NOVA IMS Information Management School

COMPANY IN FOCUS

BOOKME

Akylai Kulzhanova e20211850
Luis Silvano r20201479
Miguel Tomé r20201644
Pedro Sousa r20201611
Sila Sefer e20211875

BOOKME

Index

1. Introduction and Methodology

1.1 Introduction

The company BookMe is a firm that operates in the hospitality sector. They interact with their clients through their international website. Their main objective is to grant quality and conditions in their services (Comfort, Reception Schedule, Food and Drink Available, Accommodation Location, Wi-Fi Service, Accommodation Amenities, Staff, Online Booking, Price Quality Relationship, Room Space, Check-Out, Check-in, Cleanliness and Bar Service) to future traveller's. They recruited our team, for we predict which customers have a high probability of churn depending on their satisfaction.

1.2 Methodology

In order to predict the probability of a customer according to their satisfaction and behaviour, firstly our team explored the given dataset, analysing the characteristics of the clients, the levels of satisfaction and the information about the reservation. We observed the training set, to understand which values were in need to be handled (evaluate their structure, the datatypes of each variable, and recognized the missing values and outliers of each variable). Secondly, we start pre-processing the raw data in each dataset (treating missing values and outliers, transforming categorical data into numerical, creating new variables, evaluating and checking data coherence, applying Spearman and Pearson Correlation, ANOVA test, and comparing the importance to the models as a way to perform feature selection, and finally, scaling variables). Before moving on the third big part of the project we prepared both the validation dataset and the test dataset exactly the same as we did the train. Thirdly, we implemented different models in our project and test their assessment, comparing the train and validation data set, some of these models were Logistic Regression, K-Nearest Neighbours, Decision Trees, Neural Networks, Random Forest, Gradient Boosting and Naive Bayes, all regarding to classification problems, this find out which model is more competent to predict the probability of churn depending on customers satisfaction and behaviour. Finally, after having the idea of the best model, we created the column of Churn to the test dataset in order to get the final results whether a customer churn or not.

2. Exploration

In Exploration Criteria, our aim was to explore which values we had and which attributes we will concern from the training dataset (the first look of the dataset). We understand when we apply the code to get the shape of the table, we had 15589 clients and 22 attributes to work it (Data Types Table). After realizing, how is the structure of our dataset our team decided to read the types of each attribute, noticing some categorical data that would need to be treated in order to use some of the models we wanted to use. Next, we compared the frequency of missing values from the training data set each (Missing Values Table in Percentage). We notice that the variable Year of Birth had 195 missing values (or 1.25%) so theoretically we could futurely remove it. Then, we want to evaluate the BookMe clients from their room preferences, their age, if they are regular clients, their satisfaction to their Reward Points. From **longevity** we could conclude that most of the clients are regular clients and some mistakes in the dataset, since we had some 'y' answers (Longevity Histogram), in the **year of birth** the clients that uses more BookMe were born between 1960 and 2000, and therefore middle-aged adults, we also noticed that clients born between 1960 and 1980 are more likely to churn than others (Clients Age Histogram). We also notice a small preference for room single rooms, then doubles, and with very few suite (Room Type Histogram), and that most clients use BookMe to business travels (Type of Travel Histogram).

After understanding better our dataset, we did an individual analysis of each service with the rates given by clients. Starting from the service **Comfort Satisfaction Level** (Comfort Satisfaction Graph), if the rate of comfort is bigger or equal to four, we have more no-churn than churns. Meaning that, clients that rated their level of comfort with 4 or more don't tend to churn, whereas if the rate is less than 4, they usually churn from the company, so overall comfort level is a good measure whereas a client will churn or not. From **Reception Schedule Satisfaction** (Reception Schedule Graph) we can conclude all ratings levels have more no-churns than churns. So, all clients are equally satisfied with their reception schedule, another this is that the values of each rate are really similar so it's hard to take any conclusion here. In **Food and Drink Satisfaction** (Food and Drink Graph), we can conclude that if the rate is 0 or higher than 3, we have more no-churns than churns. So, that means that when the client is satisfied with the food and drink service, they are more likely to don't churn, the fact of having fewer churns when the rate is 0, is a curious case possibly an error in the questionnaire or from the client itself. In **Location Satisfaction** (Location Graph) we can conclude we have more churns than no churns from the clients when the rating is three. Observing the ratings are similar, except when rate is 3 or 5, because clients churn more and churn less, respectively, so the location of the accommodation doesn't have a big impact whether a client will churn or not. **Wi-Fi Satisfaction** (Wi-Fi Graph) in this service we must have the attention that we have ratings with values (6), so it occurs some problem in the questionnaire**.** So, after considering all rates of 6 as 5, we understand the clients tend more to churn when ratings from Wi-Fi satisfaction are lesser or equal than 2, so the Wi-Fi satisfaction is a factor whether a client churn or not. For **Amenities Satisfaction** (Amenities Graph), we can conclude when ratings (0, 4, or 5) the clients tend to no-churn, so we keep the same idea of some kind of problem in rating 0. But overall, the fact of having extra services (gym, pool, or spa) and their satisfaction is a major factor of whether a client will churn or not, since smaller rating means more churns. In **Staff Satisfaction** (Staff Graph), we can conclude higher ratings (4 or 5) in Staff we have fewer churns from the clients. This means when clients have support from staff and when they are well cared from workers, reduces the churns by the clients. For **Online Booking, Price Quality** (Online Booking Price Quality Graph)**, Room Space** (Room Space Graph)**, Check-Out** (Check-Out Graph) and **Cleanliness Satisfaction** (Cleanliness Graph) after observing the numbers we can conclude that higher rates (4 and 5) lead to less churns and more no churns, and lower rates leads to more churns, so we can say they are good to see if a client will churn or not. For **Check-In** (Check-In Graph) and **Bar Service Satisfaction** (Bar Service Graph), we conclude that the idea is similar but clients accept 'less' quality, since when ratings (3 to 5) clients are more likely to don't churn than churn. So that means, the clients are satisfied when the rating is bigger than 2. Otherwise, when the rating is (1 or 2) the clients are used to churn more than no-churn. **Overall, in satisfaction levels, higher rates means that clients are less likely to churn, lower rates lead to more churns, some confusing results to rate 0, from practically no answer to no fitting reason**. Finally, to conclude, we saw the system of client's rewards (Reward Points Box Plot) and were able to stablish that the middle 50% of customers have between 4445 and 5649 points, where the minimum and maximum points obtain by a client is 409 and 6940 points, respectively. Overall, we can observe a highly skewed observation to the left.

During the exploration part of the project, we took the conclusion using mostly seaborn library to get the graphs necessary to get this conclusion, we used simple and more complex feature to improve the visualization that will be better explain in *chapter 4.*

3. *Pre-Processing*

In Pre-Processing Criteria, the objective was to take care, cleaning, transforming, and reducing errors of the raw data of the respective datasets. We decide to treat the outliers and missing values instead of deleting them, we change the data types so we can do extensive analysis with different models that don't allow non-numerical data, we also review the coherence of the data, we perform a feature selection, with a correlation test, ANOVA test and importance to some models, and lastly, we scale the data.

So, starting with the pre-processing itself, we made a copy of our initial table of train dataset, we index our variable **'Cust_ID',** in order to save this information about the client and to have a better view of our dataset. Next move was to **treat missing values,** first we confirmed their position, then we made a copy of the variable to separate it from the rest, next we scale the variable with a minimum/maximum scaler and with this we were able to use the K-Nearest Neighbours imputer with 5 neighbours, after that we did an inverse scaler to get the real values of the year of birth of each customer and then we confirmed that there were no more missing values, so all that was left to do was to eliminate the **'Year_Birth'** present on the train dataset and add this new one with no missing values. The process of treating missing values and the use of KNN imputer as a form of treating them will be better explain in *chapter 4.*

Our next move was to **create new variables**, more specifically two new ones, the first that defines if a customer is a female or not and the second where from the year of birth variable, we took information about the client's age. So better explaining this process, for the variable **'Female'** we create a restriction on the first 3 letters **('Mr. or Ms.')** from each name in our dataset and save this as variable. Then using the dummy variable, we define that when is 'Mr.' is zero and when it's 'Ms.' is one, after this we rename it to **'Female'** and delete the **'Name'** variable, after a small analysis we saw a small difference and notice a few more women as clients (Gender Histogram). For the variable **'Age'** was simpler, we simply created a new variable, already called age where we subtracted the current year by the year of birth of each customer and then we transform this float variable into an integer one.

Then we had to **check the coherence** of our dataset, as notice before, we had some problems with the variables **'Wi-Fi Satisfaction'** and **'Longevity'.** In the first one we had ratings equal to six, and after checking the business needs of the company this did not make sense, so after some consideration we considered this rate as fives that were misplaces and used a replaced function to do this transformation. In the second one, we used the same thinking as before and to arrange the values of **'Longevity'** we used also the **replace** function to change the values (y) into (yes). Also, in data coherence, we noted the minimum age was 7 years old and maximum age was 85, we did not consider this as a problem as we consider this as being sons of clients who filled in the satisfaction questionnaire.

The next step was to **transform categorical data into numerical**, which would be a problem in some models afterwards. We treated the variables '**Type of Travel', 'Type of Room', 'Churn', and 'Longevity'**, for the first two variables we separated them and use the dummy variables function, creating new columns which we renamed them (a leisure travel column which said 1 if a customer did come as leisure, 0 otherwise; a single room which said 1 if the room chosen was single, a suite variable which said 1 if the room was a suite, and if the room chosen was double would appear as 0 in both variables), for the other two variables we transformed churn and no-churn into 1 and 0, respectively, and yes/no into 1/0.

Now, we decided to **treat the outliers** from the variable Reward Points as saw before, and since we only had two variables where it was possible to see have missing values, it was relatively easy to see. We decided that we won't remove outliers with Interquartile Method, because we believe that this valuable information and removing them is losing significant information from our dataset. So, we decide to keep them and apply multiple method that will be better explain in *chapter 4,* but we ended up using the box-cox transformation to treat them and the data was less dispersed.

After handling the outliers, the next step was **feature selection**, the first method we used was simply checking the variance, to see if some independent variable had variance equal to zero or close, this wasn't the case. Then we checked both

Pearson correlation and Spearman correlation which gave similar results, and with these we looked for high correlated variables (we decided a threshold of >70%) and therefore there wasn't high correlated variables, and we look also for predictors that weren't correlated with the predicted variable, and found four (Reception Schedule, Location, Suite Room and Rewards Points), before choosing to delete or not we decided to analyse all methods first. After correlations we checked the ANOVA test, and we asked for the five variables that performed worse in the ANOVA test, the results were Reception Schedule, Location, Suite Room, Reward Points and Leisure Travel. And when compared to the correlation selection we notice that most variables are the same, but before choosing what to delete we wanted to make sure this variable weren't important to the realization of the models, so we did a trial of two models, Decision Tree, and Random Forest. And after performing a horizontal bar char in both, we realize they were similar, and that Amenities Satisfaction and Comfort Satisfaction were really important to the models, in the other hand we checked that the fact a client choose a suite or not is not really important to the models (and although the Wi-Fi Satisfaction scores low in Decision Trees, it scores higher in Random Forest). So, after some consideration and comparing all techniques in a Feature Selection Table, and we decide only to delete the variable **'Suite'**, since it was presence in all criteria.

To finish the pre-processing of the train dataset, we scale the data with minmax scaler after separating the dependent variable from the independents, and we achieve the final table to perform the models. But before we move on to the modelling and performance assessment (*chapter 5 and chapter 7*) we pre-process both validation and test datasets, this pre-processing was exactly the same way we did the train dataset, and since we didn't delete any rows before we didn't have any problem. It's important to refer that when scaling the data, we scale it to the train data (or using the same scaler in better words). In Original Train_c and Scaled Train_c we can see a sample of both the final pre-processed table of train and the scaled train dataset.

