Data Mining Project

Tomer Lahav (208394452) | Dan Michaely (325625945)

;

Data information

We have 18 types of attributes:

```
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   27213 non-null  object
 1   weekend_nights       27213 non-null  int64
 2   week_nights          27213 non-null  int64
 3   room_type            27213 non-null  object
 4   board_type           19045 non-null  object
 5   n_adults             27213 non-null  int64
 6   n_less_12            27213 non-null  int64
 7   n_more_12            27213 non-null  int64
 8   booked_tour          27213 non-null  int64
 9   n_requests           27213 non-null  int64
 10  lead_time            26794 non-null  float64
 11  purchase_type        22366 non-null  object
 12  n_p_cacellation      27213 non-null  int64
 13  n_p_not_cacellation  27213 non-null  int64
 14  repeated             27213 non-null  int64
 15  price                23808 non-null  float64
 16  date                 27213 non-null  object
 17  is_canceled          27213 non-null  int64
```

Figure 1. Data attributes printed in python

Attribute	Description
ID	The ID of the current reservation – nominal attribute
weekend_nights	Number of weekend nights booked – an integer
week_nights	Number of week nights booked – an integer
room_type	The type of room – nominal attribute
board_type	The type of board – nominal attribute
n_adults	Number of adults in the order – an integer
n_less_12	Number of children aged less than 12 – an integer
n_more_12	Number of children aged more than 12 – an integer
booked_tour	Indicates whether a tour was booked – a Boolean value
n_requests	Number of special requests made by the guest – an integer
lead_time	Number of days between the reservation date and the arrival date – a real number
purchase_type	Type of purchase made - nominal attribute
n_p_cacellation	Number of previous reservations that were canceled by the customer prior to the current reservation – an integer
n_p_not_cacellation	Number of previous reservations not canceled by the customer prior to the current reservation – an integer
repeated	Indicates whether the reservation is a repeat reservation – a Boolean value
price	Price of the reservation – a real number
date	Date of the reservation – a date
is_canceled	Whether the order is canceled – a Boolean value

;

Values examples from the head of the database:

	ID	weekend_nights	week_nights	room_type	board_type	n_adults	n_less_12	n_more_12	booked_tour	n_requests
0	INN09588	1	5	Room_Type 1	half board	2	0	0	0	2
1	INN07691	0	3	Room_Type 1	NaN	2	0	0	0	0
2	INN32192	0	2	Room_Type 4	half board	1	0	0	0	1
3	INN32218	1	2	Room_Type 1	NaN	2	0	0	0	0
4	INN02994	1	3	Room_Type 4	half board	2	0	1	0	2

lead_time	purchase_type	n_p_cacellation	n_p_not_cacellation	repeated	price	date	is_canceled
34.0	Online	0	0	0	108.4	11/28/2018	0
365.0	NaN	0	0	0	NaN	11/03/2018	1
148.0	Online	0	0	0	137.3	05/06/2018	0
502.0	Offline	0	0	0	127.0	9/26/2018	1
32.0	Offline	0	0	0	110.0	10/19/2017	0

Figure 2. Examples for the attributes values. We can see the value difference between the different types of attributes, as well as some missing values – NaN.

;

Data statitics

Firstly, we printed several basic statistic of our data:

	ID	weekend_nights	week_nights	room_type	board_type	n_adults	n_less_12	n_more_12	booked_tour	n_requests	lead_time	purchase_type	n_p_cacellation	n_p_not_cacellation	repeated	price	date	is_canceled
count	27213	27213.000000	27213.000000	27213	19045	27213.000000	27213.000000	27213.000000	27213.000000	27213.000000	26794.000000	22366	27213.000000	27213.000000	27213.000000	23808.000000	27213	27213.000000
unique	27213	NaN	NaN	7	4	NaN	NaN	NaN	NaN	NaN	NaN	5	NaN	NaN	NaN	NaN	553	NaN
top	INN09588	NaN	NaN	Room_Type 1	half board	NaN	NaN	NaN	NaN	NaN	NaN	Online	NaN	NaN	NaN	NaN	10/13/2018	NaN
freq	1	NaN	NaN	21084	14591	NaN	NaN	NaN	NaN	NaN	NaN	14306	NaN	NaN	NaN	NaN	188	NaN
mean	NaN	0.812810	2.197332	NaN	NaN	1.845221	0.052989	0.053357	0.031750	0.621100	102.952377	NaN	0.021975	0.155404	0.026421	123.455494	NaN	0.327674
std	NaN	0.869317	1.403576	NaN	NaN	0.519715	0.266150	0.268688	0.175336	0.785642	103.498942	NaN	0.346697	1.728693	0.160387	35.136566	NaN	0.469374
min	NaN	0.000000	0.000000	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	NaN	0.000000	0.000000	0.000000	20.000000	NaN	0.000000
25%	NaN	0.000000	1.000000	NaN	NaN	2.000000	0.000000	0.000000	0.000000	0.000000	21.000000	NaN	0.000000	0.000000	0.000000	100.300000	NaN	0.000000
50%	NaN	1.000000	2.000000	NaN	NaN	2.000000	0.000000	0.000000	0.000000	0.000000	69.000000	NaN	0.000000	0.000000	0.000000	119.450000	NaN	0.000000
75%	NaN	2.000000	3.000000	NaN	NaN	2.000000	0.000000	0.000000	0.000000	1.000000	153.000000	NaN	0.000000	0.000000	0.000000	140.000000	NaN	1.000000
max	NaN	7.000000	17.000000	NaN	NaN	4.000000	6.000000	4.000000	1.000000	5.000000	532.000000	NaN	13.000000	57.000000	1.000000	560.000000	NaN	1.000000

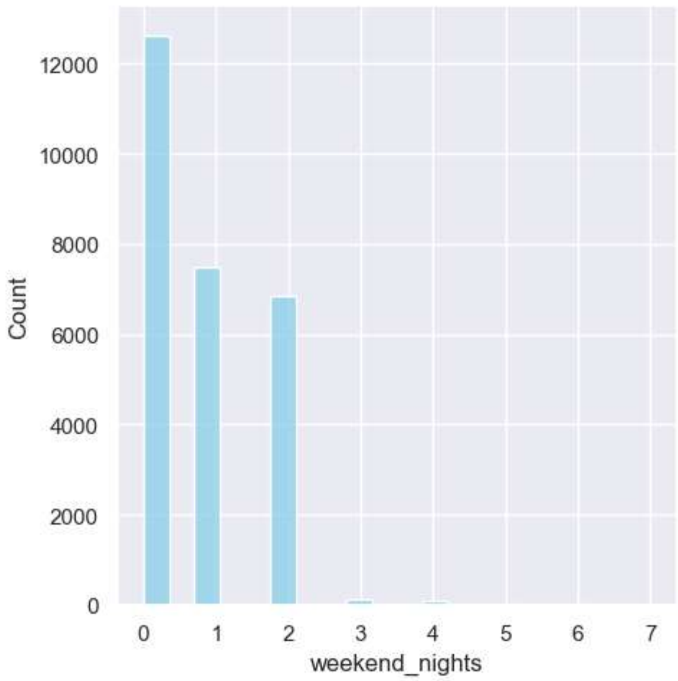
Figure 3. Basic statitstics for each attribute

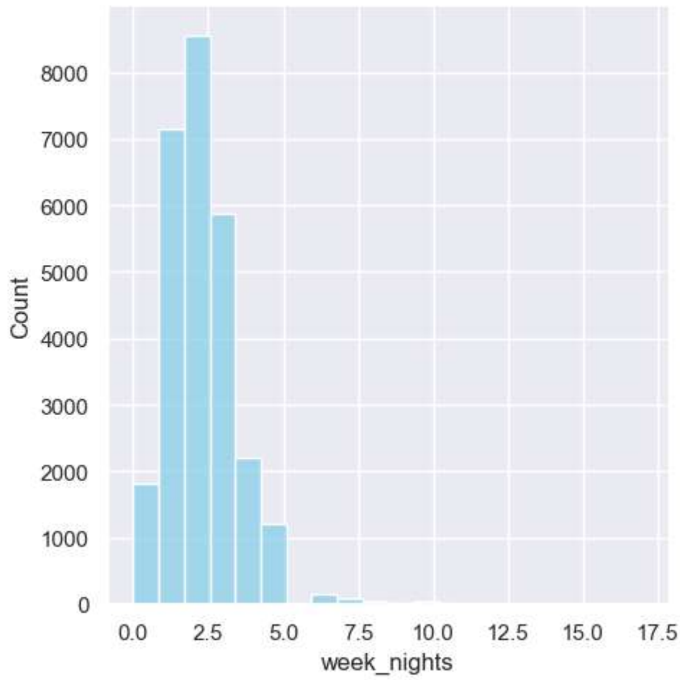
We can notice that for nominal attributes we get 'NaN' for statistics like mean, std etc. That is because the attributes do not have numerical value and thus we cannot calculate those kind of statistics for them.

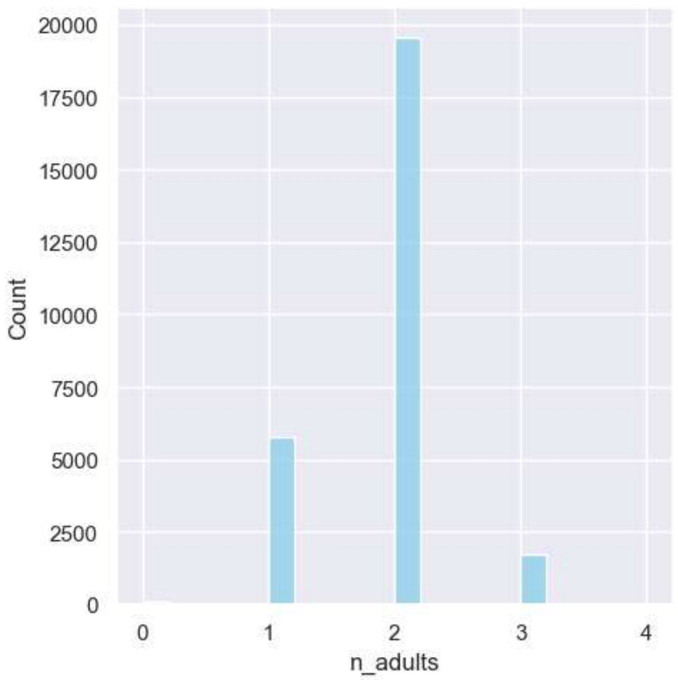
We will display the distribution of the attributes values (excluding the ID) in order to learn more about them.

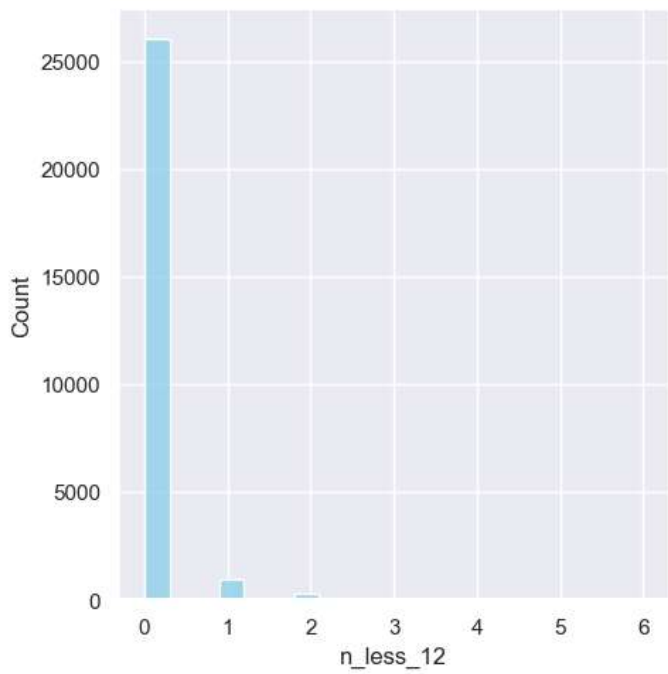
Numerical attributes:

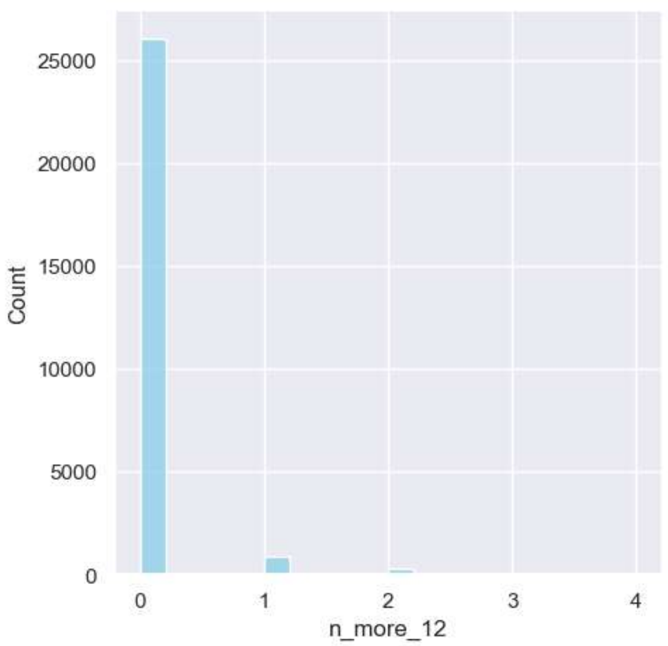
;

