PROJECT REPORT

on

“**FLIGHT PRICE PREDICTION MODEL ”**

# Submitted to

### Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur

### In partial fulfilment of the requirement of

**Bachelor of Commerce (Computer Application)**

# Submitted by

Mohit M Khule

Ayush T Goutam

Akansha Kamble

# Under the Guidance of

Ms. Neha Mittal



**Dr. Ambedkar Institute of Management Studies & Research DeekshaBhoomi Nagpur-440012**

**(2024-2025)**

CERTIFICATE

#### This is to certify that **Mohit M Khule , Ayush T Goutam and Akansha Kamble** has satisfactorily completed the project work entitled **“FLIGHT PRICE PREDICTION MODEL”** in not less than one academic session. This also certify that this project work is the result of the candidate’s own work and is of sufficiently high standard to warrant its presentation for the B.Com (Computer Application ) program.

To the best of my knowledge this project or its part has not been submitted to this university or any other university for any Degree.

**Guide Signature**

(Ms. Neha Mittal)

**Internal Examiner External Examiner**

##### Place: Nagpur

**Date:**  **Director**

**DECLARATION**

We, Mohit M Khule, Ayush T Goutam and Akansha Kamble hereby declare that the project entitled “FLIGHT PRICE PREDICTION MODEL” is the outcome of our own research work based on personal study during academic session 2024– 2025 and has not been submitted previously for award of any degree to this university or any other university.

Mohit M Khule

Ayush T Goutam

Akansha Kamble

ACKNOWLEDGEMENT

“Words have never expressed human sentiments. This only an attempt to express our deep gratitude which comes from our heart.”

It is a great pleasure for us to express our deep feeling of gratitude to our respected guideMs. Neha Mittal **(Asst. Professor,** **Dr. Ambedkar Institute of Management Studies & Research, Nagpur)** for his/her great encouragement and unfailing support which provided needed moral and confidence to carry on our work.

We are grateful to the Dr. S. S. Fulzele, (Director, Dr. Ambedkar Institute of Management Studies & Research, Nagpur) for making all facilities available for our work.

It is with profound gratitude that we wish to express our indebtedness to **Dr. Nirzar Kulkarni** (**Associate. Director, Dr. Ambedkar Institute of Management Studies & Research, Nagpur**) for his invaluable guidance and supervision in completion of this project work.

We are great to our parents for their lovable support. Last but not least we are thankful to our friends and other faculty member for their direct and indirect help for completion of this work.

**Content Page**

1. Introduction
2. Objectives
3. Preliminary system analysis .Present System in use
   1. Flaws in Present System
   2. Need of New System
   3. Feasibility study
   4. Project Category
4. Software Engineering Paradigm applied
   1. Modules
   2. System/Modular Chart
5. Software and Hardware Requirement Specification
6. Detailed system analysis
   1. Data Flow Diagram
   2. Numbers of Modules and Process Logic
   3. Data Structures and Tables
   4. Entity Relationship Diagram
7. System Design
   1. Form Design
   2. Source Code
   3. Input / output screen
8. Testing and validation checks
9. System Security and Measures
10. Implementation, Evaluation and maintenance.
11. Future scope of the project
12. Suggestion and conclusions
13. Bibliography and References

**Introduction**

The aviation industry has undergone rapid growth and transformation in recent years, resulting in increased accessibility and affordability of air travel for passengers worldwide. However, one of the most significant challenges faced by travelers is the unpredictable nature of flight ticket prices. Airlines implement dynamic pricing strategies based on various factors such as demand, seasonality, route popularity, available seat capacity, time of booking, and even competitor pricing strategies. This constant fluctuation in airfare makes it difficult for travelers to determine the best time to book their tickets at the most economical rates.To address this issue, this thesis presents a predictive model for flight price estimation using machine learning techniques. By analyzing historical flight data and identifying key influencing parameters, our system aims to provide users with accurate price forecasts before booking their flights. The proposed model considers multiple factors, such as airline carriers, departure and arrival times, duration of flights, class of service, layovers, and airport locations, to generate a reliable prediction of future airfare trends. This research not only assists travelers in making cost-effective booking decisions but also enhances transparency in airline pricing structures.The system is developed using Python, a powerful programming language widely used in data science and machine learning applications. The predictive model is built using various machine learning algorithms, including regression techniques, decision trees, and ensemble learning methods, to achieve high accuracy in price prediction. Additionally, the system is deployed as an interactive web application using Streamlit, providing users with an intuitive and user-friendly interface to input their flight details and receive price predictions instantly. Furthermore, a database is integrated into the system to store user queries, historical pricing trends, and prediction results for future analysis and model improvement.This thesis thoroughly explores the complete development lifecycle of the system, covering requirement gathering, system analysis, design, implementation, testing, and future enhancements. The research aims to bridge the gap between unpredictable airfare fluctuations and traveler expectations by leveraging artificial intelligence and big data analytics. With the integration of advanced data processing techniques, our proposed system aspires to revolutionize the understanding of flight ticket pricing, enabling users to make more informed and strategic booking decisions.

**Objective**

These detailed objectives ensure that the **Flight Price Prediction and Booking System** is not only accurate and efficient but also user-friendly, secure, and scalable. By addressing these objectives, the system aims to revolutionize the way users predict and book flight tickets, providing a seamless and enjoyable experience.

make it formal objective

**Formal Objectives of the Flight Price Prediction and Booking System**

The **Flight Price Prediction and Booking System** is designed to address the limitations of traditional flight booking platforms by leveraging advanced machine learning techniques, modern web technologies, and robust database management. Below are the formal objectives of the system:

**1. Accurate Flight Price Prediction**

* **Objective**: To develop a highly accurate machine learning model capable of predicting flight prices based on dynamic factors such as departure time, arrival time, airline, number of stops, and cabin class.
* **Details**:
  + Utilize a **Random Forest Regressor** or other advanced machine learning algorithms to ensure precise price predictions.
  + Train the model on a comprehensive dataset that includes historical flight prices, seasonal trends, and demand fluctuations.
  + Continuously update the model with new data to maintain and improve prediction accuracy over time.

**2. User-Friendly Interface**

* **Objective**: To design an intuitive and interactive web application that allows users to input travel details and receive real-time flight price predictions.
* **Details**:
  + Develop a responsive and visually appealing user interface using **Streamlit**.
  + Incorporate interactive elements such as dropdown menus, date and time pickers, and sliders for user inputs.
  + Provide real-time feedback and display the predicted price prominently along with a breakdown of influencing factors.

**3. Efficient Data Management**

* **Objective**: To implement a robust database system for storing and managing flight prediction data.
* **Details**:
  + Use **SQLite** for efficient data storage and retrieval.
  + Design a structured database schema with tables to store user inputs, predicted prices, and historical data.
  + Enable efficient querying and retrieval of historical data for analysis and reporting.

**4. Customization for User Preferences**

* **Objective**: To allow users to customize predictions based on their specific preferences.
* **Details**:
  + Provide options for users to input the number of passengers, preferred cabin class, and airline.
  + Adjust the predicted price dynamically based on user inputs, such as multiplying the base price by the number of passengers or applying multipliers for different cabin classes.
  + Enable users to compare prices for different combinations of preferences.

**5. Historical Data Analysis**

* **Objective**: To provide users with insights into historical flight price trends.
* **Details**:
  + Store all predictions and user inputs in the database for future analysis.
  + Enable users to view historical data in tabular or graphical formats.
  + Provide insights such as average prices for specific routes, price trends over time, and the impact of factors like cabin class and airline on prices.

**6. Scalability and Performance**

* **Objective**: To ensure the system is scalable and can handle increasing user traffic.
* **Details**:
  + Design the system to handle a large number of concurrent users.
  + Use efficient algorithms and data structures to ensure fast predictions.
  + Optimize the database for quick querying and data retrieval.
  + Deploy the system on a cloud platform for scalability and reliability.

**7. System Security**

* **Objective**: To implement security measures to protect user data and ensure system integrity.
* **Details**:
  + Encrypt sensitive data before storing it in the database.
  + Validate all user inputs to prevent SQL injection and other attacks.
  + Implement optional user authentication for administrators to access historical data.
  + Regularly update the system to patch vulnerabilities and ensure compliance with security standards.

**8. Seamless Integration with External Systems**

* **Objective**: To enable integration with external systems for enhanced functionality.
* **Details**:
  + Integrate with real-time flight data APIs to fetch live prices and availability.
  + Provide options for users to book flights directly through the system (future enhancement).
  + Enable integration with payment gateways for seamless transactions (future enhancement).

**9. User Feedback and Continuous Improvement**

* **Objective**: To collect user feedback and continuously improve the system.
* **Details**:
  + Provide a feedback mechanism for users to report issues or suggest improvements.
  + Regularly update the machine learning model with new data to improve accuracy.
  + Continuously enhance the user interface based on user feedback and usability testing.

**10. Documentation and Support**

* **Objective**: To provide comprehensive documentation and support for users and administrators.
* **Details**:
  + Create user guides and tutorials to help users navigate the system.
  + Provide technical documentation for administrators to manage the system.
  + Offer customer support through email, chat, or a dedicated helpdesk (future enhancement).

**Preliminary System Analysis**

**a. Present System in Use**

The current system for flight price prediction and booking primarily relies on static pricing algorithms and third-party APIs. These systems are integrated into existing flight booking platforms and provide users with price estimates based on predefined rules and historical data. However, they lack the ability to adapt to real-time market changes and dynamic factors such as demand surges, fuel price fluctuations, and seasonal variations.

