INN Hotels Project

Context

A significant number of hotel bookings are called off due to cancellations or no-shows. The typical reasons for cancellations include change of plans, scheduling conflicts, etc. This is often made easier by the option to do so free of charge or preferably at a low cost which is beneficial to hotel guests but it is a less desirable and possibly revenue-diminishing factor for hotels to deal with. Such losses are particularly high on last-minute cancellations.

The new technologies involving online booking channels have dramatically changed customers' booking possibilities and behavior. This adds a further dimension to the challenge of how hotels handle cancellations, which are no longer limited to traditional booking and guest characteristics.

The cancellation of bookings impact a hotel on various fronts:

1. Loss of resources (revenue) when the hotel cannot resell the room.
2. Additional costs of distribution channels by increasing commissions or paying for publicity to help sell these rooms.
3. Lowering prices last minute, so the hotel can resell a room, resulting in reducing the profit margin.
4. Human resources to make arrangements for the guests.

Objective

The increasing number of cancellations calls for a Machine Learning based solution that can help in predicting which booking is likely to be canceled. INN Hotels Group has a chain of hotels in Portugal, they are facing problems with the high number of booking cancellations and have reached out to your firm for data-driven solutions. You as a data scientist have to analyze the data provided to find which factors have a high influence on booking cancellations, build a predictive model that can predict which booking is going to be canceled in advance, and help in formulating profitable policies for cancellations and refunds.

Data Description

The data contains the different attributes of customers' booking details. The detailed data dictionary is given below.

Data Dictionary

- Booking_ID: unique identifier of each booking
- no_of_adults: Number of adults
- no_of_children: Number of Children
- no_of_weekend_nights: Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
- no_of_week_nights: Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel
- type_of_meal_plan: Type of meal plan booked by the customer:
    - Not Selected – No meal plan selected

    - Meal Plan 1 – Breakfast
    - Meal Plan 2 – Half board (breakfast and one other meal)
    - Meal Plan 3 – Full board (breakfast, lunch, and dinner)
- required_car_parking_space: Does the customer require a car parking space? (0 - No, 1- Yes)
- room_type_reserved: Type of room reserved by the customer. The values are ciphered (encoded) by INN Hotels.
- lead_time: Number of days between the date of booking and the arrival date
- arrival_year: Year of arrival date
- arrival_month: Month of arrival date
- arrival_date: Date of the month
- market_segment_type: Market segment designation.
- repeated_guest: Is the customer a repeated guest? (0 - No, 1- Yes)
- no_of_previous_cancellations: Number of previous bookings that were canceled by the customer prior to the current booking
- no_of_previous_bookings_not_canceled: Number of previous bookings not canceled by the customer prior to the current booking
- avg_price_per_room: Average price per day of the reservation; prices of the rooms are dynamic. (in euros)
- no_of_special_requests: Total number of special requests made by the customer (e.g. high floor, view from the room, etc)
- booking_status: Flag indicating if the booking was canceled or not.

Importing necessary libraries and data

In [1]:
```
# suppress all warnings
import warnings
warnings.filterwarnings("ignore")

from statsmodels.tools.sm_exceptions import ConvergenceWarning
warnings.simplefilter("ignore", ConvergenceWarning)

#import libraries needed for data manipulation
import pandas as pd
import numpy as np

#display floating numbers with 5 decimal places
pd.set_option('display.float_format', lambda x: '%.5f' % x)

# unlimited number of displayed columns, limit of 200 for displayed rows
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 200)

#import libraries needed for data visualization

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# split the data into random train and test subsets
from sklearn.model_selection import train_test_split
```

```
# using statsmodels to build our model
import statsmodels.stats.api as sms
import statsmodels.api as sm

# to compute VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

# to compute and display decision trees
from statsmodels.tools.tools import add_constant
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# tuning different models
from sklearn.model_selection import GridSearchCV


# for different sklearn metric scores
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    plot_confusion_matrix,
    precision_recall_curve,
    roc_curve,
    make_scorer,
)
```

Data Overview

- Observations -Sanity Checks

In [2]:

```
#import dataset named 'INNHotelsGroup.csv'

hotels = pd.read_csv('INNHotelsGroup.csv')

# read first five rows of the dataset

hotels.head()
```

Out[2]:

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	le
0	INN00001	2	0	1	2	Meal Plan 1	0	Room_Type 1	
1	INN00002	2	0	2	3	Not Selected	0	Room_Type 1	
2	INN00003	1	0	2	1	Meal Plan 1	0	Room_Type 1	
3	INN00004	2	0	0	2	Meal Plan 1	0	Room_Type 1	

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	le
4	INN00005	2	0	1	1	Not Selected	0	Room_Type 1	

In [3]:
```
hotels.shape
```

Out[3]:
```
(36275, 19)
```

In [4]:
```
hotels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 19 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Booking_ID                            36275 non-null  object 
 1   no_of_adults                          36275 non-null  int64  
 2   no_of_children                        36275 non-null  int64  
 3   no_of_weekend_nights                  36275 non-null  int64  
 4   no_of_week_nights                     36275 non-null  int64  
 5   type_of_meal_plan                     36275 non-null  object 
 6   required_car_parking_space            36275 non-null  int64  
 7   room_type_reserved                    36275 non-null  object 
 8   lead_time                             36275 non-null  int64  
 9   arrival_year                          36275 non-null  int64  
 10  arrival_month                         36275 non-null  int64  
 11  arrival_date                          36275 non-null  int64  
 12  market_segment_type                   36275 non-null  object 
 13  repeated_guest                        36275 non-null  int64  
 14  no_of_previous_cancellations          36275 non-null  int64  
 15  no_of_previous_bookings_not_canceled  36275 non-null  int64  
 16  avg_price_per_room                    36275 non-null  float64
 17  no_of_special_requests                36275 non-null  int64  
 18  booking_status                        36275 non-null  object 
dtypes: float64(1), int64(13), object(5)
memory usage: 5.3+ MB
```

Observations

- There are 36,274 rows and 19 columns.
- `Booking_ID`, `type_of_meal_plan`, `room_type_reserved`, `market_segment_type`, and `booking_status` are object type, while the rest are numeric in nature.
  - Booking_ID is just an identifier for each hotel guest.

In [5]:
```
hotels.isnull().sum()
```

Out[5]:
```
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
```

In [6]:

```
hotels.duplicated().sum()
```

Out[6]: 0

Observations

- There are no missing values.
- There are no duplicated values.

In [7]:

```
# create a copy of the data so that the original dataset is not changed.

df = hotels.copy()
```

In [8]:

```
# drop Booking ID variable, since it is just an identifier

df.drop(columns=['Booking_ID'], inplace=True)
```

Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A thorough analysis of the data, in addition to the questions completed below, will help to approach the analysis in the right manner and generate insights from the data.

In [9]:

```
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max

	count	mean	std	min	25%	50%	75%	max
no_of_adults	36275.00000	1.84496	0.51871	0.00000	2.00000	2.00000	2.00000	4.00000
no_of_children	36275.00000	0.10528	0.40265	0.00000	0.00000	0.00000	0.00000	10.00000
no_of_weekend_nights	36275.00000	0.81072	0.87064	0.00000	0.00000	1.00000	2.00000	7.00000
no_of_week_nights	36275.00000	2.20430	1.41090	0.00000	1.00000	2.00000	3.00000	17.00000
required_car_parking_space	36275.00000	0.03099	0.17328	0.00000	0.00000	0.00000	0.00000	1.00000
lead_time	36275.00000	85.23256	85.93082	0.00000	17.00000	57.00000	126.00000	443.00000
arrival_year	36275.00000	2017.82043	0.38384	2017.00000	2018.00000	2018.00000	2018.00000	2018.00000
arrival_month	36275.00000	7.42365	3.06989	1.00000	5.00000	8.00000	10.00000	12.00000
arrival_date	36275.00000	15.59700	8.74045	1.00000	8.00000	16.00000	23.00000	31.00000
repeated_guest	36275.00000	0.02564	0.15805	0.00000	0.00000	0.00000	0.00000	1.00000
no_of_previous_cancellations	36275.00000	0.02335	0.36833	0.00000	0.00000	0.00000	0.00000	13.00000
no_of_previous_bookings_not_canceled	36275.00000	0.15341	1.75417	0.00000	0.00000	0.00000	0.00000	58.00000
avg_price_per_room	36275.00000	103.42354	35.08942	0.00000	80.30000	99.45000	120.00000	540.00000
no_of_special_requests	36275.00000	0.61966	0.78624	0.00000	0.00000	0.00000	1.00000	5.00000

Observations:

- The number of adults ranges from 0 to 4, number of children from 0 to 10. Children maximum seems high, might require a check.
- Range on weekends and weeknights seems reasonable, though 7 weekends might also require a check.
- At least 75% of hotel guests do not require a parking space.
- On average, guests book 85 days in advance. Between 75th percentile and max is a large difference, suggests many outliers.
- We have two years of data, 2017 and 2018 (latter consists of more information).
- At least 75% of guests are not repeating customers.
- On average, the price per room is 103 euros. Between 75th percentile and max is a large difference, suggests many outliers.

Leading Questions:

1. What are the busiest months in the hotel?
2. Which market segment do most of the guests come from?
3. Hotel rates are dynamic and change according to demand and customer demographics. What are the differences in room prices in different market segments?
4. What percentage of bookings are canceled?
5. Repeating guests are the guests who stay in the hotel often and are important to brand equity. What percentage of repeating guests cancel?
6. Many guests have special requirements when booking a hotel room. Do these requirements affect booking cancellation?

In [10]:

```
# define a function to plot a boxplot and a histogram along the same scale
```

```python
def histbox(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined
    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (box, hist) = plt.subplots(
        nrows=2,                                            # Number of rows of the subplot grid = 2
                                                                # boxplot first then histogram created below
        sharex=True,                                        # x-axis same among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},        # boxplot 1/3 height of histogram
        figsize=figsize,                                    # figsize defined above as (12, 7)
    )
    # defining boxplot inside function, so when using it say histbox(df, 'cost'), df: data and cost: feature

    sns.boxplot(
        data=data, x=feature, ax=box, showmeans=True, color="chocolate"
    )  # showmeans makes mean val on boxplot have star, ax =
    sns.histplot(
        data=data, x=feature, kde=kde, ax=hist, bins=bins, color = "darkgreen"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=hist, color = "darkgreen"
    )  # For histogram if there are bins in potential graph

    # add vertical line in histogram for mean and median
    hist.axvline(
        data[feature].mean(), color="purple", linestyle="--"
    )  # Add mean to the histogram
    hist.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [11]:

```python
# define a function to create labeled barplots

def bar(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
```

```
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage
plt.show()  # show the plot
```

Question 1: What are the busiest months in the hotel?

In [12]:

```
df['arrival_month'].value_counts()[:6]
```

Out[12]:

```
10    5317
9     4611
8     3813
6     3203
12    3021
11    2980
Name: arrival_month, dtype: int64
```

Observations:

- The top 6 busiest months for the hotel are all in the last half of the year:
  - October, then September, August, June, December, and November.

Question 2: Which market segment do most of the guests come from?

In [13]:
```
df.head()
```

Out[13]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	lead_time	arri
0	2	0	1	2	Meal Plan 1	0	Room_Type 1	224	
1	2	0	2	3	Not Selected	0	Room_Type 1	5	
2	1	0	2	1	Meal Plan 1	0	Room_Type 1	1	
3	2	0	0	2	Meal Plan 1	0	Room_Type 1	211	
4	2	0	1	1	Not Selected	0	Room_Type 1	48	

In [14]:
```
print("Most guests come from the", df['market_segment_type'].max(), "market segment.")
```

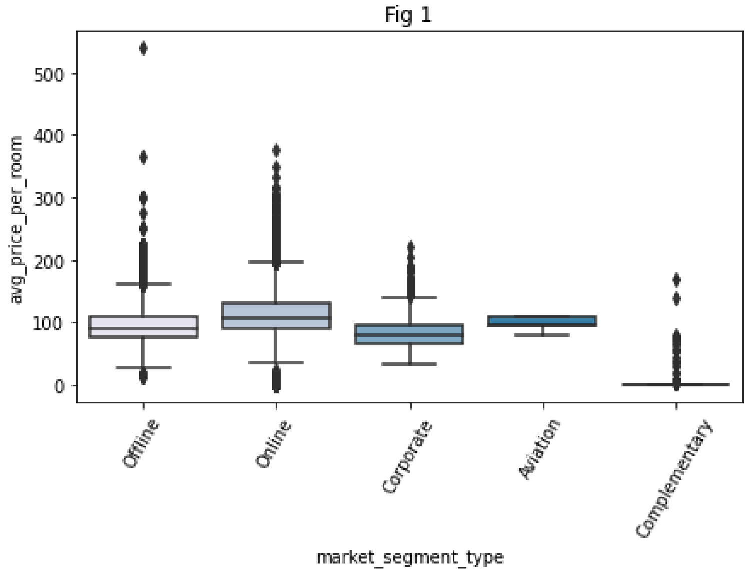
```
Most guests come from the Online market segment.
```

Question 3: Hotel rates are dynamic and change according to demand and customer demographics. What are the differences in room prices in different market segments?

In [15]:
```
df.groupby("market_segment_type")["avg_price_per_room"].mean()
```

Out[15]:
```
market_segment_type
Aviation         100.70400
Complementary      3.14176
Corporate         82.91174
Offline           91.63268
Online           112.25685
Name: avg_price_per_room, dtype: float64
```

In [16]:
```
plt.figure(figsize=(7,4))
plt.title("Fig 1")
sns.boxplot(x = "market_segment_type", y = "avg_price_per_room", data = df, palette = 'PuBu')
plt.xticks(rotation = 60)
plt.show()
```



Observations

- Rooms booked online have high variations in prices.
- The offline and corporate room prices are almost similar.
- Complementary market segment gets the rooms at very low prices, which makes sense.

