

Submitted by Harini S (231801049)

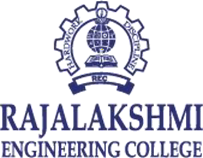
Hemalatha L (231801055)

**AI23331 - FUNDAMENTALS OF MACHINE LEARNING**

**Department of Artificial Intelligence and Data Science**

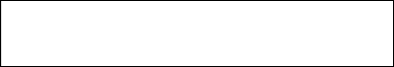
**Rajalakshmi Engineering College ,Thandalam**

**Nov 2024**



BONAFIDE CERTIFICATE

**NAME……………………………………………………………………………………. ACADEMIC YEAR…………………SEMESTER…………. BRANCH …………….**

**UNIVERSITY REGISTER No. **

Certified that this is the bonafide record of work done by the above students in the Mini Project titled " " in the subject **AI23331 – FUNDAMENTALS OF MACHINE**

**LEARNING** during the year **2024 - 2025.**

**Signature of Faculty – in – Charge Submitted for the Practical Examination held on -----------------------**

**Internal Examiner External Examiner**

**TABLE OF CONTENTS**

**CHAPTER NO. TITLE PAG NO. ABSTRACT**

1. **INTRODUCTION (CHAPTER 1)**
   1. GENERAL
   2. NEED FOR THE STUDY
   3. OVERVIEW OF THE PROJECT
   4. OBJECTIVE OF THE STUDY
   5. ALGORITHMS USED
2. **SYSTEM REQUIREMENT**
   1. HARWARE REQUIREMENTS
   2. SOFTWAREREQUIREMEN
3. **SYSTEM OVERVIEW**
   1. IMPLEMENTATION
   2. ALGORITHM
   3. SOURCE CODE
4. **RESULT AND DISCUSSION**
5. **CONCLUSION**
6. **REFERENCE**

# ABSTRACT

Urban traffic congestion and unpredictable travel times significantly impact mobility and efficiency in modern cities, emphasizing the need for accurate and reliable travel time prediction systems. This project presents an innovative solution by developing a machine learning-powered ***Tra\_Well*** system that uses linear regression to predict travel times for road journeys. The model considers critical factors such as route length, traffic conditions, and time of day—variables that are intricately linked to travel efficiency.

To ensure precision and robustness, the dataset undergoes comprehensive preprocessing, including scaling numerical features like route length and encoding traffic conditions to reflect their dynamic nature. These steps ensure balanced contributions from all features, allowing the model to learn nuanced patterns effectively. The trained linear regression model generates accurate travel time predictions, enabling users to make informed decisions about route selection.

The model’s performance is rigorously evaluated using metrics such as Mean Squared Error (MSE) and R-squared, which confirm its reliability and effectiveness. Beyond numerical predictions, the project integrates an interactive mapping feature, offering users a visual representation of the optimal route alongside key insights like distance and predicted travel time.

This system delivers a user-friendly, cost-effective, and highly interpretable tool that supports travellers, urban planners, and transportation managers. By providing actionable insights and enhancing route planning efficiency, ***Tra\_Well*** contributes to improved traffic management, reduced delays, and elevated travel experiences in urban environments.

**CHAPTER 1**

**INTRODUCTION**

**1.1 GENERAL**

Efficient route planning is a critical challenge in urban mobility systems, as traffic congestion and delays not only disrupt daily schedules but also have far-reaching effects on logistics, economic productivity, and environmental sustainability. These delays lead to increased fuel consumption, higher transportation costs, and elevated levels of air pollution, making effective traffic management an urgent necessity for modern cities. A reliable system to predict travel times and recommend optimal routes can significantly improve mobility for travelers while enabling urban planners to optimize infrastructure. By leveraging machine learning to analyze traffic patterns, travel demand, and route characteristics, such systems can deliver real-time, actionable insights. This transformative approach not only benefits individual commuters but also aids policymakers in developing more efficient and sustainable urban transportation networks. As traffic systems grow increasingly complex, predictive solutions like *Tra\_Well* are essential for addressing the multifaceted challenges of urban mobility.

**1.2 NEED FOR THE STUDY**

With the increasing complexity of urban traffic, manual estimations of travel time are no longer sufficient to meet the demands of modern commuters. The unpredictability of traffic patterns, influenced by factors like congestion, time of day, and road conditions, requires advanced tools for accurate predictions. Integrating a machine learning model can provide actionable insights by analyzing real-time data, route lengths, traffic conditions, and other critical parameters. Such systems can dynamically adapt to changing conditions, offering users more reliable travel time estimates. By predicting travel times, systems like *Tra\_Well* can significantly reduce delays and optimize route selection. This enhances trip planning, allowing travelers to make informed decisions about their journeys. Furthermore, it supports urban planners and transportation managers in designing efficient road networks. The combination of accurate predictions and interactive visualization empowers both individuals and policymakers. Ultimately, this improves the overall travel experience while contributing to smarter traffic management strategies.

