**COLLEGE CODE-** 3116

**COLLEGE NAME-** Misrimal Navajee Munoth Jain Engineering College

**DEPARTMENT-** CSE

**STUDENT NM ID-**9705C6DD32DA63983D74022A05D23043

**ROLL NO.-** 311623104022

**DATE-** 07-05-2025

**TECHNOLOGY-** AUTONOMOUS VEHICLES AND ROBOTICS

**SUBMITTED BY-**

1) Kandiyuri Kalyana Varadan(kvaradan2005@gmail.com)

2)Arunachalam T(arunachalama867@gmail.com)

3) SriPrasath S(sriprasath0106@gmail.com)

4) Ajay S(ajay01cbi@gmail.com)

5) Anuprasad C(anuprasadspidey07@gmail.com)

**Phase 5: Project Demonstration & Documentation**

Title: AUTONOMOUS VEHICLES AND ROBOTICS

# Abstract:

The **AUTONOMOUS VEHICLES AND ROBOTICS** project aims to revolutionize vehicle accessibility by leveraging artificial intelligence, natural language processing, and IoT (Internet of Things) technologies. In its final phase, the system integrates advanced AI models to vehicle issues, real-time health data collection from IoT devices, and secure data management, while ensuring scalability and seamless integration. This document provides a comprehensive report of the project’s completion, covering the system demonstration, technical documentation, perform vehicle metrics, source code, and testing reports. The project is designed to handle large-scale operations with robust data security measures, providing accurate vehicle recommendations in real-time. Screenshots and codebase snapshots will be included for a full understanding of the system's architecture and functionality.

|  |  |  |
| --- | --- | --- |
| **S.NO.** | **TOPIC** | **PAGE.NO.** |
| 1 | Project Demonstration | 3 |
| 2 | Project Documentation | 3-4 |
| 3 | Feedback and Final Adjustments | 4 |
| 4 | Final Project Report | 4-5 |
| 5 | Project Handover and Future | 6-8 |
| 6 | Screenshots of Working Project | 9-11 |

# Project Demonstration

# Overview:

The Autonomous vehicles and robotics will be demonstrated to stakeholders, showcasing its features, performance improvements, and functionality. This demonstration highlights the system’s real-time responses, IoT data integration, security measures, and performance scalability.

# Demonstration Details:

# The demonstration will include the following components:

# Real-Time Navigation: Live showcasing of the autonomous vehicle’s ability to navigate a dynamic environment using onboard sensors and machine learning algorithms.

# IoT Data Integration: Display of how the system ingests and processes IoT sensor data (e.g., traffic signals, weather data) to make intelligent driving decisions.

# Security Measures: Overview of the cybersecurity framework protecting vehicle communication and data exchange.

# Performance Testing: Demonstration of the system under various loads and scenarios to illustrate scalability and fault tolerance.

# User Interface: A walkthrough of the monitoring dashboard or control interface used by operators or administrators

# Outcome:

By the end of the demonstration, the system’s ability to handle real-world scenarios, ensure data security, and deliver health insights through IoT integration will be showcased to the stakeholders.

# Project Documentation

# Overview:

Comprehensive documentation for the Autonomous vehicles and robotics is provided to detail every aspect of the project. This includes system architecture, AI model details, code explanations, and usage guidelines for both users and administrators.

# Documentation Sections:

* + **System Architecture:** Diagrams illustrating the complete system, including AI algorithms.
  + **Code Documentation:** Source code and explanations for all code modules, including AI training scripts, API integrations for IoT devices, and chatbot interactions.
  + **User Guide:** A manual for end users explaining how to interact with the AI assistant and how to interpret vehicle data and recommendations.
  + **Administrator Guide:** Instructions for system maintenance, monitoring, and performance testing procedures.
  + **Testing Reports:** Detailed reports on performance metrics, load testing, and data security evaluations.

# Outcome:

All critical components of the system will be well-documented, providing a clear guide for future development, deployment, or system scaling.

# Feedback and Final Adjustments

# Overview:

Feedback from the project demonstration will be collected from instructors, stakeholders, and a broader group of test users. This feedback will be used to make final refinements before project handover.