Question 4: What percentage of bookings are canceled?

In [17]:
```
print ('The percentage of bookings that are cancelled is',
       round(df[df['booking_status'] == "Canceled"].shape[0] / df.shape[0] * 100, 2),'%')
```

```
The percentage of bookings that are cancelled is 32.76 %
```

Question 5: Repeating guests are the guests who stay in the hotel often and are important to brand equity. What percentage of repeating guests cancel?

In [18]:
```
repeat = df[df["repeated_guest"] == 1]

repeat.shape[0]
```

Out[18]:
```
930
```

In [19]:
```
print ('The percentage of bookings that are cancelled by repeating guests is',
       round(repeat[repeat['booking_status'] == "Canceled"].shape[0] / repeat.shape[0] * 100, 2),'%')
```

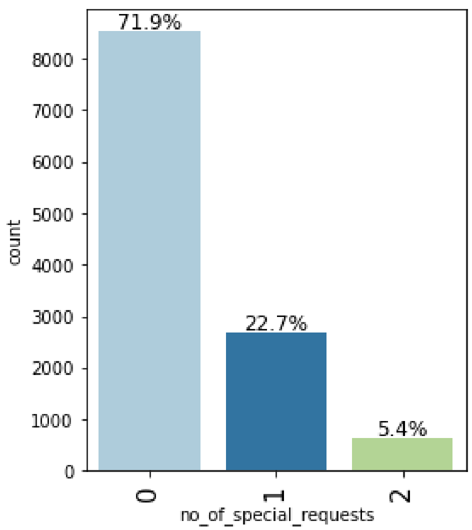
```
The percentage of bookings that are cancelled by repeating guests is 1.72 %
```

Question 6: Many guests have special requirements when booking a hotel room. Do these requirements affect booking cancellation?

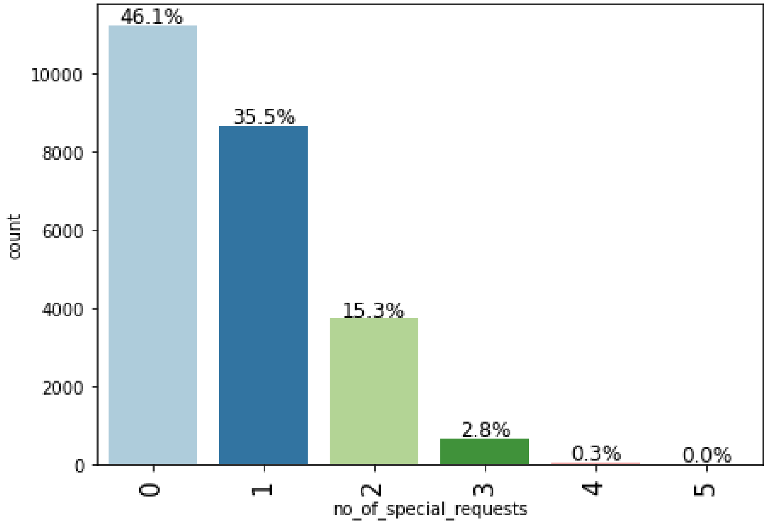
In [20]:
```
df.groupby("no_of_special_requests")["booking_status"].value_counts()
```

Out[20]:
```
no_of_special_requests  booking_status
0                       Not_Canceled      11232
                        Canceled           8545
1                       Not_Canceled       8670
                        Canceled           2703
2                       Not_Canceled       3727
                        Canceled            637
3                       Not_Canceled        675
4                       Not_Canceled         78
5                       Not_Canceled          8
Name: booking_status, dtype: int64
```

In [21]:
```
canceled = df[df['booking_status'] == "Canceled"]
kept = df[df['booking_status'] == "Not_Canceled"]

bar(canceled, 'no_of_special_requests', perc=True)
```



In [22]:
```
bar(kept, 'no_of_special_requests', perc=True)
```

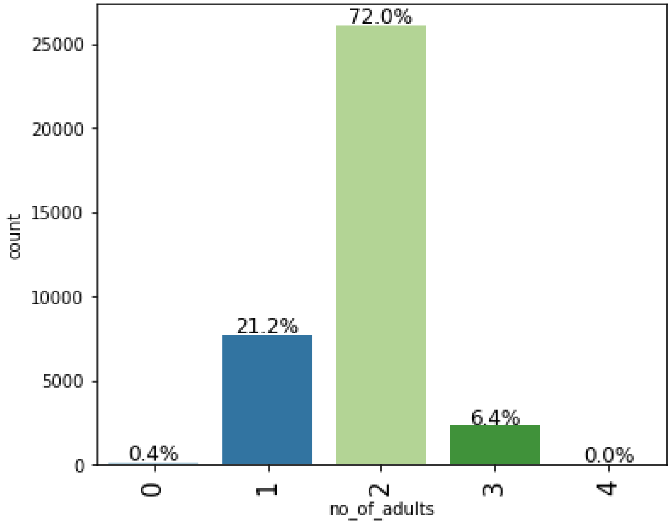


Observations:

- Number of special requests:
    - 3 or more: no bookings were canceled
    - 2: over 85% were not canceled
    - 1: over 75% were not canceled
    - 0: around 56% of bookings were not canceled
- Canceled vs Not Canceled
    - Out of bookings that happened to be canceled, 71.9% had 0 special requests.
    - Out of bookings that happened to NOT be canceled, 46.1% had 0 special requests, and 35.5% had 1.

Univariate Analysis

In [23]:
```
bar(df, 'no_of_adults', perc=True)
```

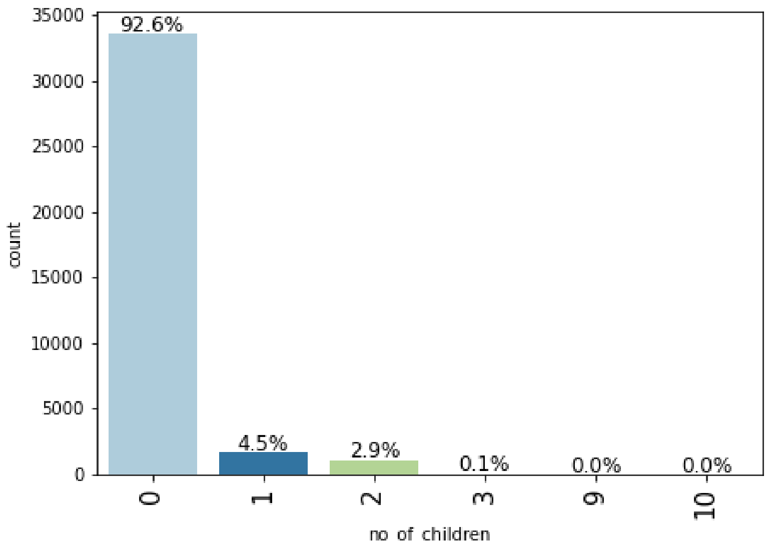


Observations:

- 72% of the bookings were made for 2 adults.
- Many outliers, more so on the lower end of the range.

In [24]:

```
bar(df, 'no_of_children', perc=True)
```



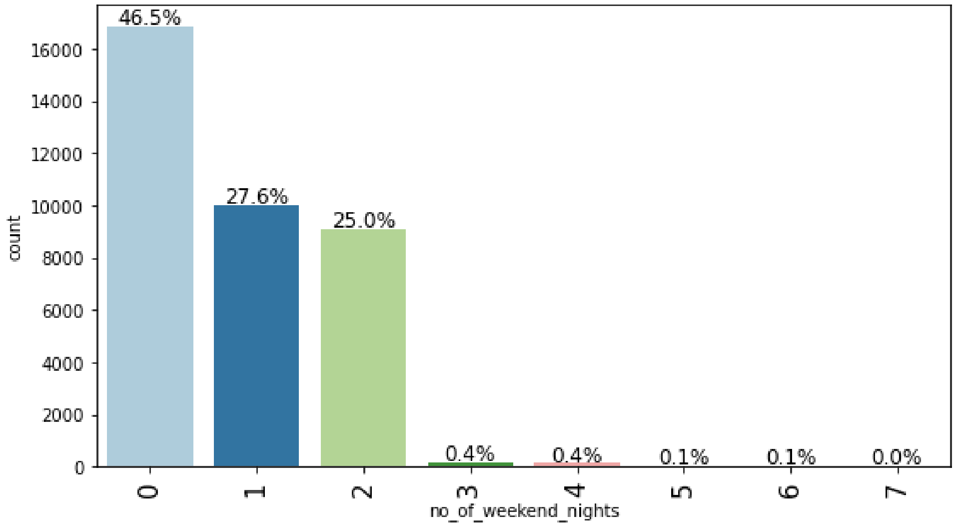
Observations:

- 93% of the customers didn't make reservations for children.

- There are some values in the data where the number of children is 9 or 10, which is highly unlikely.
- We will replace these values with the maximum value of 3 children.

In [25]:
```python
# replacing 9, and 10 children with 3

df["no_of_children"] = df["no_of_children"].replace([9, 10], 3)
```

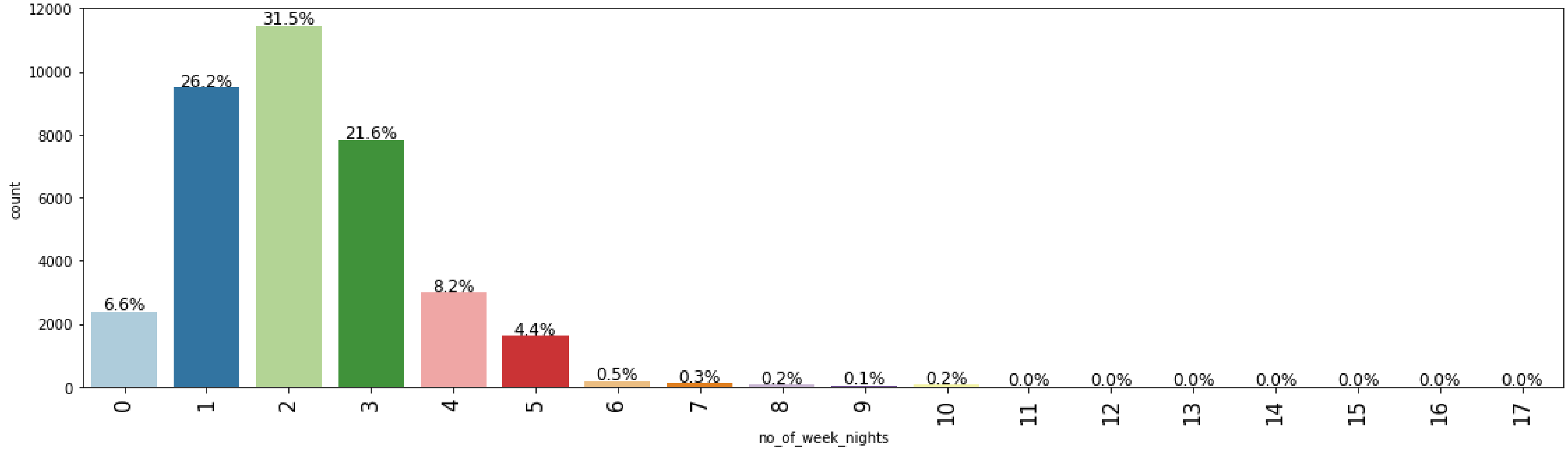
In [26]:
```python
bar(df, 'no_of_weekend_nights', perc=True)
```



Observations:

- 46.5% of the customers do not plan to spend the weekend in the hotel.
- The percentage of customers planning to spend 1 or 2 weekends in the hotel is almost the same

In [27]:
```python
bar(df, 'no_of_week_nights', perc=True)
```

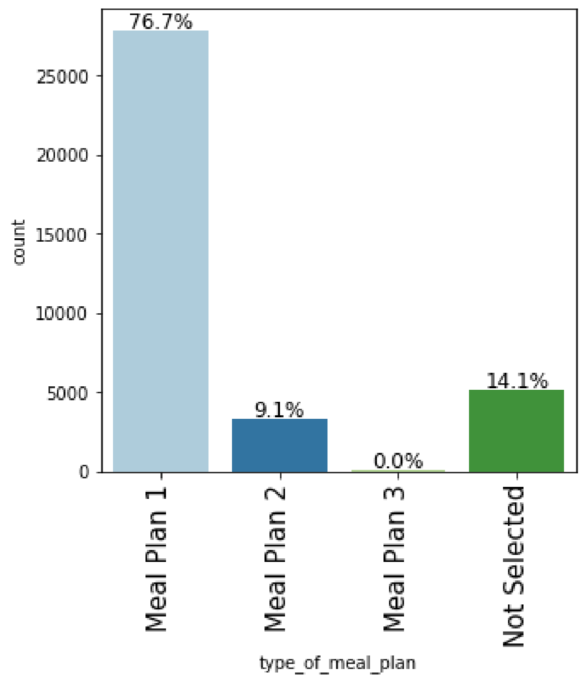


Observations:

- Most bookings are made for 2 nights (31.5%) followed by 1 night (26.2%).
- A very small proportion of customers made the booking for more than 10 days.

