Summer Internship Report

by Vedant Singh

Submission date: 15-Jul-2025 01:22PM (UTC+0530)
Submission ID: 2715320303
File name: Vedant_Summer_Internship_Plag_check.pdf (3.29M)
Word count: 7607
Character count: 41505

Summer Internship Report

ORIGINALITY REPORT

2%	1%	1%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Hertfordshire Student Paper	<1%
2	www.coursehero.com Internet Source	<1%
3	Submitted to Technological Institute of the Philippines Student Paper	<1%
4	predictivethought.com Internet Source	<1%
5	"Machine Learning for Networking", Springer Science and Business Media LLC, 2020 Publication	<1%
6	shura.shu.ac.uk Internet Source	<1%
7	Tanya Buddi, Rohit Kandakatla, Rameshrao Nitin Kotkunde, Upadrasta Ramamurty, Asma Perveen. "Multi-Disciplinary Research and	<1%

Summer Internship Report

on

Crime-Sensitive Navigation System for Safer Urban Travel

(May 30, 2025 – July 17, 2025)

by

VEDANT SINGH

(Third Year, 6th Semester Undergraduate, Information Technology)

Indian Institute of Information Technology, Sonepat

Haryana - 131021

Under the supervision of

Professor D.S. KUSHWAHA

(Computer Science and Engineering Department)

Motilal Nehru National Institute of Technology Allahabad,

Prayagraj – 211004, U.P. India

July 2025

Date : July 17, 2025

To,

The Faculty In-charge

SNFCE, MNNIT Allahabad

Prayagraj-211004

Uttar Pradesh, India

Through – Head of Department, CSED

Subject: Submission of summer internship report and request to issue the Internship Completion Certificate.

Respected Sir,

I, **Vedant Singh,** B.Tech., Third year, I.T. Branch student at IIIT Sonepat, Haryana, am writing to submit my internship report and request the issuance of my internship completion certificate. It has been a privilege to be at Motilal Nehru National Institute of Technology, Allahabad, Prayagraj, as an intern from **May 30, 2025 – July 17, 2025**.

During my internship I had the opportunity to work on **Project title**: **Crime-Sensitive Navigation System for Safer Urban Travel**, which enhanced my skills and knowledge in the field of Information Technology and Computer Science and Engineering.

Enclosed with this letter, is my internship report, detailing my contribution and learning outcomes. I kindly request you to review my report and issue the internship completion certificate at your earliest convenience. Thank you for the opportunity and support.

Yours Sincerely,

Vedant Singh
Roll No. 12212030
Information Technology
Indian Institute of Information Technology Sonepat
Sonepat, Haryana, 131021
Contact No.: +91 6387269537
Email: svedant2103@gmail.com

Date: July 17, 2025

To,
The Head,
Computer Science and Engineering Department (CSED)
MNNIT Allahabad
Prayagraj-211004
Uttar Pradesh, India

Subject: Submission of summer internship report and request to issue the Internship Completion Certificate.

Respected Sir,

I, Vedant Singh, B.Tech III year, I.T. Branch student at IIIT Sonepat, Haryana, am writing to submit my internship report and request the issuance of my internship completion certificate. It has been a privilege to be at Motilal Nehru National Institute of Technology Allahabad, Prayagraj, as an intern from May 30, 2025- July 17 2025.

During my internship, I had the opportunity to work on project title: Crime Sensitive Navigation System for Safer Urban Travel, which enhanced my skills and knowledge in the field of Information Technology.

Enclosed with this letter, is my internship report, detailing my contribution and learning outcomes. I kindly request you to review my report and issue the internship completion certificate at your earliest convenience.

Thank you for the opportunity given and support provided to me.

Your Sincerely,

(Vedant Singh)
Roll No. 12212030
IIIrd year student, IT Branch
IIIT Sonepat, Haryana- 136119
Contact no. +91 6387269537
Email: svedant2103@gmail.com

UNDERTAKING

I, Vedant Singh, student of B.Tech, IIIrd year, IT Branch at IIIT Sonepat, Haryana, hereby declare that the project report entitled "Crime Sensitive Navigation System for Safer Urban Travel" submitted for the summer internship program at the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology (MNNIT), is my original work and has not been submitted earlier, either partly or fully, to any other university or any other institution for the award of any degree, diploma, or any other purpose. I further declare that all the information and data given in the report is genuine and correct to the best of my knowledge and belief.

Place: MNNIT, Allahabad

(Vedant Singh)
B.Tech IIIrd year, 6th Semester
IT Branch
IIIT Sonepat, Haryana- 136119

CERTIFICATE

This is to certify that the report entitled **"Crime-Sensitive Navigation System for Safer Urban Travel",** submitted by Vedant Singh, a B.tech 3rd year (6th semester) student of Information Technology Department at IIIT Sonepat, Haryana, is a bonafide record of the project work carried out by him during his summer internship program **during 30th May to 17th July 2025** in the Department of Computer Science and Engineering at Motilal Nehru Institute of Technology Allahabad (MNNIT), Prayagraj.

Place: MNNIT Allahabad

(Prof. D.S. Kushwaha)
Internship Supervisor
Computer Science and Engineering Department (CSED)
MNNIT, Allahabad
PRAYAGRAJ

ACKNOWLEDGEMENT

I, Vedant Singh, would like to extend my deep sense of gratitude to Professor D. S. Kushwaha, Department of Computer Science and Engineering, my Mentor for summer internship programme, who has given me opportunity to work with him on the project entitled “Crime Sensitive Navigation System for Safer Urban Travel”. Without his able guidance and support, it would have not been possible to complete the task given to me. I once again extend my thanks to D. S. Kushwaha.

I further deeply acknowledge and extend my regards to Head, Computer Science and Engineering, MNNIT Allahabad and faculties for providing me support whenever required.

I am also thankful to the staff of SNFCE for their full support and help rendered by them.

At last, I can not forget to thank my mother, father and sisters who rendered guidance from time to time during period of internship.

I take this opportunity to express my gratitude towards everyone who has been instrumental in the successful completion of this project file.

Place: MNNIT, Allahabad

(Vedant Singh}
B.Tech. IIIrd year, 6th Semester
IT Branch
IIIT Sonepat, Haryana- 136119

ABSTRACT

Internship focused on the design and development of a crime-sensitive navigation system aimed at suggesting safest travel routes within urban areas. The project utilizes spatial crime data to help individuals avoid high-risk areas by suggesting alternative, less-crime paths based on GIS mapping and graph network analysis. With the increasing demand for public safety in urban areas, such systems are becoming essential components, especially for vulnerable populations like pedestrians, late-night workers, delivery agents, working professionals, and other such kinds of individuals.

The aim of this project was to analyze how crime risk scores can be linked with traditional navigation systems. The system consists of several key factors like time of occurrence, location, severity, and frequency of crime incidents. These factors are used to calculate a risk score for various parts of a city in India. Crime data was visualized using interactive maps to highlight crime hotspots, and a risk-aware routing model was developed to recommend the safest possible path between a source and destination.

To implement this routing model, road network graphs were created for each city within India using OpenStreetMap GIS data through OSMnx library. For calculating the safest shortest path, Dijkstra's algorithm was used in two ways - one considering only the distance measure to find the shortest path, and another factored in crime risk scores to avoid dangerous routes. The results were visualized on an interactive map built using Folium (GIS tool), with red paths indicating the shortest route (irrespective of safest routes), and green paths indicating the safest route (even if longer). Crime zones were shown using red marks, and detailed information was added to show risk scores and city details.

The overall application demonstrates the potential of combining crime analytics with navigation systems to create smarter, safety-aware urban mobility solutions. It provides a user-friendly interface that can help reduce exposure to high-risk areas. This project shows how data science, machine learning, and geospatial analysis can be integrated together to build socially impactful solutions that promote safer and more informed travel decisions in urban areas.

CONTENTS

Section	Description	Page No.
	UNDERTAKING	I
	CERTIFICATE	Ii
	ACKNOWLEDGEMENT	Iii
	ABSTRACT	Iv
	CONTENTS	V
	LIST OF FIGURES	Vii
	LIST OF TABLES	Ix
CHAPTER 1	**INTRODUCTION**	1-5
	1.1 Background and Motivation	1
	1.2 Problem Outline	1
	1.3 Project Objectives	2
	1.4 Overview of Tools and Technologies Used	2
	1.5 Project Methodology	3
	1.6 Limitations	4
	1.7 Summary	5
CHAPTER 2	**LITERATURE REVIEW**	7-11
	2.1 Introduction	7
	2.2 Previous Studies of Crime Prediction and Safest Routing	8
	2.3 Machine Learning Techniques used in Prediction	8
	2.4 Gap Identified in Literature Review	10
	2.5 Advancements Over Prior Approaches	11
	2.6 Summary	11
CHAPTER 3	**IMPLEMENTATION**	13-31
	3.1 Introduction	13
	3.2 Code Snippet	13
	3.2.1 Data Collection	13
	3.2.2 Data Preprocessing	15
	3.2.3 Model Training and Evaluation	21
	3.2.4 Analyze Crime Hotspot using Folium (GIS tool)	25
	3.2.5 Creating the Road Network using OSMnx library	27
	3.2.6 Finding Shortest and Safest Path using Dijkstra's Algorithm	29
	3.2.7 Summary	31

CHAPTER 4	**RESULTS AND DISCUSSION**	33-40
	4.1 Dataset Description	33
	4.2 Evaluation Metrices	34
	4.3 Results and Analysis	34
	4.3.1 Initial plots based on EDA	34
	4.3.2 Comparision of different models based on mean accuracy scores	37
	4.3.3 Visualize Crime Hotspots on Folium Map for multiple cities	38
	4.3.4 Visualize Safest and Shortest Paths using Dijkstra's Algorithm on Folium Map for multiple cities	40
CHAPTER 5	**CONCLUSION AND FUTURE SCOPE**	43
	5.1 Conclusion	43
	5.2 Future Scope	43
	REFERENCES	45-46
	APPENDIX	47

****

LIST OF FIGURES

Figure No.	Figure Title	Page No.
Fig 2.1	Project Workflow	7
Fig 3.1	Code snippet illustrating data loading process	15
Fig 3.2	Code snippet for dropping rows with missing values	15
Fig 3.3	Code snippet for filling missing values with random placeholder	15
Fig 3.4	Code snippet for dropping the 'Weapon Used' column	16
Fig 3.5	Code snippet for converting date columns into standard format	16
Fig 3.6	Code snippet for categorizing whether crime occurred in day/night	17
Fig 3.7	Code snippet for assigning custom severity scores to each type of crimes	17
Fig 3.8	Code snippet for performing encoding on 'Case Closed' attribute	18
Fig 3.9	Code snippet for creating new feature 'risk score' for analysis	18
Fig 3.10	Code snippet for loading geo-logical data from external source	19
Fig 3.11	Code snippet for removing unwanted attribute from geo-logical data	19
Fig 3.12	Code snippet for merging and exporting geo-logical data as final dataset	20
Fig 3.13	Code snippet for normalizing 'risk score' feature using Min-Max Scaler	20
Fig 3.14	Code snippet for classifying risk levels into three categories	21
Fig 3.15	Code snippet for categorizing risk levels into different bins	22
Fig 3.16	Code snippet for training LR model for 80-20 hold-out ratio	23
Fig 3.17	Code snippet for training SVC model for 80-20 hold-out ratio	23
Fig 3.18	Code snippet for training XGB model for 80-20 hold-out ratio	24
Fig 3.19	Code snippet for training LightGBM model for 80-20 hold-out ratio	24
Fig 3.20	Code snippet for training MLP model for 80-20 hold-out ratio	25

Fig 3.21	Code snippet for extracting unique cities	26
Fig 3.22	Code snippet for counting Risk Level categories	26
Fig 3.23	Code snippet for visualizing crime heatmaps using Folium	27
Fig 3.24	Code snippet for creating the road network using OSMnx library	28
Fig 3.25	Code snippet for defining source to destination points for pathfinding	29
Fig 3.26	Code snippet for assigning weights to each road segment	30
Fig 3.27	Code snippet for identifying shortest path using Dijkstra's algorithm	30
Fig 3.28	Code snippet for identifying safest path using Dijkstra's algorithm	31
Fig 4.1	Comparison of number of crimes happened in day vs night	35
Fig 4.2	Visualizing Crime Count by Crime Type (Top 10 Crimes)	35
Fig 4.3	Visualizing Number of Crimes by Day of the Week	36
Fig 4.4	Visualizing High-Risk Crimes That Occurred at Night within each city	36
Fig 4.5	Visualizing Distribution of Calculated Risk Scores through KDE	37
Fig 4.6	Visualizing Comparision of different models based on mean accuracy scores	38
Fig 4.7	Visualizing Crime Hotspots on Folium Map for 'Mumbai' city	39
Fig 4.8	Visualizing Crime Hotspots on Folium Map for 'Indore' city	39
Fig 4.9	Visualizing Shortest and Safest path on Folium Map for 'Delhi' city	40
Fig 4.10	Visualizing Shortest and Safest path on Folium Map for 'Pune' city	41

LIST OF TABLES

Table No.	Table Title	Page No.
4.1	Dataset Description	33
4.2	Comparison of Proposed Model vs Referenced Model Performance Based on Mean Accuracy Scores	37

CHAPTER-1

INTRODUCTION

1.1 Background and Motivation

In past few years, individuals residing in urban areas lack safety problems which has become a growing concern, especially with the increasing frequency of crimes reported in metropolitan areas. In today's world, some amazing navigation pre-build applications like Google Maps have made commuting tasks easier, but they are totally oriented towards time, distance from source to destination, and traffic measures —not safety. While these currently running applications are effective for finding the fastest shortest route from source to destination travel in metropolitan areas, they do not consider the presence of crime-zone areas, which can cause serious risks to individuals, especially during late-night working hours or in unfamiliar locations.

