B.Tech. BCSE498J - Project-II

METRO LINE PREDICTION USING MACHINE LEARNING MODEL

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

21BCE0512	**JAIVARDHAN TAMMINANA**
21BCE0920	**SOUMESH PADHI**

Under the Supervision of

Dr. Ebenezer Juliet S

Associate Professor Sr.

School of Computer Science and Engineering

(SCOPE)

April 2025

DECLARATION

I hereby declare that the project entitled Metro Line Prediction using Machine Learning Model submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Prof. Dr. Ebenezer Juliet S.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree ordiploma in this institute or any other institute or university.

Place : Vellore

Date : 21-04-2025

T-Jaivardhan

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled Metro Line Prediction using Machine Learning Model submitted by Jaivardhan Tamminana (21BCE0512), **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by him under my supervision during Winter Semester 2024-2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 21-04-2025

[illegible]

Signature of the Guide

VELLORE INSTITUTE OF TECHNOLOGY (VIT) (Deemed to be University under section 3 of UGC Act, 1956) School of Computer Science & Engineering (SCOPE) Vellore-632 014, T.N., India

[illegible]

Internal Examiner

[illegible]

External Examiner

[illegible]

Umadevi K S

Computer Science and Engineering

ACKNOWLEDGEMENTS

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

My sincere thanks to Dr. Jaisankar N, Dean - School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence.

I express my profound appreciation to Dr. Umadevi K S, the Head of the Computer Science and Engineering Department, for her insightful guidance and continuous support. Her expertise and advice have been crucial in shaping throughout the course. Her constructive feedback and encouragement have been invaluable in overcoming challenges and achieving goals.

I am immensely thankful to my project supervisor, Dr. Ebenezer Juliet S, for her dedicated mentorship and invaluable feedback. Her patience, knowledge, and encouragement have been pivotal in the successful completion of this project. My supervisor’s willingness to share her expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies. Her support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

Jaivardhan Tamminana

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	**Full Form**
SQ KM	Square Kilometer
ML	Machine Learning
CO_2	Carbon Dioxide
IT	Information Technology
API	Application Programming Interface
RMSE	Root Mean Square Error
MSE	Mean Square Error
CPU	Central Processing Unit
GPU	Graphics Processing Unit
OS	Operating System
RAM	Random Access Memory
SSD	Solid State Drive
CNN	Convolutional Neural Network
RNN	Recursive Neural Network
IoT	Internet of Things
NLP	Natural Language Processing
JSON	JavaScript Object Notation
ETL	Extract, Transform, Load
PDF	Portable Document Format
HTTP	Hypertext Transfer Protocol
SMOTE	Synthetic Minority Oversampling Technique
CSV	Comma Separated Values
HTML	Hypertext Markup Language
JWT	JSON Web Token
KMS	Kilometers
RFC	Random Forest Classifier
ML	Machine Learning
UI	User Interface

ABSTRACT

Urban planning today faces growing challenges due to increasing populations, rapid urbanization, and climate change. Traditional methods struggle to meet the demand for sustainable and resilient cities. This project explores how machine learning can help by analyzing factors such as population density, environmental risks (earthquakes, floods, and wind hazards), socio-economic influences, commute time, connectivity to transportation hubs (airports, railways, and ports), and existing infrastructure conditions (inadequate or overburdened). We also incorporate age distribution data to understand demographic trends and categorize natural disaster risks as low, medium, or high for better preparedness.

By leveraging large datasets and predictive models, our approach guides smarter, more adaptive city planning. We integrate data-driven insights with urban planning principles to predict risks and develop solutions for resilience. Machine learning algorithms identify patterns among environmental factors, urban growth, and transportation networks, offering practical insights for safer, more efficient, and sustainable cities. We also analyze trade-offs between economic growth and environmental sustainability through simulations of various planning strategies.

Our project supports urban planners and policymakers in making informed decisions about infrastructure, zoning, disaster preparedness, transportation, and sustainability initiatives. By incorporating machine learning, we enhance risk assessments and create a flexible model applicable to cities worldwide. Moving forward, we will refine the model with real-time data and expand its scope for greater comprehensiveness.

Keywords: *Urban Planning, Machine Learning, Population Density, Environmental Risks, Climate Change, Predictive Modelling, Sustainable Development, Disaster Preparedness, Commute Time, Transportation Connectivity, Age Distribution, Infrastructure Assessment, Sustainability.*

Chapter 1

1. INTRODUCTION

1.1 BACKGROUND

Urbanization in India has significantly intensified transportation challenges, with traffic congestion and rising carbon emissions emerging as critical concerns in most urban canters. As cities expand rapidly, road infrastructure is increasingly unable to meet the growing demand, leading to prolonged commute times, frequent traffic gridlocks, and a surge in greenhouse gas emissions [1]. Public transport systems, particularly buses, are struggling to handle the growing commuter population, often resulting in overcrowded vehicles, inefficiencies, and dissatisfaction among users [2]. This transportation crisis is further compounded by its environmental implications, as urban areas grapple with deteriorating air quality and higher rates of respiratory illnesses and cardiovascular diseases linked to pollution [3]. These issues underscore the pressing need for innovative, sustainable transportation solutions that address both mobility and environmental concerns.

Metro rail systems have emerged as a transformative option in this context, offering a sustainable and high-capacity alternative to traditional road-based transportation. Metro systems have demonstrated their ability to significantly reduce traffic congestion, promote eco-friendly commuting, and lower the overall carbon footprint of cities [4]. However, planning and implementing metro networks present unique challenges, especially in India’s diverse urban environments. District-specific traffic patterns, variations in population density, and exposure to environmental risks, such as floods, earthquakes, and air pollution hotspots, must be meticulously accounted for [5][6]. Without precise and comprehensive planning, metro systems risk being underutilized or failing to address the intended problems effectively.

To tackle these challenges, this project integrates advanced machine learning (ML) techniques into metro planning. By analyzing extensive datasets encompassing traffic flow, vehicular emissions, population dynamics, and natural hazard risks, ML models can provide accurate predictions and actionable insights [7]. This data-driven approach empowers urban planners to optimize metro routes, anticipate future demand, and align systems with environmental sustainability goals [8]. ML-based planning also ensures metro systems are adaptive to evolving urban conditions, making them resilient, efficient, and scalable. By addressing these critical aspects, this project seeks to create a replicable framework for metro planning that supports India's broader objectives of sustainable urbanization and eco-friendly mobility.

1.2 MOTIVATION

The motivation for this project stems from the urgent need to tackle two deeply interconnected challenges that urban India faces today: traffic congestion and carbon emissions. As cities like Chennai experience rapid growth, the number of vehicles on the roads continues to increase exponentially, resulting in unprecedented pressure on urban infrastructure (NITI Aayog, 2021). The average travel speeds in these cities are steadily declining, which not only hampers daily productivity but also exacerbates environmental and economic stresses (Tiwari et al., 2016). Traffic congestion creates a cascade of negative effects, including extended commute times, elevated fuel consumption, and significantly higher greenhouse gas emissions (Srinivas et al., 2019). These emissions, in turn, degrade air quality, contributing to severe environmental and public health concerns such as respiratory diseases, cardiovascular issues, and the broader impacts of climate change (Chauhan et al., 2020).

Metro rail systems have emerged as one of the most promising solutions to these issues. They offer an efficient, high-capacity alternative to road-based transportation, providing a sustainable means of mobility that significantly reduces both traffic congestion and vehicular emissions (Nair et al., 2020). Despite their potential, the planning and execution of metro systems are highly complex and require meticulous attention to a variety of dynamic factors. Effective metro planning involves not just the physical design and engineering aspects but also a deep understanding of traffic patterns, urban population growth, and environmental impacts (Verma et al., 2017). Without proper integration of these factors, metro systems risk being underutilized or failing to achieve their full potential in addressing urban mobility challenges.

This project addresses these complexities by incorporating machine learning (ML) models into the metro planning process. ML offers a cutting-edge approach to solving these challenges by analyzing vast datasets that include traffic flow, carbon emission levels, and passenger demand (Vohra & Sharma, 2021). These models enable the prediction of future traffic dynamics and environmental impacts, offering insights that are critical for data-driven decision-making. By simulating various scenarios, ML tools can help urban planners optimize metro routes, ensuring that these systems effectively alleviate congestion and minimize emissions while meeting the needs of rapidly growing urban populations (Kumar & Muralidharan, 2022).

Moreover, this project aligns with India's broader goals of reducing greenhouse gas emissions and transitioning to sustainable urban transportation systems, as outlined in the country's national climate change initiatives (Ministry of Environment, Forest and Climate Change, 2021). By integrating technology-driven approaches such as ML, the project not only addresses immediate urban mobility challenges but also contributes to the long-term vision of creating environmentally friendly and economically viable cities. The use of ML models in metro planning marks a significant advancement in urban transportation, offering an innovative framework for addressing the pressing issues of congestion and pollution in India’s urban centers (Indian Green Building Council, 2020).

1.3 SCOPE OF THE PROJECT

This project aims to integrate data-driven insights with urban transportation planning, leveraging advanced machine learning (ML) techniques to address the critical challenges of traffic congestion and carbon emissions. The focus lies in analyzing extensive datasets, including traffic density, vehicular emission statistics, and urban infrastructure details, to inform the strategic planning and expansion of metro lines across various districts in India. By identifying patterns in traffic flow, emission levels, and population growth, the project seeks to pinpoint the key factors that should guide decisions on where and how metro lines are developed, ensuring alignment with both mobility demands and environmental sustainability goals [1][8].

The ML model designed through this initiative will provide precise forecasting of future transport demands, enabling urban planners to anticipate the needs of a growing and dynamic population. This includes accounting for the environmental implications of sustained vehicular emissions [9]. The model will be trained using high-quality data derived from metro projects in major cities like Chennai and Delhi, where issues such as severe traffic congestion and air pollution have reached critical levels [14]. These cities serve as ideal case studies due to their advanced urban infrastructure and the urgent need for sustainable solutions [1]. The insights generated by the ML model will empower planners to design metro systems that are not only efficient and cost-effective but also capable of significantly reducing carbon footprints and mitigating urban congestion [5].

Moreover, this project aims to create a scalable and replicable framework that can be adapted for metro planning in other cities, catering to their unique challenges and growth trajectories. By fostering a cleaner, more efficient public transportation network, the project contributes to broader national goals of sustainable development and urban resilience [8]. Ultimately, the integration of ML technology in metro planning represents a forward-thinking approach that not only enhances the functionality of urban transport systems but also paves the way for a greener, more liveable future for Indian cities [1].

Chapter 2

2. PROJECT DESCRIPTION AND GOALS

2.1 LITERATURE REVIEW

1) Sumathy Eswaran, M. A. J. Bosco, and Rajalakshmi, “A Study on Traffic Forecast for Metro Railway Expansion in Chennai,” *Indian Journal of Science and Technology*, vol. 9, no. 39, pp. 1-9, Oct. 2016, doi: 10.17485/ijst/2016/v9i39/95429.

This paper examines the feasibility of metro rail expansion in Chennai with a focus on traffic congestion, public transport demand and carbon emissions. The authors conduct a survey of potential metro users, analyzing travel patterns, purposes, and preferences. They project the future traffic growth and emphasize the importance of adopting eco-friendly, low-carbon transportation systems like metro rails. The study highlights the potential for metro systems to alleviate road congestion, reduce travel times, and lower carbon emissions. Key findings from the survey show high public support for metro rail, with commuters seeking improved comfort, reduced travel time, and affordable fares. The paper also discusses the potential revenue benefits of metro systems based on trip length and passenger preferences.

2) R. Goel and G. Tiwari, “Promoting Low Carbon Transport in India: Case Study of Metro Rails in Indian Cities,” UNEP Risø Centre on Energy, Climate and Sustainable Development, Technical University of Denmark, June 2014.

This paper is part of a larger UNEP-sponsored project focusing on sustainable transportation in India. The paper emphasizes the need for metro systems to achieve low-carbon urban growth, citing that India’s transport sector is responsible for 13% of the country’s energy-related CO_2 emissions. The study compares metro rail projects in Delhi, Bangalore, Chennai and Mumbai, discussing the socio-economic benefits, environmental impacts, and challenges faced by these systems. A significant portion of the paper is dedicated to discussing the life cycle assessment (LCA) of emissions from metro systems, showing that while metros can reduce overall emissions, the construction and operation phases still contribute to CO_2 emissions. The study underscores the importance of public transportation in mitigating climate change but also calls for more comprehensive assessments of environmental impacts.

3) S. Jain and G. Tiwari, "Evaluating the Impact of Metro Rail on Urban Transport Systems: A Case Study of Delhi Metro," *Transportation Research Part A: Policy and Practice*, vol. 45, no. 1, pp. 80-94, Jan. 2011, doi: 10.1016/j.tra.2010.09.002.

This paper focuses on the **impact of metro rail systems on urban mobility** in Indian cities, specifically Delhi. The authors explore the shift from road-based transport to rail-based public transportation due to the introduction of the metro system. The study uses a comprehensive data analysis of travel behaviors, passenger demand, and traffic congestion before and after the implementation of the Delhi Metro. The research demonstrates that the metro significantly reduces traffic congestion on major corridors, leading to shorter travel times and improved public transport usage. Additionally, the paper discusses the broader environmental impact, particularly the reduction in carbon emissions attributed to fewer private vehicles on the road. However, it also highlights challenges such as limited coverage and access/egress issues, which affect the metro's overall efficiency. The study calls for better integration between metro and other public transport modes to maximize benefits.

4) A. Mohan, "Public Transport Policy and the Role of Rail-Based Systems in India," *Journal of Transportation Planning and Technology*, vol. 31, no. 3, pp. 225-245, 2008, doi: 10.1080/03081060802106114.

This paper delves into the **policy framework surrounding public transport in India** and the increasing role of rail-based systems like metros. The paper discusses the economic, environmental, and social justifications for metro rail projects, presenting them as key to tackling urban challenges like congestion and pollution. The study compares various Indian cities, including Kolkata, Delhi, and Bangalore, where metro systems have been implemented or planned. It outlines the cost-effectiveness of metro systems versus road-based transport options, emphasizing that metros not only reduce traffic congestion but also contribute to lower greenhouse gas emissions. However, the paper warns against over-optimism, noting that many metro projects suffer from inflated benefits and underestimation of costs, leading to budget overruns and delayed implementation.

5) T. Flyvbjerg, A. Bruzelius, and W. Rothengatter, "Cost Underestimation and Benefit Overestimation in Rail Projects: A Global Perspective," *International Journal of Project Management*, vol. 24, no. 6, pp. 481-488, Aug. 2006, doi: 10.1016/j.ijproman.2006.07.006.

This paper provides a global examination of **cost estimation and benefit analysis issues in rail projects**, including metro systems. Flyvbjerg and his co-authors analyse over 200 rail-based transportation projects across the world and find a consistent pattern of underestimation of costs and overestimation of benefits, particularly in developing countries. The study also touches upon the **environmental claims** of metro projects, noting that while metros are marketed as eco-friendly solutions, the actual environmental benefits, such as reduced carbon emissions, are often lower than predicted due to incomplete project execution and unanticipated delays. The authors recommend more realistic budgeting and planning, as well as the use of advanced forecasting methods for better predictions of demand and cost, to prevent underperformance of these large infrastructure projects.

2.2 GAPS IDENTIFIED

1) Absence of District-Level Planning in Metro Rail Projects

Many of the available studies focus on city-wide data for traffic congestion, emissions, and population density. However, they often overlook the importance of district-level analysis, which is critical for making informed decisions about where metro systems should be expanded or added. City-wide data may not capture the nuanced variations in traffic patterns, environmental impacts, and commuter demand that differ across districts. As urban centres grow, certain districts face unique challenges such as higher congestion, limited infrastructure, or greater environmental risks due to their proximity to industrial zones or pollution hotspots. Without district-level planning, metro systems may fail to meet the needs of specific areas, leading to underutilization in some districts while creating unnecessary congestion in others. Research focusing on local variations in traffic flow, population density, and emissions is critical to ensure that metro networks are effectively designed to address these disparities and optimize resource allocation [1][6].

For instance, in cities like Chennai and Delhi, districts closer to the central business district (CBD) or industrial hubs often experience significantly higher levels of congestion and air pollution. Metro systems designed without considering these district-specific factors could overlook key areas of high commuter demand or fail to alleviate congestion in the most critical regions. Incorporating district-level planning would allow for the development of more targeted and efficient metro networks, ensuring that metro expansions address the root causes of traffic congestion and emissions in each district.

2) Limited Use of Machine Learning for Traffic and Emission Predictions

Another critical gap in the existing literature is the underutilization of machine learning (ML) techniques in modelling and predicting traffic patterns and emissions. Most of the studies on metro systems use traditional statistical models such as regression analysis or time-series forecasting, which, while useful, have limitations in handling the complexity and volume of data involved in urban transportation planning. These methods often fail to capture non-linear relationships between variables and cannot account for the dynamic changes in urban traffic conditions, weather patterns, or population mobility. Moreover, they are generally unable to incorporate real-time data or adapt to evolving urban conditions.

Machine learning techniques, on the other hand, can process large and complex datasets with high accuracy, providing deeper insights into traffic dynamics, emissions levels, and the impact of various factors like weather, accidents, and public holidays. For example, ML models like decision trees, neural networks, or deep learning algorithms can be trained on historical traffic data, weather conditions, and emissions levels to predict future traffic flows and carbon emissions under different scenarios. These models are capable of simulating the effects of new metro expansions on traffic patterns and emissions with greater precision than traditional methods, making them invaluable tools for urban planners and policymakers [7][1].

The integration of ML into metro rail project planning could also support the development of adaptive systems that continuously learn from new data and adjust planning strategies accordingly. Such systems would help planners make more accurate predictions and optimize metro routes, capacity, and schedules based on real-time data, thereby improving the overall efficiency of metro systems.

3) Lack of Longitudinal Analysis of Metro Impact on Carbon Emissions

Existing studies often focus on the immediate or short-term benefits of metro systems, such as reduced congestion and lower emissions during peak hours. However, there is a lack of comprehensive longitudinal analysis of metro systems' long-term impact on carbon emissions and their potential to offset initial pollution from construction activities. Metro systems are often promoted as a key solution to reducing the environmental impact of urban transportation, but their effectiveness in achieving long-term reductions in greenhouse gas emissions remains underexplored.

For instance, the carbon footprint of building a metro system can be substantial due to the energy-intensive construction process, including tunnelling, station building, and rail installation. While metro systems can lead to significant reductions in emissions by promoting public transportation over private vehicles, these benefits may take several years or even decades to fully materialize, depending on the speed of adoption and the level of integration with other sustainable modes of transport. Therefore, a longitudinal study examining the cumulative effect of metro systems over time—factoring in both construction-related emissions and the subsequent decrease in vehicular emissions—is essential for understanding the long-term environmental benefits of metro expansion projects [9].

Such an analysis could also help identify strategies for maximizing the long-term environmental gains of metro systems, such as integrating renewable energy sources for powering metro operations or implementing policies to encourage greater use of public transport. A more comprehensive understanding of the long-term environmental effects would contribute to more accurate projections and better decision-making regarding future metro projects.

4) City-Specific Data and Limited Generalizability

Another major limitation of the existing research is that the data analysed in most papers is often city-specific, which restricts the generalizability of findings. Many studies focus on a single city, such as Delhi or Chennai, and assess the effectiveness of metro systems within the context of that city's unique traffic dynamics and environmental conditions. While these studies provide useful insights for the respective cities, they do not offer sufficient guidance for metro planning in other cities with different demographic, economic, and geographic characteristics.

For example, the traffic patterns, emission levels, and commuter behaviours in a city like Bangalore, with its distinct topography and rapid IT sector-driven growth, may differ significantly from those in Kolkata, which has a more traditional, industrial economy. As a result, metro expansion strategies and their predicted impact on traffic and emissions cannot be universally applied across all cities. A broader, more comparative approach that incorporates data from multiple cities would help to develop more adaptable, scalable metro planning frameworks that can be customized to fit the unique characteristics of various urban centres across India [8].

In conclusion, while existing research on metro rail systems in India provides valuable insights into urban mobility and emissions reduction, several critical gaps need to be addressed. These include the absence of district-level planning, limited use of advanced machine learning techniques, the lack of longitudinal analysis of metro systems' long-term impact on carbon emissions, and the over-reliance on city-specific data. Addressing these gaps through more detailed, data-driven approaches would enhance the effectiveness of metro rail systems in tackling urban congestion, reducing carbon emissions, and contributing to sustainable urban development. By incorporating machine learning, longitudinal studies, and broader data sets, urban planners can design metro systems that are not only efficient and adaptive but also capable of delivering long-term environmental benefits.

2.3 OBJECTIVES

1) Specific: Developing a Machine-Learning Model

The primary goal is to develop a machine-learning model that incorporates both traffic congestion and carbon emission data to optimize the planning of metro lines in specific districts within major Indian cities. By using data-driven insights, the model will pinpoint critical areas for metro expansion, ensuring that metro systems are not only well-placed to reduce traffic but also to contribute significantly to reducing the overall carbon footprint of urban areas. In doing so, this approach will improve urban mobility and contribute to cleaner, more sustainable transport systems.

Metro rail systems have been proven to reduce congestion and lower emissions when strategically integrated into city layouts (Verma et al., 2017). However, many cities still struggle with poorly coordinated metro expansions, where routes may not be adequately designed to serve the areas with the most traffic and highest emission levels. This project aims to remedy this gap by employing machine learning models that can account for dynamic traffic flows, population density, and emission hotspots, providing a more nuanced and optimized approach to metro planning.

2) Measurable: Achieving Accuracy through Validation

The success of the project will be measured by its predictive accuracy, specifically in forecasting traffic congestion and emission levels in targeted districts. To evaluate the effectiveness of the model, it will be validated using historical traffic and carbon emission data from at least three major cities—Chennai, Delhi, and Bangalore. These cities have been chosen due to their high urbanization, significant congestion issues, and varying levels of pollution, making them ideal test cases for the model.

Chennai experiences high levels of traffic congestion and air pollution, especially in its central business district and industrial hubs (Tiwari et al., 2016).

Delhi, the capital city, has one of the highest vehicular emission rates in the country, and its transportation network is under significant stress (Srinivas et al., 2019).

Bangalore, a city known for its tech industry growth, faces severe traffic bottlenecks due to rapid urbanization and limited public transport options (Nair et al., 2020).

Using these cities as case studies will allow the project to assess the model's robustness and ensure that it can generalize across different urban environments. By leveraging data from government databases and open-source traffic and emission datasets, the model's predictions can be validated against real-world outcomes, ensuring high accuracy in forecasting future trends.