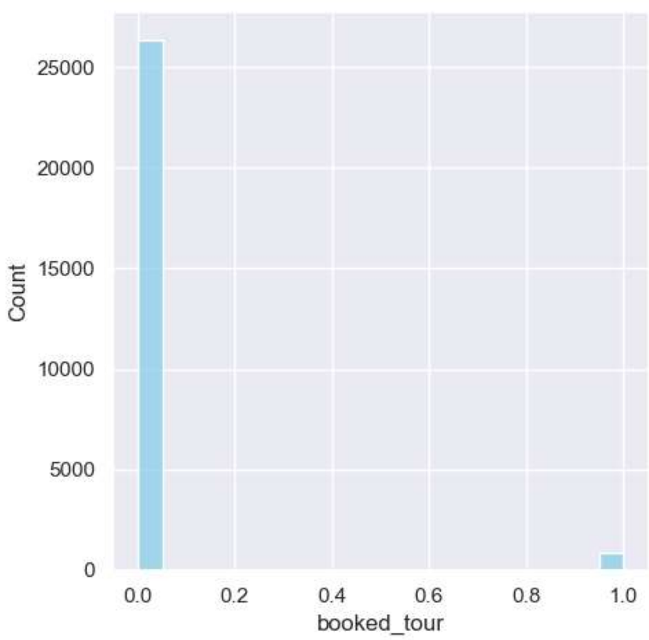

12000
10000
8000
6000
4000
2000
0
Count
0 1 2 3 4 5 6 7
weekend_nights

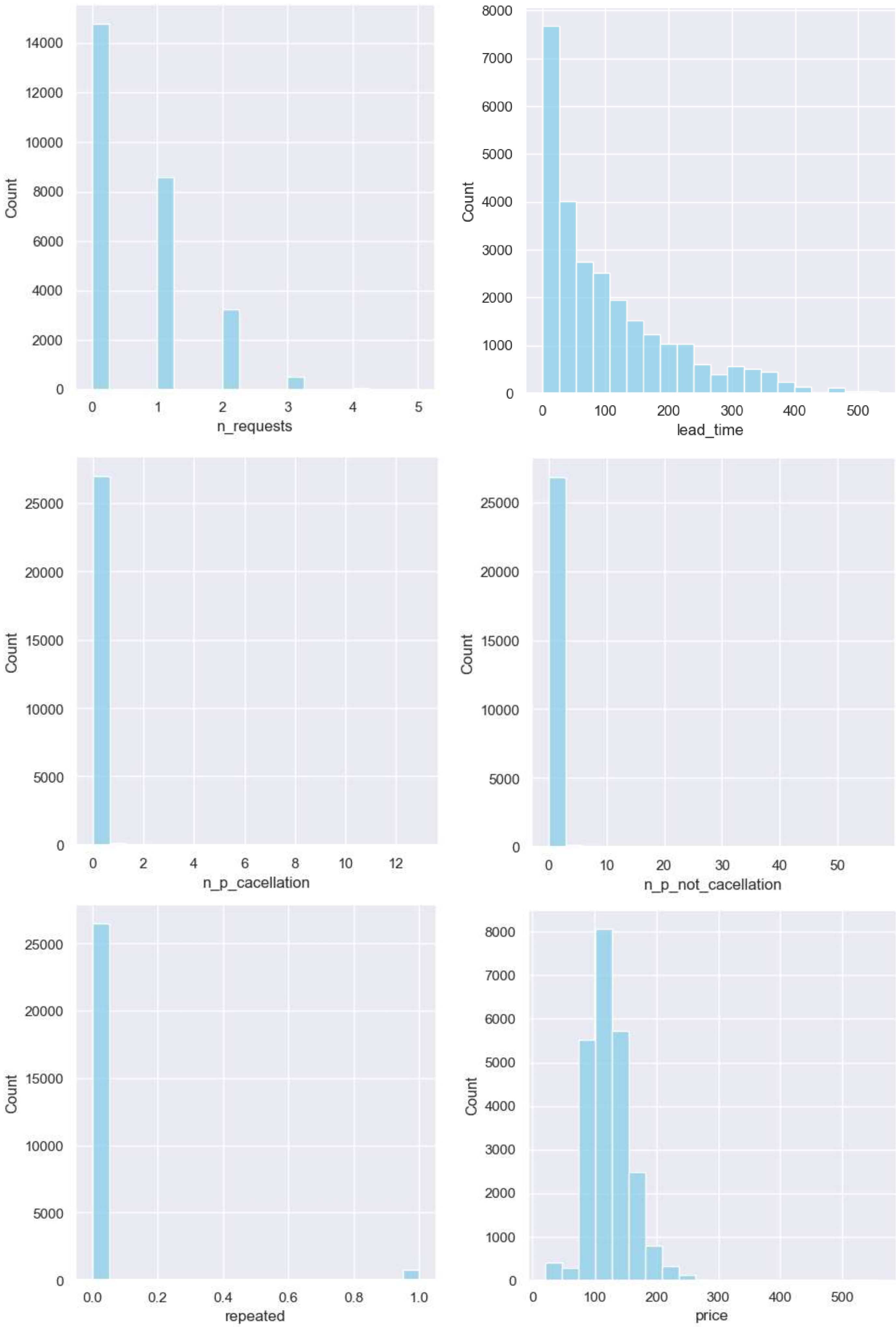

8000
7000
6000
5000
4000
3000
2000
1000
0
Count
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
week_nights


20000
17500
15000
12500
10000
7500
5000
2500
0
Count
0 1 2 3 4
n_adults

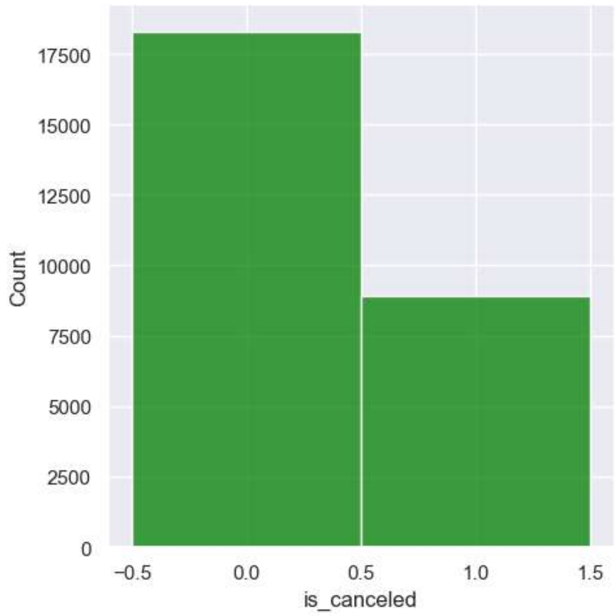

25000
20000
15000
10000
5000
0
Count
0 1 2 3 4 5 6
n_less_12


25000
20000
15000
10000
5000
0
Count
0 1 2 3 4
n_more_12


25000
20000
15000
10000
5000
0
Count
0.0 0.2 0.4 0.6 0.8 1.0
booked_tour

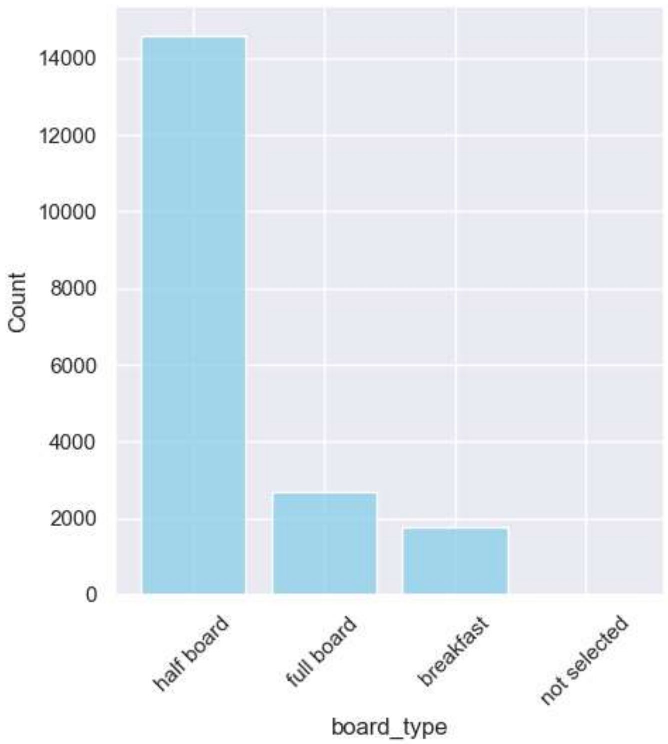


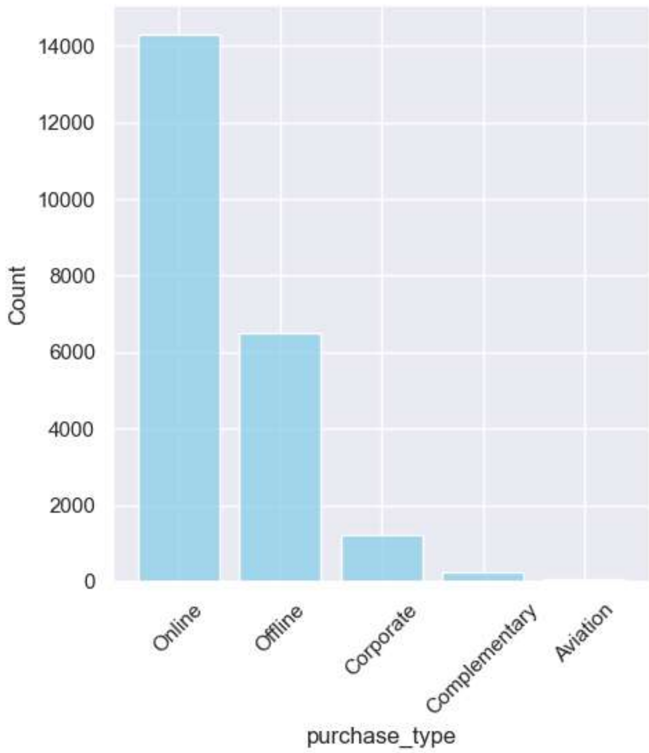
Count
n_requests
Count
lead_time
Count
n_p_cacellation
Count
n_p_not_cacellation
Count
repeated
Count
price



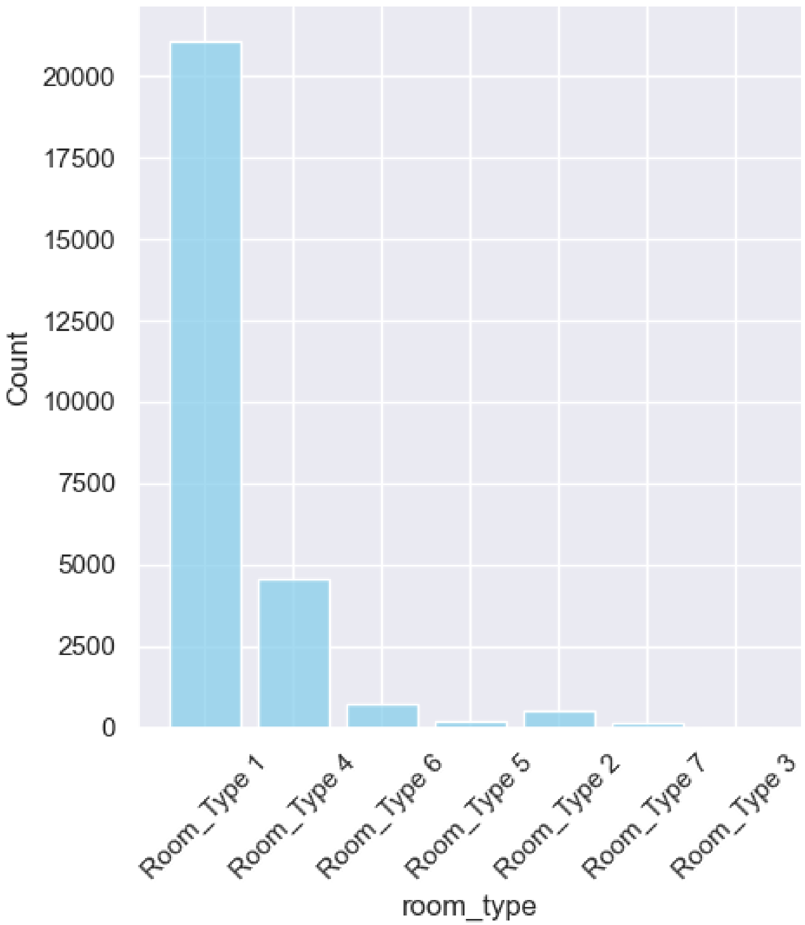
Most of the interesting data about the distributions could have been infered from the basic statistics, for example we can understand that most orders are not repeated because the mean of the attribiute was low, but it helps visualize the data.

