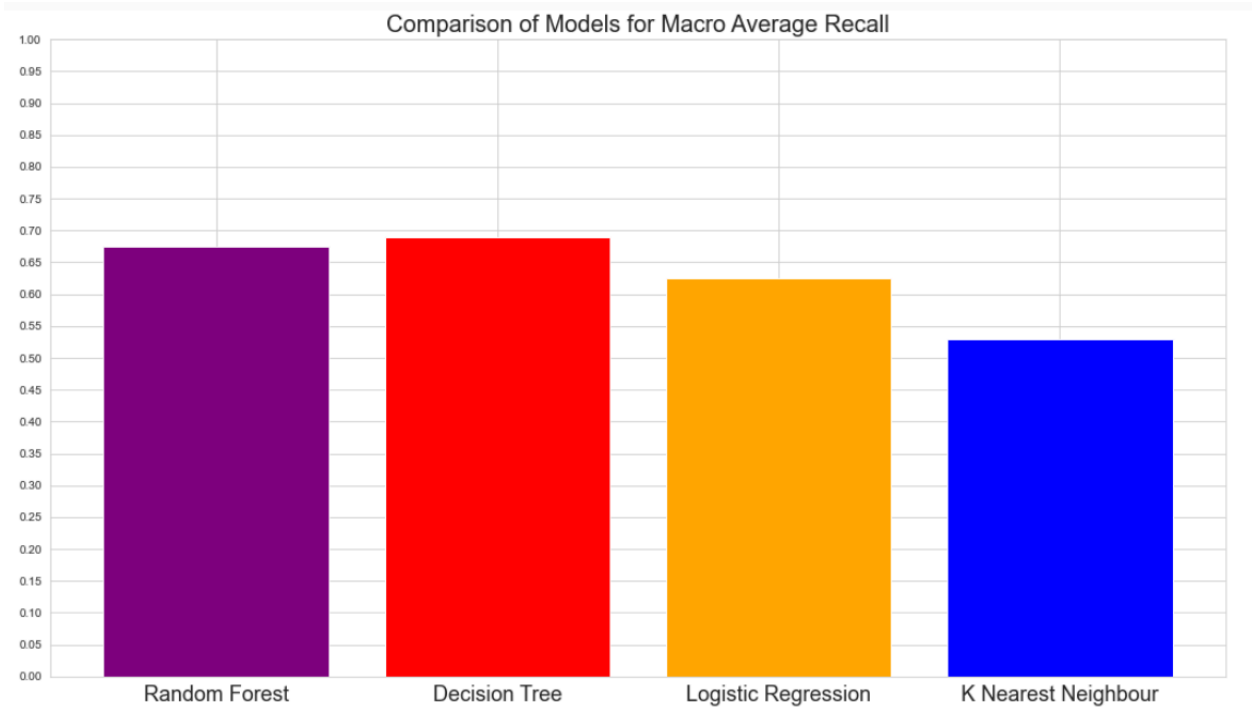


Figure A.3: Macro Average Recall of Models

Decision Tree had the highest Macro Average Recall while KNN had the lowest.

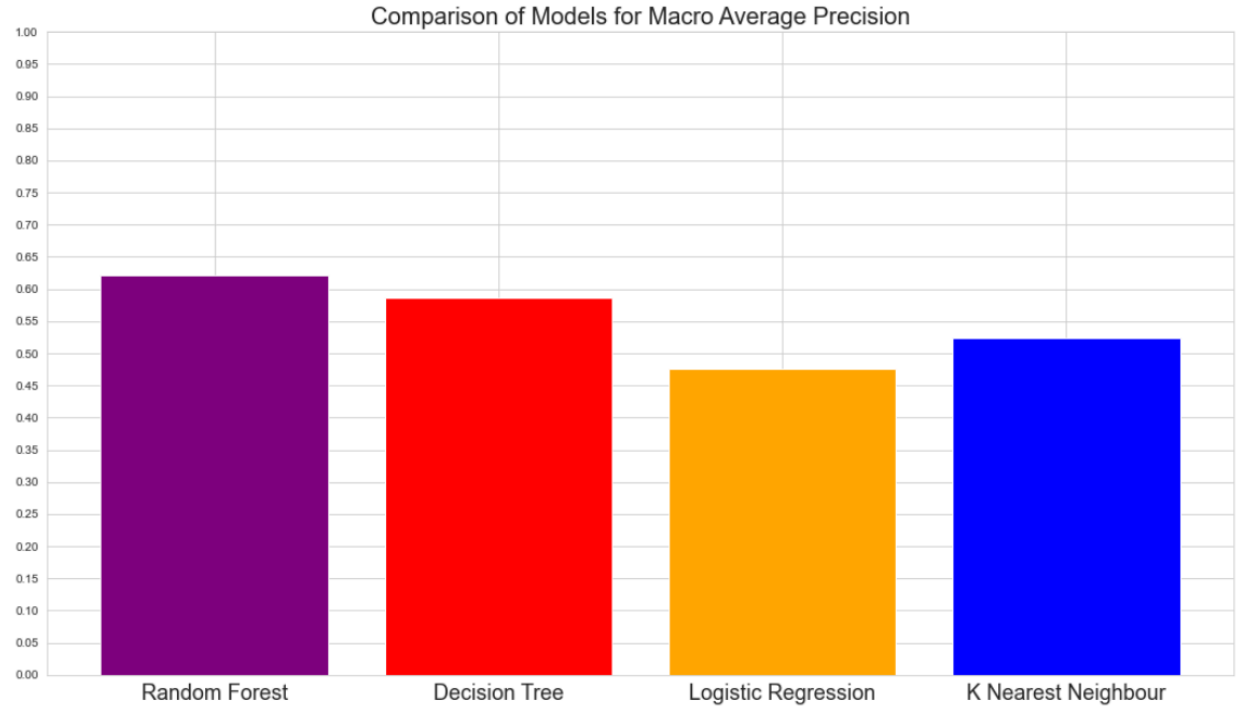


Figure A.4: Macro Average Precision of Models

Random Forest had the highest Macro Average Precision while Logistic Regression had the lowest.

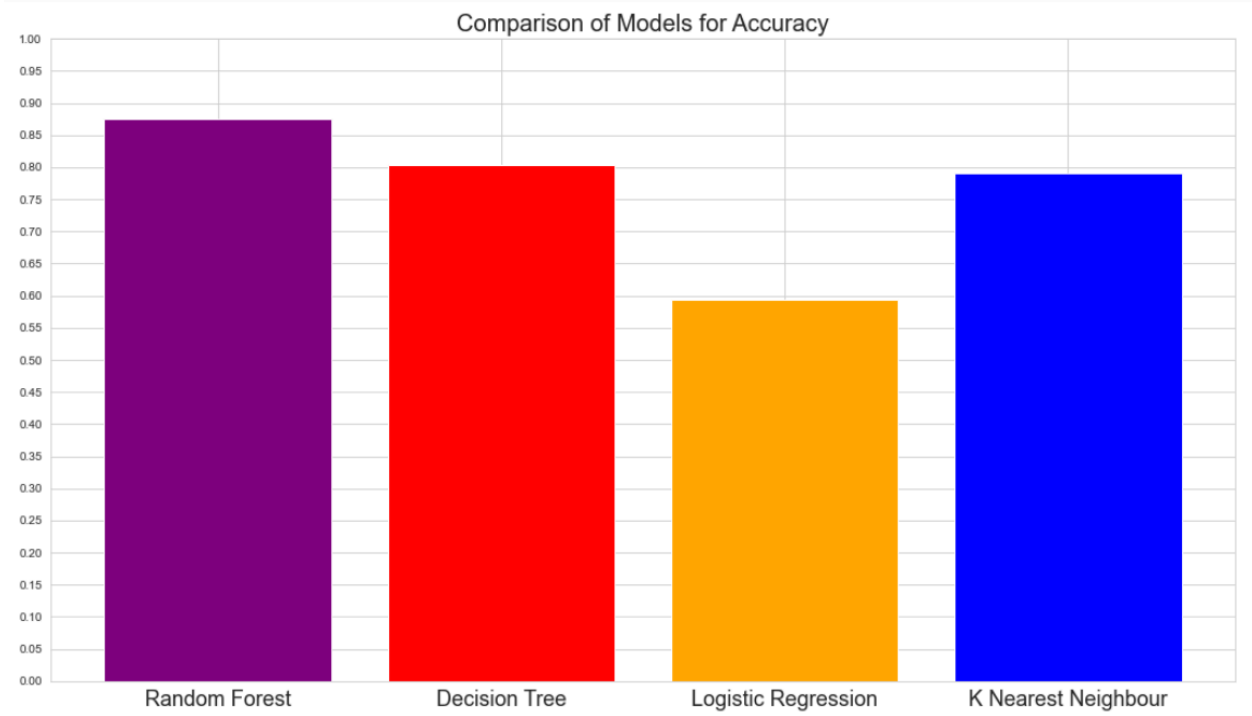


Figure A.5: Accuracy of Models

Random Forest had the highest accuracy while Logistic Regression had the lowest.