4. Creativity and Other Self-Study

As we said in *chapter 3* in the realization of this project, we decided to not delete any customer, this includes missing information and outliers. In this process we use self-study techniques in order to obtain the desirable results, so to start to with the KNN imputer we used it to fill the missing values. The point of KNN method is to identify 'k' samples in the variable (in this case was Year of Birth) that are similar or close in the space. This 'k' was the number of neighbours or samples we will use to estimate the value of the missing data, and for k we end up using five, since it presented the best results. Before using this type of method is important to scale the data since it was necessary to produce good results, we scale it from 0 to 1, and afterwards we inversed the scale to get the true values of all data.

Next, we have the treatment of outliers we performed in the variable **'RewardPoints'** (Reward Points Graph) We agreed that these outliers were vital information about the customers that could be used in the modelling, so in order that these variables affect the model wrongly due to its outliers we decided to rescale the variable. To start we tried to see how the variable was distributed, as we saw before in *chapter 2*, and then we tried the first technique that was the **log scale** (Reward Points Log Scale Graph)**,** which is a way of showing numerical data over a very large range of values into a compact way, the results given were not the best, since the skew was even more marked by the log function. The next technique we tried was the **square root scale** (Reward Points Square Root Scale Graph), which transforms the range of values in such form that if the value of x is four times bigger than w, the results is only multiplied by two and not by 4, the results given here were best than the previous one, but still worse than the original, so we also rejected. Then we tried the **reciprocal transformation scale** (Reward Points Reciprocal Scale Graph), which is an example of an inverse distribution, and the reciprocal (inverse) of a random variable with a reciprocal distribution itself has a reciprocal distribution, the results given were not the desirable too. So as the final technique we performed a **box-cox scale** (Reward Points Box Cox Scale Graph), which is a transformation of non-normal dependent variables into a normal distribution shape, where we defiend both the fitted data and the value of the lambda used to perform the best transformation, beside saving the transformed the variable, we also saved the value of lambda to be able to transform the validation and test dataset with exactly the same value. After this we plot the difference of both graphs and saw that these were the best results and the one that we implemented into our final dataset, removing the then the previous variable. But explaining better, we took the original data, with a non-normal distribution, and returned the fitted data and the lambda value that was used to fit the transformation of non-normal distribution to a normal distribution.

Besides the treatment made in pre-processing we also looked for improving the visualization of the graphs made during the realization of the project. To start, we used a mixed of the libraries of matplotlib and seaborn, using seaborn to create the graph itself and matplotlib to add some features, as title, legend, labels, and more that we learned in class. Furthermore, we tried to facilitates the view and understanding of these graphs by adding a label for each bar, in order to have a clear idea of the values, we also increase the x limit in order that label didn't be cut by the bord of the graph, and we had a padding to the label to implement a space between the label and the bar. Some other parameters we used to improve the visualization of the graph were 'hue' to legend some graphs and compare churn and no-churn, colour/palette to put the graphs in a common theme, 'shrink' to separate the bins of the histogram and doesn't look like a huddle, and 'bins' that defined a specific number of bins or categories of the histograms. Besides these more specific, we used some functions more classic, as title, xlabe, ylabel, xticks and legend.

5. *Modelling*

After treating all the data, it was time to divide our dataset into training set, validation set and test set to implement different predictive algorithms to find the most precise one. Starting with the training phase, the bigger the dataset, the better, to build the model and create a certain pattern based on the data, then the validation test to access the model that we used during the training phase and finally the test set to provide an unbiased estimate of the final model.

Starting with a **Logistic Regression Model**, where using a binary outcome, models the probability given an input variable. So that is what we did. We began by Importing the Logistic Regression Model, create the model itself to fit with the dataset, to predict the probabilities and make evaluations regarding that. As parameters, we only defined the class weight, that defines the wight of the classes, as balanced in order to values of the dependent variable to automatically adjust weights inversely proportional to class frequencies.

Secondly, we used the **K-Nearest Neighbours Model**, where it stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K-NN algorithm. After we implement the code, we were not happy with the outcome of our values, so we decided to find the number of neighbours that best suited our model. By using a for loop from one to ten we assess that the best option was to use 6/8 neighbours. After that, we rewrite the code with the number of neighbours equal to six after we establish and reassess the results, whose came with a better outcome.

Next up, we try **Decision Trees** to continue comparing with the previous models. This predictive model is a mapping from observations about an item to conclusions about its target value. In the tree structures, leaves represent classifications (also referred to as labels), non-leaf nodes are features, and branches represent conjunctions of features that lead to the classifications. Just like with the K-Nearest Neighbours Model, we were not happy with the outcome of our results of the default model, since our defined three has a depth of 30, 1993 nodes and a total of 997 leaves, and in our test was likely to have a problem with overfitting. So, we decided to try to discover both the best depth and number of leaf nodes for our model by iterating them in two separated for loops, and using f1-score to compare and evaluate them. After this, we rewrite the code with the best depth and leave notes possible (with max depth=10; max leaf nodes=500; and the entropy criteria) and the results were much more satisfying, and this difference was noticeable in the draw of the model.

Our final model given in class was the **Neural Networks**, a structure inspired by the human brain, mimicking the way that biological neurons signal to one another to learn and improve their accuracy over time. We started by defining two functions, which it is going to receive a model to the function and will interact ten times and return the mean time taken and the mean accuracy given in both train and validation data set, for each one of them and with that we put that information in a simple table (Table Accuracy NN) , we did this since the scores changes every time, so this way we have a clear idea of the approximated score. Since the first results was from a default model, it was not appropriate for our problem, we had to change them by creating a new code to evaluate our model with different parameters. The parameters we chose to evaluate were: activation, which defines the weighted sum of the inputs when transforming into outputs (tanh or logistic); the size and number of hidden layers or layers between input and output (1 with 15, or 2 both with 15); the learning rate, which controls how quickly the model is adapted to the problem in question (between constant or inverse scaling); the initial learning rate (0.1 or 0.01); and the solver or optimization solver (stochastic gradient descent or stochastic gradient-based optimizer). It's important to mention that the size of hidden layers, since we follow the rules given by the teacher, we used 2/3 of the number of independent variables plus 1, given a final result of 15, and then checked and it followed the restrictions of being between the number of dependent variables and the number of independent variables, and it was also smaller than twice the number of variables (42). To optimize the five parameters, we decided to check each one of the combinations to each other to return the best combination possible, and since we already define the 'hidden_layers_size' as fifteen, the process was easier but still very slow. After this we were ready to fit these new parameters in our model and we obtain a better evaluation than the default model.

After we went through the models that were taught to us in classes we decided to go abroad and search for other models that could fit the best for our problem. And these models will be better explored in *chapter 6.*

6. Other Predictive Modelling

The first model we tried in our own was **Random Forest**, a classifier that contains several decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. After applying this model since the results were very satisfying, we did not need to apply any changes and we moved on to the next one.

The next model that we used that were not part of the ones taught in class was **the Gradient Boosting Machine Classifier model**, which combine different weak learning models together to create a strong predictive model, being Decision trees usually used when doing gradient boosting. A point to mention is that this type of model can be very slow to train, this is because trees must be created and added sequentially. But we still tried this model, and although we defined some parameters the results were not as satisfied as expected, so with this in mind we tried the following model with hopes of resolving this kind of a disappointing model.

So, as said, we also tried the **Histogram Gradient Boosting Machine Classifier model**, this approach is part of the Gradient Boosting Machines and because it is also a gradient boosting that implements this technique and train faster decision trees around the input variable. As before, this model is also somewhat difficult to train since it's also a gradient boosting machine. So, we tried the model with hopes of having better f1-score than before. And since we believed that this model had potential, we did some trials of grid searches to get the best parameters and end up with the learning rate or shrinkage at 0.1, the loss as binary cross entropy or logistic, this loss is optimized using a gradient descent, the max number of bins as 125, the maximum depth as 6 and the max number of leaf nodes as 25. After this, we just used our custom function to get the results.

Finally, the last model that we used was the **Naive Bayes Algorithm**, based on Bayes theorem, it predicts based on the probability of something for solving classification problems. Like the previous models every time we change the parameters our values for our validation would decrease and our model would become weaker, so we decided to keep the default settings and not change our model giving us still a good outcome.

7. Performance and Assessment

After writing the algorithm for the different models and make adjustments in the parameters to better fit in our problem it was time to assess their performance to choose the best one, based on the values of training and validation. Remember that were hoping that training and validation be as high as possible in the f1 score, but at the same time the difference between the two models don't be too much, since we don't want to overestimate (with the train dataset), or even underestimate. There were a few models that we could disposal already from the beginning. It's important to mention that in this part of the project we only consider the final form of each model, this being the best results of each model only, with the transformations and experiments we did to define the parameters. And in order to obtain the results and to pay special attention to the f1 score, we first created a function that received the true value of the predicted variable and the variable predicted by the model, of both datasets, and returned information like the recall, f1score (the most important), the accuracy, the precision and many others.

From the models that were given to us in the classes the most satisfying one was Neural Networks model (Neural Networks Table) with a high value of training and validation, with 0.9485 and 0.9341, respectively. So, after all, it was a great improvement from the default model we also tried (Default NN Table). But also, with remarkably similar values the Decision Tree model (Decision Tree Table). with values for training, 0.9457, and for validation 0.9274, which also shown a great improvement from the default model, since this one presented high level of overfitting (Default DT Table). Unfortunately, the Logistic Regression model (Logistic Regression Table) did not offer good results, with training at 0.8233 and validation at 0.8256 so we didn't even consider as a final option.

From the other four models that we study the most satisfying one was the Histogram Gradient Boosting Classifier model (Histogram Gradients Boosting Table) with the biggest value comparing to the other models in validation, 0.9445, and since it presents a really small overfitting (with an f1 score in train of 0.9546) we believe that this won't affect negatively the results in the test data. Beside the Histogram GBM Classifier other hypotheses we had as the final model was the other GBM, which presented really bad results for our surprise, with train at 0.8233 and validation at 0.8257. Random Forest (Random

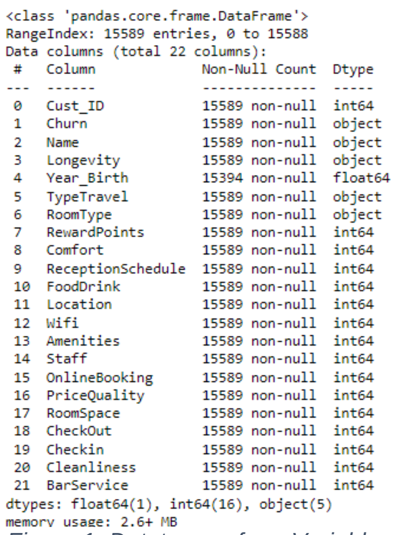
Forest Table) present also good values both in train dataset and validation dataset, with 1.0 and 0.9415, respectively, and as it’s noticeable we didn’t choose due to a bigger overfitting of the model (close to 6%). As said before, we did try another model, the Naïve Bayes Algorithm (Naive Bayes Table), but the results weren’t the desirable, so we didn’t consider with both datasets close to 0.80 in the f1 score.

After some consideration and comparing the results in a table (Model Results Table), we believe that the **Histogram Gradient Machine model will present the best results for our problem,** from all the models we gave in class and the ones we tried by our own. This is given since although the Random Forest presents high results for the f1 score in both train and validation datasets, it also presents high levels of overfitting (close to 6%). Both Decision Trees and the Neural Networks model were also good option, but the first had lower levels in validation presenting some overfitting and the second although had very similar in both datasets, the results were a little lower than the ones in the Histogram GBM model. Other models like, Logistic Regression, GBM and Naïve Bayes Algorithm were immediately discarded due to low values in both datasets.