**1.3 OBJECTIVE OF THE PROJECT**

The objective of the Tra\_well project is to develop a predictive model that estimates travel time based on route length, traffic conditions, and time of day. It also aims to incorporate interactive mapping for visualizing optimal routes, providing users with an intuitive and efficient planning tool. By evaluating model performance through metrics such as Mean Squared Error (MSE) and R-squared, the project ensures reliability while delivering a cost-effective, user-friendly system to enhance route planning and address urban transportation challenges effectively.

**1.4 OBJECTIVE OF THE STUDY**

This study has several specific objectives:

* To analyse historical travel data and geospatial information, identifying key features such as route length, traffic conditions, and time of day that influence travel time predictions.
* To develop a predictive model using machine learning techniques, optimizing its parameters to maximize accuracy and minimize prediction errors.
* To evaluate model performance using metrics such as Mean Squared Error (MSE) and R-squared, ensuring reliability and robustness.
* To provide insights into the significance of each feature influencing travel time, offering valuable interpretability for users and urban planners.
* To compare the chosen machine learning model with alternative predictive algorithms, validating its effectiveness and identifying potential improvements.

**ALGORITHM USED**

In the Tra-Well project, the Decision Tree Regression algorithm is used to predict the travel time between two cities based on input features such as route length, traffic condition, and time of day. The Decision Tree algorithm works by recursively partitioning the data, selecting the feature that best reduces the mean squared error (MSE) at each decision node. This process continues until the tree reaches a predefined maximum depth or further splits no longer improve the model's prediction accuracy. Each leaf node in the tree holds the predicted travel time, which is derived from the average travel times of the data points that reach that leaf.

The model is trained on historical travel data, where features like route length (distance between cities), traffic condition (simulated or real-time data), and time of day (hour of the day) are mapped to corresponding travel times. The model’s performance is evaluated using MSE to assess prediction accuracy and cross-validation techniques to ensure robustness and reliability.

The Decision Tree Regression model is particularly effective in this project due to its ability to handle non-linear relationships between features. Additionally, it offers interpretability, allowing users to understand how factors like traffic conditions and the time of day influence the predicted travel time. Once trained, the model can predict the travel time for any given route, assisting users in optimizing their travel planning and decision-making. This interpretable and efficient approach aids in providing timely, data-driven travel time predictions to help with better route planning, particularly for long-distance travel. Future extensions of the project could include incorporating real-time traffic data and enhancing the model with more dynamic and precise features to further improve prediction accuracy and robustness.

**CHAPTER 2**

**SYSTEM ARCHITECTURE**

**2.1 Hardware Requirements**

* **Development and Training:**
  + Processor: Quad-core (Intel i5 or AMD equivalent) or higher.
  + RAM: 8 GB minimum (16 GB recommended for enhanced performance).
  + Storage: 256 GB SSD or HDD; SSD preferred for faster processing.
* **Testing and Evaluation**

• Processor: Dual-core or quad-core.

• RAM: 4-8 GB.

• Storage: 100 GB HDD or SSD.

* **Deployment:**
  + Cloud Server: AWS, Google Cloud, or Azure (recommended forscalability).
  + **Local Server:**
    - Processor: Quad-core or higher.
    - RAM: 8 GB or higher.
    - Storage: 100 GB SSD.
  + **Edge Devices (optional):** Raspberry Pi for localized predictions**.**