**b. Flaws in Present System**

1. **Inaccurate Predictions:**
   * **Static algorithms cannot account for dynamic factors like demand surges, fuel price fluctuations, or seasonal variations.**
   * **Prices are often outdated or do not reflect real-time market conditions.**
2. **Lack of Customization:**
   * Users cannot adjust predictions based on specific preferences such as cabin class, passenger count, or preferred airlines.
   * Limited flexibility in tailoring search results to individual needs.
3. **Limited Historical Data:**
   * Most systems do not provide access to historical price trends, making it difficult for users to make informed decisions.
   * Users cannot analyze price patterns over time.
4. **Poor User Experience:**
   * Many platforms have cluttered interfaces and lack real-time interactivity.
   * Users often find it difficult to navigate and input their preferences.

**c. Need for a New System**

A new system is required to address these flaws by:

* Utilizing Machine Learning: Implementing advanced algorithms like Random Forest Regressor to provide accurate and dynamic price predictions.
* Offering Customization: Allowing users to input specific preferences such as cabin class, passenger count, and preferred airlines.
* Storing Historical Data: Providing access to historical price trends for analysis and decision-making.
* Enhancing User Experience: Designing an intuitive and interactive web interface using modern frameworks like Streamlit.

**d. Feasibility Study**

1. **Technical Feasibility:**
   * The system uses Python, Streamlit, and SQLite, which are well-suited for web applications and data management.
   * The machine learning model is trained using Scikit-learn, a robust library for predictive analytics.
   * The integration of these technologies ensures a scalable and efficientsolution.
2. **Economic Feasibility:**
   * The use of open-source tools and libraries minimizes development costs.
   * The system can be hosted on cloud platforms with minimal operational expenses.
3. **Operational Feasibility:**
   * The system is designed to be user-friendly and requires minimal training for end-users.
   * It can be easily integrated into existing flight booking platforms.

**e. Project Category**

This project falls under the category of Web Application Development with Machine Learning Integration. It combines:

* Data Science: For accurate flight price prediction using machine learning.
* Web Development: For creating an interactive and user-friendly interface.
* Database Management: For storing and managing flight prediction data.

**Software Engineering Paradigm Applied**

The **Flight Price Prediction and Booking System** follows a structured software engineering approach to ensure efficient development, scalability, and maintainability. Below is a **detailed explanation** of the paradigm applied:

**a. Modules**

The system is divided into **four main modules** to ensure modularity, separation of concerns, and ease of maintenance. Each module has a specific role and interacts with other modules to deliver the system's functionality.

**User Interface Module**:

**Purpose**: This module is responsible for handling user interaction and displaying results in a user-friendly manner.

**Functionality**:

* Collects user inputs such as:
* Source and destination cities.
* Departure and arrival dates/times.
* Number of stops.
* Airline preferences.
* Passenger count and cabin class.
* Displays the predicted flight price and historical data in an interactive and visually appealing format.

**Technologies Used**:

* **Streamlit**: A Python-based framework for building web applications. It is used to create the user interface with dropdown menus, date/time pickers, and sliders.
* **HTML/CSS**: For custom styling and animations to enhance the user experience.

**Expected Outcome**:

* A seamless and engaging user interface that allows users to input their preferences and view results in real-time.

**Prediction Module**:

* **Purpose**: This module is responsible for predicting flight prices based on user inputs.

**Functionality**:

* Processes user inputs and applies the **Random Forest Regressor** machine learning model to generate predictions.
* Adjusts the predicted price dynamically based on:
* Passenger count (e.g., multiplying the base price by the number of passengers).
* Cabin class (e.g., applying multipliers for Business or First Class).
* Encodes categorical variables (e.g., airlines, source, destination) into numerical values for model input.

**Technologies Used**:

* **Scikit-learn**: A Python library for machine learning. It is used to train and deploy the Random Forest Regressor model.
* **Pandas**: For data manipulation and preprocessing.

**Expected Outcome**:

* Accurate flight price predictions that adapt to user preferences and dynamic factors.

**Database Module**:

**Purpose**: This module manages the storage and retrieval of flight prediction data.

**Functionality**:

* Stores user inputs, predicted prices, and historical data in a structured format.
* Retrieves historical data for analysis and reporting.
* Ensures efficient querying and data retrieval for real-time operations.

**Technologies Used**:

* **SQLite**: A lightweight and efficient relational database management system. It is used to store and manage flight prediction data.
* **SQLAlchemy**: For interacting with the database using Python.

**Expected Outcome**:

* A robust and scalable database system that supports efficient data storage and retrieval.

**Admin Module**:

* **Purpose**: This module allows administrators to manage and analyze historical data.

**Functionality**:

* Provides access to historical data for analysis and reporting.
* Enables administrators to view and manage user predictions.
* Generates insights such as average prices, price trends, and the impact of factors like cabin class and airline on prices.

**Technologies Used**:

* **Streamlit**: For creating an admin dashboard.
* **SQLite**: For querying historical data.
* **Expected Outcome**:
* A comprehensive admin interface that provides valuable insights into flight pricing trends.

**b. System/Modular Chart**

* The system is designed with a **modular architecture** to ensure clear separation of responsibilities and ease of maintenance.

**User Interface Module**:

* Interacts with the user and sends inputs to the Prediction Module.
* Displays the predicted price and historical data to the user.

**Prediction Module**:

* Processes inputs, generates predictions, and sends results to the Database Module for storage.

**Database Module**:

* Stores and retrieves data for future use and analysis.

**Software & Hardware Requirement Specification**

This section outlines the essential software and hardware requirements for developing and deploying the **Flight Price Prediction Model**. The system requires a robust computing environment capable of handling large datasets, machine learning computations, and web-based deployment.

**5.1 Software Requirements**

| **Component** | **Specification** |
| --- | --- |
| **Operating System** | Windows 11 |
| **Programming Language** | Python 13.3 |
| **Libraries & Frameworks** | NumPy, Pandas, Matplotlib, Seaborn, Scikit-Learn, TensorFlow, Streamlit |
| **Database** | CSV files |
| **IDE / Code Editor** | Jupyter Notebook, VS Code |
| **Version Control** | Git & GitHub |
| **Deployment Platform** | Streamlit, local server |
| **Web Framework** | Streamlit (for interactive UI) |

**Software Justification:**

* **Python 13.3** is used for its powerful capabilities in data science, machine learning, and system integration.
* **Scikit-Learn** and **TensorFlow** facilitate efficient model development.
* **Streamlit** provides an interactive interface for real-time user interaction.
* **CSV files** are used for lightweight data storage and retrieval.

**5.2 Hardware Requirements**

| **Component** | **Minimum Requirement** | **Recommended Requirement** |
| --- | --- | --- |
| **Processor** | Ryzen 5 7000 Series | Ryzen 5 7000 Series |
| **RAM** | 8 GB | 16 GB |
| **Storage** | 256 GB SSD | 512 GB SSD |
| **GPU (for deep learning)** | Not required | NVIDIA RTX 3060 or higher |
| **Internet Connection** | Required for data fetching & model training | High-speed connection recommended |

**Hardware Justification:**

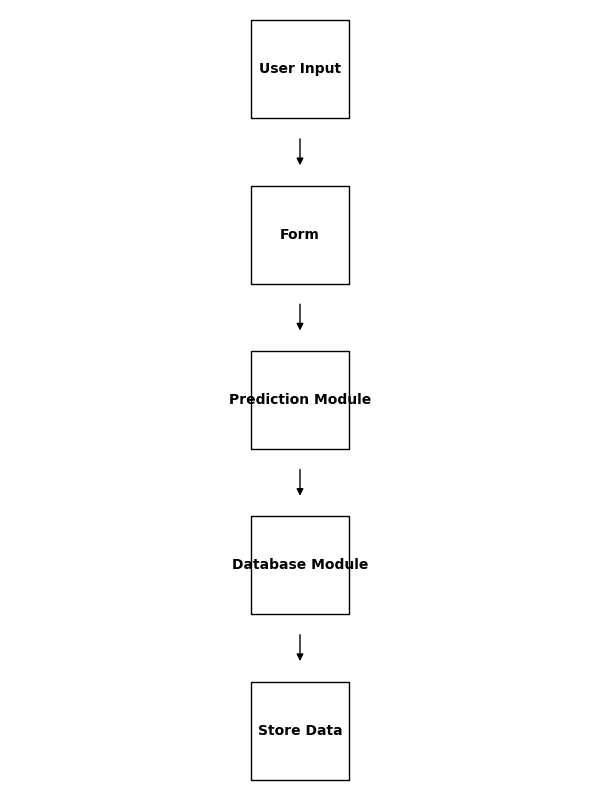
* **Ryzen 5 7000 Series** ensures efficient performance during data processing and model training.
* **16 GB RAM** is ideal for handling large datasets and machine learning computations.
* **512 GB SSD** ensures faster data access and improves system performance.
* **NVIDIA RTX 3060 or higher** is recommended if deep learning models are implemented for enhanced computational efficiency.

**Detailed System Analysis**

The **Detailed System Analysis** provides a comprehensive understanding of the system's architecture, data flow, and components. It breaks down the system into smaller, manageable parts to ensure clarity and efficiency in design and implementation. Below is a **detailed explanation** of each aspect of the system analysis:

**a. Data Flow Diagram (DFD)**

The **Data Flow Diagram (DFD)** is a graphical representation of how data moves through the system. It shows the flow of data between the user, the system modules, and the database.

**Level 0 DFD (Context Diagram)**:  


* **User Input**: The user provides inputs such as source, destination, departure/arrival times, and preferences.
* **Prediction Module**: Processes the inputs and generates a flight price prediction.
* **Database Module**: Stores the prediction and user inputs for future analysis.