Nominal data:





;



We can learn things about the popularity of certain values for each attribute from the distributions and we will use it later.

We were also requested to calculate the skewness of the attributes:

```
weekend_nights  skewness: 0.7187888765659908
week_nights  skewness: 1.5422954469230514
n_adults  skewness: -0.32416096529704014
n_less_12  skewness: 5.749665488234375
n_more_12  skewness: 5.569096359968911
n_requests  skewness: 1.1437446108569245
lead_time  skewness: 1.291447308852505
price  skewness: 0.682392611272259
```

Figure 4. Skewness of the numerical attributes

;

Attributes correaltions

We used the *corr* method in python in order to get the correlation between our numerical attributes, we used heat wave map in order to display it:

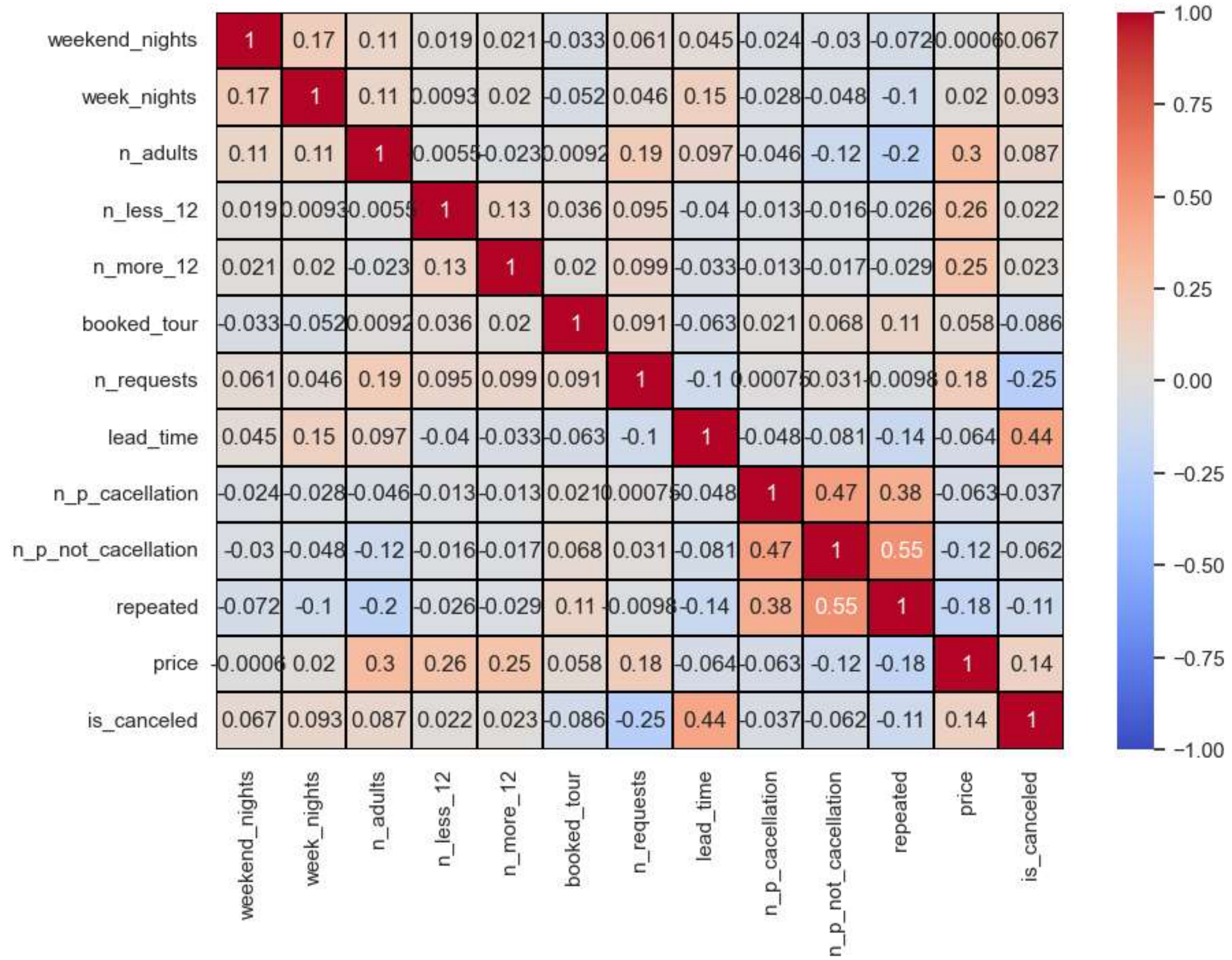


Figure 5. Correlations between numerical attributes

Strong correlation is about at least 0.3, we ignore the 1 correlations og the attributes with themselvs naturally. We can identify firstly the medium-high correlations:

- $weekend\_nights$ and $week\_nights$ – makes sense since the more nights you are staying, the more likely you are to stay more nights in the weekend.
- n_adults and $n\_requests$ – when more people are coming they are likely to have more requests.
- $price$ and the attributes n_adults, n_less_12, n_more_12, $n\_requests$ – when the reservation includes more people, the price is expected to be higher and vice versa. Also, we notice that the correlation is slightly weaker for the number of kids, probably because the price for them is lower, thus the connection is weaker. In addition, it is reasonable to think that the higher the price, the more people will request special things.

Strong correlations:

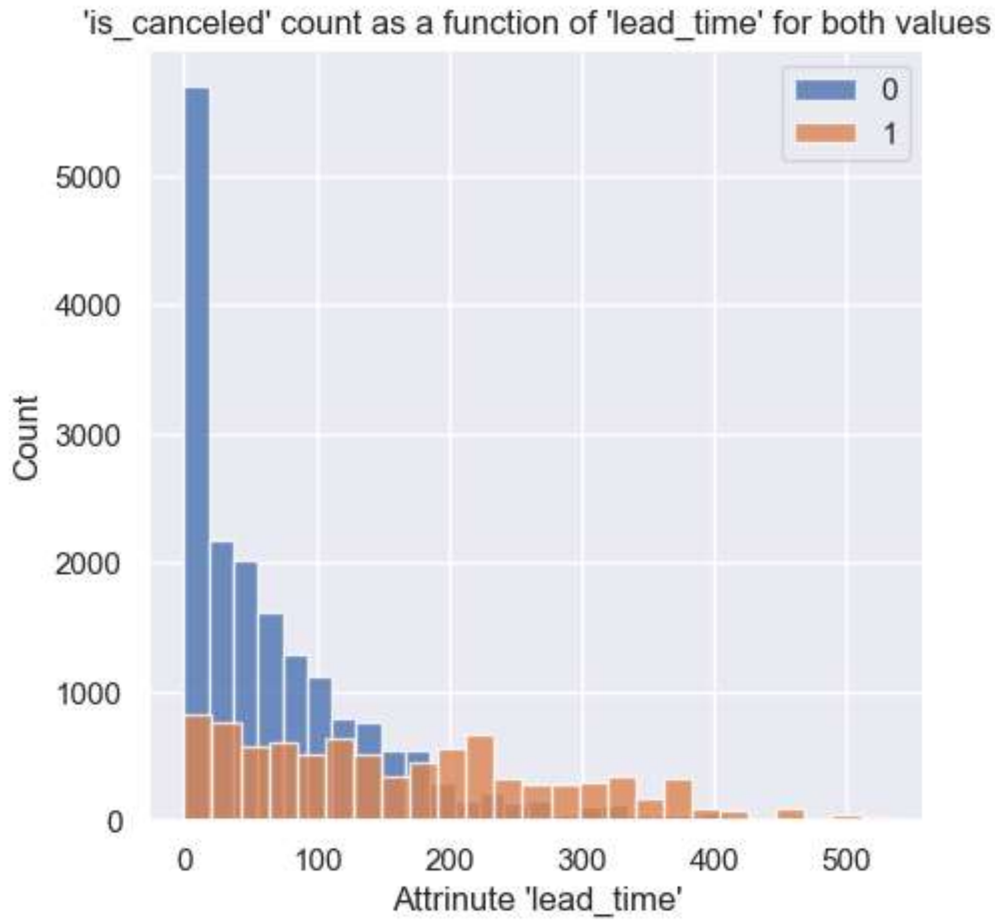
- $lead\_time$ and $is\_canceled$ – the further the reservation is, the more likely are people to cancel it due to changes in schedule, unexpected events etc.

;

- $repeated$ and $n\_p\_cancellation$, $n\_p\_not\_cancellation$ – if this is a repeated order then the cosumer had a previous one which he canceled or did not cancel. Also, the correlation is higher for $n\_p\_not\_cancellation$ which makes sense because people are more likely to go to hotels they already know and love.
- $n\_p\_cancellation$ and $n\_p\_not\_cancellation$ – also correlates strongly as we expect.

Insights from the data

Most of the interesting trends can be inferred from the previous part such as the relation between $lead\_time$ and $is\_canceled$. We can plot them as function of one another and get:

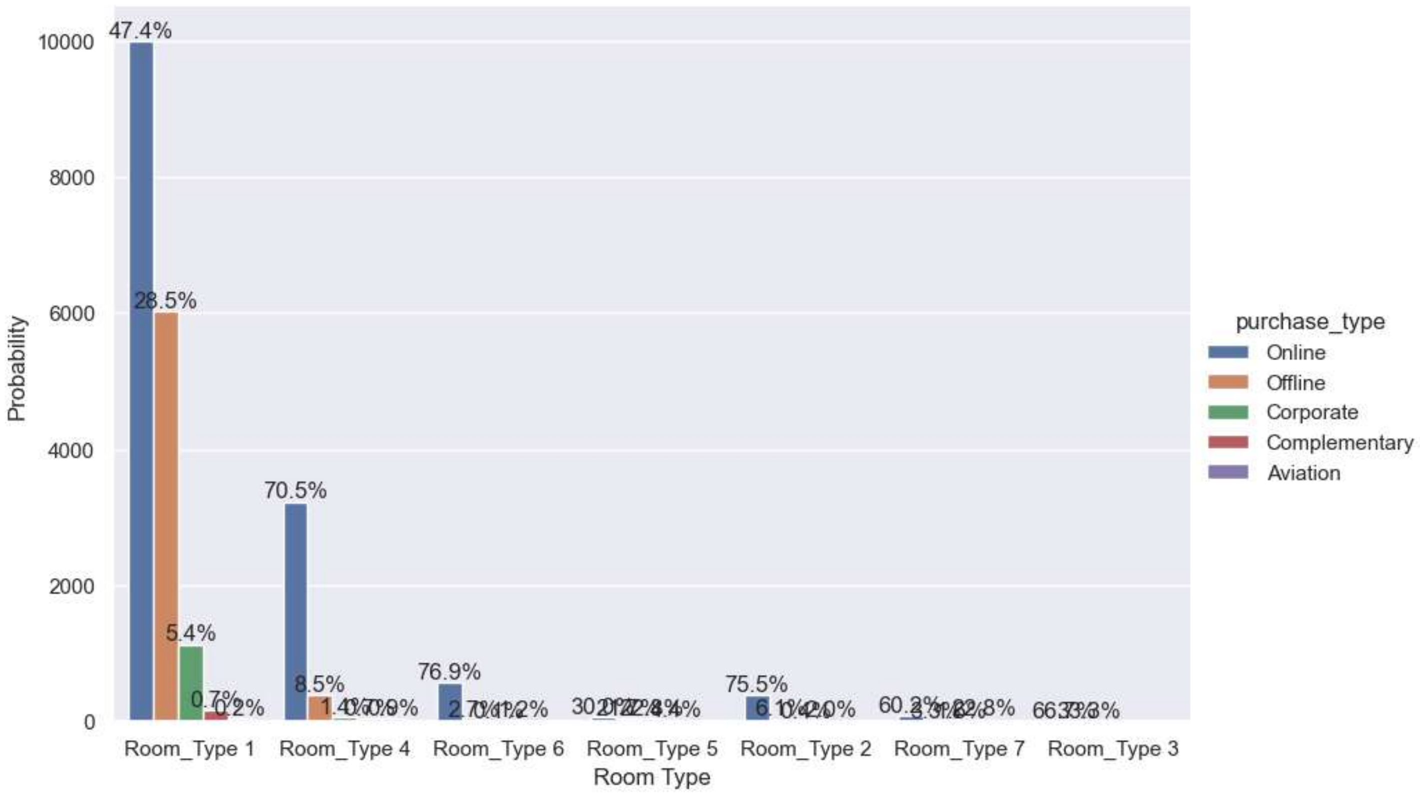


We can see that as the $lead\_time$ grows, the lower is the count for $is\_canceled = 0$ but higher for $is\_canceled = 1$.

