INN Hotels Project

Supervised Learning - Decision Trees

Pauline Zeestraten
January, 2024

Contents / Agenda

- Executive Summary
- Business Problem Overview and Solution Approach
- EDA Results
- Data Preprocessing
- Model Performance Summary
- Appendix

Executive Summary

- Our analysis shows that 32.8% of all bookings are canceled, and that canceled rooms are on average priced higher than non-canceled rooms.

- We found that a shorter lead time in bookings, results in much fewer cancellations. Approximately 75% of non cancelled bookings were made less than 100 days prior to the booked date.

- Online booking makes up 64% of the market segment designation. Online booked rooms comprised 35% of all rooms that were booked at $0, a result from last minute cancellations that are notoriously hard to get rebooked.

- Our analysis identified several factors that increase the likelihood of cancellations, and factors that influence demand on rooms, allowing for higher prices per room. All these factors should be incorporated in INSS Hotel group's dynamic pricing strategies to the customer at the time of making a reservation.

Executive Summary continued

- Our univariate analysis on “previous bookings not canceled” shows a very low number, suggesting that the INSS hotel group in Portugal does not welcome many repeat customers. This should be an area of focus. Our suggestion is to increase efforts enticing existing customers to come back in the future with their families, possibly on weekends or holidays. Our analysis shows that currently 92.6% of all bookings do not include any children, and that 46.5% of all bookings do not include any weekend night. These are great opportunities for further exploring and growth.

- Moreover, our bivariate analysis shows that repeat customers cancel far less frequently than first time customers. All the more reason to cultivate the number of repeat customers.

- Our analysis shows that “average price per room” peak between May and September, but declines steadfast thereafter, missing out on capitalizing on the increased demand for rooms in October.

- Our analysis shows a long lead time is a risk factor for cancellations.

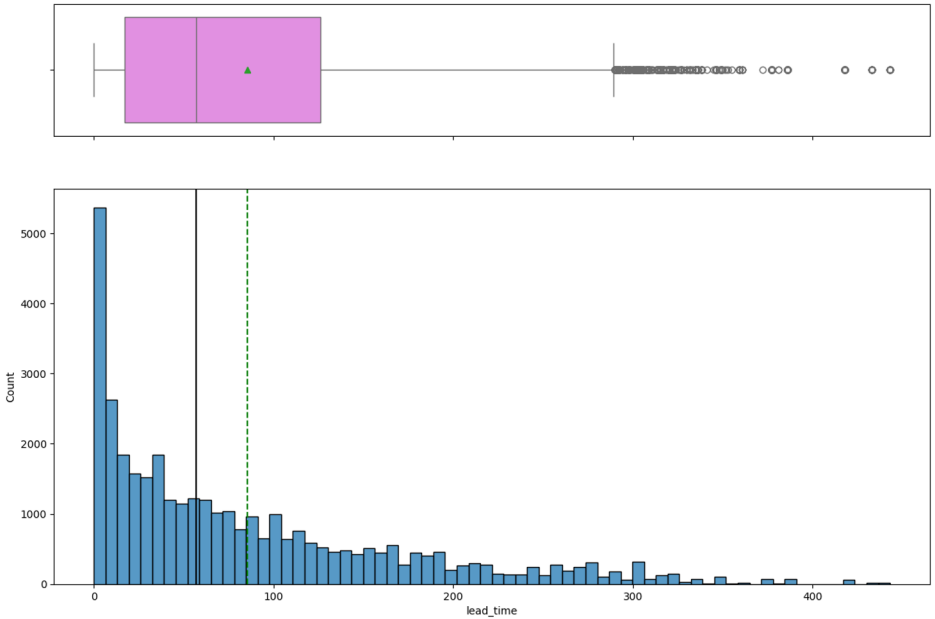
Business Problem Overview and Solution Approach

INNS Hotels Group is seeking data-driven solutions for the high number of booking cancellations at their chain of hotels in Portugal. Typical known reasons for cancellations include change of plans, scheduling conflicts, etc., and they are often made easier by the option to do so free of charge or at low cost which is beneficial to hotel guests, but not to INNS Hotels Group's bottom line.

I have analyzed the data provided by INNS Hotels Group, and in this report I share:

- Which factors have a high influence on booking cancellations.
- A predictive model that can predict which booking is likely going to be cancelled in advance.
- Advice in formulating profitable policies for cancellations and refunds.

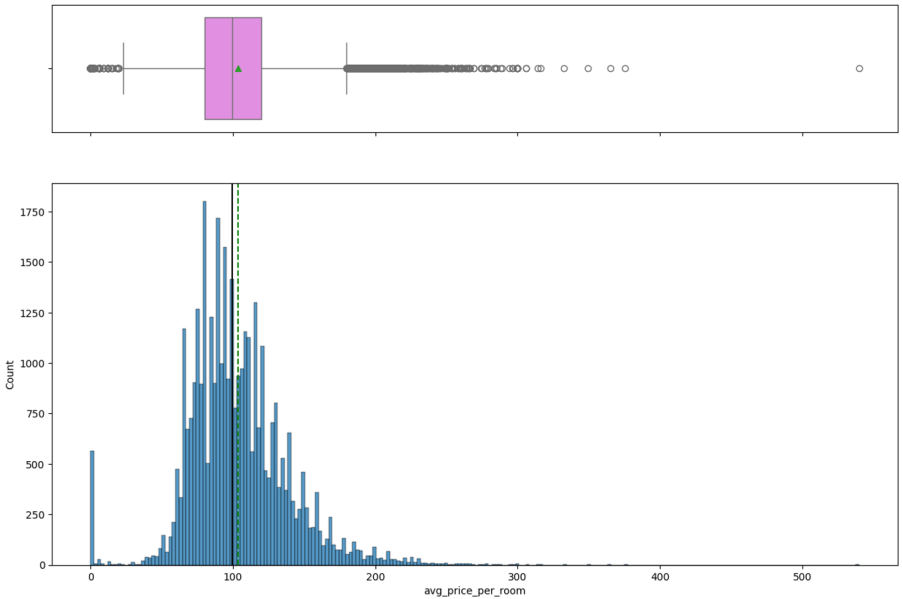
EDA - Univariate analysis - "Lead Time"



- We observe that the data for "lead time" is heavily right skewed with a long tail of outliers, while the peak of the distribution is at the very left end of the x-axis. This means the majority of bookings are made with relatively little, or very little lead time.
- To illustrate with numbers: the minimum value is 0 days, while the maximum lead time is 443 days.
- The mean is approximately 85 days, and the median 57.

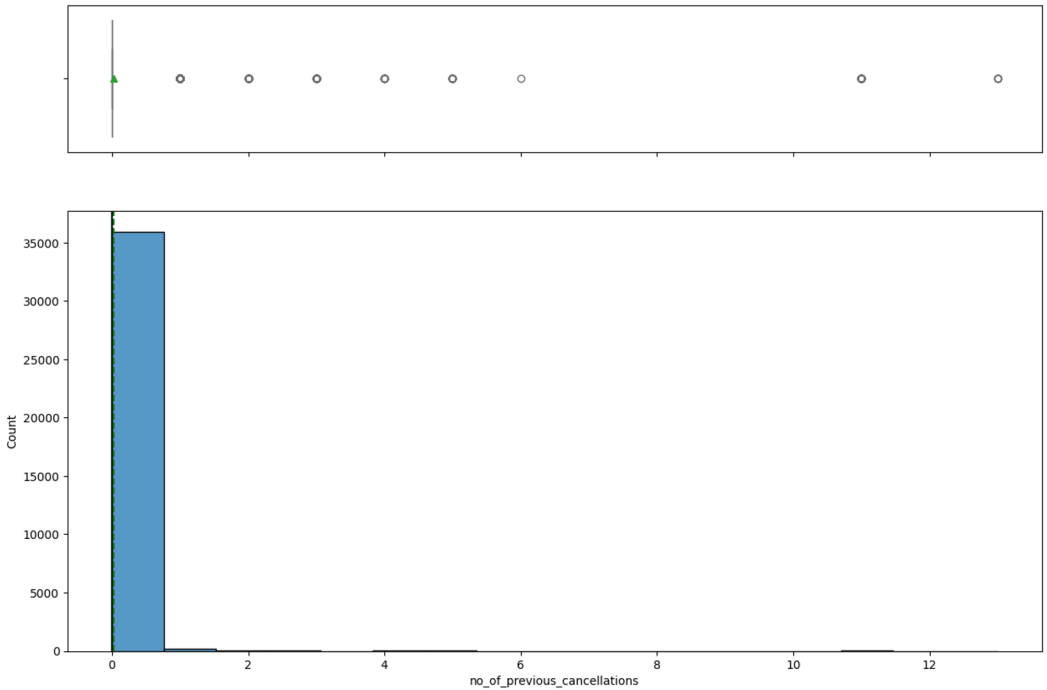
Link to Appendix slide on data background check

EDA - Univariate analysis - "Average price per room"



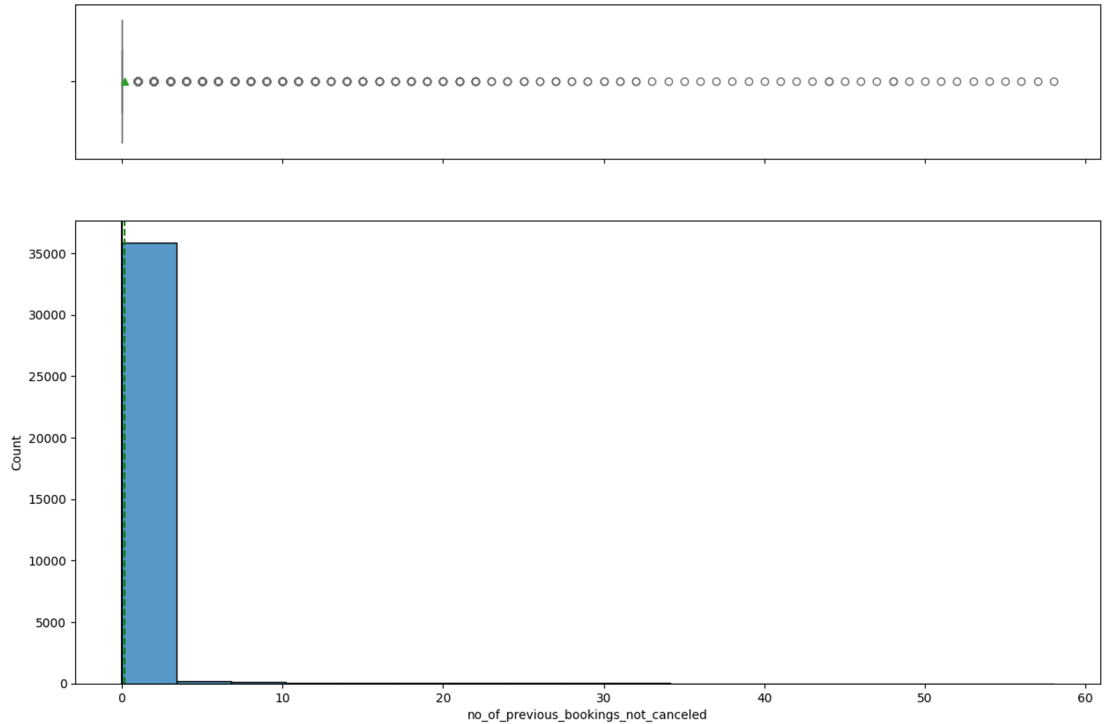
- We observe that the data for average price per room is right skewed.
- Outliers on the left of the distribution occur in the form of a concentrated peak at $0 value. These represent cancellations and no-shows.
- Outliers on the right are spread between rates of $120 and $540.
- The mean and median are $103, and $99 respectively.