**Level 1 DFD (Detailed Flow)**:

1. **User Input**:
   * The user interacts with the **User Interface Module** to input travel details.
   * Inputs include:
     + Source and destination cities.
     + Departure and arrival dates/times.
     + Number of stops.
     + Airline preferences.
     + Passenger count and cabin class.
2. **Prediction Module**:
   * The inputs are sent to the **Prediction Module**.
   * The module processes the inputs and applies the **Random Forest Regressor** model to generate a price prediction.
   * The predicted price is adjusted dynamically based on user preferences (e.g., passenger count, cabin class).
3. **Database Module**:
   * The prediction and user inputs are sent to the **Database Module** for storage.
   * The data is stored in the predictions table for future analysis.

**b. Number of Modules and Process Logic**

The system is divided into **three main modules**, each with specific functionality and process logic:

1. **User Interface Module**:
   * **Functionality**:
     + Collects user inputs and displays results.
     + Ensures a seamless and interactive user experience.
   * **Process Logic**:
     + The user interacts with the interface to input travel details.
     + The inputs are validated to ensure they are correct (e.g., arrival time must be after departure time).
     + The validated inputs are sent to the **Prediction Module**.
2. **Prediction Module**:
   * **Functionality**:
     + Processes inputs and applies the machine learning model.
     + Adjusts the price dynamically based on user preferences.
   * **Process Logic**:
     + The inputs are preprocessed (e.g., encoding categorical variables like airlines and cities).
     + The preprocessed data is fed into the **Random Forest Regressor** model.
     + The model generates a base price prediction.
     + The price is adjusted based on:
       - Passenger count (e.g., multiplying the base price by the number of passengers).
       - Cabin class (e.g., applying multipliers for Business or First Class).
     + The final price is sent to the **Database Module** for storage.
3. **Database Module**:
   * **Functionality**:
     + Stores and retrieves data for analysis and reporting.
   * **Process Logic**:
     + The prediction and user inputs are stored in the predictions table.
     + Historical data is retrieved for analysis and displayed to the user.

**c. Data Structures and Tables**

The system uses **structured data storage** to manage user inputs, predictions, and historical data. Below is a detailed explanation of the data structures and tables:

1. **Tables**:
   * predictions: Stores flight prediction data.
     + **Columns**:
       - id: Unique identifier for each prediction (Primary Key).
       - total\_stops: Number of stops in the flight.
       - journey\_day: Day of the journey.
       - journey\_month: Month of the journey.
       - dep\_hour: Departure hour.
       - dep\_min: Departure minute.
       - arrival\_hour: Arrival hour.
       - arrival\_min: Arrival minute.
       - dur\_hour: Duration in hours.
       - dur\_min: Duration in minutes.
       - airline: Airline name.
       - source: Source city.
       - destination: Destination city.
       - passenger\_count: Number of passengers.
       - cabin\_class: Cabin class (Economy, Business, First Class).
       - price: Predicted flight price.
2. **Data Structures**:
   * **Lists**: Used to store and process user inputs (e.g., list of airlines, cities).
   * **Dictionaries**: Used to map categorical variables to numerical values (e.g., airline names to encoded values).
   * **DataFrames**: Used for data manipulation and preprocessing (e.g., Pandas DataFrame).

**d. Entity Relationship Diagram (ERD)**

The **Entity Relationship Diagram (ERD)** illustrates the relationships between entities in the database. Below is the ERD for the system:

**Entities**:

1. **Predictions**:
   * Stores flight prediction data.
   * Attributes: id, total\_stops, journey\_day, journey\_month, dep\_hour, dep\_min, arrival\_hour, arrival\_min, dur\_hour, dur\_min, airline, source, destination, passenger\_count, cabin\_class, price.
2. **Users** (Optional for Future Enhancements):
   * Stores user information.
   * Attributes: id, name.

**Relationships**:

* Each **Prediction** is associated with a **User** (optional for future enhancements).
* The relationship is **one-to-many**: One user can have multiple predictions.

**Summary of Detailed System Analysis**

| **Aspect** | **Details** |
| --- | --- |
| **Data Flow Diagram** | Illustrates how data moves between the user, Prediction Module, and Database. |
| **Modules** | User Interface, Prediction, and Database Modules. |
| **Process Logic** | Input validation, price prediction, and data storage. |
| **Data Structures** | Lists, dictionaries, and DataFrames for data manipulation. |
| **Tables** | predictions table to store flight prediction data. |
| **Entity Relationship Diagram** | Shows relationships between Predictions and Users entities. |

This detailed system analysis ensures that the **Flight Price Prediction and Booking System** is designed with clarity, efficiency, and scalability in mind. Each component is carefully analyzed to ensure seamless integration and optimal performance.

**System Design**

a. Form Design

**1. Flight Details**

* **Departure City:** [ Select City ▼ ] *(Dropdown with predefined major cities)*
* **Destination City:** [ Select City ▼ ] *(Dropdown with predefined major cities)*
* **Departure Date:** [📅 Select Date] *(Date Picker)*
* **Return Date (Optional):** [📅 Select Date] *(Date Picker)*
* **Preferred Airline:** [ Select Airline ▼ ] *(Dropdown: Airline options with "Any" default)*
* **Number of Passengers:** [ 1 ▼ ] *(Dropdown: 1-10 passengers)*
* **Cabin Class:** ( ) Economy ( ) Business ( ) First-Class *(Radio Buttons)*

**2. Flight Preferences**

* **Layover Preference:**  
  ( ) Direct Flight  
  ( ) 1 Stop  
  ( ) 2+ Stops *(Radio Buttons)*
* **Flexible Dates?** [✓] Yes [ ] No *(Checkbox)*
* **Travel Duration Preference:** [\_\_\_\_\_\_] Hours (Optional) *(Numeric Input)*

**3. Price Prediction & Booking**

* **Predict Flight Price:** [🔍 Get Prediction] *(Button - Uses ML Model to Predict Price)*
* **Compare Prices with Airlines:** [🔍 Search for Best Deals] *(Button - Fetches Real-time Prices from Airlines)*
* **Save Query for Future Alerts:** [✓] Subscribe for Price Alerts *(Checkbox with Email Notifications)*

b. Source Code

1. aeroplain.ipynb

1. # IMPORTING PACKAGES

2.

3. import numpy as np

4. import pandas as pd

5. import matplotlib.pyplot as plt

6. import seaborn as sns

7.

8. sns.set()

9. # LOADING TRAINNING DATA

10. train\_data=pd.read\_excel(r"C:\Users\mohit\streamlit dev\datasets\Data\_Train.xlsx")

11. train\_data

12. # DATA CLEANING

13. pd.set\_option('display.max\_columns',None)

14. train\_data.head()

15.

16. train\_data.info()

17. #Give Distinct Count

18. train\_data['Duration'].value\_counts()

19. train\_data.dropna(inplace=True)

20. train\_data

21.

22. train\_data.isnull().sum()

23. # EDA

24. train\_data.head(1)

25. # Date Formating

26. train\_data['Journey\_Day']=pd.to\_datetime(train\_data.Date\_of\_Journey,format="%d/%m/%Y").dt.day

27.

train\_data['Journey\_Month']=pd.to\_datetime(train\_data.Date\_of\_Journey,format="%d/%m/%Y").dt.month

28. train\_data.head(1)

29. # as this data has only one year of data so we are droping the date of journey column

30. train\_data.drop(['Date\_of\_Journey'],axis=1,inplace=True)

31. #departure time is when a plane leaves the gate .

32. # similar to dat\_of\_journey we can extract value from the dep\_time

33.

34. # extracting hours

35. train\_data['dep\_hour']=pd.to\_datetime(train\_data["Dep\_Time"]).dt.hour

36.

37. #extracting minutes

38. train\_data['dep\_min']=pd.to\_datetime(train\_data["Dep\_Time"]).dt.minute

39.

40. #now we can drop dep\_time coz its of no use

41. train\_data.drop(['Dep\_Time'],axis=1,inplace=True)

42.

43.

44. train\_data.head(1)

45. #arrival time is when a plane pull up the gate .

46. # similar to dat\_of\_journey we can extract value from the dep\_time

47.

48. # extracting hours

49. train\_data['Arrival\_hour']=pd.to\_datetime(train\_data["Arrival\_Time"]).dt.hour

50.

51. #extracting minutes

52. train\_data['Arrival\_min']=pd.to\_datetime(train\_data["Arrival\_Time"]).dt.minute

53.

54. #now we can drop dep\_time coz its of no use

55. train\_data.drop(['Arrival\_Time'],axis=1,inplace=True)

56.

57.

58. len('2h 50m '.split())

59. # Time taken by plane to reach destination is called Duration

60. # It is the differnce betwwen Departure Time and Arrival time

61.

62.

63. # Assigning and converting Duration column into list

64. duration = list(train\_data["Duration"])

65.

66. for i in range(len(duration)):

67. if len(duration[i].split()) != 2: # Check if duration contains only hour or mins

68. if "h" in duration[i]:

69. duration[i] = duration[i].strip() + " 0m" # Adds 0 minute

70. else:

71. duration[i] = "0h " + duration[i] # Adds 0 hour

72.

73. duration\_hours = []

74. duration\_mins = []

75. for i in range(len(duration)):

76. duration\_hours.append(int(duration[i].split(sep = "h")[0])) # Extract hours from duration

77. duration\_mins.append(int(duration[i].split(sep = "m")[0].split()[-1])) # Extracts o

78. # Adding duration\_hours and duration\_mins list to train\_data dataframe

79.

80. train\_data["Duration\_hours"] = duration\_hours

81. train\_data["Duration\_mins"] = duration\_mins

82.

83. # droping duration

84.

85. train\_data.drop(["Duration"], axis = 1, inplace = True)

86. train\_data.head(3)

87. # handeling categorical data

88. One can find many ways to handle categorical data. Some of them categorical data are,

89.

90. Nominal data --> data are not in any order --> OneHotEncoder is used in this case

91. Ordinal data --> data are in order --> LabelEncoder is used in this case

92. train\_data["Airline"].value\_counts()

93. # From graph we can see that Jet Airways Business have the highest Price.