3) Achievable: Feasibility with Existing Data and ML Frameworks

The project is achievable using existing traffic, emissions, and transport data sourced from government databases and public transport systems. The Indian government maintains several open datasets, including the National Air Quality Index (AQI) and traffic flow data, which will serve as the foundation for training the model. Public transport data, such as ridership statistics and metro expansion plans, will also be leveraged to incorporate patterns of current metro usage and infrastructure gaps (Verma et al., 2017).

In terms of technical feasibility, widely available machine learning frameworks such as Scikit-learn and Tensor Flow will be utilized for model development. These frameworks provide extensive support for implementing algorithms like decision trees, random forests, and neural networks, which are well suited for predicting complex, non-linear relationships between traffic congestion, emissions, and metro infrastructure. Additionally, Python libraries like Pandas and NumPy will be used for data manipulation and pre-processing, enabling efficient handling of large datasets.

4) Relevant: Aligning with National Sustainability Goals

The project is highly relevant to India's broader objectives of reducing greenhouse gas emissions and improving urban transportation efficiency. As part of India's commitment to the Paris Agreement and its national climate action plan, the country has set ambitious targets for reducing emissions and promoting sustainable transport options. Public transport, particularly metro systems, plays a pivotal role in these efforts. By optimizing metro routes through data-driven insights, the project supports national sustainability initiatives, including the goal of reducing carbon emissions from the transport sector by 30-35% from 2005 levels by 2030 (Ministry of Environment, Forest and Climate Change, 2021).

The integration of machine learning into metro planning aligns with India's Smart Cities Mission, which aims to improve urban infrastructure through technology and sustainable practices. The project will also support the National Electric Mobility Mission Plan (NEMMP) and Faster Adoption and Manufacturing of Hybrid and Electric Vehicles (FAME) schemes by promoting efficient, low-emission public transportation options.

5) Time-Bound: Four-Month Completion with Specific Milestones

Month 1-2: Data collection and pre-processing, including gathering traffic congestion, emissions, and public transport datasets from government sources and city transport departments. Model development and initial training using machine-learning algorithms like decision trees and neural networks. During this phase, the model will be validated with historical data from the selected cities (Chennai, Delhi and Bangalore).

Month 3-4: Testing and refinement of the model. This phase will involve fine-tuning the model parameters to improve predictive accuracy and incorporating additional factors such as district-specific population density and air quality. Final deployment of the model, including a detailed report on the findings and actionable insights for metro line optimization in the selected districts.

2.4 PROBLEM STATEMENT

India is experiencing a huge wave of urbanisation and with each new vehicle on the road, combating traffic congestion and carbon emissions in Indian cities have become more challenging than ever. One potential candidate in this category could be metro systems, but specific traffic patterns and emissions at the district level are hardly considered in current planning procedures, nor shall they employ advanced machine learning models for precise prediction. Addressing these gaps, this project is to build a ML model that can deliver research-informed data insight for the metro line planning to coordinate on infrastructure efficiency, leading toward reduction in traffic congestion and carbon emission.

2.5 PROJECT PLAN

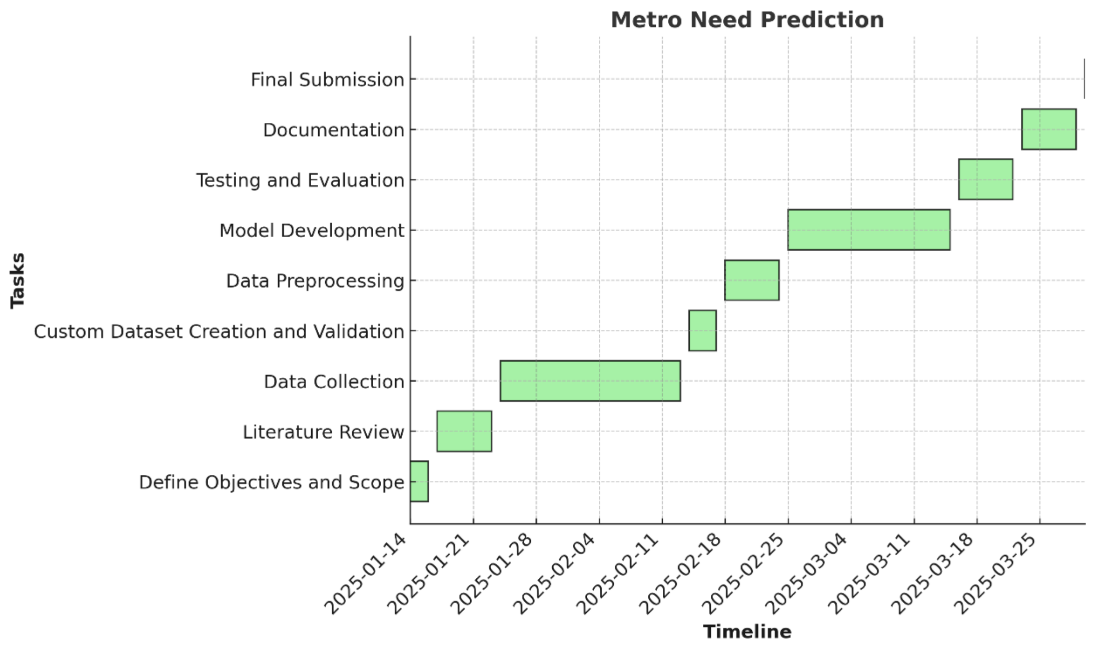


Fig. 2.5.1. Gantt chart

Phase 1: Data Collection and Pre-processing

Goals:

- Gather relevant datasets
- Combine the datasets to make a single dataset
- Clean and pre-process data for model input
- Identify data sources and obtain necessary permissions

Tasks:

1. **Identify Data Sources**:
    - Traffic congestion data: Collect traffic congestion data, including traffic density, congestion patterns, and traffic-related delays, from government sources and transport departments.
    - Public transport data: Gather data on existing metro and other public transport infrastructure.

    - Additional data: Collect data on population, district-level demographics, and air quality metrics from government websites, census reports, and open datasets.
    - Average trip time & commute length data: Collect the average trip time and trip length data of areas.
2. **Data Pre-processing**:
    - Clean Data: Handle missing data, remove outliers, and fill gaps in datasets.
    - Data Transformation: Normalize or standardize datasets (e.g., scaling numerical features).
    - Feature Selection: Select the top features based on the Cross Validation Scores for each model.
3. **Data Integration**:
    - Merge the separate datasets into a single dataset based on geographic location.

Deliverables:

- Cleaned and pre-processed dataset
- Document detailing the sources and pre-processing steps

Phase 2: Model Development and Initial Training

Goals:

- Develop initial predictive models
- Train the model using historical data
- Validate the models' performances on sample cities

Tasks:

1. **Model Selection:**
    - Evaluate machine-learning algorithms such as Decision Trees, Random Forests, and Support Vector Machines to predict metro requirement.
    - Decide on a specific machine-learning model for classification or regression tasks based on available target variables.
2. **Model Development:**
    - Split the data into training, validation, and test sets for model evaluation.
    - Use feature selection methods to identify the most important variables for prediction (e.g., population density, traffic congestion).
    - Train models on the training dataset and evaluate using validation sets.
    - Fine-tune hyper parameters and evaluate the models on test data.

3. **Validation Using Historical Data:**
    - Train the model using data from selected cities (Chennai, Delhi and Bangalore) to assess how well the model predicts outcomes like metro ridership, traffic congestion, and air quality.
    - Compare the model's predictions against historical trends to assess the model's predictive accuracy.
4. **Model Evaluation:**
    - Evaluate the models using key metrics such as accuracy, F1 score, and mean squared error (for regression tasks).
    - Perform cross-validation to ensure robustness.

Deliverables:

- Initial model prototype with basic predictions
- Report on model performance, including accuracy and any insights gained

Phase 3: Model Testing and Refinement

Goals:

- Improve the model's accuracy and efficiency
- Incorporate additional factors (district-specific features) into the model
- Ensure robustness and accuracy across all cities

Tasks:

1. **Refine Model Parameters:**
    - Fine-tune hyper parameters (e.g., tree depth, learning rate) using grid search or random search.
    - Incorporate additional data like district-specific population density, traffic patterns, or air quality metrics to improve predictions.
2. **Additional Feature Engineering:**
    - Generate new features that might be relevant, such as time-of-day traffic patterns, historical growth rates of ridership, and weather data.
    - Create interaction terms between features to capture relationships (e.g., traffic congestion * air quality).
3. **Model Testing:**
    - Conduct extensive testing using different subsets of the data (e.g., city-specific, district-specific) to validate the model's consistency across regions.
    - Test the model's robustness by simulating "what-if" scenarios (e.g., increase in metro capacity, addition of new stations).
4. **Model Optimization:**
    - Apply advanced techniques like feature scaling, ensemble methods (e.g., boosting or bagging), and neural networks to further optimize the model.

    - Check for overfitting and under fitting, and adjust model complexity accordingly.

Deliverables:

- Refined model with improved accuracy and robustness
- Report on model performance with refined predictions

Phase 4: Final Deployment and Reporting

Goals:

- Deploy the optimized model for real-world use
- Provide actionable insights for metro line optimization
- Generate a detailed final report

Tasks:

1. **Deployment:**
    - Develop a deployment plan for the model, including integration with city planning tools, transport departments, and real-time data feeds.
    - Implement the model using a cloud-based or local deployment architecture (e.g., Flask for web interface or API integration).
    - Test deployment in a controlled environment with real-time data.
2. **Actionable Insights:**
    - Provide actionable recommendations for metro line improvements, such as optimizing routes based on predicted ridership, or adjusting train schedules based on traffic congestion patterns.
    - Highlight areas in each city where new metro lines could reduce congestion and improve air quality.
3. **Final Reporting:**
    - Summarize the entire process: data collection, model development, testing, and insights.
    - Present a comprehensive report including methodology, results, and actionable insights.
    - Visualize findings using maps, graphs, and charts to clearly communicate insights to stakeholders.
4. **Presentation to Stakeholders:**
    - Prepare a presentation for key stakeholders such as urban planners, transportation authorities, and city officials.
    - Discuss model outcomes, potential implementation strategies, and expected benefits.

Deliverables:

- Deployed model with real-time data capabilities
- Detailed final report with insights on metro line optimization
- Presentation for stakeholders

Chapter 3

3. TECHNICAL SPECIFICATIONS

3.1 FUNCTIONAL REQUIREMENTS

1) Data Collection: The system must gather traffic congestion data, carbon emissions data, and public transport usage statistics from various sources such as government APIs, environmental sensors, and open data platforms, which are known to be reliable sources for urban data collection. The system should integrate real-time data feeds for continuous updates, ensuring that the model can react to dynamic urban conditions. Data must be aggregated at various levels (district, city-wide) to accommodate different analysis scopes, which is a critical feature for effective urban planning (Chen et al., 2020).

2) Data Processing: The system should include automated techniques to detect and handle erroneous or missing data, ensuring the integrity of the datasets used for analysis. It is essential for the system to support transformation processes such as normalization and feature encoding, which are crucial for improving model accuracy (Jain et al., 2020). Integration across disparate datasets (traffic, emissions, etc.) will be necessary to build a comprehensive dataset for analysis, which is a common approach in data-driven urban systems.

3) Model Development: The system should implement machine learning algorithms like regression, decision trees, or support vectors to predict traffic and emissions, techniques proven to be effective in urban applications. Tools for hyper parameter tuning (grid search, random search) will allow users to optimize models and improve predictive performance, as seen in numerous machine learning applications.

4) User Interaction: An intuitive user interface is essential for enabling users to input parameters and visualize results, even for those without technical expertise. The system should allow urban planners to test various scenarios (e.g., new metro lines), providing insights into the potential impacts of such changes on traffic and emissions. Real-time visualization of predictions through maps, heat maps, and graphs is necessary to help users understand complex data and make informed decisions.

5) Reporting: The system must automatically generate detailed reports summarizing key findings and predictions, which is a typical feature of modern data analytics systems. Users should be able to tailor reports to focus on specific parameters or cities, enabling a more personalized experience.

6) Testing and Validation: The system should evaluate the predictive models using performance metrics such as MSE or RMSE, commonly used to assess the accuracy of machine learning models in prediction tasks. Cross-validation methods, such as k-fold validation, are necessary to ensure the robustness of the models across different data subsets.

3.2 NON-FUNCTIONAL REQUIREMENTS

1) Performance: The system must process data and provide predictions within a short timeframe (e.g., under 5 minutes for large datasets), ensuring it meets the demands of real-time decision-making. Optimized querying is crucial for handling large datasets, especially in urban environments with vast amounts of traffic and emissions data.

2) Scalability: The system should easily scale to accommodate additional cities or districts, a common requirement in systems designed for urban environments. The system should be able to handle growing data volumes and increasing user demands by scaling horizontally, using cloud infrastructure (Hasan et al., 2021).

3) Reliability: The system must aim for a high prediction accuracy (at least 85%) to ensure that the outputs are reliable for urban planning decisions. The system should maintain at least 99.5% uptime, ensuring continuous service availability.

4) Usability: A simple, intuitive interface will enhance user experience, particularly for urban planners with varying levels of technical expertise. The interface should be accessible on both desktop and mobile devices, providing flexibility for users.

5) Security: Sensitive data should be encrypted to protect it during transmission and storage, ensuring compliance with data protection standards (Sharma et al., 2021). The system must implement role-based access control and secure login mechanisms to protect against unauthorized access.

3.3 FEASIBILITY STUDY

The feasibility study assesses the viability of the Metro Planning project by examining the technical, economic, and social feasibility.

3.3.1 Technical Feasibility

Technical feasibility evaluates the practicality of developing and implementing the proposed system using available technologies and resources. This analysis covers data collection, model development, user interface, and testing to ensure that the solution can be effectively executed.

1. **Data Collection and Integration:**
    - **Data Availability and Access:** A wealth of public datasets is available from government sources, city transport departments, environmental sensors, and APIs. Data related to traffic congestion, carbon emissions, and public transportation can be accessed via platforms like OpenStreetMap, government portals (e.g., India's Open Government Data Platform), and sensor networks (e.g., Smart City Projects). Many cities have also started integrating sensors for real-time traffic monitoring, making it feasible to collect dynamic data.
    - **Data Integration and Processing:** Modern data processing frameworks such as Apache Kafka and Apache Spark are designed to handle large, real-time data streams. Kafka enables the integration of diverse data sources, while Spark's distributed computing capabilities make it ideal for processing high-volume datasets. These tools allow for the efficient handling of traffic, emissions, and transport data in real-time, ensuring scalability and robustness as data grows. They also facilitate data pre-processing, cleaning, and transformation, which are critical for predictive accuracy.
    - **Scalability and Real-Time Data Handling:** Cloud infrastructure and microservices architectures will enable the system to scale easily. Technologies like Docker and Kubernetes can support containerization, allowing for flexible deployment across cloud platforms (e.g., AWS, Google Cloud, Microsoft Azure). Real-time data handling through these platforms will ensure that the model's predictions and traffic management recommendations are delivered in a timely manner.
2. **Model Development:**
    - **Predictive Modelling:** Advanced machine learning techniques such as decision trees, neural networks (e.g., deep learning for complex pattern recognition), and regression models will be used to predict traffic congestion and emissions. These models will be trained on historical traffic, emissions, and transport data to understand patterns and predict future trends. Libraries like Scikit-learn for decision trees, TensorFlow for neural networks, and XGBoost for gradient boosting models are well-established in the industry.
    - **Computational Power:** Cloud computing platforms like AWS EC2, Google Cloud, and Microsoft Azure offer the computational power necessary for training complex models, conducting data processing, and making real-time predictions. These platforms provide on-demand access to high-performance CPUs and GPUs, which are critical for machine learning tasks, especially deep learning and neural networks.
    - **Model Optimization and Retraining:** Regular model retraining and optimization will ensure that the predictions remain accurate as new data is collected. Cloud computing can easily accommodate the need for frequent model retraining by providing scalable infrastructure.

3. **User Interface and Visualization:**
    - **Data Visualization:** Interactive visualizations will be essential for urban planners, government officials, and other stakeholders to understand the model’s outputs. Open-source tools such as D3.js, Leaflet.js, and Plotly provide robust frameworks for creating dynamic, interactive maps and charts. For example, traffic congestion data can be displayed on geographic maps using Leaflet.js, and emission levels can be visualized using heatmaps with D3.js.
    - **User Experience:** React.js, Angular, and Vue.js will be used to build responsive, intuitive user interfaces that allow non-technical users to interact with the system. These frameworks are widely used in the industry and provide real-time interactivity, enabling urban planners and policymakers to input parameters, run simulations, and visualize predictions effortlessly.
    - **Mobile and Cross-Platform Accessibility:** To cater to a wide range of users, the system can be made accessible on mobile devices using frameworks like React Native, allowing urban planners and decision-makers to access real-time data and insights while on the move.
4. **Testing and Validation:**
    - **Model Evaluation:** The machine learning models will be rigorously tested using techniques like k-fold cross-validation, ensuring that the model’s predictions are not biased toward any particular dataset. Performance metrics such as Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) will be used to assess the accuracy of the models. These metrics are industry standards for regression tasks and will help evaluate model effectiveness across different scenarios.
    - **Robustness Under Different Scenarios:** The system will be tested under various urban planning scenarios, including changes in traffic patterns, air quality levels, and population growth. These tests will ensure that the system remains reliable and can handle a variety of inputs, providing consistent performance even in complex urban environments.

The technical feasibility of the project is highly favourable, thanks to the availability of open data, well-established machine learning frameworks, and scalable cloud infrastructure. The tools required for implementation are mature, and the necessary expertise is readily accessible.

3.3.2 Economic Feasibility

Economic feasibility focuses on assessing whether the project is financially viable. This involves evaluating the cost of development, ongoing operational expenses, and the potential for revenue generation and long-term financial sustainability.

1. **Initial Costs:**
    - **Development and Personnel Costs:** The project will require skilled professionals, including data scientists, machine learning engineers, urban planners, and web developers. Costs associated with hiring a team of qualified personnel will be significant, particularly for specialized roles like data scientists and machine learning experts. Moreover, acquiring licenses for proprietary software or APIs (e.g., paid datasets, third-party analytics platforms) will add to the initial investment.
    - **Cloud Infrastructure Costs:** Cloud computing platforms typically charge based on usage. The costs of using AWS, Google Cloud, or Azure for data storage, processing, and model training can vary, but on-demand instances (e.g., EC2 for AWS) and managed services (e.g., BigQuery on Google Cloud) provide a flexible pricing model. The cost will scale with the size of the datasets and frequency of model retraining, but pricing structures generally offer cost-effective solutions for initial deployment.
2. **Operational Costs:**
    - **Data Acquisition Costs:** Many datasets are available for free through government portals or open data platforms. However, real-time data from sensors, commercial traffic data providers, or additional premium datasets (e.g., satellite images, third-party air quality monitoring services) may incur ongoing costs. Additionally, the system will require continuous access to data feeds to maintain its predictions.
    - **Maintenance and Updates:** As the system is used, there will be a need for regular updates and maintenance. This includes refreshing the underlying models with new data, refining the predictive algorithms, and improving the user interface based on feedback. These activities will require both financial investment and developer time.
3. **Revenue Generation:**
    - **Cost Savings for Cities:** The tool can help cities optimize transportation networks by predicting traffic congestion and emissions, which can lead to significant cost savings. By streamlining public transport routes and schedules, improving traffic management, and reducing the impact of congestion on productivity, municipalities can lower their operational costs.
    - **Partnerships and Collaborations:** The project can generate revenue through public-private partnerships. Collaboration with government

    agencies, transport authorities, and private companies in urban mobility (e.g., ride-sharing services, infrastructure providers) can provide funding to offset operational costs.
  - **Smart City Initiatives:** Many governments are keen to support smart city initiatives. The system may be eligible for government grants and funding focused on improving urban mobility, reducing carbon emissions, and creating data-driven urban planning tools.
4. **Long-Term Financial Sustainability:**
   - **Predictive Analytics as a Service:** Once the system is developed and tested, it could be offered as a commercial service to municipalities, transportation departments, and private companies involved in urban development and infrastructure projects. This would generate a recurring revenue stream, which could help maintain and expand the system.
   - **Continued Government Funding:** Many governments fund projects that align with sustainability and smart city goals. This can provide continued financial support for both development and scaling efforts.

The economic feasibility is strong, as the project promises significant cost savings for cities, potential revenue from public-private collaborations, and sustainability through long-term partnerships. While initial investment is required, the system's ability to generate revenue and deliver savings will make it financially viable in the long term.

3.3.3 Social Feasibility

Social feasibility evaluates the potential social impact of the system, its ability to gain public acceptance, and how it benefits society, particularly in urban contexts.

1. **Improving Urban Quality of Life:**
   - **Traffic Congestion Reduction:** By predicting and mitigating traffic congestion, the system will lead to smoother traffic flow and shorter commute times. This can reduce stress for urban residents and improve overall productivity. Studies have shown that cities with optimized traffic management systems experience higher levels of satisfaction among commuters (Hamed et al., 2020).
   - **Environmental Impact:** Reducing traffic congestion directly translates into lower carbon emissions, improving air quality in dense urban areas. This will have significant health benefits, particularly in cities like Delhi and Bangalore, where air pollution is a serious issue. Cleaner air will lead to reduced healthcare costs and an overall improvement in public health.

- **Public Transport Optimization:** The predictive models can help urban transport authorities optimize bus and metro schedules, improving efficiency and reducing waiting times for passengers. This would make public transport more attractive, encouraging higher usage and reducing the reliance on private vehicles.

2. **Public Awareness and Participation:**
    - **Transparency and Informed Decision-Making:** The system will provide urban planners and the public with detailed data on traffic patterns, emissions, and public transport usage. This transparency can help build trust in government decision-making processes, as citizens will have access to the data behind infrastructure projects and urban planning decisions.
    - **Enhanced Citizen Engagement:** By sharing actionable insights, the system can foster more inclusive decision-making. Residents can provide feedback on predictions and transportation recommendations, further empowering them to shape urban development plans.
3. **Equity and Accessibility:**
    - **Serving Underserved Communities:** By incorporating public transport data and ensuring that emission reduction strategies are equally applied across all areas of a city, the system will promote greater equity. Public transport is often more critical for low-income communities, and better planning and management will ensure that services are accessible to all. The system can also help optimize the location of infrastructure like bus stops, stations, and roads to benefit underserved areas.
    - **Sustainability and Social Impact:** Reducing emissions and improving air quality will directly benefit marginalized communities that tend to be located near high-traffic zones. By addressing these disparities, the system can improve social equity and quality of life.
4. **Acceptance of New Technology:**
    - **User-Friendliness and Accessibility:** With a user-friendly interface, urban planners, policymakers, and the general public will find it easy to interact with the system. This can increase public acceptance and participation. Governments are increasingly focused on sustainable urban development, which means this system aligns well with societal goals for smart cities, making it easier to gain political and public support.

The social feasibility is robust, with the project aligning with urban sustainability goals, improving quality of life, and providing equitable benefits across communities. The system's transparency and accessibility will ensure broad public support, making it highly feasible from a social impact perspective.

