Enhancing Airbnb booking prediction through Machine-Learning

Hemanth Rao Bompally Bharath Chowdary Nuthalapati Vishal Sakkari
University of Maryland, Baltimore County
Baltimore, USA

Abstract

In this study, we use several machine learning models to predict customer bookings on Airbnb. The Hotel Booking Demand Kaggle dataset was used in the learning process [1]. This dataset has many features, such as booking times, customer demographics, room types, and cancellation information. The initial data preprocessing steps included deleting duplicates, filling up the missing values, and encoding categorical variables. These steps are necessary for data preparation for different models. In addition, the data was explored using exploratory analysis to find patterns and distributions. Feature engineering was used to transform the categorical variables into a machine-readable format.
A comparison of results for the project was done among logistic regression, naive Bayes, k-nearest neighbors, decision trees, and a random forest model. The models were compared on metrics such as accuracy, precision, recall, and F1-score. To handle class imbalance, which can affect model performance, resampling techniques like SMOTE and ADASYN have been implemented [2] [3] [4]. The best model was the random forest model, which turned out to have the best overall performance across all evaluation metrics. The findings underline the importance of careful data preprocessing and advanced feature engineering to improve the predictive accuracy of machine learning models. This study offers meaningful insights that might enable the hospitality industry to improve booking systems to better meet customer needs and achieve higher levels of efficiency.

1. INTRODUCTION

Predicting customer bookings remains one of the most significant challenges Airbnb faces. The challenge merges the complexity of individual property characteristics, user preferences, and broad market trends. This paper is going to study this multifaceted problem by trying to gauge the efficacy of a good number of popular machine learning models that try to predict customer bookings on one of the most famous platforms for short-term vacation rentals.
Logistic Regression, Naive Bayes Classifier, and k-Nearest Neighbors are models picked for their perfect balance between simplicity, interpretability, and the ability to deal with diverse data types and interactions. To implement the models, we use the Scikit-learn library, which is one of the most used libraries inside the Python ecosystem for machine learning tasks.
After that, we will proceed with exploratory data analysis, identifying key patterns and relationships within the dataset to provide a better understanding of the dynamics of Airbnb bookings. The subsequent steps of feature engineering transform the data to make it more friendly to the algorithm of machine learning.
By evaluating the performance of these models with metrics such as accuracy, precision, recall, and the F1 score, we are contributing to the body of existing literature that is primarily academic but also providing a practical guideline for supporting Airbnb hosts and property managers in making strategic decisions regarding pricing, marketing, and managing the properties to achieve optimal booking rates and, consequently, profitability. Finally, the insights gained from this analysis can support the development of even more advanced predictive models in the future for enhanced decision-making in a highly competitive rental market.

2. DESCRIPTION OF THE DATASET

Nuno Antonio, Ana Almeida, and Luis Nunes created the data used in this project, which is from the Hotel Booking Demand dataset on Kaggle [1]. It includes booking details from 2015 to 2017 for two types of hotels: City Hotel and Resort Hotel. This dataset has 119,390 rows and 32 columns, making it a strong basis for analysing customer booking behaviours and industry trends.
The dataset [1] holds detailed information about the hotels, customers, and bookings. It covers variables such as hotel type, arrival and departure dates,

number of guests - adults, children, and babies, meal type, booking channels, and more. It also includes the booking status, which is the target for our predictive modelling and can be 'Cancelled', 'No-Show', or 'Check-Out'.
This dataset [1] is essential for predicting customer bookings and discovering behavioural patterns that can help hotel managers improve their operations. It has broad implications for academic and industrial research, offering insights into customer preferences and effective revenue management strategies.
The dataset contains the following features:

1. *hotel* - A categorical variable indicating which hotel the booking was made for.
2. *is cancelled* - A binary variable indicating whether the booking was cancelled (1) or not (0).
3. *lead time* - The no. of days between the date of booking and the arrival data.
4. arrival date year - The year of the arrival date.
5. *arrival date month* - The month of the arrival date.
6. *arrival date week number* - The week number of the ar- rival date.
7. *arrival date day of month* - The day of the month of the arrival date.
8. *stays on weekend nights* - The number of weekend nights (Saturday or Sunday) the guest stayed at the ho- tel.
9. *stays in week nights* - The number of week nights (Mon- day to Friday) the guest stayed at the hotel.
10. *adults* - The number of adults included in the booking.
11. *children* - The number of children included in the book- ing.
12. *babies* - The number of babies included in the booking.
13. *meal* - The type of meal included in the booking.
14. *country* - The country of origin of the guest.
15. *market segment* - The market segment the booking was made for. The possible values are ”Direct”, ”Corpo- rate”, ”Online TA”, ”Offline TA/TO”, ”Complementary”, ”Groups”, and ”Undefined”.
16. *distribution channel* - The distribution channel the book- ing was made through. The possible values are ”Direct”, ”Corporate”, ”TA/TO”, and ”Undefined”.
17. *is repeated guest* - A binary variable indicating whether the guest has stayed at the hotel before (1) or not (0).
18. *previous cancellations* - The number of previous book- ings that were cancelled by the guest before the current bookin
19. *previous bookings not cancelled* - The number of previ- ous bookings that were not cancelled by the guest before the current booking
20. *reserved room type* - The type of room reserved by the guest.
21. *assigned room type*- The type of room assigned to the guests
22. *booking changes* - The number of changes made to the booking before the guest arrived at the hotel.
23. *deposit type* - The type of deposit made by the guest. The possible values are ”No Deposit”, ”Non Refund”, and ”Refundable”.
24. *agent* - The ID of the travel agency that made the book- ing.
25. *company*- TheIDofthecompanyorentitythatmadethe booking.
26. *days in waiting list* - The number of days the booking was on the waiting list before it was confirmed to the guest.
27. *customer type* - The type of booking. The possible val- ues are ”Transient”, ”Contract”, ”Transient-Party”, and ”Group”.
28. *adr* - The average daily rate (ADR) of the booking.
29. *required car parking spaces* - Number of car parking spaces required by the customer.
30. *total of special requests* - Number of special requests made by the customer (e.g. for a specific room, bed, etc.).
31. *reservation status* - The last status of the reservation. Possible values are ’Cancelled’, ’Check-Out’, and ’No-Show’. ’Cancelled’ means the reservation was cancelled by the customer, ’Check-Out’ means the customer has checked out,

and 'No-Show' means the customer did not show up for the reservation.

32. *reservation status date* - Date when the last status was set.

3. DATA CLEANING AND PREPARATION

In our project, we processed the full Hotel Booking Demand dataset from Kaggle. This approach allows us to take advantage of all the available data for our analysis and maximises the insight gathered from our models.

In our cleaning of the data, there are some important features in which we have found missing values in the dataset. Null values for feature "Country" denote that booking does not include information regarding the nationality of the guest. We filled up these null values with "Unknown" For the "Agent" and "Company" features, we replaced the null values with 0, suggesting that there was no booking agent or company. The decision reflects the cases where bookings were made directly by the individuals without intermediary agents.

Also, we ran into duplicate rows, which could indicate data duplication or the recording of errors. We printed out the number of these kinds of rows and afterward removed them to make our dataset unique and maintain its integrity.

One specific problem that was fixed was the presence of rows where all guest counts, meaning adults, children, and babies, were zero. Such values could mean errors or incomplete entries. We identified and removed them since they likely don't represent valid bookings.

4. EXPLORATORY DATA ANALYSIS

We performed univariate and bivariate analysis to explore the dataset. For univariate analysis, we examined the distribution of numerical and categorical variables. For bivariate analysis, we explored the rela- tionship between different variables. We created choropleth maps and line plots to visualise the data.

