Flight Fare Prediction

Submitted In Partial Fulfillment of Requirements
for the Degree Of

Bachelor of Science
(Data Science)

By

Ritika Gupta
Roll No: 31013021004
Guide
Minal Dive

S K Somaiya College
Somaiya Vidyavihar University
Vidyavihar, Mumabi - 400 077
2021-24

Somaiya Vidyavihar University

S K Somaiya College

Certificate

This is to certify that the project report on dissertation entitled “Flight Fare Prediction” is bonafide record of the dissertation work done by Ritika Gupta in the year 2023-24 under the guidance of Minal Dive, Department of Computer Science in partial fulfillment of requirement for the Bachelor of Science degree in Data Science of Somaiya Vidyavihar University.

Guide/Co-guide	Coordinator of Department
Head of the Department	Director

Date:

Place: Mumbai-77

Somaiya Vidyavihar University

S K Somaiya College

Certificate of Approval of Examiners

This is to certify that the project report on dissertation entitled “Flight Fare Prediction” is bonafide record of the dissertation work done by Ritika Gupta in partial fulfillment of requirement for the Bachelor of Science degree in Data Science of Somaiya Vidyavihar University.

External Examiner /Expert

Internal Examiner/ Guide

Date:

Place: Mumbai-77

Somaiya Vidyavihar University

S K Somaiya College

DECLARATION

I declare that this written report submission represents the work done based on my and / or others' ideas with adequately cited and referenced the original source. I also declare that I have adhered to all principles of academic honesty and integrity as I have not misinterpreted or fabricated or falsified any idea/data/fact/source/original work/ matter in my submission.

I understand that any violation of the above will be cause for disciplinary action by the college and may evoke the penal action from the sources which have not been properly cited or from whom proper permission is not sought.

____
Signature of the Student

Ritika Gupta
Name of the Student

31013021004
Roll No.

Date:

Place: Mumbai-77

Acknowledgement

I extend my deepest gratitude to everyone who contributed to the success of this project and the completion of this thesis.

Firstly, I would like to express my sincere appreciation to my thesis advisor, Dr. Swati Maurya Ma’am and Minal Dive Ma'am, whose expertise, understanding, and patience, added considerably to my graduate experience. I am exceedingly grateful for the guidance and the insights he provided throughout this research.

My sincere thanks also go to the faculty members of the Department of Computer Science at

SK Somaiya College for their insightful comments and encouragement, which have been invaluable to this study. Their perspectives and expertise were crucial in shaping both the methodology and direction of this research.

I am thankful to our director CA Monica Lodha for providing us with the facilities for smooth working of project. Lastly, and most importantly, I must express my very profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Ritika Gupta

Abstract

This project develops a comprehensive flight price prediction system aimed at assisting travelers in making informed decisions regarding flight bookings. Leveraging advanced machine learning techniques and robust data analytics, the system predicts future flight prices with high accuracy, allowing users to optimize their travel costs effectively.

The architecture of the system is built around a Python-based Flask server which acts as the backbone, facilitating dynamic data handling, and interaction between the frontend and the backend. The frontend is designed with a focus on user experience, employing responsive web design principles to ensure accessibility across various devices. The backend, integrated with a predictive model, processes vast amounts of historical and real-time flight data to forecast prices.

Significant features of the system include a user-friendly interface that allows travelers to input their travel preferences and receive instant price predictions. The predictive model at the core of the system utilizes algorithms like Random Forests and LSTM networks, which are trained on a rich dataset encompassing historical prices, airline pricing strategies, and other relevant travel factors.

Functionality testing ensures all components operate correctly and efficiently, with particular focus on form validation, API response correctness, and load handling. Performance evaluation tests the system's responsiveness, backend processing efficiency, and predictive accuracy, providing a robust framework that supports scalability and reliability.

By combining sophisticated algorithmic predictions with an intuitive user interface, the project delivers a valuable tool for budget-conscious travelers, enhancing their booking experience and enabling cost savings. The system not only demonstrates high technical proficiency but also aligns with user-centric design principles, making it a significant advancement in the travel technology arena.

Table of Contents

Chapter 1

Introduction

Project Overview:

The primary purpose of creating a flight price prediction system is to offer users, which include both everyday consumers and businesses within the travel industry, insights into flight pricing trends to make informed decisions about purchasing airfare. The goal is to decode the often-volatile pricing strategies employed by airlines and present this information through a predictive tool that anticipates future price movements based on historical and real-time data.

At its core, the Flight Price Prediction System utilizes advanced data analytics and machine learning algorithms to analyze historical flight data, current trends, and pricing patterns. The system is engineered to be user-friendly, providing a seamless interface where users can input parameters such as departure and arrival dates, destinations, and preferred airlines to receive immediate price forecasts. This project aims to democratize travel planning by making it possible for anyone to predict future flight prices with a high degree of accuracy, thus planning their trips at the most economical times.

- Consumer Perspective

From a consumer's perspective, the flight price prediction system aims to provide a strategic advantage in securing lower prices. Air travel can represent a significant expense, especially for frequent flyers and those planning vacations. A reliable prediction tool can guide users on the optimal time to buy tickets to achieve the best rates, potentially saving substantial amounts of money over time.

- Business Perspective

For businesses, particularly those in the travel sector, understanding flight pricing dynamics can improve their service offerings by providing added value to customers, such as advising on the best booking times. Additionally, travel agencies and other stakeholders can use these insights to optimize their own pricing models and promotional strategies to increase competitiveness and profitability.

Significance of the Flight Price Prediction System

The significance of a flight price prediction system extends beyond mere cost savings for individual travelers. It has broader implications for market transparency, consumer empowerment, and economic efficiency.

- Enhancing Market Transparency

This system demystifies the complexities behind flight pricing algorithms. Airlines use sophisticated models that consider multiple variables to set ticket prices. By predicting these prices, the system makes the market more transparent, enabling consumers to understand and anticipate pricing behaviors.

- Empowering Consumers

With access to predictive insights, consumers can make more empowered decisions, aligning their purchasing strategies with predicted price drops or rises. This empowerment helps level the playing field between sophisticated airline pricing algorithms and the average traveler.

- Improving Economic Efficiency

By enabling more predictable and informed purchasing, the system encourages efficiency within the air travel market. Consumers can optimize their spending, while airlines can adjust capacities and pricing strategies based on more accurate demand forecasts derived from user interaction with the tool.

Brief Description of the Technology Stack Used

A robust technology stack is crucial for developing an effective and efficient flight price prediction system. This project utilizes a combination of modern web development technologies and advanced data analytics tools to create a responsive and user-friendly interface coupled with powerful data processing capabilities.

Frontend Technologies

- HTML (HyperText Markup Language): Serves as the backbone of the user interface, structuring the content and elements on the web pages.
- CSS (Cascading Style Sheets): Provides the styling for the HTML content, dictating the layout, colors, fonts, and other visual elements of the application to ensure an attractive and intuitive user experience.
- Bootstrap: A widely used framework for developing responsive and mobile-first websites. Bootstrap offers ready-made components and utilities that speed up development and ensure consistency across different browsers and devices.
- JavaScript: Enhances the interactivity of the web application. It allows the creation of dynamic elements on web pages, such as live price updates, interactive charts, and user input forms, without needing to reload the page.

Backend Technologies

- Python: A versatile programming language known for its readability and vast ecosystem of libraries, particularly favored in data science and machine learning communities. Python handles the core computational logic, data processing, and model execution.
- Flask: A lightweight and flexible Python web application framework. Flask provides the tools to develop a web server that can serve the HTML, CSS, and JavaScript to users and handle their interactions via HTTP requests. It acts as the intermediary between the frontend and the backend, managing data flow from the server to the user interface and vice versa.

Data Analysis and Machine Learning

The predictive capability of the system is powered by machine learning algorithms implemented using Python's data science libraries, such as Pandas for data manipulation, NumPy for numerical data operations, and scikit-learn for building and deploying the predictive models. These libraries offer efficient and effective tools for large-scale data analysis, allowing the system to analyze historical flight price data and learn patterns that can forecast future prices.

Motivation:

The development of a flight price prediction system offers profound benefits and serves a dual purpose: enhancing the user experience for travelers and fostering greater economic efficiency within the air travel industry. This analysis explores the motivation behind such systems, their importance for users, and how they serve to refine decision-making processes for travelers.

Economic Motivations:

The volatility and variability in flight pricing can be a significant challenge for consumers looking to manage travel costs effectively. Airline pricing strategies are complex and multifaceted, influenced by competition, demand, fuel prices, seasonal trends, and many other factors. For the average consumer, these dynamics can be opaque and difficult to navigate.

- Cost Savings

The primary economic motivation for consumers using a flight price prediction system is the potential for cost savings. By predicting when prices for flights are likely to be lowest, the system provides an opportunity for travelers to purchase tickets at the most economical rates. This ability can lead to substantial savings, especially for frequent flyers or families planning vacations.

- Budget Planning

For many travelers, particularly those with rigid budgets, planning expenses in advance is crucial. Flight price prediction tools enable these users to foresee and plan for travel costs more accurately, ensuring that air travel remains within their financial means without compromising other aspects of their trip or overall financial health.

Technological Motivations:

The advent of big data and advancements in machine learning have provided the technological substrate necessary to tackle complex predictive tasks such as flight price forecasting. These technologies have reached a level of maturity that allows for the effective and efficient processing of large datasets, enabling the development of tools that can learn from patterns in historical data to make accurate predictions about future events.

Leveraging Big Data

Airlines and travel agencies generate vast amounts of data regarding flight prices, booking patterns, and passenger preferences. Harnessing this data through sophisticated analytical tools can uncover valuable insights that would otherwise remain hidden. Flight price prediction systems utilize these datasets to train models that understand and anticipate changes in pricing.

Machine Learning Capabilities

Machine learning offers the ability to automate the extraction of patterns from data and use these patterns to make predictions. By applying machine learning algorithms to historical flight pricing data, developers can create models that predict future price trends with a high degree of accuracy. This capability is central to the utility of flight price prediction systems.

Importance of Flight Price Prediction for Users:

- Making Air Travel More Accessible

One of the key benefits of flight price prediction systems is making air travel more accessible to a broader audience. By providing insights into when flight prices are expected to be lower, these tools help people with limited travel budgets to find opportunities that they might otherwise miss. This democratization of travel information helps level the playing field between seasoned travelers and those new to air travel.

- Strategic Planning Advantage

Flight price prediction gives users a strategic planning advantage. Understanding future price trends allows travelers to optimize their booking times and travel dates, aligning their plans with periods when prices are predicted to be lower. This not only saves money but can also lead to better travel experiences by avoiding peak and thus more crowded times.

Enhancing Decision-Making for Travelers:

- Informed Decision-Making

With access to predictive insights, travelers are no longer subject to the whims of fluctuating airline prices. Instead, they can make informed decisions based on data-driven predictions. This shift transforms how individuals approach purchasing flights, moving from a reactive to a proactive stance, where decisions are guided by strategic insight rather than spur-of-the-moment or uninformed choices.

- Confidence in Travel Planning

The certainty that comes with knowing future price trends instills confidence in travelers. They can plan their trips knowing they have secured the best possible deal, reducing the stress and uncertainty that often accompanies travel planning. This confidence can lead to increased satisfaction with the travel experience as a whole, as travelers feel empowered by their ability to manage costs effectively.

- Enhancing Travel Experiences

Beyond just saving money, knowing the best times to book flights can influence other aspects of travel, such as choosing less busy travel periods, optimizing holiday durations, and even selecting destinations based on affordability. These considerations can significantly enhance the overall travel experience, making vacations more enjoyable and less constrained by financial limitations.

The development of a flight price prediction system is motivated by both economic and technological advancements. The importance of such systems to users lies in their ability to make air travel more accessible, provide strategic planning advantages, and enhance the overall travel experience through better decision-making. As technology continues to advance, these tools will only become more refined, further revolutionizing the way travelers plan and book flights.

Scope of the Project Work:

- Enhancing Predictive Accuracy

The primary objective of this thesis is to develop a flight price prediction system with enhanced accuracy. This involves utilizing historical data sets comprising various influencing factors like dates, seasons, economic conditions, and airline pricing strategies to train a predictive model. The goal is to minimize the error margin between the predicted prices and the actual prices, thereby providing users with reliable and actionable insights.

- User-Centric Design

Another crucial objective is to ensure that the system is user-friendly and accessible to a broad audience, including those without a technical background. The interface should be intuitive, providing users with easy-to-understand insights and recommendations on the best times to purchase airline tickets. This aspect of the design focuses on user engagement and retention, ensuring that the system is practical and easy to use.

- Integration with Existing Platforms

To maximize its utility, the system aims to be seamlessly integrable with existing online travel agencies and airline booking platforms. This integration ensures that users can directly benefit from the predictive insights without needing to consult multiple sources. The implementation should support API-based integrations, allowing for real-time data sharing and updates.

- Real-Time Data Processing

The ability to process data in real-time and update predictions accordingly is another critical objective. This feature will allow the system to adjust its forecasts based on the latest available data, ensuring that the predictions remain relevant and timely, thus providing users with the most current information.

Limitations and Boundaries of the Project Scope:

- Data Availability and Quality

One of the primary limitations is the availability and quality of data. The accuracy of predictions heavily depends on the breadth and depth of historical data. However, access to comprehensive and granular data can be challenging due to privacy concerns, proprietary restrictions, or incomplete datasets. The system's effectiveness is contingent on the quality of data fed into the predictive models.

- Predictive Model Complexity

While advanced models can provide more accurate predictions, they also require more computational resources and expertise to develop and maintain. There is a trade-off between model complexity and practical usability, especially concerning real-time processing requirements. The project may need to balance sophistication with performance, ensuring that the system remains responsive and efficient.

- Economic and Regulatory Changes

The air travel industry is highly susceptible to economic fluctuations and regulatory changes, which can impact flight pricing strategies. Predictive models based on historical data might not always capture these dynamic changes effectively. This limitation acknowledges that external economic and regulatory factors could lead to less reliable predictions.

- Technological Constraints

The development and operation of a sophisticated flight price prediction system require substantial computational resources, especially when dealing with large datasets and real-time processing. Budgetary constraints or technological limitations could restrict the scope of the project, affecting the complexity of the models used and the scalability of the system.

- User Adoption and Behavior

The success of the system also depends on user adoption rates and the behavioral responses of users to the predictions made by the system. There is a risk that widespread use of the tool could alter booking patterns in a way that impacts the validity of the predictions, a phenomenon known as the "prediction feedback loop."

- Project Boundaries

The project is delimited not just by its technical and data-driven constraints but also by its focus on specific markets and consumer segments. It does not aim to predict prices for all types of travel or incorporate all possible influencing factors, such as sudden geopolitical events, which can drastically affect air travel dynamics. The system is primarily designed for standard commercial airline travel and does not cover charter flights, private jets, or cargo services.

Chapter 2

Literature Survey

The advancement of flight price prediction systems marries historical data analysis with machine learning and user interface design. This literature survey examines previous work, current technologies, and foundational insights that underpin the thesis. Recent models in flight price prediction have evolved from simple regression techniques to sophisticated machine learning algorithms like Random Forests and neural networks. These are designed to interpret data patterns and forecast prices based on variables including time of booking, seasonal trends, and promotional strategies. Platforms like Hopper and Skyscanner utilize these models, integrating predictive analytics into their services to advise users on optimal purchase times. Reliable data is critical for prediction accuracy and typically includes comprehensive historical pricing from airlines and booking platforms. The effectiveness of a prediction model heavily depends on the quality and timeliness of the data it processes. The user interface for these systems is built using web technologies such as HTML, CSS, JavaScript, and Bootstrap, allowing for the creation of responsive, dynamic user interfaces. Backend development often leverages Python and its frameworks like Flask, which supports data processing with libraries such as Pandas and NumPy, and provides a flexible environment for developing and deploying web applications.