3.4 SYSTEM SPECIFICATION

System specification defines the hardware and software requirements for Metro Planning to function effectively and provide a seamless user experience.

3.4.1 Hardware Specification

The hardware requirements must support the processing of large datasets related to urban planning attributes like population, weather conditions, and hazards (earthquake, wind, floods). The system must also support machine learning model training and inference.

1) Cloud Infrastructure: Utilize virtual machines or containers within a cloud provider (e.g., AWS, Google Cloud, or Azure).

- **CPU:** Multi-core processors (16+ cores) for efficient handling of large-scale computations.
- **Memory:** 64 GB RAM or higher for smooth processing of datasets and model training.
- **Storage:** Scalable SSD storage (1 TB+) for storing urban data attributes, models, and logs, with options for on-demand scaling.
- **GPU:** NVIDIA A100 or V100 GPUs for accelerating ML model training, especially for deep learning models or simulations.

2) Local Devices:

- **Processor:** Intel i7 (12th Gen) or AMD Ryzen 7 equivalent.
- **Memory:** 16 GB RAM or higher.
- **Storage:** 500 GB SSD.
- **GPU:** NVIDIA RTX 3060 or equivalent for testing.

3.4.2 Software Specification

The software requirements focus on enabling efficient data processing, machine learning model deployment, and interactive user interfaces.

1) Operating System:

- **Server OS:** Linux-based (Ubuntu or CentOS) for compatibility with machine learning libraries and cloud services.
- **Client OS:** Cross-platform compatibility for web and mobile applications (iOS, Android, Windows, macOS).

2) Core Technologies:

- **Machine Learning Frameworks:**
    - **Scikit-learn** for building predictive models.
    - **XGBoost** or **LightGBM** for handling structured data efficiently.
- **Database Systems:**
    - **PostgreSQL** or **MySQL** for structured data storage (e.g., population data).
    - **Geospatial Database (PostGIS):** For handling location-based data.
- **ETL Pipelines:**
    - Tools like **Apache Airflow** for automating data preprocessing workflows.

3) Data Processing Tools:

- **Pandas and NumPy:** For data manipulation and analysis.
- **Weather APIs:** For fetching real-time weather data (e.g., OpenWeatherMap API).

4) Machine Learning Tools:

- **Transformers:** If NLP is involved in processing unstructured city data.
- **GIS Software:** Integration with tools like QGIS or ArcGIS for visualizing urban planning outputs.

Chapter 4

4. DESIGN APPROACH AND DETAILS

4.1 SYSTEM ARCHITECTURE

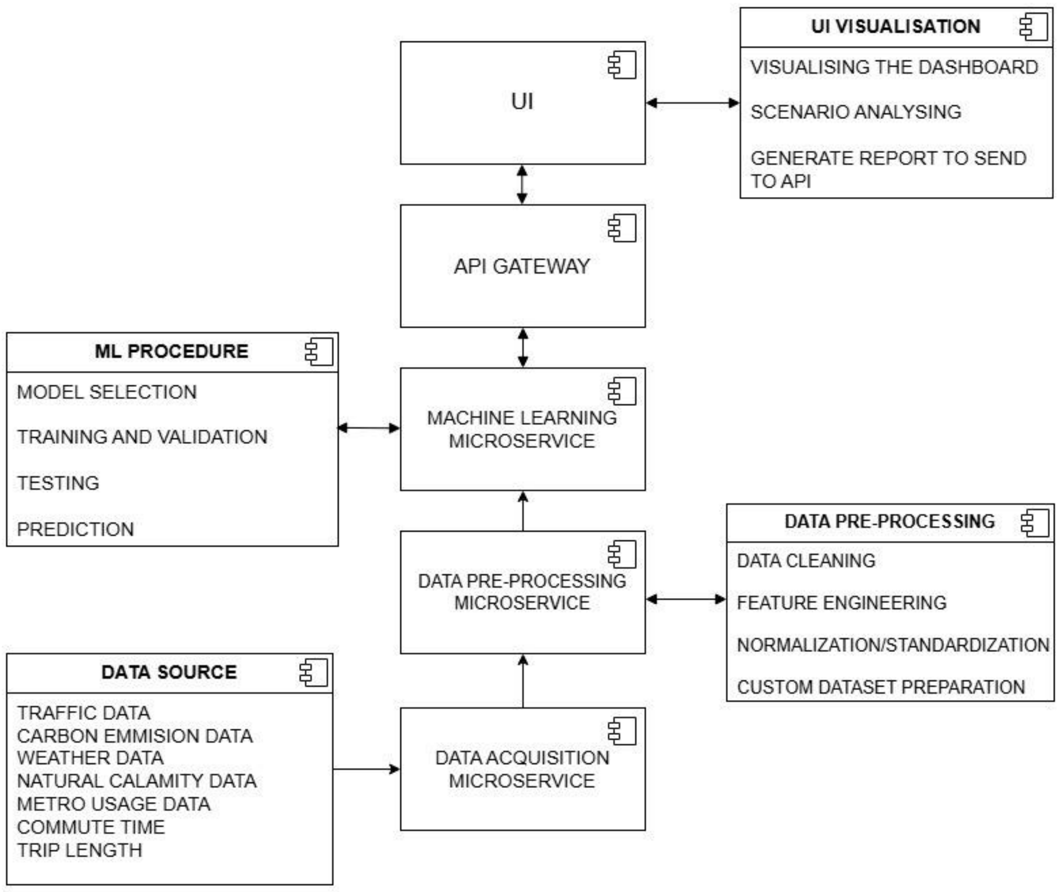


Fig. 4.1.1. System Architecture

4.1.1 Data Acquisition Microservice

This microservice is critical for gathering diverse datasets required for urban city planning, focusing on traffic, emissions, and public transport.

4.1.1.1 Functionality

1. **Traffic Data:**
   - Collect real-time and historical data from traffic monitoring systems, IoT sensors, and third-party traffic databases (e.g., TomTom, Google Maps API).
   - Incorporate data from smart city initiatives, such as vehicle counts, road usage patterns, and congestion levels.

2. **Carbon Emission Data:**
    - Gather data from environmental monitoring agencies (e.g., EPA), vehicle emission reports, and satellite data sources (e.g., Copernicus).
    - Integrate with IoT-enabled air quality monitoring stations and emission prediction studies.
3. **Public Transport Usage Data:**
    - Retrieve ridership and operational data from metro systems, bus networks, and shared mobility services (e.g., Uber, Lyft).
    - Incorporate GPS data from public transport vehicles for route optimization.

4.1.1.2 Technologies

1. **Web Scraping:**
    - Tools: Selenium, Beautiful Soup, or Scrapy for scraping structured data from public databases or transport websites.
    - Automation: Scheduling scripts using Cron Jobs or Task Schedulers for periodic data updates.
2. **APIs:**
    - Third-party APIs: Google Maps API, OpenWeather API, or traffic management systems APIs for seamless integration.
    - Custom integrations: Develop RESTful endpoints to connect to proprietary data sources securely.

4.1.2 Data Pre-Processing Microservice

This microservice handles the transformation of raw data into a clean and analysis-ready format.

4.1.1.1 Functionality

1. **Combining Dataset:**
    - Merge various datasets into a single unified dataset using *District* as the common parameter.
    - Ensure consistency in column names, data types, and categorical value formats across the dataset.
    - Handle missing or duplicate entries to avoid data distortion during merging.
    - Validate successful merging by checking for expected row and column counts post-integration.
2. **Data Cleaning:**
    - Remove duplicates, handle null values with imputation techniques, and standardize formats for compatibility across datasets.

    - Address outliers using statistical or ML-based anomaly detection methods.
    - Standardize text data (e.g., casing, spacing, special characters) for uniformity.
    - Drop redundant or irrelevant features.
3. **Feature Engineering:**
    - Generate predictive features like traffic density, emission per vehicle type, peak-hour patterns, and weekday/weekend trends.
    - Aggregate spatial data to specific zones (e.g., neighborhoods, metro zones) for focused analysis.
    - Convert categorical variables into numerical formats using encoding techniques such as one-hot encoding or label encoding.
4. **Normalization/Standardization:**
    - Normalize variables like traffic volume, emissions, and public transport ridership for uniform scale in model input.
    - Use log transformations or scaling techniques like MinMaxScaler or StandardScaler.

2. Technologies

1. **Python Libraries:**
    - **Pandas:** For handling structured data manipulation.
    - **NumPy:** For efficient numerical calculations.

.

4.1.3 Machine Learning Microservice

The machine learning microservice is at the core of predictive analytics for traffic and emissions.

4.1.3.1 Functionality

1. **Model Selection:**
    - Explore models like Logistic Regression, Random Forest, Gradient Boosting, and XGBoost for structured data.
2. **Feature Selection:**
    - Use automated techniques such as Recursive Feature Elimination with Cross Validation (RFECV) and Feature Importance Ranking.
    - Focus on maximizing model performance while minimizing overfitting and redundancy.
3. **Training and Validation:**
    - Implement k-fold cross-validation, hyperparameter tuning (e.g., Grid Search or Bayesian Optimization), and feature importance analysis.

- Include evaluation metrics like R^2, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

4. **Prediction:**
    - Predict metrics like future traffic congestion, emission levels, and public transport usage under various scenarios.
    - Enable real-time prediction using batch or streaming inference.

4.1.3.2 Technologies

1. **Frameworks:**
    - **Scikit-learn:** For classical ML algorithms.
2. **Tools:**
    - **Google Collaboratory Notebook:** For model experimentation and prototyping.

4.1.4 API Gateway

The API gateway serves as the communication hub, linking microservices with external clients and the user interface.

4.1.4.1 Functionality

1. **Centralized Access:**
    - Provide unified RESTful endpoints for accessing processed data, ML model predictions, and visualizations.
2. **Load Balancing:**
    - Ensure efficient handling of concurrent requests using load balancers.
3. **Authentication and Security:**
    - Use OAuth2.0 or JWT for secure access control and role-based permissions.

4.1.4.2 Technologies

1. **Frameworks:**
    - **FastAPI** for high-performance RESTful APIs.
    - **Flask** for lightweight microservices.
2. **Middleware:**
    - Integrate with tools like Nginx for API traffic management.

4.1.5 User Interface

The user interface ensures usability, providing stakeholders with actionable insights and interactive tools.

4.1.5.1 Functionality

1. **Visualization Dashboard:**
    - Real-time maps for traffic and emissions data using geospatial visualizations.
    - Historical trends with filtering options (e.g., date, location, and traffic zones).
2. **Scenario Analysis:**
    - Interactive tools to simulate different urban planning scenarios (e.g., new metro line routes, alternate road networks).
    - Visualize the impact of scenarios on traffic flow and emission levels.
3. **Report Generation:**
    - Export results into PDFs, spreadsheets, or interactive web formats for stakeholder presentations.

4.1.5.2 Technologies

1. **Frontend Frameworks:**
    - **React.js or Angular:** For responsive and interactive web applications.
2. **Visualization Libraries:**
    - **D3.js, Chart.js, or Plotly:** For dynamic charts, graphs, and geospatial plots.
    - **Leaflet or Mapbox:** For map-based visualizations.

4.2 DESIGN

4.2.1 Data Flow Diagram

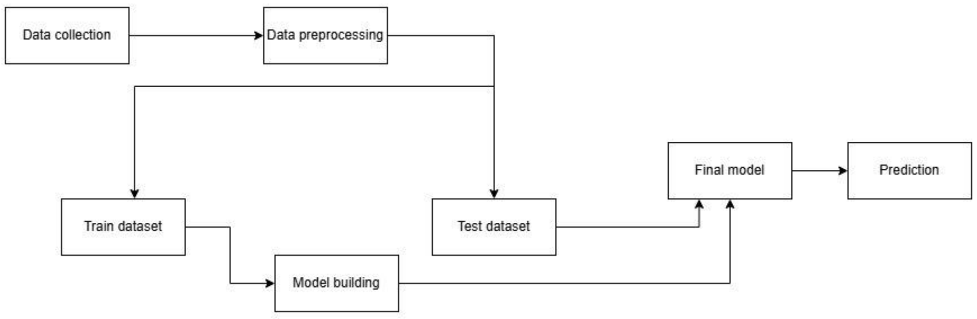


Fig 4.2.1. Data Flow Diagram

1) Data Collection

The **Data Acquisition Microservice** is responsible for gathering and aggregating diverse datasets critical for traffic and emissions analysis.

Workflow:

1. **Traffic Data Collection:**
    - Collects historical traffic data from city databases to identify patterns and trends over time.
2. **Emissions Data Collection:**
    - Aggregates emissions data from environmental monitoring agencies, IoT-enabled air quality sensors, and vehicle registration databases.
    - Incorporates satellite data (e.g., NASA or Copernicus) for macro-level emissions trends.
3. **Public Transport Data:**
    - Retrieves ridership statistics, schedules, and vehicle usage data from metro and bus services.
    - Integrates shared mobility data (e.g., bike sharing, ride hailing) for comprehensive urban mobility insights.
4. **Combining Datasets:**
    - Merge various datasets into a single unified dataset using *District* as the common parameter.
    - Ensure consistency in column names, data types, and categorical value formats across the dataset.
    - Handle missing or duplicate entries to avoid data distortion during merging.
    - Validate successful merging by checking for expected row and column counts post-integration.

Key Features:

- Automation of data retrieval via APIs and web scraping.
- Data storage in scalable cloud databases for efficient access and management.

2) Data Processing

The **Data Processing Microservice** ensures raw data is clean, structured, and ready for downstream tasks like machine learning and visualization.

Workflow:

1. **Data Cleaning:**
    - Detects and removes duplicates, fills missing values using statistical or ML-based imputation, and resolves inconsistencies in format.

    - Identifies and mitigates outliers using advanced statistical tests or anomaly detection models.

2. **Feature Engineering:**
    - Generates derived features, such as traffic density, emissions per capita, and peak-hour traffic indicators.
    - Aggregates data spatially (e.g., neighbourhoods, districts) and temporally (e.g., hourly, daily).
3. **Normalization/Standardization:**
    - Scales variables to ensure compatibility with machine learning algorithms, using techniques like MinMaxScaler or Z-score normalization.

Key Features:

- Leverages robust ETL pipelines for automated data integration and transformation.
- Utilizes libraries like Pandas, NumPy, and GeoPandas for scalable processing.

3) Model Training

The **Machine Learning Microservice** builds predictive models for traffic congestion and emissions analysis, enabling data-driven urban planning.

Workflow:

1. **Model Selection and Development:**
    - Evaluates traditional ML models (e.g., Random Forest, XGBoost) to identify the best-fit model.
    - Uses domain-specific features and historical data for model training.
2. **Training and Validation:**
    - Applies k-fold cross-validation and hyperparameter optimization techniques to improve model accuracy and prevent overfitting.
    - Utilizes techniques like SHAP or LIME to ensure model interpretability.
3. **Model Deployment:**
    - Deploys trained models as RESTful endpoints accessible through the API Gateway for real-time or batch predictions.

Key Features:

- Tracks and versions models using MLflow or DVC for reproducibility.
- Offers continuous learning capabilities by periodically retraining models with new data.

4) Predictions

Trained models provide actionable insights through predictive analytics, enabling proactive decision-making.

Workflow:

1. **Input Handling:**
    - Receives new input parameters, such as proposed metro routes or vehicle regulations, through the API Gateway.
2. **Real-Time Prediction:**
    - Generates forecasts for metrics like traffic density, emissions levels, and public transport efficiency.
    - Supports "what-if" scenario analyses, allowing stakeholders to evaluate potential urban planning strategies.
3. **Output Delivery:**
    - Serves predictions in a structured format (JSON/CSV) for integration with visualization tools or reporting systems.

Key Features:

- Highly scalable to handle concurrent prediction requests.
- Provides confidence intervals or uncertainty metrics for each prediction.

5) User Interaction

The **User Interface (UI)** serves as the primary touchpoint for stakeholders, offering an intuitive and interactive platform to explore predictions and data insights.

Workflow:

1. **Data Visualization:**
    - Real-time dashboards display traffic trends, emissions data, and model predictions using dynamic charts, maps, and graphs.
    - Geospatial visualizations (e.g., heatmaps, road congestion overlays) provide actionable insights at a glance.
2. **Scenario Analysis:**
    - Users input parameters (e.g., proposed infrastructure changes, policy adjustments) and instantly view their impact.
    - Simulations offer comparisons of multiple scenarios, highlighting optimal solutions.
3. **Reporting and Export:**
    - Generates detailed, customizable reports in formats like PDF, Excel, or interactive HTML for stakeholder presentations.

- Allows users to export raw data or visualized insights for further analysis.

Key Features:

- Built using modern frontend frameworks (React.js, Angular) for responsive and user-friendly design.
- Integrates with APIs to provide seamless communication with backend services.
- Employs libraries like Plotly, Chart.js, and Leaflet for advanced data visualization.

4.2.2 Use Case Diagram

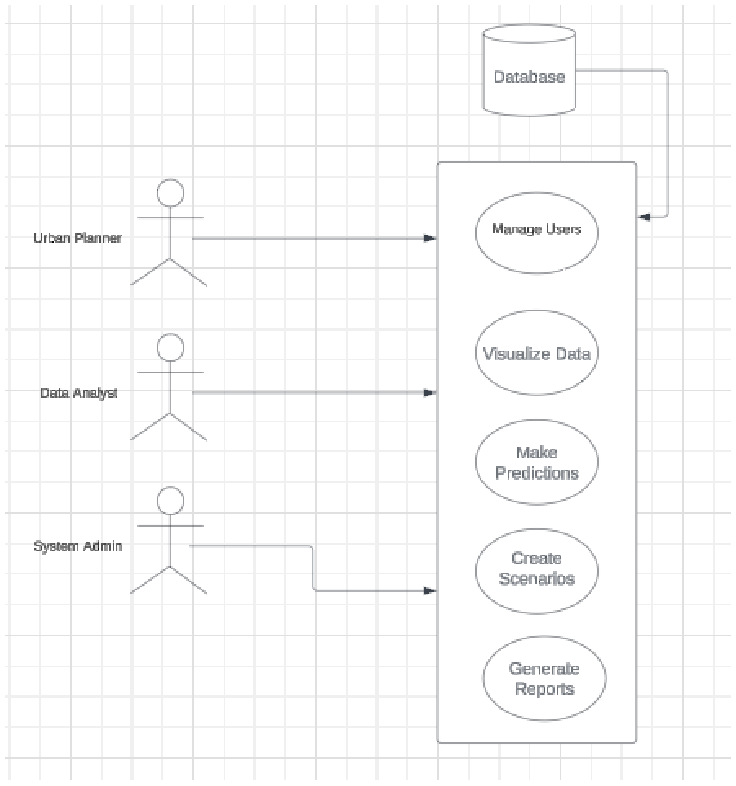


Fig 4.2.2. Use Case Diagram

The diagram represents a system designed for urban planning and decision-making. It demonstrates how different types of users interact with the system's core functionalities to achieve their goals. The system integrates data management, analytics, predictive modelling, and reporting to assist stakeholders in making informed urban planning decisions.

1) Actors and Their Roles

1. **Urban Planner**
   - **Role:** Urban Planners are key decision-makers responsible for planning city infrastructure, optimizing public transport, reducing traffic congestion, and minimizing environmental impact.
   - **Interaction with the System:**
     - **Manage Users:** Urban Planners can oversee system access by managing user accounts, assigning roles, and ensuring appropriate permissions are granted.
     - **Visualize Data:** They can view real-time and historical data insights, such as traffic trends and carbon emissions, to inform their decisions.
     - **Create Scenarios:** Urban Planners can simulate the impact of proposed policies or infrastructure projects, such as a new metro line or road widening, and analyze their feasibility.
   - **Goals:**
     - Streamline urban planning processes with accurate data.
     - Evaluate the potential impact of proposed projects on traffic and emissions.
2. **Data Analyst**
   - **Role:** Data Analysts focus on understanding the data collected by the system, interpreting trends, validating predictions, and producing insights for stakeholders.
   - **Interaction with the System:**
     - **Visualize Data:** Data Analysts use dashboards and charts to explore traffic, emissions, and public transport data trends.
     - **Make Predictions:** They input variables (e.g., traffic volume or new routes) to generate predictive analytics for congestion and emissions.
     - **Generate Reports:** Analysts produce detailed reports summarizing trends, predictions, and scenario results for presentations or further analysis.
   - **Goals:**
     - Enhance decision-making by deriving actionable insights.
     - Provide accurate, data-driven reports to stakeholders.
3. **System Admin**
   - **Role:** The System Admin oversees the technical functioning of the system, ensuring that users have access to the necessary tools and data while maintaining the integrity and security of the system.
   - **Interaction with the System:**
     - **Manage Users:** System Admins add, remove, and manage user accounts, ensuring proper access control and security protocols.

- **Goals:**
    - Ensure the system operates efficiently and securely.
    - Manage user permissions to prevent unauthorized access.

2) System Functionalities

The system provides the following key functionalities to support urban planning and decision-making:

1. **Manage Users:** Enables administrators and planners to manage system users and their roles.
    - Functions:
        - Add or remove user accounts.
        - Assign roles (e.g., planner, analyst, admin).
        - Monitor user activity for compliance and security.
2. **Visualize Data:** Provides interactive dashboards and charts to display real-time and historical data trends.
    - Features:
        - Heatmaps showing traffic congestion across different areas.
        - Charts tracking emissions levels over time.
        - Real-time updates from traffic monitoring systems and environmental sensors.
    - Importance:
        - Helps users identify problem areas, trends, and anomalies.
3. **Make Predictions:** Uses machine learning models to predict future traffic congestion and emissions levels based on user-defined parameters.
    - Features:
        - Users input parameters (e.g., proposed metro lines, traffic policies).
        - Provides confidence intervals and insights into the reliability of predictions.
    - Importance:
        - Empowers stakeholders to make proactive, data-driven decisions.
4. **Create Scenarios:** Allows users to simulate and evaluate the impact of hypothetical urban planning decisions.
    - Features:
        - Simulate the impact of adding a new metro route or altering traffic flow.
        - Compare multiple scenarios side by side.
    - Importance:

        - Helps planners choose the best course of action for urban development.
5. **Generate Reports:** Produces detailed, customizable reports summarizing data insights, predictions, and scenario outcomes.
    - Features:
        - Export reports in various formats (e.g., PDF, Excel).
        - Include visualizations, trends, and scenario results in reports.
    - Importance:
        - Facilitates effective communication with stakeholders and ensures transparency in decision-making.

3) Relationships and Interactions

- **Urban Planner to System:** Focuses on data visualization and scenario creation for strategic planning. Relies on the system for accessible, actionable insights. Collaborates with Data Analysts for scenario evaluation and reports.
- **Data Analyst to System:** Engages deeply with data visualization, predictions, and report generation. Uses predictive insights to validate or refine planning strategies.
- **System Admin to System:** Manages user roles and ensures the system is secure, reliable, and accessible.
- **Database Role:** Acts as the central storage hub for all system data, including traffic patterns, emissions data, and user activity logs. Ensures seamless data flow between functionalities like predictions and visualizations.

