ROAD ACCIDENT PREDICTION AND ANALYSIS

MAJOR PROJECT

Submitted by:

Aayushie Prasad (19104008)

Saransh Gupta (19104015)

Siddharth Khandelwal (19104058)

Under the supervision of

Dr Ankita Wadhwa (Supervisor)

Dr Neetu Sardana (Co. Supervisor)

Department of CSE/IT

Jaypee Institute of Information Technology, Noida

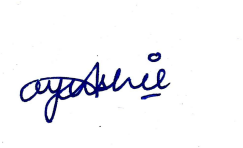
DECEMBER 2022

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and beliefs, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma from a university or other institute of higher learning, except where due acknowledgement has been made in the text.

Place: JIIT, Noida

Signature:



Date: 09 December 2022

Name: Aayushie Prasad

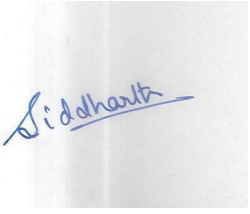
Enrolment No.: 19104008

Signature:

Name: Saransh Gupta

Enrolment No.:19104015

Signature:



Name: Siddharth Khandelwal

Enrolment No.:19104058

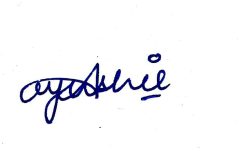
CERTIFICATE

This is to certify that the work titled “Road Accident Prediction and Analysis” submitted by Aayushie Prasad, Saransh Gupta, Siddharth Khandelwal of B.Tech of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of any other degree or diploma.

Signature of Supervisor
Name of Supervisor: Dr Ankita Wadhwa
Designation: Assistant Professor (Grade II)
Date: 09 December 2022

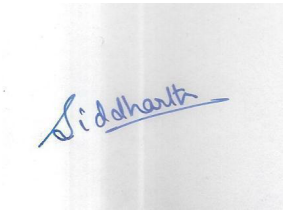
ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Dr Ankita Wadhwa, Assistant Professor (Grade II), Department of CSE/IT, Jaypee Institute of Information Technology, Noida for her generous guidance, continuous encouragement, help and valuable suggestions throughout the present work.



Aayushie Prasad (19104008)

Saransh Gupta (19104015)



Siddharth Khandelwal (19104058)

SUMMARY

Road safety is of paramount importance for the Government and the entire mobility ecosystem. While the number of road accidents has reduced over the last couple of years, still the overall number of accidents that occurred in the year 2021 is beyond alarming. It has been assessed that 90% of accident fatalities occur due to human error. A technological solution/platform which can rate driver behaviour and predict accident hot spots is the need of the hour. This will enable enforcement agencies and commuters to take preemptive actions to avoid accidents. Hence, through our project we aim to perform feature engineering after preprocessing our dataset, do unsupervised and then supervised learning using three different classification algorithms. Finally, we will do a comparative analysis to give results and precautionary measures.

We have done literature surveys on related topics and got information about prior work done in this field and also what future work can be done, or how existing systems can be improved.

The dataset used in our project is of Bangalore city. These days vehicles use a Collision Avoidance System which alerts the drivers of a possible collision beforehand. This dataset contains latitude, longitude, ward name (area where the alert was given), type of alert, date and time of alert. Second dataset is of Bangalore weather to analyse what were the weather conditions during the particular alert before the circumstance of an accident. It consists of weather conditions like temperature, humidity, wind speed etc.
We have done data cleaning and preprocessing for feature engineering so as to improve the accuracy of our model. Machine learning (ML) is used to analyse various algorithms through experience and improve results. It includes three major types of learning techniques, namely supervised and unsupervised learning.

In unsupervised learning, we have done K-means clustering to analyse whether the probability of an accident happening based on the aforementioned conditions are high or low.

Next, we have created a class/label and proceeded with supervised learning classification algorithms like SVM (Support Vector Machine), Decision Tree, Naive Bayes. The ratio of training and testing dataset is set as 70 % and 30 % respectively.

After applying the algorithms, we also calculated their accuracy which came out to be 99.6% for Decision Tree, 99.53% for SVM and Naive Bayes as 93.56%.

Since Decision Tree has the best accuracy, hence we have analysed our result from it that the major attributes that affect the probability of an accident occurring are Area, weather condition, alarm type, and time. It is usually observed that in the places where there exist severe potholes, accidents are bound to take place.

Hence, we have successfully done our feature analysis to predict the probability of accidents based on different outer environmental conditions.

List of Figures

List of Tables

Table of Contents

CHAPTER - 1

INTRODUCTION

1.1 General Introduction

Road safety is of paramount importance for the Government and the entire mobility ecosystem. While the number of road accidents has reduced over the last couple of years, still the overall number of accidents that occurred in the year 2021 is beyond alarming. It has been assessed that 90% of accident fatalities occur due to human error. “Ministry of Road and Transport, Govt. of India” envisages reducing road accidents & deaths by half by the end of 2024. There are a number of interventions being adopted to reduce the occurrence of road accidents. A technological solution/platform which can rate driver behaviour and predict accident hot spots is the need of the hour. This will enable enforcement agencies and commuters to take preemptive actions to avoid accidents.

Traffic on roads of Bangalore is not among the best in cities of India, and a recent study by Ola Cabs has confirmed the same - the average speed of vehicles at peak hours is approx 15.5 KM/hr, which is 3rd from the bottom ranking among Indian cities. But there are pockets where traffic moves at high speed as well, parts of the city where the number of accidents or potential accidents is high, at the same time at other places it is pretty low. Through the EDA of the Bangalore traffic dataset, we try to unravel some interesting observations about the roads of Bangalore. The types of alarms captured by CDS or CAS in the dataset used are as follows:

- **FORWARD COLLISION WARNINGS (FCW)**
  A FCW alerts drivers of an imminent rear-end collision with a car, truck, or motorcycle.
- **URBAN FORWARD COLLISION WARNINGS (UFCW)**
  UFCW provides an alert before a possible low-speed collision with the vehicle in front, thus assisting the driver at a low speed in densely heavy traffic.
- **HEADWAY MONITORING WARNING (HMW)**
  The headway monitoring warning (HMW) helps drivers maintain a safe following distance from the vehicle ahead of them by providing visual and audible alerts if the distance becomes unsafe.
- **LANE DEPARTURE WARNINGS (LDW)**
  The LDW provides an alert when the vehicle unintentionally departs from the driving lane without using the turn signals. If the turn signals are used when changing lanes, an alert is not generated.
  PEDESTRIANS AND CYCLIST DETECTION AND COLLISION WARNING (PCW)
  The PCW notifies the driver of a pedestrian or cyclist in the danger zone and alerts drivers of an imminent collision with a pedestrian or cyclist.
- **OVERSPEEDING**
  Detects and classifies various visible speed limit signs and provides visual indication when the vehicle’s speed exceeds the posted speed limit.

1.2 Problem Statement

There are several problems with current practices for prevention of the accidents occurring in the localities. The database we will use is available officially from many institutes and government websites. The data collected will be analysed, integrated and grouped together based on different constraints using the best-suited algorithm. This estimation will be helpful to analyse and identify the flaw and the reasons for the accidents. It will also be helpful while making roads and bridges as a reference to avoid the same problems faced before. The predictions made will be very much useful to plan the management of such problems.

1.3 Significance of The Problem

In the present world, there is a lot of data that is available on government websites and through other sources. We can use geographic data and combine it with accident data to analyse the reasons behind the huge amount of accidents that are happening in and around the world. This is for the benefit of the human race itself. By analysing the reasons a person can be aware of the causes behind the occurrence of accidents and be careful about them. Also, the authorities and government can study the analysis and work towards a future where there is less chance of accidents occurring.

1.4 Empirical Study

India is a country with a really high population which implies that manpower is really found in abundance here. In order to grow more and grow quickly, the government has started initiatives towards creating smart cities and people are running all across the country. An increase in movement results in an increase in traffic and also the dangers of accidents. Accidents depend on a lot of factors like the weather, condition of the road, speed of the vehicle and sometimes even the time of the day. Roads, nowadays, are one of the most dangerous places to be with all the dangers of accidents hovering over people all around the clock. As per recent surveys, road accidents are one of the major reasons behind the deaths of people all over the world. People and authorities are not all completely aware of the factors that lead to these accidents. Everyone wishes to prevent these but a proper study is needed to understand the reasons behind them and prevent them.

1.5 Brief Description of the Solution Approach

We are going to study the road accident alerts dataset of Bangalore and combine it with the area that the alert was received from, the time of the alert, the weather conditions of the place at that point in time, the condition of the potholes in that area, accident severity and based on all that is going to determine the probability of an accident occurring at that place at that particular point of time.
This data can be used by road authorities and the public to keep themselves safe and also to solve the issues behind the huge number of road accidents.

CHAPTER - 2

LITERATURE SURVEY

2.1 Summary of Papers Studied

Paper 1: Daniel Santos, José Saias, Paulo Quaresma and Vítor Beires Nogueira, “Machine Learning Approaches to Traffic Accident Analysis and Hotspot Prediction”, November 2021

- Objective -
  They aim to figure out what factors increase the likelihood of accidents and the severity of those accidents, develop predictive models for both the number and severity of accidents and test a predictive model to predict the likelihood of accidents on to specific road segments.

- Techniques -
  1. Rule-Based Model
  This presents the proposed approach to finding the most influential factors for accident severity and representing those factors in rule sets. The four stages of the proposed work are namely, data processing, clustering, feature selection, and rule generation.
  2. The clustering algorithm takes as input the geographical coordinates of the accidents and adds its output as the input of the predictive model. Moreover, data inputs of the predictive model also contain weather, road, and time information.
  This work can be divided into data processing, clustering, predictive model training, and prediction.

- Evaluation Metrics -
  For approach 1, the Silhouette Index was used as an evaluation measure to compare the performance of the agglomerative hierarchical clustering and the k-means algorithms. The silhouette index ranges from [-1, 1], with -1 indicating poor consistency within clusters and 1 indicating excellent consistency within clusters.
  2. Not mentioned for approach 2.

- Outcome
  The rule generation model's results successfully found patterns for fatal and non-fatal traffic accidents. According to their model, pedestrian accidents and accidents involving motorcycles are the main factors that have a higher chance of resulting in victims. Hit-and-run incidents are less likely to result in a victim. Then, in the research paper, they focused on the random forest algorithm.
  The model proved to be quite conservative with a false positive rate (FPR) of 3% & an accuracy of 73%; further development of this approach is required to improve these results.

- Critical analysis/gaps
  For future work, more recent data (2020/2021) needs to be provided to them, which will allow them to improve the proposed work. Since they have highlighted motorcycle

accidents as the main factor influencing accident severity, it would be interesting to include traffic parameters and intensity in their approaches.
When comparing these results to the literature, it is important to note that, as most examples belong to the negative class, the model that contains the higher negative/positive sample ratio is usually the one with the highest accuracy. Considering this, their work achieved an excellent FPR when compared to other works, while sensitivity was still not in the acceptable range.

Paper 2: G. Mani Kandan, Sarilya Jaiswal, Rahul Mishra & Mrs Steffina Muthukumar, "Analysis and Prediction of Road Accidents by Graphical Visualiser using ML", March 2019

- Objective -
  To analyse spatial and temporal factors that are mainly responsible for the cause of an accident and to determine likely road crash conditions.

- Techniques -
  The raw dataset is preprocessed. Then data is analysed and represented using a graph visualiser. The same preprocessed dataset is trained(75%) and tested(25%). These datasets are inputs of several model-fitting algorithms, namely, random forests, artificial neural networks & support vector machines. Then accuracy is predicted and actual comparison is done with predicted values.

- Evaluation Metrics -
  Accuracy validation is done using the k-fold validation technique where accuracy is calculated k times in the training set and mean values are compared against the test set accuracy.

- Outcome -
  No concrete result was mentioned.

- Critical Analysis/gaps -
  Only models are proposed, and no actual work is shown on a dataset.

Paper 3: Mamoudou Sangare, Sharut Gupta, Samia Bouzefrane, Soumya Banerjee, Paul Muhlethaler, "Exploring the forecasting approach for road accidents: Analytical measures with hybrid machine learning.", April 2021.

- Objective -
  This paper proposes a novel framework that combines the descriptive strength of the Gaussian Mixture Model with the high-performance classification capabilities of the Support Vector Classifier. A new approach is presented that uses the mean vectors obtained from the GMM model as input to the SVC.

- Techniques -
  This section presents the models used in this study for traffic accident forecasting. As mentioned above, the accident data including vehicle, casualty and drivers' features are collected from data.govt.uk.
  The algorithm description for the hybrid model includes steps

1. Data preprocessing
2. Data re-sampling
3. Feature selection
4. GMM
5. GMM & traffic prediction
6. SVC
7. Multi-class SVC

- Evaluation Metrics -
  4 performance metrics given below are evaluated from the confusion metric:
  • True Positives (TP) – These are the examples with 'yes' as their actual and class predicted by the model.
  • True Negatives (TN) – These are the examples with 'no' as their actual class & predicted by the model.
  • False Positives (FP) – These are the examples with 'no' as their actual class but are predicted as 'yes' by the model.
  • False Negatives (FN) – These are the examples with 'yes' as their actual class but are predicted as 'no' by the model.
  The performance estimation parameters are defined as
  • Accuracy (A): This is defined as the ratio of the number of correctly predicted examples over the total number of e.g.s. (TP + TN) / (TP + FP + FN + TN)
  • Precision (P): This is defined as the ratio of correctly predicted positive observations to the total predicted positive observations.

- Outcome -

Data re-sampling results.

Accident severity class	Training samples before SMOTE	Training samples after SMOTE
Class 1	2,044	93,170
Class 2	21,098	93,756
Class 3	93,321	93,037

Fig. 2.1.1 Research Paper Sampling

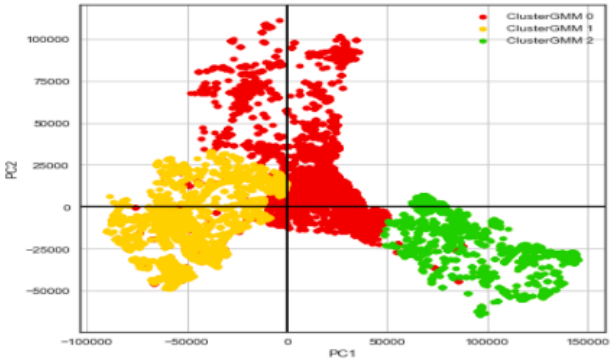


Fig. 2.1.2 Research Paper Outcome

- Critical Analysis/gaps -
  The first concerns the dataset used. This research is based on a road traffic accident dataset from the year 2017 which contains very few data samples for the no-injury and non-incapacitating injury types of accident.
  The second limitation concerns the dependence of the SVC model on parameters and attribute selection. In this study, the performance of SVC relies heavily on the feature selection results and the mean vectors obtained from the GMM.

Paper 4: Vivian Brian Lobo, Jayesh Patil, Vaibhav Patil, Dhaval Walavalkar, "Road Accident Analysis and Hotspot Prediction using Clustering.", 2021.

- Objective -
  This study aims to develop an application connected to maps that gives a user a clear idea about accident-prone areas and passing an alert about road conditions and suggesting certain ways to be safe.

- Techniques -
  This study focuses on k-means clustering, which is a part of an unsupervised ML technique. It helps in finding all kinds of unknown patterns in datasets. Data training is performed using unlabeled data. Unsupervised learning helps in performing complex processing tasks as compared to supervised learning. Their study takes various factors like age, gender, atmospheric conditions, condition of vehicles, roads, and mental state of a driver into consideration, and they apply k-means clustering to every available dataset to gain maximum accuracy. Their model offers a perfect prediction for various locations in India so that accurate reason for an accident can be predicated and then the cause can be rapidly eliminated to avoid future mishaps.
  The proposed methodology for achieving this are data gathering, data preprocessing, data training and analysis, testing and output.

