GROUP ASSIGNMENT

AICT009-4-2-IDA

INTRODUCTION TO DATA ANALYTICS

UCDF2407ICT(DI)

No.	STUDENT NAME	TP NUMBER
1.	HENG EE SERN	TP081786
2.	TAN HAO SHUAN	TP080852
3.	LAEU ZI-LI	TP083604

LECTURER NAME: TS. DR. LAW FOONG LI

HAND OUT DATE: 22nd MAY 2025

SUBMISSION DATE: 31st JULY 2025

Table of Contents

1 Introduction 4
1.1 Project Overview 4
1.2 Business Domain and Background Information 5
1.2.1 Strategic Management: KPI Achievement Analytics 5
1.2.2 The Commercial Passenger Airline Industry 5
1.3 Problem Statement 6
1.4 Aim and Objectives 6
1.5 Hypotheses 7
1.6 Scope of the Solution 8
2 Data Analytics Methodology 9
2.1 CRISP-DM (Cross-Industry Standard Process for Data Mining) 9
2.2 Explanation and Justification of Chosen Methodology 9
3 Data Understanding and Preparation 11
3.1 Data Preparation and Cleaning 11
3.1.1 Data Sourcing and Initial Inspection 11
3.1.2 Target Variable Distribution Analysis 14
3.1.3 Handing Missing Values and Special Conditions 15
3.1.4 Data Transformation 17
3.2 Exploratory Data Analysis (EDA) 19
3.2.1 Univariate Analysis 19
3.2.2 Bivariate Analysis 23
3.2.3 Correlation and Statistical Analysis 25
3.3 Feature Engineering and Selection 29
3.3.1 Feature Scaling 29
3.3.2 Generating Feature Sets with Filter, Wrapper, and Embedded Methods 31
3.3.3 Comparative Evaluation and Final Selection 38

4 Predictive Modelling 45
4.1 Logistic Regression (Laeu Zi-Li) 47
4.1.1 Introduction to Logistic Regression 47
4.1.2 Training of untuned model 47
4.1.3 Model performance 47
4.1.4 Fine-tuning model 49
4.1.5 Results and summary 51
4.2 Random Forest (Tan Hao Shuan) 53
4.2.1 Introduction to Random Forest 53
4.2.2 Training of Untuned Model 53
4.2.3 Model Performance 53
4.2.4 Fine-Tuning Model 55
4.2.5 Final Performance of Tuned Random Forest Model 56
4.3 Extreme Gradient Boosting (XGBoost) (Heng Ee Sern) 59
4.3.1 Introduction to XGBoost 59
4.3.2 Baseline Performance Evaluation 59
4.3.3 Hyperparameter Fine-Tuning with GridSearchCV 62
4.3.4 Overfitting Analysis 64
4.3.5 Final Performance of the Tuned XGBoost Model 65
5 Model Comparison and Final Selection 67
5.1 Performance Comparison of Models 67
5.2 Selecting the Champion Model 67
5.3 Model Interpretation with SHAP 68
5.3.1 Global Feature Importance 68
5.3.2 Explaining Individual Predictions 69
6 Model Deployment with Streamlit 71
6.1 Tab 1: Interactive Descriptive Analysis 71

6.2 Tab 2: Single Passenger Prediction and Feedback System 72
6.3 Tab 3: Batch Prediction via File Upload 75
7 Business Recommendations 76
7.1 Strategic Insights Derived from the Final Model's 10 Features 76
7.2 Actionable Recommendations for the Airline 77
8 Conclusion 78
8.1 Summary of Project Findings and Achievements 78
8.2 Project Limitations 78
8.3 Recommendations for Future Work 79
8.4 Social and Ethical Considerations 79
8.4.1 Social Impacts 79
8.4.2 Ethical Issues 79
9 References 81
10 Appendix 83
10.1 Appendix A: Workload Matrix 83
10.2 Appendix B: Personal Reflection Report 84

1 Introduction

1.1 Project Overview

This project presents an end-to-end data analytics solution designed to address the critical business challenge of passenger dissatisfaction in the airline industry. Using a publicly available dataset of over 120,000 passenger surveys, this report details a comprehensive workflow following the CRISP-DM methodology. The process began with data preparation and exploratory data analysis to uncover initial insights. A multi-stage feature selection process was then conducted, systematically evaluating a wide range of methods to reduce the original 23 features down to an optimal set of 10 key features.

A comparative analysis of multiple machine learning models was performed, resulting in the selection and fine-tuning of a high-performance XGBoost classifier, which achieved 94.3% accuracy on unseen data. Explainable AI (XAI) techniques using SHAP were implemented to interpret its predictions.

The final deliverable is a fully functional, interactive Streamlit dashboard. This application not only provides real-time satisfaction predictions for individual or batch passenger data but also includes a feedback system, creating a tool that offers both immediate predictive power and long-term strategic value for improving the customer experience.

1.2 Business Domain and Background Information

1.2.1 Strategic Management: KPI Achievement Analytics

The selected business domain is strategic management, mainly focused on achievement analytics. Strategic management is a clarification of a company's mission and vision to help reach its goals (Sheldon et al., 2024). A common term used in strategic management is key performance indicator (KPI), which is a numeric indicator of progress over a period towards achieving a specific goal (Kiviak, 2023).

1.2.2 The Commercial Passenger Airline Industry

Customer satisfaction in the competitive airline market today is not only a service metric, but also a strategic key performance indicator (KPI) that plays significant weight in customer loyalty, operating performance, and long-term profitability .Those airlines that monitor and enhance customer satisfaction can differentiate themselves through the provision of improved passenger experiences in an era where customer feedback is in high visibility and high influence. (Tahanisaz & Shokuhyar, 2020)

The information in this project, "Airline Passenger Satisfaction", contains complete records on passenger demographics, travel details, service quality scores, and overall satisfaction levels. It enables analysis of drivers of satisfaction such as Wi-Fi on flights, beverages and meals, luggage, and comfort levels in seating. (Dike et al., 2023) Such data can be used by management to identify trends and issues in the service process, culminating ultimately in strategic decision-making and improvement actions.

Today's customer service is provided through the end-to-end passenger experience, from pre-flight engagement, in-out service, to post-flight service, all of them essential to providing great satisfaction and fostering loyalty. With customers becoming increasingly familiar with simple digital interactions, airlines are focusing on improving in-flight interaction and offering connected experience worthy of today's customer expectations. (Shiwakoti et al., 2022)

This strategic focus on satisfaction KPIs is harmonious with higher-level goals like customer retention, brand differentiation, and market competitiveness, which are all critical elements of good strategic management.

1.3 Problem Statement

The airline industry is a highly competitive, service-based sector that heavily relies on repeat passengers, customer loyalty, and a strong brand reputation to maintain profitability and market share (Batarlienė, 2023). However, challenges arise in the operational and service aspects can lead to passenger dissatisfaction. Without proactive measures to address these issues, customer retention will suffer due to damaged reputation, resulting in financial losses (Ali, 2022). This project aims to mitigate this problem by utilizing data analytics to identify passengers likely to be dissatisfied through a predictive model, thereby enabling airlines to proactively intervene and enhance the customer flight experience.

1.4 Aim and Objectives

Aim:

- To identify passengers likely to be dissatisfied and address potential issues by developing a predictive model, enabling airline improve customer flight experience.

Objectives:

1. To preprocess and explore the dataset to understand the variables and overall satisfaction distribution.
2. To analyse the relationships between various attributes to identify key features that will accurately predict satisfaction.
3. To build and evaluate a classification model that predicts whether a passenger will be satisfied or dissatisfied based on their journey attributes.
4. To determine the most influential features that contribute to dissatisfaction from the classification model.
5. To propose strategies and improvements to the identified area of weaknesses.

1.5 Hypotheses

1. If the rating of inflight amenities and comfort (such as Seat Comfort, Leg Room Service, Cleanliness, Food and Drink, In-flight service, In-flight Wifi Service, and In-flight Entertainment) are higher, then the overall passenger satisfaction will be higher.
2. Passenger demographics will have a negligible relationship with overall passenger satisfaction.
3. Passengers travelling in Economy class will demonstrate lower overall satisfaction compared to those in Business or Economy Plus classes.
4. Flight delays (departure and arrival) will show a weak negative correlation with the overall passenger satisfaction.
5. If the passenger is returning passenger, then their overall satisfaction will be higher.

1.6 Scope of the Solution

Inclusions:

- The provided *airline_passenger_satisfaction.csv* and *data_dictionary.csv* datasets will be exclusively used for all analysis and training of the data mining model.
- CRISP-DM methodology will be applied for structured data mining and reporting.
- A comprehensive report about the process of analysis, findings, and actionable recommendations will be delivered, along with an interactive dashboard built with Streamlit designed to visualize key insights and accepting input data for prediction.
- Python libraries such as Pandas, Scikit-learn, Matplotlib, and Seaborn will be utilized for the entire data analysis and modelling process.

Exclusions:

- Real-time operational data from any airline will not be incorporated into this analysis.
- The developed model does not support live deployment or integration into an airline's existing system, it is intended as a hypothetical solution for academic purposes only.
- Detailed feasibility studies are not included for the recommendations proposed.
- The analysis is solely focused on passenger experience and will not encompass a full audit on the airline operational departments.

2 Data Analytics Methodology

2.1 CRISP-DM (Cross-Industry Standard Process for Data Mining)

For this data analytics project, the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology has been selected.

2.2 Explanation and Justification of Chosen Methodology

Cross-Industry Standard Process for Data Mining (CRISP-DM) is considered as the most common methodology within the data science field, including data mining and analytics (Hotz, 2024). It provides a structured approach that consists of 6 phases to help with data mining projects.

The first phase is the Business Understanding phase, which focus on identifying the project goals and scope. This phase mainly defines the desired outputs of the project and aligns them with business needs (Smart Vision Europe, 2020). It involves setting clear objectives by describing the primary goal from a business perspective, developing a project plan outlining the specific steps for data mining and establishing business success criteria to determine whether the project has been successful. This phase serves as a foundation for project management activities.

The second phase is the Data Understanding phase. This phase involves four key steps to produce a data quality report. These steps include data collection, data description, data exploration data quality verification. First, the relevant data from the project resources will be loaded. Then, the data will be described according to its features such as format and quantity. Afterwards, some data mining questions will be addressed using querying, visualizing or other techniques. Lastly, the quality of the data will be examined to produce a data quality report, suggesting solutions for identified problems.

The third phase is Data Preparation phase, where the selected data will be cleaned by addressing the missing data, data errors and measurement errors. New data will also be constructed by deriving new attributes from existing attributes and generating records. Afterwards, these data will be integrated and combined to form new record and values. The methods used to integrate includes merging tables that have different information about the same objects and summarizing information from multiple records or tables.

The fourth phase is Modelling phase. Modelling is carried out in many processes including modelling techniques selection, test design generation, model building and model assessing (*IBM SPSS Modeler Subscription*, 2021). Initially, an appropriate modelling technique such as decision tree or neural network is chosen to be executed. Then, a testing mechanism is developed to test the evaluate the model's effectiveness. The model is then built according to its parameter settings. Finally, the model is interpreted based on domain knowledge to determine its accuracy or effectiveness, ensuring it meets the requirements to become the final model.

The fifth phase is Evaluation phase, during which the models are compared to identify the one that best meets the business requirements. The comparison process helps to determine the reason for any deficiencies in the models. Additionally, this phase involves planning for next steps in the project. Depending on the reviews and results, a decision is made whether to proceed to the next phase or repeat the earlier data mining process.

The final phase is the Deployment phase. In this phase, a deployment, monitoring and maintenance plan is designed. Once the plans are established, a final report for the project is produced. Afterwards, a comprehensive documentation is created as a review for what went well and identify area of improvements for the project.

3 Data Understanding and Preparation

3.1 Data Preparation and Cleaning

3.1.1 Data Sourcing and Initial Inspection

The dataset selected is "Airline Passenger Satisfaction" dataset from Maven Analytics. After obtaining and loading the dataset along with the necessary libraries, the initial data inspection was carried out. Commands such as *df.head()* examines the key columns such as ID, Gender, Age, Customer Type and Flight Distance. Other commands such as *df.info()* is used to reveal the basic information about the dataset, including number of columns, entries and data types. Moreover, *df.describe()* generates the numerical columns of the data frame like count, mean, min and max. Lastly, *data_dict* shows a description of every columns in the dataset while *df.shape()* returns a tuple representing the dimensions of the data frame.

```
[ ] df.head()
```

	ID	Gender	Age	Customer Type	Type of Travel	Class	Flight Distance	Departure Delay	Arrival Delay	Departure and Arrival Time Convenience	...	On-board Service	Seat Comfort	Leg Room Service	Cleanliness	Food and Drink	In-flight Service	In-flight Wifi Service	In-flight Entertainment	Baggage Handling
0	1	Male	48	First-time	Business	Business	821	2	5.0	3	...	3	5	2	5	5	5	3	5	5
1	2	Female	35	Returning	Business	Business	821	26	39.0	2	...	5	4	5	5	3	5	2	5	5
2	3	Male	41	Returning	Business	Business	853	0	0.0	4	...	3	5	3	5	5	3	4	3	3
3	4	Male	50	Returning	Business	Business	1905	0	0.0	2	...	5	5	5	4	4	5	2	5	5
4	5	Female	49	Returning	Business	Business	3470	0	1.0	3	...	3	4	4	5	4	3	3	3	3

5 rows × 24 columns

Table 3.1: First five rows of the DataFrame

```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129880 entries, 0 to 129879
Data columns (total 24 columns):
 #   Column                            Non-Null Count   Dtype
---  ------                            --------------   -----
 0   ID                                129880 non-null  int64
 1   Gender                            129880 non-null  object
 2   Age                               129880 non-null  int64
 3   Customer Type                     129880 non-null  object
 4   Type of Travel                    129880 non-null  object
 5   Class                             129880 non-null  object
 6   Flight Distance                   129880 non-null  int64
 7   Departure Delay                   129880 non-null  int64
 8   Arrival Delay                     129487 non-null  float64
 9   Departure and Arrival Time Convenience  129880 non-null  int64
 10  Ease of Online Booking            129880 non-null  int64
 11  Check-in Service                  129880 non-null  int64
 12  Online Boarding                   129880 non-null  int64
 13  Gate Location                     129880 non-null  int64
 14  On-board Service                  129880 non-null  int64
 15  Seat Comfort                      129880 non-null  int64
 16  Leg Room Service                  129880 non-null  int64
 17  Cleanliness                       129880 non-null  int64
 18  Food and Drink                    129880 non-null  int64
 19  In-flight Service                 129880 non-null  int64
 20  In-flight Wifi Service            129880 non-null  int64
 21  In-flight Entertainment           129880 non-null  int64
 22  Baggage Handling                  129880 non-null  int64
 23  Satisfaction                      129880 non-null  object
dtypes: float64(1), int64(18), object(5)
memory usage: 23.8+ MB
```

Table 3.2: Summary of the Raw Dataset Structure

```
df.describe()
```

	ID	Age	Flight Distance	Departure Delay	Arrival Delay	Departure and Arrival Time Convenience	Ease of Online Booking	Check-in Service	Online Boarding	Gate Location	On-board Service	Seat Comfort	
count	129880.000000	129880.000000	129880.000000	129880.000000	129487.000000	129880.000000	129880.000000	129880.000000	129880.000000	129880.000000	129880.000000	129880.000000	12
mean	64940.500000	39.427957	1190.316392	14.713713	15.091129	3.057599	2.756876	3.306267	3.252633	2.976925	3.383023	3.441361	
std	37493.270818	15.119360	997.452477	38.071126	38.465650	1.526741	1.401740	1.266185	1.350719	1.278520	1.287099	1.319289	
min	1.000000	7.000000	31.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	32470.750000	27.000000	414.000000	0.000000	0.000000	2.000000	2.000000	3.000000	2.000000	2.000000	2.000000	2.000000	
50%	64940.500000	40.000000	844.000000	0.000000	0.000000	3.000000	3.000000	3.000000	3.000000	3.000000	4.000000	4.000000	
75%	97410.250000	51.000000	1744.000000	12.000000	13.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	5.000000	
max	129880.000000	85.000000	4983.000000	1592.000000	1584.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	

Table 3.3: Descriptive statistics for the numerical features

```
data_dict
```

	Field	Description
0	ID	Unique passenger identifier
1	Gender	Gender of the passenger (Female/Male)
2	Age	Age of the passenger
3	Customer Type	Type of airline customer (First-time/Returning)
4	Type of Travel	Purpose of the flight (Business/Personal)
5	Class	Travel class in the airplane for the passenger...
6	Flight Distance	Flight distance in miles
7	Departure Delay	Flight departure delay in minutes
8	Arrival Delay	Flight arrival delay in minutes
9	Departure and Arrival Time Convenience	Satisfaction level with the convenience of the...
10	Ease of Online Booking	Satisfaction level with the online booking exp...
11	Check-in Service	Satisfaction level with the check-in service f...
12	Online Boarding	Satisfaction level with the online boarding ex...
13	Gate Location	Satisfaction level with the gate location in t...
14	On-board Service	Satisfaction level with the on-boarding servic...
15	Seat Comfort	Satisfaction level with the comfort of the air...
16	Leg Room Service	Satisfaction level with the leg room of the ai...
17	Cleanliness	Satisfaction level with the cleanliness of the...
18	Food and Drink	Satisfaction level with the food and drinks on...
19	In-flight Service	Satisfaction level with the in-flight service ...
20	In-flight Wifi Service	Satisfaction level with the in-flight Wifi ser...
21	In-flight Entertainment	Satisfaction level with the in-flight entertai...
22	Baggage Handling	Satisfaction level with the baggage handling f...

Table 3.4: Data dictionary defining each feature in the dataset

```
[ ] df.shape

(129880, 24)
```

Table 3.5:Dimensions of the raw dataset

3.1.2 Target Variable Distribution Analysis

The target variable chosen is *Satisfaction*. The target variable distribution analysis is carried out to understand the nature of the prediction problem. In short, this step aims to check if the dataset is balanced. As shown in the figure below, the number of satisfied passengers is slightly lower than the number of passengers who are neutral or dissatisfied. This shows a slight class imbalance in the dataset.

Figure 3.1: Distribution of the Target Variable

```
Percentage distribution of passenger satisfaction:
Satisfaction_Numeric
1    56.553742
0    43.446258
Name: proportion, dtype: float64
```

Table 3.6: Percentage breakdown of 'Satisfied' vs. 'Neutral or Dissatisfied' passengers

3.1.3 Handing Missing Values and Special Conditions

Based on Figure, it is noticeable that *Arrival Delay* has missing values. In this situation, the best approach is to fill the missing values of the column by using the median of the column. This is because there are potential outliers that might skew the mean and cause inaccuracy, making the median more stable and representative. The command *.fillna()* replaces all the missing values with the calculated median. Afterward, the last line checks for confirmation that all missing values have been handled.

```
[ ]  # Arrival Delay has missing values.
     # Filling with medians is the best approach with potential outliers
     median_arrival_delay = df['Arrival Delay'].median()
     df['Arrival Delay'] = df['Arrival Delay'].fillna(median_arrival_delay)
     df['Arrival Delay'].isnull().sum()

     np.int64(0)
```

Code Snippet 3.1: Python code for imputing missing Arrival Delay values

Some special conditions may occur due to the flaws of the survey. For instance, some data with the rating of '0' are caused by non-applicable or not-responded conditions. These cannot be treated as a score as it will often cause misleading and might negatively impact the model's performance. A two-step strategy is designed to deal with these special cases. Firstly, all occurrence of '0' within these specific columns are replaced with a missing value using the *replace(0, np.nan)* command. Then, they will be imputed with the median of their respective columns.

```
[ ]  # Treat 0 (Not Applicable) in rating features as missing values and impute with median
     # Features that were originally ratings
     rating_features = [
         'Departure and Arrival Time Convenience', 'Ease of Online Booking', 'Check-in Service',
         'Online Boarding', 'Gate Location', 'On-board Service', 'Seat Comfort', 'Leg Room Service',
         'Cleanliness', 'Food and Drink', 'In-flight Service', 'In-flight Wifi Service',
         'In-flight Entertainment', 'Baggage Handling'
     ]

     print("Missing values before imputation of 0:")
     print(df[rating_features].isna().sum())

     # Replace 0 with np.nan
     df[rating_features] = df[rating_features].replace(0, np.nan)
     # Fill nan with median
     df[rating_features] = df[rating_features].fillna(df[rating_features].median())

     # After fillna
     print("Missing values after imputation of 0:")
     print(df[rating_features].isna().sum())
```

Code Snippet 3.2: Python code for the process of treating '0' ratings as null values and imputing them with the column median

```
Missing values before imputation of 0:
Departure and Arrival Time Convenience    0
Ease of Online Booking                    0
Check-in Service                          0
Online Boarding                           0
Gate Location                             0
On-board Service                          0
Seat Comfort                              0
Leg Room Service                          0
Cleanliness                               0
Food and Drink                            0
In-flight Service                         0
In-flight Wifi Service                    0
In-flight Entertainment                   0
Baggage Handling                          0
dtype: int64
Missing values after imputation of 0:
Departure and Arrival Time Convenience    0
Ease of Online Booking                    0
Check-in Service                          0
Online Boarding                           0
Gate Location                             0
On-board Service                          0
Seat Comfort                              0
Leg Room Service                          0
Cleanliness                               0
Food and Drink                            0
In-flight Service                         0
In-flight Wifi Service                    0
In-flight Entertainment                   0
Baggage Handling                          0
dtype: int64
```

Table 3.7: Comparison of null value counts before and after the imputation process

3.1.4 Data Transformation

Data transformation is carried out to transform the target variables into a format that machine learning algorithms can understand and process. As machine learning algorithms require numerical inputs, the original text values need to be converted. The *.map()* function creates a new column and maps the labelled category *Satisfied* and *Neutral or Dissatisfied* to numerical values 0 and 1.

```
[ ] # Target Variable
    df['Satisfaction_Numeric'] = df['Satisfaction'].map({'Satisfied': 1, 'Neutral or Dissatisfied': 0})
    print("'Satisfaction' column converted to numerical ('Satisfaction_Numeric')")
    print("Value counts for 'Satisfaction_Numeric':")
    print(df['Satisfaction_Numeric'].value_counts())

    'Satisfaction' column converted to numerical ('Satisfaction_Numeric')
    Value counts for 'Satisfaction_Numeric':
    Satisfaction_Numeric
    0    73452
    1    56428
    Name: count, dtype: int64
```

Code Snippet 3.3: Python code for converting the categorical 'Satisfaction' column into the numerical Satisfaction_Numeric target variable

Then, categorical features such as *Male* or *Female.* One-Hot encoding is used to convert these features into binary columns that contains only 1 or 0. The command *pd.get_dummies()* performs this conversion.

```
[ ] # One-Hot Encoding
    categorical_cols = ['Gender', 'Customer Type', 'Type of Travel', 'Class']
    df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
    print("Categorical columns one-hot encoded:")
    print(df_encoded.columns)

    Categorical columns one-hot encoded:
    Index(['ID', 'Age', 'Flight Distance', 'Departure Delay', 'Arrival Delay',
           'Departure and Arrival Time Convenience', 'Ease of Online Booking',
           'Check-in Service', 'Online Boarding', 'Gate Location',
           'On-board Service', 'Seat Comfort', 'Leg Room Service', 'Cleanliness',
           'Food and Drink', 'In-flight Service', 'In-flight Wifi Service',
           'In-flight Entertainment', 'Baggage Handling', 'Satisfaction',
           'Satisfaction_Numeric', 'Gender_Male', 'Customer Type_Returning',
           'Type of Travel_Personal', 'Class_Economy', 'Class_Economy Plus'],
          dtype='object')
```

Code Snippet 3.4: Implementation of One-Hot Encoding for categorical features

Before the final dataset is ready, unnecessary columns that are not needed must be removed. After removing, the final dataset is displayed.

```
[ ]  print(f"Shape of the preprocessed dataframe (rows, column): {df_processed.shape}\n")
     print("Final column names: ")
     print(df_processed.columns.tolist())

     print("\nInfo of the preprocessed dataframe: \n")
     df_processed.info()

     print("\nFirst 5 rows of the preprocessed dataframe: ")
     df_processed.head()
```

Table 3.8: A preview of the final preprocessed DataFrame

Lasty, the fully cleaned data frame is saved to ensure reproductivity. This file can be easily accessed to be used in future steps such as model training to avoid wasting time.

```
[ ]  # Save preprocessed data
     output_filename = 'airline_data_preprocessed.csv'
     output_filepath = shared_folder_path + output_filename

     try:
       df_processed.to_csv(output_filepath, index=False)
       print(f"\nPreprocessed data saved to: {output_filepath}")
     except Exception as e:
       print(f"Error saving file: {e}")

     Preprocessed data saved to: /content/drive/MyDrive/IDA Assignment Data/airline_data_preprocessed.csv
```

Code Snippet 3.5: Python code for saving the final preprocessed DataFrame to Google Drive

3.2 Exploratory Data Analysis (EDA)

3.2.1 Univariate Analysis

Following initial data inspection and cleaning, we now have a suitable dataset for further understanding. The first step was to conduct univariate analysis to examine the distribution of our numerical and categorical features. For numerical features, histograms and box plot were constructed to visualize the spread and identify any potential outliers.