In this part, we tried finding interesting trends and insights involving the nominal attributes since we discussed the other trends before.

We plotted the purchase types for the different room types:

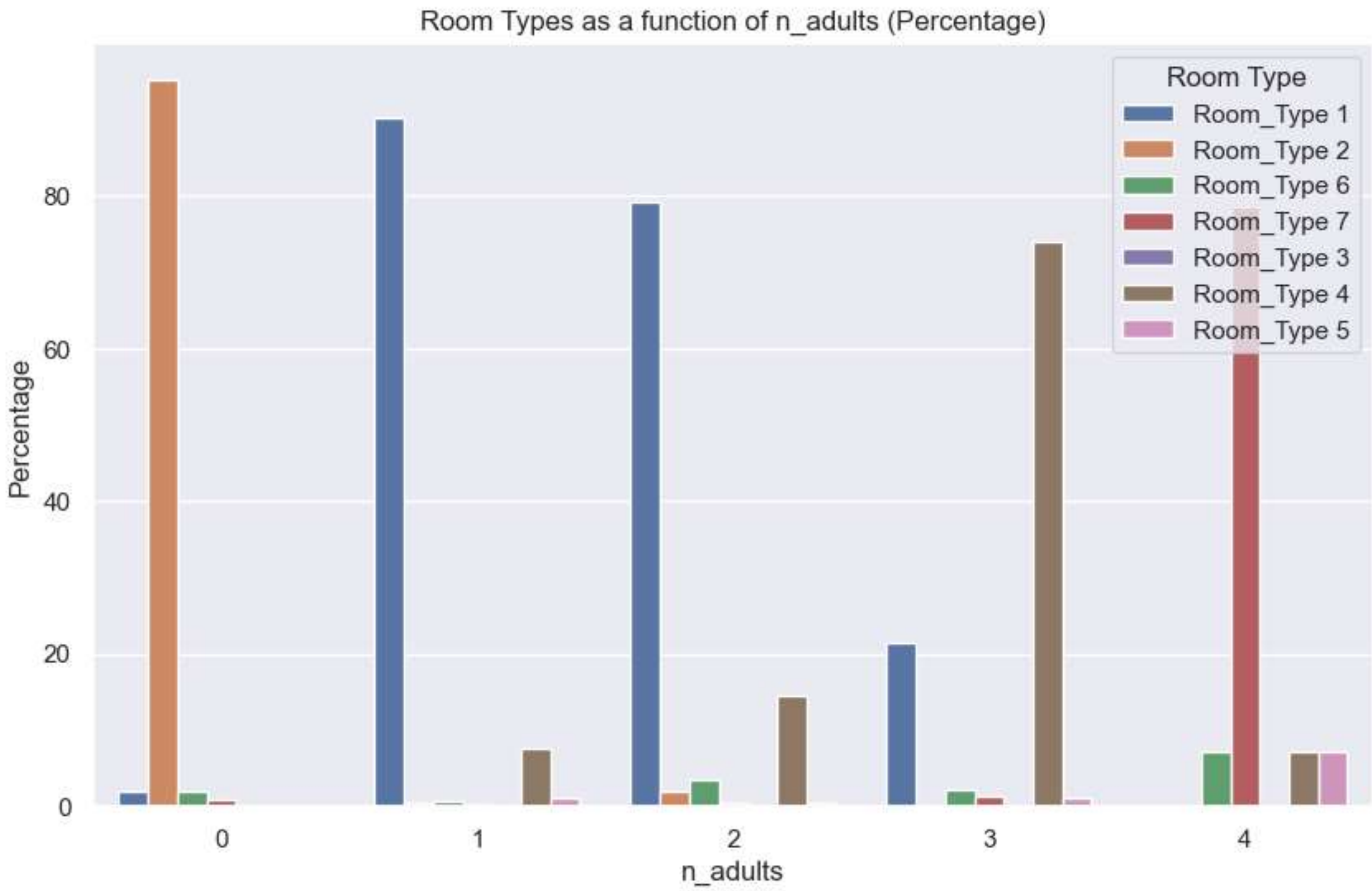
;



We can't say a lot about the room types other than $Room\_Type$ 1 and 4 becasuse there are almost no reservations for these rooms.

We can see that the percentage of reservations online made for 4 is much higher than 1. This could indicate problems with advertising Room1 online or Room4 offline.

We plotted the most common room types as function of the number of adults:



;

For any number of adults there is one room type which is significantly more popular than the others. Thus, we can infer for what purpose each room is designated:

- Room type 1 – standard room, for 1-2 adults.
- Room type 2 – most common when n_adults is 0. Probably a room for kids.
- Room type 4 – most common when n_adults is 3. Probably a slightly bigger room than room type 1 with an additional bed.
- Room type 7 – big room for 4 adults.
- Room type 3,5,6 – not very common for any number of adults, we assume these are expensive room or some kind of suite.

We tried finding interesting graphs involving the nominal attributes but the graphs above are the only ones we learned new information from.

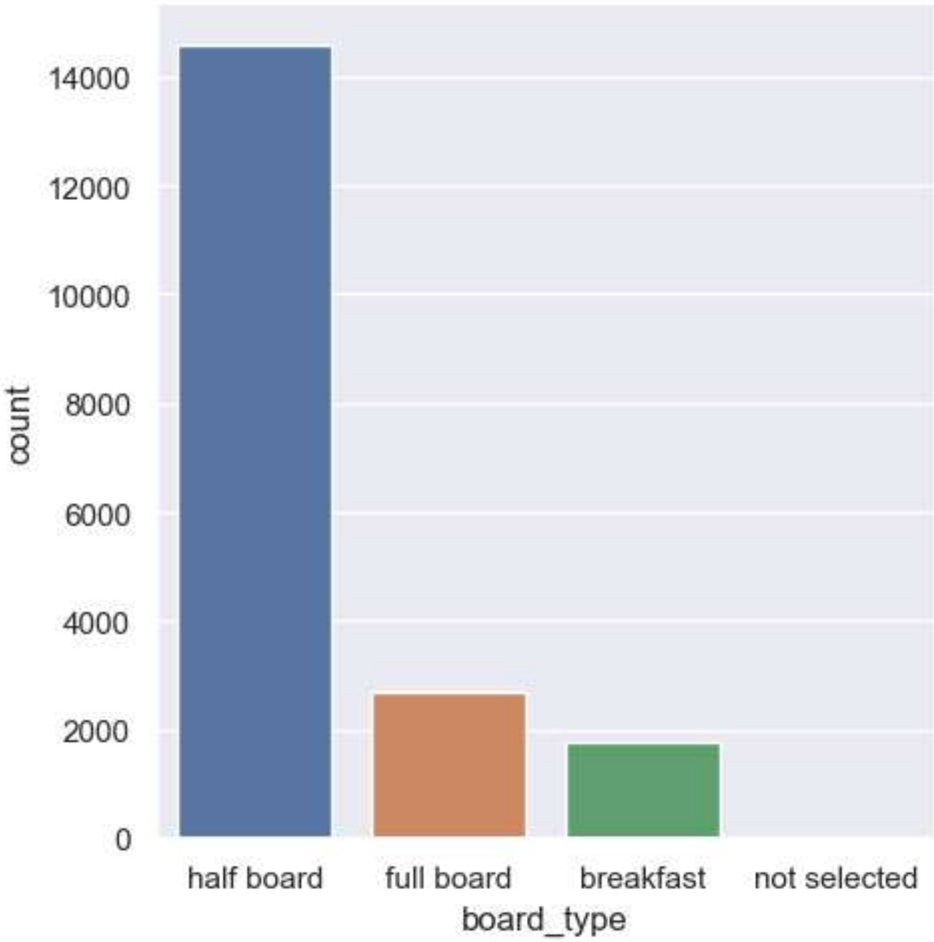
Data cleaning

As seen in Figure 1, our dataset contains missing values in four attributes: lead_time, board_type, purchase_type, and price. We addressed these missing values one attribute at a time to ensure data integrity and improve the model's performance.

- board_type:

We analyzed the distribution of values in the board_type attribute and found that the majority of orders selected 'half board.' Therefore, we filled the missing values with 'half board.'

Additionally, there was another value called 'not selected,' which appeared in only four rows. We assumed these were mistakes in the orders and filled them with the cheapest value, which is 'breakfast.'



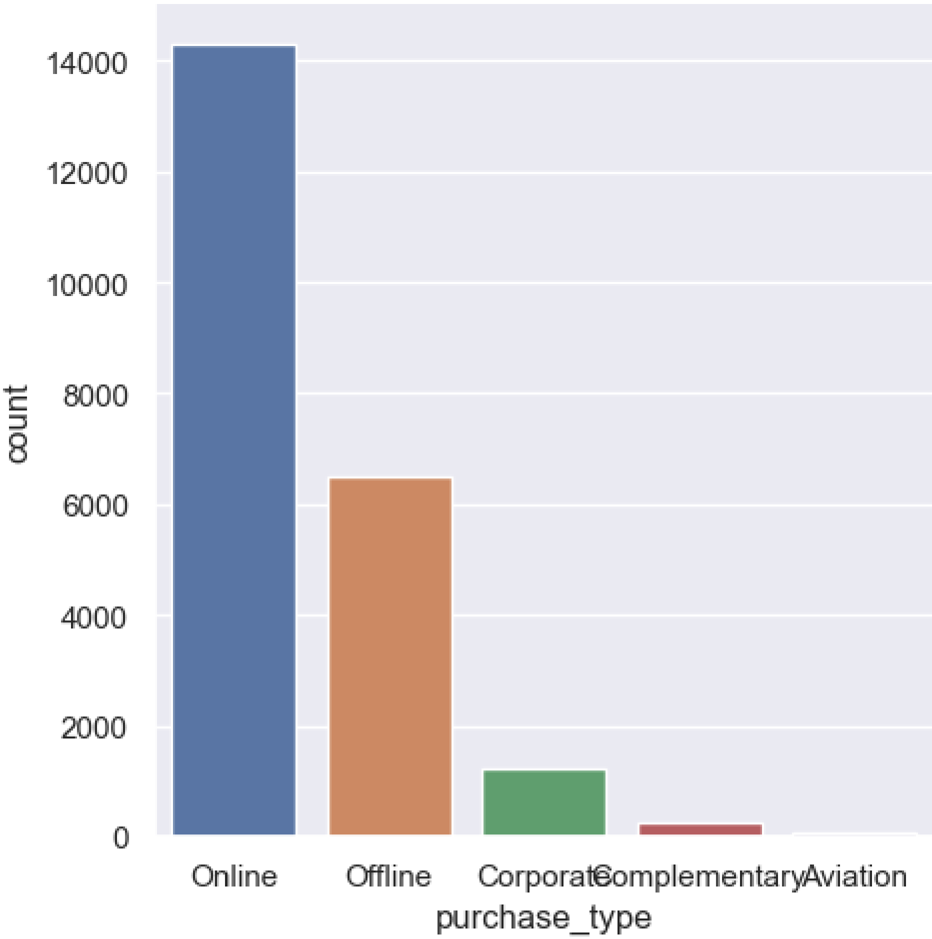
- lead_time

For the lead_time attribute, we observed that all rows with missing values also had missing values for the other attributes we needed to address. Given that these problematic rows amounted to only 419, or 1.5% of the entire dataset, we decided to drop these rows instead of filling four attributes in each row. This approach helped maintain data quality without introducing potential biases.

- purchase_type

Similar to board_type, we searched for the most common value in the purchase_type attribute, which was 'Online.' We filled all missing values with 'Online,' ensuring a consistent dataset.

```
purchase_type
Online          14306
Offline          6501
Corporate        1238
Complementary     244
Aviation           77
Name: count, dtype: int64
```



- price

For the price attribute, we aimed to avoid simply filling missing values with the mean. Instead, we sought correlations between rows and used similar rows to estimate the mean price. We developed an algorithm to search for similar rows and calculate the mean price. If similar rows were not found, the algorithm looked for more general rows close to the specific row in question. This method provided a more accurate imputation for missing prices.

;

Handling Categorical Attributes

Next, we addressed the categorical attributes: room_type, board_type, purchase_type, and date.

- date

  We replaced the date attribute by extracting only the month, as we believed it was sufficient for our analysis. This transformation helped simplify the attribute without losing significant information.