b) Results for all 4 Models AFTER Upsampling (Balanced Dataset)

	Random Forest	Decision Tree	Logistic Regression	K Nearest Neighbours
Accuracy	93.64%	87.02%	58.84%	87.10%
Macro Avg Precision	93.84%	86.82%	59.66%	87.37%
Macro Avg Recall	93.71%	86.98%	58.87%	87.23%
Macro Avg F1-Score	93.67%	86.82%	58.79%	87.08%

Figure B.1: Showing Tabular Results of Performance Metrics for all 4 Models After Upsampling

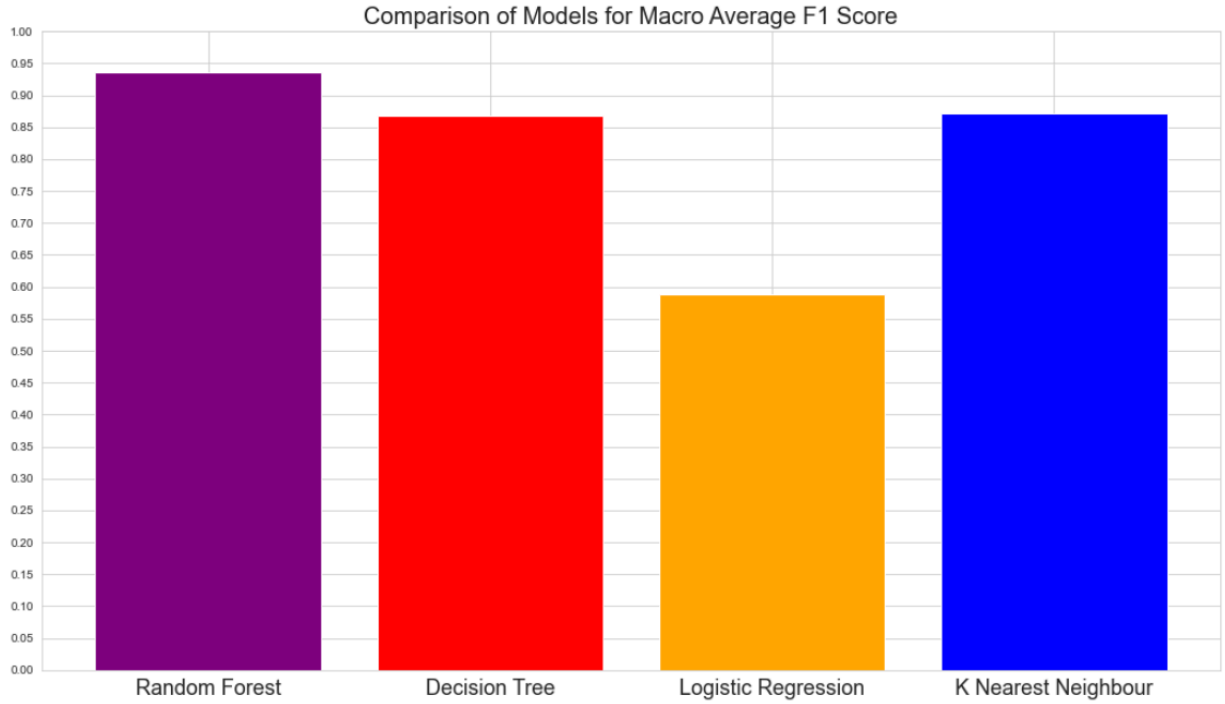


Figure B.2: Macro Average F1 Score of Models

Random Forest had the highest Macro Average F1 Score while Logistic Regression had the lowest Macro Average F1 Score.

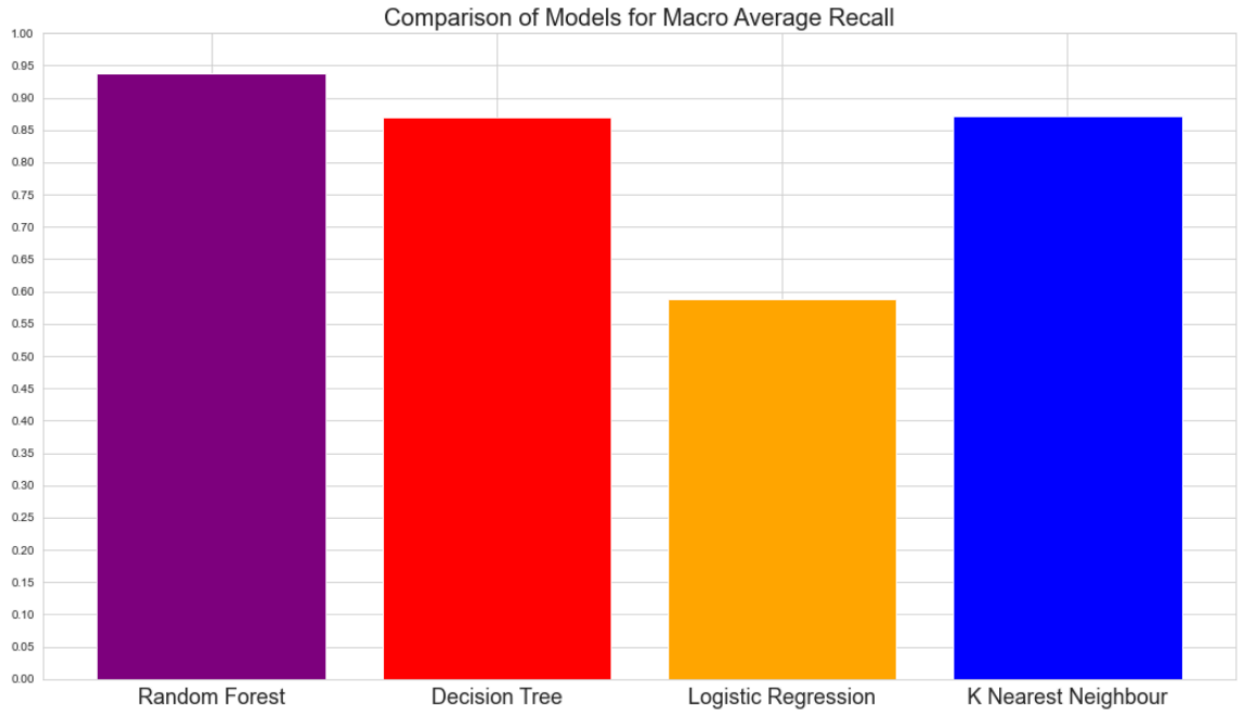


Figure B.3: Macro Average Recall of Models

Random Forest had the highest Macro Average Recall while Logistic Regression had the lowest.

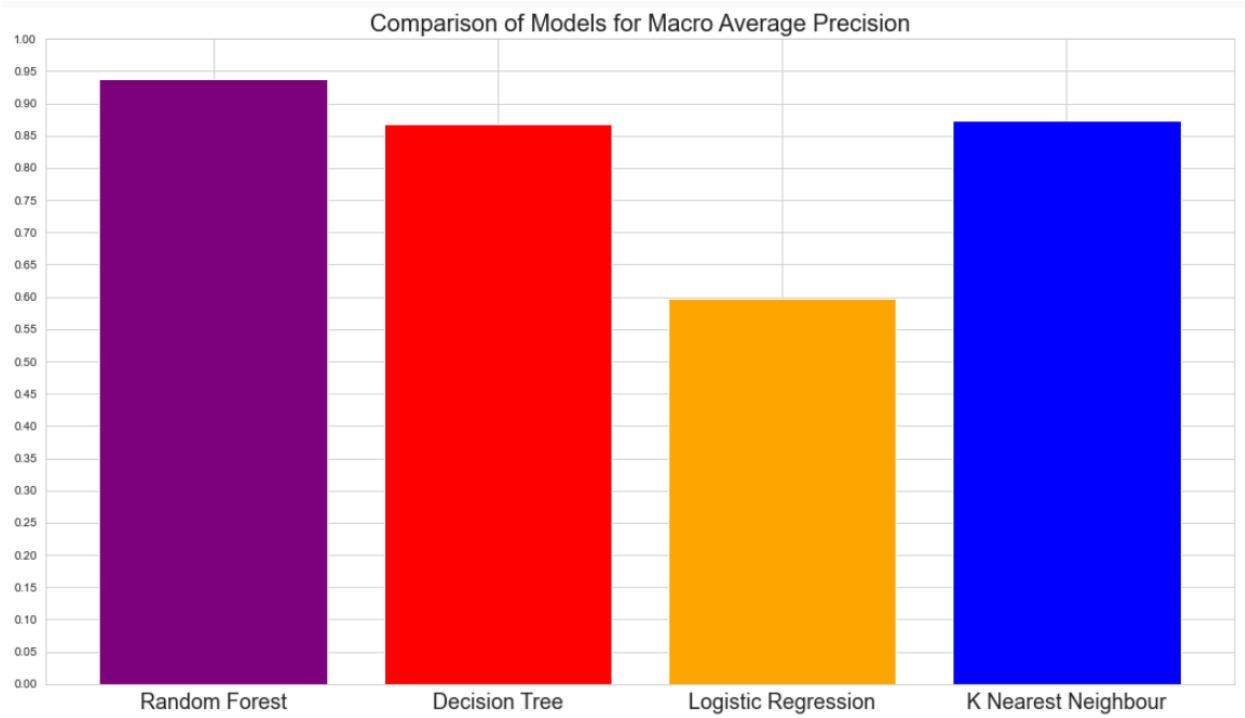


Figure B.4: Macro Average Precision of Models

Random Forest had the highest Macro Average Precision while Logistic Regression had the lowest.