EDA - Univariate analysis - “no of previous cancellations”



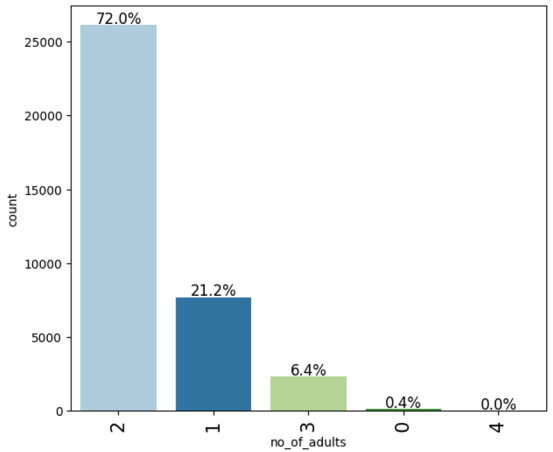
- We observe that the number of previous canceled booking reservations is low.
- Mean and Median are at 0 cancellations.
- Two right skewed extreme outliers with 13 and 11 cancellations.

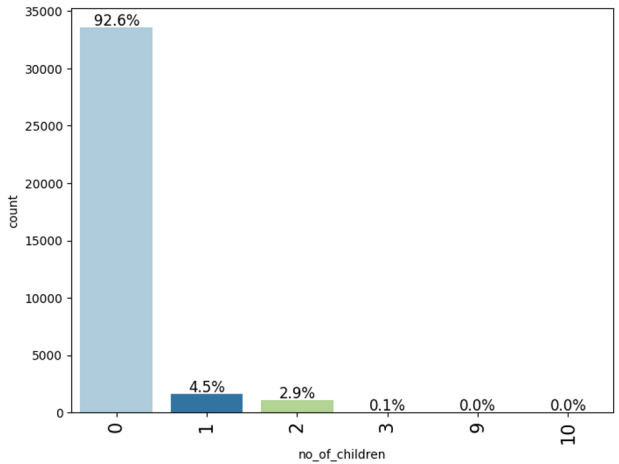
EDA - Univariate analysis - “no of previous bookings not canceled”



- Observation that “number of previous bookings not canceled” are very low, indicating not many repeat customers at all.
- The median and mean are 0 and 0.15 number of previous bookings not cancelled.

EDA - Univariate analysis - "no of adults", "no of children"

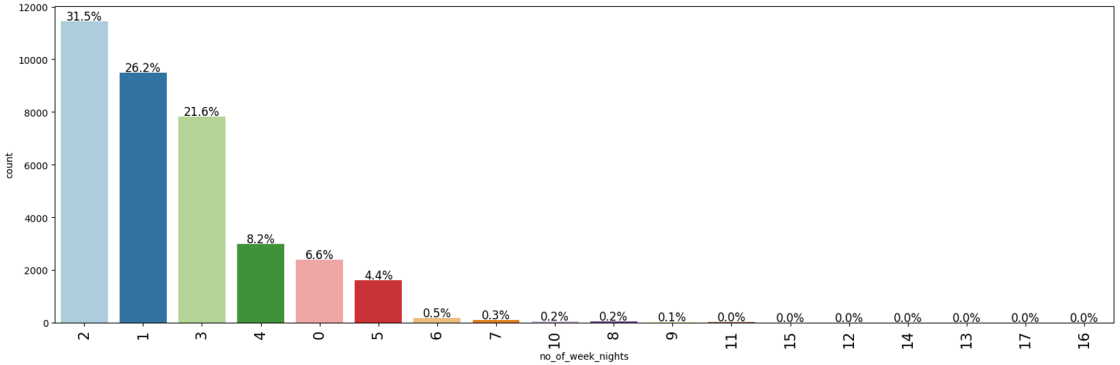




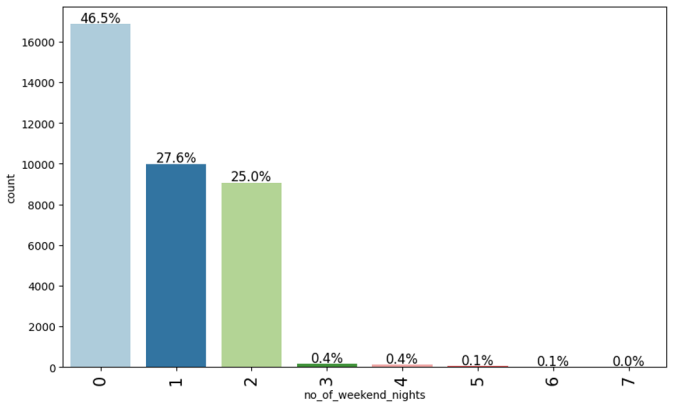
- Observation that most prevalent number of adults per booking is 2 with 72%, while the number of children most prevalent per booking is 0 with 92.6%.

- This implies very few rooms are booked with families, and this could be an area of focus for growth in the future.

EDA - Univariate analysis - "no of week nights", "no of weekend nights"



Observation on number of week nights: 2, 1, and 3 night bookings during the week comprise 79% of all bookings placed.

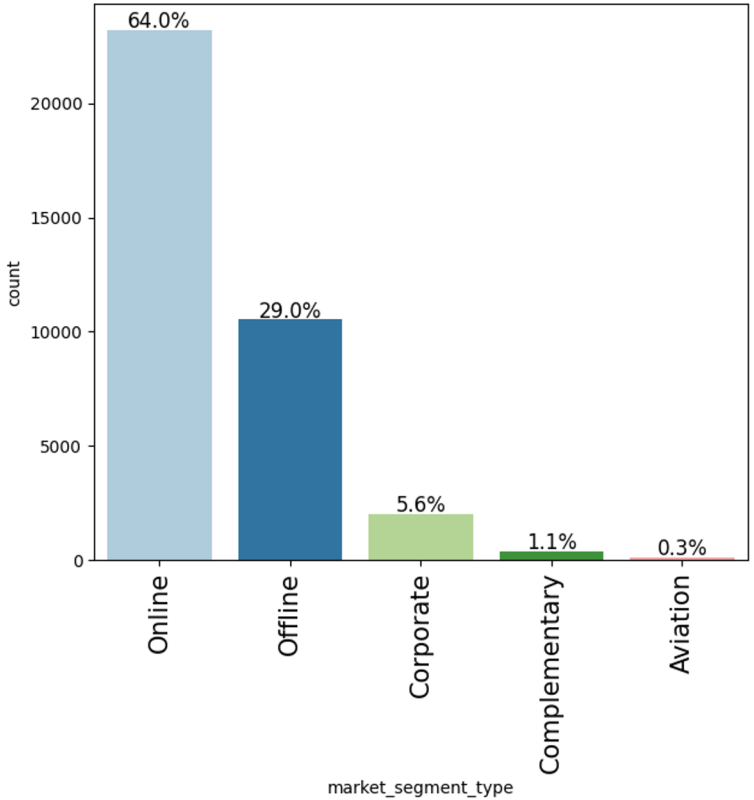


Observation on number of weekend nights: 46.5% of all bookings do not include a weekend night. This is an opportunity for growth, to be combined with focus on increase of family bookings.

EDA - Univariate analysis - "parking space," "meal plan", "room type", "arrival month", "no special requests", "booking status"

- Observation that 96.9% of all bookings do not require a parking space.
- Observation that 76.7% of all bookings request meal plan 1, and 14.1% does not select any meal plan.
- Observation that 77.5% of all bookings reserve room type 1.
- Observation that bookings based on arrival month show no dramatic peaks. The months October, September, August and June account for 46.7% of all annual bookings.
- Observation that 54.5% of bookings didn't come with a special request, 31.4% had one request.
- **Observation that 67.2% of all bookings are not cancelled, but 32.8% are!**

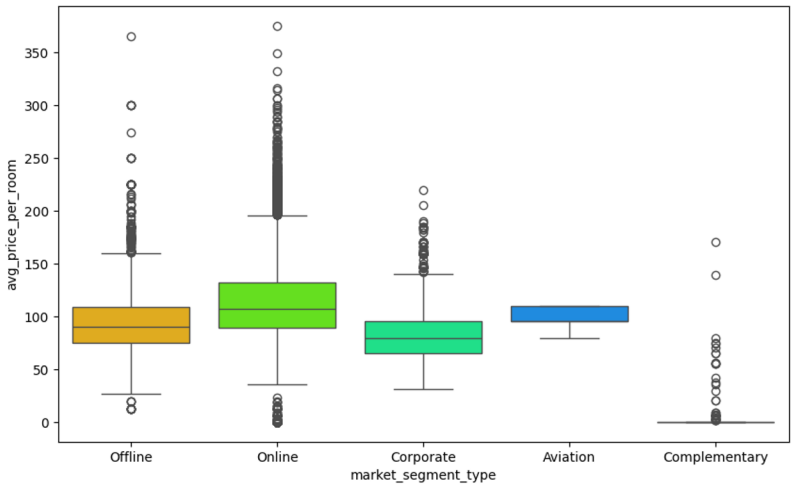
EDA - Univariate analysis - "market segment type"



- Observation that online bookings are by far the largest market segment type.
- This means this particular market segment needs extra focus and consideration when it comes to cancellations.

EDA - Bivariate analysis - "avg price per room " and "market segment type"

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x="market_segment_type",
y="avg_price_per_room", palette="gist_rainbow")
plt.show()
```




```
data.loc[data["avg_price_per_room"] == 0,
"market_segment_type"].value_counts()
```

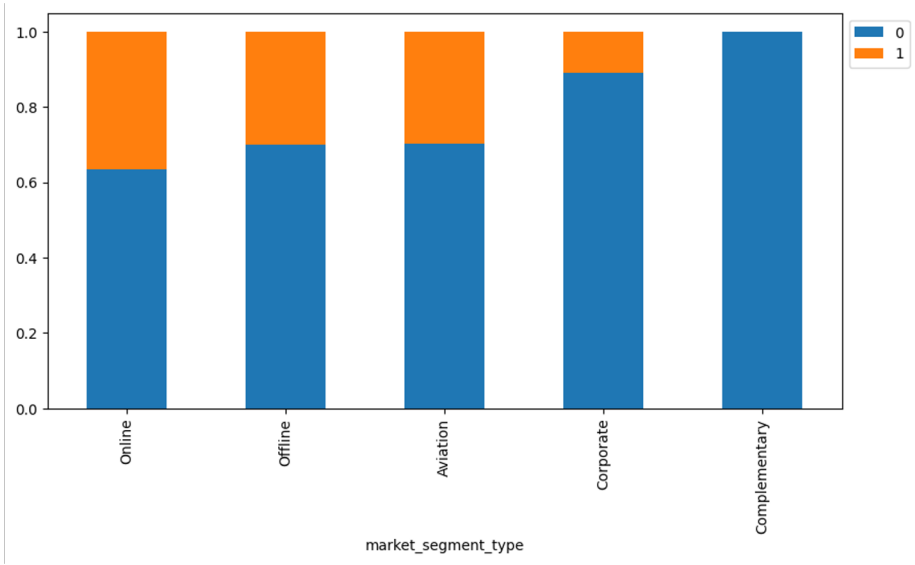
Complementary	354
Online	191

- Average price for rooms booked online are highest of all market segments. Online segment also has significant number of outliers in the $0 and low cost range, indicating high # late cancellations.
- Not clear from data if complimentary market segment is meant to be deployed as marketing strategy, or functions as contingency allocation for late cancellations and no shows, or a combination thereof.
- Market segments Corporate and Aviation, and to a degree Offline, all have no or few rooms booked at $0 or very low cost.

EDA - Bivariate analysis - "market segment type " and "booking status"

- Observation that cancellations occur most frequent in Online market segment, followed by Aviation and Offline market segments.

