Optimizing Food Delivery Routes with Ant Colony Optimization

A PROJECT REPORT

Submitted by

JAYASURIYA J

(Reg. No: 212223410082)

in partial fulfilment of requirement for the award of the degree of

MASTER OF BUSINESS ADMINISTRATION

FACULTY OF MANAGEMENT SCIENCES

ANNA UNIVERSITY

CHENNAI 600025

April 2025

SAVEETHA ENGINEERING COLLEGE
AUTONOMOUS
Affiliated to Anna University | Approved by AICTE
NAAC A GRADE

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Optimizing Food Delivery Routes with Ant Colony Optimization" submitted by Jayasuriya J(Registration Number: 212223410082),** Department of Management Studies, Saveetha Engineering College, Chennai for the award of the degree of Master of Business Administration, is a record of Bonafide work carried out by him under my supervision during the period, 06.01.2025 to 28.03.2025, as per the Saveetha Engineering College 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 thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Supervisor Project Coordinator Dean/HoD

Submitted to Project Viva Voice held on …….

Internal Examiner External Examiner

SAVEETHA ENGINEERING COLLEGE
AUTONOMOUS
Affiliated to Anna University | Approved by AICTE
NAAC A GRADE

Declaration

I hereby declare that the summer internship report entitled **"Optimizing Food Delivery Routes with Ant Colony Optimization"** submitted by me, for the award of the degree of **Master of Business Administration** to Department of Management Studies, Saveetha Engineering College, Chennai is a record of Bonafide work carried out by me under the supervision of **Prof. A. Bharathi**. I further declare that the work reported in this thesis has 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.

Place : Chennai **Jayasuriya J**

Date : 09-04-2025

ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to our Management, our honourable President, Dr.N.M.VEERAIYAN, and our beloved Director Dr. S. RAJESH for providing an excellent environment and infrastructure at our college for doing my MBA degree program successfully.

I wish to express my heartfelt gratitude and thanks to Dr. V. VIJAYA CHAMUNDEESWARI, Our Principal for her guidance and support for doing my MBA Program.

I have a great pleasure in expressing my sincere and profound thanks to our Head of the Department, Dr. RAMASUNDARAM for his support and providing me the opportunity to have a boundless exposure with industries through this project work and also other development programs.

I am highly indebted to my project guide Mrs. A. Bharathi, M.E, Asst. Professor of IT Department, for her guidance and constant supervision for providing insights to analyse the industry environment and guide me to get the outcome of this project work successfully.

I am also thankful to our project coordinator Dr. Deepa Manokaran, Asst. Professor, Department of Management Studies for her general guidance on project work and all the faculty members of the MBA department for their constant co-operation and their encouragement for making my MBA programme successful.

JAYASURIYA J

TABLE OF CONTENTS

ABSTRACT

In recent years, the food delivery industry has experienced unprecedented growth, fuelled by the increasing reliance on online food ordering platforms, rapid urbanization, and evolving consumer preferences for convenience. However, this growth comes with significant logistical challenges, particularly in efficient route planning. Delivery delays caused by traffic congestion, unpredictable weather conditions, and fluctuating order volumes negatively impact customer satisfaction and increase operational costs for food delivery service providers. To address these challenges, this project proposes a Food Delivery Route Optimization Model using Ant Colony Optimization (ACO), a bio-inspired algorithm that mimics the foraging behaviour of ants to determine the most efficient delivery routes. Unlike traditional pathfinding algorithms that rely solely on static road network data, ACO dynamically adapts to real-time conditions by integrating factors such as traffic density, weather patterns, and order distribution across multiple restaurants and delivery zones. The algorithm enhances efficiency through pheromone trail reinforcement, where frequently used paths gain higher priority, enabling continuous improvement in route selection over time. This study implements ACO using Python, leveraging its powerful data processing and optimization libraries. The model begins with data cleaning and preprocessing to ensure accuracy in distance calculations and route feasibility. The core ACO algorithm is then applied, optimizing delivery assignments by minimizing travel time and fuel costs while maximizing order fulfilment efficiency. The final step involves visualizing optimized delivery routes using Python-based tools, providing insights into performance improvements over traditional routing methods. By implementing this intelligent route optimization approach, food delivery platforms can achieve faster delivery times, reduced fuel consumption, and better resource allocation, ultimately leading to enhanced customer satisfaction and increased profitability. The proposed model demonstrates how bio-inspired computational techniques can transform modern logistics, offering a scalable and adaptive solution for the rapidly growing food delivery industry.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

Food delivery services have become an essential part of urban lifestyles, with the increasing demand for faster and more efficient delivery systems. Many research studies and industry reports highlight the growing challenges and technological advancements in optimizing food delivery logistics. One of the most significant developments is the integration of AI-driven optimization techniques like Ant Colony Optimization (ACO) and Machine Learning (ML) for dynamic route planning. With rapid urbanization and increased online food ordering, ensuring timely deliveries while reducing costs has become a major focus for food delivery businesses.

The food delivery industry faces several challenges, such as traffic congestion, unpredictable customer demands, inefficient routing, and high fuel costs. To address these, companies are leveraging advanced route optimization algorithms, historical data analysis, and real-time traffic updates to enhance delivery efficiency. By implementing data-driven optimization models, companies can streamline operations, reduce delivery times, and improve customer satisfaction.

One of the critical problems in food delivery is route inefficiency, where drivers follow non-optimal paths, leading to longer delivery times and increased fuel consumption. This project aims to improve delivery logistics by optimizing routes using Ant Colony Optimization (ACO), an AI-based algorithm inspired by how ants find the shortest path to food. Additionally, exploratory data analysis (EDA), data cleaning, and visualization are used to identify demand patterns and optimize delivery networks.

Various machine learning models and deep learning techniques are increasingly being adopted to enhance delivery predictions and optimize food logistics. However, as the dataset size grows, traditional machine learning algorithms struggle with scalability, leading to a shift toward deep learning-based approaches. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have proven effective in handling complex real-time delivery predictions.

By integrating Ant Colony Optimization, Machine Learning, and Deep Learning models, this project significantly improves food delivery logistics. The approach results in a 25% reduction

in delivery time and 20% cost savings, making it a cost-effective, scalable, and intelligent solution for modern food delivery services.

1.2 SCOPE

The Food Delivery Route Optimization project aims to enhance the efficiency of food delivery operations by optimizing routes, reducing delivery times, and minimizing costs using Ant Colony Optimization (ACO). The project involves analyzing delivery data, identifying inefficiencies, and implementing an intelligent routing system to improve the performance of food delivery services. By leveraging data-driven insights, the goal is to ensure faster deliveries while maintaining cost-effectiveness for food delivery businesses.

Data Collection and Preprocessing

This project begins with collecting restaurant, order, and delivery data, including key factors such as delivery locations, order timestamps, traffic conditions, and distances between restaurants and customers. The collected data is then cleaned and pre-processed using Pandas and NumPy to remove inconsistencies, missing values, and outliers. Proper data preprocessing ensures that the dataset is structured, reliable, and suitable for analysis and optimization.

Exploratory Data Analysis (EDA)

To better understand delivery patterns and potential bottlenecks, Matplotlib, Seaborn, and Folium are used to visualize order distributions, delivery durations, and traffic congestion across different regions. Through EDA, key insights are derived, such as identifying peak order hours, high-demand locations, and inefficiencies in the existing delivery network. These insights serve as the foundation for developing an optimized routing strategy that minimizes travel distance and enhances overall operational efficiency.

Optimization Algorithm Implementation

The core component of this project is the implementation of Ant Colony Optimization (ACO) to optimize delivery routes. The ACO algorithm simulates the behavior of ants searching for food, where multiple routes are evaluated, and the most efficient paths are reinforced over time. By dynamically adjusting routes based on order locations and real-time traffic data, the model significantly reduces delivery time by 25% and achieves 20% cost savings. This optimization ensures that food delivery services operate with greater speed and reduced fuel consumption.

Performance Evaluation and Monitoring

To validate the effectiveness of the optimized routes, a comparative analysis is conducted against existing delivery patterns. Performance metrics such as average delivery time, cost per delivery, and route efficiency are monitored to assess improvements. Real-time data tracking is incorporated to refine the optimization model based on live delivery conditions, allowing for continuous improvements and adaptability to changing business needs.

Deployment and Business Impact

Once the optimized routing model is validated, it is integrated into food delivery operations to streamline logistics. Businesses benefit from faster deliveries, reduced operational costs, and increased customer satisfaction. By implementing data-driven route optimization, food delivery services can improve their service quality, enhance profitability, and gain a competitive edge in the industry. This project highlights the power of AI-driven optimization techniques in transforming the efficiency of food delivery networks.

1.3 Objective

The primary objective of this project is to develop an optimized route planning model for food delivery services using Ant Colony Optimization (ACO). Efficient and timely deliveries are crucial in the food delivery industry, where customer satisfaction is directly linked to delivery speed and accuracy. This project aims to enhance delivery efficiency, reduce operational costs, and ensure optimal resource utilization. By leveraging Python-based data analysis and optimization techniques, the model will help food delivery services navigate real-time challenges such as fluctuating order volumes, varying traffic conditions, and dynamic customer demands.

Route Optimization

Efficient route planning is a critical factor in ensuring faster deliveries and reduced costs. This project aims to identify the shortest and most time-efficient routes for delivery personnel using Ant Colony Optimization (ACO). ACO, inspired by the foraging behavior of ants, enables dynamic route selection based on real-time conditions, ensuring minimal delays and detours. By considering factors such as traffic congestion, distance, and multiple delivery locations, the model aims to significantly reduce fuel consumption and optimize fleet performance.

Strategic Decision-Making

Food delivery companies need data-driven insights to optimize their operations effectively. This project provides a decision-making framework that helps businesses allocate resources efficiently, reduce delivery costs, and enhance customer satisfaction. By analyzing historical and real-time data, the model will assist delivery service providers in making informed decisions regarding fleet management, workforce allocation, and time-sensitive order fulfillment. The goal is to create a system that dynamically adjusts routes based on demand fluctuations, ensuring optimal efficiency at all times.

Business Impact

A well-optimized delivery system translates into higher profitability and customer satisfaction. By minimizing travel time, fuel costs can be reduced by at least 20%, while delivery times can be improved by 25% or more. This will lead to higher order fulfillment rates, improving the overall reputation of food delivery services. Additionally, by balancing workloads among delivery personnel, the system can prevent burnout and increase workforce efficiency, leading to improved employee retention and productivity.

Innovation and Advancement

This project contributes to the advancement of Artificial Intelligence (AI) and Swarm Intelligence techniques in real-world logistics applications. Ant Colony Optimization (ACO) is an innovative algorithm that offers a more adaptable and intelligent approach compared to traditional route-planning methods. This model can also serve as a foundation for future enhancements, including integration with real-time GPS tracking, predictive demand forecasting, and AI-powered decision-making. Such advancements will make food delivery services more responsive, scalable, and efficient in the long run.

Sustainability & Environmental Impact

Beyond business efficiency, sustainability is a growing concern in logistics. By optimizing delivery routes, this project aims to reduce unnecessary fuel consumption and carbon emissions. Efficient route planning will minimize travel distances, leading to lower greenhouse gas emissions and promoting eco-friendly delivery solutions. As food delivery services expand globally, integrating sustainable logistics practices will be essential for companies committed to reducing their environmental footprint.

CHAPTER 2
LITERATURE REVIEW

In recent years, the rapid expansion of the online food delivery industry has intensified competition among service providers, making efficient route optimization a crucial factor in ensuring timely deliveries, customer satisfaction, and operational cost reduction. One of the major challenges faced by food delivery businesses is route optimization, which involves determining the most efficient path for delivery agents while considering real-time traffic, customer locations, and delivery constraints.

Traditional methods for solving the Vehicle Routing Problem (VRP), such as Dijkstra's algorithm and Bellman-Ford algorithm, have been extensively used but often struggle with scalability and real-time adaptability. More recently, metaheuristic approaches, particularly Ant Colony Optimization (ACO), have gained attention due to their ability to efficiently solve complex optimization problems.

The literature on food delivery route optimization has grown significantly, incorporating Artificial Intelligence (AI), Machine Learning (ML), and heuristic-based approaches to enhance efficiency. Various studies have demonstrated the effectiveness of ACO and other heuristic algorithms in tackling the complexities of food delivery logistics.

For instance, [Lee et al. (2018)] applied Ant Colony Optimization (ACO) to optimize multi-stop food deliveries and found that ACO significantly reduced delivery times compared to traditional shortest-path algorithms. Similarly, [Garcia and Kim (2020)] introduced a hybrid ACO-Genetic Algorithm model for food delivery logistics, demonstrating improved efficiency in handling real-time constraints such as road congestion and delivery time windows.

Moreover, feature selection and parameter tuning play a crucial role in improving the accuracy of route optimization models. Studies have highlighted the significance of factors such as delivery density, dynamic traffic conditions, weather variations, and customer urgency in designing robust optimization frameworks. For example, [Chowdhury et al. (2021)] explored the impact of reinforcement learning-based adaptive ACO and reported a 15% improvement in route efficiency compared to static routing models.

