Hotel Booking Demand

Minho Cho

CS 5525 Data Analytics

Department of Computer Science
Virginia Tech
United States

December 10th, 2022

Contents

List of Figures

List of Tables

Abstract

There are three major parts when it comes to traveling. What to do, where to sleep, and what to eat. With the ever-growing demand for travel, booking accommodations is a key part of making travel plans. This paper attempts to answer questions with a data analytical approach to make observations and predictions for how and where potential hotel business owners should focus to prevent cancellations from customers in order to maximize their profits.

1 Introduction

1.1 Opening

When traveling, there are three essential things one must consider. The three are what they will be doing, what they will be eating, and finally, where they will be staying. This paper will be focusing on the last aspect, where they will be staying. It is also known as choosing one's accommodation. Among the different types of accommodations available to travelers such as hotel, motel, Airbnb, hostels, etc., hotels are still one of the most commonly booked type of accommodation. Through the analysis of this hotel booking demand dataset, we will observe and make predictions through multiple analysis and classifiers to determine the characteristics of hotel guests and provide outcomes on how potential hotel business owners can benefit through this analysis. Here are some of the potential questions we will be answering through this paper:

- What kind of hotel do guests prefer?
- Which month had the most guest arriving?
- When is the optimal time to book rooms?
- Is not providing certain meals a deal breaker for guests?
- Are guests who have made individual bookings more likely to cancel?
- What are the most influential cancellation indicators?

Through the analysis and observation of answering questions as mentioned above, we will ultimately attempt in predicting if a guest will cancel their booking based on their preferences and choices they have made on their booking.

1.2 Video Presentation

A video presentation of the final term project can be found at Hotel Booking Demand Presentation by Minho Cho via YouTube. The results presented in the presentation are subject to change as modifications have been made in the process of writing this report.

1.3 Motivation

Before we get into the analysis of the dataset, I wanted to share my motivation in choosing this topic. Ever since I have graduated high school, I have been a big traveler and I always wondered how the travel industry worked behind the scenes. As mentioned above, I believe there are three things people should consider when planning trips and I put my priority on choosing the right accommodation as my plans revolve around my accommodation. As I've been traveling for a little shy of 10 years, I've noticed the importance of this part in the travel industry, the business part of accommodations, and decided to look into this with the excuse of being it my final term project topic.

So without further ado, let's take a look at the dataset before jumping into data preprocessing.

2 Dataset Description

The **Hotel Demand Booking** dataset was obtained from the following website, Hotel Booking Demand Kaggle Page.

The dataset provided on Kaggle was taken from the Hotel Booking Demand Datasets article written by **Nuno Antonio**, **Ana Almeida**, and **Luis Nunes** for **Data in Brief, Volume 22, Februrary 2019**.

The initial dataset was later cleaned by **Thomas Mock** and **Antoine Bichat** on **#tidytuesday** in 2020.

Through the extraction and cleanup of the dataset, information regarding personal identification has been removed from the dataset.

In the Hotel Demand Booking dataset, there are a total of **32** columns.

15 of them are **categorical features** while **17** of them are **numerical features**.

Data within the dataset consists of hotel booking information for city and resort hotels with various features associated to them.

No information regarding personal traits/biases are included in the dataset.

Below is a table that goes through each column of the dataset along with a brief explanation of what the values in the columns represent.

Number	Data Column Name	Data Column Description
1	hotel	Describes if the type of the hotel is either a city hotel or resort hotel
2	is_canceled	Describes if the booking was not canceled (0) or canceled (1)
3	lead_time	Describes the number of days that have passed between the entering date of the booking into the PMS and the arrival date
4	arrival_date_year	Describes the year of arrival date
5	arrival_date_month	Describes the month of arrival date
6	arrival_date_week_number	Describes the week number of year for arrival date
7	arrival_date_day_of_month	Describes the day of arrival date
8	stays_in_weekend_nights	Describes the number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
9	stays_in_week_nights	Describes the number of weeknights (Monday to Friday) the guest stayed or booked to stay at the hotel
10	adults	Describes the number of adults
11	children	Describes the number of children
12	babies	Describes the number of babies
13	meal	Describes the type of meal booked. Categories are presented in standard hospitality meal packages: Undefined/SC – no meal package; BB - Bed & Breakfast; HB - Half board (breakfast and one other meal – typically dinner); FB - Full board (breakfast, lunch, and dinner)
14	country	Describes the country of origin of the guest Categories are represented in the ISO 3155–3:2013 format
15	market_segment	Describes the market segment designation Categories are presented as the following: TA - Travel Agents TO - Tour Operators
16	distribution_channel	Describes the booking distribution channel Categories are presented as the following: TA - Travel Agents TO - Tour Operators
17	is_repeated_guest	Describes the value indicating if the booking name was from a newcomer (0) or repeated guest (1)
18	previous_cancellations	Describes the number of previous bookings that was canceled by the customer prior to the current booking
19	previous_bookings_not_canceled	Describes the number of previous bookings not canceled by the customer prior to the current booking
20	reserved_room_type	Describes the code of room type reserved
21	assigned_room_type	Describes the code for the type of room assigned to the booking Due to hotel operation reasons or customer requests, the assigned room type may differ from the reserved room type
22	booking_changes	Describes the number of changes/amendments made to the booking between the booking being entered on the PMS and until check-in or cancellation
23	deposit_type	Describes the indication of if the customer made a deposit to guarantee the booking. Categories are presented as the following: No Deposit – no deposit was made; Non Refund – a deposit was made in the value of the total stay cost; Refundable – a deposit was made with a value under the total cost of stay
24	agent	Describes the ID of the travel agency that made the booking
25	company	Describes the ID of the entity that made/paid the booking
26	days_in_waiting_list	Describes the number of days the booking was on the waiting list before confirmation
27	customer_type	Describes the type of booking Categories are presented as the following: Contract - when the booking has an allotment or other type of contract associated with it; Group – when the booking is associated with a group; Transient – when the booking is not part of a group or contract and is not associated with another transient booking; Transient-party – when the booking is transient but is associated with at least another transient booking
28	adr	Describes the Average Daily Rate (ADR) as defined by dividing the sum of all lodging transactions by the total number of nights staying
29	required_car_parking_spaces	Describes the number of car parking spaces required by the customer
30	total_of_special_requests	Describes the number of special requests made by the customer
31	reservation_status	Describes the latest reservation status Categories are presented as the following: Canceled – booking was canceled by the customer; Check-Out – customer has checked in but already departed; No-Show – customer did not check-in and did inform the hotel
32	reservation_status_date	Describes the date at which the last status was set

Table 1: Dataset Feature Explanation Table

As the focus of this paper is to predict if a guest(s) will cancel their booking based on their preferences and selections they have made on their booking, the **dependent variable** was chosen to be the **is_canceled** feature. The **independent variable(s)** were chosen through the process of exploratory data analysis (EDA) and regression analysis which the next parts of the paper will discuss.

3 Phase I, Exploratory Data Analysis

This section presents the exploratory data analysis (EDA) of the dataset. Before we dig into any data preprocessing, we will take a look at the correlation matrix of the initial data.

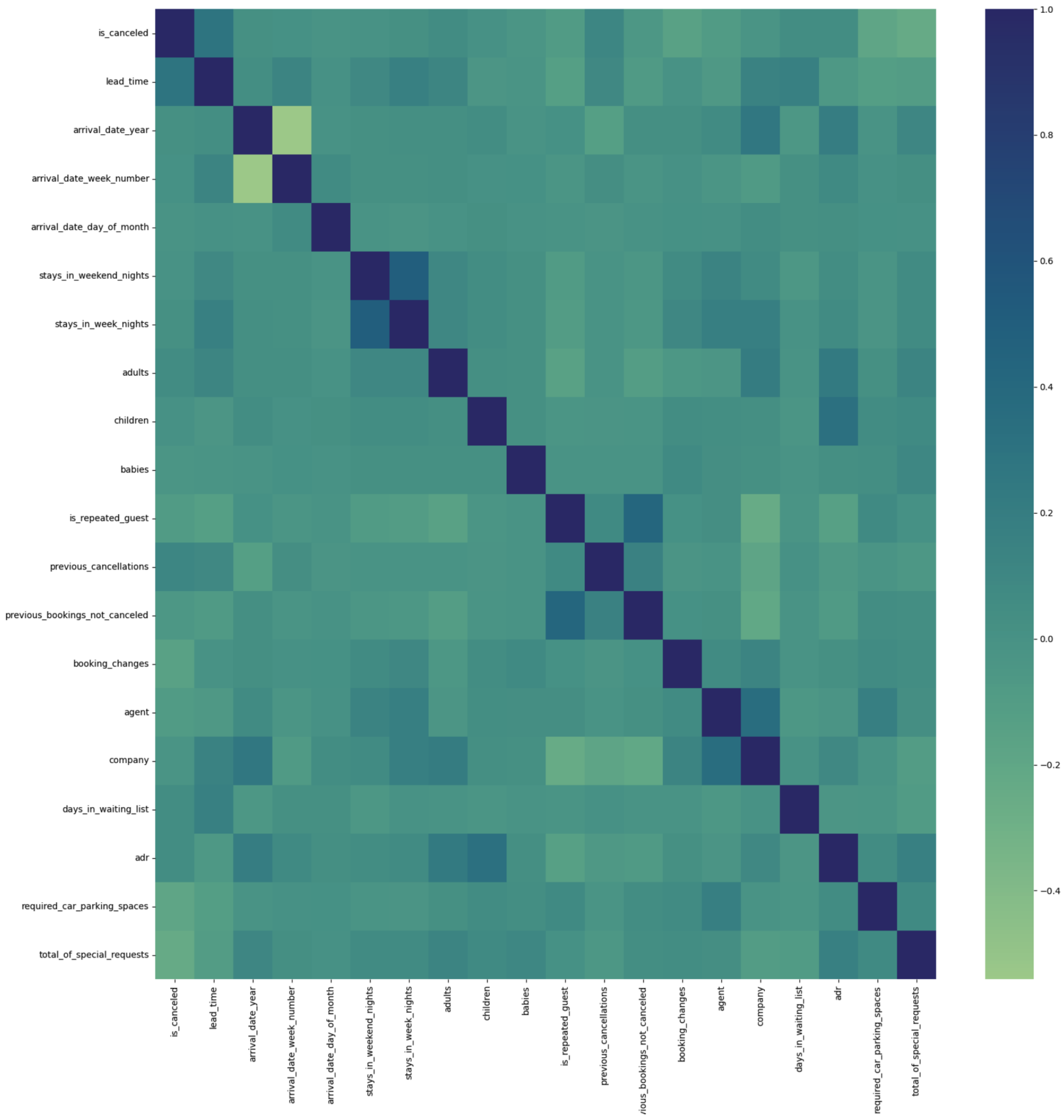


Figure 1: Initial Correlation Matrix

We can observe several things from the heatmap presented above. The number of nights stayed on a weekday and weekend seems to have a high correlation. I assume this correlation is high as when people book hotels, they usually book by the night and do not consider whether it is a weekday or a day on the weekend. There also seems to be a higher correlation between the lead time and cancellation. In addition, there seems to be a high correlation between guests who made repeated bookings and those who have not canceled their previous bookings. This could be interpreted in a way that the guest was satisfied with their previous stay which made the chance of them canceling their booking lower.
With this initial observation in mind, we will move on to data preprocessing.

3.1 Data Preprocessing

This section presents the data preprocessing steps done to the dataset.

- Data Cleaning
- Data Observations
- Dimensionality Reduction
- Feature Selection
- Variable Transformation

Afterward, the sample covariance matrix and the sample covariance matrix with the preprocessed data were obtained to check whether the preprocessed data is suitable for regression analysis and classification analysis.

3.1.1 Data Cleaning

The first step is to check if there are any missing values in the dataset. By running the appropriate command, we get the following,

```
children       4
country      488
agent      16340
company   112593
```

With this, we will get replace the missing values with appropriate values so that we can fully utilize the dataset. The following was performed to the missing values:

- Missing children values replaced with **0**
- Missing country values replaced with **Unknown**
- Transform agent ids (numerical values) to **agent** (categorical value)

- Replace NaN agent values with **No agent** (categorical value)
- Set company column value to **Corporate** if the values in their market segment and distribution channel column are **Corporate**
- Replace non-null values in company column to **Corporate**
- Replace remaining and missing values in the company column with **Individual**

The following were also performed on the dataset:

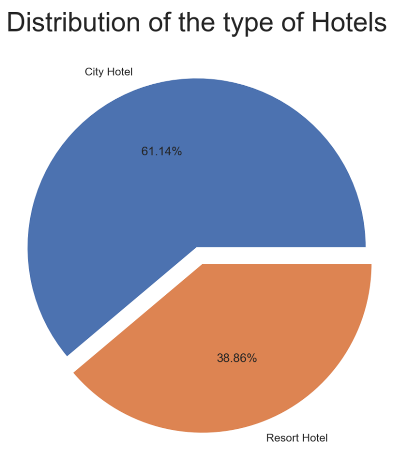
- Drop duplicate rows
- Correcting the data type of the arrival data year to object
- Binning countries to their respective continents

The reason why binning countries to their respective continents. The data was organizing the country of origin column. As there are a total of **178** different countries.
Each country was categorized into one of the following

- Africa
- Asia
- Australia
- Europe
- North America
- South America
- Unknown
- Others

3.1.2 Data Observations

This section makes general observations about the cleaned dataset. It includes graphs of different features in the hotel booking demand dataset versus the cancellation column.



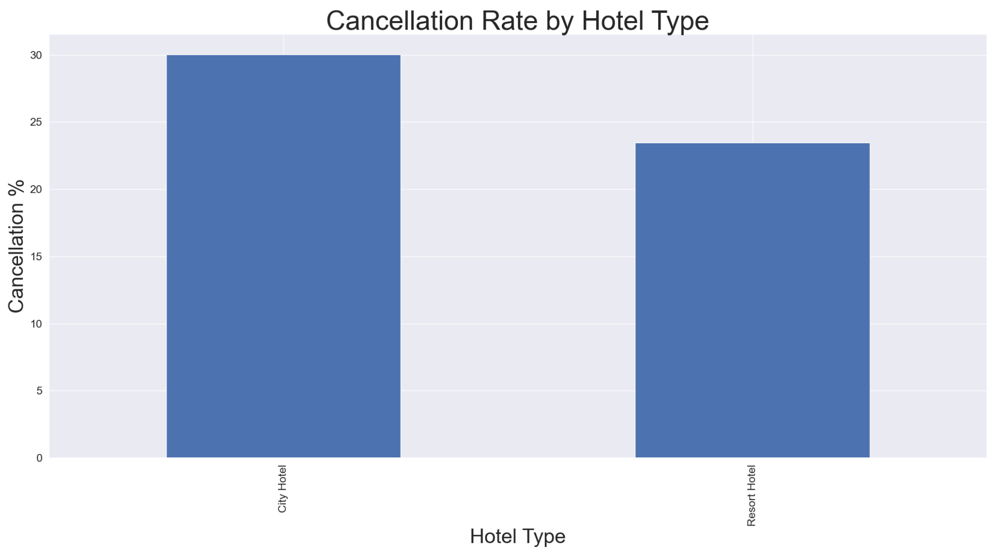
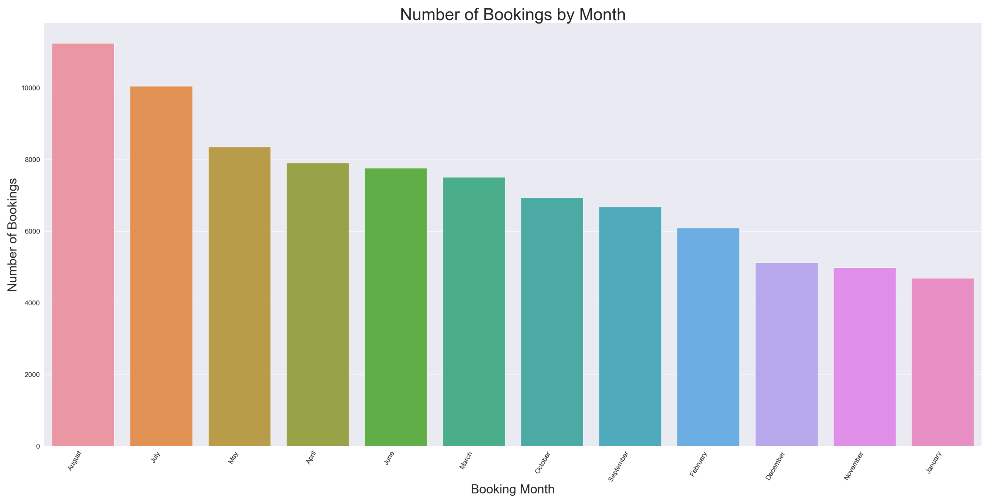


Figure 2: Booking by Hotel Type

We can see that city hotels are booked more than resort hotels. However, the cancellation rate of the city hotels are higher as well. This could lead to multiple interpretations such as the satisfaction rate being higher for resort hotels or the city hotels having features that allow easy cancellation or being offered at cheaper prices so that canceling isn't too much of a deal.



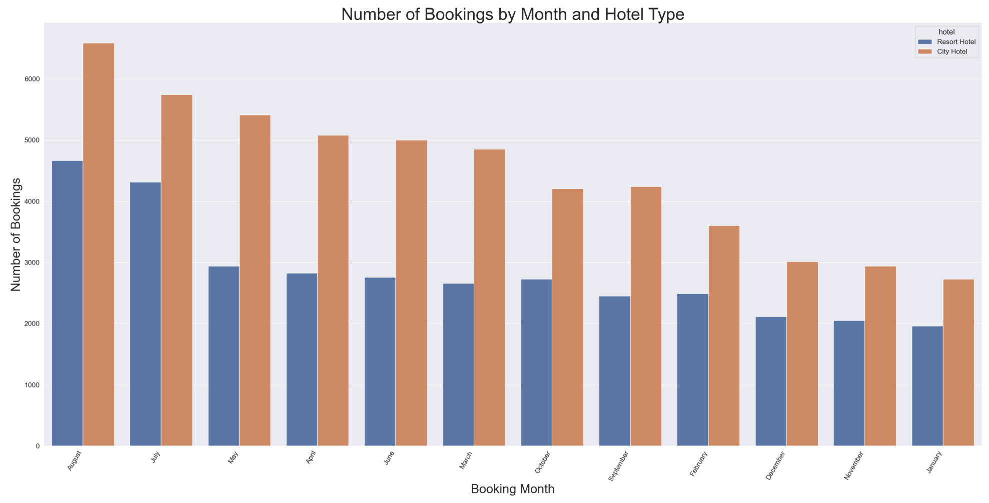


Figure 3: Booking by Month

We can observe that the typical summer months (July, August) along with the later spring months (April, May) tend to have more bookings compared to the other months. For every month, there are more bookings made at city hotels compared to resort hotels.