# Steps:

* + **Feedback Collection:** Feedback from mentors, stakeholders, and test users will be gathered via surveys and observation during the demonstration.
  + **Refinement:** Based on the feedback, improvement in sensors and monitoring devices or usability issues will be addressed.
  + **Final Testing:** After adjustments, the system will undergo final testing to ensure full functionality, usability, and scalability.

# Outcome:

Final adjustments will optimize the system for a broader rollout, ensuring that it is fully ready for real-world deployment.

# Final Project Report Submission Overview:

The final project report provides a comprehensive summary of all phases, key achievements, challenges faced, and outcomes of the AI-Powered Healthcare Assistant project. This report will include testing results, performance improvements, and future recommendations.

# Report Sections:

* + **Executive Summary:** A concise overview of the project, outlining its objectives and major achievements.
  + **Phase Breakdown:** A detailed breakdown of each phase, covering AI model development, chatbot improvements, IoT integration, and data security.
  + **Challenges & Solutions:** A section documenting the key challenges encountered, such as AI misdiagnosis or security under load, and how they were resolved.
  + **Outcomes:** A summary of the system’s current capabilities and readiness for deployment.

# Outcome:

A detailed project report will be submitted, outlining the entire journey from concept to completion.

# Project Handover and Future:

# Overview:

The projects intro for future development.

# Handover Details:

* + **Next Steps:** Suggestions for future work, including scaling the system to support more users, expanding AI capabilities, and implementing multilingual support, will be provided.

# Outcome:

The Autonomous vehicles and robotics will be officially handed over, along with recommendations for future enhancements and guidelines for system maintenance.

**CODE SNIPPET OF FINAL WORKING PROJECT:**

import tkinter as tk

from tkinter import messagebox

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import LabelEncoder

from collections import deque

graph = {}

road\_data = []

def add\_road():

src = src\_entry.get().strip()

dst = dst\_entry.get().strip()

try:

dist = float(dist\_entry.get())

traffic = float(traffic\_entry.get())

except ValueError:

messagebox.showerror("Input Error", "Distance and traffic must be numbers.")

return

bidirectional = bidir\_var.get()

if src not in graph:

graph[src] = {}

graph[src][dst] = (dist, traffic)

if bidirectional:

if dst not in graph:

graph[dst] = {}

graph[dst][src] = (dist, traffic)

road\_data.append({'src': src, 'dst': dst, 'distance': dist, 'traffic': traffic, 'cost': dist \* traffic})

messagebox.showinfo("Success", f"Road added from {src} to {dst}")

src\_entry.delete(0, tk.END)

dst\_entry.delete(0, tk.END)

dist\_entry.delete(0, tk.END)

traffic\_entry.delete(0, tk.END)

def create\_dataset():

return pd.DataFrame(road\_data)

def train\_model():

global model, le\_src, le\_dst, df

if not road\_data:

messagebox.showerror("Error", "No roads added.")

return

df = create\_dataset()

le\_src = LabelEncoder()

le\_dst = LabelEncoder()

df['src\_enc'] = le\_src.fit\_transform(df['src'])

df['dst\_enc'] = le\_dst.fit\_transform(df['dst'])

X = df[['src\_enc', 'dst\_enc', 'distance', 'traffic']]

y = df['cost']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = RandomForestRegressor(n\_estimators=100)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

messagebox.showinfo("Model Trained", f"Mean Squared Error: {mse:.3f}")

open\_predict\_window()

def predict\_cost(model, le\_src, le\_dst, src, dst, distance, traffic):

src\_enc = le\_src.transform([src])[0]

dst\_enc = le\_dst.transform([dst])[0]

X\_input = pd.DataFrame([[src\_enc, dst\_enc, distance, traffic]], columns=['src\_enc', 'dst\_enc', 'distance', 'traffic'])

return model.predict(X\_input)[0]

def bfs\_path(graph, start, goal):

queue = deque([[start]])

visited = set()

while queue:

path = queue.popleft()

node = path[-1]

if node == goal:

return path

if node not in visited:

visited.add(node)

for neighbor in graph.get(node, []):

new\_path = list(path)

new\_path.append(neighbor)

queue.append(new\_path)

return None

def display\_path\_with\_icons(source, destination):

path = bfs\_path(graph, source, destination)

if not path:

return "No path found."