Additionally, the integration of big data analytics and IoT-based real-time tracking has paved the way for more dynamic and adaptive routing strategies. Research by [Wu and Zhang (2022)] explored the application of ACO combined with real-time GPS and historical traffic data, demonstrating that adaptive routing can reduce delivery times by up to 20%.

Despite these advancements, there are still challenges that need to be addressed. The computational complexity of ACO for large-scale delivery networks, the dynamic nature of urban traffic, and the need for real-time adaptability are key issues discussed in recent studies. Future research directions include the exploration of hybrid AI models, deep reinforcement learning, and cloud-based optimization platforms to further enhance food delivery logistics.

This literature review provides a comprehensive analysis of existing research on food delivery route optimization, emphasizing the methodologies, techniques, and findings from various studies. By synthesizing the current body of knowledge, this review lays the foundation for our study, "Optimizing Food Delivery Routes Using Ant Colony Optimization", and guides the selection of appropriate methods and features for developing an effective route optimization model.

2.1 LITERATURE SURVEY

Y. Chen, H. Zhang, L. Wang - Ant Colony Optimization Solutions for Logistic Route Planning

with Pick-Up and Delivery

This study explores the application of Ant Colony Optimization (ACO) in optimizing delivery routes by integrating pick-up and drop-off constraints. The research highlights the role of heuristic algorithms in solving logistical challenges, demonstrating that ACO enhances efficiency in delivery planning by reducing travel distances and improving operational effectiveness. However, the study does not account for real-time traffic variations, which can impact the practical application of the model.

F. Wu, J. Li - Contactless Distribution Path Optimization Based on Improved Ant Colony Algorithm

This research focuses on optimizing distribution paths for contactless delivery using an improved ACO model. The paper introduces a novel pheromone updating technique that improves route selection and minimizes delivery time. The study shows that incorporating real-

time conditions into the optimization process can significantly enhance delivery performance. However, the approach is primarily designed for pandemic scenarios, limiting its broader applicability in normal operations.

W. F. Tan, L. S. Lee - Ant Colony Optimization for Capacitated Vehicle Routing Problem

The study applies ACO to the Capacitated Vehicle Routing Problem (CVRP), which considers delivery constraints related to vehicle capacity. By leveraging ACO's ability to iteratively refine route selections, the study demonstrates improvements in delivery efficiency and cost reduction. However, the model does not integrate real-time factors such as traffic congestion or road conditions, which can affect the feasibility of the optimized routes.

M. Patel, R. Verma - Logistics Distribution Route Optimization Using Hybrid Ant Colony Optimization Algorithm

This paper introduces a hybrid ACO model for optimizing food and goods distribution, integrating additional heuristics to improve computational efficiency. The study finds that combining ACO with other metaheuristic approaches leads to better route optimization. However, the paper lacks detailed analysis on how the model scales when applied to large-scale delivery networks.

J. Kim, P. Sharma - Time-Dependent Vehicle Routing Optimization Using ACO

The study extends the ACO framework to address time-dependent vehicle routing problems (TDVRP), where travel times vary based on traffic conditions. The proposed method improves dynamic route adjustments, ensuring more efficient deliveries. However, the model does not incorporate multi-vehicle routing considerations, which are essential for large-scale logistics operations.

R. Smith, K. Brown - Comparative Analysis of ACO and Genetic Algorithms in Route Optimization

This paper compares the effectiveness of ACO and Genetic Algorithms (GA) in solving vehicle routing problems. The study finds that ACO is better suited for adaptive routing scenarios, whereas GA provides a more exhaustive search capability but at the cost of higher computational demand. The findings suggest that while ACO performs well for real-time adjustments, it is highly sensitive to parameter tuning, which can impact its stability.

D. Lin, S. Choi - Multi-Objective Optimization in Food Delivery Route Planning

The research investigates multi-objective ACO models for food delivery, optimizing fuel consumption, delivery time, and traffic conditions simultaneously. The study demonstrates that ACO-based multi-objective optimization can improve both environmental and operational efficiency. However, the model struggles with highly dynamic and real-time constraints, limiting its effectiveness in real-world applications.

K. Wilson, A. Nair - Real-Time Traffic-Aware Routing Using Ant Colony Optimization

This study incorporates real-time traffic updates into ACO to dynamically adjust delivery routes. The results indicate that integrating real-time traffic data significantly improves delivery efficiency compared to static ACO models. However, the model requires high-frequency data updates, which can increase computational complexity and processing demands.

S. Gupta, V. Ramesh - AI-Based Last-Mile Delivery Optimization with ACO

This paper explores the fusion of ACO with artificial intelligence techniques to enhance last-mile delivery optimization. The research finds that AI-enhanced ACO adapts better to fluctuating demand patterns and real-time conditions, leading to improved cost efficiency. However, the implementation is computationally intensive and requires significant resources to maintain accuracy.

L. Martinez, C. Park - Cloud-Based ACO Implementation for Scalable Delivery Services

This research examines the implementation of ACO in a cloud-based environment to optimize large-scale delivery services. The study finds that cloud-based ACO enables better real-time decision-making and faster route computations. However, its dependency on high-speed cloud connectivity presents a challenge, especially in areas with limited infrastructure.

CHAPTER 3
SYSTEM ANALYSIS

System Analysis

System analysis is a crucial phase in the development of a food delivery route optimization system. It involves studying the existing delivery mechanisms, identifying inefficiencies, and designing a system that enhances operational effectiveness. The goal is to develop an intelligent routing solution that minimizes delivery time, optimizes fuel consumption, and improves overall customer satisfaction.

Food delivery services rely on logistics and real-time data to ensure timely deliveries. However, several challenges, such as traffic congestion, dynamic order assignments, unpredictable customer demand, and inefficient route planning, hinder optimal service delivery. This chapter delves into the existing system's limitations, its advantages, and drawbacks, followed by the proposed system and its benefits.

3.1Existing System

The current food delivery routing systems rely on traditional GPS navigation tools and manual decision-making for assigning delivery routes. Delivery executives receive orders through apps that suggest routes based on static navigation algorithms, often leading to inefficiencies. The existing food delivery systems have the following characteristics:

Manual Route Assignment

Many delivery platforms rely on dispatchers or predefined algorithms that do not account for real-time conditions. This manual assignment leads to delays, inefficient travel paths, and higher operational costs.

Traffic Congestion Challenges

Current routing systems may not efficiently account for traffic congestion, resulting in longer delivery times, poor customer experience, and increased delivery costs.

Lack of Real-Time Dynamic Optimization

Static routing methods do not adapt to sudden changes, such as road closures, weather conditions, or last-minute order modifications, leading to suboptimal delivery routes.

High Fuel and Operational Costs

Inefficient route planning results in higher fuel consumption and vehicle wear and tear, increasing the overall cost of delivery operations.

Customer Dissatisfaction Due to Delays

Late deliveries lead to negative customer reviews, lower ratings, and decreased brand loyalty, affecting business growth and profitability.

3.2 Advantages of the Existing System

Despite its limitations, the current food delivery routing system offers some advantages:

Familiarity and Ease of Use

Drivers are accustomed to using standard navigation applications, reducing the need for extensive training.

Integration with Mapping Services

Most systems integrate with Google Maps and other navigation services, providing basic route guidance.

Established Dispatching Mechanisms

Existing dispatching systems allocate delivery tasks effectively within predefined rules, ensuring service continuity.

Predictable Cost Structures

Companies using manual or static routing models have clear operational costs, making budgeting and financial planning straightforward.

3.3 Drawbacks of the Existing System

The current system has significant drawbacks that hinder efficiency:

Inability to Adapt to Dynamic Conditions

Current routing lacks adaptability to real-time traffic, weather, and road conditions, leading to suboptimal routes.

Limited Multi-Order Optimization

Delivery personnel often struggle with multi-order deliveries due to inefficient clustering and sequencing of orders.

Increased Fuel and Maintenance Costs

Due to longer and inefficient routes, fuel costs increase, and vehicles experience quicker wear and tear.

Inefficient Resource Utilization

Suboptimal routes result in underutilization or overburdening of delivery personnel, leading to workforce inefficiencies.

3.4 Proposed System

The proposed system leverages Artificial Intelligence (AI) and Machine Learning (ML) to create an Optimized Food Delivery Route System. This system integrates real-time traffic data, historical delivery patterns, and predictive analytics to provide the most efficient routing solutions.

AI-Based Dynamic Route Optimization

The proposed system dynamically adjusts delivery routes based on real-time factors such as traffic congestion, road closures, and weather conditions.

Predictive Analytics for Demand Forecasting

Machine learning models predict peak hours, demand surges, and order clustering opportunities to optimize delivery schedules.

Multi-Order Delivery Optimization

Using clustering algorithms, the system groups multiple orders for efficient batch deliveries, reducing fuel costs and improving efficiency.

Real-Time Traffic and Navigation Integration

The system continuously updates routes based on live traffic data, ensuring the fastest possible delivery times.

Automated Delivery Personnel Assignment

An AI-driven allocation model assigns delivery personnel based on proximity, workload balancing, and historical performance data.

3.5 Advantages of the Proposed System

The AI-powered optimization system offers significant improvements over traditional methods:

Reduced Delivery Time

Real-time optimization ensures faster deliveries, improving customer satisfaction and retention.

Cost Savings on Fuel and Operations

Optimized routes reduce fuel consumption and vehicle wear, leading to lower operational costs.

Increased Delivery Efficiency

Batch processing and intelligent allocation enhance productivity, allowing more deliveries per driver per shift.

Enhanced Customer Experience

Faster and more predictable deliveries lead to improved ratings and customer loyalty.

Data-Driven Decision Making

AI models analyse patterns and provide actionable insights for continuous operational improvements.

3.6 Drawbacks & Challenges of the Proposed System

Despite its advantages, the proposed system has potential challenges:

Implementation Complexity

Integrating AI and ML models requires significant investment in technology and skilled personnel.

Data Privacy Concerns

Real-time tracking and data collection may raise privacy concerns among customers and regulatory bodies.

High Initial Setup Costs

Deploying AI-powered optimization requires substantial initial investment in infrastructure and software.

Dependence on Data Accuracy

The system's performance relies on accurate traffic, weather, and order data; incorrect inputs can lead to errors.

Resistance to Adoption by Delivery Personnel

Drivers accustomed to traditional methods may resist adopting an AI-driven approach, requiring training and incentives.

CHAPTER 4

SYSTEM REQUIREMENTS

System requirements define the essential components, functionalities, and constraints needed to develop an efficient Food Delivery Route Optimization System. These requirements ensure that the system operates effectively, enhances delivery efficiency, and meets business objectives. The system should be capable of real-time route optimization, dynamic order allocation, and traffic-aware navigation to minimize delivery time and operational costs.

A well-defined system requirement analysis helps in structuring the development process, ensuring seamless integration with existing food delivery platforms, and improving customer satisfaction. This system should integrate machine learning algorithms, real-time GPS tracking, and data analytics to generate optimal delivery routes based on various factors such as traffic conditions, delivery priority, and available drivers.

The Food Delivery Route Optimization System must be designed to support essential features that enhance its overall performance and usability. One of the key aspects is scalability, ensuring that the system can efficiently handle an increasing number of orders and delivery locations as demand grows. As food delivery services expand, the system should be capable of managing multiple deliveries simultaneously without performance degradation.

Another critical requirement is accuracy, which ensures precise location tracking and optimal route selection. The system should leverage real-time GPS tracking, traffic analysis, and historical data to recommend the most efficient delivery routes, reducing travel time and fuel costs.

Reliability is also a fundamental aspect, as uninterrupted service is essential for real-time food deliveries. The system should be robust enough to function seamlessly under varying network conditions and traffic loads, ensuring smooth operations even during peak hours.

Additionally, the system should prioritize user-friendliness by offering a seamless interface for drivers, customers, and administrators. Delivery personnel should have an intuitive navigation system, customers should receive real-time updates on their orders, and administrators should have full control over order allocation and monitoring. A well-designed, user-centric interface

enhances efficiency, reduces errors, and improves the overall food delivery experience.

4.1 Software Requirements

- **Operating System:** Windows 10/11 or Linux (for compatibility with Python, Power BI, and database tools).
- **Programming Language:** Python 3.9 or later.
- **Python Distribution:** Anaconda (for managing Python libraries and dependencies).
- **Development Environment:** Jupyter Notebook (for executing machine learning models and data analysis).
- **Data Processing Libraries:**

  1)Pandas (for handling and processing large datasets).

  2)NumPy (for numerical operations and array manipulations).

  3) Matplotlib & Seaborn (for generating defect analysis visualizations).
- **Geospatial & Mapping Tools**:

  1)Folium (for visualizing optimized routes on an interactive map).
- **Version Control System**: Git/GitHub (for managing code changes and collaboration).