- Evaluation Metrics -
  No validation method is mentioned.

- Outcome -
  Their analysis clearly identifies regions having high count of accidents and reasons behind it. The results clearly depict the reason and their causes. A figure shows all high accident areas in India, and the pin denotes the major reason behind this. Moreover, their graphical user interface will clearly warn about such zones while driving making it much safer than the current situation.

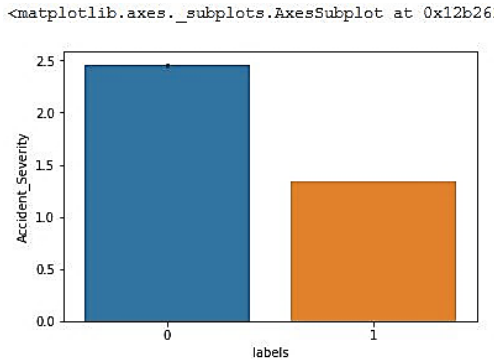


Fig. 2.2.1 Research Paper Outcome 1

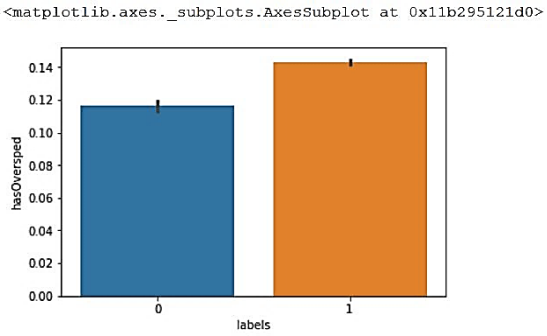


Fig. 2.2.2 Research Paper Outcome 2

- Critical Analysis/gaps -
  No validation or evaluation metric used. No future work mentioned.

2.2 Integrated Summary of the Literature Studied

sno	Paper ID	Objective	Techniques	Evaluation Metrics	Outcome	Critical Analysis/Gaps
1.	Machine Learning Approaches to Traffic Accident Analysis and Hotspot Prediction Computers 2021, 10, 157	The aim is to figure out what factors increase the likelihood of accidents and the severity of those accidents, develop predictive models for both the number and severity of accidents and test a predictive model to predict the likelihood of accidents on to specific road segments.	1. Rule-Based Model This section presents the proposed approach to finding the most influential factors for accident severity and representing those factors in rule sets. Figure 1 illustrates the four stages of the proposed work, namely, data processing, clustering, feature selection, and rule generation. 2. The clustering algorithm takes as input the geographical coordinates of the accidents and adds its output as the input of the predictive model. Moreover, data inputs of the predictive model also contain weather, road, and time information. This work can be divided into data processing, clustering, predictive model training, and prediction.	For approach 1, the Silhouette Index was used as an evaluation measure to compare the performance of the agglomerative hierarchical clustering and the k-means algorithms. The silhouette index ranges from [-1, 1], with -1 indicating poor consistency within clusters and 1 indicating excellent consistency within clusters. 2. Not clearly mentioned for approach 2.	The rule generation model's results successfully found patterns for fatal and non-fatal traffic accidents. According to the model, pedestrian accidents and accidents involving motorcycles are the main factors that have a higher chance of resulting in victims. Hit-and-run incidents are less likely to result in a victim. Then we focused on the random forest algorithm. The model proved to be quite conservative with a false positive rate (FPR) of 3% & an accuracy of 73%; further development of this approach is required to improve these results.	For future work, more recent data (2020/2021) will be provided to us that will allow us to improve the proposed work. Since we have highlighted motorcycle accidents as the main factor influencing accident severity it would be interesting to include traffic parameters and intensity in our approaches. When comparing these results to the literature, it is important to note that, as most examples belonging to the negative class, the model that contains the higher negative/positive sample ratio is usually the one with the highest accuracy. Considering this, our work achieved an excellent FPR when compared to other works, while sensitivity is still not in an acceptable range.

Table 2.2.1

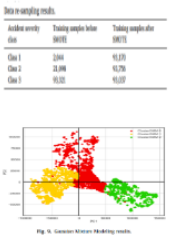
2.	Analysis and Prediction of Road Accidents by Graphical Visualiser using ML	To analyse spatial and temporal factors that are mainly responsible for the cause of an accident and to determine likely road crash conditions.	The raw dataset is preprocessed. Then data is analysed and represented using a graph visualiser. The same preprocessed dataset is trained(75%) and tested(25%). These datasets are inputs of several model-fitting algorithms, namely, random forest, artificial neural networks & support vector machines. Then accuracy is predicted and actual comparison is done with predicted values.	Accuracy validation is done using the k-fold validation technique where accuracy is calculated k times in the training set and mean values are compared against the test set accuracy,	No concrete result was mentioned.	Only models are proposed, and no actual work is shown on a dataset.
3.	Exploring the forecasting approach for road accidents: Analytical measures with hybrid machine learning.	This paper proposes a novel framework that combines the descriptive strength of the Gaussian Mixture Model with the high-performance classification capabilities of the Support Vector Classifier. A new approach is presented that uses the mean vectors obtained from the GMM model as input to the SVC.	This section presents the models used in this study for traffic accident forecasting. As mentioned above, the accident data including vehicle, casualty and drivers' features are collected from data.govt.uk. The algorithm description for the hybrid model includes steps 1. Data preprocessing 2. Data re-sampling 3. Feature selection 4. GMM 5. GMM & traffic prediction 6. SVC 7. Multi-class SVC	4 performance metrics given below are evaluated from the confusion metric: • True Positives (TP) – These are the examples with 'yes' as their actual and class predicted by the model. • True Negatives (TN) – These are the examples with 'no' as their actual class & predicted by the model. • False Positives (FP) – These are the examples with 'no' as their actual class but are predicted as 'yes' by the model. • False Negatives (FN) – These are the examples with 'yes' as their		The first concerns the dataset used. This research is based on a road traffic accident dataset from the year 2017 which contains very few data samples for the no-injury and non-incapacitating injury types of accident. The second limitation concerns the dependence of the SVC model on parameters and attribute selection. In this study, the performance of SVC relies heavily on the feature selection results and the mean vectors obtained from the GMM.

Table 2.2.2

				actual class but are predicted as 'no' by the model. The performance estimation parameters are defined as • Accuracy (A): This is defined as the ratio of the number of correctly predicted examples over the total number of e.g.s. (TP + TN)/ (TP + FP + FN + TN) • Precision (P): This is defined as the ratio of correctly predicted positive observations to the total predicted positive observations.		
4.	Road Accident Analysis and Hotspot Prediction using Clustering	This study aims to develop an application connected to maps that gives a user a clear idea about accident-prone areas and passing an alert about road conditions and suggesting certain ways to be safe.	The study takes various factors like age, gender, atmospheric conditions, condition of vehicles, roads, and mental state of a driver into consideration, and they apply k-means clustering to every available dataset to gain maximum accuracy. Their model offers a perfect prediction for various locations in India so that accurate reason for an accident can be predicated and then the cause can be rapidly eliminated to avoid future mishaps.	No metrics mentioned.	Their analysis clearly identifies regions having high count of accidents. The results clearly depict the reason and their causes. A figure shows all high accident areas in India and their graphical user interface will clearly warn about such zones while driving making it much safer than the current situation.	No future work given.

Table 2.2.3

CHAPTER - 3
REQUIREMENT ANALYSIS AND SOLUTION APPROACH

3.1 Overall Description of The Project

Road safety is of paramount importance for the Government and the entire mobility ecosystem. It has been assessed that 90% of accident fatalities occur due to human error. A technological solution/platform which can rate driver behaviour and predict accident hot spots is the need of the hour. This will enable enforcement agencies and commuters to take preemptive actions to avoid accidents. Hence, through our project we aim to perform feature engineering after preprocessing our dataset, do unsupervised and then supervised learning using three different classification algorithms. Finally, we will do a comparative analysis to give results.

3.2 Requirement Analysis

Operating System - Windows 8.1 or higher

Language Used - Python because it is currently the most important programming language for Machine Learning. Also, humans are able to interpret it easily which makes it easier to build models for machine learning.

Tools Used - Jupyter Notebook

Libraries Used:

- **Pandas -** It is used to import data and create python objects with rows and columns and can also be used to write data into the file.
- **NumPy -** It contains multi-dimensional arrays such as matrix data structures. It is used to perform statistical and algebraic mathematical operations on arrays.
- **Matplotlib** - It is a visualisation library. Used to create interactive graphs and charts
- **Seaborn -** It is again a data visualisation library based on matplotlib used for making a high-level interface for drawing statistical graphs
- **OS -** The functions the OS module provides allows us to operate on underlying Operating System tasks, irrespective of whether it is a Windows Platform, Macintosh or Linux.
- **Datetime -** Python's built-in DateTime library is one of the most common modules to manipulate date and time object data. You can create date and date-time objects, loop through a range of dates, parse and format date strings, and more
- **Sklearn** - Sklearn library comes loaded with a lot of features such as classification, regression, clustering and dimensionality reduction algorithms include k-means, K-nearest neighbors, Support Vector Machines (SVM), Decision Trees also supports python numerical and scientific libraries NumPy and SciPy
- **Time** - Use of time in Jupyter Notebook Timeit magic command in the Jupyter notebook is used to measure the time execution of small code. You don't need to import the time module from a standard library.

- **Urlib** - Urllib is a package that collects several modules for working with URLs: urllib. request for opening and reading URLs.
- **Re** - The re module provides a set of powerful regular expression facilities, which allows you to quickly check whether a given string matches a given pattern (using the match function), or contains such a pattern (using the search function).

3.3 Solution Approach

For the data analysis we'd be using the dataset available on Kaggle and applying various data analysis techniques and KNN algorithm we'll be finding some fruitful results and relationships between the attributes which can help us get some significant results.

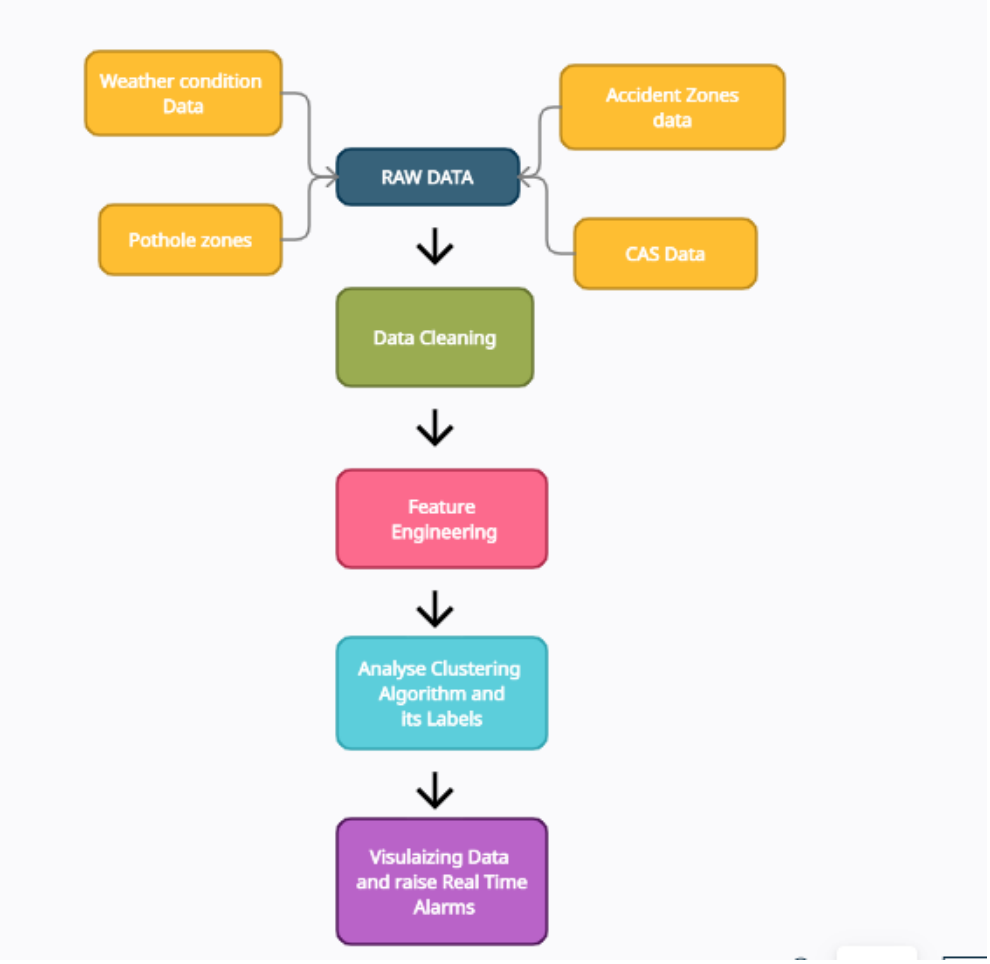
For prediction of the accidents we will be using K Means algorithm and web scraping techniques to find out all the information needed and then compiling it all together and forming a decision tree to predict the probability of an accident based on the data provided.

Python 3.7 will be used for the coding
Jupyter Notebook will be platform used

CHAPTER - 4
MODELLING AND IMPLEMENTATION DETAILS

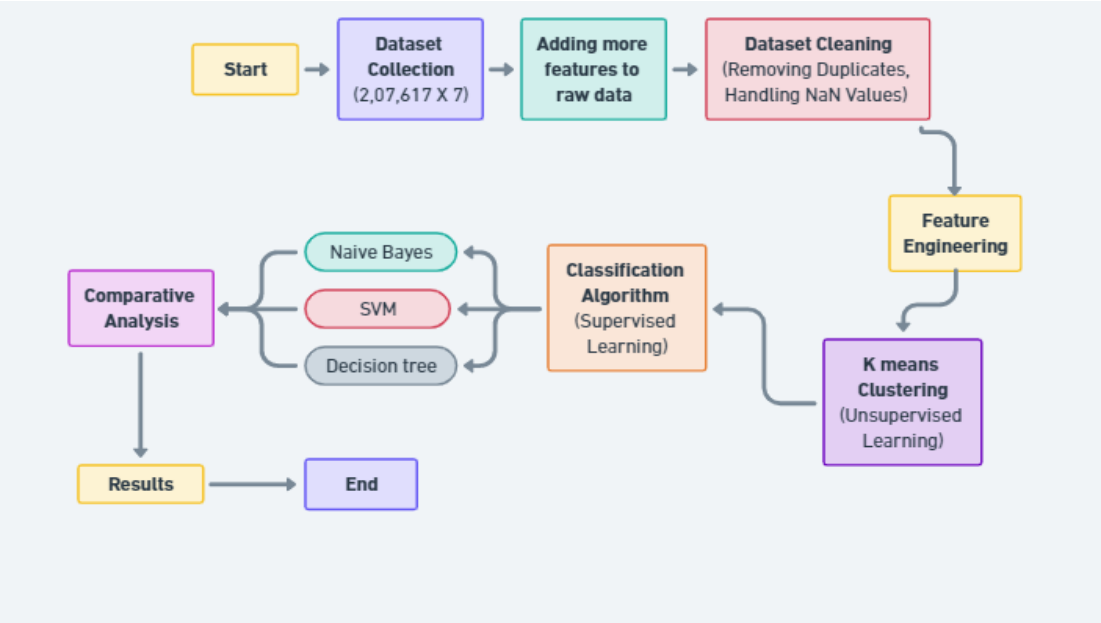
4.1 Design Diagrams

4.1.1 Class Diagrams/Control Flow Diagrams



4.1.1 Class Diagram

4.1.2 Sequence Diagrams/Activity Diagrams



4.1.2 Sequence Diagram

4.2 Implementation Details and Issues

DATASET USED

Specifications Table

Subject	Collision Avoidance System of Vehicles in Bengaluru
Specific subject area	Automobiles, CAS, Accidents
How data were acquired	It is a private dataset on Kaggle
Data format	Mixed (raw and pre-processed)
Parameters for data collection	The unit of analysis of the dataset is a vehicle. A full five-year period of data was collected (2013 to 2018). Time-related variables were accounted for based on the last day of the extraction period. The last day of the extraction period is December 31, 2018.
Description of data collection	Data was extracted via T SQL queries executed in the production server, using Microsoft SQL Studio Manager. Python was employed to perform summary statistics.
Data source location	The data came from Bangalore traffic and road accident department
Data accessibility	Data is available from https://www.kaggle.com/code/supratimhaldar/bangalore-road-accidents-basic-data-analysis/data