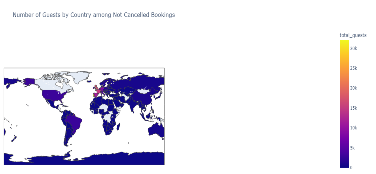

Fig 1: Display the number of guests by country among not cancelled bookings in a choropleth map

Fig 2: Price variation over time (trend) by hotel type among not cancelled bookings in a line plot

Fig 3: Price variation by year and hotel type among not cancelled bookings in a box plot

Fig 4: Number of bookings variation over time (seasonality) by hotel type among not cancelled bookings in a line plot.

Fig 5: Price variation by reserved room type among cancelled bookings in a box plot

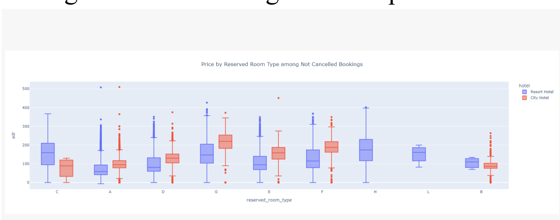

Fig 6: Price variation by reserved room type among not cancelled bookings in a box plot

We observed that the majority of bookings were made for City hotels as opposed to Resort hotels. We investigated the relationship between variables by creating choropleth maps and line plots. Our analysis revealed the following:

- Not cancelled bookings were more frequent than cancelled bookings in most countries. The highest number of bookings was made by guests from Portugal, followed by the UK, Spain, and France
- The average price per night was higher for Resort hotels than for City hotels. We also observed that the price per night increased over time for both hotel types, with seasonal fluctuations.
- The highest number of bookings was made for City hotels, followed by Resort hotels. We also observed that the number of bookings increased over time for both hotel types, with seasonal fluctuations.
- The price per night was higher for bookings made for rooms with higher levels of comfort and amenities.
- In addition, we found that most bookings were made by customers from Portugal, followed by guests from the UK and Spain.

5. FEATURE ENGINEERING

To enhance the performance of the model, we perform feature engineering techniques on the preprocessed data. The actions listed below were completed:

Splitting date columns: The year, month, and day were split out of the reservation status date column. Since it was no longer necessary, the original column was removed.

Correlation analysis: The numerical variables in the data were subjected to a correlation analysis. The plotly library was used to create a heatmap representation of the correlation matrix. The heatmap cells also showed the correlation values. The correlation values with the target variable were also plotted in a bar chart.

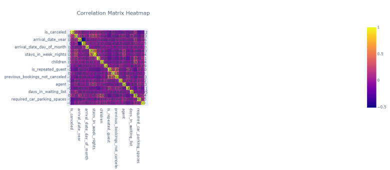

Fig 7: Perform correlation analysis on the numerical variables

Eliminating highly and poorly correlated variables: A high degree of correlation between the variables was found, and one of them was dropped.Additionally, variables that showed little correlation to the target variable were identified and eliminated. A list contained the list of pointless variables that needed to be removed.

Encoding categorical variables: The pandas library's get dummies() function was used to one-hot encode the categorical variables.

Normalisation: Min-max scaling was used to normalise the numerical variables, with the exception of the target variable. The scaling was done using each variable's minimum and maximum values. Additionally, each variable's variance was calculated and printed.

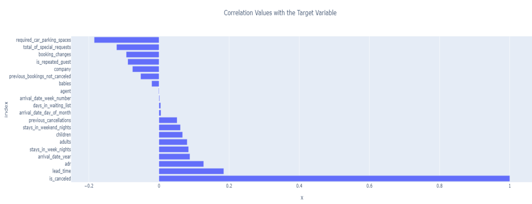

Fig 8: Correlation values with the target variable

The features of reservation status date, year, month, and day are highly correlated with arrival date year, arrival date week number, and arrival date day of month, respectively, according to the correlation matrix heatmap.

6. METHODOLOGIES

Logistic Regression :- Logistic Regression is a statistical technique for data analysis in which one or more independent variables mark an outcome variable. Measured on a dichotomous variable, the outcome is also known as a binary variable—one with only two possible outcomes. Logistic Regression is used extensively in various fields, including machine learning for binary classification problems. Logistic regression estimates the probabilities using a logistic function and is particularly helpful for understanding the contributions of several independent variables on one outcome variable.

Naive Bayes :- The Naive Bayes method is based on Bayes' Theorem, assuming that the predictors are independent of each other. In simple terms, a Naive

Bayes classifier considers that the presence of a feature in a class is unrelated to the presence of any other feature. This model is very easy to build and particularly useful for very large datasets. In addition to simplicity, Naive Bayes is known to outdo even highly sophisticated classification methods. It is particularly effective for prediction tasks that include text classification, spam filtering, and sentiment analysis.

KNN (k-nearest neighbors) :- k-Nearest Neighbors is one of the simplest, versatile, and easily implementable algorithms of supervised machine learning, applicable in both classification and regression tasks, although more often used in classification. It classifies a data point based on how its neighbors are classified. The algorithm finds the 'k' number of nearest data points around the query data point, where 'k' is a user-defined number; it then classifies the query point in accordance with the majority category or the mean of those neighbors. kNN is a non-parametric method, meaning that it does not make any assumptions about the distribution of data, making it flexible for use in real-world scenarios.

7. RESULTS

7.1 Model Performance Evaluation Metrics

To verify the reliability and efficiency of our predictive models, we have used a wide range of well-established evaluation metrics including accuracy, precision, recall, and the F1 score. These metrics capture a different meaning regarding the model's performance and are essential for multiple analytical purposes.

Accuracy: It reflects the general accuracy of the model. It is defined as the ratio between the number of correct predictions and the total number of predictions. Accuracy is highly valuable when the dataset includes well-balanced classes that are equally important.

Precision: It is another indicator of the general accuracy of the positive predictions. It is defined as the ratio of true positives to the true positive number and a number of false positives. Precision is critical to when the consequences of false positives are more severe.

Recall/recall/sensitivity: Indicates the model's ability to identify every instance that is available in the dataset. It is calculated as the ratio of true positives to false negatives and true positives. This is crucial because a false negative could be extremely harmful.

F1 score: The F1 score is an average of recall and precision. It is the harmonic average of these two metrics, to be more precise. When we wish to measure false positives and false negatives in-depth, this is especially beneficial.

These metrics were fundamental in our experiments; they provided a means to compare models under a number of conditions, such as different class distributions and data rebalancing techniques. We were able to make an end-to-end comparison of various models and methodologies, therefore defining the best suited for our particular analytical challenges.

7.2 Model Implementation and inferences

Logistic Regression: Logistic Regression is implemented using scikit-learn library in Python [3]. Logistic regression, with SMOTE resampling performed with accuracy at 77.26%, a precision of 75.62%, and a recall of 79.74%. The F1 Score was recorded at 77.62%, and the ROC AUC Score was slightly higher at 77.28%. That is a pretty good performance of the model, where it is good at distinguishing between the classes. In fact, for other class ratios—0.5, 0.33, 0.25, and 0.1—the performance metrics did not actually vary significantly, which showed great stability in the ability of the model to deal with class imbalances through SMOTE.

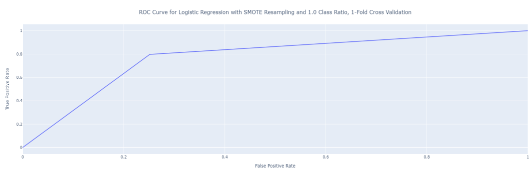

Figure 9: ROC Curve for Logistic Regression with SMOTE Resampling and 1.0 Class Ratio, 1-Fold Cross Validation.