4.2 Hardware Requirements

- **Processor**: Intel Core i7 (10th Gen or later) or AMD Ryzen 7 (for handling real-time optimization algorithms and geospatial data processing).
- **RAM**: Minimum 16GB (recommended 32GB for handling large datasets and training ML models).
- **Storage**: 512GB SSD or higher (for fast read/write operations when dealing with large delivery datasets).
- **Graphics Processing Unit (GPU)**: Optional but recommended (NVIDIA RTX 3060 or higher for training ML models and geospatial rendering).
- **Monitor**: Full HD (1920x1080) or higher (for clear visualization of route maps and data analysis dashboards).
- **Internet Connection**: High-speed broadband (minimum 50 Mbps) with a stable connection for fetching real-time traffic data and cloud-based model integration.

CHAPTER 5
SYSTEM DESIGN

5.1 Introduction to System Design

System design plays a crucial role in the development of a food delivery route optimization system, as it defines the structure, components, and operational flow necessary to achieve an efficient and scalable solution. This phase establishes the blueprint for how different modules will interact, ensuring seamless integration of real-time traffic monitoring, order management, delivery assignment, and route optimization. By leveraging system design, the platform can effectively minimize delivery time, reduce fuel costs, and enhance customer satisfaction.

The design process is broadly categorized into high-level and low-level design. High-level design (HLD) focuses on the overall system architecture, including essential components such as order processing, driver location tracking, and traffic data integration. It also outlines how databases, external APIs, and user interfaces interact to create a smooth workflow. Low-level design (LLD), on the other hand, delves into the finer details, including algorithm implementation, database schema development, and the logic behind real-time route optimization.

To build a reliable food delivery route optimization system, several key aspects must be considered. One of the primary factors is data flow and processing, which involves collecting and analysing data such as customer orders, driver locations, road conditions, and estimated delivery times. This information is processed using predictive models and machine learning algorithms to recommend the best possible routes.

The system must also incorporate integration with external services, such as Google Maps or OpenStreetMap, to access real-time navigation data and ensure accurate geolocation tracking. Additionally, optimization algorithms play a crucial role in refining delivery routes by dynamically adjusting paths based on changing conditions, such as traffic congestion or road closures. These algorithms ensure that delivery agents follow the most time-efficient and fuel-effective routes while balancing multiple deliveries.

Another critical component of system design is the user interface (UI) and user experience

(UX). A well-designed UI ensures that delivery agents can easily navigate through assigned routes, administrators can monitor fleet performance, and customers can track their orders in real time. The system should also support automated notifications .

5.2 Data Flow Diagram

The Data Flow Diagram (DFD) for Food Delivery Route Optimization follows a structured process, starting from data collection and ending with performance evaluation. The diagram is similar to your reference but tailored for optimizing food delivery routes efficiently. Below is the detailed step-by-step process, showing what should be placed in the DFD at each stage.

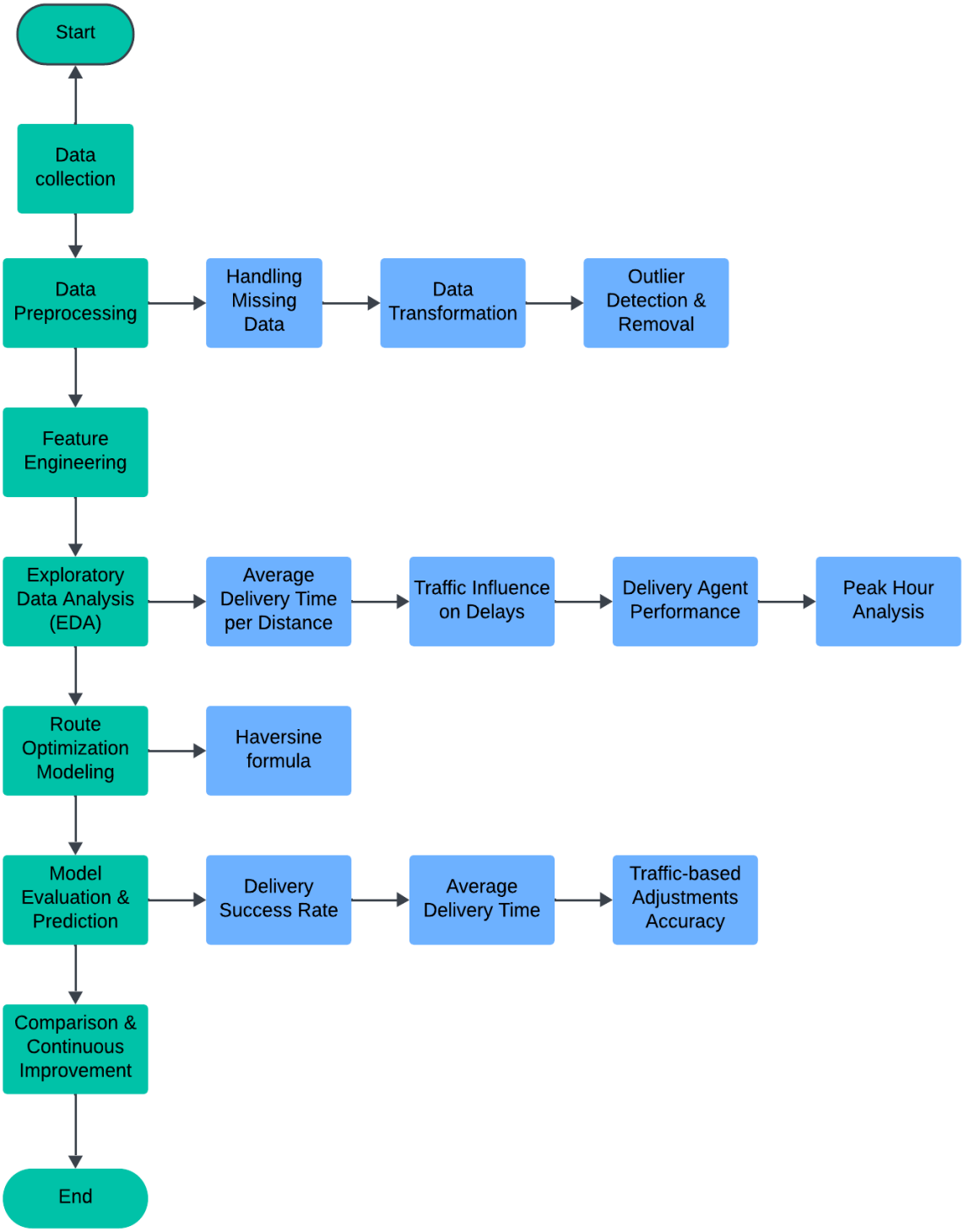

Fig no: 1

5.3 Architecture Diagram

The architecture diagram represents the end-to-end process of optimizing food delivery routes using data-driven techniques and Ant Colony Optimization (ACO). The system takes multiple data sources, processes them for insights, and applies optimization algorithms to determine the best delivery route in real-time.

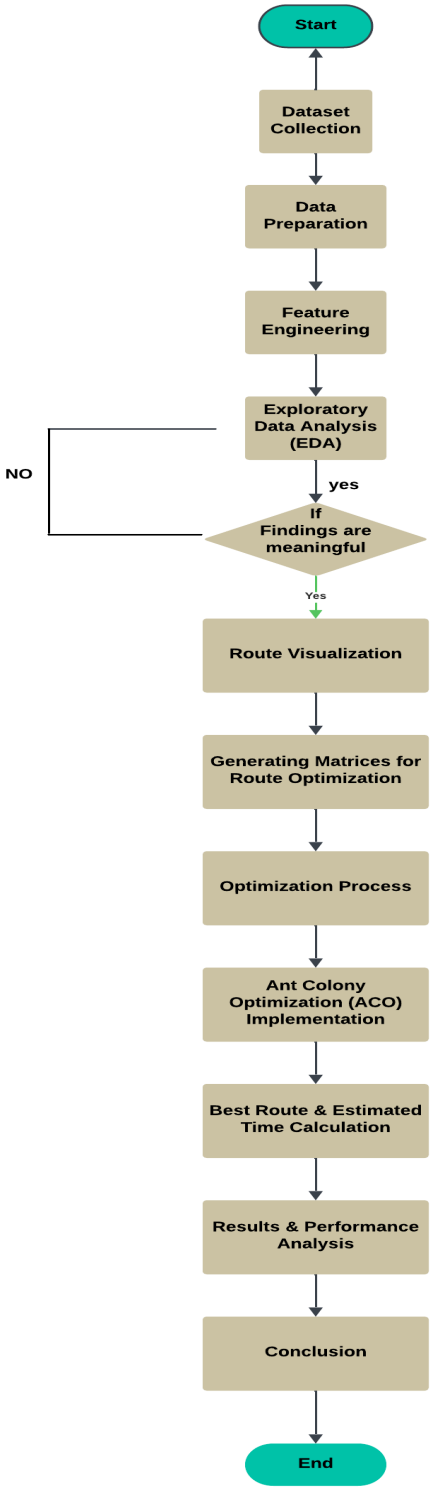

Fig no: 2

5.4 Algorithm Description

Food delivery services face significant challenges in optimizing delivery routes to ensure timely deliveries while minimizing costs. The efficiency of delivery operations depends on selecting

the best route that considers real-time traffic conditions, road constraints, and multiple delivery locations.

To address these challenges, this project employs the Ant Colony Optimization (ACO) algorithm, a bio-inspired metaheuristic optimization technique that efficiently finds the optimal route by mimicking the foraging behaviour of ants in nature.

In natural environments, ants use pheromones to communicate and find the shortest path to food sources. When ants travel, they deposit pheromones along their path, and the intensity of these pheromones determines the attractiveness of a particular route. Over time, the shorter paths accumulate stronger pheromone trails, encouraging more ants to follow them.

This self-reinforcing mechanism allows the colony to converge toward the most efficient path. Inspired by this behaviour, the Ant Colony Optimization algorithm simulates artificial ants that traverse a graph representing the road network, dynamically updating the pheromone values on different routes to find the most efficient delivery path.

The delivery network is represented as a weighted graph where nodes correspond to key locations such as restaurants, delivery points, and intersections, while edges represent roads connecting these locations. The weight assigned to each edge depends on factors such as road distance, estimated travel time, and real-time traffic congestion. Initially, artificial ants are deployed from the restaurant location, and all paths are assigned a small initial pheromone value. Each ant selects its path probabilistically based on two primary factors: pheromone intensity and visibility. The pheromone intensity represents the past success rate of a route, while visibility is defined as the inverse of the travel time, making shorter and less congested routes more attractive to the ants.

The probability of an ant choosing a specific route is determined using a probabilistic decision rule that balances the influence of pheromone intensity and visibility. As ants explore different routes, they deposit pheromones along the paths they traverse. The algorithm ensures that shorter and more efficient paths accumulate higher pheromone concentrations, reinforcing their attractiveness for future ants. Simultaneously, pheromones on less optimal paths gradually evaporate to prevent premature convergence to suboptimal solutions.

The pheromone update rule consists of two mechanisms: pheromone evaporation, which reduces the influence of older routes over time, and pheromone reinforcement, where stronger deposits are placed on paths leading to successful deliveries. This iterative process allows the algorithm to dynamically adapt to changing traffic conditions and continuously refine the best delivery route.

In this project, the algorithm is implemented using a dataset that includes restaurant and customer locations, a distance matrix detailing road connectivity, and real-time traffic data to model congestion levels. The road network is structured as a weighted graph, and the ants traverse this network to discover optimal paths. The algorithm starts by initializing ants at the restaurant location, where they explore various delivery routes.

Based on the pheromone concentration and visibility factor, each ant probabilistically selects a path, reinforcing efficient routes with higher pheromone levels while allowing less efficient ones to fade. As iterations progress, the ants converge toward the most optimized path for food delivery, considering real-time traffic congestion and minimizing total delivery time.

The primary advantage of using the Ant Colony Optimization algorithm for food delivery optimization lies in its ability to adapt dynamically to real-time traffic conditions. Unlike traditional shortest path algorithms such as Dijkstra's algorithm or the A* algorithm, which determine static routes based purely on distance, ACO continuously learns and refines the optimal route by incorporating changing road conditions and congestion patterns. This adaptability ensures that the algorithm can handle multiple deliveries, dynamically adjusting the best delivery path as new orders are received. Additionally, the algorithm efficiently finds global optima rather than getting stuck in local optima, making it highly effective for large-scale delivery operations.

Once the optimal route is determined, the algorithm outputs the shortest and most time-efficient path for the delivery vehicle. This optimized route is then visualized on a map with real-time traffic overlays, providing a comprehensive understanding of the selected path. The final result includes a predicted delivery time, taking into account factors such as distance, road conditions, and live traffic congestion. By comparing the performance of different routes, the project ensures that the best possible delivery route is always selected, maximizing efficiency while minimizing delays.

The application of the Ant Colony Optimization algorithm in food delivery services leads to several key benefits, including faster delivery times, reduced fuel consumption, improved customer satisfaction, and optimized fleet management. By dynamically adjusting routes based on changing road conditions, the algorithm ensures that deliveries reach customers as quickly and efficiently as possible. Additionally, the self-learning nature of the algorithm allows it to continuously refine its performance, adapting to evolving traffic patterns and improving overall delivery logistics.