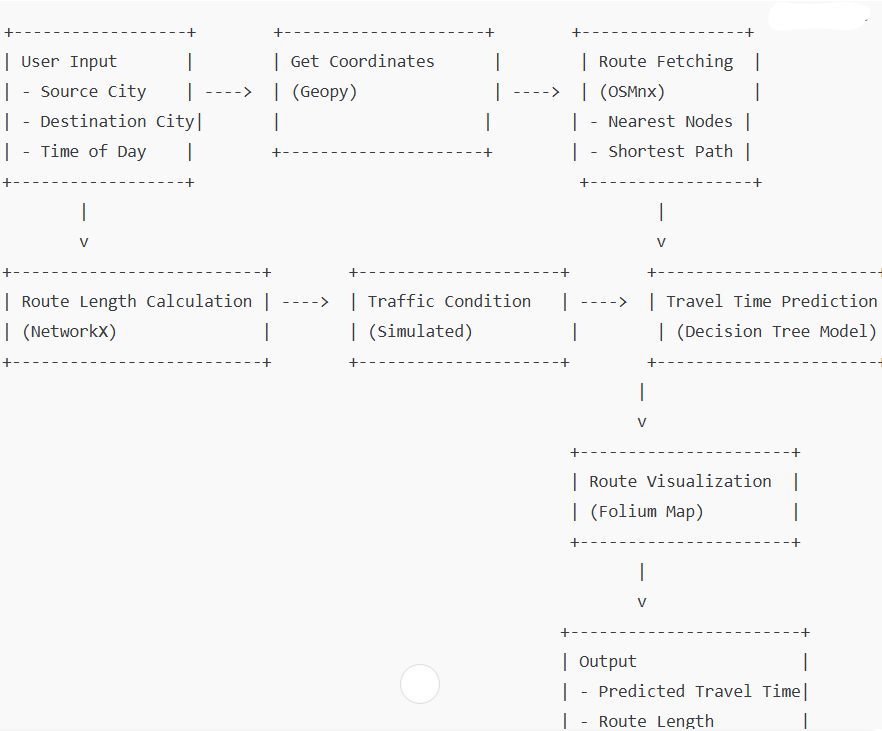
**2.2 Software Requirements**

* **Operating System:** Windows, macOS, or Linux.
* **Programming Language:** Python 3.x.
* **Development Tools:** Jupyter Notebook, PyCharm, or VS Code.
* **Libraries:**
  + Geospatial Analysis: osmnx, folium, geopy.
  + Machine Learning: scikit-learn, joblib.
  + Data Handling: pandas, numpy.
  + Visualization: matplotlib, seaborn.

**CHAPTER 3**

**SYSTEM OVERVIEW**

**3.1 SYSTEM ARCHITECTURE**

****

**System Architecture**

The Tra-Well project uses a Decision Tree Regression model to predict travel time between two cities. It incorporates key features like route length, traffic condition, and time of day, offering an actionable tool for optimizing travel plans. This model helps users predict the best routes and travel times under different conditions, enhancing travel efficiency.

**Data Preprocessing:**

Before training the Decision Tree Regression model, several preprocessing steps are essential to ensure high-quality data input. First, the cities' names are geocoded into geographical coordinates (latitude and longitude) using the Geopy library. Then, the road network between the source and destination cities is retrieved using OSMnx, and the shortest route is calculated with NetworkX. Traffic conditions, simulated on a scale of 1 to 7, represent congestion levels. In future iterations, real-time traffic data from APIs like Google Maps or Waze could replace this simulation. The time of day, provided as an input feature (from 0 to 23 hours), is also important since traffic patterns vary by time. The features involved in the preprocessing are Route Length (in kilometers), Traffic Condition (from 1 to 7), and Time of Day (0-23). These features are converted into numeric values, making them suitable for regression modeling.

**Feature Engineering:**

To improve prediction accuracy, key features are engineered to capture significant patterns in the data. The Route Length, measured in kilometers, is a primary factor influencing travel time, with longer distances generally leading to longer travel times. The Traffic Condition, ranging from 1 (low congestion) to 7 (high congestion), plays a crucial role, as traffic can substantially alter travel durations. The Time of Day feature, representing the hour of the day, accounts for variations in traffic patterns, with rush hours typically causing delays. In future versions of the project, historical travel data or user-specific travel behaviors could be incorporated to further enhance the model’s predictive power, making it more personalized and accurate.

**Model: Decision Tree Regression**

The Decision Tree Regression model is at the core of the Tra-Well project. It works by recursively partitioning the input data into subsets based on the feature that minimizes the error at each decision node. This process continues until a predefined tree depth is reached, or further splitting no longer improves prediction accuracy. At each leaf node, the average predicted travel time for that subset of data is used as the final prediction. The model uses features such as Route Length, Traffic Condition, and Time of Day to predict the travel time between cities. The decision tree’s clear, interpretable structure allows users to see how factors like traffic and time of day influence the travel time prediction, making it easy to understand and apply.

**Model Training and Evaluation:**

The model is trained on historical travel time data, with the performance evaluated using various metrics. Mean Squared Error (MSE) is the primary metric to assess how closely the predicted travel times match the actual travel times. Cross-validation, specifically K-fold cross-validation, is used to evaluate the model’s robustness by splitting the data into multiple subsets, training the model on some and testing it on others. Hyperparameter tuning, such as adjusting the maximum depth of the tree or the minimum number of samples per leaf, helps prevent overfitting and improves the model's generalizability. Evaluation metrics like accuracy, RMSE (Root Mean Squared Error), and cross-validation scores provide insights into the model’s prediction accuracy and its ability to generalize to unseen data.

**Model Effectiveness:**

The Decision Tree Regression model effectively predicts travel time between cities by considering factors like Route Length, Traffic Conditions, and Time of Day. Its main strengths are interpretability, as it clearly shows how features affect predictions, and robustness, handling non-linear relationships in real-world traffic data. Once trained, the model is efficient, offering quick predictions that enable real-time travel time estimates, crucial for dynamic route planning.