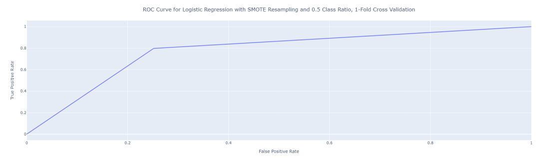

Figure 10: ROC Curve for Logistic Regression with SMOTE Resampling and 0.5 Class Ratio, 1-Fold Cross Validation

Figure 11: ROC Curve for Logistic Regression with ADASYN Resampling and 1.0 Class Ratio, 1-Fold Cross Validation

Figure 12: ROC Curve for Logistic Regression with ADASYN Resampling and 0.5 Class Ratio, 1-Fold Cross Validation

Results for SMOTE using 1.0 class ratio:

{'Confusion Matrix': Predicted 0 1
Actual
0 9563 3216
1 2535 9975, 'Accuracy':
0.7725888726323697, 'Precision':
0.7561974073231749, 'Recall':
0.7973621103117506, 'F1 Score':
0.7762343877670129, 'ROC AUC Score':
0.7728496129459996}

Results for SMOTE using a 0.5 class ratio:

{'Confusion Matrix': Predicted 0 1
Actual
0 9563 3216
1 2535 9975, 'Accuracy':
0.7725888726323697, 'Precision':
0.7561974073231749, 'Recall':
0.7973621103117506, 'F1 Score':
0.7762343877670129, 'ROC AUC Score':
0.7728496129459996}

Results for ADASYN using a 1.0 class ratio:

{'Confusion Matrix': Predicted 0 1
Actual
0 9239 3490
1 2619 9744, 'Accuracy':
0.7565359477124183, 'Precision':
0.7362853256762883, 'Recall':
0.788158214025722, 'F1 Score':
0.7613392194397781, 'ROC AUC Score':
0.7569905690287303}

Results for ADASYN using 0.5 class ratio:

{'Confusion Matrix': Predicted 0 1
Actual
0 9239 3490
1 2619 9744, 'Accuracy':
0.7565359477124183, 'Precision':
0.7362853256762883, 'Recall':
0.788158214025722, 'F1 Score':
0.7613392194397781, 'ROC AUC Score':
0.7569905690287303}

In contrast to that, resampling using ADASYN for a 1.0 class ratio provided slightly lesser performance figures: an accuracy of 75.65%, precision of 73.63%, and a recall of 78.82%. The F1 score was at 76.13%, with a ROC AUC score of 75.70%. This relative decrease in precision, as compared to SMOTE, would indicate that ADASYN may over-sample the dataset a bit, as it generates more aggressive synthetic samples with higher false positives, slightly decreasing the predictive efficiency of the model.

The comparison between SMOTE and ADASYN shows the slight superiority of SMOTE for resampling ratio, where it manages to retain both higher accuracy and higher precision. Both show the typical precision-recall tradeoff, with increases in recall usually comes at the cost of precision. This analysis suggests that SMOTE might be more applicable for an application where one needs the balance between false positives and false negatives, while ADASYN could be a preference if maximisation of true positives is essential at the cost of increased false positives.

The ROC curve visualisations further support these observations by showing the good trade-offs that the logistic regression model was able to manage within the different settings. These insights are important to the optimization of the application of the model for specific contexts, ensuring that the resampling technique chosen aligns with the predictive model's results.

Naive Bayes: Naive Bayes is implemented using scikit-learn library in Python [3]. The ROC curve for the Naive Bayes classifier using SMOTE resampling with a class ratio of 1.0, when used on a model that does pretty well, shows a smooth, balanced rise across the range of false positive rates. In this setting, it obtained around 69.77% accuracy, precision, recall, F1 score, and ROC AUC, all grouped around 67–72%. This may mean a relatively balanced performance in terms of both types of errors, indicating that the model is reasonably capable of distinguishing between the classes but far from perfect.

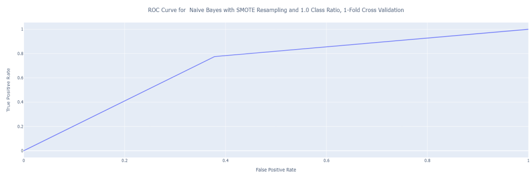

Figure 13: ROC Curve for Naive Bayes with SMOTE Resampling and 1.0 Class Ratio, 1-Fold Cross Validation.

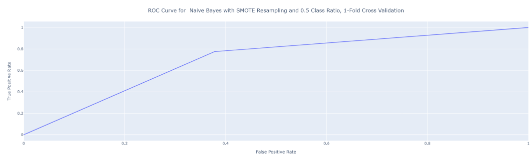

Figure 14: ROC Curve for Naive Bayes with SMOTE Resampling and 0.5 Class Ratio, 1-Fold Cross Validation.

Figure 15: ROC Curve for Naive Bayes with ADASYN Resampling and 1.0 Class Ratio, 1-Fold Cross Validation.

Figure 16: ROC Curve for Naive Bayes with ADASYN Resampling and 0.5 Class Ratio, 1-Fold Cross Validation.

Results for SMOTE using a 1.0 class ratio:
{'Confusion Matrix': Predicted 0 1
Actual
0 7946 4833
1 2811 9699, 'Accuracy':
0.6977341927320179, 'Precision':
0.6674236168455822, 'Recall':
0.7752997601918465, 'F1 Score':
0.7173285999556246, 'ROC AUC Score':
0.698550576551045}

Results for SMOTE using 0.5 class ratio:
{'Confusion Matrix': Predicted 0 1
Actual
0 7946 4833
1 2811 9699, 'Accuracy':
0.6977341927320179, 'Precision':
0.6674236168455822, 'Recall':
0.7752997601918465, 'F1 Score':
0.7173285999556246, 'ROC AUC Score':
0.698550576551045}

Results for ADASYN using a 1.0 class ratio:
{'Confusion Matrix': Predicted 0 1
Actual
0 7760 4969
1 3278 9085, 'Accuracy':
0.6713295074127212, 'Precision':
0.6464351785968407, 'Recall':
0.734853999838227, 'F1 Score':
0.6878146647991823, 'ROC AUC Score':
0.6722427749210775}

Results for ADASYN using 0.5 class ratio:
{'Confusion Matrix': Predicted 0 1
Actual
0 7760 4969
1 3278 9085, 'Accuracy':
0.6713295074127212, 'Precision':
0.6464351785968407, 'Recall':
0.734853999838227, 'F1 Score':

0.6878146647991823, 'ROC AUC Score': 0.6722427749210775}

A decrease in the class ratio with SMOTE resampling shows consistency in the metrics of accuracy, precision, recall, F1 score, and ROC AUC, suggesting that class ratio modifications do not seem to have a strong effect on the performance of the Naive Bayes classifier in this case. All these setups mirror the performance metrics of the 1.0 class ratio closely, suggesting that the Naive Bayes model maintains its classification behaviour across different levels of class balance achieved by SMOTE.
Even at the lower class ratio of 0.1, the performance metrics of the Naive Bayes classifier do not fluctuate too much. This shows that the robustness to the variation of class ratio perhaps means Naive Bayes, with the assumption of feature independence given class, might be insensitive to the class imbalance given the specific dataset and the features involved.
This could be due to the fact that the performance in metrics from the use of ADASYN resampling is slightly lower, as compared to SMOTE, for all class ratios. The accuracy, precision, recall, F1 score, and ROC AUC scores are lower by approximately 1-2 percentage points across these setups from their respective SMOTE setups. This could be an indication that the synthetic samples generated by ADASYN, which focuses more on generating samples next to the original samples that are hard to classify, do not align well with the Naive Bayes model's expectations and assumptions.

Overall, the Naive Bayes classifier has shown moderate effectiveness in handling this classification task with both SMOTE and ADASYN resampling techniques [2] [3]. The relative consistency among class ratios suggests that, within the boundaries of this dataset and the assumptions behind Naive Bayes, the effect of changing class balance is relatively minimal on model generalisation capacity. This may be useful in real-world applications where class distribution is unknown or variable, but slightly better performance with SMOTE indicates a slight preference toward its method of synthetic sample generation for this particular model and dataset.

K Nearest neighbors: K Nearest neighbors is implemented using scikit-learn library in Python [3]. Using SMOTE, the KNN model presented approximately 86.04% accuracy, a precision of 76.04%, and a recall of 89.84%. The F1 score was approximately 0.819, and the ROC AUC was around 0.837; hence, precision and recall were well balanced. These metrics were consistent across the class ratios, ranging from 1.0 to 0.25, indicating that the class distribution skew is properly handled by SMOTE.

