OPTIMIZATION OF TOURISM INDUSTRY

Ahmed Ayman Sayed	**20201002**
Ahmed Ibrahim Mahmoud	**20200007**
Ahmed Mohamed Tonsy	**20200039**
Kareem Mohamed Kadrey	**20200394**

Supervised By

Assoc. Prof. Sally Kassem
Eng. Nesma Mohamed

CAIRO UNIVERSITY

JULY, 2024

ABSTRACT

The tourism industry is a dynamic and competitive sector where new businesses face numerous challenges in making strategic decisions. These challenges include optimal resource allocation, designing attractive tour packages, and setting competitive yet profitable prices. The complexity lies in balancing profitability while navigating an ever-changing market landscape. Existing models often fall short in providing comprehensive solutions tailored to the unique needs of these startups.

Our project addresses these challenges by developing a robust decision support framework designed to assist new tourism businesses. This framework integrates advanced optimization techniques to streamline resource allocation, enhance tour package attractiveness, and refine pricing strategies. By focusing on key performance indicators, our methodology involves data-driven analysis, predictive modeling, and scenario simulations to guide businesses towards making informed decisions. This strategic approach aims to enhance operational efficiency, maximize profitability, thereby providing a competitive edge in the tourism market.

We employed a rigorous testing and validation process to ensure the reliability and effectiveness of our framework. We created a mathematical model from scratch and solved it using a metaheuristic solution with a genetic algorithm in Python. Data analysis was conducted using Excel dashboards, and we developed a user interface using C# Windows Form application with .NET Framework. Additionally, we developed an AI prediction model to forecast tour package demand based on price, minimum demand, maximum demand, and year, using data from 2019 to 2023 to predict demand for 2025 to 2028. The results demonstrated significant improvements in resource utilization and revenue generation for the businesses using our framework. Ultimately, our project offers a comprehensive solution that empowers new tourism ventures to achieve sustained success and adaptability in a constantly evolving industry.

DECLARATION

We hereby declare that our dissertation is entirely our work and genuine / original. We understand that in case of discovery of any PLAGIARISM at any stage, our group will be assigned an F (FAIL) grade and it may result in withdrawal of our Bachelor's degree.

Group members:

Name	**Signature**
Ahmed Ayman Sayed	_____
Ahmed Ibrahim Mahmoud	_____
Ahmed Mohamed Tonsy	_____
Kareem Mohamed Kadrey	_____

PLAIGRISM CERTIFICATE

This is to certify that the project entitled "**Optimization of Tourism Industry**", which is being submitted here with for the award of the "**Bachelor of Computer and Artificial Intelligence Degree" in "Operations Research and Decision Support** ". This is the result of the original work by Ahmed Ayman Sayed, Ahmed Ibrahim Mahmoud, Ahmed Mohamed Tonsy and Kareem Mohamed Kadrey under my supervision and guidance. The work embodied in this project has not been done earlier for the basis of award of any degree or compatible certificate or similar tile of this for any other diploma/examining body or university to the best of my knowledge and belief.

Turnitin Originality Report

Processed on 6-Jul-2024 00:14 PKT

ID: 300502964

Word Count: 19395

Similarity Index

10%

Similarity by Source

Internet Sources:	06%
Publications:	0 %
Student Papers:	08%

Date: 08/07/2024

Assoc. Prof. Sally Kassem

ACKNOWLEDGMENT

We would like to express our deepest gratitude to our supervisor, Assoc. Prof. Sally Kassem, for her invaluable guidance, support, and encouragement throughout the course of this project. Her insights and expertise were crucial to our success.

We also extend our sincere thanks to Eng. Nesma Mohamed for her continuous assistance and helpful feedback, which greatly contributed to the completion of our work.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Caption** **Page**

LIST OF TABLES

Table **Caption** **Page**

CHAPTER 1

INTRODUCTION

1.1 Introduction

The tourism industry plays a pivotal role in the global economy, contributing to economic growth, job creation, and cultural exchange. It is a multifaceted sector that encompasses various services such as hospitality, transportation, entertainment, and destination management. However, the industry's dynamic nature poses significant challenges for new entrants who must navigate a highly competitive and fluctuating market. The optimization of operations within the tourism industry is essential to achieve sustainable growth and profitability.

Optimization of the tourism industry involves the application of advanced analytical techniques and models to enhance decision-making processes. It aims to improve resource allocation and boost overall efficiency. Key areas of focus include pricing strategies, tour package design, resource management, and marketing effectiveness. By optimizing these elements, tourism businesses can better meet customer demands, reduce operational costs, and increase revenue.

One critical aspect of optimization is the effective allocation of limited resources, such as budget, personnel, and time, to various activities and destinations. This requires a deep understanding of market trends and competitive dynamics. Advanced data analytics and modeling techniques, such as predictive analytics, machine learning, and metaheuristic algorithms, provide the tools necessary to analyze vast amounts of data and derive actionable insights.

Another important facet is the design of tour packages that appeal to different customer segments while ensuring profitability. This involves balancing the cost of services provided with competitive pricing, all while delivering a compelling value proposition to customers. Optimization techniques help identify the most attractive combinations of services and experiences, tailored to the specific needs and preferences of target markets.

Furthermore, pricing strategies in the tourism industry must consider factors such as demand elasticity, seasonal variations, and competitive pricing. Dynamic pricing models and algorithms enable businesses to adjust prices in real-time based on market conditions, maximizing both occupancy rates and revenue. This approach requires sophisticated optimization methods to ensure prices are set optimally to attract customers while maintaining profitability.

The ultimate goal of optimization in the tourism industry is to create a resilient and adaptable business model capable of thriving in an ever-changing environment. By leveraging advanced optimization techniques, new tourism businesses can enhance their strategic decision-making capabilities, improve operational efficiency, and achieve sustainable growth. This project seeks to contribute to this objective by developing a comprehensive decision support framework tailored to the specific needs of new entrants in the tourism sector.

1.2 Problem Statement

The main challenge we're dealing with in this project is creating and using a model that helps new tourism businesses make the right decisions. These businesses need to figure out how to use their resources in the best way, create attractive tour packages, set prices that work, and use marketing channels effectively to reach the people they want to. The heart of the problem is finding a balance, a strategy that makes the most money while also meeting the many needs and wants of the customers. These businesses must figure out things like: How can they use their limited resources in different places to make customers happy? What kinds of tour packages will be the most popular? How should they set prices that make sense for their costs and what the competition is doing? And how can they use marketing to make people more aware of their brand and get them interested?

This problem is even more complicated because businesses must think about both what's happening inside their company and what's happening outside in the tourism industry. What customers want can be different and can change because of things like cultural trends, the time of year, and new travel preferences. Also, these businesses need to be ready to change things when the market changes or when something unexpected happens that could affect how people travel and what they want.

In short, this project aims to give a full solution to the many challenges new tourism businesses face. It's not just about making the most money or spending the least; it's about making decisions with the customer in mind, adapting to what's happening outside, and working toward lasting success in a field that's always changing. The framework we're developing isn't just a math model; it's like a guide that helps these businesses be strong, creative, and successful in the long run.

1.3 Proposed System

1.3.1 Aims and Objectives

The proposed system for optimizing the tourism industry aims to develop a comprehensive decision support framework tailored for new tourism businesses. The primary objectives are:

- **Optimal Resource Allocation:** To enable businesses to allocate their limited resources (budget, personnel, time) effectively across various activities and destinations.
- **Enhanced Tour Package Design:** To assist in designing attractive and profitable tour packages that cater to diverse customer segments.
- **Dynamic Pricing Strategies:** To implement pricing strategies that adjust in real-time based on market conditions, demand elasticity, and competitive pricing.
- **Improved Decision-Making:** To enhance the strategic decision-making capabilities of new tourism businesses by providing data-driven insights and recommendations.

1.3.2 Proposed System Features

The proposed decision support framework integrates several advanced analytical techniques and methodologies to provide a holistic solution. Key features of the system include:

- **Mathematical Modeling:**
  - Development of a mathematical model from scratch to encapsulate the complexities of resource allocation, tour package optimization, and pricing strategies.
  - Consideration of various constraints and objectives such as budget limitations and market competition.
- **Metaheuristic Optimization:**
  - Utilization of a genetic algorithm to solve the mathematical model.
  - Identification of near-optimal solutions within a reasonable time frame, making it suitable for the dynamic and complex nature of the tourism industry.
- **Data Analysis and Visualization:**
  - Leveraging Excel dashboards for comprehensive data analysis.

- Provision of insightful visualizations of key performance indicators, market trends, and customer behavior.

❖ **User Interface Development:**

- Development of a user-friendly interface using C# Windows Form application with .NET Framework.
- Allowing users to interact with the framework easily, input relevant data, and receive actionable insights and recommendations.

System Functionality:

❖ **Resource Allocation:**

- Assisting businesses in effectively allocating their resources to various activities and destinations based on data-driven insights and predictive analytics.

❖ **Tour Package Optimization:**

- Helping design tour packages that appeal to customers and ensure profitability by identifying the most attractive combinations of services and experiences tailored to specific customer segments.

❖ **Dynamic Pricing:**

- Implementing dynamic pricing strategies to adjust prices in real-time based on market conditions, demand elasticity, and competitive pricing, ensuring competitive prices while maximizing revenue.

1.4 Development Methodology

The development of the decision support framework for optimizing the tourism industry follows a structured methodology to ensure the creation of an effective, efficient, and user-friendly system. The methodology is divided into several key phases: requirements analysis, system design, implementation, testing, and deployment. Each phase is crucial for the successful completion of the project.

1.4.1 Requirements Analysis

The first phase involves gathering and analyzing the requirements of the new tourism businesses that the framework aims to support. This includes:

- **Identifying Key Challenges:** Understanding the primary challenges faced by new tourism businesses in resource allocation, tour package design, pricing strategies, and marketing.
- **Defining Objectives:** Establishing clear objectives for the decision support framework, such as improving operational efficiency, maximizing profitability.
- **Stakeholder Consultation:** Engaging with stakeholders, including business owners, industry experts, and potential users, to gather insights and refine requirements.

1.4.2 System Design

The system design phase translates the requirements into a detailed blueprint for the framework. This phase includes:

- **Architectural Design:** Defining the overall architecture of the system, including the integration of the mathematical model, optimization algorithm, data analysis tools, and user interface.
- **Component Design:** Designing individual components such as the genetic algorithm for optimization, Excel dashboards for data visualization, and the C# Windows Form application for the user interface.
- **Data Flow Diagrams:** Creating data flow diagrams to map out the flow of information within the system, ensuring all components work together seamlessly.

1.4.3 Implementation

In the implementation phase, the design is brought to life through coding and software development. Key activities include:

- **Model Development:** Creating a mathematical model from scratch to address the specific optimization needs of new tourism businesses.
- **Algorithm Integration:** Implementing the genetic algorithm in Python to solve the mathematical model and identify near-optimal solutions.
- **Data Analysis Tools:** Developing Excel dashboards to analyze and visualize key performance indicators, market trends, and customer behavior.
- **User Interface Development:** Building the user interface using C# Windows Form application with .NET Framework to ensure ease of use and accessibility.

1.4.4 Testing

Testing is a critical phase to ensure the reliability, accuracy, and usability of the framework. This phase includes:

- **Unit Testing:** Testing individual components to ensure they function correctly in isolation.
- **Integration Testing:** Ensuring that all components work together as intended and that data flows smoothly between them.
- **System Testing:** Evaluating the entire system in a simulated environment to identify and fix any issues.
- **User Acceptance Testing (UAT):** Conducting tests with potential users to ensure the system meets their needs and is easy to use.

1.4.5 Deployment

The final phase involves deploying the system for use by new tourism businesses. This phase includes:

- **Deployment Planning:** Creating a detailed deployment plan to ensure a smooth rollout of the system.
- **User Training:** Providing training and support to users to help them understand and effectively use the system.
- **Maintenance and Support:** Establishing a maintenance plan to address any issues that arise post-deployment and to ensure the system remains up-to-date and effective.

1.4.6 Iterative Improvement

Recognizing the dynamic nature of the tourism industry, the development methodology incorporates iterative improvement. Feedback from users and stakeholders is continuously gathered to make necessary adjustments and enhancements to the system, ensuring it remains relevant and effective over time.

1.5 Resource Requirement

1.5.1 Subsystem Requirements

- **Programming Languages and Development Tools:**
  - **Python:** For implementing genetic algorithms and other optimization routines.
  - **C# with .NET Framework:** For developing the user interface using Windows Form application.
  - **Excel:** For creating dashboards and conducting data analysis.
- **Development Environments:**
  - **IDE:** Jupyter Notebook for Python development and Visual Studio for C# development.
- **Version Control System:**
  - **GitHub:** For version control and collaborative development.
- **Optimization and Modeling Tools:**
  - **Genetic Algorithms:** For solving the mathematical model.
- **Data Analysis and Visualization:**
  - **Excel Dashboards:** For visualizing key performance indicators, market trends, and customer behavior.

1.5.2 Data Requirements

- **Sample Data Sets:**
  - **Publicly Available Tourism Data:** Relevant data sets for testing and validating the optimization model and framework.
  - **Synthetic Data:** Generated to simulate real-world scenarios.
  - **Stakeholder Data:** Data provided by stakeholders in the tourism industry.
- **Data Analysis:**
  - **Data Cleaning:** Ensuring the data is accurate and ready for analysis.
  - **Statistical Analysis:** Applying statistics and probabilities to derive actionable insights.

1.5.3 Hardware Resources

- **Computers:**
  - Laptops or desktops with sufficient processing power and memory to handle development tasks, data analysis, and testing.
- **Storage:**
  - External storage devices for backup and data management.

1.6 Report Layout

This report is structured to comprehensively document the development and implementation of the decision support framework for optimizing the tourism industry. The layout is designed to provide clear insights into each stage of the project, from problem identification to final results and conclusions.

- **Chapter 1: Introduction**
  - Provides an overview of the project, including the problem statement, objectives, and proposed system.
- **Chapter 2: Background/Existing Work**
  - Reviews existing research , Background and methodologies relevant to optimization in the tourism industry and decision support systems.
- **Chapter 3: System Design and Architecture**
  - Details the design and architectural components of the proposed system, including data flow and component specifications.
- **Chapter 4: Implementation**
  - Describes the implementation process of the mathematical model, genetic algorithm, data analysis tools, and user interface.
- **Chapter 5: Testing and Validation**
  - Covers the testing methodologies used, results from various testing phases, and validation of the system's effectiveness.
- **Chapter 6: Conclusion and Future Work**
  - Summarizes the project's achievements, discusses any limitations, and provides recommendations for future improvements and research.

❖ **References**

- Lists all the sources and literature cited throughout the report.

This structure ensures a logical flow of information, making it easy to follow the project's progression from conception to completion.

CHAPTER 2

BACKGROUND/EXISTING WORK

2.1 Introduction

The optimization of the tourism industry has garnered significant attention in research and development. Numerous studies focus on improving resource allocation, pricing strategies. However, new tourism businesses often face unique challenges due to the industry's dynamic nature and the lack of accessible decision support systems. This chapter reviews existing work in the field, highlighting limitations and gaps, and introduces the innovative aspects and design of our project.

2.2 Overview of Project

Our project comprises several key phases:

- **Initial Research and Model Development:**
  - **Search for Existing Models:** Initially, we searched for a suitable mathematical model to optimize tourism operations but found none that met our needs.
  - **Creation of Preliminary Model:** We developed a mathematical model from scratch and tested it with hypothetical data.
- **Data Collection and Analysis:**
  - **Real-life Data Acquisition:** We sourced real-life data from an online tourism company.
  - **Data Preprocessing:** Cleaning and analyzing the data to ensure its suitability for our model.
- **Algorithm Testing:**
  - **Metaheuristic Algorithms:** Implemented various algorithms in Python, including:
    - Genetic Algorithm
    - Ant Colony Optimization
  - **Algorithm Selection:** Determined that the Genetic Algorithm was most suitable due to its efficiency in time and space complexity.
- **System Development:**
  - **User Interface:** Developed a C# Windows Forms application with the .NET framework.
  - **Database Integration:** Integrated real data into a database, allowing admins to manage tour packages, budgets, and resources.

❖ **AI Prediction Model:**

- We developed an AI model to predict future tour package demand, focusing on creating packages that maximize profit.

2.3 Limitations of Project

Despite our comprehensive approach, several limitations exist:

❖ **Data Dependency:**

- The accuracy of our model's predictions and optimizations relies heavily on the quality and availability of data from tourism companies.

❖ **Scalability:**

- While designed for typical tourism company operations, scaling up for larger datasets or more complex scenarios may require significant enhancements.

2.3.1 Innovations in Project

Our project introduces several innovative elements:

❖ **Custom Mathematical Model:**

- A bespoke model tailored specifically for new tourism businesses, integrating resource allocation, pricing strategies.

❖ **Multiple Metaheuristic Algorithms:**

- Testing and implementing various algorithms to find the most efficient optimization technique, with the Genetic Algorithm proving most suitable.

❖ **AI Prediction Model:**

- An advanced AI model to forecast future demand, providing actionable insights for businesses to stay competitive.

2.3.2 Design of Project

The design of our project includes:

- ❖ **Robust Decision Support Framework:**
  - A meticulously developed and tested mathematical model supported by comprehensive data preprocessing and analysis.
- ❖ **User-Friendly Interface:**
  - A C# Windows Forms application with the .NET framework, designed for admins to manage tour packages, budgets, and resources.
- ❖ **AI Integration:**
  - An AI prediction model that analyzes historical data to forecast future trends, enabling informed decision-making and optimized operations.

2.4 Existing Work

In this section, we review the existing work and references that have influenced and supported our project. This review highlights previous research and methodologies that have contributed to the optimization of the tourism industry and decision support systems. Each reference is followed by a summary of its relevance and application to our project.

- ❖ **Reference 1:**
  - **Summary:** This book provides comprehensive insights into supply chain management, covering strategy, planning, and operational aspects. It lays the groundwork for understanding how efficient resource allocation and strategic planning can enhance operational performance, which is critical for optimizing tourism operations. The methodologies discussed in this book influenced our approach to developing a mathematical model and implementing efficient resource allocation strategies.

- ❖ **Reference 2:**
  - **Summary:** Taha's textbook is a seminal work in operations research, offering detailed explanations of optimization techniques and mathematical modeling. The concepts of linear programming, integer programming, and other optimization methods were foundational to the creation and refinement of our mathematical model. This reference guided our decision to employ mixed-integer linear programming (MILP) to solve complex optimization problems in the tourism industry.

❖ **Reference 3:**

- **Summary:** This study explores how supply chain strategies can be applied to the tourism industry to boost regional revenue. It emphasizes the importance of strategic resource management and efficient operations. The findings from this research provided a practical perspective on applying supply chain principles to tourism, which informed our approach to optimizing tour packages and resource allocation.

❖ **Reference 4:**

- **Summary:** This paper presents a decision-making framework aimed at enhancing service quality in the tourism industry while operating within budget constraints. It highlights the challenges of balancing service quality with financial limitations. The methodologies proposed in this work inspired our development of a decision support system that helps new tourism businesses make informed, cost-effective decisions.

❖ **Reference 5:**

- **Summary:** Frechtling's work on tourism demand forecasting is crucial for understanding how to predict future trends in tourism. This book provides various forecasting techniques, which we adapted and implemented in our AI prediction model. By applying these techniques, we aimed to accurately forecast demand for tour packages, thereby optimizing resource allocation and improving strategic planning.

❖ **Reference 6:**

- **Summary:** This paper discusses the management of tourism emissions by optimizing the mix of tourism demand. The approach of balancing environmental concerns with economic benefits provided insights into how we could incorporate sustainability metrics into our optimization model, aligning our project with sustainable tourism practices.

❖ **Reference 7:**

- **Summary:** This study presents an optimization model for creating personalized tourism packages. The methodologies and algorithms discussed influenced our approach to developing a customizable and adaptable model for tourism package optimization. It also highlighted the importance of personalization in meeting diverse tourist preferences.

❖ **Reference 8:**

- **Summary:** This paper explores the use of machine learning for managing and optimizing tourism information resources. The techniques and insights from this work helped us enhance our AI prediction model, enabling it to learn from historical data and provide more accurate forecasts for tourism demand.

❖ **Reference 9:**

- **Summary:** This research evaluates regional tourism competitiveness and proposes optimization strategies. The findings emphasized the importance of regional analysis and adaptability in tourism management, guiding us to consider regional factors in our model to ensure its applicability across different geographical locations.

❖ **Reference 10:**

- **Summary:** This overview of AI applications in tourism provided a broad perspective on how AI technologies can enhance various aspects of the tourism industry. The review underscored the potential of AI in demand forecasting, resource management, and decision support, which directly influenced the development of our AI prediction model and decision support framework.

CHAPTER 3

MATHEMATICAL MODEL DEVELOPMENT AND DATA PROCESSING

3.1 Introduction

This chapter outlines the development of the mathematical model and the process of data collection, preprocessing, and analysis. We will discuss the iterative process of creating the model from scratch, testing it on hypothetical data, and refining it to the final version. Subsequently, the chapter will detail the data collection, preprocessing, and analysis steps.

3.2 Preliminary Mathematical Model Development

3.2.1 Initial Model Creation

The initial phase involved creating a preliminary mathematical model from scratch. This model was designed to address the optimization problem within the tourism industry, focusing on strategic decision-making for new tourism businesses. The primary components of the initial model included decision variables, parameters, the objective function, and constraints.

3.2.2 Application on Hypothetical Data

To test the validity of the preliminary model, we applied it to a set of hypothetical data. This step was crucial for identifying potential issues and areas for improvement in the model. The results from this application highlighted several aspects that required refinement.

3.3 Model Refinement and Finalization

3.3.1 Updates to the Mathematical Model

Based on the insights gained from the preliminary model, we made several updates:

- **Addition of Parameters:** New parameters were introduced to better capture the complexities of the tourism industry.
- **Modification of Decision Variables:** The decision variables were adjusted to provide a more accurate representation of the problem.
- **Enhancement of Constraints:** Additional constraints were added to ensure the feasibility and realism of the solutions generated by the model.

3.3.2 Final Mathematical Model

The final version of the mathematical model incorporates all the updates and refinements. This model is robust and capable of providing optimal solutions for strategic decision-making in the tourism industry.

Decision Variables:

- $\boldsymbol{xij}$: Binary variable indicating whether to offer package i in season j

  (1 if offered, 0 otherwise).

Parameters:

- $\boldsymbol{n}$: Number of potential tour packages.
- $\boldsymbol{m}$: Number of seasons.
- $\boldsymbol{ci}$: Cost of designing and operating package i.
- $\boldsymbol{ck}$: Cost per unit of resource k.
- $\boldsymbol{pi}$: Price of package i.
- $\boldsymbol{br}$: Maximum budget for resources (excluding marketing).
- $\boldsymbol{Rk}$: Available amount of resource k.
- $\boldsymbol{rijk}$:Amount of resource allocation for package i in season$j$ for resource k.
- $\boldsymbol{dmin(i)}$: Minimum desired demand for package i.
- $\boldsymbol{dmax(i)}$: Maximum desired demand for package i.
- $\boldsymbol{dij}$: Demand for package i in season j.
- $\boldsymbol{mij}$: Maximum number of packages i to offer per season j.

Objective Function:

$$\boldsymbol{Maximize\ Z} = \sum_{i=1}^{n}\sum_{j=1}^{m}(\boldsymbol{p_i} * \boldsymbol{d_{ij}} * \boldsymbol{x_{ij}}) - \sum_{i=1}^{n}\sum_{j=1}^{m}(\boldsymbol{c_i} * \boldsymbol{x_{ij}}) - \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{r_{ijk}}(\boldsymbol{c\_k} * \boldsymbol{r_{ijk}} * \boldsymbol{x_{ij}})$$

Table 3.3.2 1- Objective Function

Term	Definition
$\sum_{i=1}^{n}\sum_{j=1}^{m}(\boldsymbol{p_i} * \boldsymbol{d_{ij}} * \boldsymbol{x_{ij}})$	the total revenue generated from selling tour packages.
$\sum_{i=1}^{n}\sum_{j=1}^{m}(\boldsymbol{c_i} * \boldsymbol{x_{ij}})$	the total cost of designing and operating each tour package.
$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{r_{ijk}}(\boldsymbol{c\_k} * \boldsymbol{r_{ijk}} * \boldsymbol{x_{ij}})$	the total cost of resources allocated to each package

Constraints :

1. Budget Constraints

1.1 Resource Budget

- **Description**: Ensures that the total cost of allocating resources does not exceed the available budget.

- **Mathematical Representation**:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{r_{ijk}}\left(\boldsymbol{c_k} * \boldsymbol{r_{ijk}} * \boldsymbol{x_{ij}}\right) \leq \boldsymbol{b_r} \quad \forall \boldsymbol{i}, \boldsymbol{j}$$

- **Explanation:** The summation of the cost $\boldsymbol{c_k}$ for each resource $\boldsymbol{k}$, multiplied by the amount $\boldsymbol{r_{ijk}}$ allocated to package $\boldsymbol{x_{ij}}$, must not exceed the budget $\boldsymbol{b_r}$.

2. Resource Constraints

2.1 Resource Demand

- **Description**: Ensures that the demand for each resource $\boldsymbol{k}$ does not exceed the available amount $\boldsymbol{R_k}$.

- **Mathematical Representation**:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} (\boldsymbol{r_{ijk}} * \boldsymbol{x_{ij}}) \leq \boldsymbol{R_k} \quad \forall \boldsymbol{i, j}$$

- **Explanation**: The total resource $\boldsymbol{r_{ijk}}$ allocated to package $\boldsymbol{x_{ij}}$ should not exceed the resource availability $\boldsymbol{R_k}$.

3. Demand Constraints

3.1 Minimum Package Demand

- **Description**: Ensures that the demand for each package is above a minimum threshold.

- **Mathematical Representation**:

$$\boldsymbol{d_{ij} * x_{ij} \geq d_{min(i)}} \quad \forall \boldsymbol{i, j}$$

- **Explanation**: The product of demand $\boldsymbol{d_{ij}}$ and the decision variable $\boldsymbol{x_{ij}}$ must be at least the minimum demand $\boldsymbol{d_{min(i)}}$.

3.2 Maximum Package Demand

- **Description**: Ensures that the demand for each package does not exceed a maximum threshold.

- **Mathematical Representation**:

$$\boldsymbol{d_{ij} * x_{ij} \leq d_{max(i)}} \quad \forall \boldsymbol{i, j}$$

- **Explanation**: The product of demand $\boldsymbol{d_{ij}}$ and the decision variable $\boldsymbol{x_{ij}}$ must be at most the maximum demand $\boldsymbol{d_{max(i)}}$.

4. Package Offering Constraints

4.1 Binary Nature of x_{ij}

- **Description**: Ensures that each package is either offered (1) or not offered (0).

- **Mathematical Representation**:

$$x_{ij} \in \{0,1\}$$

- **Explanation**: The decision variable x_{ij} is binary, meaning it can only take values 0 or 1.

5. Seasonal Constraints

5.1 Maximum Package Offerings per Season

- **Description**: Limits the number of packages that can be offered in a given season.

- **Mathematical Representation**:

$$\sum_{i=1}^{n} x_{ij} \leq m_{ij} \quad \forall i, j$$

- **Explanation**: The total number of packages x_{ij} offered must be less than or equal to the maximum allowed packages m_{ij} per season.

6. Non-negativity constraints

- **Description**: Ensures that all parameters involved in the model are non-negative.

- **Mathematical Representation**:

$$n, m, c_i, c_k, p_i, b_r, R_k, r_{ijk}, d_{min(i)}, d_{max(i)}, d_{ij}, m_{ij} \geq 0$$

- **Explanation**: This constraint ensures that none of the parameters take negative values.