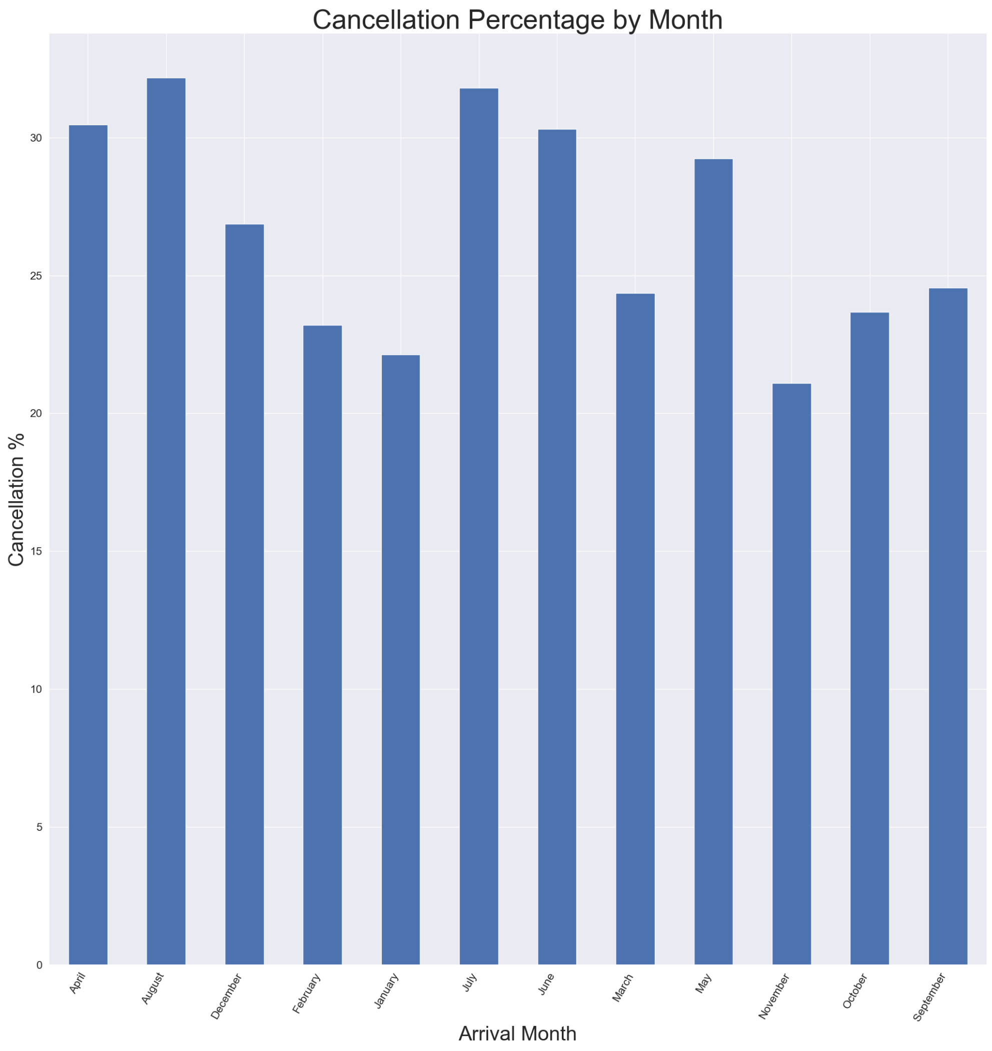


Figure 4: Cancellation by Arrival Month

In correlation with the months that guests booked the most, cancellations during these months were almost the highest. The typical summer months (August, July) and the late spring months (April, and May) show a high cancellation rate of around 30%.

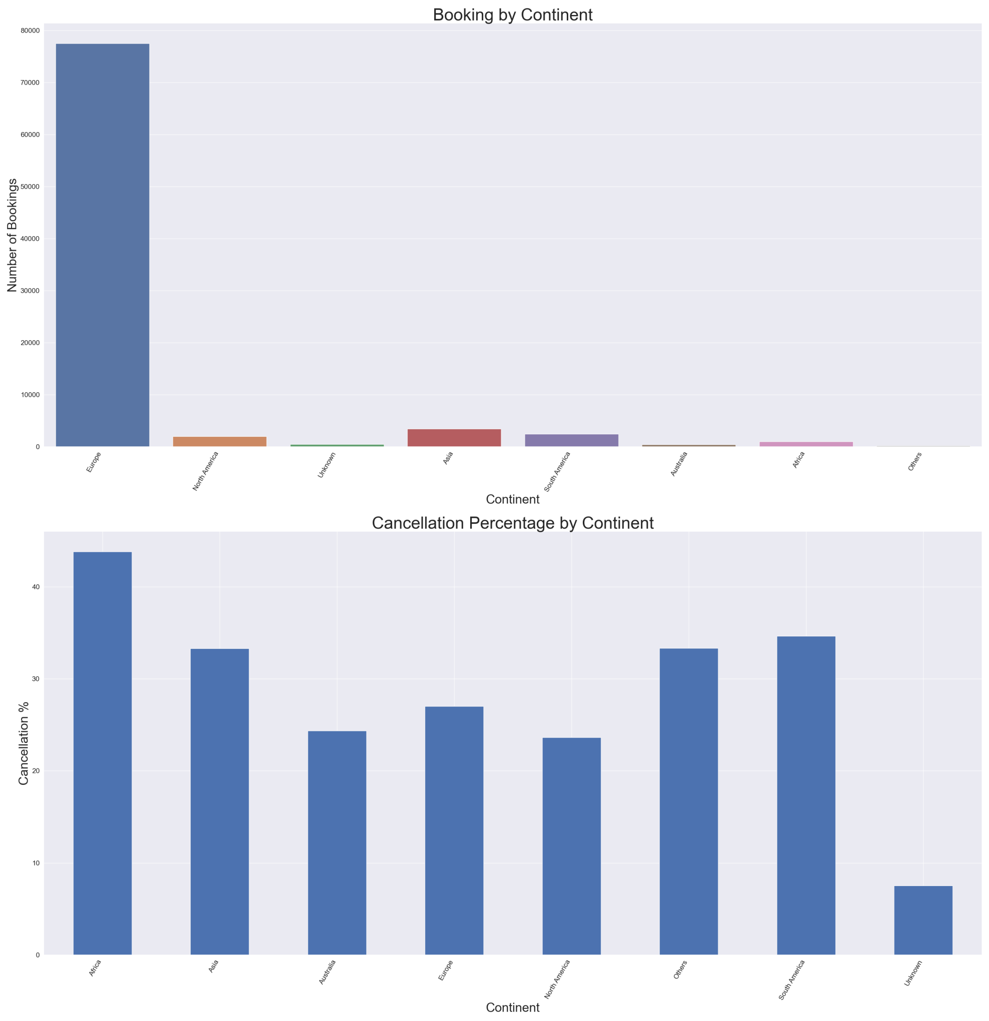


Figure 5: Booking by Continent

Most bookings were made in Europe. However, most cancellations came from Africa. This could be due to a number of reasons. I can think of transportation difficulties or denial of entry to where the hotels are situated which also could relate to the difficulty of obtaining a certain travel permit or visa to where the hotel is located.

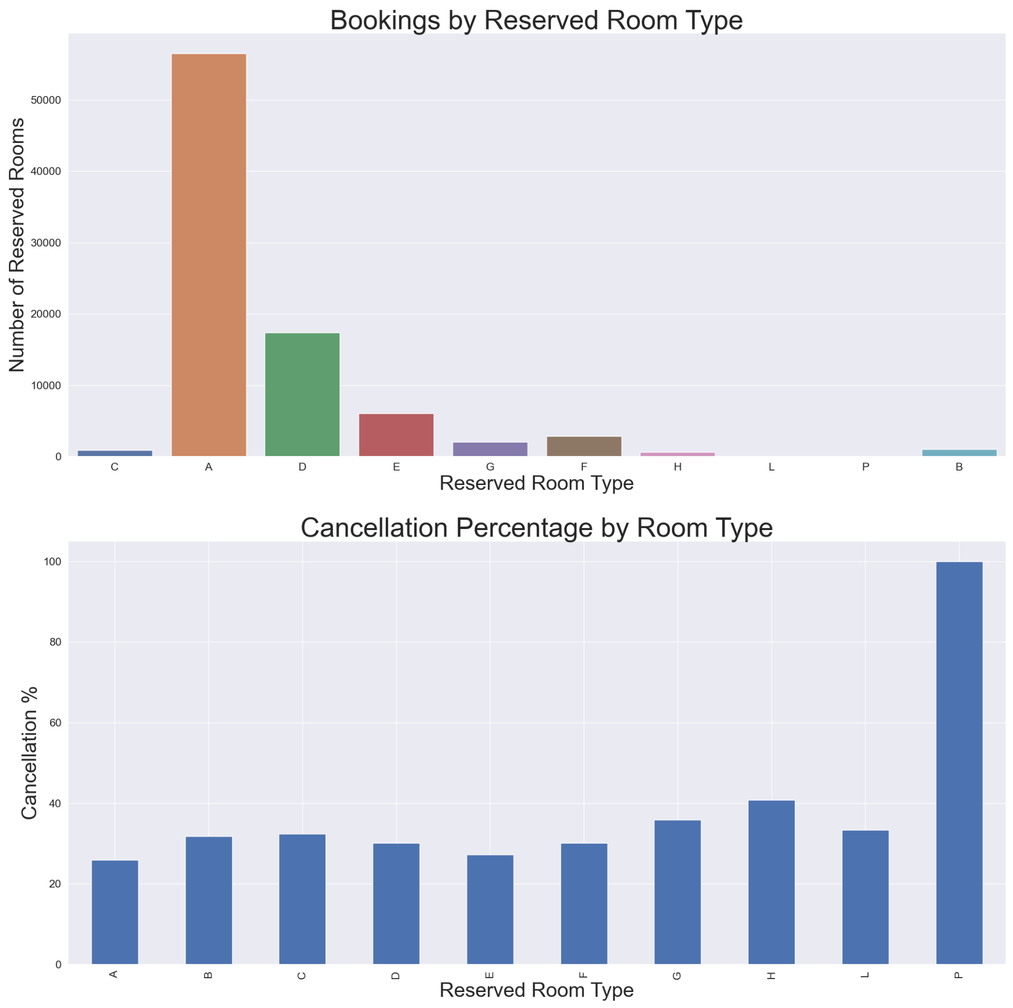


Figure 6: Booking by Room Type

We can observe here that the Room Type A was the most booked. This could be due to numerous reasons. I believe the most acceptable answer would be that Room Type A offered a cost-effective rate that most guests would want. In contrast, Room Type P was the most cancelled at 100%. There probably is some bias or a lack of data that we should consider removing from our dataset to make more accurate predictions.

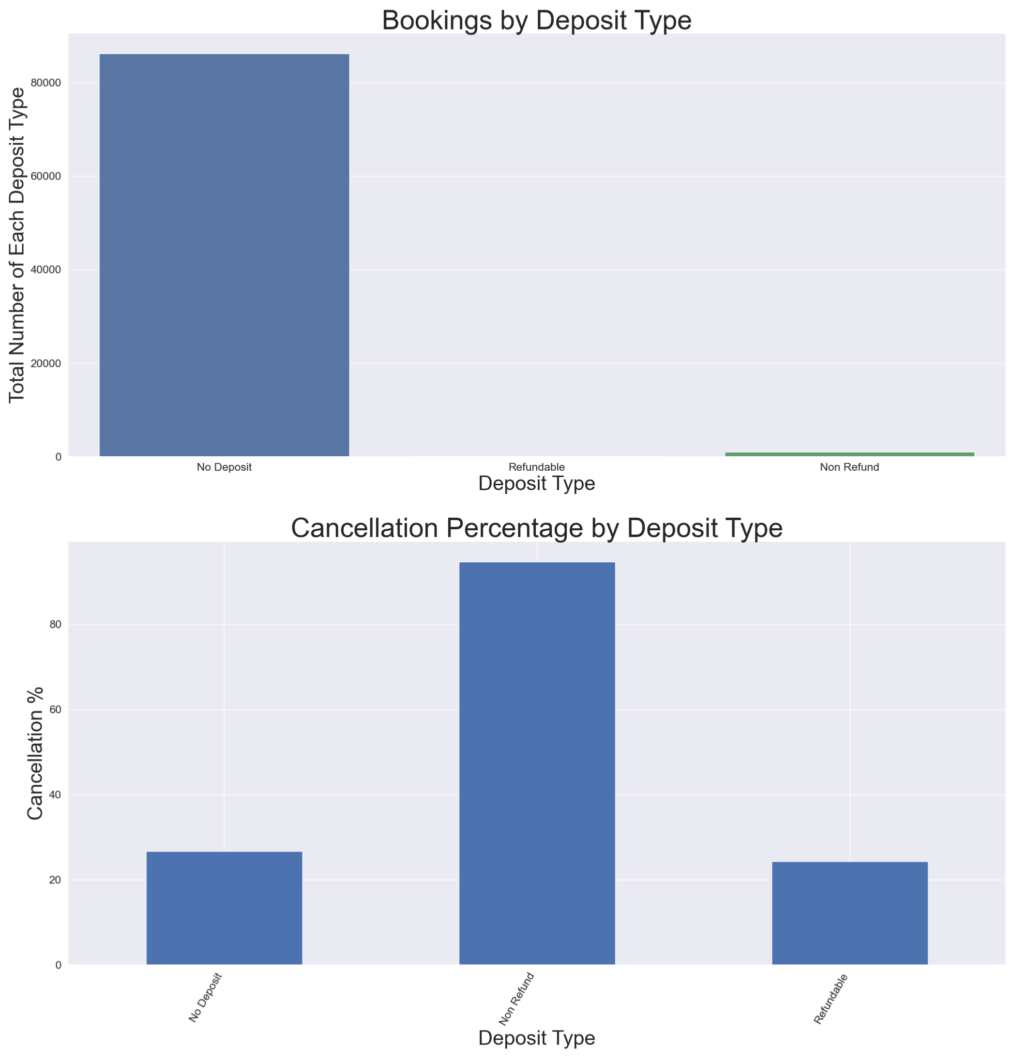
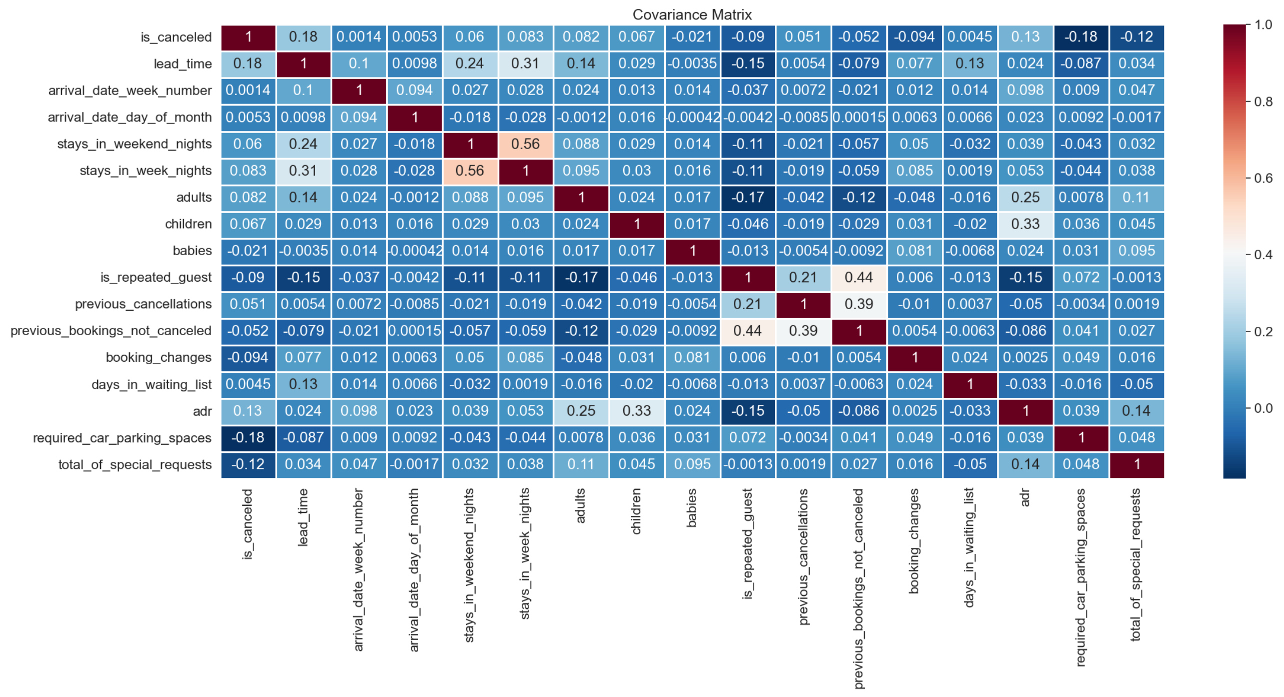


Figure 7: Booking by Deposit Type

Lastly, we can see that most bookings were done with the no deposit option. I think this goes in line with the cost-effective rate as most no deposit bookings are the cheapest. However, the cancellation rate was the highest for rooms that are not refundable which we will look into more detail as this may indicate some kind of unwanted bias in the dataset.

