FLIGHT FARE PREDICTION SYSTEM

A DISSERTATION SUBMITTED TO MANCHESTER METROPOLITAN UNIVERSITY
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

September 2024

By
Farrukh Nizam Arain
Department of Computing and Mathematics

Table of Contents

List of Tables

List of Figures

Abstract

Flight ticket prices change quickly around the world due to various factors like the airline, seat class, and flight duration. Flight aggregator websites often do not provide all the necessary information, making it hard for customers to make informed travel decisions based on their budget. This study aims to develop a robust flight fare prediction system that not only predicts the prices accurately but also identifies the key factors driving fare fluctuations. Using historical flight data sourced from Indian airlines, five algorithms were used: Linear Regression, Decision Tree Regression, Random Forest Regression, K-Nearest Neighbours (KNN), and Gradient Boosting Regression. The accuracy and evaluation metrics were optimised using the Randomized Search Cross Validation (CV) technique, with Gradient Boosting Regression emerging as the best model. To further improve prediction accuracy, three ensemble algorithms were used: Average Ensemble Regression, Voting Ensemble Regression, and Stacking Ensemble Regression, with Stacking Ensemble Regression performing the best. Shapley Additive Explanations (SHAP) algorithm was used to analyse feature importance in the chosen ensemble model, identifying seat class, flight number, days until departure, and flight duration as key factors affecting fare predictions. This approach ensures the flight prediction system is robust and accurate, helping users make informed decisions based on their budget and preferences.

Declaration

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work. This work has been carried out in accordance with the Manchester Metropolitan University research ethics procedures and has received ethical approval number 69727.

Signed: Farrukh Nizam Arain

Date: 27th September 2024

Acknowledgements

I would like to sincerely thank my supervisor, Mr. Anthony Kleerekoper, for his constant support, guidance, and encouragement during the development of this flight fare prediction system. His expertise and helpful advice were key in steering the project towards success. I genuinely appreciate the time and effort he invested in providing feedback and direction, which helped me overcome many challenges along the way.

I am also incredibly grateful to Manchester Metropolitan University and my lecturers for their ongoing support and for providing a great learning environment throughout my MSc Data Science course. The resources, facilities, and knowledge shared by the staff have been incredibly helpful in shaping my understanding of the subject. Their contributions were invaluable in enabling me to complete this project to the best of my ability.

Abbreviations

Analysis of Variance……. ANOVA
Cross Validation……..……CV
Consumer Price Index……CPI
Decision Tree Regression…… DTR
Exploratory Data Analysis……...…EDA
First Quartile……... Q1
Gradient Boosting Regression……. GBR
Interquartile Range…… IQR
K-Nearest Neighbours…….……..... KNN
K-Nearest Neighbours Regression…….……. KNNR
Linear Regression……... LR
Mean Absolute Error……... MAE
Mean Absolute Percentage Error…….…. MAPE
Mean Squared Error……. MSE
Mean Squared Logarithmic Error……. MSLE
Multilayer Perceptron…..…….…… MLP
Partial Least Squares……. PLS
R-Squared……...……. R^2
Random Forest Regression……..… RFR
Residential Sum of Squares…….……. RSS
Root Mean Squared Error……. RMSE
Root Mean Squared Logarithmic Error ……. RMSLE
Total Sum of Squares…… TSS
Shapley Additive Explanations……. SHAP
Third Quartile……. Q3

Chapter 1 - Introduction

According to a recent report by the International Air Transport Association (2023), the demand for air travel is expected to double by 2040, growing at 3.4 percent per year. This rapid growth is driven by established airlines expanding their fleets and new airlines entering the market. Arjun et al. (2022) highlighted how airlines now use dynamic pricing strategies influenced by factors like ticket price data, demand, seasonality, seat availability, fuel costs, and market competition. They found that ticket prices can vary significantly even for seats close together on the same flight, as airlines use various mathematical methods and predictive models to optimise pricing and increase profitability. Consequently, travellers are constantly looking for the best deals, which intensifies competition among airlines and travel agencies to offer the best fares and services across different budget segments.

Traditionally, airlines have used statistical methods and historical data to predict airfares. While useful, these methods often fail to account for market complexities and real-time fluctuations. Nadeem and Sivakumar (2023) noted two major shifts in fare prediction strategies: the advent of more powerful computing systems and the integration of advanced machine learning algorithms. Yamini et al. (2024) concluded that high-powered systems allow airlines to process large datasets quickly, analysing historical trends, flight details, and customer behaviour to uncover hidden patterns. Advanced machine learning algorithms can analyse massive datasets with many features, such as historical fares, seat availability, flight durations, weather conditions, and booking trends, predicting future prices with high accuracy. Gopal et al. (2024) showed that using historical flight pricing data and other relevant factors enables machine learning algorithms to accurately forecast future ticket prices. Shukla et al. (2020) noted that these recent technological improvements allow airlines to adjust their pricing strategies to balance profitability with competitive rates, providing customers with more reasonable and timely fare options.

Innovations in airfare prediction have become valuable for customers due to the sophistication of machine learning models. These models analyse a wide range of variables that influence flight prices, such as flight duration, seat class, booking timing, the number of available seats, and external factors like holidays or economic conditions. By leveraging these insights, customers can identify the best booking times, avoid price spikes, and secure lower fares. This knowledge helps customers save money and plan their trips more effectively, making informed decisions based on a comprehensive understanding of price-influencing factors.

The problem addressed in this project is the unpredictability of flight prices for customers who face difficulty in purchasing affordable tickets due to the complex and rapidly changing pricing systems adopted by airlines. Customers lack a reliable prediction system to help them identify the cheapest ticket price as per their preferences. Hence, a robust prediction system would

provide substantial value to consumers by analysing the different factors contributing to fare fluctuations.

The aim of this project is to create an effective flight fare prediction system using machine learning techniques. Several objectives are set to accomplish this goal, beginning with gathering and importing the necessary data using advanced software tools. Once collected, the data is cleaned, prepared, and transformed for analysis. Exploratory Data Analysis (EDA) is then applied to identify initial patterns or trends, helping to understand how different factors may influence flight prices. These models are fine-tuned with cross-validation techniques to improve performance. Additionally, techniques are used to identify which factors, such as booking time, seat availability, flight duration, and class have the greatest impact on price changes. This approach ensures that the fare prediction system is reliable and accurate, providing insights into the factors that affect flight pricing.

The rest of the research paper is structured as follows: The Literature Review chapter provides an overview of relevant research on flight fare prediction systems, exploring different methodologies, models, and techniques used in previous studies. The Methodology chapter outlines the process of data collection, cleaning, and preparation, the application of machine learning models to the dataset, and the techniques used to enhance model performance through cross-validation and optimisation, including feature importance analysis. The Results and Analysis chapter presents the findings from the model training and testing phases, highlighting the accuracy of predictions and identifying key factors that influence ticket prices. The Evaluation chapter critically assesses the effectiveness of the models, considering their strengths, limitations, and real-world applicability. Finally, the Conclusion summarises the key outcomes of the study, discusses its broader implications, and offers suggestions for future research.

Chapter 2 - Literature Survey

In chapter 1, the unpredictability of airfare was discussed and the need of reliable prediction system to help customers find cheaper tickets. This chapter reviews on airfare prediction models, with an emphasis on machine learning algorithms that enhance accuracy. It lays the groundwork for the next chapter, which will explain the methods chosen for this project.

Predicting airline ticket prices accurately is crucial for both travellers and airlines due to the complex dynamics of the aviation industry. Shukla et al. (2020) emphasised that effective airfare prediction is vital for optimising airline revenue, improving operational efficiency, and helping consumers find the best fares. The growing importance of this task is evident in the increasing research focused on refining airfare prediction models using various machine learning algorithms. These algorithms are continually evolving to address the complex factors influencing airfare, such as demand variability, seasonal trends, and broader economic conditions. This literature review explores the development of airfare prediction models, focusing on early studies, demand estimation models, macroeconomic factors, recent machine learning approaches, and advancements in feature engineering.

One of the pioneering studies in airfare prediction was by Etzioni et al. (2003), who introduced the HAMLET model. This model combined Time-Series Analysis, Ripper, and Q-learning. Ripper is a rule-based learning algorithm that creates classification rules from data, while Q-learning is a reinforcement learning algorithm that determines the value of actions in a changing environment. The dataset for this study was extracted from a major travel website and focused on two routes for both non-stop and round-trip flights: Los Angeles (LAX) to Boston (BOS), and Seattle (SEA) to Washington, DC (IAD). Key features included flight number, hours until departure time, current price, airline name, and route. Data was collected for 7-day round trips, with prices recorded from 21 days before each departure date at three-hour intervals, eight times a day. The model's output was either 'buy' or 'wait', guiding passengers on the best time to purchase tickets to get the lowest price. Despite its innovative approach, the HAMLET model was limited by its small dataset and lack of real-time data processing, which restricted its ability to generalise across different scenarios.

In another study, Lantseva et al. (2015) analysed airfare data for Russian cities (Moscow and St Petersburg), collected from AviaSales and Sabre for both international and domestic flights

up to 90 days before departure. They developed a regression model and found that for international flights, buying tickets early helped customers secure lower prices compared to purchasing close to the departure date. However, no clear pattern was observed for domestic flights due to high competition within the local Russian market and infrastructure lagging behind that of foreign countries. This study highlighted regional differences in airfare trends and the challenges of predicting domestic fares in certain markets.

Accurate demand estimation is critical for pricing strategies in the airline industry. Each airline operates a similar system because every flight has a limited number of seats, and they must manage demand while maximising profits. When high demand is anticipated, the price of each seat is increased significantly. Conversely, if demand is expected to decrease, the price of that flight is reduced to at least cover the service cost, as an unsold seat is a loss for the airline.
Papadakis (2014) examined how airlines use demand forecasting to adjust seat prices based on factors such as time of day, seasonality, and competition. His models employed regression techniques to estimate demand fluctuations, analysing the relationship between dependent and independent variables to predict future outcomes. While Papadakis's models provided valuable insights into demand estimation, they were limited by the diversity of flight data and did not fully capture the complex relationships between various factors affecting airfare.

Groves and Gini (2013) presented a case study where an agent was developed to optimise flight ticket purchase timings on behalf of customers. The initial steps involved feature extraction and lagged feature computation before constructing a regression model. The best model was selected based on a hold-out set. Among the machine learning algorithms tested, Partial Least Squares (PLS) regression performed well due to its resistance to collinear and irrelevant variables, although it required substantial domain knowledge for feature set generation. This study highlighted the importance of accurate feature selection in improving airfare predictions.

Macroeconomic variables, such as crude oil price and the Consumer Price Index (CPI), play a crucial role in influencing airfare pricing. Wang (2019) integrated these macroeconomic indicators with data from the Origin and Destination Survey (DB1B) and Air Carrier Statistics (T-100) to predict average airfare prices on a quarterly basis using a regression model. The proposed model achieved an R-squared score of 0.869, indicating a high level of accuracy.

However, this model was restricted by its quarterly prediction granularity and did not account for real-time data or dynamic factors such as booking time and seat availability.

Janssen (2014) employed the Linear Quantile Blended Regression methodology on a flight fare dataset from San Francisco Airport (SFO) to John F. Kennedy Airport (JFK) provided by Infare Solutions. Out of twenty-seven features, four were selected, and two additional features were derived from these. The two prominent features were the number of days until departure and whether the departure was on a weekend. While the prediction model worked well for days far from the departure time, it was less effective for the days closer to departure.

In recent years, machine learning algorithms have significantly enhanced the accuracy of airfare prediction. Rajankar et al. (2019) proposed an airline price prediction system utilising machine learning algorithms, including Linear Regression, Decision Tree, Random Forest, and K-Nearest Neighbour (KNN) Regressor. Linear Regression models calculate the relationship between a dependent variable and one or more independent variables. Decision Trees create a model that splits data into branches based on feature values. Random Forest combines multiple decision trees to improve accuracy and reduce overfitting. KNN groups data points based on the majority class of their nearest neighbours. Their study utilised a dataset from makemytrip.com, including related features such as the date of departure, time of departure, place of departure, time of arrival, place of destination, airlines, and total fare. After analysing the results, the authors concluded that KNN had the highest accuracy with an R-Squared value close to one, but the absence of seat availability data limited its predictive capabilities. The comparison in this paper highlighted KNN's superior accuracy in handling complex feature interactions, while Random Forest offered valuable insights into feature importance.

A paper by Kimbhaune et al. (2021) extended this research by applying Linear Regression, Decision Tree, and Random Forest to a larger dataset containing over 10,000 flight records. Features in the dataset included source, destination, departure date, departure time, number of stops, arrival time, price, and others. Results from these models were evaluated using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). MAE measures the average magnitude of errors in predictions, while RMSE quantifies the standard deviation of prediction errors. Random Forest generally outperformed other algorithms, but challenges in achieving

high accuracies continued, highlighting the need for advanced feature engineering and model optimisation.

Vu et al. (2018) introduced a model focusing on markets with additional features which were not present in previous models. Their model assisted buyers in their purchasing decisions by predicting price trends even without official information from airlines. The study concentrated on the Vietnamese aviation market, a rapidly growing sector in the developing world. Despite the significant dataset size, it lacked some features like the number of unsold seats. Among the models tested, Random Forest and Multilayer Perceptron (MLP) performed the best. MLP, a type of neural network with multiple layers of neurons, is effective in modelling complex, non-linear relationships. Prediction scores were further improved by combining them into a new stacked prediction model, demonstrating that features like the interval between purchase and departure date are critical in rapidly growing markets. Despite high performance, the lack of seat availability data remained a limitation.

Table 2.1 summarises the key contributions of the reviewed studies, comparing algorithms, features considered, and model accuracy.

Study	Algorithms Used	Features Considered	Dataset Size	Accuracy / R-squared	Limitations
Etzioni et al. (2003)	Time-Series, Ripper, Q-Learning	Flight details (number, airline, route)	Small dataset	N/A	Small dataset, limited features, no real-time data
Lantseva et al. (2015)	Regression Model	Departure date, competition, infrastructure quality	Moderate dataset	Moderate	No clear domestic patterns, regional differences
Papadakis (2014)	Regression Models	Time of day, seasonality, competition	Moderate dataset	Moderate	Limited data variety, complex feature relations not fully explored
Groves & Gini (2013)	PLS Regression	Flight purchase timing, lagged feature computation	Moderate dataset	High	Domain knowledge needed for feature generation
Rajankar et al. (2019)	Linear Regression, KNN, Decision Tree, Random Forest	Date/time of departure, destination, fare, place of departure and destination	Medium dataset	High (KNN: R^2 close to 1)	Lack of seat availability data, limited feature set
Kimbhaune et al. (2021)	Linear Regression, Decision Tree, Random Forest	Source, destination, date/time of departure number of stops	Large dataset	High	Accuracy challenges, advanced feature engineering needed
Vu et al. (2018)	Random Forest, MLP	Interval to departure, booking time	Large dataset	High	Missing seat data, focused on

Study	Algorithms Used	Features Considered	Dataset Size	Accuracy / R-squared	Limitations
					developing markets only
Wang (2019)	Regression Model	Oil prices, CPI, Origin/Destination data	Large dataset	High	Quarterly predictions, lack of real-time data
Janssen (2014)	Linear Quantile Blended Regression	Days from departure, weekend status	Medium dataset	Moderate	Struggled with near-departure date predictions

Table 2.1: Comparison of Methodologies by Researchers

In this chapter, studies on predicting airfares were reviewed, with a focus on the methods and algorithms used to address the challenges of fare forecasting. The comparison table highlights key differences between these approaches, providing insights into the most effective algorithms and features. The next chapter will apply machine learning techniques to develop a flight fare prediction system, incorporating lessons from this review to improve prediction accuracy.

Chapter 3 - Methodology

Building on the findings from the previous chapter, which reviewed various models for predicting airfare, this chapter outlines the methodology for developing a flight fare prediction system. It covers the selection of techniques for data collection, transformation, splitting, training, optimisation, and feature assessment.

3.1 Data Collection

The data collection phase involves gathering the relevant data to predict flight fares, emphasizing the importance of selecting relevant data sources. Data such as ticket prices, flight schedules, and other relevant variables were collected. This step is crucial as the quality and relevance of the data directly impact the accuracy of the predictive models.

3.1.1 Data Source

The dataset was sourced from the EaseMyTrip website, a well-known platform for flight bookings. Data was scraped using the Octoparse tool, focusing on flight booking options. Two separate files were collected: one for economy class and the other for business class tickets. The data was collected on February 10th, 2022, and spans a fifty-day period from February 11th to March 31st, 2022. The Dataset covers flights between the top six metro cities of India.

3.1.2 Data Structure

The dataset contains 300,261 entries across 11 columns, providing detailed information about flight fares. Most columns contain string data, with one column is an integer. This structure enables a thorough analysis of flight fares across airlines and classes. Table 3.1 summarises the dataset used in this project.

Column	Non-Null Value Count	Data Type
date	300261	object
airline	300261	object
ch_code	300261	object
num_code	300261	int64
dep_time	300261	object

Column	Non-Null Value Count	Data Type
from	300261	object
time_taken	300261	object
stop	300261	object
arr_time	300261	object
to	300261	object
price	300261	object
class	300261	object

Table 3.1: Data Frame Structure Summary

3.2 Data Preprocessing

Data preprocessing is an essential step for converting raw data into useable format. The key tasks in this step include data cleaning, normalisation, transformation, and feature engineering to ensure that the dataset is compatible with machine learning algorithms.

3.2.1 Data Cleaning

Data cleaning resolves numerous issues in the dataset, such as missing values, duplicates, errors, outliers, and inconsistencies, to improve data quality and reliability.

3.2.1.1 Merging Datasets

Before data cleaning, economy and business class datasets were merged into one by adding a 'class' column to label each row as 'Business' or 'Economy'. This merged datasets was combined using the ‘pd.concat()’ function to create a continuous index. Table 3.2 provides a merged view of the flight fare dataset that was used in later steps of the research.

date	airline	ch_code	num_code	dep_time	from	time_taken	stop	arr_time	to	price	class
11-02-2022	SpiceJet	SG	8709	18:55	Delhi	02h 10m	non-stop	21:05	Mumbai	5,953	Economy
11-02-2022	SpiceJet	SG	8157	06:20	Delhi	02h 20m	non-stop	08:40	Mumbai	5,953	Economy

Table 3.2: Sample of Merged Dataset

3.2.1.2 Duplicate Rows Removal

To avoid potential bias due to redundant records, duplicates were identified using the 'duplicated()' function, which flagged all instances of duplicate rows. These rows were displayed for further inspection and were subsequently dropped from the dataset using the 'drop_duplicates()' function to ensure that changes were made directly to the original dataset. Table 3.3 displays the list of duplicate rows that were removed.

date	airline	ch_code	num_code	dep_time	from	time_taken	stop	arr_time	to	price	Class
14-02-2022	Air India	AI	807	17:20	Delhi	15h 15m	1stop \n\t..	08:35	Mumbai	12,150	Economy
14-02-2022	Air India	AI	807	17:20	Delhi	15h 15m	1stop \n\t\..	08:35	Mumbai	12,150	Economy
13-03-2022	Air India	AI	475	13:00	Delhi	24h 35m	1stop \n\t..	13:35	Mumbai	4,780	Economy
13-03-2022	Air India	AI	475	13:00	Delhi	24h 35m	1stop \n..	13:35	Mumbai	4,780	Economy

Table 3.3: Duplicate Rows in the Dataset

3.2.1.3 Assessment of Missing Values

To prevent issues while training the dataset in the modelling phase, it was checked for missing or NaN values using the 'isnull()' method, which counts missing values in each column. No missing values were found in the data after running the code. The results displayed in Table 3.4 confirms that the dataset was complete and did not require further imputation.

Column	Missing Values
date	0
airline	0
ch_code	0
num_code	0
dep_time	0
from	0
time_taken	0
stop	0
arr_time	0
to	0

Column	Missing Values
price	0
class	0

Table 3.4: Missing Values in the Dataset

3.2.1.4 Data Type Verification and Transformation

An initial check of each feature's datatype was carried out using the 'df.info()' function. While most columns had the correct datatype, the 'price' column was identified as an object type due to commas in the values (e.g., "4,452" and "5,234"). After removing the commas from the price values, the column was transformed into the integer datatype. This transformation ensured that the 'price' column would not have any incompatibility issues, as it was imperative for the dependent feature to be in a numerical data type before starting the modelling phase of this project.

3.2.1.5 Data Inconsistencies, Identification, and Standardisation

After the initial analysis of the unique values in every column, it was identified that the 'stop' column required standardisation. The values of 'non-stop' and '2+-stop' were easily converted to 'Zero' and 'Two or More' respectively. Boolean expressions were utilised to remove unwanted escape characters and whitespace in the '1-stop' values. Table 3.5 displays the old and replaced values in the 'stop' column.

Old Values	Replaced Values
non-stop	Zero
1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t…	1-stop
2+-stop	Two or More

Table 3.5: Standardisation of 'Stop' Column

Most values in the 'time_taken' column followed a specific hours and minutes pattern (e.g., '02h 10m', '12h 15m'). However, some values like '1.03h m', '1.02h m', and '1.01h m' did not match this pattern. To avoid any potential errors during the transformation phase, regular expressions were first utilised to identify these irregular values, which were then removed to ensure data cleanliness and consistency. Table 3.6 lists the inconsistent values which were removed.

date	airline	ch_code	num_code	dep_time	from	time_taken	stop	arr_time	to	price	class
26-02-2022	GO FIRST	G8	146	05:45	Bangalore	1.03h m	One	09:10	Mumbai	5177	Economy
26-02-2022	GO FIRST	G8	146	05:45	Bangalore	1.02h m	Two or More	08:00	Kolkata	5177	Economy
26-02-2022	GO FIRST	G8	146	05:45	Bangalore	1.03h m	Two or More	09:30	Hyderabad	4337	Economy
25-02-2022	GO FIRST	G8	517	20:45	Hyderabad	1.01h m	One	21:50	Delhi	6132	Economy

Table 3.6: Inconsistent ‘Time_Taken’ Values

3.2.1.6 Verification of Negative Values in Numeric Columns

An additional check was performed to ensure that no negative values were present in the numerical columns. The ‘select_dtypes()’ function was used to filter all the numeric columns, and ‘lt(0)’ was also used to check for values less than zero. No negative values were found after executing the code, confirming the dataset's integrity and readiness for further analysis.

3.2.2 Feature Engineering

Feature engineering is critical for improving model performance by creating and selecting relevant features. This process includes deriving new features from existing data, such as extracting insights from datetime values. It may also involve categorising continuous values into ranges or frequency-based groups to help the model identify patterns more efficiently.

3.2.2.1 Calculating Days Until Flight Departure

A new column called 'flight_days_left' was introduced to calculate the number of days left until a flight’s departure, starting from a fixed date (10th February 2022), when the data was collected. The ‘date’ column was first converted from a string datatype to datetime objects, and the difference in days was subtracted from the data collection date. The original ‘date’ column

was then removed to avoid duplication. Table 3.7 shows a sample of this transformation from temporal data into a numeric feature that is more suitable for machine learning models to process efficiently.

Original date	'flight_days_left'
26-02-2022	16
25-02-2022	15
24-02-2022	14

Table 3.7: Date to 'Flight_Days_Left' Transformation

3.2.2.2 Grouping Departure and Arrival Hours

To make the 'dep_time' and 'arr_time' columns more meaningful, the hour and minute values were extracted and saved in a new data frame. An adjustment was applied: if the minute value was 0, the hour was reduced by 1, and -1 values were replaced by 23. This adjustment ensured consistency and handled edge cases effectively. The modified hour values were then grouped into predefined time-of-day slots, such as 'Late Night' and 'Morning', using the 'pd.cut()' function. Table 3.8 shows the hour ranges and their respective categories.

Rows with null values in the new 'departure_time' and 'arrival_time' columns were dropped to maintain dataset's integrity. The original columns were replaced by these new columns, simplifying the dataset. This transformation allowed the model to work with broader time categories, ultimately enhancing the pattern identification process.

Hour Range	Time of Day
00 - 03	Late Night
04 - 07	Early Morning
08 - 11	Morning
12 - 15	Afternoon
16 - 19	Evening
20 - 23	Night

Table 3.8: Hour to 'Time of Day' Mapping

3.2.2.3 Converting Flight Time into Hours

The 'time_taken' column was processed to calculate the total flight time in hours. The hour and minute components were extracted from the original strings, with missing minute values

assigned a value of 0. The minutes were converted to a fraction of an hour and added to the hour value. This new flight duration was rounded to two decimal places, and the original ‘time_taken’ column was replaced with this more precise value, as shown in Table 3.9.

Original ‘time_taken’ value	Extracted Hours	Extracted Minutes	Calculated Total Time (hours)
1h 30m	1	30	1.50
2h 45m	2	45	2.75
3h 0m	3	0	3.00
0h 50m	0	50	0.83

Table 3.9: Transformation of ‘Time_Taken’ Values

3.2.2.4 Generating Unique Flight Numbers

To create a unique identifier for each flight, the 'ch_code' and 'num_code' columns were combined into a single 'flight_number' column, separated by a hyphen. The original columns were removed to streamline the data. A sample of these transformed values are shown in Table 3.10.

ch_code	num_code	flight_number
G8	146	G8-146
AI	302	AI-302
6E	205	6E-205
SG	401	SG-401

Table 3.10: Creation of ‘Flight_Number’

3.2.2.5 Standardising and Rearranging Feature Names

Column names were revised to be more descriptive during feature engineering. For instance, ‘airline’ was renamed as 'Airline Name', and 'source_city’ was changed to ‘Departure City’. This renaming made the dataset more user-friendly and easier to understand. A full list of the changes in the column names is provided in Table 3.11.

Old Column Name	New Column Name	Description
airline	Airline Name	Name of the airline
source_city	Departure City	City from which the flight departs
time_taken	Flight Duration	Duration of the flight
stops	Number of Stops	Number of stops during the flight

Old Column Name	New Column Name	Description
arrival_city	Destination City	City where the flight arrives
price	Price	Cost of the flight
days_left	Flight Days Left	Days left until the flight departs
departure_time	Departure Time	Time of departure
arrival_time	Destination Time	Time of arrival
Flight Number	Flight Number	Concatenated Flight number

Table 3.11: Column Name Changes Summary

After renaming, the columns were rearranged into an optimised order, such as placing 'Flight Number' and 'Seat Class' next to each other, facilitating analysis that may require these features to be closely related with each other. Table 3.12 shows a sample of the dataset after rearranging of the column names.

Airline Name	Flight Number	Seat Class	Departure City	Departure Time	Number of Stops	Destination City	Destination Time	Flight Duration	Flight Days Left
SpiceJet	SG-8709	Economy	Delhi	Evening	Zero	Mumbai	Night	2.17	1
SpiceJet	SG-8157	Economy	Delhi	Early Morning	Zero	Mumbai	Morning	2.33	1

Table 3.12: Sample of Transformed Dataset

3.2.2.6 Outlier Detection and Treatment

Outlier detection was performed to maintain the dataset's quality and reliability. A subplot containing three boxplots in Figure 3.1 were plotted for numerical columns, and the Interquartile Rangle (IQR) method was applied to calculate the lower and upper bounds for potential outliers. Through Equation (3.1), the IQR is calculated as the difference between the third quartile (Q3) and the first quartile (Q1) of the dataset:

$$IQR = Q3 - Q1 \tag{3.1}$$

To define the threshold for outlier detection, the following lower and upper bounds were set using equation (3.2) and (3.3):

$$Lower\,Bound = Q1 - 1.5 \times IQR \tag{3.2}$$

$$Upper\,Bound = Q3 + 1.5 \times IQR \tag{3.3}$$

These bounds were plotted as vertical lines on the boxplots, clearly marking the threshold beyond which values are considered outliers. Annotations were added to each boxplot to indicate the specific bounds, ensuring a clear visual understanding of where the outliers are.

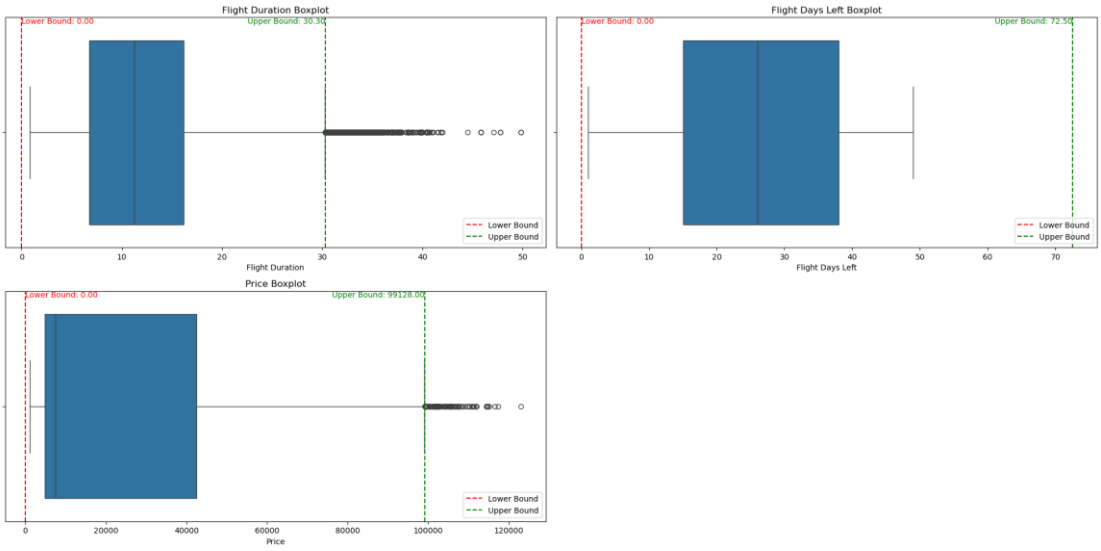


Figure 3.1: Boxplots of Numerical Columns for Outlier Detection

Upon examining the boxplots, a significant number of outliers, as in Table 3.13, were observed in the 'Flight Duration' column, with values equal to exceeding 30.30 hours. Normally, this would seem odd for domestic flights, as such durations typically indicate international travel or operational delays. However, verification with the original source confirmed that these long-duration domestic flights were valid. Domestic flights can sometimes have such long durations due to multiple layovers, contributing to these extended travel times.