icon\_path = []

for i, node in enumerate(path):

if i == 0:

icon\_path.append(f"📍{node}")

elif i == len(path) - 1:

icon\_path.append(f"🏁{node}")

else:

icon\_path.append(f"🚏{node}")

return " 🛣 ".join(icon\_path)

def open\_predict\_window():

pred\_window = tk.Toplevel(root)

pred\_window.title("Cost Predictor")

def run\_prediction():

src = pred\_src.get().strip()

dst = pred\_dst.get().strip()

try:

distance = float(pred\_distance.get())

traffic = float(pred\_traffic.get())

time\_min = float(pred\_time.get())

except ValueError:

messagebox.showerror("Input Error", "Please enter valid numbers.")

return

if src not in df['src'].unique() or dst not in df['dst'].unique():

messagebox.showerror("Error", "Source or Destination not found in training data.")

return

predicted = predict\_cost(model, le\_src, le\_dst, src, dst, distance, traffic)

path = display\_path\_with\_icons(src, dst)

time\_hours = time\_min / 60

max\_speed = distance / time\_hours if time\_hours else 0

min\_speed = (distance \* traffic) / time\_hours if time\_hours else 0

result\_text = f"Predicted Cost: {predicted:.2f}\nRoute: {path}\n\n➡️ Max Speed: {max\_speed:.2f} km/h\n⬅️ Min Speed: {min\_speed:.2f} km/h"

output.config(text=result\_text)

tk.Label(pred\_window, text="Source:").grid(row=0, column=0)

pred\_src = tk.Entry(pred\_window)

pred\_src.grid(row=0, column=1)

tk.Label(pred\_window, text="Destination:").grid(row=1, column=0)

pred\_dst = tk.Entry(pred\_window)

pred\_dst.grid(row=1, column=1)

tk.Label(pred\_window, text="Distance (km):").grid(row=2, column=0)

pred\_distance = tk.Entry(pred\_window)

pred\_distance.grid(row=2, column=1)

tk.Label(pred\_window, text="Traffic (1.0-1.7):").grid(row=3, column=0)

pred\_traffic = tk.Entry(pred\_window)

pred\_traffic.grid(row=3, column=1)

tk.Label(pred\_window, text="Time (minutes):").grid(row=4, column=0)

pred\_time = tk.Entry(pred\_window)

pred\_time.grid(row=4, column=1)

tk.Button(pred\_window, text="Predict", command=run\_prediction).grid(row=5, column=0, columnspan=2, pady=10)

output = tk.Label(pred\_window, text="", wraplength=300, justify="left")

output.grid(row=6, column=0, columnspan=2)

root = tk.Tk()

root.title("Road Network Builder")

root.geometry("400x400")

ENTRY\_WIDTH = 30

PAD\_Y = 8

tk.Label(root, text="Source:").grid(row=0, column=0, columnspan=2, pady=PAD\_Y)

src\_entry = tk.Entry(root, width=ENTRY\_WIDTH)

src\_entry.grid(row=1, column=0, columnspan=2, pady=PAD\_Y)

tk.Label(root, text="Destination:").grid(row=2, column=0, columnspan=2, pady=PAD\_Y)

dst\_entry = tk.Entry(root, width=ENTRY\_WIDTH)

dst\_entry.grid(row=3, column=0, columnspan=2, pady=PAD\_Y)

tk.Label(root, text="Distance (km):").grid(row=4, column=0, columnspan=2, pady=PAD\_Y)

dist\_entry = tk.Entry(root, width=ENTRY\_WIDTH)

dist\_entry.grid(row=5, column=0, columnspan=2, pady=PAD\_Y)

tk.Label(root, text="Traffic (1.0 - 1.7):").grid(row=6, column=0, columnspan=2, pady=PAD\_Y)

traffic\_entry = tk.Entry(root, width=ENTRY\_WIDTH)

traffic\_entry.grid(row=7, column=0, columnspan=2, pady=PAD\_Y)

bidir\_var = tk.BooleanVar()

bidir\_check = tk.Checkbutton(root, text="Bidirectional", variable=bidir\_var)

bidir\_check.grid(row=8, column=0, columnspan=2, pady=PAD\_Y)

tk.Button(root, text="➕ Add Road", width=20, command=add\_road).grid(row=9, column=0, columnspan=2, pady=PAD\_Y)