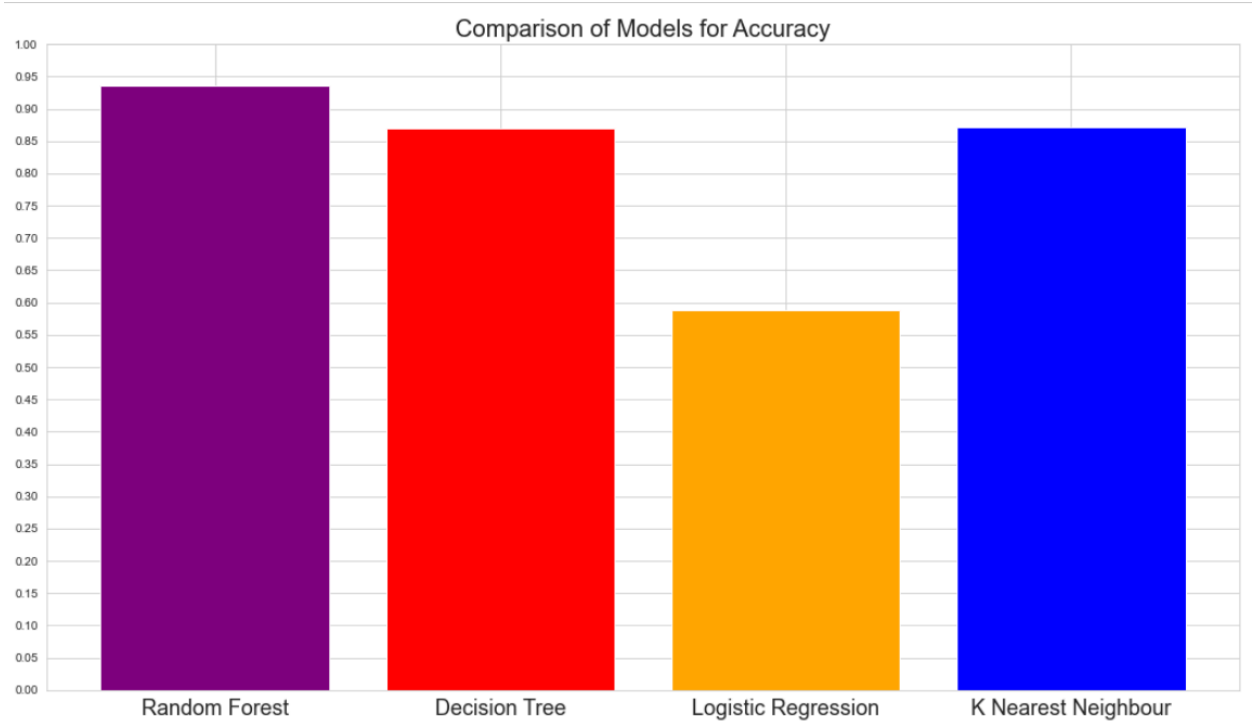


Figure B.5: Accuracy of Models

Random Forest had the highest accuracy while Logistic Regression had the lowest.

One of the objectives of our project is to achieve at least 80% for Accuracy, Macro Average Precision, Macro Average Recall and Macro Average F-1 Score for the Analytics application.

With reference to the results above:
Before upsampling, the performance metrics of the 4 models were OK or acceptable. The Random Forest model had the best performance while the Logistic Regression model had the worst performance.

After upsampling to balance the dataset, there was a significant improvement in performance for all 4 models as shown in the tables above. The Random Forest model had the best performance while the Logistic Regression model continued to have the worst performance.

Overall, the Random Forest Model is the best model. Below, we test this model on the test dataset.

c) Results of Random Forest Model on Test Data BEFORE Upsampling (Imbalanced Dataset)

```
from sklearn import metrics
randomF_predictedLabels = randomF_model.predict(testFeatures)
print(metrics.confusion_matrix(testLabels, randomF_predictedLabels))
print(metrics.classification_report(testLabels, randomF_predictedLabels))
```

```
[[1724  170    2]
 [ 304 2129   53]
 [   3   11    3]]
              precision    recall  f1-score   support

           1       0.85      0.91      0.88      1896
           2       0.92      0.86      0.89      2486
           3       0.05      0.18      0.08        17

    accuracy                           0.88      4399
   macro avg       0.61      0.65      0.62      4399
weighted avg       0.89      0.88      0.88      4399
```

```
randomF_pr = precision_score(testLabels, randomF_predictedLabels, average='macro')
randomF_rc = recall_score(testLabels, randomF_predictedLabels, average='macro')
randomF_f1 = f1_score(testLabels, randomF_predictedLabels, average='macro')
randomF_ac = accuracy_score(testLabels, randomF_predictedLabels)

print("Precision for Random Forest: ", format(randomF_pr * 100, ".2f"),"%")
print("Recall for Random Forest: ", format(randomF_rc * 100, ".2f"),"%")
print("F1 Score for Random Forest: ", format(randomF_f1 * 100, ".2f"),"%")
print("Accuracy for Random Forest: ", format(randomF_ac * 100, ".2f"),"%")
```

```
Precision for Random Forest:  60.74 %
Recall for Random Forest:  64.74 %
F1 Score for Random Forest:  61.53 %
Accuracy for Random Forest:  87.66 %
```

	On Validation Dataset	On Test Dataset
Accuracy	87.57%	87.66%
Macro Avg Precision	62.12%	60.74%
Macro Avg Recall	67.56%	64.74%
Macro Avg F1-Score	63.54%	61.53%

The performance of our Random Forest Model on both validation and test datasets are comparable with the performance on the test dataset slightly lower in general.

d) Results of Random Forest Model on Test Data AFTER Upsampling (Imbalanced Dataset)

```
randomF_predictedLabelsv3_def = randomF_modelv2_def.predict(testFeaturesv2)
print(metrics.confusion_matrix(testLabelsv2, randomF_predictedLabelsv3_def))
print(metrics.classification_report(testLabelsv2, randomF_predictedLabelsv3_def))
```

```
[[2388  134    0]
 [ 341 2183   15]
 [   0    0 2466]]
              precision    recall  f1-score   support

           1       0.88      0.95      0.91      2522
           2       0.94      0.86      0.90      2539
           3       0.99      1.00      1.00      2466

    accuracy                           0.93      7527
   macro avg       0.94      0.94      0.94      7527
weighted avg       0.94      0.93      0.93      7527
```

```
randomF_pr = precision_score(testLabelsv2, randomF_predictedLabelsv3_def, average='macro')
randomF_rc = recall_score(testLabelsv2, randomF_predictedLabelsv3_def, average='macro')
randomF_f1 = f1_score(testLabelsv2, randomF_predictedLabelsv3_def, average='macro')
randomF_ac = accuracy_score(testLabelsv2, randomF_predictedLabelsv3_def)

print("Precision for Random Forest: ", format(randomF_pr * 100, ".2f"),"%")
print("Recall for Random Forest: ", format(randomF_rc * 100, ".2f"),"%")
print("F1 Score for Random Forest: ", format(randomF_f1 * 100, ".2f"),"%")
print("Accuracy for Random Forest: ", format(randomF_ac * 100, ".2f"),"%")
```

```
Precision for Random Forest:  93.71 %
Recall for Random Forest:  93.56 %
F1 Score for Random Forest:  93.52 %
Accuracy for Random Forest:  93.49 %
```

	On Validation Dataset	On Testing Dataset
Accuracy	93.64%	93.49%
Macro Avg Precision	93.84%	93.71%
Macro Avg Recall	93.71%	93.56%
Macro Avg F1-Score	93.67%	93.52%

The performance of our Random Forest Model on both validation and test datasets are comparable with the performance on the test dataset slightly lower in general. Therefore, we will deploy this Random Forest model based on the upsampled data set because the performance on the upsampled dataset is better than that on the unbalanced dataset. This project has helped to achieve our objectives.

1.2 Discussions (Limitations of Project and Improvements that can be done in the future)

A major limitation we encountered in this project is the extremely **imbalanced dataset**. The 'Fatal' severity code only represents about 0.5% of all the samples in the original dataset. Because there are so few samples of this minority target class to learn from, the classifiers tend to favour the majority classes when making predictions. As a result, we attempted to rectify this situation by **upsampling** both the 'Fatal' and 'No Injuries' classes to match the number of samples in the class with the most samples (which is the 'Not Fatal' class). The performance of our classifiers improved significantly after running on the balanced dataset. For future improvement, we would like to try the SMOTE upsampling technique. We did not use it because we had difficulties installing the *imblearn* library. Instead of simply generating duplicates, SMOTE creates synthetic samples that are slightly different from the original samples.