Table 4.2.1

Data Description

(i) bangalore-cas-alerts.csv

Variable	Type	Description
deviceCode_deviceCode	Numeric	Alert Device Code
deviceCode_location_latitude	Numeric	Latitude of the location of alert device
deviceCode_location_longitude	Numeric	Longitude of the location of alert device
deviceCode_location_wardName	Categorical	Name of the ward from where the alert is received
deviceCode_pyld_alarmType	Categorical	The category of the alarm from the alert
deviceCode_pyld_speed	Numeric	Speed of the vehicle
deviceCode_time_recordedTime_$date	Numeric	Time at which the alert is received

Table 4.2.2

(ii) bangalore-weather.xlsx

Variable	Type	Description
weatherDate	Numeric	Date in yyyymmdd format
time	Time	Time for the weather
temperature	Numeric	Temperature at that specific time
visibility	Numeric	Measure of the visibility
condition	Categorical	The kind of weather it is at that point of time on that date

Table 4.2.3

(iii)bangalore-accidents-zone.xlsx

Variable	Type	Description
Area	Categorical	Area of the accident
Mapped_Location	Categorical	Ward of the area
Accident_Severity	Categorical	Rated according to the severity of the accident
Pothole_Severity	Categorical	Rated according to severity of the pothole

Table 4.2.4

ANALYSES

```
In [2]: train_data = pd.read_csv("bangalore-cas-alerts.csv")
        df = pd.read_csv("bangalore-cas-alerts.csv")

In [3]: train_data.shape
Out[3]: (207617, 7)

In [4]: df.dropna(inplace=True)

In [5]: train_data.drop_duplicates(inplace=True)
        train_data.shape
Out[5]: (152276, 7)
```

Fig. 4.2.1

A glance at the data clearly shows duplicate rows present in the data. So, we remove duplicates from the data.

```
In [6]: train_data.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 152276 entries, 0 to 207616
Data columns (total 7 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   deviceCode_deviceCode                 152276 non-null  int64
 1   deviceCode_location_latitude          152276 non-null  float64
 2   deviceCode_location_longitude         152276 non-null  float64
 3   deviceCode_location_wardName          152276 non-null  object
 4   deviceCode_pyld_alarmType             152276 non-null  object
 5   deviceCode_pyld_speed                 152276 non-null  int64
 6   deviceCode_time_recordedTime_$date    152276 non-null  object
dtypes: float64(2), int64(2), object(3)
memory usage: 9.3+ MB
```

Fig. 4.2.2

There are no missing values

```
In [9]: df =df.rename(columns = {'deviceCode_deviceCode':'DeviceCode',
                'deviceCode_location_latitude':'Latitude',
                'deviceCode_location_longitude' : 'Longitude',
                'deviceCode_location_wardName':'WardName',
                'deviceCode_pyld_alarmType':'AlarmType',
                'deviceCode_pyld_speed':'Speed',
                'deviceCode_time_recordedTime_$date':'RecordedDateTime'})

In [10]: lat_max = train_data.latitude.max()
         lat_min = train_data.latitude.min()
         print("Range of latitude:", lat_max, lat_min)

         lon_max = train_data.longitude.max()
         lon_min = train_data.longitude.min()
         print("Range of longitude:", lon_max, lon_min)

         Range of latitude: 13.070075035095217 12.686662673950195
         Range of longitude: 77.80682373046875 77.5081787109375
```

Fig. 4.2.3

Finding the range of coordinates between which the data belongs

```
In [11]: fig, axes = plt.subplots(figsize=(9,12))
         axes.scatter(train_data.longitude, train_data.latitude, s=0.1, alpha=0.5, c='r')
         plt.show()
```

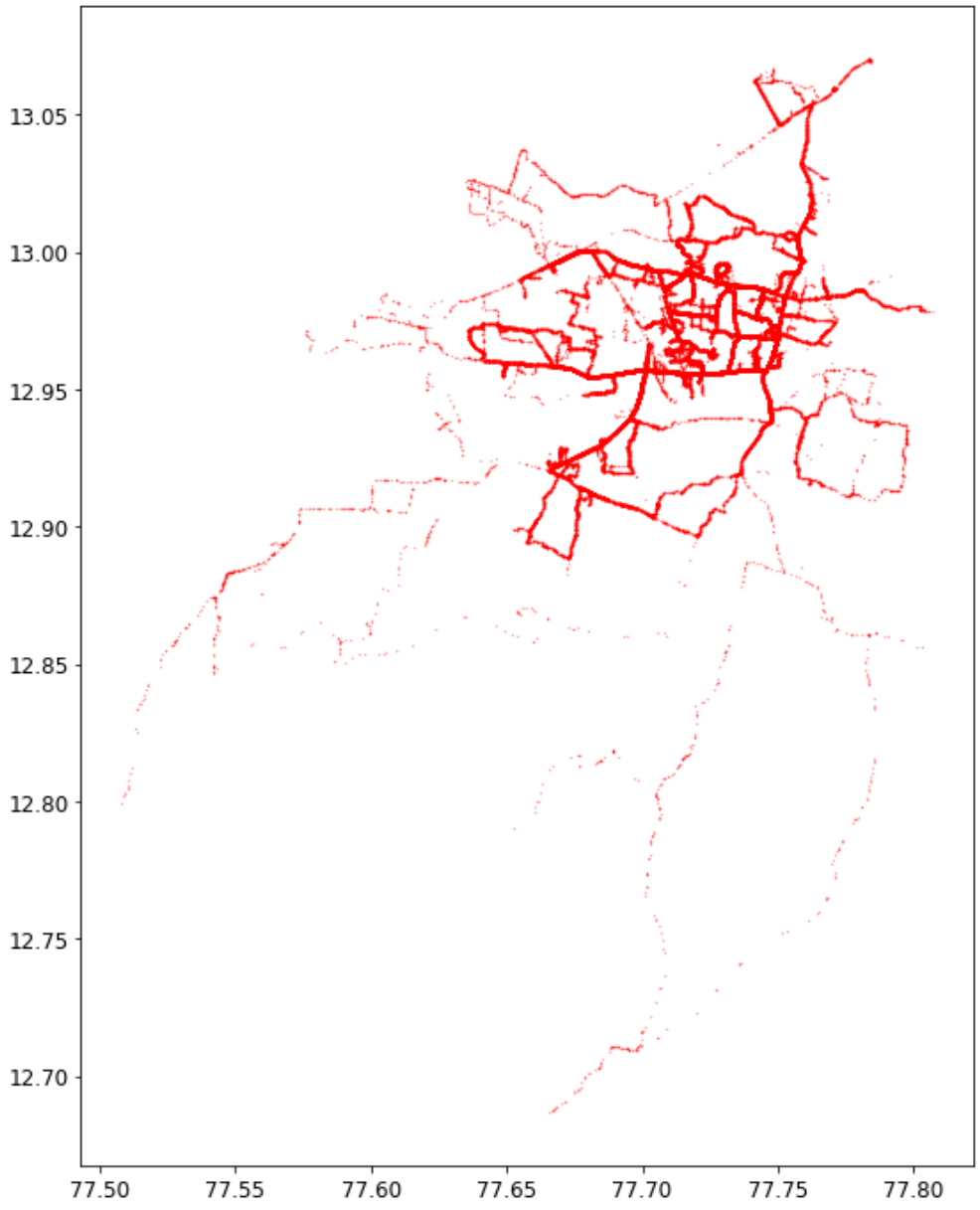


Fig 4.2.4

Simple plot of the coordinates on the road map of Bangalore.

```
In [12]: bangalore_map_img = 'https://lh3.googleusercontent.com/np8igtYRrHpe7rvJwMzVhbyUZC4Npgx5fRznofRoLVhP6zcdBW9tfD5bC4FbF2ITctElCtBrOr
         bangalore_map = plt.imread(bangalore_map_img)
         cmap = plt.get_cmap("jet")

         axes = train_data.plot(figsize=(15,20), kind='scatter',
                                x='longitude', y='latitude',
                                alpha=0.5, marker="o", s=train_data["speed"]*2,
                                c=train_data["speed"], cmap=cmap,
                                colorbar=False)

         epsilon = 0.01
         bound_box = [lon_min + epsilon, lon_max + epsilon,
                      lat_min + epsilon, lat_max + epsilon]
         im = plt.imshow(bangalore_map, extent=bound_box, zorder=0,
                    cmap=cmap, interpolation='nearest', alpha=0.7)

         axes.set_ylabel("Latitude")
         axes.set_xlabel("Longitude")
         axes.set_title('Accident Heatmap of city of Bangalore')

         # Colorbar
         speed = train_data["speed"].values
         tick_values = np.linspace(speed.min(), speed.max(), num=6, dtype=np.int64)

         cbar = plt.colorbar(im, fraction=0.05, pad=0.02)
         cbar.set_label("Speed (km / hour)")
         cbar.ax.set_yticklabels(["%d"%(val) for val in tick_values])

         plt.tight_layout()

         #output_image = os.path.join(input_data_dir, "output_bangalore_map_traffic")
         #plt.savefig(output_image + ".png", format='png', dpi=300)

         plt.show()
```

Fig. 4.2.5

Before we explore each attribute in detail, let us plot all CDS alarm data on a map of Bangalore by the coordinates specified, to generate a heatmap of locations and speed of vehicles at the time of alarm generation.

On this heatmap, magnitude of speed is represented by colour temperatures - cooler (bluish) plots indicate low speeds whereas warmer (reddish) colours represent higher speeds.

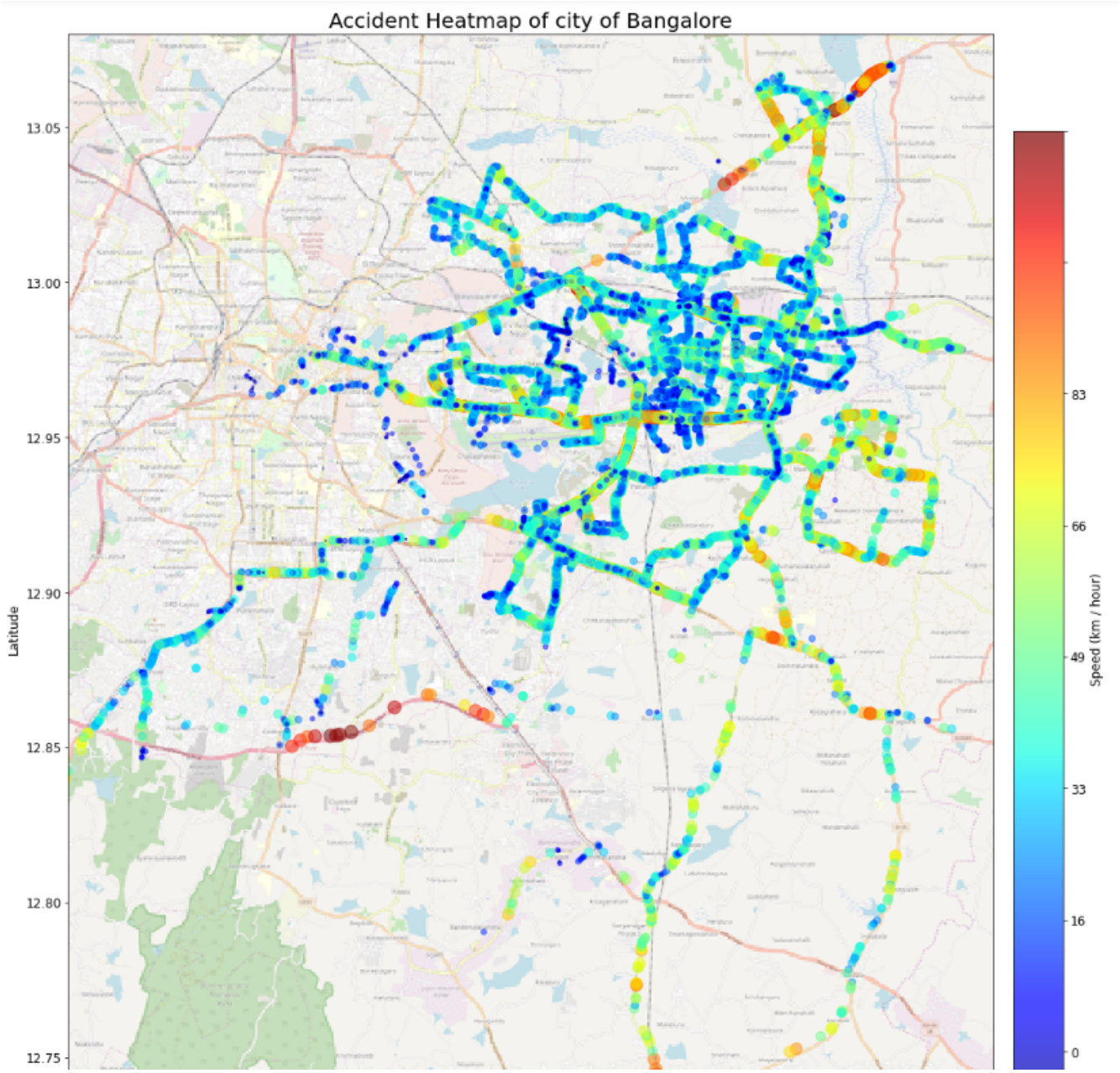

Accident Heatmap of city of Bangalore
Latitude
13.05
13.00
12.95
12.90
12.85
12.80
12.75
Speed (km / hour)
83
66
49
33
16
0

Fig. 4.2.6

```
In [15]: fig, axes = plt.subplots(figsize=(12,6))
data = train_data['wardName'].value_counts(normalize=True).sort_values(ascending=False)
data = data.head(10)
axes.bar(data.index, data*100, color={"red","blue","green","orange","brown","black","pink","purple","violet","magenta"})
axes.set_ylabel("% of CDS alarms")
axes.set_xticklabels(data.index, rotation=90)
axes.set_title("% of incidents in each ward")
plt.show()
```

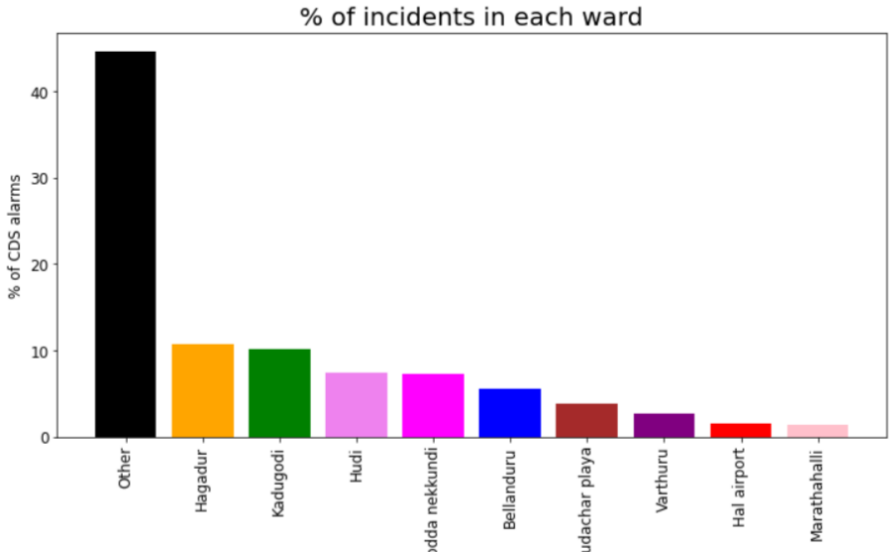


Fig. 4.2.7

It is found that the majority (~45%) incidents are captured against the ward 'Other'. Data munging is required to clean this data and fill in the proper ward name.

```
In [21]: # Apply KNN classification
# Input: Lat, Lon, Output: Ward No (since this is numeric variable)
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier(n_neighbors=1, weights='distance')
knn_clf.fit(bbmp_data[['LAT', 'LON']], bbmp_data['WARD_NO'])

# Estimate ward no in train_data from learnt model of bbmp_data
train_data['estimatedWardNo'] = knn_clf.predict(train_data[['latitude', 'longitude']])
# Estimate ward name in train_data from the ward no - ward name mapping dictionary
train_data['estimatedWardName'] = train_data['estimatedWardNo'].map(ward_dict)
```

```
In [22]: # Check accuracy
validation_orig = train_data['wardName'][~train_data['wardName'].str.contains('Other')]
validation_estimated = train_data['estimatedWardName'][~train_data['wardName'].str.contains('Other')]
accuracy = np.mean(validation_orig == validation_estimated)
print("Out of ward names which are not 'Other', % of accurate predictions =", accuracy*100)
```

```
Out of ward names which are not 'Other', % of accurate predictions = 62.588597842835135
```

```
In [23]: fig, [axes1, axes2] = plt.subplots(1, 2, figsize=(15,6))
data = train_data['estimatedWardName'].value_counts(normalize=True).sort_values(ascending=False)
data1 = data.head(10)

axes1.bar(data1.index, data1*100, color={"red","blue","green","orange","brown","black","pink","purple","violet","magenta"})
axes1.set_ylabel("% of CDS alarms")
axes1.set_xticklabels(data1.index, rotation=90)
axes1.set_title("Most dangerous wards")

data2 = data.tail(10)
axes2.bar(data2.index, data2*100, color={"red","blue","green","orange","brown","black","pink","purple","violet","magenta"})
axes2.set_ylabel("% of CDS alarms")
axes2.set_xticklabels(data2.index, rotation=90)
axes2.set_title("Most safe wards")
```

Fig. 4.2.8

Even though all the estimated ward names do not match with original ward names in the dataset, a close examination of multiple rows indicate that the estimated wards are very close (geographically) and are differing for lat/lon on the ward borders. Hence, we'll use the estimated ward names for further analysis.