This huge gap in navigation systems regarding safety concerns led to motivation for this project. The sole aim is to build a crime-sensitive navigation system that not only guides individuals from one place to another but also ensures that the path taken avoids areas known for criminal activity. By integrating crime static data with geospatial mapping and routing algorithms, the system aims to enhance personal safety and create more informed travel decisions for individuals in urban areas. This is especially beneficial for pedestrians, elderly citizens, late-night workers, delivery agents, and other travelers who may feel uncomfortable while commuting.

1.2 Problem Outline

The existing modern navigation systems lack the ability to identify risk or detect crime-prone areas while suggesting a route. Individuals are unaware if the shortest or fastest path goes through high-crime zones. As a result, people may unintentionally take routes that are dangerous where the frequency of crime-rates is high, especially in areas with poor lighting or low police deployment.

Moreover, there is limited integration of publicly available crime static data into mapping applications, even though such data can help reduce crime incidents and improve safety awareness among individuals in metropolitan areas. The absence of a safety-focused routing method creates a bridge between existing systems that this project aims to fill.

1.3 Project Objectives

The main objective of this project is to develop a crime-sensitive navigation system that promotes safety in addition to distance. The system should be capable of:

- Collecting and pre-processing real-world crime static data from public sources like – Kaggle, National Crime Record Bureau (NCRB), India Data Portal (IDP) etc.
- Mapping crime-zone areas on an interactive city map (Leaflet Map) using Folium library.
- Assigning a risk score to different road segments based on proximity to crime locations using OSMnx library.
- Applying graph-based pathfinding algorithms (like the Dijkstra algorithm) to determine both the shortest and safest routes.
- Visualizing final results on an interactive map so individuals can clearly see the safest and shortest paths, along with high-risk areas which are crime sensitive.

This system ultimately seeks to empower individuals with choices: whether to take the shortest path or the safest path as an alternative option.

1.4 Overview of Tools and Technologies Used

This project combines several Python libraries and data visualization tools that work together to build a functional, efficient, and user-friendly crime-sensitive navigation system. Some of the tools and technologies which are used in this project are listed here:

- **Pandas:** This library was used to load and prepare the crime dataset. It played a vital role in cleaning and handling missing values, organizing data columns, and creating derived features like risk scores and time categories which are essential for determining crime hotspots for analysis.
- **NumPy:** This library was used to perform some kind of numerical operations like normalization and scaling using Min-Max Scaler and other techniques. It played a vital role in adjusting the weight of features like victim's age, police deployment, and severity scores used in risk scores calculations for finding high crime-prone areas.
- **Matplotlib:** This python library was used to create some amazing static plots like bar graphs, line plots, and histograms for visualizing the distribution of crime-related features. It helped represent trends such as the number of counts of high-risk incidents or severity scores across categories. These kinds of plots provide useful insights from given raw data during the early analysis before mapping the data.

- **Seaborn:** This python library was used for generating visually appealing probabilistic or statistical plots like KDE plots and heatmaps. It made it easier to find correlations and patterns in the given raw data, such as identifying peak risk hours or comparing severity across crime types.
- **OSMnx (OpenStreetMap + NetworkX):** This python library was used to fetch and download the walkable road network data for each unique cities within India from OpenStreetMap (OSM). This allowed us to convert roads into graph networks, making it suitable for applying pathfinding algorithms (Dijkstra's algorithm).
- **NetworkX:** This library helps in constructing the city graph and applying Dijkstra's algorithm for finding the shortest and safest path among all cities to reduce high-crime risks. It enabled path calculation based not just on distance but also on crime risk levels by assigning weights to road segments.
- **Folium (GIS Tool):** This library was used for map visualization. It allows us to plot high-risk crime locations, highlight safe and unsafe paths, and add interactive popups—making the map easy to interpret and informative.

1.5 Project Methodology

The development of this crime-sensitive navigation system follows a systematic, structured and logical approach, ensuring that each step contributed effectively to the final output. This structured approach ensures accuracy, efficiency, and clarity in building and testing the system. The methodology followed in this project is structured into several stages:

- **Data Collection**: The project began with collecting a raw dataset that includes detailed crime info. from various Indian cities. The dataset contained features like city, location, time of occurrence, type of crime, severity score and much more. Understanding this dataset was important, as it formed the base for identifying crime-prone areas and assigning safety scores to different parts of the city.
- **Data Cleaning and Encoding**: Given dataset, often contains missing or irrelevant values. These were handled using standard data cleaning techniques such as removing null entries and correcting data types. Additional meaningful columns were created, such as: "Risk Score" and "Time Category" were derived. These new features added value to the dataset and supported further analysis and mapping tasks.
- **Risk-Based Mapping**: To make the data informative, crime-prone locations were plotted on an interactive map like - 'Leaflet Map' using **Folium** library. Each crime hotspot was visualized on the map using color-coded markers based on severity or risk score of that

crime in that city within India. This helped visually identify zones with a higher concentration of risky incidents, commonly referred to as crime hotspots.
Heatmaps were also generated to provide density-based plotting of the most dangerous crime-prone areas. These visual tools allow easy identification of crime patterns across cities.

- **Graph Creation**: Once the crime hotspots were analyzed, the next step was to build a graph-based model of each city's road network. This was done using **OSMnx** library, which fetches real-world road data from OpenStreetMap and converts it into a graph network structure. In this graph: **nodes** represent road intersections or endpoints, and **edges** represent road segments connecting the nodes for each city. This phase of our system allowed us to apply pathfinding algorithms efficiently.
- **Pathfinding**: For this part **Dijkstra algorithm** was used to determine two kinds of paths: **Shortest Path**: it is based purely on distance, regardless of crime data and **Safest Path**: it considers crime risk as a weight, avoiding high-risk areas irrespective of shortest path. By comparing both paths, individuals could choose whether they want a faster shortest route or a safer one, depending on their personal priorities and use cases.

1.6 Limitations

While designing such kind of system offers an innovative solution, it does also have certain limitations which are listed below:

- **Static Data**: The crime dataset is historical based on past crime incidents and not updated in real-time scenarios. Live or real-time updates would make the system more effective and scalable.
- **Data Availability**: The project relies on publicly available past crime data, which may not be consistently structured or frequently updated across all cities within India.
- **Path Sensitivity**: Some crime-sensitive locations may be very close to commonly used roads, which could cause a pathfinding algorithm to label those roads as high-risk zones even if they are well-monitored, which may result in incorrect output.
- **Route Scope**: Currently, the system supports pedestrian navigation. Expanding to other transport modes (e.g., driving, public transit) would enhance usability.
- **No User Feedback Loop**: The system doesn't currently allow individuals to report or verify risks, which could improve accuracy and performance of the system. This system lacks LLM support which could be responsible for doing this task of user feedback.

1.7 Summary

Chapter 1 begins with the **Introduction (1.1)**, which describes the background and motivation behind building a crime-sensitive navigation system to ensure safer travel in urban areas. It is followed by the **Problem Outline (1.2)**, which highlights the limitations of existing route platforms that do not consider public safety when suggesting travel paths. The **Problem Objectives (1.3)** section describes the specific aims of the project, focusing on risk-based route computation and visualization. The **Overview of Tools and Technologies (1.4)** describes the various libraries and platforms used for data analysis, route mapping, and pathfinding algorithms. This is followed by the **Project Methodology (1.5)**, which explains each step taken from data preprocessing to path generation. The chapter concludes with the **Limitations (1.6)** and a **Summary (1.7)**, which collectively define the project's scope, innovation, and the challenges faced during development of this project.

CHAPTER-2

LITERATURE REVIEW

2.1 Introduction

This part of chapter explains the nature of the literature review in relation to the study. It describes how the review of the current research is beneficial to determine the scenario of crime prediction and safe routing that leads to the selection of methodologies and algorithms. The section also introduces the overall structure of the chapter and provides the reader with a "road map" of what will follow [4].

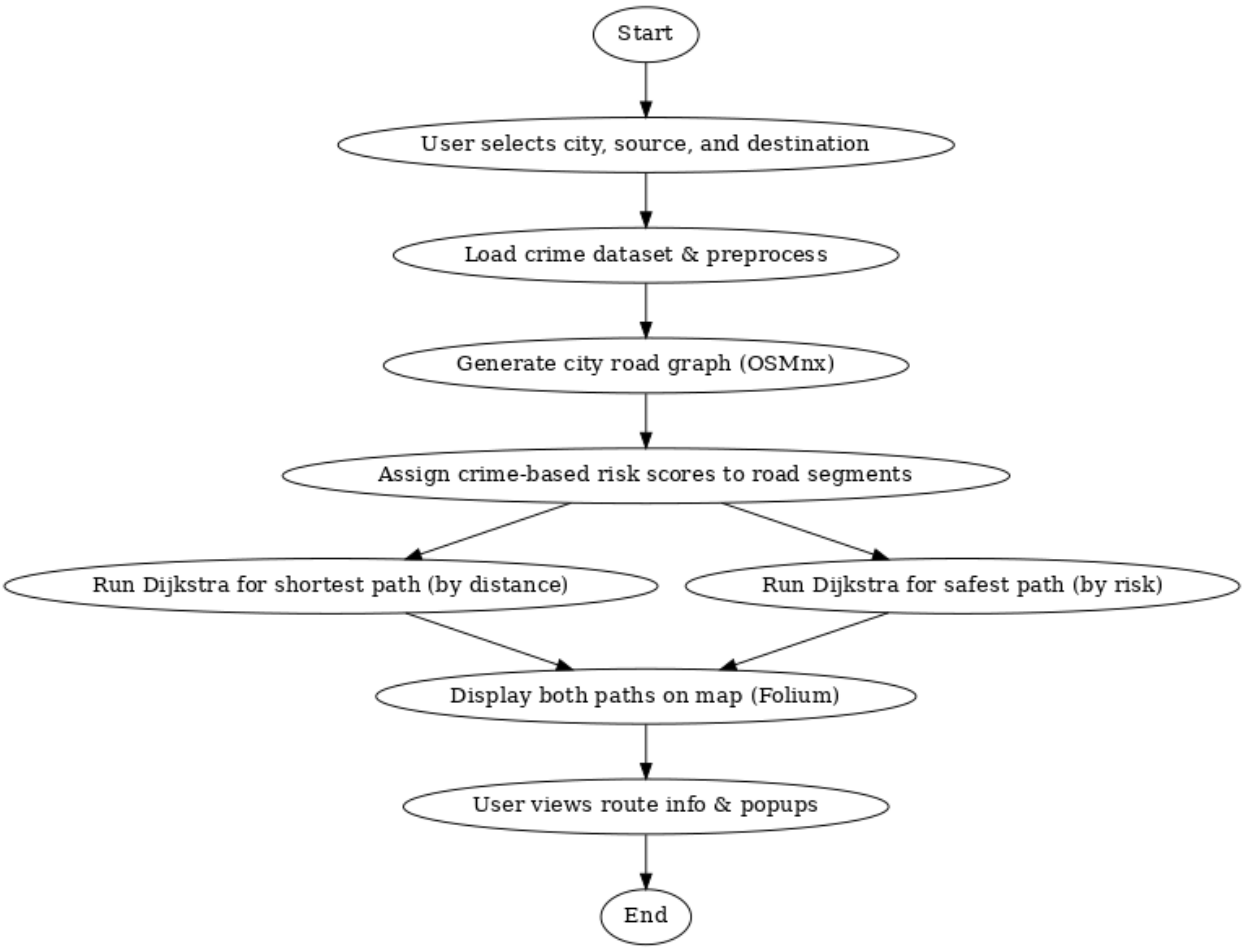


Fig 2.1: Project Workflow [2]

The rising concern over public safety in urban areas has led to a demand for smarter navigation systems that go beyond just suggesting the shortest path. While existing GPS systems and routing services focus more on distance and traffic concerns, they often fail to identify the personal safety of individuals, especially in crime-prone areas or during late hours. This chapter

discusses existing research work on crime-sensitive navigation and identifies the need for integrating crime risk modeling into urban route planning systems [3][4].