As shown in Figure 3.2, the distribution of passengers' age is largely symmetrical, with a slight skewed to the right. The histogram shows that there is a concentration of passengers in mid-adulthood, with fewer passengers at very young or older ages. The distribution of flight distance, however, is strongly right skewed, and its median is below 1000 miles. This indicates that most passengers in this dataset take short to medium haul flights, which is typical for regional travel.

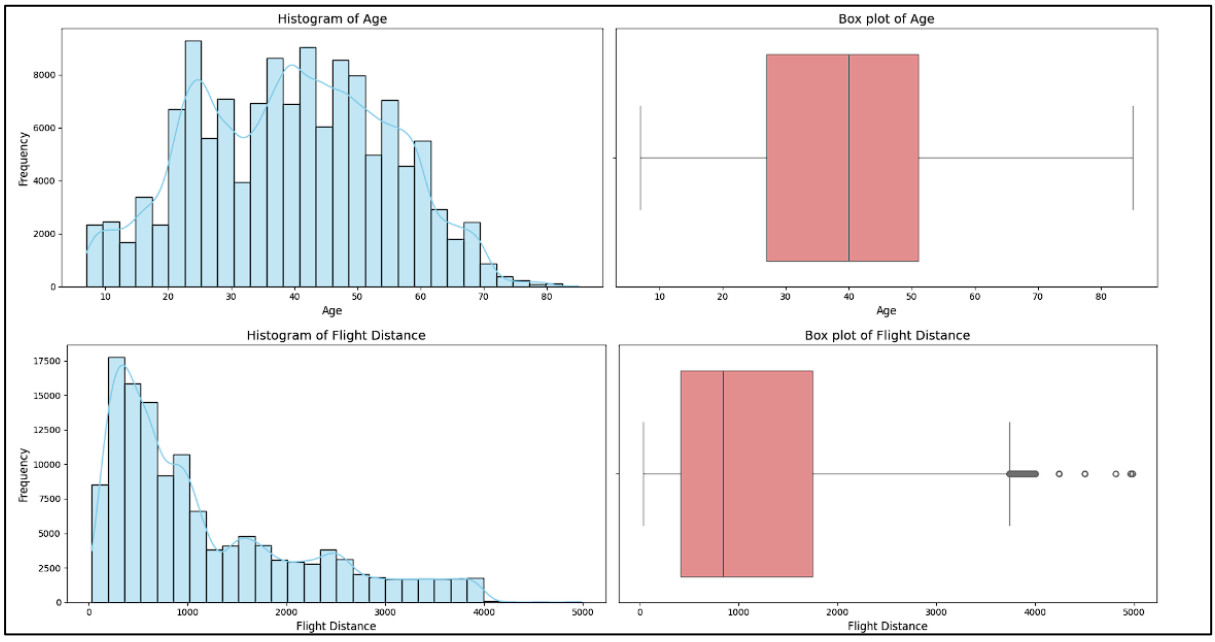


Figure 3.2: Univariate analysis of Age and Flight Distance

The box plot for arrival delay as shown in Figure 3.3 revealed a significant number of outliers, with some delays extending for many hours. This finding confirmed our earlier decision to use median for missing value imputation in this column.

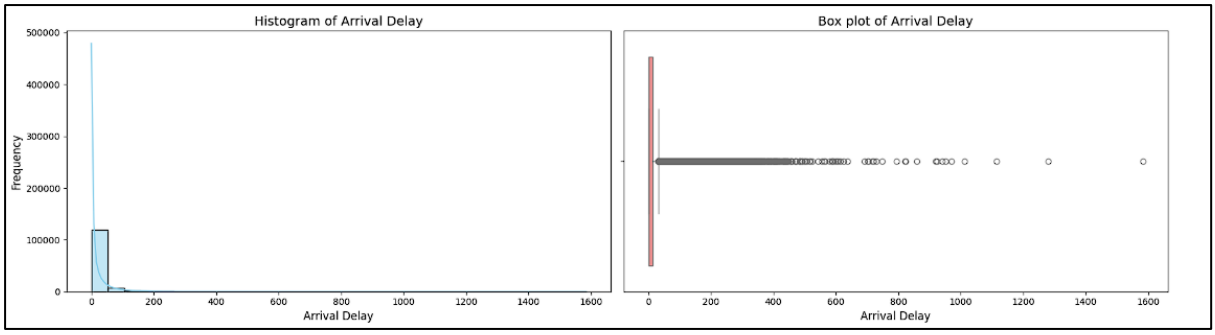


Figure 3.3: Distribution of Arrival Delay

For categorical features, bar charts were used to compare the frequency of different categories. The passengers' gender chart shows a nearly equal number of male and female passengers, indicating a balanced representation. The customer type chart reveals that returning customers outnumbered first-time customers, suggesting a degree of customer loyalty. Meanwhile, business travellers are more than double the number of personal travellers, reflecting that the airline may be more preferred by corporate clients. In the class distribution chart, business class has the most passengers, a finding which aligns with the fact that there are more business travellers in the dataset.

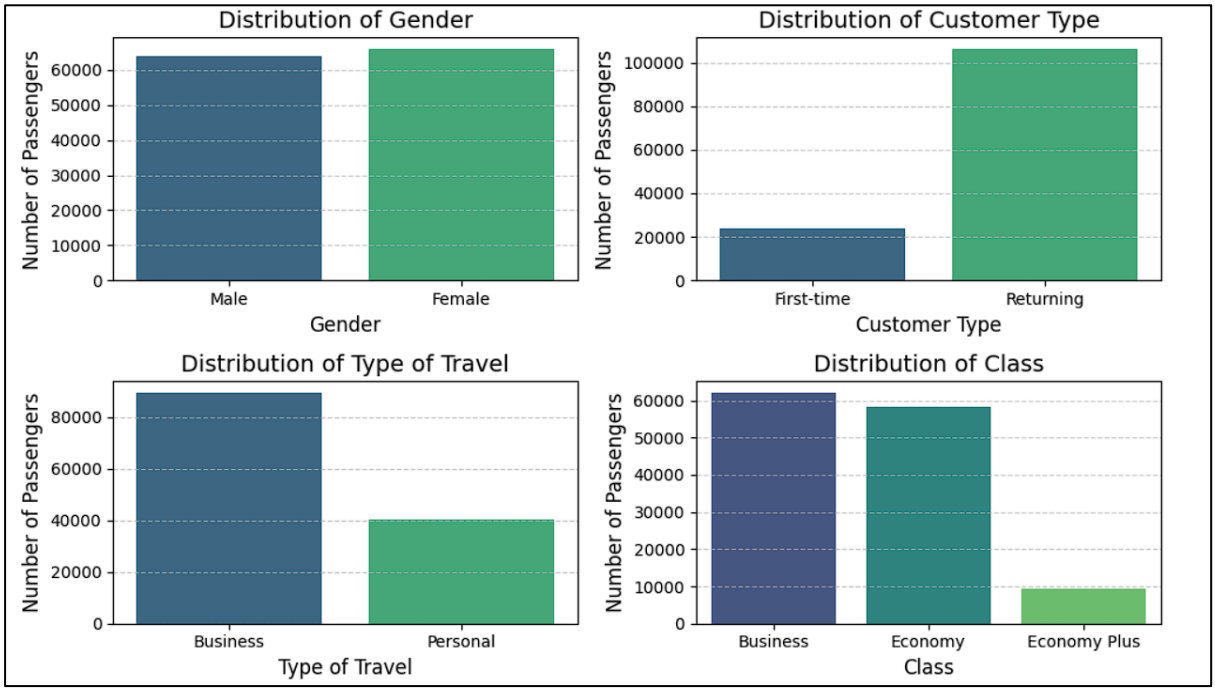
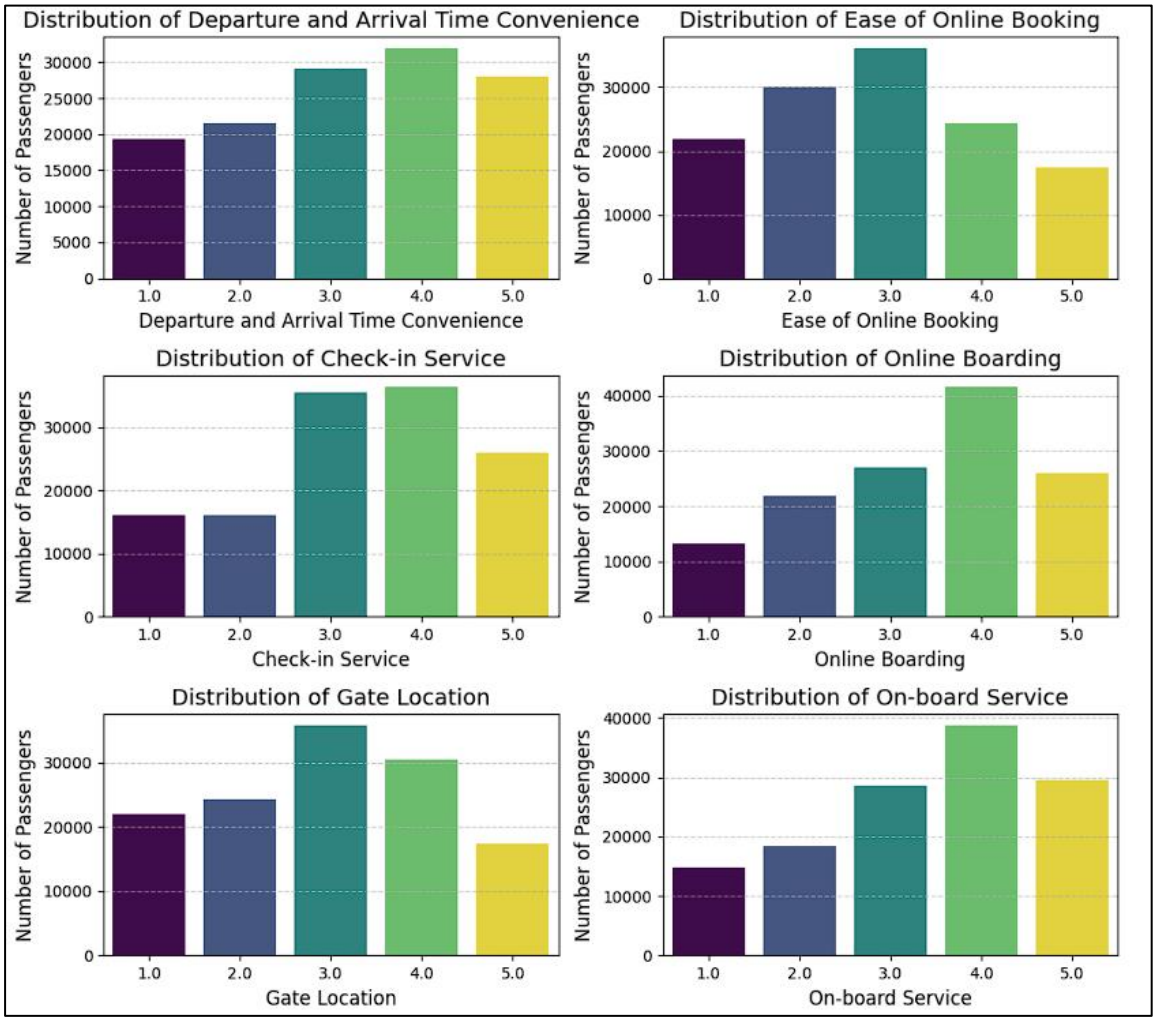


Figure 3.4: Frequency distribution of key categorical segments

To evaluate passengers' satisfaction across various service aspects, bar charts (Figure 3.5) were constructed to analyse the Likert-scale ratings. The scale ranges from 1 (lowest) to 5 (highest). Most charts show a concentration of ratings around 4.0, indicating slight positive feedback. For example, *Check-in Service*, *Online Boarding*, *On-board Service*, *Seat Comfort*, *In-flight Service*, and *Baggage Handling* received high ratings overall. In contrast, *Ease of Online Booking, Gate Location, Leg Room Service, Food and Drink, and In-flight Wifi Service* received generally lower ratings. These findings highlight a clear distinction between different service areas, indicating that while many aspects of the journey are highly rated, specific features like the booking process, gate convenience, and certain in-flight amenities like Wi-Fi and leg room are potential factors of passengers' dissatisfaction.



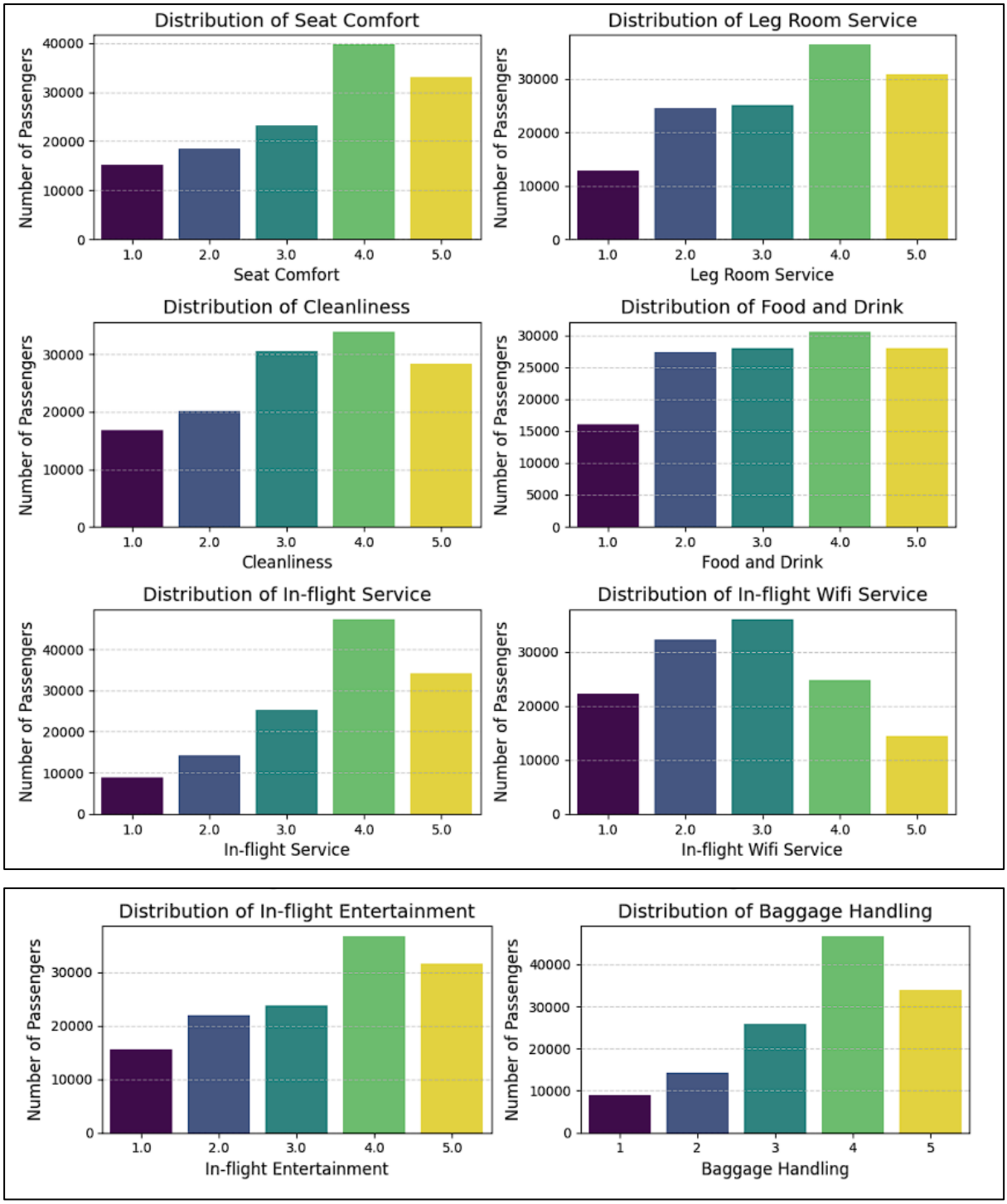


Figure 3.5: Typical service rating distribution

3.2.2 Bivariate Analysis

To further understand the relationship between passengers' ratings and satisfaction, several key features were selected for bivariate analysis. Bar charts (Figure 3.6) were plotted to visualize the satisfaction rate against the values of each feature. As illustrated in the charts for *Online Boarding, In-flight Wifi Service, and Leg Room Service*, the percentage of passenger satisfied climbs sharply when the ratings are 4 or 5. For online boarding, a rating of 5 correspond to an 87.1% satisfaction rate, whereas a rating of 3 yields only 13.8% satisfaction rate. This suggest that *Online Boarding* may be a strong driver of passenger satisfaction. For *Type of Travel* and *Class*, business travel and business class are more satisfied than personal travel and business class. Based on these findings, we suggest that strategic improvements should focus on enhancing the experience of economy class and personal travel passengers, as they represent the largest opportunity to increase satisfaction.

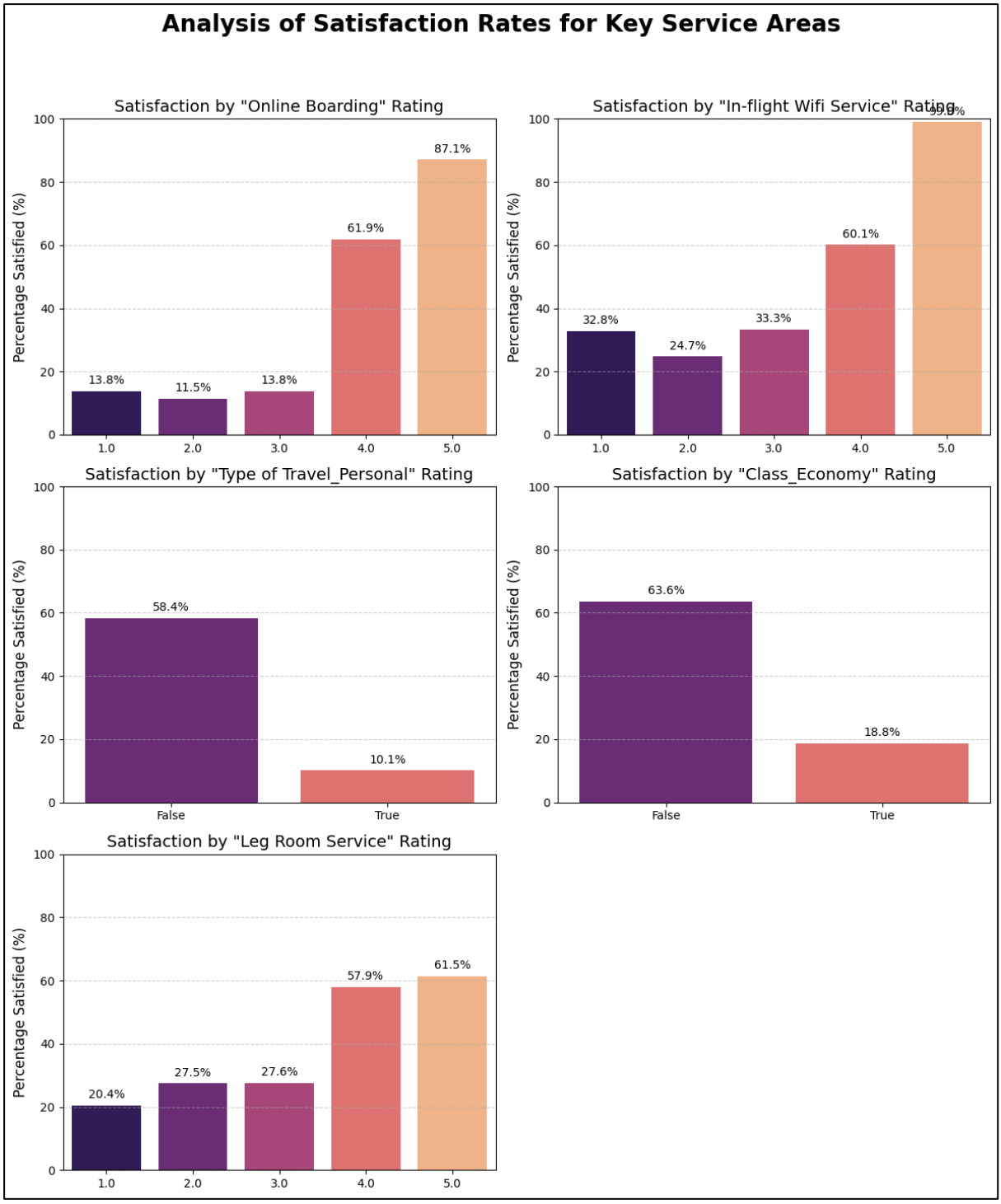


Figure 3.6: Bivariate analysis showing the direct relationship between service ratings and the percentage of satisfied passengers

3.2.3 Correlation and Statistical Analysis

To supplement the bivariate graphical analysis, we performed a comprehensive quantitative analysis of the relationships between all the features and the target variable.

Pearson Correlation Analysis

A correlation matrix heatmap (Figure 3.7) was generated to discover the linear relationships between variables. According to the heatmap, the correlation between *Departure Delay and Arrival Delay* is strong. This finding was not a surprise as they were logically related. However, the correlation between most features and the target variable (*Satisfaction_Numeric*) is relatively weak. This is a critical finding as it suggests that simple linear method are likely insufficient to determine the key drivers of satisfaction in our dataset. Therefore, we needed to explore other methods that can capture non-linear patterns.

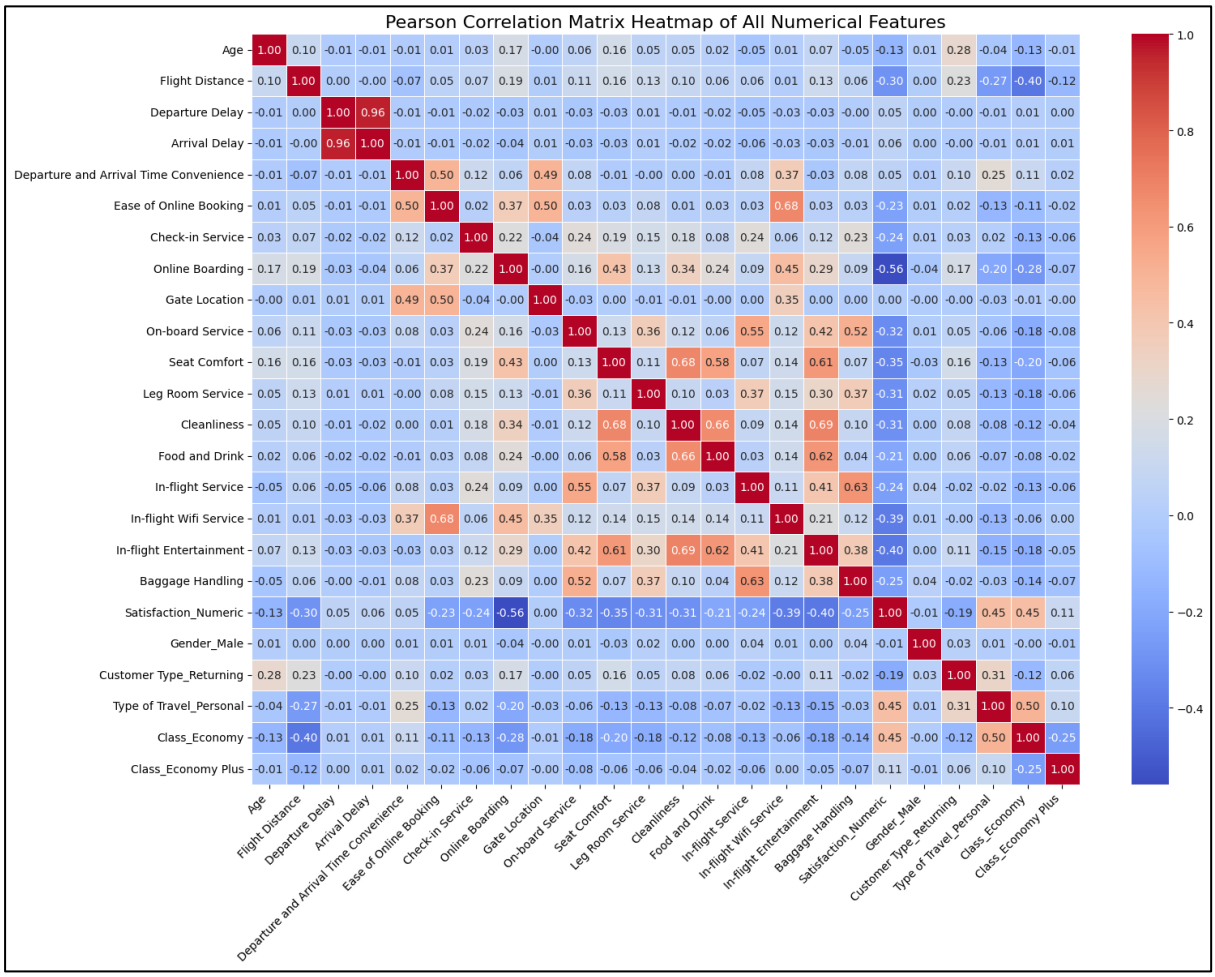


Figure 3.7: Pearson Correlation Matrix

Mutual Information

Hence, Mutual Information (MI) analysis was performed to gain a clearer picture of feature importance. The mutual dependence between *Satisfaction_Numeric* and the other features is measured by MI, capturing both linear and non-linear relationships. We obtained the degree of dependence of each variable on *Satisfaction_Numeric* and sorted them from highest to lowest as shown below.