Another limitation we have in our project is the lack of **data quality** in our dataset. The first issue is there are a lot of empty slots in the dataset. There is also an input called Nao Informado which is 'No information' in English which is also considered as empty or unusable data. The second problem is the illogical data such as the age column. There is quite a lot of data with 0 inserted as age and 00/00/0000 as the birth date. These two issues cause those particular rows of data unusable due to illogical or empty data. For age as it is too important to be left out, we had decided to do an extra step in the data preparation and insert the mean as new data to the age column that has 0 as its value. However, since there are a lot of those 0 data in age, we cannot predict a complete accuracy as we do not know the actual value because a huge shift in the data might change the overall prediction. For example, since the mean we inserted is 36, but in reality the car accidents might include a generally older passenger or driver. The solution is to ensure good data is selected. Before selecting a dataset it is important to make sure the source is reliable. For instance, the details of the content within the dataset, do most of the columns have logical content or is it empty and unusable?, are the variables collected important for the prediction and analysis of the main variable?. These are some of the factors that we need to know before selecting a dataset to work with. Then, it will more accurately reflect the real situation and ensure a more precise prediction and understanding of the relationship between variables in this project.

It seems to us that the dataset provided to us is **missing important variables** that have a significant impact on the occurrence and severity of accidents. For instance, there is no information on weather conditions, type of road, speed limit on the road, where the accident occurred, and any special feature of the road segment where the accident took place (e.g. junction, roundabout, blind corner, etc.). If this additional information can be collected by the Brazilian department of transit, we may be able to produce a better machine learning classifier.

For this project, we **only used 4 machine learning algorithms** to create our classifier. For future work, we would like to try using additional algorithms, especially Neural Networks and other ensemble algorithms such as XGBoost and AdaBoost.

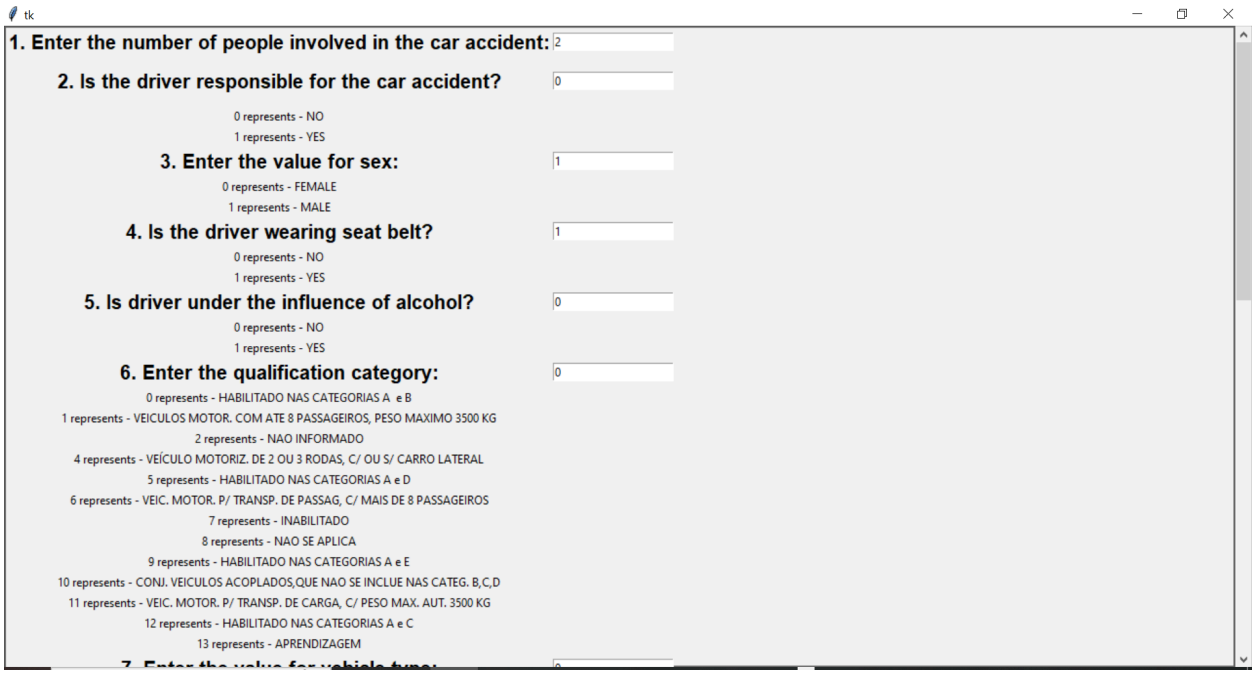
F. DEPLOYMENT

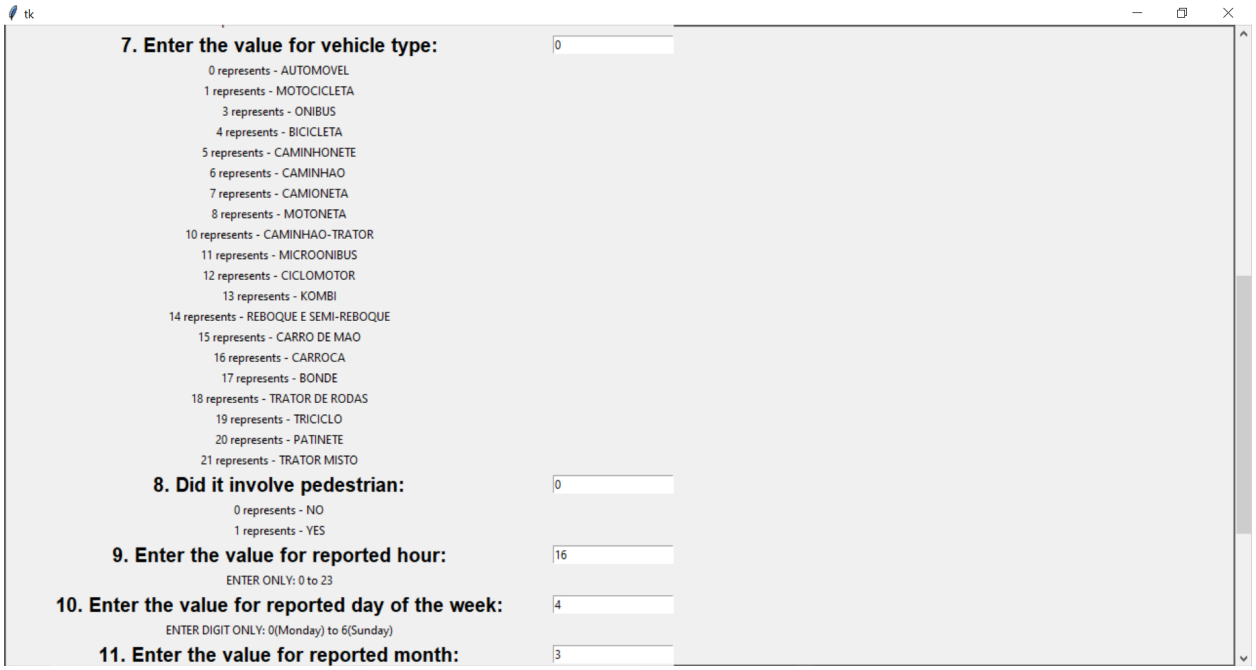
We have written Tkinter code to accept multiple inputs from the user in order to predict the severity code of the car accident.

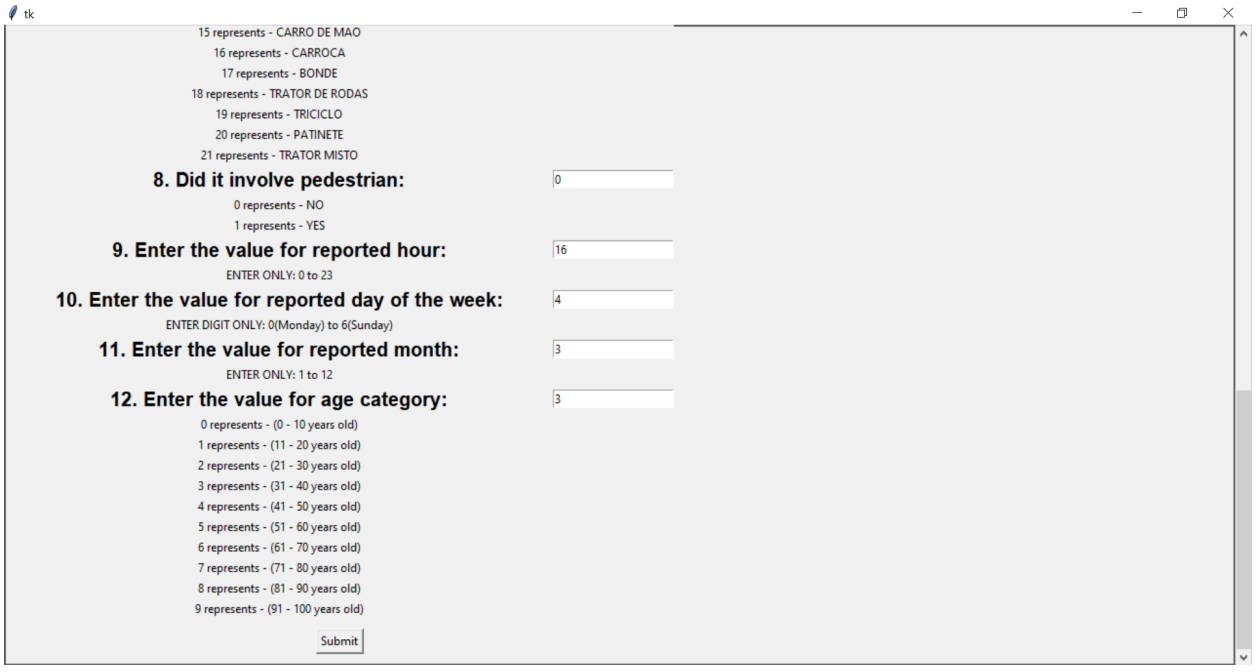
User will be asked to enter:

1. The number of people involved
2. Whether the driver was responsible for the accident
3. Sex
4. Whether the driver was wearing a seatbelt
5. Whether the driver was under the influence of alcohol
6. The qualification category
7. Vehicle Type
8. Whether a pedestrian was involved
9. The reported hour
10. The reported day of week
11. The reported month
12. Age category

To use the GUI, fill in the input fields and then click the Submit button at the bottom of the window. The default GUI only can be fitted up to a certain size and range, therefore we have to use a function called to overcome the limitation of the size and make it flexible to the user handling the input using tkinter.




tk
7. Enter the value for vehicle type:
0
0 represents - AUTOMOVEL
1 represents - MOTOCICLETA
3 represents - ONIBUS
4 represents - BICICLETA
5 represents - CAMINHONETE
6 represents - CAMINHAO
7 represents - CAMIONETA
8 represents - MOTONETA
10 represents - CAMINHAO-TRATOR
11 represents - MICROONIBUS
12 represents - CICLOMOTOR
13 represents - KOMBI
14 represents - REBOQUE E SEMI-REBOQUE
15 represents - CARRO DE MAO
16 represents - CARROCA
17 represents - BONDE
18 represents - TRATOR DE RODAS
19 represents - TRICICLO
20 represents - PATINETE
21 represents - TRATOR MISTO
8. Did it involve pedestrian:
0
0 represents - NO
1 represents - YES
9. Enter the value for reported hour:
16
ENTER ONLY: 0 to 23
10. Enter the value for reported day of the week:
4
ENTER DIGIT ONLY: 0(Monday) to 6(Sunday)
11. Enter the value for reported month:
3


tk
15 represents - CARRO DE MAO
16 represents - CARROCA
17 represents - BONDE
18 represents - TRATOR DE RODAS
19 represents - TRICICLO
20 represents - PATINETE
21 represents - TRATOR MISTO
8. Did it involve pedestrian:
0
0 represents - NO
1 represents - YES
9. Enter the value for reported hour:
16
ENTER ONLY: 0 to 23
10. Enter the value for reported day of the week:
4
ENTER DIGIT ONLY: 0(Monday) to 6(Sunday)
11. Enter the value for reported month:
3
ENTER ONLY: 1 to 12
12. Enter the value for age category:
3
0 represents - (0 - 10 years old)
1 represents - (11 - 20 years old)
2 represents - (21 - 30 years old)
3 represents - (31 - 40 years old)
4 represents - (41 - 50 years old)
5 represents - (51 - 60 years old)
6 represents - (61 - 70 years old)
7 represents - (71 - 80 years old)
8 represents - (81 - 90 years old)
9 represents - (91 - 100 years old)
Submit

Result of pressing submit button after filling out the form:

```
The numeber of people involved in the car accident entered is: 2
Is the driver resposible for the car accident: 0
The value for sex entered is: 1
Is the driver wearing seat belt: 1
The value for driver under alcohol entered is: 0
The value for qualification Category is: 0
The value for vehicle type entered is: 0
The value for involved pedestrain entered is: 0
The value for reported hour entered is: 16
The value for reported day of the week entered is: 4
The value for reported month entered is: 3
The value for age category entered is: 3
[3]
Severity of Accident: Fatal
```

**This output will be displayed in the Jupyter Notebook

From the result above, we can observe that the model predicted the severity of the accident using the user's input. The GUI only allows the user to enter the values in integer because to train the dataset, we can only utilise integer instead of categorical values such as string. Therefore the deployment works as it predicted the outcome of the accident.

G. CONCLUSION

Advantages and disadvantages

a. Random Forest

Starting off with the advantages, its performance is better for a large variety of data items than a decision tree. Secondly, a random forest can solve the problem of overfitting and can be used for regression and classification jobs. Another benefit is that random forest is not significantly affected by outliers. Data scaling is not required by random forest so even if we provide data that is not scaled, good accuracy can still be maintained.

There are disadvantages as well. It requires more effort and time to build a Random Forest than a decision tree. Secondly, the random forest's process of prediction does not take a short time when compared to other models/algorithms. Besides that, random forest is complex and needs more computational power. If it faces a large dataset, it can be demanding in terms of computation.

b. Decision Tree

There are many advantages for using the decision tree model to identify the result for the dataset. First and foremost is easy to implement, because if the user has no data preparation the dataset will still be able to use, but in one condition the training dataset and testing dataset have to be in numerical form because without the numerical form, the compiler will have error. Apart from that, We can say that the data can include generating output and results for analyzing. Not only that, the flexibility of the code is also very wide to edit, because if we want to find the entropion and the max_depth we have to edit to run the code to find the most optimal result. So we can perform trial and error in the model to get the best result.

As for the disadvantages, it is very costly to operate because the decision tree model needs time and effort to find the best nodes to operate as the machines need more time to generate the best result for each node. Not only costly to operate but also its can be overfitted because if we make the max_depth to big of the value the values will be very easy to be overfitted.

c. Logistic Regression

The advantage of logistic regression is that it is easy to implement in certain situations such as having only one dependent variable. In addition, having an option to put a clear cut off value to adjust between cases, especially using ROC curve directly as a way to choose a cutoff value. Next, logistic regression can easily extend into a multiclass classification which our dataset is.

However, some of its disadvantages is needing all the independent variables to be completely independent from each to each other, meaning that no sub-groups of dependent data that is independent to the main variable but not to each other can be inside the dataset. Next, it is not the most accurate with its prediction especially in high dimensional dataset because it will have a hard time capturing complex relationships. Therefore, it needs quite a big dataset to analyse in order to capture the most accurate correlation between each variable.

d. K-Nearest Neighbours

The advantages of K-Nearest Neighbours (KNN) is it is a popular classification and regression algorithm in machine learning that is simple and easy to understand. Some advantages of KNN include its non-parametric nature, meaning that it does not require any assumptions about the data distribution, and its ability to handle both continuous and discrete data. Additionally, KNN has low computational cost during the training phase and can be easily updated with new data.

However, there are also some disadvantages of KNN. One significant disadvantage is that it can be computationally expensive during the testing phase, especially when working with large datasets. KNN also suffers from the "curse of dimensionality," where the distance between points becomes less meaningful in high-dimensional space. Furthermore, KNN is highly sensitive to the choice of distance metric and can be influenced by the presence of noisy or irrelevant features in the dataset. Finally, KNN has a poor performance on imbalanced datasets, where the number of samples in one class is significantly higher than the other classes.

H. REFERENCE

1. AnalyticsVidhya (2023), Decision Tree Algorithm - A Complete Guide. Available at: https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/#:~:text=Root%20Nodes%20%E2%80%93%20It%20is%20the,nodes%20are%20called%20Decision%20Node(Accessed: 6 May 2023).

2. IBM (*no date*), What is a Decision Tree. Available at: https://www.ibm.com/topics/decision-trees#:~:text=A%20decision%20tree%20is%20a,internal%20nodes%20and%20leaf%20nodes(Accessed: 6 May 2023).

3. Simplilearn (2023), An Introduction to Logistic Regression in Python, Simplilearn.com. Available at: https://www.simplilearn.com/tutorials/machine-learning-tutorial/logistic-regression-in-python#advantages_of_the_logistic_regression_algorithm. (Accessed: 6 May 2023).

4. MachineLearningMastery (2023), How to Connect Model Input Data With Predictions for Machine Learning, MachineLearningMastery.com. Available at: https://machinelearningmastery.com/how-to-connect-model-input-data-with-predictions-for-machine-learning/. (Accessed: 6 May 2023).

5. Scikit (2023), 1.12. Multiclass and multioutput algorithms. Available at: https://scikit-learn.org/stable/modules/multiclass.html. (Accessed: 6 May 2023).

6. Scikit (2023), Multiclass Receiver Operating Characteristic (ROC). Available at: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html. (Accessed: 6 May 2023).