2.2 Previous Studies of Crime Prediction and Safest Routing

Several researchers have tried to integrate crime-related data into route recommendation systems to improve users' safety in metropolitan areas. A huge contribution to this area is the study titled **"Route-The Safe: A Robust Model for Safest Route Prediction Using Crime and Accidental Data"** by Shankar et al. [4]. This research focuses on combining past crime and accident data to identify risk and suggest alternative paths that reduce exposure to high crime-prone areas.

The authors used real-world datasets from NYC Open Data, applied **nested K-means clustering algorithm** to detect high-crime sensitive zones based on geographic coordinates like (latitude and longitude). These groups help isolate crime hotspots with greater precision compared to traditional area-based risk mapping. The route safety is determined using a **weighted risk formula**, which considers several parameters like - accident intensity, crime severity, and distance to risky regions. Additionally, the **K-Nearest Neighbors (KNN)** regression algorithm is applied to assign risk scores to potential paths, making the recommendation model more dynamic and adaptable in nature [3][4].

The model integrates its logic with the Google Maps API, allowing individuals to visualize both the shortest and safest routes. This makes it practical and familiar for real-world use. The system displays both route options with color codes mapping and detailed risk metrics, allowing individuals to choose between time efficiency by considering shortest path and personal safety by considering safest path [4].

2. 4 Machine Learning Techniques used in Prediction

This section demonstrates how machine learning algorithms are used in our project to support the classification and analysis of high-risk areas based on static crime data. These models were implemented to determine the safety levels of road segments and assist in route risk scoring. The following machine learning algorithms used in our proposed study are as follows-

Logistic Regression

Logistic Regression was used as a base classification model to predict whether a road network falls into a high-risk, medium-risk, or low-risk category based on risk-scores category. Its dependency makes it ideal for understanding how different features—like severity score, police deployment, or time of day—influence risk classification [1][5].

Support Vector Classifier

Support Vector Classifier was used to handle high-dimensional feature spaces, especially when there's a non-linear boundary between high-risky and safe regions. In our case, this model was used to detect whether certain feature combinations (e.g., time of crime, location density, type of crime) indicate a high-risk area or not [1][6].

XGBoost Classifier

XGBoost, model was used for accurate classification of road risk levels based on severity scores. It is well-suited for structured tabular data with mixed features (numerical and categorical). In our predictive study, it was trained on features like victim age, time category, and severity score etc. [1][7].

LightGBM Classifier

LightGBM was used as an alternative to XGBoost model for faster training on large datasets. It handles categorical features more effectively. This model was effective in learning from the scaled features in our dataset and showed higher accuracy in classifying paths into safe or unsafe regions [1][9].

Multi-Layer Perceptron Classifier

MLP, a type of neural network model, which was tested for classification using non-linear combinations of features. It includes one or more hidden layers and is capable of learning complex relationships in the dataset. In our case, MLP analyzed some dependencies between location, time, and severity variables, offering a deeper pattern recognition model than traditional classifiers.

These types of models are effective in learning from the scaled features in our dataset and showed higher accuracy in classifying paths into safe or unsafe regions for further crime analysis [1][8].

K-Nearest Neighbors

The KNN model was used both for classification and to assign risk scores to road networks based on similarity to known high-crime locations. It classifies a new data point based on the most frequent class among its kth nearest neighbors. In this project, it also played a vital role in predicting risk levels at undefined waypoints by averaging scores of nearby labeled data points [4][11].

K-Means Clustering

K-means clustering was not used for classification but for **unsupervised grouping of crime points into hotspots**. The algorithm grouped similar kinds of crimes based on geographic coordinates like – (latitude and longitude) and risk-based features. These clusters served as the base to determine risk scores for every type of crime in different regions and were key in visualizing crime hotspots on the map [4][10].

2.4 Gap Identified in Literature Review

While the referenced research study represents a solid foundation for crime-sensitive route planning, it also highlights some limitations common in existing navigation systems which are as follows:

- **Data Depth**: Many earlier approaches relied on district-level crime datasets, which lacked precision. Street or city wise-level clustering, as used in the paper, provides a more realistic risk assessment in urban areas.
- **Dynamic Risk Scoring**: Few methods assign static risk levels based on historical data, without adjusting some factors like - time of day, severity scores, and police deployment. The referenced study addresses this partially through weighted scoring but could still benefit from real-time updates.
- **Limited Algorithm Usage**: Most crime-sensitive models do not incorporate advanced graph algorithms like Dijkstra or A* for pathfinding. Instead, they rely on general heuristics algorithms, which may not suggest optimal results for both safety and shortest route navigation.
- **Lack of Deployment Frameworks**: Many existing works remain theoretical and do not provide user-friendly interfaces or web applications. By contrast, the referenced study successfully integrates its logic into an interactive system built on a real-time platform.[3][4]

2.4 Advancements Over Prior Approaches

However, while the referenced study focused primarily on clustering analysis and predictive modeling, our project builds further by:

- Integrating **road network graphs** via OpenStreetMap data using OSMNX and NetworkX library for creating road segments and doing graph analysis.
- Applying pathfinding algorithms like - **Dijkstra's algorithm** for both shortest and safest route recommendations.
- Creating **city-wise visualizations** of crime hotspots using Folium library on Leaflet Maps.
- Introducing a complete end-to-end solution—from raw crime data to a web-based route recommendation interface [optional].

Moreover, while the referenced paper used NYC data, our project adapts the approach to the Indian context, using a wide range dataset of crime reports of past 4 years from multiple Indian cities between 2020 and 2024.

2.5 Summary

This chapter began with an overview of crime-sensitive navigation systems, focusing on the importance of integrating public safety into route planning tasks. It then reviewed earlier referenced studies that utilized clustering algorithms, regression models, and spatial data to map high-risk areas in metropolitan areas. Some algorithms such as K-means, KNN, and risk-based scoring were discussed. The chapter also highlighted limitations like dependency on static data and limited real-time adaptability. These identified gaps helped shape the motivation for developing a more localized, graph-based, and visually interactive route navigation system in the proposed study.

CHAPTER-3

IMPLEMENTATION

3.1 Introduction

This chapter outlines how the proposed system was practically built using real-world crime data and modern computational tools. It covers the integration of data preprocessing, feature engineering, crime-hotspot detection, risk-based path evaluation, and user interface design [optional part].

Each module was implemented in a systematic manner, with a focus on ensuring accurate, real-time safe route recommendations. The chapter also illustrates how machine learning models, graph algorithms, and map-based interfaces were combined to meet the goal of safe urban navigation in urban areas.

3.2 Code Snippet

This section provides a detailed study of the development of our crime-sensitive navigation system. The code for this system was written in Python, primarily within the Google Colab environment. Key aspects of the implementation are explained using code and its output, captured directly from the Colab interface. These snippets illustrate the core methodologies and techniques employed in building the predictive model.

3.2.1 Data Collection

The dataset used is named **crime_dataset_india.csv**, and it was uploaded from Google Drive into the notebook environment. The dataset contains detailed information about crime incidents in various cities across India, with features like: City name, Date of occurrence, Time of occurrence, Type of crime, Time and description of each crime, Crime Code, Crime Domain, Weapon Used and many more. It involves several key steps to load and perform inspection on given raw data some of the steps include -

- **Load the Dataset into a Pandas Data frame** - Read the .csv file from google drive Path and load the file into notebook environment.
- **Initial Data Inspection** - Some basic steps involved before moving on data pre-processing and feature engineering tasks. Those steps are as follows -

Viewing top 10 records for given dataset.

Checking the structure and column types.

Get the shape and summary statistics of the dataset.

Step01 - DATA COLLECTION

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] df = pd.read_csv('/content/drive/MyDrive/CrimeProjectInternship/crime_dataset_india.csv')
```

```
df.head(n=10)
```

	Report Number	Date Reported	Date of Occurrence	Time of Occurrence	City	Crime Code	Crime Description	Victim Age	Victim Gender	Weapon Used	Crime Domain	Police Deployed	Case Closed	Date Case Closed
0	1	02-01-2020 00:00	01-01-2020 00:00	01-01-2020 01:11	Ahmedabad	576	IDENTITY THEFT	16	M	Blunt Object	Violent Crime	13	No	NaN
1	2	01-01-2020 19:00	01-01-2020 01:00	01-01-2020 06:26	Chennai	128	HOMICIDE	37	M	Poison	Other Crime	9	No	NaN
2	3	02-01-2020 06:00	01-01-2020 02:00	01-01-2020 14:30	Ludhiana	271	KIDNAPPING	48	F	Blunt Object	Other Crime	15	No	NaN
3	4	01-01-2020 05:00	01-01-2020 03:00	01-01-2020 14:46	Pune	170	BURGLARY	49	F	Firearm	Other Crime	1	Yes	29-04-2020 05:00
4	5	01-01-2020 21:00	01-01-2020 04:00	01-01-2020 16:51	Pune	421	VANDALISM	30	F	Other	Other Crime	18	Yes	08-01-2020 21:00
5	6	02-01-2020 03:00	01-01-2020 05:00	01-01-2020 17:09	Delhi	442	ASSAULT	16	M	Firearm	Violent Crime	18	Yes	30-03-2020 03:00
6	7	01-01-2020 16:00	01-01-2020 06:00	01-01-2020 14:08	Chennai	172	VEHICLE - STOLEN	64	F	Knife	Violent Crime	13	Yes	24-03-2020 16:00
7	8	02-01-2020 10:00	01-01-2020 07:00	02-01-2020 06:33	Chennai	169	COUNTERFEITING	78	X	Knife	Other Crime	8	No	NaN
8	9	04-01-2020 03:00	01-01-2020 08:00	02-01-2020 06:34	Mumbai	338	EXTORTION	41	X	Blunt Object	Other Crime	1	No	NaN
9	10	03-01-2020	01-01-2020	01-01-2020	Chennai	497	PUBLIC	29	M	Knife	Other	4	No	NaN

```
[ ] df.shape
(40160, 14)

[ ] df.describe()
```

	Report Number	Crime Code	Victim Age	Police Deployed
count	40160.000000	40160.000000	40160.00000	40160.000000
mean	20080.500000	349.360259	44.49126	10.006250
std	11593.337742	144.168205	20.22555	5.467951
min	1.000000	100.000000	10.00000	1.000000
25%	10040.750000	225.000000	27.00000	5.000000
50%	20080.500000	349.000000	44.00000	10.000000
75%	30120.250000	474.000000	62.00000	15.000000
max	40160.000000	599.000000	79.00000	19.000000

Fig 3.1: Code snippet illustrating data loading process

3.2.2 Data Preprocessing

Before any analysis, the crime dataset was cleaned and prepared. Steps involved in data pre-processing tasks -

- **Drop Rows with Missing Values** - Dropped rows where essential columns like '**Time of Occurrence'** or **'Crime Description'** are missing.

```
STEP02 - DATA PRE-PROCESSING

[ ] # Drop rows with missing critical fields
    df = df.dropna(subset=["Time of Occurrence", "Crime Description"])
```

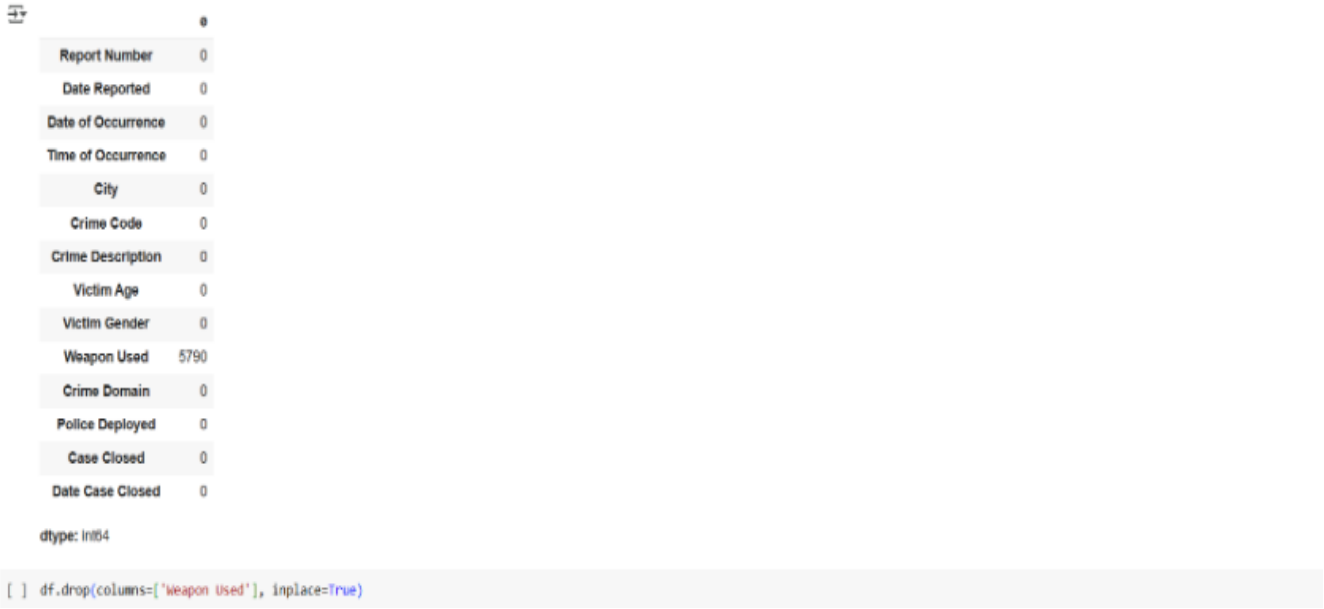
Fig 3.2: Code snippet for dropping rows with missing values

- **Fill Missing Values in Other Columns** – Replace missing values in the **'Date Case Closed'** feature with a placeholder like **'Not Closed'**.