8. Conclusions

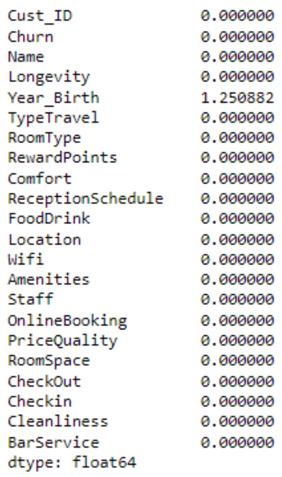
In conclusion, in the given the problem to predict the probability of churning depending on their satisfaction and behaviour we observed the dataset and did an exploration of it, and realized that most customers where middle aged adults in business travels, and that customers that rate worst in any satisfaction level, usually churn on the company, and we also saw that we had to make some adjustments to have a more precise and correct model. So then, we treated missing values and outliers, checked the coherence of the dataset, created some new variables, did a feature selection and then did a preparation of both validation and test datasets. After this phase was time to write different algorithms for the different models and adjust them with different parameters to improve them and get better results. With this, we obtained different outcomes so after some analysis and comparison we evaluated each one of them and based on the values we decided to get the results of the test dataset using the Histogram Gradient Boosting Classifier model. So we really believe that the model we implemented into our project will be a good solution to predict whether a customer will give on BookMe or not.Figure 32: ANOVA Test

9. Annex



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15589 entries, 0 to 15588
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Cust_ID             15589 non-null  int64  
 1   Churn               15589 non-null  object 
 2   Name                15589 non-null  object 
 3   Longevity           15589 non-null  object 
 4   Year_Birth          15394 non-null  float64
 5   TypeTravel          15589 non-null  object 
 6   RoomType            15589 non-null  object 
 7   RewardPoints        15589 non-null  int64  
 8   Comfort             15589 non-null  int64  
 9   ReceptionSchedule   15589 non-null  int64  
 10  FoodDrink           15589 non-null  int64  
 11  Location            15589 non-null  int64  
 12  Wifi                15589 non-null  int64  
 13  Amenities           15589 non-null  int64  
 14  Staff               15589 non-null  int64  
 15  OnlineBooking       15589 non-null  int64  
 16  PriceQuality        15589 non-null  int64  
 17  RoomSpace           15589 non-null  int64  
 18  CheckOut            15589 non-null  int64  
 19  Checkin             15589 non-null  int64  
 20  Cleanliness         15589 non-null  int64  
 21  BarService          15589 non-null  int64  