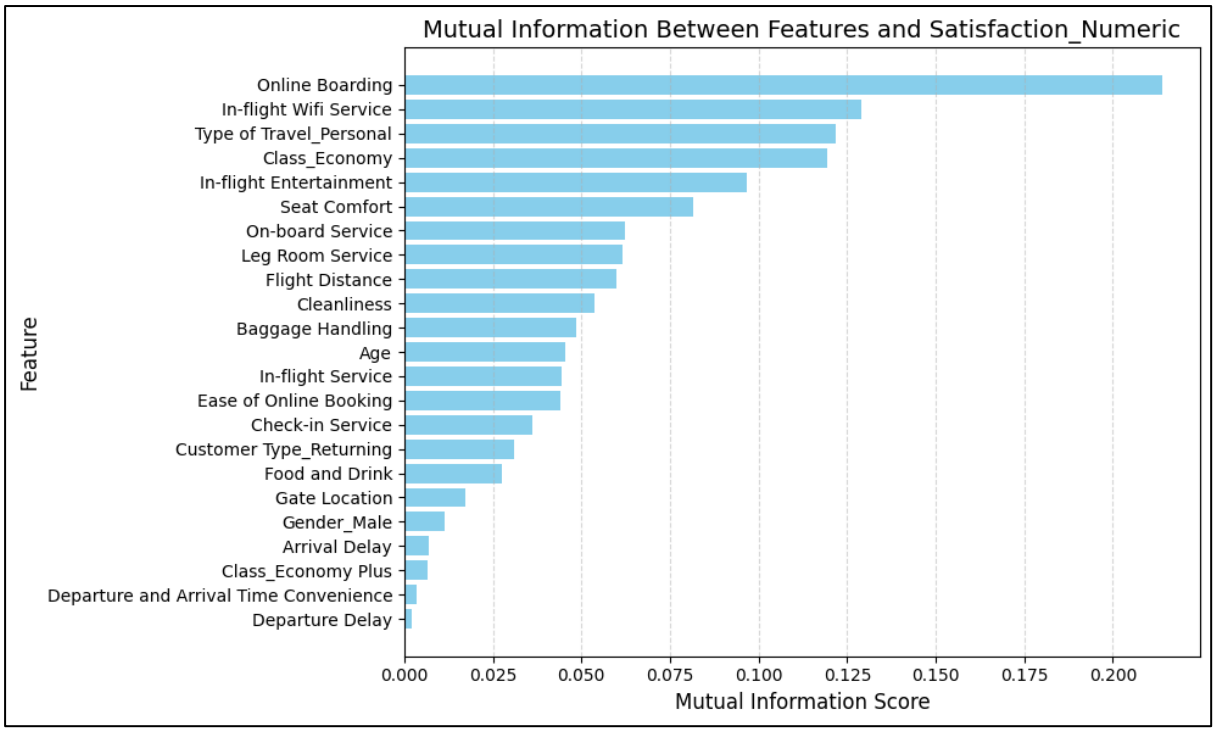


Figure 3.8: Feature Importance Ranking by Mutual Information

Chi-Squared Test

The Chi-Squared test of independence was also performed to assess the association between each feature and target variable. The results in Table 3.9 were definitive. The calculated p-value for all 23 features was less than 0.0001, which is significantly below our alpha level of 0.05. This allows us to reject the null hypothesis for every feature, confirming that there is statistically significance relationships exist between all the features and passenger satisfaction. The ranking in Figure 3.9 clearly shows which features have the most influential relationship with the target variable.

--- Chi-Squared Test Results Summary (against Satisfaction_Numeric) ---
Sorted in descending order based on Chi2 Test Values

Feature	Chi2 Statistic	P-value	Significant (alpha=0.05)
Online Boarding	49491.6338	0.0000	Yes
In-flight Wifi Service	27991.9354	0.0000	Yes
Type of Travel_Personal	26282.5210	0.0000	Yes
Class_Economy	26227.2062	0.0000	Yes
In-flight Entertainment	23044.2554	0.0000	Yes
Seat Comfort	19537.4627	0.0000	Yes
Flight Distance	18443.5420	0.0000	Yes
Leg Room Service	15064.4759	0.0000	Yes
On-board Service	14336.8449	0.0000	Yes
Cleanliness	12938.2364	0.0000	Yes
Age	11240.6006	0.0000	Yes
Baggage Handling	10820.2135	0.0000	Yes
Ease of Online Booking	10384.3535	0.0000	Yes
In-flight Service	10353.2767	0.0000	Yes
Check-in Service	8142.9462	0.0000	Yes
Food and Drink	6571.0049	0.0000	Yes
Customer Type_Returning	4493.1888	0.0000	Yes
Gate Location	3068.1729	0.0000	Yes
Arrival Delay	2051.7690	0.0000	Yes
Class_Economy Plus	1459.4350	0.0000	Yes
Departure Delay	1166.9393	0.0000	Yes
Departure and Arrival Time Convenience	565.5858	0.0000	Yes
Gender_Male	16.3521	0.0001	Yes

Table 3.9: Summary of Chi-Squared test results

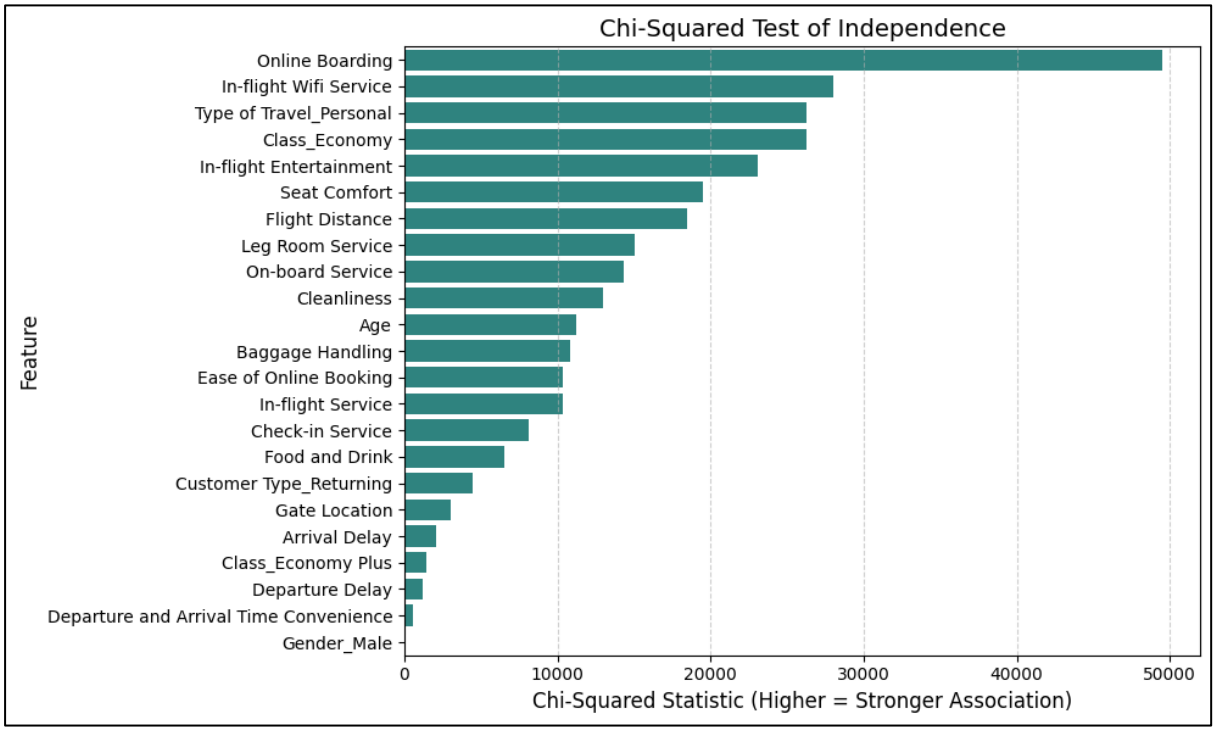


Figure 3.9: Feature Association Ranking by the Chi-Squared Statistic

ANOVA F-test

Finally, an Analysis of Variance (ANOVA) F-test was conducted. This test determines whether there are any statistically significant differences in the means of a numerical or rating-based feature between the two satisfaction groups ('Satisfied' vs. 'Dissatisfied'). A high F-statistic indicates that the means are significantly different. The ranking of features by their F-statistic is shown in Figure 3.10.

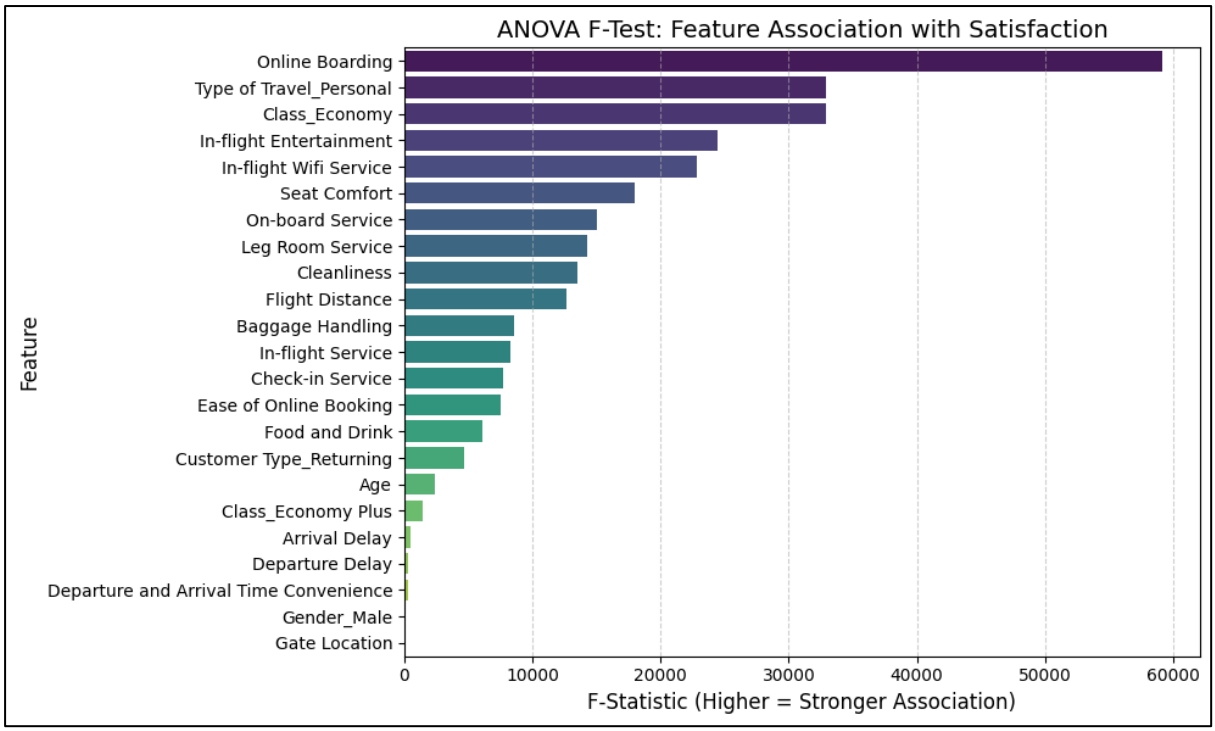


Figure 3.10: Feature Association Ranking by the ANOVA F-Statistic

Summary of Statistical Analysis

From the three statistical analyses conducted, we can observe that features such as *Online Boarding, In-flight Wifi Service, Type of Travel_Personal, Class_Economy, In-flight Entertainment, and Seat Comfort* frequently appeared at the top of the rankings. This consistent set of findings gives us high confidence that they are the key predictors of passenger satisfaction.

3.3 Feature Engineering and Selection

3.3.1 Feature Scaling

Before proceeding with the model-based feature selection process, the preprocessed dataset was separated into two components. Feature matrix (*X*) was created by removing the target variable using *drop()* method to our preprocessed data frame. Target vector (*y*) was created to store only the target variable, *Statisfaction_Numeric.*

```
[ ]  # All columns except the target variable
     X = df_processed.drop('Satisfaction_Numeric', axis=1)

     # Only the target variable
     y = df_processed['Satisfaction_Numeric']

     print("Feature matrix shape:", X.shape)
     print("Target vector shape:", y.shape)

     Feature matrix shape: (129880, 23)
     Target vector shape: (129880,)
```

Code Snippet 3.6: Python code for separating the dataset into the feature matrix (X) and the target vector (y)

Then, a final feature engineering step was conducted, which is feature scaling. The purpose of performing feature scaling is that in our dataset, there are continuous numerical features such as *Age,* measured in years, and *Flight Distance* measured in miles. These features, with different scales and units might influence the performance of feature selection methods such as Recursive Feature Elimination (RFE) with Logistic Regression due to their sensitivity towards the magnitude of features.

To address this potential problem, we applied Standardization to the continuous numerical features within the feature matrix (*X*). Scikit-learn's *StandardScaler* was used to transform the features to have a mean of 0 and a standard deviation of 1. This ensures that the importance of a feature is determined by its predictive power, not its original unit of measurement. The resulting scaled feature matrix (*X_scaled*) was used for all subsequent feature selection and modelling tasks.

```
[ ]  # Standardisation (Scaling) for numerical columns
     # Create a copy to avoid changing the original X DataFrame
     X_scaled = X.copy()

     # Initialize the scaler
     scaler = StandardScaler()

     # Fit the scaler on the numerical columns and transform them
     X_scaled[numerical_features] = scaler.fit_transform(X[numerical_features])

     print("\nFirst 5 rows of the scaled data:")
     X_scaled.head()
```

Code Snippet 3.7: Implementation of StandardScaler

First 5 rows of the scaled data:

	Age	Flight Distance	Departure Delay	Arrival Delay
0	0.566960	-0.370261	-0.333948	-0.261490
1	-0.292868	-0.370261	0.296454	0.623553
2	0.103976	-0.338179	-0.386481	-0.391644
3	0.699242	0.716512	-0.386481	-0.391644
4	0.633101	2.285515	-0.386481	-0.365613

5 rows × 23 columns

Table 3.10: A preview of the X_scaled DataFrame

3.3.2 Generating Feature Sets with Filter, Wrapper, and Embedded Methods

To ensure a robust and reliable final feature set, we generated multiple candidate sets using different approaches. Each approach has its own strength and trade-offs. Therefore, we will select the final feature set after performing evaluation.

Filter Method: Mutual Information

First, we applied a filter method using Mutual Information (MI). From 3.2.1 Correlation and Statistical Analysis, we learn that MI returns a score for dependency between two features. By using scikit-learn's mutual_info_classif() function, we calculate the MI score for every feature in our scaled dataset (*X_scaled*) against the target (*y*). The results were then visualized as a horizontal bar chart to clearly rank the features by their importance.

From the ranking shown in Table 3.11, the top 15 features were selected as charts the demonstrate a clear pattern of diminishing returns. While the top feature *Online Boarding* has a very high MI score, the score declines steadily. By the 15th feature, *Check-in Service*, the score is significantly lower. Furthermore, there is a gap of 0.005 to the 16th feature and any subsequent features would contribute even less predictive information. Therefore, selecting the top 15 features provides a comprehensive set of the most impactful variables without including the low-value features at the tail end of the distribution.

```
[18] print("Filter Method")

     # Filter method: Mutual Information

     print("Applying Filter Method: Mutual Information")

     mi_scores = mutual_info_classif(X_scaled, y, random_state=42)

     # Create dataframe to hold the feature names and their MI scores
     mi_scores_df = pd.DataFrame({'Feature': X_scaled.columns, 'MI_Score': mi_scores}).sort_values(by='MI_Score', ascending=False)

     print("Top 20 Features by Mutual Information Score:")
     print(mi_scores_df.head(20).to_string(index=False))

     # Get the list of the top 15 features from Mutual Information
     N_FEATURES_TO_SELECT = 15
     features_MI = mi_scores_df.head(N_FEATURES_TO_SELECT)['Feature'].tolist()

     print(f"List of Top {N_FEATURES_TO_SELECT} features from Mutual Information:")
     print(features_MI)

     # Visualize MI Scores
     plt.figure(figsize=(10, 8))
     sns.barplot(x='MI_Score', y='Feature', data=mi_scores_df.head(20), palette='viridis', hue='Feature', legend=False)
     plt.title('Top 20 Features Ranked by Mutual Information', fontsize=16, fontweight='bold')
     plt.xlabel('Mutual Information Score', fontsize=12)
     plt.ylabel('Feature', fontsize=12)
     plt.grid(axis='x', linestyle='--', alpha=0.6)
     plt.tight_layout()
     plt.show()
```

Code Snippet 3.8: Implementation of mutual_info_classif

```
Filter Method
Applying Filter Method: Mutual Information
Top 20 Features by Mutual Information Score:
                  Feature  MI_Score
          Online Boarding  0.213711
 In-flight Wifi Service  0.130200
Type of Travel_Personal  0.120834
           Class_Economy  0.118354
In-flight Entertainment  0.096600
            Seat Comfort  0.081531
       Leg Room Service  0.061718
        Flight Distance  0.061705
       On-board Service  0.061383
            Cleanliness  0.057721
       Baggage Handling  0.047432
      In-flight Service  0.043459
                    Age  0.042928
 Ease of Online Booking  0.042543
       Check-in Service  0.035385
         Food and Drink  0.030617
Customer Type_Returning  0.029589
           Gate Location  0.016046
             Gender_Male  0.007959
           Arrival Delay  0.007314
```

Table 3.11: Top 20 features and their corresponding Mutual Information scores

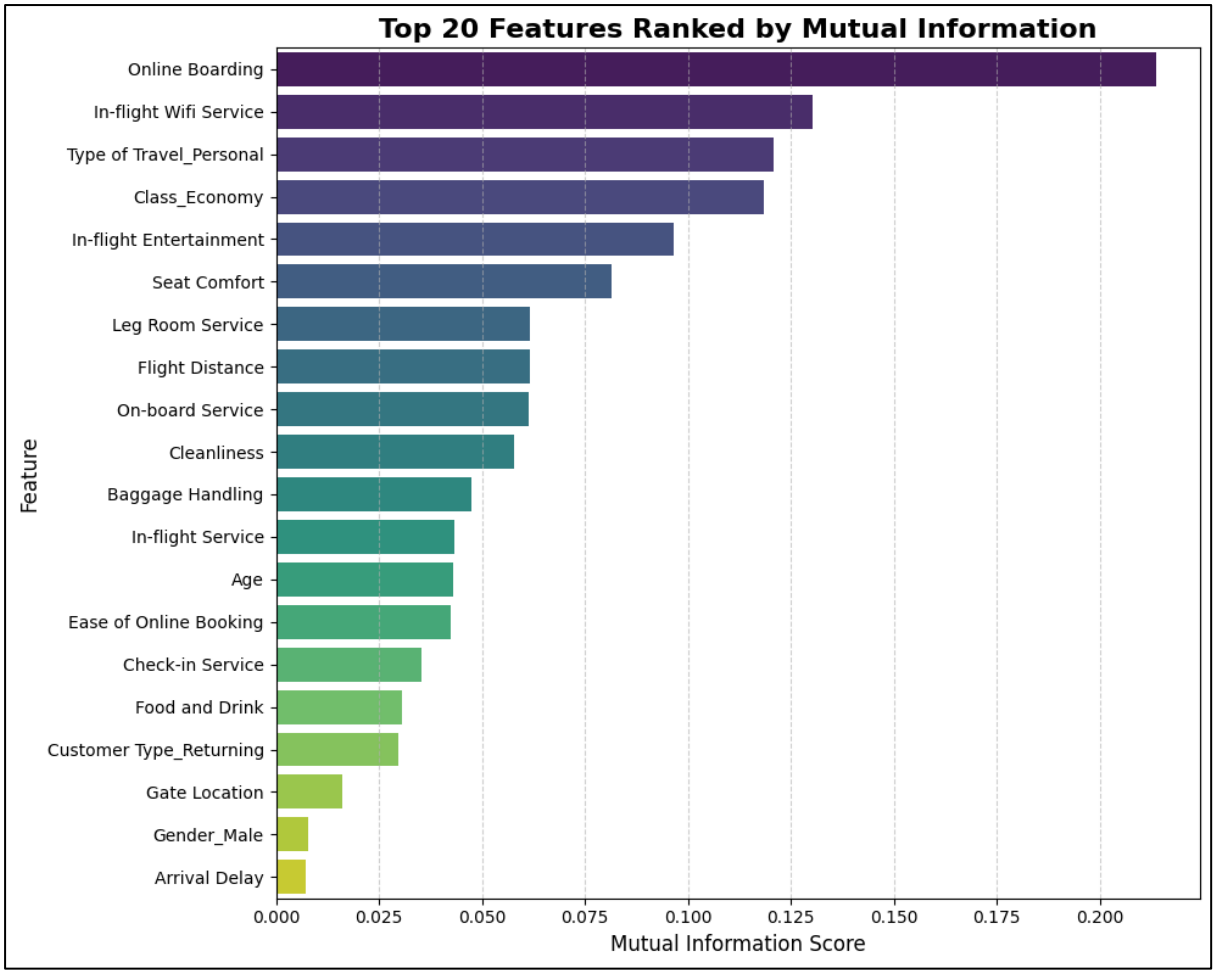


Figure 3.11: Visualization of the Top 15 features as ranked by Mutual Information

Wrapper Method: Recursive Feature Elimination (RFE)

Next, we employed a wrapper method, Recursive Feature Elimination (RFE). RFE iteratively trains a model, and in each iteration, the weakest feature is eliminated. The model's *.ranking* attribute reveals the elimination order. A rank of 1 means the feature was selected, and for the subsequent rank, the higher the rank number, the earlier the feature was removed. In our case, we use Logistic Regression for RFE because it is efficient and fast. We configured it to select the top 15 features. This number was chosen to maintain consistency with our filter method, ensuring fairness during the feature evaluation process.

```
[ ]  estimator_for_rfe = LogisticRegression(max_iter=1000, random_state=42)

     # Select top 15 features using RFE

     rfe_selector = RFE(estimator=estimator_for_rfe, n_features_to_select=15, step=1)

     print("Fitting RFE...")

     rfe_selector.fit(X_scaled, y)

     print(f"RFE fitting complete.")

     # Get features selected by RFE
     selected_mask_rfe = rfe_selector.support_

     # Use the mask to select the column names from original dataframe
     features_RFE = X_scaled.columns[selected_mask_rfe].tolist()

     print(f"List of {len(features_RFE)} features selected by RFE:")
     print(features_RFE)

     rfe_ranking_df = pd.DataFrame({
         'Feature': X_scaled.columns,
         'RFE_Rank': rfe_selector.ranking_
     }).sort_values(by='RFE_Rank', ascending=True)

     print("\nRanking of all features from RFE (Rank 1 means 'selected)':")
     print(rfe_ranking_df.to_string(index=False))

     print("Wrapper method task complete")
```

Code Snippet 3.9: Python code for initializing and running Recursive Feature Elimination (RFE)

```
Fitting RFE...
RFE fitting complete.
List of 15 features selected by RFE:
['Arrival Delay', 'Departure and Arrival Time Convenience',

Ranking of all features from RFE (Rank 1 means 'selected)':
                                Feature  RFE_Rank
                          Arrival Delay         1
                        Check-in Service         1
                  Ease of Online Booking         1
Departure and Arrival Time Convenience         1
                         Online Boarding         1
                        Leg Room Service         1
                        On-board Service         1
                           Gate Location         1
                  In-flight Wifi Service         1
                       In-flight Service         1
                             Cleanliness         1
                  Customer Type_Returning         1
                  Type of Travel_Personal         1
                            Class_Economy         1
                       Class_Economy Plus         1
                          Departure Delay         2
                         Baggage Handling         3
                                      Age         4
                  In-flight Entertainment         5
                           Food and Drink         6
                             Gender_Male         7
                          Flight Distance         8
                             Seat Comfort         9
Wrapper method task complete
```

Table 3.12: The full ranking of all features as determined by the RFE process

Embedded Methods: Lasso and Random Forest Importance

Finally, we generated two feature sets using embedded methods, where feature selection is part of the model training process.