3.4 Data Collection

3.4.1 Data Sources

Data was collected from various sources, including industry reports, historical business records, and market research surveys. These sources provided comprehensive information on tour package costs, resource availability, demand patterns, and pricing structures.

To tackle the challenges of optimizing decision-making for new tourism businesses, we utilized a combination of real-world and hypothetical data.

The real-world data included prices of tours, seasonal variations, types and locations of tour packages, and the duration of trips, sourced from Egypt Tours Portal Company. Due to confidentiality constraints, the companies were unable to share certain critical data points such as the cost of packages per tourist, cost per unit of resources, minimum, actual, and maximum demand, and the number of resources allocated per package. To address this gap, we created hypothetical data to simulate these essential variables.

3.4.2 Data Collection Process

The data collection process involved:

- **Identifying Relevant Data**: Determining the key variables and parameters needed for the model . The types of data used were obtained and virtual data were created to create the model as follows :

    - Tour Name
    - Season
    - Duration
    - Price
    - Cost of Package Per Tourist
    - Cost per unit of resources
    - Minimum Demand
    - Actual Demand
    - Maximum Demand
    - Number of Resources of Package
    - Number of Available Resources

The **Figure 3.4.2 1- Block of data used in model** below shows the format and details of the data used in the model

Figure 3.4.2 1- Block of data used in model

Tour Name	Season	Duration	Price	Cost of Package Per Tourist	Cost per unit of resource	Actual Demand	Minimum Demand	Maximum Demand	#Resources of Package	#Available Resources
Cairo Stopover Tour	Winter	1 Day	$99	$64	$10	40	50	105	15	38
2 Day Trip to Luxor from Hurghada	Winter	2 Days / 1 Night	$210	$148	$10	102	50	105	32	40
Cairo Shore Trip from Port Said & Drop O	Winter	2 Days / 1 Night	$340	$245	$10	109	50	105	9	22
5 Days Nile Cruise from Hurghada	Summer	5 Days / 4 Night	$650	$468	$20	64	50	105	9	22
Egypt Highlights from Hurghada in 2 Nigh	Summer	3 Days/ 2 Nights	$545	$399	$10	85	50	105	12	31
Tour to Cairo, Luxor & Abu Simbel from El	Winter	3 Days / 2 Night	$590	$433	$10	50	50	105	4	40
4 Days Cairo Tour Packages	Winter	4 Days / 3 Night	$430	$313	$10	105	50	105	24	27
5 Days Nile Cruise from Marsa Alam	Summer	5 Days / 4 Night	$699	$504	$20	46	50	105	12	31
6 Days Cairo, Luxor & Aswan Holiday	Winter	6 Days / 5 Night	$885	$644	$20	50	50	105	4	35
7 Days Cairo and Hurghada Holiday	Winter	7 Days / 6 Night	$855	$621	$20	85	33	85	19	21
8 Days Cairo Nile Cruise and Hurghada To	Winter	8 Days / 7 Night	$1,120	$810	$30	57	33	85	10	25
Overnight Trip to Cairo & Luxor from Mar	Winter	2 Days / 1 Night	$480	$350	$10	76	50	105	10	24
9 Days Cairo, Luxor & Hurghada Tour Pac	Winter	9 Days / 8 Night	$1,030	$743	$30	56	33	85	14	34
10 Days Egypt Tour Cairo, Nile Cruise & H	Winter	10 Days / 9 Nigh	$1,310	$953	$30	92	33	85	11	28
Luxury 11 Days Egypt Tour	Winter	11 Days / 10 Nig	$1,440	$1,050	$30	48	33	85	13	33
12 Days Hurghada, Cairo & Nile Cruise Ho	Summer	12 Days / 11 Nig	$1,520	$1,100	$40	50	28	55	26	33
13 Days Alexandria, Cairo and Nile Cruise	Summer	13 Days / 12 Nig	$1,650	$1,198	$40	32	28	55	5	25

- **Gathering Data**: Gathering data in the tourism industry presents significant challenges. Due to the competitive nature of the industry, companies are often reluctant to share sensitive information, such as detailed financials, cost structures, and proprietary demand forecasts, as these are considered trade secrets. Additionally, the dynamic and diverse nature of tourism, with its fluctuating seasonal demand, varied customer preferences, and wide range of service offerings, makes it difficult to obtain consistent and comprehensive data. This scarcity of openly available, detailed data complicates efforts to develop accurate models and decision support frameworks. Consequently, researchers often must rely on a combination of limited real-world data and well-constructed hypothetical data to conduct meaningful analysis and derive actionable insights.

- **Verifying Data**: In our project, ensuring the accuracy and reliability of the collected data was a critical step. Given the challenges in gathering comprehensive data in the tourism industry, it was essential to verify both the real-world and hypothetical data to maintain the integrity of our analysis.

  - **Real-World Data Verification**
    For the real-world data obtained from the Egypt Tours Portal Company, our verification process involved:

    - **Cross-Referencing**: We cross-referenced the provided data with publicly available information and other reputable sources to ensure consistency and accuracy.

- **Validation with Industry Experts**: We consulted with industry professionals to confirm that the data reflected current market trends and practices.

- **Anomaly Detection**: We implemented statistical methods to identify and correct any anomalies or outliers that could skew our analysis.

- **Hypothetical Data Verification**
  Given that we had to generate hypothetical data for certain key variables due to confidentiality constraints, we adopted the following measures:

  - **Logical Consistency Checks**: We ensured that the hypothetical data followed logical patterns and relationships consistent with industry norms. For example, costs per tourist and resource allocation were aligned with typical business models.

  - **Scenario Analysis**: We created multiple scenarios with varying parameters to test the robustness of our hypothetical data. This helped us understand the impact of different assumptions on our analysis outcomes.

  - **Expert Review**: The hypothetical data was reviewed by academic advisors and industry experts to ensure it was plausible and aligned with real-world conditions.

❖ **Ensuring Reliability :** To further ensure the reliability of our data:

- **Reproducibility**: We documented all data preprocessing, transformation, and analysis steps meticulously to ensure that our results could be reproduced by others.

- **Continuous Monitoring**: We continually monitored the data throughout the project to identify any potential issues or changes that could affect our analysis.

By implementing these verification processes, we were able to maintain a high level of accuracy and reliability in our data, which is crucial for developing a trustworthy decision support framework. This rigorous approach ensures that the insights and recommendations generated by our project are both valid and applicable to real-world scenarios in the tourism industry.

3.5 Data Preprocessing

3.5.1 Data Cleaning

Data cleaning involved several steps to address inconsistencies, errors, and irrelevant information in the dataset :

- **Removing Duplicates**: We identified and removed duplicate entries to prevent redundant data from skewing our analysis.

- **Handling Missing Values**: Missing data points were addressed by either imputing values based on statistical methods or excluding incomplete records, depending on the context and importance of the missing data.

- **Correcting Inaccuracies**: Any inaccuracies or errors in the data, such as incorrect dates, misspelled names, or out-of-range values, were corrected through cross-referencing with reliable sources and expert validation.

- **Standardizing Formats**: We standardized data formats for consistency, ensuring that dates, numerical values, and categorical variables were uniform across the dataset.

- **Eliminating Irrelevant Data**: Irrelevant data that did not contribute to the analysis, such as extraneous columns or outdated information, was removed to streamline the dataset.

3.5.2 Data Transformation

Data transformation was necessary to convert the cleaned data into formats suitable for analysis, enhancing its value and usability:

- **Normalization**: We normalized numerical data to a common scale, making it easier to compare and analyze variables with different units or magnitudes.

- **Categorization**: Categorical data was transformed into meaningful groups or categories. For example, tour types were classified into categories such as adventure, cultural, and leisure tours.

- **Aggregation**: Data was aggregated to create summary statistics, such as average tour prices per season or total resource costs per package, providing a higher-level view of key metrics.

- **Feature Engineering**: New features were created based on existing data to enhance the analysis. For instance, we derived metrics like the cost per tourist per day or the average duration of tours in different seasons.

- **Data Integration**: Real-world data and hypothetical data were integrated seamlessly, ensuring that all relevant variables were included and properly aligned for comprehensive analysis.

- **Temporal Adjustments**: Data was adjusted for temporal factors, such as converting seasonal data into consistent time frames to analyze trends and patterns over time.

3.6 Data Analysis

Data analysis was essential for understanding the underlying patterns, trends, and relationships within the dataset. The analysis aimed to extract actionable insights that would inform strategic decisions for optimizing resource allocation, tour package selection, and pricing strategies. Our dataset, compiled from real-world data provided by Egypt Tours Portal Company and augmented with hypothetical data, included various attributes such as tour prices, seasons, package types, destinations, and trip durations.

3.6.1 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was a critical part of our data analysis process, helping us to explore the dataset in detail, uncover underlying structures, and identify potential issues before applying more sophisticated modeling techniques. The main objectives of EDA were to:

- **Understand the Data Distribution**: We visualized the distribution of key variables to identify normality, skewness, and the presence of outliers.

- **Identify Relationships**: Through scatter plots, correlation matrices, and pair plots, we explored the relationships between variables, such as the impact of pricing on demand.

- **Detect Anomalies**: We identified outliers and anomalies that could distort the analysis and addressed them through data cleaning techniques.

- **Generate Hypotheses**: EDA helped us generate hypotheses about potential factors influencing tour success, which could be tested in subsequent analysis.

- **Descriptive Statistics**: Summarizing the central tendencies and dispersions. Here we talk about the prices of flights for each season, The following descriptive statistics provide an overview of the pricing for tour packages across different seasons: Autumn, Spring, and Summer. Each table includes key measures that help us understand the distribution and characteristics of tour prices for each season

Table 3.6.1 1- Descriptive Statistics for Price of Packages in Autumn Season

Price of packages in Autumn Season	
Mean	857.3562654
Median	735
Mode	99
Standard Deviation	671.5286057
Skewness	0.876436736
Range	3135
Minimum	15
Maximum	3150
Sum	348944
Count	407

- **Explanation**:
  - **Mean**: The average price of tour packages in Autumn is approximately $857.36.
  - **Median**: The median price is $735, indicating that half of the tour packages are priced below this value and half above.
  - **Mode**: The most frequent price point is $99.
  - **Standard Deviation**: A high standard deviation of $671.53 suggests significant variability in prices.
  - **Skewness**: A skewness of 0.88 indicates a right-skewed distribution, meaning there are more lower-priced packages with a few higher-priced outliers.

- **Range**: The range of $3135 shows the difference between the highest and lowest prices.
- **Minimum and Maximum**: The lowest price is $15, and the highest is $3150.
- **Count**: There are 407 tour packages recorded for the Autumn season.

Table 3.6.1 2- Descriptive Statistics for Price of Packages in Spring Season

Price of packages in Spring Season	
Mean	857.0771028
Median	752.5
Mode	99
Standard Deviation	666.7595876
Skewness	0.867516526
Range	3135
Minimum	15
Maximum	3150
Sum	366829
Count	428

- **Explanation**:
  - **Mean**: The average price of tour packages in Spring is approximately $857.08.
  - **Median**: The median price is $752.5.
  - **Mode**: The most frequent price point is $99.
  - **Standard Deviation**: A standard deviation of $666.76 indicates substantial price variability.
  - **Skewness**: A skewness of 0.87, similar to Autumn, shows a right-skewed distribution.
  - **Range**: The range remains $3135, highlighting significant price differences.
  - **Minimum and Maximum**: Prices range from $15 to $3150.
  - **Count**: There are 428 tour packages recorded for the Spring season.

Table 3.6.1 3- Descriptive Statistics for Price of Packages in Summer Season

Price of packages in Summer Season	
Mean	922.8933718
Median	850
Mode	99
Standard Deviation	673.7467515
Skewness	0.737913494
Range	3135
Minimum	15
Maximum	3150
Sum	320244
Count	347

- **Explanation**:
  - **Mean**: The average price of tour packages in Summer is approximately $922.89, higher than in Autumn and Spring.
  - **Median**: The median price is $850, suggesting a higher central tendency compared to other seasons.
  - **Mode**: The most frequent price point is $99.
  - **Standard Deviation**: A standard deviation of $673.75, indicating price variability similar to the other seasons.
  - **Skewness**: A lower skewness of 0.74 indicates a slightly less right-skewed distribution compared to Autumn and Spring.
  - **Range**: The range of $3135 continues to show significant price differences.
  - **Minimum and Maximum**: Prices also range from $15 to $3150.
  - **Count**: There are 347 tour packages recorded for the summer season.

Table 3.6.1 4- Descriptive Statistics for Price of Packages in Winter Season

Price of packages in Winter Season	
Mean	855.6083916
Median	720
Mode	99
Standard Deviation	678.1395147
Skewness	0.905595433
Range	3135
Minimum	15
Maximum	3150
Sum	367056
Count	429

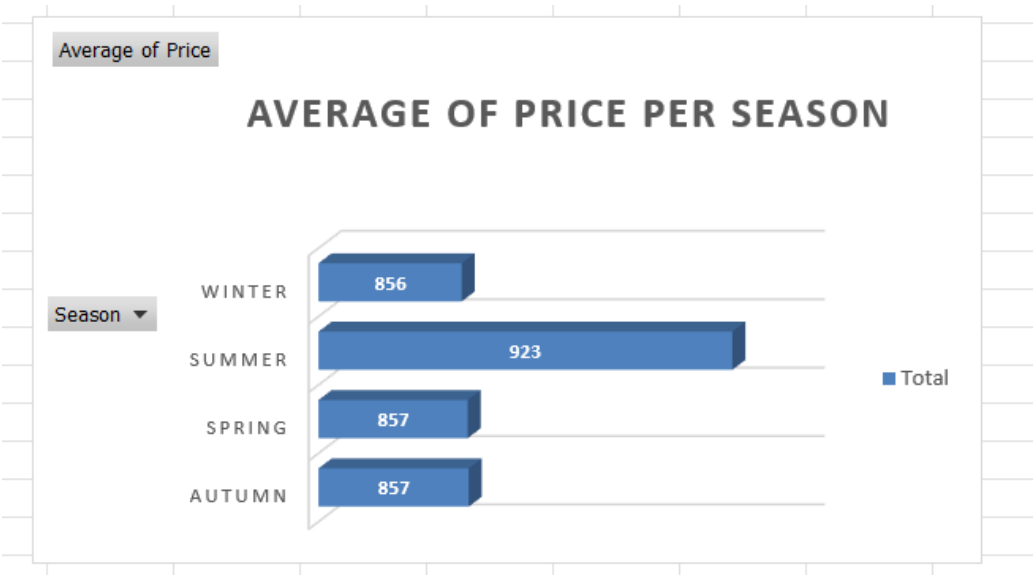
- **Explanation**
  - **Mean:** The average price of tour packages in Winter is approximately $855.61.
  - **Median:** The median price is $735, indicating that half of the tour packages are priced below this value and half above.
  - **Mode**: The most frequent price point is $99.
  - **Standard Deviation**: A high standard deviation of $678.14 suggests significant variability in prices.
  - **Skewness**: A skewness of 0.91 indicates a right-skewed distribution, meaning there are more lower-priced packages with a few higher-priced outliers.
  - **Range**: The range of $3135 shows the difference between the highest and lowest prices.
  - **Minimum and Maximum**: The lowest price is $15, and the highest is $3150
  - **Count**: There are 429 tour packages recorded for the winter season.

❖ **Visualization**: Creating charts and graphs to visualize data distributions and relationships.

The **Figure 3.6.1 1- Average of Price per Season** below explains the bar chart displays the average price of tour packages across different seasons.

- **Spring and Autumn**: Both seasons have an identical average price of $857.
- **Winter**: The average price slightly decreases to $856.
- **Summer**: The highest average price is observed in Summer at $923.

Figure 3.6.1 1- Average of Price per Season



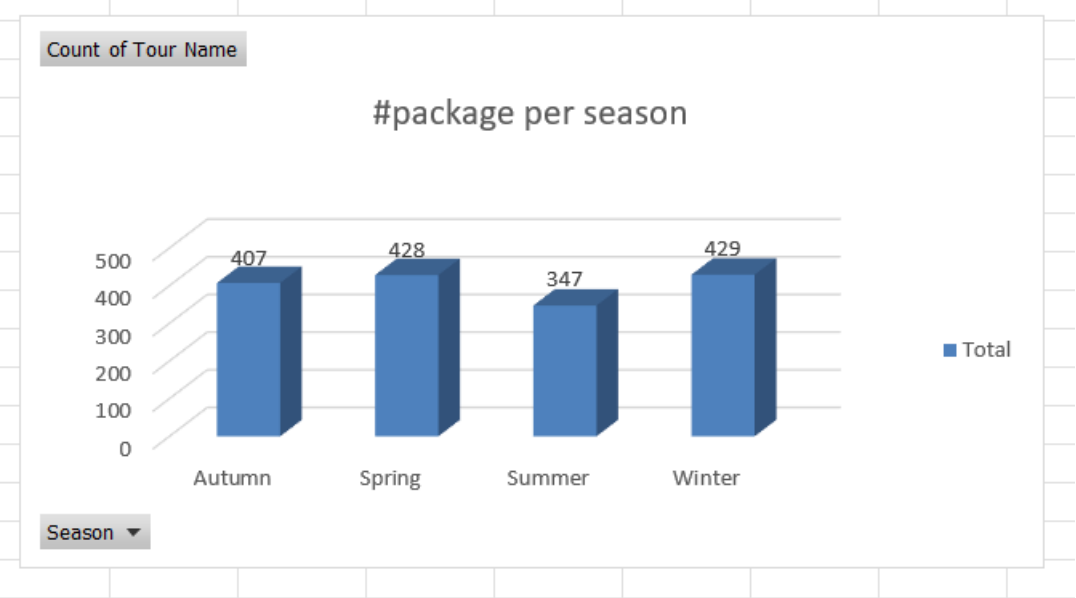
Insights:

- Summer commands a higher average price, likely due to peak travel demand and higher tourism activities.
- Spring and Autumn maintain similar pricing, possibly reflecting balanced demand and supply.
- Winter shows a slight decrease, suggesting it might be a less popular travel season compared to Summer.

The **Figure 3.6.1 2- Number of packages per season** below explains the bar chart shows the count of tour packages available in each season

- **Winter**: The highest number of tour packages (429).
- **Spring**: The second highest with 428 packages.
- **Autumn**: Offers 407 packages.
- **Summer**: The least number of packages at 347.

Figure 3.6.1 2- Number of packages per season



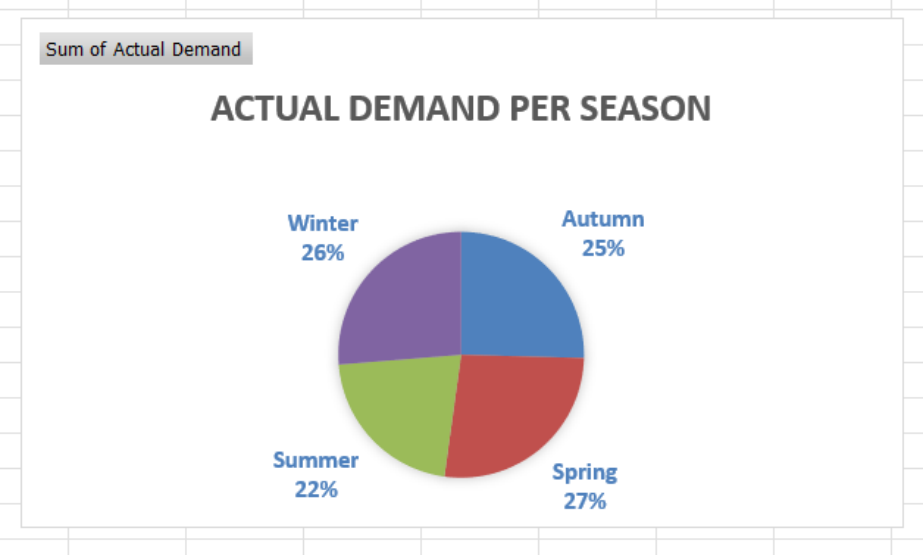
Insights:

- **Winter and Spring**: The high number of packages suggests that these seasons have significant offerings, catering to various customer preferences.
- **Summer**: Despite having the highest average price, Summer has the lowest number of packages, indicating limited availability but higher pricing strategies.

The **Figure 3.6.1 3- Actual Demand per Season** below explains the pie chart represents the actual demand distribution for tour packages across seasons.

- **Spring**: The highest demand at 27%.
- **Winter**: Follows closely with 26%.
- **Autumn**: Accounts for 25% of the demand.
- **Summer**: The lowest demand at 22%.

Figure 3.6.1 3- Actual Demand per Season



Insights:

- **Spring and Winter**: These seasons show the highest demand, aligning with the higher number of available packages.
- **Summer**: Despite having the highest average price, it has the lowest demand, which could be due to fewer package offerings and higher prices.
- **Autumn**: Shows balanced demand, reflecting its moderate pricing and package availability.

The charts provide a comprehensive view of the pricing and demand dynamics across different seasons in the tourism industry. Key observations include:

- **Price Dynamics**: Summer has the highest average prices, likely due to high demand and limited package availability.
- **Package Availability**: Winter and Spring have the highest number of packages, catering to their higher demand.
- **Demand Distribution**: Spring shows the highest demand, closely followed by Winter, while Summer has the lowest demand despite its higher prices.

These insights help new tourism businesses understand seasonal trends, allowing them to strategically plan their offerings and pricing to maximize profitability and meet customer expectations.

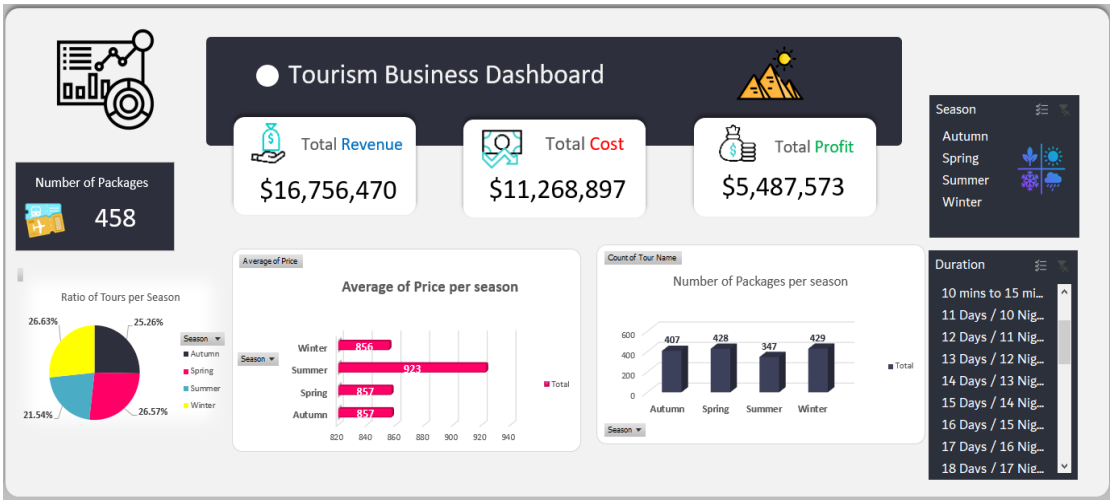
3.6.2 Creating Dashboards

A comprehensive dashboard was developed using Excel to present the analyzed data in an interactive and user-friendly format. The dashboard included:

- **Pivot Tables**: Summarizing large datasets to show trends and patterns.
- **Charts**: Visual representations of key metrics and indicators.

The **Figure 3.6.2 1- Dashboard of Data** below shows the dashboard that was created to summarize data and information in a clear, easy and comprehensive way for the user and is a summary of pivot tables and pivot charts.

Figure 3.6.2 1- Dashboard of Data



CHAPTER 4

APPLYING DIFFERENT APPROACHES TO SOLVE THE MODEL

4.1 Introduction

In the field of optimization, choosing the right approach to solve a model is crucial for achieving efficient and effective results. Two main categories of approaches are often considered: exact methods and metaheuristic methods. Each of these has its own Advantages and Disadvantages, making them suitable for different types of problems and objectives.

4.2 Difference between Exact and Metaheuristic Solution's

4.2.1 Exact Solution

- **Definition:** An exact solution method finds the optimal solution to a problem with a guarantee of accuracy. These methods explore all possible solutions to ensure the best one is found.
- **Common Techniques:**
  - **Linear Programming (LP):** Used for problems where the objective function and constraints are linear.
  - **Integer Programming (IP):** Similar to LP but some or all decision variables are restricted to integer values.
  - **Mixed-Integer Linear Programming (MILP):** Combines aspects of LP and IP, where some variables are continuous and some are integers.
  - **Dynamic Programming (DP):** Solves problems by breaking them down into simpler subproblems and solving each subproblem only once.
  - **Branch and Bound**: Systematically explores branches of solution spaces to find the optimal solution.
- **Advantages:**
  - Guarantees finding the optimal solution.
  - Provides a definitive measure of solution quality.
- **Disadvantages:**
  - Computationally expensive for large-scale problems.
  - May be impractical for very complex or large problems due to time and resource constraints.

4.2.2 Metaheuristic Solution

- **Definition:** Metaheuristic methods are high-level problem-independent algorithmic frameworks that provide near-optimal solutions by exploring the solution space more efficiently.

- **Common Techniques:**

    - **Genetic Algorithms (GA):** Inspired by natural selection, uses crossover, mutation, and selection operators.

    - **Ant Colony Optimization (ACO):** Inspired by the foraging behavior of ants, uses pheromone trails to guide the search.

- **Advantages:**
    - Capable of finding good solutions within reasonable time frames.
    - Flexible and can be adapted to various types of problems.

- **Disadvantages:**
    - No guarantee of finding the optimal solution.
    - Solution quality may vary depending on the problem and specific implementation.

4.3 Apply exact solution approach to solve the model

The Technique of mathematical model we create is a **Mixed-Integer Linear Programming (MILP)** model then we applied the exact solution approach using Mixed-Integer Linear Programming (MILP) to solve the tour package optimization model with the provided Python code.

We developed a **Mixed-Integer Linear Programming (MILP)** model to determine the optimal tour packages to offer in each season to maximize profit while adhering to budget, resource, and demand constraints. This **detailed explanation** walks through the implementation of the model using the **PuLP** library to formulate and solve the optimization problem , The PuLP library efficiently handles the optimization process, providing valuable insights for decision-making in tour package management.

4.3.1 Detailed Explanation

- **Import Libraries**:

```
import pulp
import pandas as pd
import locale
```

  - **pulp**: A library for linear programming.
  - **pandas**: Used for data manipulation and analysis.
  - **locale**: Used for currency formatting.

- **Load Data**:

```
file_path = 'C:/Users/Mohammed/OneDrive/Desktop/use data.xlsx' data =
pd.read_excel(file_path, sheet_name='Data')
```

  - Load the Excel file containing the necessary data for the model.

- **Extract Unique Values**:

```
tour_packages = data['Tour Name'].unique()
seasons = data['Season'].unique()
```

  - Extract unique tour packages and seasons from the data.

- **Create Indices**:

```
tour_package_indices = {tour: idx for idx, tour in enumerate(tour_pack-
ages)} season_indices = {season: idx for idx, season in enumerate(sea-
sons)}
```

  - Create dictionaries to map tour packages and seasons to integer indices.

❖ **Define Parameters**:

```
c_i = data['Cost of Package Per Tourist'].tolist()
c_k = data['Cost per unit of resources'].unique().tolist()
p_i = data['Price'].tolist()
b_r = 1000000  # Maximum budget for resources
R_k = data['#Available Resources'].unique().tolist()
d_min = data['Minimum Demand'].tolist()
d_max = data['Maximum Demand'].tolist()
```

- Define various parameters such as costs, prices, budgets, resources, and demand constraints.

❖ **Create Demand Pivot Table**:

```
demand_pivot = data.pivot(index='Tour Name', columns='Season', values='Actual Demand') d_ij = demand_pivot.fillna(0).values.tolist()
```

- Create a pivot table for actual demand and convert it to a list, filling missing values with zero.