dtypes: float64(1), int64(16), object(5)
memory usage: 2.6+ MB
```

Figure 1: Datatypes of our Variables



```
Cust_ID              0.000000
Churn                0.000000
Name                 0.000000
Longevity            0.000000
Year_Birth           1.250882
TypeTravel           0.000000
RoomType             0.000000
RewardPoints         0.000000
Comfort              0.000000
ReceptionSchedule    0.000000
FoodDrink            0.000000
Location             0.000000
Wifi                 0.000000
Amenities            0.000000
Staff                0.000000
OnlineBooking        0.000000
PriceQuality         0.000000
RoomSpace            0.000000
CheckOut             0.000000
Checkin              0.000000
Cleanliness          0.000000
BarService           0.000000
dtype: float64
```

Figure 2: Percentage of Missing Values each Variable

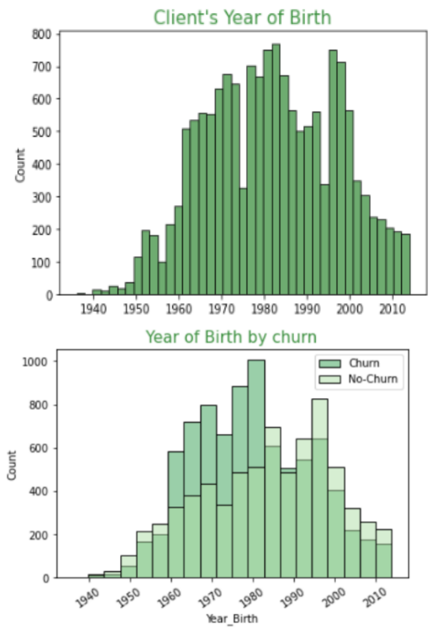


Figure 1: Year of Birth by Churn

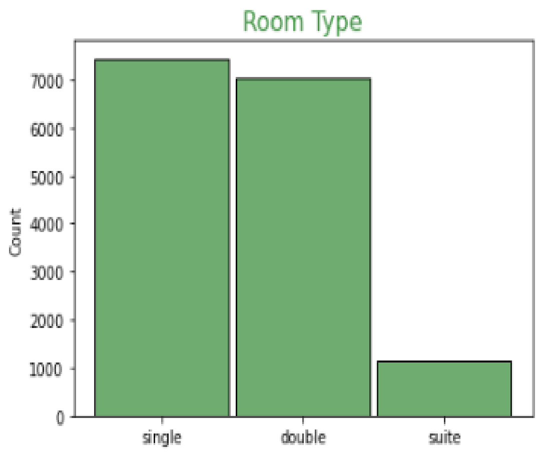


Figure 4: Room Type

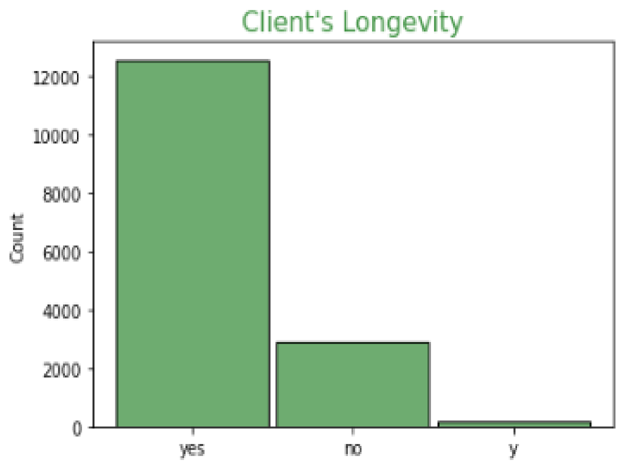


Figure 5: Longevity

s

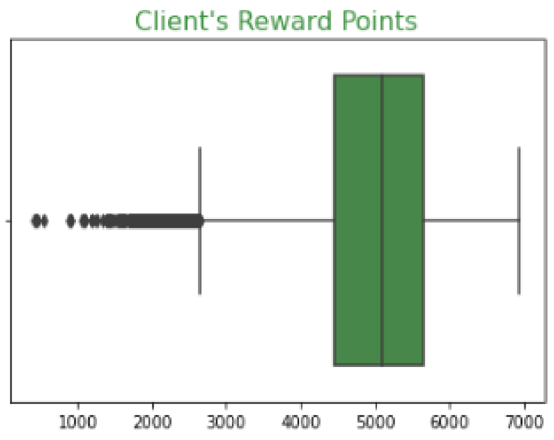


Figure 6: Reward Points Outliers

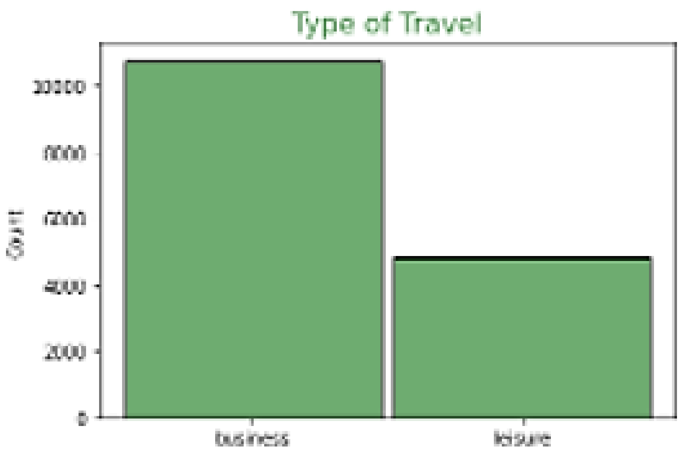


Figure 7: Type of Travel

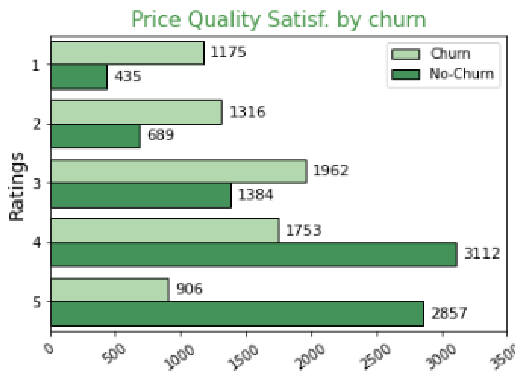


Figure 8: Price Quality Satisfaction by Churn

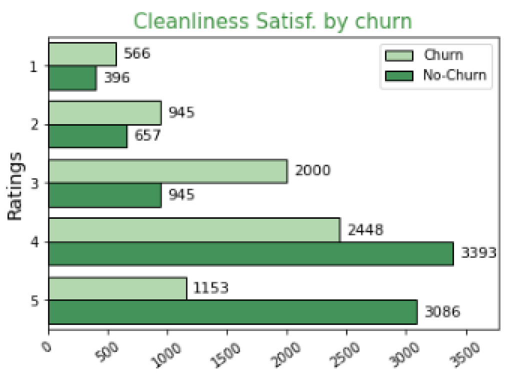


Figure 9: Cleanliness Satisfaction by Churn

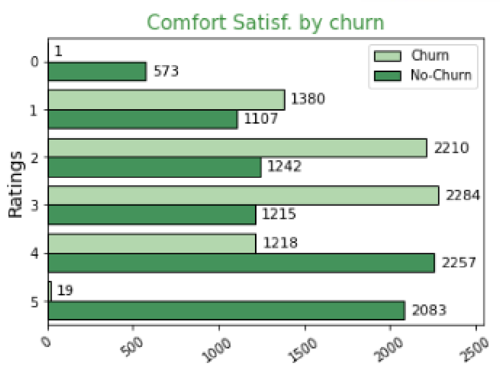


Figure 10: Comfort Satisfaction by Churn

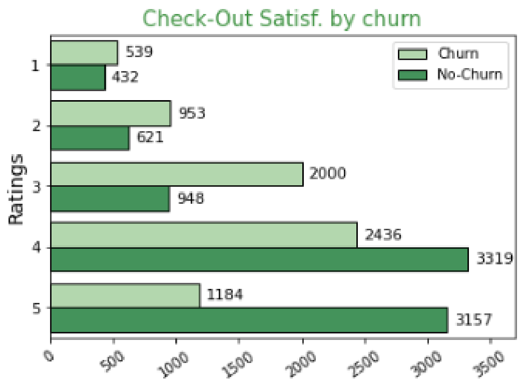


Figure 11: Check-Out Satisfaction by Churn

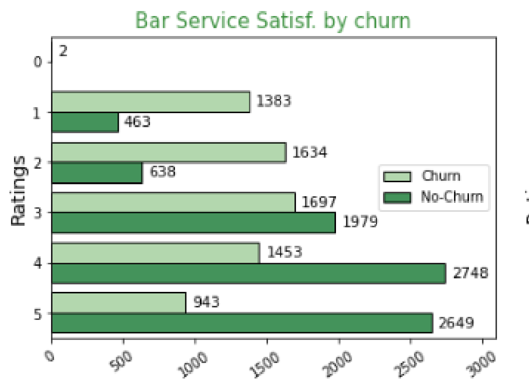


Figure 12: Bar Service Satisfaction by Churn

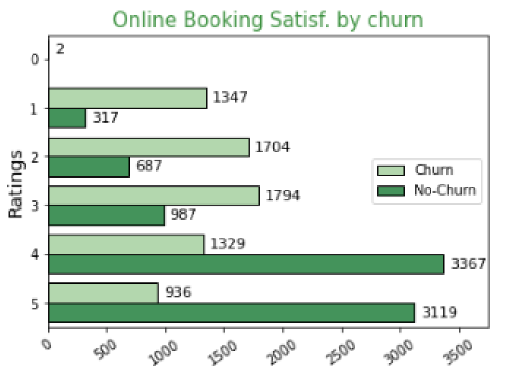


Figure 13: Online Booking Satisfaction by Churn

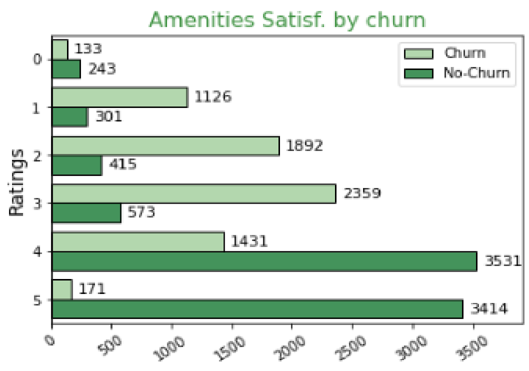


Figure 14: Amenities Satisfaction by Churn

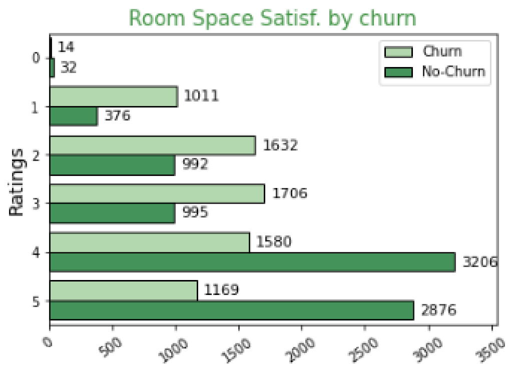


Figure 15: Room Space Satisfaction by Churn

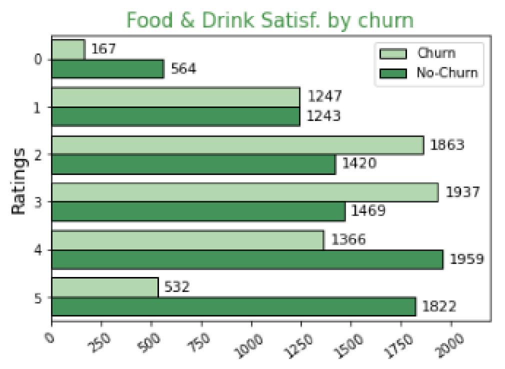


Figure 16: Food and Drink Satisfaction by Churn

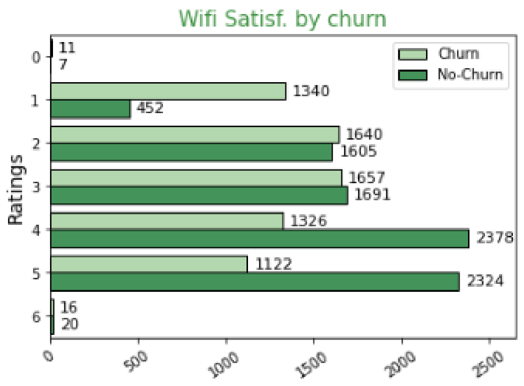


Figure 17: Wi-Fi Satisfaction by Churn

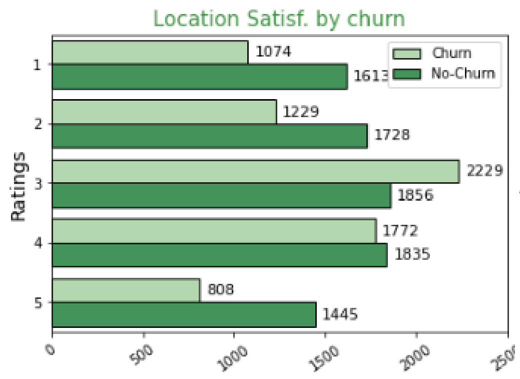


Figure 18: Location Satisfaction by Churn

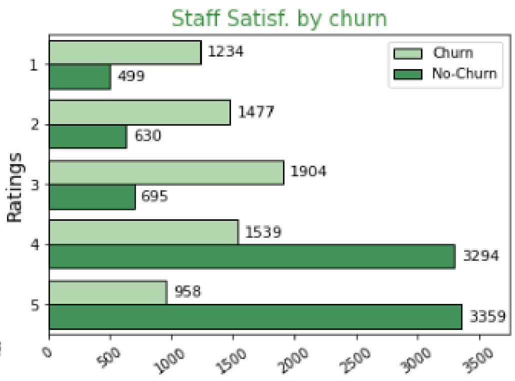


Figure 19: Staff Satisfaction by Churn

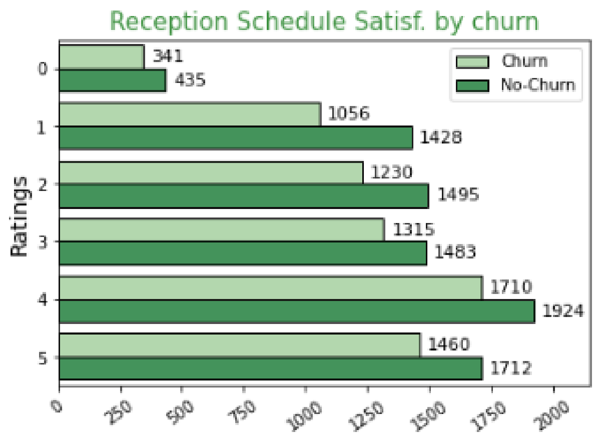


Figure 20: Reception Schedule Satisfaction by Churn

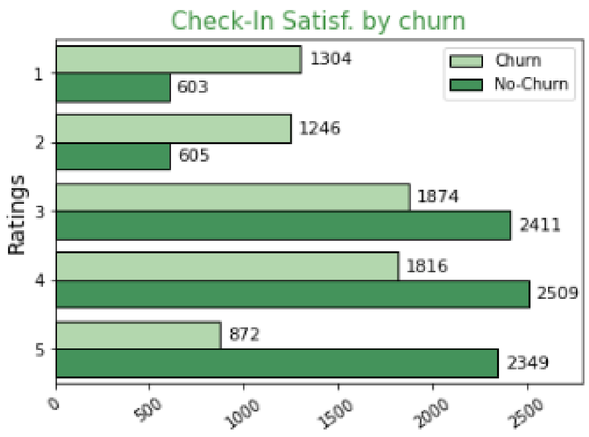
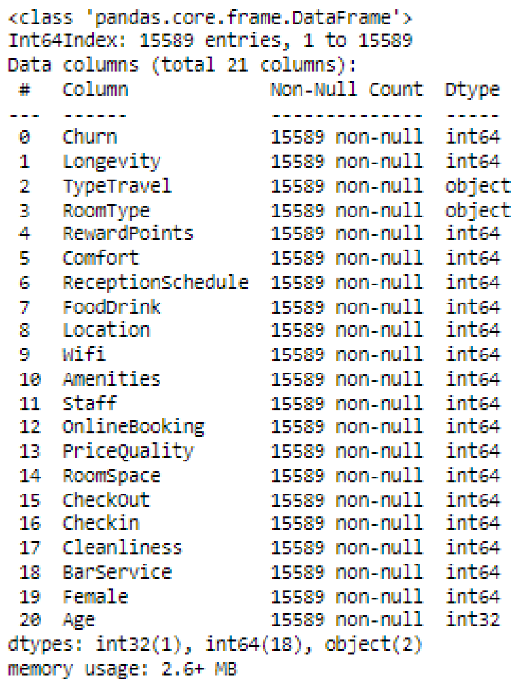


Figure 21: Check-In Satisfaction by Churn