94. # Apart from the first Airline almost all are having similar median

95. # we can use box plot on the place of catplot

96. # Airline vs Price

97. sns.catplot(y = "Price", x = "Airline", data = train\_data.sort\_values("Price", ascending = False), kind="boxen", height = 4, aspect = 3,palette="coolwarm")

98. plt.show()

99. # As Airline is Nominal Categorical data we will perform OneHotEncoding

100. #pd.get\_dummies():Converts each unique category in the Airline column into a separate binary column (1 or 0).

101. Airline = train\_data[["Airline"]]

102.

103. Airline = pd.get\_dummies(Airline, drop\_first= True)

104. Airline = Airline.astype(int) # Convert True/False to 1/0

105. Airline.head()

106. # Airline vs Price

107. sns.catplot(y = "Price", x = "Source", data = train\_data.sort\_values("Price", ascending = False), kind="boxen", height = 4, aspect = 3,palette="coolwarm")

108. plt.show()

109. # As Source is Nominal Categorical data we will perform OneHotEncoding

110.

111. Source = train\_data[["Source"]]

112.

113. Source = pd.get\_dummies(Source, drop\_first= True)

114. Source = Source.astype(int)

115. Source.head()

116.

117. train\_data['Destination'].value\_counts()

118. sns.catplot(y = "Price", x = "Destination", data = train\_data.sort\_values("Price", ascending = False), kind="boxen", height = 4, aspect = 3,palette="coolwarm")

119. plt.show()

120. # As Destination is Nominal Categorical data we will perform OneHotEncoding

121.

122. Destination = train\_data[["Destination"]]

123.

124. Destination = pd.get\_dummies(Destination, drop\_first = True)

125. Destination=Destination.astype(int)

126. Destination.head()

127. train\_data["Route"]

128.

129. # Additional\_Info contains almost 80% no\_info

130. # Route and Total\_Stops are related to each other

131.

132. train\_data.drop(["Route", "Additional\_Info"], axis = 1, inplace = True)

133. train\_data["Total\_Stops"].value\_counts()

134. # As this is case of Ordinal Categorical type we perform LabelEncoder

135. # Here Values are assigned with corresponding keys

136.

137. train\_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)

138. train\_data.head()

139. # Concatenate dataframe --> train\_data + Airline + Source + Destination

140.

141. data\_train = pd.concat([train\_data, Airline, Source, Destination], axis = 1)

142. data\_train.head()

143. data\_train.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)

144. data\_train.head()

145. data\_train.shape

146. # TEST DATASET

147. we process the test data seprately coz of dataleakage

148.

149. #LOADING TEST DATASET

150.

151. test\_data = pd.read\_excel(r"C:\Users\mohit\FLIGHT PRICE PREDICTION\datasets\Test\_set.xlsx")

152. test\_data.head(2)

153.

154. # Preprocessing

155.

156. print("Test data Info")

157. print("-"\*75)

158. print(test\_data.info())

159.

160. print()

161. print()

162.

163. print("Null values :")

164. print("-"\*75)

165. test\_data.dropna(inplace = True)

166. print(test\_data.isnull().sum())

167.

168. # EDA

169.

170. # Date\_of\_Journey

171. test\_data["Journey\_day"] = pd.to\_datetime(test\_data.Date\_of\_Journey, format="%d/%m/%Y").dt.day

172. test\_data["Journey\_month"] = pd.to\_datetime(test\_data["Date\_of\_Journey"], format = "%d/%m/%Y").dt.month

173. test\_data.drop(["Date\_of\_Journey"], axis = 1, inplace = True)

174.

175. # Dep\_Time

176. test\_data["Dep\_hour"] = pd.to\_datetime(test\_data["Dep\_Time"]).dt.hour

177. test\_data["Dep\_min"] = pd.to\_datetime(test\_data["Dep\_Time"]).dt.minute

178. test\_data.drop(["Dep\_Time"], axis = 1, inplace = True)

179.

180. # Arrival\_Time

181. test\_data["Arrival\_hour"] = pd.to\_datetime(test\_data.Arrival\_Time).dt.hour

182. test\_data["Arrival\_min"] = pd.to\_datetime(test\_data.Arrival\_Time).dt.minute

183. test\_data.drop(["Arrival\_Time"], axis = 1, inplace = True)

184.

185. # Duration

186. duration = list(test\_data["Duration"])

187.

188. for i in range(len(duration)):

189. if len(duration[i].split()) != 2: # Check if duration contains only hour or mins

190. if "h" in duration[i]:

191. duration[i] = duration[i].strip() + " 0m" # Adds 0 minute

192. else:

193. duration[i] = "0h " + duration[i] # Adds 0 hour

194.

195. duration\_hours = []

196. duration\_mins = []

197. for i in range(len(duration)):

198. duration\_hours.append(int(duration[i].split(sep = "h")[0])) # Extract hours from duration

199. duration\_mins.append(int(duration[i].split(sep = "m")[0].split()[-1])) # Extracts only minutes from duration

200.

201. # Adding Duration column to test set

202. test\_data["Duration\_hours"] = duration\_hours

203. test\_data["Duration\_mins"] = duration\_mins

204. test\_data.drop(["Duration"], axis = 1, inplace = True)

205.

206.

207. # Categorical data

208.

209. print("Airline")

210. print("-"\*75)

211. print(test\_data["Airline"].value\_counts())

212. Airline = pd.get\_dummies(test\_data["Airline"], drop\_first= True)

213. Airline=Airline.astype(int)

214. print()

215.

216. print("Source")

. print("-"\*75)

218. print(test\_data["Source"].value\_counts())

219. Source = pd.get\_dummies(test\_data["Source"], drop\_first= True)

220. Source=Source.astype(int)

221.

222. print()

223.

224. print("Destination")

225. print("-"\*75)

226. print(test\_data["Destination"].value\_counts())

227. Destination = pd.get\_dummies(test\_data["Destination"], drop\_first = True)

228. Destination=Destination.astype(int)

229.

230. # Additional\_Info contains almost 80% no\_info

231. # Route and Total\_Stops are related to each other

232. test\_data.drop(["Route", "Additional\_Info"], axis = 1, inplace = True)

233.

234. # Replacing Total\_Stops

235. test\_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)

236.

237. # Concatenate dataframe --> test\_data + Airline + Source + Destination

238. data\_test = pd.concat([test\_data, Airline, Source, Destination], axis = 1)

239.

240. data\_test.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)

241.

242. print()

243. print()

244.

245. print("Shape of test data : ", data\_test.shape)

246.

247.

248. data\_test.head()

249. # Feature Selection

250. Finding out the best feature which will contribute and have good relation with target variable. Following are some of the feature selection methods,

251.

252. 1 heatmap

253. 2 feature\_importance\_

254. 3 SelectKBest

255. data\_train.shape

256. data\_train.columns

257. X = data\_train.loc[:, ['Total\_Stops', 'Journey\_Day', 'Journey\_Month', 'dep\_hour',

258. 'dep\_min', 'Arrival\_hour', 'Arrival\_min', 'Duration\_hours',

259. 'Duration\_mins', 'Airline\_Air India', 'Airline\_GoAir', 'Airline\_IndiGo',

260. 'Airline\_Jet Airways', 'Airline\_Jet Airways Business',

261. 'Airline\_Multiple carriers',

262. 'Airline\_Multiple carriers Premium economy', 'Airline\_SpiceJet',

263. 'Airline\_Trujet', 'Airline\_Vistara', 'Airline\_Vistara Premium economy',

264. 'Source\_Chennai', 'Source\_Delhi', 'Source\_Kolkata', 'Source\_Mumbai',

265. 'Destination\_Cochin', 'Destination\_Delhi', 'Destination\_Hyderabad',

266. 'Destination\_Kolkata', 'Destination\_New Delhi']]

267. X.head()

268.

269. y = data\_train.iloc[:, 1]

270. y.head()

271. # Finds correlation between Independent and dependent attributes

272.

273. # Select only numerical columns

274. numerical\_data = train\_data.select\_dtypes(include=['number'])

275.

276. # Plot the heatmap for correlation matrix

277. plt.figure(figsize=(18, 18))

278. sns.heatmap(numerical\_data.corr(), annot=True, cmap="RdYlGn")

279. plt.show()

280.

281. # Important feature using ExtraTreesRegressor

282.

283. from sklearn.ensemble import ExtraTreesRegressor

284. selection = ExtraTreesRegressor()

285. selection.fit(X, y)

286. print(selection.feature\_importances\_)

287. #plot graph of feature importances for better visualization

288.

289. plt.figure(figsize = (12,8))

290. feat\_importances = pd.Series(selection.feature\_importances\_, index=X.columns)

291. feat\_importances.nlargest(20).plot(kind='barh')

292. plt.show()

293. # Fitting model using Random Forest

294. Split dataset into train and test set in order to prediction w.r.t X\_test

295. If needed do scaling of data

296. Scaling is not done in Random forest

297. Import model

298. Fit the data

299. Predict w.r.t X\_test

300. In regression check RSME Score

301. Plot graph

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

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

304.

305.

306. from sklearn.ensemble import RandomForestRegressor

307. reg\_rf = RandomForestRegressor()

308. reg\_rf.fit(X\_train, y\_train)

309. y\_pred = reg\_rf.predict(X\_test)

310. reg\_rf.score(X\_train, y\_train)

311. reg\_rf.score(X\_test, y\_test)

312.

313. sns.distplot(y\_test-y\_pred)

314. plt.show()

315. plt.scatter(y\_test, y\_pred, alpha = 0.5)

316. plt.xlabel("y\_test")

317. plt.ylabel("y\_pred")

318. plt.show()

319. from sklearn import metrics

320. print('MAE:', metrics.mean\_absolute\_error(y\_test, y\_pred))

321. print('MSE:', metrics.mean\_squared\_error(y\_test, y\_pred))

322. print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))

323.