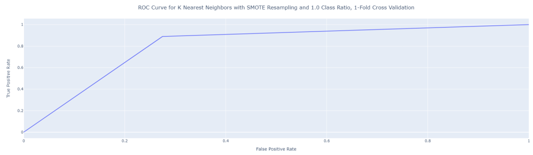

Figure 17: ROC Curve for K Nearest Neighbors with SMOTE Resampling and 1.0 Class Ratio, 1-Fold Cross Validation.

Figure 18: ROC Curve for K Nearest Neighbors with SMOTE Resampling and 0.5 Class Ratio, 1-Fold Cross Validation.

Figure 19: ROC Curve for K Nearest Neighbors with ADASYN Resampling and 1.0 Class Ratio, 1-Fold Cross Validation.

Figure 20: ROC Curve for K Nearest Neighbors with ADASYN Resampling and 0.5 Class Ratio, 1-Fold Cross Validation.

Results for SMOTE using 1.0 class ratio:
{'Confusion Matrix': Predicted 0 1
Actual
0 9268 3511
1 1383 11127, 'Accuracy': 0.8064771244414568, 'Precision':

0.7601448285284875, 'Recall':
0.8894484412470024, 'F1 Score':
0.8197288934728157, 'ROC AUC Score':
0.8073504042059411}

Results for SMOTE using 0.5 class ratio:
{'Confusion Matrix': Predicted 0 1
Actual
0 9268 3511
1 1383 11127, 'Accuracy':
0.8064771244414568, 'Precision':
0.7601448285284875, 'Recall':
0.8894484412470024, 'F1 Score':
0.8197288934728157, 'ROC AUC Score':
0.8073504042059411}

Results for ADASYN using 1.0 class ratio:
{'Confusion Matrix': Predicted 0 1
Actual
0 8613 4116
1 1008 11355, 'Accuracy':
0.795791487326638, 'Precision':
0.7339538491370952, 'Recall':
0.9184663916525115, 'F1 Score':
0.8159086009915929, 'ROC AUC Score':
0.7975551378484099}

Results for ADASYN using 0.5 class ratio:
{'Confusion Matrix': Predicted 0 1
Actual
0 8613 4116
1 1008 11355, 'Accuracy':
0.795791487326638, 'Precision':
0.7339538491370952, 'Recall':
0.9184663916525115, 'F1 Score':
0.8159086009915929, 'ROC AUC Score':
0.7975551378484099}

In contrast, ADASYN adaptive synthetic sampling with a focus on sample generation in the neighbourhood of the borderline of the minority class showed different results. The model, created using ADASYN for the class ratio of 1.0, was able to achieve an accuracy of 75.97%, precision of 73.39%, and a recall of 91.64%. The higher recall compared to precision indicates that the model, while being sensitive to detecting the positive class, made more false positive errors, which is reflected by a lower precision. The F1 score was about 0.816, and the ROC AUC was 0.795, thus showing a slight decline compared to SMOTE.

These analyses indicate that, in general, the KNN with the SMOTE resampling method outperforms that of the ADASYN approach in maintaining high precision with not much loss in terms of recall. The constant performance of SMOTE under different class ratios could reflect its robustness to not easily overfit synthetic samples. In contrast, the lower performance of ADASYN may be said to mean that the location of its synthetic samples might not always be optimal with respect to KNN's locality-based decision-making.

8.CONCLUSION AND FUTURE WORK

In our project, we experimented with Logistic Regression, Naive Bayes, and K-Nearest neighbors models, coupled with SMOTE and ADASYN resampling techniques to handle the inherent class imbalance within the dataset.

Logistic Regression, when enhanced by the SMOTE resampling technique, resulted in an approximately 77% accuracy, 76% precision, and 80% recall. The metrics presented here evidence a balanced performance, with the model showing robustness on different class distributions. Specifically, the consistency in the different SMOTE ratios is very promising regarding the stability of the model against class imbalance.

Naive Bayes yielded some interesting results, above all notable for the consistent performance across the different class ratios. For example, in the 1.0 class ratio case with SMOTE, the model maintained an accuracy around 70%, with both precision and recall clustered around the 70% mark. This consistency is reflective of the Naive Bayes suitability for applications where class distributions might change, which infers that the predictive capability for this algorithm should be reliable regardless of underlying data skew.

The K-Nearest Neighbors technique, when the SMOTE resampling was employed with a 1.0 class ratio, produced an accuracy of around 86% with a 76% precision and 90% recall. The results above show that this model is effective in maximising the true positive rates, which is critical for operational scenarios, such as booking prediction, where capturing as many actual bookings as possible is important.

Summarising, the application of SMOTE generally gave better precision and balanced recall rates across the models tested and, by this, suggests its effectiveness in our predictive tasks over ADASYN. These findings thus underline the importance of selecting the right resampling techniques to improve model performance on unbalanced datasets and give a foundational approach to improving booking prediction systems in the hospitality industry.

Looking ahead, there is potential to expand on this predictive modelling capability. We plan to increase predictive accuracy with more advanced machine learning models, such as deep learning and ensemble models. The advanced models will most definitely capture the complex and nonlinear relationships that may have been left out by simpler models, hence possibly netting a tremendous gain in predictive performance. Integrating real-time data from online booking platforms and social media sentiment with economic indicators can redefine our strategy toward a dynamic prediction model. Such a model would reflect not only current market conditions but also changes in consumer behaviour and economic landscapes, hence being more responsive and accurate in forecasting. This is part of an ongoing strategy to refine our understanding of the booking patterns of our customers in order to drive more informed decisions and strategic planning within the hospitality industry.

9. REFERENCES

1. Mostipak, Jesse. "Hotel Booking Demand." *Www.kaggle.com*, 2020, www.kaggle.com/datasets/jessemostipak/hotel-booking-demand
2. Haibo He, et al. "ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning." *IEEE Xplore*, 1 June 2008, ieeexplore.ieee.org/abstract/document/4633969?casa_token=J_CENnbbg04AAAAA:H66LkaQgQseWdiBmYNy3Puy0nrHpFfZ7OA3Io7ZXVSCE-0_bXw-pmblGkrE7HoIISMjkQqG7Ng.
3. Pedregosa, Fabian, et al. "Scikit-Learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT et AL. Matthieu Perrot Edouard Duchesnay." *Journal of Machine Learning Research*, vol. 12, 2011, pp. 2825–2830, www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf
4. Chawla, N. V., et al. "SMOTE: Synthetic Minority Over-Sampling Technique." *Journal of Artificial Intelligence Research*, vol. 16, no. 16, 1 June 2002, pp. 321–357, www.jair.org/index.php/jair/article/view/10302, https://doi.org/10.1613/jair.953.
5. He, Haibo, and Yunqian Ma. *Imbalanced Learning*. John Wiley & Sons, 7 June 2013.