```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15589 entries, 1 to 15589
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Churn              15589 non-null  int64
 1   Longevity          15589 non-null  int64
 2   TypeTravel         15589 non-null  object
 3   RoomType           15589 non-null  object
 4   RewardPoints       15589 non-null  int64
 5   Comfort            15589 non-null  int64
 6   ReceptionSchedule  15589 non-null  int64
 7   FoodDrink          15589 non-null  int64
 8   Location           15589 non-null  int64
 9   Wifi               15589 non-null  int64
 10  Amenities          15589 non-null  int64
 11  Staff              15589 non-null  int64
 12  OnlineBooking      15589 non-null  int64
 13  PriceQuality       15589 non-null  int64
 14  RoomSpace          15589 non-null  int64
 15  CheckOut           15589 non-null  int64
 16  Checkin            15589 non-null  int64
 17  Cleanliness        15589 non-null  int64
 18  BarService         15589 non-null  int64
 19  Female             15589 non-null  int64
 20  Age                15589 non-null  int32
dtypes: int32(1), int64(18), object(2)
memory usage: 2.6+ MB
```
Figure 22: Clients Age (Years)

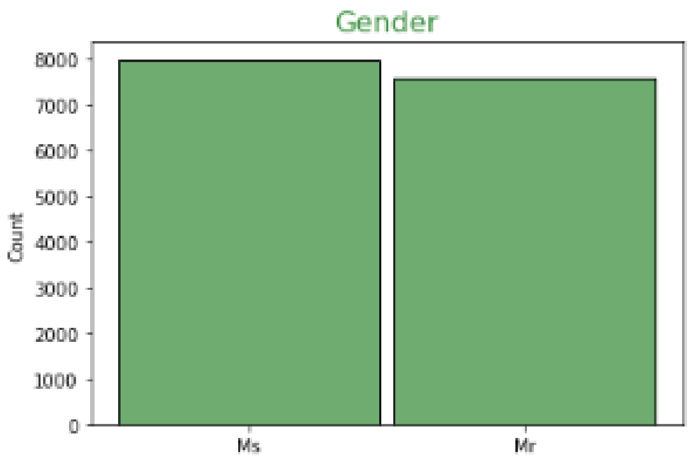


Figure 23: Clients Gender Counting

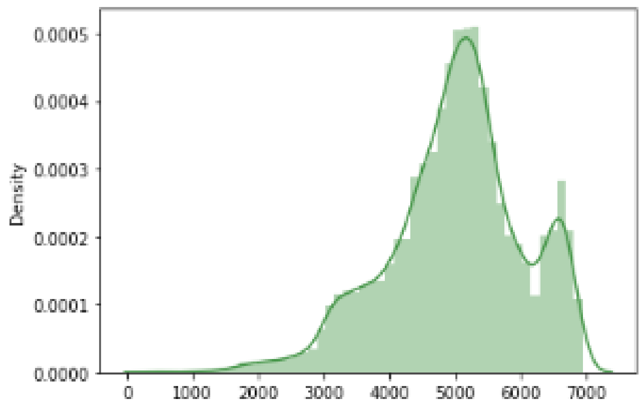


Figure 24: Original Reward Points

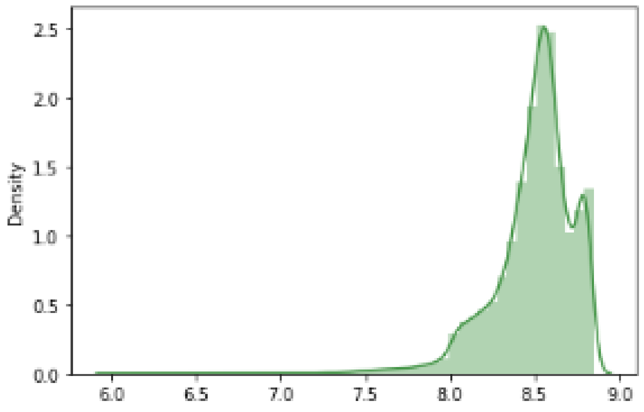


Figure 25: Reward Points Log Scale

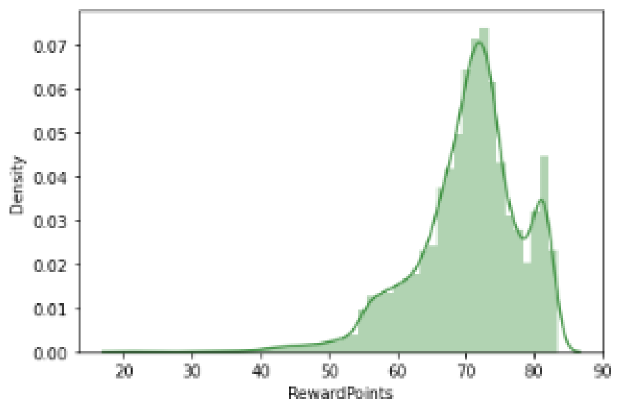


Figure 26: Square Root Scale Reward Points

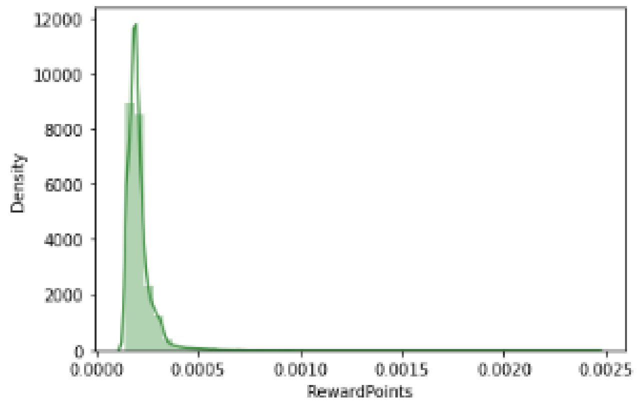


Figure 27: Reward Points Reciprocal Scale

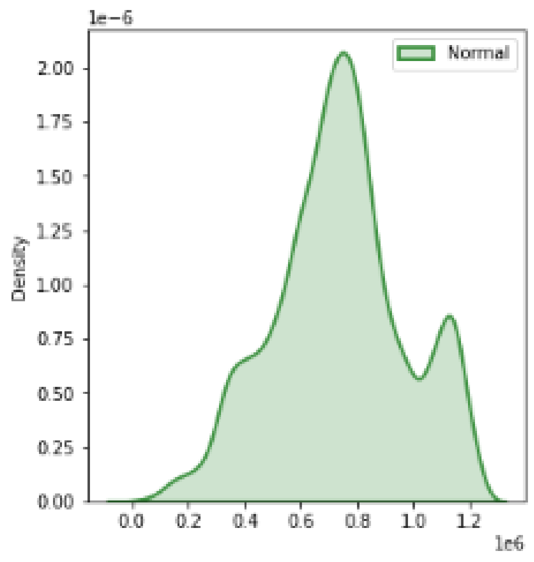


Figure 28: Box Cox Scale Density

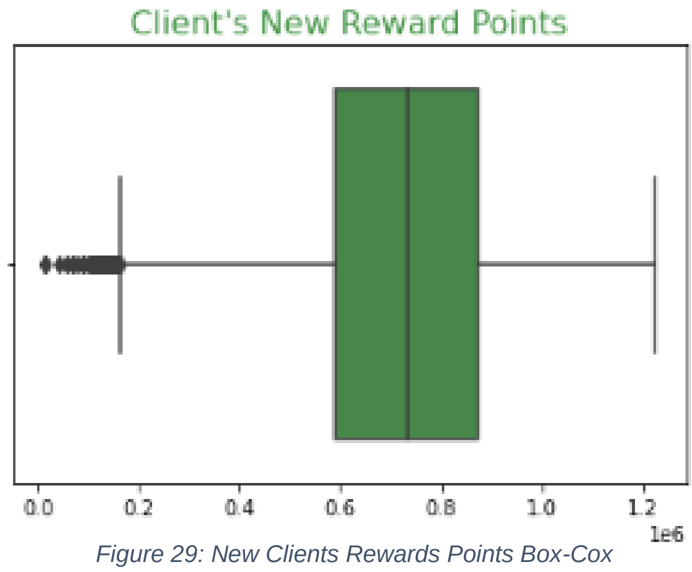
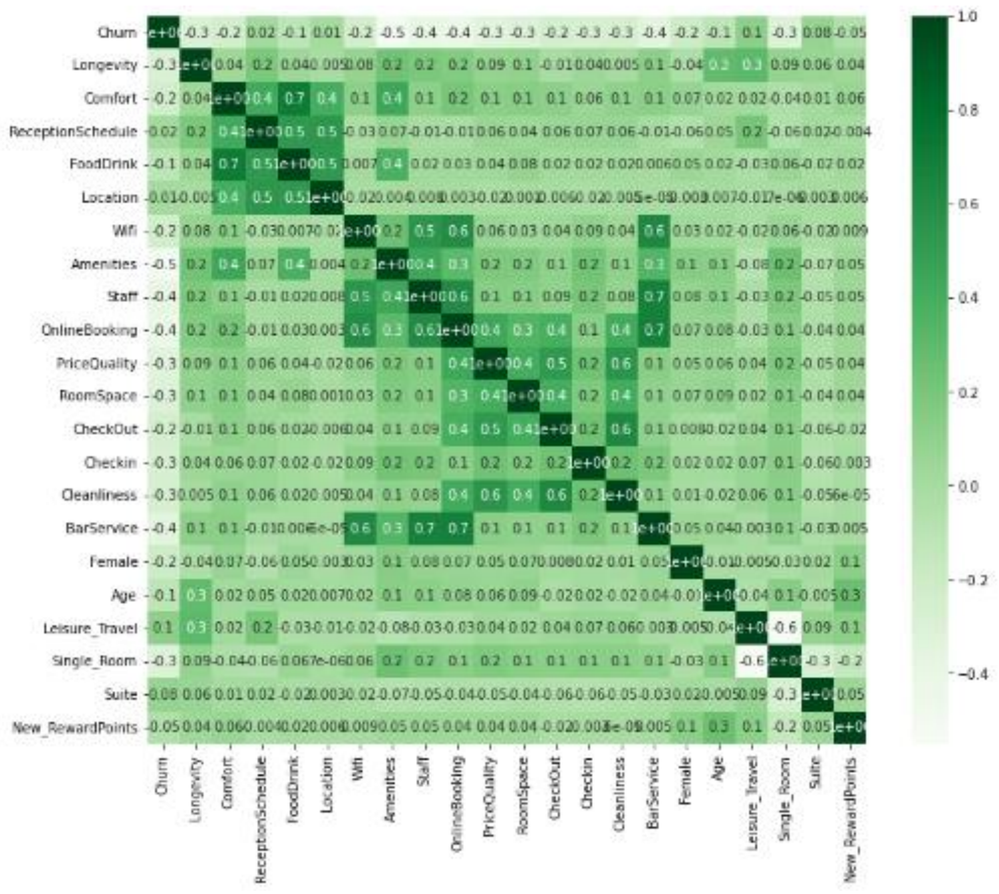


Figure 29: New Clients Rewards Points Box-Cox

Figure 30: Spearman Correlation Matrix



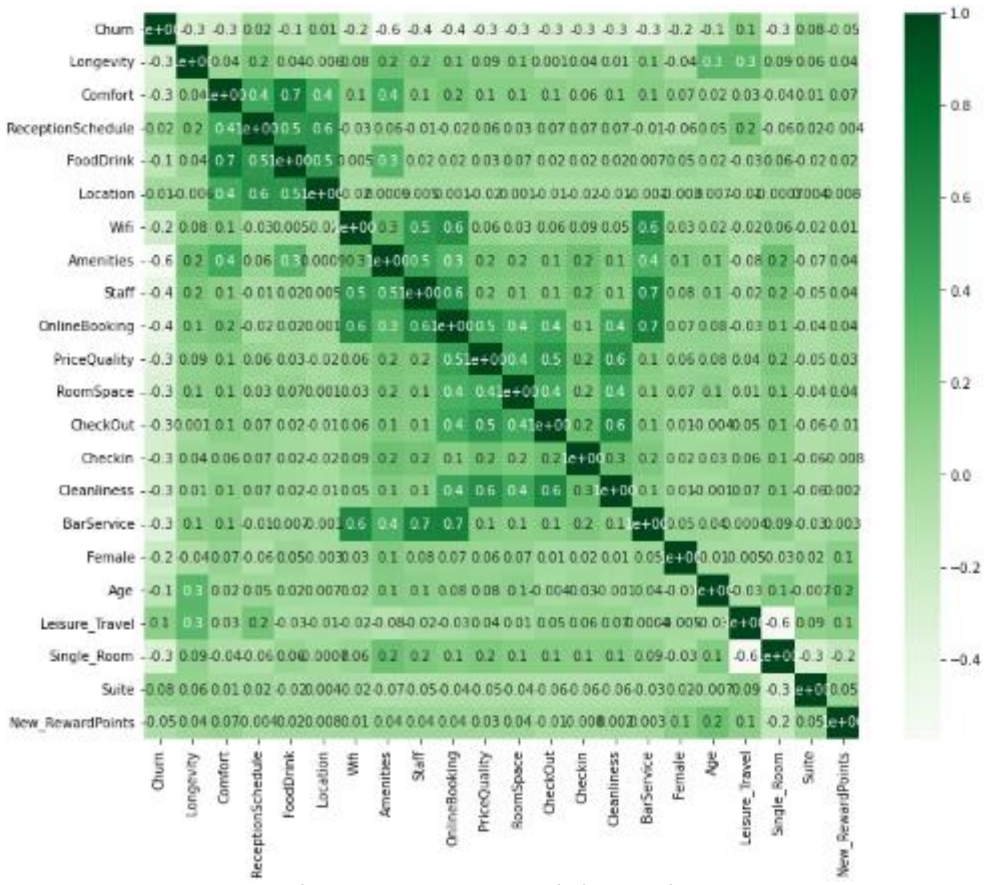
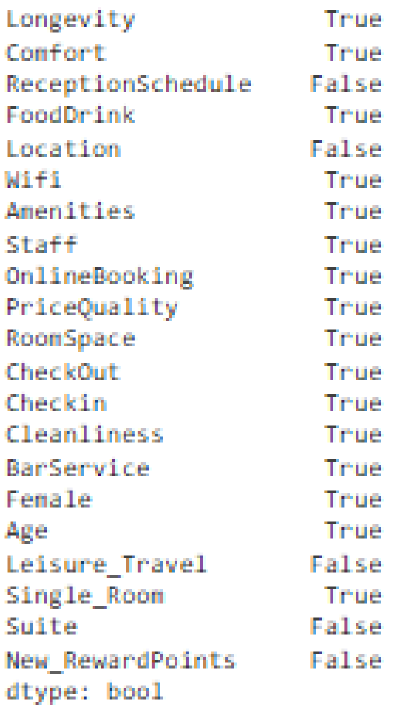
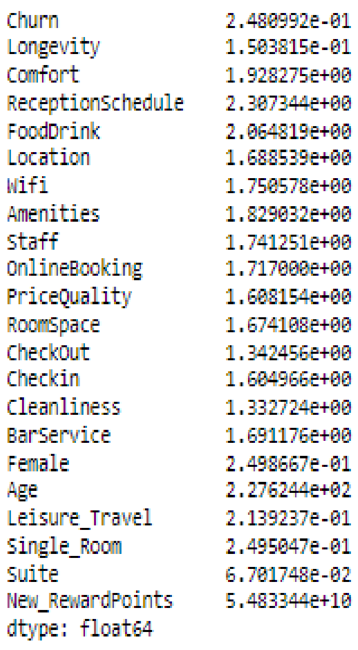


Figure 31: Pearson Correlation Matrix