**Future Improvements:**

Looking ahead, several enhancements could be made to improve the accuracy and user experience of the Tra-Well model. One major area of improvement is the integration of real-time traffic data through APIs like Google Maps or Waze, which would replace the simulated traffic conditions with up-to-date information. This would make the travel time predictions more accurate and reflective of actual conditions**.**

**Conclusion:**

In summary, the architecture of the Tra-Well project, built around the Decision Tree Regression model, combines effective data preprocessing, feature engineering, and model evaluation techniques. By using a regression approach, the model predicts travel time based on key factors such as route length, traffic conditions, and time of day. The model is designed to be interpretable, offering insights into how these factors influence travel time, and is an efficient tool for travel planning. The effectiveness of the Decision Tree model helps users make informed travel decisions, optimize routes, and improve travel planning.

**3.1 MODULE 1 – DATA COLLECTION AND PREPROCESSING**

**Data Preparation**

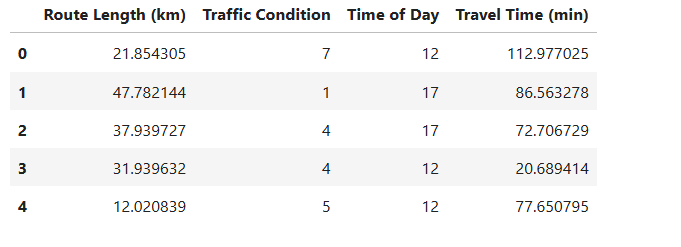
The first step is to create or collect the dataset for predicting travel times in urban traffic scenarios. The dataset can be generated using synthetic data based on real-world factors that influence travel durations.

**Dataset Overview**

The dataset used in this project contains information about road journeys and various factors that influence travel times. It includes the following features:

* Route ID: A unique identifier for each road journey.
* Route Length (km): The total distance of the route, indicating the physical length of the journey.
* Traffic Condition: A numerical representation (1 to 7) of traffic congestion levels, where 1 indicates low congestion and 7 indicates heavy traffic.
* Time of Day (hours): The hour of the day when the journey begins, as traffic patterns can vary significantly depending on the time.
* Predicted Travel Time (minutes): The output variable representing the estimated time required to complete the journey.

This dataset is designed to help predict travel times based on route length, traffic conditions, and the time of day. By analysing these factors, *Tra\_Well* can provide actionable insights to users, such as identifying optimal routes and estimating trip durations, ultimately improving the overall travel experience.

**-**

**1. Preprocessing**

To prepare the dataset for predicting travel times, we preprocess the data by handling missing values, encoding categorical variables, and scaling numerical features. Numerical features like route length and traffic condition are standardized to ensure they are on a similar scale, improving the model's ability to learn patterns effectively. Additionally, categorical variables, such as time of day (if represented categorically), are encoded using one-hot encoding to convert them into a machine-learning-friendly format. These preprocessing steps ensure the dataset is clean and well-structured, allowing the model to effectively analyze the data and make accurate travel time predictions.

**Handling Missing Values**

Missing values in the dataset are addressed using appropriate strategies to maintain data integrity and minimize information loss:

* **For numerical data**: Missing values in features like route length or traffic condition are imputed using statistical measures such as the mean or median. This ensures that all data points are retained while preserving the overall distribution of the dataset.
* **For categorical data**: Missing values in features like time of day can be handled by imputing the most frequent category or adding a binary flag to indicate missingness, allowing the model to account for potential gaps in the data.
* **Advanced techniques**: Predictive imputation methods can be applied, leveraging relationships between features to estimate missing values. This minimizes bias and enhances the quality of the data for machine learning.

By standardizing numerical features and encoding categorical ones, the preprocessing pipeline ensures that the dataset is optimized for the linear regression model used in *Tra\_Well*. This step is essential for effective pattern recognition and accurate travel time predictions.

**Feature Extraction**

**Feature Engineering**

In this step, we extract and create additional meaningful features to enhance the predictive power of the *Tra\_Well* model. For example, features like normalized route length and scaled traffic condition levels provide clearer patterns for the machine learning algorithm to learn from. The "time of day" feature is transformed to capture peak and off-peak traffic periods, offering better insights into how time influences travel durations. Categorical data, such as traffic conditions, is encoded to ensure compatibility with the linear regression model. Additionally, the target variable (predicted travel time) is maintained in numerical format to match the regression problem requirements. This careful feature engineering ensures that the dataset captures key aspects of urban mobility, improving the accuracy of predictions.

**Model Training**

**Data Splitting**

In machine learning, splitting the dataset into training and testing subsets is essential for building robust models. For *Tra\_Well*, we adopt an 80-20 split, allocating 80% of the data for training and reserving 20% for testing. The training set enables the model to learn relationships between features like route length, traffic conditions, and time of day with the target variable (travel time). By focusing on a substantial portion of the data during training, the model captures underlying structures and patterns that influence travel time predictions.