```
# Fill "Date Case Closed" with placeholder
df["Date Case Closed"] = df["Date Case Closed"].fillna("Not Closed")
```

Fig 3.3: Code snippet for filling missing values with random placeholder

- **Dropping Unwanted Columns -** The attribute named **'Weapon Used'** was dropped due to lack of consistent data. Apart from this attribute, there are no attributes in the entire dataset which have inconsistent data.



	0
Report Number	0
Date Reported	0
Date of Occurrence	0
Time of Occurrence	0
City	0
Crime Code	0
Crime Description	0
Victim Age	0
Victim Gender	0
Weapon Used	5790
Crime Domain	0
Police Deployed	0
Case Closed	0
Date Case Closed	0

dtype: int64

```
[ ] df.drop(columns=['Weapon Used'], inplace=True)
```

Fig 3.4: Code snippet for dropping the 'Weapon Used' column

- **Converting Date Related Columns to Proper Format –** The crime dataset has features like - **'Date Reported'**, **'Date of Occurrence'**, and **'Date Case Closed'**. These features were initially stored as strings. We converted them into standard datetime format so that we can easily extract day, month, and year for further analysis.
- **Feature Extraction** – Extracted Year, Month, and Weekday from the **'Date of Occurrence'** attribute from a crime dataset for better trend analysis.

```
[14] # Convert date columns to datetime
     date_cols = ["Date Reported", "Date of Occurrence", "Date Case Closed"]
     for col in date_cols:
         df[col] = pd.to_datetime(df[col], errors='coerce')
```

```
[15] # Extract features from datetime
     df["Year"] = df["Date of Occurrence"].dt.year
     df["Month"] = df["Date of Occurrence"].dt.month
     df["Weekday"] = df["Date of Occurrence"].dt.day_name()
```

Fig 3.5: Code snippet for converting date columns into standard format

- **Categorize Time of Crime Happened -** We used the **'Time of Occurrence'** attribute to find out whether the crime happened during the day or night (If the crime happened between 8 PM to 5 AM, we labeled it as **'night'** otherwise, it was labeled as '**day**'). This helped us analyze when most crimes occur.

```
# Convert "Time of Occurrence" to "Time Category"
def func(row):
    try:
        time_str = row.split()[1]
        hour = int(time_str.split(":")[0])
        return "Night" if (hour >= 20 or hour < 5) else "Day"
    except:
        return "Unknown"
df["Time Category"] = df["Time of Occurrence"].apply(func)
```

Fig 3.6: Code snippet for categorizing whether crime occurred in day/night

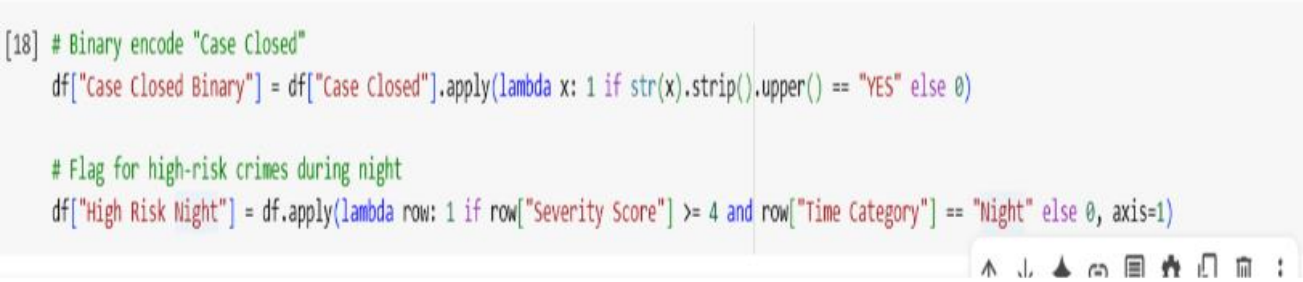
- **Assigning Severity Scores to each Crimes** - Assigned custom severity scores to different crime types for a given crime dataset. This helps us do further analysis, like identifying which cities have more serious crimes.

```
[17] # Assign severity scores
     severity_mapping = {
         "HOMICIDE": 5,
         "ASSAULT": 4,
         "KIDNAPPING": 4,
         "BURGLARY": 3,
         "VANDALISM": 2,
         "IDENTITY THEFT": 2,
         "FRAUD": 3,
         "THEFT": 2,
         "ROBBERY": 4,
         "OTHER": 1
     }
     df["Severity Score"] = df["Crime Description"].apply(lambda x: severity_mapping.get(x.upper(), 1))
```

Fig 3.7: Code snippet for assigning custom severity scores to each type of crimes

- **Binary Encode "Case Closed" Attribute –** For a crime dataset, there is a column named **'Case Closed'** which has values like – either **'Yes'** or **'No'** [binary classification]. To make this column easier to use in further analysis, we converted it into numbers like – 0 or 1 (if the case was closed **'Yes'**, we set the value to **1**, otherwise set the value to **0**). This type of encoding technique is referred to as **Binary encoding**.
- **Flag High-Risk Crimes that occurred at Night -** We wanted to identify cases that were both serious and occurred at night, as they are usually more dangerous. So, we created

a new attribute called '**High-Risk Crimes'** (If the crime's severity score was 4 or greater than 4 (like Robbery or Homicide) and it happened at night, we set the value to **1**, Otherwise, we set it to **0**).



```
[18] # Binary encode "Case Closed"
     df["Case Closed Binary"] = df["Case Closed"].apply(lambda x: 1 if str(x).strip().upper() == "YES" else 0)

     # Flag for high-risk crimes during night
     df["High Risk Night"] = df.apply(lambda row: 1 if row["Severity Score"] >= 4 and row["Time Category"] == "Night" else 0, axis=1)
```

Fig 3.8: Code snippet for performing encoding on 'Case Closed' attribute

- **Create a Target Custom "Risk Score" column** - To analyze the overall impact of each crime in each city, we created a new column called **"Risk Score"**. This score is calculated using a weighted formula based on: **Victim Age** – weight = 0.1, **Police Deployed** – weight = 0.3, **Severity score** – weight = 0.6, the formula used was -

```
# Create Risk Score (scaled)
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df[["Victim Age", "Police Deployed", "Severity Score"]] = scaler.fit_transform(
    df[["Victim Age", "Police Deployed", "Severity Score"]]
)

df["Risk Score"] = df["Victim Age"] * 0.1 + df["Police Deployed"] * 0.3 + df["Severity Score"] * 0.6
```

Fig 3.9: Code snippet for creating new feature 'risk score' for analysis

- **Extract geological data -** Now in this step, our raw crime dataset doesn't contain geo-logical data like **latitude and longitude** for each city. So, we kindly extracted the geo-logical data from **'Indiageocordinates.com'** as **'in.csv file'** for each city to analyze crime hotspots we need geo-logical data.

  Load the geo-logical data – First step, is to load the geo-logical data from **'Indiageocordinates.com'** into our notebook environment as **'in.csv file'.**

```
df2 = pd.read_csv("/content/in.csv")
df2.head(n=10)
```

	city	lat	lng	country	iso2	admin_name	capital	population	population_proper
0	Delhi	28.6100	77.2300	India	IN	Delhi	admin	32226000	16753235
1	Mumbai	19.0761	72.8775	India	IN	Mahārāshtra	admin	24973000	12478447
2	Kolkāta	22.5675	88.3700	India	IN	West Bengal	admin	18502000	4496694
3	Bangalore	12.9789	77.5917	India	IN	Karnātaka	admin	15386000	8443675
4	Chennai	13.0825	80.2750	India	IN	Tamil Nādu	admin	12395000	6727000
5	Hyderābād	17.3617	78.4747	India	IN	Telangāna	admin	10494000	6993262
6	Pune	18.5203	73.8567	India	IN	Mahārāshtra	NaN	8231000	3124458
7	Ahmedabad	23.0225	72.5714	India	IN	Gujarat	minor	8009000	5570585
8	Sūrat	21.1702	72.8311	India	IN	Gujarāt	NaN	6538000	4466826
9	Lucknow	26.8500	80.9500	India	IN	Uttar Pradesh	admin	3382000	3382000

Fig 3.10: Code snippet for loading geo-logical data from external source

Drop Unwanted Attributes - As per analysis we need some important features like – latitude, longitude and City so now by carefully observing some of the features that are irrelevant for analyzing crime patterns should be dropped to reduce the complexity in given data. Some of the unwanted attributes are like – country, iso2, admin-name, capital, population etc.

```
[33] df2.isnull().sum()
```

	0
city	0
lat	0
lng	0
country	0
iso2	0
admin_name	0
capital	143
population	0
population_proper	0

dtype: int64

```
df2.drop(columns=['country','iso2','admin_name','capital','population','population_proper'], inplace=True)
```

Fig 3.11: Code snippet for removing unwanted attribute from geo-logical data

Now merge this data with our original dataset to create one final dataset with all the relevant fields for analysis. We will export **'crimedf.csv'** as a final dataset for further crime analysis.

```
[36] # Clean city names for matching
     df2["City_clean"] = df2["city"].str.strip().str.upper()
     df["City_clean"] = df["City"].str.strip().str.upper()

[37] # Merge
     final_df = pd.merge(df, df2, on="City_clean", how="left")

[38] crime_df = pd.read_csv("/content/final_df.csv")

     crime_df = crime_df.dropna(subset=["lat", "lng"])
     crime_df.head(n=3)
```

	Report Number	Date Reported	Date of Occurrence	Time of Occurrence	City	Crime Code	Crime Description	Victim Age	Victim Gender	Crime Domain	...	Severity Score	Case Closed Binary	High Risk Night	Risk Score	Cit
0	1	2020-02-01 00:00:00	2020-01-01 00:00:00	01-01-2020 01:11	Ahmedabad	576	IDENTITY THEFT	0.086957	M	Violent Crime	...	0.25	0	0	0.358696	AHMI
1	2	2020-01-01 19:00:00	2020-01-01 01:00:00	01-01-2020 06:26	Chennai	128	HOMICIDE	0.391304	M	Other Crime	...	1.00	0	0	0.772464	C
3	4	2020-01-01 05:00:00	2020-01-01 03:00:00	01-01-2020 14:46	Pune	170	BURGLARY	0.565217	F	Other Crime	...	0.50	1	0	0.356522	

3 rows × 27 columns

```
[41] crime_df.shape
```

(25761, 25)

```
[42] # export crimedf to csv
     crime_df.to_csv("crime_df.csv", index=False)
```

Fig 3.12: Code snippet for merging and exporting geo-logical data as final dataset

Normalize Risk Score Attribute - Scaling values between 0 and 1 range using **'Min-Max-Scaler' Technique**. Adding a new feature named **'normalized risk'** into our final dataset.

```
[45] from sklearn.preprocessing import MinMaxScaler
     scaler = MinMaxScaler()
     crime_df['normalized_risk'] = scaler.fit_transform(crime_df[['Risk Score']])
```

Fig 3.13: Code snippet for normalizing 'risk score' feature using Min-Max Scaler

Risk Level Classification - To make the analysis more understandable, we assigned risk level categories (Green, Yellow, Red) for each crime based on its normalized risk score. This helped

us logically to classify crimes into different levels of severity. Then, this function was applied to the entire 'normalized risk' column in the Data Frame to create a new column named 'risk levels'.

```
def func(score):
    if score >= 0.2:
        return 'red'
    elif score > 0.1 and score < 0.2:
        return 'yellow'
    else:
        return 'green'

crime_df['risk_levels'] = crime_df['normalized_risk'].apply(func)
```

Fig 3.14: Code snippet for classifying risk levels into three categories

3.2.3 Model Training and Evaluation

Now in this section, multiple machine learning models are trained on **80-20 hold-out ratio** using numerical features as **X value** out of all feature present in the given final dataset except target attribute and **y value** would be the target variable which is **'risk score'**. Trained models like – **Logistic Regression, Support Vector Classifier, XG-Boost Classifier, Light-GBM Classifier, MLP Classifier** on a given final dataset. All models have been compared with respect to each other based on mean accuracy scores.

To prepare the data for further analysis, we categorized the 'risk score' into defined risk levels and selected relevant numerical features.

- **Risk Level Categorization using Binning Method** - Instead of using a custom function, we used the **pd.cut() function** to divide the continuous Risk Score values into three categories - (Green: Risk Score between 0.0 to 0.1, Yellow: Risk Score between 0.1 to 0.2, Red: Risk Score between 0.2 to 1.0). This created a new column called 'risk levels' with categorical values (Green, Yellow, Red).
- **Handling Missing Values** - To ensure data quality, we removed any rows where the 'risk levels' column has missing (Nan) values. This step is important to avoid issues during analysis or model training, as missing risk levels would make classification impossible for those records.