❖ **Initialize Resource Usage Matrix**:

```
r_ijk = [[[0 for _ in range(len(c_k))] for _ in range(m)] for _ in range(n)]
```

- Initialize a 3-dimensional list for resource usage.

❖ **Fill Resource Usage Matrix**:

```
for row in data.itertuples():
    i = tour_package_indices[row._1]
    j = season_indices[row.Season]
    for k in range(len(c_k)):
        r_ijk[i][j][k] = row._8  # Adjust this index based on dataframe structure
```

- Populate the resource usage matrix r_ijk with values from the data.

❖ **Define Decision Variables**:

```
x = pulp.LpVariable.dicts("x", ((i, j) for i in range(n) for j in range(m)),
cat='Binary')
```

- Define binary decision variables x[i, j] indicating whether a tour package i is offered in season j.

❖ **Objective Function**:

```
objective = pulp.lpSum(p_i[i] * d_ij[i][j] * x[i, j] for i in range(n) for j in
range(m)) - \
        pulp.lpSum(c_i[i] * d_ij[i][j] * x[i, j] for i in range(n) for j in
range(m)) - \
        pulp.lpSum(c_k[k] * r_ijk[i][j][k] * x[i, j] for i in range(n) for j in
range(m) for in range(len(c_k)))
```

- Define the objective function to maximize profit (revenue minus costs).

❖ **Create Optimization Problem**:

```
prob = pulp.LpProblem("TourPackageOptimization", pulp.LpMaximize)
prob += objective
```

- Create the MILP problem and set the objective function.

❖ **Add Constraints**:

- **Budget Constraints:**

```
for i in range(n):
    for j in range(m):
        prob += pulp.lpSum(c_k[k] * r_ijk[i][j][k] * x[i, j] for k in
range(len(c_k))) <= b_r
```

- **Resource Constraints**:

```
for k in range(len(c_k)):
    for j in range(m):
        prob += pulp.lpSum(r_ijk[i][j][k] * x[i, j] for i in range(n)) <=
R_k[k]
```

- **Demand Constraints**:

```
for i in range(n):
    for j in range(m):
        prob += d_ij[i][j] * x[i, j] >= d_min[i]
        prob += d_ij[i][j] * x[i, j] <= d_max[i]
```

- **Package Offering Constraints**:

```
for j in range(m):
    prob += pulp.lpSum(x[i, j] for i in range(n)) <= m_ij[j]
```

- **Non-existent Season Constraints**:

```
for i in range(n):
    for j in range(m):
        if demand_pivot.at[tour_packages[i], seasons[j]] == 0:
            prob += x[i, j] == 0
```

- Add constraints to the model for budget, resources, demand, package offering, and non-existent seasons.

❖ **Solve the Problem**:

```
prob.solve()
```

- Solve the MILP problem using the PuLP solver.

❖ **Calculate Results**:

- **Initialize Totals**:

```
total_revenue = 0
total_cost = 0
total_package_cost = 0
total_resource_cost = 0
```

- **Compute Totals**:

```
for i in range(n):
    for j in range(m):
        if pulp.value(x[i, j]) == 1:
            total_revenue += p_i[i] * d_ij[i][j]
            total_package_cost += c_i[i] * d_ij[i][j]
            total_resource_cost += sum(c_k[k] * r_ijk[i][j][k] *
d_ij[i][j] for k in range(len(c_k)))
total_cost = total_package_cost + total_resource_cost
profit = total_revenue - total_cost
```

- Calculate the total revenue, total cost (package and resource costs), and profit.

❖ **Print Offered Packages:**

```
offered_packages = []
for i in range(n):
    for j in range(m):
        if pulp.value(x[i, j]) == 1:
            offered_packages.append((tour_packages[i], seasons[j]))

for package, season in offered_packages:
    print('-------------------------------------------------')
    print()
    print(f"Package: {package} --> in season: {season} --> is offered.")
```

- Identify and print the tour packages that are offered.

❖ **Print Summary**:

```
num_offered_packages = len(offered_packages)
print()
print('-------------------------------------------------')
print(f"Number of packages offered: {num_offered_packages}")
print('-------------------------------------------------')

objective_value = int(pulp.value(prob.objective))
print(f"Objective value: {(locale.currency(objective_value, group-
ing=True))}")
```

- Print the number of packages offered and the objective value in currency format.

4.3.2 The Output

```
Package: 5 Days Nile Cruise from Sharm El Sheikh --> in season: Summer --> is
offered.

--------------------------------------------------------------------------------

Package: 13 Days Alexandria, Cairo and Nile Cruise Holidays --> in season: Au-
tumn --> is offered.

--------------------------------------------------------------------------------

Number of packages offered: 200

--------------------------------------------------------------------------------

Objective value: $1,950,843
```

4.4 Apply Metaheuristic Solution approach on a hypothetical data to solve the model

Metaheuristic algorithms are a class of optimization algorithms that are designed to find approximate solutions to complex problems. They are particularly useful for solving problems where the search space is large and traditional optimization methods are not feasible. In this project, we applied two metaheuristic algorithms Genetic Algorithm (GA) and Ant Colony Optimization (ACO) to optimize the selection and scheduling of tour packages over different seasons.

4.4.1 Metaheuristic Algorithms Techniques (GA , AC)

❖ **Genetic Algorithm (GA)**

- **Initialization**: A population of possible solutions (tour package offerings across seasons) is randomly generated.

```
def initialize_population():
    return np.random.randint(2, size=(population_size, n, m))
```

- **Evaluation**: Each solution in the population is evaluated based on a fitness function that considers the profit and constraints of the tour packages.

```
def evaluate_population(population):
    fitness_values = []
    for i in range(population_size):
        fitness_values.append(evaluate_solution(population[i]))
    return np.array(fitness_values)
```

- **Selection**: Pairs of solutions (parents) are selected from the population based on their fitness scores.

```
def select_parents(population, fitness_values):
    parents = np.empty((2, n, m), dtype=int)
    for i in range(2):
        parent_index = np.random.choice(range(population_size))
        parents[i] = population[parent_index]
    return parents
```

- **Crossover**: New solutions (children) are generated by combining parts of the parent solutions.

```
def crossover(parents):
    crossover_point = np.random.randint(1, n)  # crossover point
      along the packages
    child1 = np.concatenate((parents[0][:crossover_point], par-
      ents[1][crossover_point:]), axis=0)
    child2 = np.concatenate((parents[1][:crossover_point], par-
      ents[0][crossover_point:]), axis=0)
    return child1, child2
```

- **Mutation**: Random changes are introduced to the children to maintain diversity in the population.

```
def mutate(child):
    for i in range(n):
        for j in range(m):
            if random.random() < mutation_rate:
                child[i][j] = 1 - child[i][j]
    return child
```

- **Replacement**: The new generation of solutions replaces the old generation.

```
new_population = np.empty((population_size, n, m), dtype=int)
for i in range(0, population_size, 2):
    parents = select_parents(population, fitness_values)
    child1, child2 = crossover(parents)
    new_population[i] = mutate(child1)
    if i + 1 < population_size:
        new_population[i + 1] = mutate(child2)

population = new_population
```

- **Iteration**: Steps 2-6 are repeated for a specified number of generations or until convergence

```
for gen in range(generations):
        fitness_values = evaluate_population(population)
        generation_fitness.append(np.max(fitness_values))

        if np.max(fitness_values) > best_fitness:
            best_solution = population[np.argmax(fitness_values)]
            best_fitness = np.max(fitness_values)

end_time = time.time()
    convergence_time = end_time - start_time

    return best_solution, generation_fitness, best_fitness, conver-
      gence_time
```

- **Output of Genetic Algorithm:**

  Best Fitness Value Achieved: $602,950

  Time Taken for Convergence (seconds): 0.8412742614746094

❖ **Ant Colony Optimization (ACO)**

- **Initialization**: A pheromone matrix is initialized to represent the desirability of different solutions.

```
def select_next_package(pheromone_values,     heuristic_values,
    selected_packages):
  probabilities = pheromone_values * heuristic_values
  probabilities[selected_packages == 1] = 0
  if np.sum(probabilities) == 0:
     return random.randint(0, n - 1)
  probabilities /= np.sum(probabilities)
  return np.random.choice(np.arange(n), p=probabilities)
```

- **Solution Construction**: Each ant constructs a solution based on the pheromone levels and a heuristic function.

```
def construct_solution():
  selected_packages = np.zeros((n, m), dtype=int)
  season_offerings_count = np.zeros(m, dtype=int)
  for ant in range(num_ants):
     for season in range(m):
        if season_offerings_count[season] < MaxOfferingsPer-
    Season:
           probability_threshold = 0.8
           if random.random() < probability_threshold:
              current_package = select_next_package(phero-
    mone_matrix[:, season], 1 / (c_i + 1), selected_packages[:, sea-
    son])
              if selected_packages[current_package, season] =0:
                 selected_packages[current_package, season] = 1
                 season_offerings_count[season] += 1
        else:
           break
  return selected_packages

def evaluate_solution(selected_packages):
  total_profit = np.sum(p_i * np.sum(d_i * selected_packages,
    axis=1))
  total_cost = np.sum(np.dot(r_ik.T, selected_packages.T) * c_k)
  return total_profit - total_cost
```

- **Pheromone Update**: Pheromone levels are updated based on the quality of the solutions found.

```
def ant_colony_optimization():
    global pheromone_matrix  # Declare pheromone_matrix as
     global

    start_time = time.time()
    best_solution = None
    best_fitness = float('-inf')
    generation_fitness = []
    avg_fitnesses = []
```

- **Iteration**: Steps 2-3 are repeated for a specified number of iterations or until convergence.

```
for iteration in range(iterations_aco):
        solutions = [construct_solution() for _ in range(num_ants)]
        solutions = [seasonal_package_offerings(solution) for solution
    in solutions]

        fitness_values = [evaluate_solution(solution) for solution in
    solutions]
        generation_fitness.append(fitness_values)

        if max(fitness_values) > best_fitness:
            best_solution = solutions[fitness_values.index(max(fit-
    ness_values))]
            best_fitness = max(fitness_values)

        pheromone_matrix *= (1 - pheromone_decay)
        for ant in range(num_ants):
            for season in range(m):
                for package in range(n):
                    if solutions[ant][package, season] == 1:
                        pheromone_matrix[package, season] += phero-
    mone_constant / fitness_values[ant]

        avg_fitness = sum(fitness_values) / num_ants
        avg_fitnesses.append(avg_fitness)

    end_time = time.time()
    convergence_time = end_time - start_time

    return  best_solution,  generation_fitness,  best_fitness,  avg_fit-
     nesses, convergence_time
```

- **Output of Ant Colony Optimization:**

  Best Fitness Value Achieved: $564,800

  Time Taken for Convergence (seconds): 13.717854499816895

4.4.2 Analysis of Metaheuristic Algorithms' Performance

Compares the convergence of two metaheuristic algorithms Genetic Algorithm (GA) and Ant Colony Optimization (ACO) over 100 iterations. Here's a detailed explanation of the results and the graph.

❖ **Genetic Algorithm (GA)**

- **Best Fitness Value Achieved**: $602,950
- **Time Taken for Convergence**: 0.84 seconds

GA shows a high degree of fluctuation in the fitness values across the iterations. The best fitness value achieved is significantly higher than that of ACO, indicating that GA managed to find a highly optimal solution. However, the lack of smooth convergence suggests that the algorithm explores a broad search space and can jump to higher fitness values sporadically

❖ **Ant Colony Optimization (ACO)**

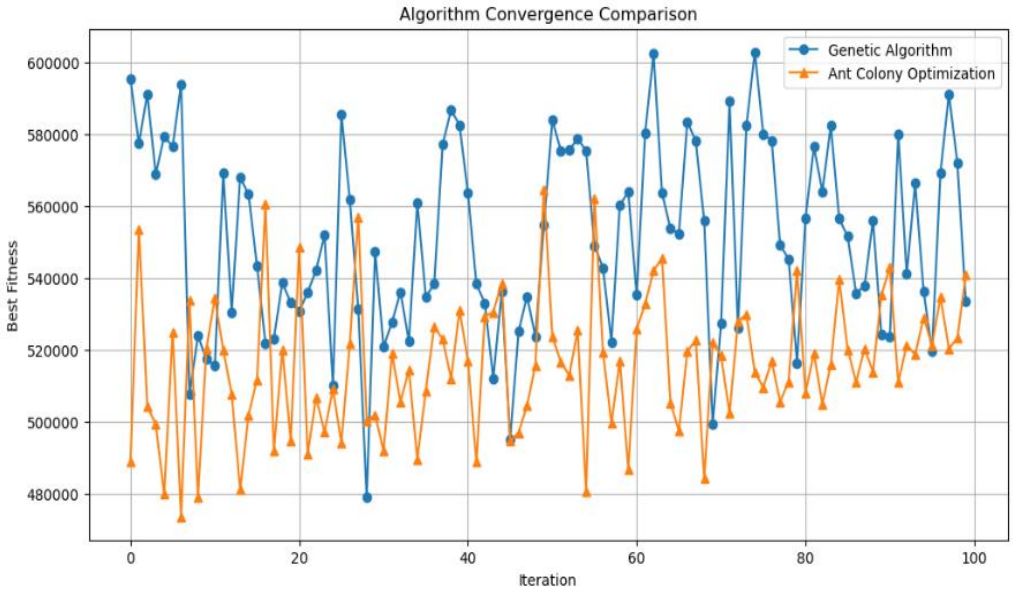
- **Best Fitness Value Achieved**: $564,800
- **Time Taken for Convergence**: 13.72 seconds

ACO is less erratic than GA. It converges more slowly, taking the longest time among the two algorithms. However, is still not as high as GA. This indicates that ACO balances exploration and exploitation but requires more time to do so.

Figure 4.4.2 1- Comparison between(GA,AC) show strengths and weaknesses of each algorithm:

- **Genetic Algorithm (GA)**: High potential for finding the optimal solution but with significant variability and less predictability in convergence.
- **Ant Colony Optimization (ACO)**: A balanced approach with steady improvement, though it requires more time to converge.

Figure 4.4.2 1- Comparison between(GA,AC)



After thorough analysis, we concluded that the choice of algorithm should depend on the specific needs of the problem, such as the importance of achieving the highest fitness value and the speed of convergence. For this problem, the Genetic Algorithm (GA) demonstrated superior performance by achieving the highest fitness value with faster convergence compared to the Ant Colony Optimization (ACO) algorithm, which offered slower convergence and lower fitness values. Thus, GA emerged as the optimal choice for its efficiency in both solution quality and convergence speed.

❖ **Recommendation:**

- If the primary goal is to achieve the best possible fitness value and time is a constraint, go with **Genetic Algorithm (GA)**.
- For a balanced approach where both fitness value and convergence stability are important, **Ant Colony Optimization (ACO)** is the best option.

4.5 Optimizing Tour Packages Using Genetic Algorithm

The aim is to optimize tour packages offered by our travel agency across different seasons to maximize profitability while managing resources efficiently. After evaluating various optimization algorithms, we chose the Genetic Algorithm (GA) due to its superior performance in achieving high fitness values.

4.5.1 Implementation of the Genetic Algorithm to solve our optimization problem

The code is divided into several parts, each handling specific tasks in the optimization process using the Genetic Algorithm (GA).

❖ **Importing Required Libraries**

```
import pandas as pd
import shutil
import tempfile
import os
import numpy as np
```

- pandas for data manipulation and analysis.
- shutil, tempfile, and os for file handling.
- numpy for numerical computations.

❖ **Loading and Reading the Excel Data**

```
def load_excel_data(file_path, sheet_name):
    temp_file = tempfile.NamedTemporaryFile(delete=False,
suffix='.xlsx')
    temp_file_path = temp_file.name
    temp_file.close()
    shutil.copy(file_path, temp_file_path)
    with pd.ExcelFile(temp_file_path) as xls:
        sheet_names = xls.sheet_names
        print(f"Sheet names: {sheet_names}")
        data_df = pd.read_excel(xls, sheet_name)
    os.remove(temp_file_path)
    return data_df

file_path = 'C:/Users/Mohammed/OneDrive/Desktop/use
data.xlsx'
data_df = load_excel_data(file_path, 'Data')
print(data_df.head())
```

- This section defines a function load_excel_data to load and read data from an Excel file. It uses a temporary file to ensure the original file is not altered. The data is loaded into a pandas DataFrame for further processing. The file_path specifies the location of the Excel file, and sheet_name specifies the sheet to be read.

❖ **Creating Dictionaries for Parameters and Variables**

```
cost_per_package = {}
price_per_package = {}
demand = {}
resource_allocation = {}
min_demand = {}
max_demand = {}
max_packages_per_season = {}
cost_per_unit_of_resources = {}

for package in tour_packages:
    package_data = data_df[data_df['Tour Name'] == pack-
age]
    cost_per_package[package] = package_data.iloc[0]['Cost
of Package Per Tourist']
    price_per_package[package] = pack-
age_data.iloc[0]['Price']
    min_demand[package] = package_data.iloc[0]['Minimum
Demand']
    max_demand[package] = package_data.iloc[0]['Maxi-
mum Demand']
    for season in seasons:
        season_data = package_data[package_data['Season']
== season]
        if not season_data.empty:
            demand[(package, season)] = sea-
son_data.iloc[0]['Actual Demand']
            resource_allocation[(package, season)] = sea-
son_data.iloc[0]['#Resources of Package']
            max_packages_per_season[season] = sea-
son_data.iloc[0]['#Available Resources']
            cost_per_unit_of_resources[season]=sea-
son_data.iloc[0]['Cost per unit of resources']

# Display the extracted data for verification
(cost_per_package,price_per_package,min_demand,
max_demand, demand, resource_allocation, max_pack-
ages_per_season)
```

- This section initializes dictionaries to store parameters and variables related to tour packages and seasons. The data is extracted from the DataFrame and stored in the respective dictionaries.

❖ **Genetic Algorithm Parameters and Population Initialization**

```
population_size = 50
generations = 100
mutation_rate = 0.01
tournament_size = 5
elitism_size = 2  # Number of top individuals to keep

def initialize_population(size, n, m):
    return np.random.randint(2, size=(size, n, m))

population=initialize_population(population_size,
len(tour_packages), len(seasons))
```

- This section sets the parameters for the genetic algorithm and defines a function initialize_population to create an initial population of random binary matrices. Each matrix represents whether a tour package is offered in a particular season.

❖ **Selection Function**

```
def tournament_selection(population, fitness_scores, tourna-
ment_size):
    selected_indices = np.random.choice(len(population), tourna-
ment_size, replace=False)
    selected = population[selected_indices]
    selected_fitness = fitness_scores[selected_indices]
    winner_index = np.argmax(selected_fitness)
    return selected[winner_index]
```

- **tournament_selection** selects parents using tournament selection.

❖ **Crossover Function**

```
def crossover(parent1, parent2):
    crossover_point = np.random.randint(parent1.shape[1])
    child1 = np.hstack((parent1[:, :crossover_point], parent2[:, crossover_point:]))
    child2 = np.hstack((parent2[:, :crossover_point], parent1[:, crossover_point:]))
    return child1, child2
```

- **crossover** performs crossover between two parents to produce children.

❖ **Mutation Function**

```
def mutate(individual, mutation_rate):
    for i in range(individual.shape[0]):
        for j in range(individual.shape[1]):
            if np.random.rand() < mutation_rate:
                individual[i][j] = 1 - individual[i][j]
```

- **mutate** introduces mutations to the individuals.

❖ **Fitness Function**

```
def calculate_fitness(individual, cost_per_package,
price_per_package, demand, resource_allocation, max_pack-
ages_per_season):
    total_revenue = 0
    total_cost = 0
    total_resource_cost = 0
    total_resource_used = {}

    for i in range(individual.shape[0]):
        for j in range(individual.shape[1]):
            if individual[i][j] == 1:
                package = tour_packages[i]
                season = seasons[j]
                demand_value = demand.get((package, season), 0)
                resource_value = resource_allocation.get((package, sea-
son), 0)
                total_revenue += price_per_package[package] * de-
mand_value
                total_cost += cost_per_package[package] * de-
mand_value
                total_resource_cost += cost_per_unit_of_resources[sea-
son] * resource_value
                if season in total_resource_used:
                    total_resource_used[season] += resource_value
                else:
                    total_resource_used[season] = resource_value

    for season in total_resource_used:
        if total_resource_used[season] > max_packages_per_sea-
son[season]:
            total_resource_cost += (total_resource_used[season] -
max_packages_per_season[season]) * 1000

    fitness = total_revenue - total_cost - total_resource_cost
    return fitness
```

- This section defines the calculate_fitness function, which evaluates the fitness of an individual by calculating the total revenue, total cost, and total resource cost. It also includes penalties for exceeding resource limits.

❖ **Main Loop for Genetic Algorithm**

```
for generation in range(generations):
    fitness_scores = np.array([calculate_fitness(ind, cost_per_package, price_per_package, demand, resource_allocation, max_packages_per_season) for ind in population])
    new_population = []
    for _ in range(population_size // 2):
        parent1 = tournament_selection(population, fitness_scores, tournament_size)
        parent2 = tournament_selection(population, fitness_scores, tournament_size)
        child1, child2 = crossover(parent1, parent2)
        mutate(child1, mutation_rate)
        mutate(child2, mutation_rate)
        new_population.append(child1)
        new_population.append(child2)

    population = np.array(new_population)
    best_fitness = np.max(fitness_scores)
    print(f'Generation {generation}: Best Fitness = ${int(best_fitness):,}')
```

- This section contains the main loop of the genetic algorithm. It iterates through the specified number of generations, calculates fitness scores, performs selection, crossover, and mutation, and evolves the population. The best fitness value for each generation is printed.

❖ **Displaying the Final Best Solution**

```
print("\nFinal Best Solution:")
print(f'Best Fitness: ${int(best_fitness):,}\n')
best_solution = population[np.argmax(fitness_scores)]
for i in range(best_solution.shape[0]):
    tour_name = tour_packages[i]
    offered_seasons = [seasons[j] for j in range(best_solution.shape[1]) if best_solution[i][j] == 1]
    not_offered_seasons = [seasons[j] for j in range(best_solution.shape[1]) if best_solution[i][j] == 0]
    print(f"Tour: {tour_name}")
    print("-----------------------------------------------------------------------")
    print(f"  Offered in: {', '.join(offered_seasons)}")
    print(f"  Not offered in: {', '.join(not_offered_seasons)}")
    print()
print("**********************************************")
 print()
```

- This section displays the final best solution, including the best fitness value achieved and the details of which tour packages are offered in which seasons.

❖ **Output of GA to solve our optimization problem**

Best Fitness: $4,206,777

Tour: Cairo Stopover Tour

Offered in: Winter
Not offered in: Spring, Summer, Autumn
****

Tour: Day Tour in Cairo and the Pyramids

Offered in: Spring
Not offered in: Winter, Summer, Autumn

****

Tour: Day Trip to Pyramids from Cairo

Offered in: Spring, Autumn
Not offered in: Winter, Summer

****

Tour: Day Trip from Luxor to Cairo by Plane

Offered in: Winter, Summer
Not offered in: Spring, Autumn

****

Tour: Saqqara & Memphis Tour from Cairo

Offered in: Winter, Spring
Not offered in: Summer, Autumn

❖ **Final Comments:**

The use of a Genetic Algorithm for optimizing tour package offerings is a powerful approach due to its ability to handle complex, multi-dimensional optimization problems. The following points highlight the strengths and considerations of our approach:

❖ **Strengths:**

- **Flexibility**: The GA can handle a variety of constraints and objectives, making it suitable for diverse optimization problems.
- **Scalability**: The algorithm can scale to larger datasets and more complex scenarios with additional constraints and variables.
- **Robustness**: GA is less likely to get trapped in local optima compared to traditional optimization methods, making them suitable for complex, non-linear problems.

CHAPTER 5

User Interface (UI) Development and AI Prediction Model

5.1 User Interface (UI) Development

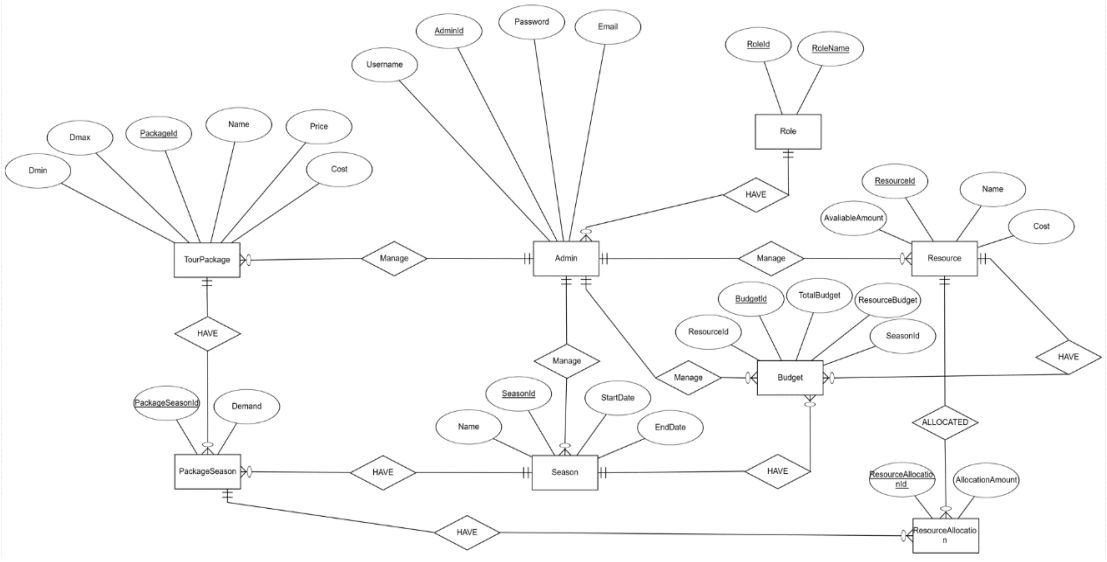
To facilitate easy interaction with the data for tourism industry professionals, we developed a user-friendly Windows Form application using C# and the .NET Framework. This UI allows administrators to manage various aspects of the tourism packages, such as prices, resources, and budgets. The development process included creating an Entity-Relationship (ER) schema, converting it to an Entity-Relationship Diagram (ERD), and then implementing it as a SQL database named **GPTourism**.