Chapter 5

5. METHODOLOGY AND TESTING

To address the binary classification problem of predicting whether a district will have metro infrastructure by 2025, we used a dataset covering all states and the 629 districts as of the 2011 census. The dataset comprised features such as State, District, risk of natural calamities (including earthquakes, floods, or wind hazards), number of households, population, decadal growth rate [13], area (in sq. km), air quality index (AQI), current infrastructure, average commute time, proximity to the nearest airport (within 100 km), and average trip length. The target variable, *Metro Present in 2025*, determines whether a metro line exists ("Yes") or not ("No") in a specific district. In order to handle missing values and encode categorical variables as necessary, we first collected and cleaned the data. To train classification models, relevant variables such as population, commuting duration, and distance-related attributes were used. We tried methods like Random Forest Classifier and Logistic Regression. Accuracy, precision, and recall were used to evaluate the model's performance, and additional hyperparameter tuning was done to maximize outcomes.

5.1 DATA COLLECTION AND PREPARATION

To create a predictive model for identifying districts that need metro line infrastructure, we chose 10 features that were considered relevant based on domain knowledge and urban planning concerns. The dataset was built to represent all 629 districts of India as per the 2011 census, ensuring comprehensive geographical and demographic coverage. The data collection approach involved gathering information from open-source databases, academic research, and manual verification.

1. **Demographic and Population data:** Population and Demographics data were obtained from open-source Indian Census dataset. Since this data was available state-wise, the dataset's basic layer was created by downloading and consolidating datasets for each state, providing thorough coverage across all districts [2].
2. **Natural Calamity Risk:** To access the vulnerability of districts to natural disasters, manual searches were conducted for each district using verified online sources, government portals, and regional disaster management websites. Each district was categorized into three risk levels—Low, Medium, or High—based on the historical frequency and severity of natural calamities reported.

3. **Connectivity to Nearest Airport:** Airport accessibility was considered a key indicator of regional connectivity. Using Google Maps, we identified the nearest airport within a 100-kilometre radius from the geographical center of each district. This information was compiled manually to ensure location-specific accuracy.
4. **Average Commute Time and Trip Length:** Data related to intra-city mobility—specifically, average commute time and trip length—was sourced from a study conducted by IIT Delhi, which provided statistically validated insights for urban and semi-urban regions. These values were directly mapped to the corresponding districts where data was available [1].
5. **Air Quality Index (AQI):** District-level AQI data was manually collected by referencing multiple environmental monitoring portals and publicly available reports. In cases where district-level AQI was not available, data was inferred from the nearest available location with similar urban characteristics [14].
6. **Metro Infrastructure Presence:** The target variable, indicating whether a district currently has or is planned to receive a metro line, was derived from a combination of government infrastructure project announcements and verified existing metro operations. This included both operational networks and officially sanctioned future projects.

Through meticulous aggregation and manual validation, we curated a clean and structured dataset that reflects the socio-economic, environmental, and infrastructural landscape of Indian districts, forming the basis for our machine learning analysis.

5.2 LIBRARIES USED

1. **Pandas**: Provides powerful tools for data manipulation and analysis. Commonly used for reading, cleaning, and transforming structured datasets such as CSV files.
2. **NumPy**: A fundamental package for scientific computing in Python, supporting multidimensional arrays, linear algebra, and a wide range of mathematical functions.
3. **Matplotlib**: A comprehensive library for creating static, animated, and interactive visualizations in Python. Often used for plotting graphs and charts to explore data.
4. **Seaborn**: Built on top of Matplotlib, this library offers a high-level interface for drawing attractive and informative statistical graphics with ease.
5. **StandardScaler**: A pre-processing technique that standardizes features by removing the mean and scaling to unit variance, improving the performance of models sensitive to feature scaling.
6. **LabelEncoder**: Encodes categorical text labels as integers. Useful for transforming non-numeric labels into a format suitable for machine learning algorithms.

7. **RFECV**: Recursive Feature Elimination with Cross-Validation. Helps select the most important features by recursively eliminating the least important ones based on model performance.
8. **CalibratedClassifierCV**: Calibrates the probability estimates of a classifier, improving the reliability of predicted probabilities for classification models.
9. **Permutation_importance**: Measures the importance of features by calculating the change in a model's score when a single feature value is randomly shuffled.
10. **Train_test_split**: Splits the dataset into training and testing sets to evaluate model performance on unseen data.
11. **RandomizedSearchCV**: A method for hyperparameter tuning that samples a wide range of parameter values randomly for efficient search over large parameter spaces.
12. **StratifiedKFold**: Ensures that each fold of cross-validation maintains the same proportion of target classes, which is especially useful for imbalanced classification tasks.
13. **LogisticRegression**: A linear classification algorithm that models the probability of a binary outcome based on input features.
14. **DecisionTreeClassifier**: A tree-based model that splits data into branches based on feature thresholds, suitable for both classification and regression tasks.
15. **RandomForestClassifier**: An ensemble learning method that builds multiple decision trees and combines their results for improved prediction accuracy and reduced overfitting.
16. **XGBClassifier**: A high-performance implementation of gradient boosting for classification problems. Known for speed and accuracy, especially on structured/tabular data.
17. **SVC (Support Vector Classifier)**: A kernel-based classification algorithm that finds the optimal hyperplane to separate classes in a high-dimensional space.
18. **f1_score, precision_score, recall_score**: Evaluation metrics used to assess classification performance. F1 score balances precision and recall, while individual precision and recall highlight different aspects of model effectiveness.
19. **Classification_report**: Generates a comprehensive summary of classification metrics including precision, recall, F1-score, and support for each class.
20. **Confusion_matrix**: Displays a matrix showing correct and incorrect predictions, providing insight into a model's classification accuracy.
21. **precision_recall_curve**: Plots precision against recall at different thresholds, useful for evaluating performance on imbalanced datasets.
22. **SMOTETomek (imblearn.combine)**: Combines SMOTE (oversampling of the minority class) with Tomek links (undersampling by removing borderline majority instances) for better handling of imbalanced datasets.
23. **RandomUnderSampler (imblearn.under_sampling)**: Balances class distribution by randomly removing samples from the majority class.
24. **Flask**: A lightweight web framework used to build APIs for model deployment and serve predictions through HTTP requests.

25. **Flask-Ngrok**: Allows Flask applications to be securely exposed to the internet using Ngrok, which is useful for testing APIs remotely.
26. **Pickle**: A Python module used for serializing and deserializing Python objects, such as saving and loading trained machine learning models.

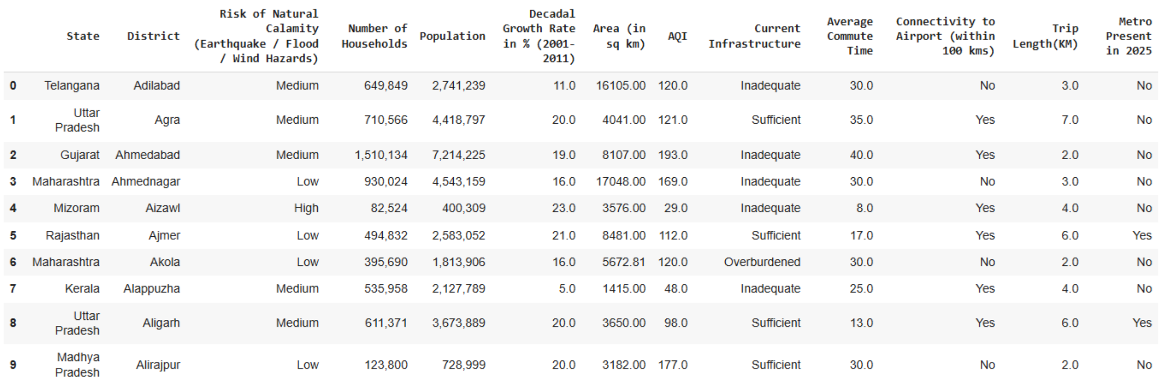
5.3 DATA PRE-PROCESSING

During the data pre-processing step, we started by collecting raw data from various sources. The first critical step was to clean up the data, beginning with the commas in the columns. We deleted any commas that interfered with appropriate numerical representation. This transformation enabled us to correctly convert the values into integers, assuring their suitability for analysis.

Next, a thorough search for missing or null values in the dataset revealed that certain rows were completely missing data in all columns. Rather of attempting to fill such large gaps, these rows were removed entirely to confirm the dataset's quality and reliability before proceeding to the next pre-processing processes.

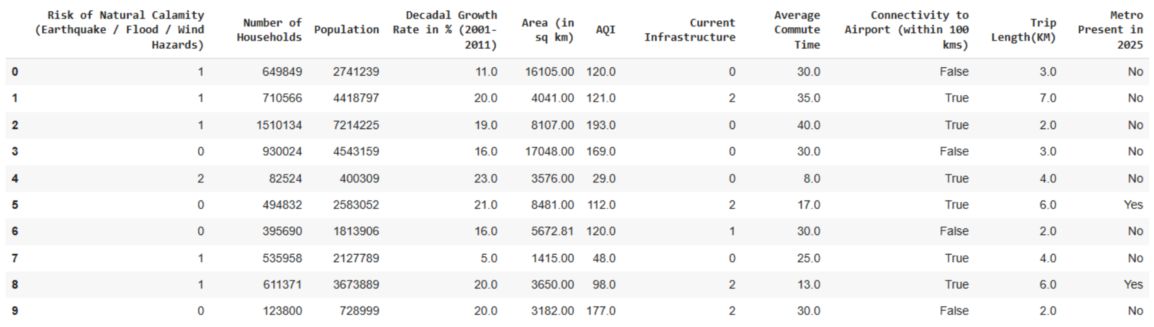
Next, we focused on encoding the dataset's categorical features. The *State* and *District* columns were removed first because they did not provide useful information for the analysis and model training. We then encoded the categorical columns. *The Risk of Natural Calamity (Earthquake / Flood / Wind Hazards)* characteristic, which had values of "Low", "Medium", and "High", was mapped to numerical values 0, 1, and 2, respectively. The *Current Infrastructure* column was similarly changed, with "Inadequate" denoted as 0, "Overburdened" as 1, and "Sufficient" as 2. For the *Connectivity to Airport (within 100 kms)* feature, "No" was mapped to False and "Yes" to True, reflecting its Boolean nature.

Finally, label encoding was used for the *Metro Present in 2025* column. The values "No" and "Yes" were encoded as binary labels, with "No" as 0 and "Yes" as 1. This guaranteed that the target variable was suitable for use in classification techniques.



	State	District	Risk of Natural Calamity (Earthquake / Flood / Wind Hazards)	Number of Households	Population	Decadal Growth Rate in % (2001-2011)	Area (in sq km)	AQI	Current Infrastructure	Average Commute Time	Connectivity to Airport (within 100 kms)	Trip Length(KM)	Metro Present in 2025
0	Telangana	Adilabad	Medium	649,849	2,741,239	11.0	16105.00	120.0	Inadequate	30.0	No	3.0	No
1	Uttar Pradesh	Agra	Medium	710,566	4,418,797	20.0	4041.00	121.0	Sufficient	35.0	Yes	7.0	No
2	Gujarat	Ahmedabad	Medium	1,510,134	7,214,225	19.0	8107.00	193.0	Inadequate	40.0	Yes	2.0	No
3	Maharashtra	Ahmednagar	Low	930,024	4,543,159	16.0	17048.00	169.0	Inadequate	30.0	No	3.0	No
4	Mizoram	Aizawl	High	82,524	400,309	23.0	3576.00	29.0	Inadequate	8.0	Yes	4.0	No
5	Rajasthan	Ajmer	Low	494,832	2,583,052	21.0	8481.00	112.0	Sufficient	17.0	Yes	6.0	Yes
6	Maharashtra	Akola	Low	395,690	1,813,906	16.0	5672.81	120.0	Overburdened	30.0	No	2.0	No
7	Kerala	Alappuzha	Medium	535,958	2,127,789	5.0	1415.00	48.0	Inadequate	25.0	Yes	4.0	No
8	Uttar Pradesh	Aligarh	Medium	611,371	3,673,889	20.0	3650.00	98.0	Sufficient	13.0	Yes	6.0	Yes
9	Madhya Pradesh	Alirajpur	Low	123,800	728,999	20.0	3182.00	177.0	Sufficient	30.0	No	2.0	No

Figure 5.2. 1 Initial 10 Samples



	Risk of Natural Calamity (Earthquake / Flood / Wind Hazards)	Number of Households	Population	Decadal Growth Rate in % (2001-2011)	Area (in sq km)	AQI	Current Infrastructure	Average Commute Time	Connectivity to Airport (within 100 kms)	Trip Length(KM)	Metro Present in 2025
0	1	649849	2741239	11.0	16105.00	120.0	0	30.0	False	3.0	No
1	1	710566	4418797	20.0	4041.00	121.0	2	35.0	True	7.0	No
2	1	1510134	7214225	19.0	8107.00	193.0	0	40.0	True	2.0	No
3	0	930024	4543159	16.0	17048.00	169.0	0	30.0	False	3.0	No
4	2	82524	400309	23.0	3576.00	29.0	0	8.0	True	4.0	No
5	0	494832	2583052	21.0	8481.00	112.0	2	17.0	True	6.0	Yes
6	0	395690	1813906	16.0	5672.81	120.0	1	30.0	False	2.0	No
7	1	535958	2127789	5.0	1415.00	48.0	0	25.0	True	4.0	No
8	1	611371	3673889	20.0	3650.00	98.0	2	13.0	True	6.0	Yes
9	0	123800	728999	20.0	3182.00	177.0	2	30.0	False	2.0	No

Figure 5.2. 2 Encoded Data

5.4 DATA VISUALIZATION

When two features in a dataset are highly correlated, it can lead to a variety of problems while training machine learning models. Our analysis showed a strong correlation between *Population* and *Number of Household*. This strong link means that the two features convey identical information, which can lead to the following issues:

1. **Multicollinearity**: Highly correlated characteristics can lead to multicollinearity in models, particularly linear regression models. When multiple correlated features provide duplicate information, the model may struggle to determine their individual effect on the target variable. As a result, the model's coefficients may become unstable, leading to inaccurate predictions and poor generalization.
2. **Overfitting**: When features are highly correlated, the model may give too much weight to one characteristic over another, thus resulting in overfitting. The model may excel in training data but struggle to apply to new data.
3. **Reduced Model Interpretability**: High correlation between features can make it harder to determine which feature contributes more to prediction.

To address this issue, we introduced a new feature, *Population Density*, which is derived by dividing *Population* by *Area (in sq km)*. This new feature captures the same essential information (the average number of people per sq km) but in a more consolidated and meaningful manner. By doing so, we reduced the redundancy caused by the high correlation.

As a final step, we dropped the *Number of Households* feature from the dataset. This decision ensured that we no longer had two features that were essentially conveying the same information, which helped reduce the risk of multicollinearity, improved model stability, and enhanced the overall interpretability of the model.

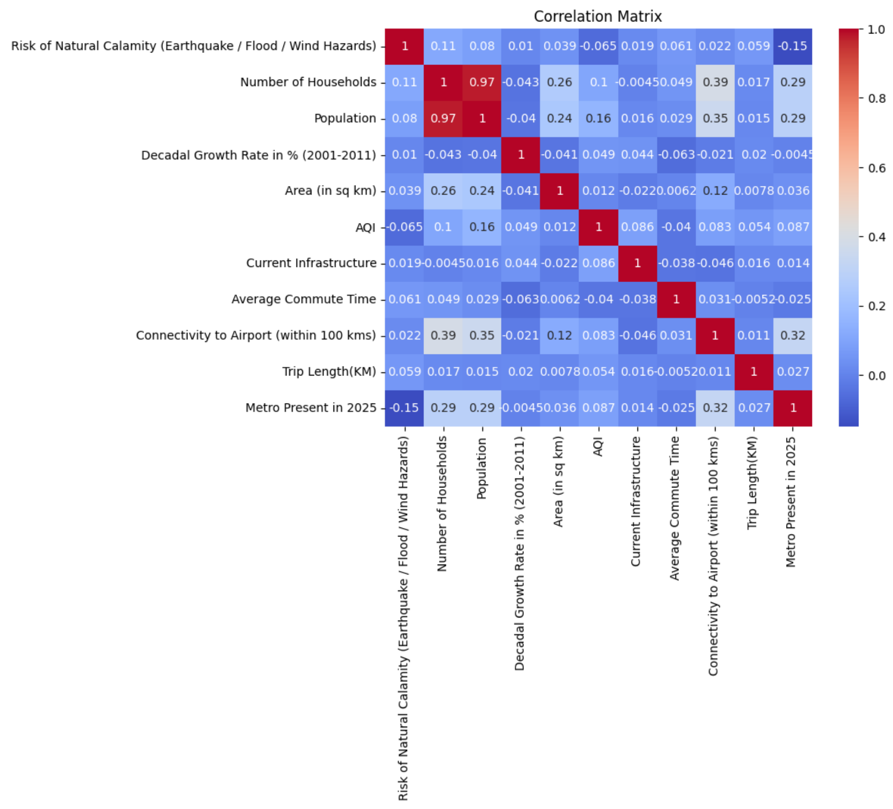


Figure 5.3. 1 Correlation with Redundant Data

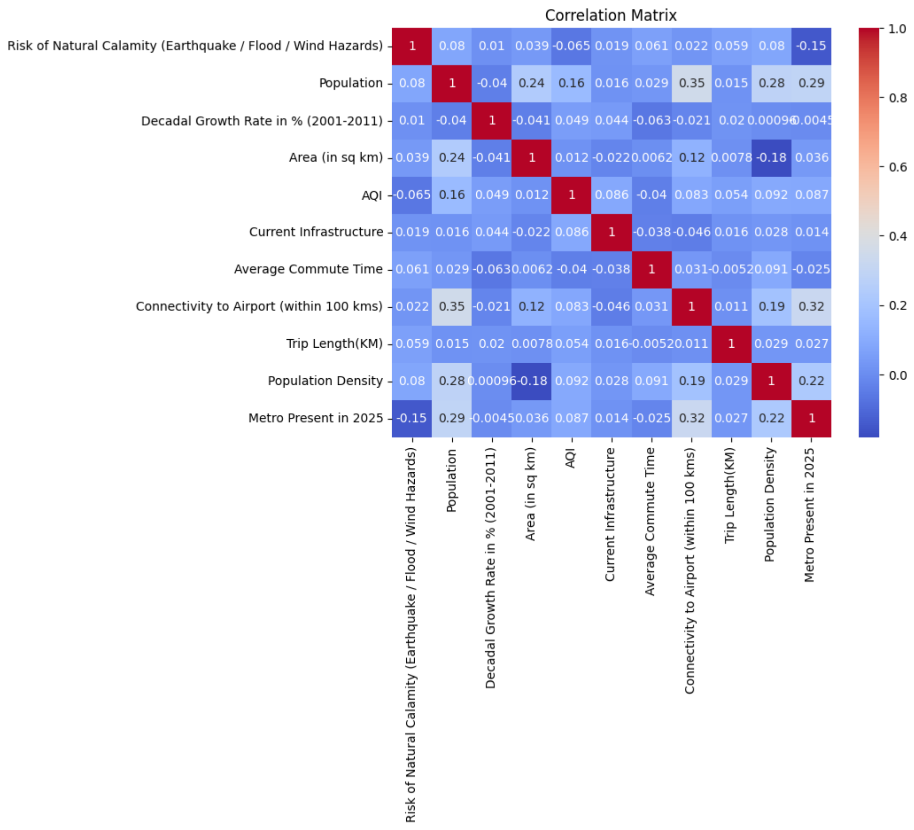
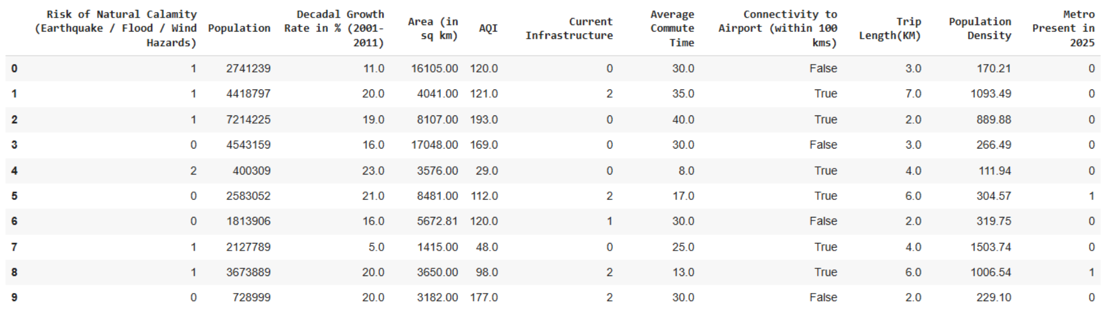


Figure 5.3. 2 Correlation after Removing Redundant Data



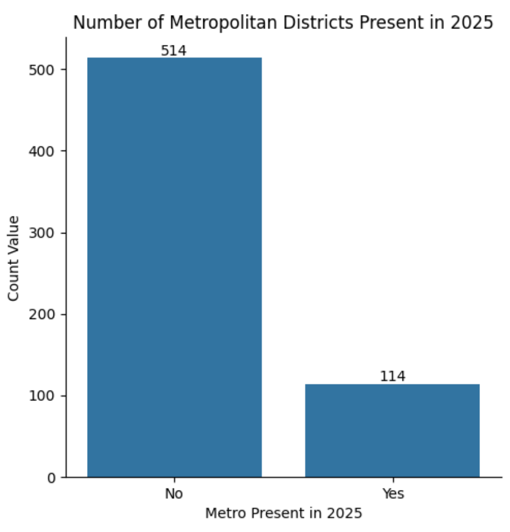
	Risk of Natural Calamity (Earthquake / Flood / Wind Hazards)	Population	Decadal Growth Rate in % (2001-2011)	Area (in sq km)	AQI	Current Infrastructure	Average Commute Time	Connectivity to Airport (within 100 kms)	Trip Length(KM)	Population Density	Metro Present in 2025
0	1	2741239	11.0	16105.00	120.0	0	30.0	False	3.0	170.21	0
1	1	4418797	20.0	4041.00	121.0	2	35.0	True	7.0	1093.49	0
2	1	7214225	19.0	8107.00	193.0	0	40.0	True	2.0	889.88	0
3	0	4543159	16.0	17048.00	169.0	0	30.0	False	3.0	266.49	0
4	2	400309	23.0	3576.00	29.0	0	8.0	True	4.0	111.94	0
5	0	2583052	21.0	8481.00	112.0	2	17.0	True	6.0	304.57	1
6	0	1813906	16.0	5672.81	120.0	1	30.0	False	2.0	319.75	0
7	1	2127789	5.0	1415.00	48.0	0	25.0	True	4.0	1503.74	0
8	1	3673889	20.0	3650.00	98.0	2	13.0	True	6.0	1006.54	1
9	0	728999	20.0	3182.00	177.0	2	30.0	False	2.0	229.10	0

Figure 5.3. 3 Data after Removing Redundant Data

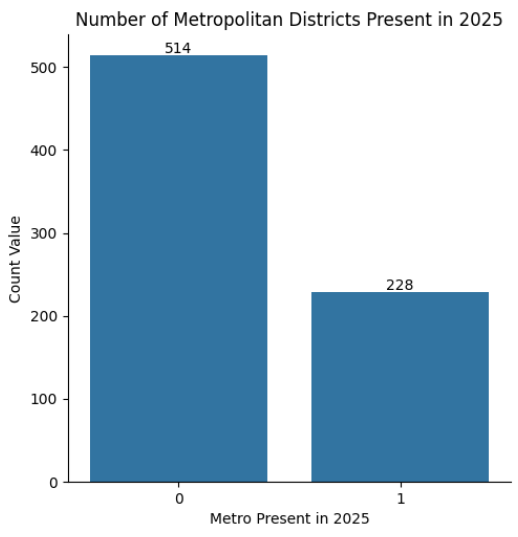
5.5 DATA AUGMENTATION

Data augmentation is a critical strategy in machine learning that increases the quantity and diversity of a dataset, hence improving model performance and generalization ability. It is especially useful when the dataset is limited or skewed, allowing the model to learn from a wider range of events. However, while data augmentation can considerably improve a model's ability to generalize to unknown data, it is critical to exercise caution while using this strategy. Excessive data augmentation can result in overfitting, in which the model learns on enhanced data that may not reflect real-world variances. This can hamper the model's capacity to generalize successfully, resulting in lower performance on real-world data. As a result, augmentation should be used with caution to ensure that the alterations introduced match realistic scenarios while still maintaining the dataset's integrity.