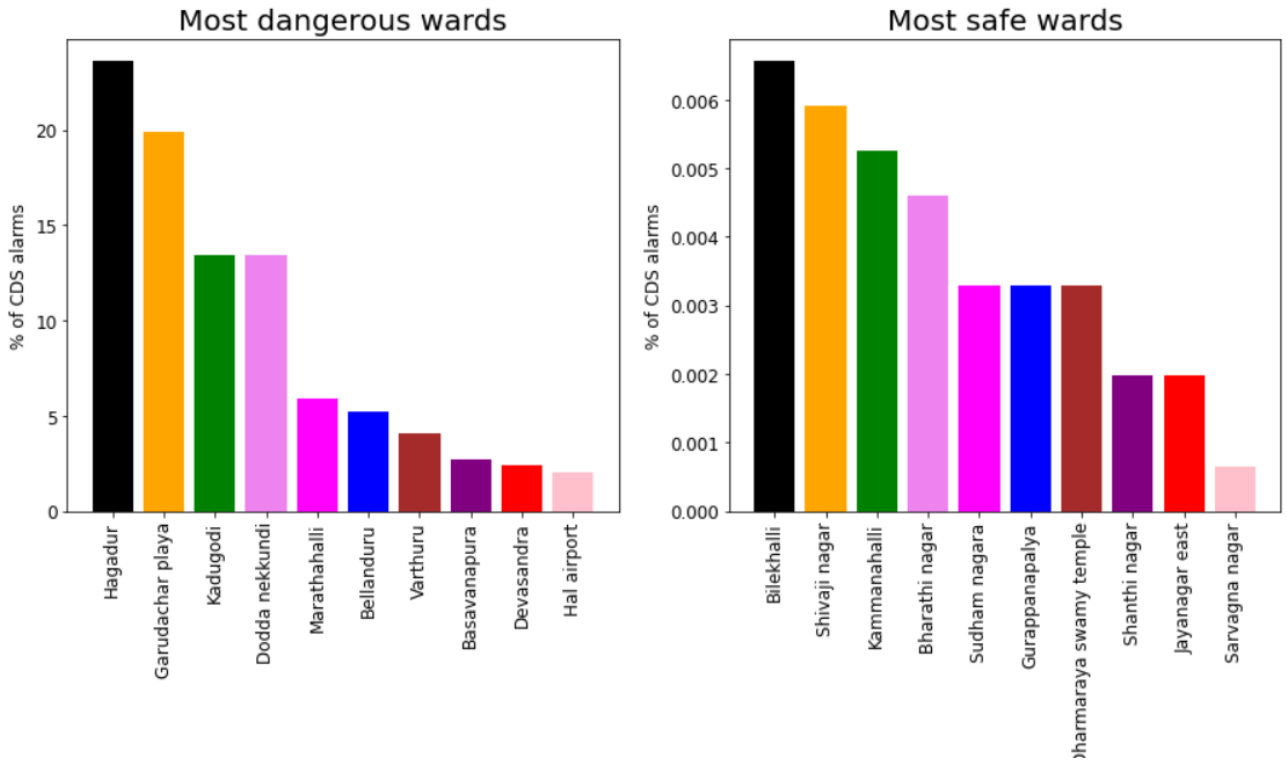


Fig. 4.2.9

```
In [24]: fig, axes = plt.subplots(figsize=(12,6))
data = train_data['alarmType'].value_counts(normalize=True)
axes.bar(data.index, data*100, color={"red","blue","green","orange","brown","black","pink","purple","violet","magenta"})
axes.set_title('% of Alarm Types')
axes.set_xlabel('')
axes.set_ylabel('%')
plt.show()
```

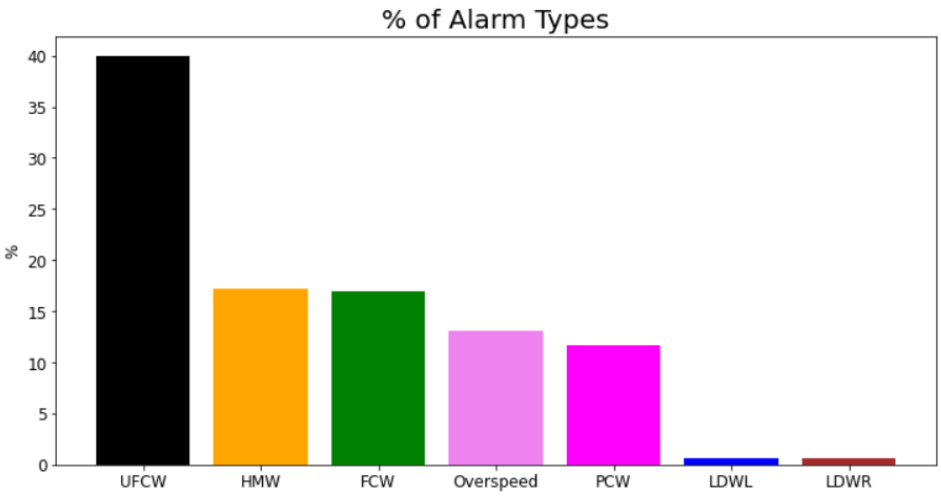


Fig. 4.2.10

Interesting observation: Majority of the alarms are of type UFCW, which are generated for potential collision at low speed. This implies that speed is not the most important reason for accidents on Bangalore roads. At the same time, this corroborates the recent finding that average speed of traffic is low in Bangalore roads.

Since HMW and FCW provide alarm on imminent rear-end collisions and speeds more than 30 kmph, both can be classified into a single category called High Speed Collision Warning (HSCW). Let's also rename UFCW to Low Speed Collision Warning (LSCW). Similarly, LDWL and LDWR can be brought under the umbrella of LDW. So, alarm types will be categorised in the following - HSCW, LSCW, PCW, LDW and Overspeed.

```
In [26]: fig, axes = plt.subplots(figsize=(12,6))
         data = train_data['alarmTypeCat'].value_counts(normalize=True)
         axes.bar(data.index, data*100, color={"red","blue","green","orange","brown","black","pink","purple","violet","magenta"})
         axes.set_title('% of Alarm Types (updated categorization)')
         axes.set_xlabel('')
         axes.set_ylabel('%')
         plt.show()
```

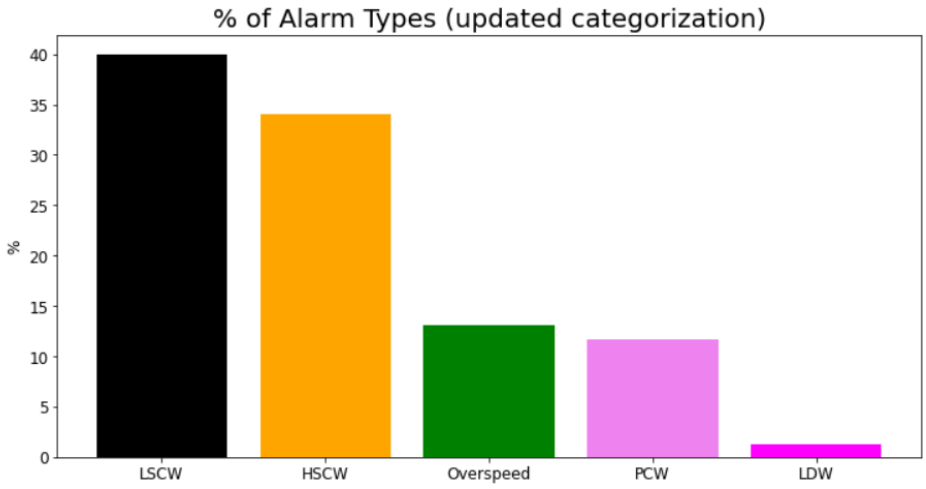


Fig. 4.2.11

The speed data that is available in the dataset is the speed recorded by buses at the time of alarm generation, and not overall speed of vehicles/buses on Bangalore roads. The highest speed that is recorded is 83 kmph, however the average speed is only 22 kmph.

```
In [27]: fig, axes = plt.subplots(figsize=(12,6))
         data = train_data['speed']
         #axes.hist(data, bins=15, color='green')
         sns.distplot(data, bins=15, color='green')
         axes.axvline(data.mean(), color='k', linestyle='dashed', linewidth=2)

         axes.set_xticks(np.arange(0, data.max()+5, 5))
         axes.set_xticklabels([str(val) for val in np.arange(0, data.max()+5, 5)])
         axes.set_xlim(0, data.max()+5)
         axes.set_xlabel('Speed in kmph')
         axes.set_title('Distribution of Speed')
         axes.grid(True)

         _ymin, _ymax = axes.get_ylim()
         axes.text(data.mean() + data.mean()/100,
                   (_ymax+_ymin)*0.5,
                   'Mean Speed: {:.2f} kmph'.format(data.mean()))

         plt.show()
```

Fig. 4.2.12

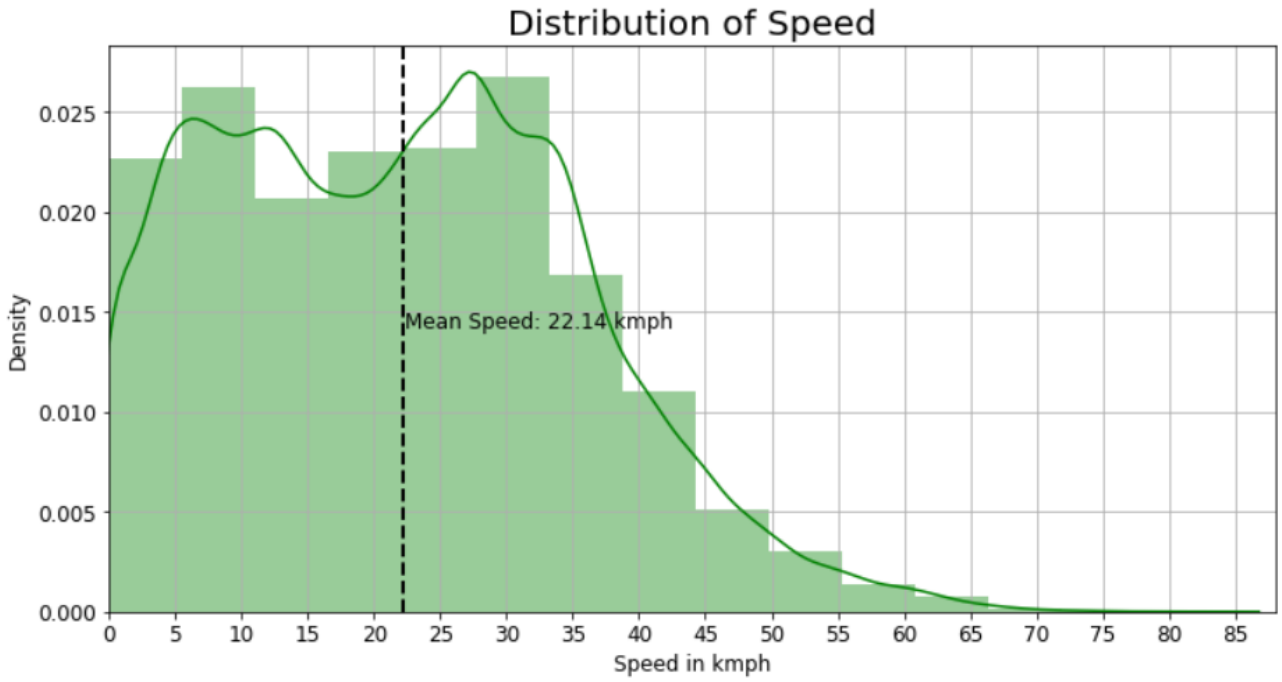


Fig. 4.2.13

The date and time in the dataset are recorded in UTC timezone, hence these should be converted to IST. Next, let's look at the distribution of data by years, months, days of week and month, and hours of day. Since the period over which the data was collected is very short, these diagrams will only provide a high level indication of the trends.

```
In [30]: fig, axes = plt.subplots(figsize=(15,6))
train_data["hour"] = train_data.recordedDateTime.dt.hour
data = train_data["hour"].value_counts(normalize=True).sort_index()

axes.bar(data.index, data, color='tan', width=0.5, zorder=0)
axes.plot(data.index, data, color='red', zorder=1)
axes.scatter(data.index, data, s=50, color='darkred', zorder=2)

axes.set_xlabel('Hour of Day')
axes.set_xticks(np.arange(1, 25))
axes.set_xticklabels([str(val) for val in np.arange(1, 25)])
axes.set_ylim(0, 0.3)
axes.set_title('% of alarms by Hours of Day')

plt.show()
```