3.1.3 Sample Covariance Matrix



	is_canceled	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	booking_changes	days_in_waiting_list	adr	required_car_parking_spaces	total_of_special_requests
is_canceled	1	0.18	0.0014	0.0053	0.06	0.083	0.082	0.067	-0.021	-0.09	0.051	-0.052	-0.094	0.0045	0.13	-0.18	-0.12
lead_time	0.18	1	0.1	0.0098	0.24	0.31	0.14	0.029	-0.0035	-0.15	0.0054	-0.079	0.077	0.13	0.024	-0.087	0.034
arrival_date_week_number	0.0014	0.1	1	0.094	0.027	0.028	0.024	0.013	0.014	-0.037	0.0072	-0.021	0.012	0.014	0.098	0.009	0.047
arrival_date_day_of_month	0.0053	0.0098	0.094	1	-0.018	-0.028	-0.0012	0.016	-0.00042	-0.0042	-0.0085	0.00015	0.0063	0.0066	0.023	0.0092	-0.0017
stays_in_weekend_nights	0.06	0.24	0.027	-0.018	1	0.56	0.088	0.029	0.014	-0.11	-0.021	-0.057	0.05	-0.032	0.039	-0.043	0.032
stays_in_week_nights	0.083	0.31	0.028	-0.028	0.56	1	0.095	0.03	0.016	-0.11	-0.019	-0.059	0.085	0.0019	0.053	-0.044	0.038
adults	0.082	0.14	0.024	-0.0012	0.088	0.095	1	0.024	0.017	-0.17	-0.042	-0.12	-0.048	-0.016	0.25	0.0078	0.11
children	0.067	0.029	0.013	0.016	0.029	0.03	0.024	1	0.017	-0.046	-0.019	-0.029	0.031	-0.02	0.33	0.036	0.045
babies	-0.021	-0.0035	0.014	-0.00042	0.014	0.016	0.017	0.017	1	-0.013	-0.0054	-0.0092	0.081	-0.0068	0.024	0.031	0.095
is_repeated_guest	-0.09	-0.15	-0.037	-0.0042	-0.11	-0.11	-0.17	-0.046	-0.013	1	0.21	0.44	0.006	-0.013	-0.15	0.072	-0.0013
previous_cancellations	0.051	0.0054	0.0072	-0.0085	-0.021	-0.019	-0.042	-0.019	-0.0054	0.21	1	0.39	-0.01	0.0037	-0.05	-0.0034	0.0019
previous_bookings_not_canceled	-0.052	-0.079	-0.021	0.00015	-0.057	-0.059	-0.12	-0.029	-0.0092	0.44	0.39	1	0.0054	-0.0063	-0.086	0.041	0.027
booking_changes	-0.094	0.077	0.012	0.0063	0.05	0.085	-0.048	0.031	0.081	0.006	-0.01	0.0054	1	0.024	0.0025	0.049	0.016
days_in_waiting_list	0.0045	0.13	0.014	0.0066	-0.032	0.0019	-0.016	-0.02	-0.0068	-0.013	0.0037	-0.0063	0.024	1	-0.033	-0.016	-0.05
adr	0.13	0.024	0.098	0.023	0.039	0.053	0.25	0.33	0.024	-0.15	-0.05	-0.086	0.0025	-0.033	1	0.039	0.14
required_car_parking_spaces	-0.18	-0.087	0.009	0.0092	-0.043	-0.044	0.0078	0.036	0.031	0.072	-0.0034	0.041	0.049	-0.016	0.039	1	0.048
total_of_special_requests	-0.12	0.034	0.047	-0.0017	0.032	0.038	0.11	0.045	0.095	-0.0013	0.0019	0.027	0.016	-0.05	0.14	0.048	1

Figure 8: Initial Covariance Matrix

From the covariance matrix, we can see that features such as babies, if a guest is a repeated guest, if the guest did not cancel their previous booking if the guest made any booking changes, and if they required parking space, etc. have little to no covariance to whether or not a guest will cancel their booking.

With the observations made from the plot visualizations and the covariance matrix, we will move on to reducing the number of features to better fit our dataset/model.

3.1.4 Dimensional Reduction

Before I performed dimensionality reduction, I went ahead and printed out the correlation against the is_canceled column.

```
arrival_date_week_number          0.001409
days_in_waiting_list              0.004464
arrival_date_day_of_month         0.005301
babies                            0.020892
previous_cancellations            0.051471
previous_bookings_not_canceled    0.052160
stays_in_weekend_nights           0.060226
children                          0.067370
adults                            0.081765
stays_in_week_nights              0.082924
is_repeated_guest                 0.089655
booking_changes                   0.093650
total_of_special_requests         0.120565
adr                               0.127970
required_car_parking_spaces       0.184229
lead_time                         0.184801
is_canceled                       1.000000
```

The following columns were removed as they were viewed as redundant and had high correlation to each other. They were also seen as irrelevant values individually.

- Children (1)
- Babies (2)
- Adults (3)
- Stays in Week Nights (4)
- Stays in Weekend Nights (5)

Columns (1), (2), and (3) were combined into a new column named **total_members**
Columns (4) and (5), were combined into a new column named **total_nights**
After creating new columns, the following columns were dropped from the dataset:

- Children
- Babies
- Adults
- Stays in Week Nights

- Stays in Weekend Nights
- Days in Waiting List
- Reservation Status
- Reservation Status Date

After dropping these features, we move on to the feature transformation of the now-modified dataset.

3.1.5 Feature Transformation

This section describes which features were transformed to better fit our model. The following features were transformed:

- Hotel
- Meal Type
- Reserved Room Type
- Deposit Type

The hotel feature was transformed to either

- Resort Hotel: 0
- City Hotel: 1

The meal feature was transformed to either

- BB: 0
- FB: 1
- HB: 2
- Undefined: 4

The reserved room type feature was transformed to either

- C: 0
- A: 1
- D: 2
- E: 3

- G: 4
- F: 5
- H: 6
- L: 7
- P: 8
- B: 9

The deposit type feature was transformed to either

- No Deposit: 0
- Refundable: 1
- Non Refund: 2

3.1.6 Feature Selection

As mentioned, this paper focuses on the cancellation of guests based on their preferences and selections theyhave made on their booking.

Therefore, the **dependent variable** is the is_cancelled feature while the **independent variable(s)** are the remaining features in the dataset.

4 Phase II, Regression Analysis

This section presents the results of running the dataset through different regression analysis. The following analysis was done on the dataset:

- F-Test Analysis
- Stepwise Regression Analysis
- Collinearity Analysis

4.1 OLS Regression Test

Through the OLS Regression Results, the F-Test analysis, stepwise regression analysis, and collinearity analysis was conducted.
Below is the initial OLS Regression Results. We will be performing the analysis mentioned above make our model more fit.

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:            is_canceled   R-squared (uncentered):                   0.421
Model:                            OLS   Adj. R-squared (uncentered):              0.420
Method:                 Least Squares   F-statistic:                              798.4
Date:                Sun, 11 Dec 2022   Prob (F-statistic):                        0.00
Time:                        17:13:28   Log-Likelihood:                         -34893.
No. Observations:               69328   AIC:                                  6.991e+04
Df Residuals:                   69265   BIC:                                  7.049e+04
Df Model:                          63
Covariance Type:            nonrobust
==================================================================================================
                                     coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------
hotel                             -0.0268      0.004     -7.214      0.000      -0.034      -0.020
lead_time                          0.0008   2.21e-05     37.014      0.000       0.001       0.001
arrival_date_week_number           0.0023      0.002      0.968      0.333      -0.002       0.007
arrival_date_day_of_month      -4.212e-05      0.000     -0.108      0.914      -0.001       0.001
meal                               0.0023      0.001      1.570      0.116      -0.001       0.005
is_repeated_guest                 -0.0144      0.010     -1.443      0.149      -0.034       0.005
previous_cancellations             0.0785      0.005     16.699      0.000       0.069       0.088
previous_bookings_not_canceled    -0.0064      0.001     -5.630      0.000      -0.009      -0.004
reserved_room_type                 0.0260      0.002     13.483      0.000       0.022       0.030
booking_changes                   -0.0382      0.002    -17.245      0.000      -0.043      -0.034
deposit_type                       0.1884      0.005     37.003      0.000       0.178       0.198
adr                                0.0008   3.92e-05     19.204      0.000       0.001       0.001
required_car_parking_spaces       -0.2362      0.006    -41.959      0.000      -0.247      -0.225
total_of_special_requests         -0.1071      0.002    -54.154      0.000      -0.111      -0.103
total_members                      0.0287      0.002     11.785      0.000       0.024       0.033
total_nights                       0.0073      0.001     11.296      0.000       0.006       0.009
arrival_date_year_2016             0.0009      0.005      0.173      0.863      -0.009       0.011
arrival_date_year_2017             0.0226      0.006      3.657      0.000       0.010       0.035
```

```
arrival_date_month_August        -0.0638      0.043     -1.498      0.134      -0.147       0.020
arrival_date_month_December      -0.0416      0.084     -0.498      0.618      -0.205       0.122
arrival_date_month_February       0.0216      0.021      1.024      0.306      -0.020       0.063
arrival_date_month_January        0.0204      0.031      0.656      0.512      -0.041       0.081
arrival_date_month_July          -0.0621      0.032     -1.929      0.054      -0.125       0.001
arrival_date_month_June          -0.0459      0.023     -2.038      0.042      -0.090      -0.002
arrival_date_month_March         -0.0004      0.012     -0.036      0.971      -0.025       0.024
arrival_date_month_May           -0.0279      0.013     -2.143      0.032      -0.053      -0.002
arrival_date_month_November      -0.0641      0.074     -0.869      0.385      -0.209       0.080
arrival_date_month_October       -0.0636      0.063     -1.003      0.316      -0.188       0.061
arrival_date_month_September     -0.0660      0.053     -1.238      0.216      -0.171       0.039
country_Asia                     -0.0888      0.016     -5.534      0.000      -0.120      -0.057
country_Australia                -0.2066      0.027     -7.597      0.000      -0.260      -0.153
country_Europe                   -0.1209      0.014     -8.561      0.000      -0.149      -0.093
country_North America            -0.1702      0.017     -9.847      0.000      -0.204      -0.136
country_Others                   -0.1294      0.039     -3.349      0.001      -0.205      -0.054
country_South America            -0.0817      0.017     -4.895      0.000      -0.114      -0.049
country_Unknown                  -0.1388      0.026     -5.344      0.000      -0.190      -0.088
market_segment_Complementary      0.2194      0.036      6.126      0.000       0.149       0.290
market_segment_Corporate          0.0965      0.029      3.316      0.001       0.039       0.154
market_segment_Direct             0.0505      0.033      1.521      0.128      -0.015       0.116
market_segment_Groups             0.0150      0.032      0.473      0.636      -0.047       0.077
market_segment_Offline TA/TO     -0.0734      0.032     -2.291      0.022      -0.136      -0.011
market_segment_Online TA          0.1723      0.032      5.404      0.000       0.110       0.235
market_segment_Undefined          0.4302      0.464      0.928      0.353      -0.478       1.339
distribution_channel_Direct      -0.0391      0.018     -2.170      0.030      -0.074      -0.004
distribution_channel_GDS         -0.0919      0.038     -2.398      0.016      -0.167      -0.017
distribution_channel_TA/TO        0.0121      0.018      0.685      0.493      -0.023       0.047
distribution_channel_Undefined    0.5754      0.232      2.480      0.013       0.121       1.030
assigned_room_type_B             -0.1315      0.013    -10.037      0.000      -0.157      -0.106
assigned_room_type_C             -0.0830      0.011     -7.879      0.000      -0.104      -0.062
assigned_room_type_D             -0.0677      0.004    -16.762      0.000      -0.076      -0.060
assigned_room_type_E             -0.0898      0.007    -13.300      0.000      -0.103      -0.077
assigned_room_type_F             -0.1668      0.010    -16.513      0.000      -0.187      -0.147
assigned_room_type_G             -0.1538      0.011    -13.670      0.000      -0.176      -0.132
assigned_room_type_H             -0.1619      0.020     -8.252      0.000      -0.200      -0.123
assigned_room_type_I             -0.1535      0.031     -4.942      0.000      -0.214      -0.093
assigned_room_type_K             -0.1658      0.041     -4.067      0.000      -0.246      -0.086
assigned_room_type_L              0.6327      0.401      1.579      0.114      -0.153       1.418
assigned_room_type_P              0.6785      0.165      4.100      0.000       0.354       1.003
agent_No agent                    0.0006      0.007      0.084      0.933      -0.013       0.014
company_Individuals               0.0456      0.015      3.020      0.003       0.016       0.075
customer_type_Group               0.0108      0.021      0.503      0.615      -0.031       0.053
customer_type_Transient           0.1088      0.009     12.434      0.000       0.092       0.126
customer_type_Transient-Party     0.0159      0.010      1.627      0.104      -0.003       0.035
==============================================================================
Omnibus:                     6849.840   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7332.439
Skew:                           0.750   Prob(JB):                         0.00
Kurtosis:                       2.461   Cond. No.                     4.87e+04
==============================================================================
Notes:
```

```
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 4.87e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
Mean Absolute Error: 0.1594882646270388
```

Each criteria was taken a look on their coefficient values, standard errors, t values, p-values, and their confidence intervals.

To perform the stepwise regression, the following features were removed as their p-values exceeded 0.05, the threshold.

- Country
- Arrival Data Year
- Market Segment
- Agent
- Company

The following were removed due to their high standard error values:

- Assigned Room Type = 'L'
- Assigned Room Type = 'P'
- Distribution Channel = 'Undefined'

This is the final OLS Regression results.

```
                            OLS Regression Results
=======================================================================================
Dep. Variable:             is_canceled   R-squared (uncentered):                   0.397
Model:                             OLS   Adj. R-squared (uncentered):              0.397
Method:                  Least Squares   F-statistic:                              1088.
Date:                 Sun, 11 Dec 2022   Prob (F-statistic):                        0.00
Time:                         17:32:14   Log-Likelihood:                          -36300.
No. Observations:                69319   AIC:                                  7.268e+04
Df Residuals:                    69277   BIC:                                  7.307e+04
Df Model:                           42
Covariance Type:             nonrobust
==================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------
hotel                           -0.0090      0.004     -2.416      0.016      -0.016      -0.002
lead_time                        0.0008   2.14e-05     36.601      0.000       0.001       0.001
arrival_date_week_number        -0.0056      0.001     -5.922      0.000      -0.007      -0.004
arrival_date_day_of_month        0.0010      0.000      4.181      0.000       0.001       0.001
meal                             0.0039      0.001      2.586      0.010       0.001       0.007
is_repeated_guest               -0.0253      0.010     -2.560      0.010      -0.045      -0.006
previous_cancellations           0.0831      0.005     17.343      0.000       0.074       0.092
previous_bookings_not_canceled  -0.0068      0.001     -6.124      0.000      -0.009      -0.005
reserved_room_type               0.0345      0.002     17.705      0.000       0.031       0.038
booking_changes                 -0.0389      0.002    -17.212      0.000      -0.043      -0.034
deposit_type                     0.1556      0.005     31.599      0.000       0.146       0.165
adr                              0.0011   3.74e-05     28.288      0.000       0.001       0.001
required_car_parking_spaces     -0.2330      0.006    -40.281      0.000      -0.244      -0.222
total_of_special_requests       -0.0820      0.002    -42.237      0.000      -0.086      -0.078
total_members                    0.0270      0.002     10.959      0.000       0.022       0.032
total_nights                     0.0048      0.001      7.496      0.000       0.004       0.006
arrival_date_month_August        0.0559      0.020      2.855      0.004       0.018       0.094
arrival_date_month_December      0.2213      0.036      6.122      0.000       0.150       0.292
arrival_date_month_February     -0.0437      0.010     -4.479      0.000      -0.063      -0.025
arrival_date_month_January      -0.0675      0.013     -5.121      0.000      -0.093      -0.042
arrival_date_month_July          0.0267      0.016      1.703      0.089      -0.004       0.057
arrival_date_month_June          0.0081      0.012      0.651      0.515      -0.016       0.033
arrival_date_month_March        -0.0414      0.007     -5.530      0.000      -0.056      -0.027
arrival_date_month_May          -0.0018      0.009     -0.194      0.846      -0.020       0.016
arrival_date_month_November      0.1628      0.033      4.978      0.000       0.099       0.227
arrival_date_month_October       0.1232      0.029      4.319      0.000       0.067       0.179
arrival_date_month_September     0.0773      0.024      3.159      0.002       0.029       0.125
distribution_channel_Direct     -0.0608      0.008     -7.422      0.000      -0.077      -0.045
distribution_channel_GDS        -0.0802      0.036     -2.217      0.027      -0.151      -0.009
distribution_channel_TA/TO       0.0584      0.008      7.657      0.000       0.043       0.073
assigned_room_type_B            -0.1682      0.013    -12.586      0.000      -0.194      -0.142
assigned_room_type_C            -0.0865      0.011     -8.154      0.000      -0.107      -0.066
assigned_room_type_D            -0.0865      0.004    -21.082      0.000      -0.095      -0.078
assigned_room_type_E            -0.1087      0.007    -15.859      0.000      -0.122      -0.095
assigned_room_type_F            -0.1904      0.010    -18.600      0.000      -0.210      -0.170
assigned_room_type_G            -0.1606      0.012    -13.962      0.000      -0.183      -0.138
```

```
assigned_room_type_H           -0.1674      0.020     -8.488      0.000      -0.206      -0.129
assigned_room_type_I           -0.1306      0.031     -4.238      0.000      -0.191      -0.070
assigned_room_type_K           -0.1874      0.041     -4.531      0.000      -0.268      -0.106
customer_type_Group             0.0508      0.021      2.387      0.017       0.009       0.093
customer_type_Transient         0.1952      0.009     22.954      0.000       0.178       0.212
customer_type_Transient-Party   0.0320      0.009      3.389      0.001       0.013       0.050
==============================================================================
Omnibus:                     5670.386   Durbin-Watson:                   1.991
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             6829.576
Skew:                           0.751   Prob(JB):                         0.00
Kurtosis:                       2.675   Cond. No.                     6.76e+03
==============================================================================
Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 6.76e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
Mean Absolute Error: 0.16637407553935074
```

5 Phase III, Classification Analysis

This section presents the results of applying different machine learning classifiers to the hotel booking dataset. The following classifiers were used:

- Decision Tree
- Logistic Regressionm
- KNN
- SVM
- Naive Bayes
- Random Forest
- Neural Network

The performance of the classifiers were compared using the following criteria:

- Confusion Matrix
- Accuracy
- Precision, Recall, F1-Score
- Specificity
- G Score
- ROC Curve

Each section presents the performance of an individual classifier. The last section, Performance Comparison, has a table comparing the performance of all the presented classifiers in one table.

5.1 Decision Tree

5.1.1 Confusion Matrix

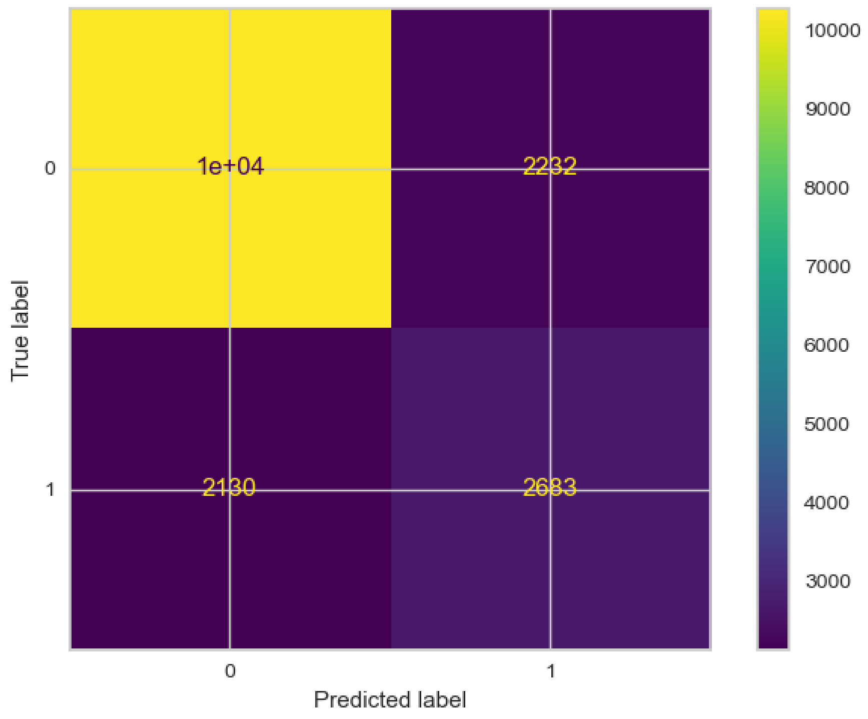


Figure 9: Decision Tree (Default) Confusion Matrix

5.1.2 Decision Tree Accuracy