In [28]:

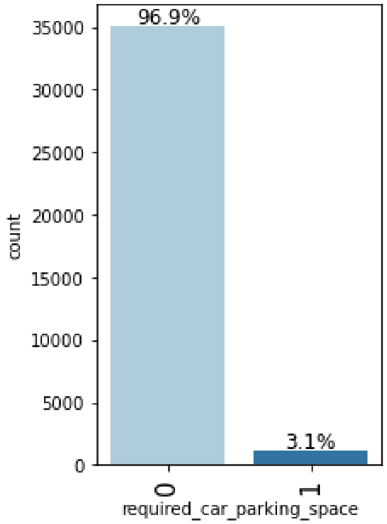
```
bar(df, 'type_of_meal_plan', perc=True)
```



Observations:

- Most of the customers prefer meal plan 1 that is only breakfast.
- 14.1% of the customers didn't select a meal plan.

In [29]:
```
bar(df, 'required_car_parking_space', perc=True)
```



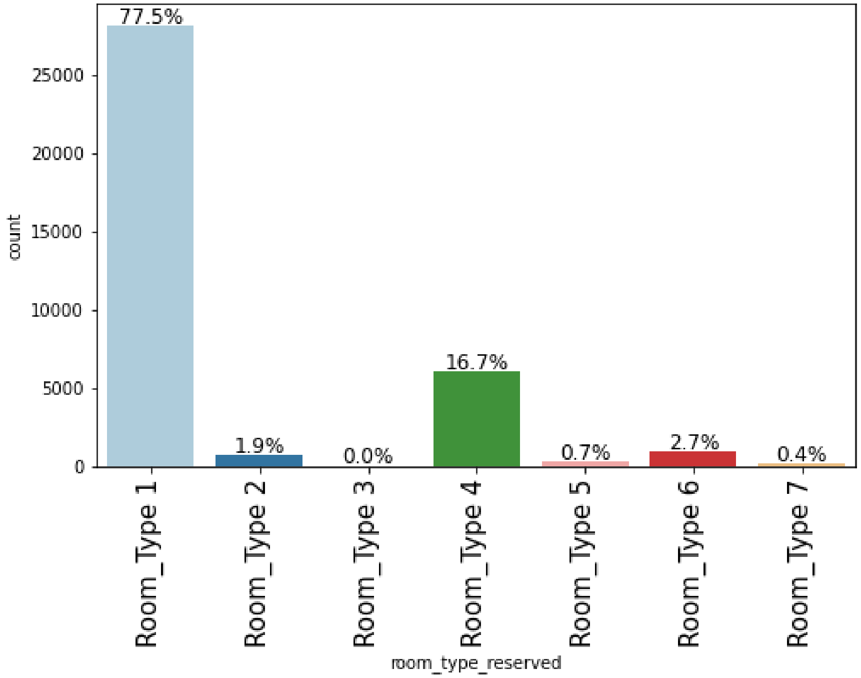
In [30]:
```
df['required_car_parking_space'].value_counts()
```

Out[30]:
```
0    35151
1     1124
Name: required_car_parking_space, dtype: int64
```

Observations:

- Does the customer require a car parking space? (0 - No, 1- Yes)
- 96.9% of the customers do not require a car parking space.

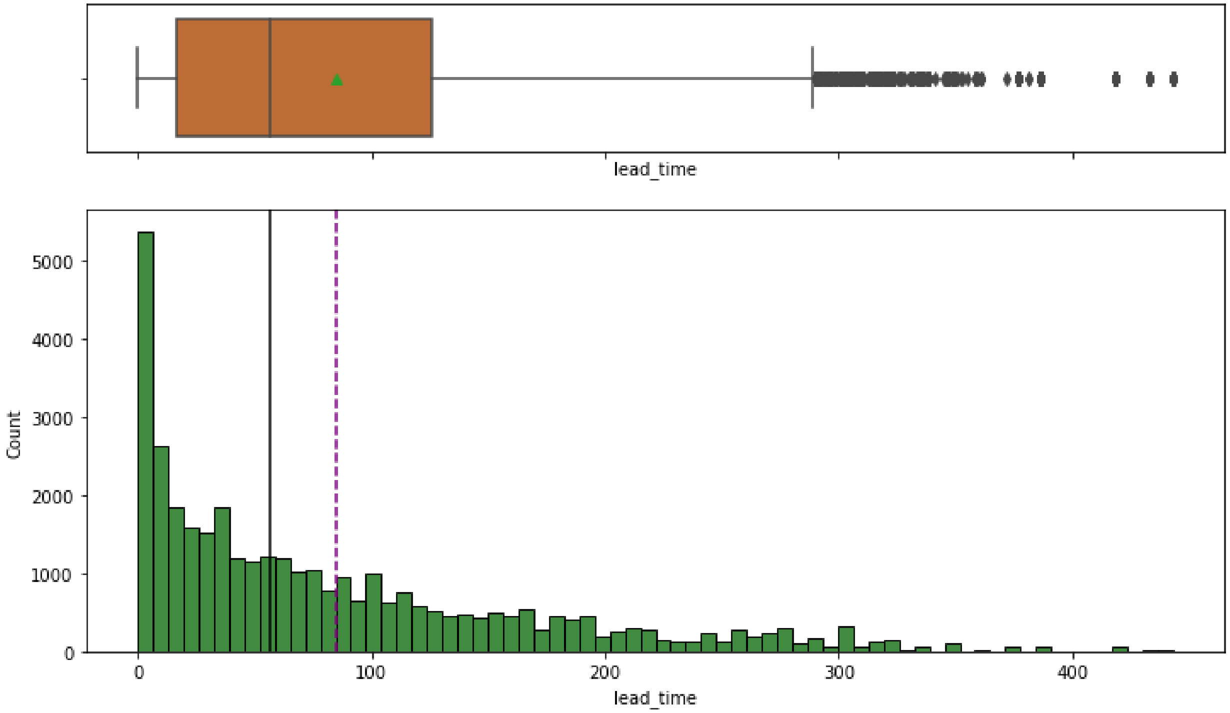
In [31]:
```
bar(df, 'room_type_reserved', perc=True)
```



Observations:

- Around 77% of the customers booked Room_Type 1 followed by 17% of the customers booking Room_Type 4.

In [32]:
```
histbox(df, 'lead_time')
```



In [33]:

```
df['lead_time'].describe()
```

Out[33]:

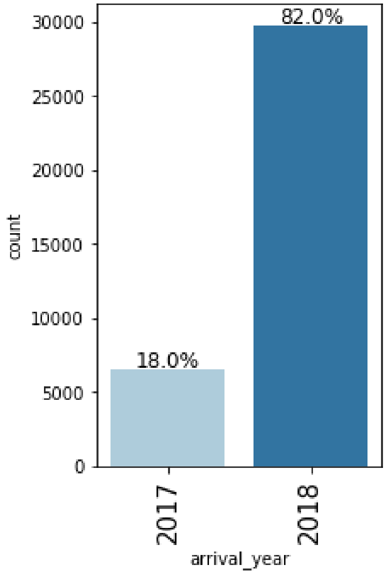
```
count    36275.00000
mean        85.23256
std         85.93082
min          0.00000
25%         17.00000
50%         57.00000
75%        126.00000
max        443.00000
Name: lead_time, dtype: float64
```

Observations:

- The average lead time is around 85 days.
- The distribution of lead time is right-skewed, and there are many outliers.
- Some customers made booking around 500 days in advance.
- Many customers have made the booking on the same day of arrival as well.

In [34]:

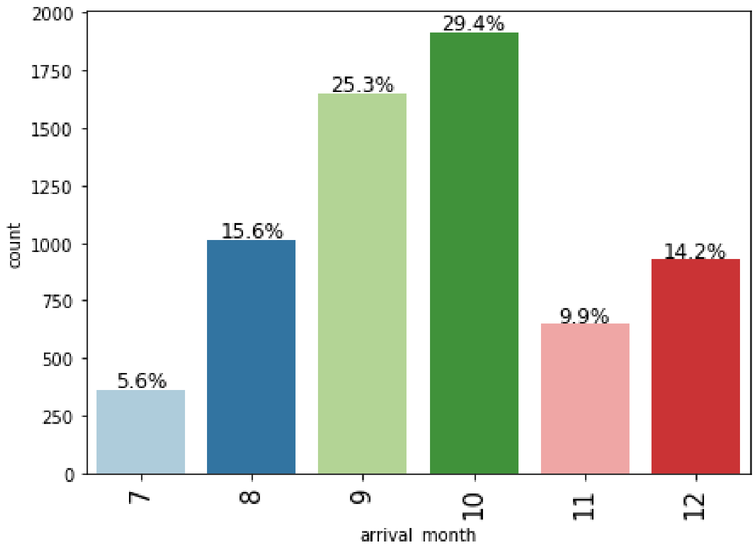
```
bar(df, 'arrival_year', perc=True)
```



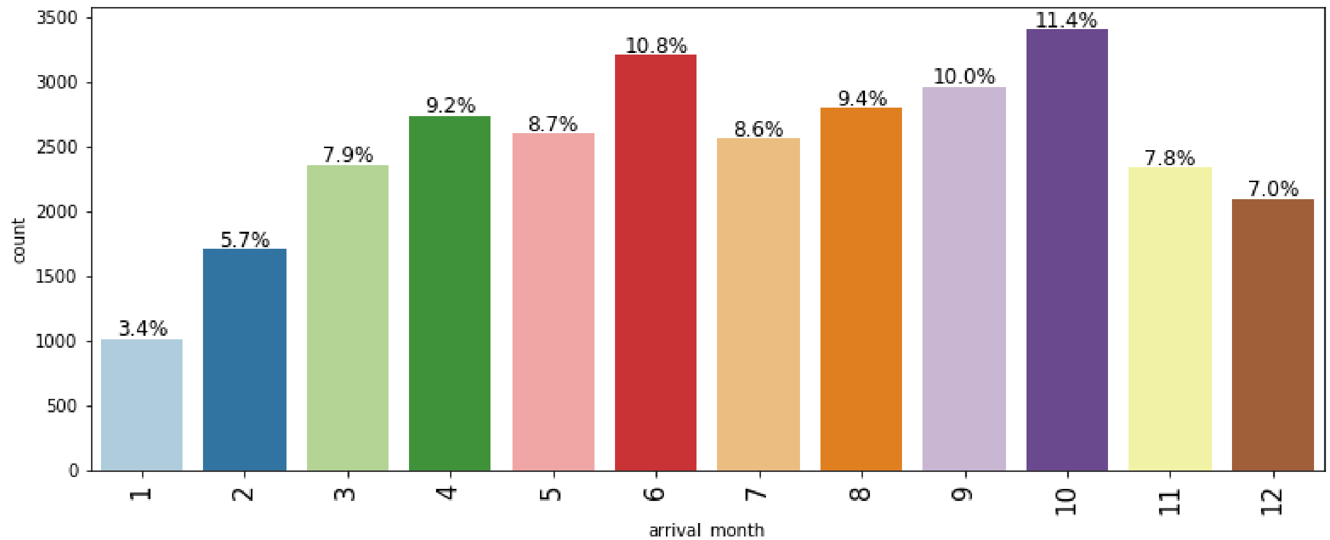
Observations:

- 82% of the hotel guests in this dataset booked a room in 2018.

In [35]:
```
year1 = df[df['arrival_year']==2017]
year2 = df[df['arrival_year']==2018]

bar(year1, "arrival_month", perc=True)
```



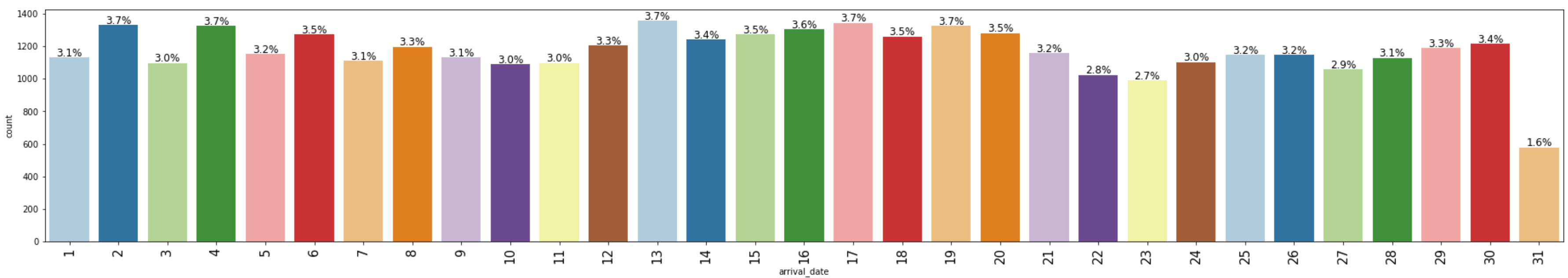
In [36]:
```
bar(year2, "arrival_month", perc=True)
```



Observations:

- 2017 bookings cover only the last 6 months of the year, 2018 covers all.
- October consists of the highest number of bookings for both years. (29% in 2017, 11% in 2018).

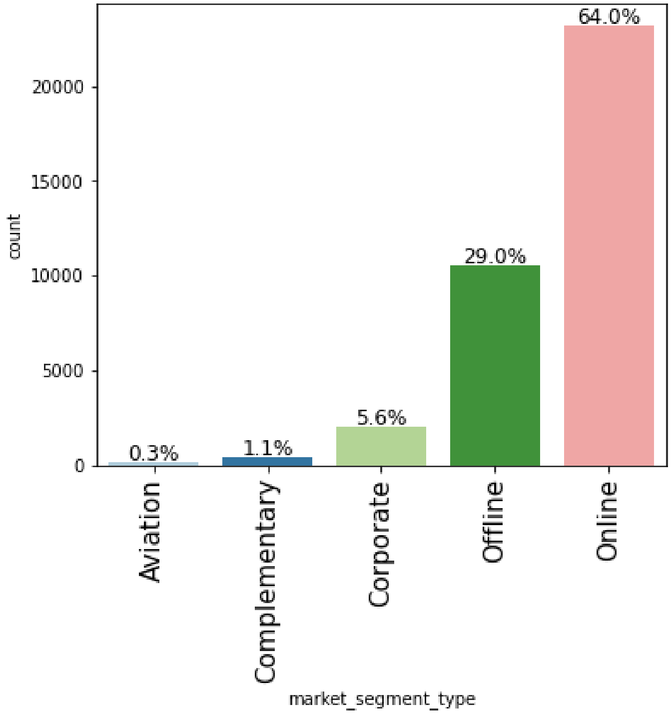
In [37]:
```
bar(df, 'arrival_date', perc=True)
```



Observations:

- Arrival date is surprisingly evenly distributed among all the days of a month.