7. Scikit (2023), sklearn.decomposition.PCA. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. (Accessed: 6 May 2023).

8. NAN (2023), The Free Dictionary. Available at: https://acronyms.thefreedictionary.com/NAN. (Accessed: 6 May 2023).

9. MachineLearningMastery (2020), What is a Confusion Matrix in Machine Learning, MachineLearningMastery.com. Available at: https://machinelearningmastery.com/confusion-matrix-machine-learning/. (Accessed: 6 May 2023).

10. Random Forests (2019). Available at: https://kevintshoemaker.github.io/NRES-746/RandomForests.html. (Accessed: 6 May 2023).

11. Editorial (2022) Pros and cons of Random Forest algorithm, RoboticsBiz. Available at: https://roboticsbiz.com/pros-and-cons-of-random-forest-algorithm/ (Accessed: May 7, 2023).

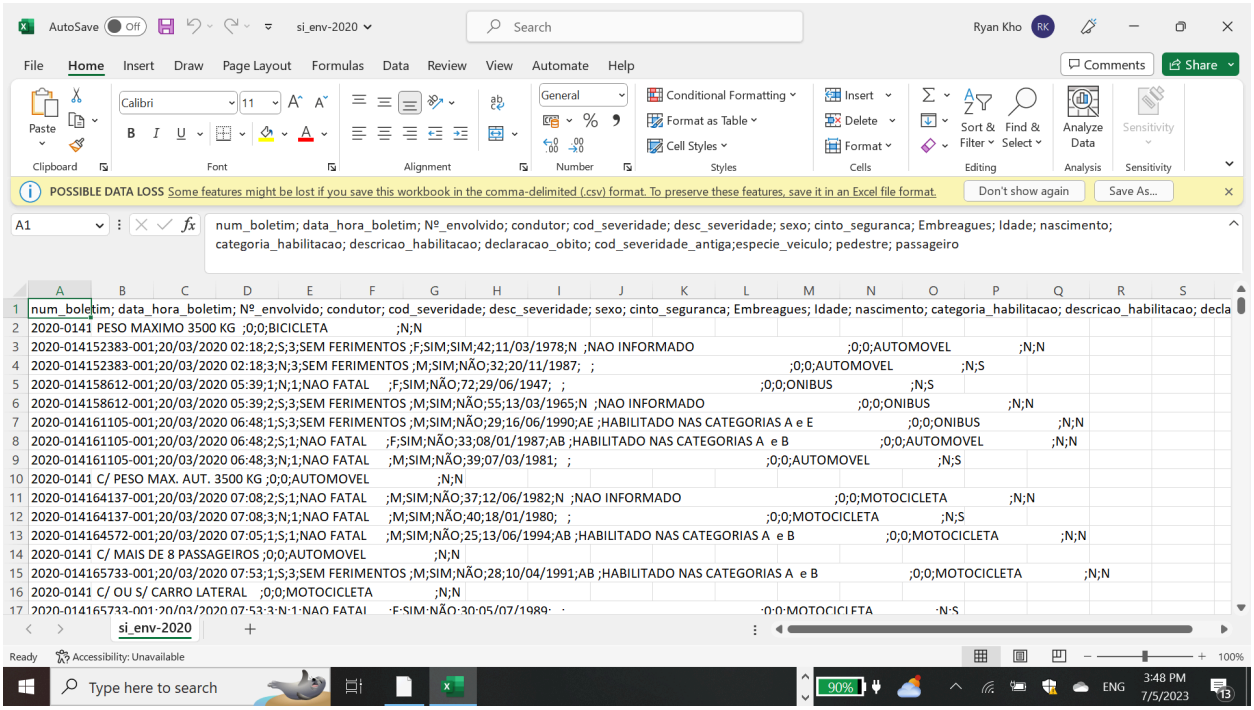
12. Wells, R. (2021) Upsampling and downsampling imbalanced data in python, wellsr.com. Available at: https://wellsr.com/python/upsampling-and-downsampling-imbalanced-data-in-python/ (Accessed: May 7, 2023).
13. Malick Sarr and MAk (2021) How to split your dataset to train, test and validation sets? [python], Malick Sarr. Available at: https://www.malicksarr.com/split-train-test-validation-python/ (Accessed: May 7, 2023).
14. Lee, W.-M. (2023) Dealing with date and time in pandas DataFrames, Medium. Towards Data Science. Available at: https://towardsdatascience.com/dealing-with-date-and-time-in-pandas-dataframes-7d140f711a47 (Accessed: May 7, 2023).

15. Data to fish (no date) Data to Fish. Available at: https://datatofish.com/replace-values-pandas-dataframe/ (Accessed: May 7, 2023).

I. APPENDIX

I.1 Sample Data Records

I.1.1The original dataset from Kaggle



si_env-2020.csv

(https://www.kaggle.com/datasets/raphaelmarconato/detran-accidents-2020)

What the dataset looks like in Jupyter Notebook Originally

	num_boletim	data_hora_boletim	Nº_envolvido	condutor	cod_severidade	desc_severidade	sexo	cinto_seguranca	Embreagues	Idade	nascimento	ca
0	2020-014152383-001	20/03/2020 02:18	1	S	1	NAO FATAL	M	NÃO	NÃO	35	06/09/1984	
1	2020-014152383-001	20/03/2020 02:18	2	S	3	SEM FERIMENTOS	F	SIM	SIM	42	11/03/1978	
2	2020-014152383-001	20/03/2020 02:18	3	N	3	SEM FERIMENTOS	M	SIM	NÃO	32	20/11/1987	
3	2020-014158612-001	20/03/2020 05:39	1	N	1	NAO FATAL	F	SIM	NÃO	72	29/06/1947	
4	2020-014158612-001	20/03/2020 05:39	2	S	3	SEM FERIMENTOS	M	SIM	NÃO	55	13/03/1965	
5	2020-014161105-001	20/03/2020 06:48	1	S	3	SEM FERIMENTOS	M	SIM	NÃO	29	16/06/1990	
6	2020-014161105-001	20/03/2020 06:48	2	S	1	NAO FATAL	F	SIM	NÃO	33	08/01/1987	
7	2020-014161105-001	20/03/2020 06:48	3	N	1	NAO FATAL	M	SIM	NÃO	39	07/03/1981	
8	2020-014164137-001	20/03/2020 07:08	1	S	3	SEM FERIMENTOS	M	SIM	NÃO	30	09/12/1989	
9	2020-014164137-001	20/03/2020 07:08	2	S	1	NAO FATAL	M	SIM	NÃO	37	12/06/1982	
10	2020-014164137-001	20/03/2020 07:08	3	N	1	NAO FATAL	M	SIM	NÃO	40	18/01/1980	
11	2020-014164572-	20/03/2020 07:05	1	S	1	NAO FATAL	M	SIM	NÃO	25	13/06/1994	

2]:

cinto_seguranca	Embreagues	Idade	nascimento	categoria_habilitacao	descricao_habilitacao	declaracao_obito	cod_severidade_antiga	especie_veiculo	pe
NÃO	NÃO	35	06/09/1984	B	VEICULOS MOTOR. COM ATE 8 PASSAGEIROS, PESO MA...	0	0	BICICLETA	
SIM	SIM	42	11/03/1978	N	NAO INFORMADO ...	0	0	AUTOMOVEL	
SIM	NÃO	32	20/11/1987		...	0	0	AUTOMOVEL	
SIM	NÃO	72	29/06/1947		...	0	0	ONIBUS	
SIM	NÃO	55	13/03/1965	N	NAO INFORMADO ...	0	0	ONIBUS	
SIM	NÃO	29	16/06/1990	AE	HABILITADO NAS CATEGORIAS A e E ...	0	0	ONIBUS	
SIM	NÃO	33	08/01/1987	AB	HABILITADO NAS CATEGORIAS A e B ...	0	0	AUTOMOVEL	
SIM	NÃO	39	07/03/1981		...	0	0	AUTOMOVEL	
SIM	NÃO	30	09/12/1989	C	VEIC. MOTOR. P/ TRANSP. DE CARGA, C/ PESO MAX....	0	0	AUTOMOVEL	
SIM	NÃO	37	12/06/1982	N	NAO INFORMADO ...	0	0	MOTOCICLETA	
SIM	NÃO	40	18/01/1980		...	0	0	MOTOCICLETA	
SIM	NÃO	25	13/06/1994	AB	HABILITADO NAS	0	0	MOTOCICLETA	

]:

Embreagues	Idade	nascimento	categoria_habilitacao	descricao_habilitacao	declaracao_obito	cod_severidade_antiga	especie_veiculo	pedestre	passageiro
NÃO	35	06/09/1984	B	VEICULOS MOTOR. COM ATE 8 PASSAGEIROS, PESO MA...	0	0	BICICLETA	N	N
SIM	42	11/03/1978	N	NAO INFORMADO ...	0	0	AUTOMOVEL	N	N
NÃO	32	20/11/1987		...	0	0	AUTOMOVEL	N	S
NÃO	72	29/06/1947		...	0	0	ONIBUS	N	S
NÃO	55	13/03/1965	N	NAO INFORMADO ...	0	0	ONIBUS	N	N
NÃO	29	16/06/1990	AE	HABILITADO NAS CATEGORIAS A e E ...	0	0	ONIBUS	N	N
NÃO	33	08/01/1987	AB	HABILITADO NAS CATEGORIAS A e B ...	0	0	AUTOMOVEL	N	N
NÃO	39	07/03/1981		...	0	0	AUTOMOVEL	N	S
NÃO	30	09/12/1989	C	VEIC. MOTOR. P/ TRANSP. DE CARGA, C/ PESO MAX....	0	0	AUTOMOVEL	N	N
NÃO	37	12/06/1982	N	NAO INFORMADO ...	0	0	MOTOCICLETA	N	N
NÃO	40	18/01/1980		...	0	0	MOTOCICLETA	N	S
NÃO	25	13/06/1994	AB	HABILITADO NAS	0	0	MOTOCICLETA	N	N

I.1.2 What the dataset looks like after cleaning and transformation

	noPeopleInvolved	driverResponsibleForAccident	severityCode	sex	woreSeatBelt	driverUnderAlcohol	qualificationCat	vehicleType	involvedPedestrian
0	1	1	2	1	0	0	1	4	0
1	2	1	1	0	1	1	0	0	0
2	3	0	1	1	1	0	0	0	0
3	1	0	2	0	1	0	0	3	0
4	2	1	1	1	1	0	0	3	0
5	1	1	1	1	1	0	9	3	0
6	2	1	2	0	1	0	0	0	0
7	3	0	2	1	1	0	0	0	0
8	1	1	1	1	1	0	11	0	0
9	2	1	2	1	1	0	0	1	0
10	3	0	2	1	1	0	0	1	0
11	1	1	2	1	1	0	0	1	0
12	2	1	1	1	1	0	6	0	0
13	1	1	1	1	1	0	0	1	0
14	2	1	2	1	1	0	4	1	0
15	3	0	2	0	1	0	0	1	0
16	1	1	2	1	1	1	7	1	0
17	2	1	1	1	1	0	0	0	0
18	1	1	2	1	1	0	0	1	0
19	2	1	1	1	1	0	0	0	0
20	3	0	1	0	1	0	0	0	0
21	1	1	2	1	1	0	0	1	0
22	2	1	1	0	1	0	1	0	0
23	1	1	2	1	1	0	0	1	0

]:

sex	woreSeatBelt	driverUnderAlcohol	qualificationCat	vehicleType	involvedPedestrian	reportedHour	reportedDayOfWeek	reportedMonth	ageCategoryNum
1	0	0	1	4	0	2	4	3	3
0	1	1	0	0	0	2	4	3	4
1	1	0	0	0	0	2	4	3	3
0	1	0	0	3	0	5	4	3	7
1	1	0	0	3	0	5	4	3	5
1	1	0	9	3	0	6	4	3	2
0	1	0	0	0	0	6	4	3	3
1	1	0	0	0	0	6	4	3	3
1	1	0	11	0	0	7	4	3	2
1	1	0	0	1	0	7	4	3	3
1	1	0	0	1	0	7	4	3	3
1	1	0	0	1	0	7	4	3	2
1	1	0	6	0	0	7	4	3	5
1	1	0	0	1	0	7	4	3	2
1	1	0	4	1	0	7	4	3	2
0	1	0	0	1	0	7	4	3	2
1	1	1	7	1	0	8	4	3	2
1	1	0	0	0	0	8	4	3	3
1	1	0	0	1	0	8	4	3	3
1	1	0	0	0	0	8	4	3	2
0	1	0	0	0	0	8	4	3	3
1	1	0	0	1	0	12	4	3	2
0	1	0	1	0	0	12	4	3	2
1	1	0	0	1	0	12	4	3	2

I.2 Python files (Main Pages)

A. Business Understanding

1. Business Background

YT Star is a consulting firm owned by 4 people, who are Ryan Kho, Starla, Cheng How and Zee Lin that is based in Brazil. It currently has 20 employees and is hired by the Brazilian government. Their mission is to investigate car accidents in Brazil. Car accidents are the major cause of unnatural deaths in the world. The Brazilian government works hard to raise awareness about the rules and regulations that must be followed when driving a vehicle on the road in order to reduce fatalities. According to a report by 360° CI agency, there were around 1.51 million road crashes in the period from 2009 to 2019, resulting in a total of 79,085 reported road deaths. The data included in the report was provided by Brazil's Federal Highway Police. In the period between 2009 and 2019, Brazil's road death rate was reduced by 26%, falling short of the target set by the World Health Organisation of a 50% drop during this time. There were as many as 5,300 road deaths in Brazil during 2020. Apart from deaths, car accidents can also bring down a country's economy in several ways, such as property damage, workplace productivity loss, traffic congestion, medical costs etc.

To reduce the number of unnecessary fatalities on the road and the economic impact of car accidents on Brazil, YT Star wil be using a dataset from the Detran (Department of Transit) database that records all the car accidents that occurred in Brazil in the year 2020. By using this dataset, YT Star will develop an analytics application that predicts the severity of car accidents and identify ways to reduce the severity of car accidents.

2. Objectives & Goals

1. To develop an analytics application that predicts the severity of car accidents, especially fatal accidents
2. To identify the factors that lead to car accidents based on the dataset
3. To support the Brazilian government's plan to make the roads a safer place for people to drive on
4. To reduce the number of fatalities on the road and Brazil's economic burden
5. To achieve at least 80% for Accuracy, Macro Average Precision, Macro Average Recall and Macro Average F-1 Score for Analytics application

3. Inventory of Resources

- Personnel: Dr Noor Aida Binti Husaini, PHD
- Data: Car Accident 2020 dataset
- Computing resources: 4 Laptops
- Software: Jupyter Notebook, Google Colaboratory

4. Requirements, Assumptions and Constraints

- The deadline for this assignment is 6 May 2023
- We are required to produce good quality results that can be easily understood
- The Kaggle dataset is open-sourced, which means that it can be seen by the public. Therefore, we are free to use the data and there should not be any data security concerns and legal issues.
- We assume that most of the data recorded are correct and verified although some may be non-verifiable .
- We are only limited to 4 laptops and 2 kinds of software but we should be able to handle a decent size of data set for modelling.

5. Business Success Criteria

- Reduce the number of fatalities from car accidents by 40%
- Give useful insights into the relationship between the severity of car accidents with the important influential factors

B. Data Understanding

At this stage, we are required to analyse the dataset, which is used for mining. Apart from acquiring the dataset from the project resources, we will need to load the data. The dataset that we have used for this assignment is from Kaggle. It is in the form of a csv file and we work on the data by loading it into Jupyter. Data understanding is important as it can help us to avoid unexpected problems during the data preparation step. Below are the steps we have taken to understand the Car Accident 2020 dataset as much as possible.

B.1 Loading and viewing the data.

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import math
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('si_env-2020.csv', sep=';', encoding='ISO-8859-1')#this csv file uses semi-colon as separator. Need to use
                                                                   #the specified encoding to solve UnicodeDecodeError
df.head(50)
```

Out[2]:

Embreagues	Idade	nascimento	categoria_habilitacao	descricao_habilitacao	declaracao_obito	cod_severidade_antiga	especie_veiculo	pedestre	passageiro
NÃO	35	06/09/1984	B	VEICULOS MOTOR. COM ATE 8 PASSAGEIROS, PESO MA...	0	0	BICICLETA	N	N
SIM	42	11/03/1978	N	NAO INFORMADO ...	0	0	AUTOMOVEL	N	N
NÃO	32	20/11/1987		...	0	0	AUTOMOVEL	N	S
NÃO	72	29/06/1947		...	0	0	ONIBUS	N	S
NÃO	55	13/03/1965	N	NAO INFORMADO ...	0	0	ONIBUS	N	N
NÃO	29	16/06/1990	AE	HABILITADO NAS CATEGORIAS A e E ...	0	0	ONIBUS	N	N
NÃO	33	08/01/1987	AB	HABILITADO NAS CATEGORIAS A e B ...	0	0	AUTOMOVEL	N	N
NÃO	39	07/03/1981		...	0	0	AUTOMOVEL	N	S
NÃO	30	09/12/1989	C	VEIC. MOTOR. P/ TRANSP. DE CARGA, C/ PESO MAX....	0	0	AUTOMOVEL	N	N
NÃO	37	12/06/1982	N	NAO INFORMADO ...	0	0	MOTOCICLETA	N	N
NÃO	40	18/01/1980		...	0	0	MOTOCICLETA	N	S

C. Data Preparation

In [172]:
```
forDP = forCC.copy()
```

In [173]:
```
forDP.head()
```

Out[173]:

	accidentRecordNum	reportedDateNTime	noPeopleInvolved	driverResponsibleForAccident	severityCode	severityDesc	sex	woreSeatBelt	driverUnderAlc
0	2020-014152383-001	2020-03-20 02:18:00	1	1	2	NAO FATAL	1	0	
1	2020-014152383-001	2020-03-20 02:18:00	2	1	1	SEM FERIMENTOS	0	1	
2	2020-014152383-001	2020-03-20 02:18:00	3	0	1	SEM FERIMENTOS	1	1	
3	2020-014158612-001	2020-03-20 05:39:00	1	0	2	NAO FATAL	0	1	
4	2020-014158612-001	2020-03-20 05:39:00	2	1	1	SEM FERIMENTOS	1	1	

5 rows × 23 columns

a) Remove Unnecessary Rows

a.1) Severity Description (severityDesc) & Severity Code (severityCode)

In [174]:
```
print(forDP['severityDesc'].value_counts())
```