Airline Name	Flight Number	Seat Class	Departure City	Departure Time	Number of Stops	Destination City	Destination Time	Flight Duration	Flight Days Left	Price
Vistara	UK-706	Economy	Delhi	Evening	Two or More	Bangalore	Afternoon	31.25	4	12222
Vistara	UK-706	Economy	Delhi	Afternoon	Two or More	Bangalore	Afternoon	33.17	4	12222
Air India	AI-9887	Economy	Delhi	Evening	Two or More	Bangalore	Evening	36.92	4	12321

Table 3.13: Sample Data with Flight Duration Outliers

Similarly, outliers in the 'Price' feature were identified, with some values exceeding INR 99,128.00. After further investigation from the source of the dataset, it became clear that these higher prices were associated with business class, which generally have higher prices than economy class seats. Table 3.14 shows that Vistara's business-class seats often exceed these thresholds, reflecting the higher class of service that this airline provides.

Airline Name	Flight Number	Seat Class	Departure City	Departure Time	Number of Stops	Destination City	Destination Time	Flight Duration	Flight Days Left	Price
Vistara	UK-706	Economy	Delhi	Evening	Two or More	Bangalore	Afternoon	31.25	4	12222
Vistara	UK-809	Business	Delhi	Afternoon	Two or More	Kolkata	Night	21.08	1	116562

Table 3.14: Sample Data with Price Outliers

Despite their extreme values, they were kept in the dataset rather than removed. The reason for keeping them in the dataset is that they represent real, valid scenarios within the aviation industry. Removing these values would risk losing valuable data about the range and variations of ticket prices and flight durations, especially when these factors play a vital role in impacting ticket prices and travel experience.

3.3 Exploratory Data Analysis

After data cleaning and feature engineering, Exploratory Data Analysis (EDA) was conducted to examine the processed dataset and uncover patterns and relationships. This phase involves summarising statistics, visualising feature distributions, and exploring correlations to understand how features interact with each other and the target variable. EDA ensures the data is well-prepared for modelling, providing insights that can help refine the feature set and improve the overall analysis.

To begin, numerical and categorical features were separated and stored in temporary variables. Appropriate models and plots then applied to extract meaningful insights from the data.

3.3.1 Descriptive Statistical Analysis

In the Exploratory Data Analysis (EDA) phase, statistical analysis summarised the main characteristics of a dataset. This included calculating measures of central tendency (mean, median, mode), measures of variability (standard deviation, variance, range), and understanding the distribution of data. This analysis helped identify patterns, detect anomalies, and form hypotheses for further analysis.

Various statistical measures were employed to describe the dataset. The mean provided the average value, the median identified the middle value when the data was sorted, and the mode indicated the most frequently occurring value. Standard deviation and variance measured the spread of the data, while the range captured the difference between the maximum and minimum values. Skewness was also calculated to measure the asymmetry of the data distribution. Positive skewness indicated a longer tail on the right side, while negative skewness indicated a longer tail on the left. The 25th and 75th percentiles provided insights into the distribution by highlighting the values below which 25% and 75% of the data fell, respectively.

The 'df.describe()' function in Pandas library was used to calculate most of these statistical measures. It provided a summary of the central tendency, dispersion, and shape of the dataset's distribution, excluding NaN values. This function calculated the count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum for each numerical column.

Method	Airline Name	Flight Number	Seat Class	Departure City	Departure Time	Number of Stops	Destination City	Destination Time	Flight Duration	Flight Days Left	Price
count	300243	300243	300243	300243	300243	300243	300243	300243	300243.000000	300243.000000	300243.000000
unique	8	1569	2	6	6	3	6	6	NaN	NaN	NaN
top	Vistara	UK-706	Economy	Delhi	Morning	One	Mumbai	Night	NaN	NaN	NaN
freq	127854	3235	206768	61343	71183	250913	59106	91551	NaN	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12.217755	26.003747	20882.294505
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	7.192916	13.560498	22694.725010

Method	Airline Name	Flight Number	Seat Class	Departure City	Departure Time	Number of Stops	Destination City	Destination Time	Flight Duration	Flight Days Left	Price
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.830000	1.000000	1105.000000
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.750000	15.000000	4783.000000
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	11.250000	26.000000	7425.000000
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	16.170000	38.000000	42521.000000
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	49.830000	49.000000	123071.000000

Table 3.15: Descriptive Statistics Summary

From the summary statistics table in Table 3.15, it was revealed that the mean flight duration was 12.22 hours, indicating that flights, on average, lasted around 12.22 hours. The standard deviation of 7.19 hours suggested significant variation inflight durations around the mean. The mean number of days left until the flight was 26 days, showing that flights were typically booked 26 days in advance. The standard deviation of 13.56 days showed considerable variation in booking times. The mean price was INR 20,882.29, suggesting that the average flight fare was approximately INR 20,882.29. The standard deviation of INR 22,694.73 reflected a high variability in flight prices.

The median and mode of the quantitative and qualitative features in the dataset were also calculated using their respective ‘median()’ and ‘mode()’ functions on the numerical columns stored in the ‘num_cols’ variable.

Feature Name	Median Value
Flight Duration	11.25
Flight Days Left	26.00
Price	7425.00

Table 3.16: Median Values of Quantitative Features

According to the median values in Table 3.16, the median flight duration was 11.25 hours, meaning half of the flights lasted 11.25 hours or less, and the other half lasted 11.25 hours or more. The median number of days left until the flight was 26 days, indicating that flights were

typically booked 26 days in advance. The median flight fare was INR 7,425, showing that half of the flight fares were INR 7,425 or lower, while the other half were higher.

Feature Name	Mode Value
Airline Name	Vistara
Flight Number	UK-706
Seat Class	Economy
Departure City	Delhi
Departure Time	Morning
Number of Stops	One
Destination City	Mumbai

Table 3.17: Mode Values of Qualitative Features

For the qualitative features, the mode values in Table 3.17 showed that Vistara was the most frequent airline, UK-706 was the most common flight number, and economy class was the most booked seat class. The most frequent departure city was Delhi, while morning being the most common departure time. One-stop flights were the most common, and Mumbai was the most frequent destination city, with night being the most common arrival time.

Skewness was calculated for the quantitative features using the 'skew()' function from the Pandas library, which measures the asymmetry of numerical columns. The features considered were flight duration, flight days left, and price. Skewness values provided insights into feature distribution, which is crucial for data preprocessing and model building.

Features	Skewness Values
Flight Duration	0.594935
Flight Days Left	-0.035948
Price	1.063585

Table 3.18: Skewness of Quantitative Features

According to the skewness values in Table 3.18, flight duration had a moderate positive skew (0.594935), indicating a higher concentration of shorter flights with a few longer ones. The flight days left had a new symmetric distribution (-0.035948), meaning that the number of days left until the flight was evenly spread. The price displayed a high positive skew (1.063585), meaning most flight prices were lower, with a few significantly higher priced.

3.3.2 Quantitative Variables Distribution Analysis

Histograms were generated to visualise the distribution of quantitative variables in the dataset. Using the ‘plt.hist()’ function from the Matplotlib library, numeric columns (num_cols) were iterated through, and for each column, a subplot was created. A histogram with 10 bins was plotted, with the x-axis labelled by the column name and the y-axis labelled 'Frequency'.

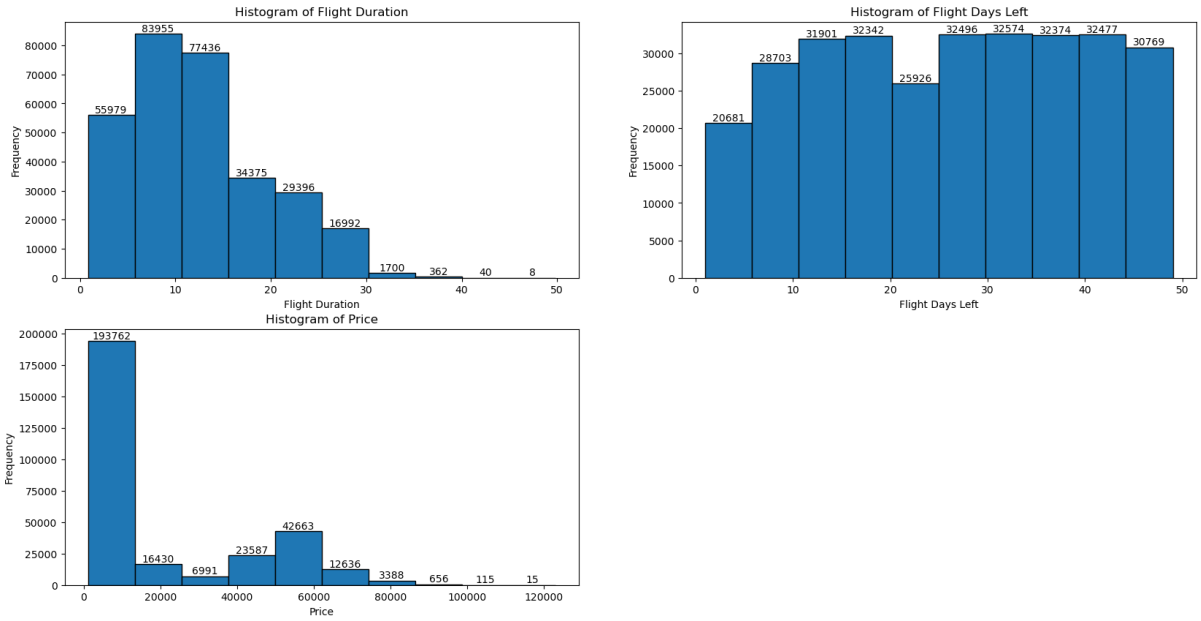


Figure 3.2: Histograms of Quantitative Variables

The histograms in Figure 3.2 revealed several key insights. Most flights lasted between 5 and 15 hours, indicating a typical operational range, with fewer long-haul flights lasting between 15 and 25 hours. Only a few flights exceeded 25 hours, suggesting a limited number of long-haul flights. The availability of flights was generally balanced throughout the booking window, with a slight dip around 15-25 before departure, stabilising after 25 days This indicates that flights remain available constantly up to 50 days in advance. Flight fares were mostly priced between INR 0 and 20,000, with a small peak around INR 60,000, indicating a market for higher-priced flights, with a very few flights exceeded INR 80,000.

3.3.3 Associations Between Quantitative Features and Price

Scatter plots were generated to visualise the relationship between quantitative variables ('Flight Duration' and 'Flight Days Left') and airfare, with points coloured by 'Seat Class'. For each

variable, a scatterplot was created using Seaborn's library. The x-axis represented the quantitative variable, and y-axis represented the mean price. The hue parameter was used to differentiate seat classes. Each plot was titled accordingly, and the axes were labelled appropriately.

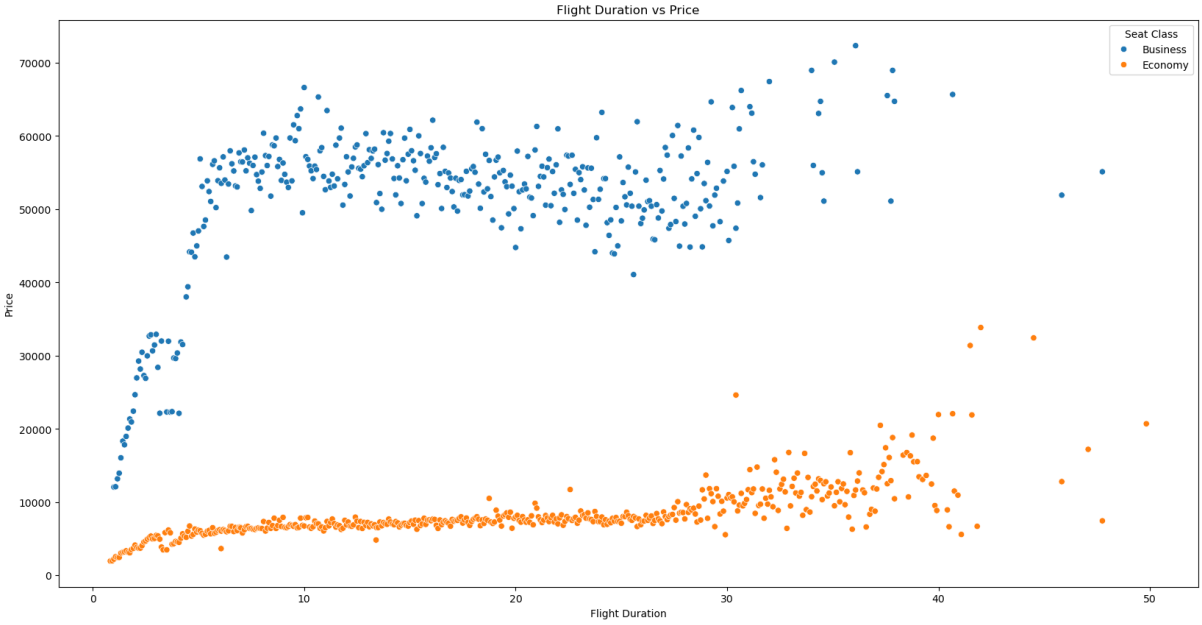


Figure 3.3: Scatter Plots of Flight Duration vs Price by Seat Class

The scatter plot in Figure 3.3 of flight duration and price showed a clear positive correlation, with prices tending to rise as flight duration increased. This trend was more obvious in business class, where prices significantly increased with longer flight durations, while economy class prices rose more gradually. There was some overlap between seat classes at longer durations, suggesting that factors beyond flight duration, such as airline and destination, influenced airfare. Though the plot did not definitely show that business class fares were always higher than economy class for the same flight duration, it confirmed that business class tickets typically occupied the higher price range.

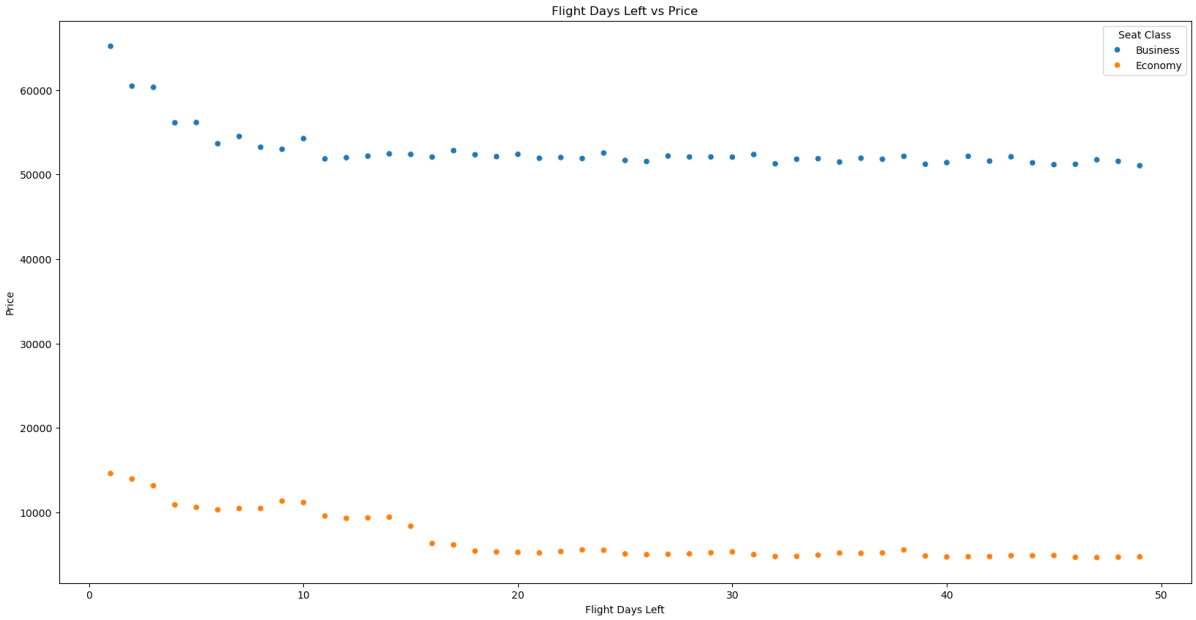


Figure 3.4: Scatter Plots of Flight Days Left vs Price by Seat Class

The scatter plot in Figure 3.4 of flight days left and price revealed a moderate negative correlation, indicating that as the number of days left until departure decreased, ticket prices tended to increase. This trend was more noticeable in economy class, where prices gradually declined with more days left, while business class prices remained high with less variation. Although this plot alone cannot account for all factors affecting price, it clearly demonstrated that prices generally rise as the departure date approaches.

3.3.4 Distribution of Qualitative Variables

Count plots were employed to visualise the distribution of qualitative variables in the dataset. Using the ‘countplot()’ function in the Seaborn library, categorical columns stored in the ‘cat_cols’ variable were iterated through, and for each column, a subplot within a 3x3 grid was created. A count plot with annotations displaying the frequency of each category was plotted, and the x-axis was labelled with the column name and the y-axis was labelled 'Frequency'.

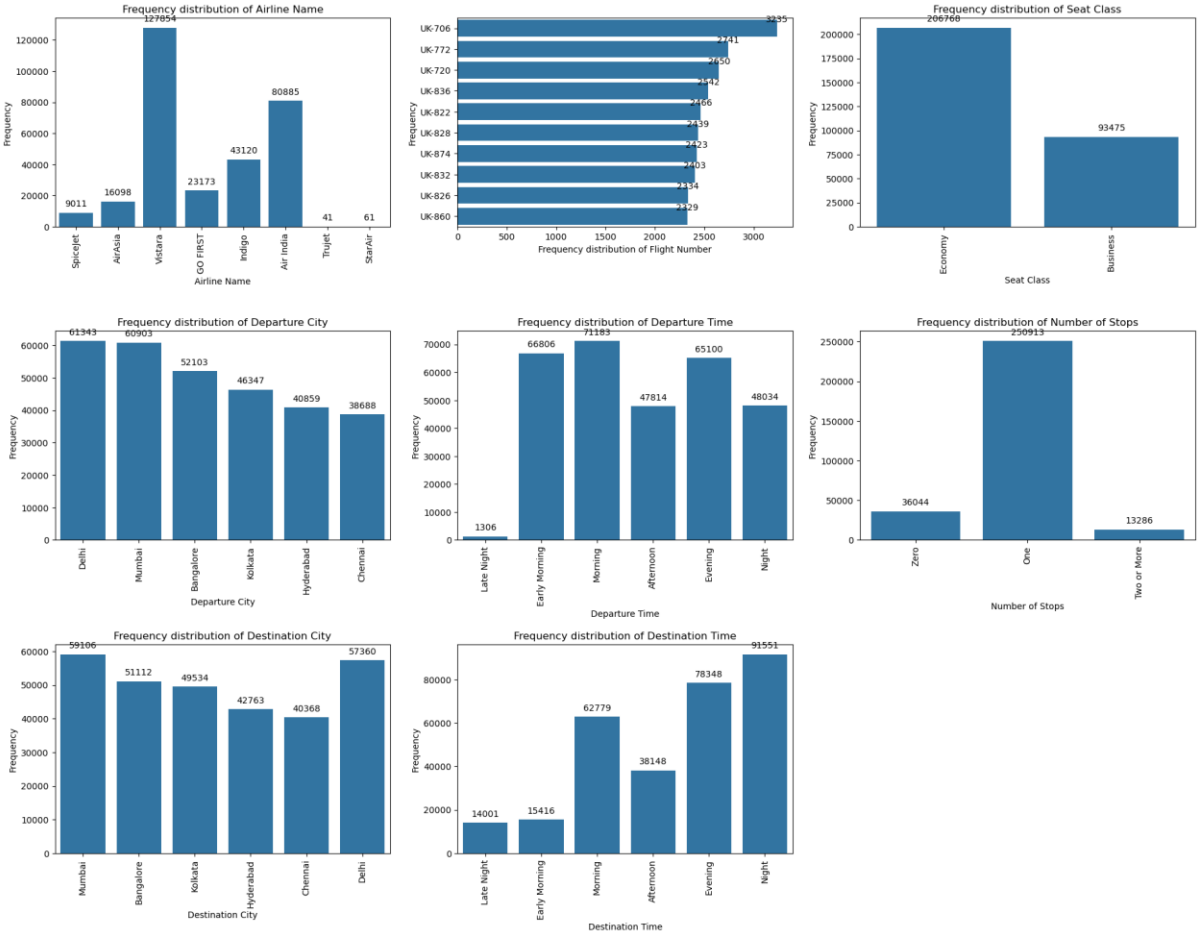


Figure 3.5: Frequency Distributions of Qualitative Features

According to Fig 3.5, Air India and Vistara dominated the market thanks to their broad networks and strong reputations, while SpiceJet and AirAsia had smaller shares, likely due to regional focus. Trujet and StarAir operated on a smaller scale, targeting niche markets. Flight number distributions showed clustering within certain ranges, indicating popular routes or busy schedules. Economy class was booked more frequently, reflecting customers' preference for cheaper options. Delhi and Mumbai emerged as major travel hubs, with Chennai showing fewer departures and arrivals, suggesting lower travel demand. Morning and early morning departures were more popular, whereas late-night departures were less in demand. One-stop flights were more common, likely due to cost considerations, while non-stop flights were less frequent. Night and evening arrivals were preferred among travellers, while late night arrivals were less common, likely due to operational constraints.

3.3.5 Associations Between Qualitative Features and Prices

Bar plots were created to explore the relationship between qualitative variables and price. For each variable, a bar plot was generated using Seaborn's bar plot function, displaying mean price for each category. The x-axis represented the qualitative variable, and the y-axis the mean price. The x-axis labels were rotated 90 degrees for better readability, and bars were annotated with their height value, representing the mean price.

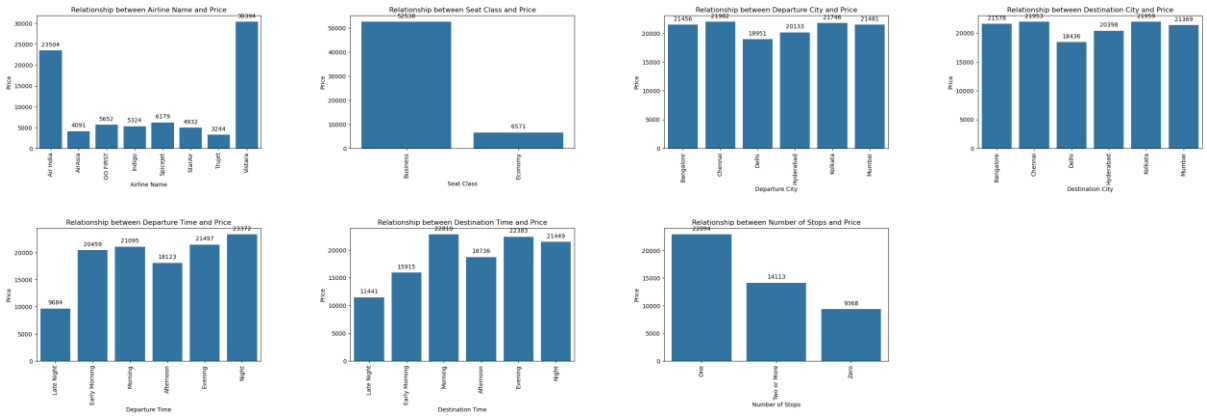


Figure 3.6: Bar Plots of Qualitive Features vs Price

The analysis in Fig 3.6 showed that Vistara had the highest average price, indicating it as a premium airline, while TruJet, AirAsia, and Star Air had the lowest prices, suggesting they were budget carriers. Business class fares were significantly higher than economy class, reflecting the additional amenities offered. Flights departing from Chennai and Kolkata were generally more expensive, while Delhi had lower fares, likely due to higher flight availability. Late-night departing flights were the cheapest, whereas night flights are the most expensive due to passenger preference. Morning arrivals had the highest prices, while late-night arrivals were the cheapest because of passenger's preference of reaching the destination during daylight. Non-stop flights tended to the least expensive than one-stop flights, likely due to a cost of additional services.

3.4 Data Transformation

Data transformation is a vital step in a machine learning project, where raw data is converted into a format more suitable for modelling. This process includes techniques such as data splitting, encoding, and scaling to ensure that machine learning algorithms can effectively learn

from the data. The goal of data transformation is to improve data quality, making it more consistent and easier for the model to interpret, ultimately improving performance and accuracy.

3.4.1 Data Splitting

Stratified Shuffle Split is a cross-validation technique that ensures each fold of the dataset maintains the same proportion of each class as the original dataset (GeekforGeeks, 2024). This method is particularly effective for handling imbalanced datasets, where certain classes are underrepresented. In the context of flight fare prediction system, this technique ensured that both the training and test sets maintain a similar distribution of the target variable, which is crucial for preserving the representativeness of the data. This method reduces bias and variance, providing a more reliable evaluation of the model's performance and ensuring it generalises well to unseen data.

The procedure began by calculating the ANOVA F-statistic and p-value for categorical columns to determine their significance in predicting flight prices. ANOVA (Analysis of Variance) is a statistical method used to compare the means of three or more groups to determine whether the differences between the means are significant (Kaufmann and Schering, 2014). The F-statistic compares the variance between group means to the variance within the groups, and a higher F-statistic indicates that the group means are significantly different from each other. The p-value shows how likely it is that any observed differences happened by chance. A lower p-value (typically less than 0.05) suggests that the differences are statistically significant, meaning they are unlikely to be occurred randomly.

Columns	F-Statistic Value	P-Value
Seat Class	2193455.0795171554	0.0
Airline Name	12294.993974923978	0.0
Number of Stops	6484.497981981782	0.0
Destination Time	883.957139765687	0.0
Departure Time	337.89053626304195	0.0

Table 3.19: Feature Ranking by F-Statistic

From the table 3.19, it is evident that all the categorical columns have extremely high F-statistic values and p-values of 0.0, indicating that the differences between the group means are highly significant. ‘Seat Class’ emerged as the most significant feature, followed by ‘Airline Name’ and ‘Number of Stops’ on the second and third place, respectively.

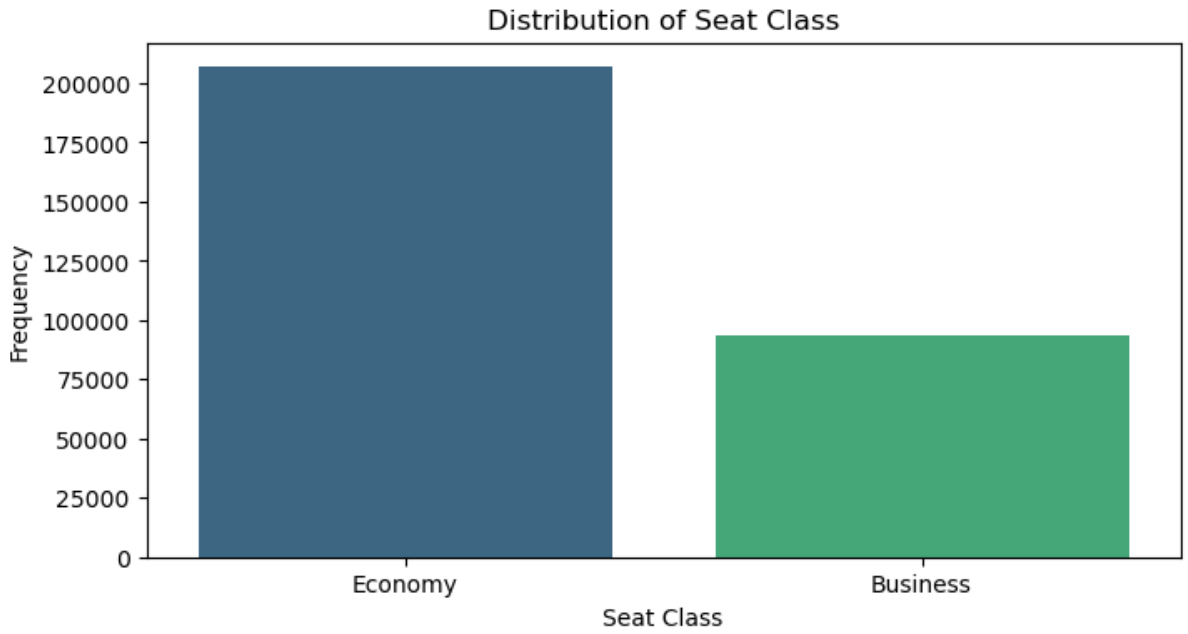


Figure 3.7: Dataset Split Using Stratified Shuffle Split

The distribution of "Seat Class" was analysed in Figure 3.7, with 68.87% of the entries in Economy class and 31.13% in Business class. The dataset was split into training and test sets using the 'StratifiedShuffleSplt()' method from 'sklearn' library, ensuring that the distribution of 'Seat Class' was preserved across both train and test sets, with 'Price ' as the target variable for prediction.