In practice, data augmentation is making minor changes to the dataset, such as adding noise, marginally altering values, or creating synthetic examples. However, if these changes are too drastic or unrealistic, they can disrupt the underlying patterns that the model is designed to learn. For example, in numerical datasets, injecting excessive noise or extreme changes can cause the model to learn from irrelevant patterns, reducing its robustness when deployed. As a result, significant consideration must be given to the magnitude and type of changes employed, ensuring that they are consistent with what would be expected in real-world data.



5.4. 1 Dataset before Augmenting



5.4. 2 Dataset after Augmenting

5.6 MODEL BUILDING

During the model building phase, the dataset was initially separated into independent variables (X) and dependent variables (Y). The independent variables are the input properties used to train the models, whereas the dependent variable reflects the desired outcome—specifically, whether a district is predicted to have metro infrastructure by 2025.

To enhance model performance and interpretability, we utilized feature selection before training. This was accomplished by Recursive Feature Elimination with Cross-Validation (RFECV), a technique that recursively removes less significant features based on model performance until an optimal subset is found. Initially, RFECV was permitted to choose the most appropriate number of features on its own. However, we discovered through experimentation that picking a fixed number of features (k = 5) resulted in more consistent and high-performing models across all classifiers. Each classification model had been evaluated using RFECV to discover the five top most influential features. These selected characteristics were saved in *model_selected_features* and utilized to train the various models. The top features of the best-performing model for each method (e.g., Random Forest, SVM, and XGBoost) were saved separately in *model_best_model_features*. This method ensured that each model was trained on the most relevant subset of data, increasing efficiency and predicting accuracy while lowering the danger of overfitting.

To alleviate class imbalance in the dataset, notably the underrepresentation of districts where metro infrastructure is expected to be required, we used the SMOTETomek approach. SMOTETomek combines SMOTE (Synthetic Minority Over-sampling Technique) with Tomek linkages to generate synthetic samples for the minority class while also removing overlapping samples that might hamper decision boundaries. This dual strategy produces a cleaner, more balanced training dataset. SMOTE works by interpolating between existing minority class samples, generating new, synthetic points that assist balance the distribution of majority and minority classes without simply replicating existing ones. The sampling_strategy='auto' parameter was used to automatically resample until class distributions were equal, and a fixed random_state=42 ensured consistency across runs.

To improve the training data, we added Gaussian noise to the feature set. This method includes applying modest, random variations from a normal distribution (mean 0, small standard deviation) to each sample in the dataset. This allows the model to learn generalized patterns rather than overfitting to specific feature values. The noise level was kept to a minimum (noise_level=0.01) to retain the integrity of the original data while introducing realistic variance. This step successfully broadens the diversity of training samples, making the model more resilient to noise and oscillations in real-world data.

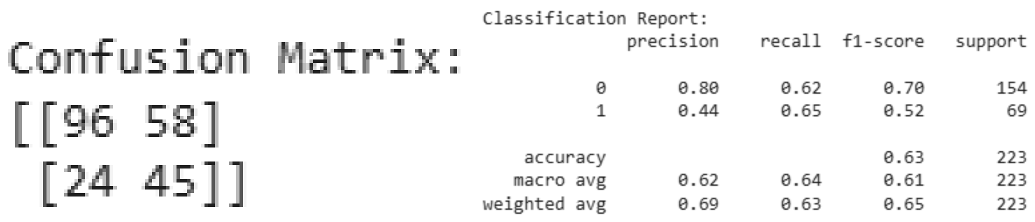
Prior to these preprocessing processes, the dataset was divided into training and testing subsets using stratified sampling to preserve class proportions. Furthermore, data types were expressly standardized where needed, such as converting categorical numerical attributes like 'Connectivity to Airport (within 100 kms)' to integer format. Collectively, these augmentation and balancing strategies ensure that the model is prepared to deal with both imbalanced data and real-world unpredictability.

5.6.1 Logistic Regression

Logistic Regression is a statistical method used for binary classification tasks, where the goal is to predict one of two possible outcomes. It models the relationship between the independent variables (features) and the dependent variable (target) using a logistic function (sigmoid), which maps predicted values to probabilities between 0 and 1. Logistic Regression predicts the probability of a specific class and uses a threshold (typically 0.5) to classify the outcome.

To identify the best hyperparameters for the logistic regression model, we used *GridSearchCV* from sklearn.model_selection. This method conducts an exhaustive search across a manually defined grid of hyperparameters, scoring each combination with 5-fold cross-validation. The grid investigated various regularization strategies, such as L1 (Lasso), L2 (Ridge), and Elastic Net penalties, as well as compatible solvers (liblinear, lbfgs, sag, saga) and regularization strengths (C values). To balance L1 and L2 penalties in Elastic Net, extra tuning was performed on the l1_ratio parameter. The F1 score was employed as a scoring metric to ensure that the selected model provides a fair trade-off between precision and recall, which is especially useful in imbalanced classification problems. After running a grid search on the training data, the best-performing model configuration was saved for future evaluation.

Following model selection, we modified the classification threshold around the default value of 0.5 to improve F1 performance on the test set. Predicted probabilities were computed with *predict_proba*, and a range of thresholds (0.30 to 0.70) were evaluated to see which value resulted in the highest F1. We calculated precision, recall, and F1 score for each threshold before deciding on the one that maximized F1 while reporting the matching precision and recall for affirmative ("Yes") predictions. This method guarantees that the final model is not only fine-tuned but also matched with our evaluation criteria, particularly in correctly identifying places that are likely to require metro infrastructure.



```
Confusion Matrix:
[[96 58]
 [24 45]]
```

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

           0       0.80      0.62      0.70       154
           1       0.44      0.65      0.52        69

    accuracy                           0.63       223
   macro avg       0.62      0.64      0.61       223
weighted avg       0.69      0.63      0.65       223
```

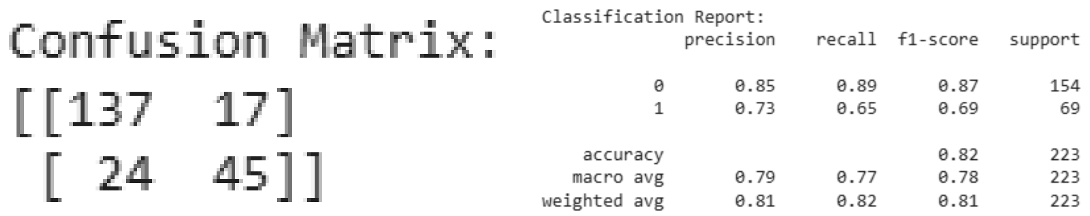
Figure 5.6.1. 1 Confusion Matrix and Classification Report of Logistic Regression

5.6.2 Random Forest Classifier

The Random Forest Classifier is a powerful ensemble learning algorithm that is widely used for classification applications. During the training phase, it constructs a large number of decision trees and outputs the class that is the average of the classes predicted by the individual trees. This technique makes use of the idea of "bagging" (bootstrap aggregating), which enhances model accuracy and resilience by lowering variance and prevents overfitting. Each tree is generated from a distinct sample of data and splits nodes based on a random subset of features, which improves generalization. Random Forest is especially well-suited to processing huge datasets with high dimensionality, and it can handle both numerical and categorical variables.

To improve the performance of the Random Forest model, we used *RandomizedSearchCV* from sklearn.model_selection. Unlike grid search, which examines all parameter combinations, Randomized Search selects a set number (n_iter) of random combinations from the parameter distributions. This enables a more efficient and computationally feasible search across a large hyperparameter space. The randomized grid varied the number of estimators, maximum tree depth, splitting criteria, feature selection approach, and class weighting choices to deal with imbalance. The F1 score was employed as the evaluation metric, and cross-validation was carried out with 5-fold stratified splits to ensure representative sampling across classes.

Once the best estimator had been discovered, we wrapped it in *CalibratedClassifierCV* to provide solid probability predictions, which are required for threshold adjustment. We next tested a variety of classification criteria (0.30 to 0.58) on the test set to determine which one maximized the F1 score. Binary predictions for each threshold were generated using the model's probability outputs and scored using F1. The best-performing threshold was kept, ensuring that the final classifier strikes an optimal balance between precision and recall. This strategy improves the model's ability to correctly identify districts that will require metro infrastructure, particularly in an imbalanced data situation.



```
Confusion Matrix:
[[137  17]
 [ 24  45]]
```

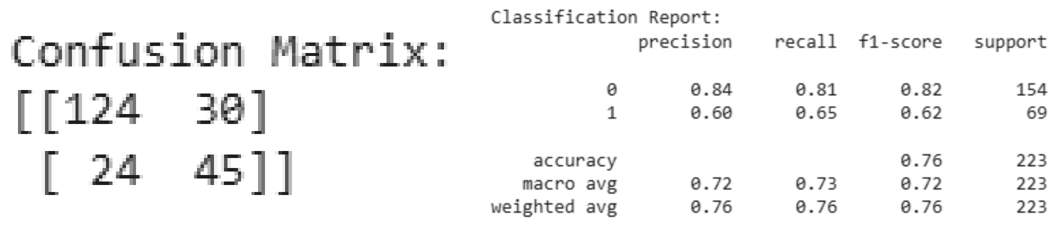
```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.89      0.87       154
           1       0.73      0.65      0.69        69

    accuracy                           0.82       223
   macro avg       0.79      0.77      0.78       223
weighted avg       0.81      0.82      0.81       223
```

Figure 5.6.2. 1 Confusion Matrix and Classification Report of Random Forest Classifier

5.6.3 XGBoost Classifier

Extreme Gradient Boosting (XGBoost) is a scalable and efficient implementation of the gradient boosting framework. XGBoost, which is well-known for its outstanding performance and speed, is frequently utilized in machine learning competitions and real-world applications that use structured/tabular data. The approach constructs decision trees in a sequential manner, with each new tree attempting to repair the flaws of the previous ones while minimizing loss through gradient descent optimization techniques. XGBoost improves on classic gradient boosting in various ways, including regularization to reduce overfitting, parallel tree construction for quicker calculation, and built-in support for missing values and class imbalance.

Following the identification of the best-performing configuration, the model was calibrated with *CalibratedClassifierCV* to improve the probability outputs via cross-validation. This calibration step guarantees that the predicted probabilities are accurately aligned with genuine likelihoods, which is especially critical for post-training threshold tweaking. To improve model performance, we increased the classification threshold above the default value of 0.5. The calibrated model generated probability estimates on the test set, and a range of thresholds (0.30 to 0.70) was evaluated to see which value maximized the F1 score. For each threshold, we computed precision, recall, and F1 score. The threshold with the best F1 score was chosen and utilized to construct the final forecasts. This stage is crucial in imbalanced classification problems like anticipating metro infrastructure requirements, when the goal is to maximize the number of "Yes" occurrences while retaining precision.



```
Confusion Matrix:
[[124  30]
 [ 24  45]]
```

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

           0       0.84      0.81      0.82       154
           1       0.60      0.65      0.62        69

    accuracy                           0.76       223
   macro avg       0.72      0.73      0.72       223
weighted avg       0.76      0.76      0.76       223
```

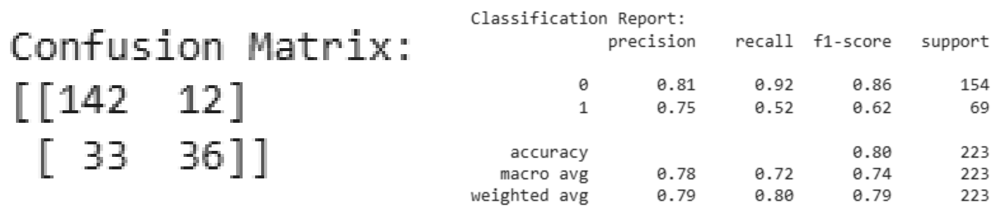
Figure 5.6.3. 1 Confusion Matrix and Classification Report of XGBoost Classifier

5.6.4 Decision Tree Classifier

The Decision Tree Classifier is a simple yet strong supervised learning technique that is widely used for classification applications. It works by recursively partitioning the dataset into subsets based on feature values in order to construct homogeneous groups with respect to the target variable. Each internal node represents a feature-based decision rule, whereas each leaf node represents a class label. This hierarchical form makes decision trees extremely interpretable because the path from root to leaf clearly illustrates how a prediction is made. Despite their simplicity, decision trees can contain complicated decision boundaries and process numerical and categorical data. However, they are prone to overfitting, which can be avoided by carefully tweaking hyperparameters or employing ensemble approaches such as Random Forest.

To improve the performance of the decision tree model, we used *RandomisedSearchCV* from sklearn.model_selection. This approach selects a predetermined number of hyperparameter possibilities, making it computationally efficient in huge search areas. The parameters investigated included several splitting criteria such as Gini and entropy, tree depth, and node splitting limitations such as min_samples_split, min_samples_leaf, and max_features. To ensure class balance, the search was undertaken using 5-fold stratified cross-validation, with the F1 score serving as the evaluation metric to account for the target variable's imbalance.

After determining the best combination of hyperparameters, the chosen model was calibrated using *CalibratedClassifierCV* to generate well-calibrated probability estimates. This stage is especially crucial for post-training threshold tweaking, since correct probabilities enable more informed decision-making about categorization boundaries. To further fine-tune the decision threshold, we tested a variety of probability cut-offs (from 0.30 to 0.70) to find the threshold that produced the highest F1 score on the test set. We calculated precision, recall, and F1 scores for each threshold and chose the one with the highest F1 value. This tuning process ensures that the final model strikes a strong balance between finding "Yes" cases and avoiding false positives, which aligns with our goal of properly predicting metro infrastructure requirement.



```
Confusion Matrix:
[[142  12]
 [ 33  36]]
```

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

           0       0.81      0.92      0.86       154
           1       0.75      0.52      0.62        69

    accuracy                           0.80       223
   macro avg       0.78      0.72      0.74       223
weighted avg       0.79      0.80      0.79       223
```

Figure 5.6.4. 1 Confusion Matrix and Classification Report of Decision Tree Classifier

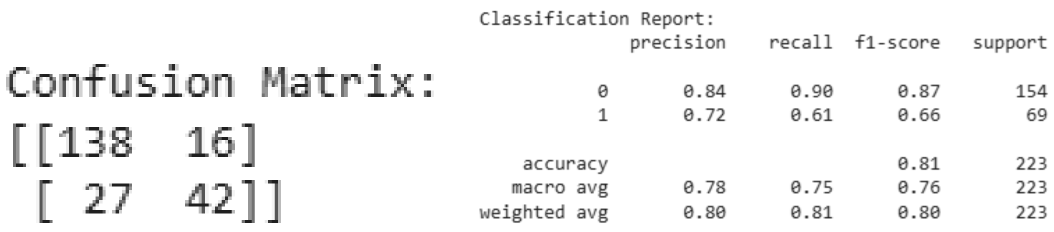
5.6.5 Support Vector Machine

Support Vector Machine (SVM) is a strong supervised learning technique that performs well in both linear and nonlinear classification applications. It works by determining the appropriate hyperplane to split the classes in the feature space, with the goal of maximizing the margin between distinct class boundaries. One of SVM's main advantages is its ability to handle high-dimensional spaces successfully while remaining robust in instances where the number of features exceeds the number of samples.

To address non-linearity in the data, we used the Radial Basis Function (RBF) kernel, which transfers the original characteristics into a higher-dimensional space using a Gaussian function. This change allows the SVM to learn complex, nonlinear decision boundaries. Among the many kernel options tested (e.g., linear, polynomial), the RBF kernel consistently outperformed the others, delivering a better balance of precision and recall on our classification task.

To find the best hyperparameters for the SVM model, we used *RandomizedSearchCV* from sklearn.model_selection, which does an efficient randomized search across a predefined distribution of hyperparameters. This method enables us to explore a large parameter space without incurring the computational overhead of a full grid search. We concentrated on tuning the regularization parameter C, which controls the trade-off between achieving a low training error and a large margin; the gamma parameter, which determines the influence of individual training examples on the decision boundary in the RBF kernel; and the class_weight, which addresses class imbalance by adjusting the penalty for misclassifications in different classes. The model's performance was assessed using 5-fold cross-validation with the F1 score as the evaluation metric, ensuring a balanced emphasis on precision and recall—which is critical for imbalanced classification tasks.

After determining the optimal parameter configuration, the model was calibrated with *CalibratedClassifierCV* to produce accurate probability estimates. These probabilities were then utilized to fine-tune the decision threshold for the test set. By evaluating thresholds ranging from 0.30 to 0.70, we discovered the value that maximized F1 score. At each step, we calculated precision, recall, and F1 score before deciding on the threshold that provided the best trade-off. This final phase guarantees that the model is not only hyperparameter-tuned but also customized to coincide with our aim of accurately identifying potential areas in need of metro infrastructure.



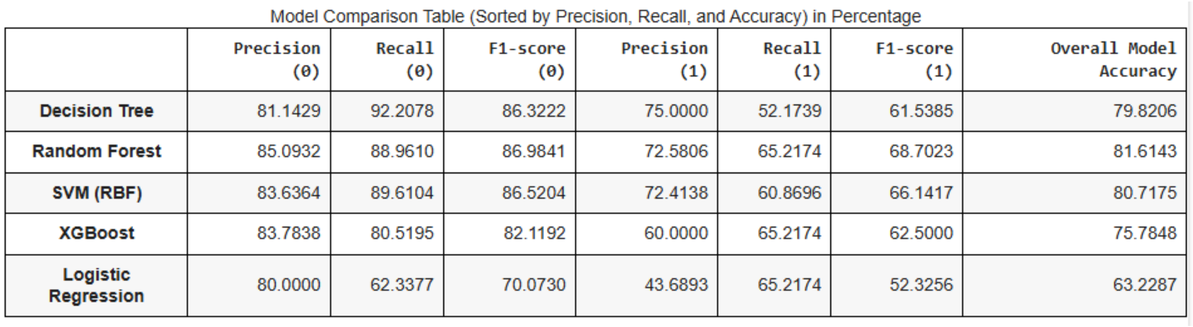
```
Confusion Matrix:
[[138  16]
 [ 27  42]]
```

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

           0       0.84      0.90      0.87       154
           1       0.72      0.61      0.66        69

    accuracy                           0.81       223
   macro avg       0.78      0.75      0.76       223
weighted avg       0.80      0.81      0.80       223
```

Figure 5.6.5. 1 Confusion Matrix and Classification Report of SVM-RBF

5.7 CHOOSING THE BEST MODEL

Among the five models tested—Decision Tree, Random Forest, SVM (RBF), XGBoost, and Logistic Regression—the Random Forest classifier had the most balanced and superior performance. It gets the highest overall accuracy of 81.61%, as well as the top F1-score for the minority class (Class 1) of 68.70%, demonstrating its ability to accurately identify affirmative situations while maintaining solid precision and recall. The SVM with an RBF kernel comes in second, with an accuracy of 80.72% and a Class 1 F1-score of 66.14%. However, Random Forest surpasses it in all essential criteria, particularly in handling the positive class with greater reliability. While the Decision Tree model performs well for the majority class (F1-score of 86.32%), it suffers with minority class recognition, scoring only 61.54% for Class 1. Meanwhile, XGBoost and Logistic Regression fall behind. XGBoost has a good accuracy of 75.78% but has a lower Class 1 F1-score of 62.50%, whereas Logistic Regression has the lowest overall accuracy (63.23%) and Class 1 F1-score (52.33%). Given the importance of both overall accuracy and minority class performance in this classification task, Random Forest emerges as the most dependable and successful model, achieving the optimum mix of precision and recall across both classes.



Model Comparison Table (Sorted by Precision, Recall, and Accuracy) in Percentage

	Precision (0)	Recall (0)	F1-score (0)	Precision (1)	Recall (1)	F1-score (1)	Overall Model Accuracy
Decision Tree	81.1429	92.2078	86.3222	75.0000	52.1739	61.5385	79.8206
Random Forest	85.0932	88.9610	86.9841	72.5806	65.2174	68.7023	81.6143
SVM (RBF)	83.6364	89.6104	86.5204	72.4138	60.8696	66.1417	80.7175
XGBoost	83.7838	80.5195	82.1192	60.0000	65.2174	62.5000	75.7848
Logistic Regression	80.0000	62.3377	70.0730	43.6893	65.2174	52.3256	63.2287

Figure 5.7. 1 Comparison of all Key Parameters

Chapter 6

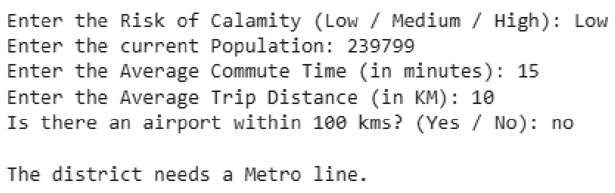
6. PROJECT DEMONSTRATION

The *get_input()* function takes user inputs on key features utilized in metro infrastructure prediction. These variables include the danger of natural disasters (Low, Medium, or High), the current population, the average commute time (in minutes), the average journey distance (in km), and whether an airport is within 100 km. The risk input is translated to a numerical scale (0 for low, 1 for medium, and 2 for high), while the airport input is converted to a boolean value. The function returns these processed inputs in a structured list format, ready for further prediction.