```
    Decision Tree (Initial) Accuracy Score: 0.748
```

5.1.3 Decision Tree Classification Report

```
Decision Tree (Initial) Classification Report:
              precision    recall  f1-score   support
           0       0.83      0.82      0.83     12517
           1       0.55      0.56      0.55      4813
    accuracy                           0.75     17330
   macro avg       0.69      0.69      0.69     17330
weighted avg       0.75      0.75      0.75     17330
```

5.1.4 Decision Tree Specificity

```
Decision Tree (Initial) Specificity: 0.822
```

5.1.5 Decision Tree ROC Curve

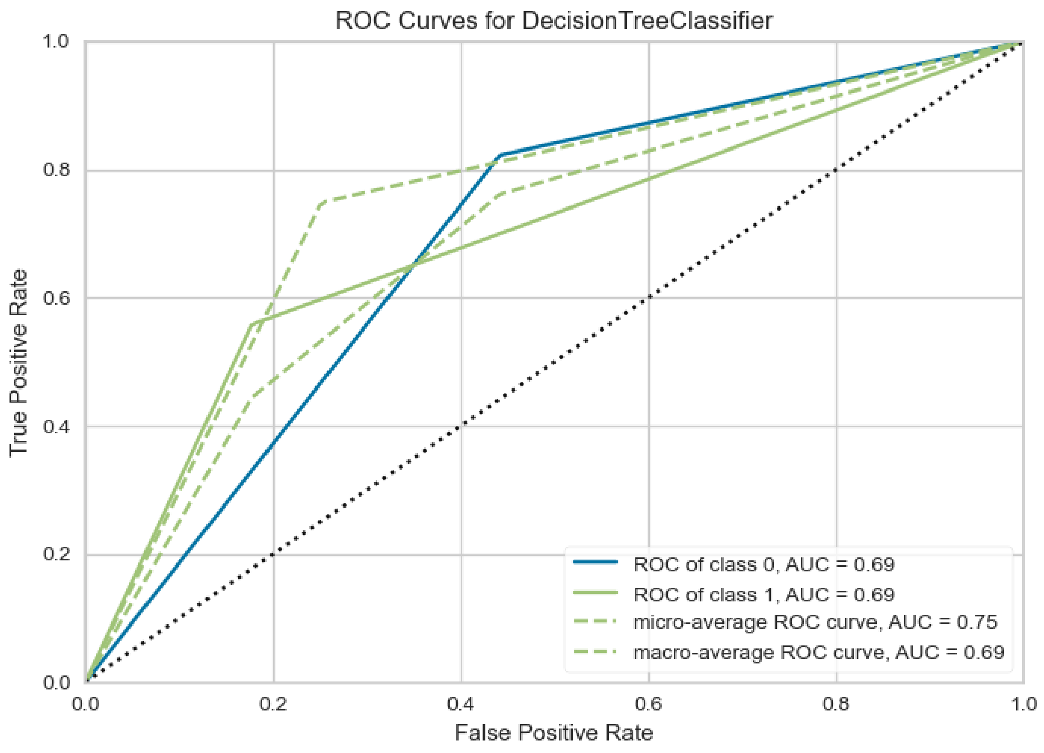


Figure 10: Decision Tree (Default) ROC Curve

5.2 Logistic Regression

5.2.1 Logistic Regression Confusion Matrix

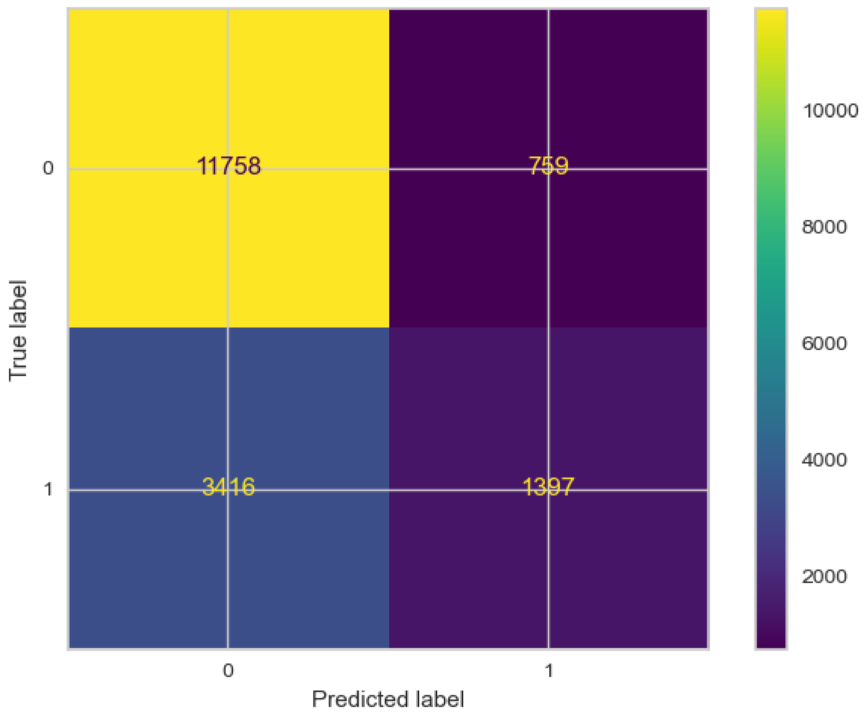


Figure 11: Logistic Regression Confusion Matrix

5.2.2 Logistic Regression Accuracy

```
    Logistic Regression Accuracy Score: 0.759
```

5.2.3 Logistic Regression Classification Report

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support
           0       0.77      0.94      0.85     12517
           1       0.65      0.29      0.40      4813
    accuracy                           0.76     17330
   macro avg       0.71      0.61      0.63     17330
weighted avg       0.74      0.76      0.72     17330
```

5.2.4 Logistic Regression Specificity

```
Logistic Regression Specificity: 0.939
```

5.2.5 Logistic Regression ROC Curve

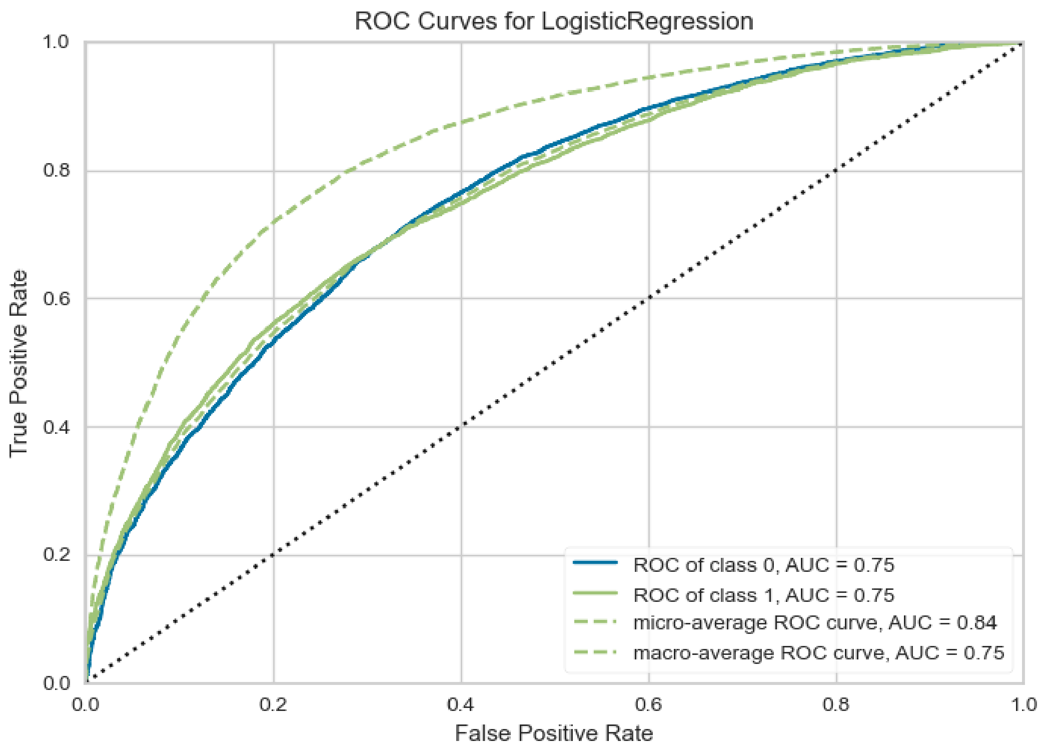


Figure 12: Logistic Regression ROC Curve

5.3 KNN

5.3.1 KNN Confusion Matrix

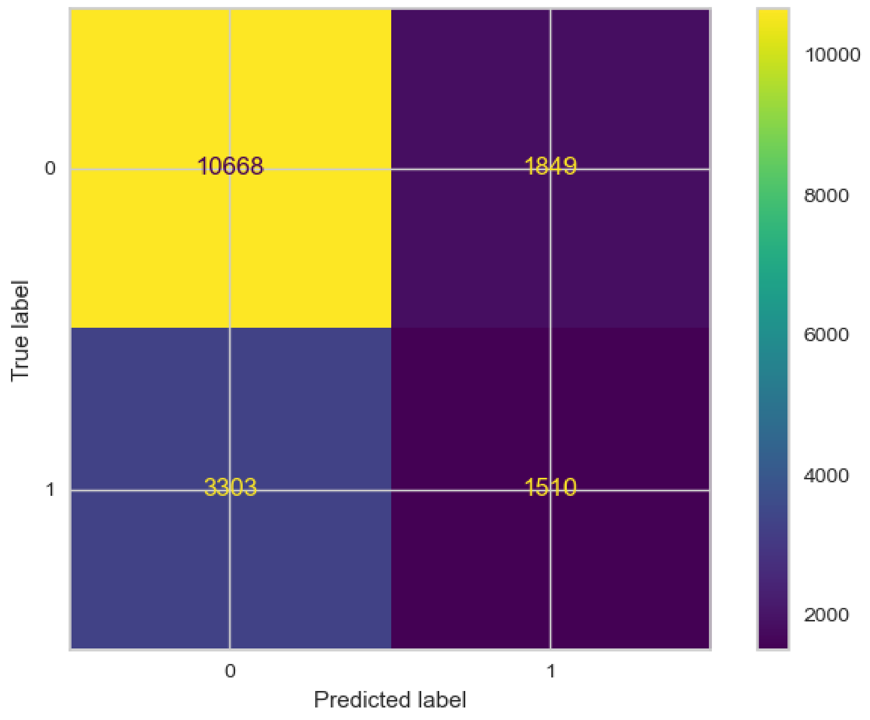


Figure 13: KNN (Default) Confusion Matrix

5.3.2 KNN Accuracy

```
    KNN (Default) Accuracy Score: 0.703
```

5.3.3 KNN Classification Report

```
KNN (Default) Classification Report:
              precision    recall  f1-score   support
           0       0.76      0.85      0.81     12517
           1       0.45      0.31      0.37      4813
    accuracy                           0.70     17330
   macro avg       0.61      0.58      0.59     17330
weighted avg       0.68      0.70      0.68     17330
```

5.3.4 KNN Specificity

```
KNN (Default) Specificity: 0.852
```

5.3.5 KNN ROC Curve

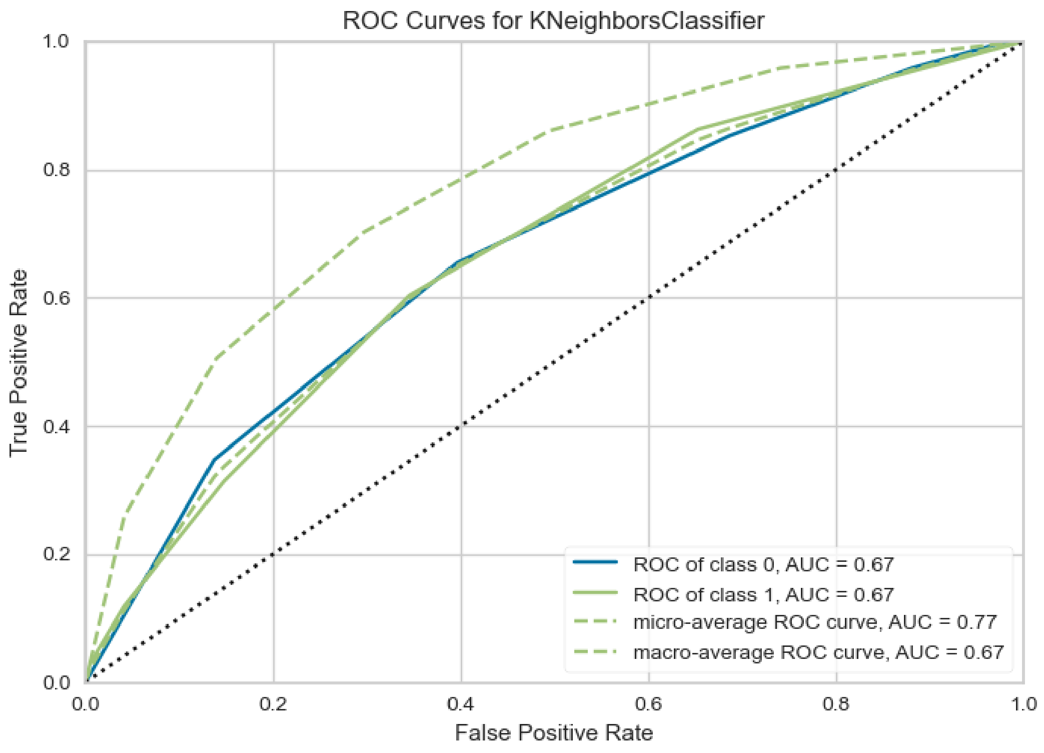


Figure 14: KNN ROC Curve

5.4 SVM

5.4.1 SVM Confusion Matrix

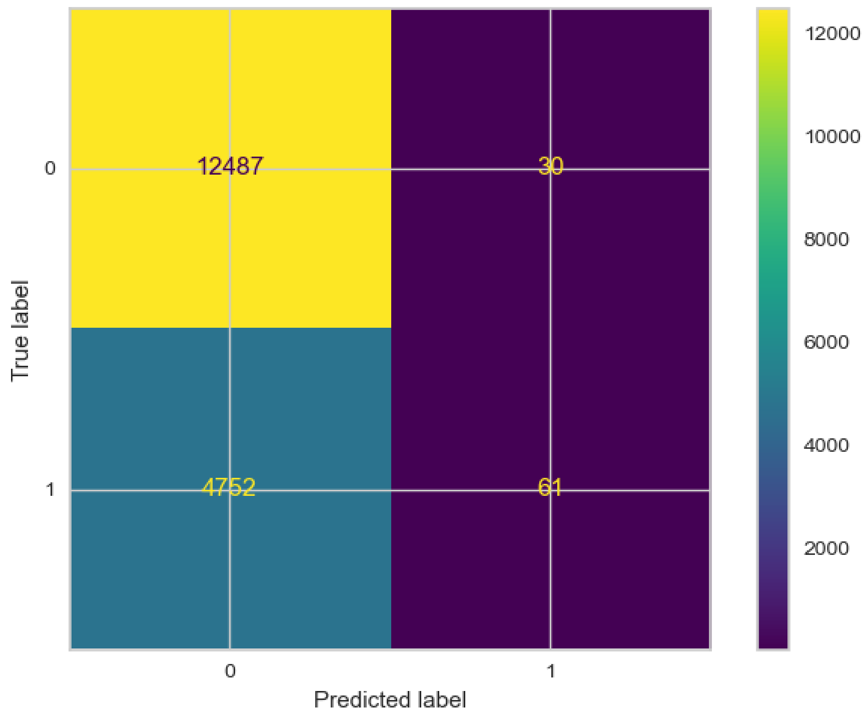


Figure 15: SVM Confusion Matrix

5.4.2 SVM Accuracy

```
SVM Accuracy Score: 0.724
```

5.4.3 SVM Classification Report

```
SVM Classification Report:
              precision    recall  f1-score   support
           0       0.72      1.00      0.84     12517
           1       0.67      0.01      0.02      4813
    accuracy                           0.72     17330
   macro avg       0.70      0.51      0.43     17330
weighted avg       0.71      0.72      0.61     17330
```