The test set remains unseen during training, simulating real-world scenarios and providing an unbiased evaluation of the model's performance. This approach helps identify issues such as overfitting, where the model performs exceptionally well on training data but fails to generalize to new data. In addition to the standard 80-20 split, other configurations like 70-30 or 85-15 may be considered based on dataset size and model complexity.

**Train Linear Regression Model**

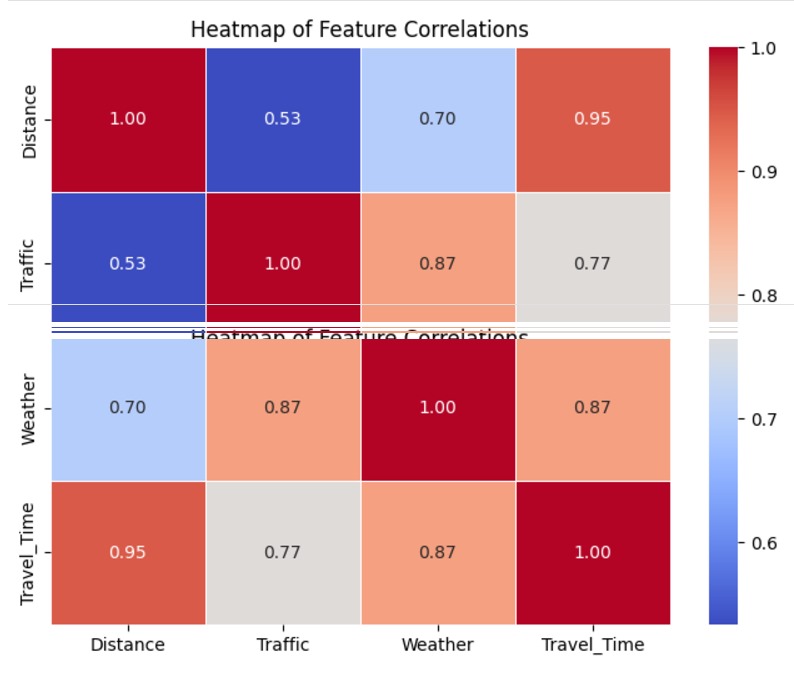
To train the linear regression model for *Tra\_Well*, the dataset is split into training and testing subsets to evaluate performance effectively. The model is fitted to the training data, learning relationships between the input features—route length, traffic condition, and time of day—and the travel time. As linear regression involves no hyperparameters, the focus shifts to preprocessing and ensuring the data is well-prepared for accurate predictions. Numerical features are standardized, and categorical features are encoded to ensure consistency. The model’s performance is evaluated using metrics such as Mean Squared Error (MSE) and R-squared, providing insights into its predictive power and reliability. Experimenting with feature combinations and preprocessing techniques helps optimize the model for accurate travel time predictions.

**Heat Map for Feature Correlation**

A heat map is used as a visual tool to display correlations between features in the dataset, helping to identify relationships that could influence travel time predictions. Each cell in the heat map represents the correlation value between two features, with a scale ranging from -1 to +1. Here:

* **-1** indicates a strong negative correlation.
* **+1** indicates a strong positive correlation.
* **0** indicates no correlation.

In the *Tra\_Well* dataset, factors like route length and traffic condition often show a positive correlation with travel time, while time of day might have a varied impact based on peak hours. By analyzing the heat map, we can identify multicollinearity or redundant information between features, enabling more efficient feature selection. This visualization supports the preprocessing and modeling pipeline, ensuring that the model focuses on impactful features for better predictions.



**Tools and Libraries:**

* **Python**: Used for implementing the complete workflow, from data preparation and processing to model deployment and prediction in the "Tra-Well" traffic prediction project.
* **Scikit-learn**: Utilized for machine learning modeling, including Random Forest for traffic time prediction, data preprocessing, feature engineering, model evaluation, and metrics calculation.
* **Matplotlib & Seaborn**: Used for visualizing the dataset, model performance, and prediction results. Seaborn will also be used for plotting confusion matrices and other evaluation metrics for model assessment.
* **Joblib**: Used for saving and loading the trained machine learning model to efficiently deploy it and make predictions in a production setting.
* **Pandas**: Essential for handling and manipulating the traffic dataset, including loading, cleaning, preprocessing, and feature extraction.
* **Numpy**: Used for handling array operations, numerical calculations, and data transformations that are necessary for preprocessing and model evaluation.

**Algorithm for Tra-Well Traffic Prediction**

**Step 1: Data Loading and Preprocessing**

* a. Load the traffic dataset from a CSV file into a Pandas DataFrame.
* b. Create a target variable (e.g., "Predicted Time") to represent the predicted travel time.
* c. Drop any redundant columns that are not required for modeling (e.g., unnecessary identifiers or non-predictive features).