- Other Categorical Attributes
  For room_type, board_type, and purchase_type, we assigned numerical labels to each category. This step facilitated easier handling of these attributes in subsequent analyse:

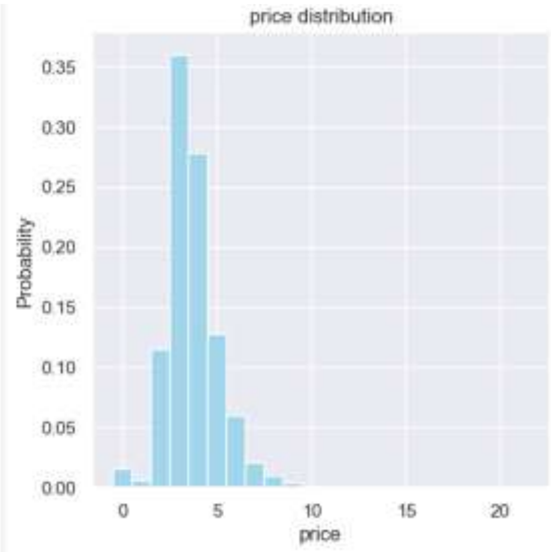
```
# room_type
room_type_mapping = {
    'Room_Type 1': 1,
    'Room_Type 2': 2,
    'Room_Type 3': 3,
    'Room_Type 4': 4,
    'Room_Type 5': 5,
    'Room_Type 6': 6,
    'Room_Type 7': 7
}
df['room_type'] = df['room_type'].map(room_type_mapping)

# board_type
board_type_mapping = {
    'not selected': 0,
    'breakfast': 1,
    'half board': 2,
    'full board': 3
}
df['board_type'] = df['board_type'].map(board_type_mapping)

# purchase_type
purchase_type_mapping = {
    'Online': 1,
    'Offline': 2,
    'Corporate': 3,
    'Complementary': 4,
    'Aviation': 5
}
```

;

Discretization of Numerical Attributes

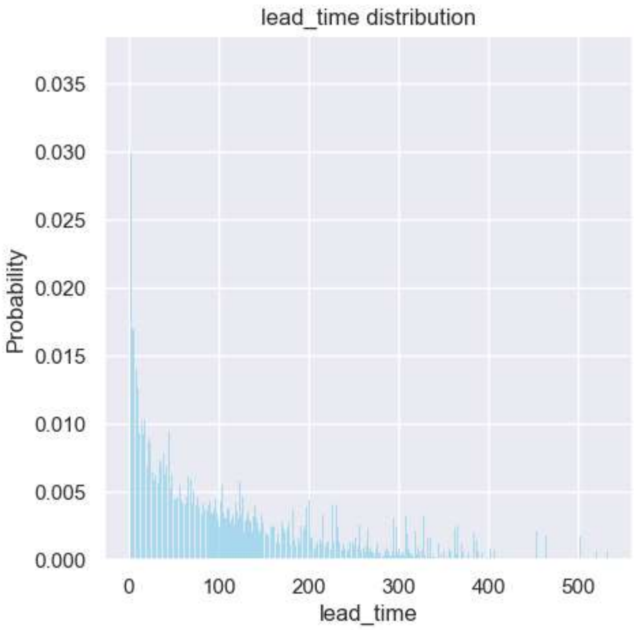
We proceeded to discretize the numerical attributes to improve model performance:

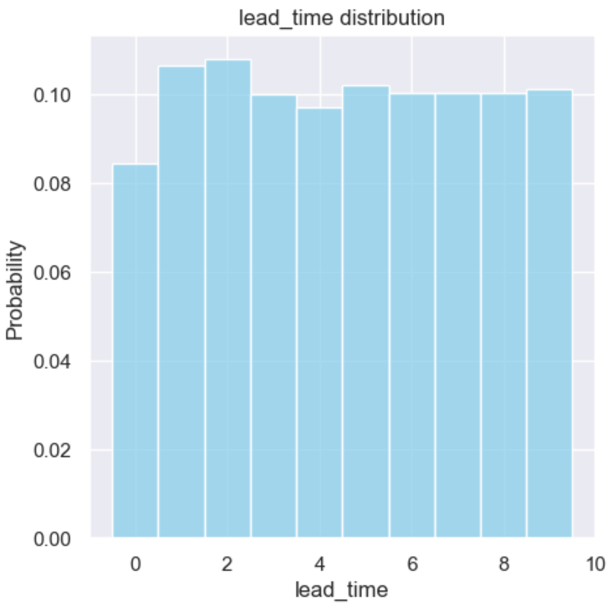
- Price:
  For price discretization, we chose Equal-width partitioning. This method divides the range of values into equal-sized intervals, helping to manage the data distribution effectively.



- Lead_time

For lead time discretization, we opted for Equal-depth partitioning. This method ensures each interval contains approximately the same number of samples, providing a balanced representation of the data.





;

Transforming Features

We transformed all other features, such as weekend_nights and week_nights, which were already numerical. The final step involved converting categorical features into dummy variables using binary values, effectively expanding these features into multiple columns with 0s and 1s to represent each category.

now the data look like this:

```
0   ID                  26774 non-null  object
1   weekend_nights      26774 non-null  float64
2   week_nights         26774 non-null  float64
3   n_adults            26774 non-null  float64
4   n_less_12           26774 non-null  float64
5   n_more_12           26774 non-null  float64
6   n_requests          26774 non-null  float64
7   lead_time           26774 non-null  float64
8   n_p_cacellation     26774 non-null  float64
9   n_p_not_cacellation 26774 non-null  float64
10  price               26774 non-null  float64
11  is_canceled         26774 non-null  int64
12  month_1             26774 non-null  float64
13  month_2             26774 non-null  float64
14  month_3             26774 non-null  float64
15  month_4             26774 non-null  float64
16  month_5             26774 non-null  float64
17  month_6             26774 non-null  float64
18  month_7             26774 non-null  float64
19  month_8             26774 non-null  float64
20  month_9             26774 non-null  float64
21  month_10            26774 non-null  float64
22  month_11            26774 non-null  float64
23  month_12            26774 non-null  float64
24  booked_tour_0       26774 non-null  float64
25  booked_tour_1       26774 non-null  float64
26  room_type_1         26774 non-null  float64
27  room_type_2         26774 non-null  float64
28  room_type_3         26774 non-null  float64
29  room_type_4         26774 non-null  float64
30  room_type_5         26774 non-null  float64
31  room_type_6         26774 non-null  float64
32  room_type_7         26774 non-null  float64
33  board_type_1        26774 non-null  float64
34  board_type_2        26774 non-null  float64
35  board_type_3        26774 non-null  float64
36  purchase_type_1     26774 non-null  float64
37  purchase_type_2     26774 non-null  float64
38  purchase_type_3     26774 non-null  float64
39  purchase_type_4     26774 non-null  float64
40  purchase_type_5     26774 non-null  float64
41  repeated_0          26774 non-null  float64
42  repeated_1          26774 non-null  float64
```

;

Addressing Data Imbalance

Before proceeding with building different models for our project, it was crucial to assess the balance of our dataset. Upon examination, we found that our data was imbalanced, with a distribution of approximately 67% non-canceled bookings to 32% canceled bookings.

is_canceled	Count	Percentage
0	18003	67.240607
1	8771	32.759393

To address this imbalance, we aimed to rebalance the dataset to a more equitable distribution of around 55% non-canceled to 45% canceled bookings. This approach allowed us to evaluate whether rebalancing would have a significant impact on our model performance.

Evaluation Metrics

When dealing with cancellation predictions, the most appropriate evaluation metric is recall. The reason behind this is the critical need to identify as many cancellations as possible, minimizing the potential financial and operational disruptions for the hotel.

However, the practical evaluation of our models is often based on accuracy, as it provides a straightforward measure of how well our model predicts both cancellations and non-cancellations. Therefore, while recall remains our primary focus, we also aim to optimize accuracy and Area Under the Receiver Operating Characteristic Curve (AUC-ROC) scores to ensure a balanced and effective model performance.

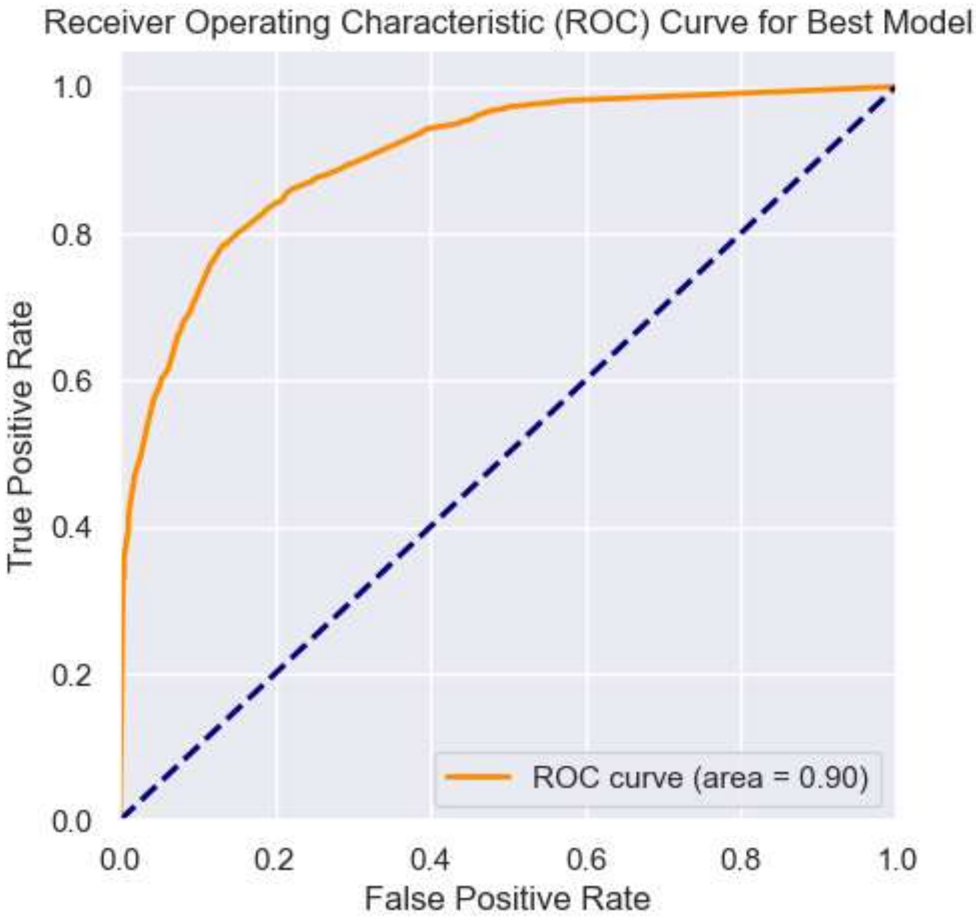
First Model: DecisionTreeClassifier

The DecisionTreeClassifier is a versatile machine learning algorithm capable of performing both classification and regression tasks. It works by splitting the data into subsets based on the most significant differentiators among the features, resulting in a tree-like model of decisions. This algorithm is particularly useful for its interpretability, as the decision tree structure can be visualized and easily understood.

After performing hyperparameter tuning, we identified the optimal parameters for our DecisionTreeClassifier. The key hyperparameters we tuned included the maximum depth of the tree, the minimum number of samples required to split an internal node, and the minimum number of samples required to be at a leaf node.

Here are the results obtained with the best hyperparameters:

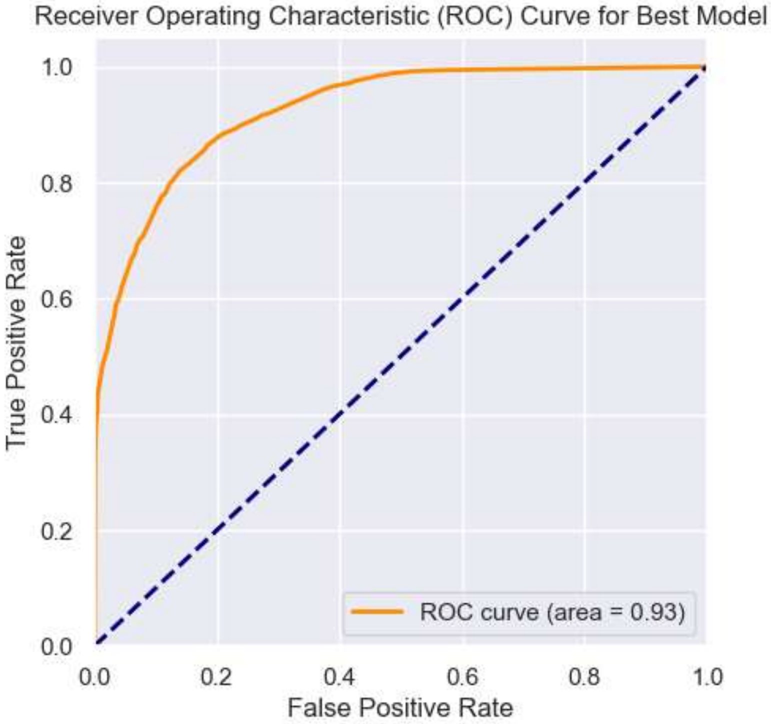
```
Accuracy of best model : 0.8383627128772034
Recall of best model : 0.6873023045639404
```