10. DESCRIPTION OF CODE

10.1 Link to code:
https://colab.research.google.com/drive/1z8ivpD14UZKi25TBzFyXlJGaf25kY9Q-?usp=sharing

10.2 Code:

```
# File Handling
import os

# Data manipulation
import pandas as pd
# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
# Resampling
from imblearn.over_sampling import
RandomOverSampler, SMOTE, ADASYN
# SciKit-Learn implementations
from sklearn.linear_model import
LogisticRegression as
SciKitLearnLogisticRegression
from sklearn.naive_bayes import MultinomialNB
as SciKitLearnNaiveBayes
from sklearn.neighbors import
KNeighborsClassifier as
SciKitLearnKNearestNeighbors
from sklearn.tree import DecisionTreeClassifier as
SciKitLearnDecisionTree
from sklearn.ensemble import
RandomForestClassifier as
SciKitLearnRandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
as SciKitLearnAdaBoostClassifier


# Evaluation metrics
```

```
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score,
roc_auc_score, roc_curve

# K-Fold Cross Validation
from sklearn.model_selection import
GridSearchCV, KFold, train_test_split,
cross_val_score
file_path = "hotel_bookings.csv"
df = pd.read_csv(file_path)
df.head()
df.tail()
print("The number of duplicate rows are: ")
print(df.duplicated().sum())
print("-" * 100)
df.drop_duplicates(inplace=True)
print(df.duplicated().sum())
df['children'].fillna(0, inplace=True)
# For country column, replace missing values with
'Unknown'
df['country'].fillna('Unknown', inplace=True)
# For agent and company columns, replace missing
values with 0
df['agent'].fillna(0, inplace=True)
df['company'].fillna(0, inplace=True)
# For all remaining rows with missing values, drop
the row
df.dropna(inplace=True)
zero_guests_mask = (df['adults'] == 0) &
(df['children'] == 0) & (df['babies'] == 0)
zero_guests_rows = df[zero_guests_mask]
# Print the number of rows where all guests are 0
print("The number of rows where all guests are 0
are: ")
print(zero_guests_rows.shape[0])
df.drop(zero_guests_rows.index, inplace=True)
zero_guests_mask = (df['adults'] == 0) &
(df['children'] == 0) & (df['babies'] == 0)
zero_guests_rows = df[zero_guests_mask]
# Print the number of rows where all guests are 0
print("The number of rows where all guests are 0
are: ")
print(zero_guests_rows.shape[0])
df.head()
df.tail()
# Distribution of numerical variables
# Get numerical columns
numerical_columns =
df.select_dtypes(include=['int64',
'float64']).columns
# Loop through numerical columns
for col in numerical_columns:
    # Get the range of data in the column
    data_range = df[col].max() - df[col].min()
    # Set the number of bins as the range of data
    num_bins = int(data_range) + 1
    # Create histogram
    plt.figure(figsize=(8, 6))
    sns.histplot(df[col], bins=num_bins, kde=True)
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.title(f'Distribution of {col}')
    plt.show()
# Distribution of categorical variables
categorical_columns =
df.select_dtypes(include=['object']).columns
for col in categorical_columns:
    plt.figure(figsize=(8, 6))
    sns.countplot(data=df, x=col)
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.title(f'Count of {col}')
    plt.show()
# Bivariate Analysis

# Display the number of guests by country among
not cancelled bookings in a choropleth map
# Filter the data to only include rows where
is_canceled is 0
not_canceled_mask = df['is_canceled'] == 0
not_canceled_df = df[not_canceled_mask]
# Group the data by country and sum the number of
guests
guests_by_country =
not_canceled_df.groupby('country')[['adults',
'children', 'babies']].sum()
# Reset the index
guests_by_country.reset_index(inplace=True)
# Rename the columns
guests_by_country.rename(columns={'adults':
'total_adults', 'children': 'total_children', 'babies':
'total_babies'}, inplace=True)
# Create a column for total number of guests
guests_by_country['total_guests'] =
guests_by_country['total_adults'] +
guests_by_country['total_children'] +
guests_by_country['total_babies']
# Sort the data by total number of guests
guests_by_country.sort_values(by='total_guests',
ascending=False, inplace=True)
# Create a choropleth map
guests_map = px.choropleth(guests_by_country,
locations='country',
color='total_guests',hover_name='country',
```

```
color_continuous_scale=px.colors.sequential.Plasma)
# Add a title
guests_map.update_layout(title_text='Number of Guests by Country among Not Cancelled Bookings', title_x=0.5)
# Display the map
guests_map.show()
# Display the number of guests by country among cancelled bookings in a choropleth map
# Filter the data to only include rows where is_canceled is 1
canceled_mask = df['is_canceled'] == 1
canceled_df = df[canceled_mask]
# Group the data by country and sum the number of guests
guests_by_country = canceled_df.groupby('country')[['adults', 'children', 'babies']].sum()
# Reset the index
guests_by_country.reset_index(inplace=True)
# Rename the columns
guests_by_country.rename(columns={'adults': 'total_adults', 'children': 'total_children', 'babies': 'total_babies'}, inplace=True)
# Create a column for total number of guests
guests_by_country['total_guests'] = guests_by_country['total_adults'] + guests_by_country['total_children'] + guests_by_country['total_babies']
# Sort the data by total number of guests
guests_by_country.sort_values(by='total_guests', ascending=False, inplace=True)
# Create a choropleth map
guests_map = px.choropleth(guests_by_country, locations='country', color='total_guests',hover_name='country', color_continuous_scale=px.colors.sequential.Plasma)
# Add a title
guests_map.update_layout(title_text='Number of Guests by Country among Cancelled Bookings', title_x=0.5)
# Display the map
guests_map.show()
# Price variation over time (trend) by hotel type among not cancelled bookings in a line plot
# Filter the data to only include rows where is_canceled is 0
not_canceled_mask = df['is_canceled'] == 0
not_canceled_df = df[not_canceled_mask]
# Group the data by hotel and arrival_date_year and get the average price
price_by_year = not_canceled_df.groupby(['hotel', 'arrival_date_year'])[['adr']].mean()
# Reset the index
price_by_year.reset_index(inplace=True)
# Rename the columns
price_by_year.rename(columns={'adr': 'average_price'}, inplace=True)
# Create a line plot of price by year and hotel type
price_by_year_line_plot = px.line(price_by_year, x='arrival_date_year',y='average_price', color='hotel', title='Average Price by Year and Hotel Type')
# Add a title
price_by_year_line_plot.update_layout(title_text='Average Price by Year and Hotel Type among Not Cancelled Bookings', title_x=0.5)
# Display the plot
price_by_year_line_plot.show()
# Price variation over time (trend) by hotel type among cancelled bookings in a line plot
# Filter the data to only include rows where is_canceled is 1
canceled_mask = df['is_canceled'] == 1
canceled_df = df[canceled_mask]
# Group the data by hotel and arrival_date_year and get the average price
price_by_year = canceled_df.groupby(['hotel', 'arrival_date_year'])[['adr']].mean()
# Reset the index
price_by_year.reset_index(inplace=True)
# Rename the columns
price_by_year.rename(columns={'adr': 'average_price'}, inplace=True)
# Create a line plot of price by year and hotel type
price_by_year_line_plot = px.line(price_by_year, x='arrival_date_year',y='average_price', color='hotel', title='Average Price by Year and Hotel Type')
# Add a title
price_by_year_line_plot.update_layout(title_text='Average Price by Year and Hotel Type among Cancelled Bookings', title_x=0.5)
# Display the plot
price_by_year_line_plot.show()
# Price variation over time (seasonality) by hotel type among not cancelled bookings in a line plot
# Filter the data to only include rows where is_canceled is 0
not_canceled_mask = df['is_canceled'] == 0
not_canceled_df = df[not_canceled_mask]
```

```
# Group the data by hotel and arrival_date_month
and get the average price
price_by_month =
not_canceled_df.groupby(['hotel',
'arrival_date_month'])[['adr']].mean()
# Reset the index
price_by_month.reset_index(inplace=True)
# Rename the columns
price_by_month.rename(columns={'adr':
'average_price'}, inplace=True)
# Define the order of the months
month_order = ['January', 'February', 'March',
'April', 'May', 'June','July', 'August', 'September',
'October', 'November', 'December']
# Sort the data by arrival_date_month
price_by_month['arrival_date_month'] =
pd.Categorical(price_by_month['arrival_date_mont
h'], categories=month_order, ordered=True)
price_by_month.sort_values(by='arrival_date_mon
th', inplace=True)
# Create a line plot of average price vs month and
hotel type
price_by_month_line_plot =
px.line(price_by_month,
x='arrival_date_month',y='average_price',
color='hotel', title='Average Price by Month and
Hotel Type')
# Add a title
price_by_month_line_plot.update_layout(title_text
='Average Price by Month and Hotel Type among
Not Cancelled Bookings', title_x=0.5)
# Display the plot
price_by_month_line_plot.show()
# Price variation over time (seasonality) by hotel
type among cancelled bookings in a line plot
# Filter the data to only include rows where
is_canceled is 1
canceled_mask = df['is_canceled'] == 1
canceled_df = df[canceled_mask]
# Group the data by hotel and arrival_date_month
and get the average price
price_by_month = canceled_df.groupby(['hotel',
'arrival_date_month'])[['adr']].mean()
# Reset the index
price_by_month.reset_index(inplace=True)
# Rename the columns
price_by_month.rename(columns={'adr':
'average_price'}, inplace=True)
# Define the order of the months
month_order = ['January', 'February', 'March',
'April', 'May', 'June','July', 'August', 'September',
'October', 'November', 'December']
# Sort the data by arrival_date_month
price_by_month['arrival_date_month'] =
pd.Categorical(price_by_month['arrival_date_mont
h'], categories=month_order, ordered=True)
price_by_month.sort_values(by='arrival_date_mon
th', inplace=True)
# Create a line plot of average price vs month and
hotel type
price_by_month_line_plot =
px.line(price_by_month,
x='arrival_date_month',y='average_price',
color='hotel', title='Average Price by Month and
Hotel Type')
# Add a title
price_by_month_line_plot.update_layout(title_text
='Average Price by Month and Hotel Type among
Cancelled Bookings', title_x=0.5)
# Display the plot
price_by_month_line_plot.show()
# Price variation by year and hotel type among not
cancelled bookings in a box plot
# Filter the data to only include rows where
is_canceled is 0
not_canceled_mask = df['is_canceled'] == 0
not_canceled_df = df[not_canceled_mask]
# Create a box plot of price vs hotel type
price_by_hotel_box_plot =
px.box(not_canceled_df, x='arrival_date_year',
y='adr', color='hotel', title='Price by Hotel Type')
# Add a title
price_by_hotel_box_plot.update_layout(title_text='
Price by Hotel Type among Not Cancelled
Bookings', title_x=0.5)
# Display the plot
price_by_hotel_box_plot.show()
# Price variation by year and hotel type among
cancelled bookings in a box plot
# Filter the data to only include rows where
is_canceled is 1
canceled_mask = df['is_canceled'] == 1
canceled_df = df[canceled_mask]
# Create a box plot of price vs hotel type
price_by_hotel_box_plot = px.box(canceled_df,
x='arrival_date_year', y='adr', color='hotel',
title='Price by Hotel Type')
# Add a title
price_by_hotel_box_plot.update_layout(title_text='
Price by Hotel Type among Cancelled Bookings',
title_x=0.5)
# Display the plot
price_by_hotel_box_plot.show()
```