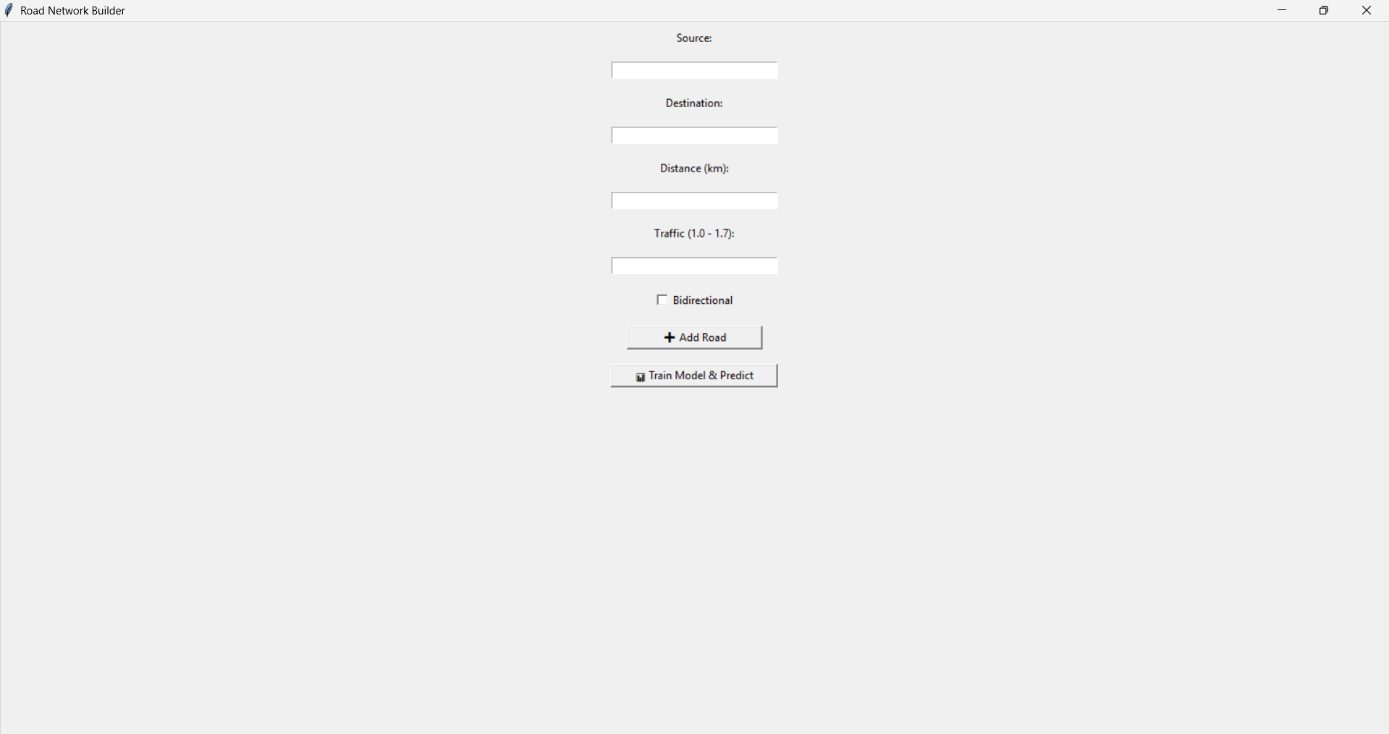
tk.Button(root, text="📊 Train Model & Predict", width=25, command=train\_model).grid(row=10, column=0, columnspan=2, pady=PAD\_Y)