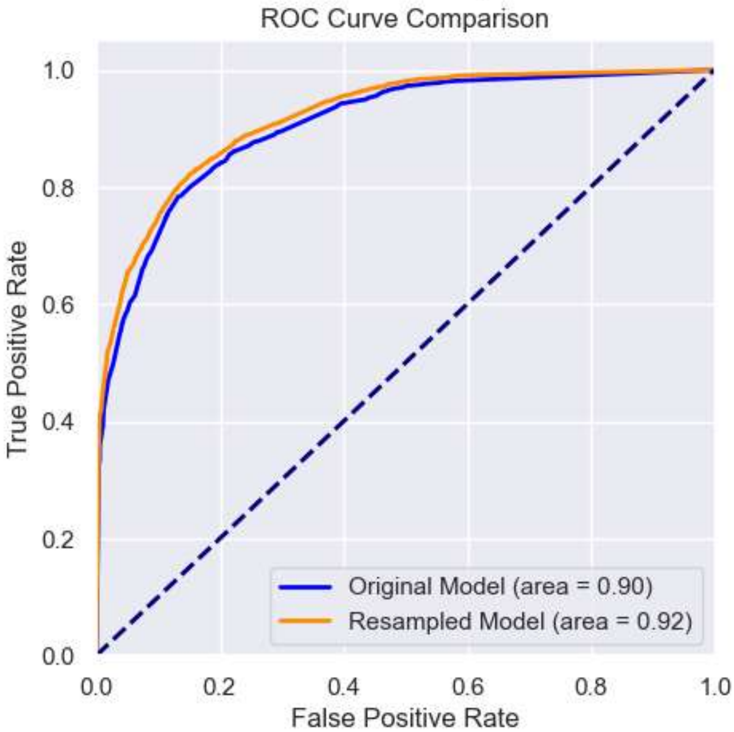
;

To understand the impact of data balancing, we compared the performance of the model on the original imbalanced dataset and the balanced dataset:

```
Accuracy of best model : 0.8271970658767104
Recall of best model : 0.7041666666666667
```




```
Mean ROC AUC for the best model: 0.9055608310947392
Mean ROC AUC for the best model with resampled data: 0.9198396093183628
```




```
T-statistic: -5.0438486717935405
P-value: 0.0006961360413936723
The difference between the models is statistically significant.
```

While the DecisionTreeClassifier showed promising results, we decided to explore additional models to ensure the robustness and effectiveness of our solution. Our goal is to find the model that offers the best overall performance in terms of accuracy, recall, and AUC-ROC.

;

Next Model: Gaussian Naive Bayes

The Gaussian Naive Bayes classifier is a probabilistic model that applies Bayes' theorem with the assumption of independence between every pair of features. It is particularly efficient and performs well with many features, provided the assumption of independence holds.

Performance Analysis

After training the Gaussian Naive Bayes model on our dataset, we observed the following results:

```
Training Accuracy: 0.4227589641434263
Training Recall: 0.9656907593778591
Testing Accuracy: 0.4323274574245593
Testing Recall: 0.9661093538183462
Confusion Matrix (Train):
      0      1
0  2156  11366
1   225   6333
Confusion Matrix (Test):
     0     1
0  756  3725
1   75  2138
```

Recall:
The recall for the Gaussian Naive Bayes model was notably high. This is beneficial for our primary objective, which is to maximize the identification of cancellations. A high recall indicates that the model is very effective at capturing the majority of true positive cancellations

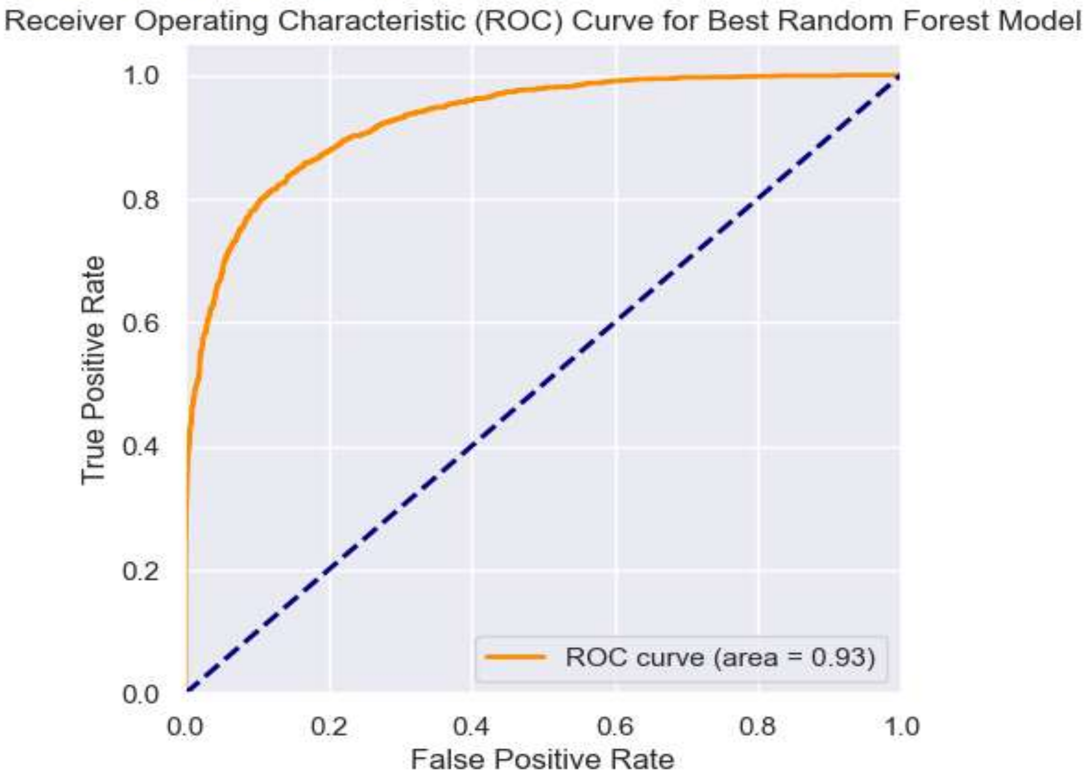
Accuracy:
Despite the high recall, the model exhibited poor accuracy. This suggests that while the model correctly identifies many cancellations, it also incorrectly labels a significant number of non-cancellations as cancellations. In other words, the model has a high false positive rate.

Next Model: Random Forest

The Random Forest classifier is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mode of the classes for classification. It is known for its robustness, ability to handle large datasets with higher dimensionality, and its effectiveness in reducing overfitting.

;

```
Fitting 10 folds for each of 81 candidates, totalling 810 fits
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
Best Accuracy Score: 0.8683764940239044
Accuracy of best model : 0.8651030773827308
Recall of best model : 0.7297785811116132
```



Accuracy:
The Random Forest model demonstrated a high accuracy, indicating that it correctly classifies a significant portion of both cancellations and non-cancellations. This balance is crucial for reliable overall performance.

Recall:
The recall score was also strong, reflecting the model's effectiveness in identifying the majority of true cancellations. This is essential for our objective of maximizing the detection of cancellations.

AUC-ROC:
The model achieved a high AUC-ROC score, suggesting excellent capability in distinguishing between cancellations and non-cancellations. A high AUC-ROC value is indicative of a robust model with good predictive power.

Given its balanced performance and robustness, the Random Forest model stands out as a good candidate for a winning model in our project. We will consider it alongside other models in our final evaluation.

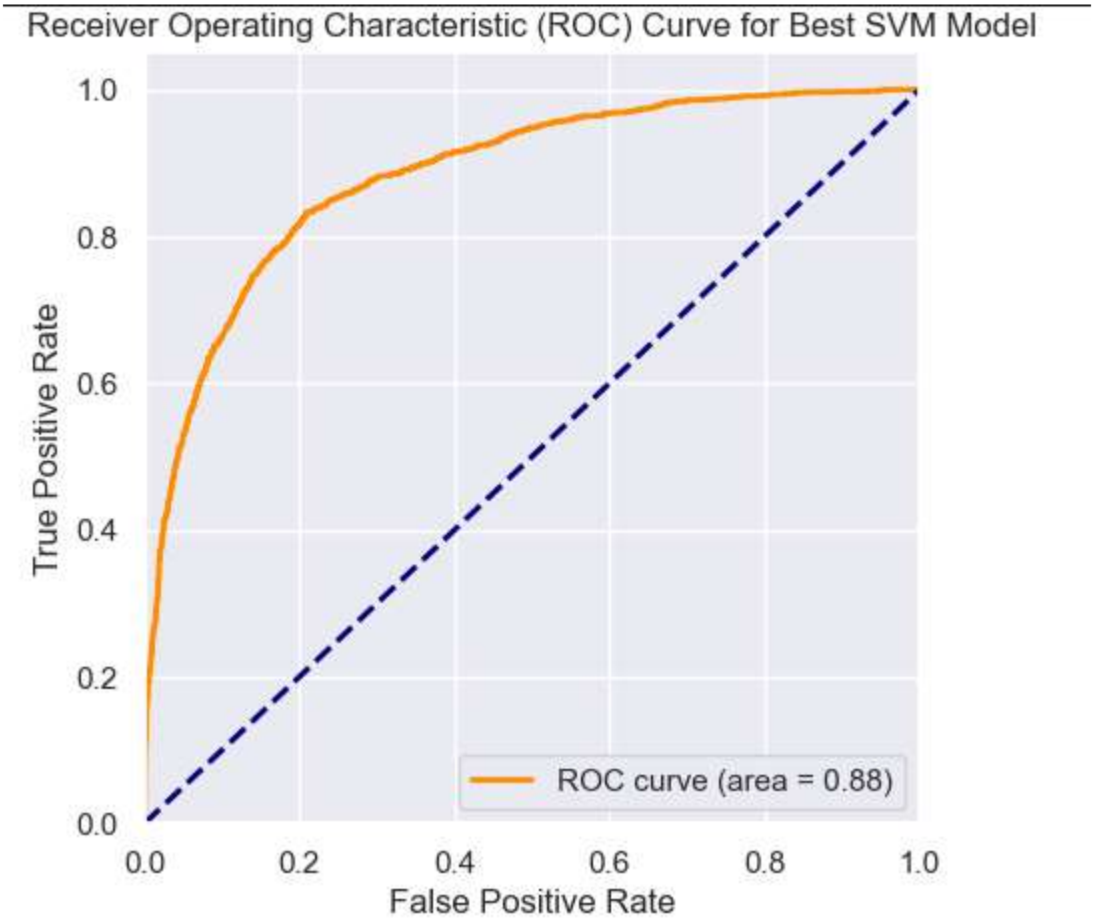
Next Model: Support Vector Machine (SVM)

Support Vector Machines (SVM) are powerful supervised learning models used for classification and regression. They work by finding the hyperplane that best separates the classes in the feature space. Despite

;

their effectiveness in many scenarios, they can be computationally intensive, especially with large datasets and high-dimensional data.

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Best Parameters: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
Best Accuracy Score: 0.8238545816733067
Accuracy of best model : 0.8220794741559606
Recall of best model : 0.6741979213737008
```



Run Time:
The SVM model's training and prediction times were significantly longer than those of the other models we evaluated. The computational cost of SVMs, particularly with large and high-dimensional datasets, can be a limiting factor.

Accuracy:
The accuracy of the SVM model was moderate but did not surpass the performance of other models like Random Forest and Gradient Boosting.

AUC-ROC:
The AUC-ROC score was also moderate, indicating that the model's ability to distinguish between cancellations and non-cancellations was not as robust as other models we tested.

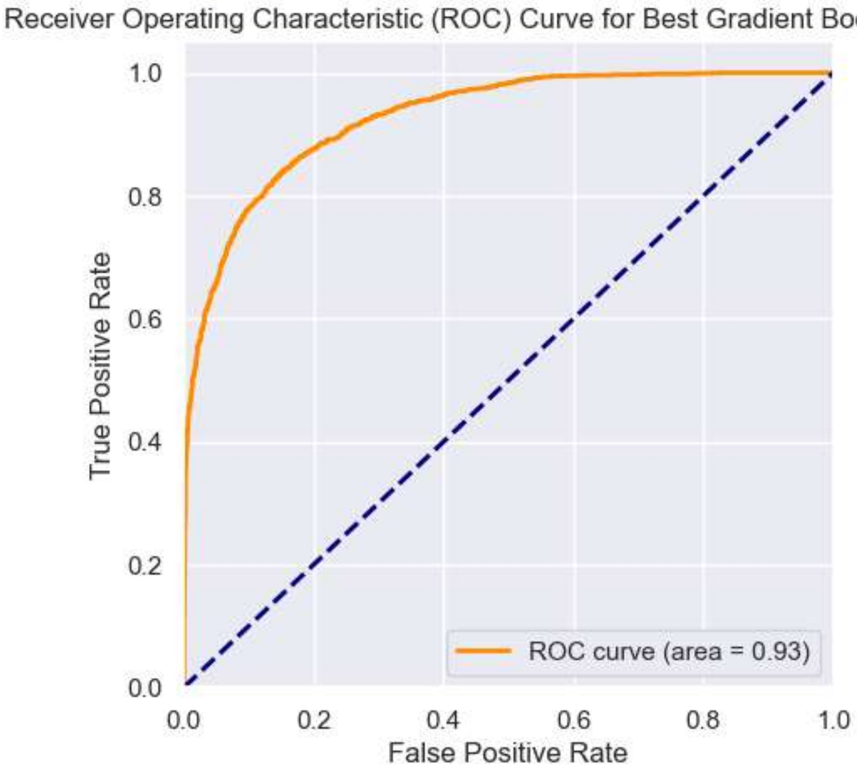
Although SVMs are known for their ability to perform well in various scenarios, their performance in this project was underwhelming. The significant run time and only moderate improvement in results led us to conclude that SVM was not the optimal choice for our dataset.

Next Model: Gradient Boosting

;

Gradient Boosting is an ensemble learning technique that builds models sequentially, where each new model attempts to correct the errors of its predecessor. It's particularly effective for both classification and regression tasks, often leading to high predictive performance.