1. **Lasso (L1 Regularization)**

   A Logistic Regression model was trained with an L1 penalty to automatically eliminate less important features by shrinking their coefficients to zero. To select the most relevant features while maintaining generalization, five-fold cross-validation (*cv=5*) was used to evaluate model performance across different subsets of data. The model tested 20 different values for the regularization strength parameter (*Cs=20*) and selected the optimal C value that balanced the model accuracy and sparsity. As a result, 20 features were retained in the *features_lasso* set, representing them what the model identified as the most predictive features of the target variable.

```
[ ]  print("---- Applying Lasso (L1 Regularisation) ----")

     lasso_cv = LogisticRegressionCV(Cs=20, cv=5, penalty='l1', solver='liblinear', random_state=42, max_iter=1000)

     print("Fitting LassoCV to find optimal regularization...")
     lasso_cv.fit(X_scaled, y)
     print("LassoCV fitting complete.")

     # The coefficients of unimportant features will be shrunk to exactly zero.
     # We get the coefficients from the fitted model.
     lasso_coefficients = lasso_cv.coef_[0]

     # Get the list offeatures selected by Lasso
     features_Lasso = X_scaled.columns[lasso_coefficients != 0].tolist()

     print(f"Lasso selected {len(features_Lasso)} features with an optimal C value of {lasso_cv.C_[0]:.4f}:")
     for features in features_Lasso:
       print(features)
```

Code Snippet 3.10: Implementation of LogisticRegressionCV with an L1 penalty

```
---- Applying Lasso (L1 Regularisation) ----
Fitting LassoCV to find optimal regularization...
LassoCV fitting complete.
Lasso selected 20 features with an optimal C value of 0.0127:
Age
Flight Distance
Arrival Delay
Departure and Arrival Time Convenience
Ease of Online Booking
Check-in Service
Online Boarding
Gate Location
On-board Service
Leg Room Service
Cleanliness
Food and Drink
In-flight Service
In-flight Wifi Service
In-flight Entertainment
Baggage Handling
Customer Type_Returning
Type of Travel_Personal
Class_Economy
Class_Economy Plus
```

Table 3.13: The final list of features selected by the optimized Lasso model

2. Random Forest Importance

Finally, a Random Forest model was trained on all 23 features. By accessing its attribute *feature_importances_* attribute, we obtain a score of each feature. The score is calculated by measuring each feature's contribution to reducing impurity across all the decisions trees in the forest. A horizontal bar chart was created to visualize the top 20 features ranked by Random Forest. From this ranking, the top 15 features from this ranking were selected to create the *features_rf_importance* set.

```
[ ]  print("\n---- Applying Random Forest Feature Importance ----")

     rf_importance_classifier = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)

     print("Fitting Random Forest to calculate feature importances...")
     rf_importance_classifier.fit(X_scaled, y)
     print("Random Forest fitting complete.")

     rf_importance_df = pd.DataFrame({
         'Feature': X_scaled.columns,
         'Importance': rf_importance_classifier.feature_importances_
     }).sort_values(by='Importance', ascending=False)

     # Get the list of top 15 features from RF Importance
     features_RF_importance = rf_importance_df.head(15)['Feature'].tolist()

     print(f"List of Top 15 features from Random Forest Importance:")
     print(features_RF_importance)

     plt.figure(figsize=(10, 8))
     sns.barplot(x='Importance', y='Feature', data=rf_importance_df.head(15), palette='plasma', hue='Feature', legend=False)
     plt.title('Top 15 Features Ranked by Random Forest Importance', fontsize=16, fontweight='bold')
     plt.xlabel('Importance Score (Higher is more important)', fontsize=12)
     plt.ylabel('Feature', fontsize=12)
     plt.grid(axis='x', linestyle='--', alpha=0.6)
     plt.tight_layout()
     plt.show()

     print("Embedded Tasks Complete.")
```

Code Snippet 3.11: Python code for training a Random Forest model to extract feature importances

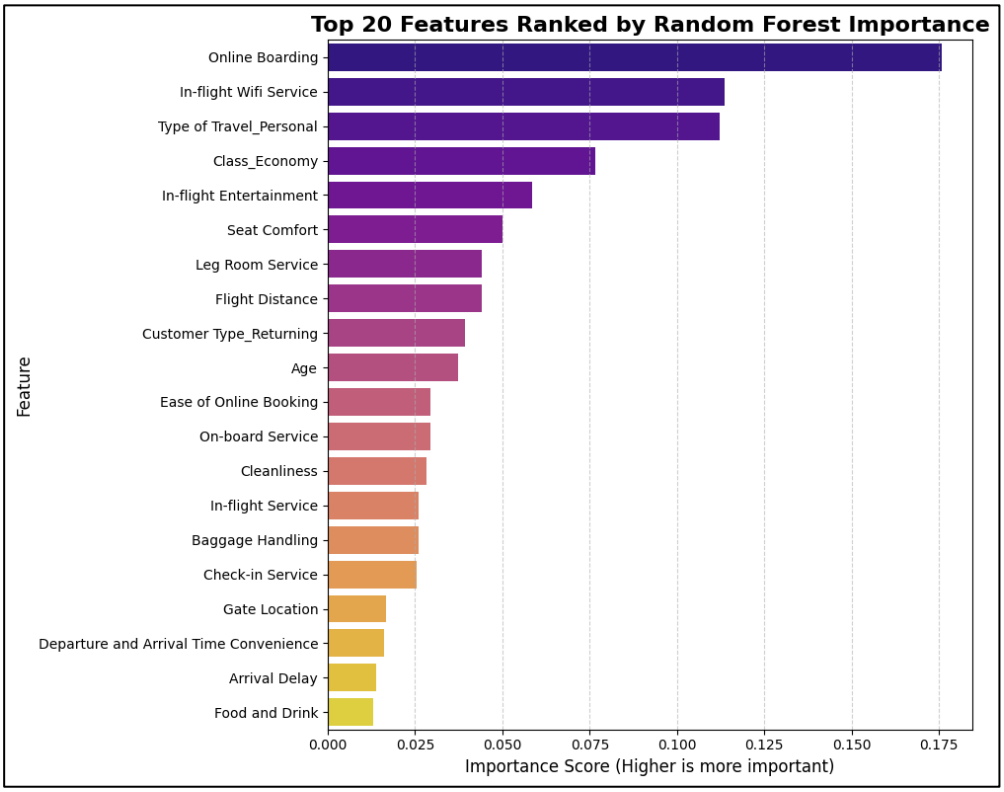


Figure 3.12: Visualization of the Top 15 features as ranked by Random Forest Importance

Feature Extraction Method: Principal Component Analysis (PCA)

In addition to feature selection methods, we also explored feature extraction using Principal Component Analysis (PCA). PCA is performed because we wanted to create a high-performance trained model while sacrificing the interpretability. We configured the PCA to select the minimal number of components to explain 95% of the total variance of our data set. This process resulted in a set of 16 components. This PCA-transformed dataset was then used as a non-interpretable benchmark in our feature evaluation section.

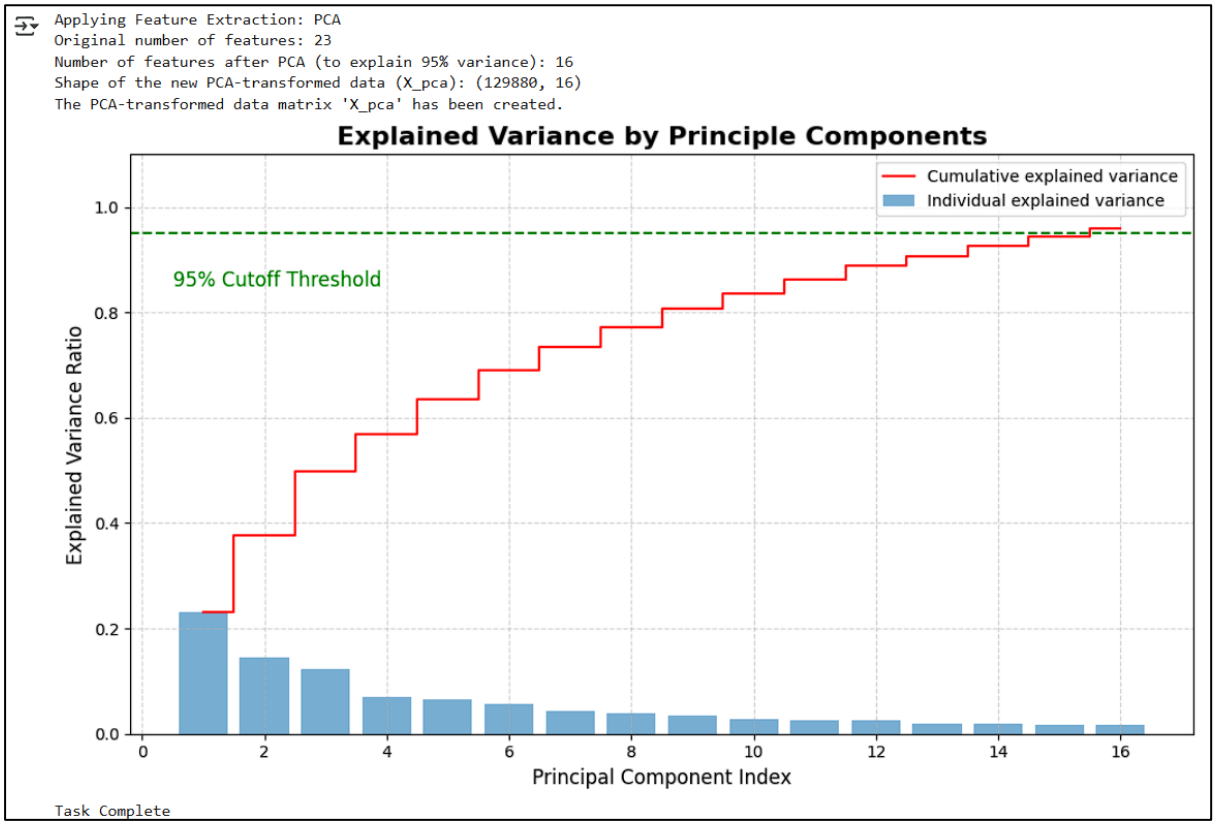


Figure 3.13: Explained Variance by Principal Components

3.3.3 Comparative Evaluation and Final Selection

Having generated four distinct feature sets, a “Grand Comparison” was conducted to evaluate which strategy produces the best results.

Grand Comparison of Feature Sets with Model Types

Each candidate feature sets, along with the original 23 features and PCA-derived set were trained with three different model types: Logistic Regression, Decision Tree, and Random Forest, without fine-tuning. We used Area Under the Receiver Operating Characteristic Curve

(ROC AUC) score, evaluated via a 5-fold cross-validation. This method was chosen because it measures the model's ability to distinguish between positive and negative classes. The higher the AUC score, the better the model performance.

The final comparison results were displayed in table form as in Table 3.14, sorted by best mean AUC. A clustered bar chart (Figure 3.14) was plotted for an easier performance comparison of different models across various feature set, with each cluster representing a feature set and each bar within the cluster corresponding to the specific model.

The findings of this comprehensive evaluation yield a clear and decisive outcome. The Random Forest classifier outperforms other algorithm consistently across all feature sets by comparing their mean AUC scores.

Among all of the feature sets, the top 15 features from RF Importance proves it is the most effective and efficient when paired with Random Forest. It achieved an AUC score of 0.9899, a score that is extremely closed to the model with all 23 features (0.9915) while minimal features were used. This result provides strong evidence that this feature set was the best performing out of all the others.

```
--- FINAL COMPARISON RESULTS (Sorted by Best Mean AUC) ---
            Feature Set                Model  Num_Features  Mean_AUC  Std_AUC
           All Features        Random Forest            23  0.991540 0.003155
    Lasso (20 features)        Random Forest            20  0.991279 0.003544
   RF Importance Top 15        Random Forest            15  0.989992 0.003550
              MI Top 15        Random Forest            15  0.988792 0.004319
             RFE Top 15        Random Forest            15  0.987501 0.004995
                    PCA        Random Forest            16  0.974606 0.010392
    Lasso (20 features) Logistic Regression            20  0.952092 0.013454
           All Features Logistic Regression            23  0.952051 0.013545
             RFE Top 15 Logistic Regression            15  0.952021 0.013574
   RF Importance Top 15 Logistic Regression            15  0.945648 0.013654
              MI Top 15 Logistic Regression            15  0.938623 0.019012
           All Features        Decision Tree            23  0.936226 0.014940
    Lasso (20 features)        Decision Tree            20  0.933939 0.015093
   RF Importance Top 15        Decision Tree            15  0.932072 0.014050
             RFE Top 15        Decision Tree            15  0.926147 0.015818
                    PCA Logistic Regression            16  0.924442 0.027841
              MI Top 15        Decision Tree            15  0.922044 0.018169
                    PCA        Decision Tree            16  0.857944 0.032974
```

Table 3.14: Grand Comparison of Model Performance (Mean AUC) across all candidate feature sets

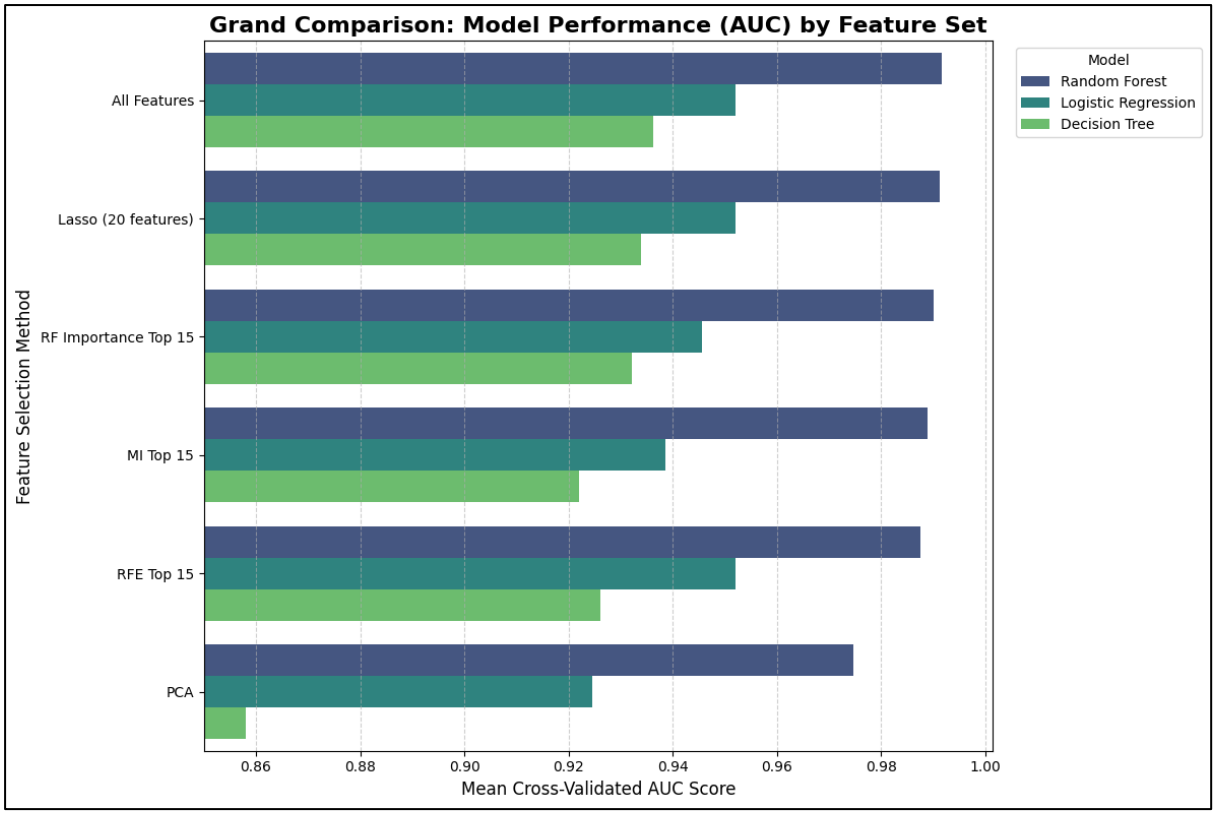


Figure 3.14: Visualization of the Grand Comparison results

Investigation of Class Imbalance (SMOTE)

From what we learned in our target variable distribution analysis, the number of neutral or dissatisfied passengers are slightly higher than satisfied passengers. Therefore, we incorporated Synthetic Minority Oversampling Technique (SMOTE) into our feature sets selection process to address the mild class imbalance of the target variable. The *imblearn.pipeline.Pipeline* class was used to incorporate SMOTE resampling step within each fold of the cross-validation process, preventing data leakage.

The results showed SMOTE had a mixed and minimal impact. While it led to negligible improvements in the mean AUC for a few model-feature set pairs, it also slightly decreased performance in others. Specifically, for our top 15 RF Importance feature set paired with a Random Forest Model, SMOTE improved the mean AUC score by only 0.000092, an insignificant amount.

Based on these findings, we concluded that SMOTE was not a necessary step for this project as it did not justify the added computational expense and model complexity. Therefore, we proceeded with the original, unbalanced data.

```
--- FINAL COMPARISON RESULTS (Sorted by Best Mean AUC) with SMOTE ---
            Feature Set                          Model  Num_Features  Mean_AUC  Std_AUC
           All Features       Random Forest + SMOTE            23  0.991569 0.003133
    Lasso (20 features)       Random Forest + SMOTE            20  0.991400 0.003392
RF Importance Top 15          Random Forest + SMOTE            15  0.990084 0.003536
              MI Top 15       Random Forest + SMOTE            15  0.988855 0.004506
             RFE Top 15       Random Forest + SMOTE            15  0.987471 0.004910
                    PCA       Random Forest + SMOTE            16  0.974990 0.010333
    Lasso (20 features) Logistic Regression + SMOTE            20  0.952013 0.012825
             RFE Top 15 Logistic Regression + SMOTE            15  0.951995 0.012964
           All Features Logistic Regression + SMOTE            23  0.951894 0.012900
RF Importance Top 15 Logistic Regression + SMOTE            15  0.945645 0.012789
              MI Top 15 Logistic Regression + SMOTE            15  0.938603 0.018484
           All Features       Decision Tree + SMOTE            23  0.935492 0.014117
    Lasso (20 features)       Decision Tree + SMOTE            20  0.933119 0.015626
RF Importance Top 15          Decision Tree + SMOTE            15  0.930372 0.013785
             RFE Top 15       Decision Tree + SMOTE            15  0.925873 0.016437
                    PCA Logistic Regression + SMOTE            16  0.924496 0.027099
              MI Top 15       Decision Tree + SMOTE            15  0.921337 0.018748
                    PCA       Decision Tree + SMOTE            16  0.857698 0.031111
```

Table 3.15: Grand Comparison of Model Performance (Mean AUC) with the SMOTE balancing technique applied

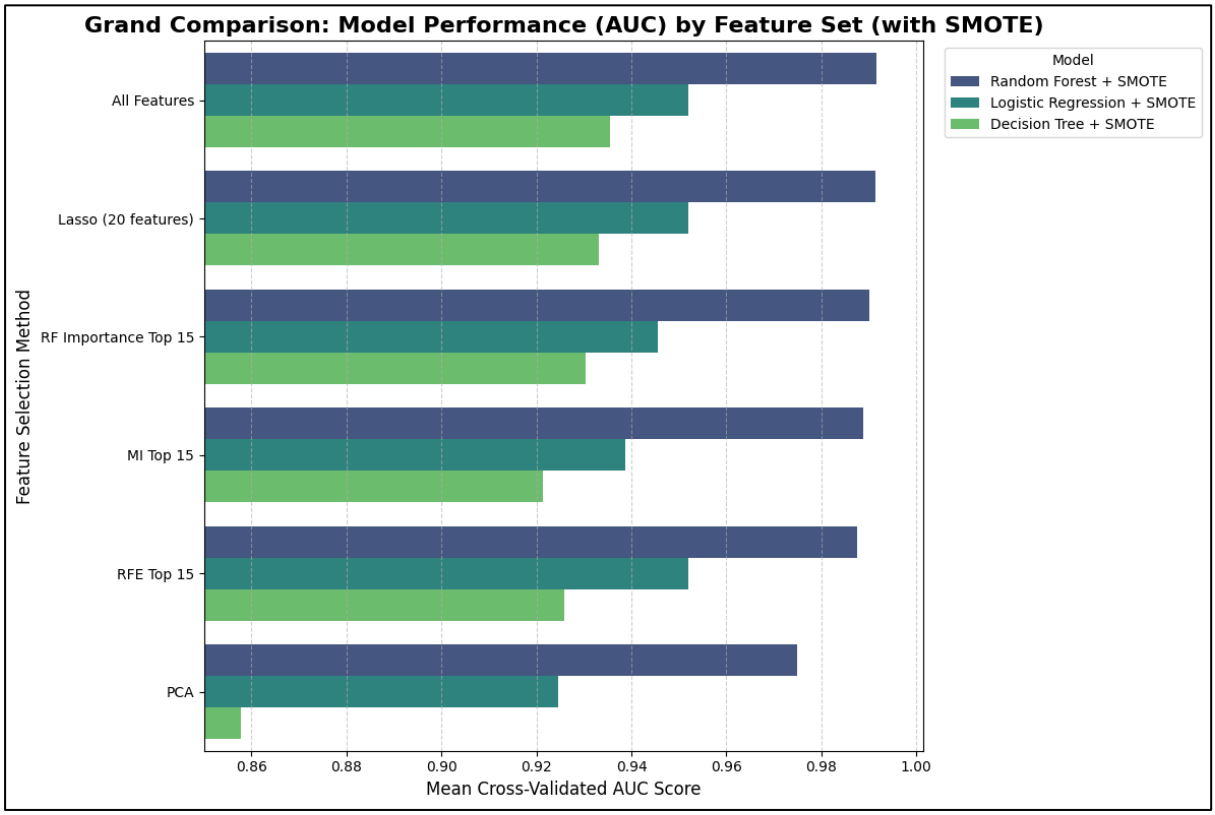


Figure 3.15: Visualization of the Grand Comparison results with SMOTE

```
--- MASTER RESULTS TABLE: ALL EXPERIMENTS SORTED BY BEST MEAN AUC ---
          Feature Set               Model Balancing  Num_Features  Mean_AUC  Std_AUC
         All Features       Random Forest     SMOTE            23  0.991569 0.003133
         All Features       Random Forest      None            23  0.991540 0.003155
  Lasso (20 features)       Random Forest     SMOTE            20  0.991400 0.003392
  Lasso (20 features)       Random Forest      None            20  0.991279 0.003544
RF Importance Top 15       Random Forest     SMOTE            15  0.990084 0.003536
RF Importance Top 15       Random Forest      None            15  0.989992 0.003550
           MI Top 15       Random Forest     SMOTE            15  0.988855 0.004506
           MI Top 15       Random Forest      None            15  0.988792 0.004319
          RFE Top 15       Random Forest      None            15  0.987501 0.004995
          RFE Top 15       Random Forest     SMOTE            15  0.987471 0.004910
                 PCA       Random Forest     SMOTE            16  0.974990 0.010333
                 PCA       Random Forest      None            16  0.974606 0.010392
  Lasso (20 features) Logistic Regression      None            20  0.952091 0.013454
         All Features Logistic Regression      None            23  0.952048 0.013542
           RFE Top 15 Logistic Regression      None            15  0.952021 0.013574
  Lasso (20 features) Logistic Regression     SMOTE            20  0.952013 0.012825
           RFE Top 15 Logistic Regression     SMOTE            15  0.951995 0.012964
         All Features Logistic Regression     SMOTE            23  0.951894 0.012900
RF Importance Top 15 Logistic Regression      None            15  0.945648 0.013654
RF Importance Top 15 Logistic Regression     SMOTE            15  0.945645 0.012789
           MI Top 15 Logistic Regression      None            15  0.938623 0.019012
           MI Top 15 Logistic Regression     SMOTE            15  0.938603 0.018484
         All Features       Decision Tree      None            23  0.936226 0.014940
         All Features       Decision Tree     SMOTE            23  0.935492 0.014117
  Lasso (20 features)       Decision Tree      None            20  0.933939 0.015093
  Lasso (20 features)       Decision Tree     SMOTE            20  0.933119 0.015626
RF Importance Top 15       Decision Tree      None            15  0.932072 0.014050
RF Importance Top 15       Decision Tree     SMOTE            15  0.930372 0.013785
          RFE Top 15       Decision Tree      None            15  0.926147 0.015818
          RFE Top 15       Decision Tree     SMOTE            15  0.925873 0.016437
                 PCA Logistic Regression     SMOTE            16  0.924496 0.027099
                 PCA Logistic Regression      None            16  0.924442 0.027841
           MI Top 15       Decision Tree      None            15  0.922044 0.018169
           MI Top 15       Decision Tree     SMOTE            15  0.921337 0.018748
                 PCA       Decision Tree      None            16  0.857944 0.032974
                 PCA       Decision Tree     SMOTE            16  0.857698 0.031111
```

Table 3.16: Master Results Table, consolidating all experiments to directly compare the impact of SMOTE

Final Feature Selection and Refinement using RFECV

Having identified the RF Importance Top 15 feature set as the most promising candidate from our grand comparison, we performed a final refinement step using Recursive Feature Elimination with Cross-Validation (RFECV) to obtain an absolutely optimal subset of features.