In [38]:
```
bar(df, 'market_segment_type', perc=True)
```

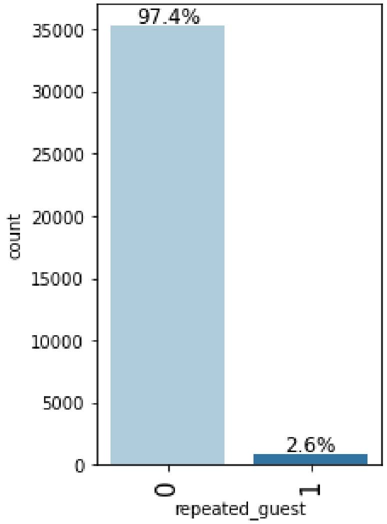


Observations:

- 64% of the hotel bookings were made online followed by 29% of the bookings which were made offline.

In [39]:

```
bar(df, 'repeated_guest', perc=True)
```



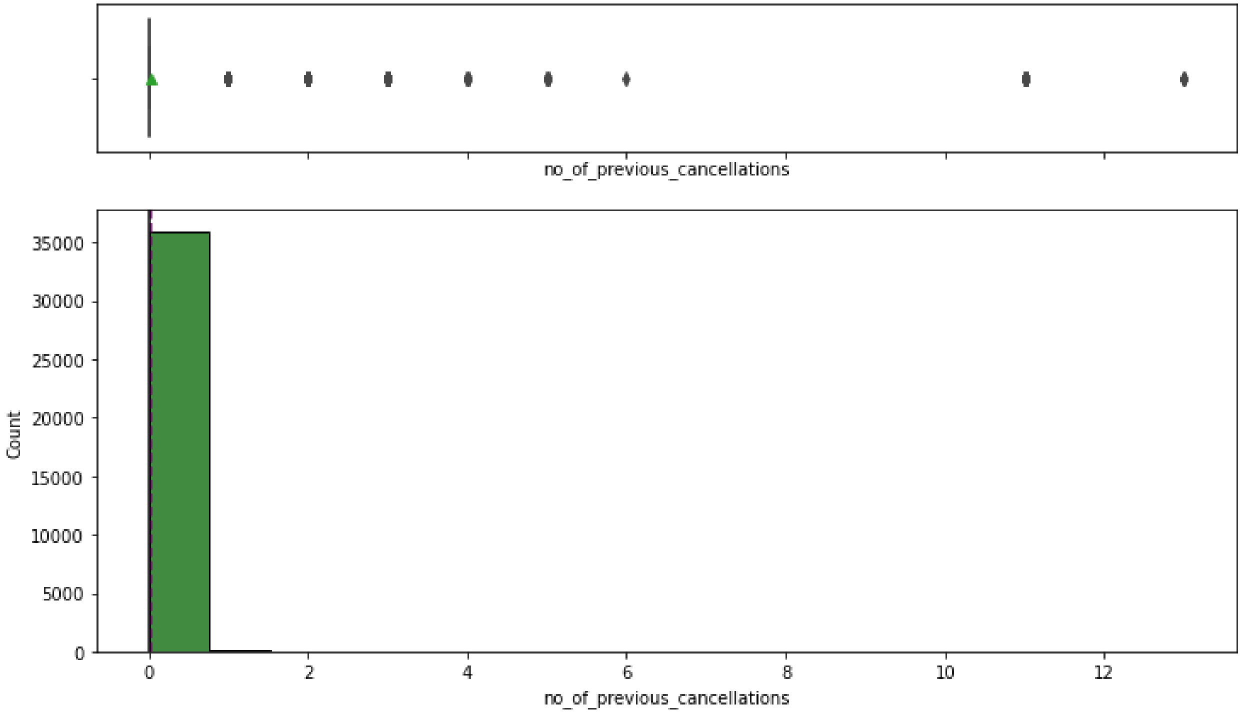
In [40]:
```
df['repeated_guest'].value_counts()
```

Out[40]:
```
0    35345
1      930
Name: repeated_guest, dtype: int64
```

Observations:

- Is the customer a repeated guest? (0 - No, 1- Yes)
- The majority of customers are not repeating guests, this might be due to the dataset only including hotel stays for two years, something to look into.

In [41]:
```
histbox(df, 'no_of_previous_cancellations')
```



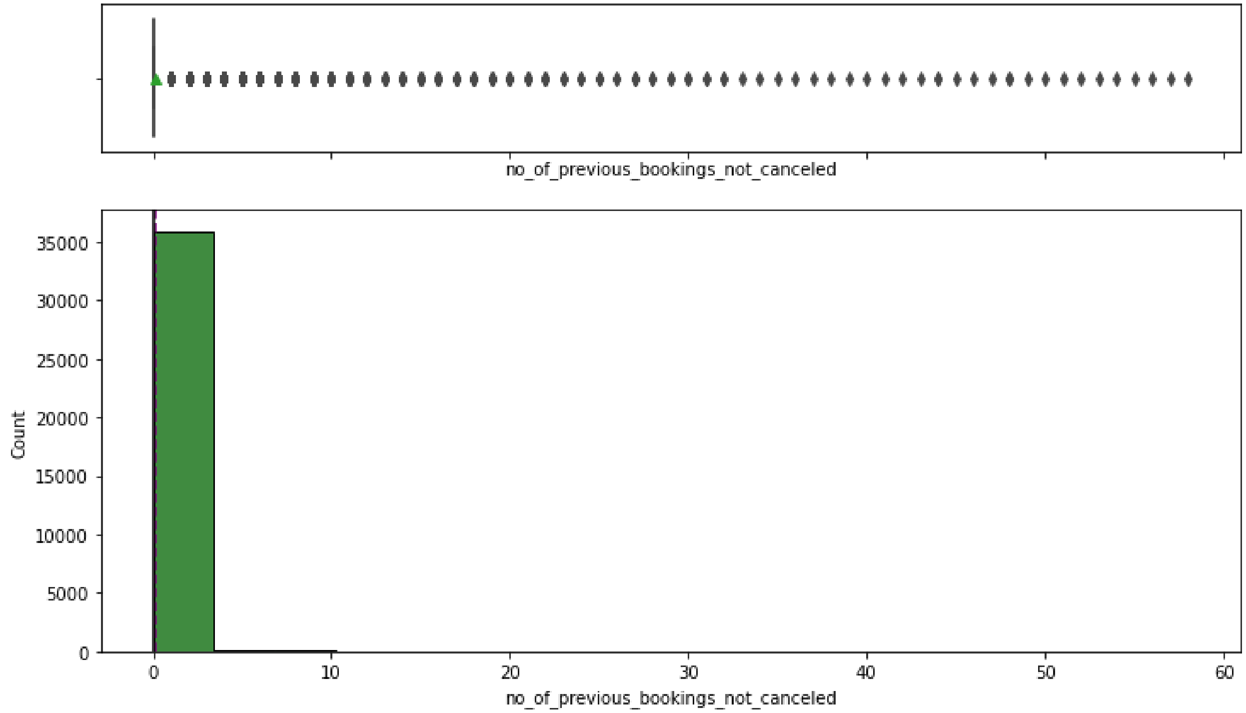
In [42]:
```
df['no_of_previous_cancellations'].value_counts()
```

Out[42]:
```
0     35937
1       198
2        46
3        43
11       25
5        11
4        10
13        4
6         1
Name: no_of_previous_cancellations, dtype: int64
```

Observations:

- Very few customers have more than one cancellation.
- Some customers canceled more than 12 times.

In [43]:
```
histbox(df, 'no_of_previous_bookings_not_canceled')
```



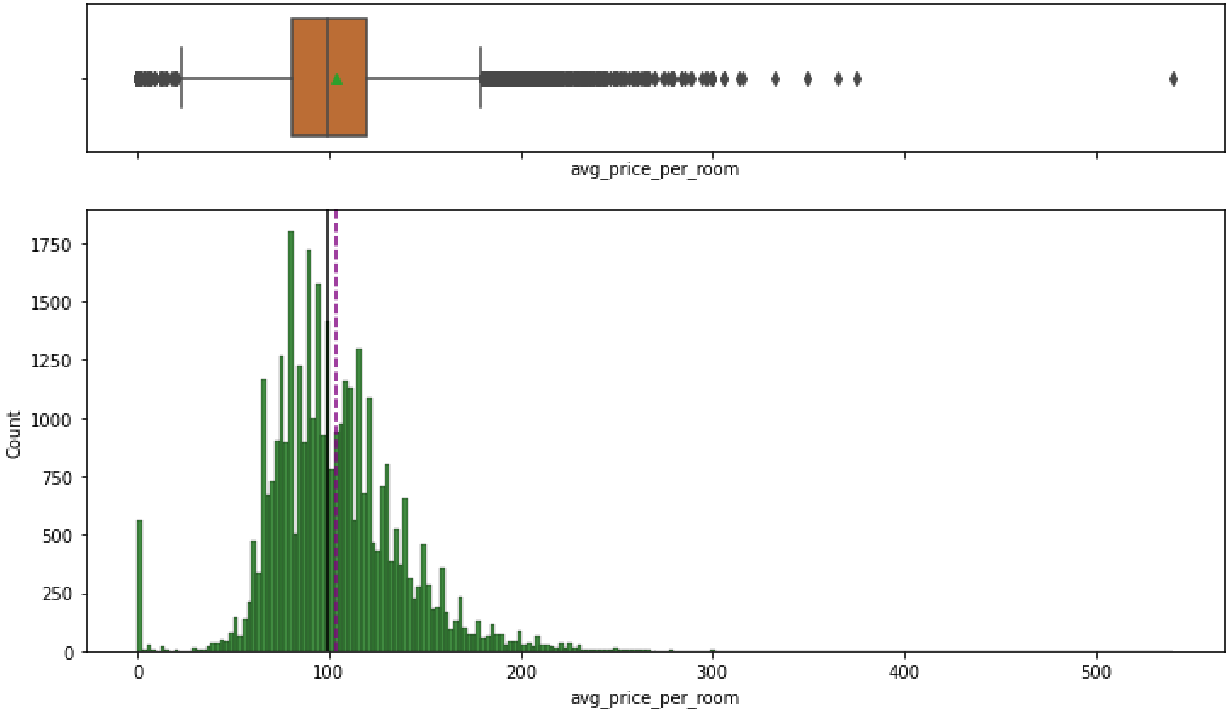
In [44]:
```
df['no_of_previous_bookings_not_canceled'].value_counts()[:10]
```

Out[44]:
```
0     35463
1       228
2       112
3        80
4        65
5        60
6        36
7        24
8        23
10       19
Name: no_of_previous_bookings_not_canceled, dtype: int64
```

Observations:

- Very few customers have more than 1 booking not canceled previously.
- Some customers have not canceled their bookings around 60 times.

In [45]:
```
histbox(df, 'avg_price_per_room')
```



In [46]:
```
#large number of avg price = 0 dollars, see what some of it looks like:

zero = df[df["avg_price_per_room"] == 0]

zero.head()
```

Out[46]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	lead_time	a
63	1	0	0	1	Meal Plan 1	0	Room_Type 1	2	
145	1	0	0	2	Meal Plan 1	0	Room_Type 1	13	
209	1	0	0	0	Meal Plan 1	0	Room_Type 1	4	
266	1	0	0	2	Meal Plan 1	0	Room_Type 1	1	
267	1	0	2	1	Meal Plan 1	0	Room_Type 1	4	

In [47]:
```
zero.groupby("market_segment_type")['avg_price_per_room'].count()
```

Out[47]:
```
market_segment_type
Complementary    354
Online           191
Name: avg_price_per_room, dtype: int64
```

In [48]:
```
df.groupby("market_segment_type")['avg_price_per_room'].count()
```

Out[48]:
```
market_segment_type
Aviation            125
Complementary       391
Corporate          2017
Offline           10528
Online            23214
Name: avg_price_per_room, dtype: int64
```

In [49]:
```
df['avg_price_per_room'].describe()
```

Out[49]:
```
count   36275.00000
mean      103.42354
std        35.08942
min         0.00000
25%        80.30000
50%        99.45000
75%       120.00000
max       540.00000
Name: avg_price_per_room, dtype: float64
```

Observations:

- The distribution of average price per room is skewed to right. There are outliers on both sides.
- The average price of a room is around ~100 euros.
- There is 1 observation where the average price of the room is more than 500 euros. This observation is quite far away from the rest of the values. Instead of dropping it, we will clip this to the upper whisker (Q3 + 1.5 * IQR).

- With rooms with a price equal to 0:
  - It makes sense that most values with room prices equal to 0 are the rooms given as complimentary service given by the hotel.
  - The rooms booked online must be a part of some promotional campaign done by the hotel.