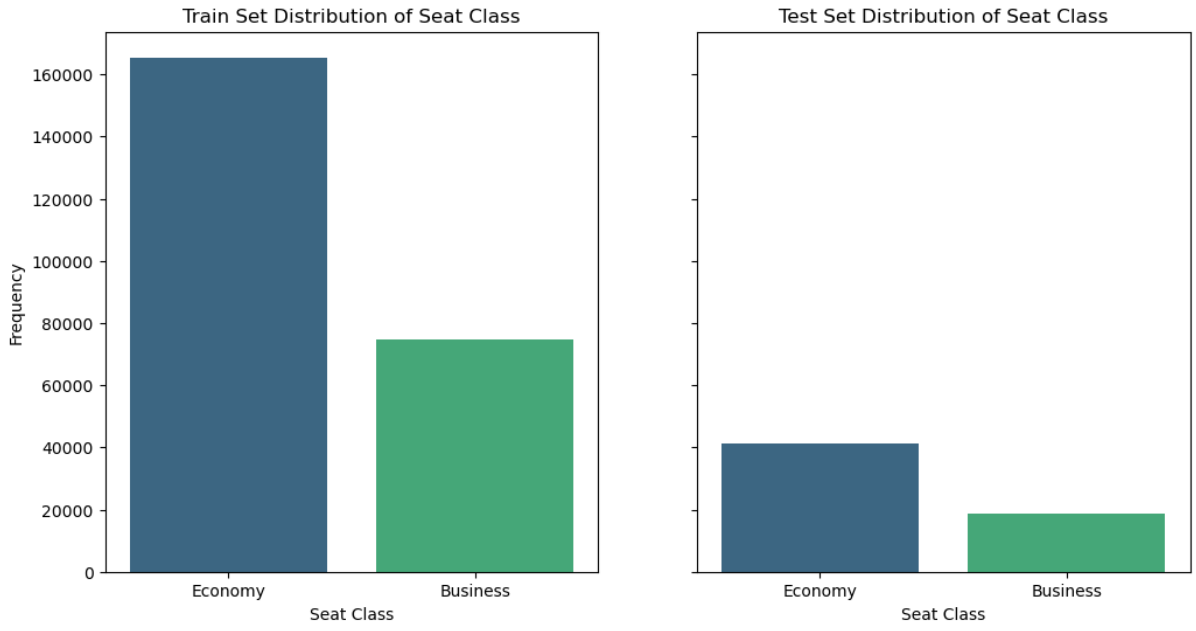


Figure 3.8: Seat Class Frequency Using Stratified Shuffle Split

After splitting of the whole dataset as per Figure 3.8, 68.87% of the data belonged to economy class and 31.13% to business class, with the same distribution maintained in both training and test sets. After splitting, the dataset was divided into features (X) and target (y), with 240,194 entriesand 60,049 entries in the test sets. This process ensured the dataset was well-stratified and ready for model building and evaluation.

3.4.2 Data Encoding

Data encoding involves converting categorical features into numerical format so machine learning algorithms can process them. Although there are numerous encoding methods, however, Ordinal Encoding and Target Encoding were used for this project.

- **Ordinal Encoding**

  Ordinal encoding converts ordinal categorical data into numerical format, preserving the meaningful order among categories (Albon and Gallatin, 2023). This is crucial for features where the order carries important information. In this flight fare prediction system, features such as 'Seat Class', 'Departure Time', 'Destination Time', and 'Number of Stops' have an inherent order. For instance, 'Seat Class' can be ranked from 'Business' to 'Economy', and 'Number of Stops' can range from 'Zero' to 'Two or More'. Ordinal Encoding ensures this order is preserved, helping the model understand the relative importance of these categories.

  In this project, ordinal categorical columns, such as 'Seat Class', 'Departure Time', 'Destination Time', and 'Number of Stops' were identified. Categories for each column were defined, and ‘OrdinalEncoder()’ function was used to transform both the training and test sets, ensuring consistency in encoding process across.

- **Target Encoding**

  Target encoding converts categorical data into numerical format by replacing each category with the mean of the target variable for that category (Galli, 2020). This method captures the relationship between categorical features and the target variable, potentially increasing model performance. In this project, target encoding was applied to nominal categorical columns such as 'Airline Name', 'Departure City', and 'Destination City', as it directly uses the target variable to encode categories, capturing the predictive power of each.

In implementing target encoding for the flight fare prediction system, the nominal categorical columns, including 'Airline Name', 'Flight Number', 'Departure City', and 'Destination City', were first listed. A ‘TargetEncoder()’ function was then used to transform the relevant nominal columns in both the training and test sets into numerical format based on the mean of the target variable for each category, ensuring consistency in the encoding process across both datasets.

3.4.3 Data Scaling

Data scaling adjusts the range of data so that different features are on a comparable scale, which is important for algorithms that rely on distance measurements, such as K-Nearest Neighbours (KNN) and Linear Regression.

- **Robust Scalar**

  Robust Scaler is a scaling technique that normalises data using the median and the interquartile range (IQR), making it less sensitive to outliers (Alsdorf and Korner, 2022). Thiis algorithm was chosen over other techniques because the flight dataset contained outliers from factors like last-minute bookings or special discounts. The Robust Scaler ensures that these outliers do not disproportionately affect the scaling process.

  In implementing the Robust Scaler for this project, it was applied to both the training and test sets, ensuring consistent scaling across both datasets.

3.5 Data Modelling

The model implementation process involves evaluating and comparing various machine learning models to determine the best fit for the data and problem requirements. This process involves selecting a range of models, training each model on the training dataset, and measuring their performance using specific evaluation metrics. These metrics are important in providing a fair comparison of the models based on their ability to predict the target variable. The aim is to select a model that generalises well to new, unseen data, ensuring accurate and reliable predictions.

A few evaluation metrics are used to gauge the model’s performance, covering aspects such as accuracy, error, and overall predictive capability. These metrics include:

- **Accuracy and R^2 scores:**

  Accuracy is generally applied in classification models, measuring the proportion of correctly predicted instances. For regression models, like linear regression, the R^2 score (coefficient of determination) is used. R^2 indicates how well the independent variables predict the dependent variable, with higher values representing better performance. This score helps determine the model's capacity to explain variance in the data, with values closer to 1 indicating better performance. The formula for R^2 in Equation (3.4) is:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.4}$$

  The numerator $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ represents the residual sum of squares (RSS), while the denominator $\sum_{i=1}^{n}(y_i - \bar{y})^2$ represents the total sum of squares (TSS). Subtracting the ratio of RSS to TSS from 1 gives the R^2 score, which ranges from 0 to 1. Higher values indicate better model performance (Frost, no date).

- **Mean Absolute Error (MAE):**

  Mean Absolute Error calculates the average of the absolute differences between predicted and actual values, providing an indication of how far predictions are, on average, from the actual values. The mathematical formula in Equation (3.5) for this metric as described by Mishra (2018) is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.5}$$

  Brownlee (2021) described that the numerator ($\sum_{i=1}^{n}|y_i - \hat{y}_i|$) represents the sum of absolute differences, while the denominator (n) is the total number of observations. MAE provides a clear measure of prediction accuracy by averaging these absolute errors.

- **Mean Squared Error (MSE):**

  Miller (2018) described Mean Squared Error as the average of the squared differences between predicted and actual values, emphasising larger errors due to squaring. This is

particularly useful for identifying models with significant outliers. He further mentioned the mathematical formula in Equation (3.6) for this metric as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.6}$$

The numerator $(\sum_{i=1}^{n}(y_i - \hat{y}_i)^2)$ represents the sum of squared differences, while the denominator (n) is the total number of observations. MSE measures the average size of errors, with larger errors having a greater impact due to the squaring.

- **Root Mean Squared Error (RMSE):**

  Root Mean Squared Error is the square root of Mean Squared Error, which makes it more interpretable, presenting errors in the same units as the target variable. The formula in Equation (3.7) as described by Hodson (2022) is:

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{3.7}$$

  In the equation above, RMSE is the square root of Mean Squared Error.

- **Mean Absolute Percentage Error (MAPE):**

  Mean Absolute Percentage Error offers a relative measure of error by expressing the absolute error as a percentage of the actual values, as explained by De Myttenaere et al. (2016). The formula for MAPE in Equation (3.8) is:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100 \tag{3.8}$$

  The numerator $\left(\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|\right)$ represents the sum of absolute percentage differences, while the denominator (n) is the total number of observations. MAPE provides a relative measure of prediction accuracy, expressed as a percentage.

3.5.1 Model Implementation

The following models were employed for the project: Linear Regression (LR), Decision Tree Regression (DTR), Random Forest Regression (RFR), K-Nearest Neighbours Regression (KNNR), and Gradient Boosting Regression (GBR). These models were chosen to represent a range of techniques, from simple linear models to more complex approaches, ensuring the most suitable model for predicting flight fares.

3.5.1.1 Linear Regression

Linear regression is a straightforward statistical method used to model the relationship between a dependent variable and one or more independent variables, as described by Kavita (2024). The objective is to find the best-fitting regression line that minimises the difference between the predicted and actual values of the dependent variable.

Linear regression model uses a mathematical formula which is in Equation (3.9) to estimate the relationship between the dependent variable (y) and the independent variables (X). This equation takes the form of a linear equation, which is described below:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon \tag{3.9}$$

In the equation, (β_0) represents the intercept, $(\beta_1, \beta_2, \ldots, \beta_n)$ are the coefficients that correspond to the independent variables, and (ϵ) is the error term, accounting for the model's imperfections. The objective through this formula is to identify the set of coefficients that minimises the sum of the squared differences between the observed predicted values. Linear regression is widely favoured to its simplicity, interpretability, and efficiency with large datasets, making it particularly useful for predicting continuous outcomes Deepanshi (2023).

In implementing this model for the project, a pipeline was created using the 'Pipeline()' function from the 'sklearn' library, with 'LinearRegression()' as the estimator. The model was trained on the training data ('X_train' and 'y_train'), and predictions were made using the test data ('X_test' and 'y_test'). Performance was evaluated using metrics such as the R^2 score, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE).

3.5.1.2 Decision Tree Regression

Decision tree regression is a non-parametric, supervised learning method designed to predict continuous outcomes. The algorithm works by splitting the data into subsets based on input feature values, forming a tree structure of decisions. Each internal node represents a decision on a feature's value, which each leaf node contains the predicted outcome (Gupta, 2017).

In contrast to linear regression, decision tree regression does not follow a single formula. However, the prediction of any given input can be expressed as the average of the target variable within the terminal node (leaf) that the input falls into. As described by Hastie et al. (2009), the prediction process for this model can be described mathematically in Equation (3.10) as:

$$\hat{y}_i = \frac{1}{|R_m|} \sum_{X_j \in R_m} y_j \tag{3.10}$$

Where Rm is the region (leaf) in which Xi resides, $|\ Rm\ |$ is the number of data points in that region Rm, and y_j are the observed target values in Rm.

To implement the decision tree model for the project, the maximum tree depth was limited to 2 to avoid overfitting and improve interpretability. A pipeline was established using the 'Pipeline()' function, with the 'DecisionTreeRegressor()' estimator. This model was trained using the training dataset, and predictions were generated on the test dataset. Model performance was assessed R^2 scores and other error metrics, including Mean Absolute Error, Root Mean Squared Error, Mean Squared Error, and Mean Absolute Percentage Error.

3.5.1.3 Random Forest Regression

Random Forest Regression is an algorithm that builds multiple decision trees during training and outputs the average prediction of the individual trees to enhance predictive accuracy and mitigate overfitting (Breiman, 2001). Unlike decision trees, where a single tree might overfit the data, random forests reduce variance and improves the model's generalisation capabilities by aggregating predictions across many trees (Shafl, 2023).

The formula Equation (3.11) is for prediction in Random Forest Regression is the average of the predictions from all individual trees:

$$\hat{y} = \frac{1}{N}\sum_{i=1}^{N}\hat{y}_i \tag{3.11}$$

In the above equation, N represents the number of trees, and $(\widehat{y_i})$ is the prediction from the (i)-th tree. By combining the outputs of multiple decision trees, the random forest model reduces the risk of overfitting and enhances predictive accuracy, especially in the presence of non-linear relationships.

In this project, the pipeline was implemented using a pipeline function from the 'sklearn' library with 'RandomForestRegressor()' as the estimator. The maximum depth for each tree was set to 2 to prevent overfitting. The pipeline was fitted to the training data and predictions were calculated on the test set. The model's performance was evaluated using R^2 score, and all the relevant error metrics were calculated, including Mean Absolute Error, Root Mean Squared Error, Mean Squared Error, and Mean Absolute Percentage Error.

3.5.1.4 K-Nearest Neighbours Regression

K-Nearest Neighbours (KNN) Regression is a non-parametric approach where the target variable is predicted by averaging the values of its k-nearest neighbours (Schott, 2019). This averaging process helps in smoothing out the predictions and reducing the impact of noise in the data. The proximity of the neighbours is typically determined using a distance metric, most commonly the Euclidean distance.

The formula in Equation (3.12) is for calculating Euclidean distance in this model can be calculated from the mathematical equation given below:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{3.12}$$

In this equation, (p) and (q) are two data points, and the sum calculates the squared difference between each dimension. This distance metric helps identify the k-nearest neighbours, whose target values are then averaged to make the final prediction.

In this project, the number of neighbours (k) was set to 100 for the model. A pipeline using the pipeline class was created with the 'KNeighboursRegressor()' estimator and fitted to the training data, allowing the model to learn the underlying patterns. The model's performance was assessed using R^2 scores and error metrics on both the training and test datasets. Additionally, error metrics such as Mean Absolute Error, Root Mean Squared Error, Mean Squared Error, and Mean Absolute Percentage Error were calculated.

3.5.1.5 Gradient Boosting Regression

Gradient Boosting Regression is a technique that builds a predictive model in stages, typically from decision trees, to correct errors from the previous models (Singh, 2018). Each model in the sequence aims to reduce a loss function, often the mean squared error, for regression tasks. The formula in Equation (3.12) is for the prediction in this model can be expressed as:

$$\hat{y} = F_0(x) + \sum_{m=1}^{M} \nu \cdot h_m(x) \tag{3.12}$$

In this formula, $(\hat{y})$ represents the predicted value, $\left(F_{0(x)}\right)$ is the initial prediction (often the mean of the target values), (M) is the total number of trees, (ν) is the learning rate, and $\left(h_{m(x)}\right)$ is the prediction from the (m)-th tree for input (x) (Friedman, 2001). This iterative approach minimises a specified loss function, improving both accuracy and robustness over time.

Considering model's implementation for this project, a pipeline was constructed using 'GradientBoostingRegressor()' as the estimator with multiple estimators as five, and a learning rate of 0.4 to balance model complexity and learning speed. The model was trained and evaluated using the R^2 score other metrics, including Mean Absolute Error, Root Mean Squared Error, Mean Squared Error, and Mean Absolute Percentage Error.

3.5.2 Hyperparameter Tuning

According to Paul (2018), hyperparameter tuning is an essential step in the machine learning pipeline, involving the selection of most favourable hyperparameters to enhance the performance of a learning algorithm. Hyperparameters, such as the learning rate, number of trees in a random forest, or the number of neighbours in K-Nearest Neighbours, are set before the training process. The aim is to find the most effective combination of hyperparameters that yields the best performance on a validation set, improving the model's generalisation ability. Methods like grid search, random search, and more advanced techniques like Bayesian optimisation are commonly used for this purpose.

For this project, Randomised Search CV was chosen as the hyperparameter optimisation technique due to its efficiency in exploring a broad range of hyperparameter values with fewer iterations compared to Grid Search CV. This approach was particularly beneficial for handling larger datasets and complex models, as discussed by Arindam (2022). Randomised Search CV was applied to all five models to increase their performance by optimising hyperparameters. The process involved evaluating random samples from predefined parameter distributions and identifying the best-performing combination. The optimised models were then assessed using performance metrics.

3.5.2.1 Linear Regression (Ridge)

As explained by Zhu (2024), ridge regression was selected due to its effectiveness in handling multicollinearity, which is important when features in the data are correlated. By introducing a regularisation term, this model helps reduce overfitting by penalising large coefficients. This ensures that the model remains simple while being robust for the unseen data.

For this project, the key hyperparameters used for Ridge Regression were ‘fit_intercept’, ‘alpha’, and ‘solver’. The ‘fit_intercept’ parameter controls whether the model should calculate the intercept (when ‘True’) or set it to zero (when ‘False’). The ‘alpha’ parameter adjusts the strength of the regularisation, with larger values applying stronger regularisation, helping avoid overfitting while potentially reducing accuracy. The ‘solver’ hyperparameter specifies the algorithm used to optimise the linear system, with options such as ‘auto’, ‘svd’, and ‘lsqr’ to influence the model's performance. Table 3.20 shows the hyperparameters and their ranges used for linear regression.

Hyperparameter	Description	Range
fit_intercept	Whether to calculate the intercept or not	[True, False]
alpha	Regularisation strength	[1e-15, 100]
solver	Algorithm used to solve the linear system	[‘auto’, ‘svd’, ‘lsqr’]

Table 3.20: Linear Regression Hyperparameters

The implementation of Randomized Search Cross-Validation ‘RandomizedSearchCV()’ for Ridge Regression started by defining a dictionary of key hyperparameters: ‘fit_intercept’, ‘alpha’, and ‘solver’. An instance of the Ridge Regression model was then created, and ‘RandomizedSearchCV()’ was applied to randomly sample hyperparameter combinations, making the process more efficient than an exhaustive grid search. The search was set to perform 10 iterations with 5-fold cross-validation, using all available processors to improve speed and efficiency. After fitting the model to the training data, the results were stored in a data frame, sorted by test scores to identify the optimal hyperparameters. A line plot was created to compare the mean training and test scores across different hyperparameter combinations, offering a visual representation of the model’s performance. With the best hyperparameters identified, a new Ridge Regression model was configured and retrained on the training data. The model was then evaluated using both training and test datasets, with error metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE) calculated to determine the positive impact of hyperparameter tuning on the model’s performance.

3.5.2.2 Decision Tree Regression

Key hyperparameters for Decision Tree Regression included ‘max_depth’, ‘min_samples_split’, and ‘min_samples_leaf’. The ‘max_depth’ parameter limits how deep the tree can grow, preventing overfitting by constraining the complexity. The range for this parameter was tested between 2 and 10. The ‘min_sample_split’ parameter determines the minimum number of samples needed to split a node, with values from 2 to 20 explored, controlling the splitting behaviour and ensuring sufficient data is available at each node. The ‘min_samples_leaf’ parameter sets the minimum number of samples needed to be present in a leaf node, with a range of 1 to 8, ensuring that smaller leaves are pruned, which improves generalisation. Table 3.21 shows the hyperparameters and their ranges for the decision tree regression model.

Hyperparameter	Description	Range
max_depth	Maximum depth of the tree	[2, 10]
min_samples_split	Minimum samples required to split an internal node	[2, 20]
min_samples_leaf	Minimum samples required to be at a leaf node	[1, 8]

Table 3.21: Decision Tree Regression Hyperparameters

The hyperparameter tuning for Decision Tree Regression involved defining a parameter grid consisting of 'max_depth', 'min_samples_split', and 'min_samples_leaf'. An instance of the Decision Tree Regressor was then created, and 'RandomizedSearchCV()' was applied to sample hyperparameter combinations randomly. This made the process more efficient than exhaustive grid search. The search performed 10 iterations with 5-fold cross-validation. After training the model on the dataset, the results were stored in a data frame and sorted by test scores to find the best hyperparameters. A line plot visualised the mean training and test scores across different combinations, helping to compare performance. Once the best hyperparameters were identified, a new Decision Tree model was configured and fitted to the training data. The model's performance was evaluated using training and test datasets, and metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE) were calculated to assess the model's improvements.

3.5.2.3 Random Forest Regression

Key hyperparameters for Random Forest Regression included 'n_estimators', 'max_depth', 'min_samples_split', and 'min_samples_leaf'. The 'n_estimators' parameter controls the number of decision trees in the forest, with a range of 100 to 500 tested. More trees usually result in better performance but at a higher computational cost. The 'max_depth', 'min_samples_split', and 'min_samples_leaf' parameters have the same function as in Decision Tree Regression, limiting the tree's depth and controlling splitting behaviour to avoid overfitting. Table 3.22 lists the number of hyperparameters, and the ranges applied for this model.

Hyperparameter	Description	Range
n_estimators	Number of trees in the forest	[100, 500]
max_depth	Maximum depth of the tree	[2, 10]
min_samples_split	Minimum samples required to split an internal node	[2, 20]

Hyperparameter	Description	Range
min_samples_leaf	Minimum samples required to be at a leaf node	[1, 8]

Table 3.22: Random Forest Regression Hyperparameters

The implementation of Randomized Search Cross-Validation for Random Forest Regression began by defining a parameter grid for ‘n_estimators’, ‘max_depth’, ‘min_samples_split’, and ‘min_samples_leaf’. After creating an instance of the Random Forest Regressor, ‘RandomizedSearchCV()’ was used to randomly sample different combinations of hyperparameters, making it more efficient than grid search. The search was configured to perform 10 iterations with 5-fold cross-validation. The model was then trained on the dataset, and results were stored in a data frame and sorted by test scores to identify the optimal hyperparameters. A line plot compared mean training and test scores across different hyperparameter combinations. After identifying the best hyperparameters, a new Random Forest model was configured and retrained on the training data. Performance was evaluated using both training and test datasets, with metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE) calculated to determine the improvement in model accuracy.

3.5.2.4 K-Nearest Neighbours Regression

The key hyperparameters for KNN Regression included ‘n_neighbors’, ‘weights’, and ‘metric’. The ‘n_neighbors’ parameter determines how many nearest points are considered when making predictions, with values ranging from 1 to 20 explored. The ‘weights’ specifies whether each neighbour should contribute equally to the prediction (uniform) or be weighted by distance (distance). The ‘metric’ parameter defines how distance between points is calculated, with options like Euclidean and Manhattan being tested. Table 3.23 specifies the list of hyperparameters and their ranges tests on the model.

Hyperparameter	Description	Range
n_neighbors	Number of nearest neighbours	[1, 20]
weights	Weighting function for neighbours	['uniform', 'distance']
metric	Distance metric for neighbour calculation	['euclidean', 'manhattan']

Table 3.23: K-Nearest Neighbours Hyperparameters

The process of hyperparameter tuning for KNN Regression began by defining a parameter grid for ‘n_neighbors’, ‘weights’, and ‘metric’. An instance of the KNN Regressor was created, and ‘RandomizedSearchCV()’ was applied to randomly explore hyperparameter combinations. The search was set for 10 iterations with 5-fold cross-validation. The model was fitted to the dataset, and the results were saved in a data frame and sorted by test scores to identify the optimal hyperparameters. A line plot was used to compare the mean training and test scores across different hyperparameter combinations, which visualised performance. After identifying the best hyperparameters, a new KNN model was configured and retrained on the training data. The model's performance was evaluated on both the training and test datasets, and metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE) were calculated to measure predictive accuracy.

3.5.2.5 Gradient Boosting Regression

Key hyperparameters for Gradient Boosting Regression included ‘n_estimators’, ‘max_depth’, and ‘learning_rate’. The ‘n_estimators’ parameter controls the number of boosting stages, with a range of 100 to 500 tested. More stages can improve performance but may lead to overfitting if not controlled by the learning rate. The ‘max_depth’ parameter limits tree depth, with a range of 2 to 10 tested to balance complexity. The ‘learning_rate’ controls how much each tree contributes to the final model, with smaller values in the range of 0.01 to 0.2 helping to prevent overfitting by scaling the contribution of each tree. Table 3.24 list the hyperparameters and their ranges for the model.

Hyperparameter	Description	Range
n_estimators	Number of boosting stages	[100, 500]
max_depth	Maximum depth of the trees	[2, 10]
learning_rate	Step size for each tree's contribution	[0.01, 0.2]

Table 3.24: Gradient Boosting Regression Hyperparameters

Hyperparameter tuning for Gradient Boosting Regression started by defining a parameter grid for ‘n_estimators’, ‘max_depth’, and ‘learning_rate’. An instance of the Gradient Boosting Regressor was created, followed by applying ‘RandomizedSearchCV()’ to sample different hyperparameter combinations. This search was performed for 10 iterations with 5-fold cross-validation. After training the model on the dataset, results were stored in a data frame and

sorted by test scores to identify the best hyperparameters. A line plot compared the mean training and test scores across various hyperparameter combinations. After identifying the best hyperparameters, a new Gradient Boosting model was configured and retrained on the training data. The performance was evaluated on both training and test datasets, and error metrics such as Mean Absolute Error.

3.5.3 Ensemble Modelling

Ensemble algorithms, as outlined by Dietterich (2000), are machine learning techniques that combine multiple models to improve prediction accuracy and robustness. These methods aggregate predictions from several base models, which may be either homogeneous (of the same type) or heterogeneous (or different types). Dietterich points out that ensemble methods address common machine learning issues such as high variance and overfitting by averaging out errors across models. Additionally, ensembles methods often generalise better to unseen data, making them highly effective in practice. The primary techniques for combining models as bagging, boosting, and stacking. Bagging reduces variance by training models on different data subsets and averaging their predictions, while boosting sequentially trains models to correct the errors of their predecessors, reducing bias. Stacking, on the other hand, trains a meta-model to combine predictions of several base models. These techniques generally lead to more accurate and robust predictions than using individual models.

The number of ensemble models were implemented in this project to combine different individual models to improve their performance.

3.5.3.1 Average Ensemble Model

The average ensemble model aggregates predictions from multiple base models to improve performance. Models such as Ridge Regression, Decision Tree Regression, Random Forest Regression, K-Nearest Neighbour Regression, and Gradient Boosting Regression were used in this project. Their predictions were averaged to produce the final output. By leveraging the strengths of each model while reducing their weaknesses, this method leads to more accurate and stable predictions (Bhatnagar, 2023).

In the flight fare prediction system, this method effectively captured a wide range of patterns in the flight fare data. The diverse base models allowed the ensemble to handle both linear and

non-linear relationships, which is essential for complex systems like airfares. A custom regressor was created, combining these multiple regression models, each optimised with the hyperparameters from cross-validation. The model was evaluated using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE). Additionally, individual predictions from each base model were visualised alongside the true values to evaluate the performance of the ensemble.

3.5.3.2 Voting Ensemble Model

Voting classifiers combine predictions from different models, which can be of different types. A decision rule, also known as voting scheme, aggregates the predictions from different models (Beyeler 2017). This can be done using two methods: hard voting and soft voting. In hard voting, the final prediction is based on the majority vote of the models (i.e., the model with the most votes determines the outcome), while in soft voting, the final prediction is the weighted average of the predicted probabilities or values from each model. In this project, soft voting was used, where weights were assigned to models based on their accuracy, meaning models with better performance had more influence on the final prediction (Bifet et al., 2018).

In the context of flight fare prediction, soft voting ensured that models with higher predictive accuracy contributed more to the final output, leading to more reliable predictions. Voting Regressor combined predictions from base models such as Ridge Regression, Decision Tree Regressor, Random Forest Regressor, K-Nearest Neighbours Regressor, and Gradient Boosting Regressor, each optimised through cross-validation. Weights were normalised based on the R^2 scores of each model to ensure that the total sum equalled to 1, The model's performance was evaluated using metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE). Additionally, visualisations were created to compare the individual predictions of each base model with the true values, highlighting the effectiveness of the ensemble approach.

3.5.3.3 Stacking Ensemble Model

Stacking is an advanced ensemble learning technique that combines multiple models to improve predictive performance. This method uses the predictions from base learners as input features for a meta-learner, which aggregates them to produce the final prediction (Kumar and Jain, 2020).

In this research project, Ridge Regression, Decision Tree Regressor, Random Forest Regressor, K-Nearest Neighbours Regressor, and Gradient Boosting Regressor served as base models, while a Linear Regression model acted as meta-learner, which was a Linear Regression model. This hierarchical approach allowed the system to capture complex patterns that individual models might overlook, thus improving predictive performance and robustness. This model evaluated using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE). Individual predictions of the base models were visualised alongside the true values to further assess performance.

3.5.4 Feature Contribution Analysis

Feature importance analysis is a technique in machine learning used to identify and quantify the contribution of each feature in a dataset to a model's predictive power. This analysis helps in understand which features are most influential in predicting the target variable, providing valuable insights into the data structure, and improving model interpretability (Shin, 2023). The importance of this technique lies in its ability to reveal the relationship between features and the target variable, like a correlation matrix, and identify irrelevant features. It also aids in improving model performance by enabling dimensionality reduction, where important features are retained, and less relevant ones are removed. This simplifies the model and enhances its efficiency. Furthermore, feature importance boosts model interpretability, making it easier to explain the model's decisions to stakeholders by highlighting the most significant features.

In the context of the flight fare prediction system, Shapley Additive Explanations (SHAP), developed by Lundberg and Lee (2017), were employed to assess feature importance. SHAP provides a method for interpreting predictions by assigning an importance value to each feature for a given prediction. Its key contributions lie in defining a class of additive feature importance measures, offering a unique solution with desirable theoretical properties. This framework unifies six pre-existing methods, addressing some limitations consistency and computational performance.