```
Longevity             True
Comfort               True
ReceptionSchedule    False
FoodDrink             True
Location             False
Wifi                  True
Amenities             True
Staff                 True
OnlineBooking         True
PriceQuality          True
RoomSpace             True
CheckOut              True
Checkin               True
Cleanliness           True
BarService            True
Female                True
Age                   True
Leisure_Travel       False
Single_Room           True
Suite                False
New_RewardPoints     False
dtype: bool
```

Figure 32: ANOVA Test



```
Churn                2.480992e-01
Longevity            1.503815e-01
Comfort              1.928275e+00
ReceptionSchedule    2.307344e+00
FoodDrink            2.064819e+00
Location             1.688539e+00
Wifi                 1.750578e+00
Amenities            1.829032e+00
Staff                1.741251e+00
OnlineBooking        1.717000e+00
PriceQuality         1.608154e+00
RoomSpace            1.674108e+00
CheckOut             1.342456e+00
Checkin              1.604966e+00
Cleanliness          1.332724e+00
BarService           1.691176e+00
Female               2.498667e-01
Age                  2.276244e+02
Leisure_Travel       2.139237e-01
Single_Room          2.495047e-01
Suite                6.701748e-02
New_RewardPoints     5.483344e+10
dtype: float64
```

Figure 33: Variance between Variables

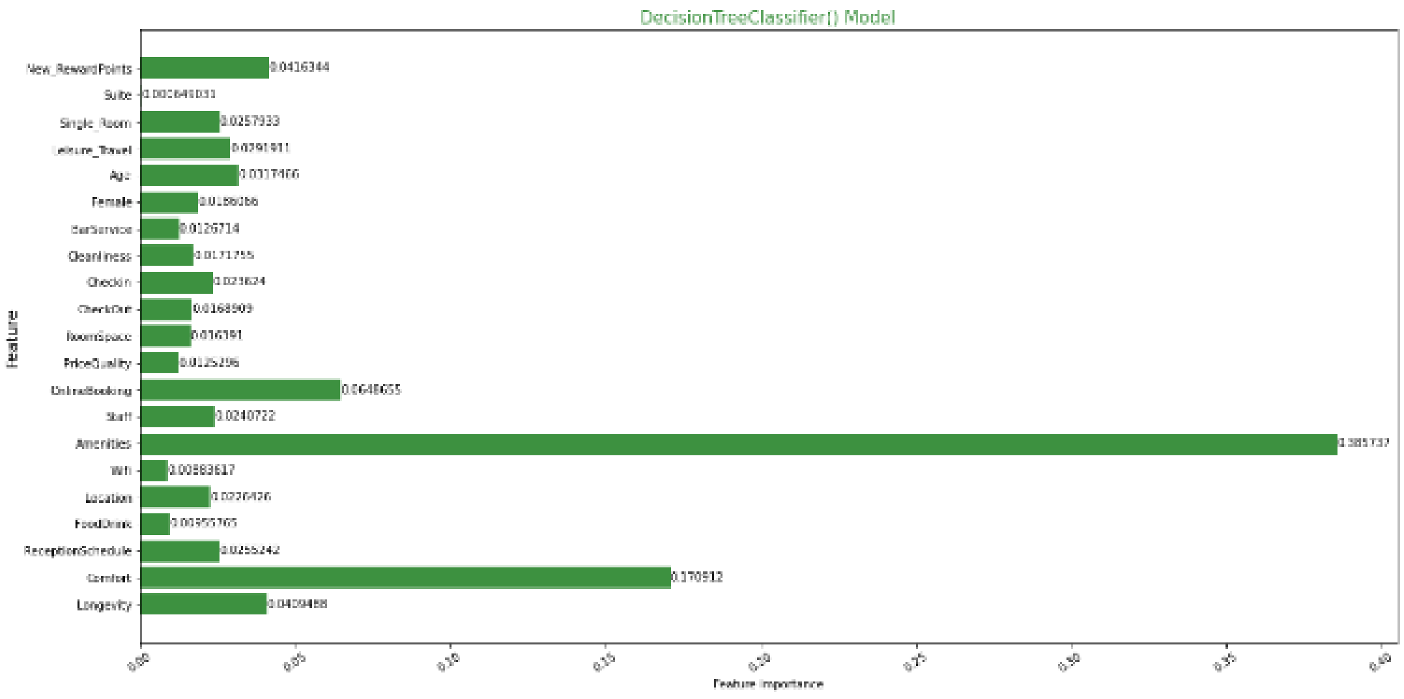


Figure 34: Decision Tree Classifier

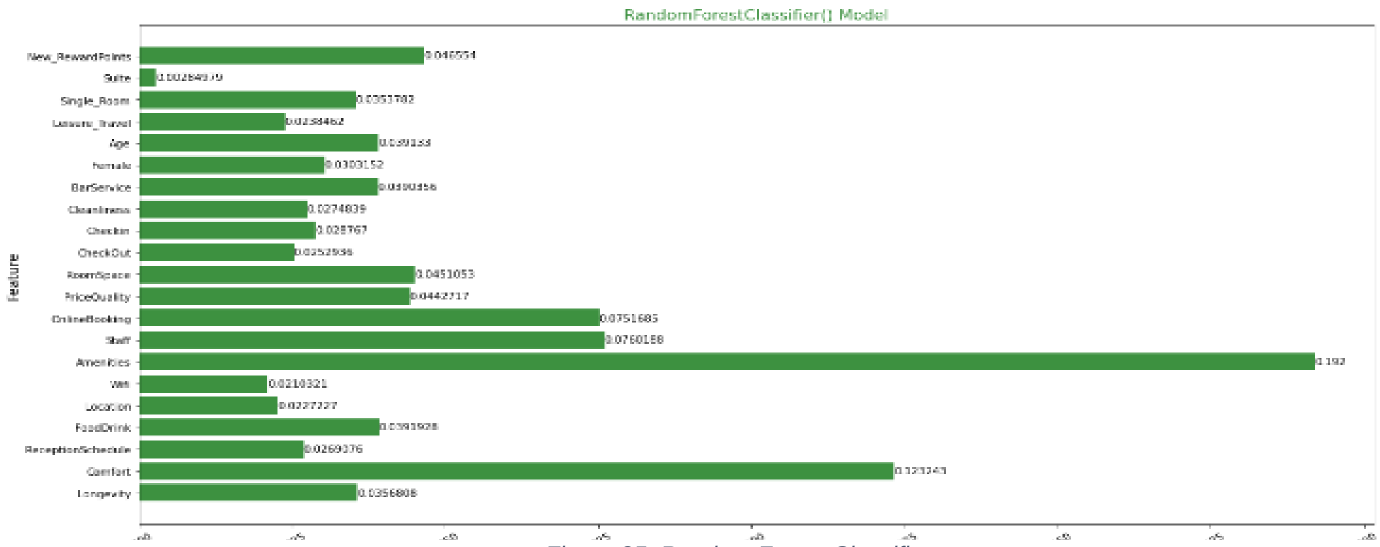
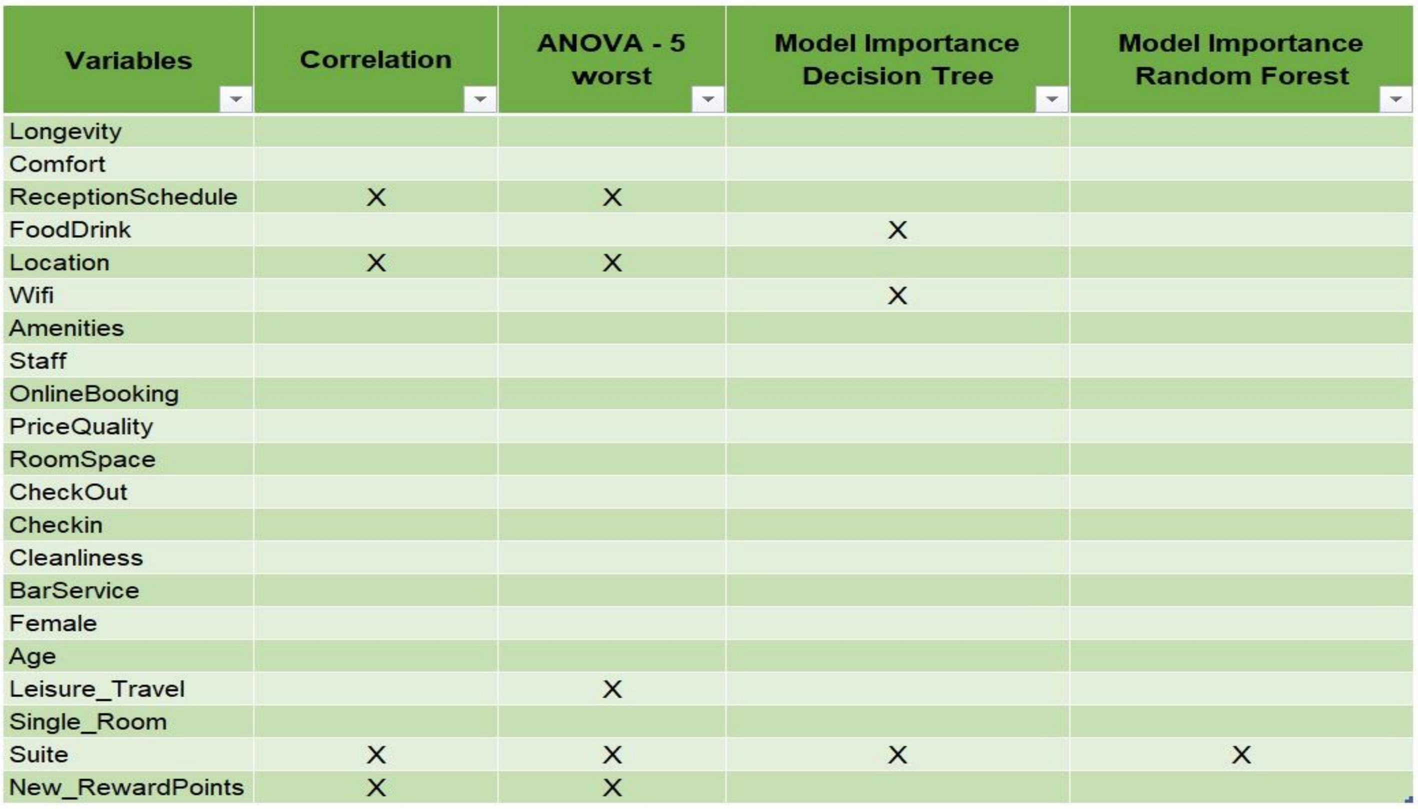
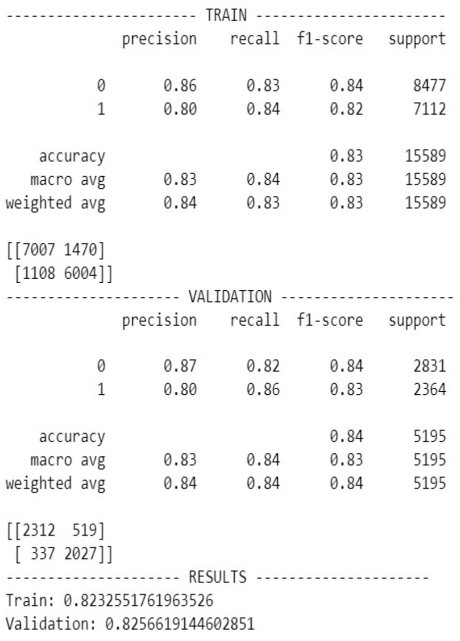


Figure 35: Random Forest Classifier



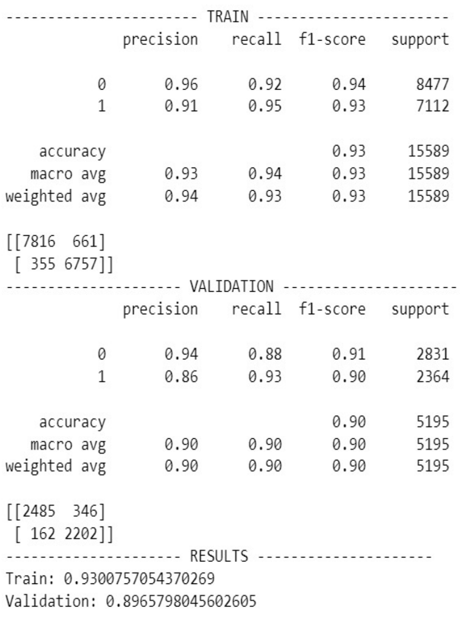
Variables	Correlation	ANOVA - 5 worst	Model Importance Decision Tree	Model Importance Random Forest
Longevity				
Comfort				
ReceptionSchedule	X	X		
FoodDrink			X	
Location	X	X		
Wifi			X	
Amenities				
Staff				
OnlineBooking				
PriceQuality				
RoomSpace				
CheckOut				
Checkin				
Cleanliness				
BarService				
Female				
Age				
Leisure_Travel		X		
Single_Room				
Suite	X	X	X	X
New_RewardPoints	X	X		

Figure 36: Feature Selection Variables used in each Model



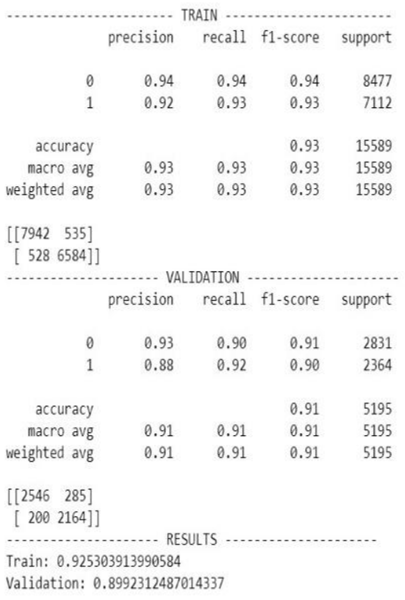
```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       0.86      0.83      0.84      8477
           1       0.80      0.84      0.82      7112

    accuracy                           0.83     15589
   macro avg       0.83      0.84      0.83     15589
weighted avg       0.84      0.83      0.83     15589