In [50]:
```
# Calculating the 25th quantile
Q1 = df["avg_price_per_room"].quantile(0.25)

# Calculating the 75th quantile
Q3 = df["avg_price_per_room"].quantile(0.75)

# Calculating IQR
IQR = Q3 - Q1

# Calculating value of upper whisker
Upper_Whisker = Q3 + 1.5 * IQR
Upper_Whisker
```

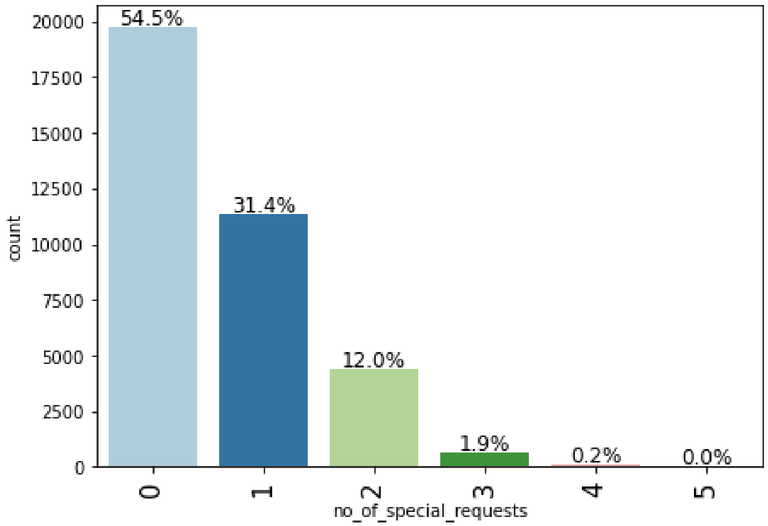
Out[50]:
```
179.55
```

In [51]:
```
# assigning the outliers the value of upper whisker
```

```
df.loc[df["avg_price_per_room"] >= 500, "avg_price_per_room"] = Upper_Whisker
```

In [52]:

```
bar(df,'no_of_special_requests', perc=True)
```

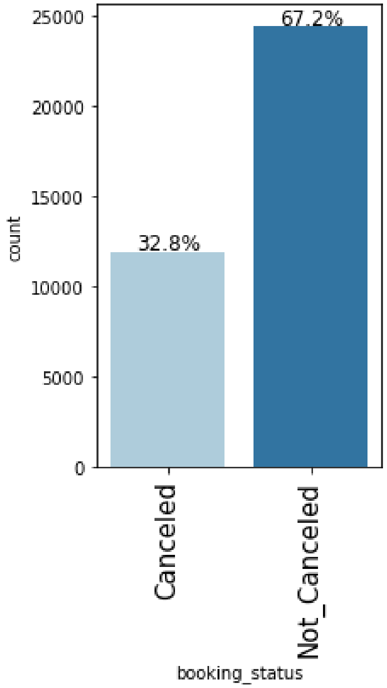


Observations:

- The majority (54.5%) of hotel guests did not put in any special requests.
- About a third put in 1 request, and 12% put in 2.

In [53]:

```
bar(df, 'booking_status', perc=True)
```



Observations:

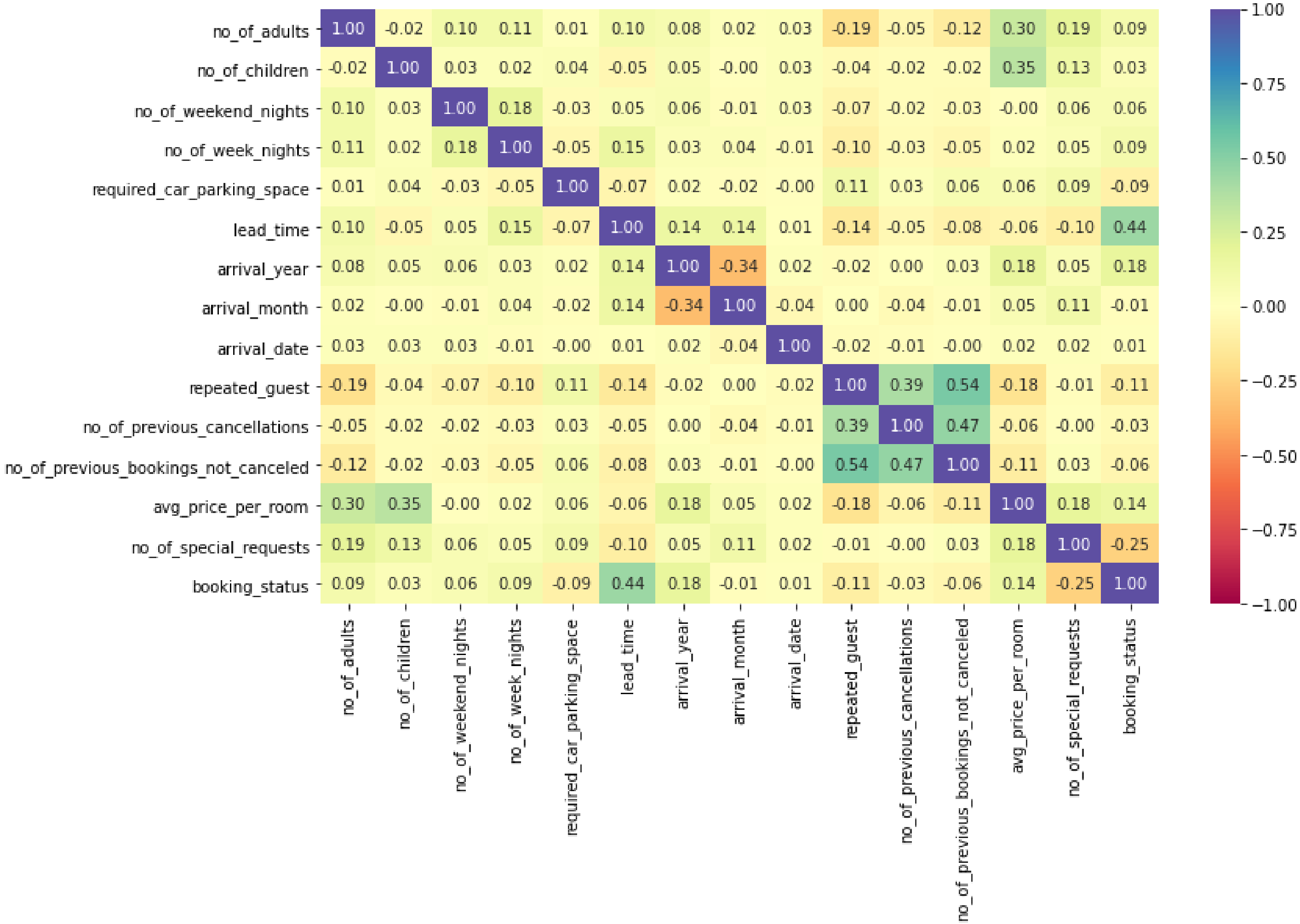
- The majority (67.2%) of hotel bookings are not cancelled.

`required_car_parking_space` and `repeated_guest` are both coded as (0 - No, 1 - Yes). **Let's do this for booking_status, code (0 - Not_Canceled, 1 - Canceled).**

In [54]:
```
df["booking_status"] = df["booking_status"].apply(lambda x: 1 if x == "Canceled" else 0)
```

Bivariate Analysis

In [55]:
```
corr_cols = df.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(12, 7))
sns.heatmap(
    df[corr_cols].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



Observations:

- There's a positive correlation between the number of customers (adults and children) and the average price per room.
  - This makes sense as more the number of customers more rooms they will require thus increasing the cost.
- There's a negative correlation between average room price and repeated guests. The hotel might be giving some loyalty benefits to the customers.
- There's a positive correlation between the number of previous bookings canceled and previous bookings not canceled by a customer and repeated guest.
- There's a positive correlation between lead time and the number of weeknights a customer is planning to stay in the hotel.
- There's a positive correlation between booking status and lead time, indicating higher the lead time higher are the chances of cancellation. We will analyze it further.
- There's a negative correlation between the number of special requests from the customer and the booking status, indicating if a customer has made some special requests the chances of cancellation might decrease. We will analyze it further.

In [56]:
```
# function to plot stacked barplot

def stack(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_values(
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 5, 6))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

In [57]:
```
# function to plot distributions with respect to target


def dist_target(data, predictor, target):

    fig, axs = plt.subplots(2, 2, figsize=(12, 10))

    target_uniq = data[target].unique()

    axs[0, 0].set_title("Distribution of target for target=" + str(target_uniq[0]))
    sns.histplot(
        data=data[data[target] == target_uniq[0]],
        x=predictor,
        kde=True,
        ax=axs[0, 0],
        color="teal",
        stat="density",
    )

    axs[0, 1].set_title("Distribution of target for target=" + str(target_uniq[1]))
    sns.histplot(
        data=data[data[target] == target_uniq[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
```

```
        color="orange",
        stat="density",
    )

    axs[1, 0].set_title("Boxplot w.r.t target")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette="gist_rainbow")

    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(
        data=data,
        x=target,
        y=predictor,
        ax=axs[1, 1],
        showfliers=False,
        palette="gist_rainbow",
    )

    plt.tight_layout()
    plt.show()
```

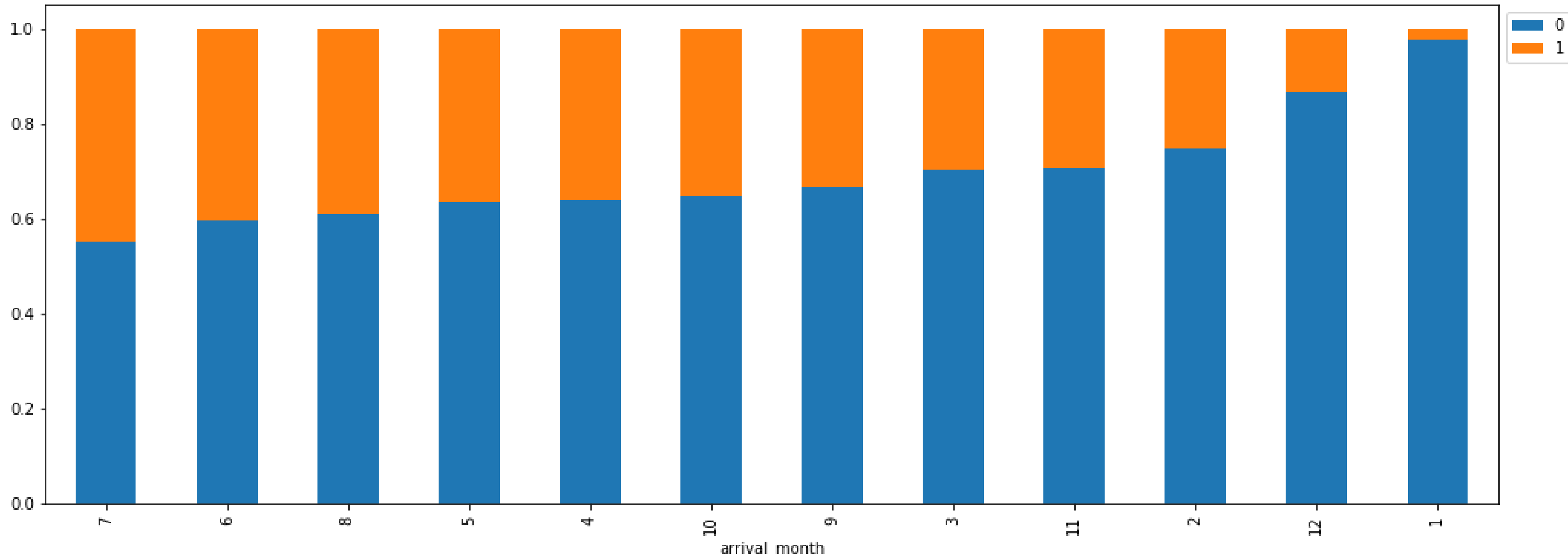
Leading Question 1 asks what the busiest months of the hotel are.

Observations:

- The top 6 busiest months for the hotel are all in the last half of the year:
  - October, then September, August, June, December, and November.

In [58]:
```
# check month of booking against cancel/not cancelled

stack(df, 'arrival_month', 'booking_status')
```

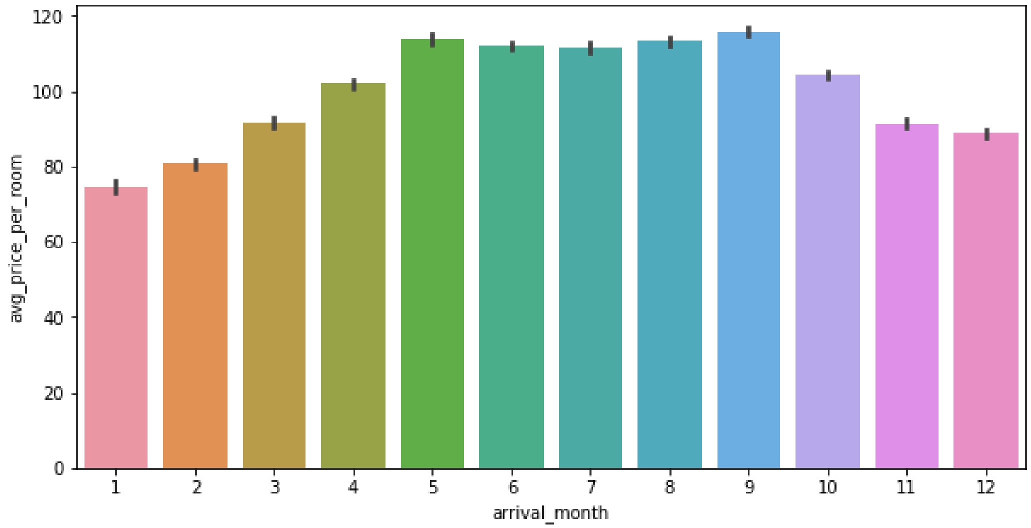
```
booking_status      0      1    All
arrival_month
All             24390  11885  36275
10               3437   1880   5317
9                3073   1538   4611
8                2325   1488   3813
7                1606   1314   2920
6                1912   1291   3203
4                1741    995   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
------------------------------------------------------------------------------------------------------------------------
```



Observations:

- Data from 2017 only includes last half of the year, so that might contribute to more cancellations being on July and August.
- The busiest months are October, then September, which both fall in the middle for number of cancellations.
- December, then January have the least number of cancellations
  - Surprising, since December is one of the busiest months.
  - January has the least bookings, so it follows that it would have fewer cancellations.