# Number of bookings variation over time (trend) by hotel type among not cancelled bookings in a line plot
# Filter the data to only include rows where is_canceled is 0
not_canceled_mask = df['is_canceled'] == 0
not_canceled_df = df[not_canceled_mask]
# Group the data by hotel and arrival_date_year and get the number of bookings
bookings_by_year = not_canceled_df.groupby(['hotel', 'arrival_date_year'])[['is_canceled']].count()
# Reset the index
bookings_by_year.reset_index(inplace=True)
# Rename the columns
bookings_by_year.rename(columns={'is_canceled': 'number_of_bookings'}, inplace=True)
# Create a line plot of number of bookings by year and hotel type
bookings_by_year_line_plot = px.line(bookings_by_year, x='arrival_date_year',y='number_of_bookings', color='hotel', title='Number of Bookings by Year and Hotel Type')
# Add a title
bookings_by_year_line_plot.update_layout(title_text='Number of Bookings by Year and Hotel Type among Not Cancelled Bookings', title_x=0.5)
# Display the plot
bookings_by_year_line_plot.show()
# Number of bookings variation over time (trend) by hotel type among cancelled bookings in a line plot
# Filter the data to only include rows where is_canceled is 1
canceled_mask = df['is_canceled'] == 1
canceled_df = df[canceled_mask]
# Group the data by hotel and arrival_date_year and get the number of bookings
bookings_by_year = canceled_df.groupby(['hotel', 'arrival_date_year'])[['is_canceled']].count()
# Reset the index
bookings_by_year.reset_index(inplace=True)
# Rename the columns
bookings_by_year.rename(columns={'is_canceled': 'number_of_bookings'}, inplace=True)
# Create a line plot of number of bookings by year and hotel type
bookings_by_year_line_plot = px.line(bookings_by_year, x='arrival_date_year',y='number_of_bookings', color='hotel', title='Number of Bookings by Year and Hotel Type')
# Add a title
bookings_by_year_line_plot.update_layout(title_text='Number of Bookings by Year and Hotel Type among Cancelled Bookings', title_x=0.5)
# Display the plot
bookings_by_year_line_plot.show()
# Number of bookings variation over time (seasonality) by hotel type among not cancelled bookings in a line plot
# Filter the data to only include rows where is_canceled is 0
not_canceled_mask = df['is_canceled'] == 0
not_canceled_df = df[not_canceled_mask]
# Group the data by hotel and arrival_date_month and get the number of bookings
bookings_by_month = not_canceled_df.groupby(['hotel', 'arrival_date_month'])[['is_canceled']].count()
# Reset the index
bookings_by_month.reset_index(inplace=True)
# Rename the columns
bookings_by_month.rename(columns={'is_canceled': 'number_of_bookings'}, inplace=True)
# Define the order of the months
month_order = ['January', 'February', 'March', 'April', 'May', 'June','July', 'August', 'September', 'October', 'November', 'December']
# Sort the data by arrival_date_month
bookings_by_month['arrival_date_month'] = pd.Categorical(bookings_by_month['arrival_date_month'], categories=month_order, ordered=True)
bookings_by_month.sort_values(by='arrival_date_month', inplace=True)
# Create a line plot of number of bookings vs month and hotel type
bookings_by_month_line_plot = px.line(bookings_by_month, x='arrival_date_month',y='number_of_bookings', color='hotel', title='Number of Bookings by Month and Hotel Type')
# Add a title
bookings_by_month_line_plot.update_layout(title_text='Number of Bookings by Month and Hotel Type among Not Cancelled Bookings', title_x=0.5)
# Display the plot
bookings_by_month_line_plot.show()
# Number of bookings variation over time (seasonality) by hotel type among cancelled bookings in a line plot

```
# Filter the data to only include rows where
is_canceled is 1
canceled_mask = df['is_canceled'] == 1
canceled_df = df[canceled_mask]
# Group the data by hotel and arrival_date_month
and get the number of bookings
bookings_by_month =
canceled_df.groupby(['hotel',
'arrival_date_month'])[['is_canceled']].count()
# Reset the index
bookings_by_month.reset_index(inplace=True)
# Rename the columns
bookings_by_month.rename(columns={'is_cancele
d': 'number_of_bookings'}, inplace=True)
# Define the order of the months
month_order = ['January', 'February', 'March',
'April', 'May', 'June','July', 'August', 'September',
'October', 'November', 'December']
# Sort the data by arrival_date_month
bookings_by_month['arrival_date_month'] =
pd.Categorical(bookings_by_month['arrival_date_
month'], categories=month_order, ordered=True)
bookings_by_month.sort_values(by='arrival_date_
month', inplace=True)
# Create a line plot of number of bookings vs
month and hotel type
bookings_by_month_line_plot =
px.line(bookings_by_month,
x='arrival_date_month',y='number_of_bookings',
color='hotel', title='Number of Bookings by Month
and Hotel Type')
# Add a title
bookings_by_month_line_plot.update_layout(title_
text='Number of Bookings by Month and Hotel
Type among Cancelled Bookings', title_x=0.5)
# Display the plot
bookings_by_month_line_plot.show()
# Price variation by reserved room type among not
cancelled bookings in a box plot
# Filter the data to only include rows where
is_canceled is 0
not_canceled_mask = df['is_canceled'] == 0
not_canceled_df = df[not_canceled_mask]
# Create a box plot of price vs reserved room type
price_by_room_type_box_plot =
px.box(not_canceled_df, x='reserved_room_type',
y='adr', color='hotel', title='Price by Reserved
Room Type')
# Add a title
price_by_room_type_box_plot.update_layout(title_
text='Price by Reserved Room Type among Not
Cancelled Bookings', title_x=0.5)
# Display the plot
price_by_room_type_box_plot.show()
# Price variation by reserved room type among
cancelled bookings in a box plot
# Filter the data to only include rows where
is_canceled is 1
canceled_mask = df['is_canceled'] == 1
canceled_df = df[canceled_mask]
# Create a box plot of price vs reserved room type
price_by_room_type_box_plot =
px.box(canceled_df, x='reserved_room_type',
y='adr', color='hotel', title='Price by Reserved
Room Type')
# Add a title
price_by_room_type_box_plot.update_layout(title_
text='Price by Reserved Room Type among
Cancelled Bookings', title_x=0.5)
# Display the plot
price_by_room_type_box_plot.show()
# Convert "reservation_status_date" to separate
columns for year, month, and day
df['reservation_status_date'] =
pd.to_datetime(df['reservation_status_date'])
df['reservation_status_date_year'] =
df['reservation_status_date'].dt.year
df['reservation_status_date_month'] =
df['reservation_status_date'].dt.month
df['reservation_status_date_day'] =
df['reservation_status_date'].dt.day
# Drop the original "reservation_status_date"
column
df.drop('reservation_status_date', axis=1,
inplace=True)

# Perform correlation analysis on the numerical
variables
numerical_variables =
df.select_dtypes(include=['int64',
'float64']).columns
# Display the correlation matrix in a heatmap with
the correlation values in the cells
correlation_matrix_heatmap =
px.imshow(df[numerical_variables].corr())
# Display the correlation values in the heatmap
cells
for i in range(len(numerical_variables)):
  for j in range(len(numerical_variables)):
    text =
correlation_matrix_heatmap.data[0].z[i][j]
    correlation_matrix_heatmap.add_annotation(
      x=j, y=i, text=round(text, 2),
showarrow=False)
```

```
# Add a title
correlation_matrix_heatmap.update_layout(title_text='Correlation Matrix Heatmap', title_x=0.5)
# Display the plot
correlation_matrix_heatmap.show()

# Displaying the correlation values with the target variable
# Get the correlation values with the target variable
correlation_values = df[numerical_variables].corr()['is_canceled'].sort_values(ascending=False)
# Display the correlation values in a bar plot
correlation_values_bar_plot = px.bar(correlation_values, x=correlation_values.values,y=correlation_values.index, orientation='h', title='Correlation Values with the Target Variable')
# Add a title
correlation_values_bar_plot.update_layout(title_text='Correlation Values with the Target Variable', title_x=0.5)
# Display the plot
correlation_values_bar_plot.show()

# Store the useless variables in a list to drop them later
useless_variables = []

# Dropping the highly correlated variables
# Drop the variable "reservation_status" as it is very highly correlated with the target variable
useless_variables.append('reservation_status')
# Drop the variable "reservation_status_date_year" as it is highly correlated with "arrival_date_year"
useless_variables.append('reservation_status_date_year')
# Drop the variable "reservation_status_date_month" as it is highly correlated with "arrival_date_week_number"
useless_variables.append('reservation_status_date_month')
# Drop the variable "reservation_status_date_day" as it is highly correlated with "arrival_date_day_of_month"
useless_variables.append('reservation_status_date_day')

# Dropping the variables with low correlation with the target variable
# Get the variables with low correlation with the target variable
low_correlation_variables = correlation_values[(correlation_values < 0.05) & (correlation_values > -0.05)]
# Print the variables with low correlation with the target variable
print('Variables with low correlation with the target variable:')
print(low_correlation_variables)
print('-' * 100)
# Drop the variables with low correlation with the target variable
useless_variables.extend(low_correlation_variables.index)