RFECV performs backward elimination by iteratively training an estimator model, and in each round, the least important feature is eliminated. Crucially, it uses cross-validation to evaluate the model's performance at every stage, allowing it to determine when the model reaches peak performance with a minimal number of features.

For our implementation, a Logistic Regression model was chosen as the internal estimator to ensure computational efficiency, as RFECV trains many models. The RFECV was configured to perform 5-fold cross-validation (*cv=5*), eliminate one feature at each iteration (*step=1*) and stop at a minimum of five features.

Finally, the RFECV results show that the optimal number of features is 10. Based on the plot, the cross-validated AUC score reaches its peak at 10 features, and including additional features provided no significant improvement. This automated process allowed us to confidently prune 5 features from our candidate set.

The final 10 features selected for all subsequent modeling are:

1. *Online Boarding*
2. *In-flight Wifi Service*
3. *Type of Travel_Personal*
4. *Class_Economy*
5. *Leg Room Service*
6. *Customer Type_Returning*
7. *On-board Service*
8. *Cleanliness*
9. *In-flight Service*
10. *Baggage Handling*

```
[42] X_subset_rfe = X_scaled[features_RF_importance]

     estimator = LogisticRegression(random_state=42)

     # Initialize RFECV
     selector_rfecv = RFECV(estimator=estimator, step=1, cv=5, scoring='roc_auc', min_features_to_select=5, n_jobs=-1)

     print("Fitting RFECV... this can take a few minutes.")
     selector_rfecv.fit(X_subset_rfe, y)
     print("RFECV fitting complete.")

     # Results
     final_optimal_features = X_subset_rfe.columns[selector_rfecv.support_].tolist()
     print("The optimal feature set is:")
     print(final_optimal_features)
     print(f"Number of features: {len(final_optimal_features)}")

     # Performance vs number of features
     plt.figure(figsize=(10, 6))
     plt.xlabel("Number of features selected")
     plt.ylabel("Cross validation score (AUC)")

     mean_scores = selector_rfecv.cv_results_['mean_test_score']
     num_features_tested = range(selector_rfecv.min_features_to_select, len(mean_scores) + selector_rfecv.min_features_to_select)

     plt.plot(num_features_tested, mean_scores)

     # Optimal point marker
     optimal_n_features = selector_rfecv.n_features_
     optimal_score = mean_scores[optimal_n_features - selector_rfecv.min_features_to_select]
     plt.plot(optimal_n_features, optimal_score, 'r*', markersize=10, label=f'Optimal point ({optimal_n_features} features)')

     plt.title("Performance vs Number of Features (RFECV)")
     plt.legend()
     plt.grid()
     plt.show()
```

Code Snippet 3.12: Python code for running RFECV to automatically determine the optimal number of features

```
Fitting RFECV... this can take a few minutes.
RFECV fitting complete.
The optimal feature set is:
['Online Boarding', 'In-flight Wifi Service', 'Type of Travel_Personal', 'Class_Economy', 'Leg Room Service', 'Customer Type_Ret
Number of features: 10
```

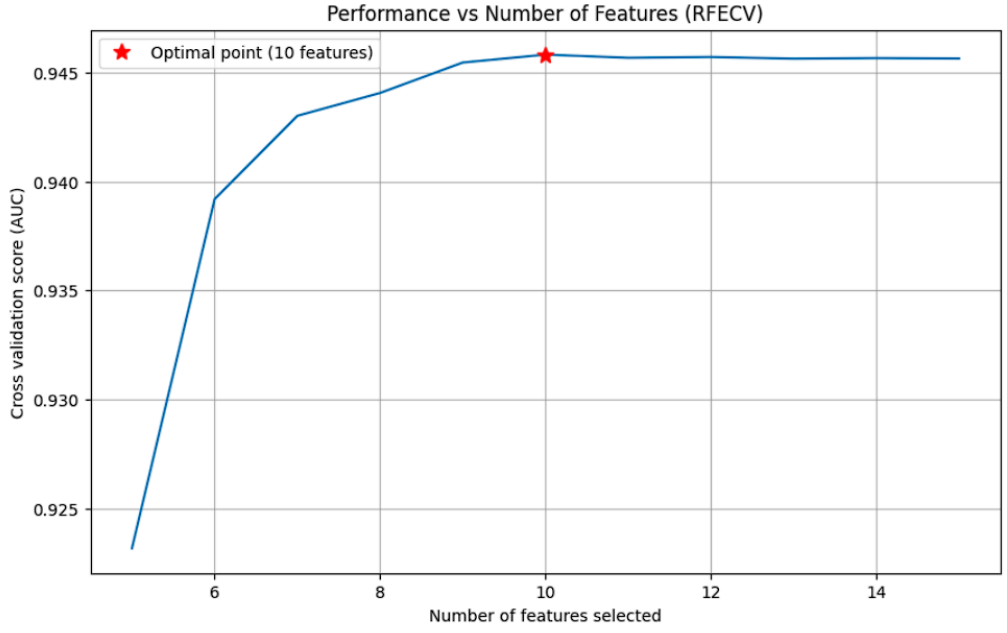


Figure 3.16: RFECV Performance Curve

4 Predictive Modelling

Using the optimal set of 10 predictive features, each team member was responsible for training and evaluating a specific model to determine which algorithm was best suited for solving the business problem.

To ensure a fair and direct comparison between the different algorithms, a standardized evaluation methodology was applied to each model.

The dataset, containing the final 10 features, was split into 80% of train set and 20% of test set. Stratified splitting was used to preserves the proportion of target variable (*Satisfaction_Numeric*) in both sets to prevent model bias. The model is then trained on the 80% train set and evaluated on the 20% unseen test set. Several key metrics were used to evaluate the model performance. First, accuracy is used to measure the overall percentage of correct predictions. Second, ROC AUC score which measures the model's ability to distinguish between satisfied and neutral or dissatisfied passengers. Third, a classification report which provides detailed classification metrics such as precision, recall, and F1-score for each class. Lastly, a confusion matrix plot that we can visualize the number of correct and incorrect predictions for each class.

```
[ ]  # FINAL VALIDATION FUNCTION

    def validate_final_model(feature_set, set_name, model_class, model_name, model_params={}):
        """
        Trains a final, specified model on a given feature set and prints/plots a
        full validation report.

        Args:
            feature_set (list): List of column names to use as features.
            model_class: The classifier class to use (e.g., RandomForestClassifier).
            set_name (str): A descriptive name for the feature set (e.g., "Vital Few").
            model_name (str): A descriptive name for the model (e.g., "Random Forest").
            model_params (dict): Dictionary of parameters to pass to the model.
        """
        full_title = f"{model_name} with {set_name}"
        print("\n" + "="*80)
        print(f"FINAL VALIDATION FOR: {full_title}")
        print("="*80)

        # 1. Prepare Data
        X_final = X_scaled[feature_set]
        y_final = y
        X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size=0.2, random_state=42, stratify=y_final)

        # 2. Initialize the model with any specified parameters
        final_model = model_class(**model_params)
        print(f"\nTraining final {model_name} model...")
        final_model.fit(X_train, y_train)
        print("Training complete.")

        # 3. Make Predictions
        y_pred = final_model.predict(X_test)
        # Check if the model has predict_proba for ROC curve
        if hasattr(final_model, "predict_proba"):
            y_pred_proba = final_model.predict_proba(X_test)[:, 1]
        else:
            y_pred_proba = None

        # 4. Evaluate and Report
        print("\n--- Performance on Hold-Out Test Set ---")
        accuracy = accuracy_score(y_test, y_pred)
        print(f"Accuracy: {accuracy:.4f}")

        if y_pred_proba is not None:
            auc = roc_auc_score(y_test, y_pred_proba)
            print(f"AUC Score: {auc:.4f}")

        print("\nClassification Report:")
        print(classification_report(y_test, y_pred, target_names=['Satisfied', 'Neutral or Dissatisfied']))

        # 5. Plot Visualizations
        # Confusion Matrix
        ConfusionMatrixDisplay.from_estimator(final_model, X_test, y_test, labels=[1, 0],
                                              display_labels=['Neutral or Dissatisfied', 'Satisfied'],
                                              cmap='Blues')
        plt.title(f'Confusion Matrix: {full_title}')
        plt.show()

        # ROC Curve (only if probabilities are available)
        if y_pred_proba is not None:
            RocCurveDisplay.from_predictions(y_test, y_pred_proba, name=full_title)
            plt.plot([0, 1], [0, 1], 'k--', label='No Skill')
            plt.title(f'ROC Curve: {full_title}')
            plt.legend()
            plt.show()
```

Code Snippet 4.1: The standardized model validation function

4.1 Logistic Regression (Laeu Zi-Li)

4.1.1 Introduction to Logistic Regression

One of the chosen algorithms is logistic regression, which is a supervised learning algorithm used for binary classification. Based on the dataset, the possible class of 1 and 0 are either "Neutral or Dissatisfied" and "Satisfied".

4.1.2 Training of untuned model

For the model training, a total of 10 final optimal features were used. The command *max_iter=1000* sets the maximum iterations for the solver to change to 1000 while the *random_state=42* sets the reproducibility seed to 42.

```
validate_final_model(
    feature_set=final_optimal_features,
    set_name="Final Optimal Features (10 Features)",
    model_class=LogisticRegression,
    model_name="Untuned Logistic Regression",
    model_params={'max_iter': 1000, 'random_state': 42}
)
```

Code Snippet 4.2: Execution of the baseline Logistic Regression validation

4.1.3 Model performance

Based on the result, the model achieved an accuracy of 87.83% and an AUC score of 0.9472. That means there are a total of 87.83% of correctly predicted dissatisfactions. Not only that, an AUC score of 0.9472 is higher than random guessing and close to perfect classifier.

```
--- Performance on Hold-Out Test Set ---
Accuracy: 0.8783
AUC Score: 0.9472
```

Table 4.1: Baseline Logistic Regression performance metrics

In addition, the logistic regression algorithm achieved a precision of 0.86 for satisfied and 0.89 for neutral or dissatisfied, having 86% and 89% of correctly predicted positives for both categories. It also has a recall of 0.85 for satisfied and 0.9 for neutral or dissatisfied, showing

that of all the positives, 85% and 90% are correctly predicted. Moreover, this model achieved the f1-score of 0.86 for satisfied and 0.89 for neutral or dissatisfied. The support or number of actual occurrences for satisfied is 11286 while for neutral or dissatisfied is 14690.

Classification Report:

	precision	recall	f1-score	support
Satisfied	0.86	0.85	0.86	11286
Neutral or Dissatisfied	0.89	0.90	0.89	14690
accuracy			0.88	25976
macro avg	0.88	0.88	0.88	25976
weighted avg	0.88	0.88	0.88	25976

Table 4.2: Baseline Logistic Regression classification report

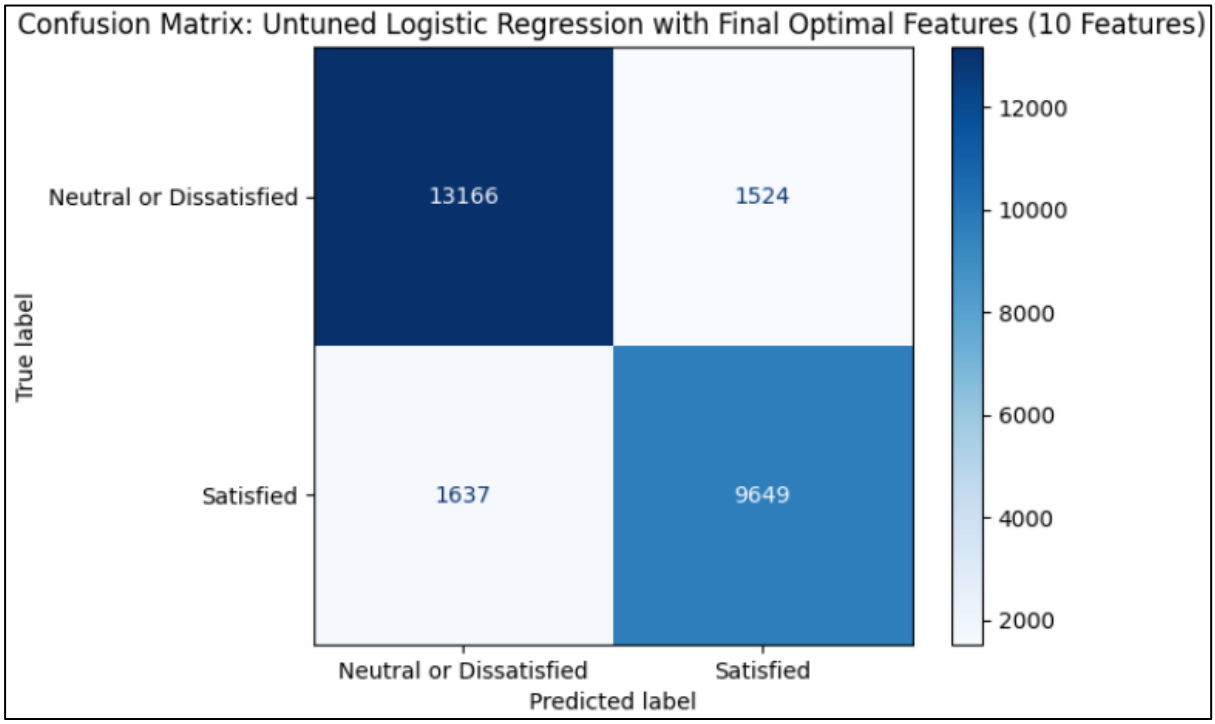


Figure 4.1: Confusion matrix for the baseline Logistic Regression model

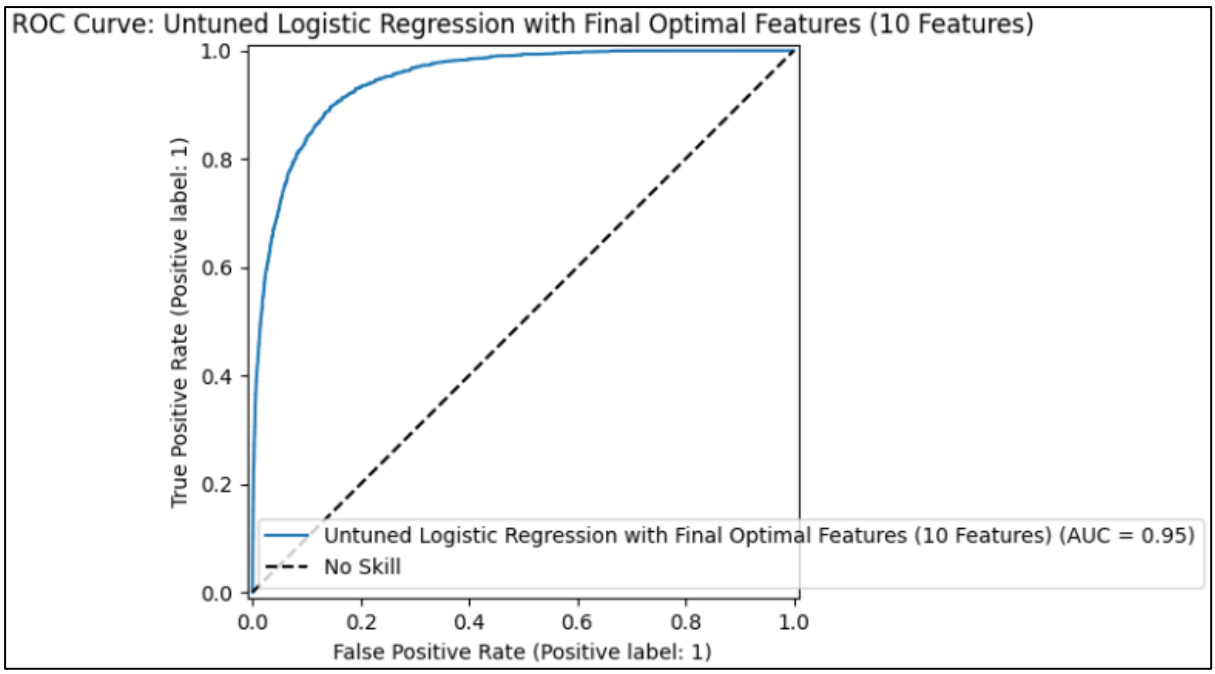


Figure 4.2: ROC Curve for the baseline Logistic Regression model

4.1.4 Fine-tuning model

The goal of fine-tuning for the logistic regression model is to maximise the AUC. For this model, hyperparameter tuning is performed using GridSearchCV. Firstly, the data preparation is carried out by selecting the 10 final optimal features to create the *X_for_tuning_lr* from the *X_scaled* feature set. The *y_for_tuning_lr* remains as target variable.

```
X_for_tuning_lr = X_scaled[final_optimal_features]
y_for_tuning_lr = y
```

Code Snippet 4.3: Data preparation for hyperparameter tuning

Then, the parameter grid, *param_grid_lr* is defined for grid search as the grid identifies the hyperparameters that is going to be tested. Some of them includes regulation strength (C), regularisation type (penalty) and solver.

```
param_grid_lr = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100], # Inverse of regularization strength
    'penalty': ['l1', 'l2'], # Regularization type
    'solver': ['liblinear'] # 'liblinear' supports both l1 and l2
}
```

Code Snippet 4.4: Hyperparameter grid for GridSearchCV

Afterwards, the GridSearchCV is carried out with the maximum iterations of 1000 and a random state of 42. The parameter grid defined before is used along with a 5-fold cross-validation (*cv=5*). Some other settings like scoring matrix set to *roc_auc*, *n_jobs=-1* and *verbose=2* helps provide detailed outputs during the search.

```
grid_search_lr = GridSearchCV(
    estimator=LogisticRegression(max_iter=1000, random_state=42),
    param_grid=param_grid_lr,
    cv=5, # Using 5-fold cross-validation
    scoring='roc_auc', # Optimize for AUC
    n_jobs=-1, # Use all available cores
    verbose=2,
    return_train_score=True
)
```

Code Snippet 4.5: GridSearchCV initialization for Logistic Regression

Now, the grid search can be executed by using the *.fit()* command.

```
print("\nStarting GridSearchCV for Logistic Regression... This will take some time.")
grid_search_lr.fit(X_for_tuning_lr, y_for_tuning_lr)
print("GridSearchCV complete.")
```

Code Snippet 4.6: Execution of the hyperparameter search

When done, the best hyperparameter with the highest AUC score is recorded and stored along with the score achieved.

```
print("\nBest hyperparameters found by GridSearchCV:")
best_params_lr = grid_search_lr.best_params_
print(best_params_lr)
```

Code Snippet 4.7: Get best parameter

```
print(f"\nBest cross-validated AUC score from the search: {grid_search_lr.best_score_:.4f}")
```

Code Snippet 4.8: Display of best parameter

Finally, the tuned model is validated using the *validate_final_model* function. This function trains the model with some fixed settings like maximum iterations being 1000 and random state to 42.

```
validate_final_model(
    feature_set=final_optimal_features,
    set_name="Final Optimal Features (10 Features)",
    model_class=LogisticRegression,
    model_name="Tuned Logistic Regression",
    model_params={**best_params_lr, 'max_iter': 1000, 'random_state': 42}
)
```

Code Snippet 4.9: Execution of the final tuned model validation

4.1.5 Results and summary

After the training of final tuned logistic regression model, it is clear that the accuracy and AUC score does not change, along with the precision, recall, f1-score and support for both category of satisfied and neutral or dissatisfied. However, there are some changes in the confusion matrix by a tiny amount of 1. Therefore, it can be concluded that although logistic regression achieved a high accuracy and performance, it suffers from limited flexibility as is it a linear classifier thus cannot model non-linear data.

```
--- Performance on Hold-Out Test Set ---
Accuracy: 0.8783
AUC Score: 0.9472
```

Table 4.3: Tuned Logistic Regression performance metrics

```
Classification Report:
                       precision    recall  f1-score   support

            Satisfied       0.86      0.85      0.86     11286
Neutral or Dissatisfied       0.89      0.90      0.89     14690

             accuracy                           0.88     25976
            macro avg       0.88      0.88      0.88     25976
         weighted avg       0.88      0.88      0.88     25976
```

Table 4.4: Tuned Logistic Regression classification report

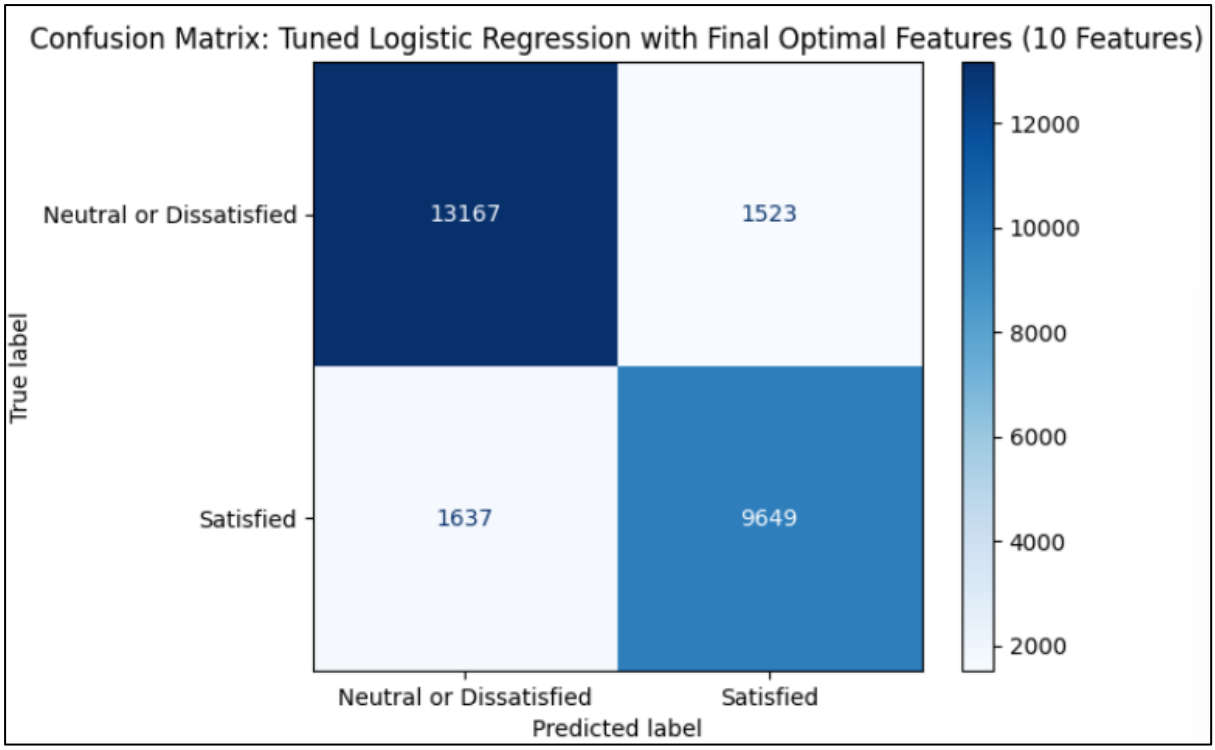


Figure 4.3: Confusion matrix for the tuned Logistic Regression model

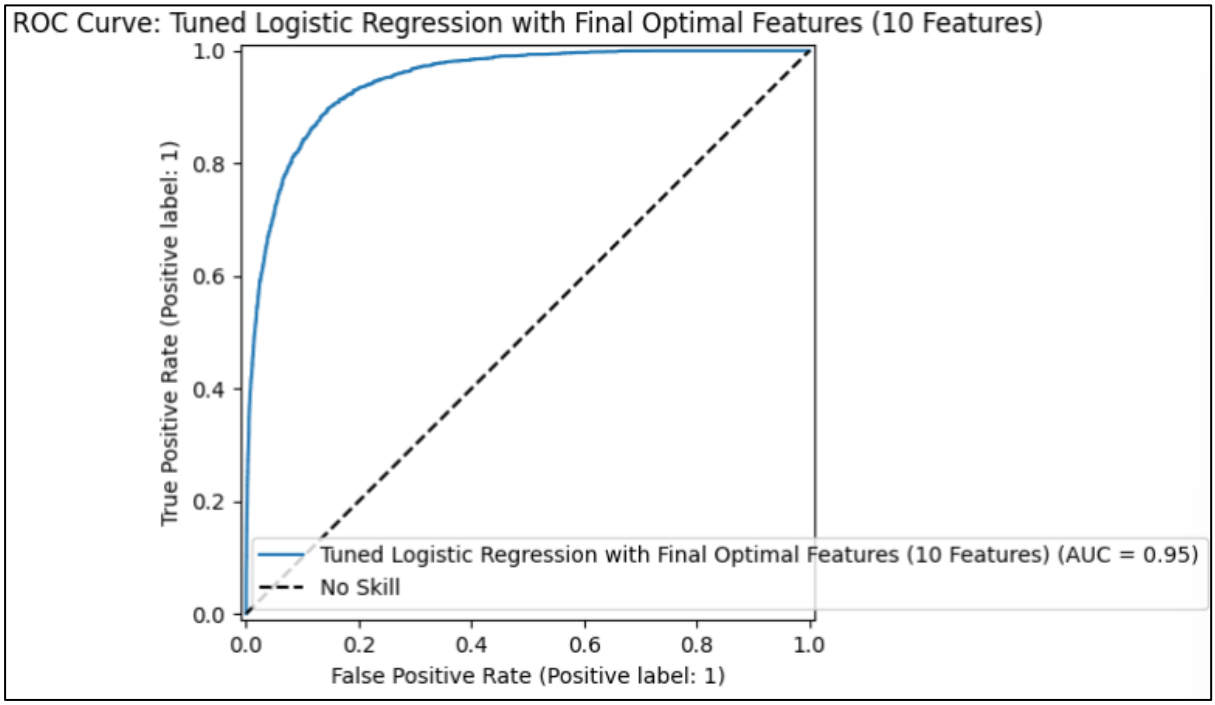


Figure 4.4: ROC Curve for the tuned Logistic Regression model

4.2 Random Forest (Tan Hao Shuan)

4.2.1 Introduction to Random Forest

Random Forest is a machine learning that combine with many decisions tree to make the prediction more accurate and stable. This machine learning works well with both classification and regression tasks. In this project, Random Forest was used for predicting the satisfaction of the passenger. It can be handling large datasets, identified the important features and giving a reliable result without much data preprocessing.