The development and evolution of flight price prediction systems have played a crucial role in enhancing the efficiency and effectiveness of travel planning. As airline ticket prices fluctuate dramatically due to a variety of factors, including demand, seasonality, fuel prices, and even global economic conditions, predictive analytics has become a critical tool in helping both consumers and companies optimize their planning and pricing strategies. This extensive exploration discusses previous work in the field, examines the current technologies employed in flight price prediction models and tools, and delves into the reliability of data sources that underpin these predictions.

Previous Work and Technologies:

Early attempts at flight price prediction were rudimentary, often relying solely on historical averages and simple econometric models. These models typically used linear regression techniques to predict future prices based on past trends. However, as the availability of data increased and computational power improved, more sophisticated methods began to be employed, including machine learning algorithms such as decision trees, random forests, and neural networks.

Machine learning has allowed for more accurate and dynamic pricing models. For example, Random Forests, a robust ensemble technique, uses multiple decision trees to make predictions by averaging the results, reducing overfitting and improving predictive accuracy. Similarly, neural networks, especially recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), are particularly adept at processing sequences of data, making them ideal for time series analysis like price trends.

In addition to these technologies, big data analytics has also played a significant role. The integration of big data technologies with machine learning algorithms enables the processing and analysis of vast amounts of data in real-time, which is essential for capturing the dynamic nature of flight pricing.

The evolution of flight price prediction technologies reveals a trajectory marked by increasing complexity and sophistication, driven by advancements in computational capabilities and the availability of vast datasets. This progression from basic econometric models to advanced machine learning and big data solutions represents a significant shift in how the travel industry approaches the challenge of dynamic pricing. Here, we delve deeper into the technologies that have shaped current predictive models, exploring not only their operational mechanics but also their implications for the future of travel technology.

- Advanced Machine Learning Techniques

Beyond the already mentioned Random Forests and neural networks, other machine learning algorithms have also contributed to the refinement of flight price prediction models. Gradient boosting machines (GBMs) and support vector machines (SVMs) have been adopted for their robustness in handling non-linear data relationships, which are common in pricing models due to external influences like economic changes or varying customer preferences.

- Neural Networks and Deep Learning

Deep learning, a subset of machine learning involving neural networks with many layers, has particularly revolutionized predictive analytics in flight pricing. Convolutional Neural Networks (CNNs), traditionally used in image recognition, have been adapted for time series data, providing improved feature extraction capabilities for sequence prediction tasks such as price forecasting. This adaptation has allowed for the inclusion of various data types, such as images of economic events or graphs of price trends, to be used directly in the prediction models, enhancing the contextual awareness of the systems.

- Integration of Big Data Technologies

The integration of big data technologies has facilitated the real-time analysis of massive datasets. Platforms like Apache Hadoop and Spark enable the processing of data at a scale and speed that traditional data processing software cannot match. These platforms support predictive modeling by allowing for the rapid execution of data-intensive operations like sorting, filtering, and aggregating large datasets, which are crucial for updating prediction models with the latest available data.

- Role of Artificial Intelligence (AI)

Artificial intelligence, particularly in the form of predictive analytics, has started to play a more prominent role in customizing pricing strategies. AI can leverage consumer behavior data to predict not only when prices are likely to increase or decrease but also how sensitive consumers are to different price points. This capability allows airlines and travel agencies to optimize their pricing strategies not just for market trends but also for individual consumer profiles, a practice known as personalized pricing.

- Challenges and Future Directions

Despite these advancements, the flight price prediction models face significant challenges, primarily concerning data quality and privacy. Ensuring the accuracy, completeness, and timeliness of data is critical, as the reliability of predictions depends heavily on the quality of input data. Furthermore, as predictive models increasingly incorporate more personal data to enhance accuracy, issues of data privacy and security become more pressing. The industry must navigate these challenges while adhering to regulatory standards and maintaining consumer trust.

Moreover, as the global landscape changes due to factors like climate change, economic shifts, and technological disruptions, predictive models need to be adaptable and robust against such uncertainties. Future research might focus on incorporating these aspects into models, using scenario analysis and stress testing to better predict under varying conditions.

The technological advancements in flight price prediction illustrate a broader trend towards more data-driven, intelligent systems in commercial applications. As these technologies continue to evolve, they promise not only to enhance the efficiency of the travel industry but also to offer more personalized and cost-effective travel experiences for consumers. The continuous improvement of these predictive tools will likely be a key factor in the sustained growth and innovation within the airline and travel industries.

Overview of Existing Flight Price Prediction Models and Tools:

Several commercial and open-source tools have been developed to predict flight prices. For instance, online travel agencies (OTAs) like Expedia and booking platforms such as Skyscanner and Kayak use sophisticated prediction algorithms to advise customers on the best times to buy their tickets. These platforms typically aggregate data from various airlines and booking systems, apply predictive models, and use the results to provide actionable insights to users.

Furthermore, specialized applications such as Hopper and Google Flights have leveraged predictive analytics to create features that predict future flight prices with considerable accuracy. Hopper, for example, uses historical data to inform users whether they should buy their tickets now or wait for a price drop. The app can notify users when the prices for their selected flights are at their lowest predicted rates.

The landscape of flight price prediction is diverse, encompassing a range of tools and technologies that cater to both consumer and industry needs. These models and tools, designed with varying levels of sophistication, have significantly altered the way consumers plan and purchase their travel. Below is a deeper look into how these systems operate and the methodologies they employ to forecast flight prices.

- Commercial Prediction Tools

Online travel agencies (OTAs) and booking platforms like Expedia, Skyscanner, and Kayak integrate advanced prediction algorithms into their search engines. These platforms harness the power of big data analytics by collecting vast amounts of information from airline databases, historical transactions, booking trends, and even seasonal variations. They then employ various predictive algorithms—ranging from linear regression models to more complex neural networks—to analyze these data sets and predict price trends. The result is not just a static listing of flight prices but a dynamic prediction about when prices are likely to increase or decrease. This allows consumers to make informed decisions about the best times to purchase airline tickets.

- Specialized Predictive Applications

Applications such as Hopper and Google Flights take a slightly different approach by focusing primarily on the predictive aspect of flight pricing. Hopper, for instance, analyzes billions of price quotes per day and uses that data to predict with considerable accuracy whether flight prices will go up or down. The app employs machine learning techniques that continuously refine their predictive accuracy by learning from new data. Consumers benefit by receiving push notifications when the algorithm determines that the price of a specific flight is at its lowest, or when it anticipates a rise in price, advising users to book quickly.

Methodologies Employed:

- Time Series Forecasting: Many flight price prediction tools use time series analysis to forecast future prices. This involves identifying patterns in historical data over time and using them to predict future trends. Techniques like ARIMA (AutoRegressive Integrated Moving Average) and Exponential Smoothing are common in these analyses.

- Machine Learning Models: More sophisticated tools deploy machine learning models that can handle non-linear relationships and interactions between multiple variables. For instance, ensemble methods like Gradient Boosting or Random Forests are popular for their ability to provide high accuracy and handle overfitting by averaging multiple deep decision trees.

- Deep Learning: For applications dealing with massive datasets or requiring the analysis of complex patterns, deep learning models such as LSTM (Long Short-Term Memory) networks are used. These are especially good at handling data where sequences and time are crucial elements, as is the case with flight prices which can fluctuate based on time-of-day, day-of-week, and proximity to the travel date.

Challenges and Innovations

While predictive technologies have advanced significantly, they still face challenges such as rapid changes in airline pricing strategies or external factors like political unrest or pandemics affecting travel norms and, consequently, flight pricing. Innovations continue to emerge in response, such as integrating broader data sets that include not just historical pricing but also real-time economic indicators, competitor pricing, and even social media trends to gauge consumer sentiment towards travel to certain destinations.

Thus, the field of flight price prediction is continually evolving, driven by advancements in data analytics and machine learning. As these technologies grow more sophisticated, they not only enhance consumer experiences by providing more accurate predictions and actionable insights but also help airlines and travel agencies optimize their pricing strategies for maximum profitability. The ongoing development in predictive tools promises even greater advancements, with potential integrations of AI to further refine personalization and pricing accuracy in the travel industry.

Data Sources and Their Reliability for Price Prediction

The reliability of flight price prediction models is heavily dependent on the quality and robustness of the data used. These models often rely on diverse data sources, including historical price data, real-time booking information, economic indicators, airline promotional calendars, and even weather forecasts.

Historical price data is the backbone of most predictive models, providing the necessary context for understanding pricing patterns and trends. However, the accuracy of predictions can be significantly impacted by the freshness of the data. In an industry as dynamic as airline travel, prices can change dramatically in a short period due to unforeseen events such as natural disasters, geopolitical tensions, or sudden changes in oil prices.

Real-time data integration is, therefore, crucial. Modern predictive systems frequently incorporate APIs that pull current data from various airlines and booking engines to stay updated with the latest price changes. Additionally, the use of advanced data cleaning and preprocessing techniques ensures the reliability of the data inputs, which in turn enhances the accuracy of the output predictions.

Expanding on the necessity and complexity of integrating various data sources into flight price prediction models highlights how sophisticated these systems have become to cater to dynamic market conditions and user needs.

- Economic Indicators and Market Dynamics: Economic indicators like inflation rates, currency fluctuations, and economic growth metrics play a crucial role in determining airfare prices. These indicators often predict general travel trends which, when analyzed alongside historical data, can provide insights into future pricing trends. For example, a strengthening economy generally leads to an increase in travel demand, pushing prices up. Conversely, economic downturns typically result in lower airfare as airlines attempt to maintain passenger volumes.

- Airline Promotional Strategies: Airlines frequently adjust their pricing based on strategic promotions and discounts which can significantly deviate from historical trends. Understanding these patterns requires predictive models to factor in promotional calendars and past marketing tactics. This involves not only tracking announcements from airlines but also predicting potential future promotions based on date, seasonality, and even competitor pricing strategies.

- Weather Forecasts and Seasonal Variability: Weather conditions and seasonal changes can drastically affect flight prices. For instance, the start of the hurricane season can lead to a drop in prices for flights to affected regions due to decreased demand. Conversely, pleasant weather can drive up prices due to increased tourist attraction. Predictive models must account for these variations by integrating weather forecasting data to more accurately determine how these factors will influence flight pricing.

- Geopolitical Events: Unexpected geopolitical events such as conflicts, terrorism, or political instability in a region can lead to sudden and significant fluctuations in air travel pricing. Models need to be designed to quickly adapt to these changes, which may involve integrating news feeds and global event databases to adjust predictions in real-time.

- Machine Learning and Artificial Intelligence: To manage the complexity of integrating and analyzing these diverse data sources, advanced machine learning and artificial intelligence technologies are employed. Techniques such as deep learning and ensemble methods can discern complex patterns and interactions within the data that may not be immediately obvious. These models can dynamically adjust to new data, learning continuously as more data becomes available.

- Data Security and Privacy: With the increasing reliance on diverse and real-time data sources, ensuring the security and privacy of the data becomes paramount. Predictive models must comply with data protection regulations such as GDPR in Europe or CCPA in California, which govern the use of personal information. This adds another layer of complexity to data integration and usage in flight price prediction models.

By addressing these intricate and interconnected factors, flight price prediction models can be developed not only to forecast prices with higher accuracy but also to be resilient enough to adapt to the ever-changing global landscape affecting air travel. This holistic approach ensures that predictive tools remain reliable and effective in delivering value to both businesses and consumers in the travel industry.

Therefore, the field of flight price prediction has seen significant technological advancements from basic econometric models to sophisticated machine learning and big data solutions. The ongoing development of these technologies, coupled with improvements in data collection and processing, promises even more accurate and timely predictions in the future. This continuous innovation not only benefits consumers by providing them with better tools for making informed decisions but also helps airlines optimize their revenue management strategies.

Chapter 3

Design and Development

The flight price prediction system is crafted to offer users a robust tool that predicts airfare costs, helping them make informed travel decisions at the most economical rates. The design and development of this system emphasize creating an intuitive, efficient, and visually appealing interface that caters to the needs of diverse users, ranging from casual vacationers to frequent business travelers.

The user interface (UI) of this system has been meticulously designed to ensure a seamless user experience. It incorporates a straightforward layout where functionality meets modern aesthetics, emphasizing clarity and ease of navigation. The UI layout leverages a clean, minimalistic design approach with a coherent color scheme and consistent typography, which not only aligns with the brand's visual identity but also improves readability and user engagement.

Under the hood, the design utilizes HTML5 and CSS3 for structuring and styling the web pages, ensuring that the content is organized semantically and is accessible across all devices and browsers. The system also integrates Bootstrap, a powerful front-end framework, to achieve responsiveness and maintain design consistency across various screen sizes. This choice supports the goal of providing a universally accessible platform, irrespective of the user's device.

In this chapter, we delve deeper into the specific components of the user interface design, from the layout choices to the functionality of form components. This includes a detailed exploration of how HTML, CSS, and Bootstrap have been utilized to create a responsive and dynamic user experience that not only looks good but also performs excellently across all platforms. This thorough design approach ensures that users can efficiently and effectively interact with the system, making the most of its capabilities to predict flight prices accurately.

User Interface Design

- Layout and Design Choices:

The user interface (UI) of the flight price prediction system is designed to be intuitive, user-friendly, and aesthetically pleasing. The layout follows a clean and minimalistic style, which helps users navigate the system effortlessly. The primary design choices include a coherent color scheme that aligns with the brand's visual identity, a consistent font style that enhances readability, and a layout that promotes a logical flow of information.

The homepage serves as the entry point and features a streamlined search interface where users can input their travel details. This page also includes visual elements such as banners and icons that are both functional and visually appealing, guiding the user's journey through the site. The use of whitespace is strategic, aiming to prevent any visual clutter that might overwhelm the user.

- HTML Structure and CSS Styling:

Within the HTML structure, thoughtful attention is paid to the use of classes and IDs to facilitate precise styling and scripting. Classes are widely utilized for elements that occur repeatedly with similar styling, such as buttons, form inputs, and content cards, ensuring that the CSS code is both reusable and efficient. IDs, on the other hand, are reserved for unique elements that require specific styling or are targeted by JavaScript functions, enhancing the specificity and functionality of the site design.

The CSS styling is meticulously crafted to ensure that every visual element aligns with the overall design philosophy of simplicity and user-friendliness. For instance, the CSS leverages advanced selectors and pseudo-elements to enhance the visual appeal without cluttering the HTML with additional markup. Pseudo-classes like `:hover` and `:focus` are used to improve interactivity, providing visual feedback on clickable elements, which is crucial for a good user experience.

The responsive nature of the design is achieved through the strategic use of media queries that adjust styles according to the screen size. This responsiveness ensures that the website's layout, typography, and navigational elements are optimally displayed on devices ranging from mobile phones to large desktop monitors. The CSS Grid and Flexbox are particularly effective in this regard, allowing for complex layouts that are both flexible and easy to maintain. Grid layouts are used for major layout frames, offering a robust structure, while Flexbox handles alignment and spacing within grid cells or standalone elements, providing flexibility and precise control over the spacing and alignment.

Typography and color schemes are carefully chosen to ensure high readability and to convey a sense of reliability and professionalism. The use of CSS variables for defining colors, font sizes, and margins promotes consistency throughout the site. This approach not only enhances the thematic coherence of the design but also simplifies maintenance and scalability by centralizing style definitions.

In addition, CSS animations and transitions are subtly incorporated to enrich user interactions without overwhelming the user interface. These animations include smooth transitions on hover effects, gradual reveals of content as it loads, and attention-grabbing yet elegant animations on call-to-action buttons. This layer of interactivity not only enhances aesthetic appeal but also engages users, making the navigation experience more intuitive and enjoyable.