5.1.1 Database Design

The database design started with creating the ER schema and ERD. The schema was then translated into SQL queries to generate the database structure. The tables in the database are as follows:

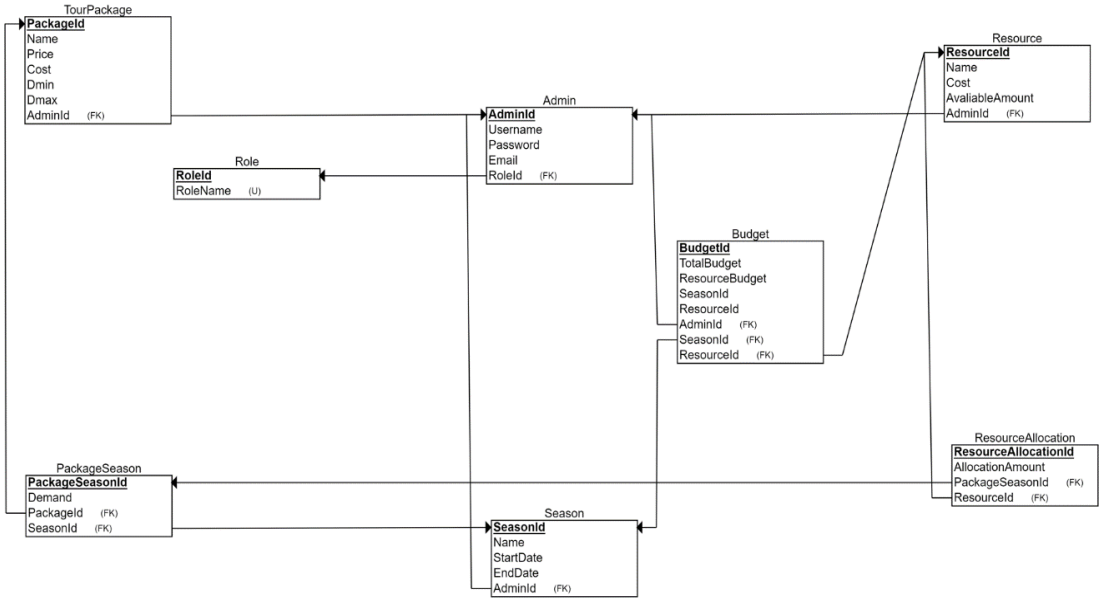
- **ER Schema (Entity Relationship Schema)**

Figure 5.1.1 1- ER Schema



❖ ER Diagram (Entity Relationship Diagram)

Figure 5.1.1 2- ER Diagram



❖ Database Tables and SQL Queries

- **Admin Table:**

```
CREATE TABLE Admin
(
 AdminId INT IDENTITY(1,1) NOT NULL,
 Username VARCHAR(255) UNIQUE NOT NULL,
 Password VARCHAR(255) NOT NULL,
 Email VARCHAR(255) UNIQUE NOT NULL,
 Role VARCHAR(50) NOT NULL,
 PRIMARY KEY (AdminId),
 FOREIGN KEY (Role) REFERENCES Role(RoleName) ON UPDATE NO
ACTION
);
```

- **TourPackage Table:**

```
CREATE TABLE TourPackage
(
 PackageId INT IDENTITY(1,1) NOT NULL,
 Name VARCHAR(255) NOT NULL,
 Price FLOAT NOT NULL,
 Cost FLOAT NOT NULL,
 Dmin INT NOT NULL, Dmax INT NOT NULL,
 AdminId INT NOT NULL, PRIMARY KEY (PackageId),
 FOREIGN KEY (AdminId) REFERENCES Admin(AdminId) ON
UPDATE CASCADE
);
```

- **Season Table:**

```
CREATE TABLE Season
(
 SeasonId INT IDENTITY(1,1) NOT NULL,
 Name VARCHAR(255) NOT NULL,
 StartDate DATE NOT NULL,
 EndDate DATE NOT NULL,
 AdminId INT NOT NULL,
 PRIMARY KEY (SeasonId), FOREIGN KEY (AdminId) REFERENCES
Admin(AdminId) ON UPDATE CASCADE
);
```

- **PackageSeason Table:**

```
CREATE TABLE PackageSeason
(
 PackageSeasonId INT IDENTITY(1,1) NOT NULL,
 Demand INT NOT NULL, PackageId INT NOT NULL,
 SeasonId INT NOT NULL,
 PRIMARY KEY (PackageSeasonId),
 FOREIGN KEY (PackageId) REFERENCES TourPackage(PackageId) ON
UPDATE NO ACTION,
 FOREIGN KEY (SeasonId) REFERENCES Season(SeasonId) ON
UPDATE NO ACTION
);
```

- **Resource Table:**

```
CREATE TABLE Resource
(
 ResourceId INT IDENTITY(1,1) NOT NULL,
 Name VARCHAR(255) NOT NULL,
 Cost FLOAT NOT NULL,
 AvailableAmount INT NOT NULL,
 AdminId INT NOT NULL,
 PRIMARY KEY (ResourceId),
 FOREIGN KEY (AdminId) REFERENCES Admin(AdminId) ON
UPDATE CASCADE
);
```

- **Budget Table:**

```
CREATE TABLE Budget
(
 BudgetId INT IDENTITY(1,1) NOT NULL,
 ResourceBudget FLOAT NOT NULL,
 ResourceId INT NOT NULL,
 SeasonId INT NOT NULL,
 AdminId INT NOT NULL,
 PRIMARY KEY (BudgetId),
 FOREIGN KEY (AdminId) REFERENCES Admin(AdminId) ON
UPDATE CASCADE,
 FOREIGN KEY (SeasonId) REFERENCES Season(SeasonId) ON
UPDATE NO ACTION,
 FOREIGN KEY (ResourceId) REFERENCES Resource(ResourceId) ON
UPDATE NO ACTION
);
```

- **ResourceAllocation Table:**

```
CREATE TABLE ResourceAllocation
(
 ResourceAllocationId INT IDENTITY(1,1) NOT NULL,
 AllocationAmount INT NOT NULL,
 PackageSeasonId INT NOT NULL,
 ResourceId INT NOT NULL,
 PRIMARY KEY (ResourceAllocationId),
 FOREIGN KEY (PackageSeasonId) REFERENCES PackageSeason(Pack-
ageSeasonId) ON UPDATE NO ACTION,
 FOREIGN KEY (ResourceId) REFERENCES Resource(ResourceId) ON
UPDATE NO ACTION );
```

- **Role Table:**

```
CREATE TABLE Role
(
 RoleId INT IDENTITY (1,1) NOT NULL,
 RoleName VARCHAR(255) NOT NULL,
 PRIMARY KEY(RoleId)
);
```

5.1.2 Relationships Between Tables

- **Admin to TourPackage:**
  One-to-Many : One Admin can manage multiple TourPackages.

- **Admin to Season:**
  One-to-Many : One Admin can manage multiple Seasons.

- **Admin to Resource:**
  One-to-Many : One Admin can manage multiple Resources.

- **Admin to Budget:**
  One-to-Many : One Admin can manage multiple Budgets.

- **TourPackage to PackageSeason:**
  One-to-Many : One TourPackage can have multiple PackageSeasons.

- **Season to PackageSeason:**
  One-to-Many : One Season can have multiple PackageSeasons.

- **PackageSeason to ResourceAllocation:**
  One-to-Many : One PackageSeason can have multiple ResourceAllocations.

- **Resource to ResourceAllocation:**
  One-to-Many : One Resource can be allocated in multiple ResourceAllocations.

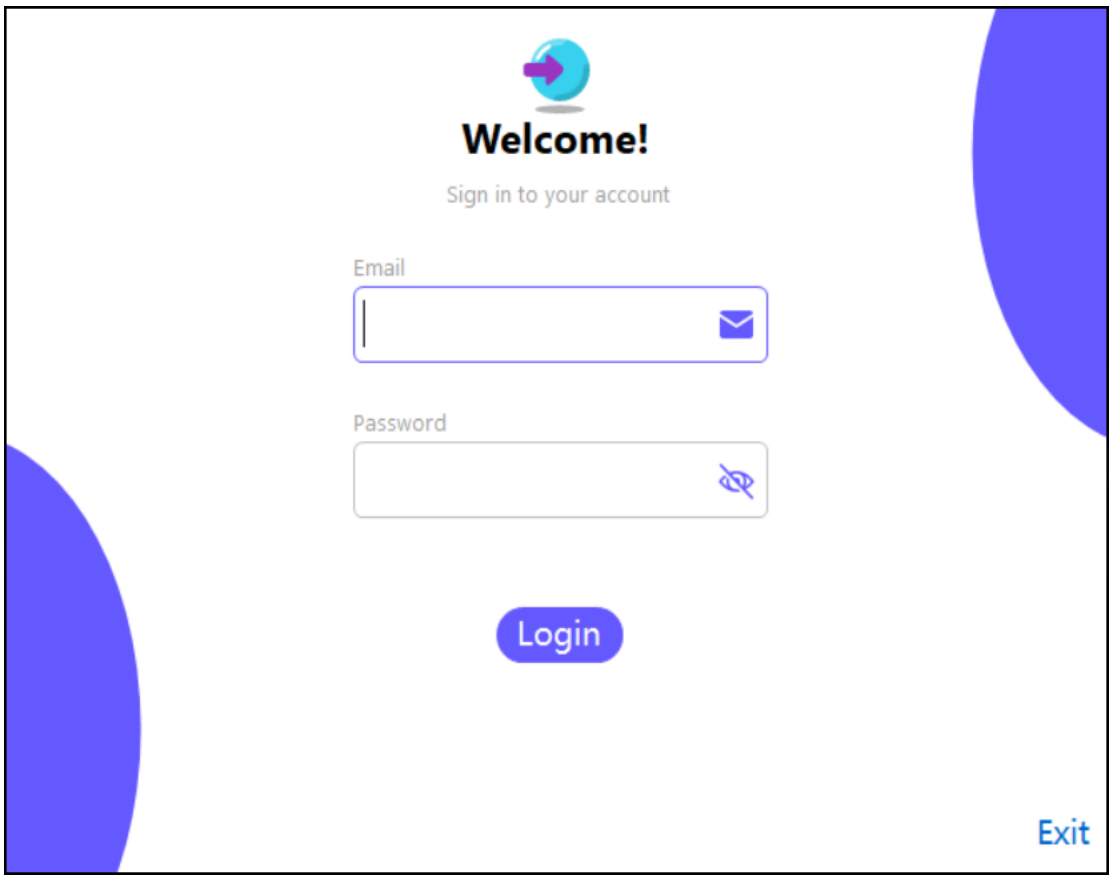
5.1.3 Windows Form Application

We developed a Windows Form application with two primary forms: a login page and an admin dashboard. The admin dashboard includes various functionalities for managing the different entities in the database.

❖ **Login Page**

The login page is designed to ensure secure access to the system. Each admin must enter their email and password correctly to gain access. The role assigned to the admin determines the functionalities they can access within the system. This role-based access control ensures that each admin can only perform actions relevant to their responsibilities.

Figure 5.1.3 1- Login Page



❖ **Admin Dashboard**

The admin dashboard is the central hub for managing all aspects of the tourism packages and related resources. The functionalities available in the dashboard vary based on the admin's role:

- **Admin Management:**
  - Add New Admin
  - Update Admin
  - View Admin
- **Tour Package Management:**
  - Add New Tour Package
  - Update Packages
  - View Packages
- **Package Season Management:**
  - Add Package Season
  - Update Package Season
  - View Package Season
- **Resource Management:**
  - Add Resource
  - Update Resource
  - View Resources
  - Allocate Resources
  - Update Allocation
  - View Allocated Resources
- **Season Management:**
  - Add New Season
  - Update Season
  - View Seasons
- **Budget Management:**
  - Add New Budget
  - Update Budget