4.2.2 Training of Untuned Model

For this model training, a total of 10 final optimal features were used. In the command, two parameters were used. The *random_state:42* sets the fixed value of 42 that can be ensure that every time run the code and get the same results. The *n_jobs=-1* tells the model to use all available cores on your machines.

```
validate_final_model(feature_set=final_optimal_features,
                     set_name="Final Optimal Features (10 Features)",
                     model_class=RandomForestClassifier,
                     model_name="Random Forest Classifier",
                     model_params={'random_state': 42, 'n_jobs': -1})
```

Code Snippet 4.10: Execution of the baseline Random Forest validation

4.2.3 Model Performance

Based on the result, this model achieved an accuracy of 93.71%, its indicating a high level of correct prediction. Its AUC score of 0.9820 also indicate the strong ability of the model to classify dissatisfied and satisfied passenger.

```
--- Performance on Hold-Out Test Set ---
Accuracy: 0.9371
AUC Score: 0.9820
```

Table 4.5: Baseline Random Forest performance metrics

From the classification report, the model possessed highly high precision and recall both classes. Specifically, it had an unhappy passenger recall of 0.91, which suggests that it was

very good at flagging unhappy passengers. The model was also well-balanced across all metrics, making it a reliable and interpretable choice for airline passenger satisfaction prediction.

Classification Report:	precision	recall	f1-score	support
Satisfied	0.94	0.91	0.93	11286
Neutral or Dissatisfied	0.93	0.96	0.95	14690
accuracy			0.94	25976
macro avg	0.94	0.93	0.94	25976
weighted avg	0.94	0.94	0.94	25976

Table 4.6: Baseline Random Forest classification report

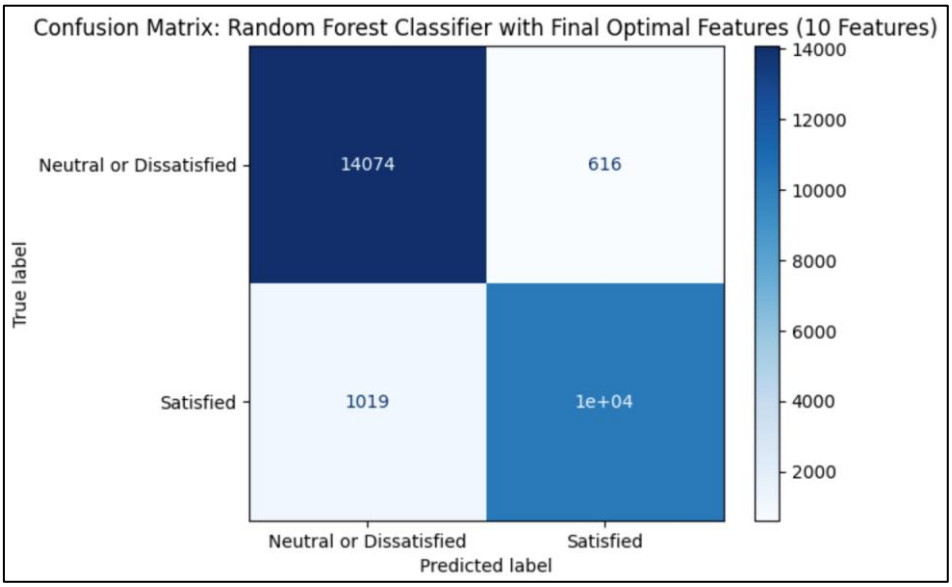


Figure 4.5: Confusion matrix for the baseline Random Forest model

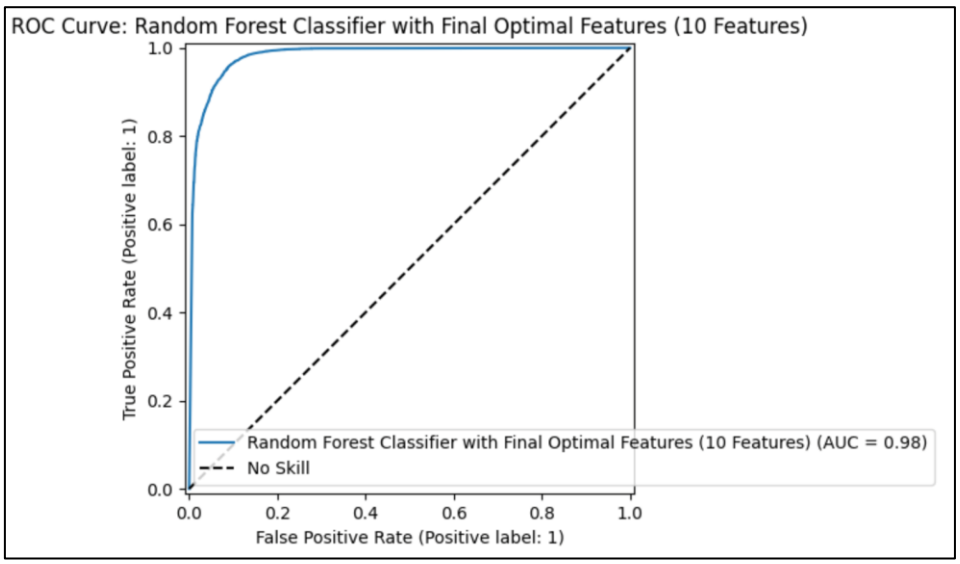


Figure 4.6: ROC Curve for the baseline Random Forest model

4.2.4 Fine-Tuning Model

The param_grid defines the hypermeters that will be utilized to optimize the Random Forest model. It changes the values of the number of trees *'n_estimors'*, the depth of the trees *'max_depth'*, the minimum samples needed to split a node *'min_samples_split'* and the minimum samples needed at a leaf node *'min_samples_leaf'*. By trying all possible combinations of these values, GridSearchCV identifies the most optimal set of model parameters for improved accuracy and generalization.

```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
```

Code Snippet 4.11: Hyperparameter grid for Random Forest GridSearchCV

Before model training, input feature and target variable was defined. *X_for_rf* only contains the 10 best optimal features of scaled data *X_scaled* to make the model work with the most indicative predictors. *Y_for_rf* contains the target variable, which is the passenger satisfaction. They are used for model training and hypermeter tuning.

```
# Split your dataset first
X_for_rf = X_scaled[final_optimal_features]  # Ensure X_scaled and final_optimal_features are defined earlier
y_for_rf = y
```

Code Snippet 4.12: Data preparation for Random Forest tuning

In order to tune the Random Forest Model, GridSearchCV with 5-fold cross-validation was utilized. It used all available combinations of hyperparameters mentioned in *param_grid* to test using accuracy as scoring. The model was trained on selected features *X_for_rf* and the target *y_for_rf*.

```
# Perform tuning with GridSearchCV
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42, n_jobs=-1), param_grid, cv=5, scoring='accuracy')
grid_rf.fit(X_for_rf, y_for_rf)
```

Code Snippet 4.13: Execution of the GridSearchCV hyperparameter search for Random Forest

After the search, the highest performing model was stored in the *best_rf* , and the best set of hyperparameters was retrieved using *grid_rf.best_params_*. The best *cross_validated* score that have been achieved upon tuning was printed for the reference purposes, aiding in the confirmation of model quality.

```
# Best estimator after tuning
best_rf = grid_rf.best_estimator_

print("Best Hyperparameters:", grid_rf.best_params_)

# Report the best score achieved during the search
print(f"\nBest cross-validated AUC score from the search: {grid_rf.best_score_:.4f}")
```

Code Snippet 4.14: Optimal hyperparameters for Random Forest

The best hypermeters were passed to existing *validate_final_model* function. The process re-trains Random Forest with the best setup and checks its performance on the test set using metrics like accuracy, AUC score, and classification report to ensure a consistent and unbiased comparison with other models.

```
# Train and validate using the best parameters
validate_final_model(
    feature_set=final_optimal_features,
    set_name="Final Optimal Features (10 Features)",
    model_class=RandomForestClassifier,
    model_name="Tuned Random Forest",
    model_params={**grid_rf.best_params_, 'random_state': 42, 'n_jobs': -1}
)
```

Code Snippet 4.15: Execution of the final tuned Random Forest validation

4.2.5 Final Performance of Tuned Random Forest Model

After the hyperparameter tuning of Random Forest model with GridSearchCV, the best hyperparameters determined were *n_estimators = 100*, *max_depth = 20*, *min_samples_split = 5*, and *min_samples_leaf = 2*. The cross-validated AUC score during the process of tuning was 0.9380 with these parameters.

After the final tuned model is created, hold-out test set was evaluated on the same using the validation function previously defined. It had 94.09% accuracy and an AUC of 0.9859, indicating fine predictive power. The model was 0.97 and 0.91 when the passengers were dissatisfied and satisfied, respectively, which indicates the model's ability to identify customers

who are susceptible to dissatisfaction. With good precision and class balance measures for both the classes, the tuned Random Forest proved to be a good and robust model for predicting airline passenger satisfaction.

```
================================================================================
FINAL VALIDATION FOR: Tuned Random Forest with Final Optimal Features (10 Features)
================================================================================

Training final Tuned Random Forest model...
Training complete.

--- Performance on Hold-Out Test Set ---
Accuracy: 0.9409
AUC Score: 0.9859

Classification Report:
                          precision    recall  f1-score   support

              Satisfied       0.95      0.91      0.93     11286
Neutral or Dissatisfied       0.93      0.97      0.95     14690

               accuracy                           0.94     25976
              macro avg       0.94      0.94      0.94     25976
           weighted avg       0.94      0.94      0.94     25976
```

Table 4.7: Final performance metrics and classification report for the tuned Random Forest model

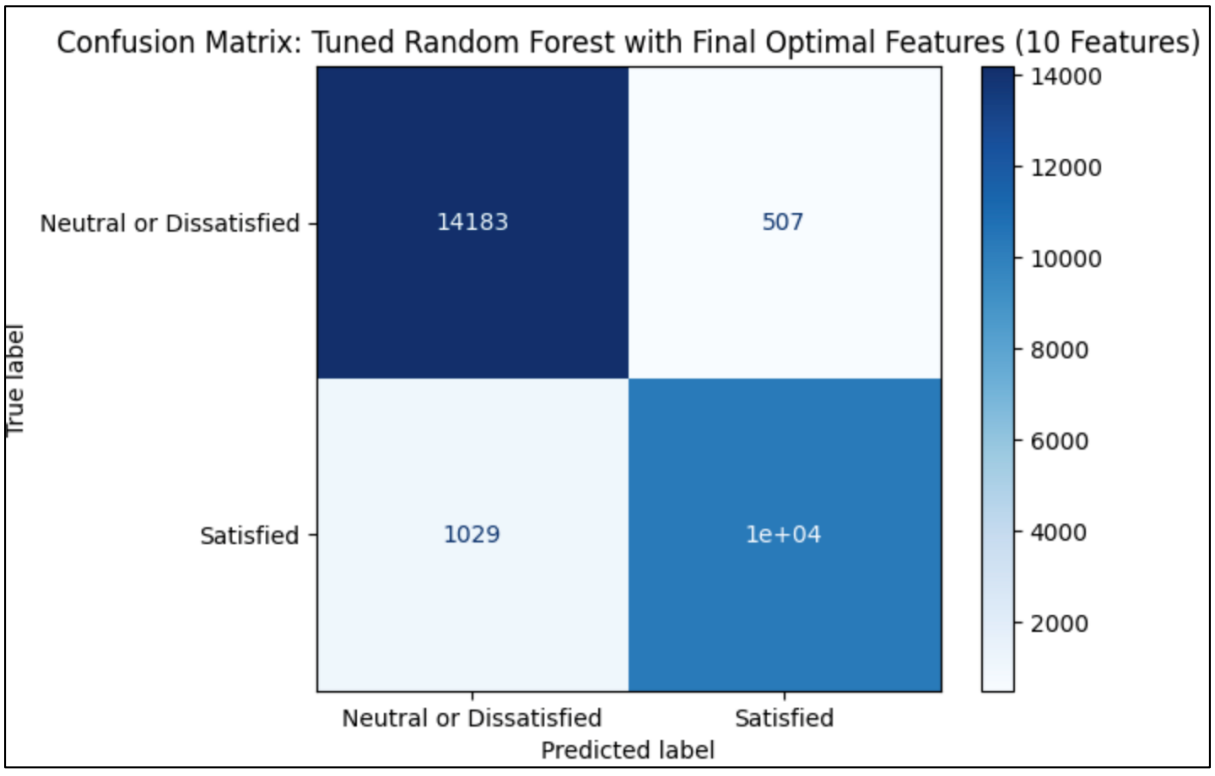


Figure 4.7: Confusion matrix for the tuned Random Forest model

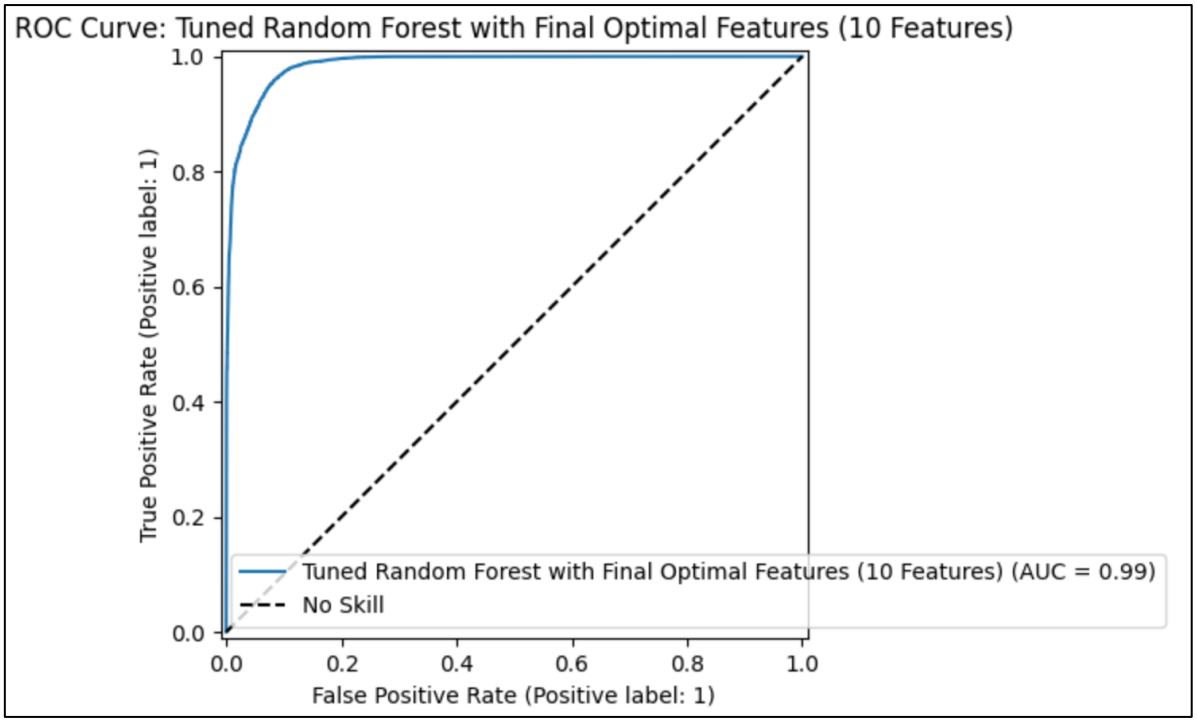


Figure 4.8: ROC Curve for the tuned Random Forest model

4.3 Extreme Gradient Boosting (XGBoost) (Heng Ee Sern)

4.3.1 Introduction to XGBoost

XGBoost (eXtreme Gradient Boosting) is a powerful and widely used machine learning algorithm based on gradient-boosted decision trees. It is renowned for its efficiency, accuracy, and scalability, making it the algorithm of choice for winning machine learning and data mining competitions (Chen & Guestrin, 2016). In this project, XGBoost is employed to capture complex non-linear relationships influencing airline passenger satisfaction, providing a robust and high-performing baseline for comparison against other models.

4.3.2 Baseline Performance Evaluation

Before performing any optimization, we first established a baseline performance evaluation for the XGBoost algorithm using its default parameters. The model was trained using our standardised evaluation methodology.

The results below shows that the XGBoost model performed exceptionally well, yielded an accuracy of 94.18% and AUC score of 0.9868 of the test set. The recall for *Neutral or Dissatisfied* class achieved 97%, meaning it correctly identified 97% of all truly neutral or dissatisfied passengers. The recall for *Satisfied* was slightly lesser at 91%. From the confusion matrix (Figure 4.9), the model produced 508 false negatives and 1003 false positives. The false negatives are the most critical business errors as it incorrectly identifies neutral or dissatisfied passengers, which is so called missing the target.

The ROC Curve (Figure 4.10) further illustrates the model's discriminatory power as the curve is very close to the top-left corner.

This strong initial result confirmed that XGBoost is a suitable algorithm for this problem. The next step would be to tune the model for further optimization, aiming to reduce the Type II error, and potentially improve the recall for *Satisfied* class.

```
[ ]  validate_final_model(
         feature_set=final_optimal_features,
         set_name="Final Optimal Features (10 Features)",
         model_class=xgb.XGBClassifier,
         model_name="XGBoost",
         model_params={'random_state': 42,'eval_metric': 'logloss'}
     )
```

Code Snippet 4.16: Execution of the baseline XGBoost model validation

```
================================================================================
FINAL VALIDATION FOR: XGBoost with Final Optimal Features (10 Features)
================================================================================

Training final XGBoost model...
Training complete.

--- Performance on Hold-Out Test Set ---
Accuracy: 0.9418
AUC Score: 0.9868

Classification Report:
                         precision    recall  f1-score   support

              Satisfied       0.95      0.91      0.93     11286
Neutral or Dissatisfied       0.93      0.97      0.95     14690

               accuracy                           0.94     25976
              macro avg       0.94      0.94      0.94     25976
           weighted avg       0.94      0.94      0.94     25976
```

Table 4.8: Performance metrics and classification report for the baseline XGBoost model

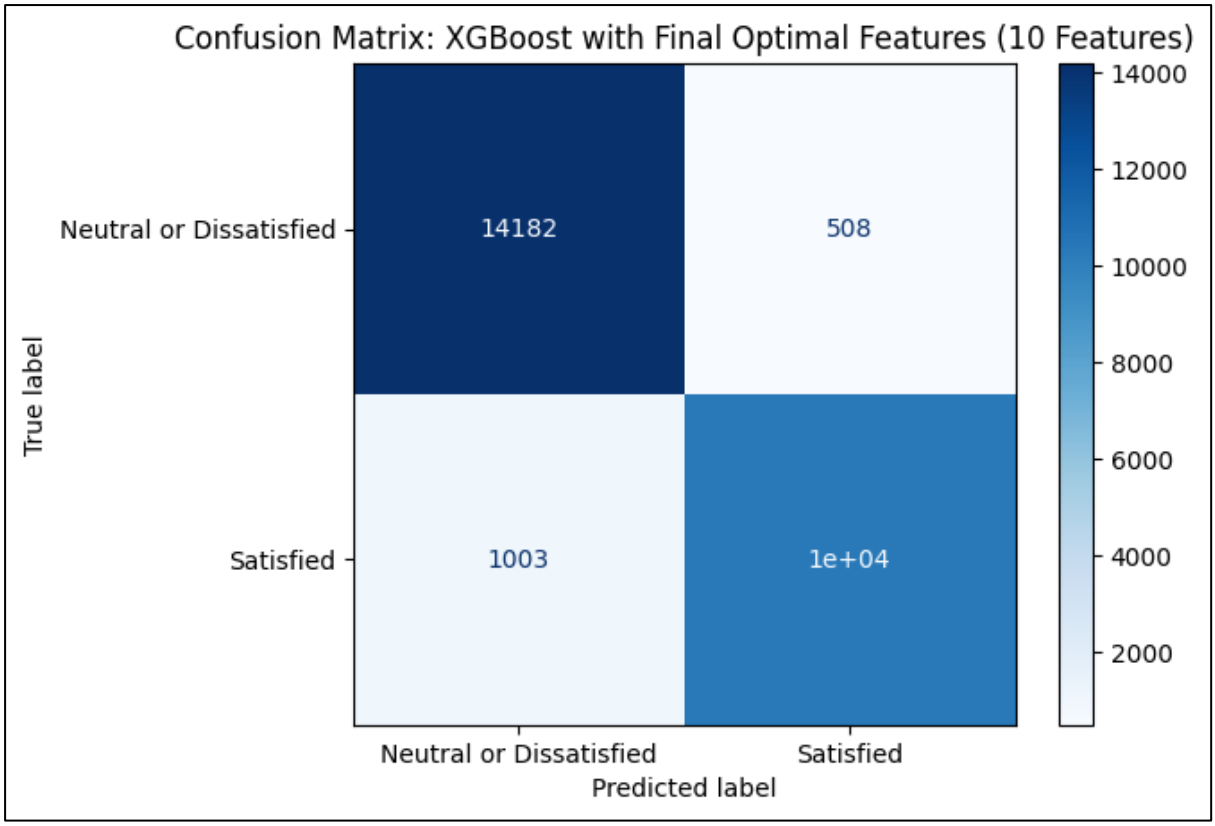


Figure 4.9: Confusion matrix for the baseline XGBoost model

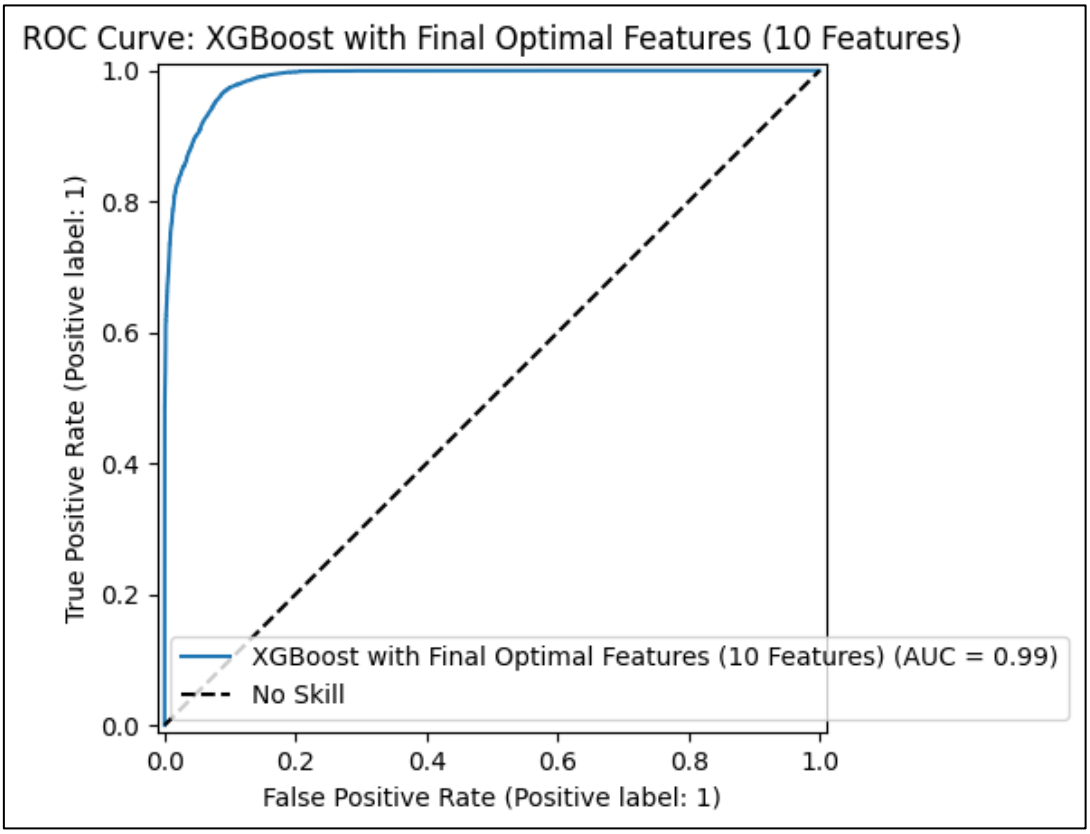


Figure 4.10: ROC Curve for the baseline XGBoost model

4.3.3 Hyperparameter Fine-Tuning with GridSearchCV

While the default settings for XGBoost provides a string result, the performance can often be enhanced by fine-tuning its parameters to the specific characteristics of the dataset. Therefore, Grid Search with Cross-Validation (GridSearchCV) was used to find the best combination of hyperparameters value.