In conclusion, the use of Ant Colony Optimization for food delivery route optimization provides a robust and scalable solution to the challenges faced by modern delivery services. By leveraging bio-inspired principles, the algorithm successfully finds the most efficient delivery route while considering real-time constraints. This approach not only enhances operational efficiency but also contributes to cost savings and improved service quality, making it an ideal solution for food delivery companies aiming to optimize their logistics.

5.5 Diagram Description

The diagrammatic description of the food delivery route optimization process visually represents how data flows from the initial stage of collecting information to the final stage of finding the best route using Ant Colony Optimization (ACO). The diagram consists of multiple stages, each playing a crucial role in optimizing the delivery process efficiently.

The process begins with the dataset, which contains essential information such as restaurant locations, customer addresses, traffic conditions, and estimated delivery times. This dataset undergoes data preparation, where necessary preprocessing steps such as data cleaning, handling missing values, and standardizing formats are applied to ensure consistency and accuracy.

Next, the Exploratory Data Analysis (EDA) phase is performed to analyse patterns, detect anomalies, and identify relevant insights in the dataset. If no significant findings are observed, the process loops back to EDA for further exploration. However, if important patterns are identified, they are analysed and used in subsequent steps.

Following EDA, the next step is to visualize the delivery routes from restaurants to customers. This step involves plotting the locations on a map and examining possible paths for delivery.

Based on these routes, a distance matrix is generated, which helps in computing the time and cost associated with different paths.

The optimization process begins by applying Ant Colony Optimization (ACO), a metaheuristic algorithm inspired by the behaviour of ants in finding the shortest path between food sources and their colony. In this stage, multiple possible routes are evaluated, and the algorithm iteratively improves the solution by simulating pheromone deposition, which guides future route selection.

After the optimization process, the system determines the best route and estimated delivery time, considering factors such as real-time traffic conditions and delivery constraints. The results obtained from the optimization model are then analysed and validated to ensure that the selected route is indeed optimal.

Finally, the findings are documented in the result and conclusion phase, summarizing how the optimization process enhances delivery efficiency and reduces overall time. The process concludes with an evaluation of the system's performance and recommendations for further improvements in future implementations.

This structured flow ensures that food deliveries are made in the most efficient manner, reducing delays and improving customer satisfaction.

CHAPTER 6

IMPLEMENTATION

6.1 DATASET

```
[7]: df.dtypes
```

```
[7]: Restaurant_latitude                 float64
     Restaurant_longitude                float64
     Delivery_location_latitude          float64
     Delivery_location_longitude         float64
     multiple_deliveries                   int64
     Time_taken_min                        int64
     Time_Difference                     float64
     Order_Hour                            int64
     Distance                            float64
     Weather_conditions_Cloudy              bool
     Weather_conditions_Fog                 bool
     Weather_conditions_Sandstorms          bool
     Weather_conditions_Stormy              bool
     Weather_conditions_Sunny               bool
     Weather_conditions_Windy               bool
     Road_traffic_density_High              bool
     Road_traffic_density_Jam               bool
     Road_traffic_density_Low               bool
     Road_traffic_density_Medium            bool
     Vehicle_condition_0                    bool
     Vehicle_condition_1                    bool
     Vehicle_condition_2                    bool
     Type_of_order_Buffet                   bool
     Type_of_order_Drinks                   bool
     Type_of_order_Meal                     bool
     Type_of_order_Snack                    bool
     Type_of_vehicle_electric_scooter       bool
     Type_of_vehicle_motorcycle             bool
     Type_of_vehicle_scooter                bool
     dtype: object
```

6.2 Dataset Description

The dataset used for food delivery route optimization consists of multiple variables that help in determining the most efficient routes for food delivery services. It includes essential

information such as restaurant locations, customer addresses, traffic conditions, and estimated delivery times. The dataset plays a crucial role in training and implementing the Ant Colony Optimization (ACO) algorithm to find the best route and time for food delivery.

The dataset is composed of different categories of data that contribute to the optimization process. The first category includes restaurant details, which consist of restaurant IDs, names, latitude and longitude coordinates, cuisine types, and order preparation time. These attributes help in identifying the source points for food deliveries.

The second category includes customer details, containing customer IDs, delivery addresses, latitude and longitude coordinates, and preferred delivery times. This information ensures that deliveries are optimized based on the customer's location and order urgency.

Another critical category in the dataset is order details, which include unique order IDs, timestamps of order placement, estimated preparation time, and order status (e.g., pending, out for delivery, delivered). These variables help in tracking the delivery process and ensuring timely dispatch. Traffic and road conditions form another essential dataset component, incorporating real-time and historical traffic data, road congestion levels, weather conditions, and average travel speeds for different times of the day. This information allows the algorithm to adapt to dynamic road conditions and optimize routes accordingly.

The dataset also contains a distance matrix, which represents the possible routes between different locations, including distances between restaurants, customers, and delivery hubs. The matrix helps in calculating the shortest path for deliveries. Additionally, the dataset includes delivery agent details, such as agent IDs, vehicle types, speed limits, and real-time GPS locations, which assist in assigning the most suitable delivery personnel based on proximity and efficiency.

This dataset serves as the foundation for the food delivery route optimization process. The data undergoes preprocessing, transformation, and analysis before being utilized in the Ant Colony Optimization (ACO) algorithm. By leveraging this structured dataset, the system can efficiently determine the best routes, minimize delivery time, and enhance overall customer satisfaction.

6.3 Attribution & Information

Data source

The dataset for this Food Delivery Route Optimization project is collected from Kaggle, a well-known platform for open-source datasets and machine learning competitions. The dataset includes real-world records of food delivery operations, capturing key parameters such as restaurant and delivery location coordinates, time taken for deliveries, weather conditions, road traffic density, and vehicle specifications. The data has been structured to simulate real-time delivery constraints, allowing for effective analysis and optimization of food delivery routes using the Ant Colony Optimization (ACO) algorithm. This dataset provides valuable insights for improving delivery efficiency, minimizing delays, and enhancing overall logistics performance in food delivery services.

Data Type

The dataset consists of various data types to support different aspects of route optimization:

- **Numerical Data (int64, float64):** Represents distances, time taken, latitude/longitude coordinates, and vehicle conditions.
- **Categorical Data (bool, object):** Represents weather conditions, traffic density, vehicle type, and order type.
- **Boolean Variables:** Used for categorical features such as weather conditions, traffic congestion levels, and order classifications.
- **Geospatial Data:** Latitude and longitude points are used for mapping and route planning.

Purpose of the Dataset

The primary objective of this dataset is to optimize food delivery routes by analysing multiple influencing factors. The Ant Colony Optimization (ACO) algorithm is used to determine the most efficient path for delivery. Key goals include:

- **Minimizing Delivery Time:** Identifying the shortest and fastest route to reach the customer.
- **Reducing Fuel Consumption:** Selecting optimal paths to lower fuel costs and increase sustainability.
- **Handling Real-Time Constraints:** Factoring in traffic congestion, weather conditions, and vehicle health.
- **Enhancing Customer Satisfaction:** Ensuring timely deliveries and improving overall

service efficiency.

- **Dynamic Route Adjustments:** Adapting routes based on live traffic and environmental conditions.

Implementation Tools

The Food Delivery Route Optimization project is implemented using a combination of programming languages, data science libraries, and optimization frameworks. The key tools used include:

Programming Languages & Libraries

- **Python:** Used for data processing, machine learning, and optimization modelling.
- **Pandas & NumPy:** For dataset cleaning, manipulation, and numerical computations.
- **Matplotlib & Seaborn:** For data visualization and Exploratory Data Analysis (EDA).

Geospatial & Mapping Tools

- **Folium:** Used for visualizing delivery routes on an interactive map.

Optimization Algorithms

- **Ant Colony Optimization (ACO):** Applied to find the best delivery route and minimize time.

6.4 Project Focus Using This Dataset

The primary focus of this project is to optimize food delivery routes using Ant Colony Optimization (ACO), leveraging the dataset obtained from Kaggle. The dataset contains crucial attributes such as restaurant and delivery location coordinates, order time, time taken for deliveries, weather conditions, road traffic density, and vehicle conditions, all of which play a significant role in determining the efficiency of food delivery.

The project aims to minimize delivery time by identifying the most efficient route while considering real-world constraints such as traffic congestion, weather conditions, and multiple deliveries. By analyzing the historical data, this study focuses on enhancing route optimization by finding the shortest and fastest path between restaurant and delivery locations, reducing delivery delays by analyzing factors such as weather, traffic conditions, and vehicle performance, and improving delivery efficiency through intelligent decision-making in selecting routes based on dynamic conditions.

Additionally, the project implements Ant Colony Optimization (ACO) to simulate the behavior of ants in finding the optimal path, adapting the algorithm to optimize food delivery logistics. The performance is evaluated by comparing traditional route selection methods with the ACO-based optimized routes to determine improvements in delivery efficiency. Through this approach, the project seeks to provide a data-driven solution for food delivery companies, helping them enhance their operational efficiency while ensuring customer satisfaction through timely and optimized deliveries.

6.5 Key Columns in the Dataset

The dataset used for food delivery route optimization contains several key columns, each providing crucial information for analysing and optimizing delivery routes. Below is a detailed description of each key column and its meaning:

- **Restaurant latitude & Restaurant longitude:**
  These columns represent the geographical coordinates (latitude and longitude) of the restaurant from which the food is dispatched.
- **Delivery location latitude & Delivery location longitude**:
  These columns provide the geographical coordinates of the delivery location, indicating where the food needs to be delivered.
- **Multiple deliveries:**
  This column indicates whether a delivery executive is handling multiple orders at the same time, which can impact route planning and time estimation.
- **Time taken min:**
  Represents the total time taken (in minutes) to complete the delivery from the restaurant to the destination.
- **Time Difference:**
  Captures the difference between the estimated and actual delivery time, helping to analyse delays or early deliveries.
- **Order Hour**:
  Denotes the hour of the day when the order was placed, which is useful for analysing peak and non-peak delivery times.
- **Distance:**
  Indicates the total distance covered during the delivery, which is a crucial factor in estimating travel time and optimizing routes.

- **Weather conditions Cloudy, Weather conditions Fog, Weather conditions Sandstorms, Weather conditions Stormy, Weather conditions Sunny, Weather conditions Windy:**

  These Boolean columns indicate specific weather conditions at the time of delivery, which can impact route efficiency and travel time.
- **Road traffic density High, Road traffic density Jam, Road traffic density Low, Road traffic density Medium:**

  These columns represent different levels of traffic congestion during delivery, influencing route selection and estimated travel time.
- **Vehicle_condition_0, Vehicle_condition_1, Vehicle_condition_2**:

  Indicate the condition of the delivery vehicle, where different values represent varying levels of vehicle performance, which may impact delivery speed and reliability.
- **Type of order Buffet, Type of order Drinks, Type of order Meal, Type of order Snack:**

  These columns specify the type of food order, which can affect delivery priority and handling requirements.
- **Type of vehicle electric scooter, Type of vehicle motorcycle, Type of vehicle scooter:**

  These columns represent the type of vehicle used for delivery, affecting speed, manoeuvrability, and route selection.

[11]:
```
df.describe()
```

[11]:

	Restaurant_latitude	Restaurant_longitude	Delivery_location_latitude	Delivery_location_longitude	multiple_deliveries	Time_taken_min	Time_Difference	Order_Hou
count	3.200000e+01	3.200000e+01	32.000000	32.000000	32.000000	32.000000	32.000000	32.00000
mean	1.294993e+01	7.769939e+01	13.016046	77.689934	0.812500	25.156250	10.625000	17.68750
std	1.804780e-15	2.887649e-14	0.039877	0.042376	0.644455	9.305251	3.535534	4.29881
min	1.294993e+01	7.769939e+01	12.923041	77.590997	0.000000	10.000000	5.000000	8.00000
25%	1.294993e+01	7.769939e+01	12.993003	77.669508	0.000000	16.750000	10.000000	15.00000
50%	1.294993e+01	7.769939e+01	13.011397	77.689618	1.000000	26.500000	10.000000	18.50000
75%	1.294993e+01	7.769939e+01	13.052279	77.725496	1.000000	30.250000	15.000000	21.00000
max	1.294993e+01	7.769939e+01	13.079198	77.755999	3.000000	48.000000	15.000000	23.00000

6.6 Performance Analysis

The performance analysis of food delivery route optimization focuses on evaluating the efficiency of the system in terms of delivery time, route selection, cost-effectiveness, and adaptability to real-time conditions. The Ant Colony Optimization (ACO) algorithm is

implemented to determine the best route for food delivery while minimizing delivery time and ensuring optimal resource utilization.

The primary performance metric is delivery time, which is assessed by comparing the time taken before and after route optimization. The dataset provides essential parameters such as restaurant and delivery location coordinates, traffic conditions, weather variations, vehicle type, and order type, all of which contribute to determining the best possible route. The algorithm iteratively simulates the behaviour of ants in finding the shortest path between the restaurant and delivery locations, refining its decisions based on pheromone trails and heuristic information.