SHAP explanations can be classified into two categories: Global Explanations and Local Explanations (Goel, 2023).

- Global explanations provide an overview of the model's behaviour across the entire dataset. They highlight which features have the most significant overall impact on the model's predictions, helping to uncover general trends such as key factors influencing flight prices. In this project, a SHAP summary plot was generated to display the contribution of each feature, ranked by importance. Additionally, a bar chart was created to visualise global feature importance, allowing stakeholders to easily identify the primary drivers behind the model's fare predictions.

- Local explanations, in contrast, focus on individual predictions, clarifying how specific features influenced the model's output for a single instance or a small subset of instances. Methods such as SHAP or LIME are commonly employed to analyse how slight changes in input features affect the predictions. For this project, SHAP values were computed for individual instances in the test dataset, with waterfall and force plots used to visualise how each feature contributed to specific fare predictions. This approach ensures that the rationale behind the model's decision-making process is transparent, fostering greater trust in the fare predictions for individual flights.

The number of different techniques and methods discussed in this chapter, such as data preprocessing, feature engineering, and model training, has built a strong and systematic foundation for evaluating the performance of flight prediction models. Through hyperparameter tuning and the application of multiple machine learning algorithms, the models are now ready for detailed analysis. The next chapter will assess the performance by employing a few key indicators and visualisations to compare predictive accuracy, offering insights into the models' behaviour and the importance of various features in determining the factors that influence changes in the ticket prices.

Chapter 4 - Results and Analysis

This chapter discusses the performance of the models used for flight fare prediction, comparing their effectiveness before and after hyperparameter tuning. Building on the machine learning techniques and data preparation processes outlined in Chapter 3, key performance indicators such as R^2, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE) were employed to analyse the predictive abilities of each model. Additionally, scatterplots were used to provide a visual comparison of predictions versus actual values, both before and after tuning. The importance of a few individual features is analysed using different types of plots, including summary plots and force plots, to further understand model behaviour.

4.1 Model Performance

To evaluate the overall performance of each model, both tabular and visual results are examined, highlighting the impact of hyperparameter tuning. Additionally, the top five ranked performances based on test scores for each model, along with line plots of the mean test and training scores, are provided in Appendix C.

4.1.1 Performance Metrics

A tabular comparison of the performance and evaluation metrics before and after tuning is presented in Table 4.1, offering a concise view of how each model has been improved:

Model	R^2 Train Score	R^2 Test Score	Mean Absolute Error	Root Mean Squared Error	Mean Squared Error	Mean Absolute Percentage Error
Linear Regression	0.909680	0.909058	4564.688168	6854.943182	4.699025e+07	44.268569
Linear Regression CV	0.909680	0.909057	4564.649866	6854.957420	4.699044e+07	44.264048
Decision Tree Regressor	0.923861	0.923509	3866.236298	6286.741673	3.952312e+07	31.805208
Decision Tree Regressor CV	0.973310	0.971723	2121.484646	3822.432476	1.461099e+07	15.396939
Random Forest Regressor	0.923863	0.923511	3866.187228	6286.670796	3.952223e+07	31.810687
Random Forest Regressor CV	0.975400	0.974022	2034.718601	3663.748436	1.342305e+07	15.009674
K-Nearest Neighbours Regressor	0.961999	0.960393	2610.922984	4523.844277	2.046517e+07	20.260696
K-Nearest Neighbours Regressor CV	0.999997	0.979919	1536.616589	3221.169909	1.037594e+07	11.474373

Model	R^2 Train Score	R^2 Test Score	Mean Absolute Error	Root Mean Squared Error	Mean Squared Error	Mean Absolute Percentage Error
Gradient Boosting Regressor	0.948939	0.948112	3287.661751	5177.897910	2.681063e+07	33.434200
Gradient Boosting Regressor CV	0.990397	0.987888	1353.885887	2501.652767	6.258267e+06	11.078824

Table 4.1: Evaluation Metrics (Pre/Post Hyperparameter Tuning)

Furthermore, each model's performance based on these metrics are analysed in detail.

4.1.1.1 Linear Regression

The Linear Regression model displayed consistent performance before and after hyperparameter tuning, as both R^2 scores for the training and test sets were unchanged. Although the model effectively captured overall trends in the data, it struggled with high fare values, which is reflected in the relatively high Mean Absolute Error and Root Mean Squared Error values. Hyperparameter tuning offered little improvement to these metrics, suggesting that Linear Regression was limited in handling the non-linear relationships found within the dataset.

4.1.1.2 Decision Tree Regression

The Decision Tree Regression initially showed strong performance, outperforming the Linear Regression model, especially in its R^2 scores and error metrics. However, the model showed signs of overfitting, as seen by the significant gap between the training and test R^2 scores. Hyperparameter tuning significantly enhanced the model's performance, as both the R^2 test score and error metrics improved. This reduction in overfitting error metrics highlight the model’s increased precision in predicting fares post-tuning.

4.1.1.3 Random Forest Regression

Prior to tuning, the Random Forest Regression demonstrated excellent generalisation, with closely aligned R^2 scores for both the training and test sets. After hyperparameter tuning, the model’s performance improved even further, resulting in higher R^2 scores and substantial reductions in the error metrics. The model’s predictive accuracy was enhanced after tuning, making it one of the most effective models for flight fare prediction.

4.1.1.4 K-Nearest Neighbours Regression

The K-Nearest Neighbours (KNN) Regression revealed the highest accuracy before tuning, as indicated by its R^2 scores. However, overfitting was apparent due to the discrepancy between the training and test scores. After hyperparameter tuning, the R^2 scores became more balanced, and there was a significant decrease in error metrics such as Mean Absolute Error and Mean Absolute Percentage Error. This indicates that the model's generalisation improved with tuning, making it an exceptionally reliable choice for predicting flight fares with increased precision and reduced overfitting.

4.1.1.5 Gradient Boosting Regression

The Gradient Boosting Regressor balanced accuracy and generalisation well before tuning. After tuning, its performance improved markedly, with a higher R^2 test score and significantly reduced error metrics. However, a slight increase in Mean Squared Error (MSE) was observed, likely due to a few larger errors that were amplified by squaring. Despite this, the model's overall precision and ability to handle larger fare values post-tuning solidify its position as one of the top-performing models in this study.

4.1.2 Graphical Analysis

Scatterplots visually demonstrate the relationship between predicted and actual values for each model, both before and after hyperparameter tuning. Below is a summary of the scatterplots for each model:

4.1.2.1 Linear Regression

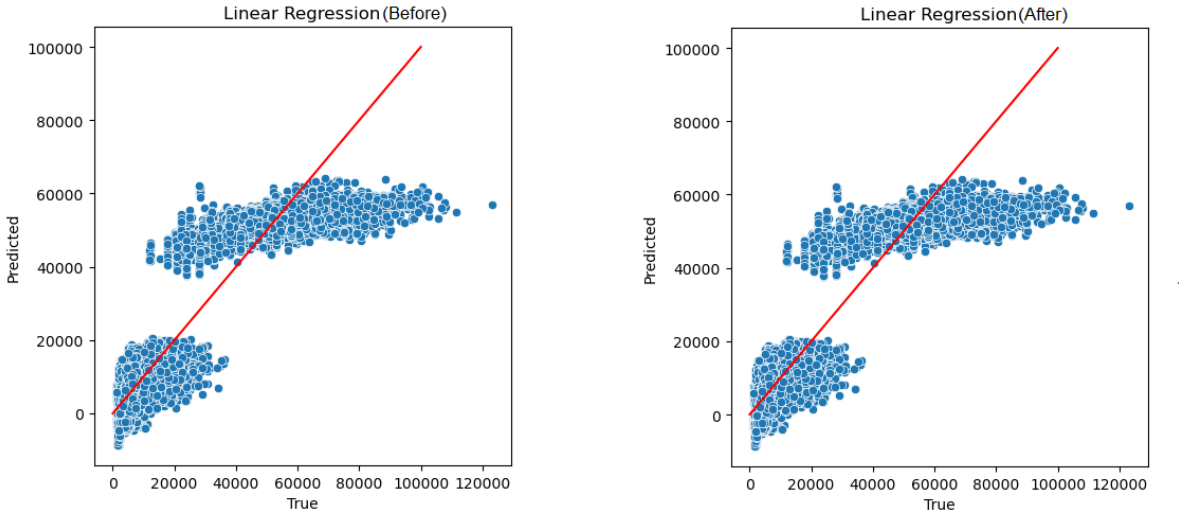


Figure 4.1: Linear Regression (Pre/Post Hyperparameter Tuning)

The scatterplots in Figure 4.1 for Linear Regression remained nearly identical before and after tuning, with predictions moderately clustered along the red line. However, for larger fare values, the model's predictions deviated more significantly from actual values. This visual pattern confirmed the model's inability to precisely predict higher fares, consistent with the observed error metrics.

4.1.2.2 Decision Tree Regression

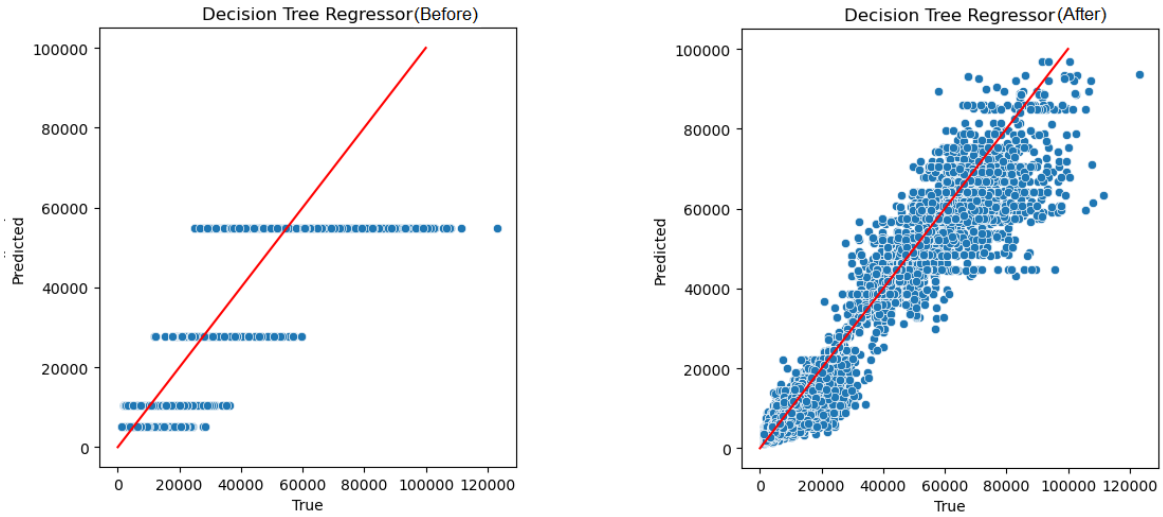


Figure 4.2: Decision Tree Regression (Pre/Post Hyperparameter Tuning)

As per the scatterplots in Figure 4.2, the Decision Tree Regressor scatterplot displayed distinct horizontal bands before tuning, a sign of stepwise prediction typical in decision tree models.

After tuning, the scatterplot revealed better alignment of predicted and actual values, reflecting the improved performance and reduced overfitting. The scatterplot post-tuning showed fewer abrupt changes in predictions, demonstrating the model's improved ability to handle various fare values more smoothly.

4.1.2.3 Random Forest Regression

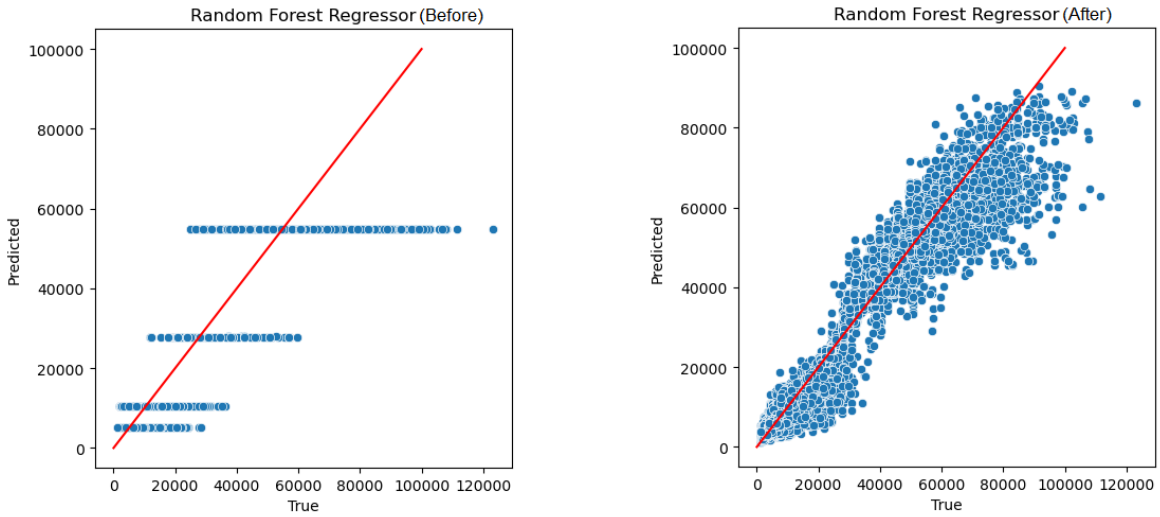


Figure 4.3: Random Forest Regression (Pre/Post Hyperparameter Tuning)

The Random Forest Regressor performed consistently both before and after tuning. While the pre-tuning scatterplot in Figure 4.3 showed some horizontal banding, tuning reduced this effect, and predictions aligned more closely along the red line. This improved alignment mirrored the reduction in Mean Absolute Error and Root Mean Squared Error, confirming the enhanced accuracy of the Random Forest Regressor after tuning.

4.1.2.4 K-Nearest Neighbours Regression

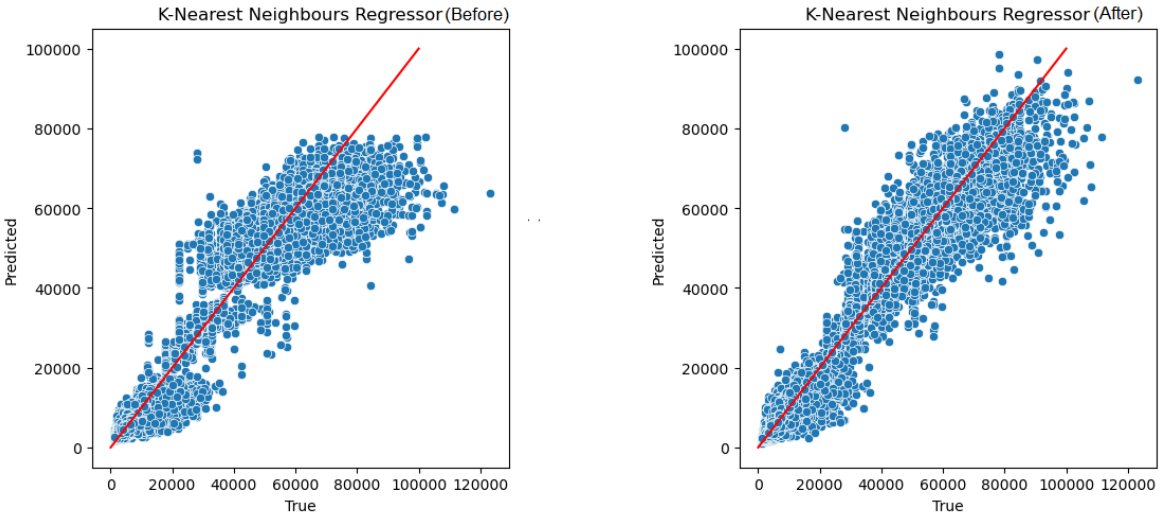


Figure 4.4: K-Nearest Neighbours Regression (Pre/Post Hyperparameter Tuning)

The pre-tuning scatterplots in Figure 4.4 for KNN displayed a tight clustering of points around the red line, signifying high accuracy. After tuning, the scatterplot showed a slight relaxation in clustering, indicative of reduced overfitting. Nonetheless, the model's predictions remained closely aligned with actual values, confirming its reliability as a predictive tool.

4.1.2.5 Gradient Boosting Regression

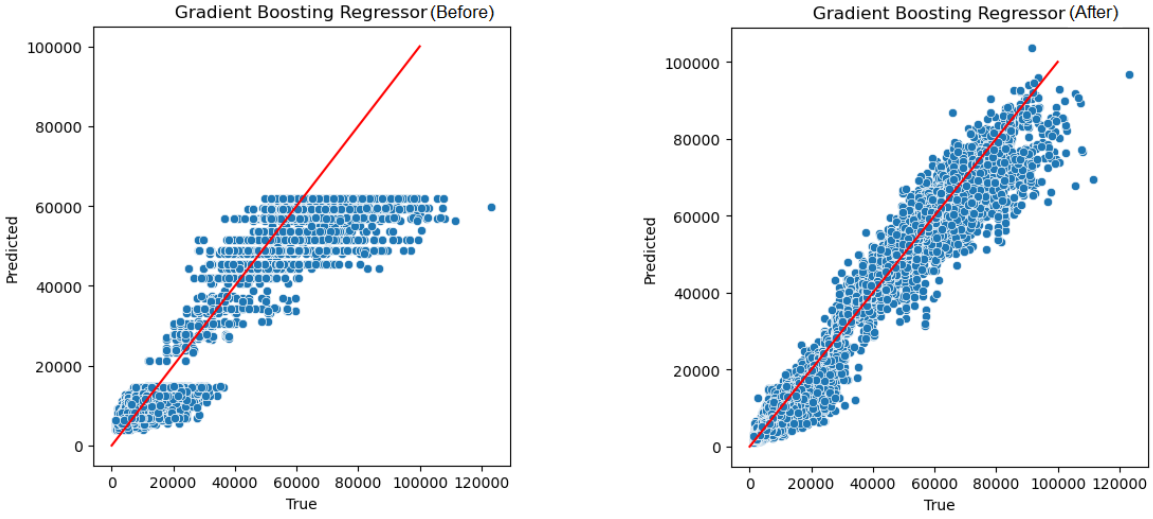


Figure 4.5: Gradient Boosting Regression (Pre/Post Hyperparameter Tuning)

The Gradient Boosting Regressor scatterplot in Figure 4.5, before tuning, demonstrated a good fit between predicted and actual values. After tuning, this fit improved further, particularly for higher fare values, as the scatterplot points were more tightly clustered along the red line. This

visual improvement corresponded to the reduced error metrics and better generalisation observed post-tuning.

4.2 Ensemble Model Implementation

Following hyperparameter tuning, the combined performance of the ensemble models was evaluated using various metrics and scatterplots for a thorough analysis.

4.2.1 Performance Metrics

The performance of the ensemble models was evaluated using standard evaluation metrics, summarised in Table 4.2. In addition to that, Appendix D includes visualisations comparing the predictions of the average, voting, and stacking ensemble regression models with actual flight data. These models demonstrate the alignment of individual model predictions with the true values, highlighting the effectiveness of averaging, voting, and stacking techniques in enhancing the prediction accuracy through their distinct approaches.

Model	R^2 Train Score	R^2 Test Score	Mean Absolute Error	Root Mean Squared Error	Mean Squared Error	Mean Absolute Percentage Error
Average Regressor	0.984773	0.977983	1982.434551	3372.846863	1.137610e+07	16.439087
Voting Regressor	0.985260	0.978408	1953.471924	3340.179027	1.115680e+07	16.103120
Stacking Regressor	0.993958	0.988685	1317.834810	2417.974372	5.846600e+06	10.850822

Table 4.2: Ensemble Model Performance Comparison

In addition to that, each model's performance based on the performance metrics is analysed in detail.

4.2.1.1 Average Ensemble Regression

The Average Ensemble Regression model performed well, showing a good fit for the training data, and successfully generalising to unseen data. The Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) indicate that the model provided reasonably accurate predictions, though there remains scope for refinement. The Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE) further demonstrate the model's reliability in predicting flight fares.

4.2.1.2 Voting Ensemble Regression

The Voting Ensemble Regression model marginally outperforms the Average Regressor, displaying a better fit and superior generalisation capabilities. The slightly lower Mean Absolute Error and Root Mean Squared Error point to improved prediction accuracy. The Mean Squared Error and Mean Absolute Percentage Error also support the enhanced performance of this model, making the Voting Regressor a stronger contender for flight fare prediction.

4.2.1.3 Stacking Ensemble Regression

The Stacking Ensemble Regression model delivered the best performance among the ensemble approaches. It fit the training data excellently and generalised well to test data. The significantly lower Mean Absolute Error and Root Mean Square Error signify high prediction accuracy. Furthermore, the MSE underscores the Stacking Regressor's effectiveness in minimising prediction error.

4.2.2 Graphical Performance

Scatterplots in Figure 4.6 were used to visually evaluate the performance of the ensemble models, providing an illustrative comparison of their predictive accuracy.

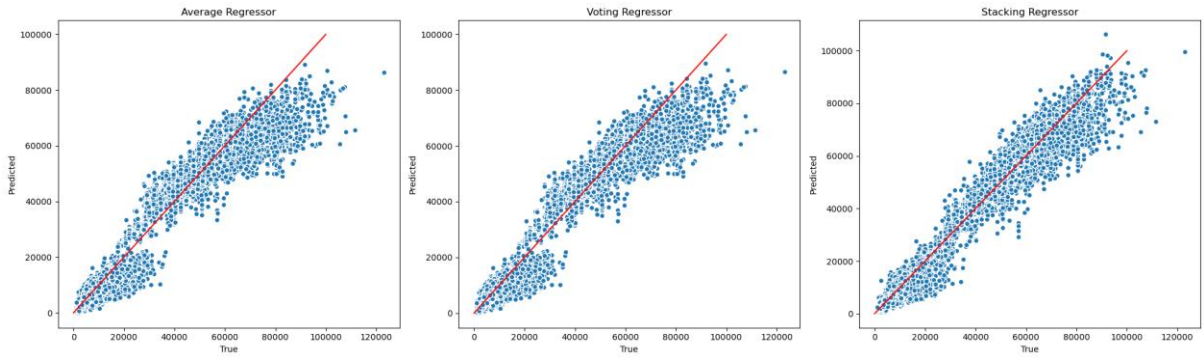


Figure 4.6: Ensemble Models Performance via Scatterplots

Below is a detailed discussion of each model's performance, as observed from the scatterplot results:

4.2.2.1 Average Ensemble Regression

The scatterplot for the Average Ensemble Regression model indicates a strong correlation between the actual and predicted values. Most data points are clustered around the diagonal line, suggesting fairly accurate predictions. However, a few outliers, particularly at the higher

fare ranges, reveal instances where the model overpredicts or underpredicts fares. This suggests that while the model is consistent, there is potential for improving its predictive accuracy.

4.2.2.2 Voting Ensemble Regression

The Voting Ensemble Regression scatterplot displays a similar trend to the Average Regressor, but with data points more closely aligned to the ideal prediction line. This suggests more accurate predictions across the range of fares, with fewer substantial outliers. The clustering of data points near the diagonal line confirms the model's improved generalisation ability, particularly in the mid-to-high fare range. The reduced spread of data points signifies better predictive reliability compared to the Average Regressor.

4.2.2.3 Stacking Ensemble Regression

The scatterplot for the Stacking Ensemble Regression model shows the tightest clustering of data points around the diagonal, indicating the highest level of prediction accuracy. There are fewer outliers, and the points remain consistently close to the ideal prediction line. This demonstrates that the Stacking Regressor effectively reduces both overprediction and underprediction errors, making it the most dependable model among the ensemble approaches, especially for higher fare predictions. The scatterplot clearly supports the conclusion that the Stacking Ensemble Regression delivers superior overall performance.

4.3 Feature Importance Analysis

The Stacking Ensemble model was selected due to its superior performance compared to other models. This analysis aims to identify which features most significantly influence ticket price fluctuations.

4.3.1 Global Explanations

To evaluate global feature importance, Shapley Additive Explanations (SHAP) summary plots and feature importance bar charts were employed to determine the impact of features across the entire dataset in predicting prices.

4.3.1.1 SHAP Summary Plot

The SHAP summary plot visualises the distribution and direction of feature impacts across the dataset, highlighting the most influential features in price prediction. On the y-axis, features are ranked by importance, while the x-axis displays SHAP values, which represent their influence on the prediction. The colour gradient, from blue to red, indicates feature values, with blue representing lower values and red higher values.

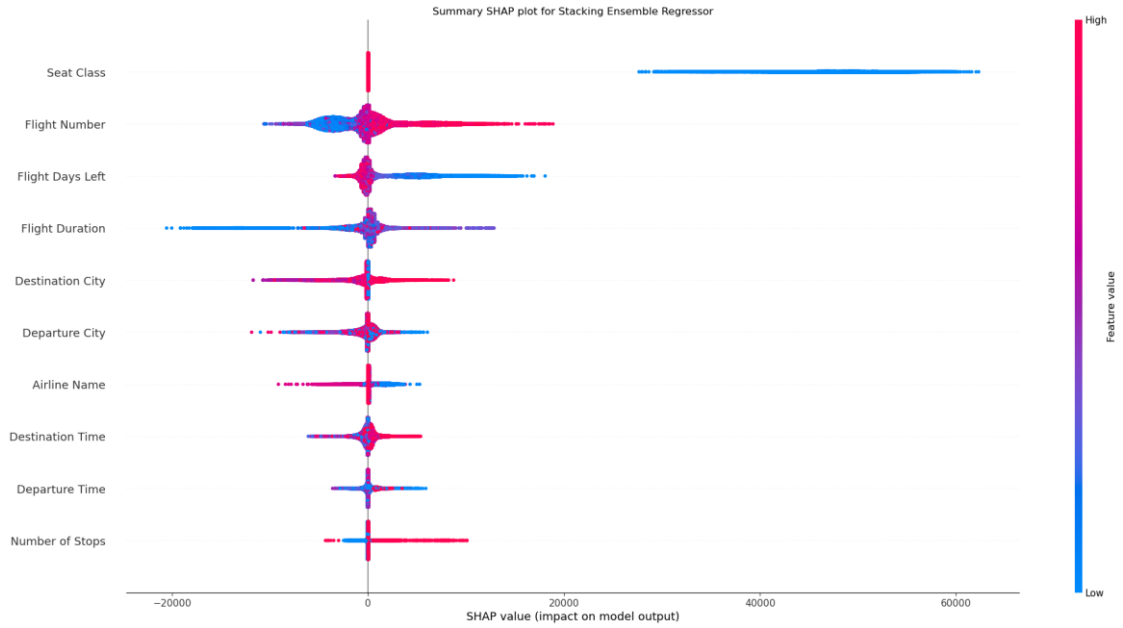


Figure 4.7: SHAP Summary Plot

The plot in Figure 4.7 reveals that seat class has the greatest effect on ticket prices, heavily influencing the final fare. Higher SHAP values, particularly those in blue, represent instances where seat classes for economy also increase the model's output, indicating that even in lower seat classes, ticket prices can rise significantly under certain conditions, such as peak demand or last-minute bookings. Flight number is also a major factor, potentially due to variables such as flight timing, route popularity, or airline. The number of days until departure also significantly affects prices, with closer departure dates leading to higher costs. Longer flights generally incur higher costs. The airline selected influences prices, with the SHAP values reflecting differences in pricing between carriers. Departure and destination cities impact fares, likely due to variations in distance, demand, and connectivity. Additionally, the times of departure and arrival influence pricing, possibly reflecting peak travel times. Lastly, flights with multiple stops are typically cheaper, though some exceptions exist.

4.3.1.2 Feature Importance Bar Plot

The Feature Importance Bar Plot ranks features based on their average SHAP values, highlighting their relative importance in the model. The x-axis represents average importance values, while the y-axis lists the features in descending order of impact.

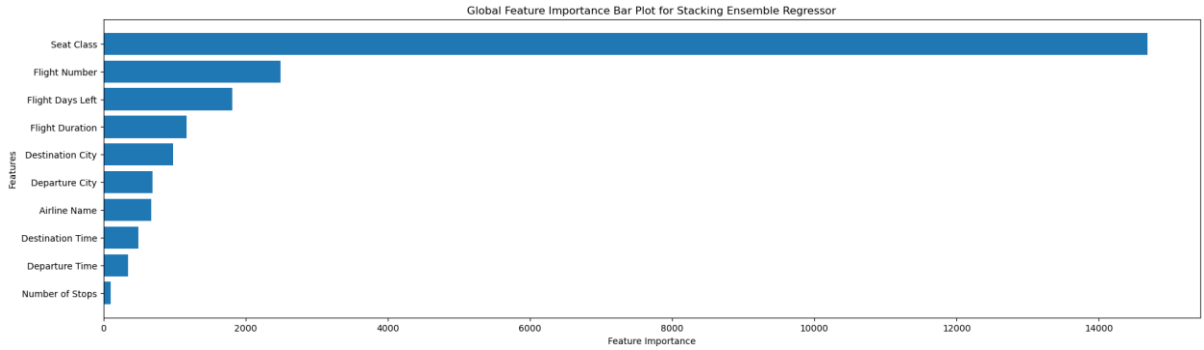


Figure 4.8: Feature Importance Bar Plot

According to the bar plot in Figure 4.8, seat class exerts the most influence on flight fares, with higher classes commanding higher prices due to added amenities and service levels. Flight number affects pricing, often determined by airline policies, aircraft type, and route popularity. Tickets purchased closer to departure tend to be more expensive due to reduced availability and increased demand. Longer flights generally have higher fares due to greater operational costs. The airline name also influences fares, with certain airlines charging more for better services or desirable routes. Departure and destination cities impact fares, with major hubs or popular destinations often having higher prices. Additionally, peak travel times raise prices, and flights with multiple stops tend to be cheaper due to longer journey times and inconvenience.