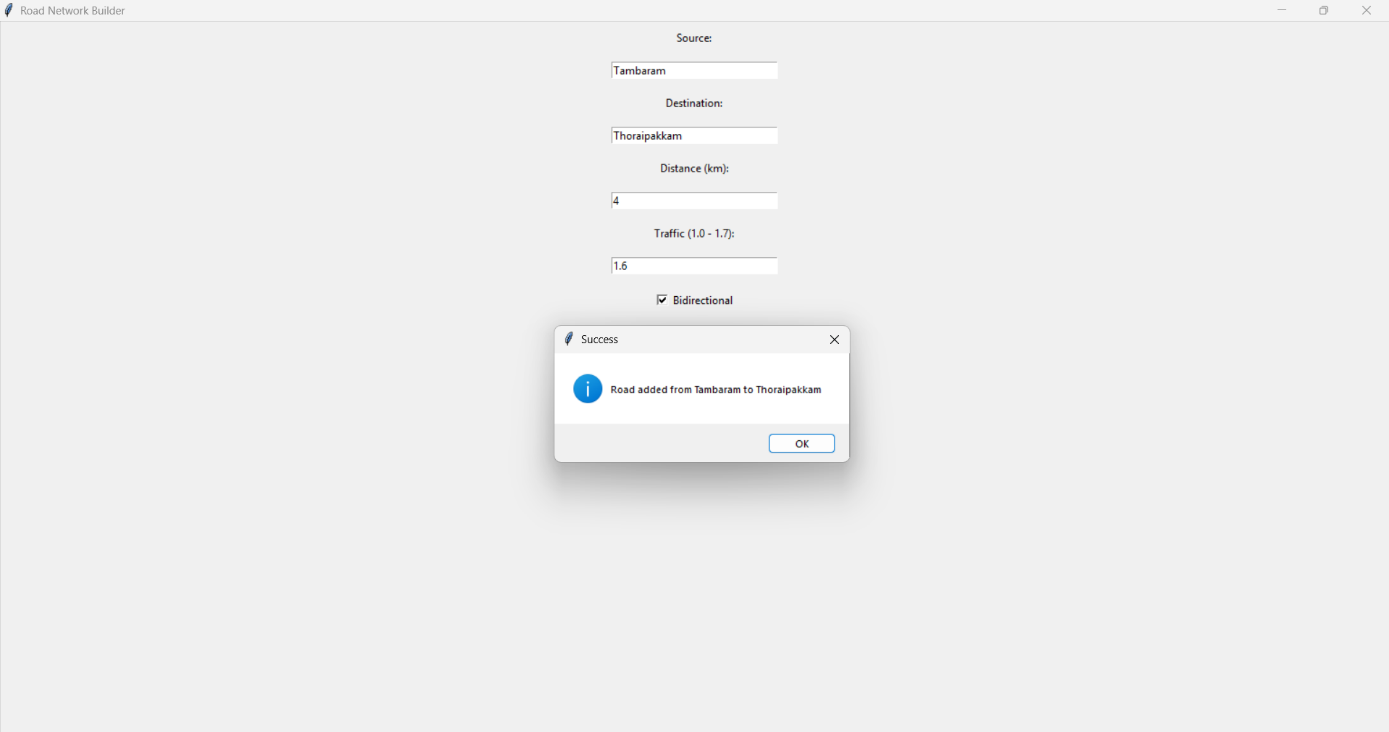
root.grid\_columnconfigure(0, weight=1)