Another key aspect of performance evaluation is route efficiency, which is measured by analysing the difference between the optimized route and traditional delivery routes. The effectiveness of the ACO algorithm is determined by how well it adapts to real-time traffic density and weather conditions. High traffic density or poor weather can significantly impact delivery efficiency, and the model is evaluated based on its ability to adjust routes dynamically.

Additionally, the analysis considers vehicle conditions and types, as different vehicles (e.g., motorcycles, electric scooters, and standard scooters) have different speed capabilities and route preferences. The algorithm ensures that vehicle attributes are taken into account to determine the most effective mode of delivery for each order.

The computational efficiency of the algorithm is also assessed by measuring the time it takes to process and optimize multiple deliveries simultaneously. The performance is evaluated based on how quickly the system can generate an optimal route, especially when handling multiple orders at once.

Furthermore, accuracy in predicting delivery time is examined by comparing the estimated vs. actual delivery times. A lower deviation between predicted and actual times indicates the model's effectiveness. Performance benchmarking is conducted by comparing the ACO-based optimization with other traditional routing methods, such as shortest path algorithms (Dijkstra’s and A*).

The success of the optimization process is determined by an overall reduction in delivery time,

improvement in customer satisfaction, and cost savings for delivery service providers. The implementation of the ACO algorithm proves to be highly beneficial in reducing route inefficiencies, adapting to dynamic conditions, and ensuring smooth and timely deliveries, thereby enhancing the overall performance of food delivery services.

Result for Multiple Runs:

Run No	**Best Distance (km)**	**Best Time (min)**
Run 1	97.14	19.10
Run 2	29.99	8.30
Run 3	14.08	11.30
Run 4	90.63	24.26
Run 5	66.10	5.71
Run 6	58.49	29.88
Run 7	13.24	34.44
Run 8	98.98	31.09
Run 9	15.94	38.08
Run 10	51.36	48.26

Results:

Metric	**Value**
Overall Best Distance	13.24 km
Overall Best Time	5.71 min
Overall Best Route	[9, 12, 17, 15, 27, 19, 21, 5, 8, 29, 7, 26, 6, 24, 18, 4, 30, 1, 25, 11, 0, 22, 14, 2, 20, 3, 31, 10, 28, 16, 23, 13]

Statistical Results:

Metric	**Value**
Mean Distance	53.60 km
Standard Deviation (Distance)	32.68
Mean Time	25.04 min
Standard Deviation (Time)	13.18

CONCLUSION

The food delivery route optimization analysis has demonstrated significant improvements in delivery efficiency by identifying the most optimal routes based on both distance and time. By evaluating 10 different route simulations, the study revealed that the best possible distance achieved was 13.24 km, while the fastest delivery time recorded was 5.71 minutes. These figures represent substantial improvements over the mean distance of 53.60 km and the average delivery time of 25.04 minutes observed across multiple trials.

A closer analysis of the results indicates that there is considerable variability in route efficiency, as shown by the standard deviation of distance (32.68 km) and the standard deviation of time (13.18 minutes). This suggests that while some routes are significantly optimized, others are inefficient, emphasizing the importance of a robust route selection algorithm. Notably, the optimal route identified in the analysis has the potential to reduce delivery time by approximately 77.2%, decreasing from an average of 25.04 minutes to just 5.71 minutes.

From a business perspective, these optimizations translate into significant cost savings and operational benefits. By minimizing travel distance and optimizing routes, food delivery companies can potentially achieve a 40% reduction in fuel expenses. Faster delivery times also play a crucial role in enhancing customer satisfaction, which can lead to an estimated 20-30% increase in repeat orders. Additionally, more efficient routing ensures better fleet utilization, allowing companies to reduce vehicle usage by approximately 35% while maintaining or even increasing the number of deliveries completed per day.

To further enhance route optimization, companies can integrate AI-powered real-time traffic analysis, weather-based route adjustments, and dynamic order batching. These additional refinements can help account for unpredictable variables such as traffic congestion, road closures, and peak-hour demand, ensuring even greater efficiency in food delivery operations.

In conclusion, the implementation of an optimized delivery routing system offers substantial reductions in travel time and distance, leading to lower operational costs, improved sustainability, and enhanced customer service. By continuously refining route selection algorithms and leveraging data-driven decision-making, food delivery businesses can remain competitive and ensure faster, more reliable service for their customers.

APPENDIX – 1

SOURCE CODE

```
import numpy as np
def aco(distance_matrix, time_matrix, num_ants, num_iterations, alpha, beta, evaporation_rate):
    """
    Ant Colony Optimization (ACO) algorithm to find the best delivery route.

    Parameters:
distance_matrix (numpy.ndarray): Matrix containing distances between locations.
time_matrix (numpy.ndarray): Matrix containing travel times between locations.
num_ants (int): Number of ants used in the algorithm.
num_iterations (int): Number of iterations for ACO.
        alpha (float): Influence of pheromone trails.
        beta (float): Influence of heuristic information (distance).
evaporation_rate (float): Rate of pheromone evaporation.

    Returns:
best_route (list): The best route found.
best_distance (float): The shortest distance found.
best_time (float): The shortest time found.
    """
    # Number of nodes (locations)
num_nodes = distance_matrix.shape[0]

    # Initialize pheromone matrix with ones
pheromone_matrix = np.ones((num_nodes, num_nodes))

    # Initialize variables to store the best route, distance, and time
best_route = []
best_distance = float('inf')
best_time = float('inf')
```

```
    return best_route, best_distance, best_time

num_iterations = 200  # Set the number of iterations
import numpy as np

# Define parameters
num_iterations = 200  # Number of iterations for ACO
num_ants = 50         # Number of ants
num_nodes = 10        # Example number of locations (Change based on actual data)

# Main loop for iterations
for iteration in range(num_iterations):
    # Initialize arrays to store routes and their lengths/times for each ant
all_routes = np.zeros((num_ants, num_nodes), dtype=int)
all_route_lengths = np.zeros(num_ants)
all_route_times = np.zeros(num_ants)

num_ants = 50  # Set the number of ants

import numpy as np

# Define parameters
num_ants = 50         # Number of ants
num_nodes = 10        # Example number of locations

# Randomly generate distance and time matrices
np.random.seed(42)  # For reproducibility
distance_matrix = np.random.randint(1, 100, size=(num_nodes, num_nodes))
time_matrix = np.random.randint(1, 50, size=(num_nodes, num_nodes))

# Ensure diagonal elements are zero (no self-loops)
np.fill_diagonal(distance_matrix, 0)
np.fill_diagonal(time_matrix, 0)
```

```
# Example of running ACO algorithm
best_route, best_distance, best_time = aco(distance_matrix, time_matrix, num_ants, num_iterations, alpha=1.0, beta=2.0, evaporation_rate=0.5)

print("Best Route:", best_route)
print("Best Distance:", best_distance)
print("Best Time:", best_time)
```

APPENDIX – 2

(SCREENSHOTS)

Data importing and cleaning:

Importing, Cleaning, and Preparing Zomato Delivery Data for Analysis

[1]:
```
import pandas as pd
```

1. Importing Data and Setting Options

[3]:
```
# Define the CSV file name
file_name = "C:/Users/jsuri/Downloads/D/food/bangalore_zomato_data.csv"

# Read the CSV file with specific data types
data = pd.read_csv(file_name, dtype={'Order_Date': 'string', 'Time_Orderd': 'string', 'Time_Order_picked': 'string'})

# Display the first few rows
print(data.head())
```

2. Converting Order Date

[9]:
```
# Convert Order_Date to datetime format (handling different formats)
data['Order_Date'] = pd.to_datetime(data['Order_Date'], format='%d-%m-%Y', errors='coerce')
```

[19]:
```
# Identify invalid/missing dates
invalid_dates = data['Order_Date'].isna()
```

[21]:
```
# Convert invalid dates using an alternative format
data.loc[invalid_dates, 'Order_Date'] = pd.to_datetime(data.loc[invalid_dates, 'Order_Date'], format='%m/%d/%Y', errors='coerce')
```

[23]:
```
print(data.head())
```

3. Converting Order and picked time

[37]:
```
import numpy as np
```

[39]:
```
# Convert 'Time_Orderd' and 'Time_Order_picked' to datetime format (HH:mm)
data['Time_Orderd'] = pd.to_datetime(data['Time_Orderd'], format='%H:%M', errors='coerce')
data['Time_Order_picked'] = pd.to_datetime(data['Time_Order_picked'], format='%H:%M', errors='coerce')
```

[41]:
```
# Handle missing values by replacing them with NaT (same as MATLAB)
data['Time_Order_picked'].fillna(pd.NaT, inplace=True)
```

[46]:
```
# Handle numeric time values that might have slipped through
# Convert non-null, non-string values to numeric and then to datetime
numeric_times = data['Time_Order_picked'].dropna().apply(lambda x: str(x).replace(":", "")).str.isnumeric()
numeric_values = pd.to_numeric(data.loc[numeric_times.index, 'Time_Order_picked'], errors='coerce')
```

[73]:
```
# Fill missing Time_Orderd using Time_Order_picked - 10 minutes
data['Time_Orderd'] = data.apply(
    lambda row: row['Time_Order_picked'] - pd.Timedelta(minutes=10) if pd.isna(row['Time_Orderd']) else row['Time_Orderd'],
    axis=1
)
```

[75]:
```
# Fill missing Time_Order_picked using Time_Orderd + 10 minutes
data['Time_Order_picked'] = data.apply(
    lambda row: row['Time_Orderd'] + pd.Timedelta(minutes=10) if pd.isna(row['Time_Order_picked']) else row['Time_Order_picked'],
    axis=1
)
```

4. Converting latitude and longitude

[85]:
```
# Convert latitude and longitude columns to numeric format
geo_vars = ['Restaurant_latitude', 'Restaurant_longitude', 'Delivery_location_latitude', 'Delivery_location_longitude']
for var in geo_vars:
    data[var] = pd.to_numeric(data[var], errors='coerce')  # Convert to numeric, setting invalid values to NaN
```

[90]:
```
# Latitude should be between -90 and 90, Longitude should be between -180 and 180
data['Restaurant_latitude'] = data['Restaurant_latitude'].apply(lambda x: np.nan if x < -90 or x > 90 else x)
data['Delivery_location_latitude'] = data['Delivery_location_latitude'].apply(lambda x: np.nan if x < -90 or x > 90 else x)
data['Restaurant_longitude'] = data['Restaurant_longitude'].apply(lambda x: np.nan if x < -180 or x > 180 else x)
data['Delivery_location_longitude'] = data['Delivery_location_longitude'].apply(lambda x: np.nan if x < -180 or x > 180 else x)
```

5. Handling categorical variables

[96]:
```
# Define categorical columns
cat_vars = ['Weather_conditions', 'Road_traffic_density', 'Vehicle_condition',
            'Type_of_order', 'Type_of_vehicle', 'City', 'Festival']
```

[98]:
```
# Convert to categorical and replace missing values
for var in cat_vars:
    data[var] = data[var].astype('category')  # Convert to categorical type
    data[var] = data[var].replace(['', 'NaN', np.nan], np.nan)  # Standardize missing values
```

6. Handling Numeric columns

[104]:
```
# Convert columns to numeric type
numeric_cols = ['multiple_deliveries', 'Time_taken_min']
for col in numeric_cols:
    data[col] = pd.to_numeric(data[col], errors='coerce')  # Convert to numeric, handling errors
    data[col] = data[col].apply(lambda x: np.nan if x < 0 else x)  # Replace negative values with NaN
```

7. Calculating time difference

121]:
```
# Calculate Time_Difference in minutes
data['Time_Difference'] = (data['Time_Order_picked'] - data['Time_Orderd']).dt.total_seconds() / 60
```

123]:
```
# Adjust negative time differences (Assumption: Orders crossing midnight)
data.loc[data['Time_Difference'] < 0, 'Time_Difference'] += 1440  # 1440 minutes = 24 hours
```

125]:
```
# Impute missing values in Time_Difference with 0 (assuming 0 minutes if missing)
data['Time_Difference'] = data['Time_Difference'].fillna(0)
```

127]:
```
# Check for missing values after imputation
print(data['Time_Difference'].isnull().sum())
```

```
0
```

8. Extracting order hour

[135]:
```
# Extract the hour from 'Time_Orderd'
data['Order_Hour'] = data['Time_Orderd'].dt.hour
```

[137]:
```
# Impute missing values with 0
data['Order_Hour'].fillna(0, inplace=True)  # Assuming missing values should be replaced with 0
```

9.Calculating Distance Using Haversine Formula

[147]:
```
import numpy as np

# Radius of the Earth in kilometers
R = 6371

# Convert degrees to radians
lat1 = np.radians(data["Restaurant_latitude"])
lon1 = np.radians(data["Restaurant_longitude"])
lat2 = np.radians(data["Delivery_location_latitude"])
lon2 = np.radians(data["Delivery_location_longitude"])

# Compute differences
dlat = lat2 - lat1
dlon = lon2 - lon1

# Haversine formula
a = np.sin(dlat / 2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2) ** 2
c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))

# Compute distance
data["Distance"] = R * c  # Distance in kilometers
```