324. # RMSE/(max(DV)-min(DV))

325.

326. 2090.5509/(max(y)-min(y))

327. metrics.r2\_score(y\_test, y\_pred)

328. # Hyperparameter Tuning

329. Choose following method for hyperparameter tuning

330. RandomizedSearchCV --> Fast

331. GridSearchCV

332. Assign hyperparameters in form of dictionery

333. Fit the model

334. Check best paramters and best score

335. from sklearn.model\_selection import RandomizedSearchCV

336. #Randomized Search CV

337.

338. # Number of trees in random forest

339. n\_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]

340. # Number of features to consider at every split

341. max\_features = ['auto', 'sqrt']

342. # Maximum number of levels in tree

343. max\_depth = [int(x) for x in np.linspace(5, 30, num = 6)]

344. # Minimum number of samples required to split a node

345. min\_samples\_split = [2, 5, 10, 15, 100]

346. # Minimum number of samples required at each leaf node

347. min\_samples\_leaf = [1, 2, 5, 10]

348. # Create the random grid

349.

350. random\_grid = {'n\_estimators': n\_estimators,

351. 'max\_features': max\_features,

352. 'max\_depth': max\_depth,

353. 'min\_samples\_split': min\_samples\_split,

354. 'min\_samples\_leaf': min\_samples\_leaf}

355. # Random search of parameters, using 5 fold cross validation,

356. # search across 100 different combinations

357. rf\_random = RandomizedSearchCV(estimator = reg\_rf, param\_distributions = random\_grid,scoring='neg\_mean\_squared\_error', n\_iter = 10, cv = 5, verbose=2, random\_state=42, n\_jobs = 1)

358.

359.

360. rf\_random.fit(X\_train,y\_train)

361. rf\_random.best\_params\_

362. prediction = rf\_random.predict(X\_test)

363. plt.figure(figsize = (8,8))

364. sns.distplot(y\_test-prediction)

365. plt.show()

366. plt.figure(figsize = (8,8))

367. plt.scatter(y\_test, prediction, alpha = 0.5)

368. plt.xlabel("y\_test")

369. plt.ylabel("y\_pred")

370. plt.show()

371. print('MAE:', metrics.mean\_absolute\_error(y\_test, prediction))

372. print('MSE:', metrics.mean\_squared\_error(y\_test, prediction))

373. print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, prediction)))

374. # Save the model to reuse it again

375. import pickle

376. # open a file, where you ant to store the data

377. file = open('flight\_rf.pkl', 'wb')

378.

379. # dump information to that file

380. pickle.dump(reg\_rf, file)

381. with open(r"C:\Users\mohit\streamlit dev\flight\_rf.pkl", "rb") as file:

382. forest = pickle.load(file)

383. y\_prediction = forest.predict(X\_test)

384. metrics.r2\_score(y\_test, y\_prediction)

Main.py

1

2.

3. import streamlit as st

4. import pickle, base64

5. import pandas as pd

6. import sqlite3

7. from PIL import Image

8.

9. # Database setup

10. def init\_db():

11.     conn = sqlite3.connect("flight\_predictions.db")

12.     c = conn.cursor()

13.     c.execute("""

14.         CREATE TABLE IF NOT EXISTS predictions (

15.             id INTEGER PRIMARY KEY AUTOINCREMENT,

16.             total\_stops INTEGER,

17.             journey\_day INTEGER,

18.             journey\_month INTEGER,

19.             dep\_hour INTEGER,

20.             dep\_min INTEGER,

21.             arrival\_hour INTEGER,

22.             arrival\_min INTEGER,

23.             dur\_hour INTEGER,

24.             dur\_min INTEGER,

25.             airline TEXT,

26.             source TEXT,

27.             destination TEXT,

28.             passenger\_count TEXT,

29.             cabin\_class TEXT,

30.             price REAL

31.         )

32.     """)

33.     conn.commit()

34.     conn.close()

35.

36. init\_db()

37.

38. # Load the trained model

39. model = pickle.load(open("flight\_rf.pkl", "rb")

40.

41. # Custom CSS for styling and animations

42. st.markdown("""

43.     <style>

44.         .main {

45.             background-color: #0e1117;

46.             color: #ffffff;

47.             font-family: 'Arial', sans-serif;

48.         }

49.         .header {

50.             text-align: center;

51.             font-size: 42px;

52.             font-weight: bold;

53.             color: #00ffcc;

54.             margin-bottom: 20px;

55.             text-shadow: 2px 2px 4px #000000;

56.             animation: fadeIn 2s ease-in-out;

57.         }

58.         .sub-header {

59.             text-align: center;

60.             font-size: 24px;

61.             color: #cccccc;

62.             margin-bottom: 40px;

63.             animation: slideIn 1.5s ease-in-out;

64.         }

65.         .stImage {

66.             display: block;

67.             margin: 0 auto;

68.             border-radius: 15px;

69.             box-shadow: 0 4px 8px rgba(0, 255, 204, 0.3);

70.             animation: zoomIn 1s ease-in-out;

71.         }

72.         .form-container {

73.             background-color: rgba(30, 30, 30, 0.9);

74.             padding: 20px;

75.             border-radius: 15px;

76.             box-shadow: 0 4px 8px rgba(0, 255, 204, 0.3);

77.             margin-bottom: 20px;

78.             animation: fadeIn 1.5s ease-in-out;

79.         }

80.         .stButton>button {

81.             background-color: #00ffcc;

82.             color: #000000;

83.             font-size: 18px;

84.             font-weight: bold;

85.             padding: 10px 24px;

86.             border: none;

87.             border-radius: 8px;

88.             cursor: pointer;

89.             transition: background-color 0.3s ease;

90.         }

91.         .stButton>button:hover {

92.             background-color: #00ccaa;

93.         }

94.         .stSelectbox>div>div>select {

95.             background-color: #333333;

96.             color: #ffffff;

97.             border: 1px solid #00ffcc;

98.             border-radius: 8px;

99.             padding: 8px;

100.         }

101.         .stDateInput>div>div>input {

102.             background-color: #333333;

103.             color: #ffffff;

104.             border: 1px solid #00ffcc;

105.             border-radius: 8px;

106.             padding: 8px;

107.         }

108.         .stTimeInput>div>div>input {

109.             background-color: #333333;

110.             color: #ffffff;

111.             border: 1px solid #00ffcc;

112.             border-radius: 8px;

113.             padding: 8px;

114.         }

115.         .stDataFrame {

116.             background-color: #1e1e1e;

117.             color: #ffffff;

118.             border-radius: 15px;

119.             box-shadow: 0 4px 8px rgba(0, 255, 204, 0.3);

120.         }

121.         .stSuccess {

122.             background-color: #00ffcc;

123.             color: #000000;

124.             border-radius: 8px;

125.             padding: 10px;

126.             font-size: 18px;

127.             font-weight: bold;

128.             text-align: center;

129.         }

130.         @keyframes fadeIn {

131.             from { opacity: 0; }

132.             to { opacity: 1; }

133.         }

134.         @keyframes slideIn {

135.             from { transform: translateX(-100%); }

136.             to { transform: translateX(0); }

137.         }

138.         @keyframes zoomIn {

139.             from { transform: scale(0.9); opacity: 0; }

140.             to { transform: scale(1); opacity: 1; }

141.         }

142.         .footer {

143.             text-align: center;

144.             padding: 20px;

145.             background-color: rgba(30, 30, 30, 0.9);

146.             border-radius: 15px;

147.             margin-top: 40px;

148.             animation: fadeIn 2s ease-in-out;

149.         }

150.         .footer a {

151.             color: #00ffcc;

152.             text-decoration: none;

153.             margin: 0 10px;

154.         }

155.         .footer a:hover {

156.             text-decoration: underline;

157.         }

158.         .aviation-section {

159.             background-color: rgba(30, 30, 30, 0.9);

160.             padding: 20px;

161.             border-radius: 15px;

162.             margin-bottom: 20px;

163.             box-shadow: 0 4px 8px rgba(0, 255, 204, 0.3);

164.         }

165.         .aviation-section img {

166.             border-radius: 15px;

167.             margin-bottom: 20px;

168.         }

169.         .aviation-section h2 {

170.             color: #00ffcc;

171.             font-size: 28px;

172.             margin-bottom: 10px;

173.         }

174.         .aviation-section p {

175.             color: #cccccc;

176.             font-size: 18px;

177.             line-height: 1.6;

178.         }

179.         .video-container {

180.             position: fixed;

181.             top: 0;

182.             left: 0;

183.             width: 100%;

184.             height: 100%;

185.             z-index: 0;

186.             overflow: hidden;

187.         }

188.         .video-container video {

189.             min-width: 100%;

190.             min-height: 100%;

191.             object-fit: cover;

192.         }

193.         .main-content {

194.             position: relative;

195.             z-index: 1;

196.             padding: 20px;

197.             background: rgba(255, 255, 255, 0.1); /\* Light overlay for readability \*/

198.             border-radius: 10px;

199.         }

200.     </style>

201.     """, unsafe\_allow\_html=True)

202.

203. # Function to generate HTML for embedding a background video

204. def get\_video\_html(video\_path):

205.     with open(video\_path, "rb") as video\_file:

206.         video\_base64 = base64.b64encode(video\_file.read()).decode()

207.

208.     return f"""

209.     <div>

210.     <div class="video-container">

211.         <video autoplay loop muted>

212.             <source src="data:video/mp4;base64,{video\_base64}" type="video/mp4">

213.         </video>

214.     </div>

215.     </div>

216.     """

217.

218. # Path to video

219. video\_path = r"C:\Users\mohit\FLIGHT PRICE PREDICTION\image\3612113-hd\_1920\_1080\_30fps.mp4"

220.