```
bins = [0.0, 0.1, 0.2, 1.0]
labels = ['Green', 'Yellow', 'Red']
crime_df['risk_levels'] = pd.cut(crime_df['Risk Score'], bins=bins, labels=labels, include_lowest=True)

X = crime_df.select_dtypes(include=['int64', 'float64']).drop(columns=['Risk Score'])
y = crime_df['risk_levels']

crime_df = crime_df.dropna(subset=['risk_levels'])
```

Fig 3.15: Code snippet for categorizing risk levels into different bins

- **Model training -** In this step, after preprocessing and preparing the crime dataset, multiple ML models were applied to classify crime records into risk levels (Green, Yellow, Red) based on numerical features.
  Data Splitting – Given dataset was divided into training and testing sets using the 'train-test-split' method. 80% of the data was used for training, and 20% was kept for testing. [used 80-20 hold-out ratio]
  Data Scaling – All numerical features are scaled or normalized using **'Standard Scaler'** so that all the features have the same range (mean = 0, standard deviation = 1). This helps improve model performance and efficiency.
  Model Setup - Several classification machine learning models were used like - Logistic Regression, Support Vector Classifier, XG-Boost Classifier, Light-GBM Classifier, MLP Classifier etc. because our target feature **'risk-levels'** is categorical. Also, here we have used one parameter named **'class-weight = 'balanced''** while training every model is used to handle any imbalance in given data in the risk level classes.

  Model Evaluation using K-Fold Cross Validation - To evaluate the model's performance, we used **K-Fold Cross Validation (with 5 splits)**:

  The dataset is divided into 5 parts.

  In each iteration, 4 parts are used for training and 1 part for testing.

  This process repeats 5 times, and the average mean accuracy scores are calculated for each machine learning model.

  Final Mean Accuracy Result - The average mean accuracy for every model is calculated and we observed, that the highest mean accuracy score was **99% achieved by XG-Boost Classifier** and **Light-GBM Classifier** and the lowest mean accuracy score we got **96% by Support Vector Classifier** on 80-20 hold-out ratio.

Model01 - LOGISTIC REGRESSION

```
# Scale numerical features
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 5-Fold Cross Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
LR = LogisticRegression(max_iter=1000,class_weight='balanced')
LR_scores = cross_val_score(LR, X_scaled, y, cv=kf, scoring='accuracy')

{
    'Model': 'Logistic Regression',
    'Accuracy Mean': np.mean(LR_scores),
}
```

```
{'Model': 'Logistic Regression',
 'Accuracy Mean': np.float64(0.9844726749659186)}
```

Fig 3.16: Code snippet for training LR model for 80-20 hold-out ratio

Model02 - SUPPORT VECTOR CLASSIFIER

```
# Scale numerical features
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 5 Fold Cross Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
Svc = SVC(class_weight='balanced',probability=True)
Svc_scores = cross_val_score(Svc, X_scaled, y, cv=kf, scoring='accuracy')

{
    'Model': 'Support Vector Classifier',
    'Accuracy Mean': np.mean(Svc_scores)
}
```

```
{'Model': 'Support Vector Classifier',
 'Accuracy Mean': np.float64(0.9671596205792 2)}
```

Fig 3.17: Code snippet for training SVC model for 80-20 hold-out ratio

MODEL03 - XGBOOST CLASSIFIER

+ Code + Text

```
# Scale numerical features
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score, KFold
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

le = LabelEncoder()
y_encoded = le.fit_transform(y)

# 5-Fold Cross Validation
kf = KFold(n_splits=5, shuffle=True)
XGB = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss',class_weight='balanced')
XGB_scores = cross_val_score(XGB, X_scaled, y_encoded, cv=kf, scoring='accuracy')

{
    'Model': 'XGBOOST Classifier',
    'Accuracy Mean': np.mean(XGB_scores)
}
```

```
{'Model': 'XGBOOST Classifier',
 'Accuracy Mean': np.float64(0.9997670807453416)}
```

Fig 3.18: Code snippet for training XGB model for 80-20 hold-out ratio

MODEL04 - LIGHTGBM CLASSIFIER

```
# Scale numerical features
from lightgbm import LGBMClassifier
from sklearn.model_selection import cross_val_score, KFold
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 5-Fold Cross Validation
kf = KFold(n_splits=5, shuffle=True)
LGBM = LGBMClassifier(random_state=42,class_weight='balanced')
LGBM_scores = cross_val_score(LGBM, X_scaled, y, cv=kf, scoring='accuracy')

{
    'Model': 'LightGBM Classifier',
    'Accuracy Mean': np.mean(LGBM_scores)
}
```

Fig 3.19: Code snippet for training LightGBM model for 80-20 hold-out ratio

MODEL05 - MLP CLASSIFIER

```
[ ] # Scale numerical features
    from sklearn.neural_network import MLPClassifier
    from sklearn.model_selection import cross_val_score, KFold
    from sklearn.preprocessing import StandardScaler
    from sklearn.metrics import accuracy_score

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # 5-Fold Cross Validation
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    MLP = MLPClassifier(max_iter=15)
    MLP_scores = cross_val_score(MLP, X_scaled, y, cv=kf, scoring='accuracy')

    {
        'Model': 'MLP Classifier',
        'Accuracy Mean': np.mean(MLP_scores)
    }

→ {'Model': 'MLP Classifier', 'Accuracy Mean': np.float64(0.9885096934427633)}
```

Fig 3.20: Code snippet for training MLP model for 80-20 hold-out ratio

3.2.4 Analyze Crime Hotspot using Folium (GIS tool)

In this section, will better understand and visualize how crime risk is spread widely across different cities within India. We created crime heatmaps. These plots help in identifying crime-prone areas or hotspots — cities where crimes with higher risk levels are concentrated. This kind of analysis provides an intuitive, interactive and easy-to-read view of crime concentration by risk level across Indian cities. Cities like **Delhi, Mumbai, or Bangalore** may appear as hotspots if they have more red or yellow crimes. Cities with mostly green crimes may be considered relatively safer.

To create interactive maps that show crime locations and risk levels, we used a Python library called **Folium [interactive GIS tool]**.

- **Install Folium Python Library using pip command.**
- **Extracting Unique City Names** - To create city-wise visualizations (like heatmaps), we first need a list of all the unique cities present in our crime dataset. Now list out all unique cities present in a given dataset in order to create crime heatmaps. This list helps us:
  Loop over each city to generate individual crime heatmaps or visualizations.
  Analyze and compare crime patterns across different cities.

```
# Get list of unique cities
cities = crime_df['City'].unique()
cities
```

```
array(['Ahmedabad', 'Chennai', 'Pune', 'Delhi', 'Mumbai', 'Bangalore',
       'Jaipur', 'Lucknow', 'Patna', 'Kanpur', 'Meerut', 'Indore'],
      dtype=object)
```

Fig 3.21: Code snippet for extracting unique cities

- **Counting Risk Level Categories** - After risk level is assigned (Red, Yellow, Green) to each crime based on its calculated risk score, we used this code to count how many crimes fall into each category. It gives a quick summary to know which type of risk is most common in the dataset. In this dataset, many crimes fall into the red category (high risk), followed by the yellow (moderate risk) and green category (low risk).

```
crime_df['risk_levels'].value_counts()
```

risk_levels	count
Red	17674
Yellow	5190
Green	2897

dtype: int64

Fig 3.22: Code snippet for counting Risk Level categories

- **Creating Crime Heatmap Visualization for Each City** - In this step, we used the Folium library to create interactive heatmaps that show crime density and risk scores across different cities. These heatmaps help in visualizing hotspots of crime in different cities. The darker/redder areas indicate higher risk based on severity, victim info, and police deployment [risk levels].

```python
import folium
from folium.plugins import HeatMap
from IPython.display import display
import numpy as np

city_heatmaps = {}

for city in cities:
    city_data = crime_df[crime_df['City'] == city].dropna(subset=['lat', 'lng', 'Risk Score'])
    city_data['lat'] += np.random.normal(0, 0.001, size=len(city_data))
    city_data['lng'] += np.random.normal(0, 0.001, size=len(city_data))

    # Create map centered on city
    m = folium.Map(location=[city_data['lat'].mean(), city_data['lng'].mean()], zoom_start=12)

    # Add heatmap layer
    heat_data = city_data[['lat', 'lng', 'Risk Score']].values.tolist()
    HeatMap(heat_data, radius=15, blur=10).add_to(m)

    for _, row in city_data.iterrows():
        popup_text = f"""
        <b>Crime:</b> {row['Crime Description']}<br>
        <b>Time:</b> {row['Time of Occurrence']}<br>
        <b>Lat, Lng:</b> {round(row['lat'], 4)}, {round(row['lng'], 4)}<br>
        <b>Risk Score:</b> {round(row['Risk Score'], 2)}
        """
        folium.CircleMarker(
            location=[row['lat'], row['lng']],
            radius=4,
            color='red',
            fill=True,
            fill_color='red',
            fill_opacity=0.6,
            popup=folium.Popup(popup_text, max_width=300)
        ).add_to(m)

    city_heatmaps[city] = m
```

Fig 3.23: Code snippet for visualizing crime heatmaps using Folium

3.2.5 Creating the Road Network using OSMnx library

In this step, we used OpenStreetMap (OSM) data along with OSMNX and Network-X libraries to generate road network graphs for each city based on identified crime hotspots. This helps visualize the city's road layout and understand how crimes are distributed around the road network. This part of the project helps us visually identify danger zones in a city and prepare for the next step, which is finding the safest and shortest routes using Dijkstra's algorithm. Some steps involved in this to create road network data -

- **Importing Required Libraries** – Used OSMnx to fetch and create road network data for cities and NetworkX was used to handle graph operations.
- **Filtering Coordinates** - From the cleaned and preprocessed dataset, we selected crime locations (latitude and longitude) and risk scores for each city. These coordinates represent the hotspots where crimes frequently occur. If no data is found for the city, just skip it.
- **Generating Road Network** - For each city, we extracted the walkable road network data using OSMnx library from OpenStreetMap (OSM). We use the OpenStreetMap-based OSMNX library to create a walkable road network graph of the city. This helps us analyze how individuals move within the city and allows us to later apply pathfinding algorithms like Dijkstra's algorithm.
- **Highlight High-Risk Crime Hotspots** - High-risk crime hotspots are defined as those with a risk score of 0.2 or more (this threshold can be changed). These locations are plotted as red circles on the map. Each circle shows a popup message with the risk score when clicked, helping us understand the intensity of risk in that area within each city.

```
import osmnx as ox
import folium
import numpy as np

def create_city_network_data(city_name, crime_data, high_risk_threshold=0.2):
    # Filter data for city
    city_crime_df = crime_df[crime_df['City'] == city_name].dropna(subset=['lat', 'lng', 'Risk Score'])
    if city_crime_df.empty:
        print(f"No data for {city_name}")
        return None, None, None

    # Generate road graph
    G = ox.graph_from_place(f"{city_name}, India", network_type='walk')

    # Create folium map centered on city
    lat_center = city_crime_df['lat'].mean()
    lng_center = city_crime_df['lng'].mean()
    fmap = folium.Map(location=[lat_center, lng_center], zoom_start=13, tiles='CartoDB dark_matter')

    # Plot high-risk crime hotspots as red circles
    high_risk_crimes = city_crime_df[city_crime_df['Risk Score'] >= high_risk_threshold]
    for _, row in high_risk_crimes.iterrows():
        folium.CircleMarker(
            location=(row['lat'], row['lng']),
            radius=5,
            color='red',
            fill=True,
            fill_opacity=0.7,
            popup=f"Risk Score: {row['Risk Score']:.2f}"
        ).add_to(fmap)
```

Fig 3.24: Code snippet for creating the road network using OSMnx library

3.2.6 Finding Shortest and Safest Path using Dijkstra's Algorithm

In this section, we use the Dijkstra algorithm to find two types of routes within each city's road network - the shortest path based purely on physical distance and the safest path based on crime risk levels. This stage enables individuals to prioritize between the fastest route or the safest route depending upon the scenarios, based on the crime distribution in a city. It makes our navigation system smarter and more helpful in real-life, especially in urban or metropolitan areas with varying safety levels.

- Initially, we used the first and last crime locations in the given crime dataset as default source and destination points to identify shortest and safest path to reduce crime rates within each city. These can be customized if needed. These coordinates are converted into the nearest nodes on the city's road network graph (created earlier using OSMnx).