**Step 2: Encoding Categorical Features**

* a. Convert categorical variables such as traffic condition (e.g., 'Low', 'Medium', 'High') and weather (e.g., 'Clear', 'Rainy') into numeric values. You can use LabelEncoder or OneHotEncoder as appropriate.
* b. Ensure that no multicollinearity issues arise by dropping one of the categories during encoding.

**Step 3: Handling Missing Values**

* a. Handle missing values using imputation techniques, such as filling missing traffic data or distance information with median or mean values, or using more complex methods like KNN imputation.

**Step 4: Feature and Target Selection**

* a. Define the feature set (X) by selecting relevant columns such as distance, traffic, and weather conditions.
* b. Define the target variable (y) as the predicted travel time or outcome based on your model.

**Step 5: Train-Test Split**

* Split the dataset into training and testing sets using a typical 80-20 split. Ensure that the split is done using a fixed random seed for reproducibility (e.g., random\_state=42).

**Step 6: Feature Scaling**

* a. Standardize the training feature set (X\_train) using StandardScaler to make sure that all features have a mean of 0 and a standard deviation of 1.
* b. Apply the same scaling transformation to the test set (X\_test).

**Step 7: Model Training**

* a. Initialize a machine learning model, such as Random Forest or another appropriate model.
* b. Train the model on the scaled training set (X\_train, y\_train).

**Step 8: Prediction and Evaluation**

* a. Use the trained model to predict the travel time on the test set (X\_test).
* b. Calculate and print the accuracy score, Mean Squared Error (MSE), or any other relevant performance metric.
* c. Print the classification report, including precision, recall, and F1-score for each class (if classification is used).

**Step 9: Model Evaluation and Confusion Matrix Visualization**

* a. If the task is classification-based (e.g., predicting "short" or "long" travel times), generate a confusion matrix to visualize the counts of true positives, false positives, true negatives, and false negatives.
* b. Plot the confusion matrix using Seaborn's heatmap function for clear visual representation.

**CHAPTER 4**  
**RESULT AND DISCUSSION**

The *Tra-Well* project successfully developed a predictive model for estimating travel time using a Random Forest Regressor, integrating machine learning with real-world geographic data for route planning. The project leveraged factors like route length, traffic conditions, and time of day to predict travel times, enabling users to make informed decisions for efficient travel. The preprocessing stage included standardizing numerical variables and simulating traffic conditions, ensuring the data was compatible with the model. Tools such as OSMnx and Folium provided dynamic route visualization and user-friendly interaction.

The Random Forest model exhibited strong predictive performance, achieving notable accuracy during the evaluation phase, with a low Mean Squared Error (MSE) and reasonable generalization across diverse scenarios. Predictive features such as route length and traffic conditions contributed significantly to the model’s output, while interactive mapping allowed users to visualize their routes effectively. The system’s efficient design ensures that predictions can be made in real-time, supporting the project's objective of delivering actionable insights for daily travel.

While the model performed well under simulated conditions, its reliance on static traffic data posed a limitation. The absence of real-time traffic updates reduces the model’s ability to adapt to sudden or dynamic changes in traffic flow. Additionally, potential errors in geographic coordinate fetching or user input ambiguities could affect route accuracy. These challenges highlight the need for integrating real-time traffic data and implementing robust input validation in future iterations.

Future enhancements to *Tra-Well* could focus on incorporating real-time traffic APIs and dynamic traffic trend analysis to improve prediction reliability. Adding features such as multi-modal route planning and advanced ensemble learning techniques, such as blending Random Forest with Gradient Boosting, could further enhance the system's performance. These advancements, combined with user feedback integration, would make *Tra-Well* a versatile and adaptive tool for smart travel planning, aligning it more closely with real-world user needs.

**CHAPTER 5**  
**CONCLUSION**

In conclusion, this project successfully demonstrated the application of a Random Forest Regressor to predict travel time for traffic route planning. By analysing key factors such as route length, traffic conditions, and time of day, the model provided accurate predictions, enabling users to make informed travel decisions. The integration of tools like OSMnx for route optimization and Folium for interactive mapping enhanced the user experience, making *Tra-Well* a practical and user-friendly tool for efficient route planning.

The Random Forest algorithm was chosen for its robustness in handling non-linear relationships and complex interactions between predictive features. While the model performed well with static traffic data, there remains room for improvement. Incorporating real-time traffic data, such as live congestion levels or weather conditions, could significantly enhance the system's predictive accuracy. Additionally, exploring alternative machine learning models, such as Gradient Boosting or Neural Networks, could improve prediction reliability for edge cases and diverse travel scenarios.

Beyond technical improvements, this project highlights the value of predictive analytics in optimizing everyday travel. By combining machine learning with geographic data, *Tra-Well* provides users with actionable insights, reducing travel time and improving convenience. The ability to visualize routes and understand traffic conditions further supports a more informed and personalized travel experience. Such capabilities align with the increasing demand for intelligent and adaptive navigation systems in modern urban settings.