5.4.4 SVM Specificity

```
SVM Specificity: 0.998
```

5.5 Naive Bayes

5.5.1 Naive Bayes Confusion Matrix

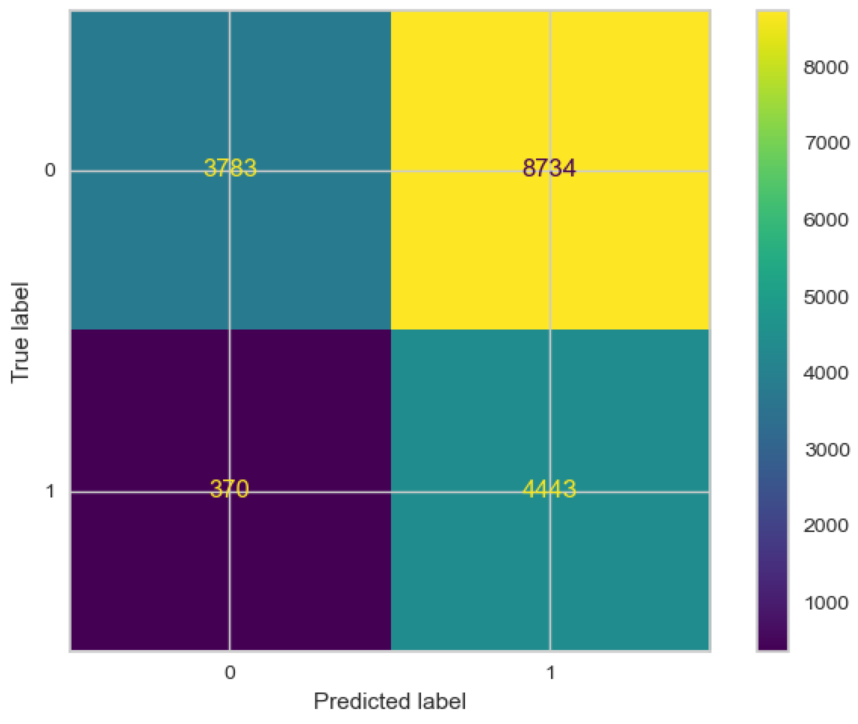


Figure 16: Naive Bayes Confusion Matrix

5.5.2 Naive Bayes Accuracy

```
Naive Bayes Accuracy Score: 0.475
```

5.5.3 Naive Bayes Classification Report

```
Naive Bayes Classification Report:
              precision    recall  f1-score   support
           0       0.91      0.30      0.45     12517
           1       0.34      0.92      0.49      4813
    accuracy                           0.47     17330
   macro avg       0.62      0.61      0.47     17330
weighted avg       0.75      0.47      0.46     17330
```

5.5.4 Naive Bayes Specificity

```
Naive Bayes Specificity: 0.302
```

5.5.5 Naive Bayes ROC Curve

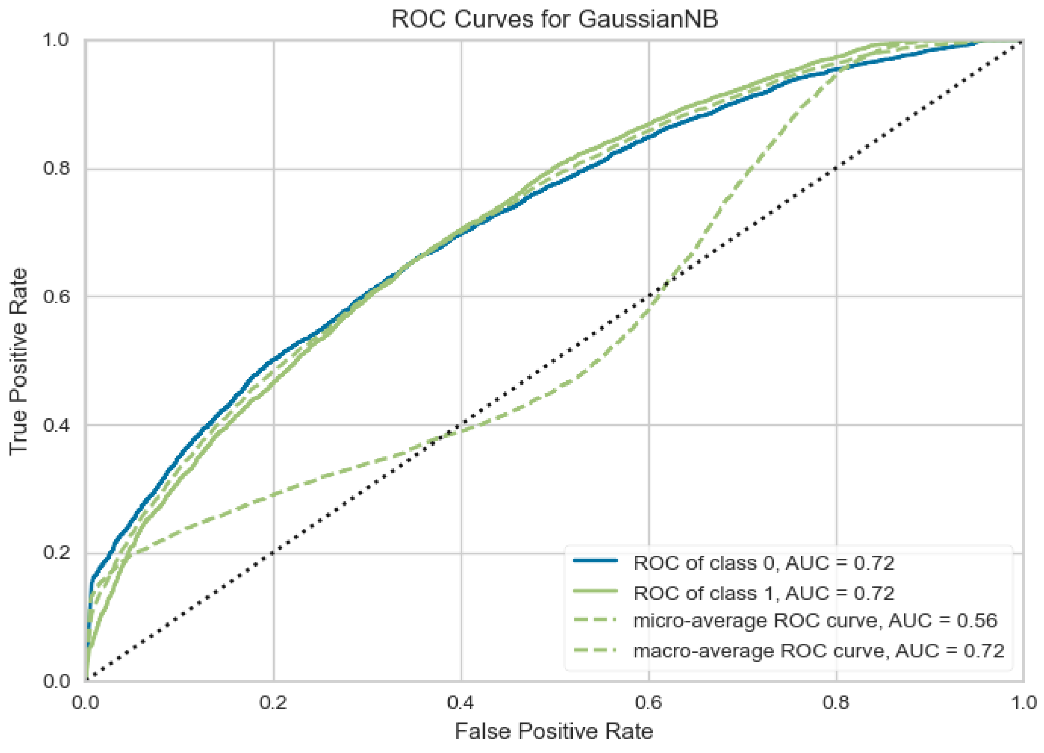


Figure 17: Naive Bayes ROC Curve

5.6 Random Forest (Gini)

5.6.1 Random Forest (Gini) Confusion Matrix

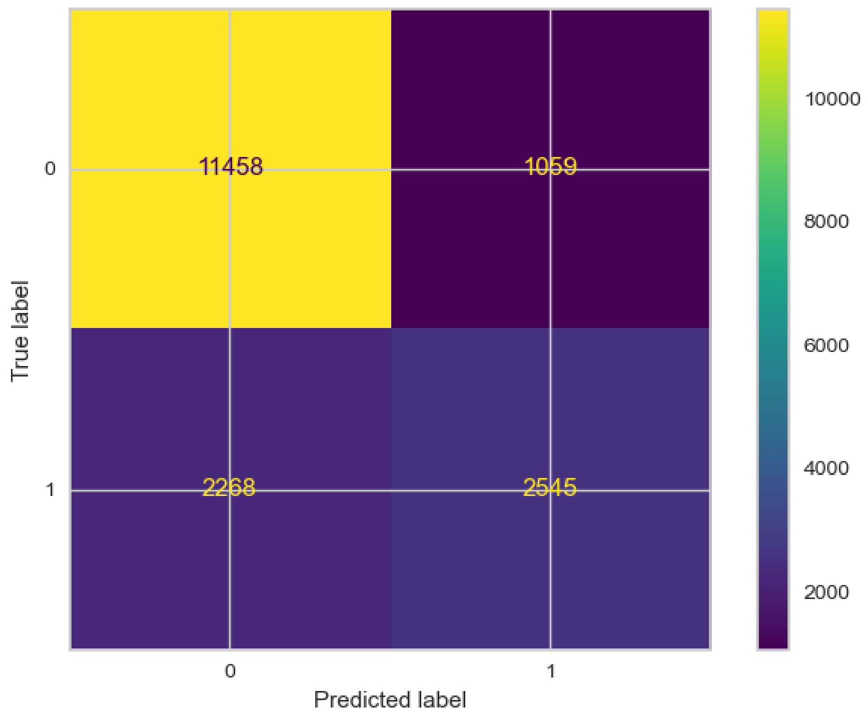


Figure 18: Random Forest (Gini) Confusion Matrix

5.6.2 Random Forest (Gini) Accuracy

```
Random Forest Accuracy Score: 0.808
```

5.6.3 Random Forest (Gini) Classification Report

```
Random Forest Classification Report:
              precision    recall  f1-score   support
           0       0.83      0.92      0.87     12517
           1       0.71      0.53      0.60      4813
    accuracy                           0.81     17330
   macro avg       0.77      0.72      0.74     17330
weighted avg       0.80      0.81      0.80     17330
```

5.6.4 Random Forest (Gini) Specificity

```
Random Forest Specificity: 0.915
```

5.6.5 Random Forest (Gini) ROC Curve

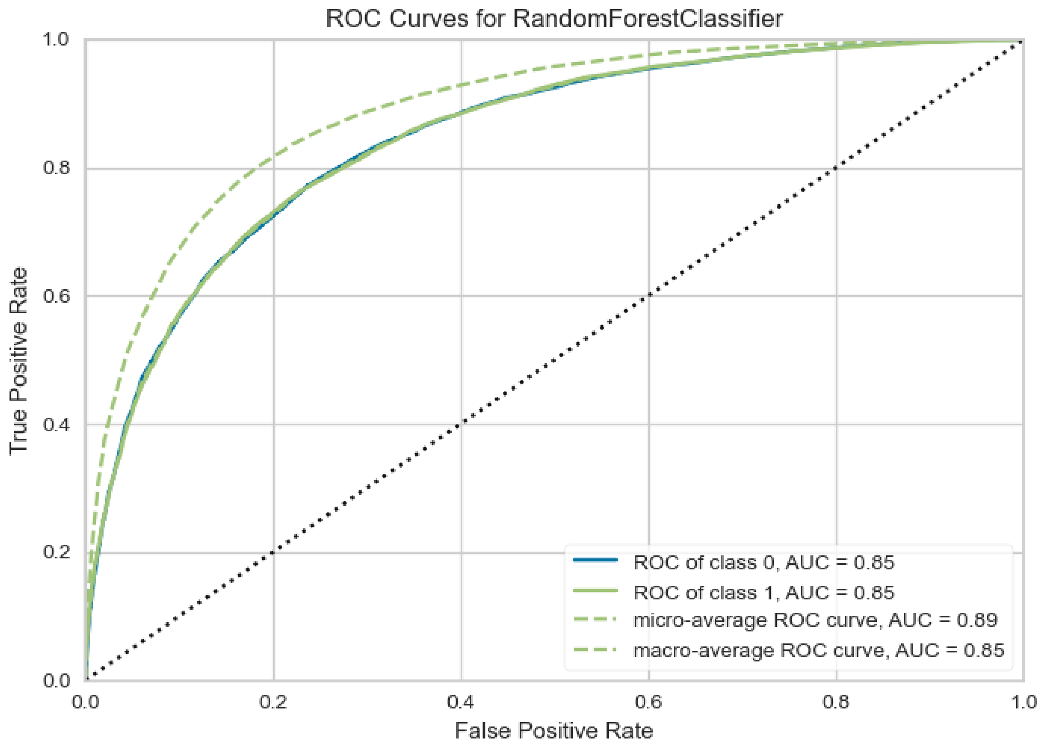


Figure 19: Random Forest (Gini) ROC Curve

5.7 Random Forest (Entropy)

5.7.1 Random Forest (Entropy) Confusion Matrix

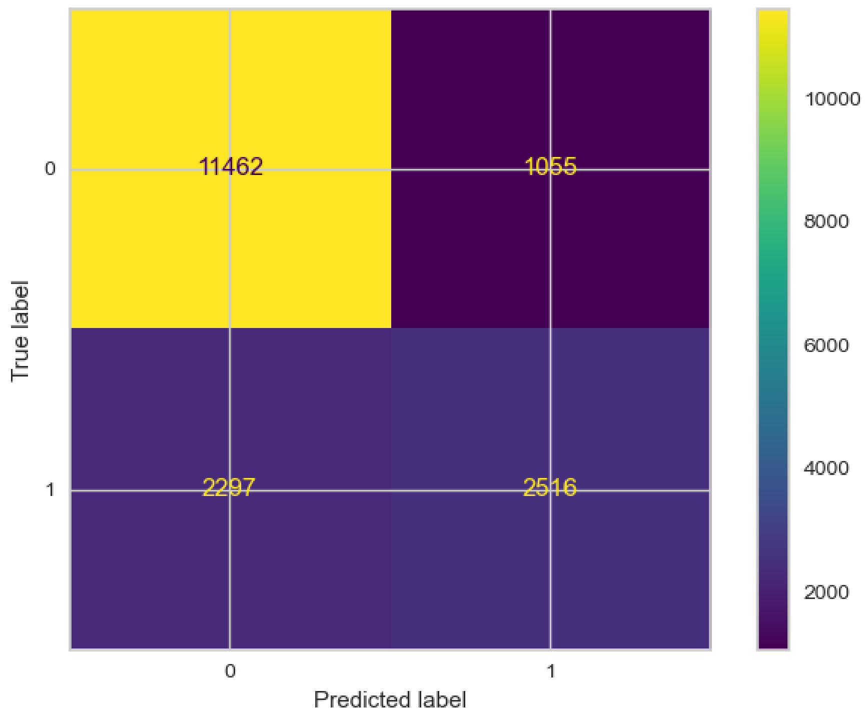


Figure 20: Random Forest (Entropy) Confusion Matrix

5.7.2 Random Forest (Entropy) Accuracy

```
Random Forest (Entropy) Accuracy Score: 0.807
```

5.7.3 Random Forest (Entropy) Classification Report

```
Random Forest (Entropy) Classification Report:
              precision    recall  f1-score   support
           0       0.83      0.92      0.87     12517
           1       0.70      0.52      0.60      4813
    accuracy                           0.81     17330
   macro avg       0.77      0.72      0.74     17330
weighted avg       0.80      0.81      0.80     17330
```

5.7.4 Random Forest (Entropy) Specificity

```
Random Forest (Entropy) Specificity: 0.916
```

5.7.5 Random Forest (Entropy) ROC Curve

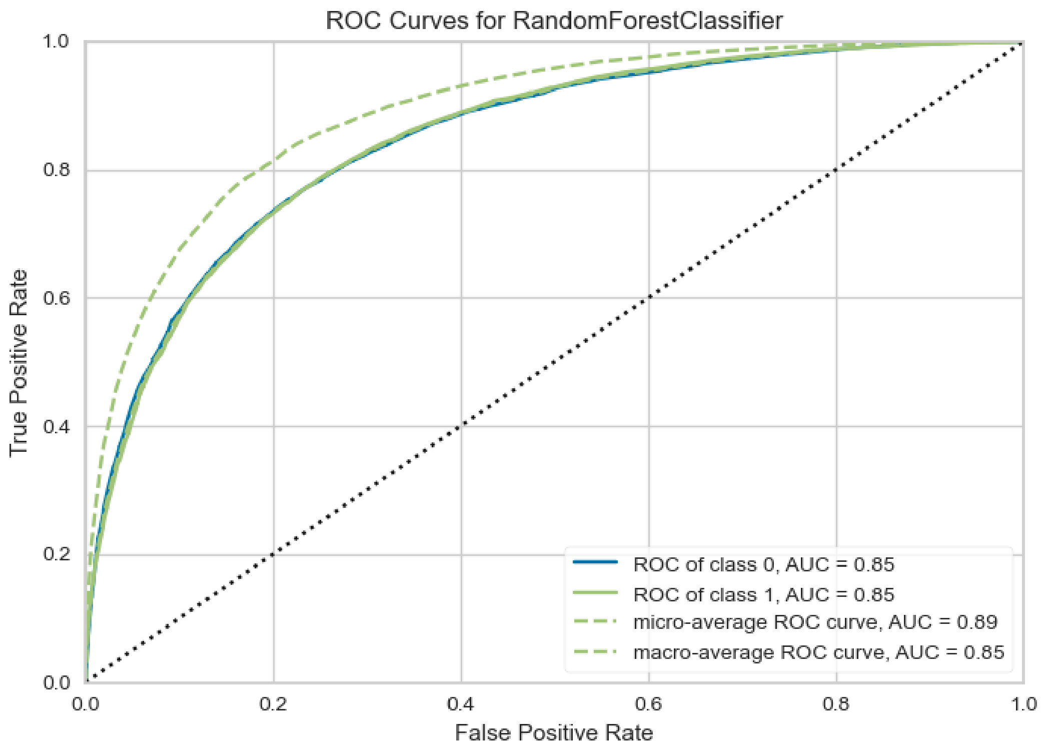


Figure 21: Random Forest (Entropy) ROC Curve

5.8 Random Forest (Log Loss)

5.8.1 Random Forest (Log Loss) Confusion Matrix

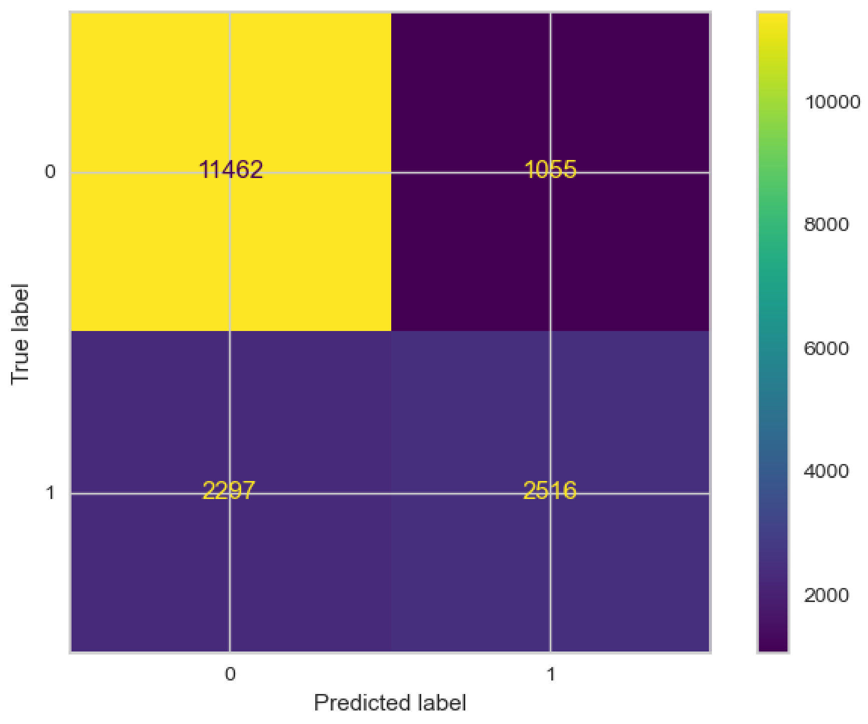


Figure 22: Random Forest (Log Loss) Confusion Matrix

5.8.2 Random Forest (Log Loss) Accuracy

```
Random Forest (Log Loss) Accuracy Score: 0.807
```

5.8.3 Random Forest (Log Loss) Classification Report

```
Random Forest (Log Loss) Classification Report:
              precision    recall  f1-score   support
           0       0.83      0.92      0.87     12517
           1       0.70      0.52      0.60      4813
    accuracy                           0.81     17330
   macro avg       0.77      0.72      0.74     17330
weighted avg       0.80      0.81      0.80     17330
```