In [59]:
```
# relationship between average room price and arrival month

plt.figure(figsize=(10, 5))
sns.barplot(x = "arrival_month", y="avg_price_per_room", data=df)
plt.show()
```



Observations:

- Months May - September are around the same in terms of room price.
- Busiest month (October) room price lower than expected

Fig 1 (from Leading Question 3): compares market segment to average room price.

Observations:

- The average room price is:
  - highest for the most common market segment type: Online.
  - lowest for Complementary
- Every market segment except Complementary ranges around $100 for room price.
- Offline has the most outliers, followed by Online, Corporate, and Complementary.

In [60]:

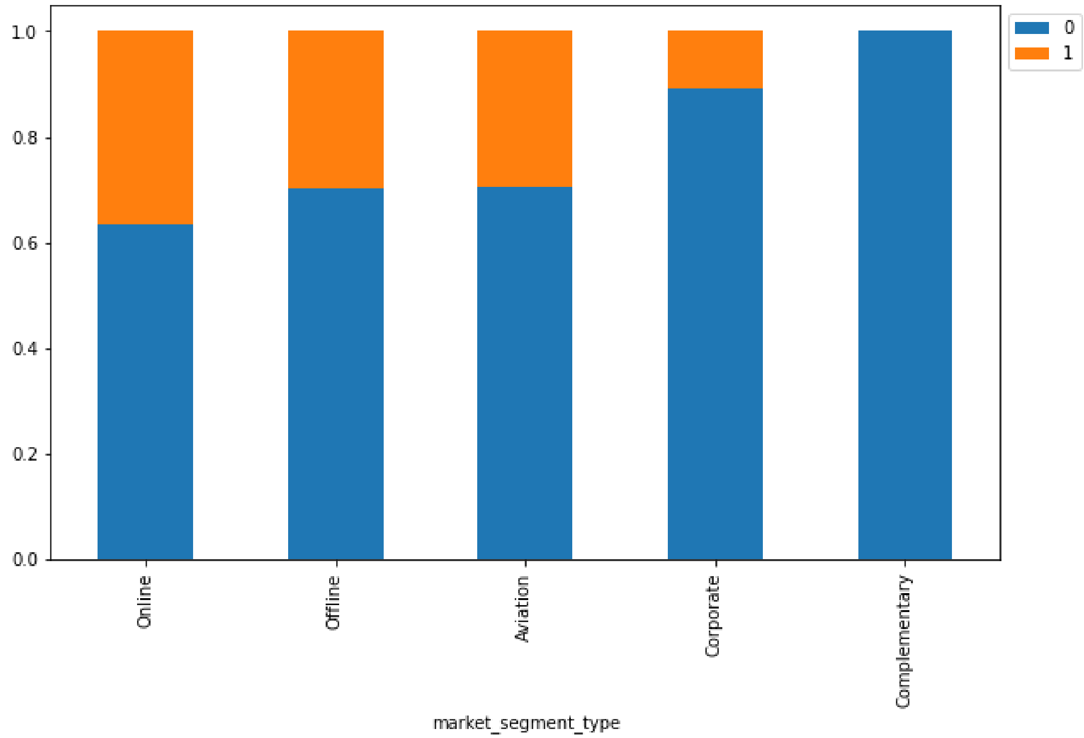
```
# compare market segment to bookings

df.groupby("market_segment_type")["booking_status"].value_counts()
```

Out[60]:

```
market_segment_type  booking_status
Aviation             0                    88
                     1                    37
Complementary        0                   391
Corporate            0                  1797
                     1                   220
Offline              0                  7375
                     1                  3153
Online               0                 14739
                     1                  8475
Name: booking_status, dtype: int64
```

In [61]:
```
stack(df, 'market_segment_type', 'booking_status')
```

```
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
------------------------------------------------------------------------------------------------------------------------
```



Observations:

- Around 40% of the online booking were canceled.
- Bookings made offline are less prone to cancellations.
- Corporate segment shows very low cancellations.

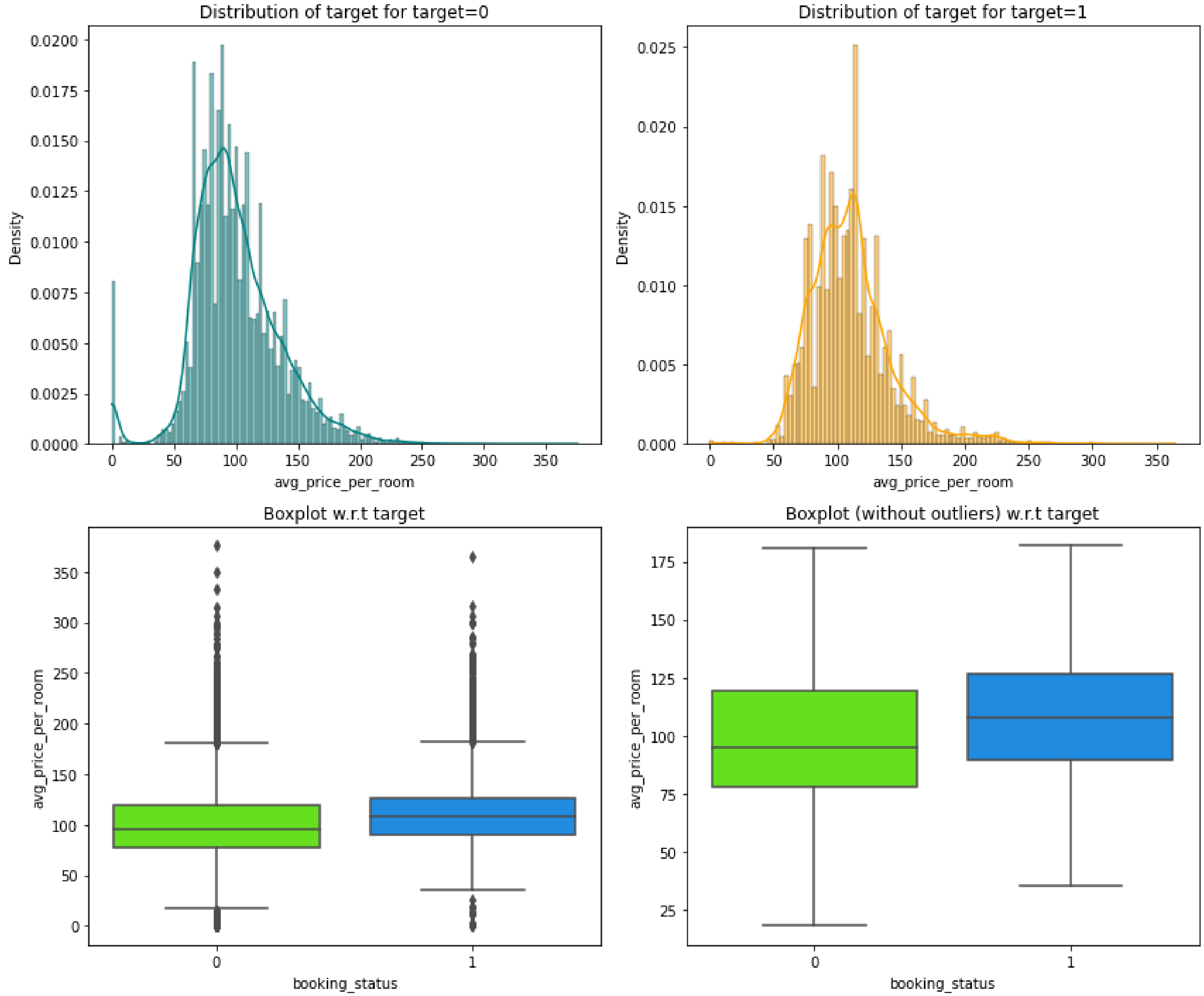
In [62]:
```
df.groupby("booking_status")["avg_price_per_room"].value_counts()
```

Out[62]:
```
booking_status  avg_price_per_room
0               65.00000              758
                75.00000              578
                0.00000               539
```

```
                     95.00000             483
                     90.00000             459
                                          ...
1                    284.10000              1
                     299.33000              1
                     306.00000              1
                     316.00000              1
                     365.00000              1
Name: avg_price_per_room, Length: 4941, dtype: int64
```

In [63]:

```
dist_target(df, 'avg_price_per_room', 'booking_status')
```

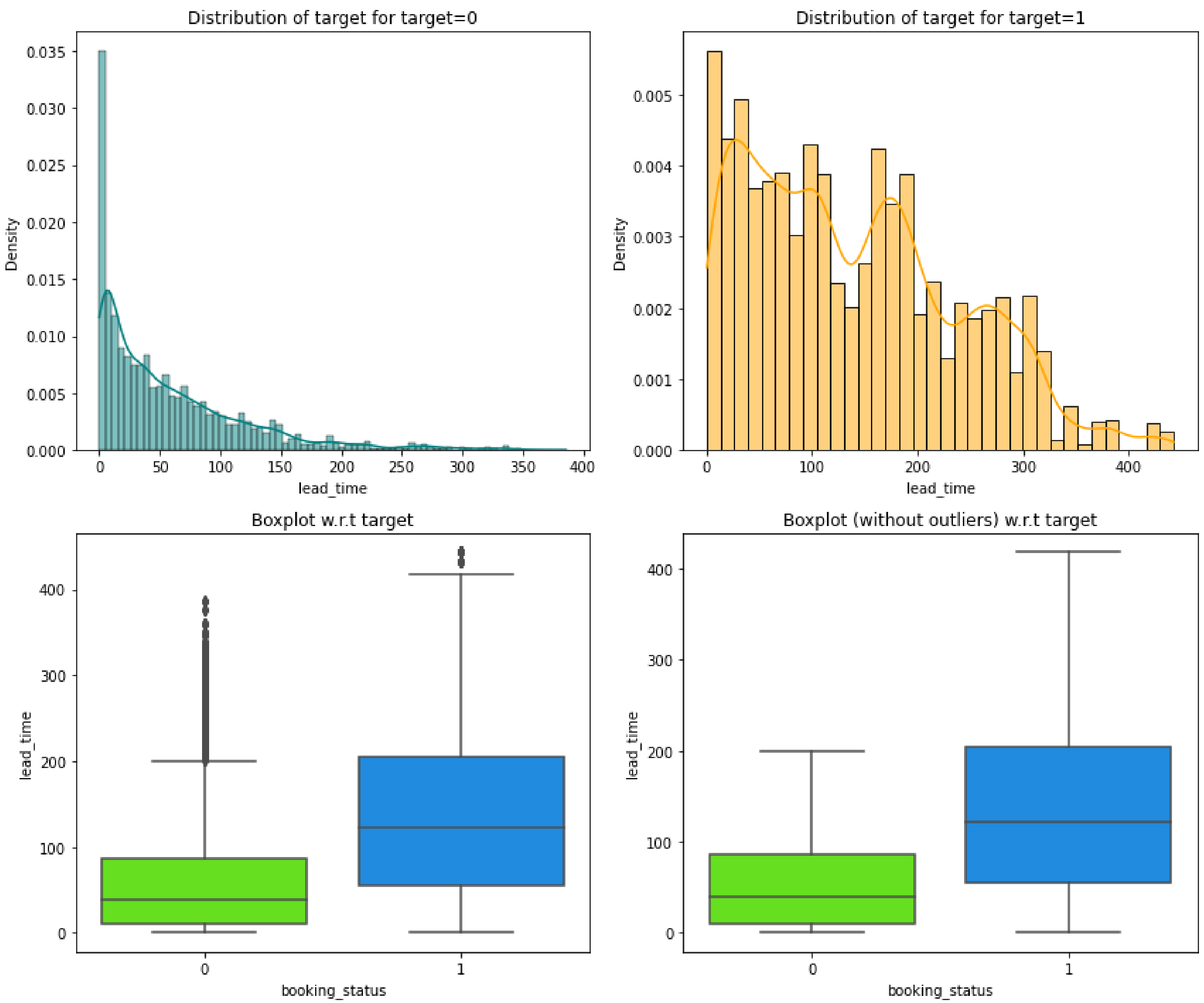


Observations:

- The distribution of price for canceled bookings and not canceled bookings is quite similar.
- The prices for the canceled bookings are slightly higher than the bookings which were not canceled.

In [64]:
```
# relationship between lead time and booking status
```

```
dist_target(df, 'lead_time', 'booking_status')
```



Observations:

- There's a big difference in the median value of lead time for bookings that were canceled and bookings that were not canceled.
- Higher the lead time higher are the chances of a booking being canceled.

Leading Question 5 details how many repeating guests have cancellations.

In the dataset, 2.6% are repeating guests. The percentage of bookings that are cancelled by repeating guests is 1.72 %. The percentage of bookings that are cancelled in general is 32.8%.