221. # Inject video background

222. st.markdown(get\_video\_html(video\_path), unsafe\_allow\_html=True)

223.

224. # UI Enhancements

225. st.image(r"C:\Users\mohit\FLIGHT PRICE PREDICTION\image\DALL·E 2025-03-10 20.10.46 - A professional airline logo with a modern and sleek design, featuring a blue and white color scheme with a flying airplane icon and the name 'SkyWings.webp", use\_container\_width=True)

226. st.markdown('<div class="header">✈️ Ready to Take Off?</div>', unsafe\_allow\_html=True)

227. st.markdown('<div class="sub-header">Experience the future of flight booking with SkyWings.</div>', unsafe\_allow\_html=True)

228.

229. # Aviation Trip Section

230. st.markdown('<div class="aviation-section">', unsafe\_allow\_html=True)

231. st.markdown('<h2>✈️ Soar Through the Skies with SkyWings</h2>', unsafe\_allow\_html=True)

232. st.image(r"C:\Users\mohit\FLIGHT PRICE PREDICTION\image\d.webp", use\_container\_width=True)

233. st.markdown("""

234.     <p>

235.         At SkyWings, we believe that every journey should be extraordinary. Whether you're flying for business or leisure,

236.         our state-of-the-art aircraft and world-class service ensure a seamless and luxurious experience.

237.         From the moment you step on board, you'll be immersed in comfort and elegance, soaring above the clouds

238.         with unparalleled views of the world below.

239.     </p>

240.     <p>

241.         Our commitment to excellence extends beyond the skies. With SkyWings, you're not just booking a flight –

242.         you're embarking on an adventure. Let us take you to new heights, where the journey is as memorable as the destination.

243.     </p>

244.     """, unsafe\_allow\_html=True)

245. st.markdown('</div>', unsafe\_allow\_html=True)

246.

247. # Flight Details Form

248. with st.form("flight\_details\_form"):

249.     st.markdown('<div class="form-container">', unsafe\_allow\_html=True)

250.     col1, col2 = st.columns(2)

251.     with col1:

252.         source = st.selectbox("From", ["NYC", "Delhi", "Kolkata", "Mumbai", "Chennai"])

253.     with col2:

254.         destination = st.selectbox("To", ["Dhaka", "Delhi", "New\_Delhi", "Hyderabad", "Kolkata"])

255.

256.     col3, col4 = st.columns(2)

257.     with col3:

258.         dep\_date = st.date\_input("Departure Date")

259.     with col4:

260.         dep\_time = st.time\_input("Departure Time")

261.

262.     col5, col6 = st.columns(2)

263.     with col5:

264.         arr\_date = st.date\_input("Arrival Date")

265.     with col6:

266.         arr\_time = st.time\_input("Arrival Time")

267.

268.     Total\_stops = st.selectbox("Total Stops", [0, 1, 2, 3, 4])

269.     airline = st.selectbox("Airline", [

270.         "Jet Airways", "IndiGo", "Air India", "Multiple carriers", "SpiceJet",

271.         "Vistara", "GoAir", "Multiple carriers Premium economy", "Jet Airways Business",

272.         "Vistara Premium economy", "Trujet"

273.     ])

274.

275.     passenger\_count = st.selectbox("Passenger", ["1 Adult", "2 Adults", "3 Adults", "4 Adults"])

276.     cabin\_class = st.selectbox("Cabin Class", ["Economy", "Business", "First Class"])

277.

278.     if st.form\_submit\_button("Show Flight"):

279.         st.markdown('</div>', unsafe\_allow\_html=True)

280.         dep\_datetime = pd.Timestamp.combine(dep\_date, dep\_time)

281.         arr\_datetime = pd.Timestamp.combine(arr\_date, arr\_time)

282.         if arr\_datetime <= dep\_datetime:

283.             st.error("Arrival time must be after departure time!")

284.             st.stop()

285.

286.         duration = arr\_datetime - dep\_datetime

287.         dur\_hour = duration.seconds // 3600

288.         dur\_min = (duration.seconds % 3600) // 60

289.

290.         airlines\_map = {name: [int(i == j) for i in range(11)] for j, name in enumerate([

291.             "Jet Airways", "IndiGo", "Air India", "Multiple carriers", "SpiceJet",

292.             "Vistara", "GoAir", "Multiple carriers Premium economy", "Jet Airways Business",

293.             "Vistara Premium economy", "Trujet"

294.         ])}

295.         airline\_encoding = airlines\_map.get(airline, [0] \* 11)

296.

297.         source\_map = {city: [int(i == j) for i in range(4)] for j, city in enumerate(["NYC", "Delhi", "Kolkata", "Mumbai", "Chennai"])}

298.         destination\_map = {city: [int(i == j) for i in range(5)] for j, city in enumerate(["Dhaka", "Delhi", "New\_Delhi", "Hyderabad", "Kolkata"])}

299.         source\_encoding = source\_map.get(source, [0] \* 4)

300.         destination\_encoding = destination\_map.get(destination, [0] \* 5)

301.

302.         features = [

303.             Total\_stops, dep\_date.day, dep\_date.month, dep\_time.hour, dep\_time.minute,

304.             arr\_time.hour, arr\_time.minute, dur\_hour, dur\_min,

305.             \*airline\_encoding, \*source\_encoding, \*destination\_encoding

306.         ]

307.         prediction = model.predict([features])

308.         base\_price = round(prediction[0], 2)

309.

310.         # Adjust price based on passenger count

311.         num\_adults = int(passenger\_count.split()[0])  # Extract number of adults

312.         final\_price = base\_price \* num\_adults

313.

314.         # Adjust price based on cabin class

315.         if cabin\_class == "Business":

316.             final\_price \*= 9

317.         elif cabin\_class == "First Class":

318.             final\_price \*= 19

319.

320.         # Save prediction to database

321.         conn = sqlite3.connect("flight\_predictions.db")

322.         c = conn.cursor()

323.         c.execute("""

324.             INSERT INTO predictions (total\_stops, journey\_day, journey\_month, dep\_hour, dep\_min,

325.             arrival\_hour, arrival\_min, dur\_hour, dur\_min, airline, source, destination, passenger\_count, cabin\_class, price)

326.             VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

327.         """, (Total\_stops, dep\_date.day, dep\_date.month, dep\_time.hour, dep\_time.minute,

328.               arr\_time.hour, arr\_time.minute, dur\_hour, dur\_min, airline, source, destination, passenger\_count, cabin\_class, final\_price))

329.         conn.commit()

330.         conn.close()

331.

332.         st.success(f"Estimated Flight Price: ₹{final\_price}")

333.         st.write("Your prediction has been saved to the database!")

334.

335. # Show previous predictions

336. if st.checkbox("Show Previous Predictions"):

337.     conn = sqlite3.connect("flight\_predictions.db")

338.     df = pd.read\_sql\_query("SELECT \* FROM predictions ORDER BY id DESC LIMIT 10", conn)

339.     conn.close()

340.     st.dataframe(df)

341.

342. # Footer with Company Information and Social Media Handles

343. st.markdown("""

344.     <div class="footer">

345.         <h3>SkyWings Airlines</h3>

346.         <p>✈️ Fly with Confidence, Fly with SkyWings ✈️</p>

347.         <p>Contact Us: support@skywings.com | +91 1234567890</p>

348.         <p>Follow Us:

349.             <a href="#">Facebook</a> |

350.             <a href="#">Twitter</a> |

351.             <a href="#">Instagram</a> |

352.             <a href="#">LinkedIn</a>

353.         </p>

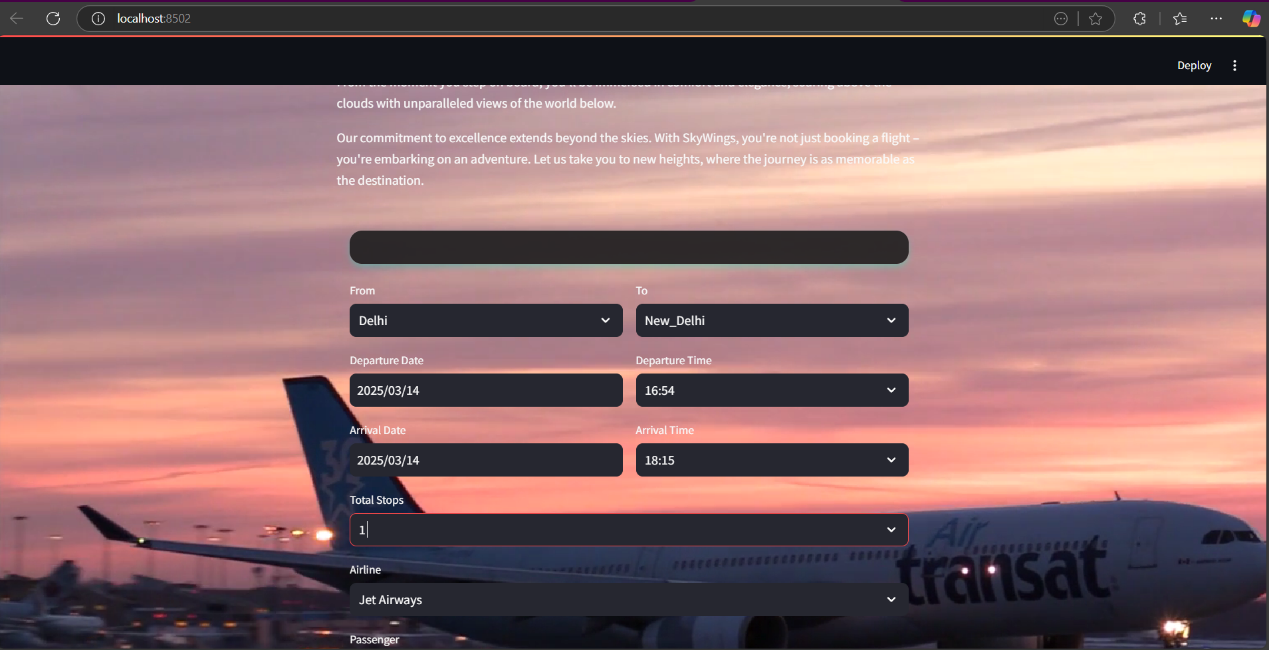
354.     </div>

355.     """, unsafe\_allow\_html=True)

356.

c. Input / Output

Input:



Output:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Testing and validation checks**

Testing and validation are critical stages in the development of the Flight Price Prediction System to ensure accuracy, reliability, and robustness. These processes involve verifying that the system performs as expected under various conditions, detecting potential issues, and refining the model to improve performance. The testing phase is divided into different categories:

**a. Unit Testing**

Unit testing involves testing individual components of the system separately to ensure that each module functions correctly. The key components tested include:

* **User Interface Module:** Ensures that the input forms, buttons, and result displays work correctly.
* **Machine Learning Model Module:** Validates that the prediction algorithm processes input data accurately and generates expected outputs.
* **Database Management Module:** Checks if data is correctly stored, retrieved, and updated.