```
stacked_barplot(data, "market_segment_type", "booking_status")
```

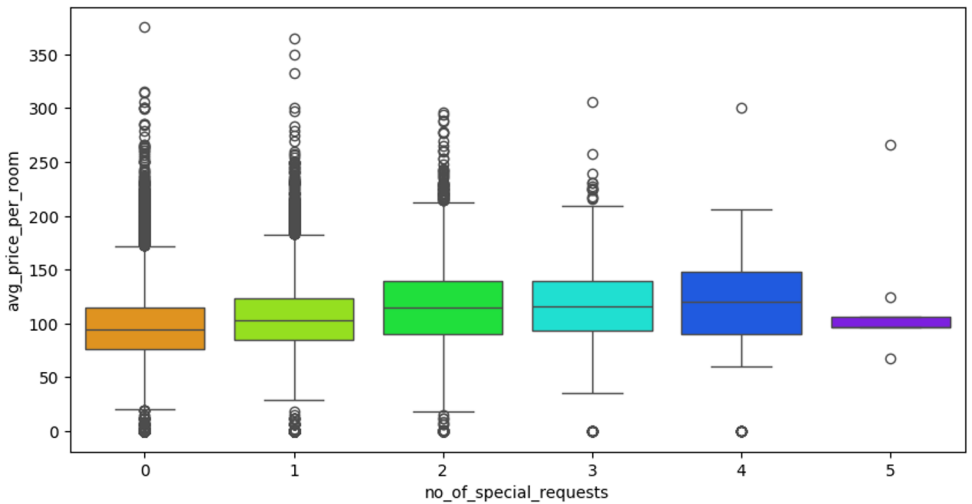



```
0=  Blue =  not canceled
1=  Orange= canceled
```

booking_status	0	1	All
market_segment_type			
All	24390	11885	36275
Online	14739	8475	23214
Offline	7375	3153	10528
Corporate	1797	220	2017
Aviation	88	37	125
Complementary	391	0	391

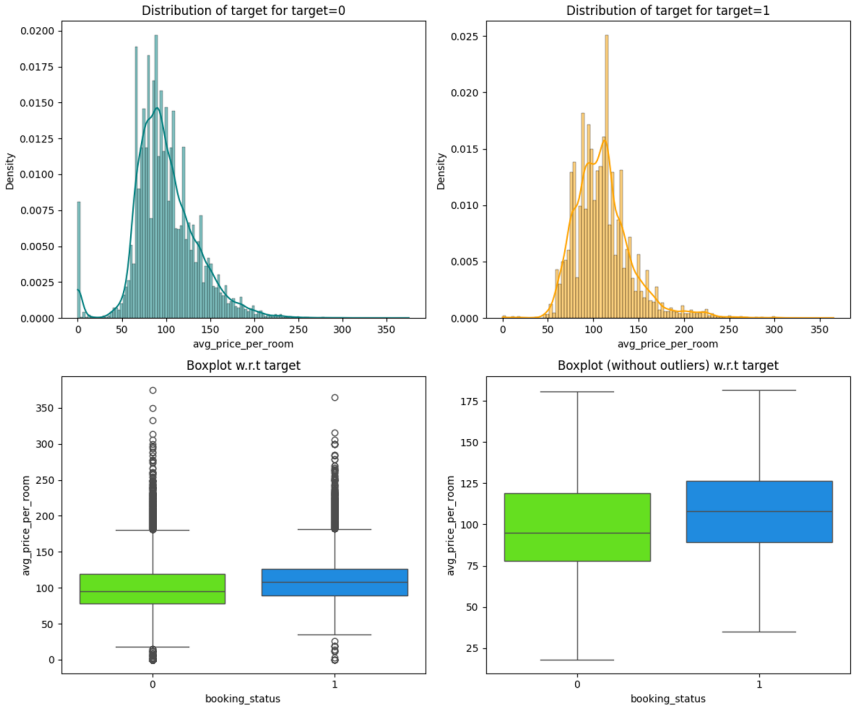
EDA - Bivariate analysis - "special requests", "booking status", "avg price p rm"

- The no of special requests when booking a room has following impact (see appendix for code and plot): Observation that ~43% of cancellations of bookings occur with zero special requests, followed by 23.7% of cancellations with one special request, and 14.5% of cancellations with two special requests.



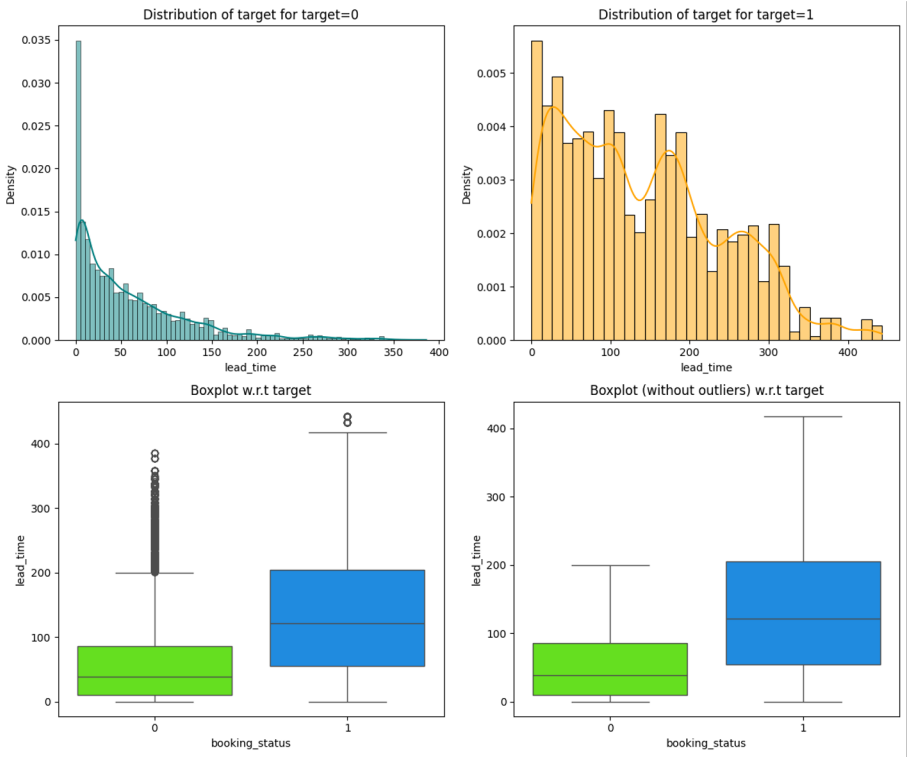
- Observation on plot to the left: Average price of the room has a linear correlation with the number of special requests made. The price for a room increases when the number of requests, up to four specifically, increase.

EDA - Bivariate analysis - "booking status" and "avg price per room"



- Observation that the average price of the rooms that are cancelled are higher than of the rooms that are not cancelled.
- This is obviously at a big disadvantage to INSS Hotel Group.

EDA - Bivariate analysis - "booking status" and "lead time"



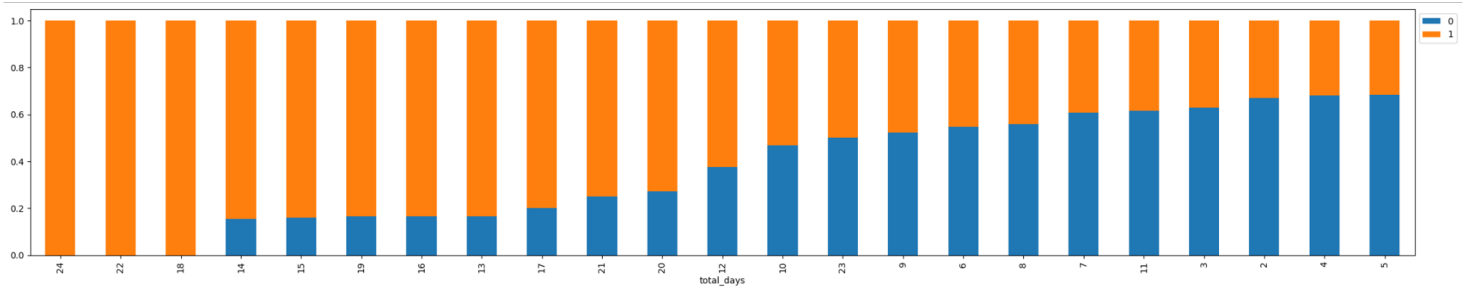
- Observation that a shorter lead time in bookings results in much fewer cancellations.
- Approximately 75% of non cancelled bookings were made less than 100 days prior to the booked date.

EDA - Bivariate analysis - "booking status" and "no of family members"

- Observation that the percentage of cancellations is highest among families with 4 family members at ~44%.
- Families with respectively 3, 5, and 2 family members, all have percentage rates of cancellations at around 35%.

- "booking status" and "total days"

- Observation that percentage of cancellation is 50% or higher for reservations with duration of stay longer than 10 days.



EDA - Bivariate analysis - "booking status" and "repeated guest"

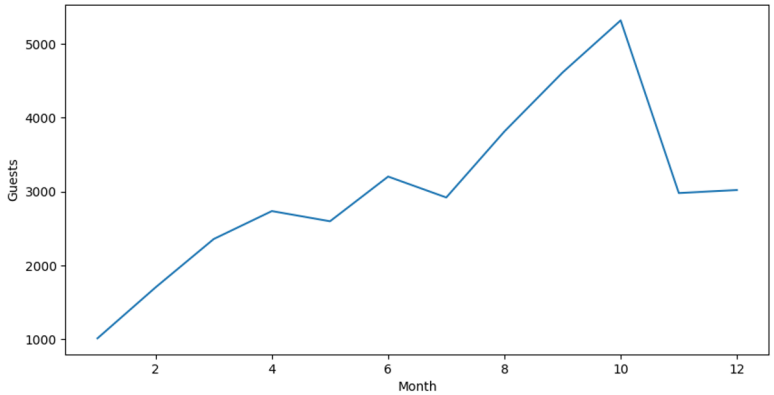
- Observation that repeated guests cancel far fewer with 1.72%, than first time customers at 33.5%.

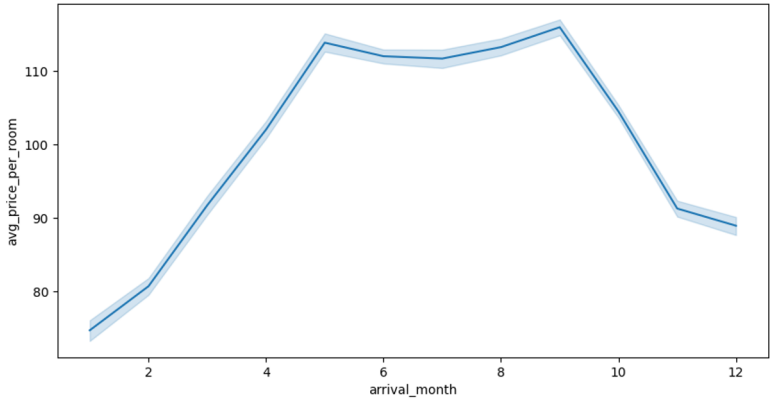
- "booking status" and "arrival month"

- Observation that the summer months July (45%), June (40%), and August (39%) account for the largest share of cancelled bookings over the year.

EDA - Bivariate analysis - "arrival month", "guests, "avg price per room"

- Observation that the month October is by far the busiest month of the year with more than 5000 guests. January the least with ~1000 guests staying.
- Observation that "average price per room" has two peaks around May and Sept, and then rapidly lowers during the remainder of the year, missing out on capitalizing on demonstrated increased demand for rooms in October.





Data Preprocessing - duplicates and missing values:

- Duplicate value check: There are no duplicate values in the dataset.

```
data.duplicated().sum() ## Complete the code to check duplicate entries in the data
```

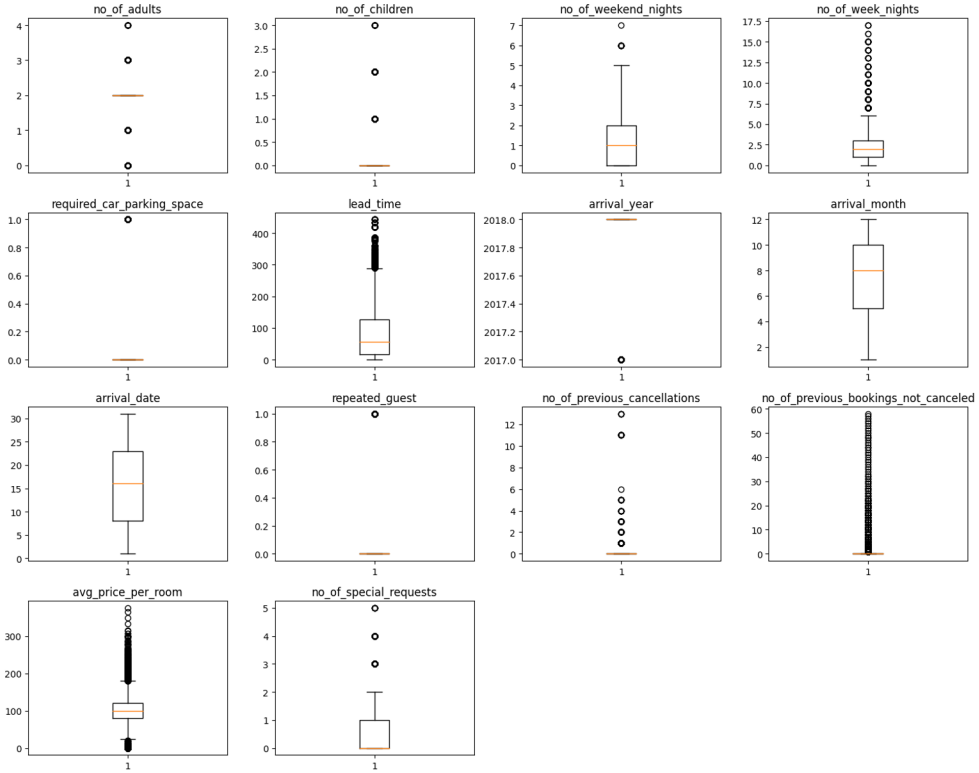
0

- Missing value treatment: There are no missing values that need treatment.