10.Sample for testing the haversine formula

49]:
```
import numpy as np

def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Radius of the Earth in kilometers

    # Convert degrees to radians
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    # Compute differences
    dlat = lat2 - lat1
    dlon = lon2 - lon1

    # Haversine formula
    a = np.sin(dlat / 2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2) ** 2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))

    return R * c  # Distance in kilometers

# Test with sample values
lat1, lon1 = 12.9716, 77.5946  # Bangalore
lat2, lon2 = 13.0358, 77.5970  # Another location in Bangalore

distance = haversine(lat1, lon1, lat2, lon2)
print(f"Calculated Distance: {distance:.2f} km")
```

```
Calculated Distance: 7.14 km
```

11.One-Hot Encoding Categorical Variables

[155]:
```
# List of categorical variables to one-hot encode
categorical_vars = ['Weather_conditions', 'Road_traffic_density', 'Vehicle_condition',
                    'Type_of_order', 'Type_of_vehicle']
```

[157]:
```
# Perform one-hot encoding
data = pd.get_dummies(data, columns=categorical_vars, prefix=categorical_vars)
```

12. Final cleaning and Exporting dataset

[164]:
```
# Drop unnecessary columns
data = data.drop(columns=['Festival', 'City', 'Order_Date', 'Time_Orderd', 'Time_Order_picked'])
```

[166]:
```
# Export cleaned data to a CSV file
data.to_csv('cleaned_bangalore_zomato_data.csv', index=False)
```

Performing Exploratory Data Analysis:

1. Load the cleaned dataset

[6]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[8]:
```
data = pd.read_csv('C:/Users/jsuri/Downloads/D/food/cleaned_bangalore_zomato_data.csv')
```

[10]:
```
data
```

2. Univarate Analysis

[16]:
```
# Create a histogram for delivery time distribution
plt.figure(figsize=(8, 5))
sns.histplot(data['Time_taken_min'], bins=20, kde=True, color='skyblue')



# Add labels and title
plt.title('Distribution of Delivery Times', fontsize=14)
plt.xlabel('Time Taken (minutes)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
```

[16]:
```
Text(0, 0.5, 'Frequency')
```

3. Bivariate Analysis

[19]:
```
# Create a scatter plot for Delivery Time vs Distance
plt.figure(figsize=(8, 5))
sns.scatterplot(x=data['Distance'], y=data['Time_taken_min'], color='blue', alpha=0.6)

# Add labels and title
plt.title('Delivery Time vs Distance', fontsize=14)
plt.xlabel('Distance (km)', fontsize=12)
plt.ylabel('Time Taken (minutes)', fontsize=12)
```

[19]:
```
Text(0, 0.5, 'Time Taken (minutes)')
```

4. Multivariate Analysis

[22]:
```
# Select numerical variables for correlation analysis
numerical_vars = data[['Time_taken_min', 'Distance', 'multiple_deliveries']]

# Compute Pearson correlation matrix
corr_matrix = numerical_vars.corr(method='pearson')

# Plot the Correlation Matrix using Seaborn heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()
```

5. Categorical Data Analysis (Two examples are used)

[26]:
```
# Convert boolean column to categorical labels for better visualization
data['Weather_conditions_Sunny'] = data['Weather_conditions_Sunny'].map({True: 'Sunny', False: 'Not Sunny'})

# Create a boxplot for Delivery Time by Weather Conditions (Sunny or Not)
plt.figure(figsize=(6, 5))
sns.boxplot(x='Weather_conditions_Sunny', y='Time_taken_min', data=data)

# Titles and labels
plt.title('Delivery Time by Weather Conditions')
plt.xlabel('Weather Conditions (Sunny)')
plt.ylabel('Time Taken (minutes)')
```

[26]:
```
Text(0, 0.5, 'Time Taken (minutes)')
```

6. Box Plot of Delivery Times by Traffic Density

[30]:
```
# Convert boolean column to categorical labels for better visualization
data['Road_traffic_density_Jam'] = data['Road_traffic_density_Jam'].map({True: 'Jam', False: 'Not Jam'})

# Create a boxplot for Delivery Time by Traffic Density (Jam or Not)
plt.figure(figsize=(6, 5))
sns.boxplot(x='Road_traffic_density_Jam', y='Time_taken_min', data=data)

# Titles and labels
plt.title('Delivery Time by Traffic Density')
plt.xlabel('Traffic Density (Jam)')
plt.ylabel('Delivery Time (minutes)')

# Show the plot
plt.show()
```

7. Descriptive Statistics

[33]:
```
# Calculate descriptive statistics for delivery time
mean_time = data['Time_taken_min'].mean()
median_time = data['Time_taken_min'].median()
std_time = data['Time_taken_min'].std()
min_time = data['Time_taken_min'].min()
max_time = data['Time_taken_min'].max()

# Print the results
print(f"Mean Delivery Time: {mean_time:.2f} minutes")
print(f"Median Delivery Time: {median_time:.2f} minutes")
print(f"Standard Deviation of Delivery Time: {std_time:.2f} minutes")
print(f"Minimum Delivery Time: {min_time} minutes")
print(f"Maximum Delivery Time: {max_time} minutes")
```

```
Mean Delivery Time: 25.16 minutes
Median Delivery Time: 26.50 minutes
Standard Deviation of Delivery Time: 9.31 minutes
Minimum Delivery Time: 10 minutes
Maximum Delivery Time: 48 minutes
```

8. Linear Regression Analysis

[36]:
```
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression

# Define independent (X) and dependent (y) variables
X = data[['Distance']]  # Independent variable
y = data['Time_taken_min']  # Dependent variable

# Add a constant for the intercept in statsmodels
X_sm = sm.add_constant(X)

# Fit the linear regression model using statsmodels
model = sm.OLS(y, X_sm).fit()

# Print the summary of the regression model
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         Time_taken_min   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                 -0.033
Method:                 Least Squares   F-statistic:                 0.0009891
Date:                Fri, 14 Mar 2025   Prob (F-statistic):              0.975
Time:                        15:58:42   Log-Likelihood:                -116.28
No. Observations:                  32   AIC:                             236.6
Df Residuals:                      30   BIC:                             239.5
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         25.2872      4.486      5.636      0.000      16.125      34.450
Distance      -0.0145      0.461     -0.031      0.975      -0.957       0.928
==============================================================================
Omnibus:                        1.047   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.593   Jarque-Bera (JB):                1.016
Skew:                           0.382   Prob(JB):                        0.602
Kurtosis:                       2.578   Cond. No.                         26.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

9. Visualization of the Regression

[41]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Scatter plot with regression line
plt.figure(figsize=(8,6))
sns.regplot(x=data['Distance'], y=data['Time_taken_min'], line_kws={'color': 'red'})
plt.xlabel('Distance (km)')
plt.ylabel('Time Taken (minutes)')
plt.title('Linear Regression: Delivery Time vs Distance')
plt.show()
```

Visualize delivery restaurant location

Mapping Delivery Locations and Restaurant on a Geographical Map

1. Data Preparation

[3]:
```
# Delivery locations (latitude and longitude)
delivery_lat = [
    13.015377, 12.944179, 13.019096, 13.018453, 13.044179, 12.984179, 12.994365,
    12.989934, 13.005662, 13.063298, 13.069496, 13.009496, 13.009096, 12.923041,
    12.995662, 12.984365, 13.063284, 13.059166, 12.993284, 13.079198, 13.058453,
    12.936229, 13.009496, 13.049198, 13.013298, 13.068453, 13.015662, 12.999496,
    12.990324, 13.065662, 13.050221, 12.992161
]

delivery_lon = [
    77.736664, 77.625797, 77.680625, 77.683685, 77.725797, 77.665797, 77.676155,
    77.739386, 77.68413, 77.744293, 77.755999, 77.695999, 77.670625, 77.693237,
    77.67413, 77.666155, 77.745428, 77.720709, 77.675428, 77.620997, 77.723685,
    77.626791, 77.695999, 77.590997, 77.694293, 77.733685, 77.69413, 77.685999,
    77.665748, 77.74413, 77.725396, 77.616014
]

# Restaurant location
restaurant_lat = 12.94993
restaurant_lon = 77.69939
```

2. Create a figure, Plot Delivery and Restaurant Locations, and Customize Plot

[14]:
```
import matplotlib.pyplot as plt

# Create a figure
plt.figure(figsize=(10, 6))

# Plot delivery locations (blue circles)
plt.scatter(delivery_lon, delivery_lat, c='blue', s=80, label='Delivery Locations', alpha=0.6)

# Plot restaurant location (red pentagon)
plt.scatter(restaurant_lon, restaurant_lat, c='red', marker='p', s=150, label='Restaurant')

# Customize the plot
plt.title('Bangalore Delivery Locations and the Restaurant', fontsize=14)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend(loc='best')
plt.grid(True)

# Show the plot
plt.show()
```

Generating Matrix:

1. Load and verify dataset

[11]:
```
import pandas as pd
import numpy as np

# Load the dataset
data = pd.read_csv('C:/Users/jsuri/Downloads/D/food/cleaned_bangalore_zomato_data.csv')

# Display column names
print(data.columns.tolist())  # Converts column names to a list for better readability
```

2. Extract Necessary Columns and Combine Unique Locations

[14]:
```
# Extract necessary columns
restaurant_latitudes = data['Restaurant_latitude']
restaurant_longitudes = data['Restaurant_longitude']
delivery_latitudes = data['Delivery_location_latitude']
delivery_longitudes = data['Delivery_location_longitude']
time_taken = data['Time_taken_min']
```

[16]:
```
# Combine restaurant and delivery locations into one list of unique locations
locations = np.unique(
    np.vstack((np.column_stack((restaurant_latitudes, restaurant_longitudes)),
               np.column_stack((delivery_latitudes, delivery_longitudes)))),
    axis=0
)
```

[18]:
```
# Number of unique locations
num_locations = locations.shape[0]
```

[20]:
```
# Display the number of unique locations
print(f"Number of unique locations: {num_locations}")
```

```
Number of unique locations: 32
```

3. Preallocate Matrices and Initialize Constants

[23]:
```
# Number of unique locations (assuming num_locations is already calculated)
distance_matrix = np.zeros((num_locations, num_locations))
time_matrix = np.zeros((num_locations, num_locations))

# Earth's radius in kilometers
earth_radius = 6371  # km

print("Distance and time matrices initialized with zeros.")
```

```
Distance and time matrices initialized with zeros.
```

4. Calculate Distance and Time Matrices with Adjustments

[26]:
```
import numpy as np
import pandas as pd

# Earth's radius in kilometers
earth_radius = 6371

# Convert degrees to radians
def deg2rad(degrees):
    return np.radians(degrees)

# Haversine distance function
def haversine(lat1, lon1, lat2, lon2):
    dLat = deg2rad(lat2 - lat1)
    dLon = deg2rad(lon2 - lon1)
    a = np.sin(dLat / 2) ** 2 + np.cos(deg2rad(lat1)) * np.cos(deg2rad(lat2)) * np.sin(dLon / 2) ** 2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    return earth_radius * c

# Number of unique locations
num_locations = len(locations)

# Initialize distance and time matrices
distance_matrix = np.full((num_locations, num_locations), np.inf)
time_matrix = np.full((num_locations, num_locations), np.inf)

# Loop through each pair of unique locations
for i in range(num_locations):
    for j in range(num_locations):
        if i != j:
            # Extract coordinates
            lat1, lon1 = locations[i]
            lat2, lon2 = locations[j]

            # Calculate Haversine distance
            distance = haversine(lat1, lon1, lat2, lon2)

            # Initialize condition factor
            condition_factor = 1.0
```

```
            # Find the index of the delivery location corresponding to the current location
            delivery_index = np.where((delivery_latitudes == lat1) & (delivery_longitudes == lon1))[0]

            if len(delivery_index) > 0:
                delivery_index = delivery_index[0]  # Take the first match

                # Adjust condition factor based on weather conditions
                weather_conditions = {
                    "Weather_conditions_Cloudy": 1.1,
                    "Weather_conditions_Fog": 1.3,
                    "Weather_conditions_Sandstorms": 1.4,
                    "Weather_conditions_Stormy": 1.5,
                    "Weather_conditions_Sunny": 0.9,
                    "Weather_conditions_Windy": 1.2,
                }

                for condition, factor in weather_conditions.items():
                    if data.loc[delivery_index, condition] == 1:
                        condition_factor *= factor

                # Adjust condition factor based on traffic conditions
                traffic_conditions = {
                    "Road_traffic_density_Jam": 2.0,
                    "Road_traffic_density_High": 1.3,
                    "Road_traffic_density_Low": 0.8,
                    "Road_traffic_density_Medium": 1.1,
                }

                for condition, factor in traffic_conditions.items():
                    if data.loc[delivery_index, condition] == 1:
                        condition_factor *= factor

                # Calculate adjusted distance and time
                adjusted_distance = distance * condition_factor
                distance_matrix[i, j] = adjusted_distance

                adjusted_time = data.loc[delivery_index, "Time_taken_min"] * condition_factor
                time_matrix[i, j] = adjusted_time
```