4.3.2 Local Explanations

For individual flight fare predictions, local explanations were generated using SHAP waterfall and force plots, which provide a detailed breakdown of how each feature contributes to a specific prediction.

4.3.2.1 SHAP Waterfall Plot

The SHAP waterfall plot visualises how the model's average prediction is adjusted by each feature's SHAP value to arrive at the final fare prediction. The x-axis displays the predicted

flight fare values, showing how each feature shifts the prediction up or down from the model's baseline. The average prediction is denoted as E[f(x)], and the y-axis ranks the features by their importance. The final prediction, f(x), is reached after all feature contributions are considered.

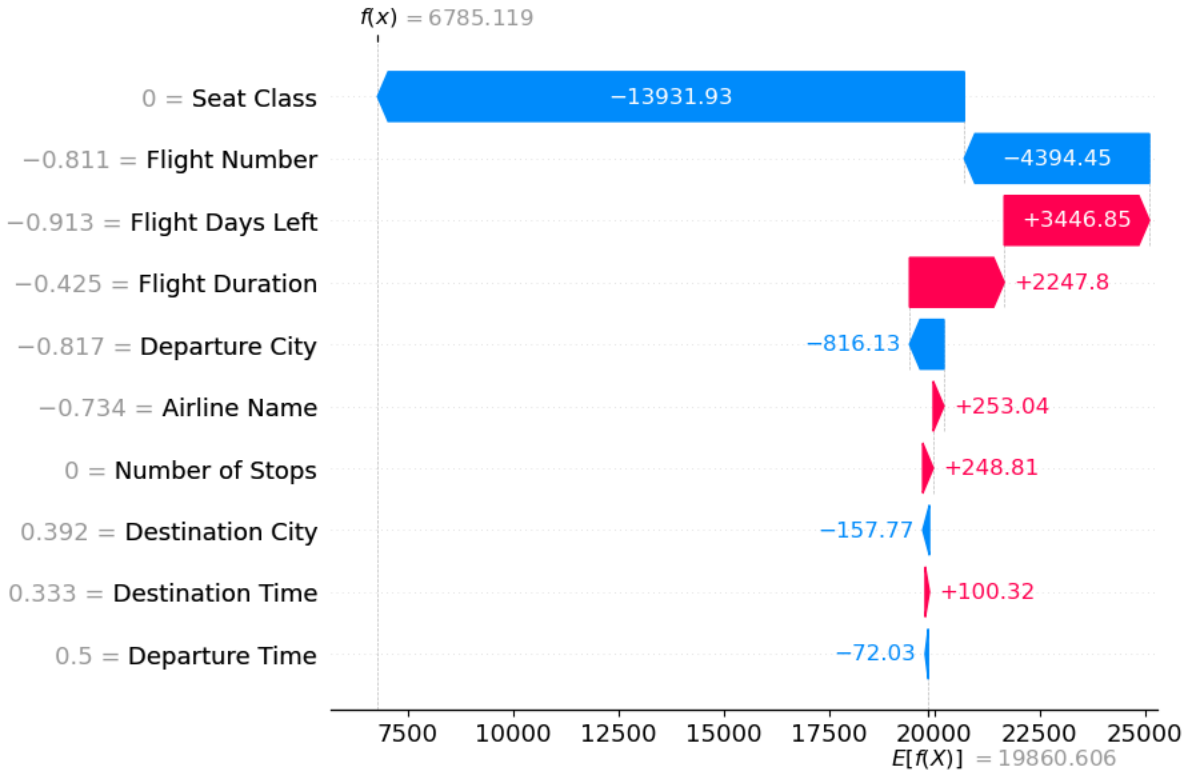


Figure 4.9: SHAP Waterfall Plot (Instance 1)

In the Figure 4.9 plot of a particular instance, the waterfall plot starts with an average prediction of INR 19,860. Seat class dramatically reduces the fare by INR 13,932, suggesting economy seats are much cheaper. Flight number further lowers the fare by INR 4,394, possibly due to flight scheduling or route popularity. A last-minute booking increases the fare by INR 3,447, reflecting the higher prices closer to departure. Longer flight durations add INR 2,248 to the fare due to increased costs. Departing from a particular city reduces the fare by INR 816, while the airline choice adds INR 253, with premium carriers charging more. The number of stops has no significant impact, and the destination city marginally decreases the fare by INR 158. Departure and arrival times show minimal effects, with changes of around INR 100 or less.

4.3.2.2 SHAP Force Plot

The SHAP force plot demonstrates how individual features affect a specific prediction by moving it from the baseline value to the final predicted fare, highlighting both positive and negative influences of each feature.

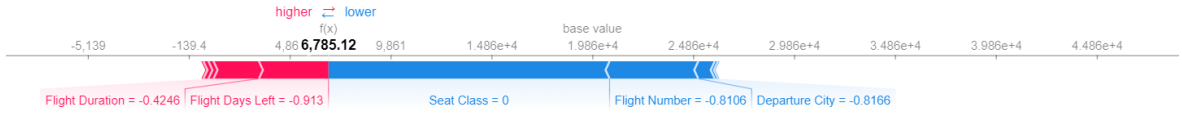


Figure 4.10: SHAP Force Plot (Instance 1)

In Figure 4.10, the SHAP force plot estimates a fare of INR 6,785.12. The baseline value (average prediction) is INR 19,860. The seat class feature has a SHAP value of 0, meaning it does not influence the predicted fare for this instance. The flight days left reduces the fare with a SHAP value of -0.913, indicating that booking closer to the departure date has a slight negative effect on the fare. Flight duration decreases the fare by a SHAP value of -0.4246. The flight number reduces the fare with a SHAP value of -0.8106, suggesting that certain flight numbers are associated with lower fares. The departure city also slightly lowers the fare, contributing a SHAP value of -0.8166. These SHAP values collectively explain how the model's prediction was adjusted from the baseline to reach the final fare of INR 6,785.12.

To smoothly transition into the next chapter, it is vital to consolidate the insights obtained from the feature importance analysis and the performance of the models. With the critical factors affecting flight fares identified, the following chapter will provide an in-depth evaluation of how well the models have captured these patterns. The next chapter will comprehensively assess the performance of the flight fare prediction models, focusing on accuracy, efficiency, and reliability before and after hyperparameter tuning. It will also explore various ensemble methods, emphasising their influence on enhancing model performance. Additionally, the chapter will evaluate the significance of feature importance in shaping predictions, offering insights and recommendations for improving future airfare predictions.

Chapter 5 - Evaluation

Following the detailed performance analysis in Chapter 4, this chapter provides a deeper evaluation of the models used for flight fare prediction. Using the same performance metrics, the models are assessed based on accuracy, generalisation capacity, and the significance of key features in influencing predictions. The impact of hyperparameter tuning and ensemble methods is also examined, offering a comprehensive overview of the models' strengths and limitations in addressing the project's objectives.

5.1 Evaluation Criteria

The evaluation relies on the metrics outlined in chapter 4, including R^2 Score, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE). These metrics were chosen to provide a balanced assessment of model performance. For instance, R^2 Score indicates the proportion of variance in flight fares explained by the model, while MAE and RMSE highlight the magnitude of errors. MAPE, expressed as a percentage, enables comparison across models. Each metric emphasises different aspects of performance, ensuring a comprehensive evaluation.

5.2 Model Performance Evaluation

This section evaluates model performance from three key perspectives: individual model effectiveness, the role of ensemble methods, and the importance of features in predicting flight fares.

5.2.1 Individual Model Evaluation

The individual models, including Linear Regression, Decision Tree Regression, Random Forest Regression, K-Nearest Neighbours Regression, and Gradient Boosting Regression, were assessed both before and after hyperparameter tuning. As discussed in Chapter 4, tuning generally improved the performance of complex models, particularly Random Forest and Gradient Boosting, which showed reduced error rates and improved R^2 scores. These models better captured complex relationships within the data, resulting in more accurate fare predictions. Simpler models, such as Linear Regression and Decision Tree, showed less improvements and struggled with the non-linear patterns in the dataset.

5.2.2 Ensemble Model Evaluation

Ensemble methods such as Average Ensemble Regression, Voting Ensemble Regression, and Stacking Ensemble Regression to improve prediction accuracy combining the strengths of individual models. Among these, the Stacking Ensemble Regression demonstrated the best performance, reducing errors and producing more reliable results than any single model. This indicates that ensemble methods can significantly enhance prediction accuracy, particularly in complex tasks like flight fare forecasting.

5.2.3 Evaluation of Feature Importance

Shapley Additive Explanations (SHAP) were used to assess the importance of key features in predicting flight fares. The global SHAP summary plots revealed that seat class, flight number, and days till departure were the top three factors influencing fare predictions. Seat class had a strong impact, consistently ranking as a key feature driving fare prices. Other features, such as flight duration, destination city, and departure city, also showed notable influence, though to a lesser extent. Local SHAP force and waterfall plots provided further insights into how these features impacted individual predictions. The combined global and local SHAP analysis offered a detailed understanding of model interpretability, validating that the models effectively captured key drivers of fare variability in line with domain knowledge.

5.3 Critical Analysis of Results

The evaluation results demonstrate strong performance from the more complex models, such as Gradient Boosting and Random Forest Regression, which effectively captured intricate relationships within the data and maintained low error rates. Although these models showed minor overfitting, the overfitting did not significantly affect their predictive accuracy. Simpler models like Linear Regression and Decision Tree Regression provided faster processing and were less prone to overfitting, but they delivered lower accuracy and higher error metrics. Despite this, simpler models played a critical role in establishing baseline benchmarks, highlighting the improvements achieved with more advanced algorithms. Ensemble methods, particularly the Stacking Ensemble Regression, further improved accuracy by combining the strengths of multiple models, minimising errors, and demonstrating strong generalisation on test data. Although there is room for further enhancement through model tuning and feature

engineering, the system proved highly effective in forecasting flight prices, with only minor limitations.

5.4 Limitations of Evaluation

Several limitations were encountered during the evaluation. On the major challenges was the computational constraint posed by developing the project on a less powerful machine without a dedicated GPU. This significantly impacted the processing times, particularly for more complex models such as Random Forest, Gradient Boosting, and feature importance assessments using SHAP. Processing took several hours, limiting the extent of hyperparameter optimisation, leaving some model configurations untested. Additionally, the dataset lacked key external factors known to influence flight fares, such as oil prices, public holidays, and major events, which, if included, could have provided deeper insights and further enhanced prediction accuracy. The evaluation was also limited to a specific set of metrics, and future analyses could explore additional techniques to further enhance accuracy and reduce error rates.

5.5 Recommendations for Future Work

To address the limitations encountered, future work should prioritise access to more powerful computing resources, such as high-performance machines with GPUs, to significantly reduce processing times and enable more extensive hyperparameter tuning. Incorporating external factors, such as oil prices, public holidays, and promotions, into the dataset could further improve the accuracy of fare predictions. In addition, employing alternative evaluation metrics like Adjusted R^2, Mean Squared Logarithmic Error (MSLE), or Root Mean Squared Logarithmic Error (RMSLE) would provide a more comprehensive evaluation of model performance. Exploring advanced ensemble techniques or deep learning models may also enhance prediction accuracy, particularly when handling complex, non-linear data. By addressing these aspects, future efforts will likely result in more precise and efficient flight fare predictions.

This thorough evaluation has provided insights into the strengths, weaknesses, and limitations of the models used for flight fare prediction, as well as identifying potential avenues for improvement. The findings from this chapter lay the groundwork for the final conclusions and recommendations, which will be explored in the next chapter. In that chapter, the overall

achievements of this project will be summarised, along with reflections on how challenges were addressed and suggestions for future work.

Chapter 6 - Conclusion

This final chapter consolidates the key themes, achievements, and challenges encountered during the development of the flight fare prediction system. It reflects on the methodologies used, the limitations faced and outlines potential directions for future work. The chapter aims to provide a cohesive summary of the insights gained, the outcomes achieved, and lessons learned throughout the project.

6.1 Restatement of the Work Undertaken

The aim of the project was to design and evaluate machine learning models capable of predicting flight fares accurately while identifying key factors that significantly influence fare prices. To achieve this, several models, including Linear Regression, Decision Trees, Random Forests, K-Nearest Neighbours (KNN), and Gradient Boosting, were implemented. These models underwent rigorous hyperparameter tuning to enhance performance, and ensemble methods—Average, Voting, and Stacking Ensemble Regressions—were explored to improve predictive accuracy. A comprehensive feature importance analysis was carried out using SHAP values, which revealed how factors like airline, seat class, flight duration, and days left until departure impacted fare prices. Models were evaluated using metrics such as R^2, MAE, RMSE, and MAPE, and SHAP analysis was key in making predictions more interpretable, helping to explain how various factors contributed to fare predictions.

6.2 Findings from the Work

The project's findings provided important insights into both model performance and feature importance. Gradient Boosting emerged as the best-performing individual model, achieving the highest accuracy across all performance metrics after hyperparameter tuning. However, ensemble methods, particularly Stacking Ensemble Regression, excelled at combining the strengths of individual models, delivering superior predictive accuracy compared to Average and Voting Ensemble methods. SHAP analysis further revealed that seat class, flight duration, and the number of days before departure were the most influential factors determining flight prices. These results underscored the capacity of machine learning models to produce transparent and interpretable predictions, particularly in fare forecasting.

6.3 Reiteration of Achievements

One of the project's key accomplishments was the successful development of a flight fare prediction system that produced highly accurate results. Among all models tested, Stacking Ensemble Regression proved to be the top performer, blending the strengths of models like Random Forest and Gradient Boosting to capture complex data patterns. Another notable achievement was the improvement in model performance through hyperparameter tuning, which particularly benefited Decision Trees, Random Forests, and Gradient Boosting. These models, which initially faced overfitting issues, demonstrated significant gains in generalisation after optimisation. Moreover, SHAP analysis enhanced the interpretability of the models by providing a detailed understanding of how specific features—such as seat class, flight duration, and departure time—influenced fare predictions.

6.4 Positive Aspects

This project demonstrated several strengths, most notably the structured approach taken to model development and fine-tuning. Ensemble techniques, especially Stacking Ensemble Regression, provided a clear advantage over standalone models by combining their predictive capabilities and improving overall performance. The effective use of hyperparameter tuning allowed models like Decision Trees and Gradient Boosting to be optimised, resulting in enhanced predictive power. Additionally, the use of various performance metrics ensured a comprehensive evaluation of model suitability for the task. The SHAP analysis was instrumental in improving the system's interpretability, a vital factor in machine learning applications where transparency and explainability are increasingly important.

6.5 Addressing Challenges

Throughout the project, several challenges arose, with computational limitations being the most significant. The lack of access to a GPU led to longer processing times, especially for complex models like Random Forests and Gradient Boosting, as well as for SHAP-based feature importance analysis. Hyperparameter tuning became a time-consuming process that required careful prioritisation of promising parameter configurations.

To manage the lack of GPU support, the entire night was allocated for processing all algorithms, reserving the daytime for analysing results, amending the code, and developing subsequent parts of the project. In terms of hyperparameter tuning, Randomised Search CV was opted instead of Grid Search CV, as the former is known to be more efficient in exploring a wide range of parameter configurations without the exhaustive nature of grid search.

6.6 Suggestions for Future Work

Future work could explore integrating external variables, such as fuel prices, holiday schedules, and major events, to offer a more comprehensive understanding of factors influencing flight fares. The inclusion of such variables could enhance the model's ability to predict fare fluctuations more accurately. Another avenue for future research involves applying time-series techniques to track flight price changes dynamically over time, enabling real-time fare predictions and dynamic pricing strategies. Additionally, using cloud-based or GPU-powered environments could facilitate deeper exploration of hyperparameter configurations, overcoming the computational constraints faced during this project. More complex models, like deep learning, could also be explored to capture non-linear patterns in the data that simpler models struggled with, while expanding evaluation metrics, such as Adjusted R^2 or logarithmic errors like MSLE, could provide further insights into model performance.

6.7 Personal Reflection

This project provided an excellent opportunity to apply the machine learning techniques learned throughout my MSc Data Science course. Concepts from various modules, including data analysis, feature engineering, and model optimisation, were central to the development of the flight fare prediction system. The hands-on experience with hyperparameter tuning and model evaluation reinforced understanding of these techniques, while SHAP analysis highlighted the importance of model interpretability. A key lesson learned was the importance of computational resources in machine learning projects. The limitations encountered due to the absence of a GPU underscored the need for efficient model selection, early validation, and cautious tuning. Reflecting on the project, securing access to cloud-based or GPU-powered systems earlier could have allowed for a broader exploration of potential model configurations.

6.8 Final Thoughts

In conclusion, this project successfully met its objective of building a flight fare prediction system that balanced accuracy and interpretability. The Stacking Regressor demonstrated a strong capacity for predicting fares, while SHAP analysis provided transparency into the factors influencing these predictions. Despite some limitations—such as computational constraints and the lack of certain external variables—the project navigated these challenges and offered valuable insights into airfare prediction. The project has laid a strong foundation for future work, with numerous opportunities for improvement and expansion. The skills acquired, particularly in machine learning model development, hyperparameter tuning, and explainable AI, will undoubtedly be beneficial in future data science roles, where balancing model complexity with real-world applicability is crucial.

References

Albon, C. and Gallatin, K. (2023) *Machine learning with python cookbook.* 2nd ed., Sebastopol: O'Reilly Media Inc.

Arindam (2022) *Hyperparameter tuning using randomized search.* Analytics Vidhya. [Online] [Accessed on 18th August 2024] https://www.analyticsvidhya.com/blog/2022/11/hyperparameter-tuning-using-randomized-search/

Alsdorf, M. and Korner, C. (2022) *Mastering azure machine learning.* 2nd ed., Birmingham: Packt Publishing Ltd.

Arjun, K.P., Rawat, T., Singh, R. and Sreenarayanan, N.M. (2022) 'Flight fare prediction using machine learning.' *In* Mehra, R., Meesad, P., Peddoju, S.K., Rai, D.S. (eds) *Computational intelligence and smart communication.* Vol. 1672, Cham: Springer Nature Switzerland, pp. 89–99.

Beyeler, M. (2017) *Machine learning for opencv.* Birmingham: Packt Publishing Ltd.

Bhatnagar, S. (2023) *Ensemble methods in machine learning.* Medium. [Online] [Accessed on 28th August 2024] https://medium.com/@shashank25.it/ensemble-methods-in-machine-learning-2d4cc7513c77

Bifet, A., Holmes, G., Gavaldà, R. and Pfahringer, B. (2018) *Machine learning for data streams: with practical examples in moa.* London: The MIT Press.

Breiman, L. (2001) 'Random forests.' *Machine Learning.* 45 pp. 5-32.

Brownlee, J. (2021) *Regression metrics for machine learning.* Machine Learning Mastery. [Online] [Accessed on 7th August 2024] https://machinelearningmastery.com/regression-metrics-for-machine-learning/

de Myttenaere, A., Golden, B., Le Grand, B. and Rossi, F. (2016) 'Mean absolute percentage error for regression models.' *Neurocomputing*, 192 pp. 38-48.

Deepanshi (2023) *All you need to know about your first Machine Learning model – linear regression.* Analytics Vidhya. [Online] [Accessed on 13th July 2024] https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-your-first-machine-learning-model-linear-regression/

Dietterich, T.G. (2000) ‘Ensemble methods in machine learning.’ *Multiple Classifier Systems. MCS 2000. Lecture Notes in Computer Science,*1857 pp. 1-15.

Etzioni, O., Tuchinda, R., Knoblock, C. and Yates, A. (2003) ‘To buy or not to buy: mining airfare data to minimize ticket purchase price.’ *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '03)*, New York: Association for Computing Machinery, pp. 119–128.

Friedman, J. H. (2001) ‘Greedy function approximation: a gradient boosting machine.’ *Annals of statistics,* 29(5) pp. 1189-1232.

Frost, J. (no date) *How to interpret r-squared in regression analysis.* Statistics By Jim. [Online] [Accessed on 2nd August 2024] https://statisticsbyjim.com/regression/interpret-r-squared-regression/

Galli, S. (2020) *Python feature engineering cookbook.* 1st ed., Birmingham: Packt Publishing Ltd.

Gopal, P.V., Kumar, B.P., Kumar, V.A., Akhtar, M.J. and Sivanagaraju, P. (2024) ‘Flight fare prediction using machine learning algorithms.’ *International journal of creative research thoughts (IJCRT)*, 11(1) pp. 634-643.

GeekForGeeks. (2024) *Sklearn.stratifiedshufflesplit() function in python.* [Online] [Accessed on 11th August 2024] https://www.geeksforgeeks.org/sklearn-stratifiedshufflesplit-function-in-python/

Groves, W. and Gini, M. (2013) ‘An agent for optimizing airline ticket purchasing.’ *12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013),* pp. 1341-1342.

Goel, A. (2023) Model Explainability using SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations). Medium. [Online] [Accessed: 19th August 2024] https://medium.com/@anshulgoel991/model-exploitability-using-shap-shapley-additive-explanations-and-lime-local-interpretable-cb4f5594fc1a

Gupta, P. (2017) *Decision trees in machine learning.* Towards Data Science. [Online] [Accessed on 3rd July 2024] https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

Hastie, T., Tibshirani, R. and Friedman, J. (2009) *The elements of statistical learning.* 2nd ed. New York: Springer.

Hodson, T.O. (2022) ‘Root-mean-square error (rmse) or mean absolute error (mae): when to use them or not.’ *Geoscientisifc Model Development*, 15(14) pp. 5481-5487.

International Air Transport Association. (2023) *Global outlook for air transport.* [Online] [Accessed on 20th June 2024] https://www.iata.org/en/iata-repository/publications/economic-reports/global-outlook-for-air-transport----june-2023/

Janssen, T. (2014) *A Linear Quantile Mixed Regression Model for Prediction of Airline Ticket Prices.* Bachelor's thesis. Radboud University, Nijmegen.

Kavita (2024) *Linear regression: a comprehensive guide.* Analytics Vidhya. [Online] [Accessed on 17th August 2024] https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/#:~:text=Linear%20regression%20has%20two%20main,the%20independent%20variable%20is%20zero

Kaufmann, J. and Schering, A. (2014) ‘Analysis of variance anova.’ *In* Balakrishnan, N., Colton, T., Everitt, B., Piegorsch, W., Ruggeri, F. and Teugels, J.L. (eds) *Wiley statsref: statistics reference online.* 1st ed., Hoboken: John Wiley & Sons, pp.

Kimbhaune, V., Donga, H., Trivedi, A., Mahajan, S. and Mahajan, V. (2021) ‘Flight Fare Prediction System’. *EasyChair Preprint, 5542.*

Kumar, A. and Jain, M. (2020) *Ensemble learning for ai developers: learn bagging, stacking, and boosting methods with use cases.* Berkeley: APress.

Lantseva, A., Mukhina, K., Nikishova, A., Ivanov, S., and Knyazkov, K. (2015) ‘Data-driven modelling of airlines pricing.’ *Procedia Computer Science,* 66, pp. 267-276.

Lundberg, S.M. and Lee, S-I. (2017) ‘A unified approach to interpreting model predictions.’ *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems,* 30, pp. 4768–4777.

Miller, L. (2018) *Function, gradient descent, and univariate linear regression.* Medium. [Online] [Accessed on 20th September 2024] https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5fe69815fd

Mishra, A. (2018) *Metrics to evaluate your machine learning algorithm.* Towards Data Science. [Online] [Accessed on 17th August 2024] https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

Nadeem, R. and Sivakumar, T. (2023) 'Flight fare forecasting: A machine learning approach to predict ticket prices.' *In* Chaki, N., Roy, N.D., Debnath, P., Saeed, K. (eds) *Proceedings of international conference on data analytics and insights.* Vol. 727, Singapore: Springer Nature Switzerland, pp. 703–713.

Papadakis, M. (2014) 'Predicting airfare prices.' *Clerk Maxwell.*

Paul, S. (2018) *Hyperparameter optimization in machine learning models.* DataCamp. [Online] [Accessed on 16th August 2024] https://www.datacamp.com/tutorial/parameter-optimization-machine-learning-models

Rajankar, S., Sakharkar, N. and Rajankar, O.S. (2019) 'Predicting the price of a flight ticket with the use of machine learning algorithms.' *International Journal of Scientific & Technology Research, 8,* pp. 3297-3300.

Schott, M. (2019) *K-nearest neighbors (knn) algorithm for machine learning.* Medium. [Online] [Accessed on 6th August 2024] https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26

Shafl, A. (2023) *Random forest classification with scikit-learn.* DataCamp. [Online] [Accessed on 1st July 2024] https://www.datacamp.com/tutorial/random-forests-classifier-python

Shin, T. (2023) *Understanding feature importance in machine learning.* Built In. [Online] [Accessed on 15th September 2024] https://builtin.com/data-science/feature-importance

Shukla, J., Srivastava, A. and Chauhan, A. (2020) 'Airline price prediction using machine learning.' *International Journal of Research in Engineering, IT and Social Sciences,* 10(5) pp. 15-18.

Singh, H. (2018) *Understanding gradient boosting machines.* Towards Data Science. [Online] [Accessed on 16th August 2024] https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab

Vu, V. H., Minh, Q.T. and Phung, P. H. (2018) 'An airfare prediction model for developing markets.' *2018 International Conference on Information Networking (ICOIN),* pp. 765-770.

Wang, T., Pouyanfar, S., Tian, H., Tao, Y., Alonso, M., Luis, S. and Chen, S-C. (2019) 'A framework for airfare price prediction: a machine learning approach.' *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI),* pp. 200-207.

Yamini, A., Bhavani, K., Asha, M., Sridhar, G. and Sravanti, M. (2024) ‘Airplane fare prediction using machine learning.’ *International journal of creative research thoughts (IJCRT)*, 12(5) pp. 476-482.

Zhu, R. (2024) *Ridge regression and the bias-variance trade-off*. GitHub. [Online] [Accessed: 23 September 2024] https://teazrq.github.io/stat432/rnote/Ridge/RidgeReg.html

Appendices

Appendix A - Terms of Reference

Department of Computing and Mathematics Computing and Digital Technology Postgraduate Programmes Terms of Reference Coversheet	
Student name:	Farrukh Nizam Arain
University I.D.:	23651952
Academic supervisor:	Anthony Kleerekoper
External collaborator (optional):	
Project title:	Flight Fare Prediction System
Degree title:	MSc Data Science with Placement Year
Project unit code:	(6G7V0007_2324_9S)
Credit rating:	60
Start date:	14th June 2024
ToR date:	03rd July 2024
Intended submission date:	27th September 2024
Signature and date student:	Farrukh Nizam Arain (03rd July 2024)
Signature and date external collaborator (if involved):	

This sheet should be attached to the front of the completed ToR and uploaded with it to Moodle.

Project Aims:

The aim of the project is to develop a strong and accurate flight fare prediction system using multiple machine learning algorithms and to identify the features that most notably impact fare prices.
To achieve the aims of the project, the following objectives are defined:

1. Conduct a thorough literature review on existing flight fare prediction models and various machine learning techniques.
2. Collect and preprocess flight fare datasets of different airlines that are publicly available from various sources on the internet.
3. Implement and compare different machine learning algorithms for flight fare prediction, including but not limited to linear regression, decision trees, random forests, and gradient boosting.
4. Compare and evaluate the performances of all selected algorithms using appropriate metrics (e.g., RMSE, MAE, R^2).
5. Select the best-performing models and fine-tune them through various methods for optimal performance.
6. Analyse all the independent features in the dataset to determine which factors contribute the most to fare fluctuations.
7. Document all the steps of the entire process, including results and analysis, in a detailed project report.

Learning Outcomes:

The project will allow the development of the following knowledge and skills:

1. Choose, customise, and integrate core techniques to build sophisticated solutions for real-world Data Science problems.
2. Process and analyse, effectively and efficiently, data of varying scales and from heterogeneous sources, formats, and systems, using a range of suitable languages, tools, and environments.
3. Build data science products with good software practices such as in code reuse, separation of concerns, modularity, testing, and documentation.
4. Express resource-intensive computational tasks, such as those involving big data and complex models, using suitable parallel, concurrent, and distributed processing constructs and frameworks.

Project Description:

The dataset pertaining to flight fares contains extensive information about the pricing of airline tickets across various routes, dates, and times. This data is important for both customers and airline companies. Customers use it to find and book flights at the best possible rates, making decisions based on factors such as departure and arrival times, flight duration, and airline preferences (Zmuda 2022). Airlines, on the other hand, use it for revenue management, and strategic pricing. They analyse fare trends, demand patterns, and competitive pricing to optimise their pricing strategies, maximize occupancy rates, and enhance profitability (Slotnick, Klapper and Kerr 2023). By understanding the dynamics of flight fares, airlines can make informed decisions about pricing adjustments, promotional offers, and capacity planning.