```
# checking for missing data (empty cells)
data.isnull().sum()
```

```
Booking_ID                              0
no_of_adults                            0
no_of_children                          0
no_of_weekend_nights                    0
no_of_week_nights                       0
type_of_meal_plan                       0
required_car_parking_space              0
room_type_reserved                      0
lead_time                               0
arrival_year                            0
arrival_month                           0
arrival_date                            0
market_segment_type                     0
repeated_guest                          0
no_of_previous_cancellations            0
no_of_previous_bookings_not_canceled    0
avg_price_per_room                      0
no_of_special_requests                  0
booking_status                          0
dtype: int64
```

Data Preprocessing - Let's check for outliers in the data:



We are treating the right skewed outliers for "Average price per room" as follows:

```
# Calculating the 25th quantile
Q1 = data["avg_price_per_room"].quantile(0.25)
# Calculating the 75th quantile
Q3 = data["avg_price_per_room"].quantile(0.75)
## Complete the code to calculate 75th qntle.
# Calculating IQR  IQR = Q3 - Q1
# Calculating value of upper whisker
Upper_Whisker = Q3 + 1.5 * IQR
Upper_Whisker
179.55
# assigning outliers value of upper whisker
data.loc[data["avg_price_per_room"] >= 500,
"avg_price_per_room"] = Upper_Whisker
```

Data Preprocessing - for Logistic Regression Modeling

- No Feature Engineering needed.
- We want to predict which bookings will be canceled.
- Before we proceed to build a model, we'll have to encode categorical features.
- We'll split the data into train and test to be able to evaluate the model that we build on the train data.

```
Shape of Training set :  (25392, 28)
Shape of test set :  (10883, 28)
Percentage of classes in training set:
0   0.67064
1   0.32936
Name: booking_status, dtype: float64
Percentage of classes in test set:
0   0.67638
1   0.32362
Name: booking_status, dtype: float64
```

```
X = data.drop(["booking_status"], axis=1)
Y = data["booking_status"]
# adding constant
X = sm.add_constant(X) ## Complete the code to add constant to X
X = pd.get_dummies(X, drop_first=True) ## Complete the code to create dummies for X

# Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.3, random_state = 1) ## Complete the code to split the data into train test in the ratio 70:30 with random_state = 1
```

Data Preprocessing - for Decision Tree Modeling

- We want to predict which bookings will be canceled.
- Before we proceed to build a model, we'll have to encode categorical features.
- We'll split the data into train and test to be able to evaluate the model that we build on the train data.
- This is the same procedure as for logistic regression modeling.

```
Shape of Training set :  (25392, 28)
Shape of test set :  (10883, 28)
Percentage of classes in training set:
0   0.67064
1   0.32936
Name: booking_status, dtype: float64
Percentage of classes in test set:
0   0.67638
1   0.32362
Name: booking_status, dtype: float64
```

```
X = data.drop(["booking_status"], axis=1)
Y = data["booking_status"]

#converting categorical values into dummies
X = pd.get_dummies(X, drop_first= True) ## Complete the code to create dummies for X

# Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.30, random_state = 1, stratify=Y) ##
Complete the code to split the data into train test in the ratio 70:30 with random_state = 1
```

Model Performance Summary

- Overview of the final ML model and its parameters
- Summary of most important features used by the ML model for prediction

***Note**: You can use more than one slide if needed*

Model Performance Summary - logistic regression models

Training performance comparison:

	Logistic Regression-default Threshold	Logistic Regression-0.37 Threshold	Logistic Regression-0.42 Threshold
Accuracy	0.80545	0.79265	0.80132
Recall	0.63267	0.73622	0.69939
Precision	0.73907	0.66808	0.69797
F1	0.68174	0.70049	0.69868

Testing performance comparison:

	Logistic Regression-default Threshold	Logistic Regression-0.37 Threshold	Logistic Regression-0.42 Threshold
Accuracy	0.80465	0.79555	0.80345
Recall	0.63089	0.73964	0.70358
Precision	0.72900	0.66573	0.69353
F1	0.67641	0.70074	0.69852

- Logistic Regression model with threshold = 0.37 has the highest F-1 score of 0.70
- The difference between training and test results is minima (<5%), which means there is no case of overfitting, and the data is not overly complicated.
- F1 score is the key metric to maximize, as this will decrease both the False Positives and False Negatives, which is the objective for INSS Hotel Group in bringing costs down.

Model Performance Summary - decision tree models

Training performance comparison:

	Decision Tree sklearn	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Accuracy	0.99437	0.83554	0.90981
Recall	0.98570	0.78339	0.91862
Precision	0.99708	0.73299	0.82572
F1	0.99136	0.75735	0.86969

Test set performance comparison:

	Decision Tree sklearn	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Accuracy	0.86529	0.83212	0.86015
Recall	0.79445	0.76921	0.84044
Precision	0.79445	0.73205	0.75873
F1	0.79445	0.75017	0.79750

- Decision Tree with post-pruning gives the highest F1 score of 0.79750.
- F1 score is the key metric in this analysis for INSS Hotel Group, as it minimizes both the False Positives and False Negatives.

APPENDIX

Data background and contents -Python functions for dataset:

```
hotel = pd.read_csv('/content/drive/MyDrive/INNHotelsGroup.csv') ## Fill the blank to read the data
data = hotel.copy()
data.head() ## Complete the code to view top 5 rows of the data
data.tail() ## Complete the code to view  last 5 rows of the data
data.shape ## Complete the code to view dimensions of the data
```

- There are 36,275 rows and 19 columns in this dataset.

```
data.info()
```

- There are no missing values.
- Of the 19 columns, the following 5 have string as datatype: "Booking ID", "Type of meal plan", "Room type reserved", "Market segment Type", and "Booking status". All other columns are numerical with either integers or floats as datatype.

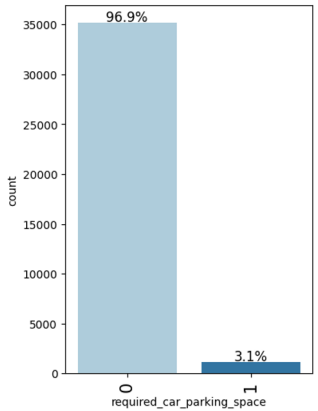
```
data = data.drop(['Booking_ID'], axis=1) ## Complete the code to drop the Booking_ID column from the dataframe
data.head()
```

EDA - Univariate Analysis - python functions

- `data.describe().T ## Complete the code to print the statistical summary of the data`
- `histogram_boxplot(data, "lead_time")`
- `histogram_boxplot(data, 'avg_price_per_room') ## Complete the code to create histogram_boxplot for average price per room`
- `data[data["avg_price_per_room"] == 0]`
- `histogram_boxplot(data, 'no_of_previous_cancellations') ## Complete the code to create histogram_boxplot for number of previous booking cancellations`
- `histogram_boxplot(data, 'no_of_previous_bookings_not_canceled') ## Complete the code to create histogram_boxplot for number of previous booking not canceled`

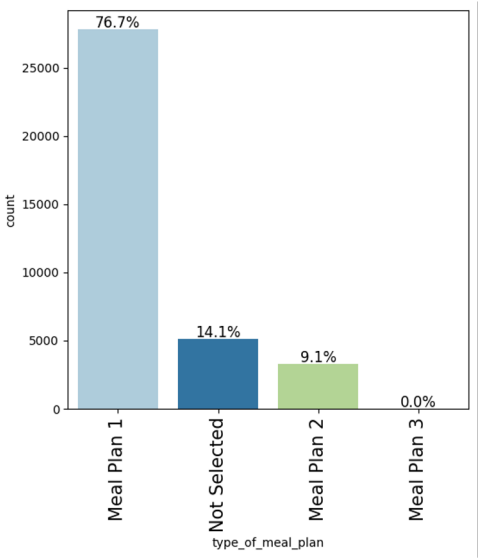
EDA - Univariate Analysis - python functions + plots

- `labeled_barplot(data, "no_of_adults", perc=True)`
- `labeled_barplot(data, 'no_of_children', perc=True) ## Complete the code to create labeled_barplot for number of children`
- `labeled_barplot(data, 'required_car_parking_space', perc=True) ## Complete the code to create labeled_barplot for car parking space`

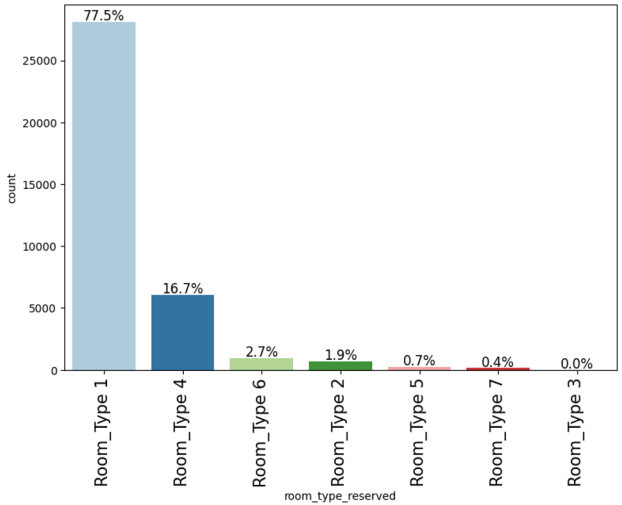


EDA - Univariate Analysis - “type of meal plan” - “room type reserved”

```
labeled_barplot(data,'type_of_meal_plan', perc=True) ## Complete the code to create labeled_barplot for type of meal plan
```

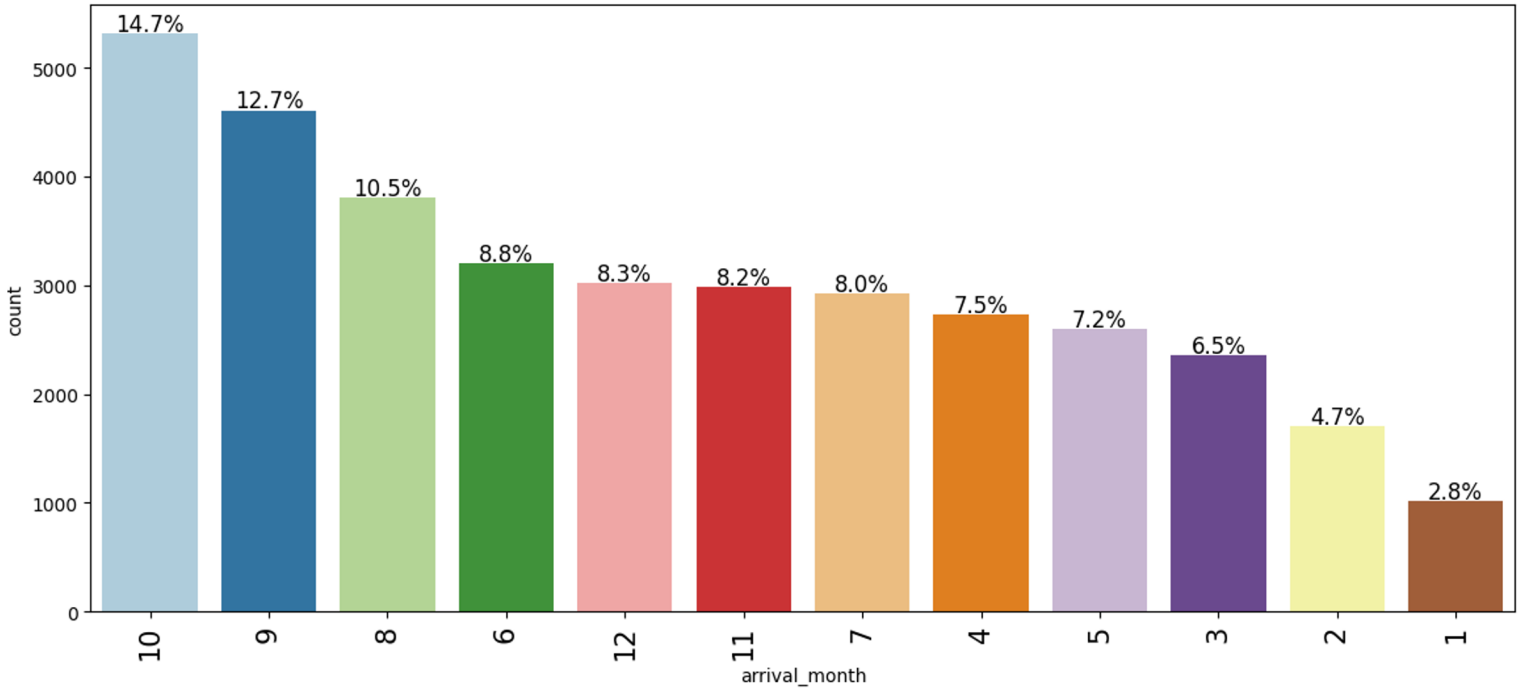