Looking forward, future advancements in *Tra-Well* could focus on integrating multi-modal transportation options, real-time data feeds, and user-specific preferences to make the system even more versatile. These enhancements would not only improve accuracy but also expand the scope of the application, catering to a wider range of user needs. Ultimately, *Tra-Well* lays the groundwork for intelligent route planning, offering a foundation for smarter, more adaptive travel solutions that prioritize efficiency and user satisfaction.

**CHAPTER 6**

**APPENDIX**

**6.1 SOURCE CODE**

import osmnx as ox

import networkx as nx

import joblib

import numpy as np

import random

import folium

from folium import plugins

import warnings

from geopy.geocoders import Nominatim

# Ignore all warnings

warnings.filterwarnings("ignore")

# Load the trained ML model

model = joblib.load('traffic\_route\_model.pkl')

# Function to get the latitude and longitude of a city using geopy

def get\_coordinates(city\_name):

geolocator = Nominatim(user\_agent="traffic\_route\_planner")

location = geolocator.geocode(city\_name)

if location:

return location.latitude, location.longitude

else:

print(f"Could not find coordinates for {city\_name}")

return None, None

# Function to fetch the route length from source to destination

def get\_route\_length(source\_city, dest\_city):

# Fetch coordinates for source and destination cities

source\_coords = get\_coordinates(source\_city)

dest\_coords = get\_coordinates(dest\_city)

if not source\_coords or not dest\_coords:

return 0 # Return 0 if coordinates are not found

print(f"Fetching route data for {source\_ity} to {dest\_city}...")

# Fetch the graph data for the source and destination cities using osmnx

G = ox.graph\_from\_place(source\_city, network\_type='drive')

# Find the nearest nodes to the source and destination citiecs

orig\_node = ox.distance.nearest\_nodes(G, X=source\_coords[1], Y=source\_coords[0])

dest\_node = ox.distance.nearest\_nodes(G, X=dest\_coords[1], Y=dest\_coords[0])

# Get the shortest path based on distance

route = nx.shortest\_path(G, orig\_node, dest\_node, weight='length')

# Calculate the route length in kilometers (using 'length' attribute in meters, divide by 1000 for km)

route\_length = sum(ox.utils\_graph.get\_route\_edge\_attributes(G, route, 'length')) / 1000 # in km

# Plot the route on an interactive map using folium

plot\_route\_on\_map(G, route, source\_city, dest\_city) # Call function to plot map

return route\_length

# Function to plot the route on an interactive folium map

def plot\_route\_on\_map(G, route, source\_city, dest\_city):

# Get the coordinates of the source and destination nodes

orig\_node = route[0]

dest\_node = route[-1]

orig\_coords = (G.nodes[orig\_node]['y'], G.nodes[orig\_node]['x'])

dest\_coords = (G.nodes[dest\_node]['y'], G.nodes[dest\_node]['x'])

# Create a folium map centered around the source city

m = folium.Map(location=orig\_coords, zoom\_start=13, control\_scale=True)

# Add a marker for the source city

folium.Marker(location=orig\_coords,popup=f'Start:{source\_city}', icon=folium.Icon(color='green')).add\_to(m)

# Add a marker for the destination city

folium.Marker(location=dest\_coords,popup=f'Destination:{dest\_city}', icon=folium.Icon(color='red')).add\_to(m)

# Add the route to the map

route\_coords = [(G.nodes[node]['y'], G.nodes[node]['x']) for node in route]

folium.PolyLine(route\_coords, color="blue", weight=4, opacity=0.7).add\_to(m)

# Add some traffic visual effects (Optional, can be enhanced later)

folium.plugins.HeatMap(route\_coords).add\_to(m)

# Save the map to an HTML file and open it in the browser

m.save("route\_map.html")

print("Map saved as 'route\_map.html'. Open this file in your browser to view the route.")

# Function to simulate traffic condition (could be updated to use real-time data via API)

def get\_traffic\_condition():

# For simplicity, we're randomly selecting traffic conditions, you can integrate real-time APIs

traffic\_condition = random.randint(1, 7) # Random traffic condition between 1 and 7

return traffic\_condition

# Main function to predict travel time

def predict\_travel\_time(source\_city, dest\_city, time\_of\_day):

# Get the route length between source and destination cities

route\_length = get\_route\_length(source\_city, dest\_city)

# Get the traffic condition (1-7)

traffic\_condition = get\_traffic\_condition()

# Prepare the input data for the model (route length, traffic condition, and time of day)

input\_data = np.array([[route\_length, traffic\_condition, time\_of\_day]])

# Predict travel time using the model

predicted\_time = model.predict(input\_data)