To achieve the project's aims, a methodical approach is used, starting with the collection and preprocessing of the relevant data from various reliable public sources, ensuring that it is free from discrepancies and ready to be processed through relevant machine learning models. Machine learning algorithms, including linear regression, decision trees, random forests, and others, are selected, configured, and implemented to predict flight fares. Each model's performance is evaluated through metrics like Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2) to ensure accuracy and reliability. A critical aspect of the project is to evaluate the feature importance within these models, identifying which factors significantly impact fare prices, both positively and negatively. This analysis will enable airlines to better comprehend and manage their pricing strategies and make more educated decisions about fare trends.

Various tools and techniques are employed for the successful implementation of the project. Data preprocessing consist of techniques such as data cleaning, normalization, and transformation. Several recognised machine learning algorithms are implemented using Python as the sole programming language, with libraries such as Pandas, Matplotlib, Scikit-learn, and others. Different feature importance techniques are utilised to analyse the effects of all independent features on the target feature through the relevant scores from tree-based models, SHAP values, and others.

The expected outcomes of the project include a robust flight fare prediction system that accurately predicts the fares by learning from the historical dataset using machine learning models. Additionally, a comprehensive analysis of feature importance provides key insights into the independent factors which dominantly effect flight fares. Deliverables of the project include a detailed report covering the entire project process, including methodology, results, and conclusions.

References:

Slotnick, D., Klapper, E., Kerr, R. *What airline fare classes tell you about your ticket,* https://thepointsguy.com/guide/airline-fare-classes/, 3rd September 2023.

Zmuda, D. *Understanding Airline Fare Information,* https://www.travelmiles101.com/ understanding-airline-fare-information/, 28th July 2022.

Evaluation Plan:

The results are evaluated through well-known quantitative methods using statistical metrics such as R^2, RMSE, MAE, to assess every selected model's performance. Apart from judging and comparing performances of the models, these statistical methods also help to reduce errors and other issues. In addition to comparing the accuracy of the models, all relevant independent features in the dataset are evaluated to determine the impact of different features on prediction accuracy. Some of the feature evaluation methods utilised include prediction accuracy scores, SHAP values, and others.

Activity Schedule:

Following are the tasks and activities that are followed for the accomplishment of the project:

S.No.	Task	Start Date	End Date
1.	**Terms of Reference and Ethics Application** • Defining project aims, results, descriptions, evaluations, and deliverables schedule. • Submitting the ethics application into the system after approval from the supervisor.	14/06/2024	07/07/2024
2.	**Literature Review** • Reviewing relevant literature on flight fare prediction and its implementation via machine learning models.	01/07/2024	21/07/2024
3.	**Data Collection** • Recognizing reliable data sources from reputable repositories on the internet. • Collecting historical fare data, flight schedules, and other relevant information.	21/07/2024	23/07/2024
4.	**Data Cleaning** • Handling missing values, removing null values and duplicates, and resolving inconsistencies. • Standardizing all features in the dataset. • Transforming data as per requirements.	23/07/2024	26/07/2024
5.	**Feature Engineering** • Identifying and selecting relevant features influencing flight fares. • Creating new features based on requirements and needs.	26/07/2024	29/07/2024
6.	**Data Analysis** • Analysing the relationship between independent and target features both statistically and visually.	29/07/2024	04/08/2024
7.	**Machine Learning Models Implementation** • Splitting the transformed dataset for training and testing purposes. • Applying appropriate encoding and scaling algorithms to the dataset. • Selecting and implementing appropriate models.	04/08/2024	11/08/2024
8.	**Models Evaluation & Tuning** • Evaluating, comparing, and tuning hyperparameters to improve the models' results.	11/08/2024	18/08/2024
9.	**Feature Importance** • Analyse the importance of all the features in influencing the price of flights. • Visualise all the results through different functions.	18/08/2024	25/08/2024
10.	**Documentation and Report Writing** • Documenting methodology, experiments, analysis, results, and conclusions. • Including appropriate tables and visualisations. • Preparing a presentation summarising key insights and recommendations.	25/08/2024	27/09/2024

Appendix B – Source Code

```
# Importing basic python libraries
import numpy as np
import pandas as pd
import datetime
import math

# Importing visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Importing splitting libraries
from scipy.stats import f_oneway
from sklearn.model_selection import StratifiedShuffleSplit

# Importing encoding libraries
from sklearn.preprocessing import OrdinalEncoder
from category_encoders import TargetEncoder

# Importing scaling libraries
from sklearn.preprocessing import RobustScaler

# Importing Pipeline libraries
from sklearn.pipeline import Pipeline

# Importing mean cross validation libraries
from sklearn.model_selection import cross_val_score

# Importing machine learning model libraries
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor


# Importing R2 Sccore and error calculation libraries
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

# Importing cross validation search libraries
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV

# Importing ensemble libraries
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.ensemble import VotingRegressor
from sklearn.ensemble import StackingRegressor

# Importing SHAP libraries
import shap
shap.initjs()


# Importing the dataset
business = pd.read_csv('../Dataset/business.csv')
economy = pd.read_csv('../Dataset/economy.csv')

# Including class of the flight in the dataset
business['class'] = 'Business'
economy['class'] = 'Economy'

# Concatenating the two datasets
df = pd.concat([economy, business], ignore_index=True)


# Checking the structure of the dataset
df.info()
```

```
# Checking for duplicate rows
print("Number of duplicate rows are:", df.duplicated(keep=False).sum())

# Displaying the duplicated rows
df[df.duplicated(keep=False)]


# Dropping the duplicated rows
df.drop_duplicates(inplace=True)

# Checking for duplicates again to be sure
print("Number of duplicate rows are: ", df.duplicated(keep=False).sum())


# Checking for missing / NaN values in the dataset
df.isnull().sum()


# Checking Data Types and sample values of each column
df.info()


# Checking a sample of price column before converting it to int64
df.sample(10)


# Removing ',' in the price column before converting it to int64
df['price'] = df['price'].str.replace(',', '').astype('int64')

# Checking a sample of price column after converting it to int64
df.sample(10)


# Viewing the unique values in all the columns
for col in df.columns:
    print(f'{col}: {df[col].unique()}')


# In the stop column, cleaning the stopover / layover data
df['stop'] = df['stop'].str.replace(r'1-stop[\s\S]*', '1-stop', regex=True)
df['stop'] = df['stop'].replace('non-stop ', 'Zero')
df['stop'] = df['stop'].apply(lambda x: 'One' if '1-stop' in x else x)
df['stop'] = df['stop'].replace('2+-stop', 'Two or More')


# Displaying the rows in the time_taken column where after first character, a decimal point is present
df[df['time_taken'].str.contains(r'^[0-9]\.', regex=True)]

# Deleting these observations to clean the data and also to standardise it
df = df[~df['time_taken'].str.contains(r'^[0-9]\.', regex=True)]


# Checking for negative values in the numeric columns
for col in df.select_dtypes(include=['int64', 'float64']).columns:
    print(f'{col}: {df[col].lt(0).sum()}')

# Removing the negative values from the numeric columns
for col in df.select_dtypes(include=['int64', 'float64']).columns:
    df = df[~df[col].lt(0)]


# Substracting date value with 10-02-2022 to get the number of days left for the flight
df['flight_days_left'] = df['date'].apply(lambda x: (datetime.datetime.strptime(x, '%d-%m-%Y')
                                                     - datetime.datetime.strptime('10-02-2022', '%d-%m-%Y')).days)

# Dropping the date column to avoid redundancy
df.drop(['date'], axis=1, inplace=True)
```

```
# In dep_time column, extracting hour and saving in a new dataset of hour column name
temp = pd.DataFrame(df['dep_time'].str.split(':', expand=True).to_numpy().astype(int), columns=['hour', 'minute'])

# Adjusting hour based on minute value
temp['adjusted_hour'] = temp.apply(lambda row: (row['hour'] - 1) if row['minute'] == 0 else row['hour'], axis=1)
temp['adjusted_hour'] = temp['adjusted_hour'].replace(-1, 23)

# Defining the bins and labels for the departure time
bin_edges = [0, 4, 8, 12, 16, 20, 24]
labels = ["Late Night", "Early Morning", "Morning", "Afternoon", "Evening", "Night"]

# Using pd.cut to categorize dep_time based on the adjusted hour
df['departure_time'] = pd.cut(temp['adjusted_hour'], bins=bin_edges, labels=labels, right=False, include_lowest=True)

# Removing rows where departure_time is null
df = df[~df['departure_time'].isnull()]

# Dropping the dep_time column
df.drop(['dep_time'], axis=1, inplace=True)


# In arr_time column, extracting hour and saving in a new dataset of hour column name
temp = pd.DataFrame(df['arr_time'].str.split(':', expand=True).to_numpy().astype(int), columns=['hour', 'minute'])

# Adjusting hour based on minute value
temp['adjusted_hour'] = temp.apply(lambda row: (row['hour'] - 1) if row['minute'] == 0 else row['hour'], axis=1)
temp['adjusted_hour'] = temp['adjusted_hour'].replace(-1, 23)

# Defining the bins and labels for the arrival time
bin_edges = [0, 4, 8, 12, 16, 20, 24]
labels = ["Late Night", "Early Morning", "Morning", "Afternoon", "Evening", "Night"]

# Using pd.cut to categorize dep_time based on the adjusted hour
df['arrival_time'] = pd.cut(temp['adjusted_hour'], bins=bin_edges, labels=labels, right=False, include_lowest=True)

# Removing rows where arrival_time is null
df = df[~df['arrival_time'].isnull()]

# Dropping the arr_time column
df.drop(['arr_time'], axis=1, inplace=True)


# Extracting the time taken from the time_taken column and save in a new dataset
temp = pd.DataFrame(df['time_taken'],columns=['time_taken'])

# Extracting the hours and minutes from the time_taken column
temp = temp['time_taken'].str.extract(r'(\d+)h (\d+)m')
temp.columns = ['hours', 'minutes']

# If minutes is null, then assigning 0 to it
temp['minutes'] = temp['minutes'].fillna(0)

# Dividing minutes by 60 in a new float column
temp['minutes_calculated'] = temp['minutes'].astype(float)/60

# Concatenating hours and minutes calculated to get the total time taken in hours and round it to 2 decimal places
temp['time_taken'] = temp['hours'].astype(float) + temp['minutes_calculated'].round(2)

# Dropping the hours and minutes columns
temp.drop(['hours', 'minutes', 'minutes_calculated'], axis=1, inplace=True)

# Replacing the time_taken column with the new time_taken column
df['time_taken'] = temp['time_taken']


# Concatenating the ch_code and num_code to get the flight number for every row
df['flight_number'] = df['ch_code'] + '-' + df['num_code'].astype(str)

# Dropping the ch_code and num_code columns
df.drop(['ch_code', 'num_code'], axis=1, inplace=True)


# Changing the column names for standardisation
df.columns = ['Airline Name', 'Departure City', 'Flight Duration', 'Number of Stops',
              'Destination City', 'Price', 'Seat Class', 'Flight Days Left', 'Departure Time', 'Destination Time', 'Flight Number']

# Rearranging the columns
df = df[['Airline Name', 'Flight Number', 'Seat Class', 'Departure City', 'Departure Time',  'Number of Stops', 'Destination City',
         'Destination Time', 'Flight Duration', 'Flight Days Left','Price']]
```

```
# Extracting the numeric columns
num_cols = df.select_dtypes(include=['int64', 'float64']).columns
print(num_cols)

# Determining the number of rows and columns for subplots
num_plots = len(num_cols)
num_cols_subplot = 2
num_rows_subplot = (num_plots + 1) // num_cols_subplot

# Creating boxplot of numeric columns in a single loop in a subplot
plt.figure(figsize=(20, 5 * num_rows_subplot))
for i, col in enumerate(num_cols):
    plt.subplot(num_rows_subplot, num_cols_subplot, i + 1)
    sns.boxplot(x=df[col])

    # Calculating IQR bounds
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = max(Q1 - 1.5 * IQR, 0)
    upper_bound = Q3 + 1.5 * IQR

    # Plotting IQR bounds
    plt.axvline(lower_bound, color='r', linestyle='--', label='Lower Bound')
    plt.axvline(upper_bound, color='g', linestyle='--', label='Upper Bound')

    # Annotating the bounds
    plt.text(lower_bound, plt.ylim()[1] * 0.95, f'Lower Bound: {lower_bound:.2f}', color='r', ha='left')
    plt.text(upper_bound, plt.ylim()[1] * 0.95, f'Upper Bound: {upper_bound:.2f}', color='g', ha='right')

    plt.title(f'{col} Boxplot')
    plt.legend(loc='lower right')

# Adjusting the layout and displaying the plots
plt.tight_layout()
plt.show()


# Masking flight duration greater than 30.30 hours of flight duration
mask = df['Flight Duration'] >= 30.30

# Displaying the rows where flight duration is greater than 30.30 of flight duration
df[mask]


# Masking price greather than INR 99,128.00
mask =df['Price'] >= 99128.00

# Displaying the rows where price is greater than INR 99,128.00
df[mask]


# Extracting the numeric columns
num_cols = df.select_dtypes(include=['int64', 'float64']).columns

# Extracting the categorical columns
cat_cols = df.select_dtypes(include=['object', 'category']).columns


# Describing all columns
df.describe(include='all')


# Calculating median and mode of quantitative and qualitative features in the dataset
print("Median of quantitative features in the dataset:", df[num_cols].median())
print("Mode of qualitative features in the dataset:", df[cat_cols].mode().iloc[0])


# Calculating and displaying skewness of quantitative features in the dataset
print("Skewness of quantitative features in the dataset:", df[num_cols].skew())
```

```
# Value Distribution of Quantitative Variables
plt.figure(figsize=(20,10))
for i, col in enumerate(num_cols):
    plt.subplot(2, 2, i+1)
    plt.ticklabel_format(style='plain')
    count, bins, patches = plt.hist(df[col], bins=10, edgecolor='black')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.title(f'Histogram of {col}')
    for count, bin, patch in zip(count, bins, patches):
        height = patch.get_height()
        bin_center = bin + (bins[1] - bins[0]) / 2
        plt.text(bin_center, height + 3, str(int(count)), ha = 'center', va='bottom')
plt.show()



# Plotting Quantitative Variables vs Price Associations
for col in ['Flight Duration', 'Flight Days Left']:
    plt.figure(figsize=(20, 10))
    sns.scatterplot(x=col, y='Price', data = df, hue='Seat Class')
    plt.title(f'{col} vs Price')
    plt.xlabel(col)
    plt.ylabel('Price')
    plt.show()


# Plotting Quantitative Variables vs Price Associations
for col in ['Flight Duration', 'Flight Days Left']:
    plt.figure(figsize=(20, 10))
    sns.scatterplot(x=col, y='Price', hue='Seat Class', data = df.groupby(['Seat Class', col])['Price'].mean().reset_index())
    plt.title(f'{col} vs Price')
    plt.xlabel(col)
    plt.ylabel('Price')
    plt.show()


# Seting up the matplotlib figure with a 3x3 grid of subplots
fig, axes = plt.subplots(3, 3, figsize=(20, 15))
fig.tight_layout(pad=5.0)

# Increasing the space between subplots
fig.subplots_adjust(hspace=0.5, wspace=0.2)

# Flattening the axes array for easy iteration
axes_flat = axes.flatten()

# Plotting the countplot for each categorical column
for i, col in enumerate(cat_cols):

    if cat_cols[i] == 'Flight Number':
        ax = axes_flat[i]
        top_flight_numbers = df[col].value_counts().nlargest(10).to_frame().reset_index()
        top_flight_numbers.columns = [col, 'Frequency']
        sns.barplot(y=col, x='Frequency', data=top_flight_numbers, color='C0', ax=ax)
        ax.set_label(col)
        ax.set_ylabel('Frequency')
        ax.set_xlabel(f'Frequency distribution of {col}')
        ax.tick_params(axis='y', rotation=0)
        for p in ax.patches:
            ax.annotate(f'{int(p.get_width())}', (p.get_width(), p.get_y() + p.get_height() / 2.),
                        ha='center', va='center', xytext=(0, 10), textcoords='offset points')
    else:
        ax = axes_flat[i]
        sns.countplot(x=col, data=df, color='C0', ax=ax)
        ax.set_xlabel(col)
        ax.set_ylabel('Frequency')
        ax.set_title(f'Frequency distribution of {col}')
        ax.tick_params(axis='x', rotation=90)
        for p in ax.patches:
            ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height()),
                        ha='center', va='center', xytext=(0, 10), textcoords='offset points')

# Hiding the unused subplot
axes_flat[-1].set_visible(False)

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

```
# Setting up the matplotlib figure with a grid of subplots
fig, axes = plt.subplots(2, 4, figsize=(30, 10))
fig.tight_layout(pad=5.0)
fig.subplots_adjust(hspace=0.6, wspace=0.4)  # Increase spacing between plots
axes_flat = axes.flatten()

# Plotting the barplot for each categorical column
for i, col in enumerate(['Airline Name', 'Seat Class', 'Departure City',
                         'Destination City', 'Departure Time', 'Destination Time',
                         'Number of Stops']):
    ax = axes_flat[i]
    sns.barplot(x=col, y='Price', data=df.groupby(col)['Price'].mean().reset_index(),
                color="C0", ax=ax)
    ax.set_xlabel(col)
    ax.set_ylabel('Price')
    ax.set_title(f'Relationship between {col} and Price')
    ax.tick_params(axis='x', rotation=90)
    for p in ax.patches:
        ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', xytext=(0, 10), textcoords='offset points')


# Hide any unused subplots
for j in range(i + 1, len(axes_flat)):
    fig.delaxes(axes_flat[j])

# Displaying the plot
plt.show()


# Calculating ANOVA F-statistic for categorical columns
def anova_test(df, feature, target):
    groups = [df[target][df[feature] == category] for category in df[feature].unique()]
    f_stat, p_val = f_oneway(*groups)
    return f_stat, p_val

# Evaluating the ANOVA test for all categorical columns
anova_results = {}
for feature in cat_cols:
    f_stat, p_val = anova_test(df, feature, 'Price')
    anova_results[feature] = (f_stat, p_val)

# Converting the results to a DataFrame for easier ranking and plotting
anova_df = pd.DataFrame.from_dict(anova_results, orient='index', columns=['F-statistic', 'p-value'])

# Sorting the DataFrame by F-statistic for better visualization
anova_df = anova_df.sort_values(by='F-statistic', ascending=False)

# Printing and ranking all the features
print("Ranking of features based on F-statistic:")
for rank, (feature, row) in enumerate(anova_df.iterrows(), start=1):
    print(f"{rank}. {feature} - F-statistic: {row['F-statistic']}, p-value: {row['p-value']}")

# Best feature based on F-statistic saved in a new variable
best_feature = anova_df.index[0]

# Plotting the F-statistics
plt.figure(figsize=(12, 6))
sns.barplot(x=anova_df.index, y='F-statistic', data=anova_df, palette='viridis')
plt.xticks(rotation=45)
plt.title('ANOVA F-statistics for Categorical Features')
plt.xlabel('Categorical Features')
plt.ylabel('F-statistic')
plt.show()


# Checking distribution of the feature with highest F-statistic
seat_class_distribution = df[best_feature].value_counts(normalize=True)
print(f"Distribution of {best_feature}:")
print(seat_class_distribution)

# Plotting the distribution of the best feature
plt.figure(figsize=(8, 4))
sns.countplot(x=best_feature, data=df, palette='viridis')
plt.title(f'Distribution of {best_feature}')
plt.xlabel(best_feature)
plt.ylabel('Frequency')
plt.show()
```

```
# Initialising the required variables of predictor, categorical feature,
# random state and test size
predictor = 'Price'
categorical_feature = best_feature
rs = 42
ts = 0.20


# Resetting the index of the DataFrame to ensure continuous indices
df = df.reset_index(drop=True)

# Initialising StratifiedShuffleSplit with the desired test size
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

# Performing the split using the categorical feature
for train_index, test_index in split.split(df, df[categorical_feature]):
    df_train = df.loc[train_index].reset_index(drop=True)
    df_test = df.loc[test_index].reset_index(drop=True)

# Verifying the distribution in the splits
train_seat_class_distribution = df_train[categorical_feature].value_counts(normalize=True)
test_seat_class_distribution = df_test[categorical_feature].value_counts(normalize=True)

print(f"Train set distribution of {categorical_feature}:")
print(train_seat_class_distribution)
print(f"Test set distribution of {categorical_feature}")
print(test_seat_class_distribution)

# Plotting the distributions
fig, axes = plt.subplots(1, 2, figsize=(12, 6), sharey=True)
sns.countplot(x=categorical_feature, data=df_train, palette='viridis', ax=axes[0], order=['Economy', 'Business'])
axes[0].set_title(f'Train Set Distribution of {categorical_feature}')
axes[0].set_xlabel(categorical_feature)
axes[0].set_ylabel('Frequency')

sns.countplot(x='Seat Class', data=df_test, palette='viridis', ax=axes[1], order=['Economy', 'Business'])
axes[1].set_title(f'Test Set Distribution of {categorical_feature}')
axes[1].set_xlabel(categorical_feature)
axes[1].set_ylabel('Frequency')

plt.show()


# Splitting the dataset into the Training set and Test set
X_train = df_train.drop(predictor, axis=1)
y_train = df_train[predictor]

X_test = df_test.drop(predictor, axis=1)
y_test = df_test[predictor]

# Displaying the shape of the training and test sets
X_train.shape, y_train.shape, X_test.shape, y_test.shape


# Listing ordinal categorical columns
ordinal_cat_cols = ['Seat Class', 'Departure Time', 'Destination Time', 'Number of Stops']

# Defining the categories for each feature
seat_class_categories = ['Business', 'Economy']
time_categories = ['Late Night', 'Early Morning', 'Morning', 'Afternoon', 'Evening', 'Night']
stops_categories = ['Zero', 'One', 'Two or More']

# Creating the encoder with the specified categories
encoder = OrdinalEncoder(categories=[seat_class_categories, time_categories, time_categories, stops_categories])

# Fitting and transforming the training data
X_train[ordinal_cat_cols] = encoder.fit_transform(X_train[ordinal_cat_cols])

# Transforming the test data
X_test[ordinal_cat_cols] = encoder.transform(X_test[ordinal_cat_cols])


# Listing Nominal categorical columns
nominal_cat_cols = ['Airline Name', 'Flight Number', 'Departure City', 'Destination City']

# Initialising the Target Encoder
te = TargetEncoder()

# Applying Target Encoding to the categorical columns
X_train[nominal_cat_cols] = te.fit_transform(X_train[nominal_cat_cols], y_train)
X_test[nominal_cat_cols] = te.transform(X_test[nominal_cat_cols])
```

```
# Initialising the RobustScaler
rs = RobustScaler()

# Applying RobustScaler to the X_train and X_test
X_train = rs.fit_transform(X_train)
X_test = rs.transform(X_test)


# Creating a Linear Regression model pipeline
lr_pipeline = Pipeline([
    ('linear_regression', LinearRegression())
])

# Fitting the model pipeline
lr_pipeline.fit(X_train, y_train)

# Predicting the test set results
y_pred = lr_pipeline.predict(X_test)

# Evaluating model performance
lr_train_Score = lr_pipeline.score(X_train, y_train)
lr_Test_Score = lr_pipeline.score(X_test, y_test)

# Displaying the scores
print('Linear Regression Accuracy Train Score is: {}'.format(lr_train_Score))
print('Linear Regression Accuracy Test Score is: {}'.format(lr_Test_Score))

# Calcuating evaluation metrics
lr_MAE = mean_absolute_error(y_test, y_pred)
lr_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
lr_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
lr_MSE = mean_squared_error(y_test, y_pred)

# Displaying the evaluation metrics
print('Mean Absolute Error:', lr_MAE)
print('Root Mean Squared Error:', lr_RMSE)
print('Mean Squared Error:', lr_MSE)
print('Mean Absolute Percentage Error:', lr_MAPE)


# Creating a Decision Tree Regressor instance via pipeline
dtr_pipeline = Pipeline([
    ('decision tree', DecisionTreeRegressor(max_depth=2))
])

# Fitting the model
dtr_pipeline.fit(X_train, y_train)

# Evaluating model performance
dtr_train_Score = dtr_pipeline.score(X_train, y_train)
dtr_Test_Score = dtr_pipeline.score(X_test, y_test)


# Displaying the accuracy scores
print('Decision Tree Regression Accuracy Train Score is: {}'.format(dtr_train_Score))
print('Decision Tree Regression Accuracy Test Score is: {}'.format(dtr_Test_Score))

# Calculating the predictions
y_pred = dtr_pipeline.predict(X_test)

# Calcuating evaluation metrics
dtr_MAE = mean_absolute_error(y_test, y_pred)
dtr_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
dtr_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
dtr_MSE = mean_squared_error(y_test, y_pred)

# Displaying the evaluation metrics
print('Mean Absolute Error:', dtr_MAE)
print('Root Mean Squared Error:', dtr_RMSE)
print('Mean Squared Error:', dtr_MSE)
print('Mean Absolute Percentage Error:', dtr_MAPE)
```

```
# Creating a Random Forest Regressor Instance via pipeline
rfr_pipeline = Pipeline([
    ('random forest', RandomForestRegressor(max_depth=2))
])

# Fitting the Model
rfr_pipeline.fit(X_train, y_train)

# Evaluating model performance
rfr_train_Score = rfr_pipeline.score(X_train, y_train)
rfr_Test_Score = rfr_pipeline.score(X_test, y_test)

# Displaying the scores
print('Random Forest Regression Accuracy Train Score is: {}'.format(rfr_train_Score))
print('Random Forest Regression Accuracy Test Score is: {}'.format(rfr_Test_Score))

# Calculating the predictions
y_pred = rfr_pipeline.predict(X_test)

# Calcuating evaluation metrics
rfr_MAE = mean_absolute_error(y_test, y_pred)
rfr_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
rfr_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
rfr_MSE = mean_squared_error(y_test, y_pred)

# Displaying the evaluation metrics
print('Mean Absolute Error:', rfr_MAE)
print('Root Mean Squared Error:', rfr_RMSE)
print('Mean Squared Error:', rfr_MSE)
print('Mean Absolute Percentage Error:', rfr_MAPE)


# Creating pipeline for K-Nearest Neighbours Regressor
knnr_pipeline = Pipeline([
    ('knn', KNeighborsRegressor(n_neighbors=100))
])

# Fitting the K-Nearest Neighbours Regressor Model
knnr_pipeline.fit(X_train, y_train)

# Evaluating the model performance
knnr_train_Score = knnr_pipeline.score(X_train, y_train)
knnr_Test_Score = knnr_pipeline.score(X_test, y_test)

# Displaying the scores
print('KNN Regression Accuracy Train Score is: {}'.format(knnr_train_Score))
print('KNN Regression Accuracy Test Score is: {}'.format(knnr_Test_Score))

# Calculating the predictions
y_pred = knnr_pipeline.predict(X_test)

# Calcuating the evaluation metrics
knnr_MAE = mean_absolute_error(y_test, y_pred)
knnr_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
knnr_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
knnr_MSE = mean_squared_error(y_test, y_pred)

# Displaying the evaluation metrics
print('Mean Absolute Error:', knnr_MAE)
print('Root Mean Squared Error:', knnr_RMSE)
print('Mean Squared Error:', knnr_MSE)
print('Mean Absolute Percentage Error:', knnr_MAPE)
```

```
# Creating a Gradient Boosting Regressor instance via pipeline
gbr_pipeline = Pipeline([
    ('gradient boosting', GradientBoostingRegressor(n_estimators=5, learning_rate=0.4))
])

# Fitting the Model
gbr_pipeline.fit(X_train, y_train)

# Evaluating model performance
gbr_train_Score = gbr_pipeline.score(X_train, y_train)
gbr_Test_Score = gbr_pipeline.score(X_test, y_test)

# Displaying the scores
print('Gradient Boosting Regressionn Accuracy Train Score is: {}'.format(gbr_train_Score))
print('Gradient Boosting Regression Accuracy Test Score is: {}'.format(gbr_Test_Score))

# Calculating the predictions
y_pred = gbr_pipeline.predict(X_test)

# Calcuating evaluation metrics
gbr_MAE = mean_absolute_error(y_test, y_pred)
gbr_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
gbr_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
gbr_MSE = mean_squared_error(y_test, y_pred)

# Displaying the evaluation metrics
print('Mean Absolute Error:', gbr_MAE)
print('Root Mean Squared Error:', gbr_RMSE)
print('Mean Squared Error:', gbr_MSE)
print('Mean Absolute Percentage Error:', gbr_MAPE)


# Displaying all the scores of the models in a dataframe
im_scores = pd.DataFrame({
    'Model': ['Linear Regression', 'Decision Tree Regressor', 'Random Forest Regressor',
              'K-Nearest Neighbours Regressor', 'Gradient Boosting Regressor'],
    'R2 Train Score': [lr_train_Score, dtr_train_Score, rfr_train_Score, knnr_train_Score, gbr_train_Score],
    'R2 Test Score': [lr_Test_Score, dtr_Test_Score, rfr_Test_Score, knnr_Test_Score, gbr_Test_Score],
    'Mean Absolute Error': [lr_MAE, dtr_MAE, rfr_MAE, knnr_MAE, gbr_MAE],
    'Root Mean Squared Error': [lr_RMSE, dtr_RMSE, rfr_RMSE, knnr_RMSE, gbr_RMSE],
    'Mean Squared Error': [lr_MSE, dtr_MSE, rfr_MSE, knnr_MSE, gbr_MSE],
    'Mean Absolute Percentage Error': [lr_MAPE, dtr_MAPE, rfr_MAPE, knnr_MAPE, gbr_MAPE]
})