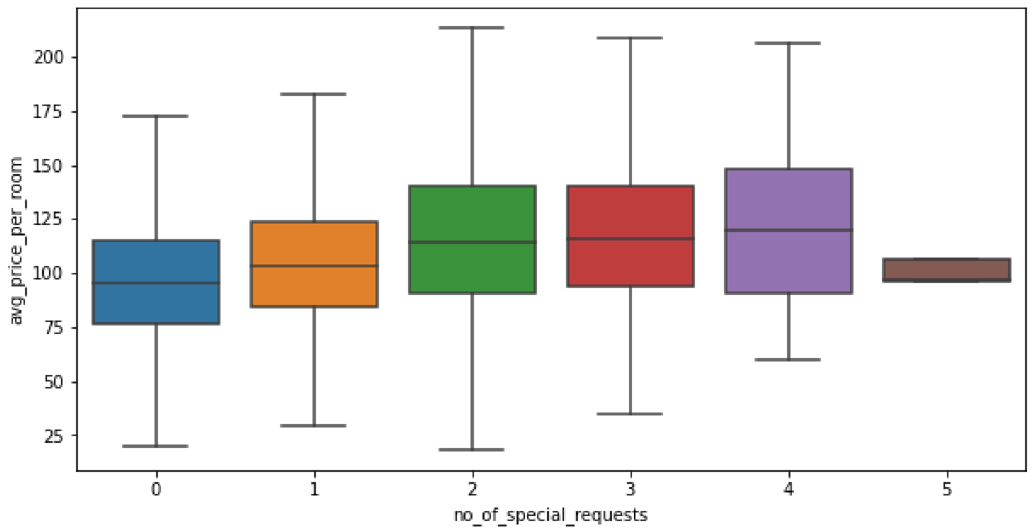
Leading Question 6 details how special requests affect booking status/cancellations.

Observations:

- Number of special requests:
  - 3 or more: no bookings were canceled
  - 2: over 85% were not canceled
  - 1: over 75% were not canceled
  - 0: around 56% of bookings were not canceled
- Canceled vs Not Canceled
  - Out of bookings that happened to be canceled, 71.9% had 0 special requests.
  - Out of bookings that happened to NOT be canceled, 46.1% had 0 special requests, and 35.5% had 1.

In [65]:

```
# relationship between special requests and average room price

plt.figure(figsize=(10, 5))
sns.boxplot(
    data=df,
    x="no_of_special_requests",
    y="avg_price_per_room",
    showfliers=False,  # turning off the outliers
)
plt.show()
```



Observations:

- The median prices of the rooms where some special requests were made by the customers are slightly higher than the rooms where customer didn't make any requests.

Generally people travel with their spouse and children for vacations or other activities. Let's create a new dataframe of the customers who traveled with their families and analyze the impact on booking status.

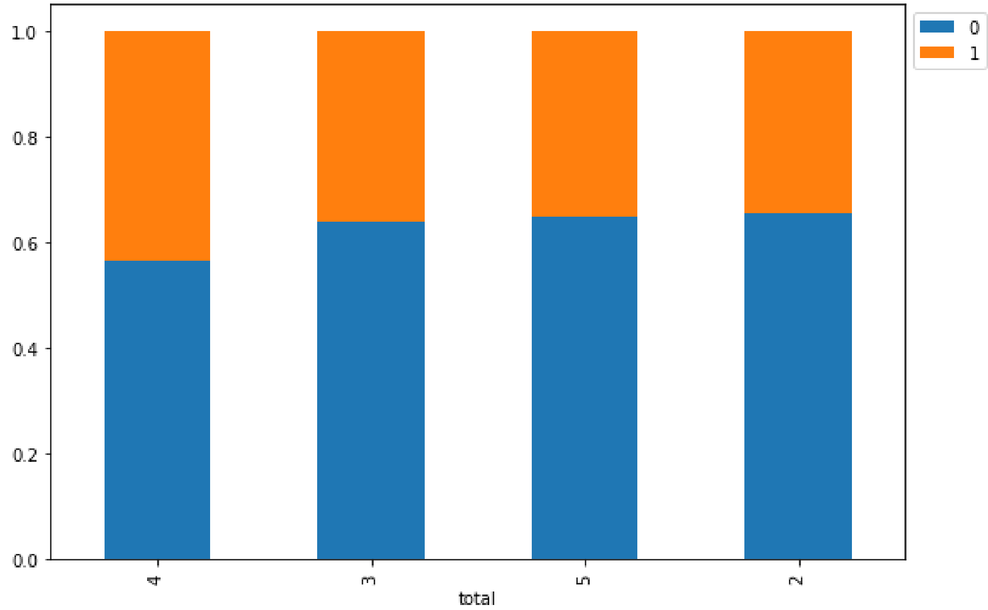
In [66]:
```
family = df[(df["no_of_children"] >= 0) & (df["no_of_adults"] > 1)]

family['total'] = family['no_of_children'] + family['no_of_adults']

family.head()
```

Out[66]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	lead_time	arri
0	2	0	1	2	Meal Plan 1	0	Room_Type 1	224	
1	2	0	2	3	Not Selected	0	Room_Type 1	5	
3	2	0	0	2	Meal Plan 1	0	Room_Type 1	211	
4	2	0	1	1	Not Selected	0	Room_Type 1	48	
5	2	0	0	2	Meal Plan 2	0	Room_Type 1	346	

In [67]:
```
stack(family, 'total', 'booking_status')
```

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



Observations:

- There's a ~40% chance of a booking getting canceled if the booking is made for 4 family members.
- Surprising that November and December, two of the busiest months, had so few bookings with family groups.
- February - May had a solid number of family bookings, but it looks like about half of those were cancelled.

Let's do a similar analysis for the customer who stay for at least a day at the hotel.

In [68]:
```
stay_data = df[(df["no_of_week_nights"] > 0) & (df["no_of_weekend_nights"] > 0)]
stay_data.shape
```

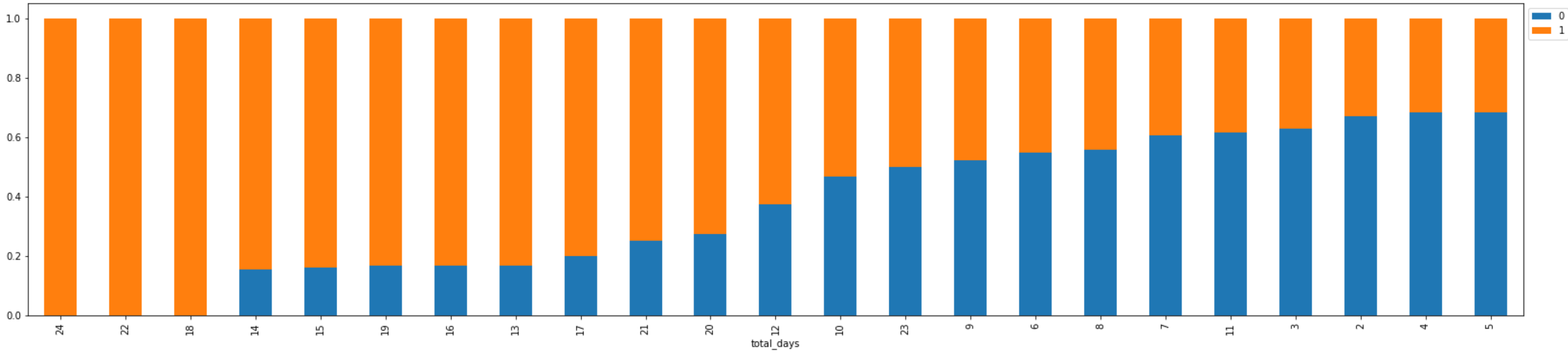
Out[68]: (17094, 18)

In [69]:
```
stay_data["total_days"] = (stay_data["no_of_week_nights"] + stay_data["no_of_weekend_nights"])
```

In [70]:
```
stack(stay_data, "total_days", "booking_status")
```

```
booking_status      0     1    All
total_days
All             10979  6115  17094
3                3689  2183   5872
4                2977  1387   4364
5                1593   738   2331
2                1301   639   1940
6                 566   465   1031
7                 590   383    973
```

```
8                  100    79    179
10                  51    58    109
9                   58    53    111
14                   5    27     32
15                   5    26     31
13                   3    15     18
12                   9    15     24
11                  24    15     39
20                   3     8     11
19                   1     5      6
16                   1     5      6
17                   1     4      5
18                   0     3      3
21                   1     3      4
22                   0     2      2
23                   1     1      2
24                   0     1      1
------------------------------------------------------------------------------------------------------------------------
```

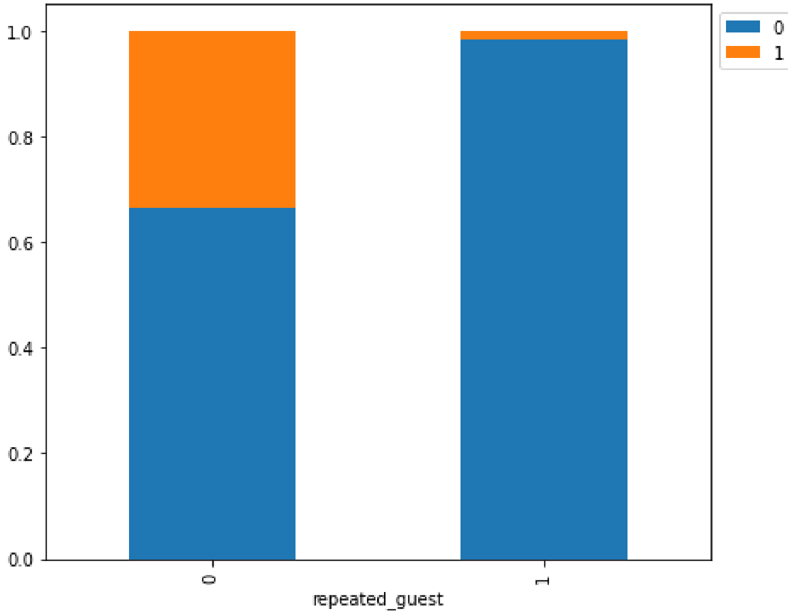


- The general trend is that the chances of cancellation increase as the number of days the customer planned to stay at the hotel increases.

In [71]:

```
stack(df, "repeated_guest", "booking_status")
```

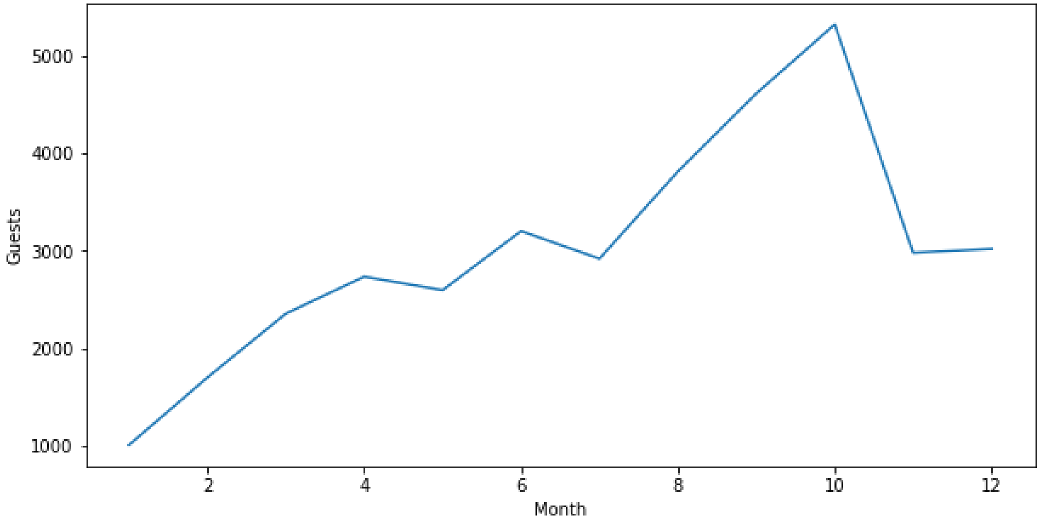
```
booking_status      0      1    All
repeated_guest
All             24390  11885  36275
0               23476  11869  35345
1                 914     16    930
------------------------------------------------------------------------------------------------------------------------
```



- There are very few repeat customers but the cancellation among them is very less.
- This is a good indication as repeat customers are important for the hospitality industry as they can help in spreading the word of mouth.
- A loyal guest is usually more profitable for the business because they are more familiar with what is on offer at a hotel they have visited before.
- Attracting new customers is tedious and costs more as compared to a repeated guest.