```
labeled_barplot(data,'room_type_reserved', perc=True) ## Complete the code to create labeled_barplot for room type reserved
```



EDA - Univariate Analysis - “arrival month”

```
labeled_barplot(data, 'arrival_month', perc=True) ## Complete the code to create labeled_barplot for arrival month
```

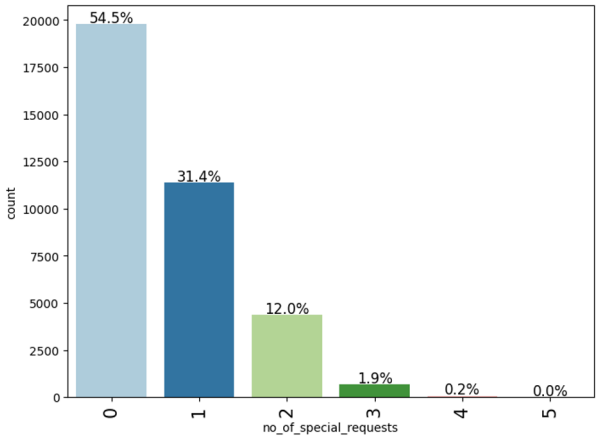


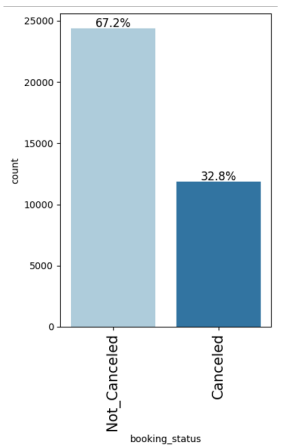
EDA - Univariate Analysis - “market segment type”, “requests”, “booking status”

- `labeled_barplot(data, 'market_segment_type', perc=True)` ## Complete the code to create labeled_barplot for market segment type

- `labeled_barplot(data,'no_of_special_requests', perc=True)` ## Complete the code to create labeled_barplot for number of special requests

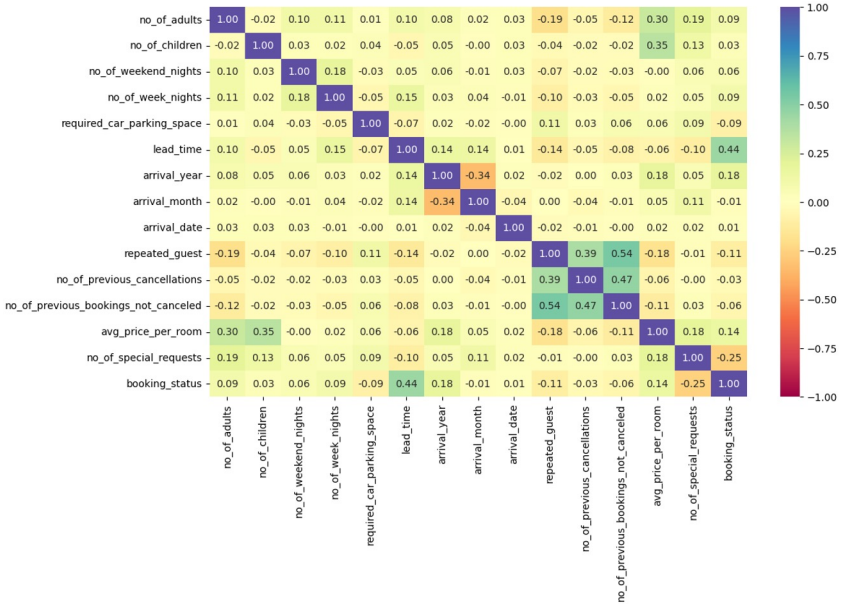
- `labeled_barplot(data,'booking_status', perc=True)` ## Complete the code to create labeled_barplot for booking status





EDA - Bivariate Analysis - correlation heatmap

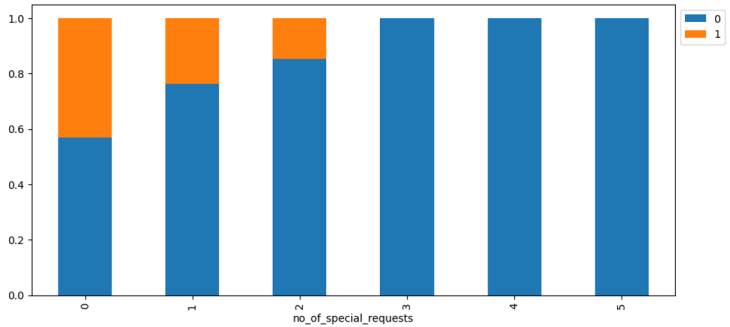
```
cols_list = data.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(12, 7))
sns.heatmap(data[cols_list].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral")
plt.show()
```



Observation that variable ‘lead time’ has a positive correlation with the variable ‘booking status’

EDA - Bivariate Analysis - “no special requests”, “avg price rm”, “booking status”

```
stacked_barplot(data,'no_of_special_requests', 'booking_status') ## Complete the code to plot stacked barplot for no of special requests and booking status
```



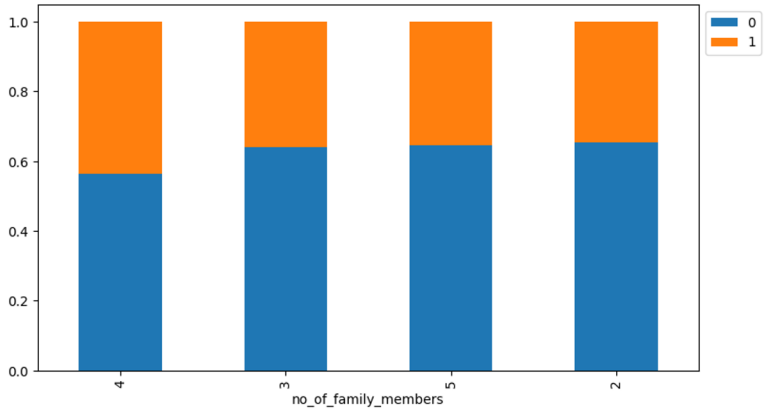
booking_status	0	1	All
no_of_special_requests			
All	24390	11885	36275
0	11232	8545	19777
1	8670	2703	11373
2	3727	637	4364
3	675	0	675
4	78	0	78
5	8	0	8

```
plt.figure(figsize=(10, 5))
sns.boxplot(data=data, x='no_of_special_requests', y='avg_price_per_room', palette='gist_rainbow') ## Complete the code to create boxplot for no of special requests and average price per room (excluding the outliers)
plt.show()
```

EDA - Bivariate Analysis - “lead time”, “no family members”, “booking status”

```
distribution_plot_wrt_target(data,'lead_time', 'booking_status') ## Complete the code to find distribution of lead time wrt booking status
```

```
stacked_barplot(family_data, 'no_of_family_members', 'booking_status') ## Complete the code to plot stacked barplot for no of family members and booking status
```

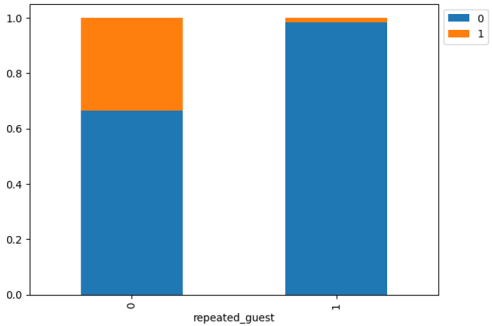


booking_status	0	1	All
no_of_family_members			
All	18456	9985	28441
2	15506	8213	23719
3	2425	1368	3793
4	514	398	912
5	11	6	17

EDA - Bivariate Analysis - “total days”, “repeated guest” and “booking status”

```
stacked_barplot(stay_data, 'total_days','booking_status') ## Complete the code to plot stacked barplot for total days and booking status
```

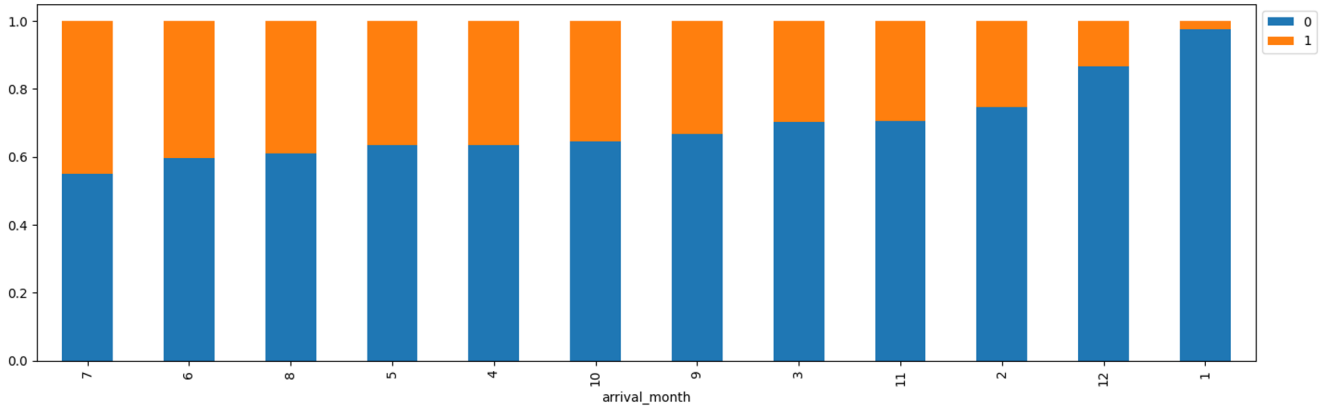
```
stacked_barplot(data, 'repeated_guest', 'booking_status') ## Complete the code to plot stacked barplot for repeated guests and booking status
```



booking_status	0	1	All
repeated_guest			
All	24390	11885	36275
0	23476	11869	35345
1	914	16	930

EDA - Bivariate Analysis - “arrival month” and “booking status”

```
stacked_barplot(data,'arrival_month', 'booking_status') ## Complete the code to plot stacked barplot for arrival month and booking status
```



booking_status	0	1	All
arrival_month			
All	24390	11885	36275
10	3437	1880	5317
9	3073	1538	4611
8	2325	1488	3813
7	1606	1314	2920
6	1912	1291	3203
4	1741	995	2736
5	1650	948	2598
11	2105	875	2980
3	1658	700	2358
2	1274	430	1704
12	2619	402	3021
1	990	24	1014

EDA - Bivariate Analysis - line plots “arrival month” “guests” “avg price p room”