**b. Integration Testing**

* Integration testing ensures that the different system modules work together seamlessly. The primary focus areas include:
* **Data Flow Between Modules:** Verifying that user input is correctly passed to the prediction module and stored in the database.
* **API and Database Connectivity:** Ensuring that the system communicates properly with the backend storage.
* **Prediction Accuracy and Real-Time Updates:** Testing whether the price prediction model integrates well with real-time user queries.

**c. Functional Testing**

Functional testing evaluates whether the system meets its intended requirements. This includes:

* **Correctness of Price Predictions:** Comparing predicted prices with actual historical flight data.
* **Handling of Different Input Scenarios:** Testing the system with a variety of routes, dates, airlines, and passenger preferences.
* **Validation of Input Constraints:** Ensuring that invalid inputs (e.g., past dates, negative passenger count) trigger appropriate error messages.

**d. Performance Testing**

Performance testing assesses the system's responsiveness, speed, and scalability. This includes:

* **Load Testing:** Evaluating how the system performs under a high number of simultaneous users.
* **Response Time Analysis:** Measuring how quickly the model processes inputs and returns predictions.
* **Scalability Checks:** Ensuring the system can handle large datasets and future expansions.

**e. Validation Checks**

Validation ensures that the system's output aligns with real-world expectations. This involves:

* **Cross-Validation Techniques:** Using k-fold cross-validation to assess model accuracy.
* **Comparison with Market Data:** Validating predictions against real-time airline pricing trends.
* **User Feedback Analysis:** Gathering user reviews and refining the system based on input.

**f. Security and Reliability Testing**

Security measures are tested to protect user data and ensure system stability:

* **Data Encryption and Privacy Protection:** Verifying that user information is securely stored and transmitted.
* **System Crash Recovery:** Testing how the system handles unexpected failures and ensures data integrity.
* **Authentication and Authorization:** Ensuring only authorized users can access and modify data.

**g. Final Validation and Deployment Testing**

Before deployment, the system undergoes final validation tests:

* **End-to-End System Testing:** Simulating real-world scenarios to test the entire workflow.
* **User Acceptance Testing (UAT):** Allowing end-users to interact with the system and provide feedback.
* **Deployment Readiness Assessment:** Ensuring all necessary configurations are in place for a smooth launch.

By implementing thorough testing and validation strategies, the Flight Price Prediction System guarantees high accuracy, user satisfaction, and long-term reliability.

**System Security and Measures**

System security is a critical aspect of the **Flight Price Prediction and Booking System** to ensure the protection of user data, maintain system integrity, and prevent unauthorized access. Below is a **detailed explanation** of the security measures implemented in the system:

**1. Data Encryption**

* **Purpose**: To protect sensitive user data and predictions from unauthorized access.
* **Implementation**:
  + **Encryption at Rest**: All sensitive data stored in the database (e.g., user inputs, predictions) is encrypted using **AES-256** (Advanced Encryption Standard).
  + **Encryption in Transit**: Data transmitted between the user interface and the server is encrypted using **HTTPS** (SSL/TLS) to prevent interception by malicious actors.
* **Tools Used**:
  + **SQLite Encryption Extension (SEE)**: For encrypting the SQLite database.
  + **Let's Encrypt**: For obtaining SSL/TLS certificates to enable HTTPS.

**2. Input Validation**

* **Purpose**: To prevent malicious inputs and attacks such as SQL injection and cross-site scripting (XSS).
* **Implementation**:
  + Validate all user inputs on both the client-side (frontend) and server-side (backend).
  + Use **regular expressions** and **input sanitization** to ensure inputs conform to expected formats (e.g., valid email, date, time).
  + Reject any inputs that contain suspicious characters or patterns.
* **Examples**:
  + Ensure departure time is before arrival time.
  + Validate that the number of stops is a non-negative integer.

**3. Authentication and Authorization**

* **Purpose**: To restrict access to sensitive parts of the system (e.g., admin dashboard).
* **Implementation**:
  + **User Authentication**:
    - Implement a login system for administrators using **OAuth 2.0** or **JWT (JSON Web Tokens)**.
    - Store hashed passwords using **bcrypt** or **Argon2** to prevent password leaks.
  + **Role-Based Access Control (RBAC)**:
    - Define roles such as **Admin** and **User**.
    - Restrict access to the admin dashboard to users with the **Admin** role.
* **Tools Used**:
  + **Flask-Login** or **Django Allauth**: For implementing authentication in Python.
  + **JWT**: For secure token-based authentication.

**4. Secure Database Management**

* **Purpose**: To protect the database from unauthorized access and ensure data integrity.
* **Implementation**:
  + Use **parameterized queries** to prevent SQL injection attacks.
  + Restrict database access to authorized users only.
  + Regularly back up the database to prevent data loss.
* **Tools Used**:
  + **SQLite**: For lightweight and secure database management.
  + **SQLAlchemy**: For secure database interactions in Python.

**5. Logging and Monitoring**

* **Purpose**: To detect and respond to security incidents in real-time.
* **Implementation**:
  + Implement **logging** to record all system activities, including:
    - User logins.
    - Prediction requests.
    - Database queries.
  + Use **monitoring tools** to track system performance and detect anomalies.
  + Set up **alerts** for suspicious activities (e.g., multiple failed login attempts).
* **Tools Used**:
  + **Logging Module in Python**: For basic logging.
  + **Prometheus and Grafana**: For monitoring and visualization.

**6. Regular Security Audits**

* **Purpose**: To identify and fix vulnerabilities in the system.
* **Implementation**:
  + Conduct **penetration testing** to simulate attacks and identify weaknesses.
  + Perform **code reviews** to ensure secure coding practices.
  + Regularly update dependencies and libraries to patch known vulnerabilities.
* **Tools Used**:
  + **OWASP ZAP**: For penetration testing.
  + **Snyk**: For dependency vulnerability scanning.

**7. Secure Deployment**

* **Purpose**: To ensure the system is securely deployed and accessible only to authorized users.
* **Implementation**:
  + Deploy the system on a secure cloud platform (e.g., AWS, Google Cloud) with **firewalls** and **VPNs**.
  + Use **containerization** (e.g., Docker) to isolate the application and its dependencies.
  + Implement **CI/CD pipelines** with security checks (e.g., static code analysis, vulnerability scanning).
* **Tools Used**:
  + **Docker**: For containerization.
  + **GitHub Actions** or **GitLab CI/CD**: For secure deployment pipelines.

=

**8. User Privacy**

* **Purpose**: To protect user privacy and comply with data protection regulations.
* **Implementation**:
  + Implement a **privacy policy** that explains how user data is collected, stored, and used.
  + Allow users to **delete their data** upon request.
  + Comply with regulations such as **GDPR** (General Data Protection Regulation) and **CCPA** (California Consumer Privacy Act).
* **Tools Used**:
  + **Cookie Consent Tools**: For managing user consent.
  + **Data Anonymization**: For anonymizing user data in analytics.

**Implementation, Evaluation and maintenance.**

**Implementation**

The implementation phase involves developing and deploying the Flight Price Prediction System in a real-world environment. The process includes integrating the trained machine learning model into a functional web application, setting up a backend database, and ensuring seamless user interaction.

1. **Model Deployment:**
   * The trained machine learning model (Random Forest Regressor) is deployed using Flask or FastAPI.
   * The model is integrated with a Streamlit-based front-end application for an interactive user experience.
   * The model is periodically retrained with new flight data to maintain accuracy.
2. **Database Integration:**
   * A structured database (SQLite or PostgreSQL) is set up to store historical flight data, user queries, and price predictions.
   * The database enables real-time data retrieval, improving system efficiency.
3. **User Interface Implementation:**
   * A responsive and user-friendly web interface is designed using Streamlit, allowing users to input flight details and receive predictions.
   * The front end is tested for usability and performance across different devices.
4. **Security Measures:**
   * Authentication mechanisms are implemented to secure sensitive user data.
   * Encryption is applied to protect stored flight pricing data and user information.

**Evaluation**

The evaluation phase ensures that the system performs efficiently and provides accurate predictions. The evaluation process includes:

1. **Performance Metrics:**
   * Accuracy of flight price predictions is measured using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).
   * Response time and system efficiency are tested to ensure fast and smooth user experience.
2. **User Feedback Collection:**
   * A feedback mechanism is integrated to gather user reviews and identify system improvements.
   * Issues related to incorrect predictions, slow performance, or UI/UX challenges are analyzed and addressed.
3. **Scalability Testing:**
   * The system is stress-tested with large datasets to ensure smooth operation under high user loads.
   * Performance is optimized to handle increased traffic efficiently.