5.8.4 Random Forest (Log Loss) Specificity

```
Random Forest (Log Loss) Specificity: 0.916
```

5.8.5 Random Forest (Log Loss) ROC Curve

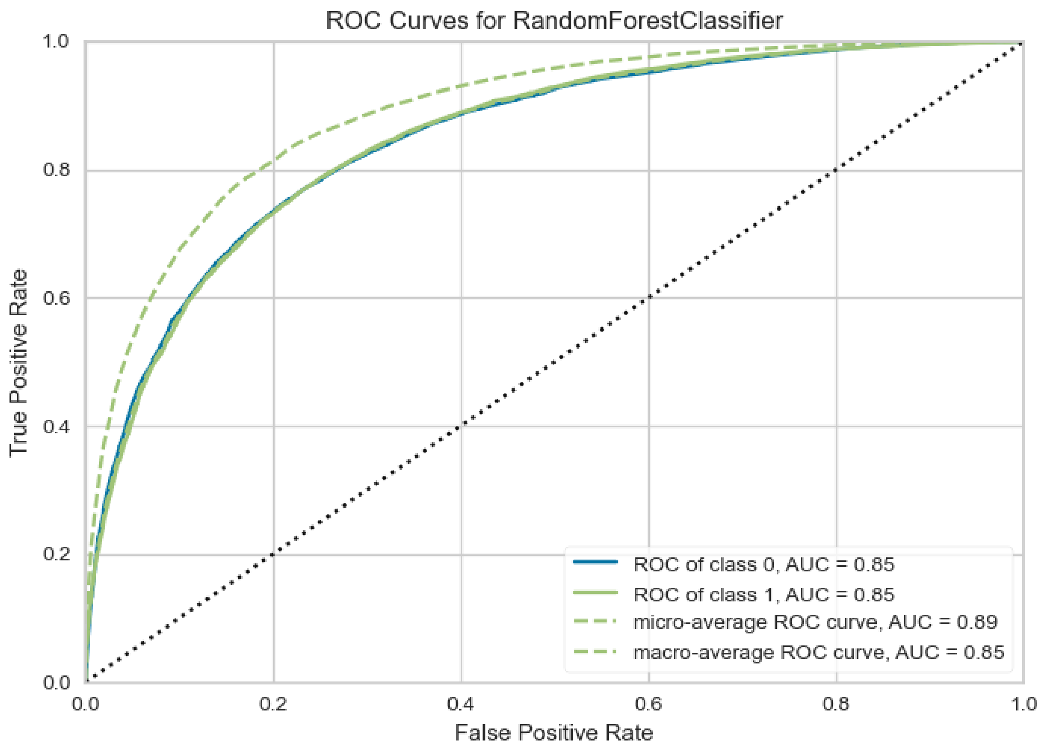


Figure 23: Random Forest (Log Loss) ROC Curve

5.9 Neural Network

5.9.1 Neural Network Confusion Matrix

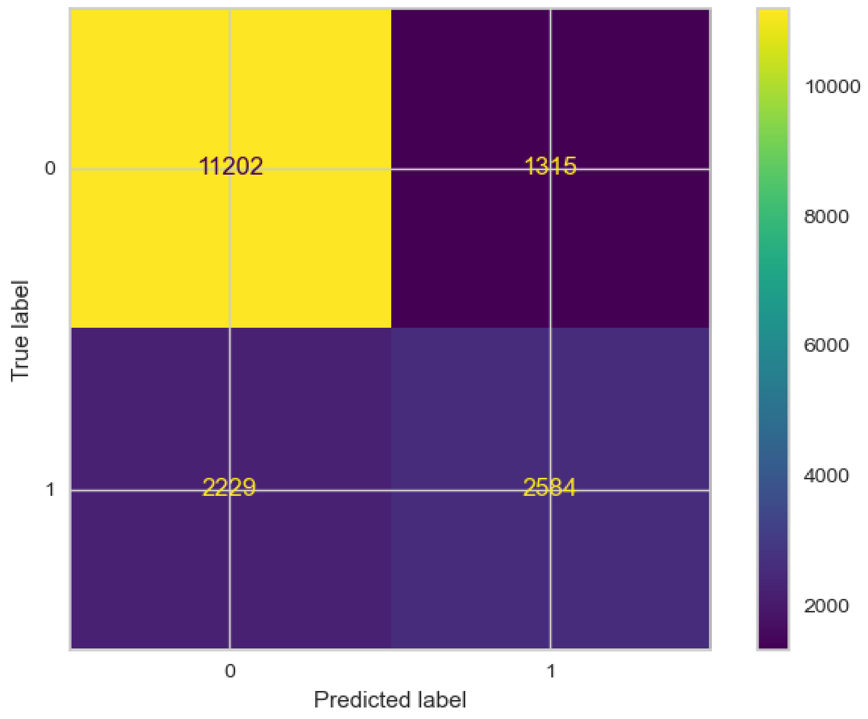


Figure 24: Neural Network Confusion Matrix

5.9.2 Neural Network Accuracy

```
    Neural Network Accuracy Score: 0.795
```

5.9.3 Neural Network Classification Report

```
Neural Network Classification Report:
              precision    recall  f1-score   support
           0       0.83      0.89      0.86     12517
           1       0.66      0.54      0.59      4813
    accuracy                           0.80     17330
   macro avg       0.75      0.72      0.73     17330
weighted avg       0.79      0.80      0.79     17330
```

5.9.4 Neural Network Specificity

```
Neural Network Specificity: 0.895
```

5.9.5 Neural Network ROC Curve

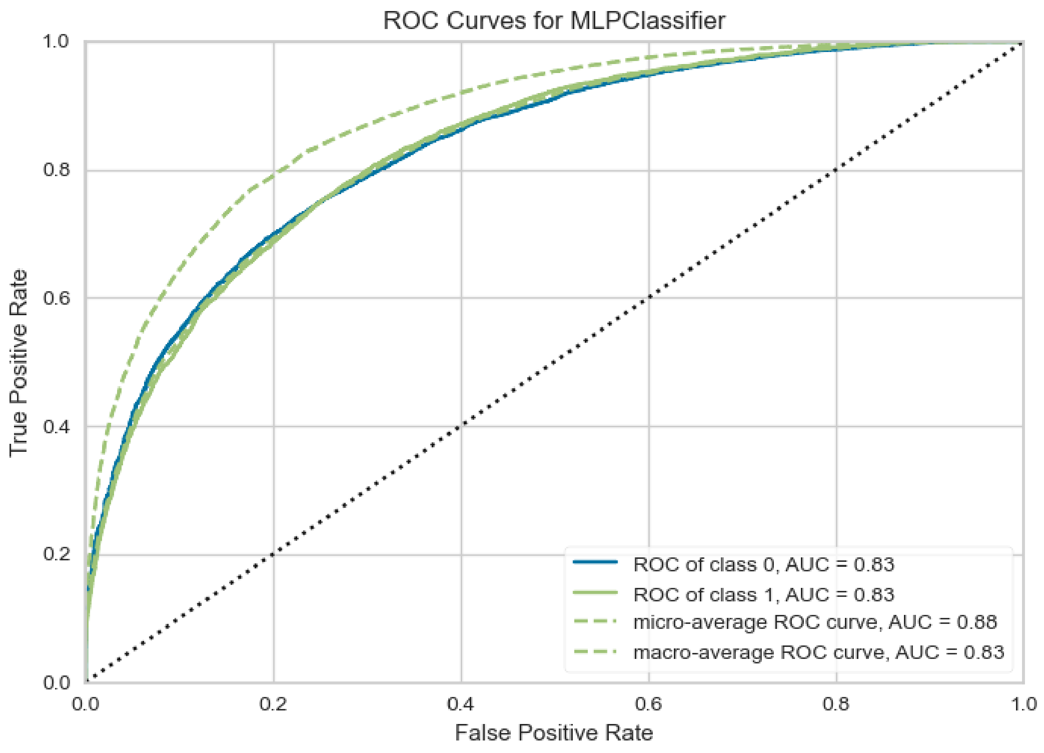


Figure 25: Neural Network ROC Curve

5.10 Performance Comparison

This section has a table of all the performances of the classifier that were mentioned in the previous section. The purpose of this section is to have a easy comparison table to see which classifier performed the best in which method.

	Decision Tree	**Logistic Regression**	**KNN**	**SVM**	**Naive Bayes**	**Random Forest**	**Random Forest (Entropy)**	**Random Forest (Log Loss)**	**Neural Network**
Confusion Matrix	10285 2232 2130 2683	11758 759 3416 1397	10668 1849 3303 1510	12487 30 4752 61	3783 8734 370 4443	11458 1059 2268 2545	11452 1055 2297 2516	11462 1055 2297 2516	11202 1315 2229 2584
Accuracy	74.8%	75.9%	70.3%	72.4%	47.5%	80.8%	80.7%	80.7%	79.5%
Precision	75%	74%	68%	71%	75%	80%	80%	80%	79%
Recall	75%	76%	70%	72%	47%	81%	81%	81%	80%
F1 Score	75%	72%	68%	61%	46%	80%	80%	80%	79%
Specificity	82.2%	93.9%	85.2%	99.8%	30.2%	91.5%	91.6%	91.6%	89.5%

Table 2: Classifier Performance Comparison Table

For accuracy, the Random Forest (Gini) outperformed all the other classifiers with an 80.8%. However, the Random Forest (Entropy) and Random Forest (Log Loss) was almost the same as they were both only 0.1% shy at 80.7%. The runner-up to these was the Neural Network at 79.5%.

For precision, the three Random Forest classifiers performed the highest at 80%. The runner-up was the Neural Network at 79%.

For the Recall, the three Random Forest classifiers performed the highest at 81%. The runner-up was the Neural Network at 80%.

For the F1 score, the three Random Forest classifiers performed the highest at 80%. The runner-up was the Neural Network at 79%.

For specificity, the SVM outperformed the rest with a 99.8%. The Naive Bayes did the worst at 30.2%. This is the worst performance statistic across all classifiers and performance evaluators.
From this observation, we can see that the Random Forest (Gini) usually performed the best. It does have the highest accuracy and as most people consider the accuracy of making a prediction one of the most important statistics, we could go ahead and say that the Random Forest (Gini) is the most suitable classifier to make predictions to the dataset used in this project.

One thing to notice is that the Neural Network also performed with statistics almost the same as the Random Forest (Gini).

6 Contribution and Conclusion

6.1 Acquired Knowledge

From this project, I have learned a couple of things. I think one of the takeaways from this project is that there is not a single classifier that is ideal. Even though the Random Forest classifier worked best for my project, I am sure that there could be multiple factors behind this. I definitely think the way I preprocessed my data affected the performance of all the classifiers and it turned out that the Random Forest classifier happened to work best with it. Even with the same dataset, I think the way you preprocess it will greatly influence your end result. I probably have replaced, removed, and aggregated features different from what others have from this dataset. I also believe that with all the different parameter tunings that can be done to classifiers, even through grid search, it is extremely difficult to find the ideal case for a classifier. One of the big takeaways I got from this project is that data preprocessing is as important or even more important than regression analysis and running classifiers. Another lesson I learned is that classifiers are extremely delicate and changing tuning can bring out drastically different performance results.

6.2 Best Performing Classifiers

In section 5 Classification Analysis, various machine learning classifiers were taken into application and their performances were taken. A summary of their results Performance Comparison Table can be found here. From the table, we can infer the following:

- Classifier with Highest Accuracy: Random Forest (Gini)
- Classifier with Highest Precision: Random Forest (All Three), Neural Network
- Classifier with Highest Recall: Random Forest (All Three)
- Classifier with Highest F1-Score: Random Forest (All Three)
- Classifier with Highest Specificity: SVM

Overall, the Random Forest classifier performed the best. The runner-up was the Neural Network classifier.

6.3 Improving Classification Performance

I believe there are multiple ways to improve the performance of all the classifiers that were used in this project. The ones that I could think of are the following

1. Reduce Dimensionality
2. Collect More Data
3. Algorithm Tuning

4. Ensemble Learning

The first one is reducing the dimensionality of the dataset even more. Even though I have reduced the number of features in my data preprocessing step, there could be features I may have misinterpreted and either overvalued its significance. I remember for the type of reserved rooms, there were about 5+ different types of rooms. I only removed two of them as they showed a high standard error compared to the other types of rooms. I believe further analysis into column values as such would have the possibility of increasing the performance of the classifiers.

The second way is collecting more data. Having more data entries may help in building a more accurate model. This does not necessarily mean increasing the dimensionality. In fact, we probably want to avoid increasing the dimensionality. I believe with more relevant data handled with the right preprocessing procedures would increase the performances of the classifiers.

The third method is algorithm tuning. As we know, there are multiple ways of tuning classifiers. Just for the Random Forest classifier, you can set different tuning settings such as the different number of trees in the forest, maximum number of features, maximum number of leaves etc. Through different combinations of tuning parameters, we may be able to increase our classifier performances. As we saw in our performance results, the Neural Network classifier was not too behind from the statistics the Random Forest classifier provided. I believe by the right algorithm tuning, the Neural Network may surpass the performance of the Random Forest classifier.

And finally, we could use ensemble learning such as bagging and boosting. I was not informed enough about ensemble learning so I could not apply it to this final project. From my research, bagging could reduce prediction variance while boosting would be able to minimize training errors. I think applying these would have improved the performance of our models in a way.

7 Reference

This section includes all the references used to write the final term project report and the presentation slides for the final term project.

`Code Used in Homework, Class of CS 5525 Data Analytics`
`https://www.youtube.com/watch?v=Nd76acrM7XU`
`https://travel.state.gov/content/travel/en/us-visas/visa-information-resources/fees/coun`
`https://www.kaggle.com/datasets/jessemostipak/hotel-booking-demand`
`https://www.youtube.com/watch?v=Nd76acrM7XU`
`https://www.youtube.com/watch?app=desktop&v=O2L2Uv9pdDA`
`https://mljar.com/machine-learning/catboost-vs-random-forest/`
`https://www.iconfinder.com/icons/4530480/bill_business_hotel_invoice_tax_icon`
`https://www.sciencedirect.com/science/article/pii/S2352340918315191`
`https://scikit-learn.org/stable/index.html`
`https://www.liveabout.com/alternate-tuning-guide-1711764`
`https://www.flaticon.com/free-icon/travel_5219574`
`https://www.flaticon.com/free-icon/hotel_5900195`
`https://pandas.pydata.org/docs/reference/api/pandas.crosstab.html`
`https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/`
`https://pub.towardsai.net/covariance-matrix-visualization-using-seaborns-heatmap-plot`
`-64332b6c90c5`
`https://www.geeksforgeeks.org/how-to-perform-an-f-test-in-python/`
`https://www.geeksforgeeks.org/how-to-calculate-confidence-intervals-in-python/`
`https://towardsdatascience.com/stepwise-regression-tutorial-in-python-ebf7c782c922`

8 Appendix

This section includes all the code written in Python 3 that was run to obtain the results in this report. To view the code online, visit Hotel Booking Demand Github Repository.

8.1 Read Me

This section includes the content of the **readme.txt**. It describes the required python files to obtain the results presented in the final project report along with instructions on how to run each python file.
There are three python files for this project:

- phase1.py
- phase2.py
- phase3.py

Each python file is responsible for each phase of the final project report.

8.1.1 How to Run Phase1.py

There are no particular instructions to run phase1.py. All you have to do is run it to generate all the plots and the data preprocessed dataset.

8.1.2 How to Run Phase2.py

This python file includes the code from phase1.py that obtains the data preprocessed dataset.

8.1.3 How to Run Phase3.py

This python file includes the code from phase1.py and phase2.py that obtains the data preprocessed dataset.
Every classifier is defined as a function. At the very end of the code, you can comment the classifier you want do not want to run. All classifiers will run at once on default. When you run the individual classifier functions, you will get the following results:

- Confusion Matrix
- Accuracy Score
- Classification Report
- Specificity
- ROC Curve

8.2 Code

This section includes all code written in Python 3 that was run to obtain the results in this report with comments.