# Dropping the useless variables
df.drop(useless_variables, axis=1, inplace=True)

# Converting categorical variables to numerical variables
# Get the categorical variables
categorical_variables = df.select_dtypes(include=['object']).columns
# Print the unique values of each categorical variable
for variable in categorical_variables:
    print(variable, df[variable].unique(), sep=': ')
# Encode the categorical variables
df = pd.get_dummies(df, columns=categorical_variables, drop_first=True)

# Normalize the numerical variables except the target variable
numerical_variables = df.select_dtypes(include=['int64', 'float64']).columns.drop('is_canceled')
for variable in numerical_variables:
    # Get the minimum and maximum values
    minimum, maximum = df[variable].min(), df[variable].max()
    # Normalize the variable
    if minimum != maximum:
        df[variable] = (df[variable] - minimum) / (maximum - minimum)
    else:
        df[variable] = 0
# Print the first 5 rows of the data
print('First 5 rows of the data:')
print(df.head())
print('-' * 100)
# Print the variance of each variable
print('Variance of each variable:')
print(df.var())
```

```
def evaluate_model(model, X, y,
resampling_strategy_name, class_ratio, folds,
model_name):
   # Get the predictions
   y_pred = model.predict(X)

   # Get the confusion matrix
   _confusion_matrix = pd.crosstab(
       y, y_pred, rownames=['Actual'],
colnames=['Predicted'])

   # Get the accuracy score
   _accuracy_score = accuracy_score(y, y_pred)

   # Get the precision score
   _precision_score = precision_score(y, y_pred,
zero_division=1)

   # Get the recall score
   _recall_score = recall_score(y, y_pred,
zero_division=1)

   # Get the f1 score
   _f1_score = f1_score(y, y_pred)

   # Get the roc auc score
   _roc_auc_score = roc_auc_score(y, y_pred)

   # Display the roc curve
   _roc_curve = roc_curve(y, y_pred)
   roc_curve_plot = px.line(x=_roc_curve[0],
y=_roc_curve[1])
   # Add a title
   roc_curve_plot.update_layout(
       title_text=f'ROC Curve for {model_name}
with {resampling_strategy_name} Resampling and
{class_ratio} Class Ratio, {folds}-Fold Cross
Validation', title_x=0.5)
   # Add x-axis and y-axis labels
   roc_curve_plot.update_xaxes(title_text='False
Positive Rate')
   roc_curve_plot.update_yaxes(title_text='True
Positive Rate')
   # Display the plot
   roc_curve_plot.show()

   # Return the confusion matrix, accuracy score,
precision score, recall score, f1 score, and roc auc
score
   return _confusion_matrix, _accuracy_score,
_precision_score, _recall_score, _f1_score,
_roc_auc_score
```

```
 X, y = df.drop('is_canceled', axis=1),
df['is_canceled']
resampling_strategies = {
      # 'No Resampling': None,
      # 'Random Oversampling':
RandomOverSampler(random_state=42),
      'SMOTE': SMOTE(random_state=42),
      'ADASYN': ADASYN(random_state=42),
   }

# Class ratios to resample the data for each
resampling strategy
class_ratios = [1.0, 0.5, 0.33, 0.25, 0.1]

# Different number of folds for k-fold cross-
validation
num_folds_list = [3, 5, 10]
# Defining the models
models = {
   'Logistic Regression': {
      'Sklearn': SciKitLearnLogisticRegression(),
   },
   'Naive Bayes': {
      'Sklearn': SciKitLearnNaiveBayes(),
   },
   'K Nearest Neighbors': {
      'Sklearn': SciKitLearnKNearestNeighbors(),
   },
   'Decision Tree': {
      'Sklearn': SciKitLearnDecisionTree(),
   }


   }
train_val_split_ratio = 0.8  # 80% for
train/validation
test_split_ratio = 0.2  # 20% for test

# Store the results in a dictionary
results = {}

# Loop over the resampling strategies
for resampling_strategy_name,
resampling_strategy in
resampling_strategies.items():
   # Store the results for each resampling strategy
in a dictionary
   results[resampling_strategy_name] = {}

      # Loop over the class ratios
   for class_ratio in class_ratios:
      # Store the results for each class ratio in a
dictionary
```

```
results[resampling_strategy_name][class_ratio] =
{}

        # Resample the data
        if resampling_strategy is not None:
            X_resampled, y_resampled =
resampling_strategy.fit_resample(X, y)
        else:
            X_resampled, y_resampled = X, y

            # Split into train and test sets
        X_train_val, X_test, y_train_val, y_test =
train_test_split(X_resampled, y_resampled,
test_size=test_split_ratio, random_state=42)

         # Create the model
        model = SciKitLearnLogisticRegression()
        model.fit(X_train_val, y_train_val)

        _confusion_matrix, _accuracy_score,
_precision_score, _recall_score, _f1_score,
_roc_auc_score = evaluate_model(
                model, X_test, y_test,
resampling_strategy_name, class_ratio, 1, 'Logistic
Regression')


results[resampling_strategy_name][class_ratio] = {
                'Confusion Matrix': _confusion_matrix,
                'Accuracy': _accuracy_score,
                'Precision': _precision_score,
                'Recall': _recall_score,
                'F1 Score': _f1_score,
                'ROC AUC Score': _roc_auc_score,
            }

# Print the results
for resampling_strategy_name in
resampling_strategies:
    for class_ratio in class_ratios:
        print('Results for {} using {} class
ratio:'.format(resampling_strategy_name,
class_ratio))

print(results[resampling_strategy_name][class_rati
o])
        print('-' * 100)
# Initialize a variable to store the results
results = {}
```

```
for resampling_strategy_name,
resampling_strategy in
resampling_strategies.items():
        # Store the results for each resampling
strategy in a dictionary
        results[resampling_strategy_name] = {}
        # Iterate over class ratios
        for class_ratio in class_ratios:
            # Store the results for each class ratio in a
dictionary

results[resampling_strategy_name][class_ratio] =
{}
            # Resample the data
            if resampling_strategy is not None:
                X_resampled, y_resampled =
resampling_strategy.fit_resample(
                    X, y)
            else:
                X_resampled, y_resampled = X, y

            # Split into train and test sets
            X_train_val, X_test, y_train_val, y_test =
train_test_split(
                X_resampled, y_resampled,
test_size=test_split_ratio, random_state=42)

            # Create the model
            model = SciKitLearnNaiveBayes()

            # Train the model
            model.fit(X_train_val, y_train_val)

            # Evaluate the model on the test set
            _confusion_matrix, _accuracy_score,
_precision_score, _recall_score, _f1_score,
_roc_auc_score = evaluate_model(
                model, X_test, y_test,
resampling_strategy_name, class_ratio, 1, ' Naive
Bayes')

            # Store the results in a dictionary

results[resampling_strategy_name][class_ratio] = {
                'Confusion Matrix': _confusion_matrix,
                'Accuracy': _accuracy_score,
                'Precision': _precision_score,
                'Recall': _recall_score,
                'F1 Score': _f1_score,
                'ROC AUC Score': _roc_auc_score,
            }
```

```
# Print the results
for resampling_strategy_name in
resampling_strategies:
      for class_ratio in class_ratios:
         print('Results for {} using {} class
ratio:'.format(
            resampling_strategy_name, class_ratio))

print(results[resampling_strategy_name][class_rati
o])
         print('-' * 100)
 results = {}

# Iterate over resampling techniques
 for resampling_strategy_name,
resampling_strategy in
resampling_strategies.items():
      # Store the results for each resampling
strategy in a dictionary
      results[resampling_strategy_name] = {}
      # Iterate over class ratios
      for class_ratio in class_ratios:
         # Store the results for each class ratio in a
dictionary

results[resampling_strategy_name][class_ratio] =
{}
         # Resample the data
         if resampling_strategy is not None:
            X_resampled, y_resampled =
resampling_strategy.fit_resample(
               X, y)
         else:
            X_resampled, y_resampled = X, y

         # Split into train and test sets
         X_train_val, X_test, y_train_val, y_test =
train_test_split(
            X_resampled, y_resampled,
test_size=test_split_ratio, random_state=42)

         # Create the model
         model = SciKitLearnKNearestNeighbors()

         # Train the model
         model.fit(X_train_val, y_train_val)

         # Evaluate the model on the test set
         _confusion_matrix, _accuracy_score,
_precision_score, _recall_score, _f1_score,
_roc_auc_score = evaluate_model(
            model, X_test, y_test,
resampling_strategy_name, class_ratio, 1, 'K
Nearest Neighbors')

         # Store the results in a dictionary

results[resampling_strategy_name][class_ratio] = {
            'Confusion Matrix': _confusion_matrix,
            'Accuracy': _accuracy_score,
            'Precision': _precision_score,
            'Recall': _recall_score,
            'F1 Score': _f1_score,
            'ROC AUC Score': _roc_auc_score,
         }

   # Print the results
 for resampling_strategy_name in
resampling_strategies:
      for class_ratio in class_ratios:
         print('Results for {} using {} class
ratio:'.format(
            resampling_strategy_name, class_ratio))

print(results[resampling_strategy_name][class_rati
o])
         print('-' * 100)
```