Fig. 4.2.14

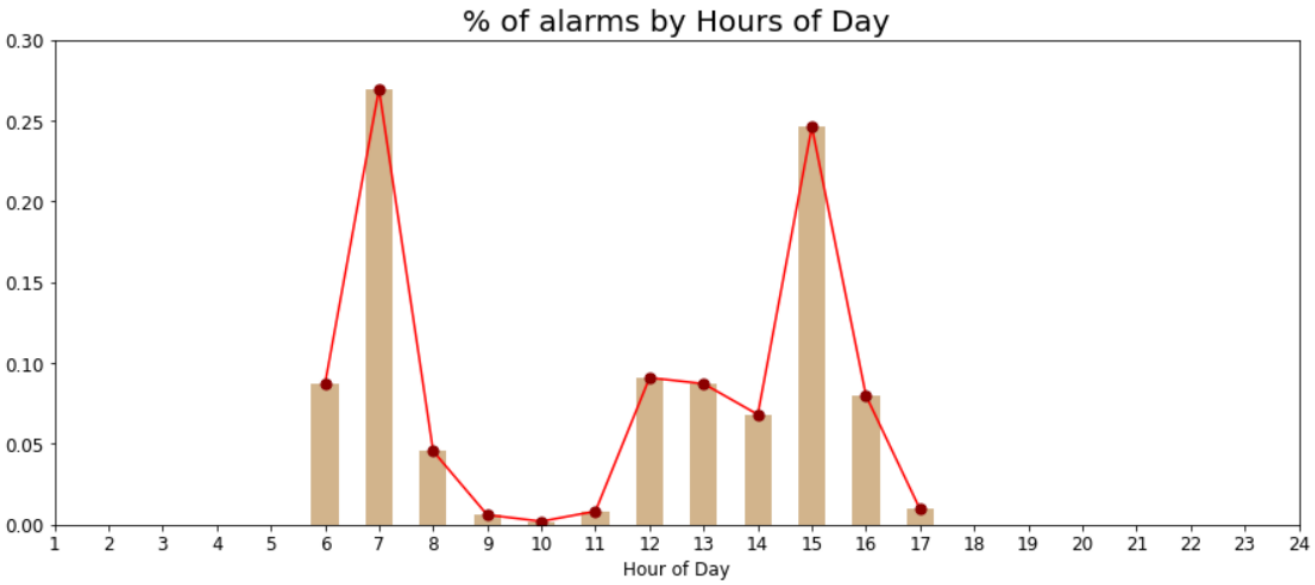
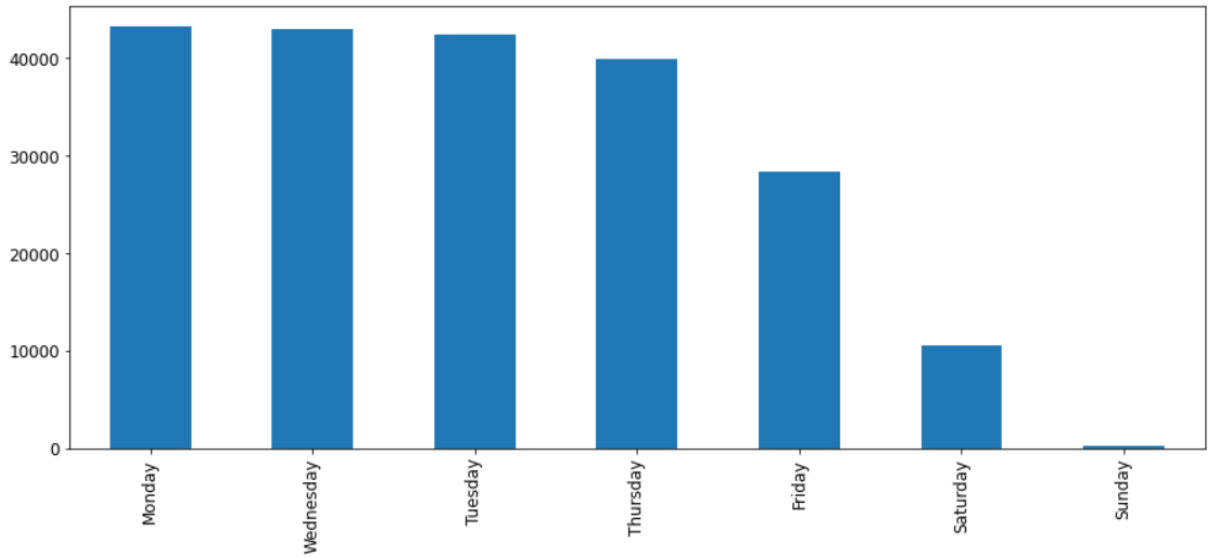


Fig. 4.2.15

In [38]:
```
plt.figure(figsize=[15,6])
df.Weekday.value_counts().plot(kind='bar')
plt.show()
```



4.2.16

In a week, Monday to Thursday recorded most of the incidents, with the number gradually going down on Friday and weekends. Sunday has the least number of alarms generated.
Among the days of a month, there's a peak on the 19th. It needs further study to find if that is a significant one.
In a day, early morning (7-8AM) and early evening (3-4PM) recorded the majority of the incidents. While early morning is a known peak-hour, evening peak traffic usually starts after 4PM. Hence further study is required to analyse this finding.

In [31]:
```
fig, axes = plt.subplots(2, 2, figsize=(20,10), sharey=True)

# LSCW
sns.kdeplot(data_lscw.speed, color='green', shade=True, ax=axes[0][0])
axes[0][0].axvline(data_lscw.speed.mean(), color='green', linestyle='dashed', linewidth=2)
axes[0][0].set_title('Low Speed Collisions')

# LSCW
sns.kdeplot(data_hscw.speed, color='red', shade=True, ax=axes[0][1])
axes[0][1].axvline(data_hscw.speed.mean(), color='red', linestyle='dashed', linewidth=2)
axes[0][1].set_title('High Speed Collisions')

# LSCW
sns.kdeplot(data_speed.speed, color='darkorange', shade=True, ax=axes[1][0])
axes[1][0].axvline(data_speed.speed.mean(), color='darkorange', linestyle='dashed', linewidth=2)
axes[1][0].set_title('Over Speeding')

# LSCW
sns.kdeplot(data_pcw.speed, color='magenta', shade=True, ax=axes[1][1])
axes[1][1].axvline(data_pcw.speed.mean(), color='magenta', linestyle='dashed', linewidth=2)
axes[1][1].set_title('Lane Departure and Pedestrian Collision')

plt.tight_layout()
plt.show()
```

Fig. 4.2.17

A distribution of speed for each of these accident/alarm types.

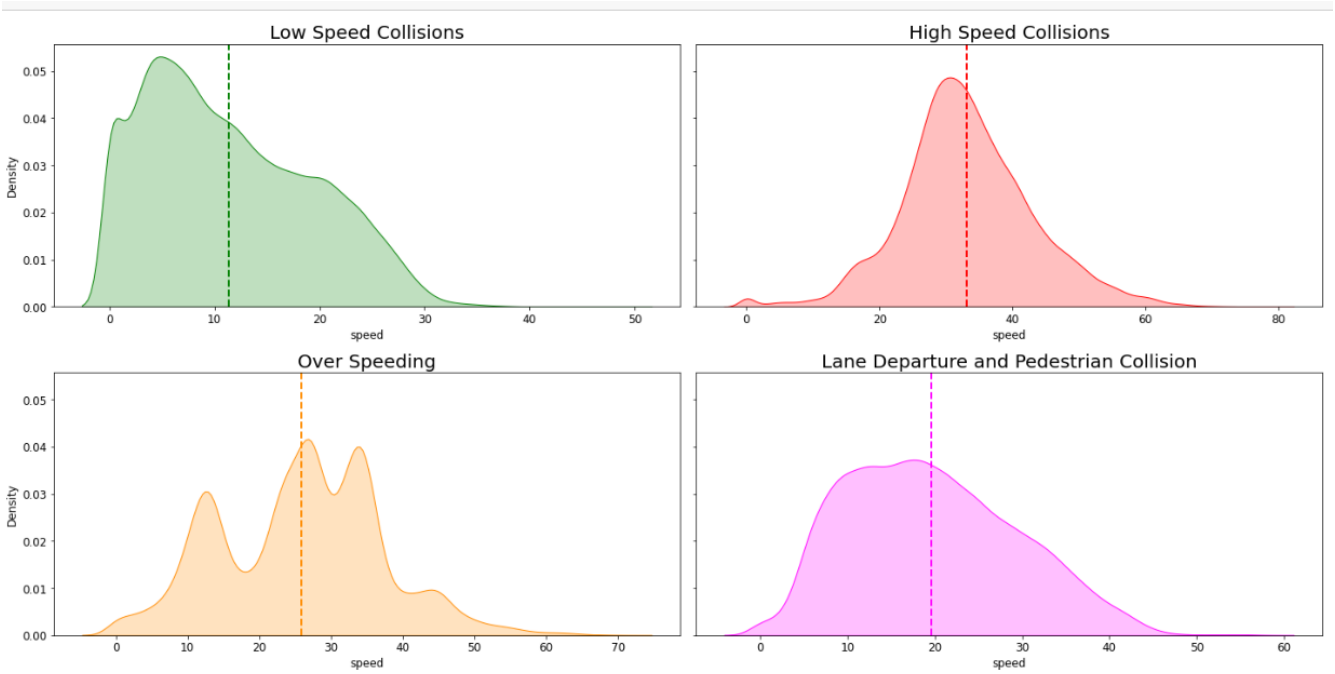


Fig. 4.2.18

Observations:

1. Low speed collisions happen at an average speed of 11-12 kmph.
2. High speed collisions happen at an average speed of approx 35 kmph, with many instances of higher speeds.
3. Overspeeding is reported at an average of 25 kmph speed.
4. Average speed of lane departures without indicators, and collisions with pedestrians and cyclists is 20 kmph.

```
In [32]: fig, axes = plt.subplots(figsize=(18,8))
         data = train_data['estimatedWardName'].value_counts(normalize=True).sort_values(ascending=False)
         data = data.head(10)
         ward_top = data.index

         ward_top_data = train_data[train_data.estimatedWardName.isin(ward_top)]
         sns.countplot(x='estimatedWardName', hue='alarmTypeCat', data=ward_top_data, ax=axes)

         axes.legend(title='Alarm Type')
         axes.set_xlabel('')
         axes.set_ylabel('# of Incidents')
         axes.set_title('What\'s going on in most dangerous wards?')

         plt.show()
```

Fig 4.2.19

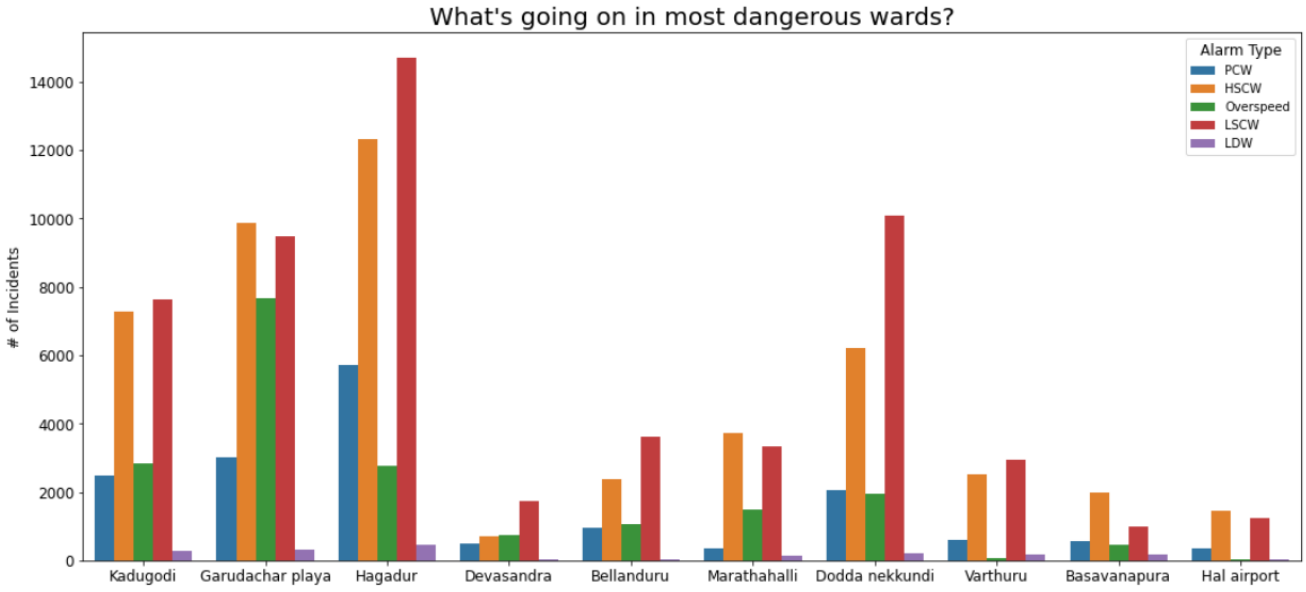


Fig. 4.2.20

Observations:

1. Most incidents of high and low speed collisions happen in Hagadur.
2. Overspeeding is most common in Garudachar Playa.
3. Hagadur is infamous for collisions with pedestrians and cyclist as well, not to mention lane change without indicators.

```
In [33]: plt.figure(figsize=[15,6])
         sns.kdeplot(df.Speed,shade=True,color='y')
         plt.axvline(df.Speed.mean(),linestyle='dashed',linewidth='2',color='k',label=df.Speed.mean())
         plt.legend(loc='best')
         plt.title('Average Speed of the buses from 6AM to 6PM')
         plt.show()
```

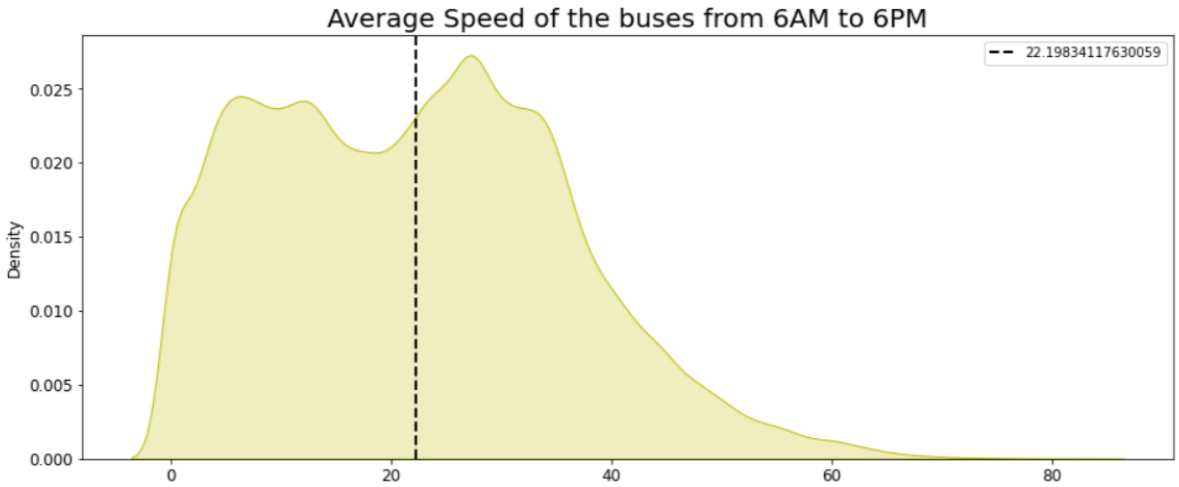


Fig. 4.2.21

```
In [34]: fig, axes = plt.subplots(3, 2, figsize=(20,10))


sns.kdeplot(df.Speed[df.AlarmType=='PCW'],shade=True,ax=axes[0][0])
axes[0][0].axvline(df.Speed[df.AlarmType=='PCW'].mean(),linestyle='dashed',color='k',label='PCW '+str(df.Speed[df.AlarmType=='PCW
axes[0][0].legend(loc='best')



sns.kdeplot(df.Speed[df.AlarmType=='FCW'],color='g',shade=True,ax=axes[0][1])
axes[0][1].axvline(df.Speed[df.AlarmType=='FCW'].mean(),linestyle='dashed',color='k',label='FCW '+str(df.Speed[df.AlarmType=='FCW
axes[0][1].legend(loc='best')



sns.kdeplot(df.Speed[df.AlarmType=='Overspeed'],color='y',shade=True,ax=axes[1][0])
axes[1][0].axvline(df.Speed[df.AlarmType=='Overspeed'].mean(),linestyle='dashed',color='k',label='Overspeed '+str(df.Speed[df.Ala
axes[1][0].legend(loc='best')


sns.kdeplot(df.Speed[df.AlarmType=='HMW'],color='cyan',shade=True,ax=axes[1][1])
axes[1][1].axvline(df.Speed[df.AlarmType=='HMW'].mean(),linestyle='dashed',color='k',label='HMW '+str(df.Speed[df.AlarmType=='HMW
axes[1][1].legend(loc='best')


sns.kdeplot(df.Speed[df.AlarmType=='LDW'],color='m',shade=True,ax=axes[2][0])
axes[2][0].axvline(df.Speed[df.AlarmType=='LDW'].mean(),linestyle='dashed',color='k',label='LDW '+str(df.Speed[df.AlarmType=='LDW
axes[2][0].legend(loc='best')
plt.show()
```