8.2.1 phase1.py

```
# general imports
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# prevent warnings import
import warnings
warnings.filterwarnings("ignore")

# custom functions
## sample covariance matrix function
### the following function generates a heatmap
### of the sample covariance matrix
def covariance_matrix(dataset):
    plt.figure(figsize=(24,12))
    sns.set(font_scale=1.5)
    corr = dataset.corr()
    sns.heatmap(corr,
                cbar=True,
                cmap='RdBu_r',
                annot=True,
                linewidth=1,
                xticklabels=corr.columns,
                yticklabels=corr.columns,
                )
    plt.title("Covariance Matrix")
    plt.rc('axes', titlesize=50)
    plt.tight_layout()
    plt.show()

## function to bin each country to their continent
### the following function performs binning
### from country to their respective continent
def country_to_continent(country):
    if country in africa:
        return 'Africa'
    elif country in asia:
        return 'Asia'
    elif country in australia:
        return 'Australia'
    if country in europe:
        return 'Europe'
    elif country in north_america:
        return 'North America'
    elif country in south_america:
        return 'South America'
    elif country in unk:
        return 'Unknown'
    elif country in Others:
        return 'Others'

# import data
original_csv =
'https://raw.githubusercontent.com/mc811mc/hotel-booking-demand/main/dataset/hotel_bookings.csv'

data = pd.read_csv(original_csv)
df = data.copy()

# prints out information used to create the dataset explanation table in the final project report
print(df.info())

# plots the heatmap for the correlation matrix of the preprocessed dataset
plt.figure(figsize=(40, 40))
sns.heatmap(df.corr(), cmap='crest')
plt.title('Correlation Matrix')
plt.xticks(rotation=60,ha='right')
plt.show()

#################
# data cleaning #
#################

# check dataset for missing values
print(f'Number of Missing Values per Column Before Data Cleaning:
\n{df.isnull().sum()[df.isnull().sum() > 0]}\n')

# fill missing values in the children column with '0'
df.children.fillna(0,inplace=True)

# fill missing values in the country column with 'Unknown'
df.country.fillna('Unknown',inplace=True)

# replace agent ids (numerical) values in the agent column with 'Agent' (categorical)
df.loc[df.agent.isnull() == False,'agent'] = 'Agent'

# fill null values in the agent column with 'No agent'
df.agent.fillna('No agent',inplace = True)

# put in 'Corporate' in the company column
# if market segment and distribution channel column is both 'Corporate'
df.loc[((df.market_segment == 'Corporate') | (df.distribution_channel == 'Corporate'))
& (df.company.isnull()), 'company'] ='Corporate'

# replace non-null values in company column as 'Corporate'
df.loc[df.company.isnull() == False,'company'] = 'Corporate'

# fill in remaining and missing values in the company column with 'Individuals'
```

```
df.company.fillna('Individuals',inplace=True)

# correct the data type of the column arrival date year
df.arrival_date_year = df.arrival_date_year.astype(object)

# drop all duplicate rows in the dataset
df.drop_duplicates(inplace = True)

# check dataset for missing values
print(f'Number of Missing Values per Column After Data Cleaning:
\n{df.isnull().sum()[df.isnull().sum() > 0]}\n')

# this section handles binning countries into continents
# within the country of origin column.
# each country is categorized to one of the following: africa, asia, australia, europe,
north america, south america,
# unknown, or others
## check the number of countries before binning to respective continents
print(f'Number of Continents Before Binning: {df.country.unique().size}')

## binning country column to its respective continents
africa = ['MOZ','BWA','MAR','ZAF','AGO','ZMB','ZWE','DZA','TUN','CAF',
'NGA','SEN','SYC','CMR','MUS','COM','UGA','CIV',
       'BDI','EGY','MWI','MDG','TGO','DJI','STP','
       ETH','RWA','BEN','TZA','GHA','KEN','GNB','BFA','LBY','MLI','NAM',
       'MRT','SDN','SLE']

asia = ['OMN','CN','IND','CHN','ISR','KOR','ARE','HKG','IRN','CYP','KWT','MDV',
'KAZ','PAK','IDN','LBN','PHL','AZE','BHR',
     'THA','MYS','ARM','JPN','LKA','JOR','SYR','SGP','SAU','VNM','QAT','UZB',
     'NPL','MAC','TWN','IRQ','KHM','BGD','TJK',
     'TMP','MMR','LAO']

australia = ['AUS']

europe = ['PRT','GBR','ESP','IRL','FRA','ROU','NOR','POL','DEU','BEL','CHE',
'GRC','ITA','NLD','DNK','RUS','SWE','EST',
       'CZE','FIN','LUX','SVN','ALB','UKR','SMR','LVA','SRB','AUT','BLR',
       'LTU','TUR','HUN','HRV','GEO','AND','SVK',
       'MKD','BIH','BGR','MLT','ISL','MCO','LIE','MNE']

north_america = ['USA', 'MEX', 'PRI', 'CRI', 'CUB', 'HND', 'NIC', 'GAB', 'PAN', 'SLV', 'GTM']

south_america = ['ARG', 'BRA', 'CHL', 'URY', 'COL', 'VEN', 'SUR', 'PER', 'ECU', 'BOL', 'PRY', 'GUY']

unk = ['Unknown']

Others = ['CYM','CPV','JAM','GIB','JEY','GGY','FJI','NZL','DOM',
'PLW','BHS','KNA','IMN','VGB','GLP','UMI','MYT','FRO',
       'BRB','ABW','AIA','DMA','PYF','LCA','ATA','ASM','NCL','KIR','ATF']

# modify country column to make only continents appear
```

```
df.country = df.country.apply(country_to_continent)

# check to see if there are only continents
print(f'Number of Continents After Binning: {df.country.unique().size}')

# this section makes plots about the dataset
plt.rc('axes', titlesize=40)
plt.rc('axes', labelsize=30)

## plot hotel type booking ratio
plt.rcParams['figure.figsize'] = [25,25]
plt.subplot(2, 1, 1)
plt.pie(df.hotel.value_counts().values,
        labels=df.hotel.value_counts().index,
        autopct='%.2f%%',
        explode=[0, 0.1])
plt.title('Distribution of the type of Hotels')

## plot cancellation rate
plt.subplot(2, 1, 2)
crosstab_table=pd.crosstab(df.hotel, df.is_canceled, margins=True)
crosstab_table['cancel_percent']=crosstab_table[1] * 100 / crosstab_table['All']
crosstab_table.drop('All', axis = 0)['cancel_percent'].plot.bar()
plt.title('Cancellation Rate by Hotel Type')
plt.xlabel('Hotel Type')
plt.ylabel('Cancellation %')
plt.show()

## plot arrival date month
plt.rcParams['figure.figsize']=[40, 40]
plt.subplot(2, 1, 1)
sns.countplot(x='arrival_date_month',
              data=df,
              order=df.arrival_date_month.value_counts().index)
plt.title('Number of Bookings by Month')
plt.xlabel('Booking Month')
plt.ylabel('Number of Bookings')
plt.xticks(rotation=60,ha='right')

plt.subplot(2, 1, 2)
sns.countplot(x='arrival_date_month',
              data=df,
              order=df.arrival_date_month.value_counts().index,
              hue='hotel')
plt.title('Number of Bookings by Month and Hotel Type')
plt.xlabel('Booking Month')
plt.ylabel('Number of Bookings')
plt.xticks(rotation=60,ha='right')
plt.show()

## plot cancellation by arrival date month
plt.figure(figsize=(25, 25))
```

```
crosstab_table=pd.crosstab(df.arrival_date_month, df.is_canceled, margins=True)
crosstab_table['cancel_percent']= crosstab_table[1] * 100 / crosstab_table['All']
crosstab_table.drop('All', axis=0)['cancel_percent'].plot.bar()
plt.title('Cancellation Percentage by Month')
plt.xlabel('Arrival Month')
plt.ylabel('Cancellation %')
plt.xticks(rotation=60,ha='right')
plt.show()

## plot booking made by continent
plt.figure(figsize=(40, 40))
plt.subplot(2, 1, 1)
sns.countplot(x='country',data=df)
plt.title('Booking by Continent')
plt.xlabel('Continent')
plt.ylabel('Number of Bookings')
plt.xticks(rotation=60,ha='right')

## plot cancellation per continent
plt.subplot(2, 1, 2)
crosstab_table=pd.crosstab(df.country, df.is_canceled, margins=True)
crosstab_table['cancel_percent']= crosstab_table[1] * 100 / crosstab_table['All']
crosstab_table.drop('All', axis=0)['cancel_percent'].plot.bar()
plt.title('Cancellation Percentage by Continent')
plt.xlabel('Continent')
plt.ylabel('Cancellation %')
plt.xticks(rotation=60,ha='right')
plt.show()

## plot type of room reserved
plt.figure(figsize=(25, 25))
plt.subplot(2, 1, 1)
sns.countplot(x='reserved_room_type',data=df)
plt.title('Bookings by Reserved Room Type')
plt.xlabel('Reserved Room Type')
plt.ylabel('Number of Reserved Rooms')

## plot cancellation rate by type of room
plt.subplot(2, 1, 2)
crosstab_table=pd.crosstab(df.reserved_room_type,df.is_canceled,margins=True)
crosstab_table['cancel_percent']=crosstab_table[1]*100/crosstab_table['All']
crosstab_table.drop('All',axis=0)['cancel_percent'].plot.bar()
plt.title('Cancellation Percentage by Room Type')
plt.xlabel('Reserved Room Type')
plt.ylabel('Cancellation %')
plt.show()

## plot deposit type
plt.figure(figsize=(25, 25))
plt.subplot(2, 1, 1)
sns.countplot(x='deposit_type',data=df)
plt.title('Bookings by Deposit Type')
plt.xlabel('Deposit Type')
plt.ylabel('Total Number of Each Deposit Type')

## plot cancellation rate by deposit type
plt.subplot(2, 1, 2)
crosstab_table=pd.crosstab(df.deposit_type,df.is_canceled,margins=True)
crosstab_table['cancel_percent']=crosstab_table[1]*100/crosstab_table['All']
crosstab_table.drop('All',axis=0)['cancel_percent'].plot.bar()
plt.title('Cancellation Percentage by Deposit Type')
plt.xlabel('Deposit Type')
plt.ylabel('Cancellation %')
plt.xticks(rotation=60,ha='right')
plt.show()

# plot sample covariance matrix
covariance_matrix(df)

# choosing variables
correlation = df.corr()['is_canceled'].abs().sort_values(ascending = True)
print("Printing out Correlation")
print()
print(correlation)

# dimensionality reduction
## create total_members column that includes children, babies, and adults
df['total_members'] = df.children + df.babies + df.adults

## create total nights column that includes stays in week nights and stays in weekend nights columns
df['total_nights'] = df.stays_in_week_nights + df.stays_in_weekend_nights

## drop the aggregated columns in the prior steps
df.drop(columns=['children', 'babies', 'adults',
'stays_in_week_nights', 'stays_in_weekend_nights'], inplace=True)

## drop irrelevant columns to improve model
df.drop(columns=['days_in_waiting_list', 'reservation_status','reservation_status_date'],inplace=True)

# feature encoding
df['hotel'] = df['hotel'].map({'Resort Hotel' : 0, 'City Hotel' : 1})

df['meal'] = df['meal'].map({'BB' : 0, 'FB': 1, 'HB': 2, 'SC': 3, 'Undefined': 4})

df['reserved_room_type'] = df['reserved_room_type'].map({'C': 0, 'A': 1, 'D':
2, 'E': 3, 'G': 4, 'F': 5, 'H': 6, 'L': 7, 'P': 8, 'B': 9})
df['deposit_type'] = df['deposit_type'].map({'No Deposit': 0, 'Refundable': 1, 'Non Refund': 3})

# feature selection
## get rid of rows that have no guests
df_temp=df.loc[~((df.total_members==0) & (df.is_canceled==0))]

## get rid of rows that have no number of nights stayed
df_temp=df_temp.loc[~((df_temp.total_nights==0) & (df_temp.is_canceled==0))]
```

```
df_preprocessed = pd.get_dummies(df_temp, drop_first=True)

# attribute variables
X = df_preprocessed.drop(columns='is_canceled')

# target variable
y = df_preprocessed.is_canceled
```

8.2.2 phase2.py

```
# general imports
import numpy as np
import pandas as pd

from sklearn import linear_model
from sklearn.model_selection import train_test_split

# regression analysis imports
import scipy.stats as st
from sklearn import metrics
import statsmodels.api as sm

# prevent warnings import
import warnings
warnings.filterwarnings("ignore")

# custom functions
## function to bin each country to their continent
### the following function performs binning
### from country to their respective continent
def country_to_continent(country):
    if country in africa:
        return 'Africa'
    elif country in asia:
        return 'Asia'
    elif country in australia:
        return 'Australia'
    if country in europe:
        return 'Europe'
    elif country in north_america:
        return 'North America'
    elif country in south_america:
        return 'South America'
    elif country in unk:
        return 'Unknown'
    elif country in Others:
        return 'Others'

# import data
original_csv =
'https://raw.githubusercontent.com/mc811mc/hotel-booking-demand/main/dataset/hotel_bookings.csv'
```

```
data = pd.read_csv(original_csv)
df = data.copy()

#################
# data cleaning #
#################

# fill missing values in the children column with '0'
df.children.fillna(0,inplace=True)

# fill missing values in the country column with 'Unknown'
df.country.fillna('Unknown',inplace=True)

# replace agent ids (numerical) values in the agent column with 'Agent' (categorical)
df.loc[df.agent.isnull() == False,'agent'] = 'Agent'

# fill null values in the agent column with 'No agent'
df.agent.fillna('No agent',inplace = True)

# put in 'Corporate' in the company column
# if market segment and distribution channel column is both 'Corporate'
df.loc[((df.market_segment == 'Corporate') | (df.distribution_channel == 'Corporate'))
& (df.company.isnull()), 'company'] ='Corporate'

# replace non-null values in company column as 'Corporate'
df.loc[df.company.isnull() == False,'company'] = 'Corporate'

# fill in remaining and missing values in the company column with 'Individuals'
df.company.fillna('Individuals',inplace=True)

# correct the data type of the column arrival date year
df.arrival_date_year = df.arrival_date_year.astype(object)

# drop all duplicate rows in the dataset
df.drop_duplicates(inplace = True)

# this section handles binning countries into continents
# within the country of origin column.
# each country is categorized to one of the following: africa, asia, australia,
europe, north america, south america,
# unknown, or others

## binning country column to its respective continents
africa = ['MOZ','BWA','MAR','ZAF','AGO','ZMB','ZWE','DZA','TUN','CAF','NGA',
'SEN','SYC','CMR','MUS','COM','UGA','CIV',
       'BDI','EGY','MWI','MDG','TGO','DJI','STP','ETH','RWA','BEN','TZA',
       'GHA','KEN','GNB','BFA','LBY','MLI','NAM',
       'MRT','SDN','SLE']

asia = ['OMN','CN','IND','CHN','ISR','KOR','ARE','HKG','IRN','CYP',
'KWT','MDV','KAZ','PAK','IDN','LBN','PHL','AZE','BHR',
     'THA','MYS','ARM','JPN','LKA','JOR','SYR','SGP','SAU','VNM',
     'QAT','UZB','NPL','MAC','TWN','IRQ','KHM','BGD','TJK',
     'TMP','MMR','LAO']

australia = ['AUS']

europe = ['PRT','GBR','ESP','IRL','FRA','ROU','NOR','POL','DEU','BEL',
'CHE','GRC','ITA','NLD','DNK','RUS','SWE','EST',
       'CZE','FIN','LUX','SVN','ALB','UKR','SMR','LVA',
       'SRB','AUT','BLR','LTU','TUR','HUN','HRV','GEO','AND','SVK',
       'MKD','BIH','BGR','MLT','ISL','MCO','LIE','MNE']

north_america = ['USA', 'MEX', 'PRI', 'CRI', 'CUB', 'HND',
'NIC', 'GAB', 'PAN', 'SLV', 'GTM']

south_america = ['ARG', 'BRA', 'CHL', 'URY', 'COL', 'VEN',
'SUR', 'PER', 'ECU', 'BOL', 'PRY', 'GUY']

unk = ['Unknown']

Others = ['CYM','CPV','JAM','GIB','JEY','GGY','FJI','NZL','DOM',
'PLW','BHS','KNA','IMN','VGB','GLP','UMI','MYT','FRO',
       'BRB','ABW','AIA','DMA','PYF','LCA','ATA','ASM','NCL','KIR','ATF']

# modify country column to make only continents appear
df.country = df.country.apply(country_to_continent)

# dimensionality reduction
## create total_members column that includes children, babies, and adults
df['total_members'] = df.children + df.babies + df.adults

## create total nights column that includes stays in week nights and stays in weekend nights columns
df['total_nights'] = df.stays_in_week_nights + df.stays_in_weekend_nights

## drop the aggregated columns in the prior steps
df.drop(columns=['children', 'babies', 'adults',
'stays_in_week_nights', 'stays_in_weekend_nights'], inplace=True)

## drop irrelevant columns to improve model
df.drop(columns=['days_in_waiting_list', 'reservation_status','reservation_status_date'],inplace=True)

# feature encoding
df['hotel'] = df['hotel'].map({'Resort Hotel' : 0, 'City Hotel' : 1})

df['meal'] = df['meal'].map({'BB' : 0, 'FB': 1, 'HB': 2, 'SC': 3, 'Undefined': 4})

df['reserved_room_type'] = df['reserved_room_type'].map({'C': 0, 'A': 1, 'D': 2, 'E': 3,
'G': 4, 'F': 5, 'H': 6, 'L': 7, 'P': 8, 'B': 9})
df['deposit_type'] = df['deposit_type'].map({'No Deposit': 0, 'Refundable': 1, 'Non Refund': 3})

# features to be dropped through regression analysis
df.drop(columns=['country', 'arrival_date_year', 'market_segment', 'agent', 'company'],inplace=True)
df.drop(df.loc[df['assigned_room_type']=='L'].index, inplace=True)
df.drop(df.loc[df['assigned_room_type']=='P'].index, inplace=True)
df.drop(df.loc[df['distribution_channel']=='Undefined'].index, inplace=True)

# feature selection
## get rid of rows that have no guests
df_temp=df.loc[~((df.total_members==0) & (df.is_canceled==0))]

## get rid of rows that have no number of nights stayed
df_temp=df_temp.loc[~((df_temp.total_nights==0) & (df_temp.is_canceled==0))]

df_preprocessed = pd.get_dummies(df_temp, drop_first=True)

# attribute variables
X = df_preprocessed.drop(columns='is_canceled')

# target variable
y = df_preprocessed.is_canceled

# split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# association analysis, f-test analysis, stepwise regression analysis, collinearity analysis
## perform ols
model_ols = sm.OLS(y_train, X_train).fit()
print(model_ols.summary())
y_pred_ols = model_ols.predict(X_test)

print(f'Mean Absolute Error: {metrics.mean_squared_error(y_test, y_pred_ols)}')
```