- View Budget

- **Role Management:**
  - Add New Role
  - Update Role
  - View Role

Figure 5.1.3 2- Add New Admin Page

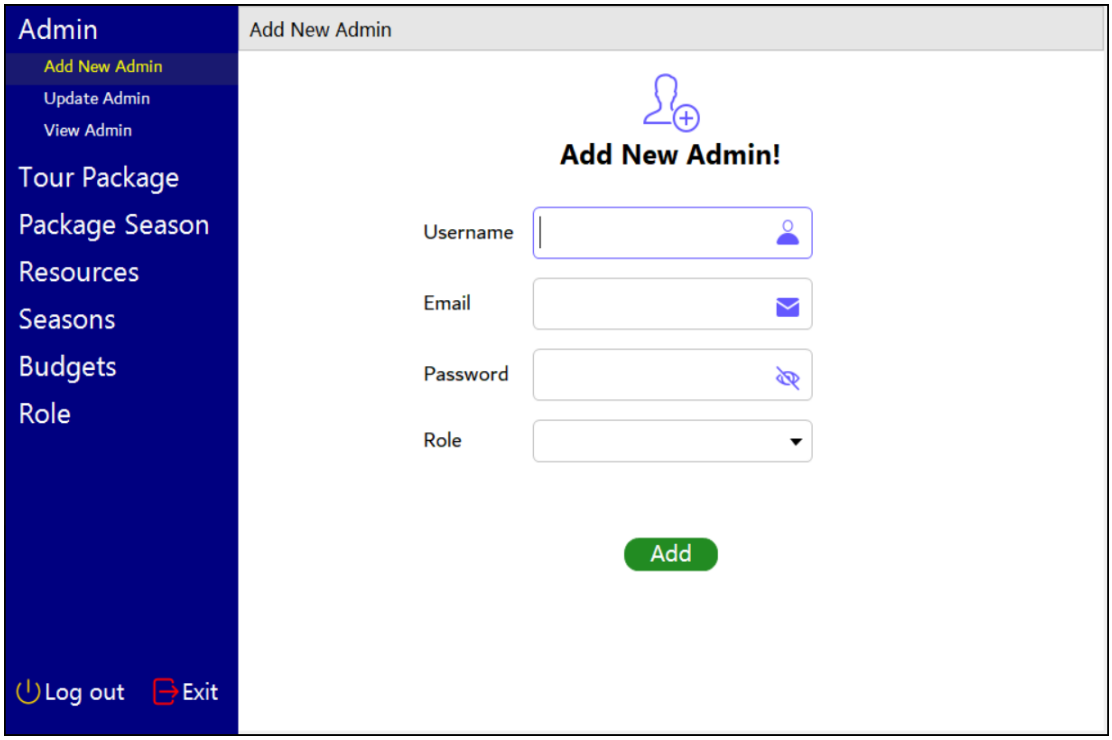


Figure 5.1.3 3- Update Admin Page

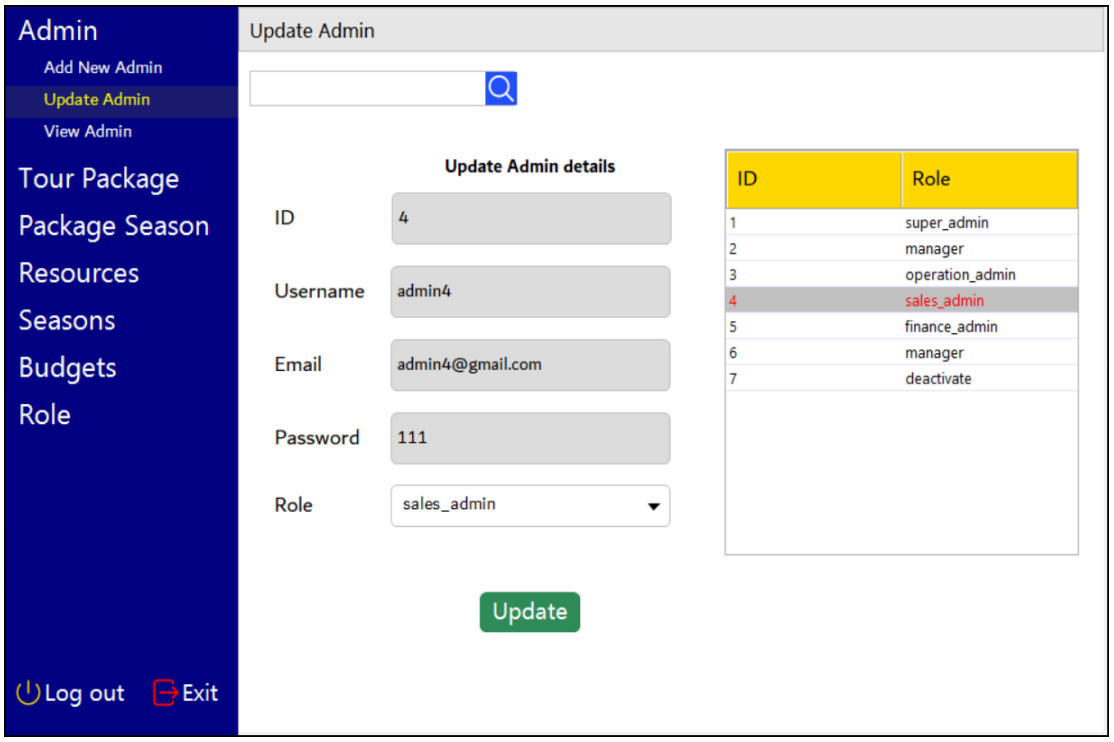


Figure 5.1.3 4- View Admin Page

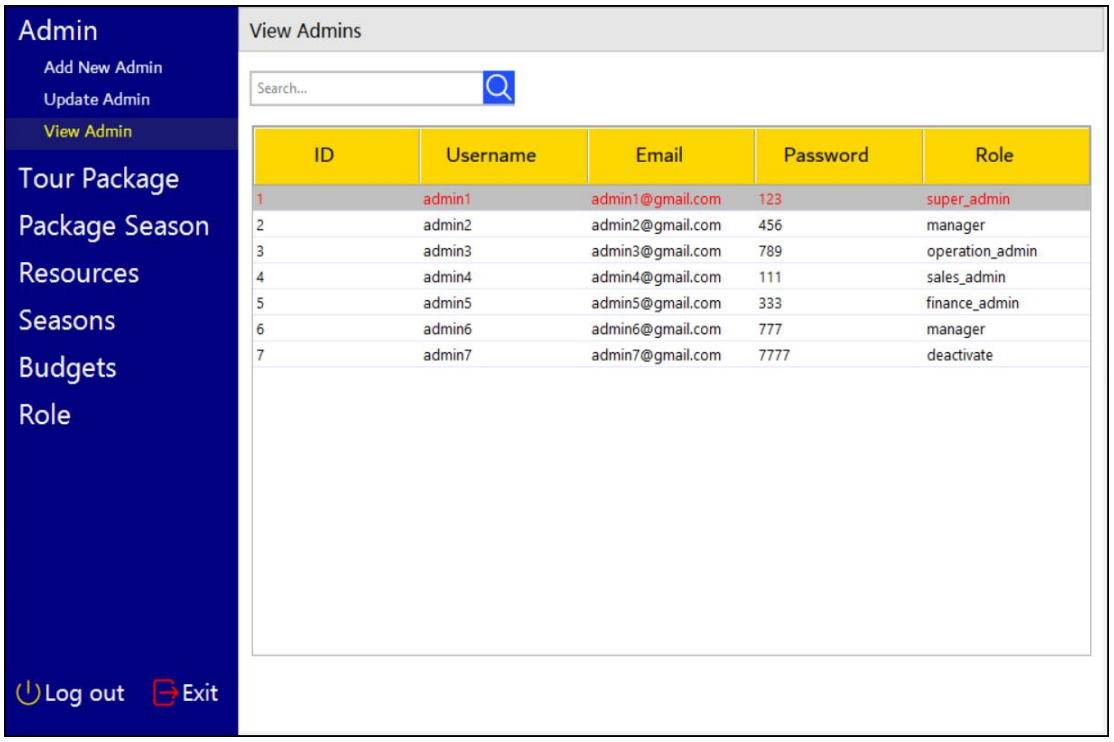


Figure 5.1.3 5- Add New Package Page

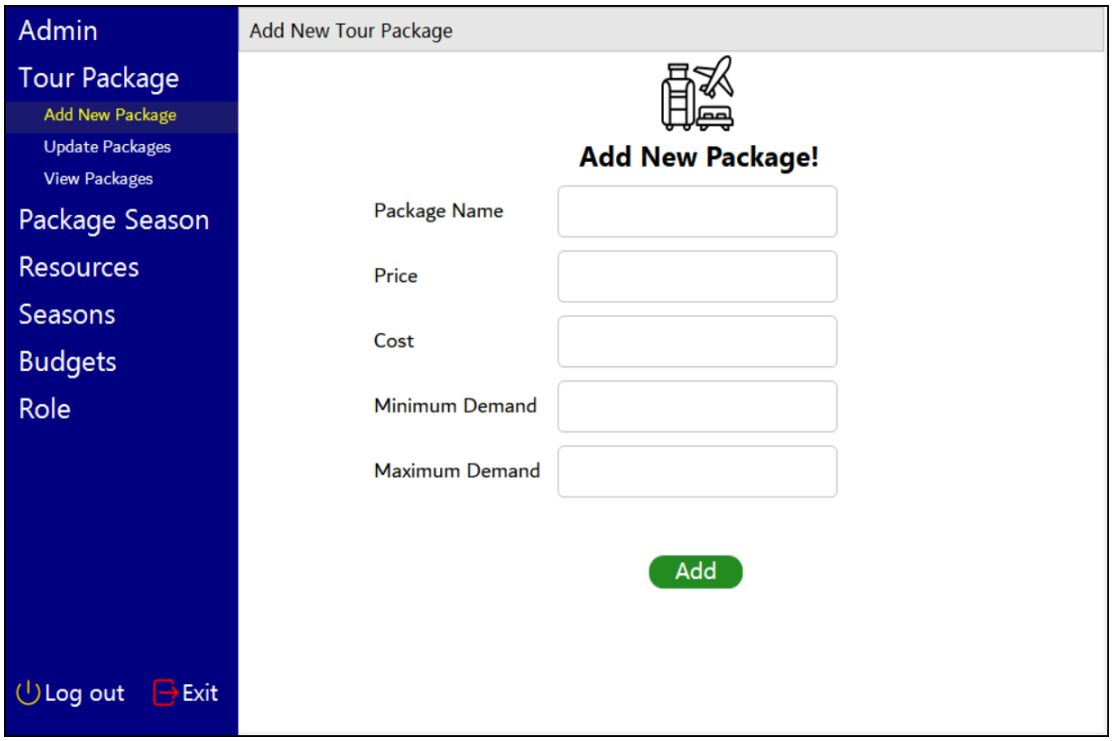


Figure 5.1.3 6- Update Package Page

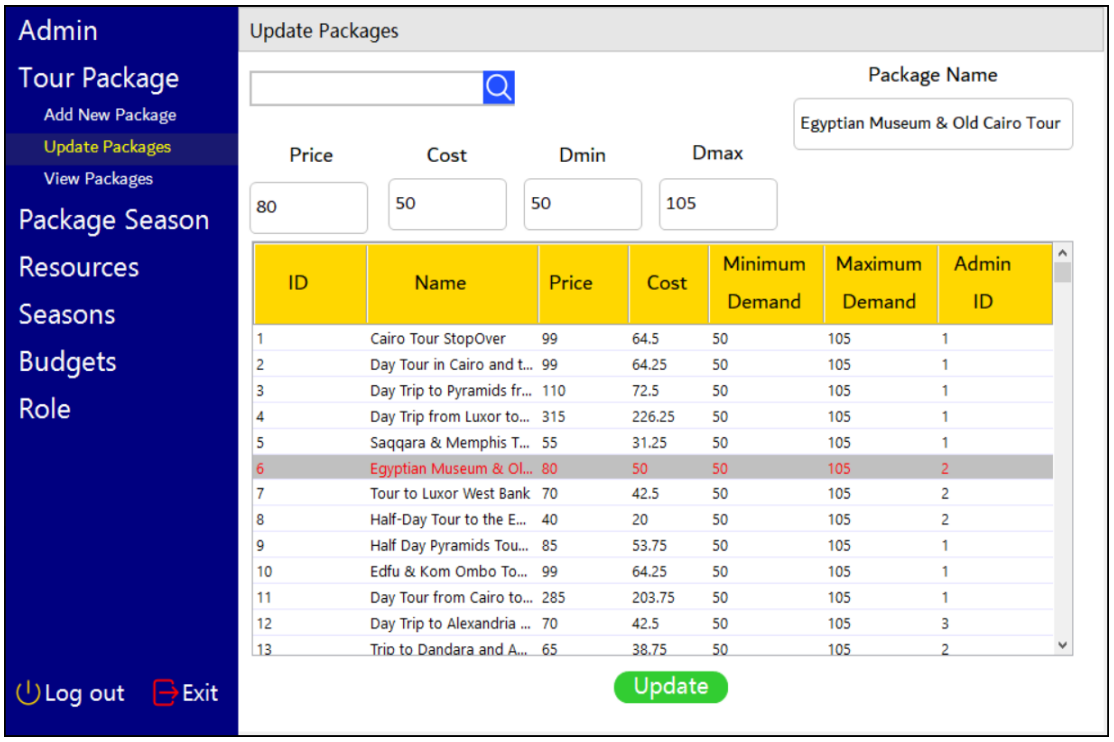


Figure 5.1.3 7- View Package Page

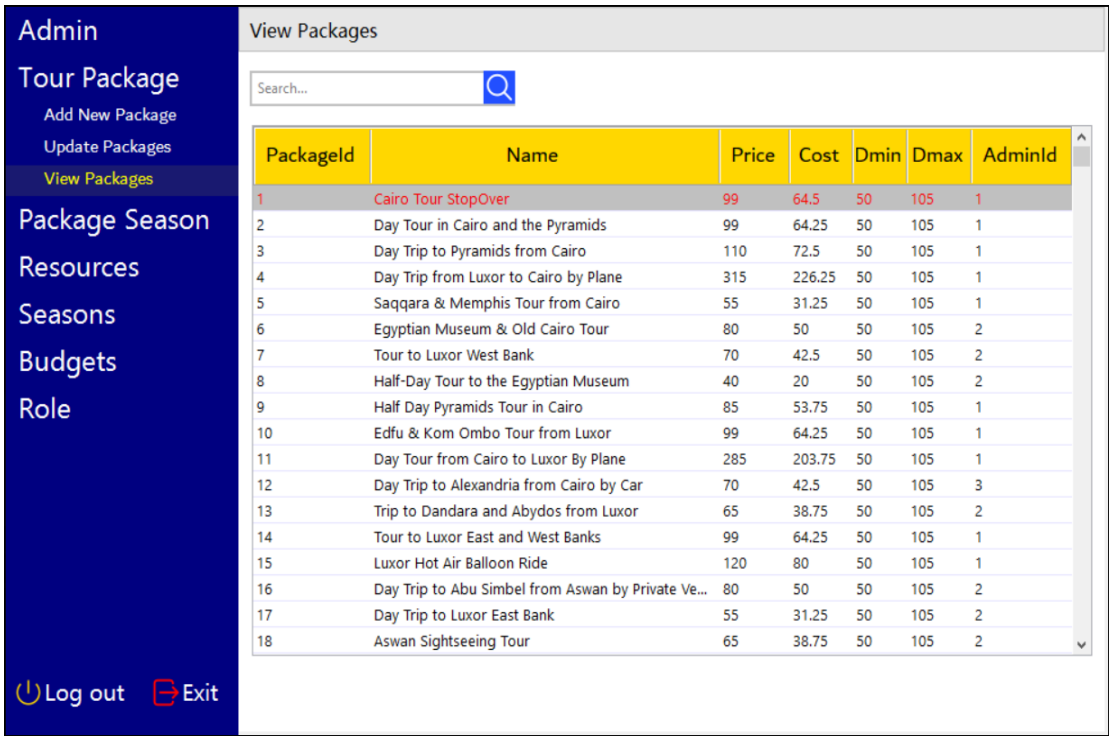


Figure 5.1.3 8- Add Package Season Page

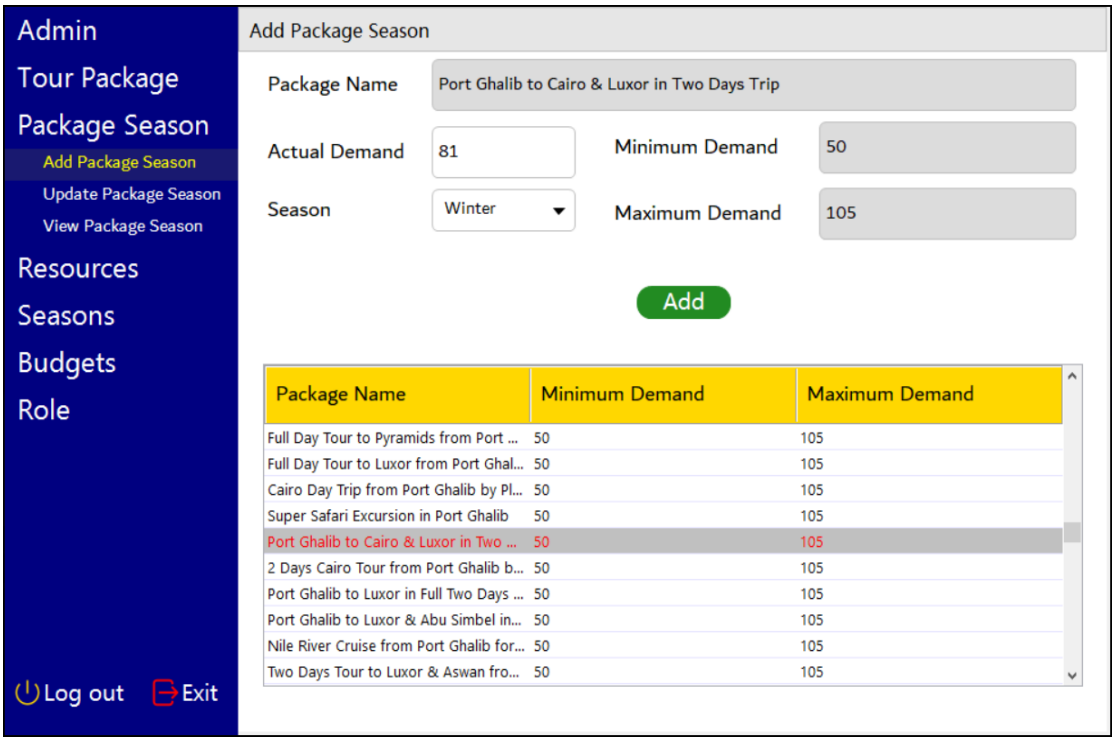


Figure 5.1.3 9- Update Package Season Page

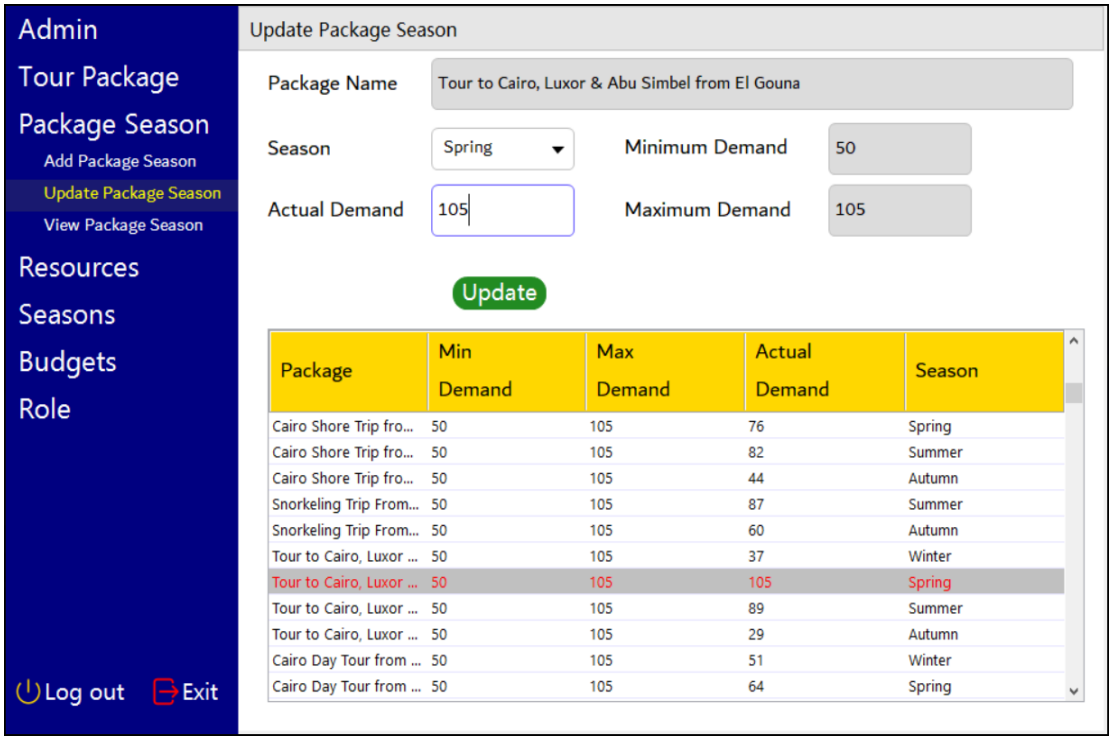


Figure 5.1.3 10- View Package Season Page

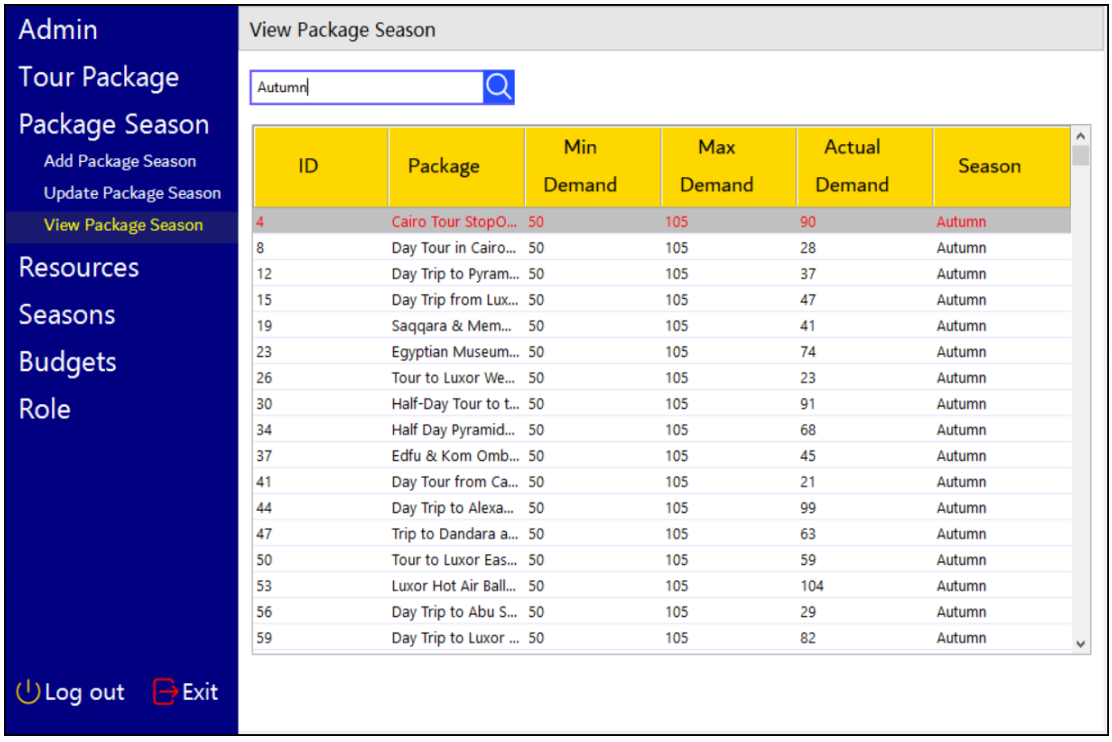


Figure 5.1.3 11- Add Resource Page

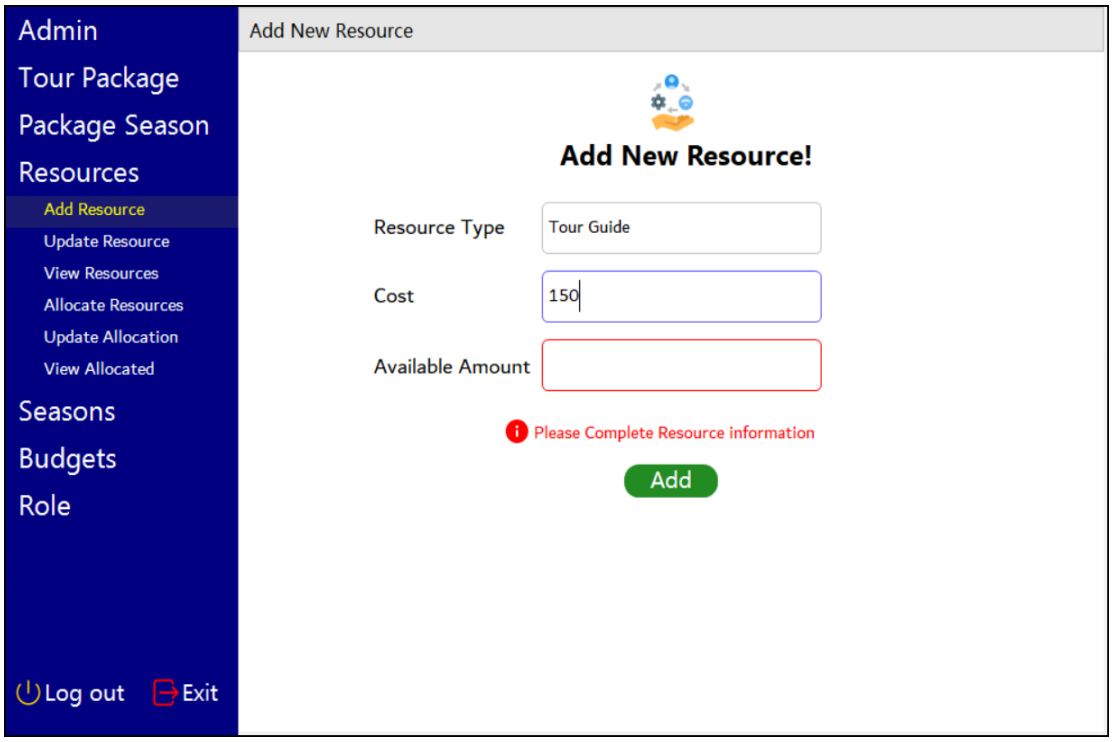


Figure 5.1.3 12- Update Resource Page

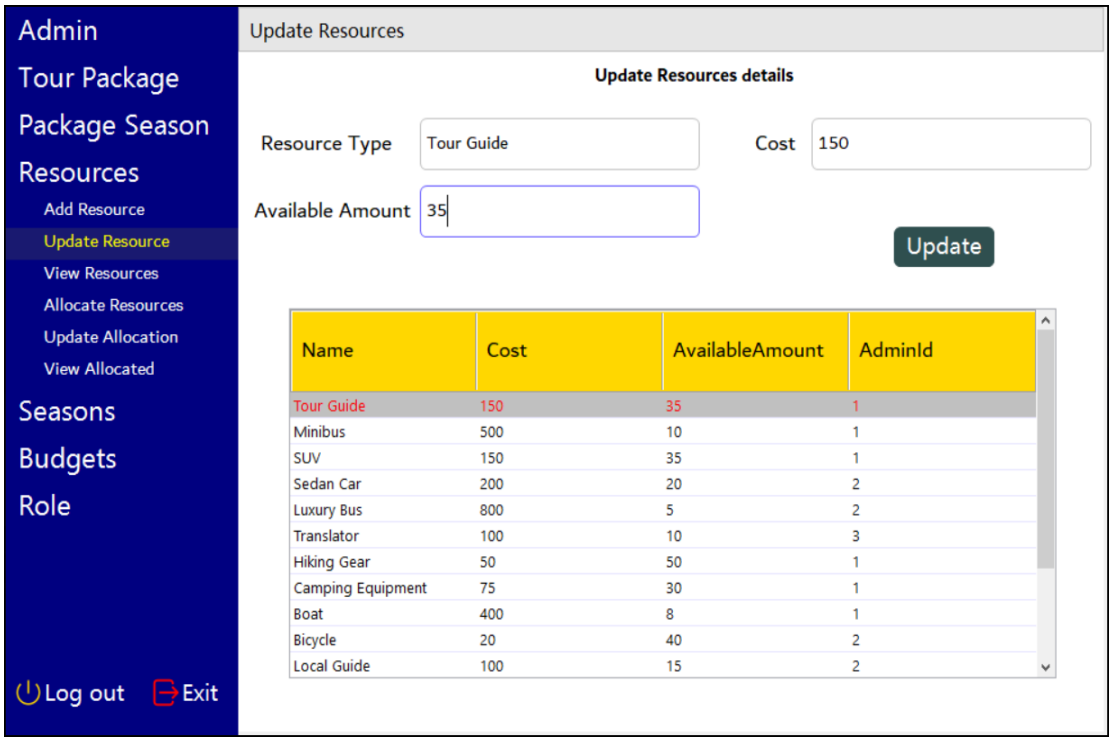


Figure 5.1.3 13- View Resources Page

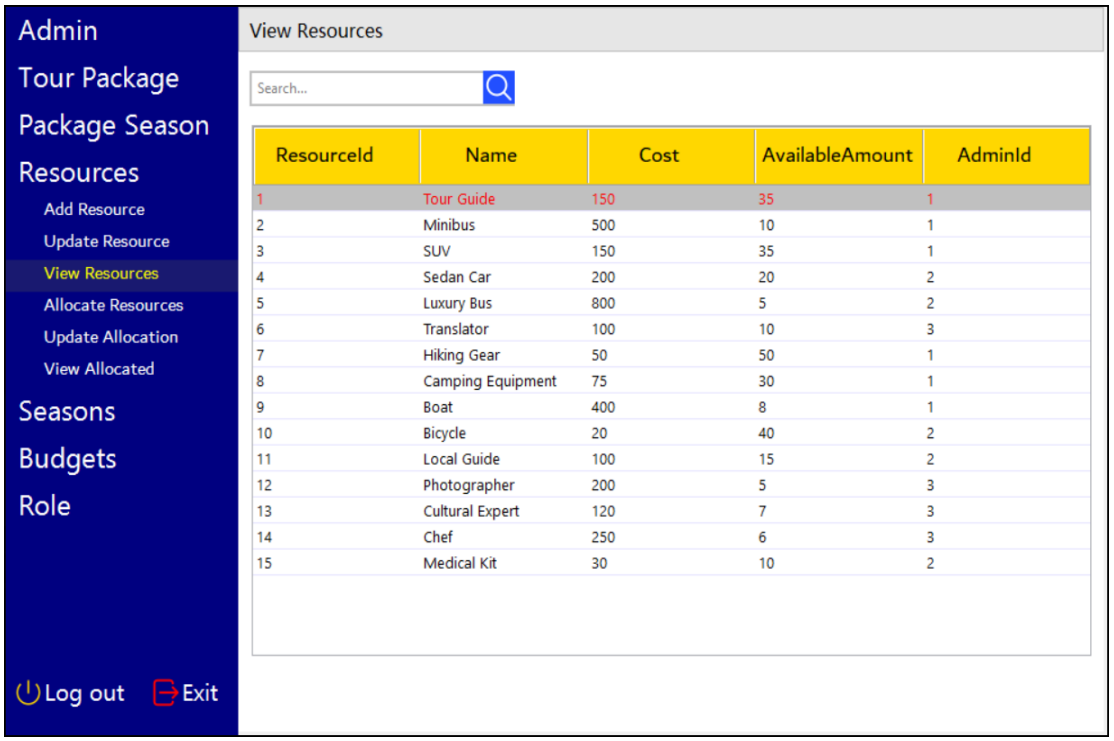


Figure 5.1.3 14- Allocate Resources Page

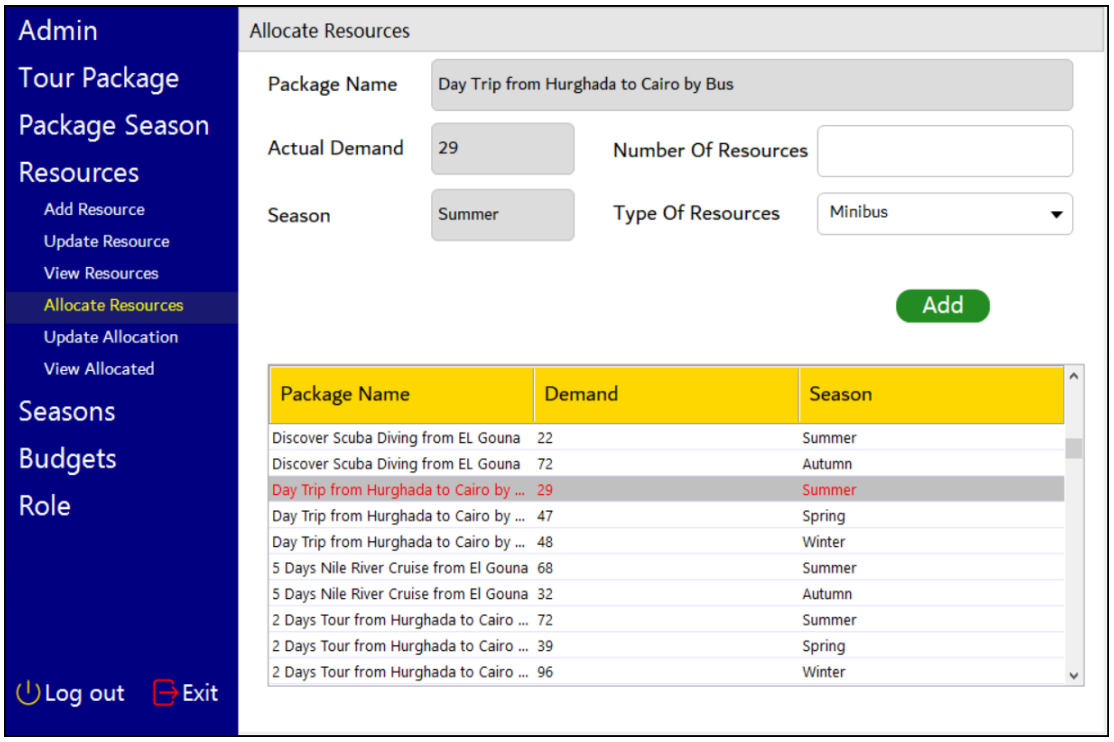


Figure 5.1.3 15- Update Allocation Resources

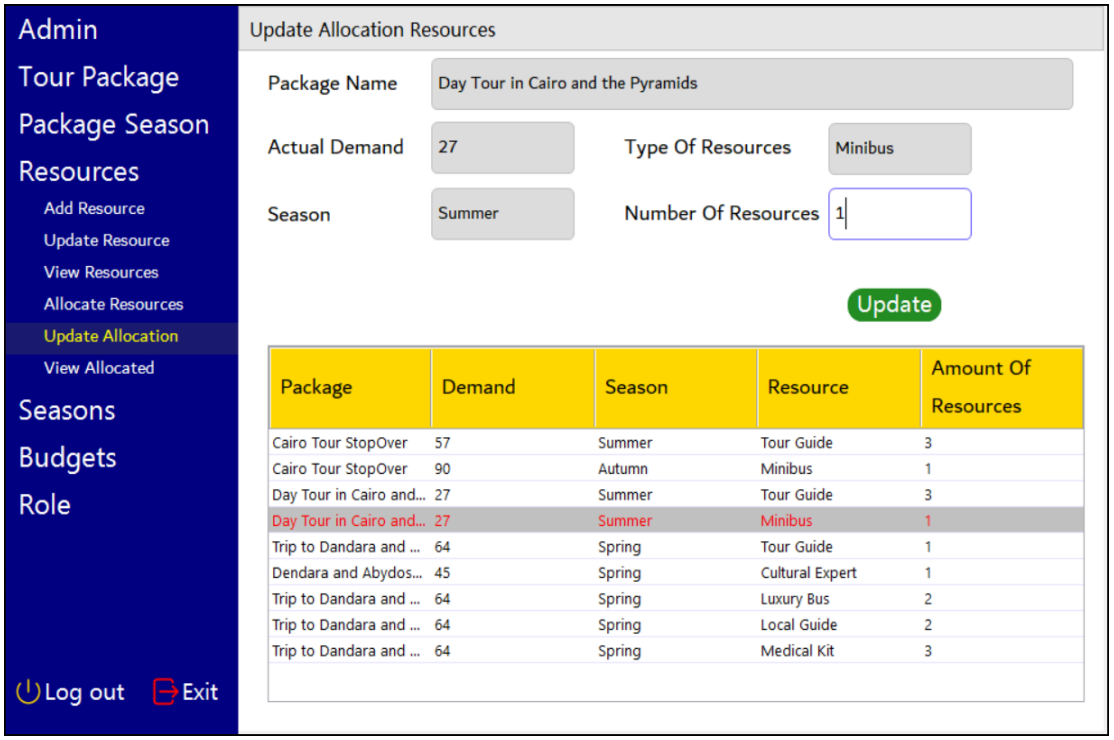


Figure 5.1.3 16- View Allocated Resources Page

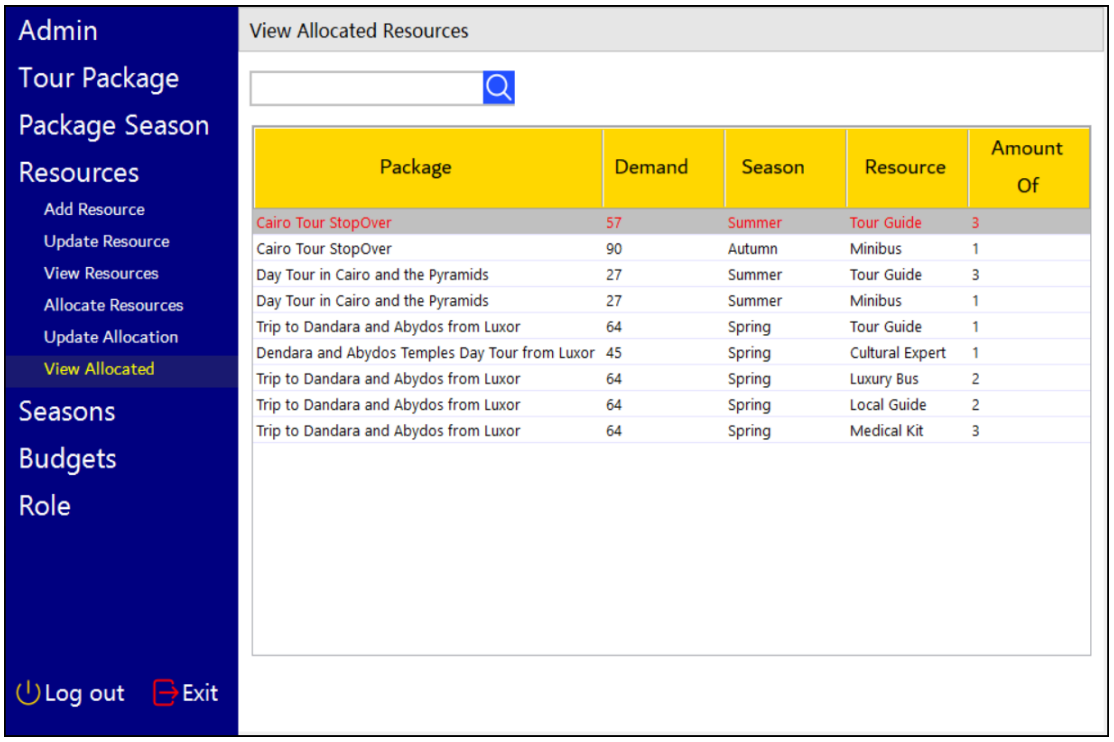


Figure 5.1.3 17- Add New Season Page

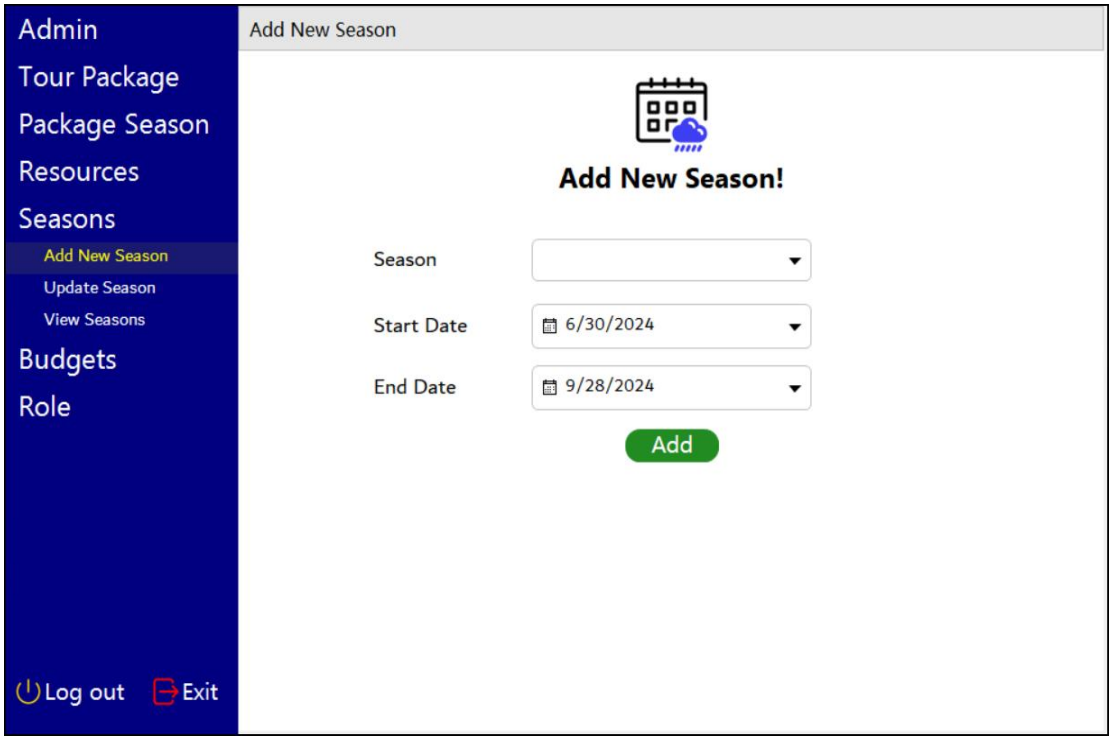


Figure 5.1.3 18- Update Season Page

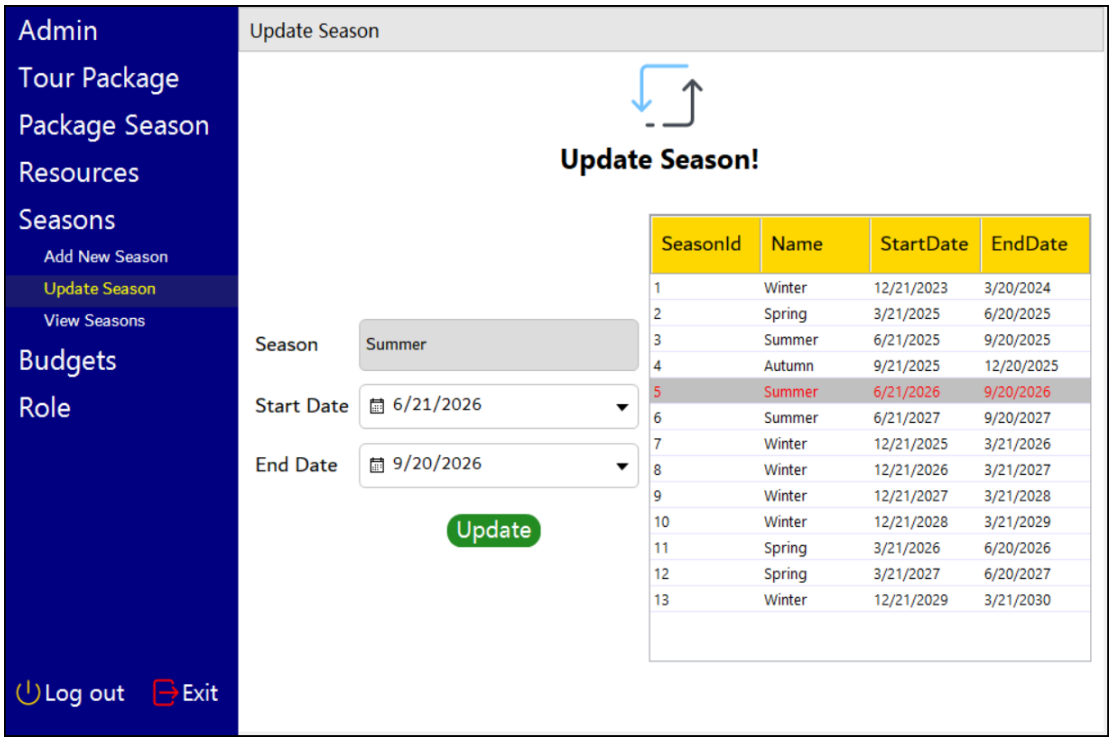


Figure 5.1.3 19- View Seasons Page

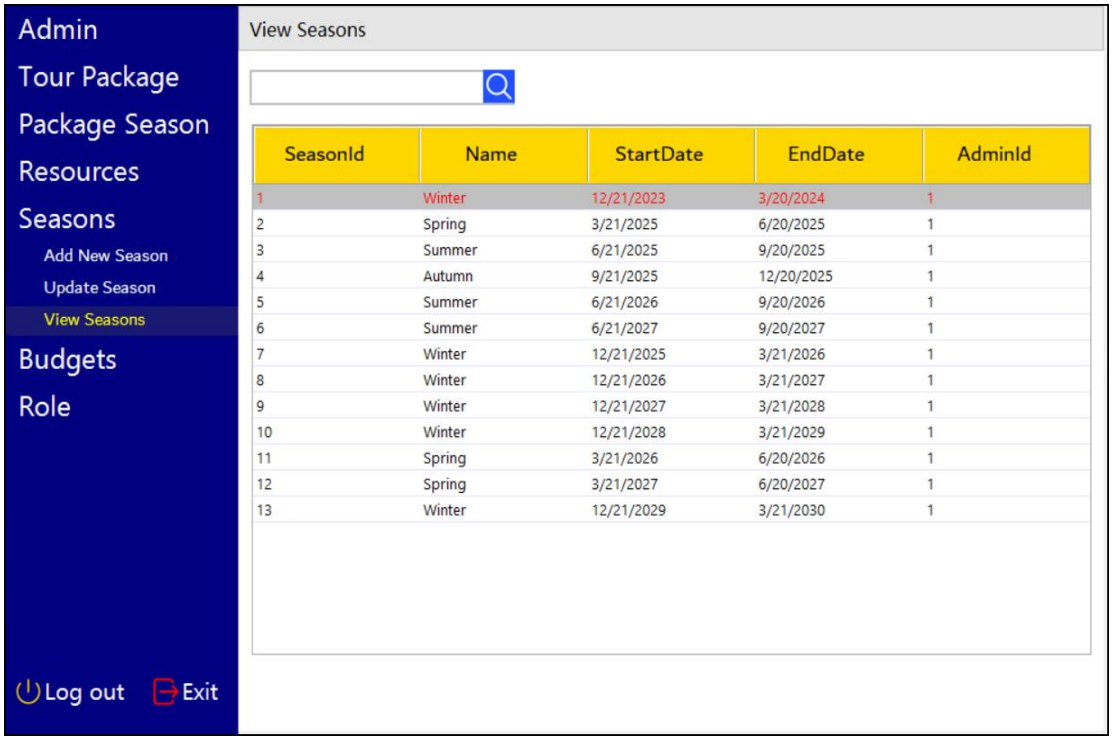


Figure 5.1.3 20- Add New Budget Page

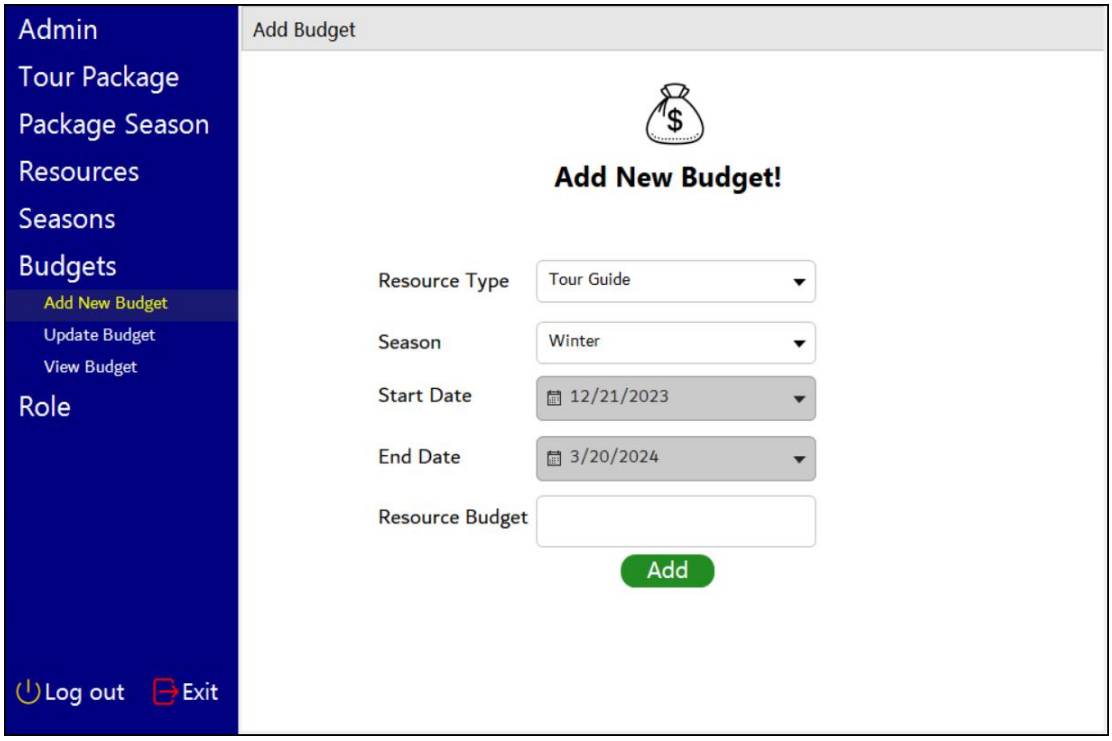


Figure 5.1.3 21- Update Budget Page

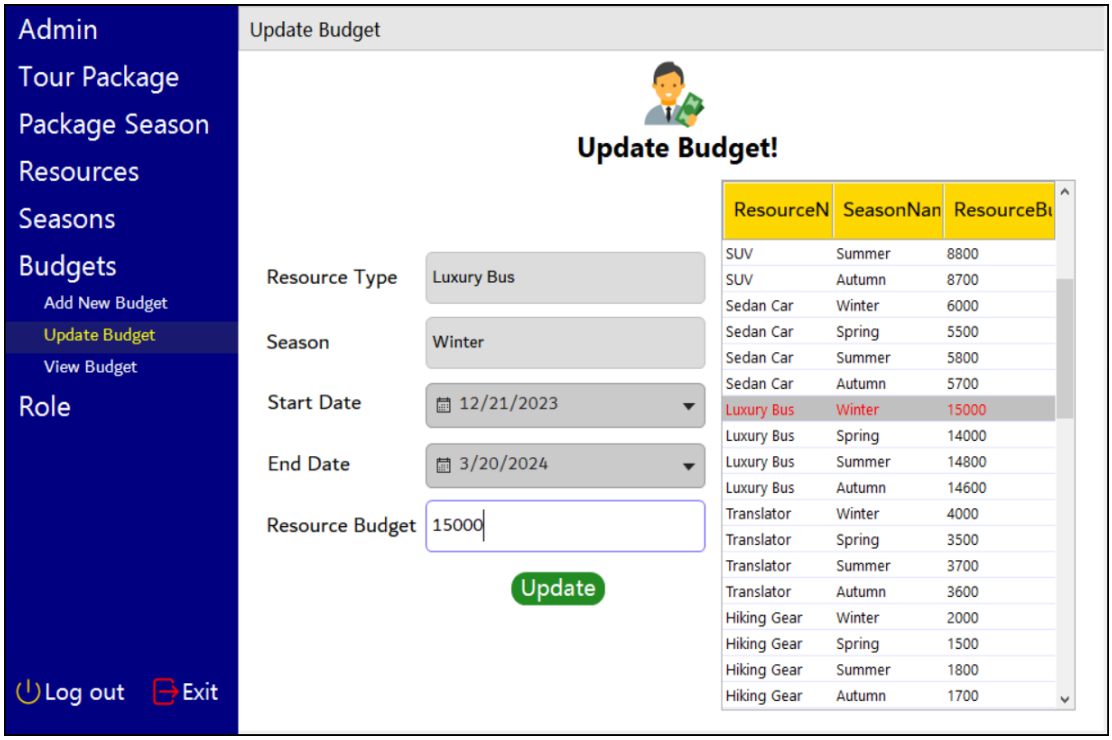


Figure 5.1.3 22- View Budget Page

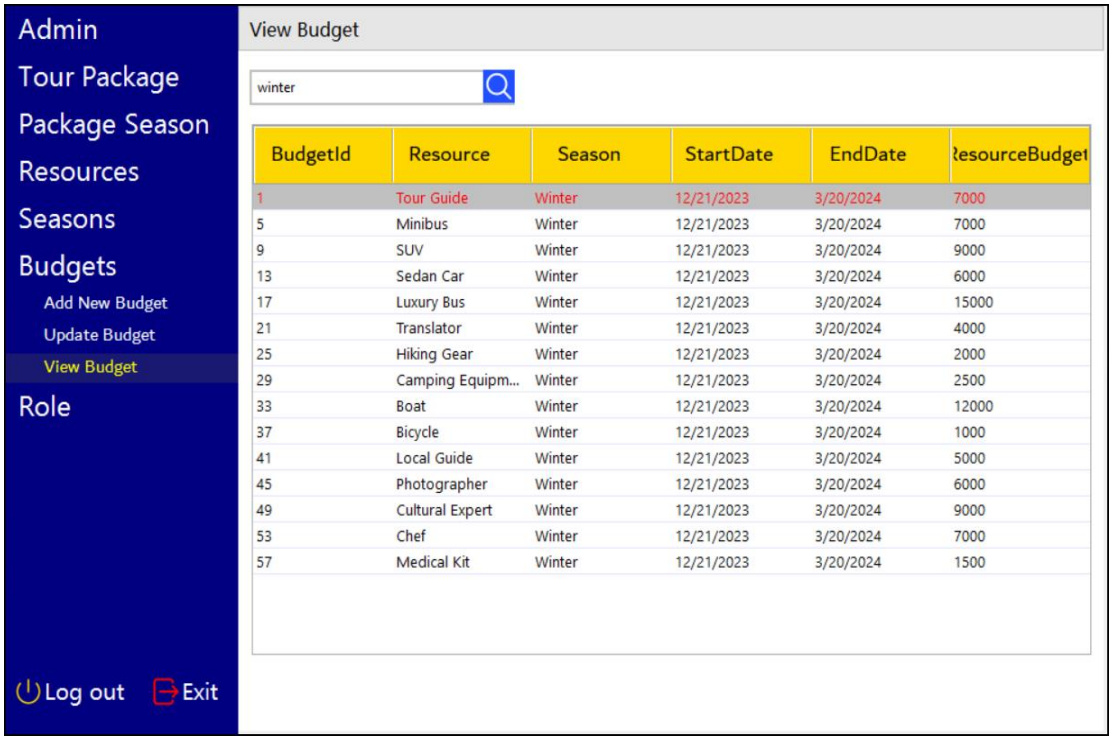


Figure 5.1.3 23- Add New Role Page

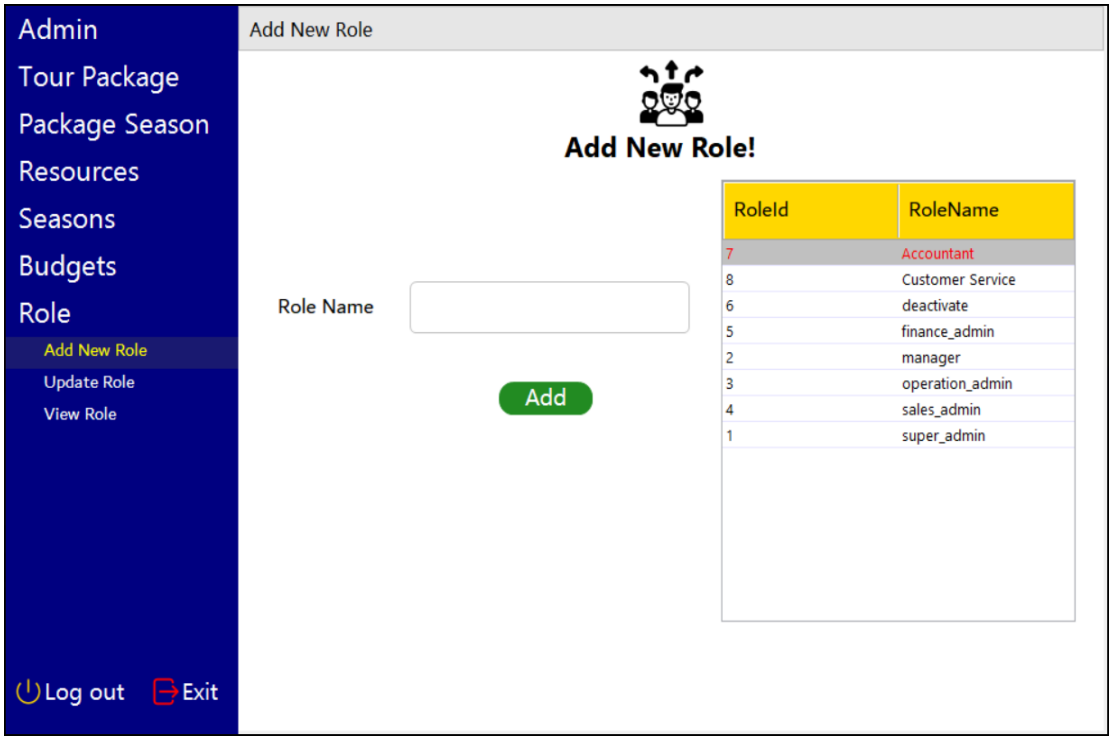


Figure 5.1.3 24- Update Role Page

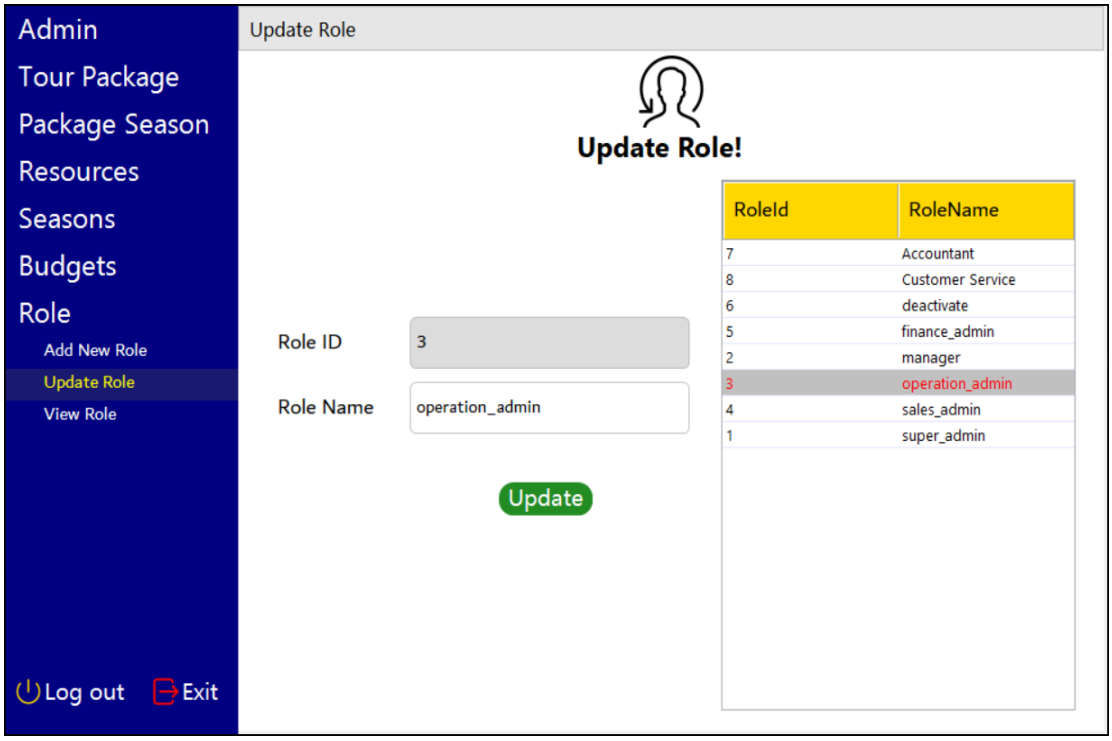
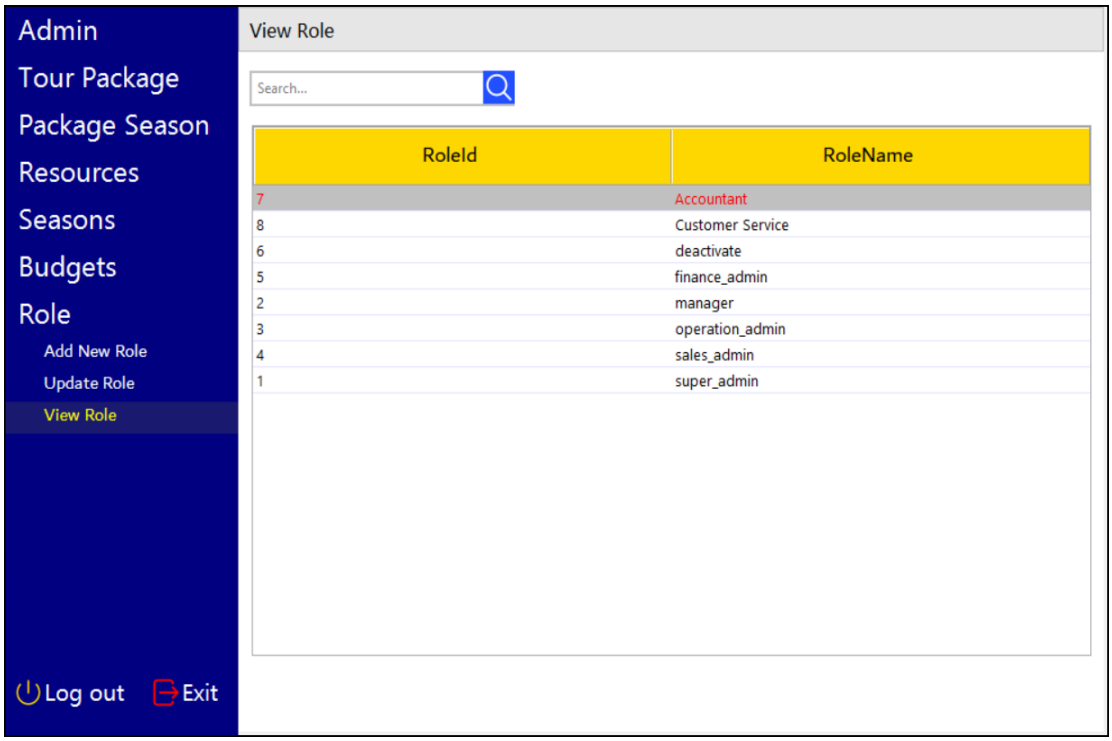


Figure 5.1.3 25- View Role Page



Each admin's access is tailored according to their role, ensuring that they can perform their duties efficiently while maintaining the security and integrity of the data.

❖ Admin Roles and Their Functionalities

- **Super Admin:**
  - Full access to all functionalities.
  - Can manage other admins, including creating and assigning roles.
  - CRU (Create , Read , Update) operations on all tables (Admin, TourPackage, Season, PackageSeason, Resource, Budget, ResourceAllocation and Role).

- **Manager:**
  - Can manage tour packages, seasons, resources, and budgets.
  - Cannot manage other admins or assign roles.
  - CRU (Create , Read , Update) operations on TourPackage, Season, Resource, Budget, and ResourceAllocation.
  - View and update data but cannot manage other admins.

- **Finance Admin:**

  - Focused on managing budgets and costs.
  - Can view and update budget-related information.
  - CRU (Create , Read , Update) operations on Budget.
  - View operations on Resource, TourPackage, Season, and ResourceAllocation.

- **Operations Admin:**

  - Manages operational aspects such as resource allocation and availability.
  - Can view and update resource-related information.
  - Can manage tour packages and seasons to some extent.
  - CRU (Create , Read , Update) operations on Resource, TourPackage, Season, and ResourceAllocation.
  - View operations on Budget.

- **Sales Admin:**

  - Focused on sales and marketing aspects.
  - Can create and update tour packages.
  - Can view and manage demand data for different seasons.
  - CRU (Create , Read , Update) operations on TourPackage and PackageSeason.
  - View operations on Season, Resource, Budget, and ResourceAllocation.

- **Deactivated:**

  - This admin cannot access the system as they are no longer with the company.

5.1.4 Integration with AI Prediction Model

The data managed through the UI will be utilized in the AI Prediction Model to forecast package demands. This integration aims to enhance decision-making for tourism industry stakeholders by providing accurate predictions based on historical and real-time data.

5.2 Scope of the AI Prediction Model

This model focuses on developing a predictive model using machine learning techniques to forecast demand for tour packages. It involves data collection from historical tour package data, seasonal trends, and pricing information. Additionally, future work includes creating an optimization algorithm to allocate resources effectively and forecast budgets for resources. The scope encompasses data collection, model development, evaluation, and the implementation of a graphical user interface (GUI) to facilitate user interaction. This model aims to enhance decision-making, improve resource utilization, and increase profitability for tourism companies. Future enhancements may include integrating real-time data and improving the accuracy of predictions through advanced algorithms.

5.3 Literature View of the AI Prediction Model

5.3.1 Overview of AI Techniques in Demand Prediction

Artificial Intelligence (AI) techniques have been widely adopted in demand prediction across various industries, including tourism. These techniques help in forecasting demand accurately by analyzing historical data and identifying patterns. Here are some key AI techniques used in demand prediction:

- **Linear Regression**: A simple and commonly used technique that models the relationship between a dependent variable and one or more independent variables. It helps in understanding how changes in factors like price and season affect demand.
- **Decision Trees**: These are tree-like models used for both classification and regression tasks. They split data into subsets based on the value of input features, making it easier to predict demand based on different conditions.
- **Random Forest**: An extension of decision trees, this technique uses multiple trees to improve prediction accuracy. It reduces the risk of overfitting and provides more reliable demand forecasts.
- **Support Vector Regressor (SVR):** SVR is a powerful technique used for classification and regression tasks. It finds the best boundary that separates different classes of data, helping in making precise demand predictions.