```
# grouping the data on arrival months and extracting the count of bookings
monthly_data = data.groupby(["arrival_month"])["booking_status"].count()

# creating a dataframe with months and count of customers in each month
monthly_data = pd.DataFrame(
{"Month": list(monthly_data.index), "Guests": list(monthly_data.values)}
)

# plotting the trend over different months
plt.figure(figsize=(10, 5))
sns.lineplot(data=monthly_data, x="Month", y="Guests")
plt.show()

plt.figure(figsize=(10, 5))
sns.lineplot(data, x='arrival_month', y='avg_price_per_room') ## Complete the code to create lineplot
between average price per room and arrival month
plt.show()
```

Data Preprocessing - outlier detection using boxplot

```
# outlier detection using boxplot
numeric_columns = data.select_dtypes(include=np.number).columns.tolist()
# dropping booking_status numeric_columns.remove("booking_status")
plt.figure(figsize=(15, 12))

for i, variable in enumerate(numeric_columns):
plt.subplot(4, 4, i + 1)
plt.boxplot(data[variable], whis=1.5)
plt.tight_layout()
plt.title(variable)
plt.show()
```

Model Building - Logistic Regression

```
# fitting logistic regression model
logit = sm.Logit(y_train, X_train.astype(float))
lg = logit.fit(disp=False) ## Complete the code to fit logistic regression
print(lg.summary()) ## Complete the code to print summary of the model
```

```
                           Logit Regression Results                           
==============================================================================
Dep. Variable:         booking_status   No. Observations:                25392
Model:                          Logit   Df Residuals:                    25365
Method:                           MLE   Df Model:                           26
Date:                Sun, 28 Jan 2024   Pseudo R-squ.:                 0.3305
Time:                        00:14:58   Log-Likelihood:                -10753.
converged:                      False   LL-Null:                       -16060.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================================
                                   coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
no_of_adults                     0.0219      0.038      0.582      0.561      -0.052       0.096
no_of_children                   0.0794      0.063      1.255      0.209      -0.045       0.203
no_of_weekend_nights             0.1490      0.020      7.522      0.000       0.110       0.188
no_of_week_nights                0.0357      0.012      2.913      0.004       0.012       0.060
required_car_parking_space      -1.6388      0.137    -11.972      0.000      -1.907      -1.371
lead_time                        0.0163      0.000     63.004      0.000       0.016       0.017
arrival_year                    -0.0013      0.000     -9.762      0.000      -0.002      -0.001
arrival_month                   -0.0669      0.006    -11.128      0.000      -0.079      -0.055
arrival_date                     0.0030      0.002      1.555      0.120      -0.001       0.007
repeated_guest                  -2.0249      0.742     -2.727      0.006      -3.480      -0.570
no_of_previous_cancellations     0.3526      0.102      3.456      0.001       0.153       0.552
no_of_previous_bookings_not_canceled -1.3048  0.880     -1.483      0.138      -3.029       0.419
avg_price_per_room               0.0198      0.001     27.513      0.000       0.018       0.021
no_of_special_requests          -1.4763      0.030    -48.812      0.000      -1.536      -1.417
type_of_meal_plan_Meal Plan 2    0.0512      0.065      0.793      0.428      -0.075       0.178
type_of_meal_plan_Meal Plan 3   10.7679    180.757      0.060      0.952    -343.510     365.045
type_of_meal_plan_Not Selected   0.2636      0.053      5.009      0.000       0.160       0.367
room_type_reserved_Room_Type 2  -0.4178      0.133     -3.137      0.002      -0.679      -0.157
room_type_reserved_Room_Type 3   1.1052      1.797      0.615      0.539      -2.417       4.628
room_type_reserved_Room_Type 4  -0.2483      0.053     -4.663      0.000      -0.353      -0.144
room_type_reserved_Room_Type 5  -0.6408      0.215     -2.975      0.003      -1.063      -0.219
room_type_reserved_Room_Type 6  -0.9110      0.155     -5.896      0.000      -1.214      -0.608
room_type_reserved_Room_Type 7  -1.4095      0.299     -4.714      0.000      -1.996      -0.823
market_segment_type_Complementary -22.9396 7618.070     -0.003      0.998    -1.5e+04    1.49e+04
market_segment_type_Corporate   -0.9608      0.276     -3.480      0.001      -1.502      -0.420
market_segment_type_Offline     -1.8906      0.264     -7.160      0.000      -2.408      -1.373
market_segment_type_Online      -0.0896      0.261     -0.343      0.732      -0.602       0.422
================================================================================================
```

- We observe many coefficients of the features listed are negatively correlated.

Model Building - Logistic Regression

```
checking_vif(X_train)
```

	feature	VIF
0	const	39497686.20788
1	no_of_adults	1.35113
2	no_of_children	2.09358
3	no_of_weekend_nights	1.06948
4	no_of_week_nights	1.09571
5	required_car_parking_space	1.03997
6	lead_time	1.39517
7	arrival_year	1.43190
8	arrival_month	1.27633
9	arrival_date	1.00679
10	repeated_guest	1.78358
11	no_of_previous_cancellations	1.39569
12	no_of_previous_bookings_not_canceled	1.65200
13	avg_price_per_room	2.06860
14	no_of_special_requests	1.24798

- The model is then checked for multicollinearity with the VIF test.
- We observe there is no multicollinearity present in the data, as the VIF scores for all the features are well below 5.
- We can ignore the VIF value for the constant and the dummy (categorical) features. The latter are not included in the screenshot, as they are not relevant.

Model Building - Logistic Regression

```
logit1 = sm.Logit(y_train, X_train1.astype(float)) ## Complete the code to train logistic regression on X_train1 and y_train
lg1 = logit1.fit(disp=False) ## Complete the code to fit logistic regression
print(lg1.summary()) ## Complete the code to print summary of the model
```

Logit Regression Results

Dep. Variable:	booking_status	No. Observations:	25392
Model:	Logit	Df Residuals:	25370
Method:	MLE	Df Model:	21
Date:	Sat, 27 Jan 2024	Pseudo R-squ.:	0.3282
Time:	22:13:07	Log-Likelihood:	-10810.
converged:	True	LL-Null:	-16091.
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	-915.6391	120.471	-7.600	0.000	-1151.758	-679.520
no_of_adults	0.1088	0.037	2.914	0.004	0.036	0.182
no_of_children	0.1531	0.062	2.470	0.014	0.032	0.275
no_of_weekend_nights	0.1086	0.020	5.498	0.000	0.070	0.147
no_of_week_nights	0.0417	0.012	3.399	0.001	0.018	0.066
required_car_parking_space	-1.5947	0.138	-11.564	0.000	-1.865	-1.324
lead_time	0.0157	0.000	59.213	0.000	0.015	0.016
arrival_year	0.4523	0.060	7.576	0.000	0.335	0.569
arrival_month	-0.0425	0.006	-6.591	0.000	-0.055	-0.030
repeated_guest	-2.7367	0.557	-4.916	0.000	-3.828	-1.646
no_of_previous_cancellations	0.2288	0.077	2.983	0.003	0.078	0.379
avg_price_per_room	0.0192	0.001	26.336	0.000	0.018	0.021
no_of_special_requests	-1.4698	0.030	-48.884	0.000	-1.529	-1.411
type_of_meal_plan_Meal Plan 2	0.1642	0.067	2.469	0.014	0.034	0.295
type_of_meal_plan_Not Selected	0.2860	0.053	5.406	0.000	0.182	0.390
room_type_reserved_Room_Type 2	-0.3552	0.131	-2.709	0.007	-0.612	-0.098
room_type_reserved_Room_Type 4	-0.2828	0.053	-5.330	0.000	-0.387	-0.179
room_type_reserved_Room_Type 5	-0.7364	0.208	-3.535	0.000	-1.145	-0.328
room_type_reserved_Room_Type 6	-0.9682	0.151	-6.403	0.000	-1.265	-0.672
room_type_reserved_Room_Type 7	-1.4343	0.293	-4.892	0.000	-2.009	-0.860
market_segment_type_Corporate	-0.7913	0.103	-7.692	0.000	-0.993	-0.590
market_segment_type_Offline	-1.7854	0.052	-34.363	0.000	-1.887	-1.684

- Next, we check for features that are statistically insignificant as indicated by their high P-values. We set the P-value threshold at > 0.05, and features with higher P-values than this threshold are removed from the updated model.

Model Building - Logistic Regression - converting coefficients to odds

	const	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	required_car_parking_space	lead_time	arrival_year	arrival_month	repeated_guest
Odds	0.00000	1.11491	1.16546	1.11470	1.04258	0.20296	1.01583	1.57195	0.95839	0.06478
Change_odd%	-100.00000	11.49096	16.54593	11.46966	4.25841	-79.70395	1.58331	57.19508	-4.16120	-93.52180

- Above table is incomplete as I couldn't capture all the features in a screenshot.
- The results based on coefficients and odds can be interpreted as follows:

  Holding all other features constant, a unit change in coefficient "no_of_adults" will increase the odds of a cancellation by 1.11491 times, or a 11.49% increase in odds of having a cancellation.

```
print("Training performance:")
log_reg_model_train_perf = model_performance_classification_statsmodels(lg1
Complete the code to check performance on X_train1 and y_train
log_reg_model_train_perf
```

Training performance:

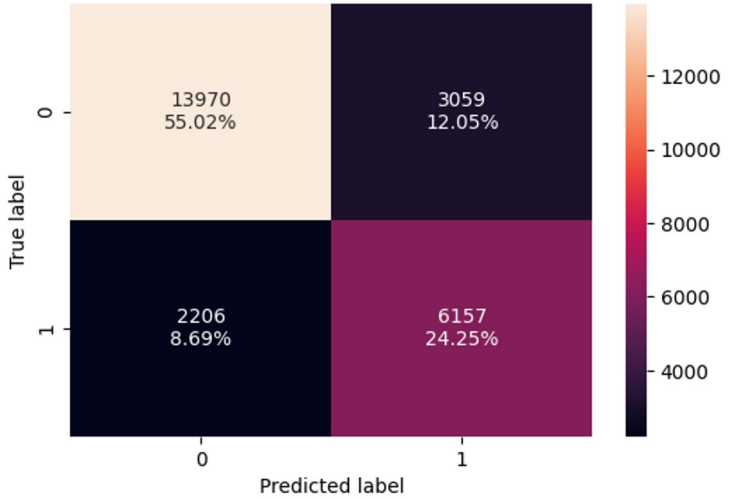
	Accuracy	Recall	Precision	F1
0	0.80545	0.63267	0.73907	0.68174

Model Performance Evaluation and Improvement - Logistic Regression AUC-ROC curve

- Let's see if the F1 score can be improved further, by changing the model threshold using AUC-ROC Curve.