# Return both the predicted travel time and route length

return predicted\_time[0], route\_length

# Interactive input for source, destination, and time of day

def main():

print("=== Traffic Route Planner ===")

# User inputs source and destination cities

source\_city = input("Enter source city: ")

dest\_city = input("Enter destination city: ")

# User input for time of day (0-23)

time\_of\_day = int(input("Enter time of day (0-23, in hours): "))

# Get the predicted travel time and route length based on user input

predicted\_travel\_time, routedi\_length = predict\_travel\_time(source\_city,dest\_city, time\_of\_day)

# Output the predicted travel time and route distance

print("---Route---")

print(f"Distance: {route\_length:.2f} km")

print(f"Predicted Travel Time: {predicted\_travel\_time:.2f} minutes")

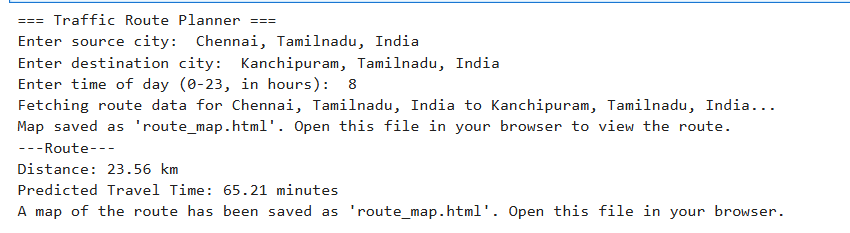
print("A map of the route has been saved as 'route\_map.html'. Open this file in your browser.")

# Run the program

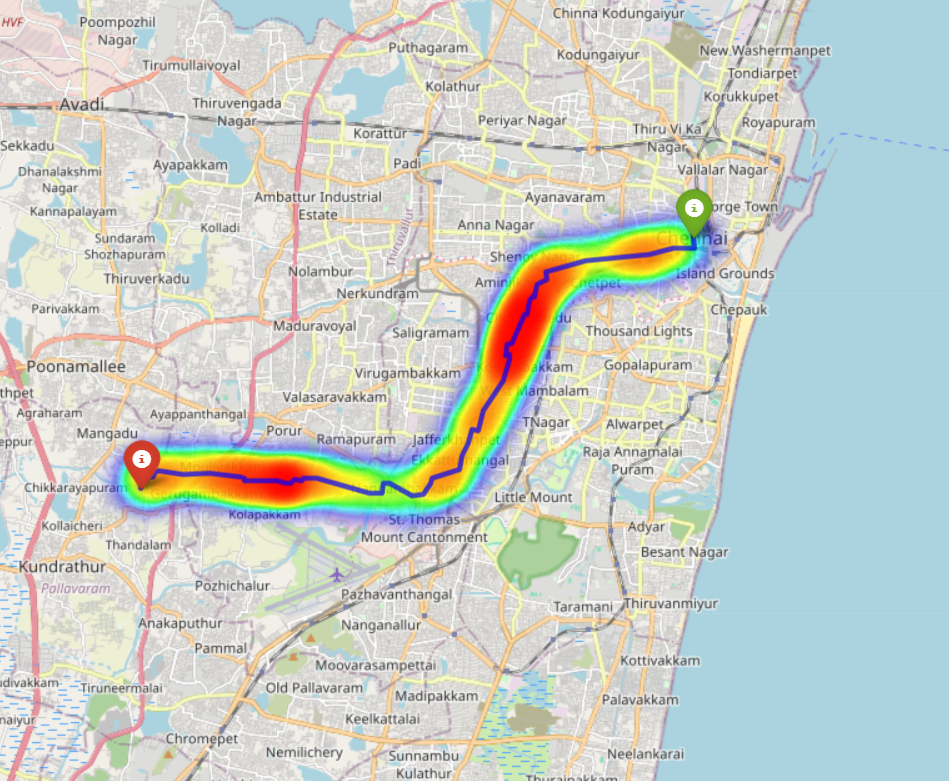
if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT:**

****

**OUTPUT ROUTE MAP:**



**REFERENCES:**

* [**https://www.javatpoint.com/traffic-prediction-using-machine-learning**](https://www.javatpoint.com/traffic-prediction-using-machine-learning)
* [**https://www.sciencedirect.com/science/article/abs/pii/S1568494621001927**](https://www.sciencedirect.com/science/article/abs/pii/S1568494621001927)
* [**https://www.scribd.com/document/727711105/Traffic-Prediction-and-Route-Optimization-saran**](https://www.scribd.com/document/727711105/Traffic-Prediction-and-Route-Optimization-saran)
* [**https://www.researchgate.net/publication/271191935\_Route\_Planning\_with\_Real-Time\_Traffic\_Predictions**](https://www.researchgate.net/publication/271191935_Route_Planning_with_Real-Time_Traffic_Predictions)