```
Fitting 10 folds for each of 243 candidates, totalling 2430 fits
Best Parameters: {'learning_rate': 0.2, 'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimator
s': 100}
Best Accuracy Score: 0.8664342629482071
Accuracy of best model : 0.8624141021810576
Recall of best model : 0.7433348395842747
```



Accuracy:
The Gradient Boosting model achieved high accuracy, comparable to the results from the Random Forest model. This indicates a strong overall performance in predicting cancellations accurately.

Recall:
The recall score was also impressive, ensuring that a significant number of actual cancellations were correctly identified, which aligns with our primary objective.

AUC-ROC:
The AUC-ROC score was high, reflecting the model's excellent ability to distinguish between cancellations and non-cancellations.

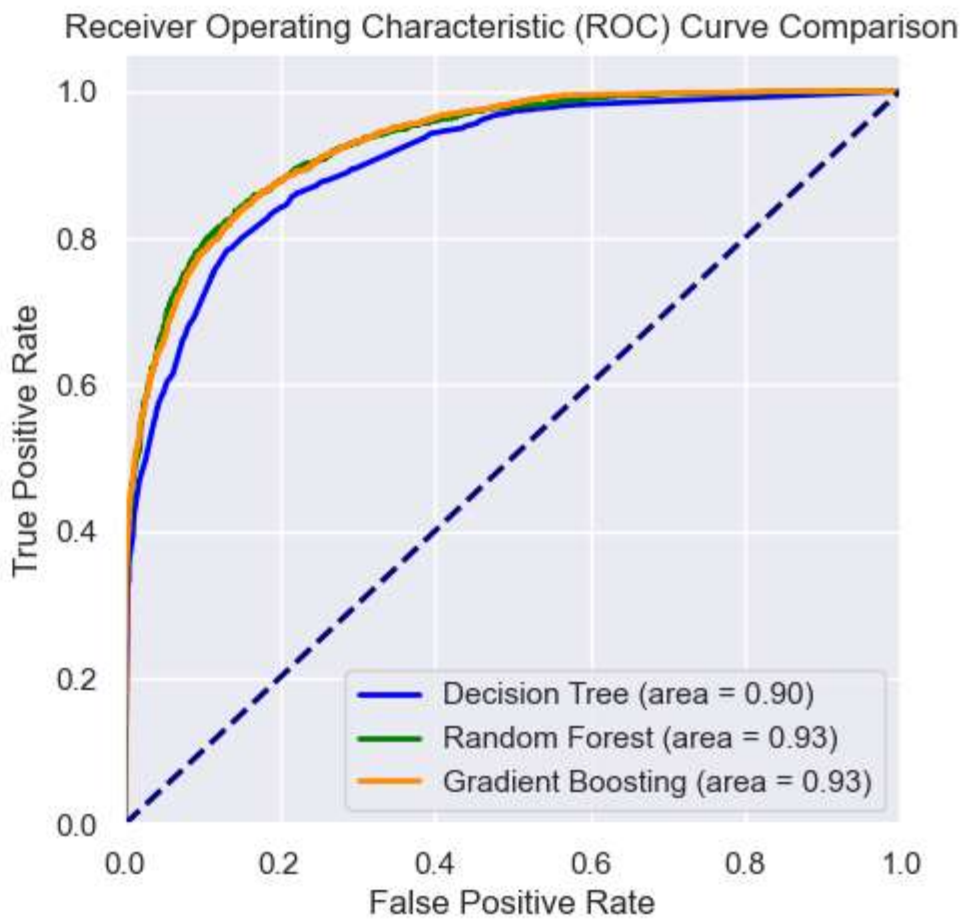
Comparison with Random Forest

Given that both Gradient Boosting and Random Forest delivered robust results, we decided to perform a more detailed comparison using statistical tests and visual tools:

;

T-Test Comparison:
We conducted a t-test to determine if the differences in performance metrics between the two models were statistically significant. This helps us understand if one model consistently outperforms the other.

ROC Curve Analysis:
We plotted the ROC curves for both models to visually compare their performance. The ROC curve helps us see how well each model distinguishes between the positive and negative classes across different threshold values.




```
Decision Tree vs Random Forest: T-statistic = 9.591623731788204, P-value = 1.01365833827967e-21
Decision Tree vs Gradient Boosting: T-statistic = 9.636610409906295, P-value = 6.566450837452832e-22
Random Forest vs Gradient Boosting: T-statistic = 0.37069307402860285, P-value = 0.7108720152014794
The difference between the Decision Tree and Random Forest is statistically significant.
The difference between the Decision Tree and Gradient Boosting is statistically significant.
The difference between the Random Forest and Gradient Boosting is not statistically significant.
```

Despite the similarities, we chose the Gradient Boosting model as our final model for the following reasons:

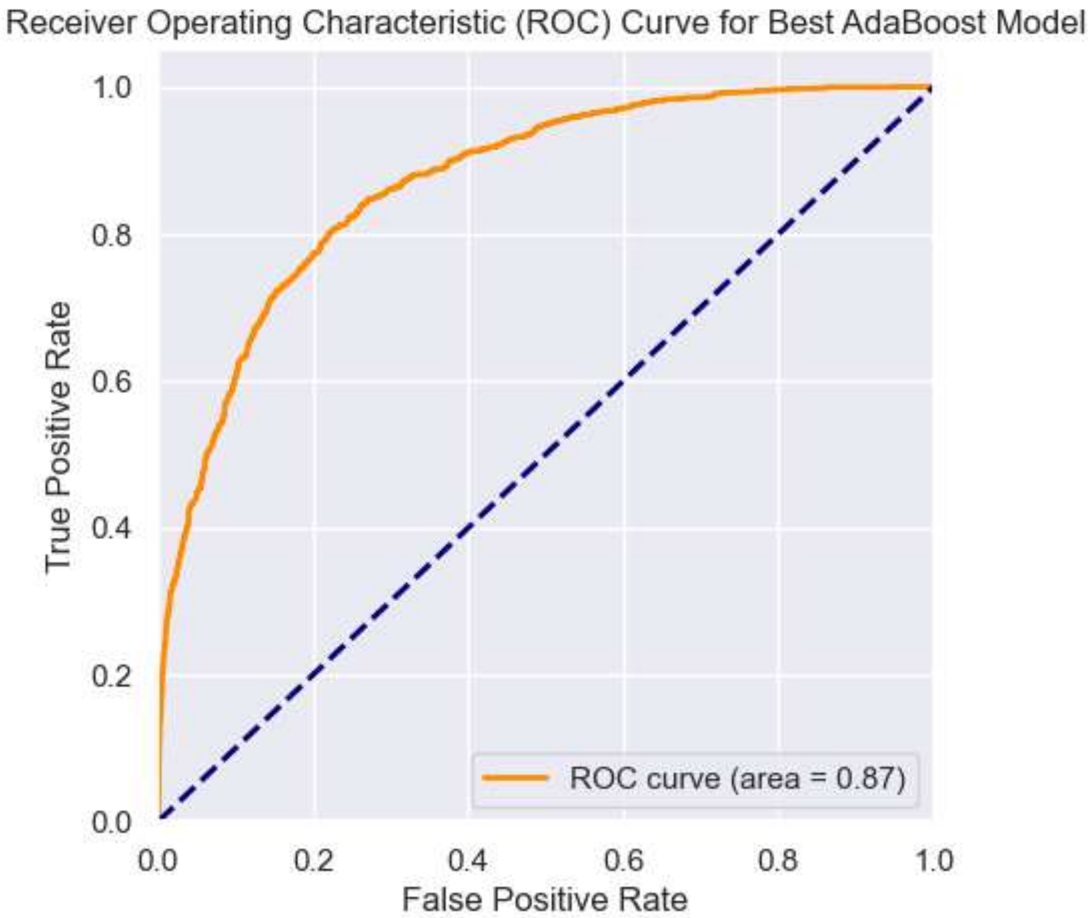
- Consistency: Gradient Boosting showed slightly more consistent results across different metrics and datasets.
- Implementation: The model's sequential nature allows for more control and fine-tuning, which can be beneficial for future improvements and iterations.

;

AdaBoost (Adaptive Boosting)

AdaBoost, is an ensemble learning method that combines multiple weak classifiers to create a strong classifier. It focuses on improving the performance of models by emphasizing the misclassified instances and adjusting the weights of these instances to guide the learning process.

```
Best Parameters: {'learning_rate': 1.0, 'n_estimators': 200}
Best Accuracy Score: 0.807121513944223
Accuracy of best model : 0.8069913355243502
Recall of best model : 0.6601897876186172
```



Accuracy:
The accuracy of the AdaBoost model was significantly lower compared to GB and RF, indicating a higher rate of incorrect predictions.

Recall:
Although recall is crucial for our primary objective, AdaBoost did not perform well in this metric either, missing a significant number of actual cancellations.

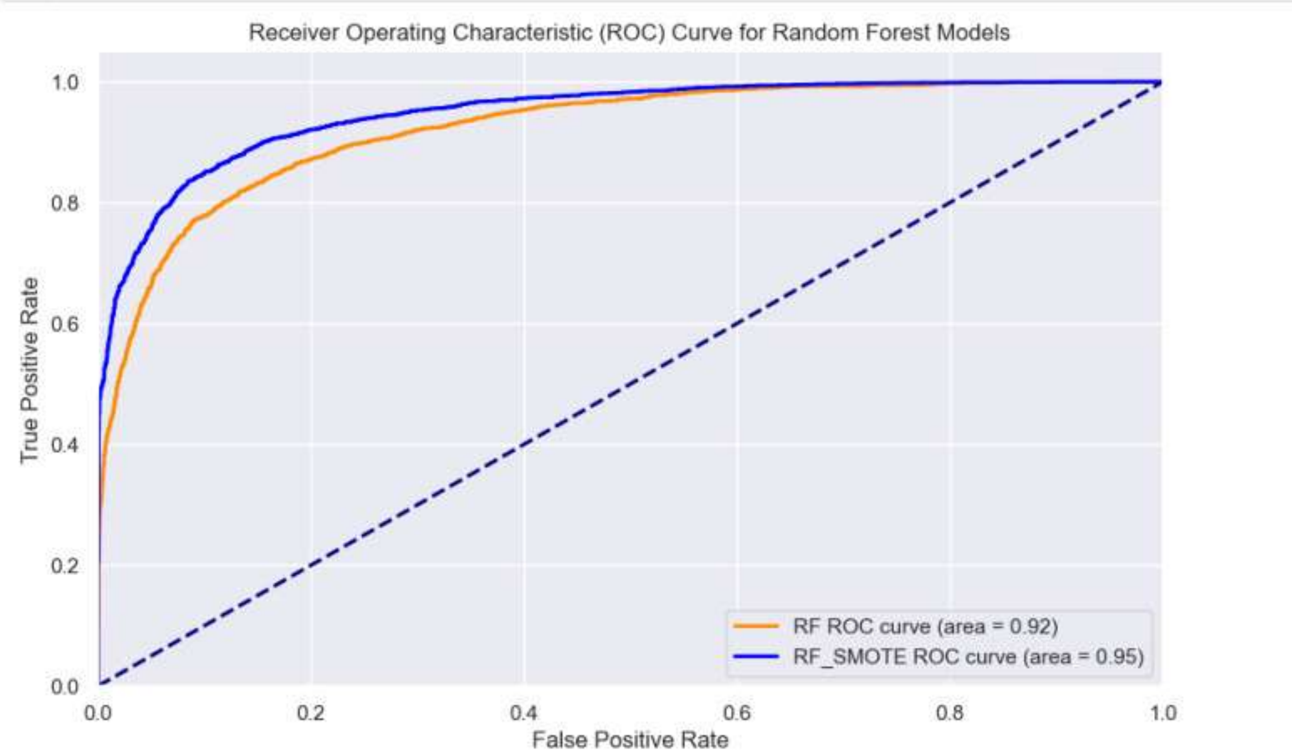
AUC-ROC:
The AUC-ROC score for AdaBoost was also lower, showing that it struggled more with distinguishing between cancellations and non-cancellations compared to GB and RF.

;

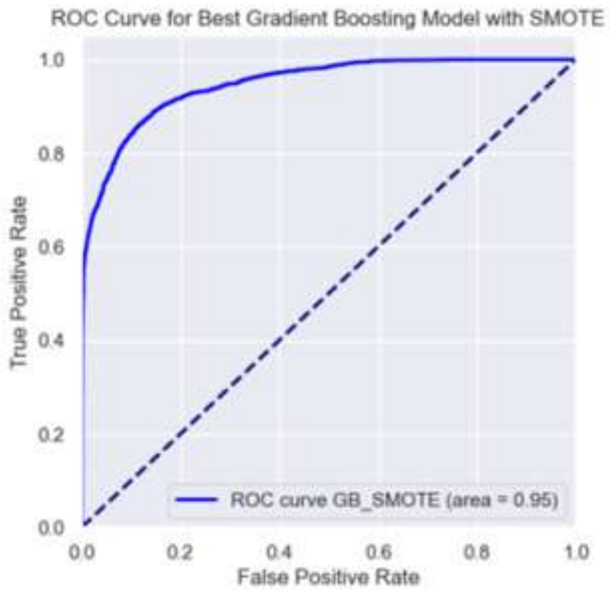
SMOTE

SMOTE is a method used to address class imbalance in our datasets. Instead of just copying the underrepresented samples, it generates new samples by combining similar ones. This approach helps create a more balanced dataset, which improves the performance of machine learning models by giving them a better variety of examples to learn from. By doing this, this method helps models to more accurately identify and predict instances from the minority class.

We added SMOTE for both of our leading models – Random Forest and Gradient Boosting to try to improve them. We got the following results:



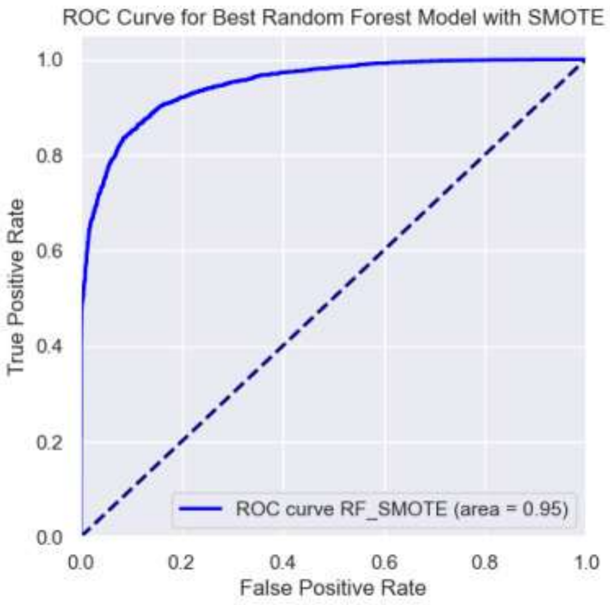