Fig. 4.2.22

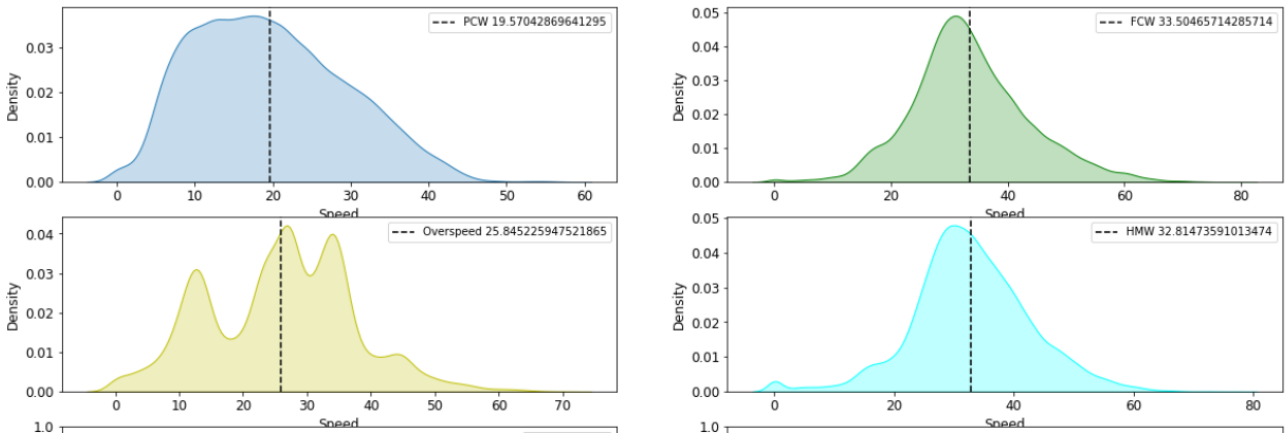


Fig. 4.2.23

Graphs of each type of alert have been drawn and it tells us overspeeding is not a cause for many accidents. On an average the speed of buses comes to around 22 km/hr

Observations

1. Data is provided only for the month of Feb,Mar,Apr,Jun,Jul
2. Data is only for the year 2018
3. Data is provided only between the time of 6AM to 6PM.

```
In [39]: df.groupby('WardName')['Hour'].value_counts().sort_values(ascending=False)

Out[39]: WardName        Hour
         other           7       31940
                         15      27710
                         6       11634
                         16       9332
                         13       8573
                                  ...
         Gurappanapalya  9           1
         Hudi            10          1
         Jogupalya       16          1
         Basavanapura    9           1
         Halsoor         12          1
         Name: Hour, Length: 280, dtype: int64

In [40]: df.groupby('WardName')['AlarmType'].value_counts().sort_values(ascending=False)

Out[40]: WardName        AlarmType
         other           UFCW         45950
                         HMW          18940
                         FCW          18907
                         PCW          13472
                         Overspeed    13450
                                      ...
         J P Nagar       HMW              1
         Singasandra     Overspeed        1
         Sudham Nagara   HMW              1
         Jayanagar East  HMW              1
         Jogupalya       HMW              1
         Name: AlarmType, Length: 247, dtype: int64
```

Fig. 4.2.24

PREDICTION MODEL

```
In [4]: bdf = bdf.rename(columns = {'deviceCode_time_recordedTime_$date':'timestamp'})

In [5]: bdf['timestamp'] = pd.to_datetime(bdf['timestamp'])
        bdf['eventDate'] = pd.to_datetime(bdf['timestamp'])
        bdf['eventDate'] = bdf['eventDate'].dt.strftime('%Y%m%d')

In [6]: bdf['e_hour'] = pd.to_datetime(bdf['timestamp'], format = '%H:%M:%S').dt.hour
        bdf['ehourCat'] = 0
        bdf['ehourCat'] = np.where((bdf['e_hour'] >= 0) & (bdf['e_hour'] < 6), 1, bdf['ehourCat'])
        bdf['ehourCat'] = np.where((bdf['e_hour'] >= 6) & (bdf['e_hour'] < 10), 2, bdf['ehourCat'])
        bdf['ehourCat'] = np.where((bdf['e_hour'] >= 10) & (bdf['e_hour'] < 16), 3, bdf['ehourCat'])
        bdf['ehourCat'] = np.where((bdf['e_hour'] >= 16) & (bdf['e_hour'] < 21), 4, bdf['ehourCat'])
        bdf['ehourCat'] = np.where((bdf['e_hour'] >= 21) & (bdf['e_hour'] < 24), 5, bdf['ehourCat'])
```

Fig. 4.2.25

We extract the date from the timestamp using the inbuilt functions of pandas and datetime libraries.

We take the hour during which the event occurred and map them to different categories to later study if the vehicle was observed during peak hours, early hours, regular hours and so on

```
In [7]: from time import time
        from datetime import timedelta
        import datetime
        from urllib.request import urlopen
        import re
        from pandas import ExcelWriter

        def date_splitter(dateToSplit):
            dateToSplit = dateToSplit.split('-', 1)
            year = dateToSplit[0]
            dateToSplit = dateToSplit[1].split('-', 1)
            month = dateToSplit[0]
            day = dateToSplit[1]
            return year, month, day

        def extractTime(s):
            time = re.findall('\d+:\d+', s)
            return time

        def extractConditions(s):
            x = re.findall('"small",h:"', s)
            condList = []
            for i in range(len(x)):
                s = s.split('"small",h:"', 1)
                s = s[1].split('."},{s', 1)
                condList.append(s[0])
                s = s[1]
            return condList

        def extractVisibility(s):
            s = s.replace(' ', '')
            re1 = '(\\d+)'
            re2 = ''
            re3 = 'km'
```

Fig. 4.2.26

Currently, we have features such as speed, area, latitude, longitude, the time when the observation was made. In addition to these features, we believe it would be beneficial to have the weather conditions when these events were observed. In order to do so, we scrape the weather data of Bangalore for the period of February until July 2018 from: https://www.timeanddate.com/weather/india/bangalore

```
In [9]: bwdf = pd.read_excel('bangalore-weather.xlsx')

In [10]: bwdf.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4636 entries, 0 to 4635
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   weatherDate  4636 non-null   int64
 1   time         4636 non-null   object
 2   temperature  4636 non-null   int64
 3   visibility   4636 non-null   int64
 4   condition    4636 non-null   object
dtypes: int64(3), object(2)
memory usage: 181.2+ KB

In [11]: bwdf.head(10)
```

Out[11]:

	weatherDate	time	temperature	visibility	condition
0	20180731	05:30	20	8	Partly cloudy
1	20180731	06:00	20	6	Partly cloudy
2	20180731	07:00	21	6	Broken clouds
3	20180731	07:30	22	6	Broken clouds
4	20180731	08:00	22	6	Broken clouds
5	20180731	08:30	22	6	Broken clouds
6	20180731	09:00	23	6	Broken clouds
7	20180731	09:30	24	6	Broken clouds

Fig. 4.2.27

We read the scraped data file now to get a glimpse of the contents.

```
In [11]: bwdf['w_hour'] = pd.to_datetime(bwdf['time'], format= '%H:%M').dt.hour
         bwdf['hourCat'] = 0
         bwdf['hourCat'] = np.where((bwdf['w_hour'] >= 0) & (bwdf['w_hour'] < 6), 1, bwdf['hourCat'])
         bwdf['hourCat'] = np.where((bwdf['w_hour'] >= 6) & (bwdf['w_hour'] < 10), 2, bwdf['hourCat'])
         bwdf['hourCat'] = np.where((bwdf['w_hour'] >= 10) & (bwdf['w_hour'] < 16), 3, bwdf['hourCat'])
         bwdf['hourCat'] = np.where((bwdf['w_hour'] >= 16) & (bwdf['w_hour'] < 21), 4, bwdf['hourCat'])
         bwdf['hourCat'] = np.where((bwdf['w_hour'] >= 21) & (bwdf['w_hour'] < 24), 5, bwdf['hourCat'])
         bwdf = bwdf.drop_duplicates(subset = ['weatherDate', 'hourCat'], keep = 'first')
         bwdf['ehourCat'] = bwdf['hourCat']
         bwdf['weatherDate'] = bwdf['weatherDate'].astype(str)

In [12]: bdf['weatherDate'] = bdf['eventDate']
         bdf['weatherDate'] = bdf['weatherDate'].astype(str)

In [13]: b1 = pd.merge(bdf, bwdf, on = ['weatherDate', 'ehourCat'], how = 'left')
```

Fig. 4.2.28

To merge the two dataframes on a mutual column, we create an extra column with the same name weatherDate in our previous dataframe bdf

Our merged dataset now looks like shown below

```
In [14]: b1.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 207617 entries, 0 to 207616
Data columns (total 17 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   deviceCode_deviceCode           207617 non-null  int64
 1   deviceCode_location_latitude    207617 non-null  float64
 2   deviceCode_location_longitude   207617 non-null  float64
 3   deviceCode_location_wardName    207617 non-null  object
 4   deviceCode_pyld_alarmType       207617 non-null  object
 5   deviceCode_pyld_speed           207617 non-null  int64
 6   timestamp                       207617 non-null  datetime64[ns, UTC]
 7   eventDate                       207617 non-null  object
 8   e_hour                          207617 non-null  int64
 9   ehourCat                        207617 non-null  int64
 10  weatherDate                     207617 non-null  object
 11  time                            190275 non-null  object
 12  temperature                     190275 non-null  float64
 13  visibility                      190275 non-null  float64
 14  condition                       190275 non-null  object
 15  w_hour                          190275 non-null  float64
 16  hourCat                         190275 non-null  float64
dtypes: datetime64[ns, UTC](1), float64(6), int64(4), object(6)
memory usage: 28.5+ MB
```

Fig. 4.2.29

The dataset we have currently does not mention about the accident prone zones of Bengaluru. This information was obtained from: http://bengaluru.citizenmatters.in/top-10-places-where-most-accidents-in-bengaluru-take-place-7260
We mapped the areas which we have in our dataset to the nearest accident prone zone marked in the link based on the number of fatalities reported by the traffic police. We performed this task manually.
Similarly, we also obtained the pothole data of Bengaluru from this link and repeated the task of mapping the pothole centric areas to the areas in our initial dataset. The pothole data was obtained manually from: https://www.potholeraja.com/live-potholeview

```
In [17]: badf = pd.read_excel('bangalore-accident-zones.xlsx')
```

```
In [18]: badf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Area               50 non-null     object
 1   Mapped_Location    50 non-null     object
 2   Accident_Severity  50 non-null     object
 3   Pothole_Severity   50 non-null     object
dtypes: object(4)
memory usage: 1.7+ KB
```

```
In [19]: badf.head(10)
```

Out[19]:

	Area	Mapped_Location	Accident_Severity	Pothole_Severity
0	A Narayanapura	Electronic City	High	High
1	Agaram	Electronic City	High	High
2	Banasavadi	Banasavadi	Low	Low
3	Basavanapura	K R Puram	Medium	High
4	Bellanduru	Electronic City	High	High
5	Benniganahalli	Banasavadi	Low	Low
6	Bharathi Nagar	K R Puram	Medium	High
7	BTM Layout	Madivala	High	Medium

Fig. 4.2.30

Some of the columns do not have enough entries compared to our initial entries. This is due to the fact that the weather scraper could not find information for all the hours when the events were recorded by the Collision Avoidance System. Hence, we replace the NaN data for which we could not find information with the Mode values, that is the most frequently occurring values of certain columns which we will use for our final prediction.

```
In [24]: for column in ['temperature', 'visibility', 'condition']:
             b[column].fillna(b[column].mode()[0], inplace=True)
```

```
In [25]: b.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 207617 entries, 0 to 207616
Data columns (total 21 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   deviceCode_deviceCode           207617 non-null  int64
 1   deviceCode_location_latitude    207617 non-null  float64
 2   deviceCode_location_longitude   207617 non-null  float64
 3   Area                            207617 non-null  object
 4   Alarm_Type                      207617 non-null  object
 5   Plying_Speed                    207617 non-null  int64
 6   timestamp                       207617 non-null  datetime64[ns, UTC]
 7   eventDate                       207617 non-null  object
 8   e_hour                          207617 non-null  int64
 9   ehourCat                        207617 non-null  int64
 10  weatherDate                     207617 non-null  object
 11  time                            190275 non-null  object
 12  temperature                     207617 non-null  float64
 13  visibility                      207617 non-null  float64
 14  condition                       207617 non-null  object
 15  w_hour                          190275 non-null  float64
 16  hourCat                         190275 non-null  float64
 17  Mapped_Location                 207617 non-null  object
 18  Accident_Severity               207617 non-null  object
 19  Pothole_Severity                207617 non-null  object
 20  hasOversped                     207617 non-null  int32
dtypes: datetime64[ns, UTC](1), float64(6), int32(1), int64(4), object(9)
memory usage: 34.1+ MB
```

Fig. 4.2.31

In [28]:
```
df = b.copy()
df['hasOversped'] = np.where(b.hasOversped == 1, 'Yes', 'No')
df['visibility'] = np.where(b.visibility == 0, 'Low', 'High')
df['ehourCat'] = b['ehourCat'].map({1: 'Early', 2: 'PeakM', 3: 'RegularM'})
b['Accident_Severity'] = b['Accident_Severity'].map({'High': 3, 'Medium': 2, 'Low': 1})
b['Pothole_Severity'] = b['Pothole_Severity'].map({'High': 3, 'Medium': 2, 'Low': 1})
b['Alarm_Type'] = b['Alarm_Type'].map({'PCW': 1, 'FCW': 2, 'Overspeed': 3, 'HMW': 4, 'UFCW': 5, 'LDWL': 6, 'LDWR': 7})
b['condition'] = b['condition'].map({'Clear': 1, 'Sunny': 2, 'Passing clouds': 3,
       'Broken clouds': 4, 'Scattered clouds': 5, 'Fog': 6, 'Haze': 7, 'Partly cloudy': 8,
       'Mild': 9, 'Drizzle. Broken clouds': 10})
b['Area'] = b['Area'].map({'Kadugodi': 1, 'Garudachar Playa': 2, 'Hudi': 3, 'Other': 4, 'Devasandra': 5,
       'Hagadur': 6, 'Bellanduru': 7, 'Marathahalli': 8, 'Dodda Nekkundi': 9, 'Varthuru': 10,
       'HAL Airport': 11, 'Vijnana Nagar': 12, 'Konena Agrahara': 13, 'A Narayanapura': 14,
       'C V Raman Nagar': 15, 'Jeevanbhima Nagar': 16, 'HSR Layout': 17, 'Domlur': 18, 'Jogupalya': 19,
       'Hoysala Nagar': 20, 'New Tippasandara': 21, 'Benniganahalli': 22, 'Singasandra': 23,
       'Basavanapura': 24, 'Halsoor': 25, 'Agaram': 26, 'Shantala Nagar': 27, 'Sampangiram Nagar': 28,
       'Sudham Nagara': 29, 'Dharmaraya Swamy Temple': 30, 'Chickpete': 31, 'Banasavadi': 32,
       'Horamavu': 33, 'Kacharkanahalli': 34, 'Kammanahalli': 35, 'Vijnanapura': 36, 'Ramamurthy Nagar': 37,
       'K R Puram': 38, 'BTM Layout': 39, 'Madivala': 40, 'Gurappanapalya': 41, 'J P Nagar': 42, 'Sarakki': 43,
       'Jaraganahalli': 44, 'Vasanthpura': 45, 'Hemmigepura': 46, 'Yelchenahalli': 47,
       'Jayanagar East': 48, 'Bharathi Nagar': 49, 'other': 4})

writer = ExcelWriter('bangalore-consolidated-data.xlsx')
b.to_excel(writer, index = False, sheet_name = 'Sheet1')
df.to_excel(writer, index = False, sheet_name = 'Sheet2')
writer.save()
```