Overall, the combination of a well-structured HTML foundation with robust and responsive CSS styling ensures that the flight price prediction system is both beautiful and functional, providing a seamless and engaging user experience that aligns with the latest web standards and user expectations.

Functional Components:

The functional components of the flight price prediction system are crucial for user interaction and data processing. Central to these interactions are the form components used for data input, which are carefully designed to facilitate ease of use and accurate data collection.

Form Components for Data Input:

The system utilizes a range of HTML form elements, each serving a specific function:

- Text Inputs: These are used for user entries such as names or destinations. Attributes like \`placeholder\` offer hints about the required input format, while \`required\` and \`pattern\` attributes ensure that the input matches specific criteria, improving form validation and user experience.
- Date Pickers: For travel dates, HTML5 \`date\` input types are employed, providing users with a convenient way to select dates. These pickers are integrated with JavaScript to handle constraints such as disallowing past dates or ensuring the return date is after the departure date.
- Dropdown Lists: To select the number of passengers or class of service (economy, business, first class), \`<select>\` elements with predefined options are used. This ensures data consistency and improves the backend processing.
- Checkboxes and Radio Buttons: These are used for selections that require boolean choices like opting in for insurance, or choosing a preferred airline, where only one choice is applicable.
- Submit Button: Styled prominently, the submit button triggers data submission. It's enhanced with JavaScript to validate form data before submission, providing immediate feedback to users if there are errors.

Each form component is styled with CSS for consistency and clarity, ensuring that elements are visually harmonious and functionally clear.

Role of Bootstrap in Responsive Design

Bootstrap plays a pivotal role in ensuring that the website is responsive and accessible across all devices. Here’s how Bootstrap enhances the design:

- Grid System: Bootstrap’s grid system is used to create a responsive layout structure. It divides the screen into a series of rows and columns, making it easy to place and align form components dynamically based on the screen size. This system adjusts automatically to fit the screen, ensuring that the form remains usable on both small mobile devices and large desktop screens.
- Predefined Classes: Bootstrap offers a wide range of predefined CSS classes that help in quickly designing form components. For example, classes like `form-control` are used to ensure that form fields are optimal width and are easily readable. Bootstrap’s `btn` classes are used to style buttons that are visually compelling and consistent.
- Components: Bootstrap includes various ready-made components like modals, dropdowns, alerts, and tooltips that can be integrated into the website to enhance functionality and interactivity without compromising on responsiveness.
- Utility Classes: For fine-tuning, Bootstrap’s utility classes like `mt-3` (margin-top), `text-center`, or `d-none` (display none) provide quick ways to adjust the styling of elements without adding additional CSS. This speeds up the development process and ensures that adjustments do not affect the responsive nature of the design.
- JavaScript Plugins: Bootstrap’s JavaScript plugins are used to enhance form components. For example, modal dialogs for confirming form submissions, or collapse components for advanced search options maintain functionality across different devices and browsers.

By leveraging Bootstrap’s features, the design ensures that the form components are not only functional but also adaptable to any device, enhancing the user experience and accessibility of the flight price prediction system.

Chapter 4

Backend Implementation and Model Integration

This chapter delves into the technical specifics of setting up and integrating the backend functionalities for the flight price prediction system, leveraging Python and Flask to create a robust server environment. The implementation covers the complete backend architecture, from server setup and routing to predictive model integration and potential database usage. Each section of the backend implementation is crucial for ensuring the system operates efficiently and accurately in real-time environments.

Introduction to Flask and its Ecosystem

- Flask at a Glance: Flask is a micro web framework for Python, known for its simplicity and flexibility. Unlike more heavyweight frameworks like Django, Flask provides the bare essentials to get a web application running quickly, making it an excellent choice for small to medium projects, and for developers who prefer finer control over the components they use.
- Why Flask for Web Development: Flask's lightweight and modular design is particularly appealing in projects requiring a custom approach without the overhead of unnecessary features. It supports extensions that can add application features as if they were implemented in Flask itself. This modularity makes it ideal for projects like a flight price prediction system, where specific user interactions and data processing mechanisms are needed.

Setting Up the Python Flask Server

- Installation and Initial Setup: The first step in using Flask is setting up the development environment. This involves installing Python, followed by Flask using pip, Python's package manager. The simplicity of Flask's installation and configuration underscores its appeal to developers who need to prototype rapidly.

- Project Structure: Structuring a Flask project involves organizing the code, templates, and static files. This organization is crucial for maintaining clean code, especially as the application scales. Typically, a Flask project includes directories for templates (HTML files), static files (CSS and JavaScript), and a main directory for Python scripts. This separation helps in managing different aspects of the application such as presentation, business logic, and application configuration separately.

Configuring Flask Applications

- Understanding Flask Configurations: Flask can be configured to suit different development needs. Configuration variables can control from debugging support, database details, to application secret keys. Flask's ability to be configured from a single file makes managing these settings across various development, testing, and production environments easier.
- Environment Specific Configuration: Managing different configurations for development, testing, and production environments is critical. Flask allows developers to inherit base configurations and override them for specific environments. This practice reduces errors and enhances security by separating development configurations from production ones.

Designing Routes and Endpoints in Flask

- Flask Routing Basics: Routes in Flask are used to direct users to different parts of a web application based on the URL. Flask uses decorators to link functions to specific URLs, making it intuitive to handle different HTTP methods like GET and POST.
- Endpoints for Dynamic Data Processing: For a flight price prediction system, defining endpoints for submitting user inputs and retrieving predictions is crucial. These endpoints handle data dynamically, accepting inputs and providing outputs in various formats (like

JSON), making Flask particularly useful due to its request and response handling capabilities.

Integrating Front-end and Back-end

- Dynamic Interaction: The core of a flight price prediction system lies in its ability to dynamically interact with user inputs. Flask seamlessly integrates with front-end technologies, allowing for responsive updates to the UI based on back-end data processing. This integration is facilitated through AJAX, JSON, or simple HTML forms.
- Security and Data Handling: As Flask applications often handle sensitive user data, securing endpoints is paramount. Flask supports various methods to secure applications, including CSRF protection and SSL encryption. Proper handling and validation of user data to prevent common vulnerabilities like SQL injection and Cross-site Scripting (XSS) are also integral to Flask's ecosystem.

Flask in Production

- Deploying Flask Applications: Deploying a Flask application involves more than just uploading code to a server. It requires setting up a web server gateway interface (WSGI) server, like Gunicorn, and a web server like Nginx that can serve static files and handle client requests by passing them to the WSGI server.
- Performance and Scalability: While Flask is suitable for small to medium applications, concerns about scalability can arise as the application grows. However, with proper architecture design, use of caching, load balancing, and database optimizations, Flask applications can scale effectively to handle increased loads.

Flask's flexibility, ease of use, and the robust ecosystem make it an excellent choice for developing web applications like a flight price prediction system. By leveraging Flask's capabilities in routing, configuration, and integration with other technologies, developers can create efficient, secure, and scalable web applications.

This comprehensive exploration offers a foundational understanding of setting up and developing a Flask-based application for dynamic data processing tasks such as flight price prediction, covering everything from initial setup to deployment and scaling, without delving into actual code. This narrative not only illuminates Flask's practicality and versatility but also underscores its suitability for projects requiring a high degree of customization and control over the web development process.

Integration of Predictive Model

The integration of a predictive model into a web application like a flight price prediction system involves several critical steps that ensure the model not only functions correctly but also efficiently process input data and returns predictions in real-time. This section delves deep into how a predictive model, typically developed in Python, is incorporated into the Flask-based backend, how it processes input data, and how the front-end and backend communicate dynamically.

Understanding Model Integration

Model integration in the context of a flight price prediction system involves embedding a machine learning model into a web application environment, specifically within a Flask-based backend. This process ensures that the predictive model, once trained and validated, can be efficiently used to forecast flight prices in real time as users interact with the application. Here's a detailed breakdown of the key steps involved in understanding and executing model integration:

1. Model Development and Training

The first step in integrating a predictive model is the development and training phase. This involves selecting the appropriate machine learning algorithms based on the problem at hand, which in this case is predicting flight prices. Algorithms such as linear regression, decision trees, random forests, or neural networks might be chosen depending on the complexity required and the nature of the data.

- Data Collection: It starts with gathering extensive historical data on flight prices, which may include features like flight duration, airline, time of booking, seasonal trends, and promotional offers.
- Feature Engineering: This step involves creating meaningful variables from the data that can significantly impact the predictions. This could include extracting day of the week from date variables, calculating days to departure, or categorizing flights into different price bands.
- Model Training: Using the prepared dataset, the model is trained to identify patterns and learn from the data. This phase may involve splitting the data into training and testing sets to validate the model's performance.
- Model Evaluation: After training, the model is evaluated using suitable metrics like RMSE (Root Mean Square Error) for regression tasks. It's crucial to assess the model's accuracy and its ability to generalize to unseen data.

2. Model Serialization

Once the model is trained and validated, it must be saved or serialized for later use in the application. Serialization involves converting the model into a binary format or a file that can be loaded and used directly without retraining.

- Using Pickle/Joblib: Python provides libraries like `pickle` and `joblib` for serializing and deserializing Python objects. `joblib` is particularly efficient for objects that carry large NumPy arrays internally, as is often the case with trained machine learning models.
- Model Storage: The serialized model file is stored on the server or in the cloud, ensuring that it can be easily accessed by the Flask application upon startup or upon user request.

3. Model Deployment

Deploying the model into the Flask application involves loading the serialized model into the application's environment and making it ready for real-time data prediction.

- Loading the Model: At the startup of the Flask app, the serialized model is loaded into memory. This setup minimizes the response time when a prediction request is made.
- API Setup: Flask routes are configured to handle requests that involve model predictions. For instance, a route may be dedicated to receiving input parameters from the user interface, passing them to the model, and sending back the predicted flight prices.

4. Continuous Monitoring and Updating

Post-deployment, continuous monitoring of the model's performance is crucial due to the dynamic nature of flight pricing. As new data becomes available or as user feedback is collected, the model may require retraining or fine-tuning to maintain or improve its accuracy.

- Feedback Loop: Implementing a mechanism for users to provide feedback on the accuracy of predictions can help in gathering additional data that might be used to further refine the model.
- Periodic Updates: Depending on the model's performance and the availability of new data, periodic updates might be necessary. This could involve retraining the model with updated data or tweaking the algorithm to better capture the dynamics of flight pricing.

Understanding and implementing model integration within a Flask-based flight price prediction system is a multifaceted process that involves meticulous planning, execution, and ongoing management. By following these steps, developers can ensure that the predictive model is not only accurate and efficient but also robust and scalable to meet the demands of real-world users.

Data Processing in the Predictive Model

Data processing is a critical stage in the development and operation of a predictive model, especially in complex domains like flight price prediction. This process encompasses several key steps: data collection, data cleaning, feature engineering, and data transformation. Each step ensures that the data is optimized for the algorithms used in the model, enhancing the accuracy and reliability of predictions. Below, we elaborate on these crucial phases in the context of a flight price prediction model.

1. Data Collection

Data collection is the foundational step where relevant data is gathered from various sources. In the case of flight price prediction, data can be sourced from airline databases, travel booking websites, historical flight price records, weather services, and economic indicators such as fuel prices and exchange rates.

- Historical Prices: Includes past data on flight prices, which is critical for time series forecasting models.
- Flight Details: Data such as departure and arrival times, airports, airline companies, and flight durations.
- External Factors: Includes holidays, school vacations, special events, and any other data that might affect flight prices.

2. Data Cleaning

Once data is collected, it often contains inconsistencies, errors, or missing values that must be addressed before further processing. Data cleaning includes:

- Handling Missing Data: Deciding whether to fill in missing values using techniques like imputation, or to remove data points that contain missing values.
- Removing Outliers: Identifying and excluding data points that represent extreme cases, which are unlikely to recur and could skew the model results.
- Error Correction: Rectifying any errors in the data, which might include typographical errors, duplicates, or incorrect data entries.

3. Feature Engineering

Feature engineering involves creating predictive features from the raw data that significantly enhance the model’s performance. This step is crucial in transforming practical data inputs into forms that are meaningful and usable by the model.

- Temporal Features: Extracting time-related features like the day of the week, month, or part of the day from date-time fields.
- Categorical Encoding: Transforming categorical data, such as airlines or destinations, into numerical values through techniques like one-hot encoding or label encoding.
- Interaction Features: Creating new features by combining two or more existing variables, e.g., an interaction term between 'holiday' and 'weekend' might capture higher prices during long weekends.

4. Data Transformation

After cleaning and engineering features, data transformation is applied to normalize or scale the data appropriately, which aids in the convergence and performance of the machine learning algorithms.

- Scaling: Standardization or normalization of features, especially when using models like SVM or neural networks, to ensure that all input features contribute equally to the prediction.
- Normalization: Applying transformations such as log-normalization to skewed data to make the distribution more symmetric and conducive to modeling.

5. Data Splitting

Finally, the data is split into training and testing sets. This practice allows the model to learn on a designated set of data (training set) and validate its predictions on unseen data (testing set).

- Training Set: Used to train the model, comprising about 70-80% of the data.

- Validation Set: Sometimes, a portion of the training set is further separated out to validate the model during training (used in tuning the model's hyperparameters).
- Testing Set: Used only after the model configuration is finalized to assess its performance, representing about 20-30% of the data.

Effective data processing forms the backbone of a robust predictive model, particularly in a complex and dynamic field like flight price prediction. By meticulously collecting, cleaning, engineering, transforming, and properly splitting the data, one can build a predictive model that is not only accurate but also robust against real-world variations and capable of providing reliable predictions that can significantly benefit both consumers and businesses in the travel industry.

Dynamic Data Handling and Frontend-Backend Interaction

Dynamic data handling and the interaction between the frontend and backend are critical components of modern web applications, particularly in systems like flight price prediction platforms. These elements ensure that the application can respond in real-time to user inputs, process data effectively, and provide accurate and timely outputs. This section delves into the intricacies of handling dynamic data and the synchronization between the frontend and backend layers of a flight price prediction application.

1. Frontend-Backend Interaction

The interaction between the frontend and backend is facilitated through well-defined APIs (Application Programming Interfaces) that manage the data flow between the user interface and the server where the processing occurs. Here's how this typically functions in a flight price prediction system:

- API Requests: The frontend sends requests to the backend via APIs. These requests could be for fetching the latest flight prices, submitting user preferences, or retrieving prediction results.
- Data Exchange Formats: Data exchanged between the frontend and backend is typically formatted as JSON (JavaScript Object Notation), which is lightweight and easily handled

by web technologies. JSON structures allow complex data like flight schedules and pricing forecasts to be parsed and manipulated with ease.

- Asynchronous Communication: Modern web applications often use asynchronous JavaScript (AJAX) for backend communication. This method allows the webpage to request small chunks of data from the server without having to reload the entire page. This is essential for updating flight prices in real-time without user intervention.

2. Dynamic Data Handling

Handling dynamic data involves not only processing the data in real-time but also ensuring that the data remains consistent, accurate, and reflective of the latest available information. Key considerations include:

- Caching Mechanisms: Implementing caching strategies to reduce the load on the backend and speed up response times for frequently requested data. For instance, common flight searches could be cached at the server or client level, reducing the need to repeatedly process the same requests.
- Concurrency Control: Managing concurrency is vital, especially when dealing with data like flight prices that can change frequently and unpredictably. Optimistic or pessimistic locking mechanisms can be used to prevent data conflicts when multiple users are interacting with the system simultaneously.

3. Backend Processing

The backend is where the bulk of the data processing and machine learning computation takes place. Essential components include:

- Data Pipelines: Automated pipelines that fetch, process, and analyze data in a streamlined manner. These pipelines handle tasks from data ingestion and cleaning to feature extraction and model inference.
- Model Deployment: Machine learning models are deployed on the server, where they can process input data received from the frontend and return predictions. These models might

be running continuously or invoked as needed, depending on the architecture and requirements of the application.