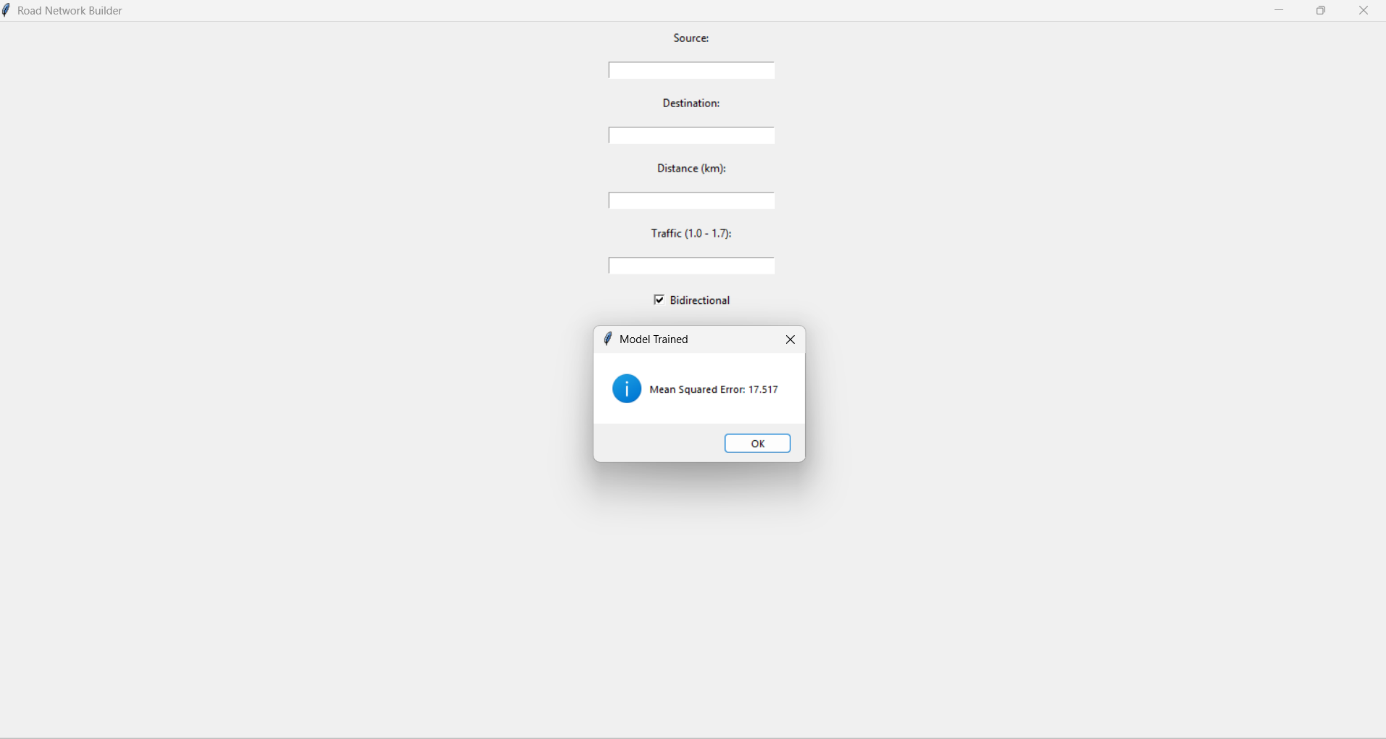
root.grid\_columnconfigure(1, weight=1)

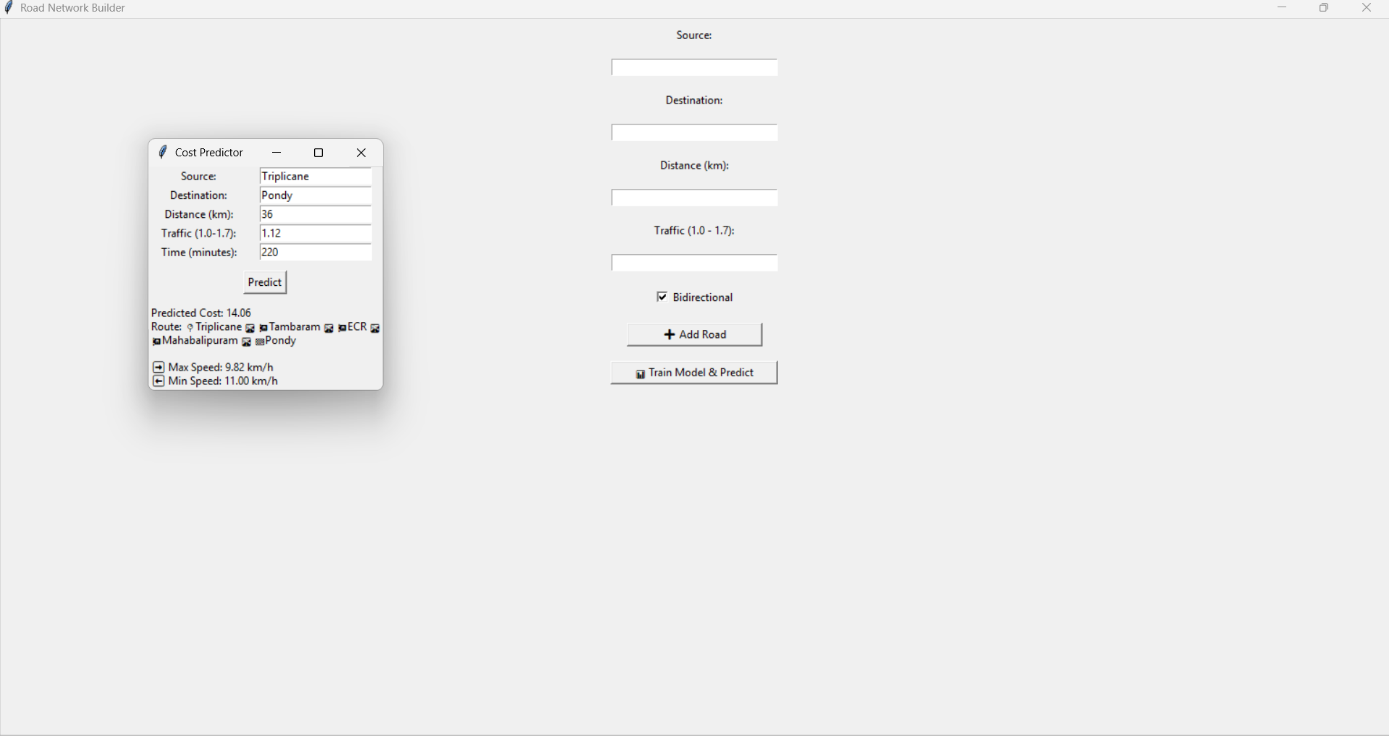
root.mainloop()