In [29]:
```
del b['deviceCode_deviceCode'], b['deviceCode_location_latitude'], b['deviceCode_location_longitude']
del b['w_hour'], b['Mapped_Location'], b['timestamp'], b['e_hour'], b['weatherDate']
del b['hourCat'], b['time'], b['temperature'], b['eventDate'], b['Plying_Speed']

del df['deviceCode_deviceCode'], df['deviceCode_location_latitude'], df['deviceCode_location_longitude']
del df['w_hour'], df['Mapped_Location'], df['timestamp'], df['e_hour'], df['weatherDate']
del df['hourCat'], df['time'], df['temperature'], df['eventDate'], df['Plying_Speed']
```

Fig. 4.2.32

We now map all the columns we need to its numerical equivalent in the form of flags

In [34]:
```
from sklearn.cluster import KMeans
X = b.values.astype(np.float)
kmeans = KMeans(n_clusters = 2, max_iter = 2000, algorithm = 'full').fit(X)
kmf2labels = kmeans.labels_
kmf2labels = kmf2labels.tolist()
print('Finished clustering using K-Means')
```

```
C:\Users\sidby\AppData\Local\Temp\ipykernel_21772\42063032.py:2: DeprecationWarning: `np.float` is a deprecated alias for the b
uiltin `float`. To silence this warning, use `float` by itself. Doing this will not modify any behavior and is safe. If you spe
cifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  X = b.values.astype(np.float)
```

```
Finished clustering using K-Means
```

In [35]:
```
len(kmf2labels)
```

Out[35]:
```
207617
```

In [36]:
```
b['labels'] = kmf2labels
df['labels'] = kmf2labels
df['labels'] = df['labels'].map({0: 'High', 1: 'Low'})
```

Fig. 4.2.33

We create two sets of clusters here: one for low likelihood accident zones whereas the other one for high likelihood accident zones.

```
In [37]: b.head(10)
```

Out[37]:

	Area	Alarm_Type	ehourCat	visibility	condition	Accident_Severity	Pothole_Severity	hasOversped	labels
0	1	1	1	0	3	2	2	0	1
1	1	1	1	0	3	2	2	0	1
2	2	2	1	0	3	2	2	0	1
3	2	2	1	0	3	2	2	0	1
4	3	3	1	0	3	2	2	1	1
5	3	3	1	0	3	2	2	1	1
6	1	4	1	0	3	2	2	0	1
7	1	4	1	0	3	2	2	0	1
8	3	3	1	0	3	2	2	1	1
9	3	3	1	0	3	2	2	1	1

```
In [38]: df.head(10)
```

Out[38]:

	Area	Alarm_Type	ehourCat	visibility	condition	Accident_Severity	Pothole_Severity	hasOversped	labels
0	Kadugodi	PCW	Early	Low	Passing clouds	Medium	Medium	No	Low
1	Kadugodi	PCW	Early	Low	Passing clouds	Medium	Medium	No	Low
2	Garudachar Playa	FCW	Early	Low	Passing clouds	Medium	Medium	No	Low
3	Garudachar Playa	FCW	Early	Low	Passing clouds	Medium	Medium	No	Low
4	Hudi	Overspeed	Early	Low	Passing clouds	Medium	Medium	Yes	Low
5	Hudi	Overspeed	Early	Low	Passing clouds	Medium	Medium	Yes	Low

Fig. 4.2.34

1) Decision Tree:

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules. Below diagram illustrates the basic flow of decision trees for decision making with labels (Rain(Yes), No Rain(No)).

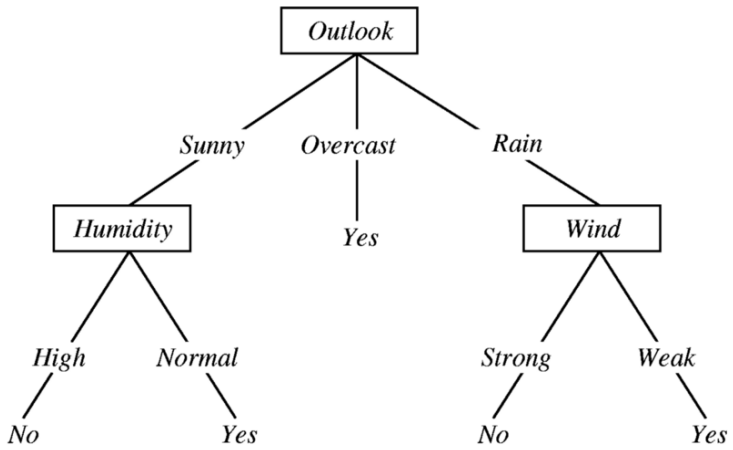


Fig. 4.2.35

Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.
Tree models where the target variable can take a discrete set of values are called classification trees. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Classification And Regression Tree (CART) is the general term for this.

Approach to make decision tree
While making a decision tree, at each node of the tree we ask different types of questions. Based on the asked question we will calculate the information gain corresponding to it.

Information Gain
Information gain is used to decide which feature to split on at each step in building the tree. Simplicity is best, so we want to keep our tree small. To do so, at each step we should choose the split that results in the purest daughter nodes. A commonly used measure of purity is called information. For each node of the tree, the information value measures how much information a feature gives us about the class. The split with the highest information gain will be taken as the first split and the process will continue until all children nodes are pure, or until the information gain is 0.

Gini Impurity
First let's understand the meaning of Pure and Impure.

Pure
Pure means, in a selected sample of dataset all data belongs to the same class (PURE).

Impure
Impure means, data is a mixture of different classes.

`

Definition of Gini Impurity
Gini Impurity is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were randomly classified according to the distribution of class labels from the data set.
If our dataset is Pure then the likelihood of incorrect classification is 0. If our sample is a mixture of different classes then the likelihood of incorrect classification will be high.

Steps for Making decision tree
- Get a list of rows (dataset) which are taken into consideration for making a decision tree (recursively at each node).
- Calculate uncertainty of our dataset or Gini impurity or how much our data is mixed up etc.
- Generate a list of all questions which needs to be asked at that node.
- Partition rows into True rows and False rows based on each question asked.
- Calculate information gain based on gini impurity and partition of data from the previous step.
- Update highest information gain based on each question asked.
- Update best question based on information gain (higher information gain).
- Divide the node on the best question. Repeat again from step 1 again until we get pure nodes (leaf nodes)

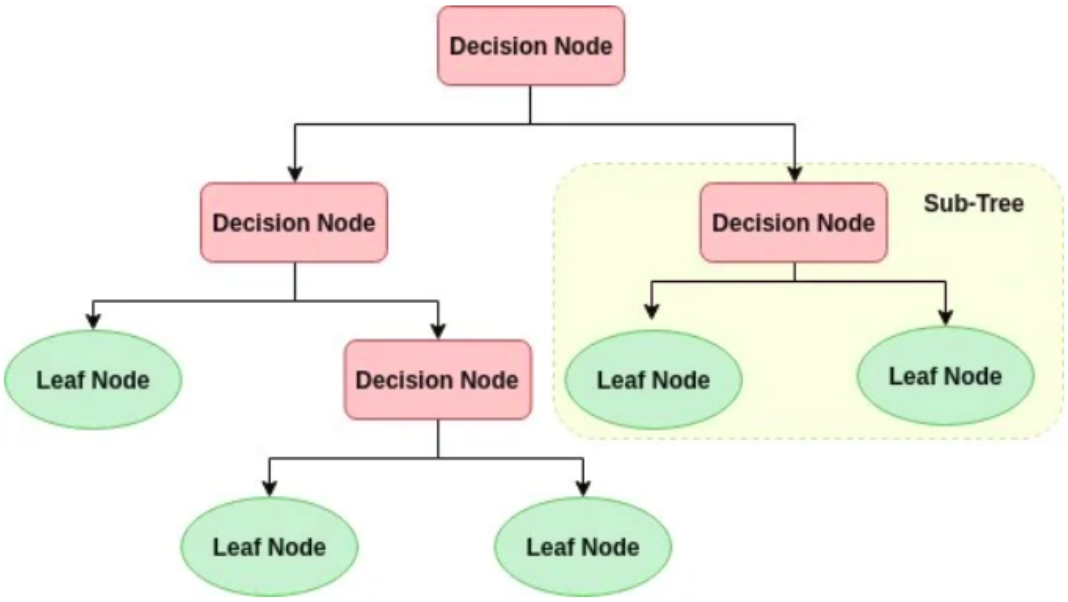


Fig. 4.2.36

```
In [41]: x = b.drop('labels', axis=1)
         y = b['labels']
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
         from sklearn.tree import DecisionTreeClassifier
         classifier = DecisionTreeClassifier()
         classifier.fit(X_train, y_train)
         print(classifier)

         DecisionTreeClassifier()

In [42]: feature_names = df.columns[:8]
         feature_names

Out[42]: Index(['Area', 'Alarm_Type', 'ehourCat', 'visibility', 'condition',
                'Accident_Severity', 'Pothole_Severity', 'hasOversped'],
               dtype='object')
```

Fig. 4.2.37

Training the model using decision tree ->

In [44]:
```
from sklearn import tree
text_representation = tree.export_text(classifier)
print(text_representation)
```

```
|--- feature_0 <= 5.50
|   |--- class: 1
|--- feature_0 >  5.50
|   |--- feature_0 <= 6.50
|   |   |--- feature_4 <= 1.50
|   |   |   |--- class: 0
|   |   |--- feature_4 >  1.50
|   |   |   |--- class: 1
|   |--- feature_0 >  6.50
|   |   |--- feature_4 <= 4.50
|   |   |   |--- class: 0
|   |   |--- feature_4 >  4.50
|   |   |   |--- feature_0 <= 7.50
|   |   |   |   |--- class: 1
|   |   |   |--- feature_0 >  7.50
|   |   |   |   |--- feature_4 <= 8.50
|   |   |   |   |   |--- feature_0 <= 8.50
|   |   |   |   |   |   |--- feature_4 <= 6.50
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- feature_4 >  6.50
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_0 >  8.50
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_4 >  8.50
|   |   |   |   |   |--- feature_0 <= 9.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_0 >  9.50
|   |   |   |   |   |   |--- class: 0
```

Fig. 4.2.38

```
In [98]: from sklearn.naive_bayes import GaussianNB
         from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

         NaiveBayes = GaussianNB()

         NaiveBayes.fit(X_train,y_train)

         predicted_values = NaiveBayes.predict(X_test)
         print(accuracy_score(y_test, predicted_values))
         print("Naive Bayes's Accuracy is: ", x)
```

```
0.9356757537809459
Naive Bayes's Accuracy is:          Area  Alarm_Type  ehourCat  visibility  condition  Accident_Severity  \
0          1           1         1           0          3                  2
1          1           1         1           0          3                  2
2          2           2         1           0          3                  2
3          2           2         1           0          3                  2
4          3           3         1           0          3                  2
...      ...         ...       ...         ...        ...                ...
207612     1           5         3           0          4                  2
207613     1           5         3           0          4                  2
207614     6           3         3           0          4                  2
207615     6           2         3           0          4                  2
207616     6           1         3           0          4                  2
```

Fig. 4.2.39

Accuracy through Naive Bayes's model is coming out to be 93.56%

```
In [103]: from sklearn.svm import SVC
          from sklearn.preprocessing import MinMaxScaler
          norm = MinMaxScaler().fit(X_train)
          X_train_norm = norm.transform(X_train)
          X_test_norm = norm.transform(X_test)
          SVM = SVC(kernel='poly', degree=3, C=1)
          SVM.fit(X_train_norm,y_train)
          predicted_values = SVM.predict(X_test_norm)
          x = accuracy_score(y_test, predicted_values)
          print("SVM's Accuracy is:",x)
```

```
SVM's Accuracy is: 0.9993256911665542
```

Fig. 4.2.40

Accuracy through SVM model is coming out to be 99.93%

```
In [100]: y_pred = classifier.predict(X_test)
          print(y_pred)
          from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
          print(confusion_matrix(y_test, y_pred))
          print(classification_report(y_test, y_pred))
          print(accuracy_score(y_test, y_pred))
```

```
[1 0 0 ... 1 1 1]
[[ 8705     0]
 [    0 32819]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      8705
           1       1.00      1.00      1.00     32819

    accuracy                           1.00     41524
   macro avg       1.00      1.00      1.00     41524
weighted avg       1.00      1.00      1.00     41524

0.9986995472497833
```

Fig. 4.2.41

Accuracy through Decision Tree model is coming out to be 99.86%

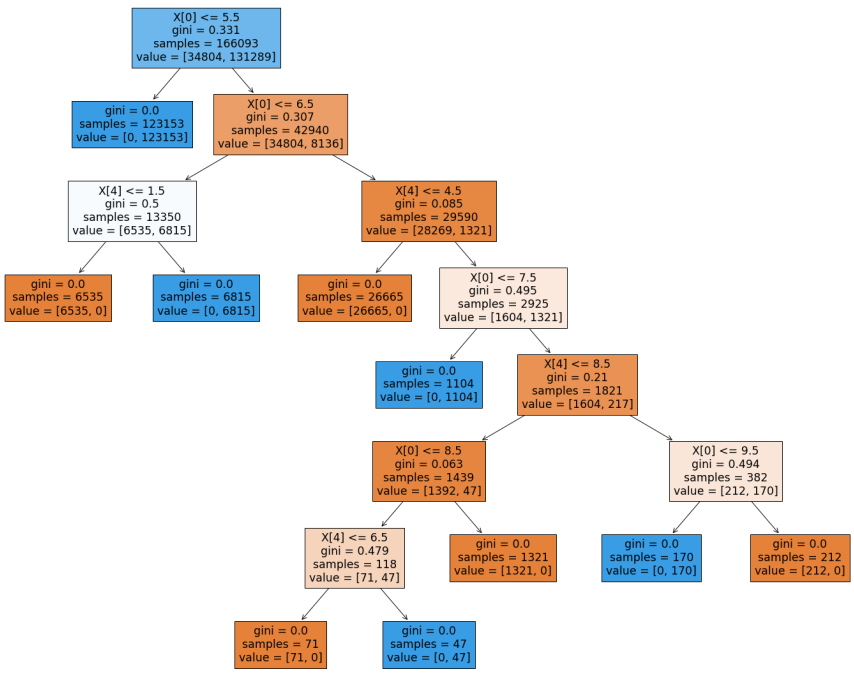


Fig. 4.2.42

CLUSTER ANALYSIS

```
In [51]: sns.violinplot(x = 'labels', y = 'Accident_Severity', data = b)
Out[51]: <AxesSubplot:xlabel='labels', ylabel='Accident_Severity'>
```

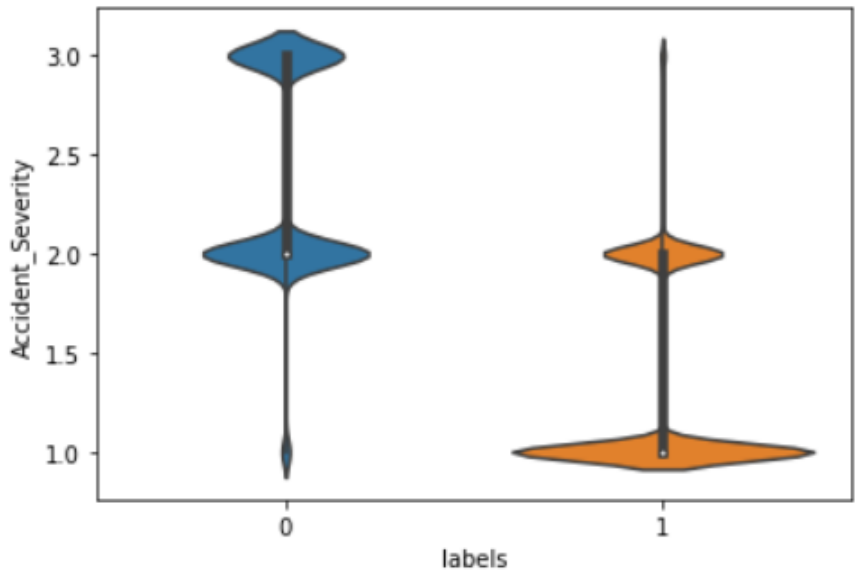


Fig. 4.2.43

```
In [52]: sns.violinplot(x = 'labels', y = 'hasOversped', data = b)
Out[52]: <AxesSubplot:xlabel='labels', ylabel='hasOversped'>
```