```
Fitting 10 folds for each of 243 candidates, totalling 2430 fits
Best Parameters for GB_SMOTE: {'learning_rate': 0.1, 'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_est
imators': 200}
Best Accuracy Score for GB_SMOTE: 0.8744424703925386
Accuracy of best GB_SMOTE model: 0.8740301876146142
Recall of best GB_SMOTE model: 0.8336538461538462
```



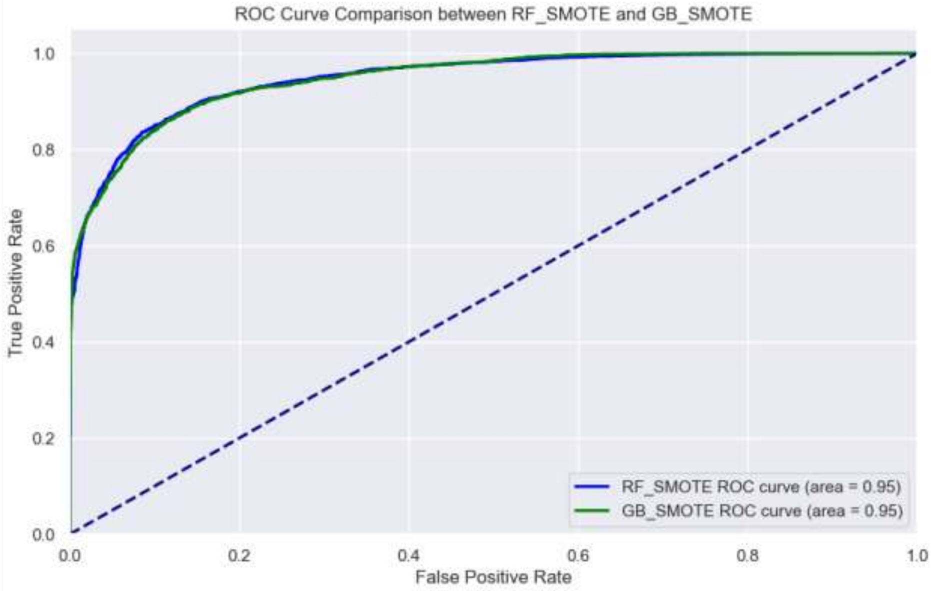

```
CV Accuracy Scores for GB_SMOTE: 0.8744424703925386 ± 0.007201776739914781
```

;

```
Fitting 10 folds for each of 81 candidates, totalling 810 fits
Best Parameters for RF_SMOTE: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200}
Best Accuracy Score for RF_SMOTE: 0.8785811903665678
Accuracy of best RF_SMOTE model: 0.8785442234447736
Recall of best RF_SMOTE model: 0.8285256410256411
```




```
CV Accuracy Scores for RF_SMOTE: 0.8785811903665678 ± 0.009586292090997744
CV Accuracy Scores for GB_SMOTE: 0.8744424703925386 ± 0.007201776739914781
T-statistic for RF_SMOTE vs GB_SMOTE: 1.035533998659503, P-value: 0.31413215702299147
The difference between RF_SMOTE and GB_SMOTE is not statistically significant.
```



Firstly, we can notice that adding SMOTE gives us better results compared to the regular model. We can also see that Random Forest + SMOTE and Gradient Boosting + SMOTE gives us practically the same quality of results.

Thus, we will add SMOTE to the selected model which will still be Gradient Boosting for the reasons explained before.

;

The final stage of our project involved loading the test data, applying the same preprocessing steps we performed on the training set, and using our Gradient Boosting model to generate predictions. Below is a detailed explanation of this process:

We began by loading the test dataset, which contained information about new hotel bookings that we needed to predict cancellations for.

We applied the same data cleaning and preprocessing steps to the test data as we did for the training data. This ensured that our model would be able to interpret the test data correctly.

Finally, we used our trained Gradient Boosting model (+Smote) to make predictions on the preprocessed test data.

Applying model and methods

Firstly, we loaded the data and printed the basic information like before:

```
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   9072 non-null   object
 1   weekend_nights       9072 non-null   int64
 2   week_nights          9072 non-null   int64
 3   room_type            9072 non-null   object
 4   board_type           6355 non-null   object
 5   n_adults             9072 non-null   int64
 6   n_less_12            9072 non-null   int64
 7   n_more_12            9072 non-null   int64
 8   booked_tour          9072 non-null   int64
 9   n_requests           9072 non-null   int64
 10  lead_time            8947 non-null   float64
 11  purchase_type        7461 non-null   object
 12  n_p_cacellation      9072 non-null   int64
 13  n_p_not_cacellation  9072 non-null   int64
 14  repeated             9072 non-null   int64
 15  price                7942 non-null   float64
 16  date                 9072 non-null   object
dtypes: float64(2), int64(10), object(5)
memory usage: 1.2+ MB
```

We see that there are missing values in the same 4 attributes like before: board type, price, purchase type and lead time.

After applying the data cleaning methods detailed above, we got:

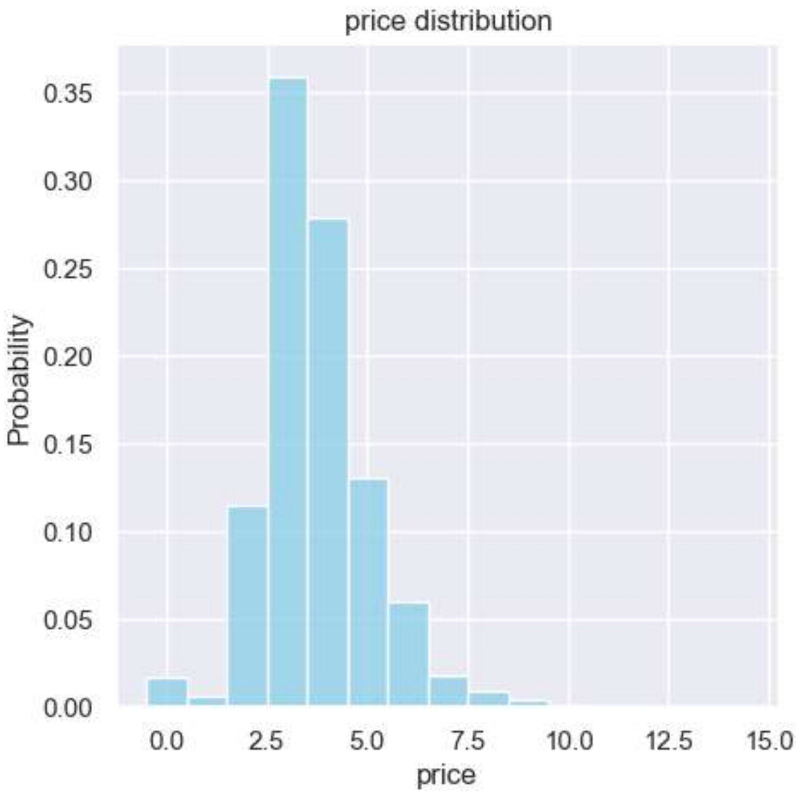
```
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   8930 non-null   object
 1   weekend_nights       8930 non-null   int64
 2   week_nights          8930 non-null   int64
 3   room_type            8930 non-null   object
 4   board_type           8930 non-null   object
 5   n_adults             8930 non-null   int64
 6   n_less_12            8930 non-null   int64
 7   n_more_12            8930 non-null   int64
 8   booked_tour          8930 non-null   int64
 9   n_requests           8930 non-null   int64
 10  lead_time            8930 non-null   float64
 11  purchase_type        8930 non-null   object
 12  n_p_cacellation      8930 non-null   int64
 13  n_p_not_cacellation  8930 non-null   int64
 14  repeated             8930 non-null   int64
 15  price                8930 non-null   float64
 16  date                 8930 non-null   object
```

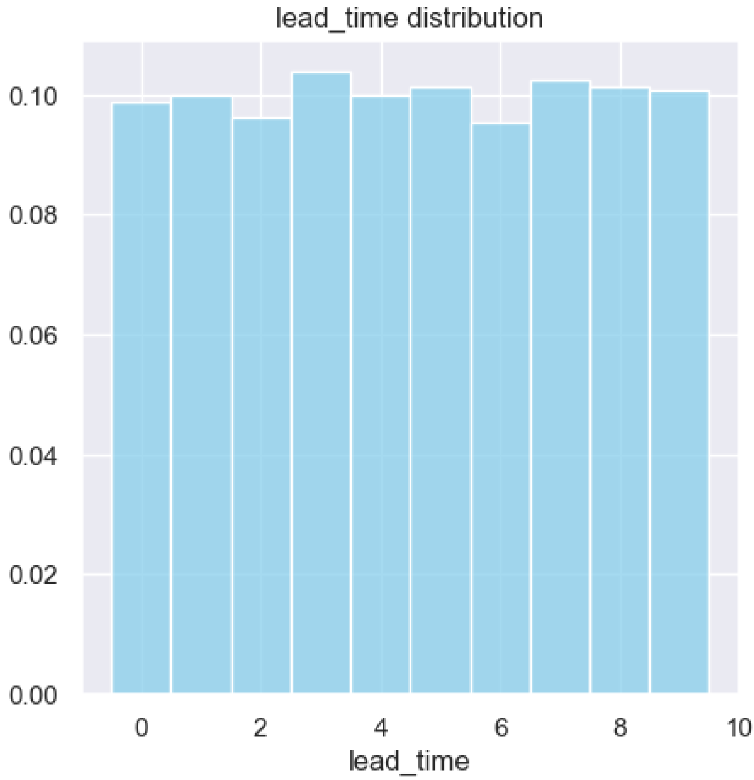
;

We notice that there are less entries – 8930 compared to 9072 – because we dumped several entries, mainly as part of the data cleaning for the lead time attribute. Yet now we have no missing value for no attribute.

After that, we applied the methods from the previous parts to discretize the continuous data, transforming it into distinct, categorical intervals for more effective model performances.

The results:





After that we applied the model in order to create predictions and saved the results in 'hotels_test_pre.csv'.

;