- **Neural Networks**: These are complex models inspired by the human brain, capable of capturing intricate patterns in data. Neural networks, particularly deep learning models, are highly effective in predicting demand for tourism packages due to their ability to learn from large datasets.
- **Gradient Boosting**: This technique builds models sequentially, with each model trying to correct the errors of the previous one. It is effective in improving the accuracy of demand predictions.

- **K-Nearest Neighbors (KNN)**: KNN predicts demand based on the similarity of new data points to the existing data. It is simple and effective for small datasets but can be computationally intensive for larger ones.

These AI techniques provide powerful tools for accurately forecasting demand in the tourism industry, helping companies optimize their resources and improve profitability.

5.3.2 Applications of AI in Tourism Demand Prediction

AI techniques have been widely adopted in the tourism industry to predict demand for various tour packages. These techniques help businesses make informed decisions, optimize resources.

- **Tour Package Forecasting**: Companies use machine learning models to forecast the demand for specific tour packages during different seasons. By analyzing historical data, these models can predict which packages will be popular, helping businesses prepare in advance.
- **Dynamic Pricing**: AI models can predict demand fluctuations and adjust prices accordingly. This helps maximize revenue by offering competitive prices during low demand periods and higher prices when demand is high.
- **Resource Allocation**: Predictive models help in planning and allocating resources like tour guides, vehicles, and accommodation. By understanding future demand, companies can ensure they have the right resources in place to meet customer needs.

5.3.3 Challenges and Limitations

While AI offers significant advantages in demand prediction for tourism, there are several challenges and limitations to consider:

- **Data Quality**: Accurate predictions rely on high-quality data. Incomplete, outdated, or incorrect data can lead to inaccurate predictions. Ensuring data quality is a continuous challenge for businesses.
- **Model Complexity**: Some AI models are highly complex and require specialized knowledge to develop and maintain. This can be a barrier for small businesses with limited technical expertise.
- **Overfitting**: AI models can sometimes overfit historical data, meaning they perform well on past data but poorly on new, unseen data. This issue requires careful model validation and regular updates.
- **Scalability**: Implementing AI models at scale can be challenging, especially for large datasets and real-time predictions. It requires robust infrastructure and efficient algorithms to handle large volumes of data.

- **Cost**: Developing and deploying AI models can be expensive. The cost of data collection, storage, processing, and skilled personnel can be significant, especially for small to medium-sized enterprises.

By addressing these challenges and limitations, businesses can effectively leverage AI to enhance their demand prediction capabilities and improve overall operational efficiency.

5.4 Methodology of AI Prediction Model

5.4.1 Data Collection and Preprocessing

- **Data Sources**
  The data for this project was collected from online platforms of Egyptian tourism companies. These sources provided a comprehensive dataset on various tour packages, seasons, and related resources.

- **Data Description**
  The dataset consists of multiple sheets, each containing specific information:

  - **PackageSeason**: Details about the association between packages and seasons.
    - Columns: PackageSeasonId, Demand, PackageId, SeasonId, Year

  - **TourPackage**: Information about different tour packages.
    - Columns: PackageId, Name, Price, Cost, Dmin, Dmax, AdminId

  - **Season**: Information about different seasons.
    - Columns: SeasonId, Name, StartDate, EndDate, AdminId

  - **Resource**: Various resources required for the packages.
    - Columns: ResourceId, Name, Cost, AvailableAmount, AdminId

  - **ResourceAllocation**: Allocation of resources to packages.
    - Columns: ResourceAllocationId, AllocationAmount, PackageSeasonId, ResourceId

  - **Role**: Roles of individuals involved.
    - Columns: RoleId, RoleName

- **Budget**: Budget-related information.
  - Columns: BudgetId, ResourceBudget, ResourceId, SeasonId, AdminId

❖ **Data Cleaning and Transformation**

To ensure the data's quality and usability, the following preprocessing steps were undertaken:

- **Handling Missing Values**: Missing data points were identified and appropriately handled through imputation or removal to ensure the dataset's integrity.
- **Data Merging**: Relevant data from multiple sheets were merged to create a consolidated dataset for demand prediction. For instance, data from the PackageSeason, TourPackage, and Season sheets were combined.

Figure 5.4.1 1- Merging data for demand to select feature

```
# Merge data for demand prediction
demand_data
= package_season_data.merge(tour_package_data, on='PackageId').merge(season_data, on='SeasonId')
```

- **Feature Selection**: Important features like PackageId, SeasonId, Year, Cost, Price, Dmin, and Dmax were selected for the prediction model.
- **Standardization**: The features were standardized to ensure that they are on a similar scale, which helps improve the performance of machine learning algorithms.
- **Splitting the Data**: The cleaned and transformed data was split into training and testing sets to evaluate the prediction models effectively.

5.4.2 Feature Selection and Engineering

❖ **Feature Selection Criteria**

In our project, feature selection is crucial to enhance the predictive performance of our models. We considered several criteria to choose the most relevant features:

- **Relevance to Demand Prediction**: We selected features that have a direct or indirect impact on the demand for tour packages. This includes the package details, seasonal information, cost, and price factors as mentioned in **Figure 5.4.1 1-Merging data for demand to select future.**

- **Data Availability**: Only features with complete and accurate data were chosen to ensure the integrity of our predictions.