```
# Optimal threshold as per AUC-ROC curve
# The optimal cut off would be where true positive rate is high and false positive rate is low
fpr, tpr, thresholds = roc_curve(y_train, lg1.predict(X_train1))

# creating confusion matrix
confusion_matrix_statsmodels(lg1, X_train1, y_train, threshold=optimal_threshold_auc_roc )
## Complete the code to create the confusion matrix for X_train1 and y_train with optimal_threshold_auc_roc as threshold
```



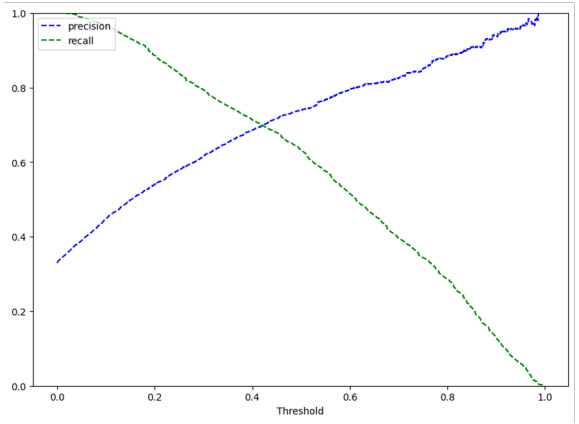
Training performance:

	Accuracy	Recall	Precision	F1
0	0.79265	0.73622	0.66808	0.70049

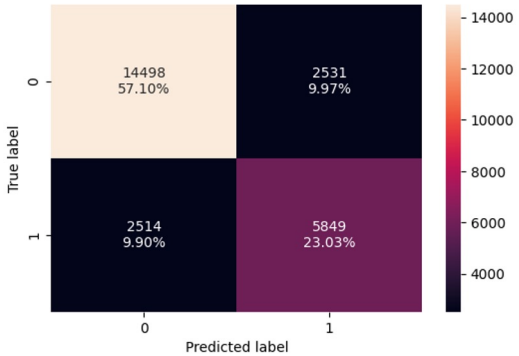
Model Performance Evaluation and Improvement - Logistic Regression

Precision-Recall Curve

- Let's use Precision-Recall curve and see if we can find a better threshold




```
# creating confusion matrix
confusion_matrix_statsmodels(lg1, X_train1, y_train,
threshold=optimal_threshold_curve)
## Complete the code to create the confusion matrix for
X_train1 and y_train with optimal_threshold_curve as
threshold
```



Training performance:

	Accuracy	Recall	Precision	F1
0	0.80132	0.69939	0.69797	0.69868

Model Performance Evaluation and Improvement - code for logistic regression model test performance comparison

```
# test performance comparison
models_test_comp_df = pd.concat(
[
log_reg_model_test_perf.T,
log_reg_model_test_perf_threshold_auc_roc.T,
log_reg_model_test_perf_threshold_curve.T,
],axis=1,)
models_test_comp_df.columns = [
"Logistic Regression-default Threshold",
"Logistic Regression-0.37 Threshold",
"Logistic Regression-0.42 Threshold",
]

print("Testing performance comparison:")
models_test_comp_df ## Complete the code to compare test performance
```

Model Building - Decision Tree - mention the model building steps

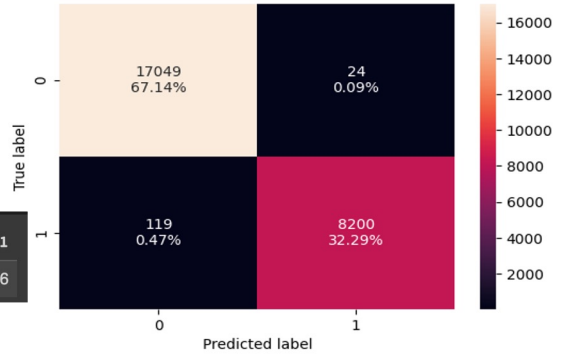
Step 1: Fit the decision tree on train data

```
model = DecisionTreeClassifier(random_state=1)
model.fit(X_train, y_train)
## Complete the code to fit decision tree on train data
```

Step 2: Check model performance on training set

```
confusion_matrix_sklearn(model, X_train, y_train)
## Complete the code to create confusion matrix for train data
```

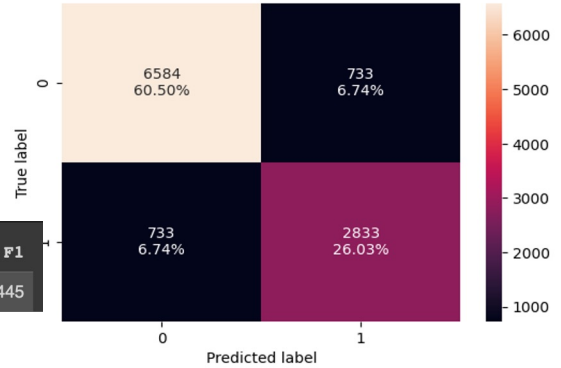
	Accuracy	Recall	Precision	F1
0	0.99437	0.98570	0.99708	0.99136



Step 3: Check model performance on test set

```
confusion_matrix_sklearn(model, X_test, y_test)
## Complete the code to create confusion matrix for test data
decision_tree_perf_test = model_performance_classification_sklearn
(model, X_test, y_test)
## Complete the code to check performance on test
decision_tree_perf_test
```

	Accuracy	Recall	Precision	F1
0	0.86529	0.79445	0.79445	0.79445



Step 4: Model evaluation: Discrepancy between train and test performance scores

Decision tree overfits and needs to pruned.

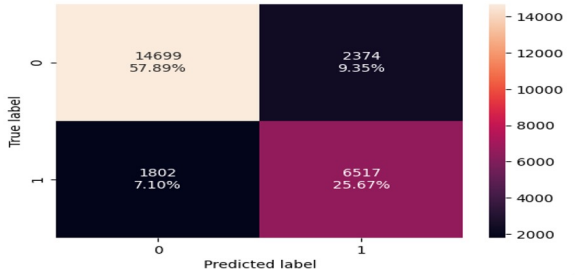
Model Performance Evaluation and Improvement - Decision Tree Pre-Pruning

- We will first try to improve the model performance by trying the pre-pruning technique.

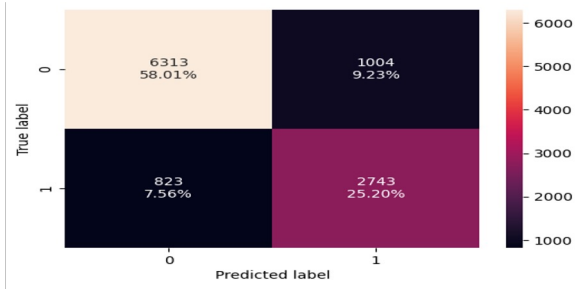
  See the # green text for details on **'hyper parameter tuning' using Grid Search:**

```
# Choose the type of classifier.
estimator = DecisionTreeClassifier(random_state=1, class_weight="balanced")
# Grid of parameters to choose from
parameters = {"max_depth": np.arange(2, 7, 2),"max_leaf_nodes": [50, 75, 150, 250],"min_samples_split":
[10, 30, 50, 70],}
# Type of scoring used to compare parameter combinations
acc_scorer = make_scorer(f1_score)
# Run the grid search
grid_obj = GridSearchCV(estimator, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)
# Set the clf to the best combination of parameters
estimator = grid_obj.best_estimator_
# Fit the best algorithm to the data.
estimator.fit(X_train, y_train)
```

Model Performance Evaluation and Improvement - Decision Tree pre-pruning



	Accuracy	Recall	Precision	F1
0	0.83554	0.78339	0.73299	0.75735



	Accuracy	Recall	Precision	F1
0	0.83212	0.76921	0.73205	0.75017

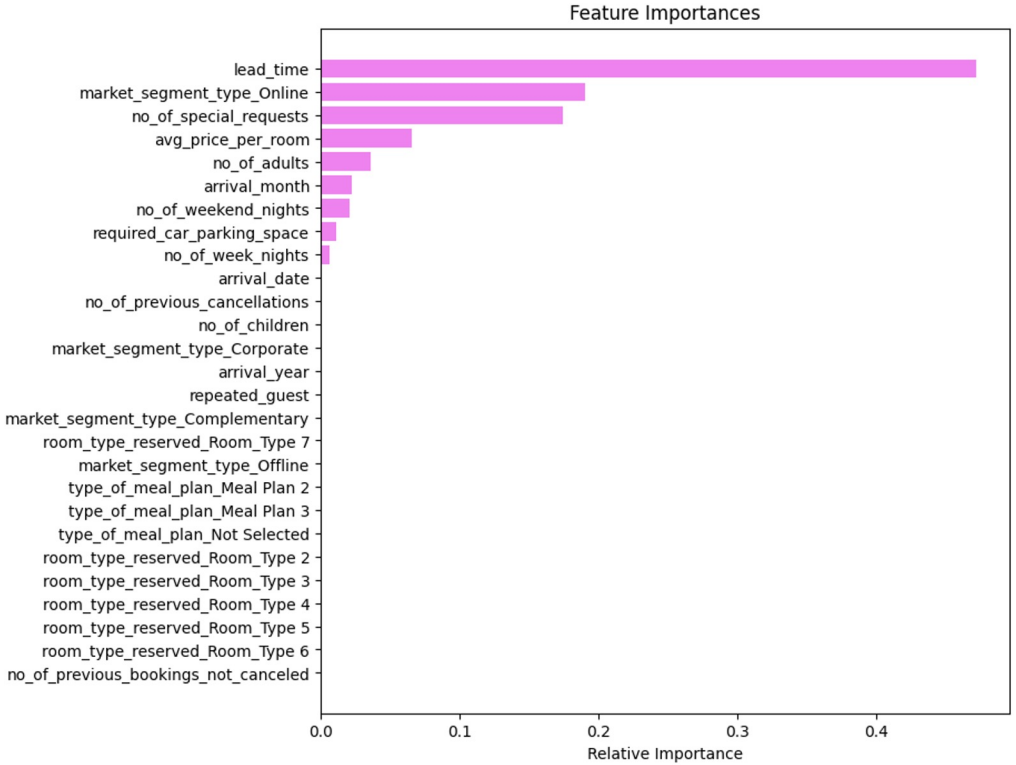
Next, we will be checking the performance on training and test set

- `confusion_matrix_sklearn(estimator, X_train, y_train)`
  `## Complete the code to create confusion matrix for train data`
- `decision_tree_tune_perf_train = model_performance_classification_sklearn(estimator, X_train, y_train) ## Complete the code to check performance on train set`
- `Decision_tree_tune_perf_train`

- `confusion_matrix_sklearn(estimator, X_test, y_test) ## Complete the code to create confusion matrix for test data`
- `decision_tree_tune_perf_test = model_performance_classification_sklearn(estimator, X_test, y_test) ## Complete the code to check performance on test set`
- `decision_tree_tune_perf_test`

We observe that after pre-pruning, the train and test data performance are much better in line, so less overfitting of the data.

Model Performance Evaluation and Improvement - Decision Tree pre-pruning



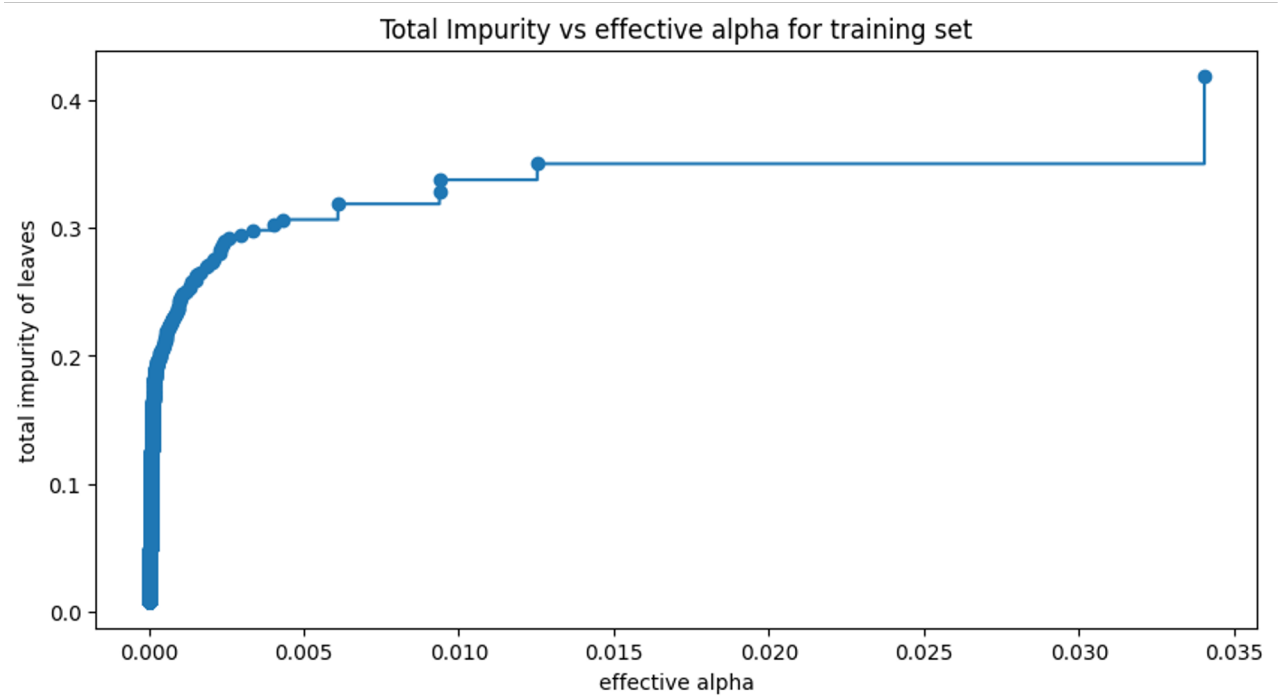
- Observe the length of the pink bars to determine which Features are of greater or less importance in the pre-pruned Decision Tree.

Model Performance Evaluation and Improvement - Decision Tree post-pruning

Now we will try to improve the performance of the decision tree by using a post pruning technique called **Cost Complexity pruning.**

	ccp_alphas	impurities
0	0.00000	0.00833
1	-0.00000	0.00833
2	0.00000	0.00833
3	0.00000	0.00833
4	0.00000	0.00833
...	...	...
1648	0.00938	0.32791
1649	0.00941	0.33732
1650	0.01253	0.34985
1651	0.03405	0.41794
1652	0.08206	0.50000

1653 rows × 2 columns



Model Performance Evaluation and Improvement - Decision Tree post-pruning

Great Learning POWER AHEAD

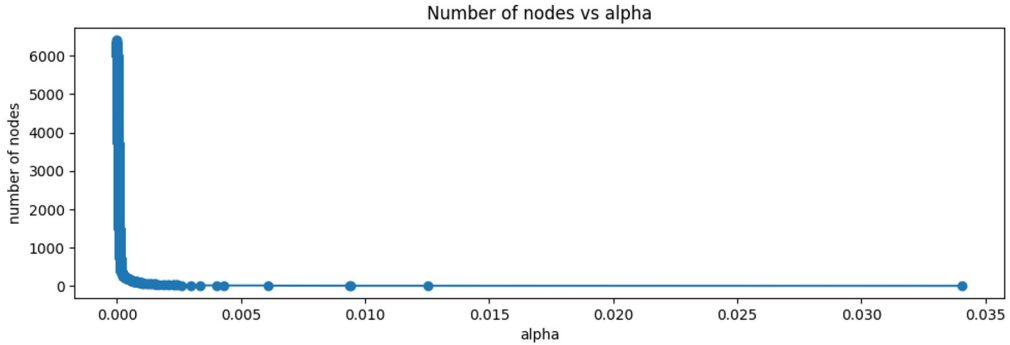
Next, we train a decision tree using effective alphas.

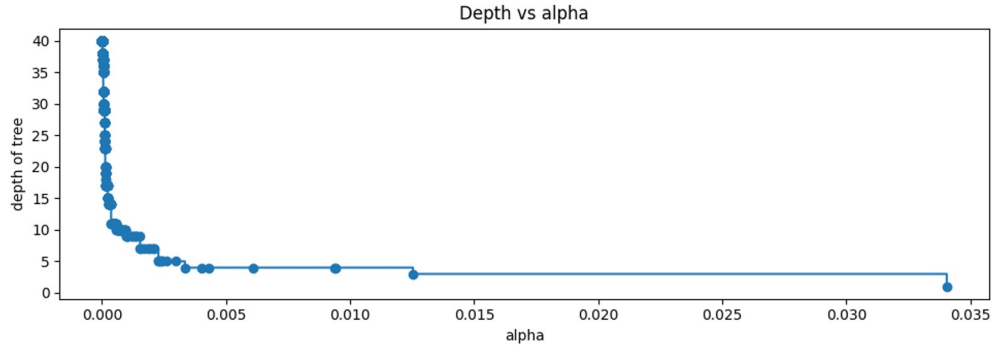
The last value in ccp_alphas is the alpha value that prunes the whole tree, leaving the tree, clfs[-1], with one node.