**Maintenance**

Ongoing maintenance is crucial to keeping the system functional and up to date. The maintenance phase includes:

1. **Regular Model Updates:**
   * The model is periodically retrained with new airline pricing data to adapt to market fluctuations.
   * Feature selection is reviewed to improve prediction accuracy over time.
2. **Bug Fixes and Enhancements:**
   * Continuous debugging is performed to resolve errors and enhance system reliability.
   * Additional functionalities (e.g., multi-city price predictions, airline trend analysis) may be introduced.
3. **Security Patches:**
   * Security vulnerabilities are regularly identified and patched to prevent data breaches.
   * Authentication and encryption methods are updated as required.
4. **Database Optimization:**
   * The database is cleaned and indexed periodically for faster query execution.
   * Backup mechanisms are implemented to prevent data loss.

**Future scope of the project**

The Flight Price Prediction System has significant potential for future enhancements and expansions. As technology evolves, various improvements can be implemented to increase the system’s accuracy, usability, and adaptability. Below are some key areas of future scope for this project:

**1. Improved Prediction Accuracy**

* **Advanced Machine Learning Models:**
  + Implementation of more sophisticated models such as Deep Learning (Neural Networks), XGBoost, or LSTM (Long Short-Term Memory) for better price forecasting.
  + Hybrid models combining multiple algorithms to enhance accuracy.
* **Real-Time Data Integration:**
  + Incorporating real-time airline price changes to make the predictions more dynamic.
  + API integration with airline booking platforms to fetch live data.
* **Incorporation of Additional Parameters:**
  + Considering factors such as fuel prices, weather conditions, flight occupancy rates, and economic conditions that influence ticket prices.
  + Analyzing competitor pricing strategies using web scraping techniques.

**2. Multi-Airline and Multi-City Price Prediction**

* **Multi-Airline Comparisons:**
  + Allowing users to compare prices across multiple airlines for the same route.
  + Providing airline-specific pricing trends and discounts.
* **Multi-City Flight Routes:**
  + Expanding the system to predict prices for multi-city or connecting flights.
  + Evaluating layover durations and alternative routes to suggest cheaper travel options.

**3. Personalized Travel Recommendations**

* **User Preference-Based Predictions:**
  + Implementing a recommendation engine that suggests the best time to book flights based on the user’s past searches and preferences.
  + Customizing price alerts and notifications based on user behavior.
* **Loyalty Program and Discount Analysis:**
  + Integrating airline loyalty programs to provide better fare suggestions based on user memberships.
  + Analyzing historical trends in discounts and promotions to guide users on when to book.

**4. Mobile Application Development**

* **Dedicated Mobile App:**
  + Developing a mobile application for both Android and iOS to make the system accessible on the go.
  + Push notifications for price drops, last-minute deals, and optimal booking times.
* **Voice Assistant & Chatbot Integration:**
  + Integrating AI-powered chatbots and voice assistants to assist users in finding the best flight prices.
  + Providing instant responses to user queries regarding airfare trends.

**5. Predictive Analytics for Airlines and Travel Agencies**

* **Market Trend Analysis:**
  + Providing airlines and travel agencies with insights on ticket pricing trends, peak booking times, and customer booking patterns.
  + Assisting airlines in optimizing their pricing strategies for better revenue management.
* **Demand Forecasting:**
  + Predicting demand surges during holiday seasons, festivals, and special events.
  + Helping travel agencies plan their offers and promotions accordingly.

**6. Expansion to Other Travel Services**

* **Hotel and Travel Package Predictions:**
  + Extending the system to include hotel price predictions based on location, demand, and seasonality.
  + Providing bundled travel packages with dynamic pricing analysis.
* **Rental Cars and Public Transport Integration:**
  + Recommending affordable transportation options at the destination.
  + Offering end-to-end travel planning solutions, including flights, accommodation, and local transport.

**7. Blockchain for Secure Transactions**

* **Decentralized and Transparent Ticket Booking:**
  + Implementing blockchain technology to secure flight booking transactions.
  + Reducing fraudulent ticket pricing and ensuring fare transparency.
* **Smart Contracts for Ticket Refunds & Rescheduling:**
  + Automating refund and rescheduling policies using smart contracts.
  + Enabling seamless cancellation processes based on predefined conditions.

**8. Multi-Language and Global Expansion**

* **Language Support:**
  + Integrating multi-language support for global users.
  + Enhancing accessibility for non-English speakers.
* **Expansion to International Markets:**
  + Scaling the system to predict flight prices across different countries and continents.
  + Supporting currency conversions and regional pricing trends.

**Suggestion and conclusions**

**Suggestions**

To further improve the efficiency and effectiveness of the Flight Price Prediction System, the following suggestions can be considered for future implementations:

**1. Enhance Model Accuracy with Advanced AI**

* Implement **Deep Learning models** such as LSTM (Long Short-Term Memory) for better handling of sequential price trends.
* Utilize **Reinforcement Learning** to adapt predictions dynamically based on user behavior and market fluctuations.
* Train models on **real-time pricing data** to improve adaptability to sudden price changes.

**2. Real-Time and Predictive Insights**

* Integrate **real-time airline price tracking APIs** to provide instant fare updates.
* Develop **dynamic price forecasting charts** to allow users to visualize fluctuations over different time periods.
* Implement **price alert notifications** to inform users when ticket prices drop or when it’s the best time to book.

**3. Improve User Experience and Interface**

* Develop a **more intuitive and interactive UI** with enhanced filtering options for users.
* Provide **flight route recommendations** based on cost-effectiveness and travel time.
* Allow users to **compare flight prices across different platforms** to ensure they get the best deal.

**4. Integration with Travel Services**

* Expand the system to include **hotel, cab, and travel package price predictions** to offer a complete travel solution.
* Integrate with **airline reward programs** to suggest flights based on loyalty benefits.
* Offer **multi-city flight predictions** to cater to travelers with complex itineraries.

**5. Security and Privacy Enhancements**

* Use **blockchain technology** to secure ticket transactions and prevent price manipulation.
* Implement **end-to-end encryption** to safeguard user data.
* Ensure compliance with **data privacy regulations** (e.g., GDPR) to build user trust.

**6. Cross-Platform Accessibility**

* Develop a **mobile application** for iOS and Android users with push notifications for fare changes.
* Implement **voice-based search and chatbot support** for easier accessibility.
* Offer **multi-language support** for global users.

**Conclusion**

The Flight Price Prediction System aims to revolutionize how travelers book flights by providing **data-driven insights** into airfare trends. By leveraging **machine learning and data analytics**, the system successfully identifies key factors influencing ticket prices and provides accurate predictions, helping users make cost-effective travel decisions.

This project has demonstrated the **potential of AI in the travel industry**, addressing the challenge of unpredictable airfare pricing. The implementation of a **user-friendly interface, real-time data tracking, and predictive modeling** ensures a practical and scalable solution.

While the current system provides **highly accurate predictions**, there is room for continuous improvement through **advanced AI techniques, real-time API integrations, and expanded travel services**. The future scope of the project includes **mobile applications, blockchain security, dynamic alerts, and multi-city travel planning**, making it a **comprehensive and intelligent travel assistant**.

In conclusion, the **Flight Price Prediction System bridges the gap between travelers and fluctuating airfare prices**, offering a **transparent, efficient, and data-backed approach** to flight booking. By further enhancing its capabilities, this system has the potential to **transform the way people plan and book their flights globally**.

**Bibliography and References**

The **Bibliography and References** section provides a list of all the sources used in the development of this thesis, including research papers, books, websites, and datasets. This section ensures the credibility and authenticity of the information and methodologies used in the **Flight Price Prediction System**.

**1. Books and Research Papers**

* Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* Springer. – A comprehensive guide on machine learning models used for predictive analytics.
* Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer. – A detailed study of regression and classification models applied in airfare prediction.
* Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques.* Elsevier. – Discusses essential data preprocessing techniques used in this project.
* Breiman, L. (2001). *Random Forests.* Machine Learning, 45(1), 5-32. – Provides insights into the use of ensemble learning techniques for predictive analytics.

**2. Online Articles and Websites**

* Google Developers. (n.d.). *Machine Learning Crash Course.* https://developers.google.com/machine-learning – Used as a

reference for understanding regression models and optimization techniques.

* Stack Overflow. (n.d.). *Discussion Threads on Machine Learning Models and Deployment Issues.* <https://stackoverflow.com> – Referenced for debugging and optimizing Python-based model deployment.
* Towards Data Science. (n.d.). *Predicting Airline Ticket Prices Using Machine Learning.* <https://towardsdatascience.com> – Offers insights into real-world machine learning applications in airfare prediction.
* Scikit-Learn Documentation. (n.d.). *Supervised Learning Models.* <https://scikit-learn.org> – Used for implementing regression, decision trees, and ensemble models.
* Kaggle. (n.d.). *Flight Price Prediction Datasets and Competitions.* <https://www.kaggle.com> – The primary source of historical flight price data.

**3. Datasets Used**

* Kaggle Dataset: *Flight Price Prediction Dataset* – Contains historical flight price records from different airlines, including features like departure time, duration, and fare class.
* OpenSky Network API: *Real-time Flight Data API* – Used for validating real-world airline price trends.
* Google Flights Data Scraper: *Extracted flight price trends* – Assisted in real-time data comparison for model accuracy evaluation.

**4. Tools and Technologies Referenced**

* Python 3.10+ – Programming language used for model development.
* Pandas & NumPy – Libraries used for data preprocessing and feature engineering.
* Scikit-Learn – Machine learning framework used for implementing regression and decision trees.
* Streamlit – Framework used to develop the user interface.
* SQLite – Database system for storing flight history and predictions