- Security and Data Privacy: Ensuring that all data exchanges between the frontend and backend are secure, using protocols like HTTPS to encrypt data in transit. Additionally, implementing proper authentication and authorization practices to protect sensitive user data and system integrity.

The dynamic interaction between the frontend and backend, coupled with effective data handling strategies, forms the core of a robust flight price prediction system. This setup not only enhances user experience by providing real-time, accurate information but also maintains high operational efficiency and reliability. Advanced techniques in web development and backend processing ensure that the system can handle complex datasets and user interactions at scale, making it an indispensable tool in the travel and tourism industry.

Integrating a predictive model into a Flask application for flight price prediction involves careful planning around data handling, model management, and seamless interaction between the frontend and backend. The process requires attention to detail in terms of data preprocessing, model serialization and deployment, and the dynamic communication between client-side and server-side components. By effectively managing these aspects, the application not only provides valuable predictions to users but also enhances user experience through efficient and responsive design.

Chapter 5

Testing and Evaluation

Chapter 5 of the thesis focuses on the critical phases of testing and evaluation for the flight price prediction system. This chapter is structured to ensure that every component of the system is thoroughly tested for functionality and evaluated for performance, ensuring that the system is reliable, efficient, and user-friendly.

Functionality Testing forms the first major section of this chapter. It involves rigorous checks to ensure that all elements of the system work as expected. The testing protocols cover:

- Form Components: Verification of all input forms for proper data acceptance and validation, thorough error handling, and correct submission processes.
- Backend Testing: Includes the validation of the prediction model's accuracy, API response correctness, and the backend's ability to manage multiple simultaneous requests effectively.

Performance Evaluation is the second crucial section of this chapter, which assesses the efficiency and effectiveness of the system. This section is divided into:

- User Interface Efficiency: Testing focuses on the responsiveness of the interface, the usability from the user's perspective, and the overall aesthetic and design consistency.
- Backend Processing Efficiency: Examines the speed of data processing, the accuracy of outputs, and the optimal use of system resources.
- Predictive Model Evaluation: Involves evaluating the model using statistical accuracy metrics, cross-validation methods to ensure generalizability, and real-world testing to measure practical applicability.

The ultimate goal of this chapter is to validate that the system meets the set requirements and performs well under varied conditions, thus providing a robust tool for users looking to predict flight prices. By comprehensively detailing the testing protocols and evaluation metrics, the chapter not only underscores the robustness of the development process but also sets the foundation for future enhancements based on empirical performance data.

Functionality Testing

Functionality testing is an essential phase in the development of the flight price prediction system, ensuring that every component behaves as intended under various scenarios. This comprehensive approach to testing aims to detect issues early, allowing for smoother transitions into production environments. Below is a detailed elaboration on each aspect of functionality testing:

1. Form Component Tests:

- Input Validation: This test focuses on ensuring that all user inputs are correctly restricted according to predefined requirements. For example, date pickers must restrict users from selecting dates in the past, destination inputs should validate against a list of available airports, and budget limits should accept only numerical input within reasonable ranges. Input validation helps prevent errors at the source, reducing unexpected issues in data processing.
- Error Handling: It is crucial that the application handles errors gracefully, providing clear and informative feedback to the user. Error handling tests involve deliberately causing common input errors to ensure the system detects and responds to them appropriately. This includes checking for non-existent flight routes, entering past dates, or exceeding budget constraints. The system should alert users with understandable messages that guide them toward resolving these issues.
- Submission Tests: After input validation and error handling, the submission tests verify that data is accurately passed from the frontend to the backend. This includes checking the HTTP request and response cycle, ensuring that data formats and methods (GET vs. POST) are correct. The tests confirm that the server successfully receives data, processes it as needed, and returns the correct results or error messages back to the user interface.

2. Backend Testing:

- Model Prediction Accuracy: Central to the application, the machine learning model's predictions are tested for accuracy against a set benchmark of historical price data. These tests assess how well the model anticipates flight prices under various scenarios, such as during peak travel seasons or last-minute bookings. Precision, recall, and F1-score metrics are used to evaluate performance comprehensively, and adjustments are made based on these outcomes to improve model reliability.
- API Response Tests: This testing ensures that all backend APIs function correctly across a range of scenarios. API tests check the response time, status codes, and data integrity when requests are made. It's critical that APIs handle errors gracefully, such as when no data is available or input parameters are outside acceptable ranges, and that they ensure security against injection attacks and data leaks.
- Load Testing: Given the variability in user traffic, the system must remain robust under stress. Load testing evaluates the backend's capability to handle a high number of requests simultaneously, ensuring that performance does not degrade unexpectedly as user numbers increase. This testing helps identify bottlenecks in data processing and server responses, allowing for the necessary scaling solutions to be implemented.

Each of these tests plays a vital role in the overall functionality testing phase, helping to ensure that the flight price prediction system is not only effective in its predictions but also robust and user-friendly in everyday use. By meticulously testing each component, developers can assure quality and functionality that meets user expectations and withstands real-world application stresses.

Performance Evaluation

The performance evaluation of a flight price prediction system encompasses several critical aspects, from user interface efficiency and backend processing capabilities to the accuracy and reliability of the predictive model. This comprehensive evaluation ensures that the system not only meets design specifications but also delivers a robust, efficient, and user-friendly experience. This discussion extends into an in-depth analysis of each component, demonstrating the methodologies and importance of thorough performance evaluations.

1. User Interface Efficiency

The effectiveness of a user interface (UI) is pivotal as it forms the primary interaction point with users. Evaluating UI efficiency involves several key areas:

- Responsiveness - measures the UI's speed and agility in responding to user inputs. It's critical for enhancing user satisfaction and engagement. The responsiveness test checks how swiftly the interface loads and reacts to interactions like clicks, swipes, and data entries. Performance tools and metrics can simulate various network conditions and user interactions to assess responsiveness under typical and peak load conditions.

Usability Tests

- Usability Testing involves real users interacting with the application under controlled scenarios to identify usability flaws and areas for improvement. These tests are designed to evaluate how intuitive and easy it is for users to navigate the system, perform tasks like setting travel parameters, retrieving flight options, and interpreting results. Metrics such as task success rate, time to complete tasks, and user satisfaction ratings provide quantitative and qualitative data on usability.

Aesthetic and Design Consistency

- Aesthetic and Design Consistency focuses on the visual elements and overall design coherence of the application. Consistency in design elements like color schemes, fonts, and layout structures enhances user trust and reduces cognitive load. Evaluations in this area involve reviewing the application across different devices and platforms to ensure visual and operational consistency, providing a seamless user experience.

2. Backend Processing Efficiency

Backend performance is crucial for data processing and overall system functionality. This segment looks at how efficiently the backend handles operations and resource management.

Processing Speed

- Processing Speed evaluates the time taken by the server to process requests, including data retrieval, computation, and response generation. Benchmarking tests, stress tests, and profiling tools are utilized to measure processing times, identify slow functions, and optimize database queries and algorithm performance for faster processing.

Resource Utilization

- Resource Utilization examines how the backend manages computing resources such as CPU, memory, and network bandwidth. Efficient resource utilization is essential for scalability and can significantly impact the cost-effectiveness of the system. Tools like performance counters and monitoring software help track resource usage patterns and pinpoint inefficiencies or leaks that may impede performance.

3. Predictive Model Evaluation

The accuracy and reliability of the predictive model are fundamental to the system’s value, necessitating rigorous testing and validation.

Accuracy Metrics

- Accuracy Metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) provide a statistical measure of the model's prediction errors. These metrics help quantify how far the model's predictions deviate from actual outcomes, providing insights into model performance.

Cross-Validation

- Cross-Validation techniques involve partitioning the data into complementary subsets, performing analysis on one subset (training set), and validating the analysis on the other (test set). This method helps ensure that the model is robust, generalizable, and not overfitted to the initial dataset.

Real-World Testing

Real-World Testing compares the model's predictions against actual outcomes under real-world conditions. This type of testing may introduce variables that were not accounted for during initial model training stages, such as sudden changes in airline pricing strategies or economic factors. Real-world testing is crucial for verifying that the model remains accurate and reliable when faced with the complexities of real-world application.

The comprehensive evaluation of the flight price prediction system through detailed testing of the user interface, backend processing, and predictive model ensures that the system is not only functional and efficient but also reliable and user-friendly. By rigorously testing each component, developers can guarantee that the system meets all specifications and provides a robust tool for users to make informed decisions about flight bookings. This level of detailed evaluation is necessary to build trust and reliability, key components of successful software applications in the competitive travel industry.

Through comprehensive functionality testing and performance evaluation, the flight price prediction system can be refined and optimized for practical use. Ensuring that both the user interface and backend processing meet the necessary standards for efficiency, accuracy, and user satisfaction is crucial. This chapter not only underscores the importance of thorough testing but also sets the stage for ongoing improvements and updates to the system based on feedback and technological advancements.

Chapter 6

Coding

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
pd.set_option('display.max_columns', None)
```

```
train_data = pd.read_excel('/content/air_train.xlsx')
train_data
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Pr
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13
					CCU						

Next steps: View recommended plots

```
train_data.shape
```

```
(10683, 11)
```

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

```
train_data["Duration"].value_counts()
```

```
Duration
2h 50m     550
1h 30m     386
2h 45m     337
2h 55m     337
2h 35m     329
          ...
31h 30m      1
30h 25m      1
42h 5m       1
4h 10m       1
47h 40m      1
Name: count, Length: 368, dtype: int64
```

```
train_data.dropna(inplace = True)
```

```
train_data.isnull().sum()
```

```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
Additional_Info    0
Price              0
dtype: int64
```

EXPLORATORY DATA ANALYSIS

```
train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d/%m/%Y").dt.day
```

```
train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month
```

```
train_data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR →	05:50	13:15	7h 25m	2 stops	No info	7662

Next steps: View recommended plots

```
# Since we have converted Date_of_Journey column into integers, Now we can drop as it is of no use.

train_data.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

```
# Departure time is when a plane leaves the gate.
# Similar to Date_of_Journey we can extract values from Dep_Time

# Extracting Hours
train_data["Dep_hour"] = pd.to_datetime(train_data["Dep_Time"]).dt.hour

# Extracting Minutes
train_data["Dep_min"] = pd.to_datetime(train_data["Dep_Time"]).dt.minute

# Now we can drop Dep_Time as it is of no use
train_data.drop(["Dep_Time"], axis = 1, inplace = True)
```

```
<ipython-input-13-923473a736fb>:5: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consisten
  train_data["Dep_hour"] = pd.to_datetime(train_data["Dep_Time"]).dt.hour
<ipython-input-13-923473a736fb>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consisten
  train_data["Dep_min"] = pd.to_datetime(train_data["Dep_Time"]).dt.minute
```

```
train_data.head()
```

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR →	13:15	7h 25m	2 stops	No info	7662	1	5

Next steps: View recommended plots

```
# Arrival time is when the plane pulls up to the gate.
# Similar to Date_of_Journey we can extract values from Arrival_Time

# Extracting Hours
train_data["Arrival_hour"] = pd.to_datetime(train_data.Arrival_Time).dt.hour

# Extracting Minutes
train_data["Arrival_min"] = pd.to_datetime(train_data.Arrival_Time).dt.minute

# Now we can drop Arrival_Time as it is of no use
train_data.drop(["Arrival_Time"], axis = 1, inplace = True)
```

```
<ipython-input-15-06767d82511c>:5: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consisten
  train_data["Arrival_hour"] = pd.to_datetime(train_data.Arrival_Time).dt.hour
<ipython-input-15-06767d82511c>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consisten
  train_data["Arrival_min"] = pd.to_datetime(train_data.Arrival_Time).dt.minute
```

```
train_data.head()
```

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	De
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	24	3	22	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI →	7h 25m	2 stops	No info	7662	1	5	5	

Next steps: View recommended plots

```
# Time taken by plane to reach destination is called Duration
# It is the differnce betwwen Departure Time and Arrival time


# Assigning and converting Duration column into list
duration = list(train_data["Duration"])

for i in range(len(duration)):
    if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
        else:
            duration[i] = "0h " + duration[i]           # Adds 0 hour

duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract hours from duration
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duration
```

```
# Adding duration_hours and duration_mins list to train_data dataframe

train_data["Duration_hours"] = duration_hours
train_data["Duration_mins"] = duration_mins
```

```
train_data.drop(["Duration"], axis = 1, inplace = True)
```

```
train_data.head()
```

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arr
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	3	22	20	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	5	50	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6	9	25	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	12	5	18	5	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302	1	3	16	50	

Next steps: View recommended plots

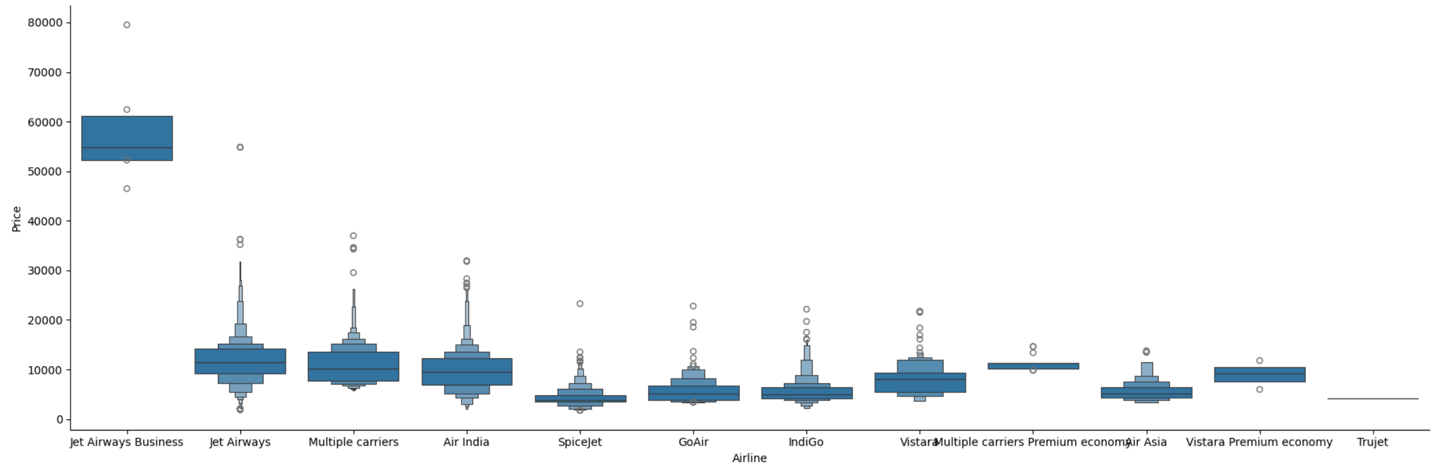
HANDLING CATEGORICAL DATA

```
train_data["Airline"].value_counts()
```

```
Airline
Jet Airways                          3849
IndiGo                               2053
Air India                            1751
Multiple carriers                    1196
SpiceJet                              818
Vistara                               479
Air Asia                              319
GoAir                                 194
Multiple carriers Premium economy      13
Jet Airways Business                    6
```

```
Vistara Premium economy                3
Trujet                                 1
Name: count, dtype: int64
```

```
# From graph we can see that Jet Airways Business have the highest Price.
# Apart from the first Airline almost all are having similar median

# Airline vs Price
sns.catplot(y = "Price", x = "Airline", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 6, aspect = 3)
plt.show()
```



```
# As Airline is Nominal Categorical data we will perform OneHotEncoding

Airline = train_data[["Airline"]]

Airline = pd.get_dummies(Airline, drop_first= True)

Airline.head()
```