```
[ ] import networkx as nx
    from folium import PolyLine
```

```
[ ] def dijkstra_algorithm(G, fmap, crime_data, start_points=None, end_points=None, city="Unknown", high_risk_threshold=0.2):
        if crime_df.empty:
            print(f"No crime data to process routes in {city}")
            return fmap

        # Define source and destination nodes
        if start_points is None:
            start_points = (crime_df.iloc[0]['lat'], crime_df.iloc[0]['lng'])
        if end_points is None:
            end_points = (crime_df.iloc[-1]['lat'], crime_df.iloc[-1]['lng'])

        start_node = ox.distance.nearest_nodes(G, start_points[1], start_points[0])
        end_node = ox.distance.nearest_nodes(G, end_points[1], end_points[0])
```

2. **Edge Risk Weighting** - For every edge (road segment) in the graph, we check if there are any **high-risk crime spots** nearby. If a road passes through or near a crime hotspot (within a small radius), we **increase its risk weight**.

```
        # Assign weights to edges based on crime proximity
        for u, v, k, data in G.edges(keys=True, data=True):
            data['length'] = data.get('length', 1)
            midpoint = ((G.nodes[u]['y'] + G.nodes[v]['y']) / 2, (G.nodes[u]['x'] + G.nodes[v]['x']) / 2)
            nearby_crimes = crime_df[
                ((crime_df['lat'] - midpoint[0]).abs() < 0.002) &
                ((crime_df['lng'] - midpoint[1]).abs() < 0.002)
            ]
            data['risk'] = 5 if not nearby_crimes.empty and nearby_crimes['Risk Score'].mean() > high_risk_threshold else 1
```

Fig 3.25: Code snippet for defining source to destination points for pathfinding

- **Assigning Risk Weight to Each Segment** - For every edge (road segment) in the road network graph, we check if there are any high-risk crime hotspots nearby or not. If a road passes through or near a crime hotspot (within a small radius), we increase its risk weight.

```
# Assign weights to edges based on crime proximity
for u, v, k, data in G.edges(keys=True, data=True):
    data['length'] = data.get('length', 1)
    midpoint = ((G.nodes[u]['y'] + G.nodes[v]['y']) / 2, (G.nodes[u]['x'] + G.nodes[v]['x']) / 2)
    nearby_crimes = crime_df[
        ((crime_df['lat'] - midpoint[0]).abs() < 0.002) &
        ((crime_df['lng'] - midpoint[1]).abs() < 0.002)
    ]
    data['risk'] = 5 if not nearby_crimes.empty and nearby_crimes['Risk Score'].mean() > high_risk_threshold else 1
```

Fig 3.26: Code snippet for assigning weights to each road segment

- **Determining Shortest Path** - The popular Dijkstra algorithm uses distance (length) as weight to find the shortest path with respect to high-crime zones. The resulting path is shown as a dashed red line on the map. Source and Destination points are mentioned with blue and purple markers. A popup message shows the total distance of this path in meters.

```
# Calculate shortest path using Dijkstra's algorithm
try:
    shortest_path = nx.shortest_path(G, start_node, end_node, weight='length')
    shortest_distance = nx.shortest_path_length(G, start_node, end_node, weight='length')
    points_short = [(G.nodes[n]['y'], G.nodes[n]['x']) for n in shortest_path]
    PolyLine(points_short, color='red', dash_array='5,5', weight=3, popup=f"Shortest Path: {shortest_distance:.2f} meters").add_to(fmap)
    folium.Marker(points_short[0], icon=folium.Icon(color='blue'),
                  popup='Start Point (Shortest)').add_to(fmap)
    folium.Marker(points_short[-1], icon=folium.Icon(color='purple'),
                  popup='End Point (Shortest)').add_to(fmap)
except Exception as e:
    print(f"Could not compute shortest path in {city}: {e}")
```

Fig 3.27: Code snippet for identifying shortest path using Dijkstra's algorithm

- **Determining Safest Path -** The same algorithm is applied again, but this time using our custom risk score as the weight. The resulting low-risk path is shown via a green line on the folium map. Similar markers and popups message are added for visualization. The safest path may be longer in distance but avoids areas with high crime risk.

```
    # Add safest path (based on risk)
    try:
        safest_path = nx.shortest_path(G, start_node, end_node, weight='risk')
        safest_distance = nx.shortest_path_length(G, start_node, end_node, weight='risk')
        points_safe = [(G.nodes[n]['y'], G.nodes[n]['x']) for n in safest_path]
        PolyLine(points_safe, color='green', weight=6, popup=f"Safest Path: {safest_distance:.2f} units").add_to(fmap)
        folium.Marker(points_safe[0], icon=folium.Icon(color='blue'),
                      popup='Start Point (Safest)').add_to(fmap)
        folium.Marker(points_safe[-1], icon=folium.Icon(color='purple'),
                      popup='End Point (Shortest)').add_to(fmap)
    except Exception as e:
        print(f"Could not compute safest path in {city}: {e}")

    return fmap
```

Fig 3.28: Code snippet for identifying safest path using Dijkstra's algorithm

3.2.7 Summary

This section has transformed the conceptual framework of the project into a functional navigation system that supports safer urban travel. The procedure began with preprocessing real-world crime data, followed by creating visualizations to highlight identify high-risk zones within Indian cities. Using tools like Folium, OSMnx, and NetworkX, interactive maps and road networks were developed to analyze city layouts and crime distributions.

Machine learning models were trained to classify areas based on risk levels, while graph algorithms like Dijkstra's were applied to determine both shortest and safest routes between selected locations. These paths were then displayed on interactive maps with informative markers and risk-based color codes. Overall, this section combines several techniques to meet the project's objective of enabling safer travel within cities.

CHAPTER-4

RESULTS AND DISCUSSION

4.1 Dataset Description

The dataset used in this project consists of recorded crime incidents across multiple Indian cities. It was taken from an open Kaggle repository titled *Indian Crimes Dataset*. The dataset highlights various dimensions of each incident, allowing for detailed spatial and crime risk analysis.

It contains over **40,000+ rows**, with each row representing a unique crime event. The dataset includes important features such as:

Table 4.1: Dataset Description

Feature Name	Description
City	Name of the city where the crime was reported
Crime Description	Type or category of the crime (e.g., theft, assault)
Time of Occurrence	Approximate time or period when the crime occurred
Victim Age	Age of the victim involved in the reported incident
Severity Score	A numerical score assigned to indicate the seriousness of the incident
Police Deployed	Count of law enforcement personnel assigned or recorded for that case
Latitude and Longitude	Geographical coordinates of the incident location
Case Closed	Indicates whether the case was resolved ("Yes" or "No")

Some additional derived features such as **Time Category**, **Case Closed Binary**, **High Risk at Night**, and **Risk Score** were created during the preprocessing steps to make risk-based route decisions by individuals. These enhancements allowed for a more accurate classification of safe and unsafe zones across the multiple Indian cities.

This dataset formed the base of the project's analysis and was used to visualize crime hotspots, train classification models, and generate safe urban navigation routes.

4.2 Evaluation Metrices

To view the performance of several machine learning models used in risk classification and route safety prediction, standard mean accuracy metrics were applied. These metrics help in understanding how accurately the models classify between high-risk and low-risk areas.

Mean Accuracy Score

Evaluation Metric used in this project is **mean accuracy**, calculated through **5-fold cross-validation**. Accuracy refers to the number of correct predictions made by a classification model out of the total number of predictions. In this context, it reflects how well each model classifies a location or road segment as safe, moderate-risk, or high-risk.

Formula used – Accuracy = No. of Correct Predictions / Total No. of Predictions

In **K-Fold Cross-Validation**, the dataset is split into *K* parts (folds). Each time, one-fold is used for testing, and the others are used for training. This process is repeated *K* times, and the accuracy scores are averaged:

$$\text{Mean Accuracy} = \frac{1}{K}\sum_{i=1}^{K}\text{Accuracy}_i$$

Using mean accuracy from cross-validation provides a more reliable estimate of real-world performance, especially when dealing with geographic data where variation exists across different cities or zones.

4.3 Results and Analysis

4.3.1 Initial plots based on EDA

Various plots were created using Seaborn and Matplotlib library to find patterns and trends in crime dataset:

- **Number of Crimes in Day vs Night** – This figure shows a count plot to compare how many crimes occurred during the day versus night. In this plot we compared crime counts between day and night.

```
# Time Category
sns.set(style="darkgrid")
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x="Time Category", palette="coolwarm")
plt.title("Crime Count: Day vs Night")
plt.xlabel("Time of Day")
plt.ylabel("Number of Crimes")
plt.tight_layout()
plt.show()
```

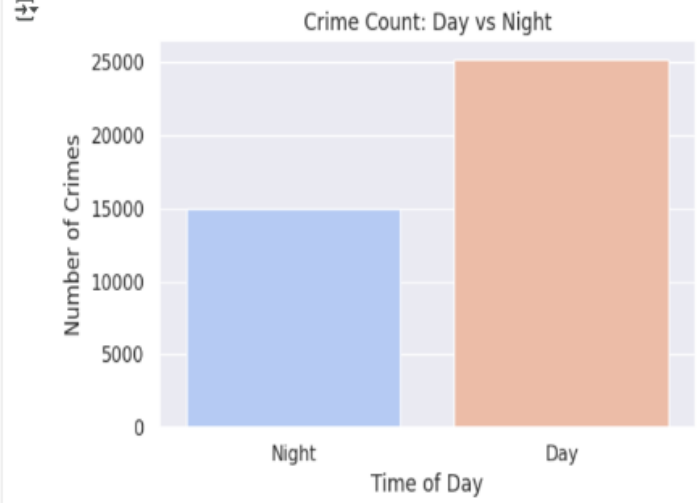


Fig 4.1: Comparison of number of crimes happened in day vs night

- **Crime Count by Crime Type (Top 10 Crimes)** - This figure shows the most common types of crimes using a horizontal bar plot. This plot shows the top 10 most frequent crimes that occurred within each city.

```
# Top 10 Crime Types
plt.figure(figsize=(10, 5))
top_crimes = df["Crime Description"].value_counts().head(10)
sns.barplot(x=top_crimes.values, y=top_crimes.index, palette="viridis")
plt.title("Top 10 Crime Types")
plt.xlabel("Number of Cases")
plt.ylabel("Crime Type")
plt.tight_layout()
plt.show()
```

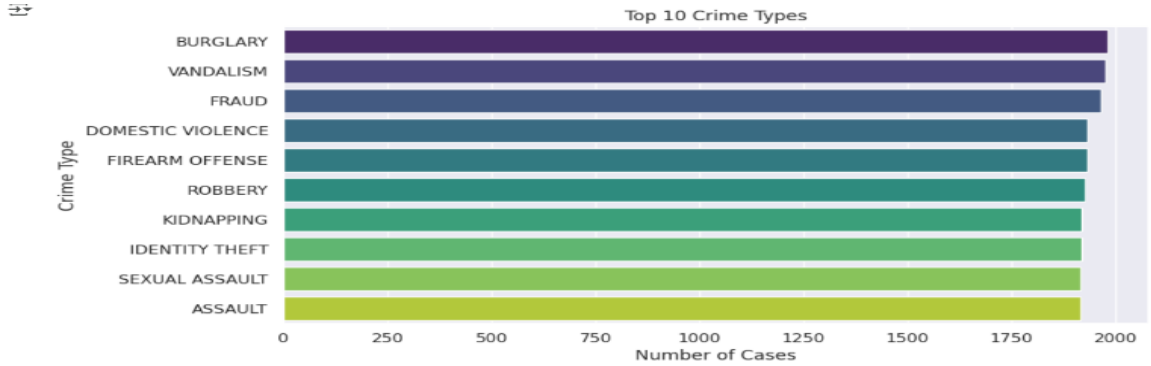


Fig 4.2: Visualizing Crime Count by Crime Type (Top 10 Crimes)

- **Number of Crimes by Day of the Week –** This figure shows how crimes are spread across days of the week using a count plot. This plot shows which day has the most or least number of crimes.