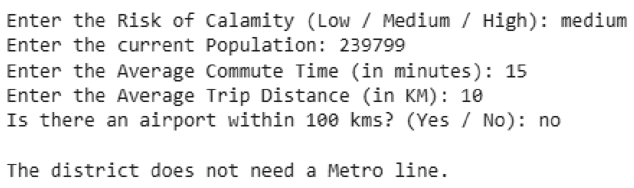
The *predict()* function uses the processed user input to forecast the requirement for a metro line in the specified district. The input list is first turned into a DataFrame with the proper column names. Numerical features, such as disaster risk, population, commuting time, and trip length, are scaled using a pre-fitted scaler (rfc_scaler), while the boolean airport feature remains unchanged. These components are then integrated into a final feature set, which is fed into a pre-trained, calibrated Random Forest model (calibrated.rfc). The model generates a probability score, which is then compared to a predetermined threshold (best_thresh) to establish the final classification. Based on this prediction, the function determines whether the district requires a metro line or not.

For the UI part, a form captures user input like state, district, population, commute time, and trip length. Dropdown menus, input fields, and buttons are strategically placed to guide users through the prediction process. The UI dynamically adjusts district options based on the selected state using JavaScript. When a user selects a state from the dropdown, JavaScript dynamically populates the corresponding district options, this is implemented using an event listener on the state dropdown, which updates the district dropdown options based on pre-defined mappings stored in a JavaScript object.

The form sends user data to the backend API hosted on Google Collab using
Flask-NGROK. Once a prediction is received, the result is displayed in a user-friendly manner. For example, an alert box will show a message saying, “Prediction: Metro needed” or “Prediction: Metro not needed.” Future improvements might include replacing alert boxes with visually enhanced modal dialogs or result cards.

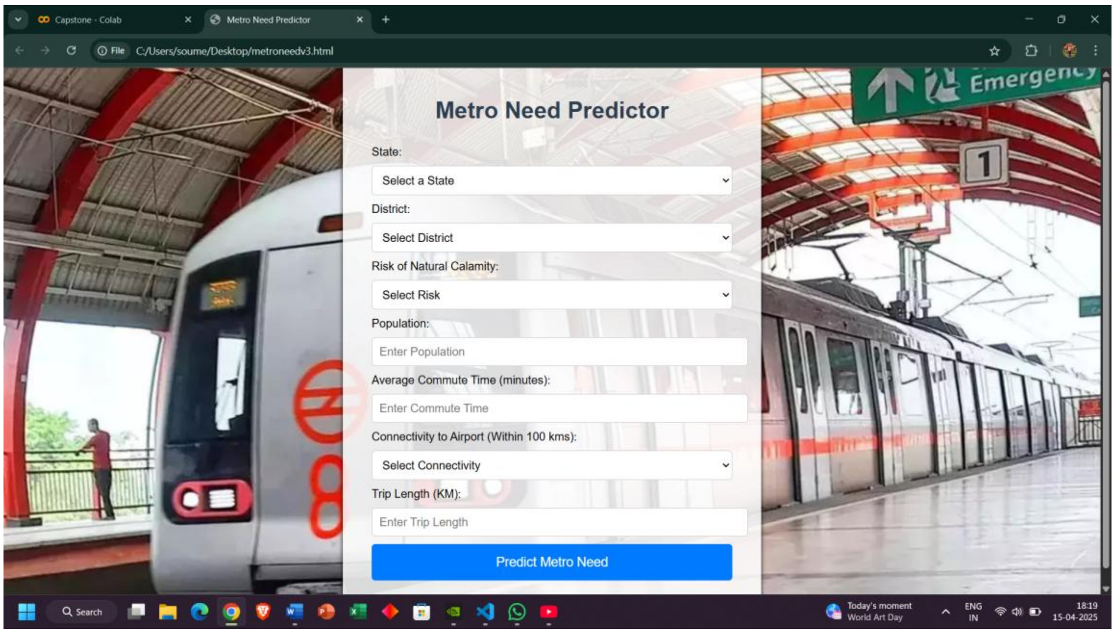


```
Enter the Risk of Calamity (Low / Medium / High): Low
Enter the current Population: 239799
Enter the Average Commute Time (in minutes): 15
Enter the Average Trip Distance (in KM): 10
Is there an airport within 100 kms? (Yes / No): no

The district needs a Metro line.
```



```
Enter the Risk of Calamity (Low / Medium / High): medium
Enter the current Population: 239799
Enter the Average Commute Time (in minutes): 15
Enter the Average Trip Distance (in KM): 10
Is there an airport within 100 kms? (Yes / No): no

The district does not need a Metro line.
```

Figure 6.1 Demo Outputs



Metro Need Predictor

State: Select a State

District: Select District

Risk of Natural Calamity: Select Risk

Population: Enter Population

Average Commute Time (minutes): Enter Commute Time

Connectivity to Airport (Within 100 kms): Select Connectivity

Trip Length (KM): Enter Trip Length

Predict Metro Need

Figure 6. 2 Webpage for front-end interaction with the model

Chapter 7

7. RESULT AND DISCUSSION

The trained models achieved testing accuracies ranging from 63% to 82%, with the Random Forest Classifier (RFC) emerging as the top performer. The table below summarizes the performance of the main models evaluated during development:

Model	Precision for No	Recall for No	F1 Score for No	Precision for Yes	Recall for Yes	F1 Score for Yes	Testing Accuracy
Random Forest Classifier	0.8509	0.8896	0.8694	0.7258	0.6522	0.687	81.61%
SVM (RBF)	0.8364	0.8961	0.8652	0.7241	0.609	0;6614	80.72%
Decision Tree Classifier	0.8143	0.9221	0.8632	0.75	0.5217	0.6154	79.82%
XGBoost	0.8378	0.8052	0.8212	0.6	0.6522	0.625	75.78%
Logistic Regression	0.8	0.6234	0.7073	0.4369	0.6522	0.5233	63.23%

Table 7. 1 Comparison of Models

Compared to an earlier version of this project—where XGBoost was the best model with an accuracy of 75% and a recall of 0.62 for the positive class—the current RFC-based model shows a clear performance improvement. In addition to higher overall accuracy, it delivers more balanced precision-recall outcomes, especially for identifying metro-eligible districts. This progress reflects improvements in feature selection, model tuning, and data pre-processing strategies that were not as mature in the earlier version.

The final model works by taking input features and using a supervised machine learning algorithm to predict outcomes for each district. The main features are:

1. Risk of Natural Calamity – A categorical value representing the risk of natural disasters in a district.
2. Population – A numerical value for the total district population.
3. Average Commute Time – The average time (in minutes) residents spend commuting within the district.
4. Airport Presence – A Boolean value indicating whether an airport is located nearby.
5. Average Trip Distance – The typical distance (in km) traveled by individuals in the district.

The model workflow includes the following steps:

1. Data Pre-processing: Cleaning and standardizing the data through normalization, encoding categorical variables, and handling missing values.
2. Feature Selection: Selecting the most relevant features algorithmically to prevent overfitting and boost model efficiency.
3. Pattern Learning: Training a model such as Random Forest to detect relationships between input features and the target variable.
4. Prediction: Using the trained model to predict the likelihood of metro infrastructure presence in new districts based on their input characteristics.

For instance, a district with high population, lengthy average commute times, and long travel distances may exhibit signs of urban sprawl and limited public transport connectivity—conditions that make metro systems highly beneficial. The model captures such relationships effectively. Similarly, the presence of an airport indicates regional accessibility, though the absence of one doesn’t necessarily imply poor infrastructure—it may simply reflect the district's geographic or economic role.

While the current model delivers promising results, several challenges were encountered:

1. Data Scarcity – With only 629 districts in the dataset, the model’s learning was constrained, especially for underrepresented classes.
2. Outdated Data – The reliance on 2011 Census data means demographic and infrastructural changes over the past decade are not reflected.
3. New District Formation – The model does not account for newer districts formed after 2011, which may have unique urban characteristics.
4. Feature Limitations – Important factors like economic output, road density, or existing public transport were excluded due to data unavailability.
5. Label Ambiguity – The target label (metro presence in 2025) may not perfectly correspond to actual need, as policy and political considerations play a role.
6. Class Imbalance – Fewer districts with metro systems introduce prediction bias unless balanced through resampling techniques.
7. Evolving Planning Criteria – Static historical data may not fully capture present-day planning trends, introducing potential misalignments.

Despite these limitations, the current RFC-based model offers a more refined and accurate predictive framework than previous attempts. Going forward, future work will focus on expanding the dataset, integrating newer and more granular urban indicators, and refining model architecture to better reflect real-time infrastructure needs. This will enable more precise, adaptable, and scalable tools to support data-driven metro planning across diverse Indian districts.

Chapter 8

8. INDIVIDUAL CONTRIBUTION

My Contribution

I, Jaivardhan Tamminana, was responsible for the machine learning models in the Metro Line Prediction using Machine Learning Model project, contributing significantly to its technical foundation. My primary focus was to evaluate, implement, and optimize the machine learning models that powered the prediction system. After experimenting with several algorithms and analyzing their performance on the dataset, I determined that the Random Forest Classifier (RFC) was the most suitable model for this project. Its ability to handle complex datasets and deliver precise predictions made it the ideal choice.

In addition to model development, I was also actively involved in building and curating the dataset used for training and evaluation. This included collecting district-level information from multiple sources, organizing key variables such as population, risk of natural calamities, average commute time, and proximity to airports, and ensuring consistency and quality across entries. The dataset, prepared collaboratively with Soumesh, played a critical role in achieving close to the desired model performance.

Once the RFC model was selected, I worked extensively on fine-tuning its parameters to maximize its accuracy and efficiency. This involved iterative testing and in-depth analysis to ensure the model performed optimally under various conditions. The machine learning models I developed were designed to process user inputs seamlessly and generate accurate predictions, becoming the core engine of the project.

With my implementation in place, I worked closely with Soumesh to integrate the machine learning functionality with the Flask API and front-end interface. Together, these components formed a cohesive system that delivered accurate predictions with an intuitive user experience. My contributions ensured the system was robust, efficient, and technically sound, forming a solid backbone for the project’s success.

Chapter 9

9. CONCLUSION

The development and implementation of this machine learning model mark a significant advancement in applying data-driven approaches to metro infrastructure planning. Among all tested models, the Random Forest Classifier emerged as the best performer, achieving a test accuracy of 81.6%, with a precision of 72.58% and recall of 65.21% for the minority class (districts with metro presence). This is particularly important given the class imbalance in the dataset. Crucially, model selection prioritized the precision-recall trade-off over accuracy, ensuring the model was more effective at identifying underserved districts likely in need of metro systems—despite their underrepresentation in the data.

The final model uses five essential features: population, average commute time, average trip distance, risk of natural calamity, and proximity to an airport. These features were chosen for their direct link to urban mobility challenges and infrastructure demand. For instance, longer commute times and travel distances often point to inadequate public transport and urban sprawl, while areas at higher risk of natural disasters may require resilient transit options like metro networks. Airport presence serves as a proxy for broader connectivity, although its significance varies by region.

Despite promising results, the model is constrained by several factors, including the limited size and age of the dataset (2011, 629 districts) and the narrow scope of features. Important aspects such as economic viability, political context, and land-use characteristics were not included due to data availability. Additionally, newly formed districts post-2011 are not accounted for, and this limits the model’s applicability in the present context.

Future work will aim to expand the dataset, update feature sources with real-time and socio-economic indicators, and enhance the model architecture to improve generalizability and precision. This foundation paves the way for a scalable, AI-powered decision-support framework to inform urban mobility strategies across diverse Indian regions.

Chapter 10

10. REFERENCES

[1] Tiwari, G., Nishant, Transportation Research and Injury Prevention Programme, Indian Institute of Technology Delhi, TIGTHAT project, UK Medical Research Council, Global Challenges, & Transport Research & Injury Prevention Programme. (2018). Travel to work in India: current patterns and future concerns (J. Woodcock, Ed.) [Report]. Transport Research & Injury Prevention Programme, Indian Institute of Technology Delhi, New Delhi. https://tripc.iitd.ac.in/assets/publication/TravelToWorkInIndia.pdf

[2] Government of India. (n.d.). **Census of India**. Retrieved from https://censusindia.gov.in

[3] M. Kumar, S. Mehta, and V. Gupta, "Leveraging smart city initiatives for sustainable urban mobility," *International Journal of Urban Sustainable Development*, vol. 15, no. 3, pp. 216-225, 2021.

[4] "XGBoost Documentation," *XGBoost*, [Online]. Available: https://xgboost.readthedocs.io/. [Accessed: Feb. 2025].

[5] Ministry of Housing and Urban Affairs. (n.d.). **National Urban Transport Policy**. Government of India. Retrieved from https://mohua.gov.in

[6] Yang, Y., Zhang, P., & Ni, S. (2014). Assessment of the Impacts of urban rail transit on metropolitan Regions using System Dynamics Model. Transportation Research Procedia, 4, 521–534. https://doi.org/10.1016/j.trpro.2014.11.040

[7] Kuriakose, P. N., & Bhattacharjee, J. (2021). India's public transport systems: the role of metro rail. In Contemporary South Asian studies (pp. 131–152). https://doi.org/10.1007/978-3-030-76878-2_8

[8] AlKhereibi, A. H., Wakjira, T. G., Kucukvar, M., & Onat, N. C. (2023). Predictive machine learning algorithms for metro ridership based on urban land use policies in support of Transit-Oriented Development. Sustainability, 15(2), 1718. https://doi.org/10.3390/su15021718

[9] Sharma, N., Dhyani, R., & Gangopadhyay, S. (2013). Critical issues related to metro rail projects in India. Journal of Infrastructure Development, 5(1), 67–86. https://doi.org/10.1177/0974930613488296

[10] Wrótny, M., & Bohatkiewicz, J. (2021). Traffic Noise and Inhabitant Health—A comparison of road and rail noise. Sustainability, 13(13), 7340. https://doi.org/10.3390/su13137340

[11] Aboul-Atta, T. A., & Elmaraghy, S. B. E. (2022). Factors affecting performance improvement of the metro system in cities. Journal of Engineering and Applied Science, 69(1). https://doi.org/10.1186/s44147-022-00078-4

[12] Padmavathi, B. & H.O.D Public Administration Govt.Degree College Sitaphalmandi, Hyderabad. (n.d.). URBAN TRANSPORT: A STUDY OF METRO RAIL PROJECTS IN INDIA. In www.ijmer.in (pp. 198–201). https://ijmer.s3-ap-southeast-1.amazonaws.com/pdf/volume8/volume8-issue8(6)-2019/22.pdf

[13] Press Information Bureau. (2020, February 17). *Development of Metro Rail Projects in the Country*. Retrieved April 16, 2025, from https://pib.gov.in/PressReleasePage.aspx?PRID=1602755

[14] CPCB | Central Pollution Control Board. (n.d.). CPCB. https://cpcb.nic.in/

[15] Sumathy Eswaran, M. A. J. Bosco, and Rajalakshmi, "A Study on Traffic Forecast for Metro Railway Expansion in Chennai," Indian Journal of Science and Technology, vol. 9, no. 39, pp. 1-9, Oct. 2016, doi: 10.17485/ijst/2016/v9i39/95429.

[16] R. Goel and G. Tiwari, "Promoting Low Carbon Transport in India: Case Study of Metro Rails in Indian Cities," UNEP Risø Centre on Energy, Climate and Sustainable Development, Technical University of Denmark, June 2014.

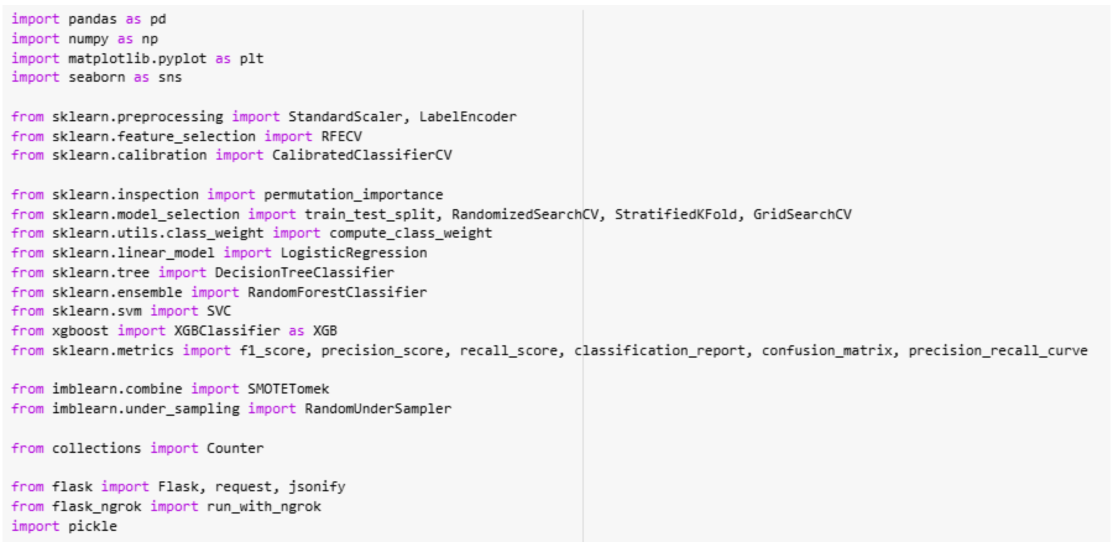
[17] S. Jain and G. Tiwari, "Evaluating the Impact of Metro Rail on Urban Transport Systems: A Case Study of Delhi Metro," Transportation Research Part A: Policy and Practice, vol. 45, no. 1, pp. 80-94, Jan. 2011, doi: 10.1016/j.tra.2010.09.002.

[18] A. Mohan, "Public Transport Policy and the Role of Rail-Based Systems in India," Journal of Transportation Planning and Technology, vol. 31, no. 3, pp. 225-245, 2008, doi: 10.1080/03081060802106114.

[19] T. Flyvbjerg, A. Bruzelius, and W. Rothengatter, "Cost Underestimation and Benefit Overestimation in Rail Projects: A Global Perspective," International Journal of Project Management, vol. 24, no. 6, pp. 481-488, Aug. 2006, doi: 10.1016/j.ijproman.2006.07.006.

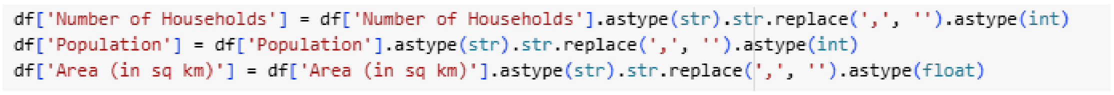
Chapter 11

11. APPENDIX A- SAMPLE CODE



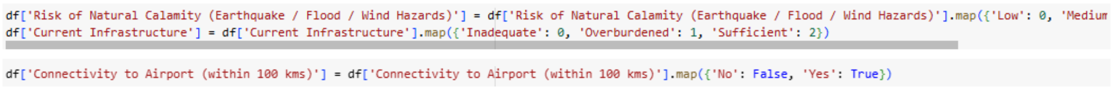
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection import RFECV
from sklearn.calibration import CalibratedClassifierCV

from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split, RandomizedSearchCV, StratifiedKFold, GridSearchCV
from sklearn.utils.class_weight import compute_class_weight
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier as XGB
from sklearn.metrics import f1_score, precision_score, recall_score, classification_report, confusion_matrix, precision_recall_curve

from imblearn.combine import SMOTETomek
from imblearn.under_sampling import RandomUnderSampler

from collections import Counter

from flask import Flask, request, jsonify
from flask_ngrok import run_with_ngrok
import pickle
```

Figure 11. 1 Libraries Used Snippet



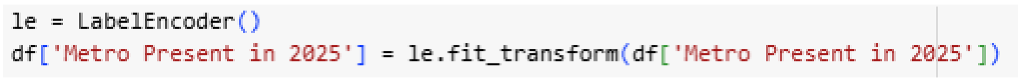
```python
df['Number of Households'] = df['Number of Households'].astype(str).str.replace(',', '').astype(int)
df['Population'] = df['Population'].astype(str).str.replace(',', '').astype(int)
df['Area (in sq km)'] = df['Area (in sq km)'].astype(str).str.replace(',', '').astype(float)
```

Figure 11. 2 Data Pre-processing Code Snippet



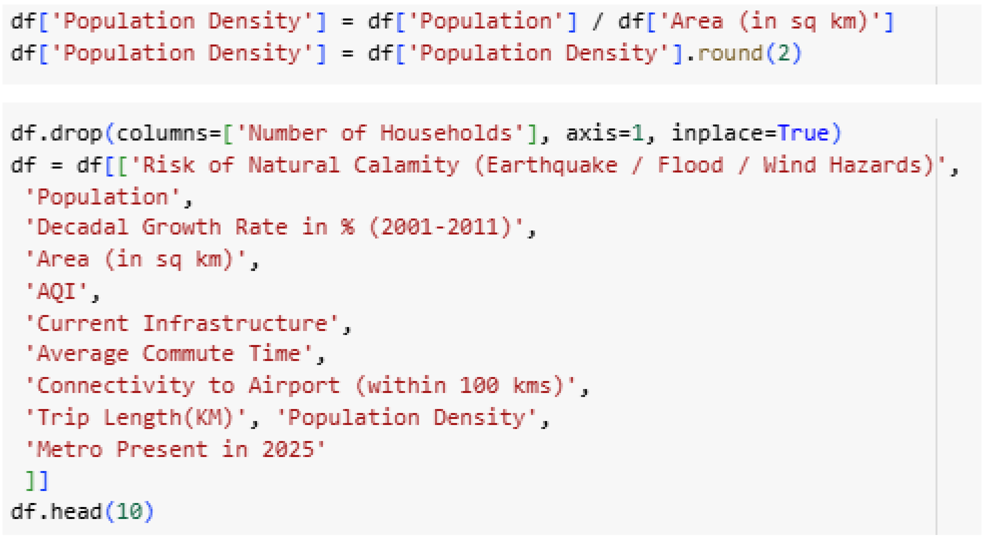
```python
df['Risk of Natural Calamity (Earthquake / Flood / Wind Hazards)'] = df['Risk of Natural Calamity (Earthquake / Flood / Wind Hazards)'].map({'Low': 0, 'Mediu
df['Current Infrastructure'] = df['Current Infrastructure'].map({'Inadequate': 0, 'Overburdened': 1, 'Sufficient': 2})

df['Connectivity to Airport (within 100 kms)'] = df['Connectivity to Airport (within 100 kms)'].map({'No': False, 'Yes': True})
```

Figure 11. 3 Feature Mapping Code Snippet