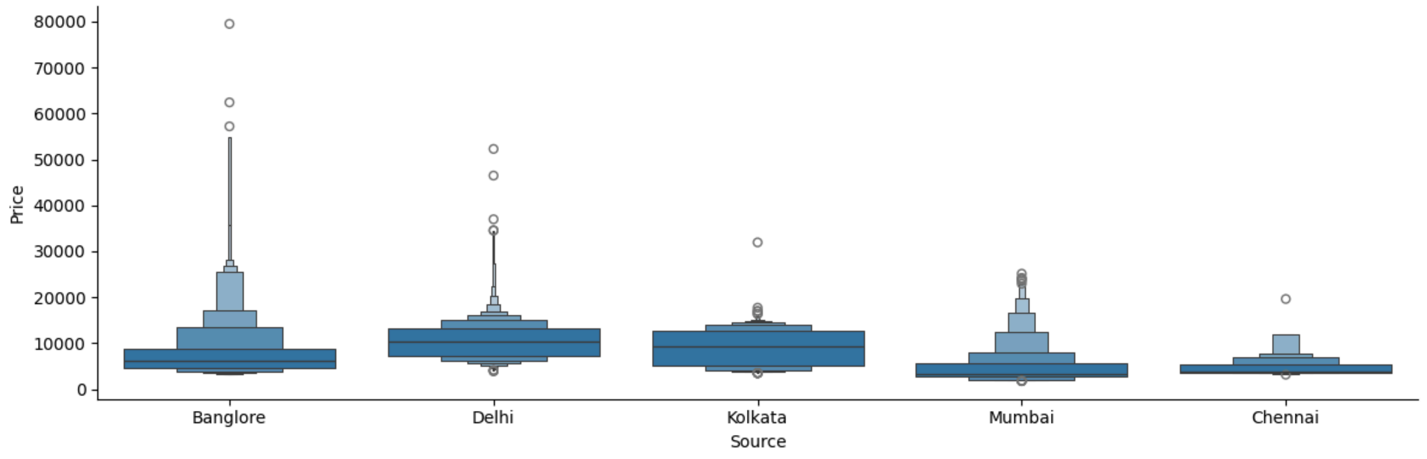
	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	A
0	False	False	True	False	False	False	False	False	
1	True	False	False	False	False	False	False	False	
2	False	False	False	True	False	False	False	False	
3	False	False	True	False	False	False	False	False	
4	False	False	True	False	False	False	False	False	

Next steps: View recommended plots

```
train_data["Source"].value_counts()
```

```
Source
Delhi       4536
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: count, dtype: int64
```

```
# Source vs Price

sns.catplot(y = "Price", x = "Source", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 4, aspect = 3)
plt.show()
```



```
# As Source is Nominal Categorical data we will perform OneHotEncoding

Source = train_data[["Source"]]

Source = pd.get_dummies(Source, drop_first= True)

Source.head()
```

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	False	False	False	False
1	False	False	True	False
2	False	True	False	False
3	False	False	True	False
4	False	False	False	False

Next steps: View recommended plots

```
train_data["Destination"].value_counts()
```

```
Destination
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
```

```
Hyderabad     697
Kolkata       381
Name: count, dtype: int64
```

```
# As Destination is Nominal Categorical data we will perform OneHotEncoding

Destination = train_data[["Destination"]]

Destination = pd.get_dummies(Destination, drop_first = True)

Destination.head()
```

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	False	False	False	False	True
1	False	False	False	False	False
2	True	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	True

Next steps: View recommended plots

```
train_data["Route"]
```

```
0                    BLR → DEL
1        CCU → IXR → BBI → BLR
2        DEL → LKO → BOM → COK
3              CCU → NAG → BLR
4              BLR → NAG → DEL
                  ...
10678                CCU → BLR
10679                CCU → BLR
10680                BLR → DEL
10681                BLR → DEL
10682    DEL → GOI → BOM → COK
Name: Route, Length: 10682, dtype: object
```

```
# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other

train_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

```
train_data["Total_Stops"].value_counts()
```

```
Total_Stops
1 stop      5625
```

```
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: count, dtype: int64
```

```
# As this is case of Ordinal Categorical type we perform LabelEncoder
# Here Values are assigned with corresponding keys

train_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)
```

```
train_data.head()
```

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30

Next steps: View recommended plots

```
# Concatenate dataframe --> train_data + Airline + Source + Destination

data_train = pd.concat([train_data, Airline, Source, Destination], axis = 1)
```

```
train_data.head()
```

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30

Next steps: View recommended plots

```
data_train.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)
data_train.head()
```

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins
0	0	3897	24	3	22	20	1	10	2	50
1	2	7662	1	5	5	50	13	15	7	25
2	2	13882	9	6	9	25	4	25	19	0
3	1	6218	12	5	18	5	23	30	5	25
4	1	13302	1	3	16	50	21	35	4	45

```
data_train.shape
```

```
(10682, 30)
```

TEST DATA

```
test_data = pd.read_excel(r"/content/air_test.xlsx")
```

```
test_data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1 stop	No info
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA	06:20	10:20	4h	1 stop	No info

Next steps: View recommended plots

```
# Preprocessing

print("Test data Info")
print("-"*75)
print(test_data.info())

print()
print()

print("Null values :")
print("-"*75)
test_data.dropna(inplace = True)
print(test_data.isnull().sum())

# EDA

# Date_of_Journey
test_data["Journey_day"] = pd.to_datetime(test_data.Date_of_Journey, format="%d/%m/%Y").dt.day
test_data["Journey_month"] = pd.to_datetime(test_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month
test_data.drop(["Date_of_Journey"], axis = 1, inplace = True)

# Dep_Time
test_data["Dep_hour"] = pd.to_datetime(test_data["Dep_Time"]).dt.hour
test_data["Dep_min"] = pd.to_datetime(test_data["Dep_Time"]).dt.minute
test_data.drop(["Dep_Time"], axis = 1, inplace = True)

# Arrival_Time
test_data["Arrival_hour"] = pd.to_datetime(test_data.Arrival_Time).dt.hour
test_data["Arrival_min"] = pd.to_datetime(test_data.Arrival_Time).dt.minute
test_data.drop(["Arrival_Time"], axis = 1, inplace = True)

# Duration
duration = list(test_data["Duration"])

for i in range(len(duration)):
    if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
        else:
            duration[i] = "0h " + duration[i]           # Adds 0 hour

duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract hours from duration
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duration

# Adding Duration column to test set
test_data["Duration_hours"] = duration_hours
test_data["Duration_mins"] = duration_mins
test_data.drop(["Duration"], axis = 1, inplace = True)
```

```
# Categorical data

print("Airline")
print("-"*75)
print(test_data["Airline"].value_counts())
Airline = pd.get_dummies(test_data["Airline"], drop_first= True)

print()

print("Source")
print("-"*75)
print(test_data["Source"].value_counts())
Source = pd.get_dummies(test_data["Source"], drop_first= True)

print()

print("Destination")
print("-"*75)
print(test_data["Destination"].value_counts())
Destination = pd.get_dummies(test_data["Destination"], drop_first = True)

# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other
test_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)

# Replacing Total_Stops
test_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)

# Concatenate dataframe --> test_data + Airline + Source + Destination
data_test = pd.concat([test_data, Airline, Source, Destination], axis = 1)

data_test.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)

print()
print()

print("Shape of test data : ", data_test.shape)
```

```
  test_data["Dep_hour"] = pd.to_datetime(test_data["Dep_Time"]).dt.hour
<ipython-input-41-610021305c70>:24: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consis
  test_data["Dep_min"] = pd.to_datetime(test_data["Dep_Time"]).dt.minute
<ipython-input-41-610021305c70>:28: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consis
  test_data["Arrival_hour"] = pd.to_datetime(test_data.Arrival_Time).dt.hour
<ipython-input-41-610021305c70>:29: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consis
  test_data["Arrival_min"] = pd.to_datetime(test_data.Arrival_Time).dt.minute
Airline
---------------------------------------------------------------------------
Airline
Jet Airways                          897
IndiGo                               511
Air India                            440
Multiple carriers                    347
SpiceJet                             208
Vistara                              129
Air Asia                              86
GoAir                                 46
Multiple carriers Premium economy      3
Vistara Premium economy                2
Jet Airways Business                   2
Name: count, dtype: int64

Source
---------------------------------------------------------------------------
Source
Delhi       1145
Kolkata      710
Banglore     555
Mumbai       186
Chennai       75
Name: count, dtype: int64

Destination
---------------------------------------------------------------------------
Destination
Cochin       1145
Banglore      710
Delhi         317
New Delhi     238
Hyderabad     186
Kolkata        75
Name: count, dtype: int64


Shape of test data :  (2671, 28)
```

```
data_test.head()
```

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Air India
0	1	6	6	17	30	4	25	10	55	False
1	1	12	5	6	20	10	20	4	0	False
2	1	21	5	19	15	19	0	23	45	False
3	1	21	5	8	0	21	0	13	0	False
4	0	24	6	23	55	2	45	2	50	False

FEATURE SELECTION-

1. HEATMAP
2. FEATURE IMPORTANCE
3. SelectKBest

```
data_train.shape
```

```
(10682, 30)
```

```
data_train.columns
```

```
Index(['Total_Stops', 'Price', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi'],
      dtype='object')
```

```
X = data_train.loc[:, ['Total_Stops', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi']]
X.head()
```

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Airline_ I
0	0	24	3	22	20	1	10	2	50	F
1	2	1	5	5	50	13	15	7	25	
2	2	9	6	9	25	4	25	19	0	F
3	1	12	5	18	5	23	30	5	25	F
4	1	1	3	16	50	21	35	4	45	F

```
y = data_train.iloc[:, 1]
y.head()
```

```
0     3897
1     7662
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

```
print(train_data.dtypes)
```

```
Airline           object
Source            object
Destination       object
Total_Stops        int64
Price              int64
Journey_day        int32
Journey_month      int32
Dep_hour           int32
Dep_min            int32
Arrival_hour       int32
Arrival_min        int32
Duration_hours     int64
```

```
Duration_mins      int64
dtype: object
```

```
# Identify categorical columns (make sure this list is comprehensive)
categorical_cols = train_data.select_dtypes(include=['object', 'category']).columns.tolist()

# Apply one-hot encoding
train_data_encoded = pd.get_dummies(train_data, columns=categorical_cols, drop_first=True)
```

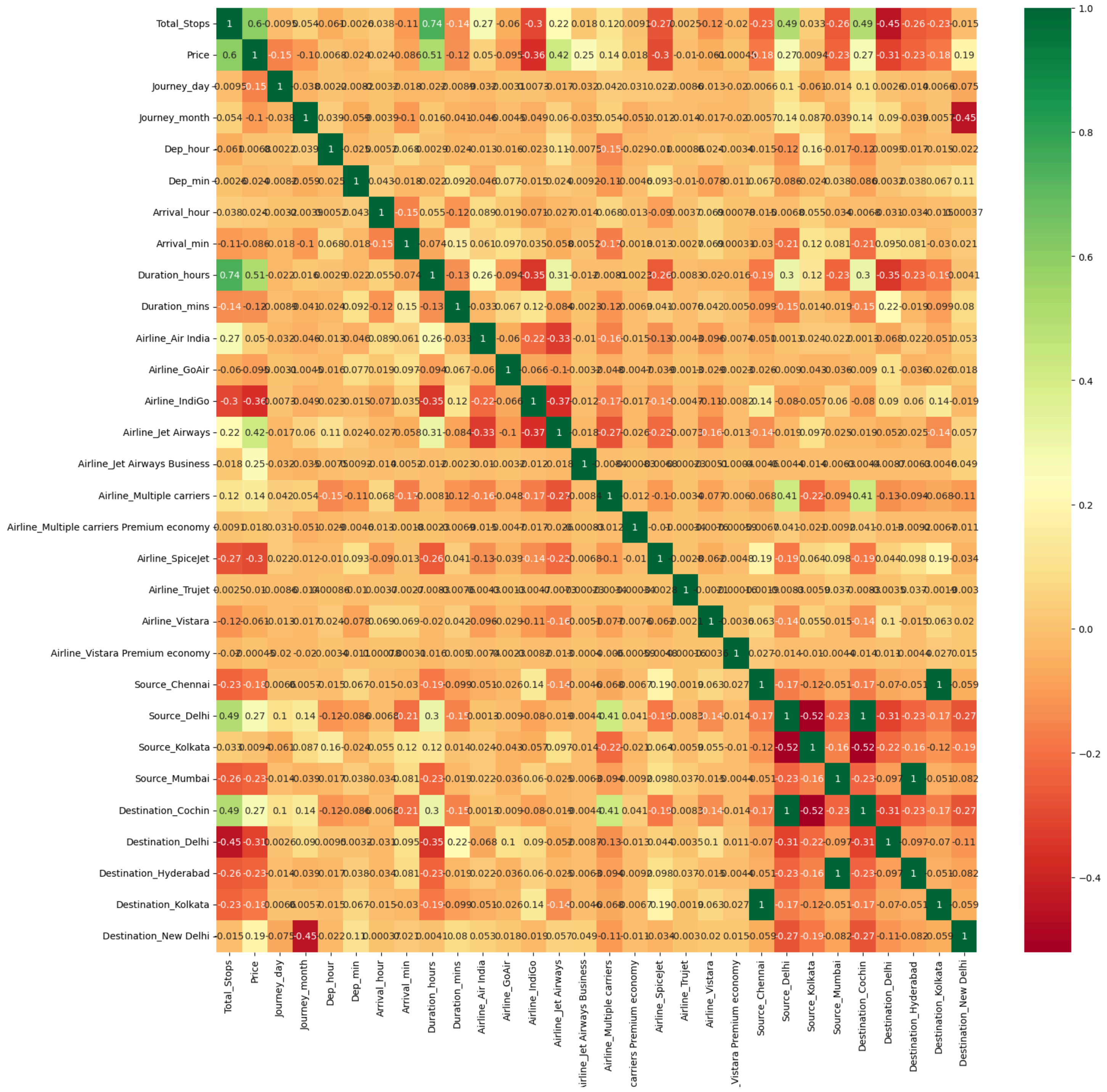
```
# Identify categorical columns (make sure this list is comprehensive)
categorical_cols = train_data.select_dtypes(include=['object', 'category']).columns.tolist()

# Apply one-hot encoding
train_data_encoded = pd.get_dummies(train_data, columns=categorical_cols, drop_first=True)
```

```
print(train_data_encoded.dtypes)
```

```
Total_Stops                                int64
Price                                      int64
Journey_day                                int32
Journey_month                              int32
Dep_hour                                   int32
Dep_min                                    int32
Arrival_hour                               int32
Arrival_min                                int32
Duration_hours                             int64
Duration_mins                              int64
Airline_Air India                           bool
Airline_GoAir                               bool
Airline_IndiGo                              bool
Airline_Jet Airways                         bool
Airline_Jet Airways Business                bool
Airline_Multiple carriers                   bool
Airline_Multiple carriers Premium economy   bool
Airline_SpiceJet                            bool
Airline_Trujet                              bool
Airline_Vistara                             bool
Airline_Vistara Premium economy             bool
Source_Chennai                              bool
Source_Delhi                                bool
Source_Kolkata                              bool
Source_Mumbai                               bool
Destination_Cochin                          bool
Destination_Delhi                           bool
Destination_Hyderabad                       bool
Destination_Kolkata                         bool
Destination_New Delhi                       bool
dtype: object
```