[[7007 1470]
 [1108 6004]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.87      0.82      0.84      2831
           1       0.80      0.86      0.83      2364

    accuracy                           0.84      5195
   macro avg       0.83      0.84      0.83      5195
weighted avg       0.84      0.84      0.84      5195

[[2312  519]
 [ 337 2027]]
-------------------- RESULTS --------------------
Train: 0.8232551761963526
Validation: 0.8256619144602851
```

Figure 37: Logistic Regression Model



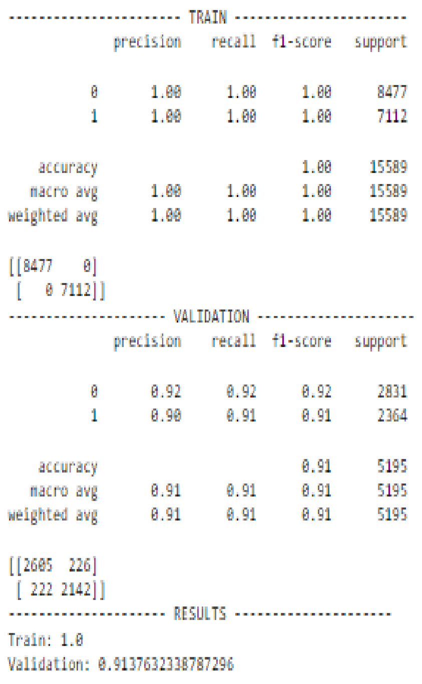
```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       0.96      0.92      0.94      8477
           1       0.91      0.95      0.93      7112

    accuracy                           0.93     15589
   macro avg       0.93      0.94      0.93     15589
weighted avg       0.94      0.93      0.93     15589

[[7816  661]
 [ 355 6757]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.94      0.88      0.91      2831
           1       0.86      0.93      0.90      2364

    accuracy                           0.90      5195
   macro avg       0.90      0.90      0.90      5195
weighted avg       0.90      0.90      0.90      5195

[[2485  346]
 [ 162 2202]]
-------------------- RESULTS --------------------
Train: 0.9300757054370269
Validation: 0.8965798045602605
```

Figure 38: Defaul K-Nearest Neighbours Model



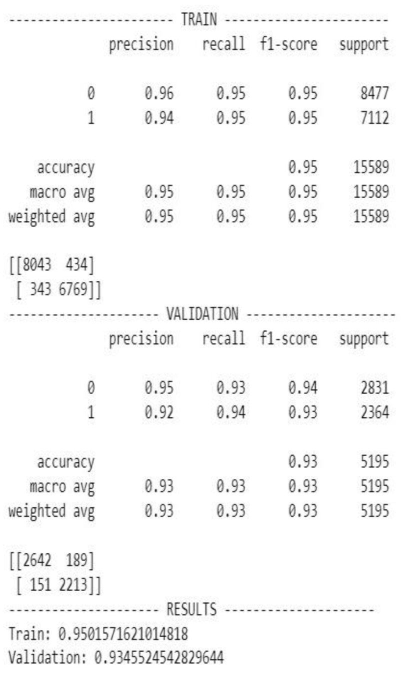
```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      8477
           1       0.92      0.93      0.93      7112

    accuracy                           0.93     15589
   macro avg       0.93      0.93      0.93     15589
weighted avg       0.93      0.93      0.93     15589

[[7942  535]
 [ 528 6584]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.93      0.90      0.91      2831
           1       0.88      0.92      0.90      2364

    accuracy                           0.91      5195
   macro avg       0.91      0.91      0.91      5195
weighted avg       0.91      0.91      0.91      5195

[[2546  285]
 [ 200 2164]]
-------------------- RESULTS --------------------
Train: 0.925303913990584
Validation: 0.8992312487014337
```

Figure 39: K-Nearest Neighbours Model



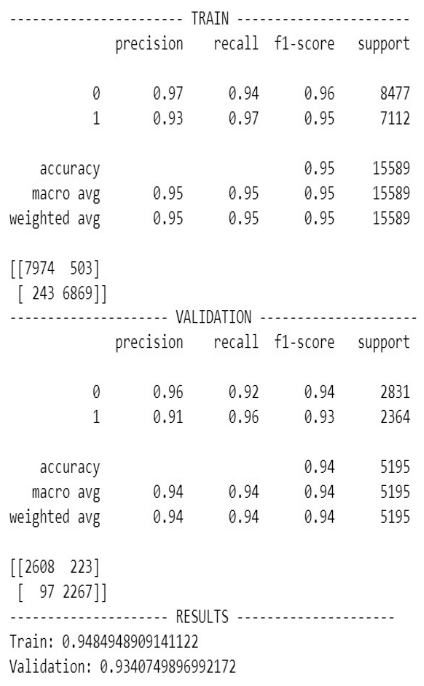
```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      8477
           1       1.00      1.00      1.00      7112

    accuracy                           1.00     15589
   macro avg       1.00      1.00      1.00     15589
weighted avg       1.00      1.00      1.00     15589

[[8477    0]
 [   0 7112]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      2831
           1       0.90      0.91      0.91      2364

    accuracy                           0.91      5195
   macro avg       0.91      0.91      0.91      5195
weighted avg       0.91      0.91      0.91      5195

[[2605  226]
 [ 222 2142]]
-------------------- RESULTS --------------------
Train: 1.0
Validation: 0.9137632338787296
```

Figure 40: Default Decision Tree Model



```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       0.96      0.95      0.95      8477
           1       0.94      0.95      0.95      7112

    accuracy                           0.95     15589
   macro avg       0.95      0.95      0.95     15589
weighted avg       0.95      0.95      0.95     15589

[[8043  434]
 [ 343 6769]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.95      0.93      0.94      2831
           1       0.92      0.94      0.93      2364

    accuracy                           0.93      5195
   macro avg       0.93      0.93      0.93      5195
weighted avg       0.93      0.93      0.93      5195

[[2642  189]
 [ 151 2213]]
-------------------- RESULTS --------------------
Train: 0.9501571621014818
Validation: 0.9345524542829644
```

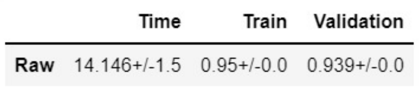
Figure 41: Final Decision Tree Model



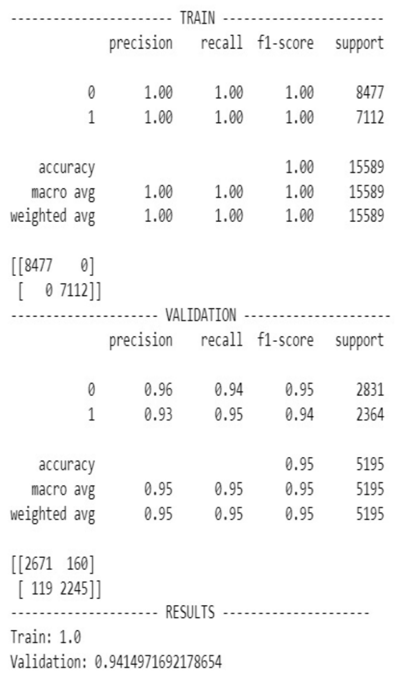
```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       0.97      0.94      0.96      8477
           1       0.93      0.97      0.95      7112

    accuracy                           0.95     15589
   macro avg       0.95      0.95      0.95     15589
weighted avg       0.95      0.95      0.95     15589

[[7974  503]
 [ 243 6869]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.96      0.92      0.94      2831
           1       0.91      0.96      0.93      2364

    accuracy                           0.94      5195
   macro avg       0.94      0.94      0.94      5195
weighted avg       0.94      0.94      0.94      5195

[[2608  223]
 [  97 2267]]
-------------------- RESULTS --------------------
Train: 0.9484948909141122
Validation: 0.9340749896992172
```

Figure 42: Neural Networks Model

Figure 43: Default Neural Networks Model

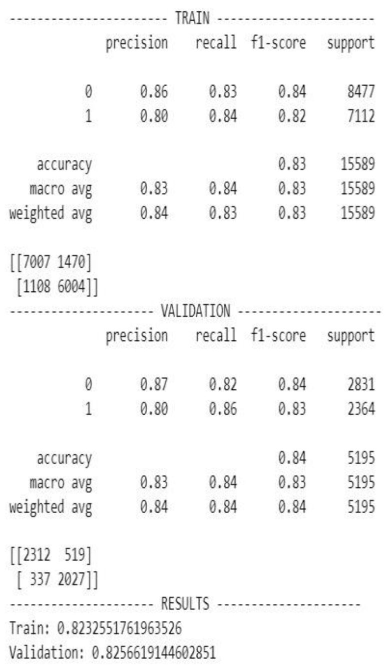


	Time	Train	Validation
Raw	14.146+/-1.5	0.95+/-0.0	0.939+/-0.0



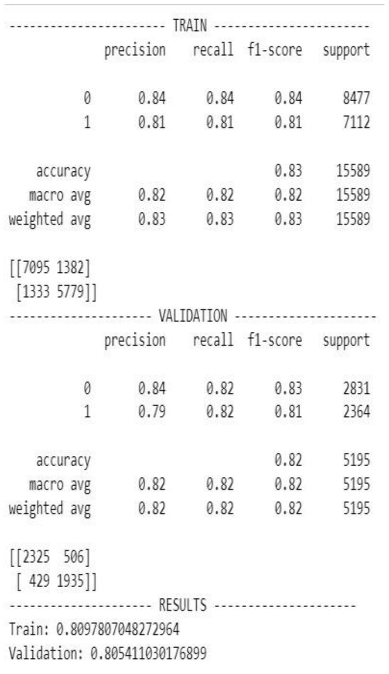
```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      8477
           1       1.00      1.00      1.00      7112

    accuracy                           1.00     15589
   macro avg       1.00      1.00      1.00     15589
weighted avg       1.00      1.00      1.00     15589

[[8477    0]
 [   0 7112]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.96      0.94      0.95      2831
           1       0.93      0.95      0.94      2364

    accuracy                           0.95      5195
   macro avg       0.95      0.95      0.95      5195
weighted avg       0.95      0.95      0.95      5195