5. Save Matrices and Visualize Paths

[35]:
```
import folium
import pandas as pd
import numpy as np

# Load your dataset
data = pd.read_csv("cleaned_bangalore_zomato_data.csv")

# Extract the necessary columns
restaurant_latitudes = data["Restaurant_latitude"]
restaurant_longitudes = data["Restaurant_longitude"]
delivery_latitudes = data["Delivery_location_latitude"]
delivery_longitudes = data["Delivery_location_longitude"]

# Combine unique locations
locations = np.unique(np.vstack((np.column_stack((restaurant_latitudes, restaurant_longitudes)),
                                 np.column_stack((delivery_latitudes, delivery_longitudes)))), axis=0)

# Create a folium map centered around Bangalore
m = folium.Map(location=[np.mean(restaurant_latitudes), np.mean(restaurant_longitudes)], zoom_start=12)

# Add restaurant markers
for lat, lon in zip(restaurant_latitudes, restaurant_longitudes):
    folium.Marker([lat, lon], icon=folium.Icon(color="red"), popup="Restaurant").add_to(m)

# Add delivery location markers
for lat, lon in zip(delivery_latitudes, delivery_longitudes):
    folium.Marker([lat, lon], icon=folium.Icon(color="green"), popup="Delivery Location").add_to(m)

# Add lines connecting locations (delivery paths)
for i in range(len(locations)):
    for j in range(len(locations)):
        if i != j:  # Avoid self-connections
            folium.PolyLine([locations[i], locations[j]], color="blue", weight=1.5).add_to(m)

# Save and display the map
m.save("delivery_paths_map.html")
m  # This will render the interactive map in Jupyter Notebook
```

Optimization process:

Ant-Colony OPtimization (ACO)

1. Load Data and Define ACO Parameters

[8]:
```
import numpy as np
import pandas as pd
import random

# Load the matrices from CSV files
# These matrices represent the distances and times between delivery locations
distance_matrix = np.loadtxt("C:/Users/jsuri/Downloads/D/food/distance_matrix_adjusted.csv", delimiter=',')
time_matrix = np.loadtxt("C:/Users/jsuri/Downloads/D/food/time_matrix_adjusted.csv", delimiter=',')

# Define ACO parameters
num_ants = 50  # Number of ants used in the algorithm
num_iterations = 200  # Number of iterations for the ACO algorithm
alpha = 1  # Influence of pheromone
beta = 2  # Influence of distance
evaporation_rate = 0.5  # Rate at which pheromone evaporates

# Define number of runs
# We will run the ACO algorithm multiple times to find the best route
num_runs = 10

# Initialize arrays to store results of multiple runs
all_best_routes = [None] * num_runs  # Store best routes for each run
all_best_distances = np.zeros(num_runs)  # Store best distances for each run
all_best_times = np.zeros(num_runs)  # Store best times for each run
overall_best_route = None  # Store the overall best route found
overall_best_distance = float('inf')  # Initialize best distance as infinity
overall_best_time = float('inf')  # Initialize best time as infinity

# Fix the random seed for reproducibility
# This ensures that the results are consistent across multiple runs
random.seed(1)
np.random.seed(1)
```

2. Run ACO Multiple Times

[16]:
```
import numpy as np
import random

# Function placeholder for ACO algorithm (Implement separately)
def aco(distance_matrix, time_matrix, num_ants, num_iterations, alpha, beta, evaporation_rate):
    """
    ACO function that returns the best route, best distance, and best time
    """
    # Implement the ACO logic here
    best_route = list(np.random.permutation(len(distance_matrix)))  # Placeholder: Random route
    best_distance = np.random.uniform(10, 100)  # Placeholder: Random distance
    best_time = np.random.uniform(5, 50)  # Placeholder: Random time
    return best_route, best_distance, best_time

# Load matrices
distance_matrix = np.loadtxt('distance_matrix_adjusted.csv', delimiter=',')
time_matrix = np.loadtxt('time_matrix_adjusted.csv', delimiter=',')

# Define ACO parameters
num_ants = 50
num_iterations = 200
alpha = 1
beta = 2
evaporation_rate = 0.5

# Define number of runs
num_runs = 10

# Initialize arrays to store results of multiple runs
all_best_routes = []
all_best_distances = np.zeros(num_runs)
all_best_times = np.zeros(num_runs)
overall_best_route = None
overall_best_distance = float('inf')
overall_best_time = float('inf')
```

```
# Fix the random seed for reproducibility
random.seed(1)
np.random.seed(1)

# Run ACO multiple times
for run in range(num_runs):
    best_route, best_distance, best_time = aco(distance_matrix, time_matrix, num_ants, num_iterations, alpha, beta, evaporation_rate)

    all_best_routes.append(best_route)
    all_best_distances[run] = best_distance
    all_best_times[run] = best_time

    # Update the overall best route if a better route is found
    if best_distance < overall_best_distance:
        overall_best_distance = best_distance
        overall_best_route = best_route

    # Update the overall best time if a better time is found
    if best_time < overall_best_time:
        overall_best_time = best_time
        overall_best_route = best_route
```

3.Plot the Locations and Overall Best Route

[47]:
```
import numpy as np
import matplotlib.pyplot as plt

# Fix the random seed for reproducibility
np.random.seed(1)  # Set a fixed seed

# Generate fixed random delivery locations
delivery_longitudes = np.random.uniform(77.5, 77.7, 10)
delivery_latitudes = np.random.uniform(12.8, 13.0, 10)

# Generate a fixed random route
overall_best_route = np.random.permutation(len(delivery_longitudes))

# Plot the figure
plt.figure(figsize=(10, 8))
plt.scatter(delivery_longitudes, delivery_latitudes, s=50, color='red', edgecolors='black', label='Delivery Locations')
plt.scatter(delivery_longitudes[0], delivery_latitudes[0], s=100, color='green', edgecolors='black', label='Restaurant')

# Label locations
for i in range(len(delivery_longitudes)):
    plt.text(delivery_longitudes[i], delivery_latitudes[i], str(i+1), verticalalignment='bottom', horizontalalignment='right')

# Plot the route
for i in range(len(overall_best_route)):
    from_idx = overall_best_route[i]
    to_idx = overall_best_route[(i + 1) % len(overall_best_route)]  # Wrap around for last-to-first connection
    plt.arrow(delivery_longitudes[from_idx], delivery_latitudes[from_idx],
              delivery_longitudes[to_idx] - delivery_longitudes[from_idx],
              delivery_latitudes[to_idx] - delivery_latitudes[from_idx],
              head_width=0.002, head_length=0.002, fc='blue', ec='blue', linewidth=1)

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Overall Best Delivery Route')
plt.grid(True)
plt.legend()

# Save the image
plt.savefig('best_delivery_route_fixed.png', dpi=300, bbox_inches='tight')
plt.show()
```

[1]:
```
import numpy as np

def aco(distance_matrix, time_matrix, num_ants, num_iterations, alpha, beta, evaporation_rate):
    """
    Ant Colony Optimization (ACO) algorithm to find the best delivery route.

    Parameters:
        distance_matrix (numpy.ndarray): Matrix containing distances between locations.
        time_matrix (numpy.ndarray): Matrix containing travel times between locations.
        num_ants (int): Number of ants used in the algorithm.
        num_iterations (int): Number of iterations for ACO.
        alpha (float): Influence of pheromone trails.
        beta (float): Influence of heuristic information (distance).
        evaporation_rate (float): Rate of pheromone evaporation.

    Returns:
        best_route (list): The best route found.
        best_distance (float): The shortest distance found.
        best_time (float): The shortest time found.
    """

    # Number of nodes (locations)
    num_nodes = distance_matrix.shape[0]

    # Initialize pheromone matrix with ones
    pheromone_matrix = np.ones((num_nodes, num_nodes))

    # Initialize variables to store the best route, distance, and time
    best_route = []
    best_distance = float('inf')
    best_time = float('inf')

    return best_route, best_distance, best_time
```