# Displaying the scores
im_scores


# Displaying True vs Predicted Analysis scatterplots of all the models
plt.figure(figsize=(15, 10))

# Linear Regression
plt.subplot(2, 3, 1)
sns.scatterplot(x=y_test, y=lr_pipeline.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Linear Regression')
plt.xlabel('True')
plt.ylabel('Predicted')
```

```
# Decision Tree Regressor
plt.subplot(2, 3, 2)
sns.scatterplot(x=y_test, y=dtr_pipeline.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Decision Tree Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')

# Random Forest Regressor
plt.subplot(2, 3, 3)
sns.scatterplot(x=y_test, y=rfr_pipeline.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Random Forest Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')

# KNN Regressor
plt.subplot(2, 3, 4)
sns.scatterplot(x=y_test, y=knnr_pipeline.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('K-Nearest Neighbours Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')

# Gradient Boosting Regressor
plt.subplot(2, 3, 5)
sns.scatterplot(x=y_test, y=gbr_pipeline.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Gradient Boosting Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')

# Adjusting the layout and displaying the plots
plt.tight_layout()
plt.show()


# Displaying True v Predicted scatterplots of all the models in a single plot
plt.figure(figsize=(20, 10))
plt.scatter(y_test, lr_pipeline.predict(X_test), label='Linear Regression', color='blue')
plt.scatter(y_test, dtr_pipeline.predict(X_test), label='Decision Tree Regressor', color='red')
plt.scatter(y_test, rfr_pipeline.predict(X_test), label='Random Forest Regressor', color='green')
plt.scatter(y_test, knnr_pipeline.predict(X_test), label='K-Nearest Neighbours Regressor', color='purple')
plt.scatter(y_test, gbr_pipeline.predict(X_test), label='Gradient Boosting Regressor', color='orange')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color = 'black')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted for all models')
plt.legend()
plt.show()


# Declaring relevant hyperparameters for RandomizedSearchCV for Ridge regression
lr_cv_parameters = {
    'fit_intercept': [True, False],
    'alpha': [1e-15, 1e-10, 1e-8, 1e-3, 1e-2, 1, 5, 10, 20, 30, 35, 40, 45, 50, 55, 100],
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']
}

# Creating an instance for Ridge Regression
ridge = Ridge()

# Creating an instance for RandomizedSearchCV
ridge_cv = RandomizedSearchCV(estimator=ridge, param_distributions=lr_cv_parameters,
                              n_iter=10, cv=5, n_jobs=-1, verbose=1, return_train_score=True,
                              random_state=42)

# Fitting the model
ridge_cv.fit(X_train, y_train)

# Displaying all randomized search results in a dataframe
lr_cv_results = pd.DataFrame(ridge_cv.cv_results_)
lr_cv_results.sort_values(by='rank_test_score', inplace=True)
lr_cv_results = lr_cv_results[['params', 'mean_train_score', 'std_train_score',
                               'mean_test_score', 'std_test_score', 'rank_test_score']].head(5).reset_index(drop=True)

# Displaying the randomized search results
lr_cv_results
```

```
# Plotting the mean test and train scores on a line plot
plt.figure(figsize=(10, 5))

plt.plot(lr_cv_results['mean_train_score'], marker='o', label='Train Score')
plt.plot(lr_cv_results['mean_test_score'], marker='o', label='Test Score')

# Adding title and labels
plt.title('Mean Train and Test Scores vs. Hyperparameter Combinations')

# Labeling X and Y labels
plt.xlabel('Hyperparameter Combination')
plt.ylabel('Mean Score')

# Adding legend
plt.legend()

# Displaying the plot
plt.show()


# Displaying the best parameters
lr_cv_best_params = ridge_cv.best_params_
lr_cv_best_fit_intercept = lr_cv_best_params['fit_intercept']
lr_cv_best_alpha = lr_cv_best_params['alpha']
lr_cv_best_solver = lr_cv_best_params['solver']

# Fitting the model with the best parameters
lr_pipeline = Pipeline([
    ('ridge_cv', Ridge(fit_intercept=lr_cv_best_fit_intercept, alpha=lr_cv_best_alpha, solver=lr_cv_best_solver))
])

# Fitting the model
lr_pipeline.fit(X_train, y_train)

# Evaluate model performance
lr_cv_train_Score = lr_pipeline.score(X_train, y_train)
lr_cv_Test_Score = lr_pipeline.score(X_test, y_test)

# Displaying the scores
print('Linear Regression CV Train Score is:', lr_cv_train_Score)
print('Linear Regression CV Test Score is:', lr_cv_Test_Score)

# Calculating the error metrics
y_pred = lr_pipeline.predict(X_test)
lr_cv_MAE = mean_absolute_error(y_test, y_pred)
lr_cv_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
lr_cv_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
lr_cv_MSE = mean_squared_error(y_test, y_pred)

# Displaying the error metrics
print('Mean Absolute Error:', lr_cv_MAE)
print('Root Mean Squared Error:', lr_cv_RMSE)
print('Mean Squared Error:', lr_cv_MSE)
print('Mean Absolute Percentage Error:', lr_cv_MAPE)


# Creating a parameter grid for RandomizedSearchCV for Decision Tree Regressor
dtr_cv_parameters = {
    'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8]
}

# Creating an instance for Decision Tree Regressor
dtr = DecisionTreeRegressor()

# Creating an instance for RandomizedSearchCV
dtr_cv = RandomizedSearchCV(estimator=dtr, param_distributions=dtr_cv_parameters,
                            n_iter=10, cv=5, n_jobs=-1, verbose=1, return_train_score=True,
                            random_state=42)

# Fitting the model
dtr_cv.fit(X_train, y_train)

# Displaying all randomized search results in a dataframe
dtr_cv_results = pd.DataFrame(dtr_cv.cv_results_)
dtr_cv_results.sort_values(by='rank_test_score', inplace=True)
dtr_cv_results = dtr_cv_results[['params', 'mean_train_score', 'std_train_score', 'mean_test_score',
                                 'std_test_score', 'rank_test_score']].head(5).reset_index(drop=True)

# Displaying the randomized search results
dtr_cv_results
```

```
# Plotting the mean test and train scores on a line plot
plt.figure(figsize=(10, 5))

plt.plot(dtr_cv_results['mean_train_score'], marker='o', label='Train Score')
plt.plot(dtr_cv_results['mean_test_score'], marker='o', label='Test Score')

# Adding title and labels
plt.title('Mean Train and Test Scores vs. Hyperparameter Combinations')


# X and Y labels
plt.xlabel('Hyperparameter Combination')
plt.ylabel('Mean Score')

# Adding legend
plt.legend()

# Displaying the plot
plt.show()


# Applying the best parameters to the Decision Tree Regressor
dtr_cv_best_params = dtr_cv.best_params_
dtr_cv_best_max_depth = dtr_cv_best_params['max_depth']
dtr_cv_best_min_samples_split = dtr_cv_best_params['min_samples_split']
dtr_cv_best_min_samples_leaf = dtr_cv_best_params['min_samples_leaf']

# Creating a Decision Tree Regressor Instance via Pipeline with the best parameters
dt_pipeline = Pipeline([
    ('decision tree', DecisionTreeRegressor(max_depth=dtr_cv_best_max_depth,
                                            min_samples_split=dtr_cv_best_min_samples_split,
                                            min_samples_leaf=dtr_cv_best_min_samples_leaf))
])

# Fitting the Decision Tree regressor Model
dt_pipeline.fit(X_train, y_train)

# Calculating Accuracy Scores of the Decision Tree Regressor Model
dtr_cv_train_Score = dt_pipeline.score(X_train, y_train)
dtr_cv_Test_Score = dt_pipeline.score(X_test, y_test)

# Displaying the Accuracy Scores
print('Decision Tree Regressor CV Train Score is:', dtr_cv_train_Score)
print('Decision Tree Regressor CV Test Score is:', dtr_cv_Test_Score)

# Calculating the error metrics
y_pred = dt_pipeline.predict(X_test)

dtr_cv_MAE = mean_absolute_error(y_test, y_pred)
dtr_cv_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
dtr_cv_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
dtr_cv_MSE = mean_squared_error(y_test, y_pred)

# Displaying the error metrics
print('Mean Absolute Error:', dtr_cv_MAE)
print('Root Mean Squared Error:', dtr_cv_RMSE)
print('Mean Squared Error:', dtr_cv_MSE)
print('Mean Absolute Percentage Error:', dtr_cv_MAPE)


# Declaring hyperparameters for RandomizedSearchCV for Random Forest Regressor
rfr_cv_parameters = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8]
}

# Creating an instance for Random Forest Regressor
rfr = RandomForestRegressor()
```

```
# Declaring hyperparameters for RandomizedSearchCV for Random Forest Regressor
rfr_cv_parameters = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8]
}

# Creating an instance for Random Forest Regressor
rfr = RandomForestRegressor()

# Creating an instance for RandomizedSearchCV
rfr_cv = RandomizedSearchCV(estimator=rfr, param_distributions=rfr_cv_parameters,
                            n_iter=10, cv=5, n_jobs=-1, verbose=1, return_train_score=True,
                            random_state=42)

# Fitting the model
rfr_cv.fit(X_train, y_train)

# Displaying all randomized search results in a dataframe
rfr_cv_results = pd.DataFrame(rfr_cv.cv_results_)
rfr_cv_results.sort_values(by='rank_test_score', inplace=True)

# Displaying the randomized search results
rfr_cv_results = rfr_cv_results[['params','mean_train_score', 'std_train_score',
                                 'mean_test_score', 'std_test_score',
                                 'rank_test_score']].head(5).reset_index(drop=True)
rfr_cv_results


# Plotting the mean test and train scores on a line plot
plt.figure(figsize=(10, 5))

plt.plot(rfr_cv_results['mean_train_score'], marker='o', label='Train Score')
plt.plot(rfr_cv_results['mean_test_score'], marker='o', label='Test Score')

# Adding title and labels
plt.title('Mean Train and Test Scores vs. Hyperparameter Combinations')

# X and Y labels
plt.xlabel('Hyperparameter Combination')
plt.ylabel('Mean Score')

# Adding legend
plt.legend()

# Displaying the plot
plt.show()


# Applying the best parameters to the Random Forest Regressor
rfr_cv_best_params = rfr_cv.best_params_
rfr_cv_best_n_estimators = rfr_cv_best_params['n_estimators']
rfr_cv_best_max_depth = rfr_cv_best_params['max_depth']
rfr_cv_best_min_samples_split = rfr_cv_best_params['min_samples_split']
rfr_cv_best_min_samples_leaf = rfr_cv_best_params['min_samples_leaf']


# Fitting the Random Forest Regressor Model
rf_pipeline.fit(X_train, y_train)

# Calculating the accuracy scores of the Random Forest Regressor Model
rfr_cv_train_Score = rf_pipeline.score(X_train, y_train)
rfr_cv_Test_Score = rf_pipeline.score(X_test, y_test)

# Displaying the accuracy scores
print('Random Forest Regressor Train Score is:', rfr_cv_train_Score)
print('Random Forest Regressor Test Score is:', rfr_cv_Test_Score)

# Calculate mean squared error of random forest regressor model
y_pred = rf_pipeline.predict(X_test)

rfr_cv_MAE = mean_absolute_error(y_test, y_pred)
rfr_cv_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
rfr_cv_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
rfr_cv_MSE = mean_squared_error(y_test, y_pred)

# Displaying the error metrics
print('Mean Absolute Error:', rfr_cv_MAE)
print('Root Mean Squared Error:', rfr_cv_RMSE)
print('Mean Squared Error:', rfr_cv_MSE)
print('Mean Absolute Percentage Error:', rfr_cv_MAPE)
```

```
# Declaring hyperparameters for RandomizedSearchCV for K-Nearest Neighbours Regressor
knnr_cv_parameters = {
    'n_neighbors': np.arange(1, 20, 1),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

# Creating an instance for K-Nearest Neighbours Regressor
knn = KNeighborsRegressor()

# Creating an instance for RandomizedSearchCV
knn_cv = RandomizedSearchCV(estimator=knn, param_distributions=knnr_cv_parameters,
                            n_iter=10, cv=5, n_jobs=-1, verbose=1,
                            return_train_score=True, random_state=42)

# Fitting the model
knn_cv.fit(X_train, y_train)

# Displaying all randomized search results in a dataframe
knn_cv_results = pd.DataFrame(knn_cv.cv_results_)
knn_cv_results.sort_values(by='rank_test_score', inplace=True)

# Displaying the randomized search results
knn_cv_results = knn_cv_results[['params', 'mean_train_score', 'std_train_score',
                                 'mean_test_score', 'std_test_score',
                                 'rank_test_score']].head(5).reset_index(drop=True)

# Displaying the randomized search results
knn_cv_results


# Plotting the mean test and train scores on a line plot
plt.figure(figsize=(10, 5))

plt.plot(knn_cv_results['mean_train_score'], marker='o', label='Train Score')
plt.plot(knn_cv_results['mean_test_score'], marker='o', label='Test Score')


# Adding title and labels
plt.title('Mean Train and Test Scores vs. Hyperparameter Combinations')

# X and Y labels
plt.xlabel('Hyperparameter Combination')
plt.ylabel('Mean Score')

# Adding legend
plt.legend()

# Displaying the plot
plt.show()


# Applying the best parameters to the Decision Tree Regressor
dtr_cv_best_params = dtr_cv.best_params_
dtr_cv_best_max_depth = dtr_cv_best_params['max_depth']
dtr_cv_best_min_samples_split = dtr_cv_best_params['min_samples_split']
dtr_cv_best_min_samples_leaf = dtr_cv_best_params['min_samples_leaf']

# Creating a Decision Tree Regressor Instance via Pipeline with the best parameters
dt_pipeline = Pipeline([
    ('decision tree', DecisionTreeRegressor(max_depth=dtr_cv_best_max_depth,
                                            min_samples_split=dtr_cv_best_min_samples_split,
                                            min_samples_leaf=dtr_cv_best_min_samples_leaf))
])

# Fitting the Decision Tree regressor Model
dt_pipeline.fit(X_train, y_train)

# Calculating Accuracy Scores of the Decision Tree Regressor Model
dtr_cv_train_Score = dt_pipeline.score(X_train, y_train)
dtr_cv_Test_Score = dt_pipeline.score(X_test, y_test)
```

```
# Displaying the Accuracy Scores
print('Decision Tree Regressor CV Train Score is:', dtr_cv_train_Score)
print('Decision Tree Regressor CV Test Score is:', dtr_cv_Test_Score)

# Calculating the error metrics
y_pred = dt_pipeline.predict(X_test)

dtr_cv_MAE = mean_absolute_error(y_test, y_pred)
dtr_cv_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
dtr_cv_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
dtr_cv_MSE = mean_squared_error(y_test, y_pred)

# Displaying the error metrics
print('Mean Absolute Error:', dtr_cv_MAE)
print('Root Mean Squared Error:', dtr_cv_RMSE)
print('Mean Squared Error:', dtr_cv_MSE)
print('Mean Absolute Percentage Error:', dtr_cv_MAPE)


# Declaring hyperparameters for RandomizedSearchCV for Gradient Boosting Regressor
gbr_cv_parameters = {
    'n_estimators' : [100, 200, 300, 400, 500],
    'max_depth' : [2, 3, 4, 5, 6, 7, 8, 9, 10],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
}


# Creating an instance for Gradient Boosting Regressor
gbr = GradientBoostingRegressor()

# Creating an instance for RandomizedSearchCV
gbr_cv = RandomizedSearchCV(estimator=gbr, param_distributions=gbr_cv_parameters,
                            n_iter=10, cv=5, n_jobs=-1, verbose=1,
                            return_train_score=True, random_state=42)

# Fitting the model
gbr_cv.fit(X_train, y_train)

# Displaying all randomized search results in a dataframe
gbr_cv_results = pd.DataFrame(gbr_cv.cv_results_)
gbr_cv_results.sort_values(by='rank_test_score', inplace=True)

# Displaying the randomized search results
gbr_cv_results = gbr_cv_results[['params','mean_train_score', 'std_train_score',
                                 'mean_test_score', 'std_test_score',
                                 'rank_test_score']].head(5).reset_index(drop=True)

# Displaying the randomized search results
gbr_cv_results


# Plotting the mean test and train scores on a line plot
plt.figure(figsize=(10, 5))

plt.plot(gbr_cv_results['mean_train_score'], marker='o', label='Train Score')
plt.plot(gbr_cv_results['mean_test_score'], marker='o', label='Test Score')

# Adding title and labels
plt.title('Mean Train and Test Scores vs. Hyperparameter Combinations')

# X and Y labels
plt.xlabel('Hyperparameter Combination')
plt.ylabel('Mean Score')

# Adding legend
plt.legend()

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

```
# Applying the best parameters to the Gradient Boosting Regressor
gbr_cv_best_params = gbr_cv.best_params_
gbr_cv_best_n_estimators = gbr_cv_best_params['n_estimators']
gbr_cv_best_max_depth = gbr_cv_best_params['max_depth']
gbr_cv_best_learning_rate = gbr_cv_best_params['learning_rate']

# Creating a Gradient Boosting Regressor Instance via Pipeline
gb_pipeline = Pipeline([
    ('gradient boosting', GradientBoostingRegressor(n_estimators=gbr_cv_best_n_estimators,
                                                    max_depth=gbr_cv_best_max_depth,
                                                    learning_rate=gbr_cv_best_learning_rate))
])

# Fitting the model
gb_pipeline.fit(X_train, y_train)


# Calculate error metrics of Gradient Boosting Regressor
gbr_cv_train_Score = gb_pipeline.score(X_train, y_train)
gbr_cv_Test_Score = gb_pipeline.score(X_test, y_test)

# Displaying the accuracy scores
print('Gradient Boosting Regressor Train Score is:', gbr_cv_train_Score)
print('Gradient Boosting Regressor Test Score is:', gbr_cv_Test_Score)

# Calculating mean squared error of random forest regressor
y_pred = gb_pipeline.predict(X_test)

gbr_cv_MAE = mean_absolute_error(y_test, y_pred)
gbr_cv_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
gbr_cv_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
gbr_cv_MSE = mean_squared_error(y_test, y_pred)

# Displaying the evaluation metrics
print('Mean Absolute Error:', gbr_cv_MAE)
print('Root Mean Squared Error:', gbr_cv_RMSE)
print('Mean Squared Error:', gbr_cv_MSE)
print('Mean Absolute Percentage Error:', gbr_cv_MAPE)


# Disaplying True vs Predicted Analysis of the improved models through
# RandomizedSearchCV
plt.figure(figsize=(15, 10))

# Linear Regression
plt.subplot(2, 3, 1)
sns.scatterplot(x=y_test, y=ridge_cv.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Linear Regression')
plt.xlabel('True')
plt.ylabel('Predicted')

# Decision Tree Regressor
plt.subplot(2, 3, 2)
sns.scatterplot(x=y_test, y=dtr_cv.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Decision Tree Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')


# Random Forest Regressor
plt.subplot(2, 3, 3)
sns.scatterplot(x=y_test, y=rfr_cv.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Random Forest Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')

# KNN Regressor
plt.subplot(2, 3, 4)
sns.scatterplot(x=y_test, y=knn_cv.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('K-Nearest Neighbours Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')
```

```
# Gradient Boosting Regressor
plt.subplot(2, 3, 5)
sns.scatterplot(x=y_test, y=gbr_cv.predict(X_test))
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Gradient Boosting Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')

# Adjusting the layout and displaying the plots
plt.tight_layout()
plt.show()


# Comparing the scores of of the models with and without RandomizedSearchCV()
# algorithm implementation
im_im_cv_scores = pd.DataFrame({
    'Model': ['Linear Regression', 'Linear Regression CV', 'Decision Tree Regressor',
              'Decision Tree Regressor CV',
              'Random Forest Regressor', 'Random Forest Regressor CV',
              'K-Nearest Neighbours Regressor',
              'K-Nearest Neighbours Regressor CV', 'Gradient Boosting Regressor',
              'Gradient Boosting Regressor CV'],
    'R2 Train Score': [lr_train_Score, lr_cv_train_Score, dtr_train_Score,
                       dtr_cv_train_Score, rfr_train_Score,
                       rfr_cv_train_Score, knnr_train_Score, knnr_cv_train_Score,
                         gbr_train_Score, gbr_cv_train_Score],
    'R2 Test Score': [lr_Test_Score, lr_cv_Test_Score, dtr_Test_Score, dtr_cv_Test_Score,
                      rfr_Test_Score,
                      rfr_cv_Test_Score, knnr_Test_Score, knnr_cv_Test_score,
                      gbr_Test_Score, gbr_cv_Test_Score],
    'Mean Absolute Error': [lr_MAE, lr_cv_MAE, dtr_MAE, dtr_cv_MAE, rfr_MAE,
                            rfr_cv_MAE, knnr_MAE, knn_cv_MAE,
                            gbr_MAE, gbr_cv_MAE],
    'Root Mean Squared Error': [lr_RMSE, lr_cv_RMSE, dtr_RMSE, dtr_cv_RMSE, rfr_RMSE,
                                rfr_cv_RMSE, knnr_RMSE,
                                knn_cv_RMSE, gbr_RMSE, gbr_cv_RMSE],
    'Mean Squared Error': [lr_MSE, lr_cv_MSE, dtr_MSE, dtr_cv_MSE, rfr_MSE, rfr_cv_MSE,
                           knnr_MSE, knn_cv_MSE,
                           gbr_MSE, gbr_cv_MSE],
    'Mean Absolute Percentage Error': [lr_MAPE, lr_cv_MAPE, dtr_MAPE, dtr_cv_MAPE, rfr_MAPE,
                                       rfr_cv_MAPE, knnr_MAPE,
                                       knn_cv_MAPE, gbr_MAPE, gbr_cv_MAPE]
})

# Displaying the scores
im_im_cv_scores



# Displaying True v Predicted plot of all the models in a single plot
plt.figure(figsize=(20, 10))
plt.scatter(y_test, ridge_cv.predict(X_test), label='Linear Regression', color='blue')
plt.scatter(y_test, dtr_cv.predict(X_test), label='Decision Tree Regressor', color='red')
plt.scatter(y_test, rfr_cv.predict(X_test), label='Random Forest Regressor', color='green')


plt.scatter(y_test, knn_cv.predict(X_test), label='K-Nearest Neighbours Regressor', color='purple')
plt.scatter(y_test, gbr_cv.predict(X_test), label='Gradient Boosting Regressor', color='orange')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color = 'red')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted for all models')
plt.legend()
plt.show()


# Initialising a Average Regressor function
class AverageRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, regressors):
        self.regressors = regressors

    def fit(self, X, y):
        for regressor in self.regressors:
            regressor.fit(X, y)
        return self

    def predict(self, X):
        predictions = np.column_stack([regressor.predict(X)
                                       for regressor in self.regressors])
        return np.mean(predictions, axis=1)
```

```
# Initialising a Average Regressor function
class AverageRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, regressors):
        self.regressors = regressors

    def fit(self, X, y):
        for regressor in self.regressors:
            regressor.fit(X, y)
        return self

    def predict(self, X):
        predictions = np.column_stack([regressor.predict(X)
                                       for regressor in self.regressors])
        return np.mean(predictions, axis=1)

# Creating the Average Regressor
average_regressor = AverageRegressor([
    Ridge(fit_intercept=lr_cv_best_fit_intercept, alpha=lr_cv_best_alpha,
          solver=lr_cv_best_solver),
    DecisionTreeRegressor(max_depth=dtr_cv_best_max_depth,
                          min_samples_split=dtr_cv_best_min_samples_split,
                          min_samples_leaf=dtr_cv_best_min_samples_leaf),
    RandomForestRegressor(n_estimators=rfr_cv_best_n_estimators,
                          max_depth=rfr_cv_best_max_depth),
    KNeighborsRegressor(n_neighbors=knnr_cv_best_n_neighbors,
                        weights=knnr_cv_best_weights, metric=knnr_cv_best_metric),
    GradientBoostingRegressor(n_estimators=gbr_cv_best_n_estimators,
                              max_depth=gbr_cv_best_max_depth,
                              learning_rate=gbr_cv_best_learning_rate)
])


# Fitting the Average Regressor
average_regressor.fit(X_train, y_train)

# Calculating Accuracy Scores
average_train_score = average_regressor.score(X_train, y_train)
average_test_score = average_regressor.score(X_test, y_test)



# Displaying the scores
print('Average Regressor - Accuracy Score for train data is: {}'.format(average_train_score))
print('Average Regressor - Accuracy Score for test data is: {}'.format(average_test_score))

# Calculating mean squared error
y_pred = average_regressor.predict(X_test)

average_MAE = mean_absolute_error(y_test, y_pred)
average_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
average_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
average_MSE = mean_squared_error(y_test, y_pred)

# Displaying the evaluation metrics
print('Average Regressor - Mean Absolute Error:', average_MAE)
print('Average Regressor - Root Mean Squared Error:', average_RMSE)
print('Average Regressor - Mean Squared Error:', average_MSE)
print('Average Regressor - Mean Absolute Percentage Error:', average_MAPE)

# Plotting predictions
plt.figure(figsize=(20, 6))
xt = X_train[:20]
yt = y_train[:20]
pred_lr = average_regressor.regressors[0].predict(xt)
pred_dtr = average_regressor.regressors[1].predict(xt)
pred_rfr = average_regressor.regressors[2].predict(xt)
pred_knnr = average_regressor.regressors[3].predict(xt)
pred_gbr = average_regressor.regressors[4].predict(xt)
pred_average = average_regressor.predict(xt)

fig, ax = plt.subplots(figsize=(10,3))
ax.plot(pred_lr, "ys--", alpha=0.5, label="Linear Regression")
ax.plot(pred_dtr, "m*--", alpha=0.5, label="Decision Tree")
ax.plot(pred_rfr, "b^-.", alpha=0.5, label="Random Forest")
ax.plot(pred_knnr, "c^--", alpha=0.5, label="K-Nearest Neighbours")
ax.plot(pred_gbr, "gd:", alpha=0.5, label="Gradient Boosting")
ax.plot(pred_average, "r*-", alpha=0.5, ms=10, label="Average Regressor")
ax.plot(yt.values, "ko", alpha=0.5, ms=3, label="True Data")
ax.tick_params(axis="x", which="both", bottom=False, top=False, labelbottom=False)
ax.set_ylabel("predicted")
ax.set_xlabel("training samples")
ax.legend(loc="best")
ax.set_title("Average Regressor predictions and their average");
plt.show()
```

```
# Initialising Voting Ensemble Regression implementation by R² test scores
# dictioanry
r2_scores = {
    'ridge': lr_cv_Test_Score,
    'dtr': dtr_cv_Test_Score,
    'rf': rfr_cv_Test_Score,
    'knn': knnr_cv_Test_score,
    'gb': gbr_cv_Test_Score
}

# Normalising the scores to sum up to 1
total_score = sum(r2_scores.values())
weights = {model: score / total_score for model, score in r2_scores.items()}

# Converting weights to a list
weights_list = list(weights.values())


# Initialising Voting Ensemble Regression implementation by R² test scores
# dictioanry
r2_scores = {
    'ridge': lr_cv_Test_Score,
    'dtr': dtr_cv_Test_Score,
    'rf': rfr_cv_Test_Score,
    'knn': knnr_cv_Test_score,
    'gb': gbr_cv_Test_Score
}


# Normalising the scores to sum up to 1
total_score = sum(r2_scores.values())
weights = {model: score / total_score for model, score in r2_scores.items()}

# Converting weights to a list
weights_list = list(weights.values())

# Displaying the weights
print("Weights for Voting Regressor:", weights_list)

# Creating and fitting the Voting Regressor with performance-based weights
ve_cv_pipeline = Pipeline(
    steps=[
        ("voting", VotingRegressor(
            estimators=[
                ("ridge", Ridge(fit_intercept=lr_cv_best_fit_intercept,
                                alpha=lr_cv_best_alpha, solver=lr_cv_best_solver)),
                ('dtr', DecisionTreeRegressor(max_depth=dtr_cv_best_max_depth,
                                              min_samples_split=dtr_cv_best_min_samples_split,
                                              min_samples_leaf=dtr_cv_best_min_samples_leaf)),
                ('rf', RandomForestRegressor(n_estimators=rfr_cv_best_n_estimators,
                                             max_depth=rfr_cv_best_max_depth)),
                ('knn', KNeighborsRegressor(n_neighbors=knnr_cv_best_n_neighbors,
                                            weights=knnr_cv_best_weights,
                                            metric=knnr_cv_best_metric)),
                ('gb', GradientBoostingRegressor(n_estimators=gbr_cv_best_n_estimators,
                                                 max_depth=gbr_cv_best_max_depth,
                                                 learning_rate=gbr_cv_best_learning_rate))
            ],
            weights=weights_list  # Assigning performance-based weights to the base regressors
        ))
    ]
)

# Fitting the Voting Regressor
ve_cv_pipeline.fit(X_train, y_train)

# Calculating Accuracy Scores
ve_cv_train_score = ve_cv_pipeline.score(X_train, y_train)
ve_cv_test_score = ve_cv_pipeline.score(X_test, y_test)

# Displaying the scores
print('Voting Regressor - Accuracy Score for train data is: {}'.format(ve_cv_train_score))
print('Voting Regressor - Accuracy Score for test data is: {}'.format(ve_cv_test_score))

# Calculating the prediction
y_pred = ve_cv_pipeline.predict(X_test)
```

```
# Calculating the error metrics
ve_cv_MAE = mean_absolute_error(y_test, y_pred)
ve_cv_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
ve_cv_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
ve_cv_MSE = mean_squared_error(y_test, y_pred)

# Displaying the evaluation metrics
print('Voting Regressor - Mean Absolute Error:', ve_cv_MAE)
print('Voting Regressor - Root Mean Squared Error:', ve_cv_RMSE)
print('Voting Regressor - Mean Squared Error:', ve_cv_MSE)
print('Voting Regressor - Mean Absolute Percentage Error:', ve_cv_MAPE)

# Plotting predictions
plt.figure(figsize=(20, 6))
xt = X_train[:20]
yt = y_train[:20]
estimators = ve_cv_pipeline.named_steps['voting'].estimators_
pred_lr = estimators[0].predict(xt)
pred_dtr = estimators[1].predict(xt)
pred_rfr = estimators[2].predict(xt)
pred_knnr = estimators[3].predict(xt)
pred_gbr = estimators[4].predict(xt)
pred_average = ve_cv_pipeline.predict(xt)

fig, ax = plt.subplots(figsize=(10,3))
ax.plot(pred_lr, "ys--", alpha=0.5, label="Linear Regression")
ax.plot(pred_dtr, "m*--", alpha=0.5, label="Decision Tree")
ax.plot(pred_rfr, "b^-.", alpha=0.5, label="Random Forest")
ax.plot(pred_knnr, "c^--", alpha=0.5, label="K-Nearest Neighbours")
ax.plot(pred_gbr, "gd:", alpha=0.5, label="Gradient Boosting")
ax.plot(pred_average, "r*-", alpha=0.5, ms=10, label="Voting Regressor")
ax.plot(yt.values, "ko", alpha=0.5, ms=3, label="True Data")
ax.tick_params(axis="x", which="both", bottom=False, top=False, labelbottom=False)
ax.set_ylabel("predicted")
ax.set_xlabel("training samples")
ax.legend(loc="best")
ax.set_title("Voting Regressor predictions and their average");


# Create a pipeline with Stacking Regressor
se_cv_pipeline = Pipeline(
    steps=[
        ("stacking", StackingRegressor(
            estimators=[
                ("ridge", Ridge(fit_intercept=lr_cv_best_fit_intercept,
                                alpha=lr_cv_best_alpha, solver=lr_cv_best_solver)),
                ('dtr', DecisionTreeRegressor(max_depth=dtr_cv_best_max_depth,
                                              min_samples_split=dtr_cv_best_min_samples_split,
                                              min_samples_leaf=dtr_cv_best_min_samples_leaf)),
                ('rf', RandomForestRegressor(n_estimators=rfr_cv_best_n_estimators,
                                             max_depth=rfr_cv_best_max_depth)),
                ('knn', KNeighborsRegressor(n_neighbors=knnr_cv_best_n_neighbors,
                                            weights=knnr_cv_best_weights,
                                            metric=knnr_cv_best_metric)),
                ('gb', GradientBoostingRegressor(n_estimators=gbr_cv_best_n_estimators,
                                                 max_depth=gbr_cv_best_max_depth,
                                                 learning_rate=gbr_cv_best_learning_rate))
            ],
            final_estimator=LinearRegression()
        ))
    ]
)

# Fitting the Stacking Regressor
se_cv_pipeline.fit(X_train, y_train)

# Calculating the accuracy scores
se_cv_train_score = se_cv_pipeline.score(X_train, y_train)
se_cv_test_score = se_cv_pipeline.score(X_test, y_test)
```

```
# Displaying the scores
print('Stacking Regressor - Accuracy Score for train data is: {}'.format(se_cv_train_score))
print('Stacking Regressor - Accuracy Score for test data is: {}'.format(se_cv_test_score))

# Calculating the prediction
y_pred = se_cv_pipeline.predict(X_test)

# Calculating the error metrics
se_cv_MAE = mean_absolute_error(y_test, y_pred)
se_cv_RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
se_cv_MAPE = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
se_cv_MSE = mean_squared_error(y_test, y_pred)

# Displaying the evaluation metrics
print('Stacking Regressor - Mean Absolute Error:', se_cv_MAE)
print('Stacking Regressor - Root Mean Squared Error:', se_cv_RMSE)
print('Stacking Regressor - Mean Squared Error:', se_cv_MSE)
print('Stacking Regressor - Mean Absolute Percentage Error:', se_cv_MAPE)

# Plotting predictions
plt.figure(figsize=(20, 6))
xt = X_train[:20]
yt = y_train[:20]

# Predictions calculation
estimators = se_cv_pipeline.named_steps['stacking'].named_estimators_
pred_lr = estimators['ridge'].predict(xt)
pred_dtr = estimators['dtr'].predict(xt)
pred_rfr = estimators['rf'].predict(xt)
pred_knnr = estimators['knn'].predict(xt)
pred_gbr = estimators['gb'].predict(xt)
pred_stacking = se_cv_pipeline.predict(xt)
fig, ax = plt.subplots(figsize=(10,3))
ax.plot(pred_lr, "ys--", alpha=0.5, label="Linear Regression")
ax.plot(pred_dtr, "m*--", alpha=0.5, label="Decision Tree")
ax.plot(pred_rfr, "b^-.", alpha=0.5, label="Random Forest")
ax.plot(pred_knnr, "c^--", alpha=0.5, label="K-Nearest Neighbours")
ax.plot(pred_gbr, "gd:", alpha=0.5, label="Gradient Boosting")
ax.plot(pred_stacking, "r*-", alpha=0.5, ms=10, label="Stacking Regressor")
ax.plot(yt.values, "ko", alpha=0.5, ms=3, label="True Data")
ax.tick_params(axis="x", which="both", bottom=False, top=False, labelbottom=False)
ax.set_ylabel("predicted")
ax.set_xlabel("training samples")
ax.legend(loc="best")
ax.set_title("Stacking Regressor predictions and their average");
plt.show()


# Displaying all the relevant scores of ensemble models in a dataframe
ensemble_scores = pd.DataFrame({
    'Model': ['Average Regressor', 'Voting Regressor', 'Stacking Regressor'],
    'R2 Train Score': [average_train_score, ve_cv_train_score, se_cv_train_score],
    'R2 Test Score': [average_test_score, ve_cv_test_score, se_cv_test_score],
    'Mean Absolute Error': [average_MAE, ve_cv_MAE, se_cv_MAE],
    'Root Mean Squared Error': [average_RMSE, ve_cv_RMSE, se_cv_RMSE],
    'Mean Squared Error': [average_MSE, ve_cv_MSE, se_cv_MSE],
    'Mean Absolute Percentage Error': [average_MAPE, ve_cv_MAPE, se_cv_MAPE],
})

# Display ensemble scores
ensemble_scores


# Displaying all the ensemble models in a single plot
plt.figure(figsize=(20, 6))

# Average Regressor
plt.subplot(1, 3, 1)
plt.scatter(x=y_test, y=average_regressor.predict(X_test), edgecolor='white')
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Average Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')
```