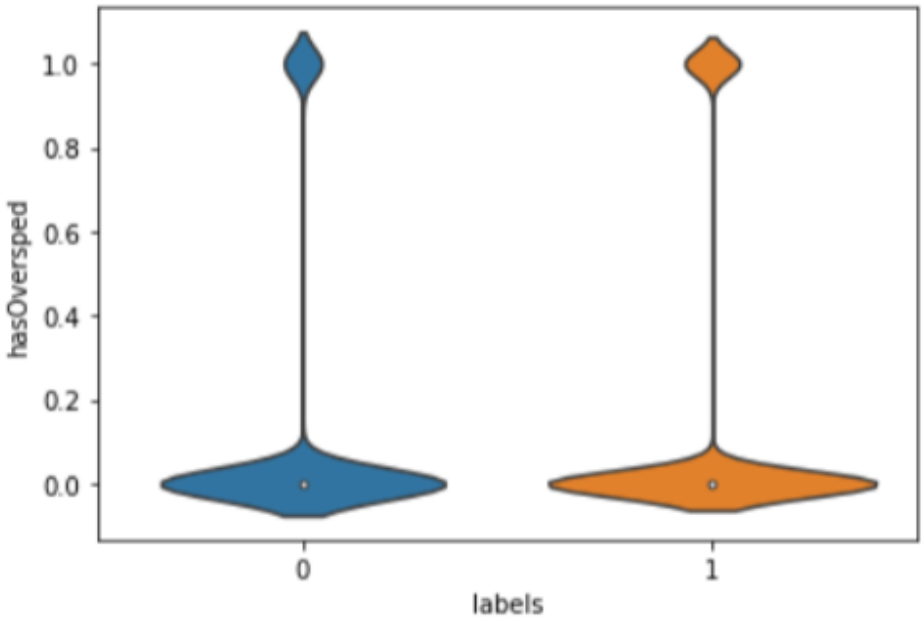


Fig. 4.2.44

```
In [54]: sns.violinplot(x = 'labels', y = 'ehourCat', data = b)
```

```
Out[54]: <AxesSubplot:xlabel='labels', ylabel='ehourCat'>
```

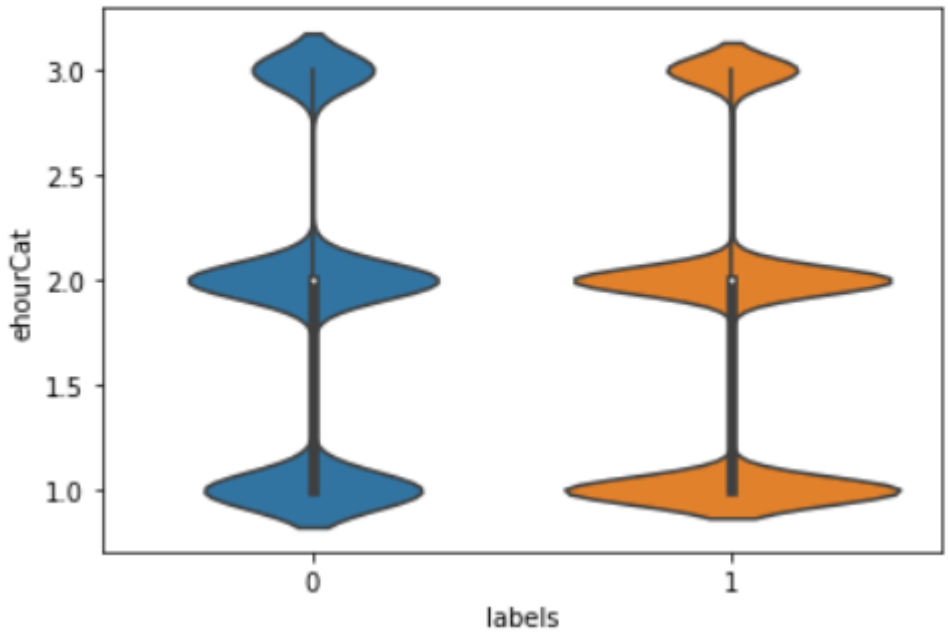


Fig. 4.2.45

```
In [55]: sns.violinplot(x = 'labels', y = 'Pothole_Severity', data = b)
```

```
Out[55]: <AxesSubplot:xlabel='labels', ylabel='Pothole_Severity'>
```

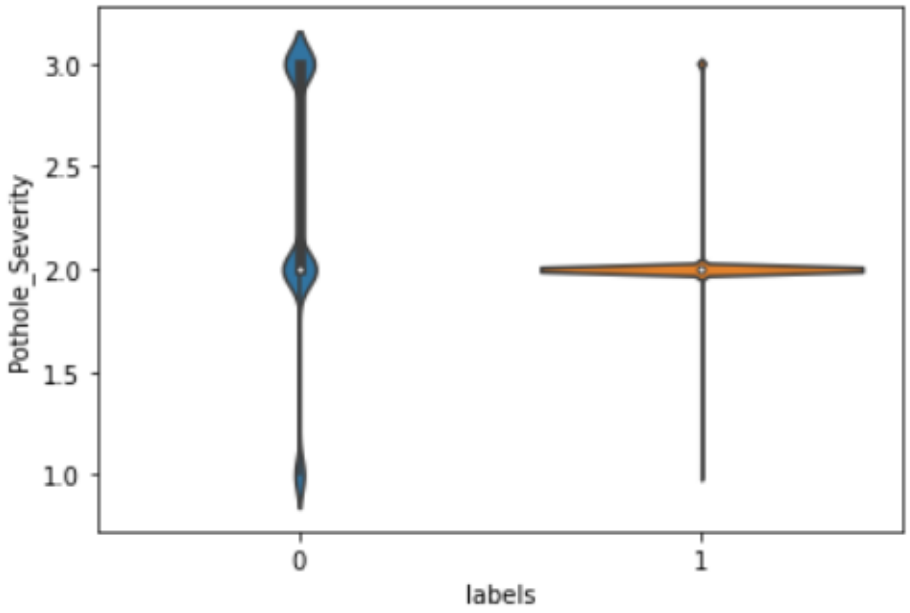


Fig. 4.2.46

4.3 Risk Analysis and Mitigation

1. Passengers should try to avoid using the most dangerous wards during the peak times(7-8 and 15-16) on weekdays.
2. All the vehicles should have a proper gap between them especially duting traffic or slow movements of vehicles so that we can avoid the collision of vehicles for which the maximum alerts are raised.
3. Pedestrians should be careful while crossing roads mainly in the areas of Kadugodi, Hagadur.
4. Drivers should have a control on speed in the Hudi area as many alerts has been raised for Overspeeding
5. Minimum 3 sec gap between vehicles to avoid collision. Strict rules to fine drivers who violate this rule would result in reducing FCW alerts.
6. From the hours provided, we can have shift timing work in office so that vehicles are equally distributed on all hours on roads which will lead to less traffic or we can provide work from home options.
7. Following the traffic rules such a speed control and traffic signals so that pedestrians can be safe. Fine drivers who violate traffic rules and max speed limit to avoid PCW. Provide foot over bridge for road crossing

CHAPTER - 5

FINDINGS, CONCLUSION, AND FUTURE WORK

5.1 Findings

Out of 3 machine learning algorithms: Decision Tree, Naive Bayes, Support Vector Machine. The Support Vector Machine gives the best accuracy for prediction of probability of an accident happening. We further studied the Decision Tree in detail for the same dataset.
The final analysis gives the following results:

Factors	**HIGH Accident Severity Prediction**	**LOW Accident Severity Prediction**
Wards	Hagadur, Garudachar playa, Kadugodi, Dodda nekkundi, Marathahalli, Bellanduru, Varthuru, Basavanapura, devasandra, Hal airport.	Shivaji nagar, Bilekhali, Bharathi nagar, Sudham nagara, Sarvagna nagar, Kammanahalli, Gurappanapalya, Shanthi nagar, Dharmaraya swamy temple, Jayanagar.
Alarm Type	LSCW: Low Speed Collision Warning	PCW: Pedestrians and Cyclist detection and Collision Warning.
Pothole Severity	Medium	High
Weather Condition	Fog, Passing clouds, Broken Clouds	Sunny, Haze, Scattered Clouds, Partly clouds
Duration	Peak	Early, Peak
Overspeed	NO	NO
Suggestions	● Passengers should try to avoid using the most dangerous wards during the peak times(7-8 and 15-16) on weekdays. ● Pedestrians should be careful while crossing roads mainly in the areas of Kadugodi, Hagadur. ● Minimum 3 sec gap between vehicles to avoid collision. ● Proper installation of Traffic lights and fine drivers who violate traffic rules. ● Raise alerts on worsening weather condition.	● Drive in the prescribed speed limits on the various roads. Always remember that “Speed thrills but kills”. ● Always put on helmets, seat belts and other safety equipments before driving a bicycle/ motor cycle/vehicle. ● Never use mobile phones or ear phones while driving. ● Proper maintenance of roads with potholes. ● Fine drivers who violate traffic rules and max speed limit to avoid PCW. Provide foot over bridge for road crossing.

Table 5.1.1

5.2 Conclusion

We successfully applied K Means clustering on our data and divided all the instances into two categories concluding the probability of an accident to occur as high or low. Based on these results we constructed a decision tree to get some fruitful results.

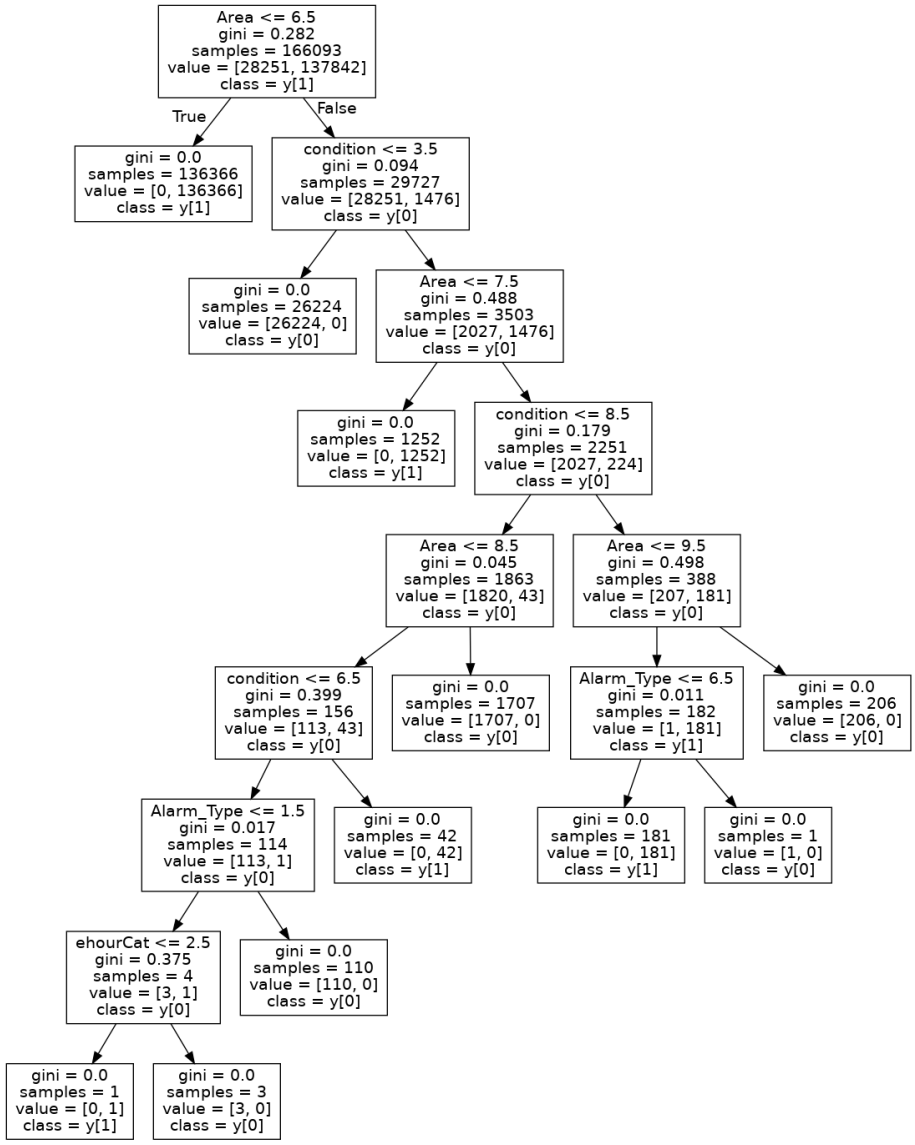


Fig. 4.2.47

This decision tree concludes that major attributes that affect the probability of an accident occurring are Area, weather condition, alarm type, and time. It is usually observed that in the places where there exist severe potholes, accidents are bound to take place. So this one attribute hasn't been considered while constructing the tree since it's effect is pretty obvious.

5.3 Future Work

For future work, more recent data (2020/2021) will be provided to us that will allow us to improve the proposed work. The first concerns the dataset used. This research is based on a road traffic accident dataset from the year 2013-18 which contains very few data samples for the no-injury and non-incapacitating injury types of accident. The second limitation concerns the dependence of the decision model on parameters and attribute selection.Since we have highlighted road accidents as the main factor influencing accident severity it would be interesting to include traffic parameters and intensity in our approaches. Moreover, to make it more feasible, we will try to make a recommender system by using these approaches that can give a prediction to the traffic accident and can warn the road user. In the future, it will be our try to create a mobile application by implementing this methodology to provide an accurate prediction to the user and make it very useful and beneficial also. Lastly, queries can be developed regarding the hotspots by which measures like traffic signals, speed breakers, toll booths could be modelled to minimise the accidents.

CHAPTER 6
REFERENCES

[1] Geodetski Vestnik, “Assessing road accidents in spatial context via statistical and non-statistical approaches to detect road accident hotspot using GIS”, September 2022

[2] Tessa Anderson, “Comparison of spatial methods for measuring road accident ‘hotspots’: A case study of London”, January 2020

[3] Mahreen Ahmed, Rafia Mumtaz, Zahid Anwar, “Spatiotemporal Clustering and Analysis of Road Accident Hotspots by Exploiting GIS Technology and Kernel Density Estimation”, May 2020

[4] Daniel Santos, José Saias, Paulo Quaresma and Vítor Beires Nogueira, “Machine Learning Approaches to Traffic Accident Analysis and Hotspot Prediction”, November 2021

[5] G. Mani Kandan, Sarilya Jaiswal, Rahul Mishra & Mrs. Steffina Muthukumar, “Analysis and Prediction of Road Accidents by Graphical Visualiser using ML”, March 2019

[6] Mamoudou Sangare, Sharut Gupta, Samia Bouzefrane, Soumya Banerjee, Paul Muhlethaler, “Exploring the forecasting approach for road accidents: Analytical measures with hybrid machine learning.”, April 2021.

[7] Vivian Brian Lobo, Jayesh Patil, Vaibhav Patil, Dhaval Walavalkar, “Road Accident Analysis and Hotspot Prediction using Clustering.”, 2021.