```
NAO FATAL          12544
SEM FERIMENTOS      9338
NAO INFORMADO        648
FATAL                113
Name: severityDesc, dtype: int64
```

In [175]:
```
print(forDP['severityCode'].value_counts())
```

```
2    12544
1     9338
0      648
3      113
Name: severityCode, dtype: int64
```

We decide all the rows with the severity code 0 (NAO INFORMADO), which means "No information". This is because we feel that a prediction, which says "No information", is not useful because "No information" can be either of the three other classes. Therefore, we feel that a prediction with "No information" has no use because it cannot tell us anything.

In [176]:
```
forDP_filtered1 = forDP[forDP['severityCode'] == 0].index
forDP.drop(forDP_filtered1 , inplace=True)
```

In [177]:
```
print(forDP['severityCode'].value_counts())
```

```
2    12544
1     9338
```

D. Modelling

D.1 Modelling with the Original Imbalanced Data

Before we can perform any modelling, we must split the dataset into train, test and validation sets. We wrote the code below so that we can split matrices or arrays into test and train subsets.

60% of the records will be allocated to training, 20% will be allocated to validation and 20% will be allocated to testing.

In [338]:
```
from sklearn.model_selection import train_test_split
modelData = forDP.copy()
```

In [339]:
```
inputVar = modelData.drop(["severityCode"], axis=1)
outputVar = modelData["severityCode"]

trainFeatures, testvalidationFeatures, trainLabels, testvalidationLabels = train_test_split(inputVar, outputVar, test_size =

validationFeatures, testFeatures, validationLabels, testLabels = train_test_split(testvalidationFeatures, testvalidationLabel
```

In [340]:
```
print("Shape of trainFeatures: ", trainFeatures.shape)
print("Shape of trainLabels: ", trainLabels.shape)
print("Shape of validationFeatures: ", validationFeatures.shape)
print("Shape of validationLabels: ", validationLabels.shape)
print("Shape of testFeatures: ", testFeatures.shape)
print("Shape of testLabels: ", testLabels.shape)
```

```
Shape of trainFeatures:  (13197, 12)
Shape of trainLabels:  (13197,)
Shape of validationFeatures:  (4399, 12)
Shape of validationLabels:  (4399,)
Shape of testFeatures:  (4399, 12)
Shape of testLabels:  (4399,)
```

1) Random Forest (Ryan Kho Yuen Thian)

1.1) Modelling with Default Hyperparameters

In [256]:
```
# 1. choose model class
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score

# 2. instantiate model
randomF_model = RandomForestClassifier(random_state=42)

# 3. fit model to data, for training purpose, training dataset
randomF_model.fit(trainFeatures, trainLabels)

# 4. predict on new data, testing data
```

E. Evaluation

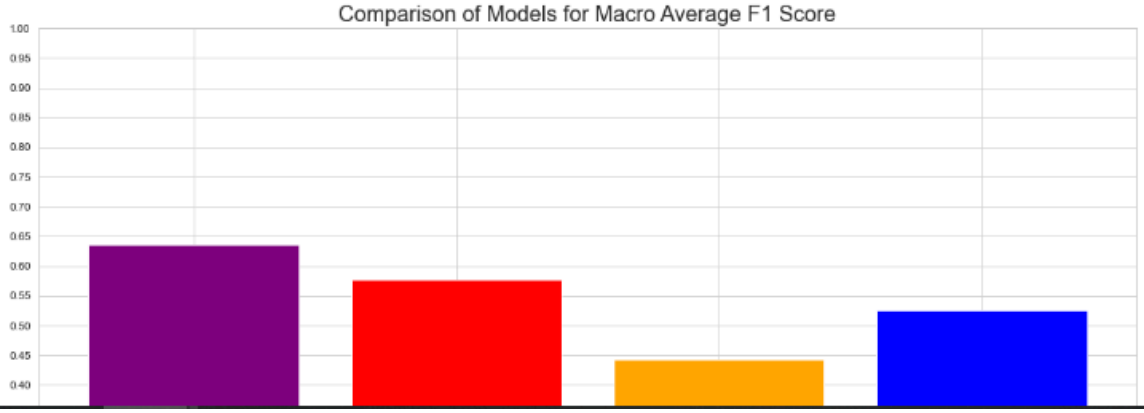
E.1) Before Upsampling of Fatal and No Injuries Cases

E.1.1) Macro Average F-1 Score of Each Model

In [281]:
```
from sklearn.metrics import f1_score
randomF_f1 = f1_score(validationLabels, randomF_predictedLabels, average='macro')
decisionT_f1 = f1_score(validationLabels, predval_dtc, average='macro')
logreg_f1 = f1_score(validationLabels, logreg_result, average='macro')
knn_f1 = f1_score(validationLabels, validationPred, average='macro')

format(knn_f1 * 100, ".2f")

print("F1 Score for Random Forest: ", format(randomF_f1 * 100, ".2f"), "%")
print("F1 Score for Decision Tree: ", format(decisionT_f1 * 100, ".2f"), "%")
print("F1 Score for Logistic Regression: ", format(logreg_f1 * 100, ".2f"), "%")
print("F1 Score for K Nearest Neighbour: ", format(knn_f1 * 100, ".2f"), "%")
```

```
F1 Score for Random Forest:  63.54 %
F1 Score for Decision Tree:  57.64 %
F1 Score for Logistic Regression:  44.27 %
F1 Score for K Nearest Neighbour:  52.60 %
```

In [282]:
```
ml_algorithms=("Random Forest","Decision Tree","Logistic Regression","K Nearest Neighbour")
f1_scores = (randomF_f1,decisionT_f1,logreg_f1,knn_f1)

colors = ("purple","red","orange","blue")
yPos = np.arange(1,5)

plt.figure(figsize=(18,10))
plt.bar(yPos,f1_scores,color=colors)
plt.xticks(yPos,ml_algorithms,fontsize=18)

#plt.set_xticklabels(ml_algorithms)

plt.yticks(np.arange(0.00, 1.01, step=0.05))
plt.grid()
plt.title("Comparison of Models for Macro Average F1 Score",fontsize=20)
plt.show()
```



F. Deployment

Below is the code that we will be asking the user to enter inputs via **Tkinter**

In [373]:

```
import tkinter as tk
from tkinter import messagebox
import pandas as pd
from tkinter import font
from tkinter import ttk

def predictSeverity(noPeopleInvolved,driverResponsibleForAccident,sex,woreSeatBelt,driverUnderAlcohol,qualificationCat,vehicl
    if(noPeopleInvolved == ''):
        noPeopleInvolved = 0
    elif(driverResponsibleForAccident == ''):
        driverResponsibleForAccident = 0
    elif(sex == ''):
        sex = 0
    elif(woreSeatBelt == ''):
        woreSeatBelt = 0
    elif(driverUnderAlcohol == ''):
        driverUnderAlcohol = 0
    elif(qualificationCat == ''):
        qualificationCat = 0
    elif(vehicleType == ''):
        vehicleType = 0
    elif(involvedPedestrian == ''):
        involvedPedestrian = 0
    elif(reportedHour == ''):
        reportedHour = 0
    elif(reportedDayOfWeek == ''):
        reportedDayOfWeek = 0
    elif(reportedMonth == ''):
        reportedMonth = 0
    elif(ageCategoryNum == ''):
        ageCategoryNum = 0

    predictCode = randomF_modelv2_def.predict([[noPeopleInvolved,driverResponsibleForAccident,sex,woreSeatBelt,driverUnderAl
    print(predictCode)
    if predictCode == [1]:
        severity = "No Injuries"
    elif predictCode == [2]:
        severity = "Not Fatal"
    elif predictCode == [3]:
        severity = "Fatal"
    return severity

def get_values():
    pplInv = int(pplinv.get())
    Drvres = int(drvres.get())
    #Sevcod = int(sevcod.get())
    Sex = int(sex.get())
    Woresb = int(woresb.get())
    Drvua = int(drvua.get())
    Quacat = int(quacat.get())
    Vehty = int(vehty.get())
    Invpe = int(invpe.get())
    Rephr = int(rephr.get())
    Repwe = int(repwe.get())
    Repmo = int(repmo.get())
    Ageca = int(ageca.get())
    print(f"The numeber of people involved in the car accident entered is: {pplInv}")
    print(f"Is the driver resposible for the car accident: {Drvres}")
```