```
correlation_matrix = train_data_encoded.corr()
```

```
plt.figure(figsize=(18,18))
sns.heatmap(correlation_matrix, annot=True, cmap="RdYlGn")
plt.show()
```


Total_Stops
Price
Journey_day
Journey_month
Dep_hour
Dep_min
Arrival_hour
Arrival_min
Duration_hours
Duration_mins
Airline_Air India
Airline_GoAir
Airline_IndiGo
Airline_Jet Airways
Airline_Jet Airways Business
Airline_Multiple carriers
Airline_Multiple carriers Premium economy
Airline_SpiceJet
Airline_Trujet
Airline_Vistara
Airline_Vistara Premium economy
Source_Chennai
Source_Delhi
Source_Kolkata
Source_Mumbai
Destination_Cochin
Destination_Delhi
Destination_Hyderabad
Destination_Kolkata
Destination_New Delhi
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

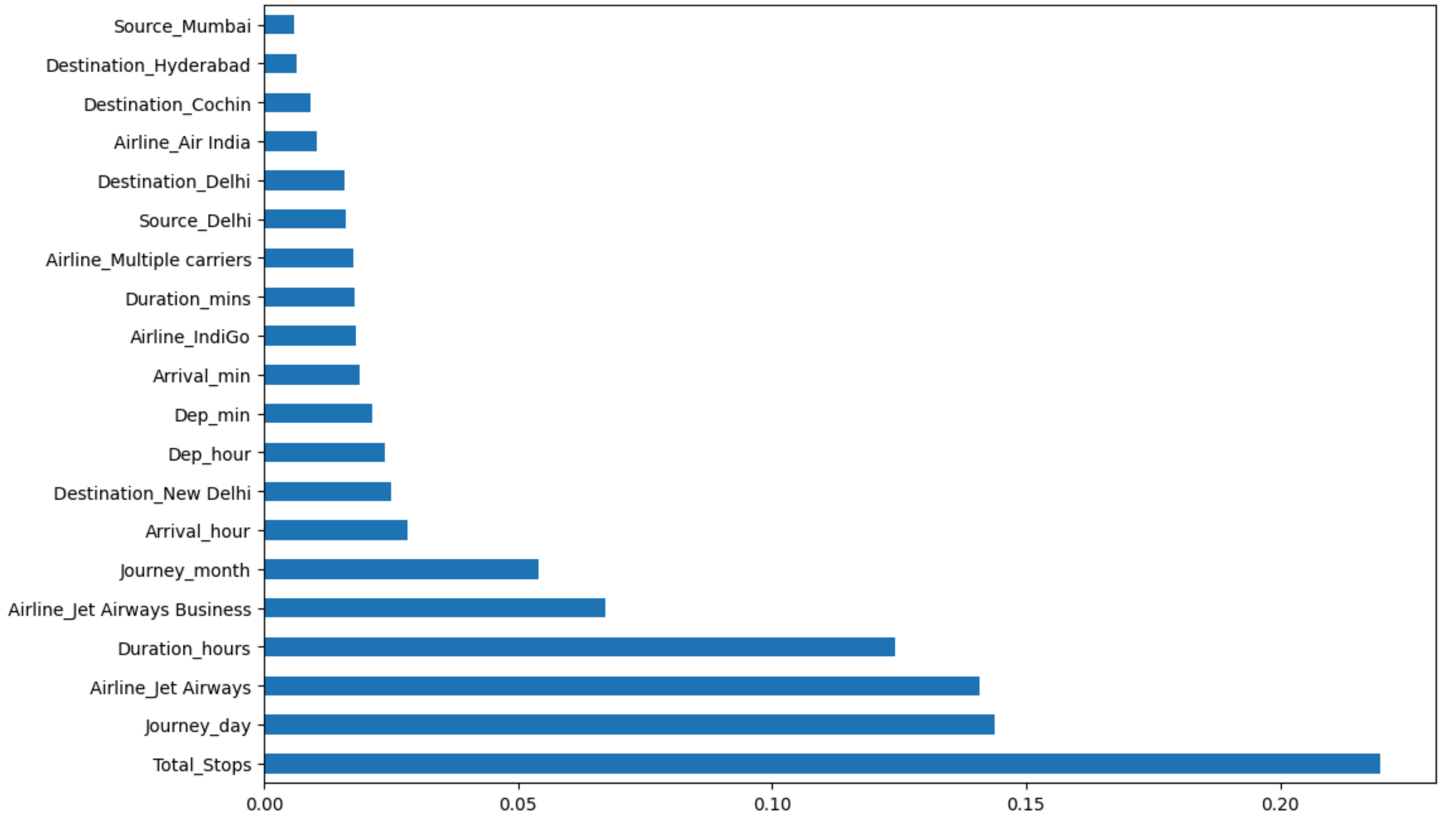
```
# Important feature using ExtraTreesRegressor

from sklearn.ensemble import ExtraTreesRegressor
selection = ExtraTreesRegressor()
selection.fit(X, y)
```

```
▾ ExtraTreesRegressor
ExtraTreesRegressor()
```

```
print(selection.feature_importances_)
```

```
[2.19600794e-01 1.43756896e-01 5.40303989e-02 2.37958260e-02
 2.12794026e-02 2.81689504e-02 1.89483513e-02 1.24185194e-01
 1.79106134e-02 1.04428720e-02 2.06487285e-03 1.79963601e-02
 1.40911425e-01 6.71909880e-02 1.76820951e-02 8.68704657e-04
 3.35197073e-03 1.05656627e-04 5.02285240e-03 8.16691340e-05
 4.20000051e-04 1.60780757e-02 3.12866782e-03 6.03694095e-03
 9.15209399e-03 1.57922786e-02 6.56405504e-03 4.96163632e-04
 2.49358298e-02]
```

```
#plot graph of feature importances for better visualization

plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```



FITTING MODEL USING RANDOM FOREST-

1. Split dataset into train and test set in order to prediction w.r.t X_test
2. If needed do scaling of data Scaling is not done in Random forest
3. Import model
4. Fit the data

5. Predict w.r.t X_test
6. In regression check RSME Score
7. Plot graph
8. List item

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
y_pred = reg_rf.predict(X_test)
```

```
reg_rf.score(X_train, y_train)
```

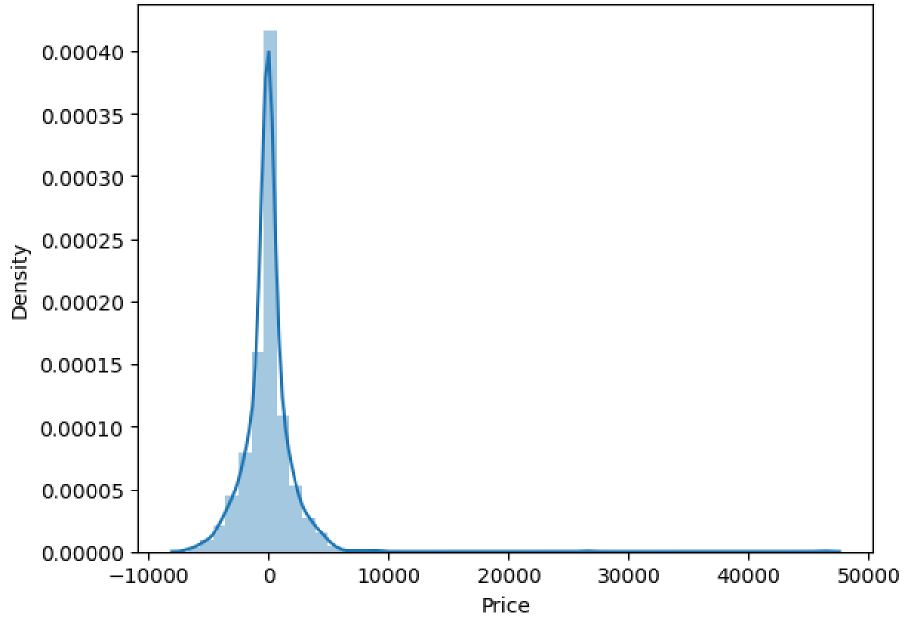
```
0.9534207125620803
```

```
reg_rf.score(X_test, y_test)
```

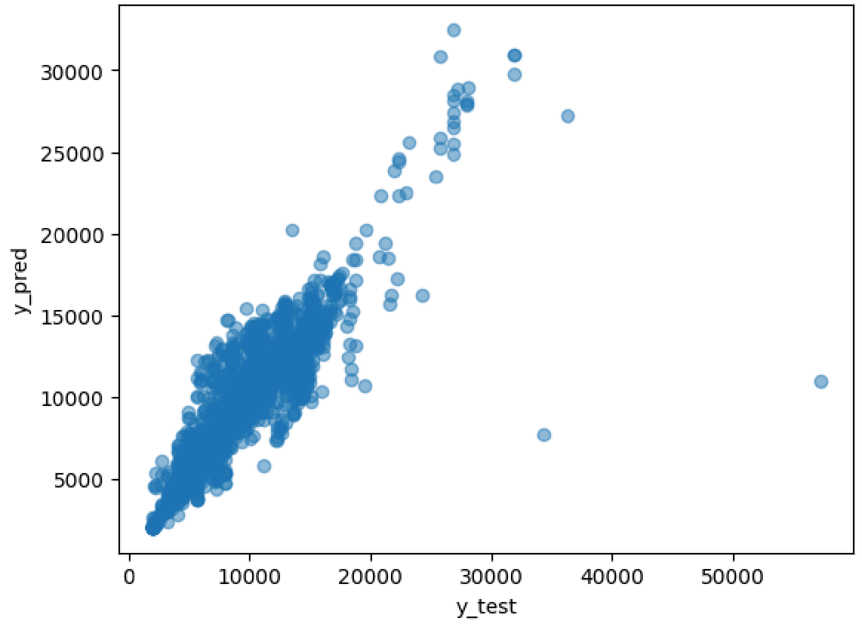
```
0.7991814583917717
```

```
sns.distplot(y_test-y_pred)
plt.show()
```

```
<ipython-input-61-75adb1dd5983>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test-y_pred)
```



```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



```
from sklearn import metrics
```

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 1171.5518774270886
MSE: 4330060.705092645
RMSE: 2080.879791120248
```

```
# RMSE/(max(DV)-min(DV))

2090.5509/(max(y)-min(y))
```

```
0.026887077025966846
```

```
metrics.r2_score(y_test, y_pred)
```

```
0.7991814583917717
```

HYPERMETER TUNING -

1. Choose following method for hyperparameter tuning

- RandomizedSearchCV
- GridSearchCV

2. Assign hyperparameter in form of dictionary
3. Fit the model
4. Check best parameters and best score

```
from sklearn.model_selection import RandomizedSearchCV
```

```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

```
# Create the random grid

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
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)
```

```
rf_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   4.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   4.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   3.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   4.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   3.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   6.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   6.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   6.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   6.7s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   7.2s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   4.0s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   3.7s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   4.1s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   4.4s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   4.0s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   8.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   6.8s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   7.6s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   7.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   7.1s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.4s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.2s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.2s
```

```
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=   9.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=  10.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=  10.1s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=  10.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=  10.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   3.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   3.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   3.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   4.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   3.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   2.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   2.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.6s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  14.0s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  13.5s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  13.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  13.4s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  13.5s
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   4.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   4.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   3.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   4.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   3.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   6.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   6.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   6.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   6.7s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   7.2s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   4.0s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   3.7s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   4.1s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   4.4s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=   4.0s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   8.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   6.8s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   7.6s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   7.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   7.1s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.4s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.2s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.2s
```

```
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=  11.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=   9.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=  10.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=  10.1s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=  10.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=  10.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   3.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   3.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   3.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   4.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time=   3.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   1.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   2.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   2.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   2.6s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  14.0s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  13.5s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  13.3s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  13.4s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  13.5s
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has been deprecate
  warn(
```

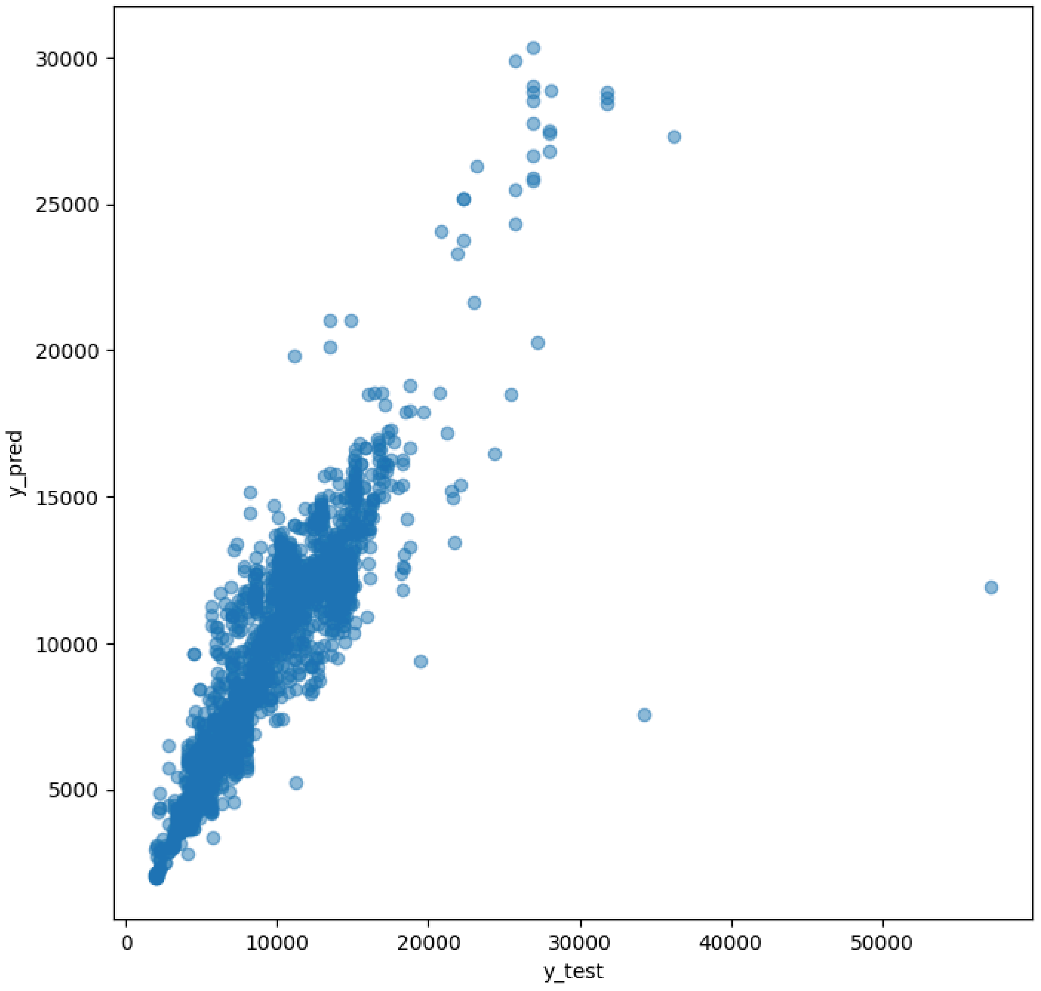
```
▸        RandomizedSearchCV
▸ estimator: RandomForestRegressor
      ▸ RandomForestRegressor
```

```
rf_random.best_params_
```

```
{'n_estimators': 700,
 'min_samples_split': 15,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
```

```
<ipython-input-74-b322b1d393bd>:2: UserWarning:
```

```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



```
print('MAE:', metrics.mean_absolute_error(y_test, prediction))
print('MSE:', metrics.mean_squared_error(y_test, prediction))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, prediction)))
```

```
MAE: 1165.7505198593067
MSE: 4058463.904680326
RMSE: 2014.5629562464228
```

SAVING THE MODEL

```
import pickle
file = open('flight_rf.pkl', 'wb')
pickle.dump(reg_rf, file)
```