```
[29] # Crime Frequency by Weekday
     plt.figure(figsize=(8, 4))
     sns.countplot(data=df, x="Weekday", order=[
         "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"],
         palette="Set2")
     plt.title("Crimes by Day of the Week")
     plt.xlabel("Weekday")
     plt.ylabel("Count")
     plt.xticks(rotation=45)
     plt.tight_layout()
     plt.show()
```

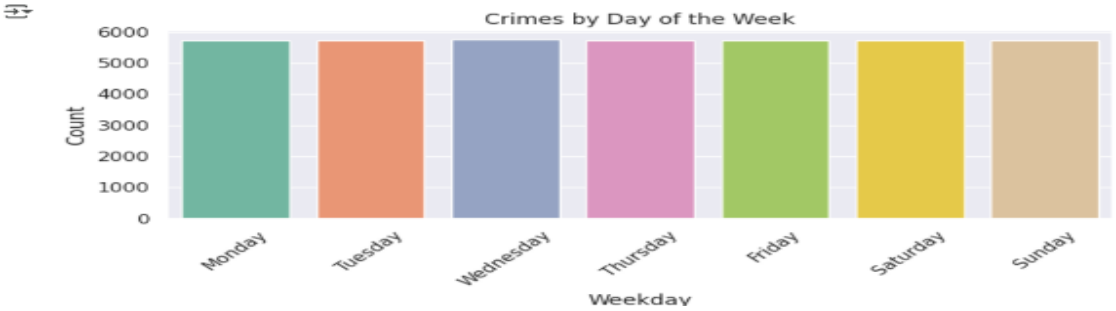


Fig 4.3: Visualizing Number of Crimes by Day of the Week

- **High-Risk Crimes That Occurred at Night** – In this figure bar plot is used to show how many crimes were classified as **'high-risk at night'**, based on our custom logic (severity + time of occurrence). This plot shows a few high-risk crimes that happened specifically during the night.

```
# Top 10 Cities with High Risk Night Crimes
plt.figure(figsize=(10, 6))
top_cities_hrn = df[df["High Risk Night"] == 1]["City"].value_counts().head(10)
sns.barplot(x=top_cities_hrn.values, y=top_cities_hrn.index, palette="rocket")
plt.title("Top 10 Cities with High Risk Crimes at Night")
plt.xlabel("Number of High Risk Night Crimes")
plt.ylabel("City")
plt.tight_layout()
plt.show()
```

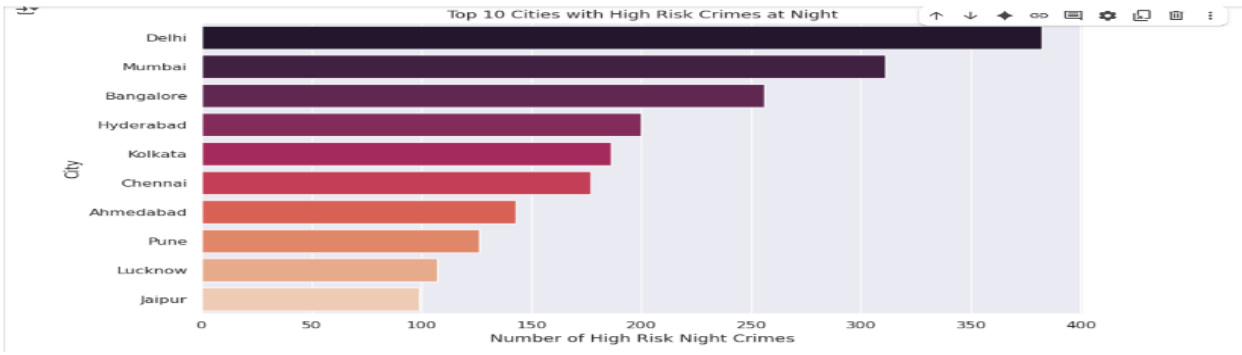


Fig 4.4: Visualizing High-Risk Crimes That Occurred at Night within each city

- **Distribution of Calculated Risk Scores** - This figure highlights histogram with a KDE (Kernel Density Estimate) line that shows how the calculated Risk Scores are distributed across all crime records. This plot helps to understand the frequency of crimes at different risk levels, ranging from low (0) to high (1).

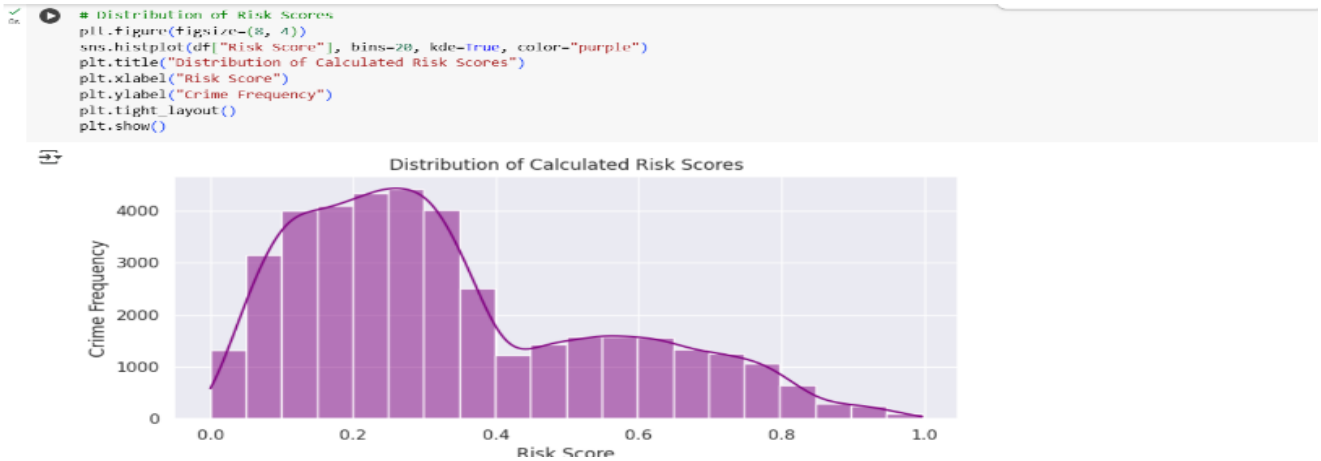


Fig 4.5: Visualizing Distribution of Calculated Risk Scores through KDE

4.3.2 Comparison of different models based on mean accuracy scores

In this section, we demonstrated the comparison of different machine learning models used in this project based on mean accuracy scores depending upon proposed system vs referenced study. This table demonstrates the overall comparison of different models based on mean accuracy scores -

Table 4.2: Comparison of Proposed Model vs Referenced Model Performance

Model	Mean Accuracy (Proposed System)	Mean Accuracy (Referenced Study)
Logistic Regression	0.98	0.74
Support Vector Classifier	0.96	Not used
XGBoost Classifier	0.99	0.83
LightGBM Classifier	0.99	Not used
Multi-Layer Perceptron	0.98	Not used
K-Nearest Neighbors (KNN)	Not used	0.79
K-Means Clustering	Not used	Used for hotspot detection only

This figure shows comparisons of various machine learning models based on mean accuracy scores. By observing carefully **XGBoost** and **LightGBM Classifier** model has achieved highest mean accuracy score, and the lowest mean accuracy score was achieved by **Support Vector Classifier** model. The respective line plot is shown below -

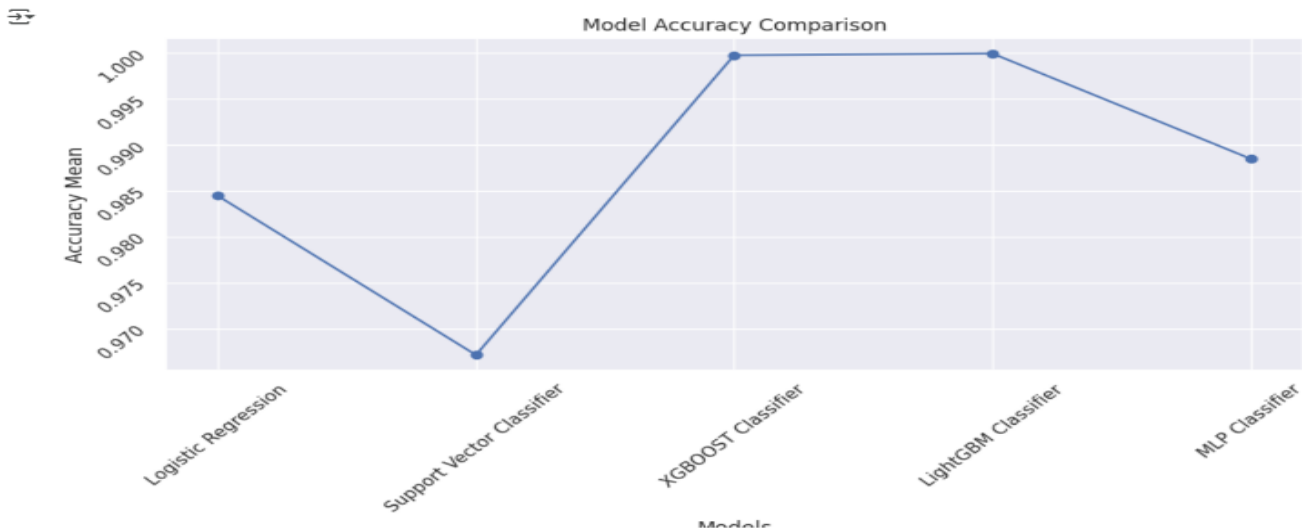


Fig 4.6: Visualizing Comparision of different models based on mean accuracy scores

4.3.3 Visualize Crime Hotspots on Folium Map for multiple cities

This image shows a heatmap of crime hotspots in the city like - **Mumbai, Indore** and many more generated using the **Folium** and **Leaflet.js** mapping libraries. The red markers at the center indicate a high-density crime hotspot, with surrounding gradient colors (yellow to blue) representing decreasing levels of crime frequency. Each marker corresponds to an individual incident.

When a user interacts with the map, a popup message appears showing detailed information about the selected crime event.

This map helps in clearly identifying urban zones with high risk, enabling the routing system to avoid such areas when suggesting safer paths. The spatial analysis of crime incidents is an essential input to the risk-based path finding algorithm used in this proposed system.

- First look crime hotspot analysis for ‘Mumbai’ city -

```
display(city_heatmaps['Mumbai'])
```

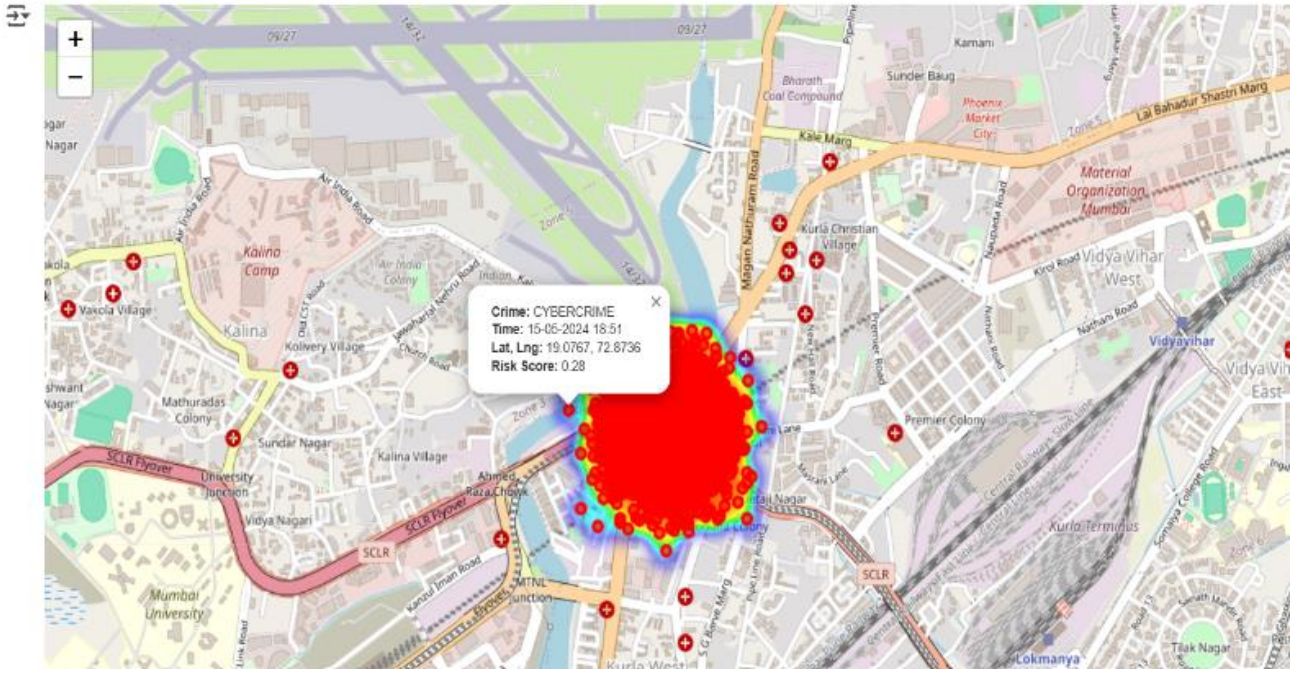


Fig 4.7: Visualizing Crime Hotspots on Folium Map for 'Mumbai' city

- Then secondly crime hotspot analysis for 'Indore' city -