Figure 5.4.2 1- Filter data for training and testing

```
# Filter data for years 2019 to 2023 for training and testing
demand_data_train = demand_data[demand_data['Year'].between(2019, 2023)]
```

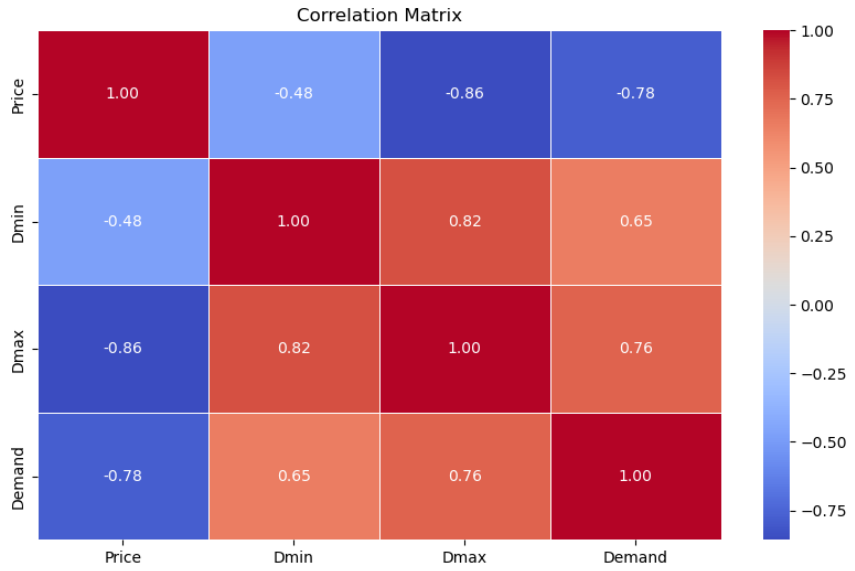
- **Correlation Analysis**: We conducted statistical tests to identify features that have a strong correlation with the demand variable, ensuring that only the most influential factors are included.

Figure 5.4.2 2- Create Correlation Matrix

```
# calculate correlation matrix
correlation_matrix = demand_data_train[['PackageId','SeasonId','Year','Price','Dmin','Dmax','Demand']].corr()

# print correlation matrix
print("Correlation matrix")
print(correlation_matrix )
```

And we have the following correlation matrix that appear a correlation between features and demand

Figure 5.4.2 3- Correlation Matrix



❖ **Feature Engineering Techniques**

Feature engineering involves creating new features or modifying existing ones to improve model performance. Here are the techniques we used:

- **Standardization**:
  - ➢ **Purpose**: Standardizing features brings all variables to a common scale, which improves the efficiency and performance of machine learning algorithms.
  - ➢ **Implementation**: The StandardScaler from sklearn.preprocessing was used to standardize the features

Figure 5.4.2 4- Standardization Part

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- **Temporal Features**:
  - ➢ **Purpose**: Including temporal features like the year helps capture trends and seasonality in demand over time.
  - ➢ **Implementation**: The Year feature was directly included in the dataset.

Figure 5.4.2 5- Temporal Features of the data

```
X = demand_data_train[['PackageId', 'SeasonId', 'Year', 'Price', 'Dmin', 'Dmax']]
```

- **Categorical Encoding**:
  - ➢ **Purpose**: Converting categorical variables into numerical formats makes them suitable for machine learning algorithms.
  - ➢ **Implementation**: Techniques like one-hot encoding were used to encode categorical variables such as PackageId and SeasonId.

Figure 5.4.2 6- Encoding Data

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
X_encoded = encoder.fit_transform(X[['PackageId', 'SeasonId']])
```

- **Interaction Features**:

  - **Purpose**: Interaction features represent the combined effect of two or more original features, providing the model with more complex relationships within the data.

  - **Example**: Interaction between SeasonId and Year can be created to capture seasonal trends over different years.

  - **Implementation**: New columns are created by multiplying existing feature value

Figure 5.4.2 7- Interaction Features for different relations

```
# Standardize the new interaction features
interaction_features
= ['Price_SeasonId_Interaction', 'Price_Year_Interaction', 'SeasonId_Year_Interaction']
scaler = StandardScaler()
demand_data_train[interaction_features] = scaler.fit_transform(demand_data_train[interaction_features])

# Now, include these interaction features in the final feature set
X = demand_data_train[['PackageId', 'SeasonId', 'Year', 'Price', 'Dmin', 'Dmax'] + interaction_features]
y = demand_data_train['Demand']

# Continue with your model training process as before
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **Domain-Specific Features**:

  - **Purpose**: Based on our understanding of the tourism industry, we engineered features such as the minimum and maximum demand capacity (Dmin and Dmax) to better capture the constraints and requirements of tour packages.

  - **Implementation**: These features were included directly in the dataset based on domain knowledge.as we mentioned in **Figure 5.4.2 5- Temporal Features of the data**

5.4.3 Data Augmentation

❖ **Rationale for Data Augmentation**

Data augmentation involves creating additional synthetic data points to supplement the original dataset. This process is crucial in scenarios where the original dataset might be limited in size or diversity. By augmenting the data, we aim to enhance the robustness and generalization capability of our machine learning models.

In our case, augmenting the dataset allows us to generate variations in the existing data, which helps in capturing a broader range of scenarios and

patterns. This is particularly beneficial in tourism demand prediction, where historical data may not fully represent all possible future trends and fluctuations.

❖ **Data Augmentation Process**

The data augmentation process involves several techniques tailored to our specific dataset and objectives:

- **Feature-based Augmentation:** We create variations in existing features such as Price, Dmin, Dmax, and temporal features like Year. This variation helps simulate different demand scenarios under varying conditions.

Figure 5.4.3 1- Feature- based Augmentation

```
# Example of feature-based augmentation
import numpy as np

# Create variations in Price
price_augmented = np.random.normal(loc=X['Price'].mean(), scale=X['Price'].std(), size=len(X))

# Create variations in Dmin and Dmax
dmin_augmented = np.random.randint(low=X['Dmin'].min(), high=X['Dmin'].max(), size=len(X))
dmax_augmented = np.random.randint(low=X['Dmax'].min(), high=X['Dmax'].max(), size=len(X))
```

- **Synthetic Sample Generation:** Using statistical methods and domain knowledge, we generate synthetic samples that resemble the original data distribution. For instance, using techniques like bootstrapping or sampling from probability distributions based on observed data patterns.

Figure 5.4.3 2- Data Generation using suitable distribution

```
# Example of synthetic sample generation
from sklearn.utils import resample

# Bootstrap resampling to generate synthetic samples
synthetic_samples = resample(X, replace=True, n_samples=len(X), random_state=42)
```

- **Augmenting Categorical Variables:** Techniques like oversampling or introducing noise can be applied to categorical variables such as PackageId and SeasonId. This enriches the dataset by ensuring a more balanced representation of different categories.

Figure 5.4.3 3- Augmenting Categorical Variables

```
from imblearn.over_sampling import RandomOverSampler

# Oversampling PackageId and SeasonId
oversampler = RandomOverSampler(random_state=42)
X_resampled, y_resampled = oversampler.fit_resample(X, y)
```

❖ **Integration with Original Data**

After generating synthetic data, the next step is to integrate it seamlessly with the original dataset:

- **Concatenation:** Synthetic data samples are concatenated with the original dataset, ensuring that the augmented dataset maintains the same structure and format.

Figure 5.4.3 4- Concatenation

```
# Concatenate synthetic data with original dataset
augmented_data = pd.concat([X, synthetic_samples], axis=0)
```

- **Validation and Quality Assurance:** Before final integration, the augmented data is validated to ensure it aligns with the original data's statistical properties and distributions. This step helps maintain the integrity and reliability of the dataset for training and testing purposes.

Figure 5.4.3 5- Validation of data augmentation

```
# Validate augmented data
from scipy.stats import ks_2samp

# Perform Kolmogorov-Smirnov test for distribution similarity
ks_statistic, p_value = ks_2samp(X['Price'], synthetic_samples['Price'])
if p_value < 0.05:
    print("Warning: Synthetic data distribution differs significantly from original data.")
```

By augmenting our dataset, we not only increase its size but also its diversity, thereby improving the predictive power and reliability of our machine learning models for tourism demand forecasting.

This approach allows us to leverage both real-world data and simulated scenarios to build more robust predictive models, capable of handling a wider range of possible outcomes in tourism demand prediction tasks.

5.4.4 Model Training and Evaluation

❖ **Selection of Machine Learning Models**

In this section, we describe the process of selecting appropriate machine learning models for predicting tourism package demand. A variety of regression models were considered to ensure robust and accurate predictions. The selected models are as follows:

- **Gradient Boosting Regressor**: This ensemble method builds multiple decision trees sequentially, where each tree corrects the errors of its predecessor. It is effective for capturing complex patterns in the data.

- **Random Forest Regressor**: Another ensemble method, the Random Forest aggregates the predictions of numerous decision trees to improve accuracy and control over-fitting.

- **Linear Regression**: A fundamental approach that models the relationship between the dependent variable and one or more independent variables using a linear equation.

- **Support Vector Regressor (SVR)**: This model uses support vector machines for regression tasks, finding the best fit by minimizing the error within a threshold.

- **K-Nearest Neighbors (KNN)**: This non-parametric method predicts the target value based on the average value of its k-nearest neighbors.

- **Decision Tree Regressor**: This model splits the data into subsets based on feature values, making decisions that lead to the most significant information gain.

- **Neural Network (MLP Regressor)**: A multi-layer perceptron, which is a type of neural network, capable of capturing intricate relationships in the data through multiple layers of nodes.

The selection of these models was based on their ability to handle different types of data distributions and their proven effectiveness in regression tasks.

Figure 5.4.4 1- Selection of Machine Learning Models and Test Hyperparameter

```python
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor

# Define models and their hyperparameters for GridSearchCV
models = {
    'Gradient Boosting': {
        'model': GradientBoostingRegressor(random_state=42),
        'params': {
            'n_estimators': [100, 200, 300],
            'learning_rate': [0.01, 0.1, 0.2],
            'max_depth': [3, 5, 7]
        }
    },
    'Random Forest': {
        'model': RandomForestRegressor(random_state=42),
        'params': {
            'n_estimators': [100, 200, 300],
            'max_depth': [None, 10, 20],
            'min_samples_split': [2, 5, 10]
        }
    },
    'Linear Regression': {
        'model': LinearRegression(),
        'params': {}
    },
    'Support Vector Regressor': {
        'model': SVR(),
        'params': {
            'kernel': ['linear', 'poly', 'rbf'],
            'C': [0.1, 1, 10]
        }
    },
    'K-Nearest Neighbors': {
        'model': KNeighborsRegressor(),
        'params': {
            'n_neighbors': [3, 5, 7],
            'weights': ['uniform', 'distance']
        }
    },
    'Decision Tree': {
        'model': DecisionTreeRegressor(random_state=42),
        'params': {
            'max_depth': [None, 10, 20],
            'min_samples_split': [2, 5, 10]
        }
    },
    'Neural Network': {
        'model': MLPRegressor(random_state=42, max_iter=1000),
        'params': {
            'hidden_layer_sizes': [(50,), (100,), (50, 50)],
            'activation': ['tanh', 'relu'],
            'solver': ['sgd', 'adam'],
            'alpha': [0.0001, 0.001]
        }
    }
}

```

❖ **Hyperparameter Tuning**

Hyperparameter tuning is crucial for optimizing the performance of machine learning models. We employed GridSearchCV to systematically search for the best combination of hyperparameters for each model. The hyperparameters and their ranges for each model are as follows:

- **Gradient Boosting Regressor**:
  - n_estimators: [100, 200, 300]
  - learning_rate: [0.01, 0.1, 0.2]
  - max_depth: [3, 5, 7]

- **Random Forest Regressor:**
  - n_estimators: [100, 200, 300]
  - max_depth: [None, 10, 20]
  - min_samples_split: [2, 5, 10]

- **Support Vector Regressor:**
  - kernel: ['linear', 'poly', 'rbf']
  - C: [0.1, 1, 10]

- **K-Nearest Neighbors Regressor:**
  - n_neighbors: [3, 5, 7]
  - weights: ['uniform', 'distance']

- **Decision Tree Regressor:**
  - max_depth: [None, 10, 20]
  - min_samples_split: [2, 5, 10]

- **Neural Network (MLP Regressor):**
  - hidden_layer_sizes: [(50,), (100,), (50, 50)]
  - activation: ['tanh', 'relu']
  - solver: ['sgd', 'adam']
  - alpha: [0.0001, 0.001]

Figure 5.4.4 2- Hyperparameter Tuning Parameters

```
from sklearn.model_selection import GridSearchCV

# Train and evaluate each model using GridSearchCV
results = {}
for name, model_info in models.items():
    grid_search = GridSearchCV(model_info['model'], model_info['params'], cv=3, scoring='r2')
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_

    y_pred = best_model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    accuracy = r2 * 100

    results[name] = {
        'model': best_model,
        'mse': mse,
        'r2': r2,
        'accuracy': accuracy,
        'y_pred': y_pred
    }

    print(f'{name}:')
    print(f'  Mean Squared Error: {mse}')
    print(f'  R-squared: {r2}')
    print(f'  Accuracy: {accuracy:.2f}%')
    print(f'  Best Hyperparameters: {grid_search.best_params_}')
    print()
```

❖ **Model Evaluation Metrics**

The evaluation of the models was based on several metrics to ensure comprehensive assessment:

- **Mean Squared Error (MSE)**: Measures the average squared difference between the predicted and actual values.
- **R-squared (R2)**: Indicates the proportion of the variance in the dependent variable that is predictable from the independent variables.
- **Accuracy**: Represents the percentage of correctly predicted values based on the R2 score.

Figure 5.4.4 3- Model Evaluation Metrics

```
from sklearn.metrics import mean_squared_error, r2_score

# Example of evaluation metrics calculation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
accuracy = r2 * 100

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
print(f'Accuracy: {accuracy:.2f}%')
```

The results for each model, including MSE, R^2, and accuracy, were computed and compared to select the best performing model.

❖ Cross-Validation

Cross-validation was used to validate the models and ensure their robustness and generalizability. We applied 3-fold cross-validation, where the data is split into three subsets.

Each model is trained on two subsets and tested on the remaining one. This process is repeated three times, with each subset used as the test set once. The average performance across the folds was considered to mitigate over-fitting and obtain a reliable estimate of the model's performance.

Figure 5.4.4 4- Cross-Validation

```
from sklearn.model_selection import train_test_split

# Example of splitting the data and performing cross-validation
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
grid_search = GridSearchCV(GradientBoostingRegressor(random_state=42), param_grid, cv=3, scoring='r2')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

y_pred = best_model.predict(X_test)

# Cross-validation results
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Best Hyperparameters: {grid_search.best_params_}')
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

❖ Summary of Model Performance

The performance of each model was summarized and visualized to compare their effectiveness in predicting tourism package demand.

Figure 5.4.4 5- Compare Models for MSE and R-Squared

```
# Plotting the results
plt.figure(figsize=(14, 8))

# Mean Squared Error
plt.subplot(2, 2, 1)
mse_values = [results[name]['mse'] for name in models]
plt.bar(models.keys(), mse_values)
plt.ylabel('Mean Squared Error')
plt.title('Mean Squared Error of Different Models')
plt.xticks(rotation=45)

# R-squared
plt.subplot(2, 2, 2)
r2_values = [results[name]['r2'] for name in models]
plt.bar(models.keys(), r2_values)
plt.ylabel('R-squared')
plt.title('R-squared of Different Models')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

This code creates a graph that compares models' performance based on MSE and R-Squared to choose the best one of them to our case.

And the following graph compares predicted demand vs actual demand for the tour packages.

Figure 5.4.4 6- Actual vs Predicted Demand

```
# Plot actual vs predicted demand for each model
plt.figure(figsize=(15, 10))

for i, (name, result) in enumerate(results.items(), 1):
    plt.subplot(3, 3, i)
    plt.scatter(y_test, result['y_pred'], alpha=0.7)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--r')
    plt.xlabel('Actual Demand')
    plt.ylabel('Predicted Demand')
    plt.title(f'Actual vs Predicted Demand - {name}')
    plt.grid(True)

plt.tight_layout()
plt.show()
```

And we will show results of testing model in 5.6 Result and Discussion

5.5 Implementation of AI Prediction Model

5.5.1 Data Integration and Merging

❖ **Merging PackageSeason, TourPackage, and Season Tables**

In the first step of data integration, we combined data from three different tables: PackageSeason, TourPackage, and Season. These tables contain crucial information about various tour packages, the seasons during which they are available, and specific details about each package and season. The merging of these tables is essential to create a comprehensive dataset that includes all relevant features required for accurate demand prediction.

The PackageSeason table provides the relationship between tour packages and seasons, indicating which packages are available in which seasons. The TourPackage table contains detailed information about each tour package, including its cost and price. The Season table holds information about different seasons. By merging these tables, we create a unified dataset that encapsulates all necessary attributes, such as package details, seasonal information, and associated demand.

The following code snippet demonstrates the merging process:

Figure 5.5.1 1- Data Integration and Merging from Different Tables

```
# Load the PackageSeason, TourPackage, and Season sheets
package_season_data = excel_data.parse('PackageSeason')
tour_package_data = excel_data.parse('TourPackage')
season_data = excel_data.parse('Season')

# Merge data for demand prediction
demand_data
= package_season_data.merge(tour_package_data, on='PackageId').merge(season_data, on='SeasonId')
```

This merging operation is performed using the merge function in pandas, which joins the tables based on common keys: PackageId and SeasonId. The result is a consolidated dataset that will be used for further analysis and model training.

❖ **Data Preparation for Model Training**

After merging the tables, the next step is to prepare the data for model training. This involves several sub-steps, including filtering the data for specific years, selecting relevant features, standardizing the features, and splitting the data into training and testing sets.

- **Filtering Data for Specific Years**

  For this project, we focus on data from the years 2019 to 2023. This range was chosen to provide enough historical data for training the model while also including recent years for more relevant predictions.

  As we mentioned in **Figure 5-4-2 1– Filter data for Training and Testing**

- **Selecting Relevant Features**

  We select specific features that are deemed important for predicting demand. These features include PackageId, SeasonId, Year, Price, Dmin, and Dmax. The target variable for our prediction is Demand.

Figure 5.5.1 2- Select Suitable Features

```
# Select features and target variable for demand prediction
X = demand_data_train[['PackageId', 'SeasonId', 'Year', 'Price', 'Dmin', 'Dmax']]
y = demand_data_train['Demand']
```

- **Standardizing the Features**

  To ensure that all features contribute equally to the model, we standardize them. Standardization transforms the data to have a mean of zero and a standard deviation of one, which helps in improving the performance of many machine learning algorithms.

  As we mentioned in **Figure 5.4.2 4- Standardization Part**

- **Splitting Data into Training and Testing Sets**

  Finally, we split the standardized data into training and testing sets. This split allows us to train the model on one subset of data and evaluate its performance on another subset that the model has not seen during training.

Figure 5.5.1 3- Split data into train and test

```
# Split into training and testing sets for demand prediction
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

By following these steps, we ensure that the data is well-prepared for training machine learning models to predict tour package demand accurately. The integrated and preprocessed dataset forms the foundation for building robust predictive models.

5.5.2 Model Implementation

In this section, we describe the implementation of various machine learning models used to predict demand for tour packages. We will explore the models in detail, including Gradient Boosting Regressor, Random Forest, Linear Regression, Support Vector Regressor, K-Nearest Neighbors, Decision Tree, and Neural Network.

❖ Gradient Boosting Regressor

Gradient Boosting Regressor is an ensemble learning technique that builds multiple decision trees sequentially. Each tree attempts to correct the errors of its predecessor. This method is powerful for handling complex datasets with non-linear relationships.

We use GridSearchCV to find the best hyperparameters for our Gradient Boosting Regressor model. This involves specifying a range of values for parameters like n_estimators, learning_rate, and max_depth, and using cross-validation to select the optimal combination.

Figure 5.5.2 1- Gradient Boosting Regressor

```
# Hyperparameter tuning for Gradient Boosting Regressor
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}
grid_search = GridSearchCV(GradientBoostingRegressor(random_state=42), param_grid, cv=3, scoring=
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# Predict on the test set
y_pred = best_model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Gradient Boosting Regressor:')
print(f'  Mean Squared Error: {mse}')
print(f'  R-squared: {r2}')
print(f'  Best Hyperparameters: {grid_search.best_params_}')
```

❖ Random Forest

Random Forest is another ensemble learning method that builds multiple decision trees and averages their predictions. Unlike Gradient Boosting, Random Forest trees are built independently and combined for the final prediction.

Figure 5.5.2 2- Random Forest

```
# Hyperparameter tuning for Random Forest
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=3, scoring='r2')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# Predict on the test set
y_pred = best_model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Random Forest:')
print(f'  Mean Squared Error: {mse}')
print(f'  R-squared: {r2}')
print(f'  Best Hyperparameters: {grid_search.best_params_}')
```

❖ Linear Regression

Linear Regression is a simple yet effective model that assumes a linear relationship between the input features and the target variable. It is often used as a baseline model.

Figure 5.5.2 3- Linear Regression

```
# Train Linear Regression
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Linear Regression:')
print(f'  Mean Squared Error: {mse}')
print(f'  R-squared: {r2}')
```

❖ Support Vector Regressor

Support Vector Regressor (SVR) is a powerful model that uses the concept of support vectors to perform regression tasks. It is effective in high-dimensional spaces and with non-linear relationships.

Figure 5.5.2 4- Support Vector Regressor

```
# Hyperparameter tuning for Support Vector Regressor
param_grid = {
    'kernel': ['linear', 'poly', 'rbf'],
    'C': [0.1, 1, 10]
}
grid_search = GridSearchCV(SVR(), param_grid, cv=3, scoring='r2')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# Predict on the test set
y_pred = best_model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Support Vector Regressor:')
print(f'  Mean Squared Error: {mse}')
print(f'  R-squared: {r2}')
print(f'  Best Hyperparameters: {grid_search.best_params_}')
```

❖ K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric method that predicts the target variable based on the k closest training examples in the feature space. It is simple and effective for many types of datasets.

Figure 5.5.2 5- K-Nearest Neighbors

```
# Hyperparameter tuning for K-Nearest Neighbors
param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance']
}
grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, cv=3, scoring='r2')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# Predict on the test set
y_pred = best_model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'K-Nearest Neighbors:')
print(f'  Mean Squared Error: {mse}')
print(f'  R-squared: {r2}')
print(f'  Best Hyperparameters: {grid_search.best_params_}')
```

❖ Decision Tree

Decision Tree is a model that splits the data into subsets based on feature values, making it easy to interpret. It can capture complex relationships but is prone to overfitting.

Figure 5.5.2 6- Decision Tree

```
# Hyperparameter tuning for Decision Tree
param_grid = {
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(DecisionTreeRegressor(random_state=42), param_grid, cv=3, scoring='r2')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# Predict on the test set
y_pred = best_model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Decision Tree:')
print(f'  Mean Squared Error: {mse}')
print(f'  R-squared: {r2}')
print(f'  Best Hyperparameters: {grid_search.best_params_}')
```

❖ **Neural Network**

Neural Network is a complex model inspired by the human brain's structure. It can capture intricate patterns in data, making it suitable for complex predictive tasks.

Figure 5.5.2 7- Neural Network

```
# Hyperparameter tuning for Neural Network
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.001]
}
grid_search = GridSearchCV(MLPRegressor(random_state=42, max_iter=1000),
                           param_grid, cv=3, scoring='r2')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# Predict on the test set
y_pred = best_model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Neural Network:')
print(f'  Mean Squared Error: {mse}')
print(f'  R-squared: {r2}')
print(f'  Best Hyperparameters: {grid_search.best_params_}')
```

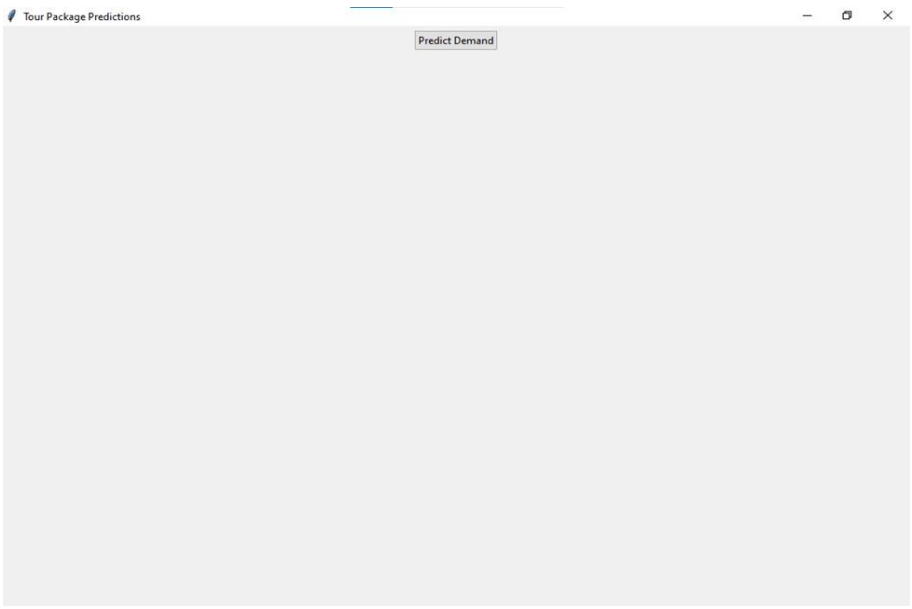
In this section, we have detailed the implementation of various machine learning models for demand prediction, highlighting the specific steps and hyperparameters used for each model. The models were trained and evaluated to select the best-performing ones for our task.

And we will clarify the accuracy of each model in the results and discussion part and indicate which of them is the best in accurately predicting demand for our model

5.5.3 Graphical User Interface (GUI) Development

The development of the Graphical User Interface (GUI) for the demand prediction model aims to create an intuitive, user-friendly platform that allows users to input relevant information, generate demand predictions, and visualize analysis results. The GUI was built using the Tkinter library in Python, which provides a robust framework for creating graphical applications.

Figure 5.5.3 1- Graphical User Interface (GUI) Development



❖ **Design and Layout**

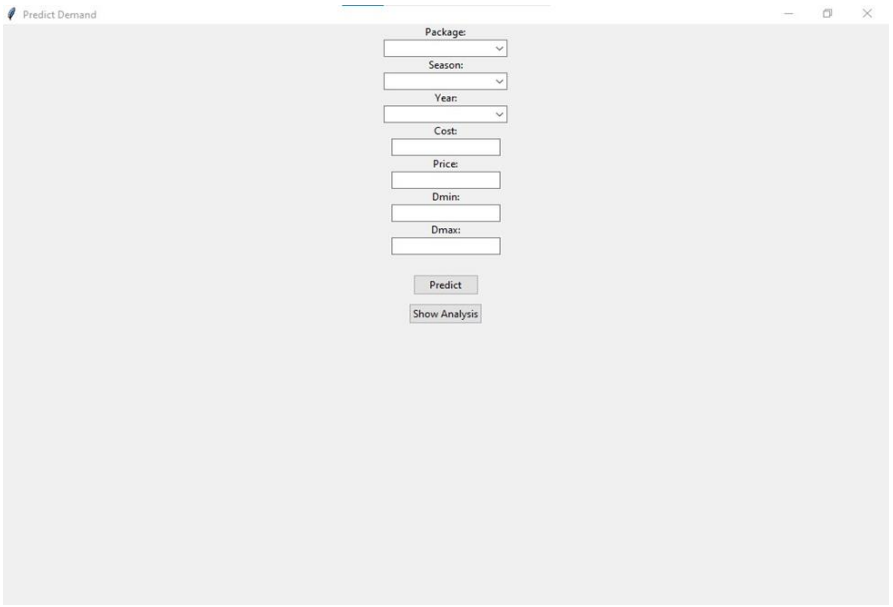
The GUI was meticulously designed to ensure ease of use and clarity. The main window features a button that opens a new window specifically for demand prediction. This secondary window is structured to guide users through the process of inputting necessary details. Dropdown menus are used for selecting the tour package, season, and year, while entry fields are provided for the user to input the price, Dmin (minimum demand), and Dmax (maximum demand). This layout ensures that all required inputs are organized logically and are easy to find and use.

Here's an example of how the main window and the demand prediction window are set up:

Figure 5.5.3 2- Design and Layout

```
# Create the main window
root = tk.Tk()
root.title("Tour Package Predictions")