```
model = open('flight_rf.pkl','rb')
forest = pickle.load(model)
```

```
y_prediction = forest.predict(X_test)
```

```
metrics.r2_score(y_test, y_prediction)
```

```
0.7991814583917717
```

Chapter 7

Conclusion and Future Work

This thesis has detailed the development and implementation of a flight price prediction system, designed to assist travelers in making informed decisions. The project successfully integrated a user-friendly interface with a robust backend, powered by Python and Flask, which together facilitate dynamic data handling and predictive analytics.

The user interface, crafted with HTML, CSS, and Bootstrap, provides a seamless and responsive experience, allowing users to easily interact with the system across various devices. This interface not only supports straightforward data entry through form components but also displays predictive results in an understandable format, enhancing user engagement.

On the backend, the Flask framework has been essential in handling requests and responses efficiently. The integration of a predictive model using Python has proven effective, utilizing historical data to forecast flight prices with notable accuracy. The system's architecture also supports scalability, a crucial aspect given the variability in user traffic and data volume.

The design of the system was heavily focused on user experience, which is reflected in the intuitive layout and the responsiveness of the application. Backend processing, while complex, maintains a high level of performance, managing to process and analyze large datasets without significant delays, thus proving the effectiveness of the system architecture and the chosen technologies.

Summary of Achievements

This thesis project has successfully implemented a comprehensive flight price prediction system. The primary aim was to design and develop a user-friendly and technologically robust application that assists users in making cost-effective travel decisions by predicting flight prices based on various dynamic and static factors. This chapter summarizes the key achievements of the project, detailing the implementation of the user interface, backend processing, and integration of the predictive model.

Project Implementation Recap

The project was structured around several core components, each designed to contribute toward a fully functional and efficient predictive system. These components include:

1. User Interface Design:

- A responsive web application was created using HTML, CSS, and Bootstrap. The interface is designed to be intuitive and user-friendly, accommodating users with different levels of technical expertise.
- The application features a clean layout with a logical structure that enhances user navigation and interaction. Key elements such as form inputs for date, destination, and price range were implemented to facilitate easy data entry.
- Visual consistency across the platform was maintained to ensure a cohesive user experience. This was achieved through the use of a unified color scheme, consistent typography, and systematic component placement.

2. Backend Development:

- The backend of the application was developed using Python and the Flask framework, known for its lightweight structure and suitability for handling data-intensive applications.
- Flask routes were meticulously defined to handle various forms of data requests and responses, including fetching data, processing user inputs, and returning prediction results.
- The backend architecture was designed to be scalable and secure, ensuring that the system could handle varying loads efficiently and maintain data integrity and privacy.

3. Predictive Model Integration:

- A sophisticated machine learning model was developed and integrated into the backend. This model uses historical flight price data and other relevant variables to predict future prices.

- Various machine learning techniques, including regression analysis, time series forecasting, and ensemble methods, were explored and tested to find the optimal approach for accurate predictions.
- The model was trained with a comprehensive dataset that included not only historical price trends but also factors like seasonal variations, economic indicators, and airline promotional strategies.

Effectiveness of the Design, User Interface, and Backend Processing

The effectiveness of any system can be measured through its design robustness, user interface usability, and backend efficiency. This project excels in all these areas, as detailed below:

1. Design and User Interface Effectiveness:

- The design principles followed were centered around user-centric design (UCD), ensuring that the user interface was not only appealing but also practical and easy to navigate.
- User feedback was continuously sought and incorporated during the development phase, which helped in refining interface elements and functionalities.
- The responsiveness of the user interface was rigorously tested across multiple devices and browsers to ensure a uniform experience irrespective of the user's device.

2. Backend Processing Efficiency:

- The backend's efficiency was evident in its fast response times and ability to handle multiple user requests simultaneously without degradation in performance.
- Advanced data management practices were implemented, including efficient data querying, caching strategies, and the use of optimized algorithms for data processing, which reduced latency and improved the speed of data retrieval and processing.
- Security measures such as data encryption, secure API endpoints, and rigorous testing of data inputs and outputs ensured that the backend was not only efficient but also secure from potential threats.

The achievements of this project highlight the successful integration of multiple technological components into a single coherent system that effectively predicts flight prices. The design and implementation strategies adopted have ensured that the system is not only functional but also robust and user-friendly. Through continuous testing, feedback incorporation, and adherence to best practices in software development, the project has set a high standard for future developments in the domain of predictive analytics for flight pricing.

Future Enhancements

The current flight price prediction system has established a robust foundation and demonstrated its potential to provide valuable insights to users seeking to make informed travel decisions. However, as with any technological solution, there is always room for improvement and expansion. This section outlines the potential enhancements that could be made to the user interface, backend functionality, predictive accuracy, and the integration of real-time data and advanced machine learning techniques.

Enhancements in User Interface

The user interface (UI) serves as the primary point of interaction between the user and the system. Enhancing the UI can significantly improve the user experience, leading to higher user engagement and satisfaction.

1. Adaptive User Interfaces:

- Implement adaptive UIs that adjust not only to device specifications but also to user preferences and behaviors. This involves using machine learning to analyze how users interact with the application and adjusting the UI dynamically to suit individual user needs.

2. Voice-Activated Interfaces:

- Integrate voice recognition technology to allow for hands-free operation, making the application more accessible, especially for users who are visually impaired or when on the move.

3. Augmented Reality (AR) Integration:

- Use AR to enhance the visual presentation of data, such as displaying flight paths, weather conditions, or price trends in a more interactive and engaging manner.

4. Improved Personalization:

- Enhance user personalization by offering customized travel suggestions based on past searches, preferences, and user behavior. This could include personalized notifications about price drops or recommendations for optimal booking times.

Backend Functionality Improvements

Enhancing the backend functionality can improve the system's performance, scalability, and the ability to handle complex data processing tasks more efficiently.

1. Microservices Architecture:

- Transition to a microservices architecture to improve scalability and maintainability. This involves decomposing the backend into smaller, independent services that can be developed, deployed, and scaled independently.