```
%%time
clfs = []
for ccp_alpha in ccp_alphas:
clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha, class_weight="balanced")
clf.fit(X_train, y_train) ## Complete the code to fit decision tree on training data
clfs.append(clf)
print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

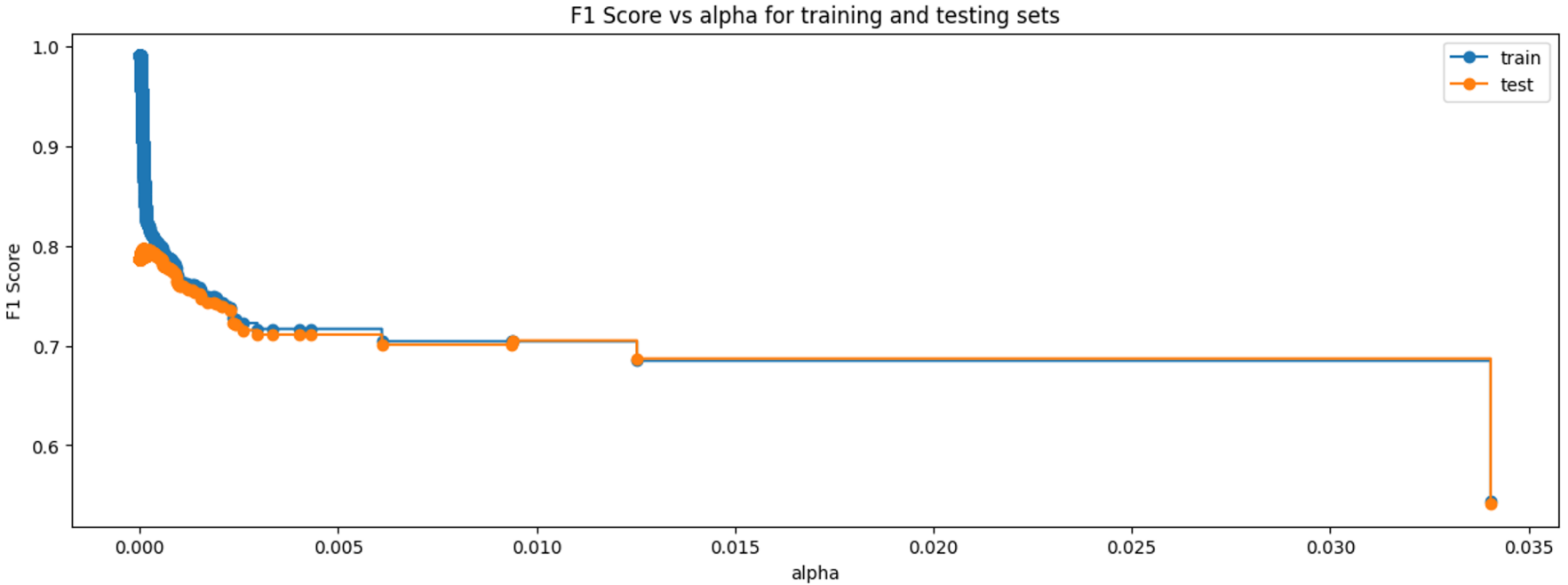
```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.08205938055988293
CPU times: user 4min 47s, sys: 0 ns, total: 4min 47s
Wall time: 4min 49s
```

Model Performance Evaluation and Improvement - Decision Tree post-pruning

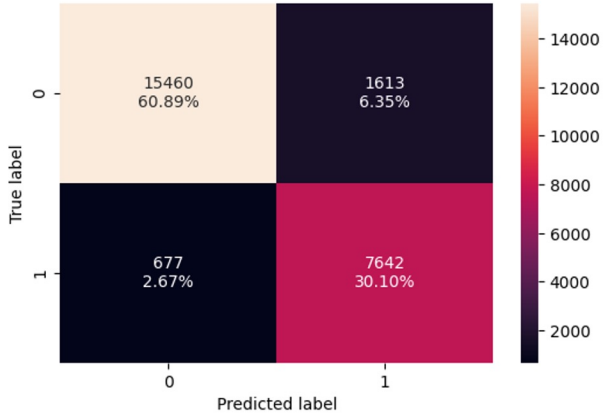




Model Performance Evaluation and Improvement - Decision Tree post-pruning



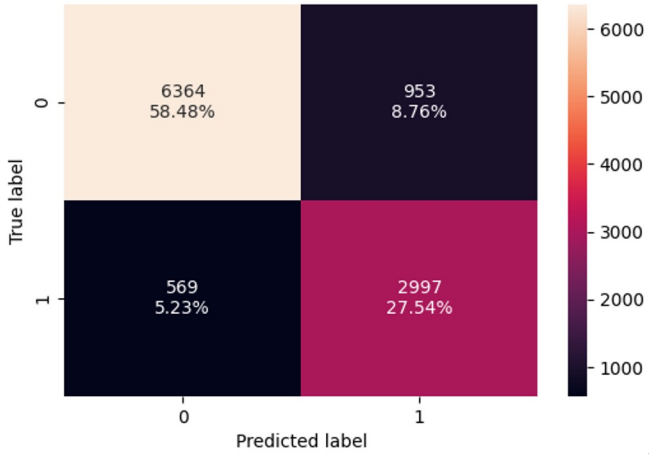
Model Performance Evaluation and Improvement - Decision Tree post-pruning



	Accuracy	Recall	Precision	F1
0	0.90981	0.91862	0.82572	0.86969

Checking performance on training set

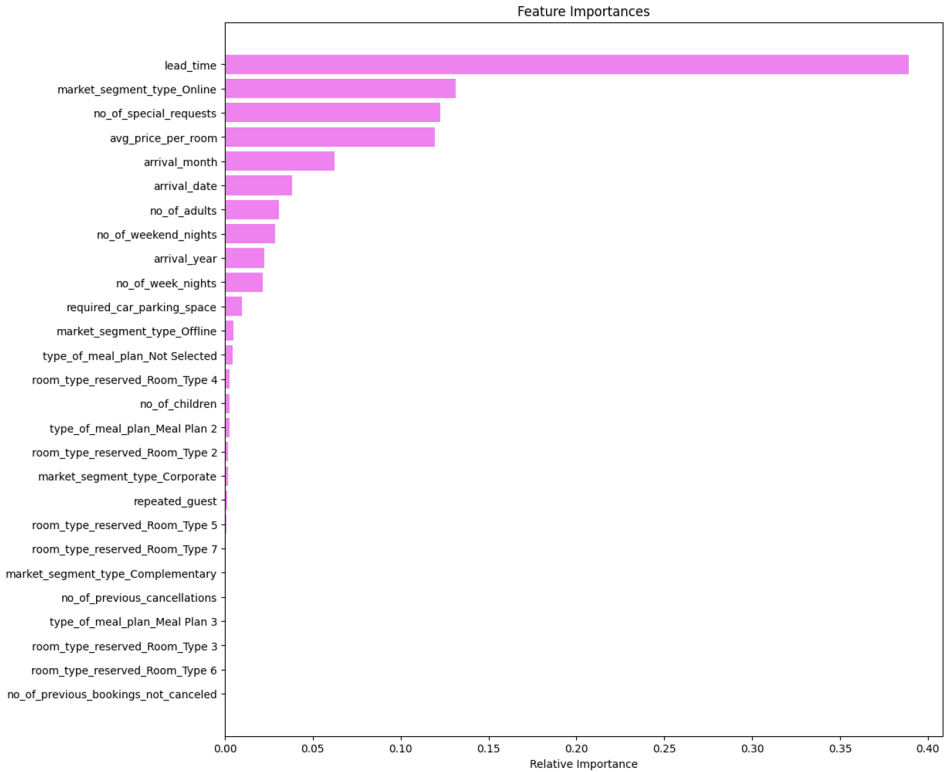
```
confusion_matrix_sklearn(best_model
, X_test, y_test)
## Complete the code to create
confusion matrix for test data on
best model
```




```
decision_tree_post_test =
model_performance_classific
ation_sklearn(best_model,
X_test, y_test)
## Complete the code to
check performance of test
set on best model
decision_tree_post_test
```

	Accuracy	Recall	Precision	F1
0	0.86015	0.84044	0.75873	0.79750

Model Performance Evaluation and Improvement - Decision Tree post-pruning



- We can observe that the most important Feature of Importance in this post pruned Decision Tree model, which delivers the highest F1 score relevant to our client, is by far the feature: “Lead Time”

- As we also saw in the bivariate analysis, the longer the lead time of a booking, the greater the probability that the booking gets cancelled.

- Other features of importance are to a lesser extent: Online market segment type, no of special requests, average price per room, arrival month.

Slide Header

- Please add any other pointers (if needed)

Great Learning

Happy Learning !