10.3 Explanation of code

Data Preparation and Preprocessing:

Importing Libraries: Imported necessary Python libraries for data manipulation (Pandas), visualisation (Matplotlib, Seaborn, Plotly), resampling techniques (imblearn), and machine learning models and evaluation metrics from scikit-learn.

Reading the Dataset: Loaded the 'hotel_bookings.csv' dataset into a Pandas DataFrame to perform operations on the data.

Handling Duplicates: Identified and removed duplicate rows from the dataset to prevent skewing the results with redundant data.

Handling Missing Values: Addressed missing or null values in several columns. Specifically, filled missing values in 'children', 'country', 'agent', and 'company' with appropriate placeholders (e.g., 0 or 'Unknown'). Removed any remaining rows with missing values that could not be imputed sensibly.

Removing Zero Guest Rows: Removed rows where the number of adults, children, and babies were all zero, as such entries do not represent valid bookings.

Exploratory Data Analysis (EDA):

Distribution of Numerical Variables: Visualised the distribution of numerical variables using histograms to understand the range and frequency of values across different numerical features.

Distribution of Categorical Variables: Plotted count plots for categorical variables to analyse the frequency of each category within the variables.

Bivariate Analysis: Generated choropleth maps to visually represent the number of guests by country for both cancelled and not cancelled bookings. Created line plots to explore trends in average price over time by hotel type and for both cancelled and not cancelled bookings.

Feature Engineering

Normalisation of Data: Normalised numerical variables to ensure that model inputs have a uniform scale. This step improves the convergence speed during the training phase of the models and impacts model performance.

Encoding Categorical Variables: Converted categorical variables to numerical format using one-hot encoding, making the data suitable for feeding into machine learning models.

Model Building and Evaluation

Resampling Strategies: Implemented different resampling strategies such as SMOTE and ADASYN to handle class imbalance in the dataset. This step helps improve model performance, particularly on minority classes.

Model Training and Validation: Trained various classifiers including Logistic Regression, Naive Bayes, K-Nearest Neighbors, and Decision Trees. Evaluated these models using metrics like accuracy, precision, recall, F1-score, and ROC AUC to understand their performance.

Cross-Validation: Applied k-fold cross-validation to assess the stability and reliability of the model performance across different subsets of the data.

Results Compilation

Compiling Results: Aggregated results from different models under varying resampling techniques and class ratios. This compilation helps in comparing the effectiveness of different models and configurations in handling the dataset.

Output Visualisation: Visualised results like ROC curves for models to provide insights into their true positive and false positive trade-offs.

Correlation Analysis: Conducted a correlation analysis to identify and eliminate highly correlated features, which can cause multicollinearity problems in machine learning models.