# Function to open a new window for demand prediction
def open_demand_window():
    demand_window = tk.Toplevel(root)
    demand_window.title("Predict Demand")

    # Package combo box
    package_label = ttk.Label(demand_window, text="Package:")
    package_label.pack()
    package_combo = ttk.Combobox(demand_window)
    package_combo['values'] = tour_package_data['Name'].tolist()
    package_combo.pack()

    # Season combo box
    season_label = ttk.Label(demand_window, text="Season:")
    season_label.pack()
    season_combo = ttk.Combobox(demand_window)
    season_combo.pack()
```

Figure 5.3.3 3- Demand Window



❖ User Input and Interaction

The interaction with the GUI is straightforward and user centric. Users begin by selecting a package from a dropdown menu. This selection dynamically updates the options available in the season dropdown menu, ensuring that only relevant seasons are displayed. The user then selects a year and inputs the price, Dmin, and Dmax values. This dynamic and responsive design helps prevent user errors and enhances the overall user experience. The interface also includes error messages and prompts to guide users in case they miss any required fields.

Figure 5.5.3 4- Load Seasons

```
def load_seasons(event):
    package_name = package_combo.get()
    package_id =
    tour_package_data[tour_package_data['Name'] == package_name]['PackageId'].values[0]
    seasons =
    package_season_data[package_season_data['PackageId'] == package_id]['SeasonId'].unique()
    season_names = season_data[season_data['SeasonId'].isin(seasons)]['Name'].tolist()
    season_combo['values'] = season_names

    # Clear all fields
    season_combo.set('')
    price_entry.delete(0, tk.END)
    dmin_entry.delete(0, tk.END)
    dmax_entry.delete(0, tk.END)
    result_label.config(text="")

package_combo.bind("<<ComboboxSelected>>", load_seasons)
```

And we can see this in this example of loading corresponding season that store in database for the package.

Figure 5.5.3 5- Cairo Stopover Package and load corresponding seasons in combo box

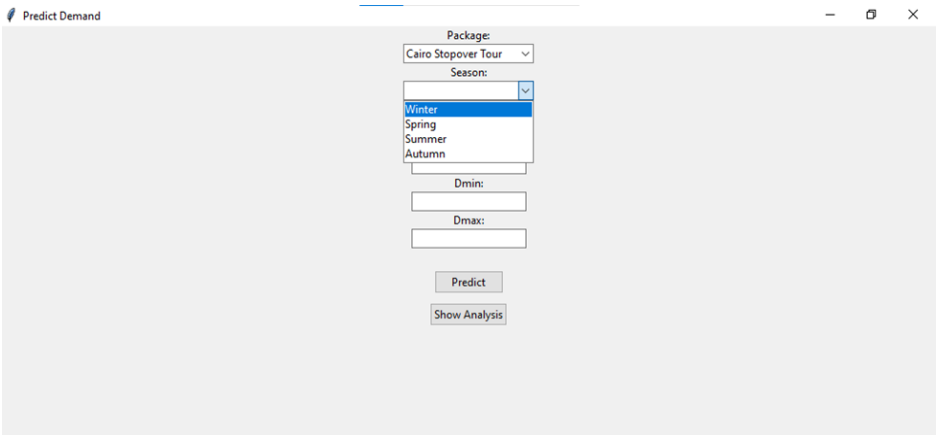
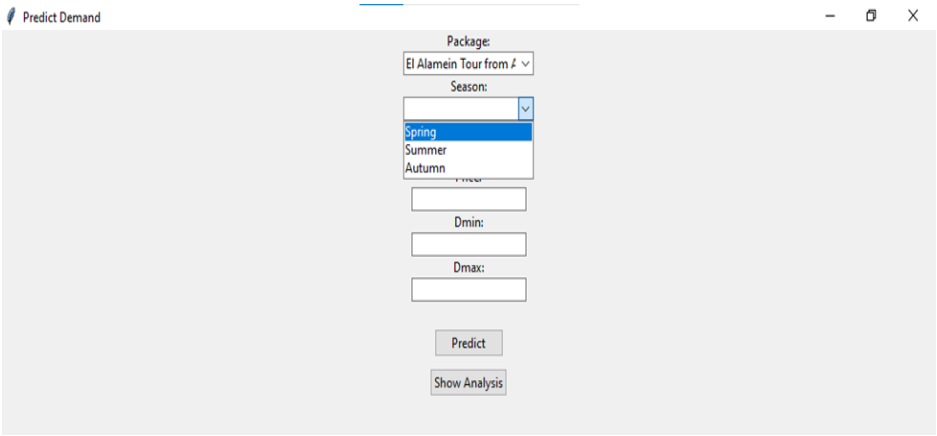


Figure 5.5.3 6- El Alamein Tour and corresponding seasons



❖ **Demand Prediction Functionality**

The core functionality of the GUI is to predict demand based on the user's input. After the user selects a package and season and inputs the year, price, Dmin, and Dmax, the system processes this information using the pre-trained Gradient Boosting Regressor model. The predicted demand is then displayed to the user in the same window. This real-time feedback allows users to quickly obtain demand predictions and make informed decisions based on the results.

Figure 5.5.3 7- Predict Demand Function

```
def predict_demand():
    package_name = package_combo.get()
    season_name = season_combo.get()
    year = int(year_combo.get())

    if not package_name or not season_name:
        result_label.config(text="Please select a package and a season.")
        return

    package_id = tour_package_data[tour_package_data['Name'] == package_name]['PackageId'].values[0]
    season_id = season_data[season_data['Name'] == season_name]['SeasonId'].values[0]
    price = float(price_entry.get())
    dmin = int(dmin_entry.get())
    dmax = int(dmax_entry.get())

    input_data = pd.DataFrame([[package_id, season_id, year, price, dmin, dmax]],
                              columns=['PackageId', 'SeasonId', 'Year', 'Price', 'Dmin', 'Dmax'])
    input_data_scaled = scaler.transform(input_data)
    prediction = int(best_model.predict(input_data_scaled)[0])

    result = f"Predicted Demand for {package_name} in {season_name} ({year}): {prediction}"
    result_label.config(text=result)
```

And we will see the examples result of demand prediction in result and discussion part.

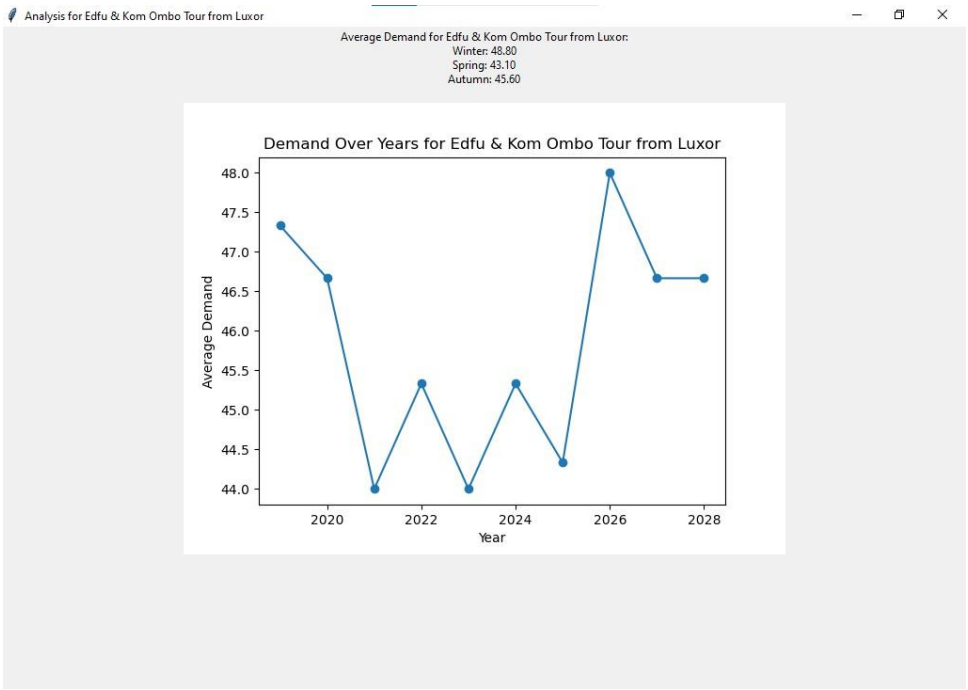
❖ **Analysis and Visualization**

In addition to demand prediction, the GUI offers robust analysis and visualization capabilities. Users can access detailed analysis of average demand for selected packages across different seasons. This analysis is presented in a new window, providing insights into historical trends and helping users understand the performance of different packages. Additionally, the GUI includes a feature to visualize demand trends over the years. This visualization is presented as a line plot, allowing users to see the changes in demand over time for a selected package. This feature helps users to identify patterns and make data-driven decisions.

Figure 5.5.3 8- Show Analysis Function

```
def show_analysis():
    package_name = package_combo.get()
    if not package_name:
        result_label.config(text="Please select a package to analyze.")
        return

    package_id = tour_package_data[tour_package_data['Name'] == package_name]['PackageId'].values[0]
    package_data = demand_data[demand_data['PackageId'] == package_id]

    analysis_window = tk.Toplevel(root)
    analysis_window.title(f"Analysis for {package_name}")

    # Example analysis: Average demand per season
    analysis_result = package_data.groupby('SeasonId')['Demand'].mean()
    analysis_result = analysis_result.reset_index()
    analysis_result = analysis_result.merge(season_data[['SeasonId', 'Name']], on='SeasonId')

    analysis_text = f"Average Demand for {package_name}:\n"
    for _, row in analysis_result.iterrows():
        analysis_text += f"{row['Name']}: {row['Demand']:.2f}\n"

    analysis_label = tk.Label(analysis_window, text=analysis_text)
    analysis_label.pack()

    # Plot demand over the years
    fig, ax = plt.subplots()
    package_data.groupby('Year')['Demand'].mean().plot(kind='line', ax=ax, marker='o')
    ax.set_title(f'Demand Over Years for {package_name}')
    ax.set_xlabel('Year')
    ax.set_ylabel('Average Demand')

    canvas = FigureCanvasTkAgg(fig, master=analysis_window)
    canvas.draw()
    canvas.get_tk_widget().pack()
```

Figure 5.5.3 9- Analysis for Edfu & Kom Ombo Tour from Luxor



And we will see the examples result of demand prediction in result and discussion part.

In this section, we detailed the design of the GUI development for the demand prediction model emphasizes a user-friendly design, interactive input fields, and robust functionality for both prediction and analysis. The intuitive design ensures that users can easily navigate the interface, input necessary details, and obtain accurate predictions and insightful analyses, all of which contribute to optimizing decision-making processes in the tourism industry.

5.6 Results and Discussion of AI Prediction Model

5.6.1 Model Performance

This section provides an in-depth analysis of the performance of various machine learning models used for demand prediction. The evaluation metrics include Mean Squared Error (MSE), R-squared (R^2), and Accuracy, along with the best hyperparameters identified through grid search.

❖ **Gradient Boosting Regressor Results**

The Gradient Boosting Regressor demonstrated exceptional performance in predicting demand. The model achieved a Mean Squared Error (MSE) of 39.19, indicating a low level of prediction error. The R-squared value of 0.93 signifies that the model explains 93.46% of the variance in the demand data, which is a very high level of accuracy. The best hyperparameters for this model were identified as a learning rate of 0.2, a maximum depth of 7, and 300 estimators. These settings enable the model to capture complex patterns in the data, leading to highly accurate predictions.

Figure 5.6.1 1- Gradient Boosting Accuracy

```
Gradient Boosting:
  Mean Squared Error: 39.190083963088576
  R-squared: 0.9346112647359792
  Accuracy: 93.46%
  Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 300}
```

❖ **Random Forest Results**

The Random Forest model also performed admirably, with a Mean Squared Error (MSE) of 41.85 and an R-squared value of 0.93, resulting in an accuracy of 93.02%. This indicates that the model is nearly as effective as the Gradient Boosting Regressor. The best hyperparameters for the Random Forest were a maximum depth of None (allowing nodes to expand until all leaves are pure or contain fewer than the minimum samples for splitting), a minimum sample split of 2, and 300 estimators. This configuration allows the model to handle complex interactions between features while maintaining robustness against overfitting.

Figure 5.6.1 2- Random Forest Accuracy

```
Random Forest:
  Mean Squared Error: 41.852644593818226
  R-squared: 0.9301687768767757
  Accuracy: 93.02%
  Best Hyperparameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 300}
```

❖ **Linear Regression Results**

Linear Regression, a simpler and more interpretable model, did not perform well for this task. The model's Mean Squared Error (MSE) was 597.89, with an R-squared value of 0.002, indicating that it explains only 0.24% of the variance in the demand data. This very low accuracy suggests that linear relationships are insufficient to capture the complexity of the demand patterns. Consequently, no hyperparameters were optimized for this model.

Figure 5.6.1 3- Linear Regression Accuracy

```
Linear Regression:
  Mean Squared Error: 597.8942008213903
  R-squared: 0.002412302810436251
  Accuracy: 0.24%
  Best Hyperparameters: {}
```

❖ **Support Vector Regressor Results**

The Support Vector Regressor (SVR) also showed poor performance, with a Mean Squared Error (MSE) of 594.47 and an R-squared value of 0.008, corresponding to an accuracy of 0.81%. The best hyperparameters for the SVR were a regularization parameter (C) of 1 and a radial basis function (rbf) kernel. Despite tuning, the SVR struggled to model the nonlinear relationships in the data effectively.

Figure 5.6.1 4- Support Vector Regressor Accuracy

```
Support Vector Regressor:
  Mean Squared Error: 594.4728881582005
  R-squared: 0.008120769987987386
  Accuracy: 0.81%
  Best Hyperparameters: {'C': 1, 'kernel': 'rbf'}
```

❖ **K-Nearest Neighbors Results**

The K-Nearest Neighbors (KNN) model performed even worse, with a Mean Squared Error (MSE) of 643.82 and an R-squared value of -0.07, resulting in a negative accuracy of -7.42%. This indicates that the KNN model not only failed to capture the demand patterns but also performed worse than a simple mean prediction. The best hyperparameters for KNN were identified as 7 neighbors and uniform weights, but these settings were insufficient for achieving reliable predictions.

Figure 5.6.1 5- K-Nearest Neighbors Accuracy

```
K-Nearest Neighbors:
  Mean Squared Error: 643.8160732752252
  R-squared: -0.07420843532169252
  Accuracy: -7.42%
  Best Hyperparameters: {'n_neighbors': 7, 'weights': 'uniform'}
```

❖ **Decision Tree Results**

The Decision Tree model performed relatively well, achieving a Mean Squared Error (MSE) of 70.54 and an R-squared value of 0.88, which corresponds to an accuracy of 88.23%. The best hyperparameters for this model included a maximum depth of None and a minimum sample split of 2. These parameters allowed the tree to fully grow and capture intricate decision boundaries within the data, resulting in strong predictive performance.

Figure 5.6.1 6- Decision Tree Accuracy

```
Decision Tree:
  Mean Squared Error: 70.5377456049638
  R-squared: 0.8823076271582367
  Accuracy: 88.23%
  Best Hyperparameters: {'max_depth': None, 'min_samples_split': 2}
```

❖ **Neural Network Results**

The Neural Network model, configured with two hidden layers of 50 neurons each, an activation function of 'tanh', an alpha value of 0.001, and a solver of 'sgd', achieved moderate performance. The Mean Squared Error (MSE) was 382.46, and the R-squared value was 0.36, resulting in an accuracy of 36.19%. Although the model was able to capture some non-linear relationships, its performance was limited compared to ensemble methods like Gradient Boosting and Random Forest**.**

Figure 5.6.1 7- Neural Network Accuracy

```
Neural Network:
  Mean Squared Error: 382.45692360109837
  R-squared: 0.36186984057502025
  Accuracy: 36.19%
  Best Hyperparameters: {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (50, 50), 'solver':
'sgd'}
```

5.6.2 Comparative Analysis of Models

❖ **Accuracy and Error Metrics**

The performance of each model is evaluated based on key metrics such as Mean Squared Error (MSE) and R-squared (R^2) values. These metrics provide insight into the accuracy and reliability of the models in predicting demand.

- **Gradient Boosting Regressor**:
  - Mean Squared Error: 39.19
  - R-squared: 0.93
  - Accuracy: 93.46%
  - Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 300}

- **Random Forest**:
  - Mean Squared Error: 41.85
  - R-squared: 0.93
  - Accuracy: 93.02%
  - Best Hyperparameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 300}

- **Linear Regression**:
  - Mean Squared Error: 597.89
  - R-squared: 0.00
  - Accuracy: 0.24%
  - Best Hyperparameters : {}

- **Support Vector Regressor**:
  - Mean Squared Error: 594.47
  - R-squared: 0.01
  - Accuracy: 0.81%
  - Best Hyperparameters: {'C': 1, 'kernel': 'rbf'}

- **K-Nearest Neighbors**:
  - Mean Squared Error: 643.82
  - R-squared: -0.07
  - Accuracy: -7.42%
  - Best Hyperparameters: {'n_neighbors': 7, 'weights': 'uniform'}

- **Decision Tree**:
  - Mean Squared Error: 70.54
  - R-squared: 0.88
  - Accuracy: 88.23%
  - Best Hyperparameters: {'max_depth': None, 'min_samples_split': 2}

- **Neural Network**:
  - Mean Squared Error: 382.46
  - R-squared: 0.36
  - Accuracy: 36.19%
  - Best Hyperparameters: {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (50, 50), 'solver': 'sgd'}

❖ **Strengths and Weaknesses of Each Model**

- **Gradient Boosting Regressor**:

  - **Strengths** :High accuracy and low MSE indicate strong predictive capabilities. The model effectively captures complex patterns in the data.

  - **Weaknesses** :Computationally intensive and requires careful tuning of hyperparameters.

- **Random Forest**:

  - **Strengths** :Robust to overfitting and handles large datasets well. Provides high accuracy similar to Gradient Boosting.

  - **Weaknesses** :Interpretation can be challenging due to the ensemble nature of the model.

- **Linear Regression**:
  - **Strengths** :Simple to implement and interpret. Useful as a baseline model.
  - **Weaknesses** :Very poor performance with high MSE and low R^2, indicating it cannot capture the complexities in the data.
- **Support Vector Regressor**:
  - **Strengths** :Effective in high-dimensional spaces. Can perform well with proper tuning.
  - **Weaknesses** :Poor performance in this context, likely due to the complexity of the data and inadequate tuning.
- **K-Nearest Neighbors**:
  - **Strengths** :Simple and intuitive. Can capture local patterns well.
  - **Weaknesses** :Very poor performance with negative R^2 and high MSE, indicating it is not suitable for this dataset.
- **Decision Tree**:
  - **Strengths** :Easy to interpret and visualize. Handles both numerical and categorical data well.
  - **Weaknesses** :Prone to overfitting without proper tuning.
- **Neural Network**:
  - **Strengths** :Capable of capturing non-linear relationships and complex patterns in the data.
  - **Weaknesses** :Requires a large amount of data and computational power. Performance in this case is moderate, indicating a need for further tuning and possibly more data.

❖ **Graphical Comparison of Actual vs Predicted Demand**

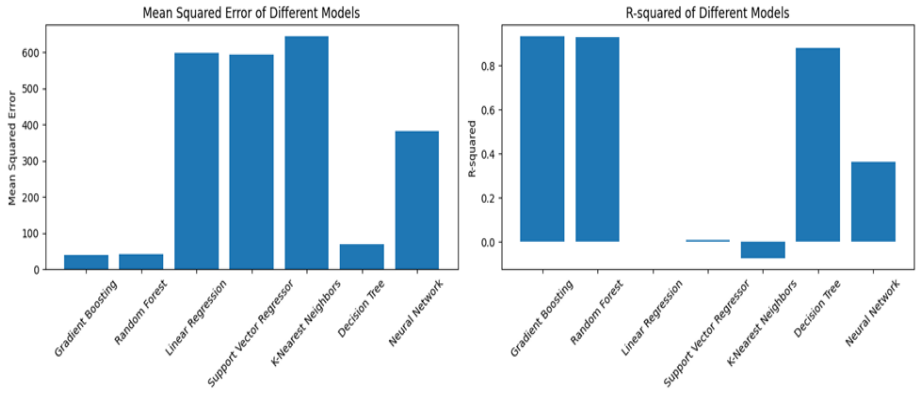
The graphical comparison of actual versus predicted demand for each model is illustrated in the figures below. These plots help visualize the accuracy and distribution of predictions made by each model.

- **Bar Charts Comparing MSE and R-squared Values**:The bar charts provide a visual comparison of Mean Squared Error (MSE) and R-squared (R^2) values for each model.

- **Mean Squared Error (MSE)** :Lower MSE values indicate better model performance. Gradient Boosting and Random Forest have the lowest MSE, showing they make the least prediction errors. Linear Regression, Support Vector Regressor, and K-Nearest Neighbors have significantly higher MSE values, indicating poor performance.

- **R-squared (R^2)** :Higher R^2 values indicate a better fit to the data. Gradient Boosting and Random Forest have the highest R^2 values, showing they explain most of the variability in the data. Linear Regression and Support Vector Regressor have near-zero or negative R^2 values, indicating they explain little to none of the data variability.

Figure 5.6.2 1- MSE and R-squared for different models



- **Scatter Plots of Actual vs Predicted Demand**:The scatter plots for each model show actual demand on the x-axis and predicted demand on the y-axis. Each point represents a data instance. The red dashed line represents perfect predictions, where actual demand equals predicted demand. The closer the points are to this line, the better the model's performance.

- **Gradient Boosting Regressor and Random Forest** :These models show a tight clustering of points around the red line, indicating strong predictive accuracy. The points follow a diagonal pattern, reflecting good model performance.

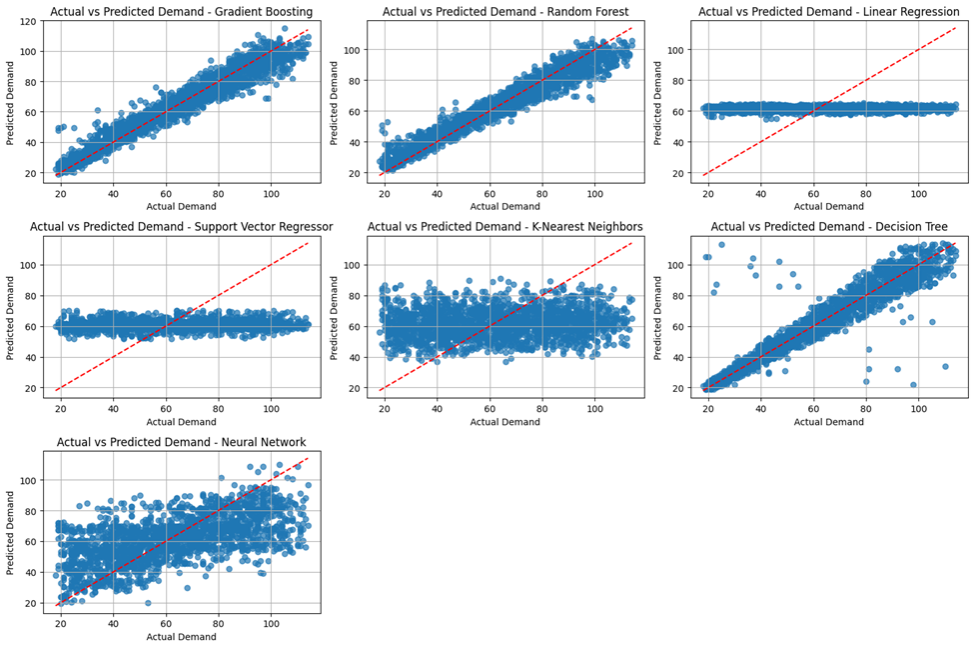
- **Linear Regression and Support Vector Regressor** :These models perform poorly, as shown by the spread of points far from the red line. This indicates significant errors in predictions.

- **K-Nearest Neighbors** :This model also shows poor performance with a wide scatter of points around the red line, indicating inaccuracies in predictions.

- **Decision Tree** :The points are relatively close to the red line, indicating a good fit, but some outliers suggest occasional prediction errors.

- **Neural Network** :The points show moderate clustering around the red line, suggesting fair performance but room for improvement.

Figure 5.6.2 2- Actual vs Predicted Demand for each model



CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this project, we have successfully developed a comprehensive solution for optimizing the tourism industry. Our approach involved several key steps:

- **Mathematical Model Creation:** We began by creating a mathematical model to understand and predict various aspects of tourism demand. This model laid the foundation for our subsequent data analysis and AI prediction efforts.

- **Data Collection and Analysis:** We collected extensive data relevant to the tourism industry and performed thorough data analysis. This step was crucial in identifying patterns and trends that informed our prediction model.

- **User Interface Development:** To facilitate the management of the tourism system, we developed a user-friendly interface using C# and .NET framework. This UI allows operators to easily interact with the system, manage resources, packages and budgets and make informed decisions based on the data insights.

- **AI Prediction Model:** We built an AI prediction model using Python and Tkinter to forecast tourism demand. This model leverages advanced machine learning techniques to provide accurate predictions of demand to a specific tour package based on different features like season and price, enabling better allocation of tourism packages.

By integrating these components, we have created a robust system that enhances the efficiency and effectiveness of tourism management. The AI prediction model's ability to forecast demand accurately helps in optimizing resource allocation, reducing waste, and improving.

6.2 Future Work

While our project has achieved significant milestones, there are several areas for future improvement and exploration:

- **Model Enhancement:** Further refinement of the AI prediction model can be achieved by incorporating additional data sources, such as real-time social media trends, weather forecasts, and economic indicators. This will enhance the model's predictive accuracy and adaptability to sudden changes in tourism demand.

- **Resource Allocation Optimization:** Future work can focus on integrating the prediction model with advanced resource allocation algorithms. This will enable automated and dynamic allocation of packages and budgets, ensuring optimal use of resources in real-time.

- **Scalability and Generalization:** Testing and adapting the model for different regions and tourism markets will be essential to ensure its scalability and general applicability. This will involve validating the model's performance across diverse geographical locations and tourism sectors.

- **Enhanced User Interface:** Developing a more sophisticated and intuitive user interface can further improve the practical application of the system. This interface should provide clear visualizations and actionable insights derived from the model's predictions.

- **Sustainability Considerations:** Future research should explore how the model can incorporate sustainability metrics. By predicting demand with an eye toward environmental and social impact, the tourism industry can move towards more sustainable practices.

REFRENCES

1. Third Edition Supply Chain Management Strategy, Planning, and Operation Peter Meindl, Stanford University Sunil Chopra, Kellogg School of Management Northwestern University.

2. Operations Research an Introduction by Hamdy A. Taha-2017.

3. Optimization of Tourism Industry by Supply chain strategy of Rokan Regional District to Increase Original Revenue Kasman Arifin ZA[1], Syapsan[2], Dewi Puspa Suri[3].

4. A Decision Method for Improving Tourism Industry Service Quality under Budget Constraints Xiaojun Chen[1], Qingliang Meng[2], Ling Zhang[2].

5. Forecasting Tourism Demand [Douglas_Frechtling].

6. Managing tourism emissions through optimizing the tourism demand mix: Concept and analysis Ya-Yen Sun[a], Pei-Chun Lin[b], James Higham[c].

7. Optimization model for designing personalized tourism packages Piya, Chefi Triki , Abdulwahab Al Maimani, Mahdi Mokhtarzadeh.

8. Digital Management and Optimization of Tourism Information Resources Based on Machine Learning Xueqiu Zhuang, Huihua Jiao, Li Kang.

9. Optimization of Tourism Management Based on Regional Tourism Competitiveness Evaluation: Evidence from Ningxia Hui Autonomous Region, China Shengrui Zhang, Lei Chi, Tongyan Zhang, Yingjie Wang.

10. Artificial Intelligence in the Tourism Industry: An Overview of Reviews Miguel Ángel García, Ana Julia Grilló.