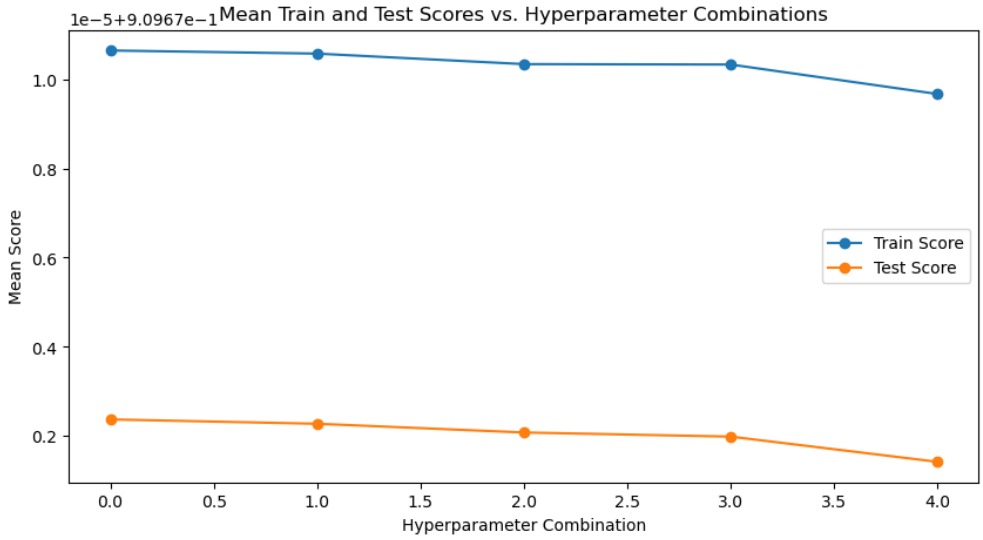
```
# Voting Regressor
plt.subplot(1, 3, 2)
plt.scatter(x=y_test, y=ve_cv_pipeline.predict(X_test), edgecolor='white')
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Voting Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')

# Stacking Regressor
plt.subplot(1, 3, 3)
plt.scatter(x=y_test, y=se_cv_pipeline.predict(X_test), edgecolor='white')
plt.plot([0, 100000], [0, 100000], color='red')
plt.title('Stacking Regressor')
plt.xlabel('True')
plt.ylabel('Predicted')

# Adjust the layout
plt.tight_layout()
plt.show()


# Disaplying True vs Predicted Analysis of the ensemble models in a pairplot in a single plot
plt.figure(figsize=(12, 6))

# Average Regressor
sns.scatterplot(x=y_test, y=average_regressor.predict(X_test), label='Average Ensemble Regressor')

# Voting Regressor
sns.scatterplot(x=y_test, y=ve_cv_pipeline.predict(X_test), label='Voting Ensemble Regressor')

# Stacking Regressor
sns.scatterplot(x=y_test, y=se_cv_pipeline.predict(X_test), label='Stacking Ensemble Regressor')

# Plot the line of perfect fit
plt.plot([0, 100000], [0, 100000], color='red')

# Labels and legend
plt.xlabel('True')
plt.ylabel('Predicted')
plt.title('True vs Predicted Analysis of Ensemble Regressors')
plt.legend()
plt.show()


# Extracting original feature names
original_feature_names = df.columns.to_list()


# Setting figure size for SHAP summary plot
plt.figure(figsize=(20, 10))

# Summarising the background data using shap.utils.sample
background = shap.utils.sample(X_train, 1)  # Use 1 sample for the background due to time constraint

# Creating a SHAP explainer for Stacking Ensemble Regressor using KernelExplainer
explainer = shap.KernelExplainer(se_cv_pipeline.predict, background)

# Calculating SHAP values
shap_values = explainer.shap_values(X_test)

# Displaying summary SHAP plot
shap.summary_plot(shap_values, X_test, plot_size=None, show=False, sort=True, feature_names=original_feature_names)

# Setting plot title
plt.title('Summary SHAP plot for Stacking Ensemble Regressor')

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

```
# If X_test is a DataFrame
if isinstance(X_test, pd.DataFrame):
    feature_names = X_test.columns
else:
    # Use original feature names if X_test is a NumPy array
    feature_names = original_feature_names[:-1]

# Calculating the sum of absolute SHAP values for each feature
feature_importance = np.abs(shap_values).mean(0)

# Creating a DataFrame for easier handling
feature_importance_df = pd.DataFrame(feature_importance, index=feature_names, columns=['importance'])

# Sorting the DataFrame by importance
feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=True)

# Plotting Global Feature Importance Bar Plot
plt.figure(figsize=(22, 6))
plt.barh(feature_importance_df.index, feature_importance_df['importance'])
plt.title('Global Feature Importance Bar Plot for Stacking Ensemble Regressor')
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.show()


# Using the column names from the original df
X_test_df = pd.DataFrame(X_test, columns=df.columns[:-1])

# Setting a random seed for reproducibility
np.random.seed(42)

# Fitting the Stacking Ensemble Regressor model
se_cv_pipeline.fit(X_train, y_train)

# Reducing the number of background samples
background = shap.sample(X_train, 15, random_state=42)  # Use 15 samples for the background

# Creating a SHAP explainer
explainer = shap.KernelExplainer(se_cv_pipeline.predict, background)



# Calculating SHAP values for the first instance
shap_values = explainer.shap_values(X_test_df.iloc[0:1, :])

# Creating an Explanation object
explanation = shap.Explanation(values=shap_values[0],
                               base_values=explainer.expected_value,
                               data=X_test_df.iloc[0, :],
                               feature_names=X_test_df.columns)

# Displaying Waterfall plot
shap.waterfall_plot(explanation)

# Disaplying Force plot
shap.force_plot(explainer.expected_value, shap_values[0], X_test_df.iloc[0, :])
```

Appendix C – Hyperparameter Tuning Results

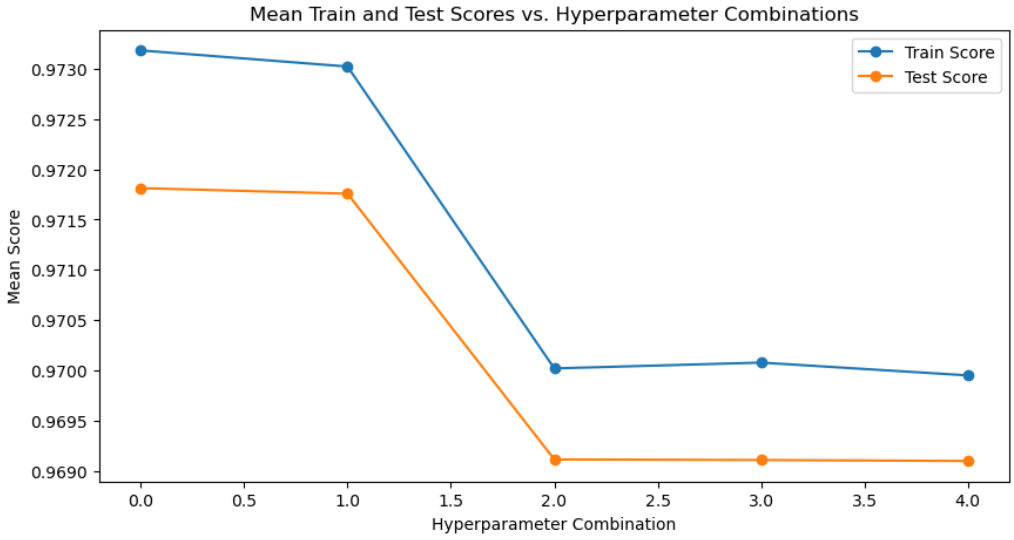
Linear Regression

Hyperparameters	Mean Train Score	Std Train Score	Mean Test Score	Std Test Score	Rank Score
{'solver': 'auto', 'fit_intercept': True, 'alp...	0.909681	0.000126	0.909672	0.000505	1
'solver': 'saga', 'fit_intercept': True, 'alp...	0.909681	0.000126	0.909672	0.000505	2
{'solver': 'svd', 'fit_intercept': True, 'alph...	0.909680	0.000126	0.909672	0.000505	3
{'solver': 'sag', 'fit_intercept': True, 'alph...	0.909680	0.000126	0.909672	0.000505	4
{'solver': 'sparse_cg', 'fit_intercept': True,...	0.909680	0.000126	0.909671	0.000505	5



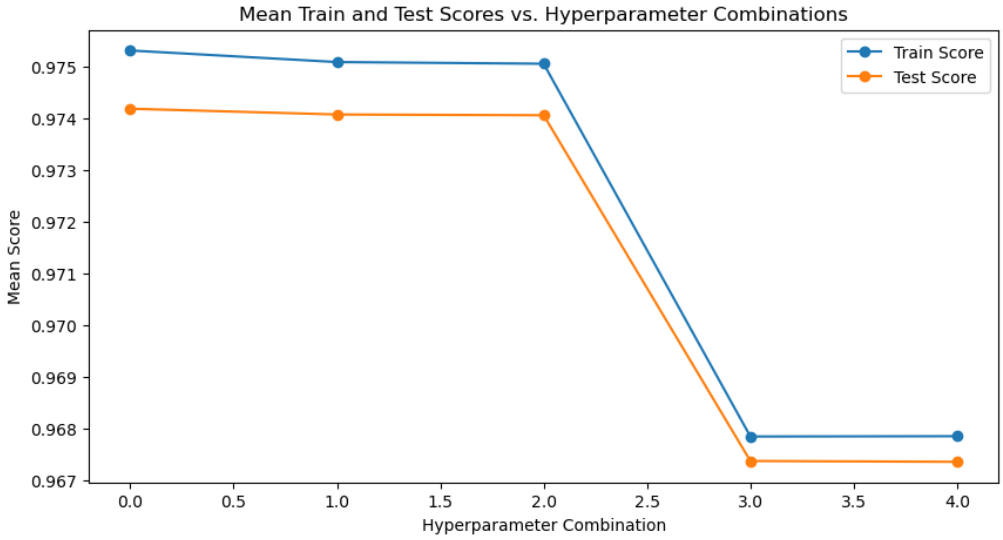
Decision Tree Regression

Hyperparameters	Mean Train Score	Std Train Score	Mean Test Score	Std Test Score	Rank Score
{'min_samples_split': 10, 'min_samples_leaf': ...	0.973179	0.000723	0.971812	0.000495	1
{'min_samples_split': 20, 'min_samples_leaf': ...	0.973019	0.000741	0.971757	0.000430	2
{'min_samples_split': 10, 'min_samples_leaf': ...	0.970022	0.000570	0.969119	0.000339	3
{'min_samples_split': 2, 'min_samples_leaf': 1...	0.970080	0.000580	0.969113	0.000374	4
{'min_samples_split': 20, 'min_samples_leaf': ...	0.969953	0.000583	0.969104	0.000322	5



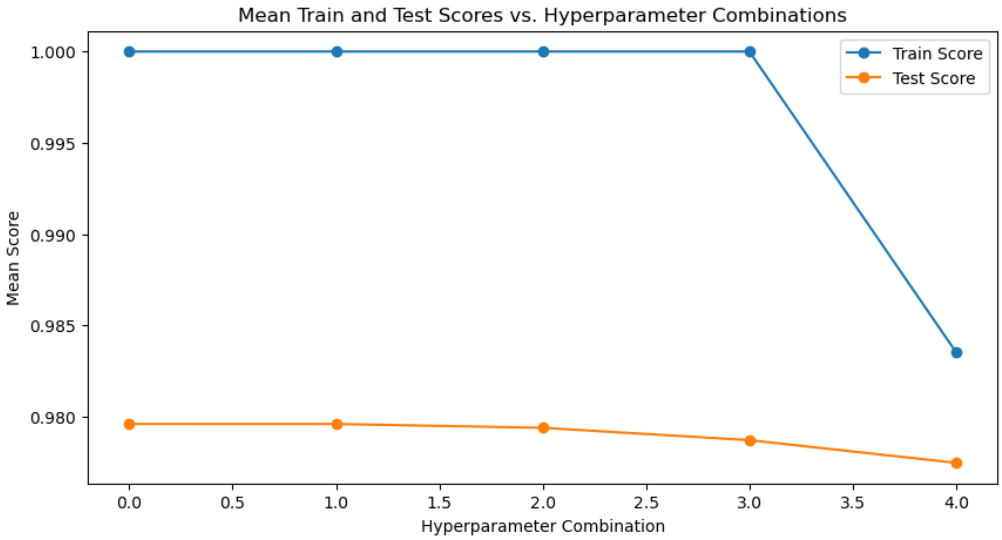
Random Forest Regression

Hyperparameters	Mean Train Score	Std Train Score	Mean Test Score	Std Test Score	Rank Test Score
{'n_estimators': 500, 'min_samples_split': 15,...	0.975302	0.000375	0.974178	0.000154	1
{'n_estimators': 100, 'min_samples_split': 20,...	0.975079	0.000376	0.974067	0.000170	2
{'n_estimators': 400, 'min_samples_split': 20,...	0.975046	0.000419	0.974053	0.000113	3
{'n_estimators': 200, 'min_samples_split': 20,...	0.967853	0.000295	0.967381	0.000153	4
{'n_estimators': 100, 'min_samples_split': 10,...	0.967859	0.000259	0.967364	0.000193	5



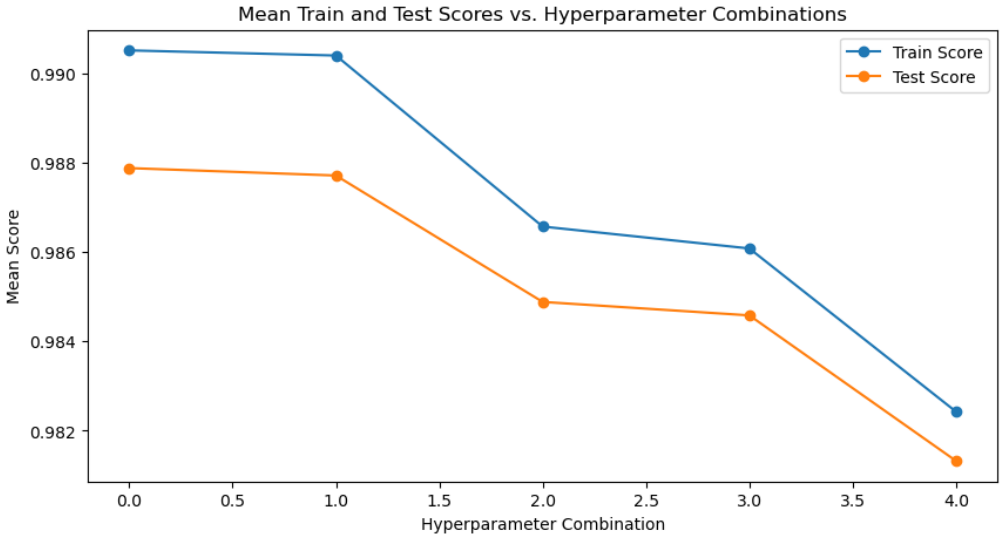
K-Nearest Neighbours Regression

Hyperparameters	Mean Train Score	Std Train Score	Mean Test Score	Std Test Score	Rank Test Score
{'weights': 'distance', 'n_neighbors': 8, 'met...	0.999997	5.950903e-07	0.979595	0.000094	1
{'weights': 'distance', 'n_neighbors': 5, 'met...	0.999997	5.950903e-07	0.979593	0.000153	2
{'weights': 'distance', 'n_neighbors': 4, 'met...	0.999997	5.950903e-07	0.979384	0.000126	3
{'weights': 'distance', 'n_neighbors': 14, 'me...	0.999997	5.950903e-07	0.978708	0.000138	4
{'weights': 'uniform', 'n_neighbors': 7, 'metr...	0.983524	4.854665e-05	0.977466	0.000165	5



Gradient Boosting Regression

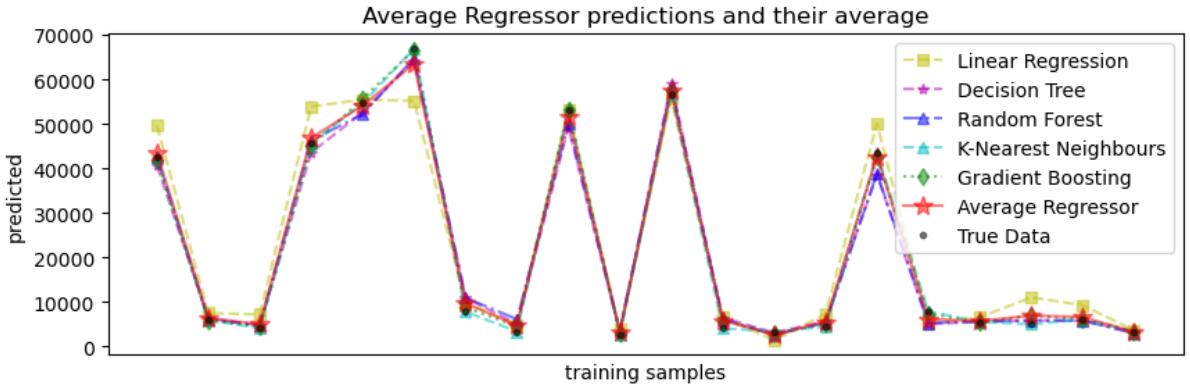
Hyperparameters	Mean Train Score	Std Train Score	Mean Test Score	Std Test Score	Rank Test Score
{'n_estimators': 400, 'max_depth': 7, 'learnin...	0.990520	0.000098	0.987884	0.000262	1
{'n_estimators': 400, 'max_depth': 8, 'learnin...	0.990406	0.000125	0.987718	0.000196	2
{'n_estimators': 400, 'max_depth': 6, 'learnin...	0.986574	0.000078	0.984886	0.000203	3
{'n_estimators': 400, 'max_depth': 5, 'learnin...	0.986083	0.000104	0.984584	0.000197	4
{'n_estimators': 400, 'max_depth': 6, 'learnin...	0.982426	0.000099	0.981323	0.000165	5


Mean Train and Test Scores vs. Hyperparameter Combinations
Train Score
Test Score
Mean Score
0.990
0.988
0.986
0.984
0.982
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Hyperparameter Combination

Appendix D - Ensemble Model Results

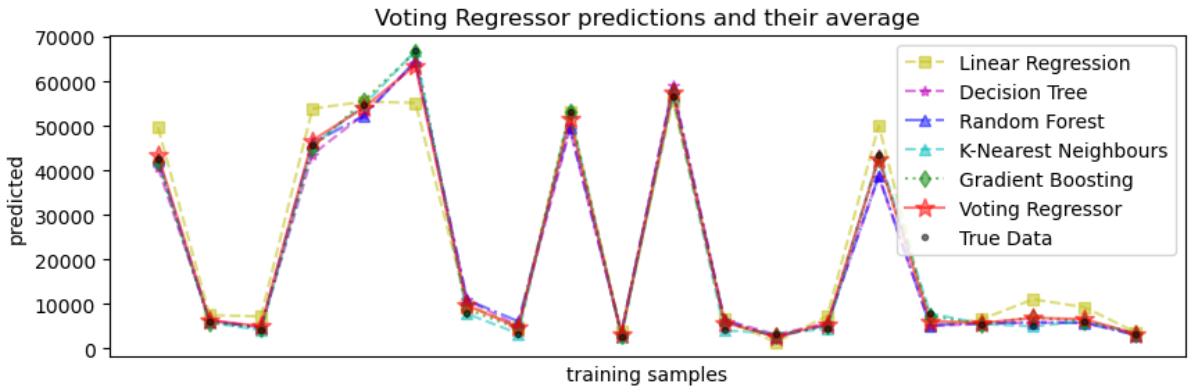
Average Ensemble Regression

Training Data Accuracy Score	Test Data Accuracy Score	Mean Absolute Error	Root Mean Squared Error	Mean Squared Error	Mean Absolute Percentage Error
0.9847734370237887	0.9779833560076324	1982.434550618309	3372.8468630354128	11376095.961487824	16.43908685591465



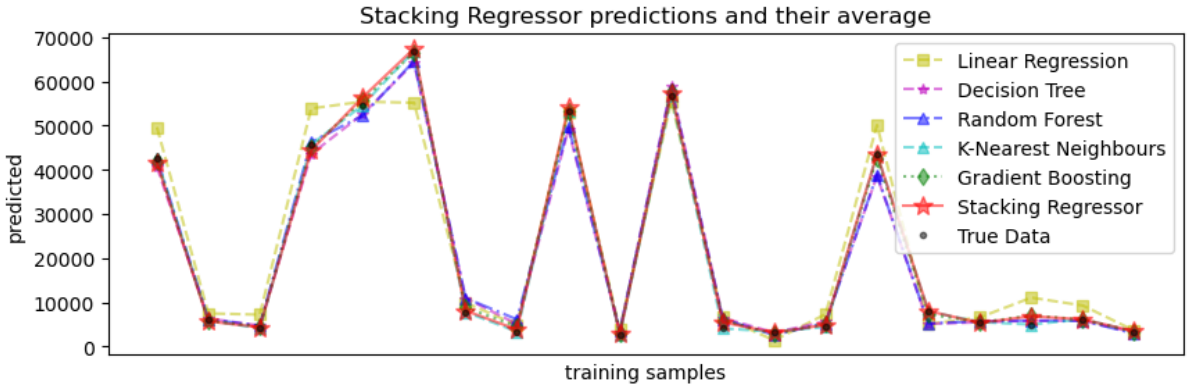
Voting Ensemble Regression

Training Data Accuracy Score	Test Data Accuracy Score	Mean Absolute Error	Root Mean Squared Error	Mean Squared Error	Mean Absolute Percentage Error
0.9852596056426026	0.9784077767067226	1953.471924201089	3340.179027454291	11156795.935445493	16.103120297539135



Stacking Ensemble Regression

Training Data Accuracy Score	Test Data Accuracy Score	Mean Absolute Error	Root Mean Squared Error	Mean Squared Error	Mean Absolute Percentage Error
0.9939580071489891	0.9886848253934475	1317.8348102402683	2417.9743717935867	5846600.06265059	10.850822178806025


Stacking Regressor predictions and their average
70000
60000
50000
40000
30000
20000
10000
0
predicted
training samples
Linear Regression
Decision Tree
Random Forest
K-Nearest Neighbours
Gradient Boosting
Stacking Regressor
True Data