```python
le = LabelEncoder()
df['Metro Present in 2025'] = le.fit_transform(df['Metro Present in 2025'])
```

Figure 11. 4 Label Encoding Code Snippet



```python
df['Population Density'] = df['Population'] / df['Area (in sq km)']
df['Population Density'] = df['Population Density'].round(2)

df.drop(columns=['Number of Households'], axis=1, inplace=True)
df = df[['Risk of Natural Calamity (Earthquake / Flood / Wind Hazards)',
 'Population',
 'Decadal Growth Rate in % (2001-2011)',
 'Area (in sq km)',
 'AQI',
 'Current Infrastructure',
 'Average Commute Time',
 'Connectivity to Airport (within 100 kms)',
 'Trip Length(KM)', 'Population Density',
 'Metro Present in 2025'
 ]]
df.head(10)
```

Figure 11. 5 Eliminating Redundancy Code Snippet

```
yes_cases = df[df['Metro Present in 2025'] == 1]
augmented_cases = yes_cases.copy()

for col in features:
  augmented_cases[col] += np.random.uniform(-30, 50, size=len(2*augmented_cases))

df = pd.concat([df, augmented_cases], ignore_index=True)
print(f"Dataset shape after augmentation: {df.shape}")
```

Figure 11. 6 Data Augmentation Code Snippet

```
x = df.drop(columns=['Metro Present in 2025'])
x.head(10)
```

```
y = df['Metro Present in 2025']
y.head(10)
```

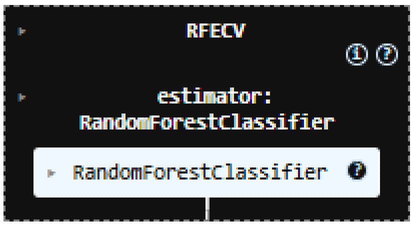
Figure 11. 7 Dependent and Independent Data Split Code Snippet

```
rf_estimator = RandomForestClassifier(random_state=42)
```

```
rf_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

```
rfecv_rf = RFECV(
    estimator=rf_estimator,
    step=1,
    cv=rf_cv,
    scoring='f1',
    n_jobs=-1
)
```

```
rfecv_rf.fit(x, y)
```

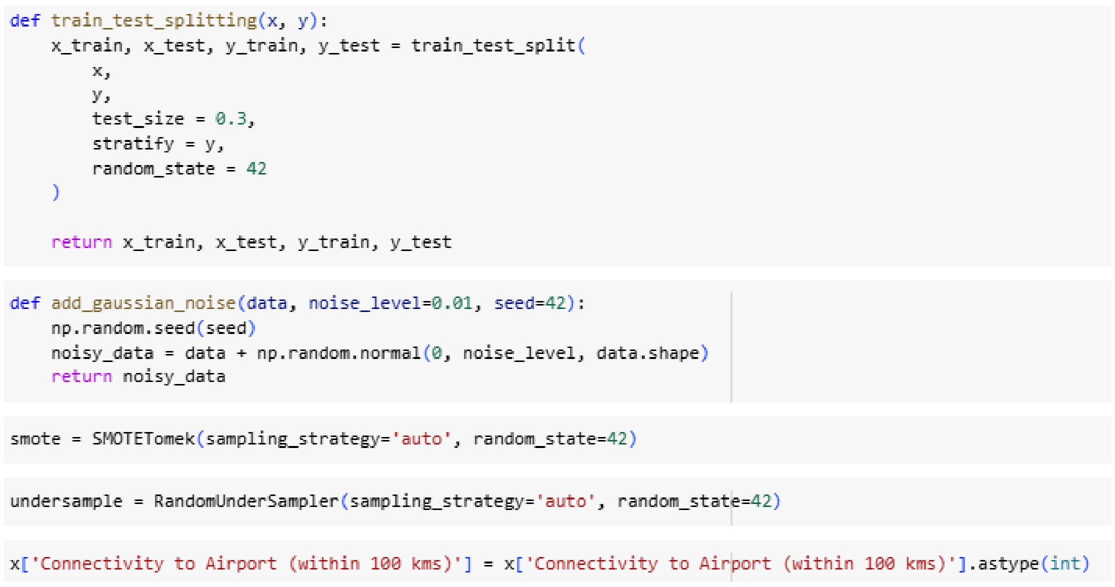


```
RFECV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
feature_ranks_rfc = list(zip(x.columns, rfecv_rf.ranking_))
feature_ranks_rfc.sort(key=lambda x: x[1])
```

```
rfc_selected_features = []
for i in range(5):
  rfc_selected_features.append(feature_ranks_rfc[i][0])
print(f"Random Forest Classifier : \n{rfc_selected_features}")
```

Figure 11. 8 Feature Selection Code Snippet



```
def train_test_splitting(x, y):
    x_train, x_test, y_train, y_test = train_test_split(
        x,
        y,
        test_size = 0.3,
        stratify = y,
        random_state = 42
    )

    return x_train, x_test, y_train, y_test
```

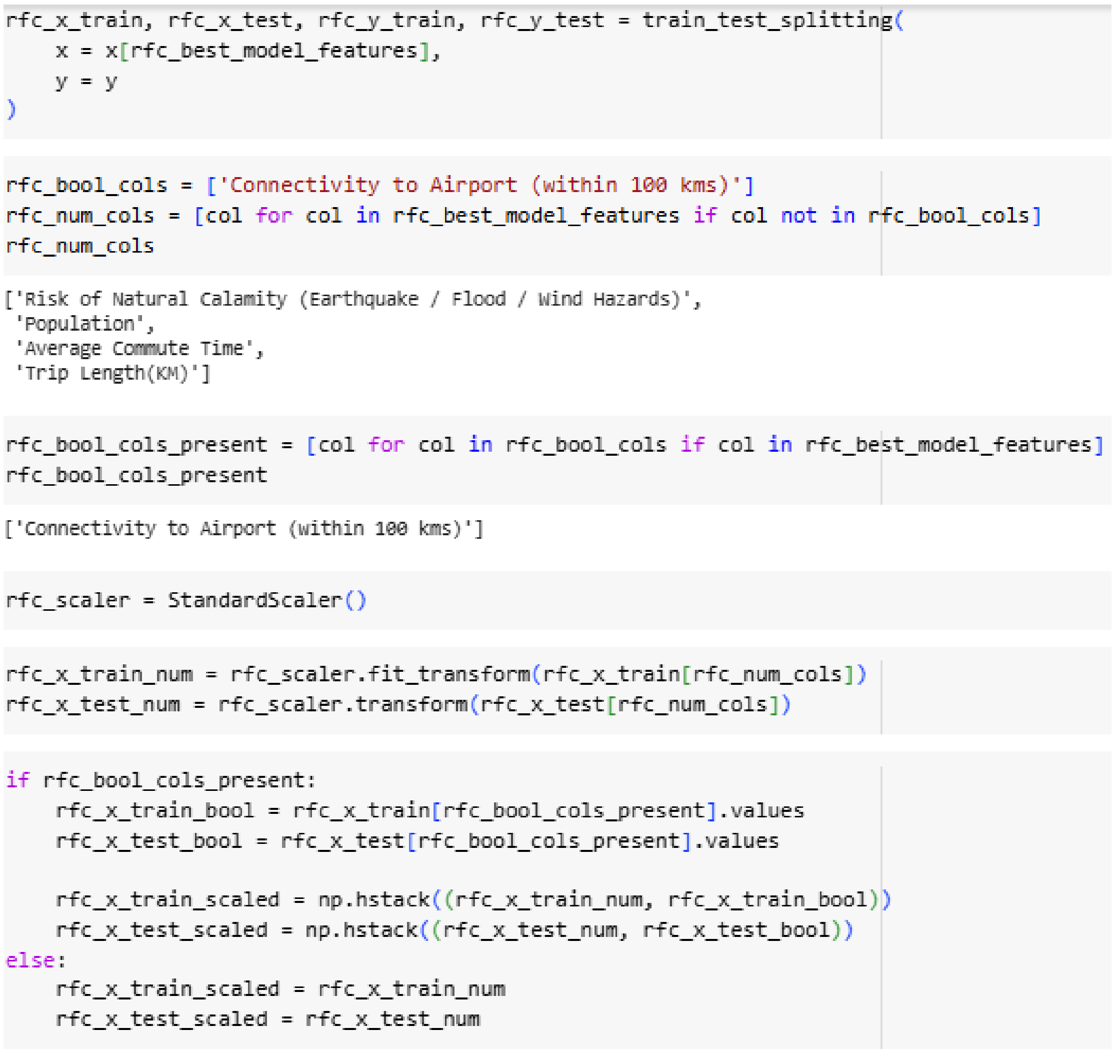
```
def add_gaussian_noise(data, noise_level=0.01, seed=42):
    np.random.seed(seed)
    noisy_data = data + np.random.normal(0, noise_level, data.shape)
    return noisy_data
```

```
smote = SMOTETomek(sampling_strategy='auto', random_state=42)
```

```
undersample = RandomUnderSampler(sampling_strategy='auto', random_state=42)
```

```
x['Connectivity to Airport (within 100 kms)'] = x['Connectivity to Airport (within 100 kms)'].astype(int)
```

Figure 11. 9 Train-Test Splitting Code Snippet



```
rfc_x_train, rfc_x_test, rfc_y_train, rfc_y_test = train_test_splitting(
    x = x[rfc_best_model_features],
    y = y
)
```

```
rfc_bool_cols = ['Connectivity to Airport (within 100 kms)']
rfc_num_cols = [col for col in rfc_best_model_features if col not in rfc_bool_cols]
rfc_num_cols
```

```
['Risk of Natural Calamity (Earthquake / Flood / Wind Hazards)',
 'Population',
 'Average Commute Time',
 'Trip Length(KM)']
```

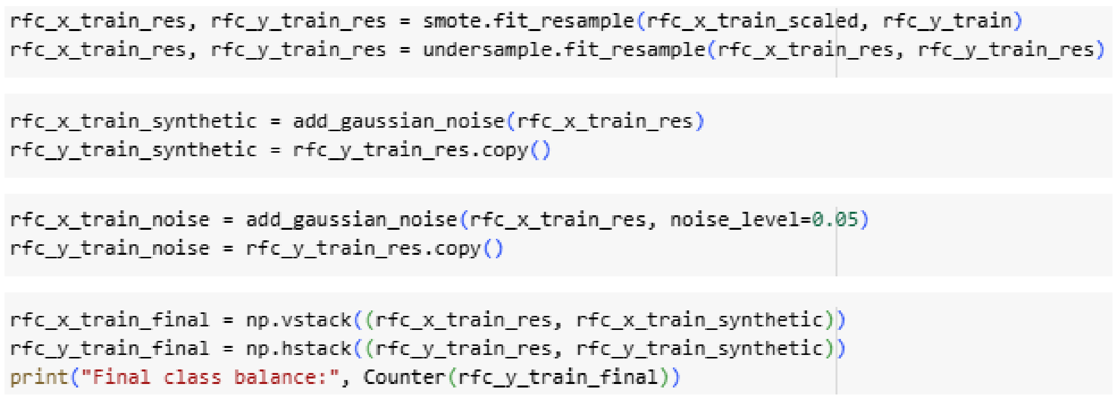
```
rfc_bool_cols_present = [col for col in rfc_bool_cols if col in rfc_best_model_features]
rfc_bool_cols_present
```

```
['Connectivity to Airport (within 100 kms)']
```

```
rfc_scaler = StandardScaler()
```

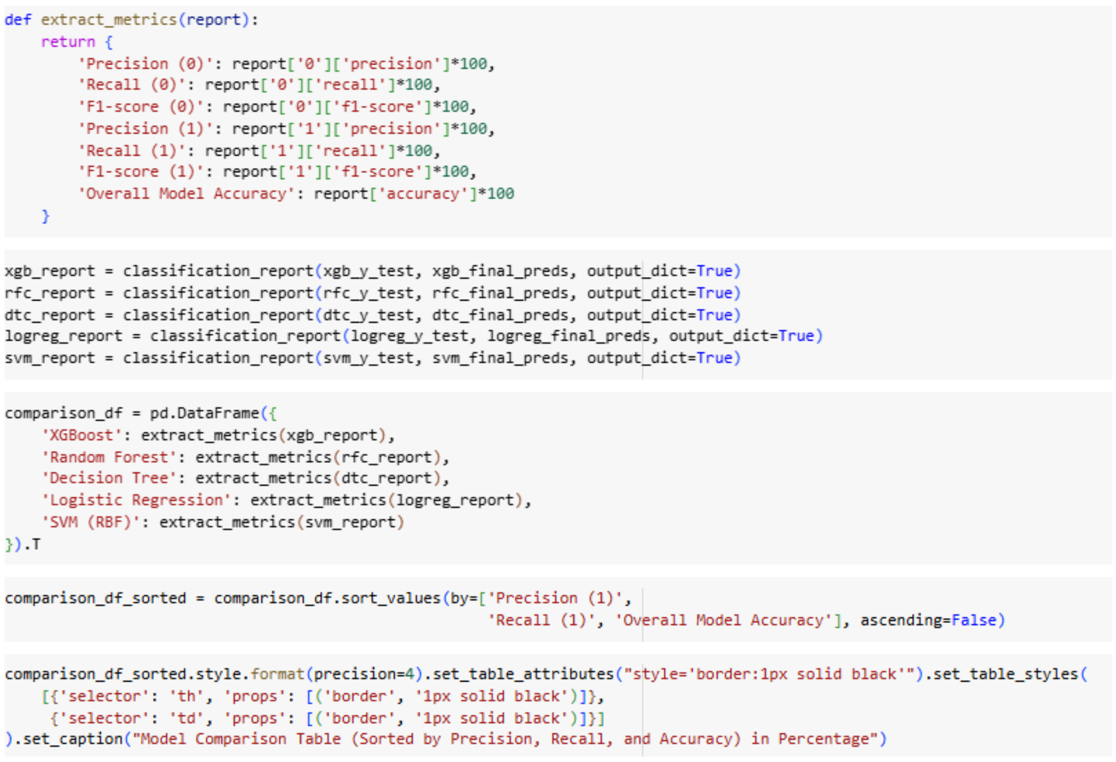
```
rfc_x_train_num = rfc_scaler.fit_transform(rfc_x_train[rfc_num_cols])
rfc_x_test_num = rfc_scaler.transform(rfc_x_test[rfc_num_cols])
```

```
if rfc_bool_cols_present:
    rfc_x_train_bool = rfc_x_train[rfc_bool_cols_present].values
    rfc_x_test_bool = rfc_x_test[rfc_bool_cols_present].values

    rfc_x_train_scaled = np.hstack((rfc_x_train_num, rfc_x_train_bool))
    rfc_x_test_scaled = np.hstack((rfc_x_test_num, rfc_x_test_bool))
else:
    rfc_x_train_scaled = rfc_x_train_num
    rfc_x_test_scaled = rfc_x_test_num
```



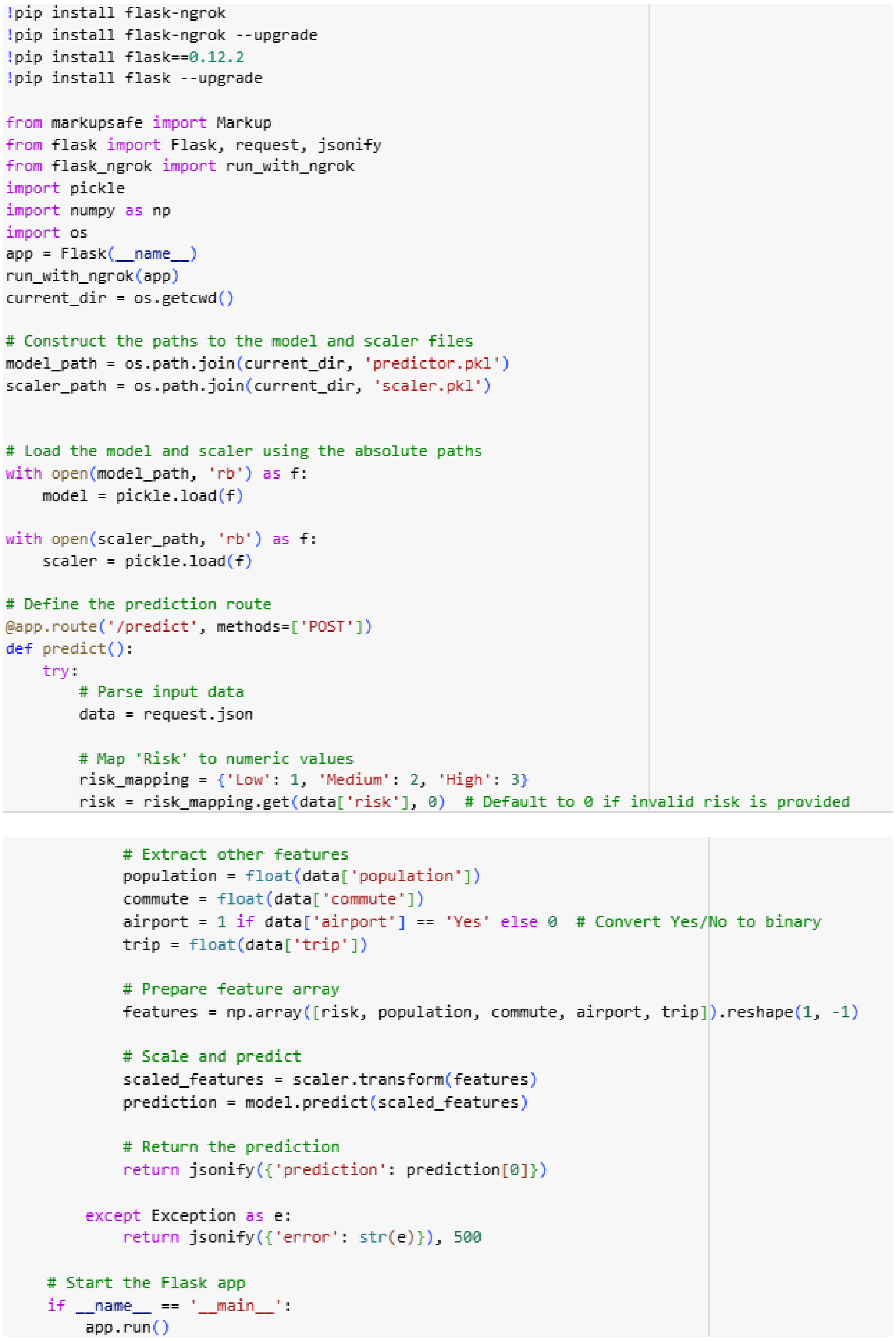
```python
rfc_x_train_res, rfc_y_train_res = smote.fit_resample(rfc_x_train_scaled, rfc_y_train)
rfc_x_train_res, rfc_y_train_res = undersample.fit_resample(rfc_x_train_res, rfc_y_train_res)

rfc_x_train_synthetic = add_gaussian_noise(rfc_x_train_res)
rfc_y_train_synthetic = rfc_y_train_res.copy()

rfc_x_train_noise = add_gaussian_noise(rfc_x_train_res, noise_level=0.05)
rfc_y_train_noise = rfc_y_train_res.copy()

rfc_x_train_final = np.vstack((rfc_x_train_res, rfc_x_train_synthetic))
rfc_y_train_final = np.hstack((rfc_y_train_res, rfc_y_train_synthetic))
print("Final class balance:", Counter(rfc_y_train_final))
```

Figure 11. 10 Model Training Code Snippet



```python
def extract_metrics(report):
    return {
        'Precision (0)': report['0']['precision']*100,
        'Recall (0)': report['0']['recall']*100,
        'F1-score (0)': report['0']['f1-score']*100,
        'Precision (1)': report['1']['precision']*100,
        'Recall (1)': report['1']['recall']*100,
        'F1-score (1)': report['1']['f1-score']*100,
        'Overall Model Accuracy': report['accuracy']*100
    }

xgb_report = classification_report(xgb_y_test, xgb_final_preds, output_dict=True)
rfc_report = classification_report(rfc_y_test, rfc_final_preds, output_dict=True)
dtc_report = classification_report(dtc_y_test, dtc_final_preds, output_dict=True)
logreg_report = classification_report(logreg_y_test, logreg_final_preds, output_dict=True)
svm_report = classification_report(svm_y_test, svm_final_preds, output_dict=True)

comparison_df = pd.DataFrame({
    'XGBoost': extract_metrics(xgb_report),
    'Random Forest': extract_metrics(rfc_report),
    'Decision Tree': extract_metrics(dtc_report),
    'Logistic Regression': extract_metrics(logreg_report),
    'SVM (RBF)': extract_metrics(svm_report)
}).T

comparison_df_sorted = comparison_df.sort_values(by=['Precision (1)',
                                                     'Recall (1)', 'Overall Model Accuracy'], ascending=False)

comparison_df_sorted.style.format(precision=4).set_table_attributes("style='border:1px solid black'").set_table_styles(
    [{'selector': 'th', 'props': [('border', '1px solid black')]},
     {'selector': 'td', 'props': [('border', '1px solid black')]}]
).set_caption("Model Comparison Table (Sorted by Precision, Recall, and Accuracy) in Percentage")
```

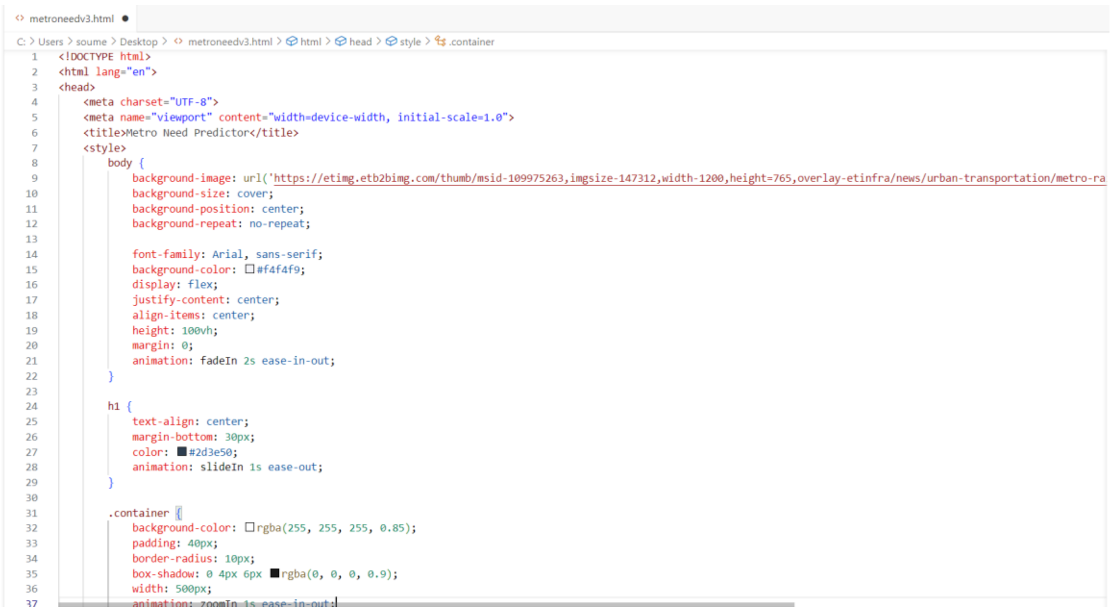
Figure 11. 11 Model Comparison Code Snippet