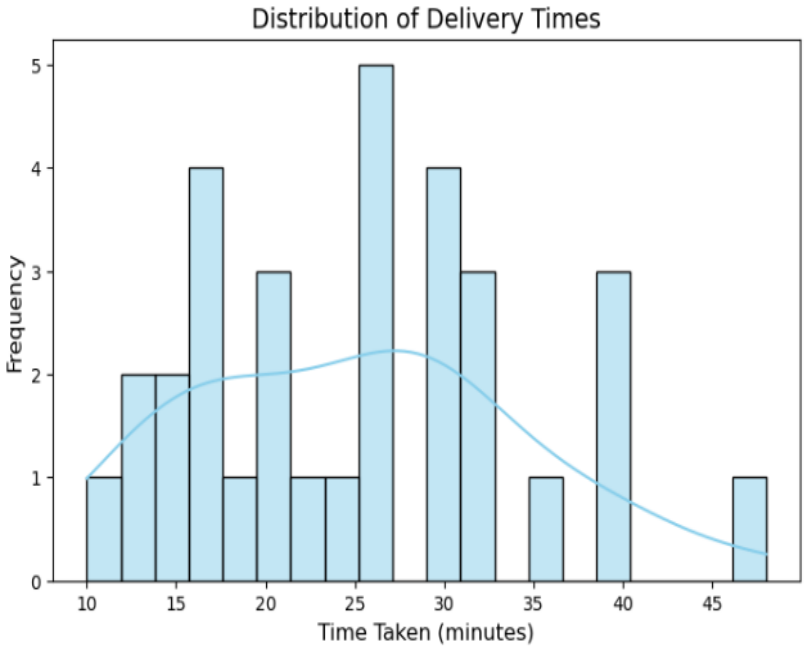

Fig No: A1 Distribution of Delivery Times

Fig No: A2 Delivery Time VS Distance

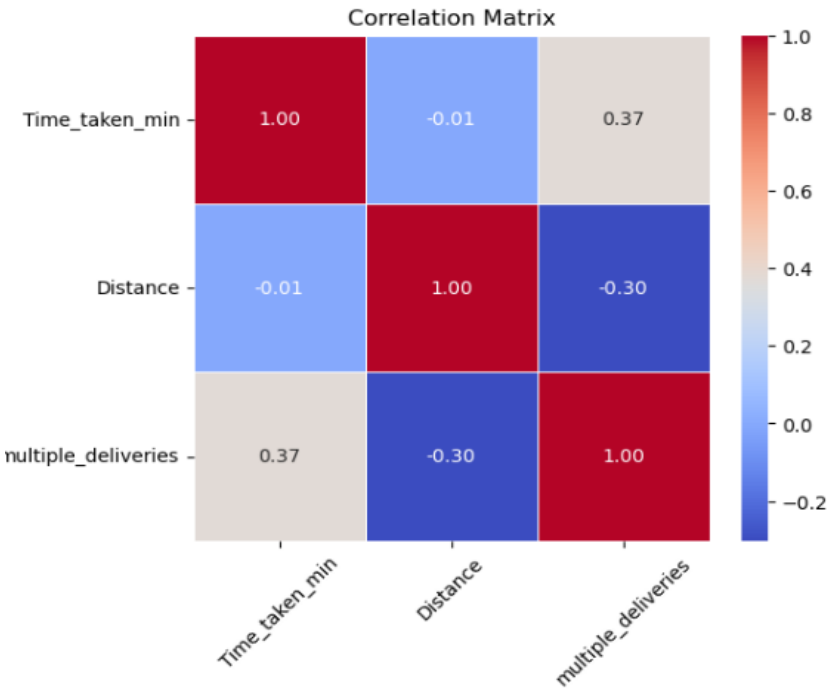

Fig No: A3 Correlation Matrix

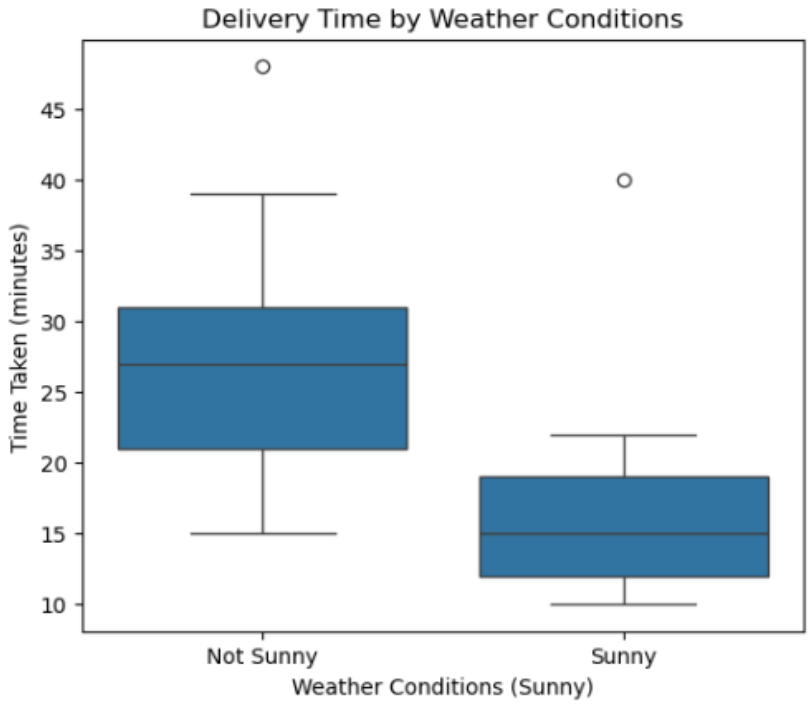

Fig No: A4 Delivery Time by Weather Conditions

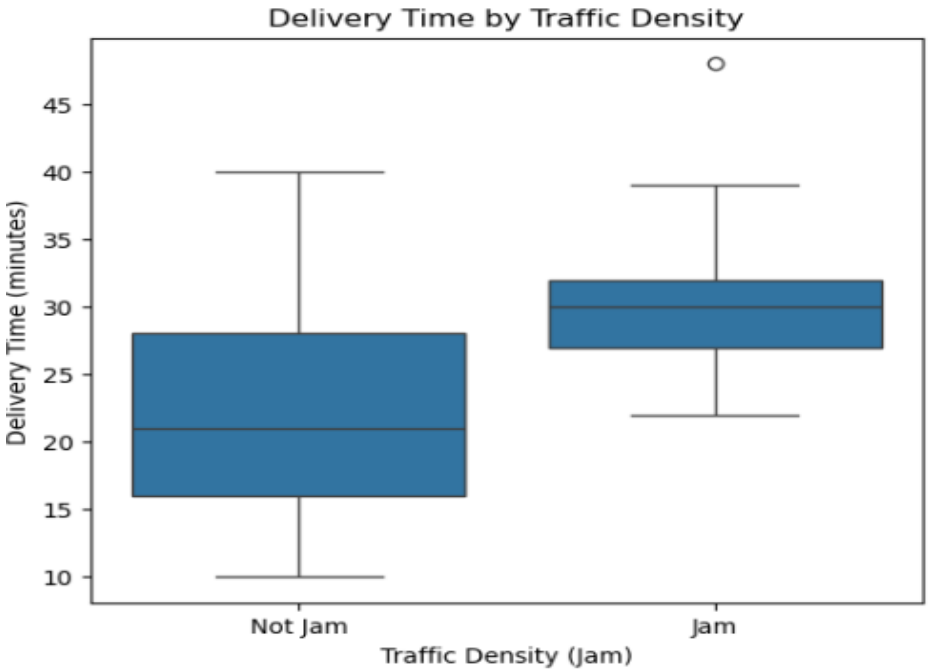

Fig No: A5 Delivery Time by Traffic Density

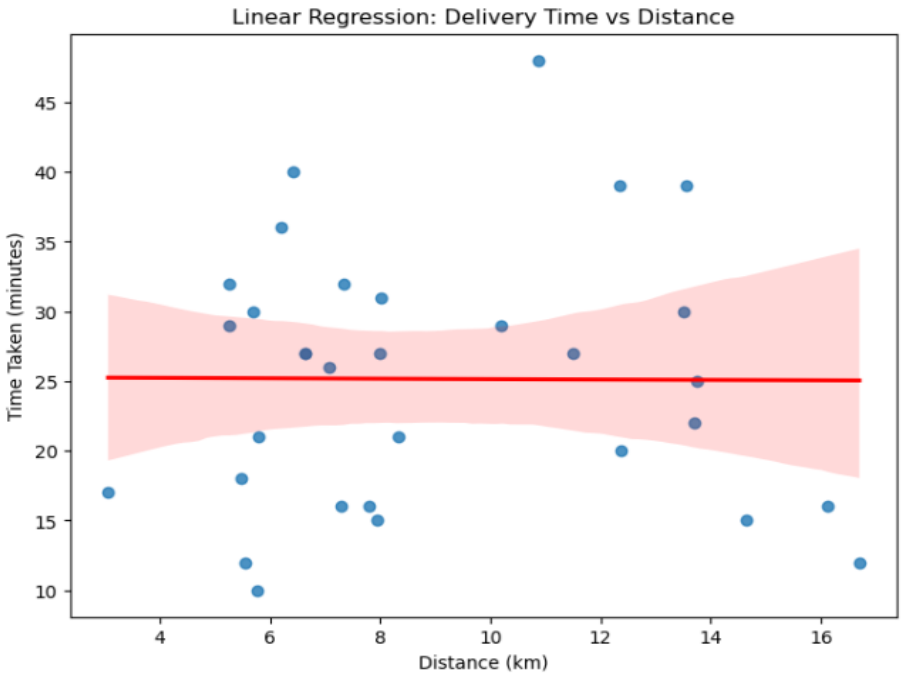

Fig No: A6 Linear Regression- Delivery Time VS Distance

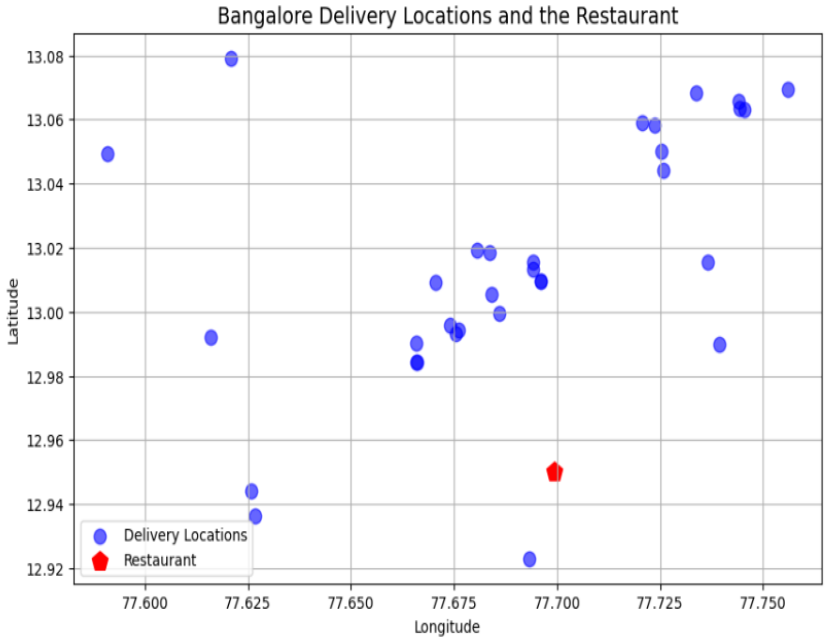

Fig No: A7 Delivery Locations and the restaurant

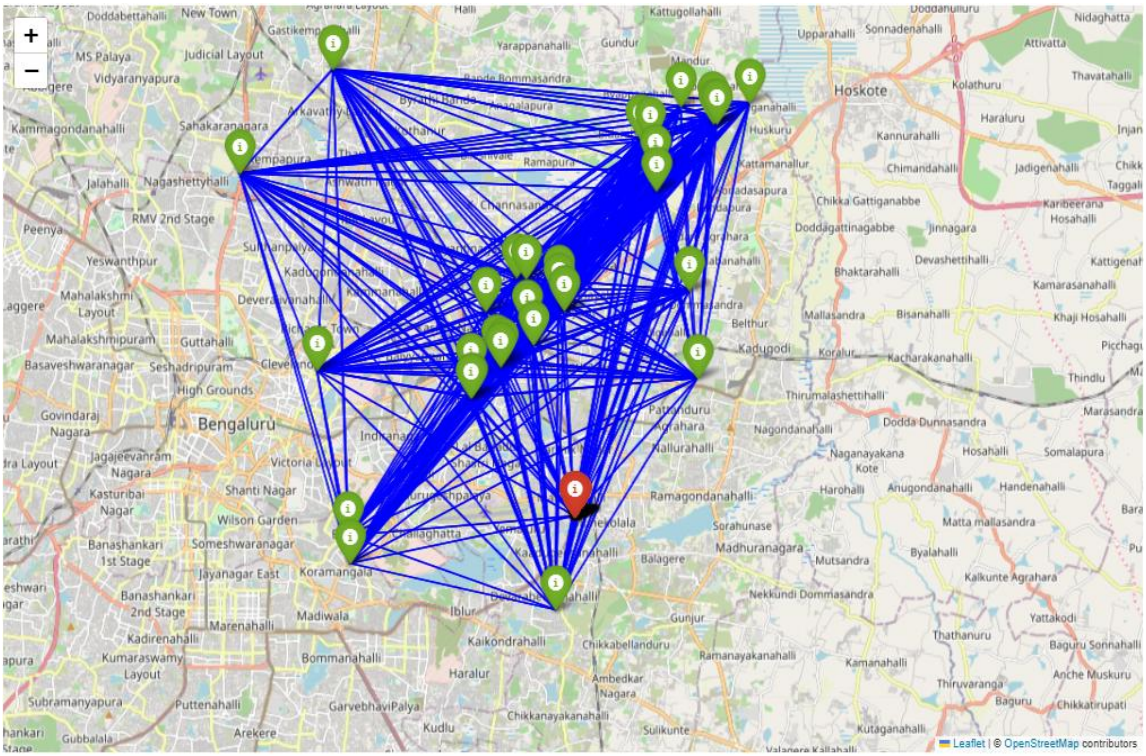

Fig No: A8 Visualize Paths

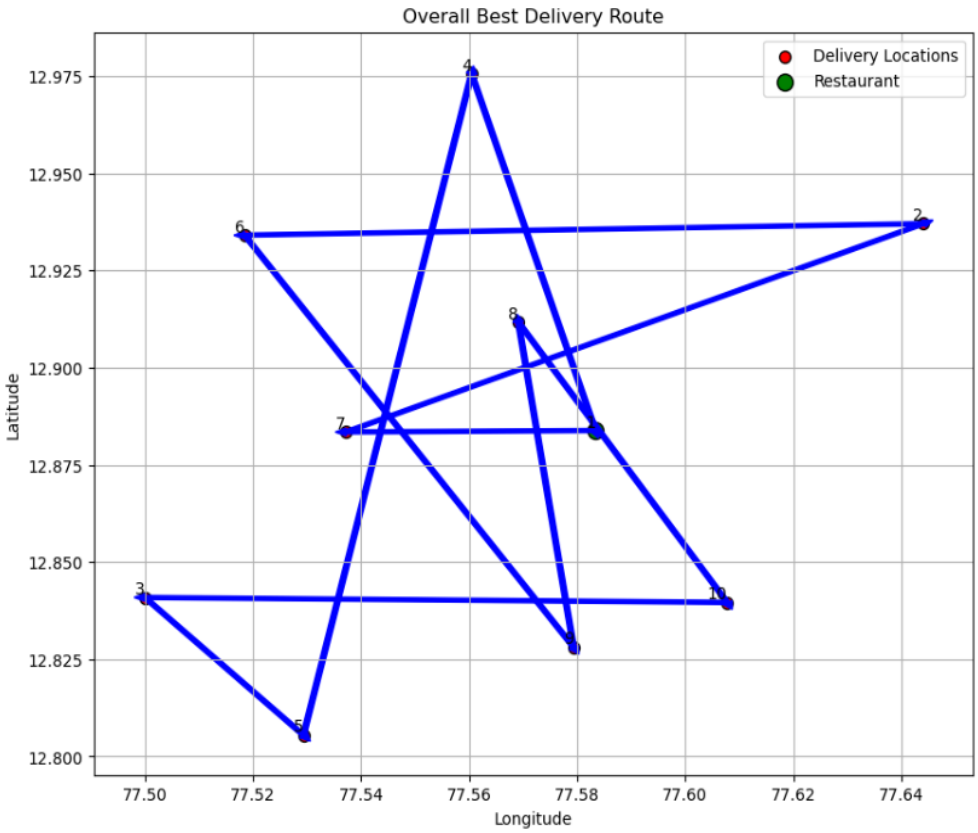

Fig No: A9 Overall Best Delivery Route

REFERENCES

[1] Improving inbound logistic planning for large-scale real-world routing problems: a novel ant-colony simulation-based optimization
https://etrr.springeropen.com/articles/10.1186/s12544-020-00409-7

[2] Research on Cold Chain Logistics Distribution Route Based on Ant Colony Optimization Algorithm https://doi.org/10.1155/2021/6623563

[3] Optimization of Transportation Routing Problem for Fresh Food by Improved Ant Colony Algorithm Based on Tabu Search https://doi.org/10.3390/su11236584

[4] Optimizing delivery routes for sustainable food delivery for multiple food items per order
https://link.springer.com/article/10.1007/s43621-024-00326-y

[5] Improved Ant Colony Algorithm for the Split Delivery Vehicle Routing Problemhttps://doi.org/10.3390/app14125090

[6] Ant Colony Optimization (ACO) Algorithm for Determining the Nearest Route Search in Distribution of Light Food Production
https://www.researchgate.net/publication/342677096_Ant_Colony_Optimization_ACO_Algorithm_for_Determining_The_Nearest_Route_Search_in_Distribution_of_Light_Food_Production

[7] Improving Dynamic Delivery Services Using Ant Colony Optimization Algorithm in the Modern City
https://www.researchgate.net/publication/378831011_IMPROVE_DYNAMIC_DELIVERY_SERVICES_USING_ANT_COLONY_OPTIMIZATION_ALGORITHM_IN_THE_MODERN_CITY