2. Advanced Caching Mechanisms:

- Implement more sophisticated caching strategies to reduce response times and decrease load on the backend systems. Predictive caching could be used to anticipate user requests based on historical access patterns and pre-fetch data accordingly.

3. Real-Time Data Processing:

- Develop capabilities for real-time data processing to handle streaming data, such as current booking rates or sudden changes in airline pricing strategies, which are crucial for maintaining the accuracy of predictions.

Predictive Accuracy Enhancements

Improving the accuracy of predictive models is crucial for maintaining the reliability and trustworthiness of the system.

1. Ensemble Methods:

- Utilize ensemble learning techniques to combine multiple predictive models to improve accuracy and reliability. This approach can help mitigate the weaknesses of individual models by leveraging their collective strengths.

2. Deep Learning Techniques:

- Explore deep learning models, such as recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), which are well-suited for time-series data like flight prices. These models can capture complex patterns and dependencies over time.

3. Feature Engineering:

- Invest in more sophisticated feature engineering to extract and select the most relevant features from the data. This might involve analyzing new data sources, such as social media sentiment, to gauge demand trends or consumer sentiment towards certain travel destinations.

Integration of Real-Time Data and Machine Learning Model Enhancements

Real-time data integration and continuous model enhancement are critical for adapting to the dynamic nature of flight pricing.

1. Real-Time Data Feeds:

- Integrate real-time data feeds to continuously update the model with the latest information, ensuring that the predictions reflect the current market conditions. This could involve partnerships with airlines and travel agencies to access their real-time booking and pricing data.

2. Continuous Learning:

- Implement a continuous learning system where the model is regularly updated with new data, allowing it to evolve and adapt over time. This would help the model stay relevant as market dynamics change.

3. Automated Model Retuning:

- Develop automated processes for periodic model re-evaluation and tuning. This ensures that the model remains optimized in response to new data or changes in underlying market trends.

The potential enhancements outlined above not only aim to improve the existing capabilities of the flight price prediction system but also seek to innovate further by incorporating cutting-edge technologies and methodologies. By focusing on user experience, backend robustness, predictive accuracy, and the integration of real-time data, the system can significantly enhance its value proposition and maintain its relevance in a rapidly evolving market. These improvements will require careful planning, additional resources, and ongoing commitment to research and development, but the long-term benefits will justify the investment, providing users with a more powerful, intuitive, and responsive tool for their travel planning needs.

References

- https://www.youtube.com/live/y4EMEpEnElQ?si=QOQAp8oz4MZNzFka
- https://youtu.be/fq2tXKSmx6s?si=1sV94iLLbCzf4WwY
- https://r.search.yahoo.com/_ylt=Awr49_zJqB5mxtkCYh9XNyoA;_ylu=Y29sbwNncTEEcG9zAzIEdnRpZAMEc2VjA3Ny/RV=2/RE=1714494921/RO=10/RU=https%3a%2f%2fmedium.com%2fgeekculture%2fflight-fare-prediction-93da3958eb95/RK=2/RS=UMxIdPo6Y_hEXf97RXauXgkIsik-
- https://r.search.yahoo.com/_ylt=Awr49_zJqB5mxtkCbB9XNyoA;_ylu=Y29sbwNncTEEcG9zAzMEdnRpZAMEc2VjA3Ny/RV=2/RE=1714494921/RO=10/RU=https%3a%2f%2fwww.analyticsvidhya.com%2fblog%2f2022%2f01%2fflight-fare-prediction-using-machine-learning%2f/RK=2/RS=Lc4ySch3VgvOeEKzdWDM5KgpWv4-
- https://r.search.yahoo.com/_ylt=AwrO6L0IqR5m4TYE2TZXNyoA;_ylu=Y29sbwNncTEEcG9zAzQEdnRpZAMEc2VjA3Ny/RV=2/RE=1714494984/RO=10/RU=https%3a%2f%2fmedium.com%2fanalytics-vidhya%2fthe-flight-ticket-price-prediction-in-python-using-scikit-learn-library-ee3aefc01224/RK=2/RS=tyR2AFgxfibDe.rl05ieHij5kaM-
- https://video.search.yahoo.com/search/video;_ylt=AwrOoZ0jqR5maqwC7IRXNyoA;_ylu=Y29sbwNncTEEcG9zAzEEdnRpZAMEc2VjA3Nj?type=E210US885G0&p=flight+fare+prediction+python+tutorial&fr=mcafee&turl=https%3A%2F%2Ftse4.mm.bing.net%2Fth%3Fid%3DOVP.XhD31U_xaNQIuqLZuStcuAEsDh%26pid%3DApi%26w%3D296%26h%3D156%26c%3D7%26p%3D0&rurl=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DRHXISdJ35-Y&tit=Flight+fare+prediction+web+app+implementation+with+deployment+-+python+for+data+science+%5C+ML+%F0%9F%94%A5likhi&pos=11&vid=259dfd9074259e20df65903b5ffcd53e&sigr=9LLVh4Pmy70e&sigt=CIERvHQvi97R&sigi=IklSaLyr3pX5
- https://github.com/sarthakkmishraa/Flight-Price-Prediction-With-Deployment

- https://github.com/Ankit-c2104/Flight-Fare-Prediction-ML-Project
- https://github.com/Rika290/Flight-Price-Prediction?tab=readme-ov-file

Appendices

Appendix A: Dataset Description

https://www.kaggle.com/datasets/shubhambathwal/flight-price-prediction

- Dataset contains information about flight booking options from the website Easemytrip for flight travel between India's top 6 metro cities. There are 300261 datapoints and 11 features in the cleaned dataset.

Appendix B: Languages used

- Python is used to preprocess, clean test and analyze the data. Both HTML and CSS is used to create the user interface for flight fare prediction application

Appendix C: Additional Statistical Analyses

- Correlation Analysis:

- Description: Measures the strength and direction of the relationship between two variables. For flight prices, this could be between price and factors like the day of the week, time of booking, or holiday periods.

- Purpose: Helps to identify dependencies between variables that could be important predictors in your model.

- Time Series Decomposition:

- Description: Breaks a time series into seasonal, trend, and residual components. Useful if you are using historical data to predict future prices.

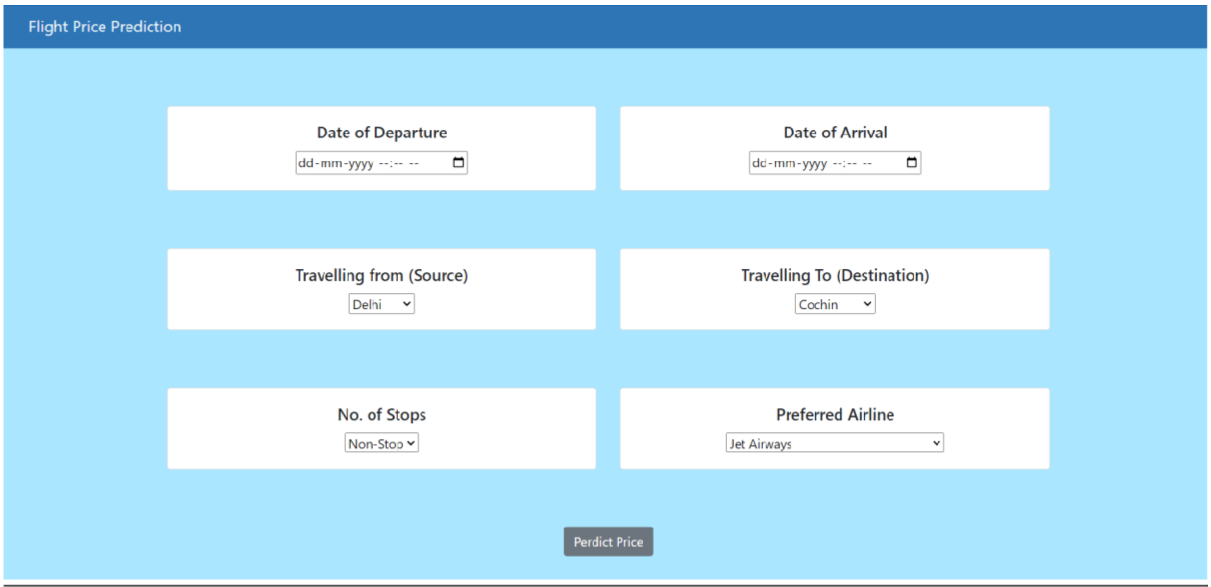
- Purpose: Helps to understand underlying patterns and trends in flight prices over time, which can be crucial for forecasting and seasonal adjustment.

- Cluster Analysis:

- Description: Involves grouping a set of objects in such a way that objects in the same group (cluster) are more similar to each other than to those in other groups. It’s particularly useful for market segmentation.

- Purpose: Can be used to discover distinct groups of flight prices based on various factors, which could inform differentiated pricing strategies or promotional targeting.

Appendix D: User Interface Screenshots


Flight Price Prediction

Date of Departure: dd-mm-yyyy --:-- --

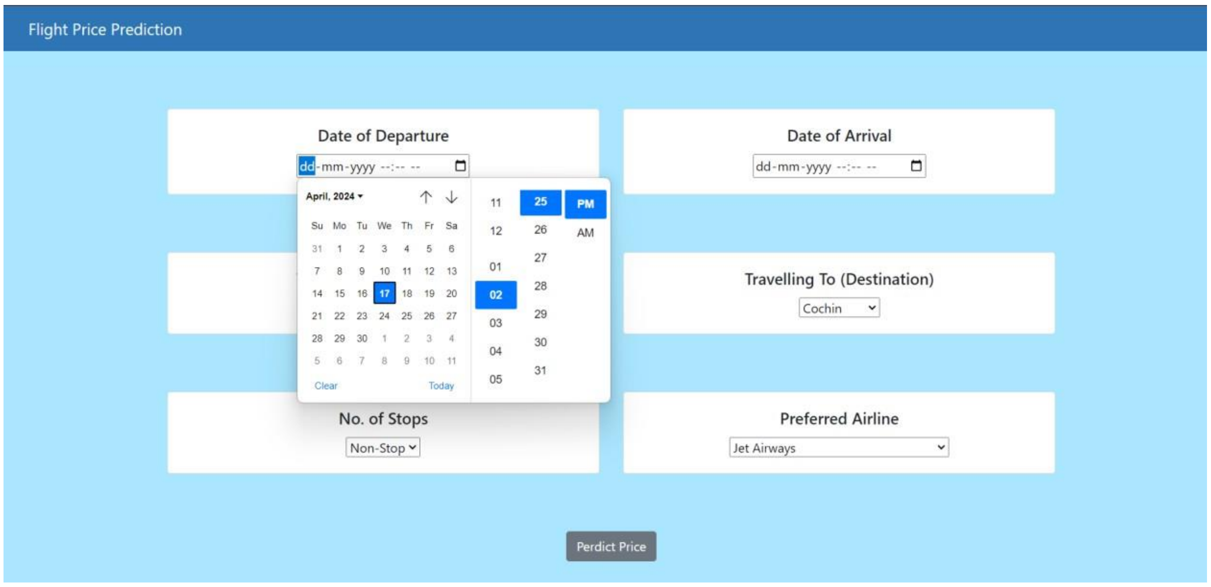
Date of Arrival: dd-mm-yyyy --:-- --

Travelling from (Source): Delhi

Travelling To (Destination): Cochin

No. of Stops: Non-Stop

Preferred Airline: Jet Airways


Flight Price Prediction

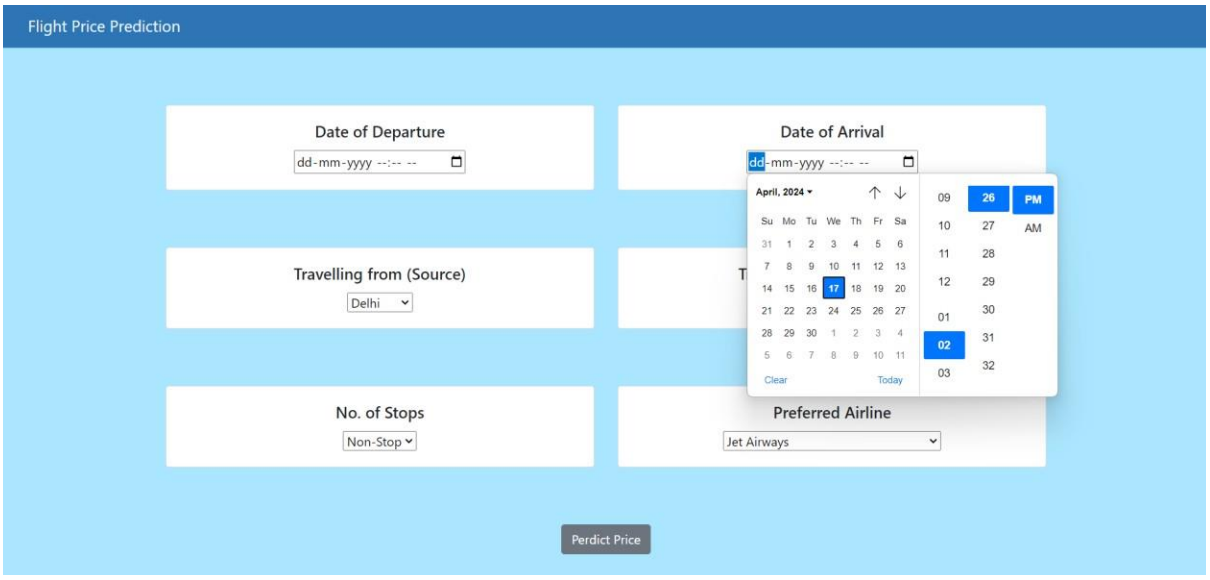
Date of Departure: dd-mm-yyyy --:-- --

Date of Arrival: dd-mm-yyyy --:-- --

Travelling To (Destination): Cochin

No. of Stops: Non-Stop

Preferred Airline: Jet Airways


Flight Price Prediction

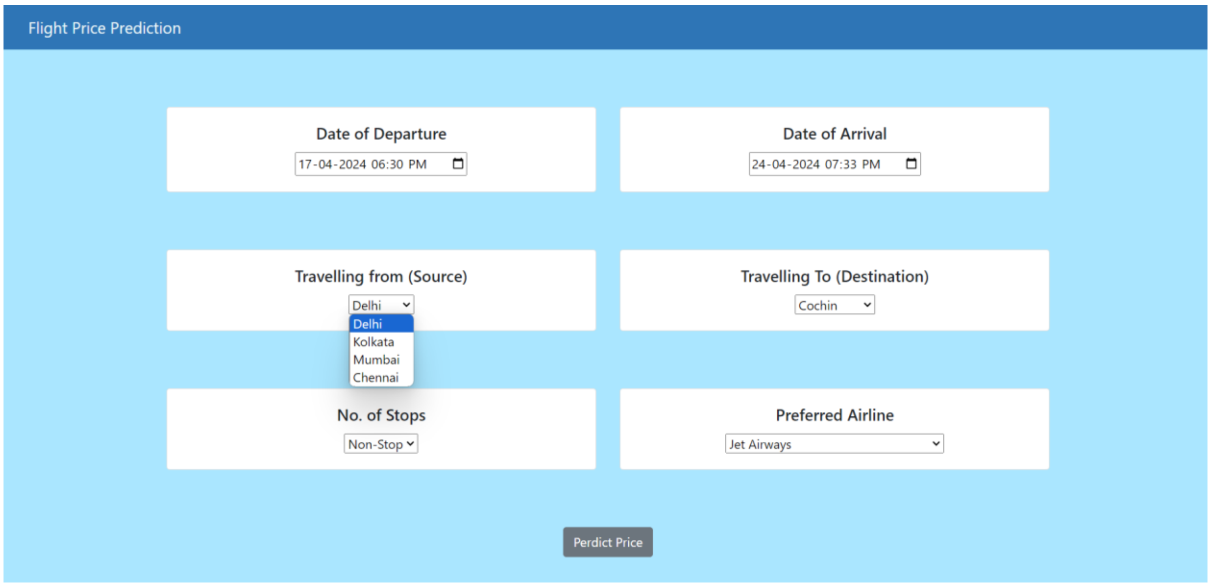
Date of Departure: dd-mm-yyyy --:-- --

Date of Arrival: dd-mm-yyyy --:-- --

Travelling from (Source): Delhi

No. of Stops: Non-Stop

Preferred Airline: Jet Airways


Flight Price Prediction

Date of Departure: 17-04-2024 06:30 PM

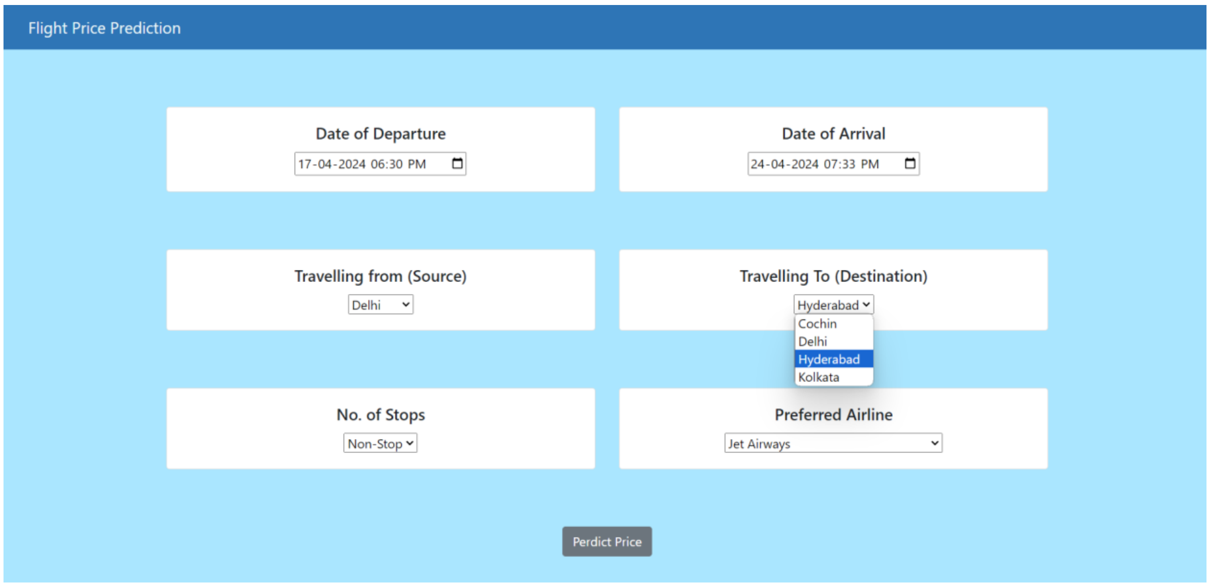
Date of Arrival: 24-04-2024 07:33 PM

Travelling from (Source): Delhi

Travelling To (Destination): Cochin

No. of Stops: Non-Stop

Preferred Airline: Jet Airways


Flight Price Prediction

Date of Departure: 17-04-2024 06:30 PM

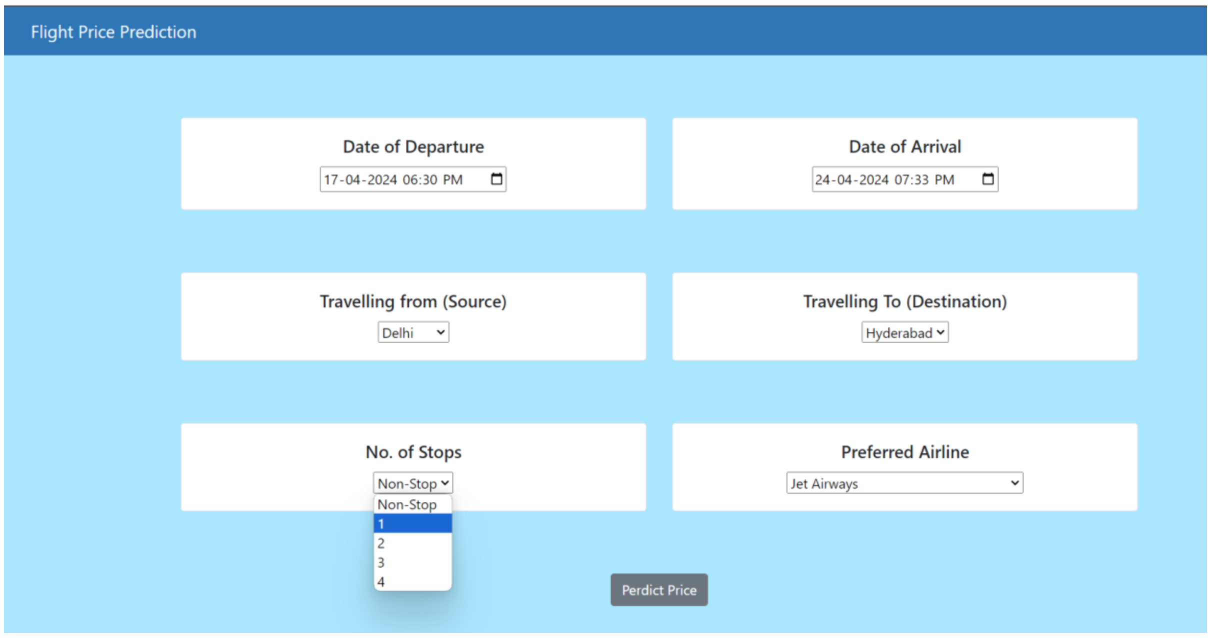
Date of Arrival: 24-04-2024 07:33 PM

Travelling from (Source): Delhi

Travelling To (Destination): Hyderabad

No. of Stops: Non-Stop

Preferred Airline: Jet Airways



Date of Departure
17-04-2024 06:30 PM

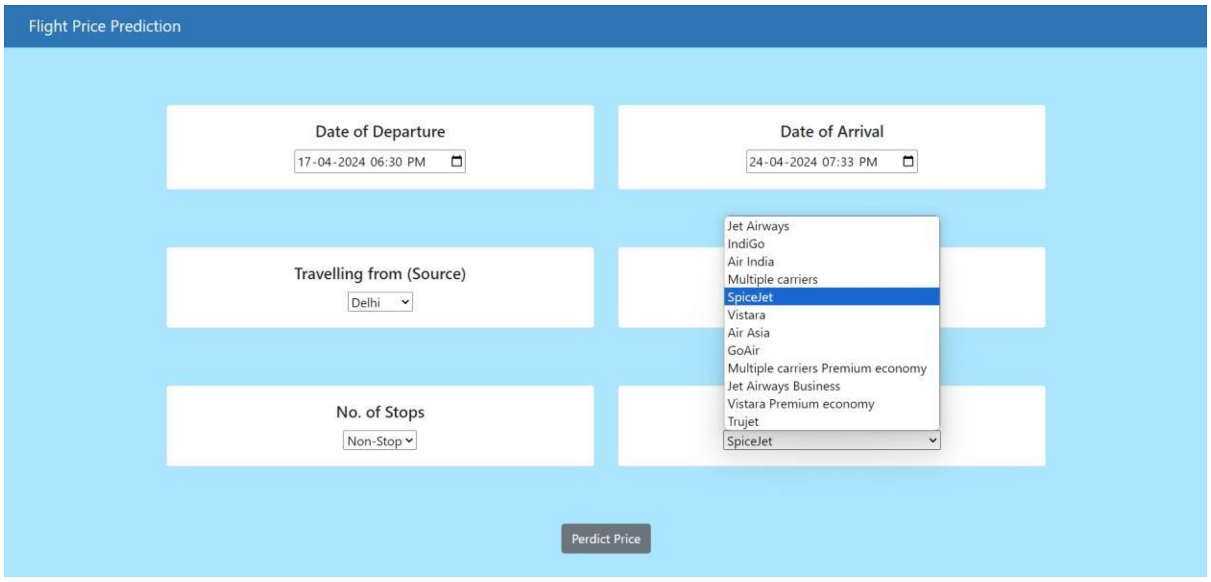
Date of Arrival
24-04-2024 07:33 PM

Travelling from (Source)
Delhi

Travelling To (Destination)
Hyderabad

No. of Stops
Non-Stop

Preferred Airline
Jet Airways



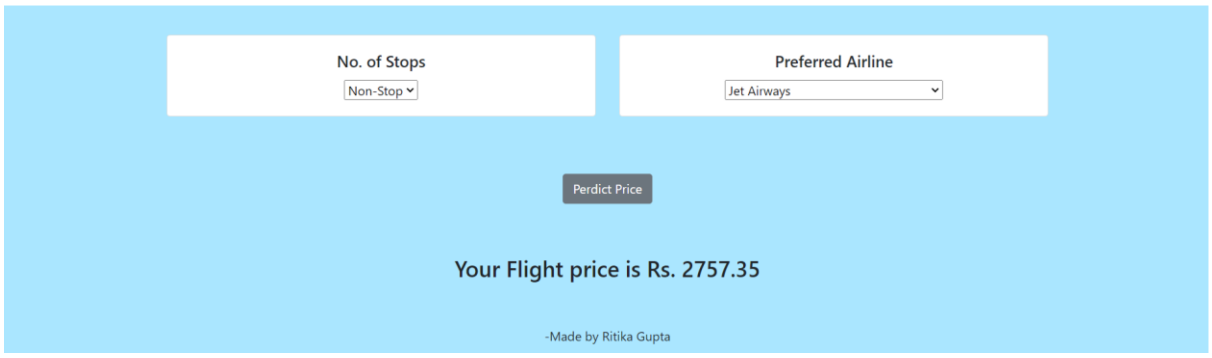
Date of Departure
17-04-2024 06:30 PM

Date of Arrival
24-04-2024 07:33 PM

Travelling from (Source)
Delhi

No. of Stops
Non-Stop

SpiceJet



No. of Stops
Non-Stop

Preferred Airline
Jet Airways

Your Flight price is Rs. 2757.35

-Made by Ritika Gupta