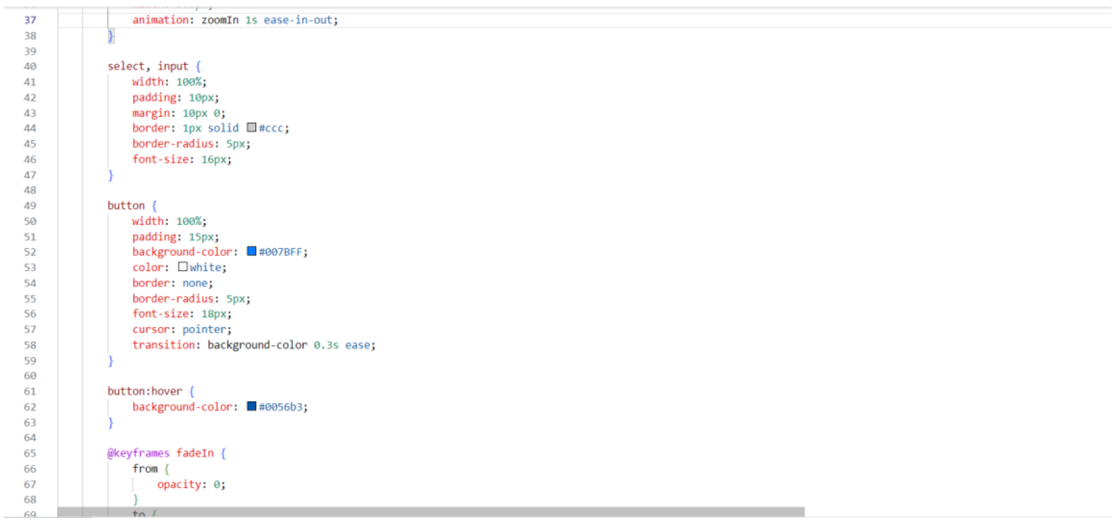
```python
!pip install flask-ngrok
!pip install flask-ngrok --upgrade
!pip install flask==0.12.2
!pip install flask --upgrade

from markupsafe import Markup
from flask import Flask, request, jsonify
from flask_ngrok import run_with_ngrok
import pickle
import numpy as np
import os
app = Flask(__name__)
run_with_ngrok(app)
current_dir = os.getcwd()

# Construct the paths to the model and scaler files
model_path = os.path.join(current_dir, 'predictor.pkl')
scaler_path = os.path.join(current_dir, 'scaler.pkl')


# Load the model and scaler using the absolute paths
with open(model_path, 'rb') as f:
    model = pickle.load(f)

with open(scaler_path, 'rb') as f:
    scaler = pickle.load(f)

# Define the prediction route
@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Parse input data
        data = request.json

        # Map 'Risk' to numeric values
        risk_mapping = {'Low': 1, 'Medium': 2, 'High': 3}
        risk = risk_mapping.get(data['risk'], 0)  # Default to 0 if invalid risk is provided
```

```python
        # Extract other features
        population = float(data['population'])
        commute = float(data['commute'])
        airport = 1 if data['airport'] == 'Yes' else 0  # Convert Yes/No to binary
        trip = float(data['trip'])

        # Prepare feature array
        features = np.array([risk, population, commute, airport, trip]).reshape(1, -1)

        # Scale and predict
        scaled_features = scaler.transform(features)
        prediction = model.predict(scaled_features)

        # Return the prediction
        return jsonify({'prediction': prediction[0]})

    except Exception as e:
        return jsonify({'error': str(e)}), 500

# Start the Flask app
if __name__ == '__main__':
    app.run()
```

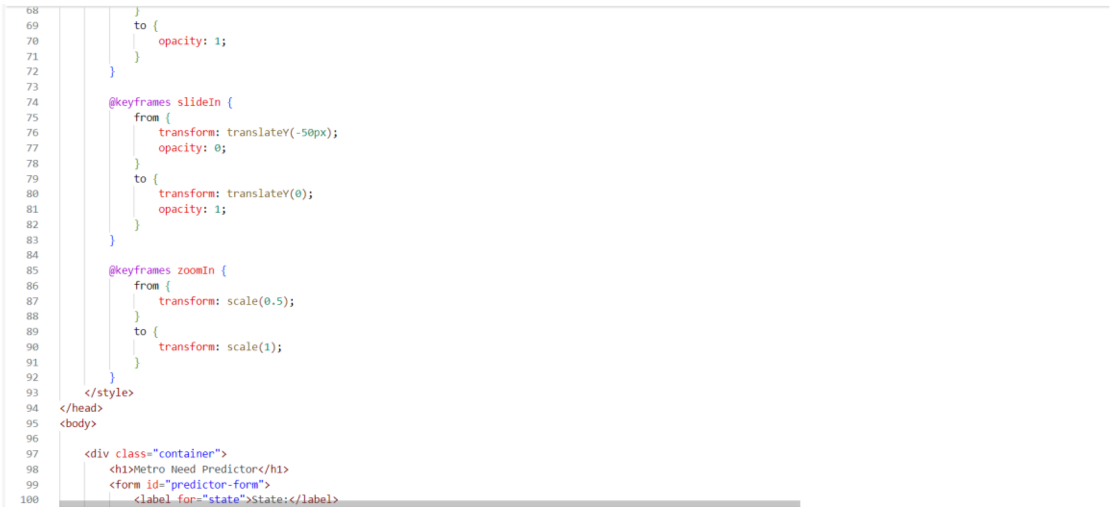
Figure 11. 12 Flask app and NGROK Setup Snippet


metroneedv3.html
C: > Users > soume > Desktop > metroneedv3.html > html > head > style > .container

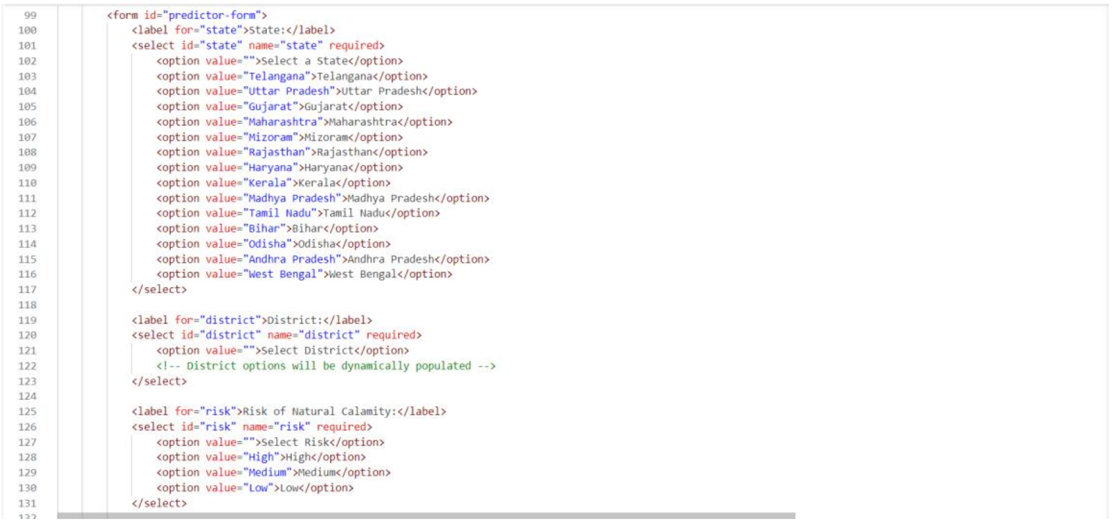
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Metro Need Predictor</title>
    <style>
        body {
            background-image: url('https://etimg.etb2bimg.com/thumb/msid-109975263,imgsize-147312,width-1200,height=765,overlay-etinfra/news/urban-transportation/metro-ra
            background-size: cover;
            background-position: center;
            background-repeat: no-repeat;

            font-family: Arial, sans-serif;
            background-color: #f4f4f9;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            animation: fadeIn 2s ease-in-out;
        }

        h1 {
            text-align: center;
            margin-bottom: 30px;
            color: #2d3e50;
            animation: slideIn 1s ease-out;
        }

        .container {
            background-color: rgba(255, 255, 255, 0.85);
            padding: 40px;
            border-radius: 10px;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.9);
            width: 500px;
            animation: zoomIn 1s ease-in-out;
```



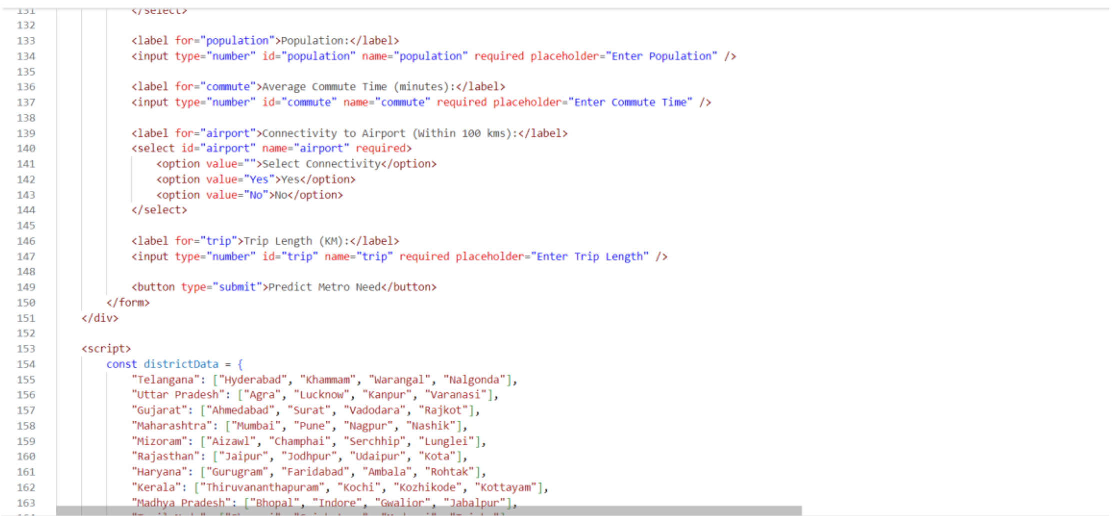
```
            animation: zoomIn 1s ease-in-out;
        }

        select, input {
            width: 100%;
            padding: 10px;
            margin: 10px 0;
            border: 1px solid #ccc;
            border-radius: 5px;
            font-size: 16px;
        }

        button {
            width: 100%;
            padding: 15px;
            background-color: #007BFF;
            color: white;
            border: none;
            border-radius: 5px;
            font-size: 18px;
            cursor: pointer;
            transition: background-color 0.3s ease;
        }

        button:hover {
            background-color: #0056b3;
        }

        @keyframes fadeIn {
            from {
                opacity: 0;
            }
            to {
```



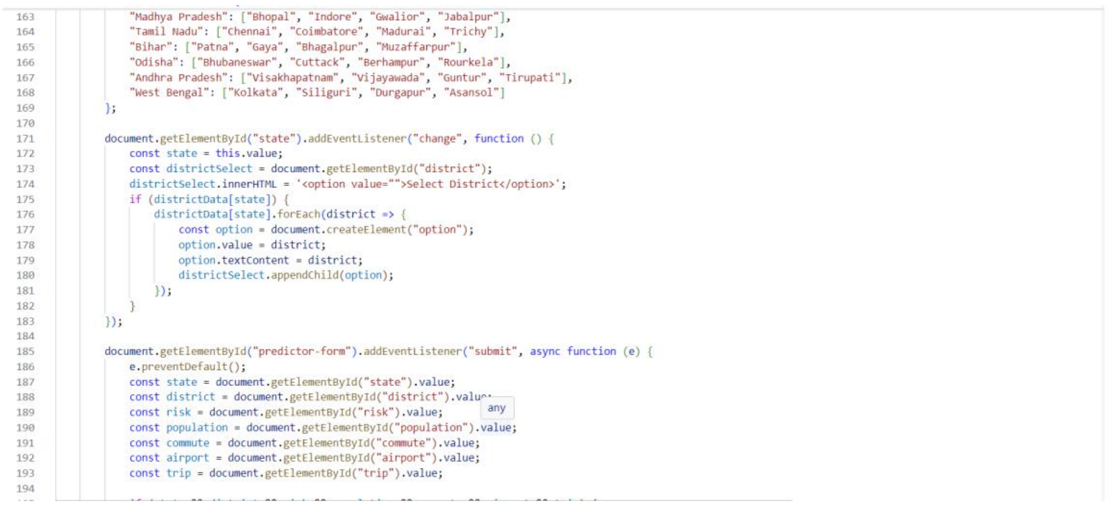
```
            }
            to {
                opacity: 1;
            }
        }

        @keyframes slideIn {
            from {
                transform: translateY(-50px);
                opacity: 0;
            }
            to {
                transform: translateY(0);
                opacity: 1;
            }
        }

        @keyframes zoomIn {
            from {
                transform: scale(0.5);
            }
            to {
                transform: scale(1);
            }
        }
    </style>
</head>
<body>

    <div class="container">
        <h1>Metro Need Predictor</h1>
        <form id="predictor-form">
            <label for="state">State:</label>
```



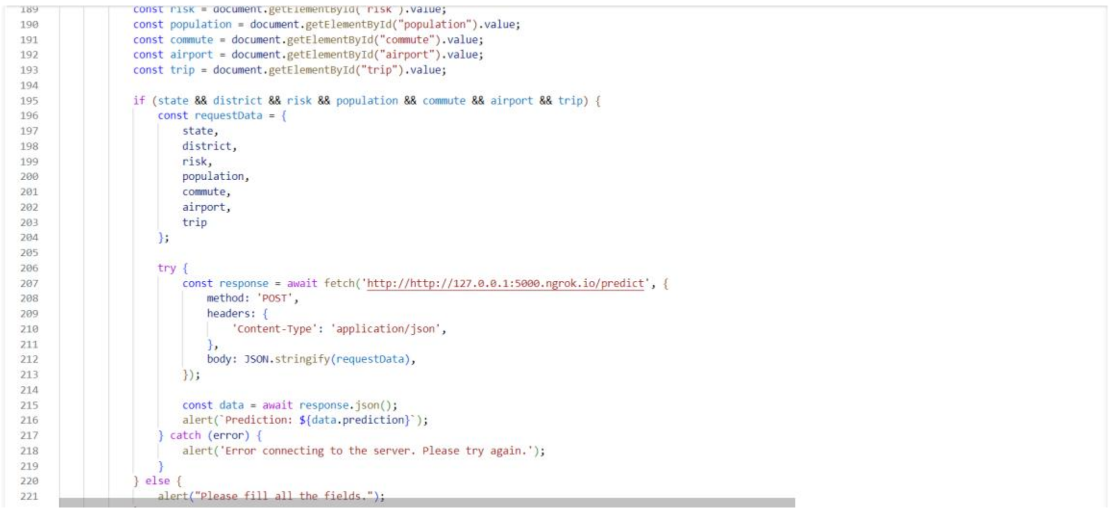
```
        <form id="predictor-form">
            <label for="state">State:</label>
            <select id="state" name="state" required>
                <option value="">Select a State</option>
                <option value="Telangana">Telangana</option>
                <option value="Uttar Pradesh">Uttar Pradesh</option>
                <option value="Gujarat">Gujarat</option>
                <option value="Maharashtra">Maharashtra</option>
                <option value="Mizoram">Mizoram</option>
                <option value="Rajasthan">Rajasthan</option>
                <option value="Haryana">Haryana</option>
                <option value="Kerala">Kerala</option>
                <option value="Madhya Pradesh">Madhya Pradesh</option>
                <option value="Tamil Nadu">Tamil Nadu</option>
                <option value="Bihar">Bihar</option>
                <option value="Odisha">Odisha</option>
                <option value="Andhra Pradesh">Andhra Pradesh</option>
                <option value="West Bengal">West Bengal</option>
            </select>

            <label for="district">District:</label>
            <select id="district" name="district" required>
                <option value="">Select District</option>
                <!-- District options will be dynamically populated -->
            </select>

            <label for="risk">Risk of Natural Calamity:</label>
            <select id="risk" name="risk" required>
                <option value="">Select Risk</option>
                <option value="High">High</option>
                <option value="Medium">Medium</option>
                <option value="Low">Low</option>
            </select>
```



```
            </select>

            <label for="population">Population:</label>
            <input type="number" id="population" name="population" required placeholder="Enter Population" />

            <label for="commute">Average Commute Time (minutes):</label>
            <input type="number" id="commute" name="commute" required placeholder="Enter Commute Time" />

            <label for="airport">Connectivity to Airport (Within 100 kms):</label>
            <select id="airport" name="airport" required>
                <option value="">Select Connectivity</option>
                <option value="Yes">Yes</option>
                <option value="No">No</option>
            </select>

            <label for="trip">Trip Length (KM):</label>
            <input type="number" id="trip" name="trip" required placeholder="Enter Trip Length" />

            <button type="submit">Predict Metro Need</button>
        </form>
    </div>

    <script>
        const districtData = {
            "Telangana": ["Hyderabad", "Khammam", "Warangal", "Nalgonda"],
            "Uttar Pradesh": ["Agra", "Lucknow", "Kanpur", "Varanasi"],
            "Gujarat": ["Ahmedabad", "Surat", "Vadodara", "Rajkot"],
            "Maharashtra": ["Mumbai", "Pune", "Nagpur", "Nashik"],
            "Mizoram": ["Aizawl", "Champhai", "Serchhip", "Lunglei"],
            "Rajasthan": ["Jaipur", "Jodhpur", "Udaipur", "Kota"],
            "Haryana": ["Gurugram", "Faridabad", "Ambala", "Rohtak"],
            "Kerala": ["Thiruvananthapuram", "Kochi", "Kozhikode", "Kottayam"],
            "Madhya Pradesh": ["Bhopal", "Indore", "Gwalior", "Jabalpur"],
```

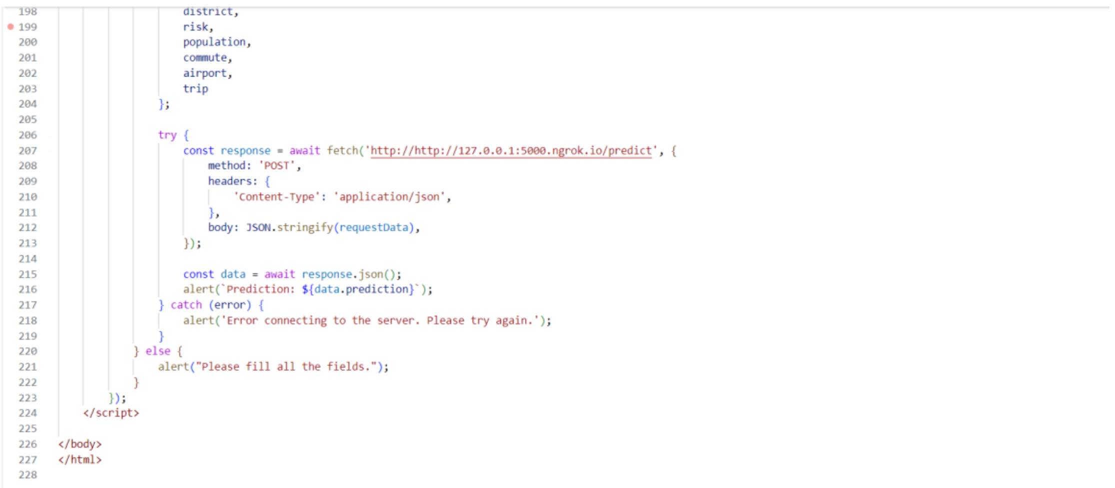


```
            "Madhya Pradesh": ["Bhopal", "Indore", "Gwalior", "Jabalpur"],
            "Tamil Nadu": ["Chennai", "Coimbatore", "Madurai", "Trichy"],
            "Bihar": ["Patna", "Gaya", "Bhagalpur", "Muzaffarpur"],
            "Odisha": ["Bhubaneswar", "Cuttack", "Berhampur", "Rourkela"],
            "Andhra Pradesh": ["Visakhapatnam", "Vijayawada", "Guntur", "Tirupati"],
            "West Bengal": ["Kolkata", "Siliguri", "Durgapur", "Asansol"]
        };

        document.getElementById("state").addEventListener("change", function () {
            const state = this.value;
            const districtSelect = document.getElementById("district");
            districtSelect.innerHTML = '<option value="">Select District</option>';
            if (districtData[state]) {
                districtData[state].forEach(district => {
                    const option = document.createElement("option");
                    option.value = district;
                    option.textContent = district;
                    districtSelect.appendChild(option);
                });
            }
        });

        document.getElementById("predictor-form").addEventListener("submit", async function (e) {
            e.preventDefault();
            const state = document.getElementById("state").value;
            const district = document.getElementById("district").valu
            const risk = document.getElementById("risk").value;      any
            const population = document.getElementById("population").value;
            const commute = document.getElementById("commute").value;
            const airport = document.getElementById("airport").value;
            const trip = document.getElementById("trip").value;

```



```
            const risk = document.getElementById("risk").value;
            const population = document.getElementById("population").value;
            const commute = document.getElementById("commute").value;
            const airport = document.getElementById("airport").value;
            const trip = document.getElementById("trip").value;

```

```
            if (state && district && risk && population && commute && airport && trip) {
                const requestData = {
                    state,
                    district,
                    risk,
                    population,
                    commute,
                    airport,
                    trip
                };

                try {
                    const response = await fetch('http://http://127.0.0.1:5000.ngrok.io/predict', {
                        method: 'POST',
                        headers: {
                            'Content-Type': 'application/json',
                        },
                        body: JSON.stringify(requestData),
                    });

                    const data = await response.json();
                    alert(`Prediction: ${data.prediction}`);
                } catch (error) {
                    alert('Error connecting to the server. Please try again.');
                }
            } else {
                alert("Please fill all the fields.");
```



```
                    district,
                    risk,
                    population,
                    commute,
                    airport,
                    trip
                };

                try {
                    const response = await fetch('http://http://127.0.0.1:5000.ngrok.io/predict', {
                        method: 'POST',
                        headers: {
                            'Content-Type': 'application/json',
                        },
                        body: JSON.stringify(requestData),
                    });

                    const data = await response.json();
                    alert(`Prediction: ${data.prediction}`);
                } catch (error) {
                    alert('Error connecting to the server. Please try again.');
                }
            } else {
                alert("Please fill all the fields.");
```

```
            }
        });
    </script>

</body>
</html>

```

Fig 11. 13 HTML, JavaScript & CSS code with integration with API Snippet