[[2671  160]
 [ 119 2245]]
-------------------- RESULTS --------------------
Train: 1.0
Validation: 0.9414971692178654
```

Figure 44: Random Forest Model



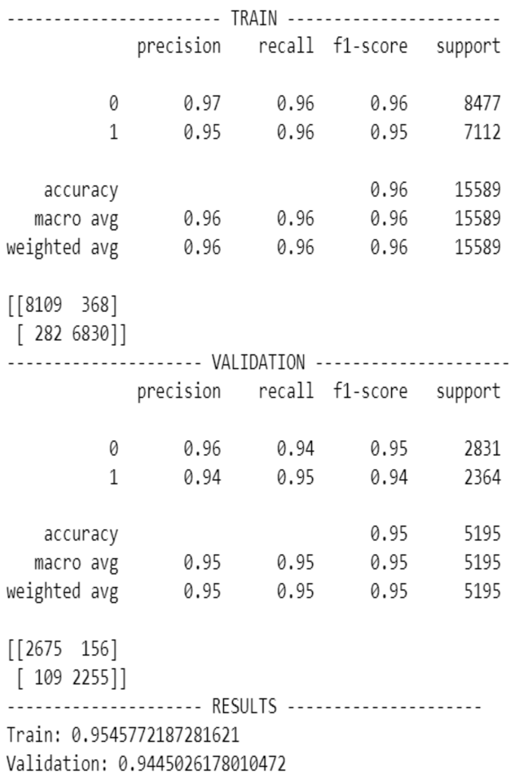
```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       0.86      0.83      0.84      8477
           1       0.80      0.84      0.82      7112

    accuracy                           0.83     15589
   macro avg       0.83      0.84      0.83     15589
weighted avg       0.84      0.83      0.83     15589

[[7007 1470]
 [1108 6004]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.87      0.82      0.84      2831
           1       0.80      0.86      0.83      2364

    accuracy                           0.84      5195
   macro avg       0.83      0.84      0.83      5195
weighted avg       0.84      0.84      0.84      5195

[[2312  519]
 [ 337 2027]]
-------------------- RESULTS --------------------
Train: 0.8232551761963526
Validation: 0.8256619144602851
```

Figure 45: Gradient Boosting Classifier Model



```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       0.84      0.84      0.84      8477
           1       0.81      0.81      0.81      7112

    accuracy                           0.83     15589
   macro avg       0.82      0.82      0.82     15589
weighted avg       0.83      0.83      0.83     15589

[[7095 1382]
 [1333 5779]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.84      0.82      0.83      2831
           1       0.79      0.82      0.81      2364

    accuracy                           0.82      5195
   macro avg       0.82      0.82      0.82      5195
weighted avg       0.82      0.82      0.82      5195

[[2325  506]
 [ 429 1935]]
-------------------- RESULTS --------------------
Train: 0.8097807048272964
Validation: 0.805411030176899
```

Figure 46: Naive Bayes Algorithm Model



```
---------------------- TRAIN ----------------------
              precision    recall  f1-score   support

           0       0.97      0.96      0.96      8477
           1       0.95      0.96      0.95      7112

    accuracy                           0.96     15589
   macro avg       0.96      0.96      0.96     15589
weighted avg       0.96      0.96      0.96     15589

[[8109  368]
 [ 282 6830]]
-------------------- VALIDATION --------------------
              precision    recall  f1-score   support

           0       0.96      0.94      0.95      2831
           1       0.94      0.95      0.94      2364

    accuracy                           0.95      5195
   macro avg       0.95      0.95      0.95      5195
weighted avg       0.95      0.95      0.95      5195

[[2675  156]
 [ 109 2255]]
-------------------- RESULTS --------------------
Train: 0.9545772187281621
Validation: 0.9445026178010472
```

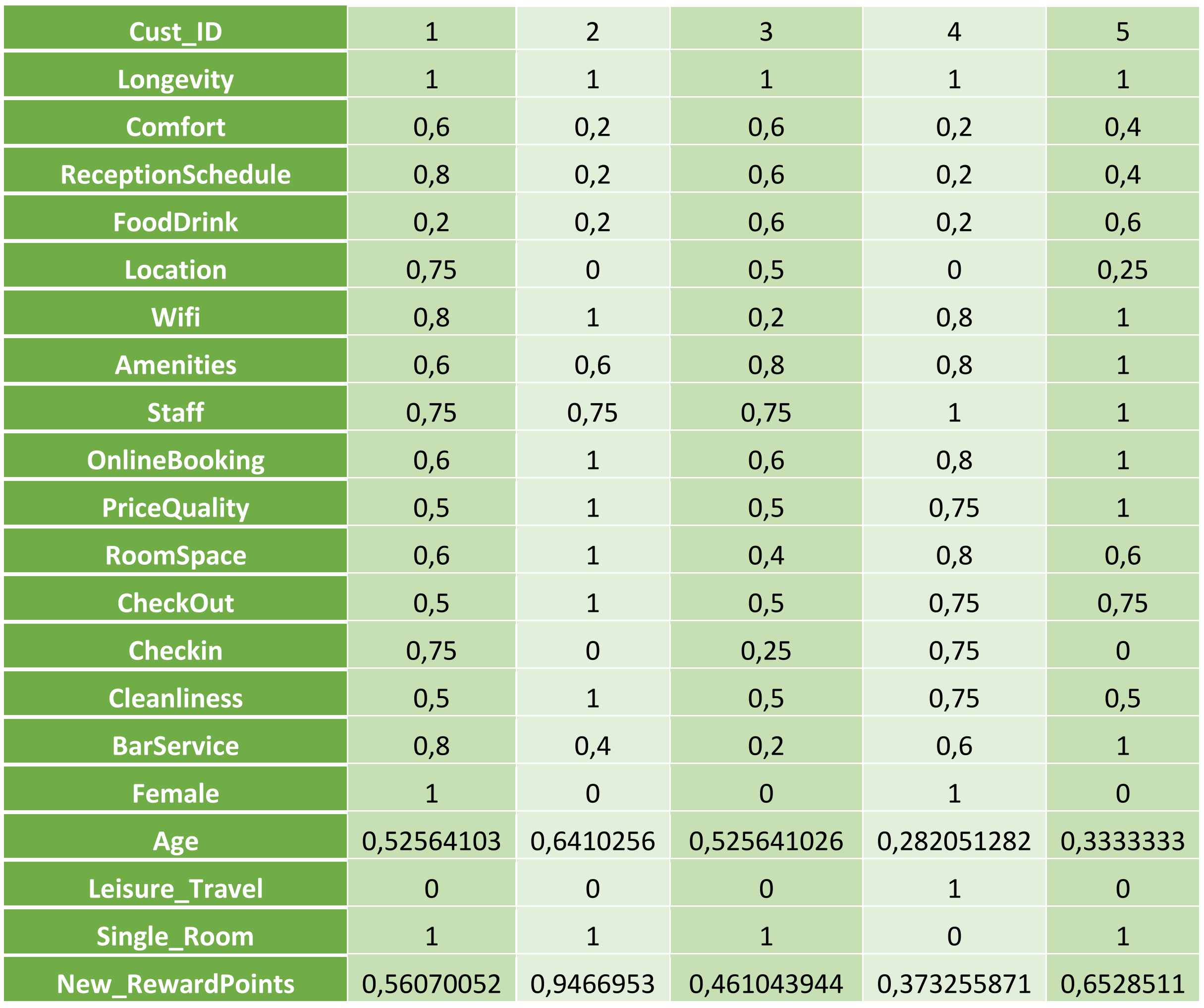
Figure 47: Histogram Gradient Boosting Model

Models	Train	Validation
Logistic Regression	0,8233	0,82566
Model K-Nearest Neighbours	0,9253	0,8992
Decision Trees	0,9457	0,9274
Neural Networks	0,9485	0,9341
Random Forest	1	0,9415
Gradient Boosting Classifier	0,8233	0,8257
Histogram Gradient Boosting Classifier	0,9546	0,9445
Naive Bayes Algorithm	0,8098	0,8054

Figure 48: Model Scores Results

Cust_ID	1	2	3	4	5
Churn	1	0	1	0	0
Longevity	1	1	1	1	1
Comfort	3	1	3	1	2
ReceptionSchedule	4	1	3	1	2
FoodDrink	1	1	3	1	3
Location	4	1	3	1	2
Wifi	4	5	1	4	5
Amenities	3	3	4	4	5
Staff	4	4	4	5	5
OnlineBooking	3	5	3	4	5
PriceQuality	3	5	3	4	5
RoomSpace	3	5	2	4	3
CheckOut	3	5	3	4	4
Checkin	4	1	2	4	1
Cleanliness	3	5	3	4	3
BarService	4	2	1	3	5
Female	1	0	0	1	0
Age	48	57	48	29	33
Leisure_Travel	0	0	0	1	0
Single_Room	1	1	1	0	1
New_RewardPoints	692533,4543	1161196,507	571533,5005	464943,917	804419,92

Figure 49: Original Train_c (transpose)



Cust_ID	1	2	3	4	5
Longevity	1	1	1	1	1
Comfort	0,6	0,2	0,6	0,2	0,4
ReceptionSchedule	0,8	0,2	0,6	0,2	0,4
FoodDrink	0,2	0,2	0,6	0,2	0,6
Location	0,75	0	0,5	0	0,25
Wifi	0,8	1	0,2	0,8	1
Amenities	0,6	0,6	0,8	0,8	1
Staff	0,75	0,75	0,75	1	1
OnlineBooking	0,6	1	0,6	0,8	1
PriceQuality	0,5	1	0,5	0,75	1
RoomSpace	0,6	1	0,4	0,8	0,6
CheckOut	0,5	1	0,5	0,75	0,75
Checkin	0,75	0	0,25	0,75	0
Cleanliness	0,5	1	0,5	0,75	0,5
BarService	0,8	0,4	0,2	0,6	1
Female	1	0	0	1	0
Age	0,52564103	0,6410256	0,525641026	0,282051282	0,3333333
Leisure_Travel	0	0	0	1	0
Single_Room	1	1	1	0	1
New_RewardPoints	0,56070052	0,9466953	0,461043944	0,373255871	0,6528511

Figure 50: Scaled Train C (transpose)

10. References

Data Science Blog. (n.d.). Retrieved from ANOVA using Python (with examples): https://www.reneshbedre.com/blog/anova.html

GeeksforGeeks. (2021, July 20). Retrieved from Python | Box-Cox Transformation: https://www.geeksforgeeks.org/box-cox-transformation-using-python/

insightsoftware. (2022, January 1). Retrieved from Top 5 Predictive Analytics Models and Algorithms: https://insightsoftware.com/blog/top-5-predictive-analytics-models-and-algorithms/

ODSC. (2019, June 24). Retrieved from Transforming Skewed Data for Machine Learning: https://opendatascience.com/transforming-skewed-data-for-machine-learning/

pandas. (n.d.). Retrieved from https://pandas.pydata.org/docs/reference/io.html

Scikit-learn. (n.d.). Retrieved from sklearn.neighbors.KNeighborsClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Scikit-Learn. (n.d.). Retrieved from sklearn.linear_model.LogisticRegression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Scikit-Learn. (n.d.). Retrieved from sklearn.neighbors.KNeighborsClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Scikit-Learn. (n.d.). Retrieved from Nearest Neighbors: https://scikit-learn.org/stable/modules/neighbors.html

Scikit-Learn. (n.d.). Retrieved from sklearn.neural_network.MLPClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Scikit-Learn. (n.d.). Retrieved from GradientBoostingClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

Scikit-Learn. (n.d.). Retrieved from sklearn.ensemble.RandomForestClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Scikit-Learn. (n.d.). Retrieved from Naive Bayes: https://scikit-learn.org/stable/modules/naive_bayes.html

Scikit-Learn. (n.d.). Retrieved from sklearn.model_selection.GridSearchCV: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Scikit-Learn. (n.d.). Retrieved from sklearn.ensemble.HistGradientBoostingClassifie: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html

seaborn. (n.d.). Retrieved from seaborn: statistical data visualization: https://seaborn.pydata.org/

Stackoverflow. (n.d.). Retrieved from https://stackoverflow.com/

w3schools. (n.d.). Retrieved from Matplotlib Pyplot: https://www.w3schools.com/python/matplotlib_pyplot.asp