8.2.3 phase3.py

```
# general imports
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

# classifier imports
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

# performance imports
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from yellowbrick.classifier import ROCAUC

# prevent warnings import
import warnings
warnings.filterwarnings("ignore")

# performance functions
## sensitivity (recall) function
def sensitivity(conf):
    return conf[1][1] / (conf[1][1] * conf[1][0])

## specificity function
def specificity(conf):
    return conf[0][0] / (conf[0][0] + conf[0][1])

# custom functions
## function to bin each country to their continent
### the following function performs binning
### from country to their respective continent
def country_to_continent(country):
    if country in africa:
        return 'Africa'
    elif country in asia:
        return 'Asia'
    elif country in australia:
        return 'Australia'
    if country in europe:
        return 'Europe'
    elif country in north_america:
        return 'North America'
    elif country in south_america:
        return 'South America'
    elif country in unk:
        return 'Unknown'
    elif country in Others:
        return 'Others'

# import data
original_csv =
'https://raw.githubusercontent.com/mc811mc/hotel-booking-demand/main/dataset/hotel_bookings.csv'

data = pd.read_csv(original_csv)
df = data.copy()

#################
# data cleaning #
#################

# fill missing values in the children column with '0'
df.children.fillna(0,inplace=True)

# fill missing values in the country column with 'Unknown'
```

```
df.country.fillna('Unknown',inplace=True)

# replace agent ids (numerical) values in the agent column with 'Agent' (categorical)
df.loc[df.agent.isnull() == False,'agent'] = 'Agent'

# fill null values in the agent column with 'No agent'
df.agent.fillna('No agent',inplace = True)

# put in 'Corporate' in the company column
# if market segment and distribution channel column is both 'Corporate'
df.loc[((df.market_segment == 'Corporate') | (df.distribution_channel == 'Corporate'))
& (df.company.isnull()), 'company'] ='Corporate'

# replace non-null values in company column as 'Corporate'
df.loc[df.company.isnull() == False,'company'] = 'Corporate'

# fill in remaining and missing values in the company column with 'Individuals'
df.company.fillna('Individuals',inplace=True)

# correct the data type of the column arrival date year
df.arrival_date_year = df.arrival_date_year.astype(object)

# drop all duplicate rows in the dataset
df.drop_duplicates(inplace = True)

# this section handles binning countries into continents
# within the country of origin column.
# each country is categorized to one of the following: africa,
asia, australia, europe, north america, south america,
# unknown, or others

## binning country column to its respective continents
africa = ['MOZ','BWA','MAR','ZAF','AGO','ZMB','ZWE','DZA','TUN',
'CAF','NGA','SEN','SYC','CMR','MUS','COM','UGA','CIV',
       'BDI','EGY','MWI','MDG','TGO','DJI','STP','ETH','RWA',
       'BEN','TZA','GHA','KEN','GNB','BFA','LBY','MLI','NAM',
       'MRT','SDN','SLE']

asia = ['OMN','CN','IND','CHN','ISR','KOR','ARE','HKG','IRN','CYP',
'KWT','MDV','KAZ','PAK','IDN','LBN','PHL','AZE','BHR',
     'THA','MYS','ARM','JPN','LKA','JOR','SYR','SGP','SAU','VNM',
     'QAT','UZB','NPL','MAC','TWN','IRQ','KHM','BGD','TJK',
     'TMP','MMR','LAO']

australia = ['AUS']

europe = ['PRT','GBR','ESP','IRL','FRA','ROU','NOR','POL','DEU','BEL',
'CHE','GRC','ITA','NLD','DNK','RUS','SWE','EST',
       'CZE','FIN','LUX','SVN','ALB','UKR','SMR','LVA','SRB','AUT',
       'BLR','LTU','TUR','HUN','HRV','GEO','AND','SVK',
       'MKD','BIH','BGR','MLT','ISL','MCO','LIE','MNE']
```

```
north_america = ['USA', 'MEX', 'PRI', 'CRI', 'CUB', 'HND',
'NIC', 'GAB', 'PAN', 'SLV', 'GTM']

south_america = ['ARG', 'BRA', 'CHL', 'URY', 'COL', 'VEN',
'SUR', 'PER', 'ECU', 'BOL', 'PRY', 'GUY']

unk = ['Unknown']

Others = ['CYM','CPV','JAM','GIB','JEY','GGY','FJI','NZL','DOM',
'PLW','BHS','KNA','IMN','VGB','GLP','UMI','MYT','FRO',
       'BRB','ABW','AIA','DMA','PYF','LCA','ATA','ASM','NCL','KIR','ATF']

# modify country column to make only continents appear
df.country = df.country.apply(country_to_continent)

# dimensionality reduction
## create total_members column that includes children, babies, and adults
df['total_members'] = df.children + df.babies + df.adults

## create total nights column that includes stays in week nights and stays in weekend nights columns
df['total_nights'] = df.stays_in_week_nights + df.stays_in_weekend_nights

## drop the aggregated columns in the prior steps
df.drop(columns=['children', 'babies', 'adults', 'stays_in_week_nights',
'stays_in_weekend_nights'], inplace=True)

## drop irrelevant columns to improve model
df.drop(columns=['days_in_waiting_list', 'reservation_status','reservation_status_date'],inplace=True)

# feature encoding
df['hotel'] = df['hotel'].map({'Resort Hotel' : 0, 'City Hotel' : 1})

df['meal'] = df['meal'].map({'BB' : 0, 'FB': 1, 'HB': 2, 'SC': 3, 'Undefined': 4})

df['reserved_room_type'] = df['reserved_room_type'].map({'C': 0, 'A': 1, 'D': 2,
'E': 3, 'G': 4, 'F': 5, 'H': 6,'L': 7, 'P': 8, 'B': 9})

df['deposit_type'] = df['deposit_type'].map({'No Deposit': 0, 'Refundable': 1, 'Non Refund': 3})

# features to be dropped through regression analysis
df.drop(columns=['country', 'arrival_date_year', 'market_segment', 'agent', 'company'],inplace=True)
df.drop(df.loc[df['assigned_room_type']=='L'].index, inplace=True)
df.drop(df.loc[df['assigned_room_type']=='P'].index, inplace=True)
df.drop(df.loc[df['distribution_channel']=='Undefined'].index, inplace=True)

# feature selection
## get rid of rows that have no guests
df_temp=df.loc[~((df.total_members==0) & (df.is_canceled==0))]

## get rid of rows that have no number of nights stayed
df_temp=df_temp.loc[~((df_temp.total_nights==0) & (df_temp.is_canceled==0))]
```

```
df_preprocessed = pd.get_dummies(df_temp, drop_first=True)

# attribute variables
X = df_preprocessed.drop(columns='is_canceled')

# target variable
y = df_preprocessed.is_canceled

# split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

#################
# decision tree #
#################
# decision tree initial case
def decision_tree_initial(X_train, X_test, y_train, y_test):
    tree_cls_no = DecisionTreeClassifier(random_state=0)
    tree_model_no = tree_cls_no.fit(X_train, y_train)

    y_pred_tree_no = tree_model_no.predict(X_test)

    tree_acc_no = accuracy_score(y_test, y_pred_tree_no)
    tree_conf_no = confusion_matrix(y_test, y_pred_tree_no)
    tree_clf_report_no = classification_report(y_test, y_pred_tree_no)

    # prints out the confusion matrix of the decision tree classifier
    print(f'Decision Tree (Initial) Confusion Matrix: \n{tree_conf_no}')
    disp = ConfusionMatrixDisplay(confusion_matrix=tree_conf_no,
                                  display_labels=tree_cls_no.classes_)
    disp.plot()
    plt.show()
    # prints out the accuracy score of the decision tree classifier
    print(f'Decision Tree (Initial) Accuracy Score: {tree_acc_no:.3f}')
    # prints out the precision, recall (sensitivity), and f1 score
    # of the decision tree classifier
    print(f'Decision Tree (Initial) Classification Report: \n{tree_clf_report_no}')
    # prints out the specificity of the decision tree classifier
    print(f'Decision Tree (Initial) Specificity: {specificity(tree_conf_no):.3f}')
    # prints out the ROC curve for the decision tree classifier
    ran_for_visualizer = ROCAUC(tree_cls_no)
    # fit the training data to the visualizer
    ran_for_visualizer.fit(X_train, y_train)
    # evaluate the model on the test data
    ran_for_visualizer.score(X_test, y_test)
    # display the visualizer
    ran_for_visualizer.show()

#######################
# logistic regression #
#######################
def logistic_regression(X_train, X_test, y_train, y_test):
    log_reg_cls = LogisticRegression(random_state=0)
    log_reg_model = log_reg_cls.fit(X_train, y_train)

    y_pred_log_reg = log_reg_model.predict(X_test)

    log_reg_acc = accuracy_score(y_test, y_pred_log_reg)
    log_reg_conf = confusion_matrix(y_test, y_pred_log_reg)
    log_reg_clf_report = classification_report(y_test, y_pred_log_reg)

    # prints out the confusion matrix of the logistic regression classifier
    print(f'Logistic Regression Confusion Matrix: \n{log_reg_conf}')
    disp = ConfusionMatrixDisplay(confusion_matrix=log_reg_conf,
                                  display_labels=log_reg_cls.classes_)
    disp.plot()
    plt.show()
    # prints out the accuracy score of the logistic regression classifier
    print(f'Logistic Regression Accuracy Score: {log_reg_acc:.3f}')
    # prints out the precision, recall (sensitivity), and f1 score
    # of the logistic regression classifier
    print(f'Logistic Regression Classification Report: \n{log_reg_clf_report}')
    # prints out the specificity of the logistic regression classifier
    print(f'Logistic Regression Specificity: {specificity(log_reg_conf):.3f}')
    # prints out the ROC curve for the logistic regression classifier
    ran_for_visualizer = ROCAUC(log_reg_cls)
    # fit the training data to the visualizer
    ran_for_visualizer.fit(X_train, y_train)
    # evaluate the model on the test data
    ran_for_visualizer.score(X_test, y_test)
    # display the visualizer
    ran_for_visualizer.show()

#######
# knn #
#######
# knn initial case
def knn_initial(X_train, X_test, y_train, y_test):
    knn_cls = KNeighborsClassifier()
    knn_model = knn_cls.fit(X_train, y_train)

    y_pred_knn = knn_model.predict(X_test)

    knn_acc = accuracy_score(y_test, y_pred_knn)
    knn_conf = confusion_matrix(y_test, y_pred_knn)
    knn_clf_report = classification_report(y_test, y_pred_knn)

    # prints out the confusion matrix of the knn classifier
    print(f'KNN (Default) Confusion Matrix: \n{knn_conf}')
    disp = ConfusionMatrixDisplay(confusion_matrix=knn_conf,
                                  display_labels=knn_cls.classes_)
    disp.plot()
    plt.show()
    # prints out the accuracy score of the knn classifier
    print(f'KNN (Default) Accuracy Score: {knn_acc:.3f}')
    # prints out the precision, recall (sensitivity), and f1 score of the knn classifier
    print(f'KNN (Default) Classification Report: \n{knn_clf_report}')
    # prints out the specificity of the knn classifier
    print(f'KNN (Default) Specificity: {specificity(knn_conf):.3f}')
    # prints out the ROC curve for the knn classifier
    ran_for_visualizer = ROCAUC(knn_cls)
    # fit the training data to the visualizer
    ran_for_visualizer.fit(X_train, y_train)
    # evaluate the model on the test data
    ran_for_visualizer.score(X_test, y_test)
    # display the visualizer
    ran_for_visualizer.show()

#######
# svm #
#######
def svm(X_train, X_test, y_train, y_test):
    svm_cls = SVC(random_state=0)
    svm_model = svm_cls.fit(X_train, y_train)

    y_pred_svm = svm_model.predict(X_test)

    svm_acc = accuracy_score(y_test, y_pred_svm)
    svm_conf = confusion_matrix(y_test, y_pred_svm)
    svm_clf_report = classification_report(y_test, y_pred_svm)

    # prints out the confusion matrix of the svm classifier
    print(f'SVM Confusion Matrix: \n{svm_conf}')
    disp = ConfusionMatrixDisplay(confusion_matrix=svm_conf,
                                  display_labels=svm_cls.classes_)
    disp.plot()
    plt.show()
    # prints out the accuracy score of the svm classifier
    print(f'SVM Accuracy Score: {svm_acc:.3f}')
    # prints out the precision, recall (sensitivity), and f1 score
    # of the svm classifier
    print(f'SVM Classification Report: \n{svm_clf_report}')
    # prints out the specificity of the svm classifier
    print(f'SVM Specificity: {specificity(svm_conf):.3f}')

###############
# naive bayes #
###############
def naive_bayes(X_train, X_test, y_train, y_test):
    gaussian_cls = GaussianNB()
    gaussian_model = gaussian_cls.fit(X_train, y_train)

    y_pred_gaussian = gaussian_model.predict(X_test)

    gaussian_acc = accuracy_score(y_test, y_pred_gaussian)
    gaussian_conf = confusion_matrix(y_test, y_pred_gaussian)
    gaussian_clf_report = classification_report(y_test, y_pred_gaussian)
    # prints out the confusion matrix of the naive bayes classifier
    print(f'Naive Bayes Confusion Matrix: \n{gaussian_conf}')
    disp = ConfusionMatrixDisplay(confusion_matrix=gaussian_conf,
                                  display_labels=gaussian_cls.classes_)
    disp.plot()
    plt.show()
    # prints out the accuracy score of the naive bayes classifier
    print(f'Naive Bayes Accuracy Score: {gaussian_acc:.3f}')
    # prints out the precision, recall (sensitivity), and f1 score
    # of the naive bayes classifier
    print(f'Naive Bayes Classification Report: \n{gaussian_clf_report}')
    # prints out the specificity of the naive bayes classifier
    print(f'Naive Bayes Specificity: {specificity(gaussian_conf):.3f}')
    # prints out the ROC curve for the naive bayes classifier
    ran_for_visualizer = ROCAUC(gaussian_cls)
    # fit the training data to the visualizer
    ran_for_visualizer.fit(X_train, y_train)
    # evaluate the model on the test data
    ran_for_visualizer.score(X_test, y_test)
    # display the visualizer
    ran_for_visualizer.show()


#################
# random forest #
#################
# random forest using gini (default)
def random_forest_default(X_train, X_test, y_train, y_test):
    ran_for_cls_gini = RandomForestClassifier(random_state=0, criterion='gini')
    ran_for_model_gini = ran_for_cls_gini.fit(X_train, y_train)

    y_pred_ran_for_gini = ran_for_model_gini.predict(X_test)

    ran_for_acc_gini = accuracy_score(y_test, y_pred_ran_for_gini)
    ran_for_conf_gini = confusion_matrix(y_test, y_pred_ran_for_gini)
    ran_for_clf_report_gini = classification_report(y_test, y_pred_ran_for_gini)

    # prints out the confusion matrix of the random forest classifier
    print(f'Random Forest Confusion Matrix: \n{ran_for_conf_gini}')
    disp = ConfusionMatrixDisplay(confusion_matrix=ran_for_conf_gini,
                                  display_labels=ran_for_cls_gini.classes_)
    disp.plot()
    plt.show()
    # prints out the accuracy score of the random forest classifier
    print(f'Random Forest Accuracy Score: {ran_for_acc_gini:.3f}')
    # prints out the precision, recall (sensitivity), and f1 score
    # of the random forest classifier
    print(f'Random Forest Classification Report: \n{ran_for_clf_report_gini}')
    # prints out the specificity of the random forest classifier
    print(f'Random Forest Specificity: {specificity(ran_for_conf_gini):.3f}')
    # prints out the ROC curve for the random forest classifier
    ran_for_visualizer = ROCAUC(ran_for_cls_gini)
    # fit the training data to the visualizer
    ran_for_visualizer.fit(X_train, y_train)
    # evaluate the model on the test data
    ran_for_visualizer.score(X_test, y_test)
    # display the visualizer
    ran_for_visualizer.show()

# random forest using entropy
def random_forest_entropy(X_train, X_test, y_train, y_test):
    ran_for_cls_entropy = RandomForestClassifier(random_state=0, criterion='entropy')
    ran_for_model_entropy = ran_for_cls_entropy.fit(X_train, y_train)

    y_pred_ran_for_entropy = ran_for_model_entropy.predict(X_test)

    ran_for_acc_entropy = accuracy_score(y_test, y_pred_ran_for_entropy)
    ran_for_conf_entropy = confusion_matrix(y_test, y_pred_ran_for_entropy)
    ran_for_clf_report_entropy = classification_report(y_test, y_pred_ran_for_entropy)

    # prints out the confusion matrix of the random forest classifier
    print(f'Random Forest (Entropy) Confusion Matrix: \n{ran_for_conf_entropy}')
    disp = ConfusionMatrixDisplay(confusion_matrix=ran_for_conf_entropy,
                                  display_labels=ran_for_cls_entropy.classes_)
    disp.plot()
    plt.show()
    # prints out the accuracy score of the random forest classifier
    print(f'Random Forest (Entropy) Accuracy Score: {ran_for_acc_entropy:.3f}')
    # prints out the precision, recall (sensitivity), and f1 score
    # of the random forest classifier
    print(f'Random Forest (Entropy) Classification Report: \n{ran_for_clf_report_entropy}')
    # prints out the specificity of the random forest classifier
    print(f'Random Forest (Entropy) Specificity: {specificity(ran_for_conf_entropy):.3f}')
    # prints out the ROC curve for the random forest classifier
    ran_for_visualizer = ROCAUC(ran_for_cls_entropy)
    # fit the training data to the visualizer
    ran_for_visualizer.fit(X_train, y_train)
    # evaluate the model on the test data
    ran_for_visualizer.score(X_test, y_test)
    # display the visualizer
    ran_for_visualizer.show()

# random forest using log loss
def random_forest_log_loss(X_train, X_test, y_train, y_test):
    ran_for_cls_log = RandomForestClassifier(random_state=0, criterion='log_loss')
    ran_for_model_log = ran_for_cls_log.fit(X_train, y_train)

    y_pred_ran_for_log = ran_for_model_log.predict(X_test)

    ran_for_acc_log = accuracy_score(y_test, y_pred_ran_for_log)
    ran_for_conf_log = confusion_matrix(y_test, y_pred_ran_for_log)
    ran_for_clf_report_log = classification_report(y_test, y_pred_ran_for_log)
    # prints out the confusion matrix of the random forest classifier
    print(f'Random Forest (Log Loss) Confusion Matrix: \n{ran_for_conf_log}')
    disp = ConfusionMatrixDisplay(confusion_matrix=ran_for_conf_log,
                                  display_labels=ran_for_cls_log.classes_)
    disp.plot()
    plt.show()
    # prints out the accuracy score of the random forest classifier
    print(f'Random Forest (Log Loss) Accuracy Score: {ran_for_acc_log:.3f}')
    # prints out the precision, recall (sensitivity), and f1 score
    # of the random forest classifier
    print(f'Random Forest (Log Loss) Classification Report: \n{ran_for_clf_report_log}')
    # prints out the specificity of the random forest classifier
    print(f'Random Forest (Log Loss) Specificity: {specificity(ran_for_conf_log):.3f}')
    # prints out the ROC curve for the random forest classifier
    ran_for_visualizer = ROCAUC(ran_for_cls_log)
    # fit the training data to the visualizer
    ran_for_visualizer.fit(X_train, y_train)
    # evaluate the model on the test data
    ran_for_visualizer.score(X_test, y_test)
    # display the visualizer
    ran_for_visualizer.show()

##################
# neural network #
##################
def neural_network(X_train, X_test, y_train, y_test):
    neural_cls = MLPClassifier(random_state=0)
    neural_model = neural_cls.fit(X_train, y_train)

    y_pred_neural = neural_model.predict(X_test)

    neural_acc = accuracy_score(y_test, y_pred_neural)
    neural_conf = confusion_matrix(y_test, y_pred_neural)
    neural_clf_report = classification_report(y_test, y_pred_neural)

    # prints out the confusion matrix of the neural network classifier
    print(f'Neural Network  Confusion Matrix: \n{neural_conf}')
    disp = ConfusionMatrixDisplay(confusion_matrix=neural_conf,
                                  display_labels=neural_cls.classes_)
    disp.plot()
    plt.show()
    # prints out the accuracy score of the neural network classifier
    print(f'Neural Network  Accuracy Score: {neural_acc:.3f}')
    # prints out the precision, recall (sensitivity), and f1 score
    # of the neural network classifier
    print(f'Neural Network Classification Report: \n{neural_clf_report}')
    # prints out the specificity of the neural network classifier
    print(f'Neural Network Specificity: {specificity(neural_conf):.3f}')
    # prints out the ROC curve for the neural network classifier
    ran_for_visualizer = ROCAUC(neural_cls)
    # fit the training data to the visualizer
    ran_for_visualizer.fit(X_train, y_train)
    # evaluate the model on the test data
    ran_for_visualizer.score(X_test, y_test)
    # display the visualizer
    ran_for_visualizer.show()

# run individual classifiers
decision_tree_initial(X_train, X_test, y_train, y_test)
logistic_regression(X_train, X_test, y_train, y_test)
knn_initial(X_train, X_test, y_train, y_test)
svm(X_train, X_test, y_train, y_test)
naive_bayes(X_train, X_test, y_train, y_test)
random_forest_default(X_train, X_test, y_train, y_test)
random_forest_entropy(X_train, X_test, y_train, y_test)
random_forest_log_loss(X_train, X_test, y_train, y_test)
neural_network(X_train, X_test, y_train, y_test)
```