```
[ ] display(city_heatmaps['Indore'])
```

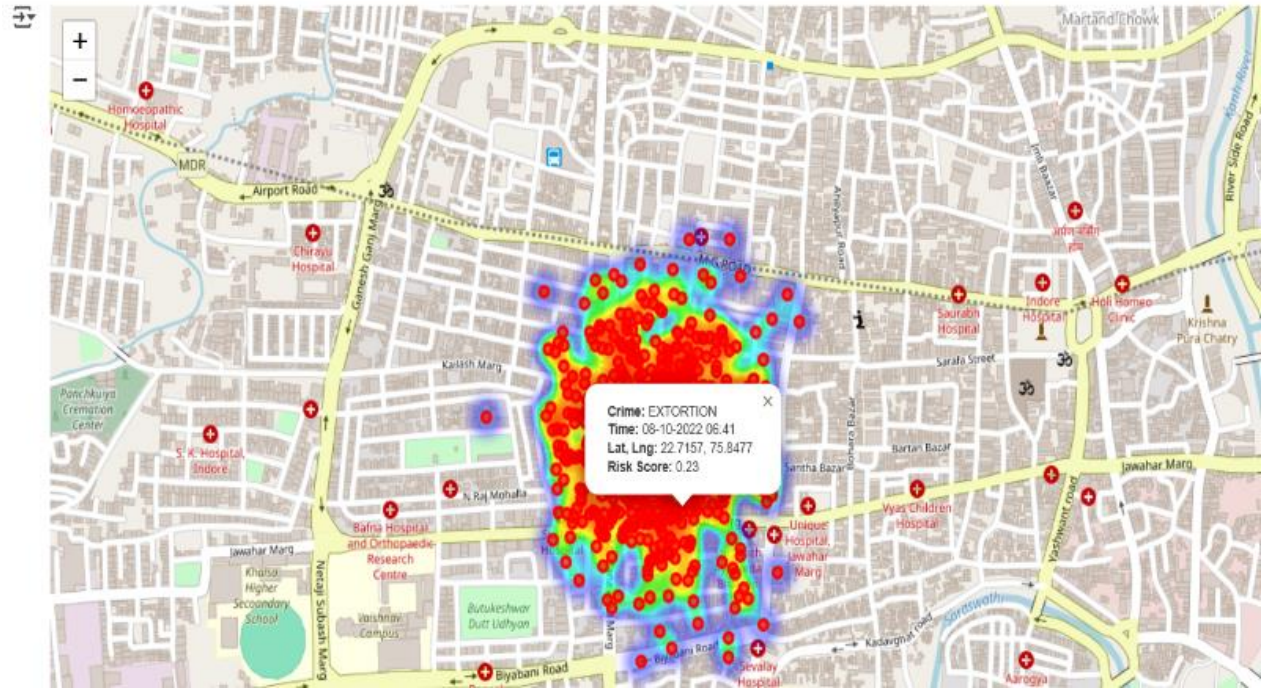


Fig 4.8: Visualizing Crime Hotspots on Folium Map for 'Indore' city

4.3.4 Visualize Safest and Shortest Paths using Dijkstra's Algorithm on Folium Map for multiple cities

This figured map shows a comparative visualization of two alternative navigation routes generated for each cities. The paths are computed using **Dijkstra's algorithm** on a crime-weighted road network, created using **OSMnx** and visualized through **Folium** on **Leaflet.js** map.

- The **green line** shows the **safest route**, which avoids areas with a high crime risk. This path is computed by minimizing exposure to locations with elevated risk scores, even if the distance is slightly longer.
- The **red dashed line** represents the **shortest route**, which prioritizes minimum physical distance without accounting for crime proximity.

This visualization allows users to make informed travel decisions by comparing risk-based routing against conventional shortest-path navigation.

- First look shortest and safest path analysis for 'Delhi' city -

Fig 4.9: Visualizing Shortest and Safest path on Folium Map for 'Delhi' city

- Then secondly safest and shortest path analysis for 'Pune' city -

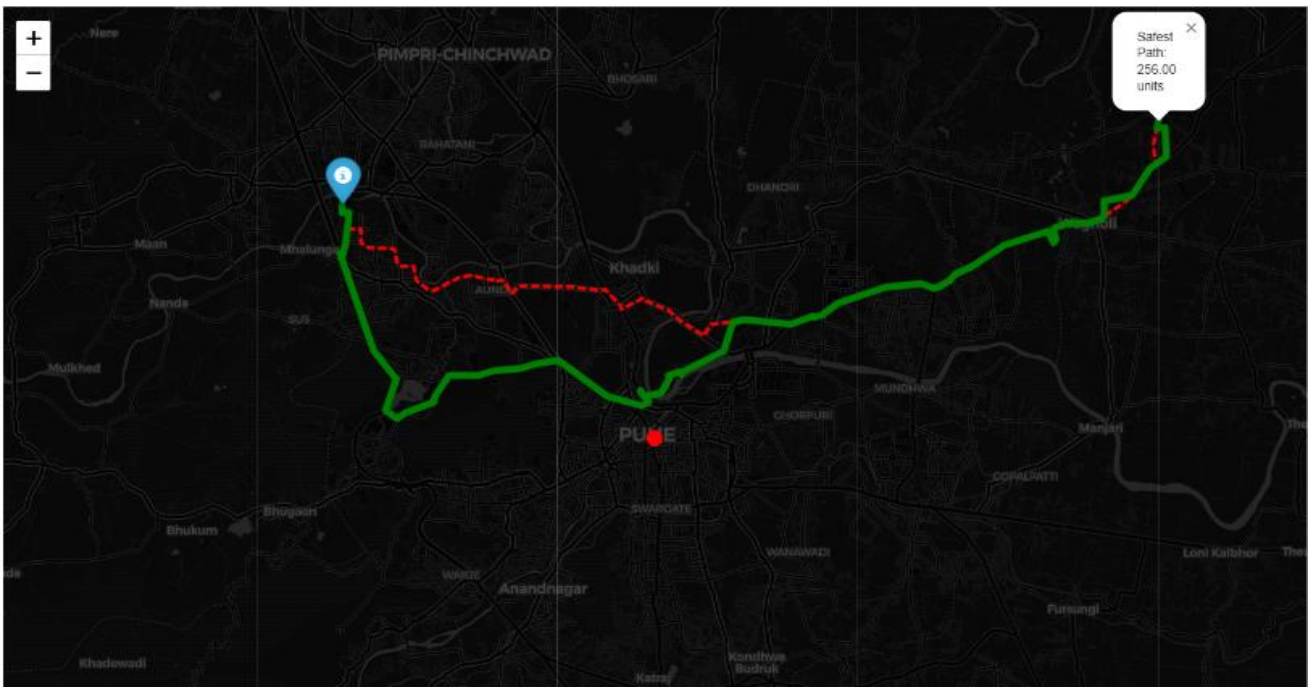


Fig 4.10: Visualizing Shortest and Safest path on Folium Map for ‘Pune’ city

CHAPTER-5

CONCLUSIONS AND FUTURE SCOPE

5.1 Conclusion

This project focused on analyzing city-level crime data to develop a smart and interactive navigation system that promotes safer urban travel. Using tools such as Pandas, Folium, OSMnx, and NetworkX, we identified and visualized high-risk zones, constructed road networks, and implemented Dijkstra's algorithm to recommend both shortest and safest routes for users.

The workflow began with cleaning and processing the crime dataset, followed by mapping crime-prone areas using heatmaps enriched with incident-specific details like type and risk score. These map plots helped identify spatial crime-sensitive patterns across different cities within India.

To convert this analysis into a functional tool, road network graphs were generated using OpenStreetMap data. Roads close to high-crime areas were penalized through weighted scoring. This enabled the algorithm to compute not only the shortest route based on distance, but also an alternate path that prioritized individuals' safety by bypassing crime prone areas —even if it meant slightly longer travel time.

5.2 Future Scope

The proposed study lays a strong foundation for integrating safety into navigation, but there are multiple domains in which it can be further enhanced. In future scope, this system can incorporate **real-time crime updates** from government or police databases to improve the accuracy and efficiency of risk assessments. Integrating **individuals' feedback or community alerts** could allow citizens to report suspicious crime activities, contributing to a more dynamic safety layer.

Moreover, the project could be extended to support **emergency routing**, guiding individuals toward nearby police stations or safer zones during high-risk scenarios. Expanding the system to handle **multi-modal routes** like public transport will make it more inclusive. Adding **custom safety preferences**—based on age, gender, or time of day—can also personalize the experience for different user groups.

Lastly, deploying this solution as a **mobile or web app with offline navigation and multilingual support** via LLM's will significantly enhance accessibility, especially for people in areas with limited internet connectivity.

REFERENCES

[1] S. G. Hegde, “Indian Crimes Dataset,” *Kaggle Datasets*, Oct. 2023. Accessed: July 9, 2025. [Online]. Available: https://www.kaggle.com/datasets/sudhanvahg/indian-crimes-dataset

[2] OpenAI, ‘*Crime-Sensitive Route Navigation System Flowchart’*, OpenAI, San Francisco, CA, USA. [Online]. Available: https://files09.oaiusercontent.com/file-5XYu2dEVqFH8PiwzUzDKbQ?se=2025-07-13T03%3A17%3A59Z&sp=r&sv=2024-0804&sr=b&rscc=maxage%3D299%2C%20immutable%2C%20private&rscd=attachment%3B%20filename%3DCrime_Safe_Routing_Flowchart.png&sig=2Gm8toGpYC%2ByJCZn0UjgCCMLtCGgaJE1ZOxos65ZNrQ%3D

[3] NYC Open Data, “Buildings Subject to HPD Jurisdiction,” *City of New York – Housing Development.* Available:https://data.cityofnewyork.us/HousingDevelopment/Buildings-Subject-to-HPD-Jurisdiction/kj4p-ruqc/about_data. Accessed: July 9, 2025.

[4] V. G. Shankar, B. L. Ram, and S. Ghosh, “Route-The Safe: A Robust Model for Safest Route Prediction Using Crime and Accidental Data,” *International Journal of Engineering and Advanced Technology (IJEAT)*, vol. 9, no. 2, pp. 34–39, 2019.

[5] P. Samarati and S. Sassi, “A Machine Learning Approach to Urban Crime Pattern Detection,” in *Proc. IEEE Int. Conf. Data Sci. Adv. Analytics*, 2021, pp. 124–131.

[6] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[7] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2016, pp. 785–794.

[8] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature*, vol. 521, pp. 436–444, May 2015.

[9] G. Ke et al., “LightGBM: A Highly Efficient Gradient Boosting Decision Tree,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 3146–3154.

[10] J. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” in *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, 1967.

[11] V. G. Shankar, B. L. Ram, and S. Ghosh, "Route-The Safe: A Robust Model for Safest Route Prediction Using Crime and Accidental Data," *Int. J. Eng. Adv. Technol.*, vol. 9, no. 2, pp. 34–39, Dec. 2019.

[12] J. O. Olusina and J. B. Olaleye, “Journey to Crime Using Dijkstra’s Algorithm,” *Nigerian Journal of Environmental Sciences and Technology*, vol. 1, no. 1, pp. 1–14, Mar. 2017. [Online]. Available: https://www.researchgate.net/publication/335387366

[13] A. Patidar, K. Khune, and S. Srividhya, “Safe Route Recommendation System for Pedestrians,” *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 4, pp. 4665–4670, Jul.–Aug. 2020. [Online]. Available: http://www.warse.org/IJATCSE/static/pdf/file/ijatcse68942020.pdf

[14] S. Levy, W. Xiong, E. Belding, and W. Y. Wang, “SafeRoute: Learning to Navigate Streets Safely in an Urban Environment,” *arXiv preprint arXiv:1811.01147*, Nov. 2018. [Online]. Available: https://arxiv.org/abs/1811.01147

[15] Y. S. Asawa, S. R. Gupta, V. V, and N. J. Jain, “User-Specific Safe Route Recommendation System,” *International Journal of Engineering Research & Technology (IJERT)*, vol. 9, no. 10, pp. 574–580, Oct. 2020. [Online]. Available: https://www.ijert.org/user-specific-safe-route-recommendation-system

[16] S. Rehan, M. Haris, M. Elaf, J. Khan, G. Siddique, and M. Ahetesham, “Crime Aware Navigation,” *International Journal of Scientific Research in Engineering and Management (IJSREM)*, vol. 8, no. 12, pp. 1–9, Dec. 2024. [Online]. Available: http://www.ijsrem.com/

[17] A. Banerjee and M. Roy, “Beyond the Shortest Route: A Survey on Quality-Aware Route Navigation for Pedestrians,” *Scispace Preprint*, 2023. [Online]. Available: https://scispace.com/pdf/beyond-the-shortest-route-a-survey-on-quality-aware-route-2lkfu7ztlb.pdf

APPENDIX

Dataset Link - https://www.kaggle.com/datasets/sudhanvahg/indian-crimes-dataset

Colab File Link -
https://colab.research.google.com/drive/1yfHEYTHvPLmKYGtjmB6Z9GJehcEMTA73#scrollTo=O3-ep2LnHgOe