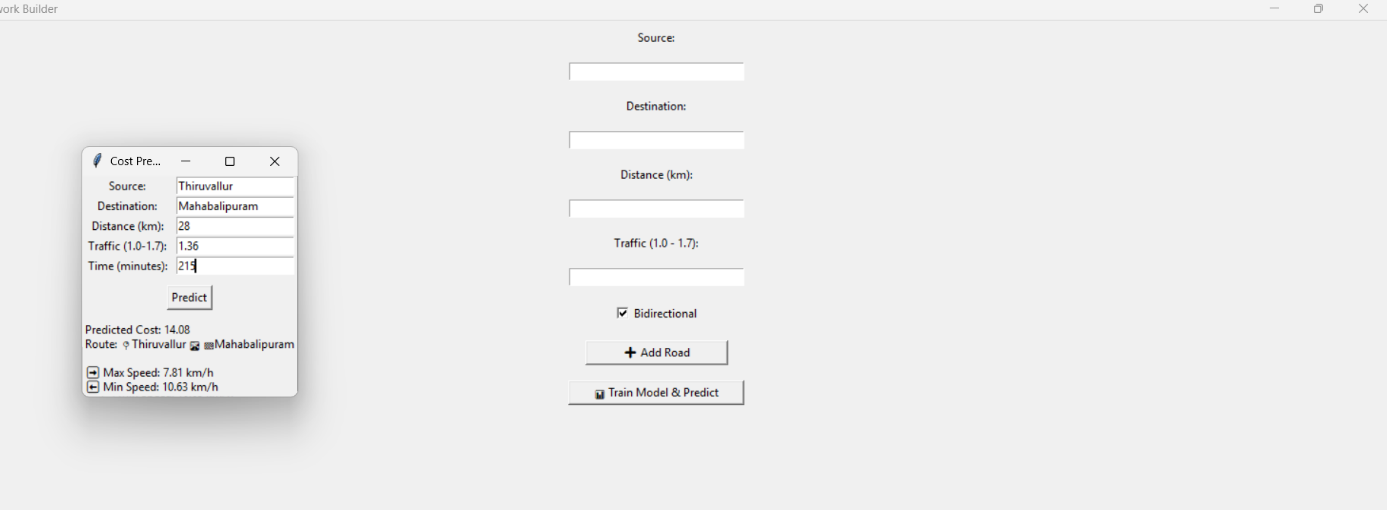
**OUTPUT FOR THE FINAL PROJECT:**

****

****

****

****

****