First, a search space was defined to explore the most influencing hyperparameters such as *n_estimators, max_depth, learning_rate, subsample, colsample_bytree.* A 3-fold cross-validation was run for 162 combinations, training a total of 486 models. As a result, *GridSearchCV* identified the best combination that yields the highest cross-validated AUC score.

The best parameters found were:

Table 4.9: Optimal hyperparameters for XGBoost as identified by GridSearchCV

Parameter	Value
colsample_bytree	0.7
learning_rate	0.1
max_depth	7
n_estimator	300
subsample	1.0

The final tuned XGBoost model demonstrates a slight improvement in overall performance. Here is a table for comparison between the performance of default XGBoost model and tuned-XGBoost Model.

Table 4.10: Performance comparison of the baseline vs. tuned XGBoost model

	Default XGBoost		Tuned XGBoost	
Metric	Satisfied	Dissatisfied	Satisfied	Dissatisfied
Precision	0.95	0.93	0.95	0.94
Recall	0.91	0.97	0.91	0.97
F1-Score	0.94	0.95	0.93	0.95
Accuracy	0.9418		0.9425	
AUC Score	0.9868		0.9870	
False Positives	1003		985	
False Negatives	508		509	

As we can see the accuracy of the model increased from 94.18% to 94.25 %, and the AUC score increased from 0.9868 to 0.9870. While this is marginal, this confirms that hyperparameter tuning process really did created a more optimized model.

The primary benefit of this tuning was the reduction in false positives. This came with a negligible cost of one additional false negative. The recall for both classes remained the same.

In short, the fine-tuning process did not dramatically increase the model's metrics. However, it creates a slightly more accurate and balanced classifier by reducing error counts while maintain high performance.

```
# Final Optimal Features for fine-tuning
X_for_tuning = X_scaled[final_optimal_features]
y_for_tuning = y

# Grid of hyperparameters to test
param_grid_xgb = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.3, 0.1, 0.05],
    'subsample': [0.5, 0.7, 1.0],
    'colsample_bytree': [0.7, 1.0]
}

# Initialize GridSearchCV
grid_search_xgb = GridSearchCV(
    estimator=xgb.XGBClassifier(random_state=42,eval_metric='logloss'),
    param_grid=param_grid_xgb,
    cv=3,
    scoring='roc_auc',
    n_jobs=-1, # Parallelization works well for GridSearchCV
    verbose=2,
    return_train_score=True
)

# Run the search
print("\nStarting GridSearchCV for XGBoost... This will take several minutes.")
grid_search_xgb.fit(X_for_tuning, y_for_tuning)
print("GridSearchCV complete.")

# Report the best parameters found
print("\nBest hyperparameters found by GridSearchCV:")
best_params_xgb = grid_search_xgb.best_params_
print(best_params_xgb)

# Report the best score achieved during the search
print(f"\nBest cross-validated AUC score from the search: {grid_search_xgb.best_score_:.4f}")

# FINAL VALIDATION OF THE TUNED XGBOOST MODEL

# Train and validate using the best parameters
validate_final_model(
    feature_set=final_optimal_features,
    set_name="Final Optimal Features (10 Features)",
    model_class=xgb.XGBClassifier,
    model_name="Tuned XGBoost",
    model_params={**best_params_xgb, 'random_state': 42, 'eval_metric': 'logloss'}
)
```

Code Snippet 4.17: Complete code for the GridSearchCV hyperparameter search and final validation

4.3.4 Overfitting Analysis

A potential risk arises for complex and highly tuned models like XGBoost is overfitting, where the model learns the training data too well, including its noise and fails to generalize to new, unseen data. To ensure the model was not overfit, we analysed the performance scores generated during the *GridSearchCV* process.

GridSearchCV provides two scores which is *mean_train_score* and *mean_test_score.* This is useful to check for overfitting of the model. If the gap between these two scores is large indicates the model is overfitted.

The mean training AUC for this model is 0.9915 whereas the mean cross-validation AUC is 0.9861. The calculated overfitting gap is 0.0054 (*mean_train_score – mean_test_score*). This is a very small gap meaning that the model performs almost as well on unseen data as it does on training data. This provides strong evidence that the model fine-tuning process was successful in creating a powerful model that generalizes well and is not overfit. We therefore can be confident in the performance of the model.

```
[ ]    # DIAGNOSING OVERFITTING AFTER GRIDSEARCHCV

    print("\nDiagnosing Overfitting for the Best Tuned XGBoost Model")

    # 1. Get the results from the GridSearchCV object
    cv_results = pd.DataFrame(grid_search_xgb.cv_results_)

    # 2. Find the row corresponding to the best parameters
    best_model_results = cv_results.loc[grid_search_xgb.best_index_]

    # 3. Extract the mean training and validation scores
    mean_train_score = best_model_results['mean_train_score']
    mean_validation_score = best_model_results['mean_test_score']
    std_validation_score = best_model_results['std_test_score']

    # 4. Print and analyze the scores
    print(f"Best Parameters: {grid_search_xgb.best_params_}")
    print(f"Mean Training AUC: {mean_train_score:.4f}")
    print(f"Mean Cross-Validation AUC: {mean_validation_score:.4f} (± {std_validation_score:.4f})")

    # 5. Calculate the gap and check for overfitting
    overfitting_gap = mean_train_score - mean_validation_score
    print(f"Overfitting Gap (Train AUC - CV AUC): {overfitting_gap:.4f}")

    if overfitting_gap > 0.05:
        print("\nWARNING: Potential significant overfitting detected.")
    elif overfitting_gap > 0.02:
        print("\nNote: Mild overfitting may be present, but the gap is likely acceptable.")
    else:
        print("\nModel appears to be well-generalized. No significant overfitting detected.")
```

Code Snippet 4.18: Python code used to extract and compare training vs. cross-validation scores

```
Diagnosing Overfitting for the Best Tuned XGBoost Model ###
Best Parameters: {'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 300, 'subsample': 1.0}
Mean Training AUC: 0.9915
Mean Cross-Validation AUC: 0.9861 (± 0.0032)
Overfitting Gap (Train AUC - CV AUC): 0.0054

Model appears to be well-generalized. No significant overfitting detected.
```

Table 4.11: Overfitting analysis, showing the gap between training and cross-validation AUC scores

4.3.5 Final Performance of the Tuned XGBoost Model

The final, tuned XGBoost model was then configured with the optimal hyperparameters and evaluated on the hold-out test set.

It achieved a final Accuracy of 94.25% and an AUC Score of 0.9870. The tuning process resulted in a slight but measurable improvement over the baseline, primarily by reducing the number of False Negative errors from 1,003 to 985.

This rigorous process of tuning and validation produced a highly accurate and robust XGBoost model, making it a very strong candidate for selection as the final champion model for this project.

```
================================================================================
FINAL VALIDATION FOR: Tuned XGBoost with Final Optimal Features (10 Features)
================================================================================

Training final Tuned XGBoost model...
Training complete.

--- Performance on Hold-Out Test Set ---
Accuracy: 0.9425
AUC Score: 0.9870

Classification Report:
                        precision    recall  f1-score   support

             Satisfied       0.95      0.91      0.93     11286
Neutral or Dissatisfied       0.94      0.97      0.95     14690

              accuracy                           0.94     25976
             macro avg       0.94      0.94      0.94     25976
          weighted avg       0.94      0.94      0.94     25976
```

Table 4.12: Final performance metrics and classification report for the tuned XGBoost model

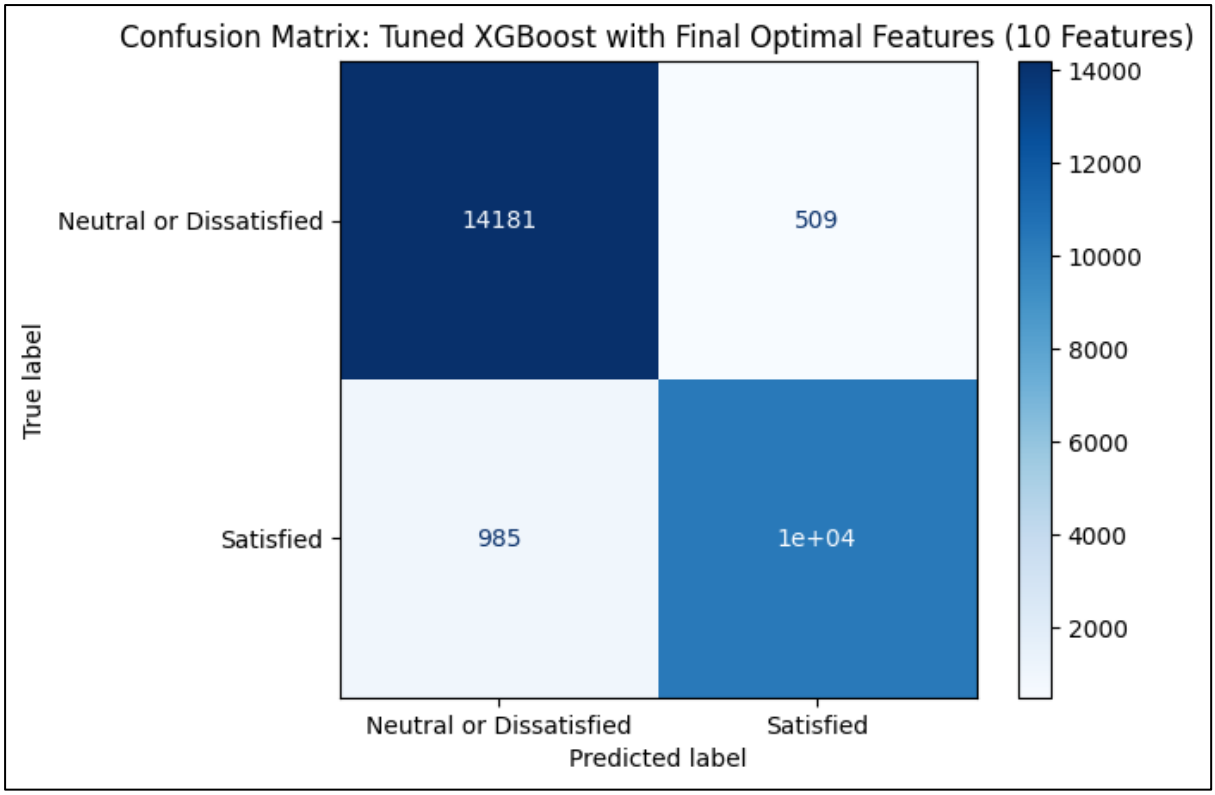


Figure 4.11: Confusion matrix for the final, tuned XGBoost model

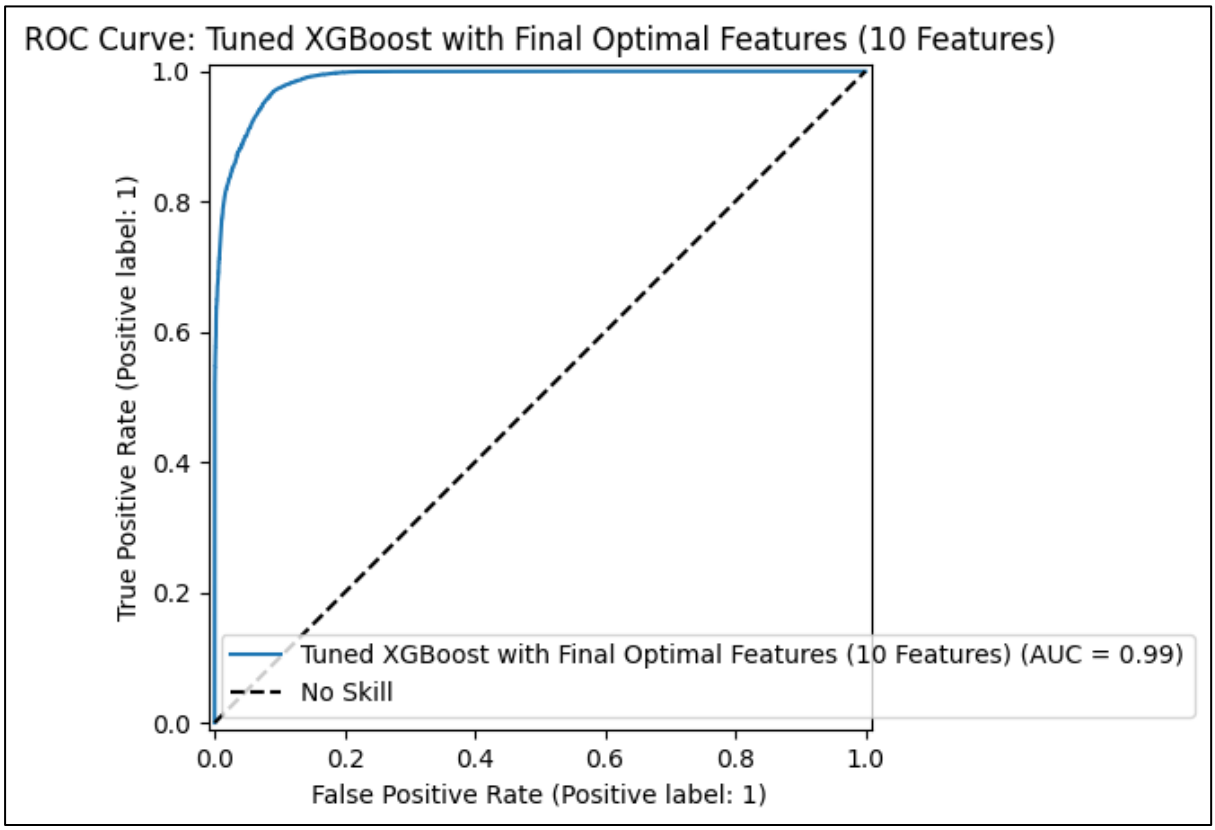


Figure 4.12: ROC Curve for the final, tuned XGBoost model

5 Model Comparison and Final Selection

5.1 Performance Comparison of Models

Following the individual development and optimization of three distinct predictive models, a final head-to-head comparison was conducted to select the single best performing model for deployment. All three models were evaluated and compared using the evaluation methodology introduced earlier.

The key performance metrics for each model are summarized in the table below.

Table 5.1: Final Model Performance Comparison on the Test Set

Metric	Tuned Logistic Regression	Tuned Random Forest	Tuned XGBoost
Accuracy	0.8783	0.9409	0.9425
AUC Score	0.9472	0.9859	0.9870
Recall (Satisfied)	0.85	0.91	0.91
Recall (Dissatisfied)	0.90	0.97	0.97
False Positives	1637	1029	985
False Negative	1523	507	509

5.2 Selecting the Champion Model

As shown in Table 5.1, the tuned XGBoost model is the clear champion. It outperforms the other models in most key metrics. XGBoost is selected for the following reasons:

1. **Superior Overall Performance:** It has the highest accuracy (94.25%) and the highest AUC score (0.9870), indicating its superior ability to classify passengers.
2. **Achieved Business Objective:** The business aim is to correctly identify dissatisfied passengers. The recall for dissatisfied class is at 97%, meaning that it correctly identifies 97% of all dissatisfied passengers.
3. **Minimal Error:** The most critical error is False Negative (Type II), and for XGBoost it is at 509 cases which is considered less. The less critical error, False Positive (Type I) of XGBoost also performs better than the other models.

Even though the tuned Random Forest has a marginally lesser False Negative, the XGBoost model was still chosen because it has significantly lesser False Positives, as we seek for an overall balance for the model.

Based on XGBoost overall performance, it was selected as the final model for this project.

5.3 Model Interpretation with SHAP

After selecting the tuned XGBoost as the final model, the final analytical step is to explain how it makes the predictions. This can be done using one of the explainable AI (XAI) method called SHAP (SHaply Additive exPlanations). It is used to calculate the impacts of each feature and reveals the nature of their influence on the predictions (Keita, 2023).

5.3.1 Global Feature Importance

In order to know the ranking of features that influence the model's predictions, a SHAP summary plot can be constructed.

From the summary plot (Figure 5.1), we can that *Type of Travel_Personal, In-flight Wifi Service,* and *Online Boarding* are the three most impactful features in the model. We can see that *Type of Travel_Personal* has a high feature value that stronly pushes the predictions towards dissatisfaction. Additionally, *Class_Economy* also pushes the predictions towards dissatisfaction. Meanwhile, we can see that *In-flight Wifi Service* has long tails on both sides, indicating that it strongly affects the predictions based on its rating. Returning customers and poor online boarding also pushes the predictions towards dissatisfaction.

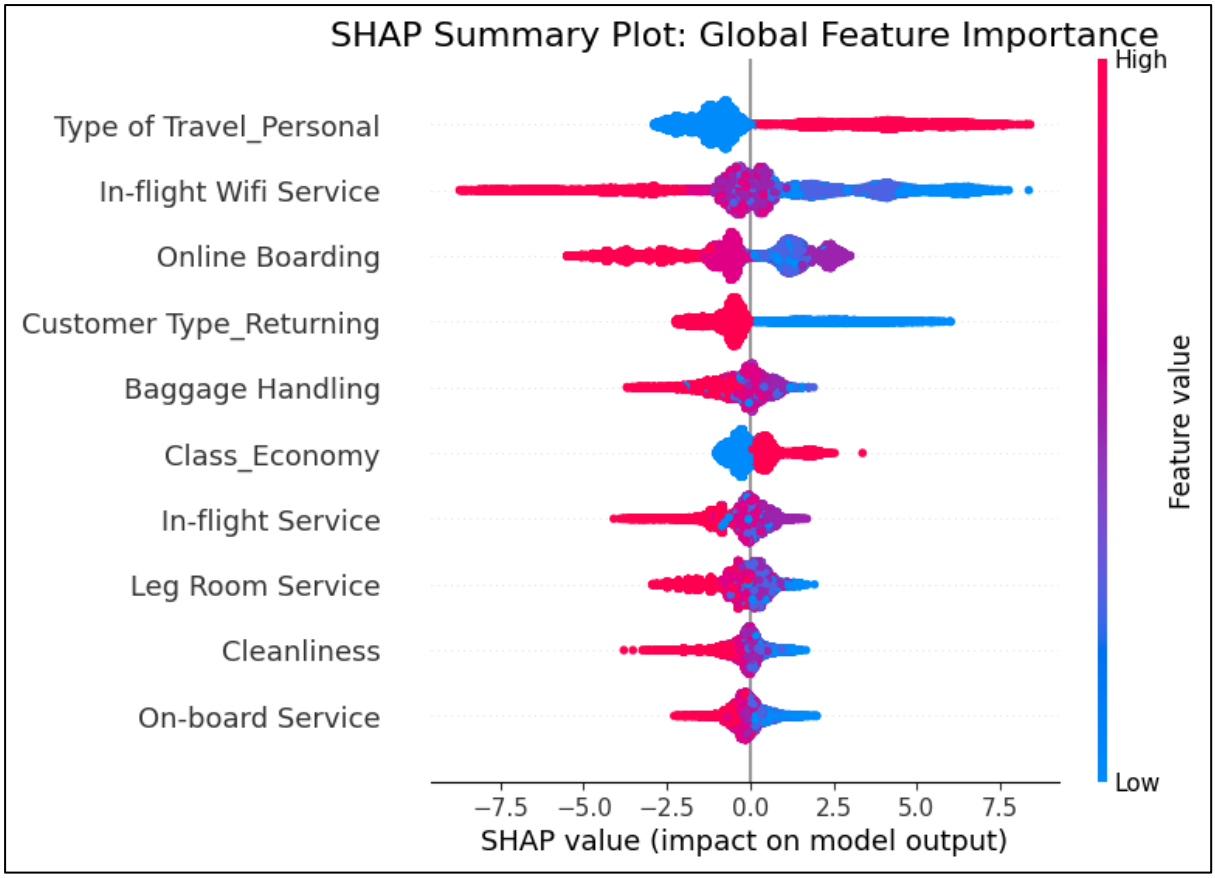


Figure 5.1: SHAP Summary Plot

5.3.2 Explaining Individual Predictions

To demonstrate the model's prediction logic, two SHAP force plot was generated, one for an instance of neutral or dissatisfied passengers, another one for an instance of satisfied passengers.

Neutral or Dissatisfied

From the force plot of a neutral or dissatisfied passengers (Figure 5.2), we can observe that every rating is low. *In-flight Wifi Service, Type of Travel_Personal and Online Boarding* strongly pushes the prediction towards neutral or dissatisfied. This aligns with our findings from the summary plot.

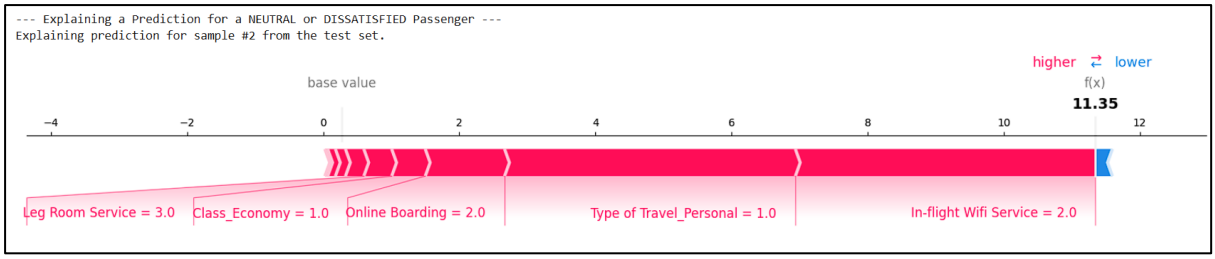


Figure 5.2: SHAP Force Plot explaining an individual prediction for a 'Neutral or Dissatisfied' passenger

Satisfied

From the force plot of a satisfied passenger (Figure 5.3), high online boarding rating, *Type of Travel_Personal*, and returning customer are the key drivers towards satisfaction. Another worth noticing finding is that features such as *Baggage Handling, In-flight Service, and Leg Room Service* push the predictions towards satisfied even though they received lower ratings. By referring to the summary plot earlier, we can observe that there are some blue dots (representing low feature values) appeared on the negative SHAP value of these features, suggesting that even when these services are rated poorly, they can still contribute to satisfaction in certain contexts. This could be due to low expectations in these areas are common, so even slightly poor ratings doesn't strongly affect satisfaction. Lastly, poor *In-flight Wifi Service* pushes towards dissatisfaction as always.

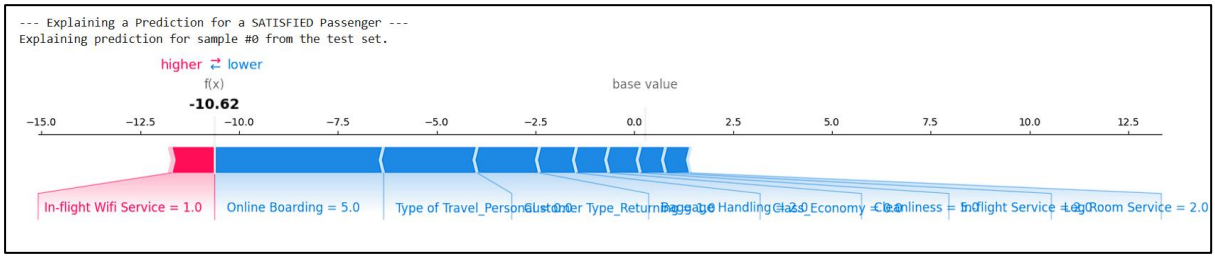


Figure 5.3: SHAP Force Plot explaining an individual prediction for a 'Satisfied' passenger

6 Model Deployment with Streamlit

To transform our descriptive analysis and predictive analysis results into a usable tool, we developed a interactive web application using the Streamlit framework. This web application serves as a dashboard for exploring data insights, generating predictions, and collecting valuable customer feedback.

This application is divided into three main functional tabs, each designed to address a specific business need.

6.1 Tab 1: Interactive Descriptive Analysis

The first tab serves as a business intelligence tool, allowing users to explore the key findings we discovered in EDA. Users get to interact with the charts to investigate the relationship themselves instead of viewing a static report.

There are three sections in this tab. The first one is a list of top 10 most impactful features for predictions. It is a direct display of features, providing immediate insight into what matter the most in passenger airline services.

Figure 6.1: The main interface of the 'Descriptive Analysis & Insights' tab

The second section is the interactive service rating analysis. There is a drop-down menu that allows users to select the key rating features to analyse its impact on the satisfaction rate by displaying a bar chat. Besides that, there is another drop-down menu that users can select

different type of passengers' segments and compare it to satisfaction rate via an intuitive pie chart.

Figure 6.2: Interactive data exploration tools

The third section is a display of actionable business recommendations in text. This is where users can get some suggestions to improve business service.

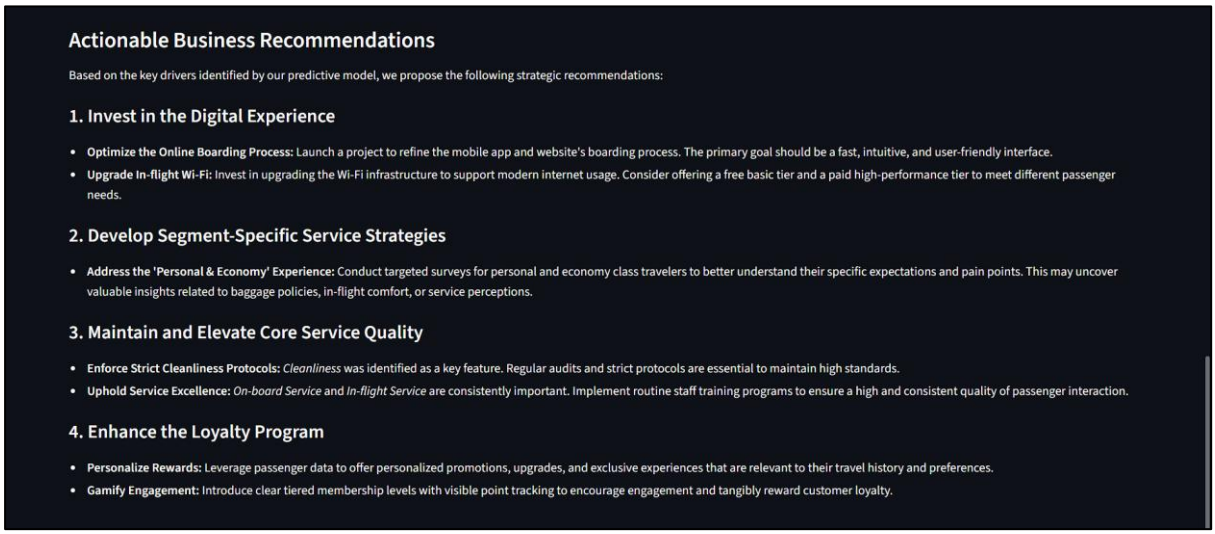


Figure 6.3: A summary of the final, data-driven business recommendations

6.2 Tab 2: Single Passenger Prediction and Feedback System

This tab is the application of our predictive model. The input form contains user-friendly sliders and drops-downs, grouped by categories allowing users to input easily. Upon submission, a

prediction (‘Satisfied’ or ‘Neutral or Dissatisfied’) will be generated using the deployed XGBoost model instantly alongside with the confidence score of the prediction. Users are then asked for optional written feedback, where qualitative data are captured logged to a *feedback_log.csv* file. This feature allows the airline to understand the detailed reason of dissatisfaction, creating valuable dataset for future analysis and model retraining.

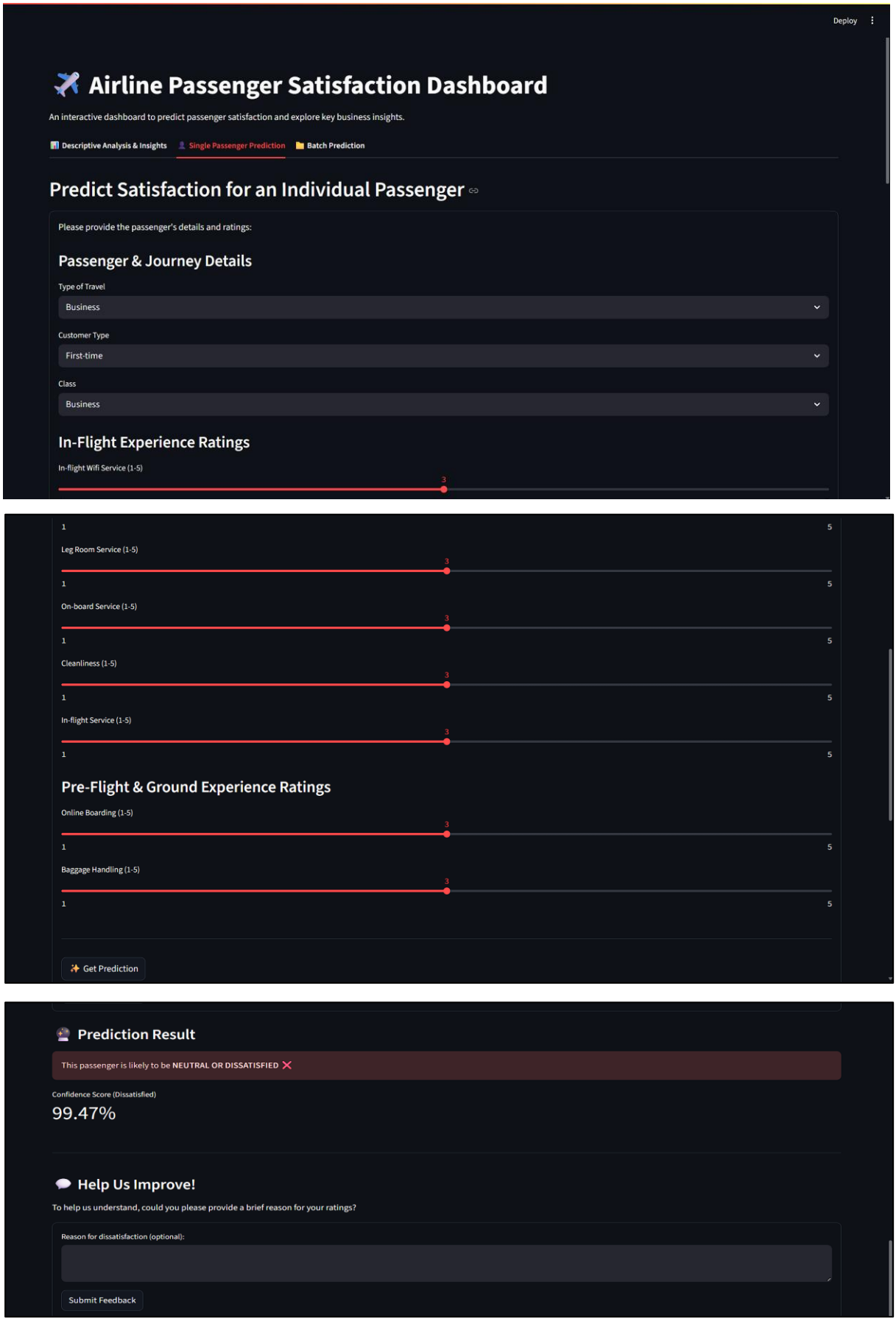


Figure 6.4: The 'Single Passenger Prediction' interface

6.3 Tab 3: Batch Prediction via File Upload

To accommodate larger scale analysis, the third tab provides a batch prediction functionality. The users can download a pre-formatted CSV sample template to cross check that their data and file is in the correct format. Then, the users can upload the CSV file for predictions. The users can preview their uploaded data, and the prediction is generated instantly. The prediction returns the original data appended with two new columns, *Predicted_Satisfaction* and *Probability_Satisfied*. The users can then download the results as a new CSV file.

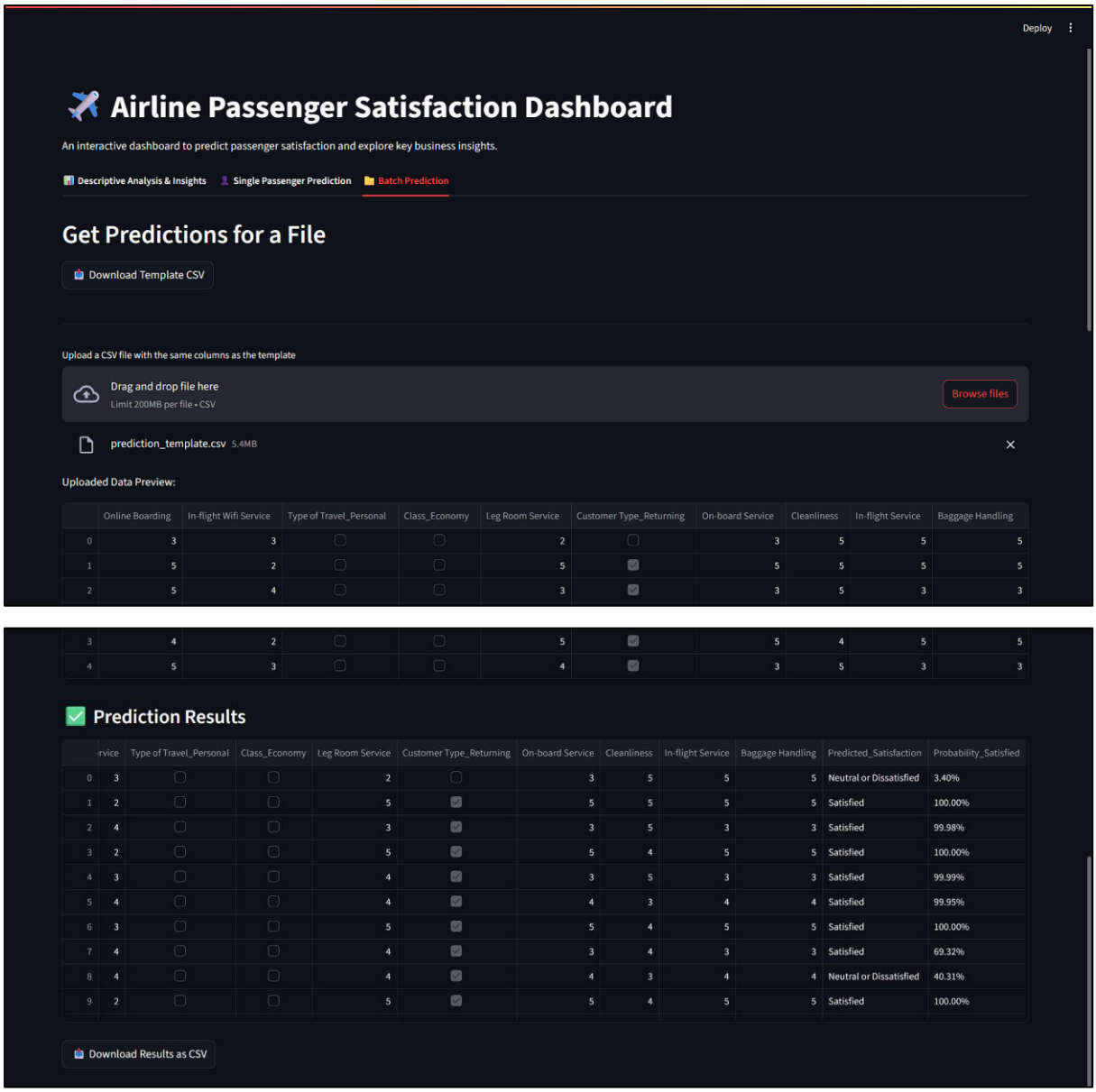


Figure 6.5: The 'Batch Prediction' interface

7 Business Recommendations

7.1 Strategic Insights Derived from the Final Model's 10 Features

The final XGBoost model identified the set of 10 features that as the key drivers of passenger satisfaction. The composition of this feature set provides several key strategic insights into the passenger experience:

1. **Digital Experience is Predominant**

   The consistent high importance of features like *Online Boarding* and *In-flight Wifi Service* indicates that passengers' digital interactions with the airline are no longer secondary, but a primary driver of their overall perception.

2. **Travel Context Defines Expectations**

   The strong predictive power of *Type of Travel* and *Class* demonstrates that a uniform service strategy is insufficient. Business and leisure travellers have distinct satisfaction drivers.

3. **Traditional Aspects Remain Crucial**

   While digital elements are the key, foundational aspects such as *In-flight Service, Leg Room Service, Cleanliness, and On-board Service* remain influential. Failing to meet these basic expectations will still lead to dissatisfaction.

4. **Returning Customers Drives Satisfaction**

   The inclusion of *Customer Type* as the key feature highlights the link between positive past experiences and future satisfaction. This reinforces the value of customer loyalty.

7.2 Actionable Recommendations for the Airline

Based on the data driven insights, we propose the following actionable recommendations:

1. **Invest in Digital Experience**
    - **a.** Launch a project to optimize the mobile app's online boarding process. Making sure that the user interface is intuitive and user friendly.
    - **b.** Upgrade Wi-Fi infrastructure to support modern internet usage. Consider offering a free basic tier and a paid high-performance tier to meet different passenger needs. Ideally, offer free high-performance Wi-Fi where feasible.

2. **Develop Segment Specific-Service Strategies**
    - **a.** Conduct targeted surveys for personal and economy class travellers to better understand their expectations and pain points. This may uncover insights on baggage handling policies.

3. **Maintain High Quality Services**
    - **a.** Enforce strict cleanliness protocols.
    - **b.** Implement routine staff training programs to uphold service quality standards.

4. **Enhance the Loyalty Program**
    - a. Offer personalized promotions, upgrades, and exclusive experiences based on passenger history and preferences.
    - b. Introduce tiered membership levels with point tracking for to encourage engagement and reward loyalty.

8 Conclusion

8.1 Summary of Project Findings and Achievements

This project successfully developed an end-to-end data analytics solution to predict airline passenger satisfaction. Following CRISP-DM methodology, we progress from understanding the business and setting up objectives, to exploratory data analysis and evaluation of multiple feature selection methodologies and machine learning models.

The key achievement was the fine-tuned XGBoost model, with accuracy of 94.25% in classifying satisfied and neutral or dissatisfied passengers. This model is built on an optimal of 10 key features, which were identified through multi-stage feature selection process.

Furthermore, the model was interpreted using SHAP, an Explainable AI technique, and was successfully deployed into a functional Streamlit dashboard. Finally, insights were gained throughout the whole journey and actionable recommendations were given. It meets the business goal of proactively identifying problems and enhance customer flight experience.

8.2 Project Limitations

While this project is successful, it is important to acknowledge its limitations:

- **Static Dataset:** This project is solely based on historical, static dataset. It does not real time operational data or live deployment into the airline's system.
- **Lack of Granular Data:** The trained model lacks more detailed information such as flight routes, aircraft type, ticket pricing, or open-ended text feedback form from passengers.
- **Correlation, not Causation:** Our model only demonstrates correlation between features and passenger satisfaction, they do not imply causation as it needs further studies.

8.3 Recommendations for Future Work

Following are the recommended future works:

- **Analysis of Feedback Data:** The qualitative data collected by the Streamlit dashboard's feedback form can be analyzed using Natural Language Processing (NLP) to get more detailed insights.
- **Model Retraining:** The model should be periodically retrained to ensure it remains accurate and adapt to changing passenger expectations.

8.4 Social and Ethical Considerations

8.4.1 Social Impacts

1. **Improved Passenger Experience:** The airline can make targeted investments to improve overall quality of travel for thousands of passengers, enhancing physical mental well-being.

2. **Accessibility:** Enhancing digital services such as online boarding and Wi-Fi services benefits digital generations and tech savvy travellers but may unintentionally exclude passengers such as elderly who are less comfortable on digital platform.

3. **Customer Empowerment:** Feedback mechanism in the dashboard allows passenger to give constructive feedback. By listening to customers' voice and act accordingly, the airline will excel in many aspects.

8.4.2 Ethical Issues

1. **Data Privacy:** The data used for the analysis must comply with data privacy regulations. The airline should follow industry best practice to ensure data privacy and security.

2. **Fairness in Loyalty Programs:** The model identified returning customers as a key feature. It would be unethical to over-reward frequent flyers and raise the bar to fly for first time customers.

In conclusion, a strong, human-centered governance framework is not just recommended, but essential for the responsible and successful implementation of this data analytics solution.

9 References

Ali, Barakat. (2022). The Effect of Firm's Brand Reputation on Customer Loyalty and Customer Word of Mouth: The Mediating Role of Customer Satisfaction and Customer Trust. International Business Research. 15. 30-30. https://doi.org/10.5539/ibr.v15n7p30

Batarlienė, N., & Slavinskaitė, N. (2023). Assessment of Factors Determining Airline Consumer Loyalty: Case Study in Lithuania. Sustainability, 15(2), 1320. https://doi.org/10.3390/su15021320

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794. https://doi.org/10.1145/2939672.2939785

Dike, S. E., Davis, Z., Abrahams, A., Anjomshoae, A., & Ractham, P. (2023). Evaluation of passengers' expectations and satisfaction in the airline industry: an empirical performance analysis of online reviews. *Benchmarking an International Journal*, *31*(2), 611–639. https://doi.org/10.1108/bij-09-2021-0563

Hotz, N. (2024, December 9). *What is CRISP DM?* Data Science PM. https://www.datascience-pm.com/crisp-dm-2/

IBM SPSS Modeler Subscription. (2021, August 17). https://www.ibm.com/docs/pt-br/spss-modeler/saas?topic=modeling-overview

Keita, Z. (2023, May 10). Explainable AI - Understanding and Trusting Machine Learning Models. https://www.datacamp.com/tutorial/explainable-ai-understanding-and-trusting-machine-learning-models

Kiviak, R. (2023). *Key performance indicator (KPI) | EBSCO*. EBSCO Information Services, Inc. | www.ebsco.com. https://www.ebsco.com/research-starters/business-and-management/key-performance-indicator-kpi

Sheldon, R., Tucci, L., & Roy, M. (2024, May 9). *strategic management*. Search CIO. https://www.techtarget.com/searchcio/definition/strategic-management

Shiwakoti, N., Hu, Q., Pang, M. K., Cheung, T. M., Xu, Z., & Jiang, H. (2022). Passengers' Perceptions and Satisfaction with Digital Technology Adopted by Airlines during COVID-19 Pandemic. *Future Transportation*, *2*(4), 988–1009. https://doi.org/10.3390/futuretransp2040055

Smart Vision Europe. (2020, June 17). *Crisp DM methodology - Smart Vision Europe*. https://www.sv-europe.com/crisp-dm-methodology/#one

Tahanisaz, S., & Shokuhyar, S. (2020). Evaluation of passenger satisfaction with service quality: A consecutive method applied to the airline industry. *Journal of Air Transport Management*, *83*, 101764. https://doi.org/10.1016/j.jairtraman.2020.101764

10 Appendix

10.1 Appendix A: Workload Matrix

Name	Task / Description / Responsibility
HENG EE SERN	- Team Leader - Proposal Writing - Exploratory Data Analysis - Feature Selection - XGBoost - Streamlit Dashboard - Model Interpretation with SHAP
LAEU ZI-LI	- Business domain - Proposal Writing - Data Analytics Methodology - Initial Data Exploration - Logistic Regression - Business Recommendations
TAN HAO SHUAN	- Background Information - Proposal Writing - Random Forest Classifier - Model Comparison and Final Selection - Conclusion

10.2 Appendix B: Personal Reflection Report

Name: HENG EE SERN

TP Number: TP081786

1. **My Contributions**

   As the group leader, I led the team along the journey of this project by following the CRISP-DM methodology. I participated in every part for confirmation. My main individual role was to develop the XGBoost model. This included fine-tuning using GridSearchCV and performing a detailed performance evaluation alongside an overfitting analysis. I also suggested the use of SHAP for the final interpretation and was a primary developer of the final Streamlit dashboard.

2. **What I learned**

   I learned how to manage a data analytics project from start to finish, from cleaning the data to deploying a working application. A key technical skill I gained is the use of SHAP. It taught me how to look at the plot and explain it, which is a very powerful tool. In addition, the feature selection process using different methods strengthened my data analysis skills.

3. **Challenges I Faced**

   The biggest technical challenge I faced is writing the code. Python for data analysis is a skill that needs to be delved into and learned.

4. **Conclusion**

   Overall, this project was a valuable experience that gave me new knowledge, taking me from a data analysis amateur to a more experienced practitioner. I now have a much better understanding of the holistic process of building and explaining a machine learning model. These skills have laid a strong foundation that I look forward to using in future projects.

Name: Laeu Zi-Li

TP Number: TP083604

1. My Contributions

For the group component, I first identified our business domain, which is strategic management. Afterwards, I help explain all the phased of the CRISP-DM methodology that we plan to use throughout the entire project. I then interpret the initial data inspection steps from data sourcing to data transformation. For the individual part, my task was to train my own model with the selected features. My algorithm used is logistic regression. After training I also fine-tune the model to compare and make conclusions.

2. What I learned

Throughout this whole project, I have learned the importance of precision especially within the project. From choosing business domain, selecting dataset to training and comparing model, every step is expected to be perfect as a slight mistake can cause misleading to the entire project.

3. Challenges I Faced

One of the largest challenges I faced is the need to adapt towards a rather new programming and coding tasks. As we do not have enough experience and skills required, we tend to struggle when it comes to the technical parts.

4. Conclusion

As conclusion, this project is one of the most exciting at the same time challenging tasks I was ever assigned to as it was my first time being exposed to data analytics. I am looking forward to deeper knowledges that are related to these fields such as AI development.

Name: Tan Hao Shuan

TP Number: TP080852

1. My Contributions

For the group activity, I focused primarily on the technical execution. I helped to perform the early EDA (Exploratory Data Analysis) and feature selection process with statistical methods like mutual information and chi-squared tests. I spearheaded the installation and hyperparameter tuning of the Random Forest model using GridSearchCV and applied our shared validation function to ensure uniform scoring.

2. What I learned

This project made me realize the significance of structure and evaluation in data analytics. I was able to realize how to select the best machine learning model based on comparing multiple performance metrics like AUC and accuracy. I also received hands-on experience in hyperparameter tuning and model verification, and this enriched my understanding of how predictive modelling is actually executed in the real world.

3. Challenges I Faced

One of the greatest challenges I faced was understanding and applying the model validation and tuning process accurately, especially when combining it with the pre-specified function of the team. I also had to correct several mistakes related to data shaping and reformatting during model training, which consumed a lot of time and needed extensive testing to correct.

4. Conclusion

Overall, this project was tough but great. It was my first attempt at building an end-to-end machine learning pipeline from data cleaning and exploration to model comparison and interpretation. It made me feel more confident in coding using Python and using such tools as Scikit-learn. I am better prepared for future machine learning-related, data science-related, and model deployment-related tasks.