In [72]:

```
# grouping the data on arrival months and extracting the count of bookings
monthly_data = df.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()
```

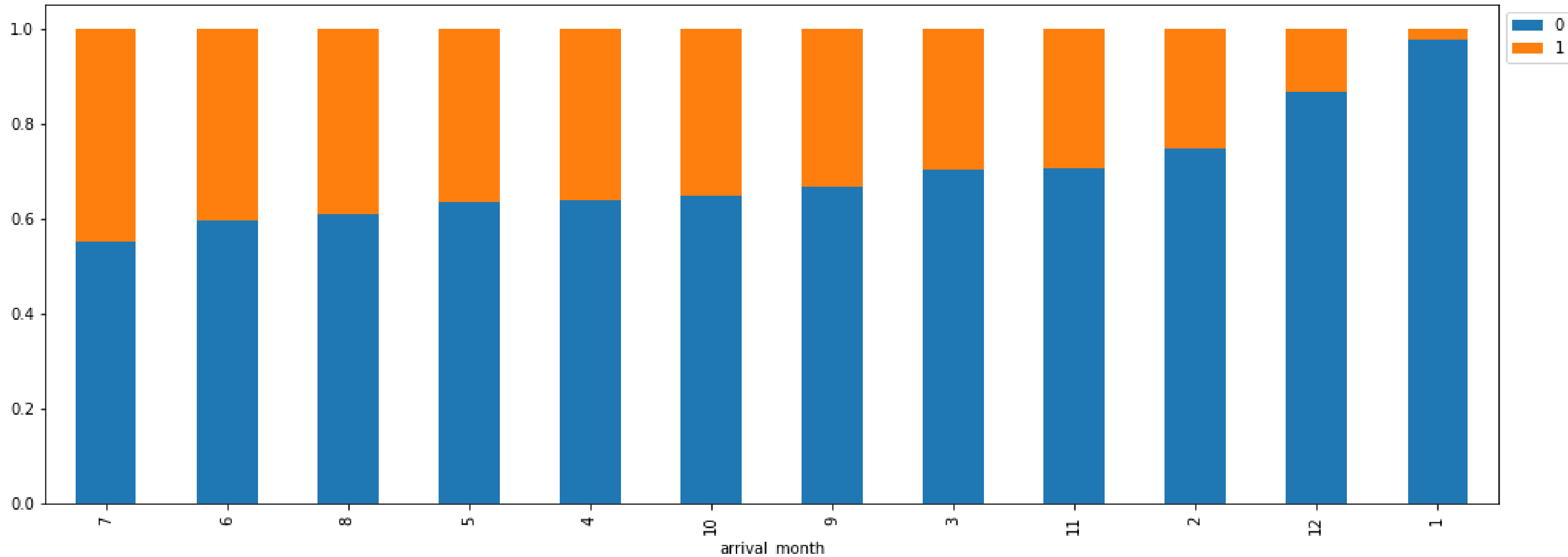


- The trend shows the number of bookings remains consistent from April to July and the hotel sees around 3000 to 3500 guests.
- Most bookings were made in October - more than 5000 bookings.
- Least bookings were made in January - around 1000 bookings.

In [73]:

```
stack(df, "arrival_month", "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
------------------------------------------------------------------------------------------------------------------------
```

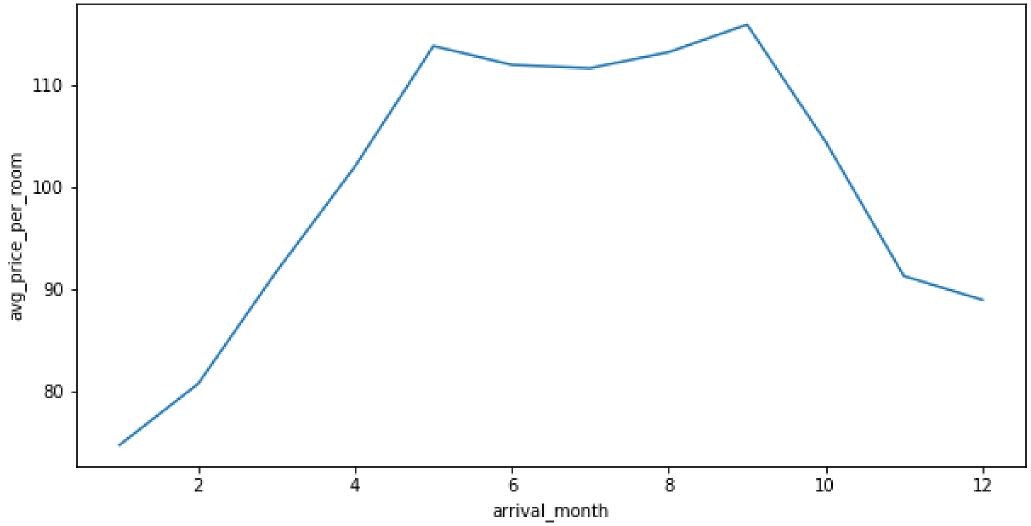


- We see that even though the highest number of bookings were made in September and October - around 40% of these bookings got canceled.
- Least bookings were canceled in December and January - customers might have traveled to celebrate Christmas and New Year.

As hotel room prices are dynamic, Let's see how the prices vary across different months.

In [74]:

```
plt.figure(figsize=(10, 5))
sns.lineplot(y=df["avg_price_per_room"], x=df["arrival_month"], ci=None)
plt.show()
```



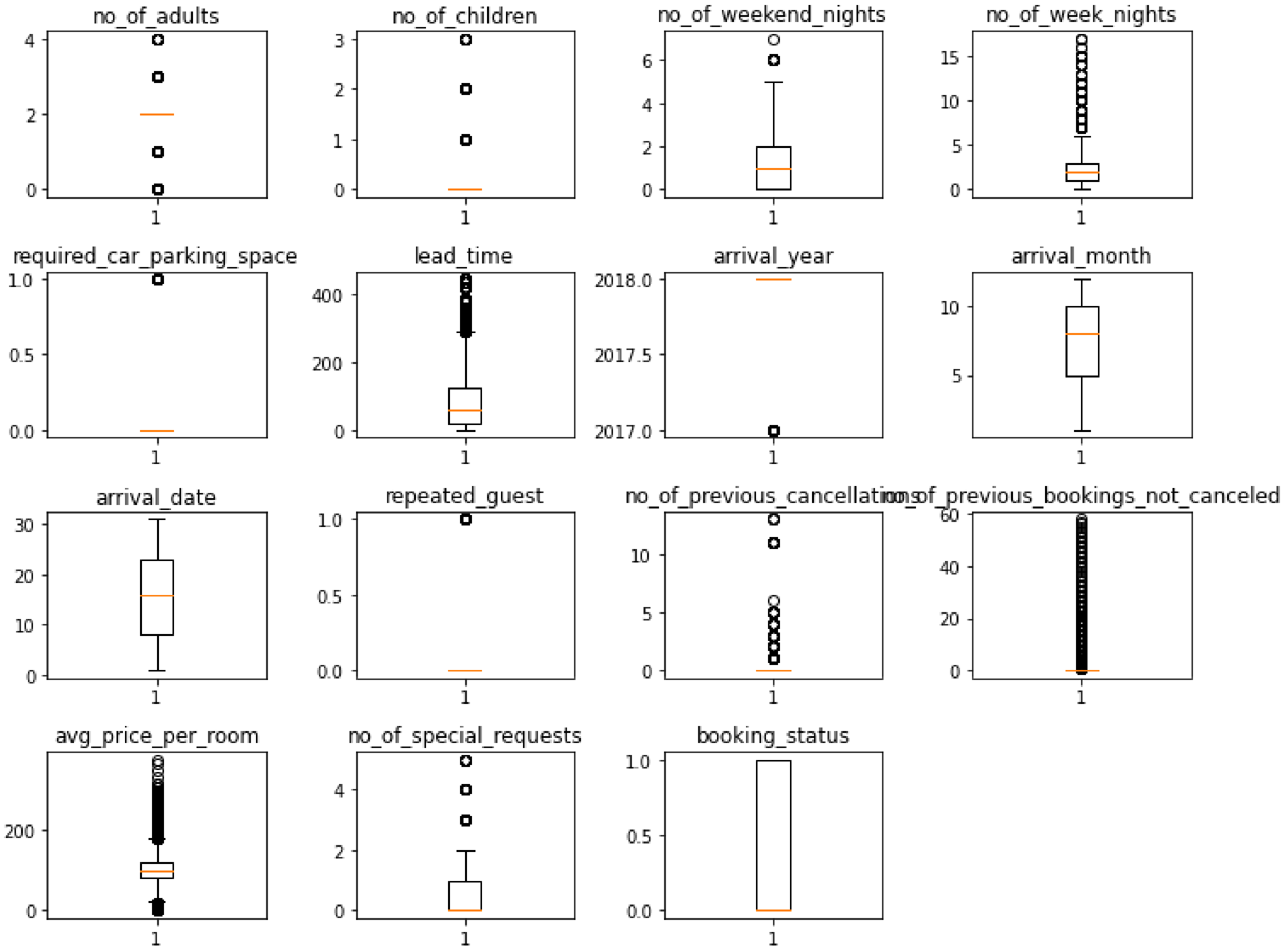
- The price of rooms is highest in May to September - around 115 euros per room.

Data Preprocessing

- Missing value treatment (not needed, no missing values)
- Feature engineering (if needed)
- Outlier detection and treatment
- Preparing data for modeling
- Any other preprocessing steps (if needed)

Outlier Detection and Treatment

In [75]:
```
# outlier detection using boxplot

num_cols = df.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(10, 8))

for i, variable in enumerate(num_cols):
    plt.subplot(4, 4, i + 1)
    plt.boxplot(df[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```



Observations:

- There are quite a few outliers in the data, notably in average room price and lead time.
- However, since they are proper values and reflect the market, we will not treat them.

Data Preparation for modeling

- From this model, want to predict which bookings will be cancelled.
- 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.

In [76]:

```
# defining the dependent and independent variables

X = df.drop(["booking_status"], axis=1)
y = df["booking_status"]
```

```
print(X.head())
print()
print(y.head())
```

```
   no_of_adults  no_of_children  no_of_weekend_nights  no_of_week_nights  \
0             2               0                     1                  2   
1             2               0                     2                  3   
2             1               0                     2                  1   
3             2               0                     0                  2   
4             2               0                     1                  1   

  type_of_meal_plan  required_car_parking_space room_type_reserved  lead_time  \
0       Meal Plan 1                           0        Room_Type 1        224   
1      Not Selected                           0        Room_Type 1          5   
2       Meal Plan 1                           0        Room_Type 1          1   
3       Meal Plan 1                           0        Room_Type 1        211   
4      Not Selected                           0        Room_Type 1         48   

   arrival_year  arrival_month  arrival_date market_segment_type  \
0          2017             10             2             Offline   
1          2018             11             6              Online   
2          2018              2            28              Online   
3          2018              5            20              Online   
4          2018              4            11              Online   

   repeated_guest  no_of_previous_cancellations  \
0               0                             0   
1               0                             0   
2               0                             0   
3               0                             0   
4               0                             0   

   no_of_previous_bookings_not_canceled  avg_price_per_room  \
0                                     0            65.00000   
1                                     0           106.68000   
2                                     0            60.00000   
3                                     0           100.00000   
4                                     0            94.50000   

   no_of_special_requests  
0                       0  
1                       1  
2                       0  
3                       0  
4                       0  

0    0
1    0
2    1
3    1
4    1
Name: booking_status, dtype: int64
```

In [77]:

```
# create dummy variables
```

```
X = pd.get_dummies(X, columns=X.select_dtypes(include=["object", "category"]).columns.tolist(), drop_first=True)

X.head()
```

Out[77]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	required_car_parking_space	lead_time	arrival_year	arrival_month	arrival_date	rep
0	2	0	1	2	0	224	2017	10	2	
1	2	0	2	3	0	5	2018	11	6	
2	1	0	2	1	0	1	2018	2	28	
3	2	0	0	2	0	211	2018	5	20	
4	2	0	1	1	0	48	2018	4	11	

In [78]:

```
# splitting the data in 70:30 ratio for train to test data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [79]:

```
print("Number of rows in train data =", X_train.shape[0])
print("Number of rows in test data =", X_test.shape[0])

print("Percentage of classes in training set:")
print(y_train.value_counts(normalize=True))
print("Percentage of classes in test set:")
print(y_test.value_counts(normalize=True))
```

```
Number of rows in train data = 25392
Number of rows in test data = 10883
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
```

Building Our Logistic Regression Model

Model evaluation criterion

Model can make wrong predictions as:

1. Predicting a customer will not cancel their booking but in reality, the customer will cancel their booking.
2. Predicting a customer will cancel their booking but in reality, the customer will not cancel their booking.

Which case is more important?

- Both the cases are important as:

- If we predict that a booking will not be canceled and the booking gets canceled (False Negative) then the hotel will lose resources and will have to bear additional costs of distribution channels.

- If we predict that a booking will get canceled and the booking doesn't get canceled (False Positive) the hotel might not be able to provide satisfactory services to the customer by assuming that this booking will be canceled. This might damage the brand equity.

How to reduce the losses?

- `f1_score` should be maximized, the greater the f1_score higher the chances of identifying both the cases correctly.

First, let's create functions to calculate different metrics and confusion matrix so that we don't have to use the same code repeatedly for each model.

- The model_performance_classification_statsmodels function will be used to check the model performance of models.
- The confusion_matrix_statsmodels function will be used to plot confusion matrix.

In [80]:

```
# defining a function to compute different metrics to check performance of a classification model built using statsmodels
def model_performance_classification_statsmodels(
    model, predictors, target, threshold=0.5
):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    threshold: threshold for classifying the observation as class 1
    """

    # checking which probabilities are greater than threshold
    pred_temp = model.predict(predictors) > threshold
    # rounding off the above values to get classes
    pred = np.round(pred_temp)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
        index=[0],
    )
```

```
    return df_perf
```

In [81]:

```
# defining a function to plot the confusion_matrix of a classification model


def confusion_matrix_statsmodels(model, predictors, target, threshold=0.5):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    threshold: threshold for classifying the observation as class 1
    """
    y_pred = model.predict(predictors) > threshold
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

In [82]:

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

# adding constant
X = sm.add_constant(X)

X = pd.get_dummies(X, drop_first=True)

# 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
)
```

In [83]:

```
# fitting logistic regression model
logit = sm.Logit(y_train, X_train.astype(float))
lg = logit.fit(disp=False)

print(lg.summary())
```

```
                           Logit Regression Results
```

```
==============================================================================
Dep. Variable:         booking_status   No. Observations:                25392
Model:                          Logit   Df Residuals:                    25364
Method:                           MLE   Df Model:                           27
Date:                Fri, 21 Jan 2022   Pseudo R-squ.:                  0.3292
Time:                        01:06:15   Log-Likelihood:                -10794.
converged:                      False   LL-Null:                       -16091.
Covariance Type:            nonrobust   LLR p-value:                     0.000
========================================================================================================
                                           coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------
const                                 -922.8266    120.832     -7.637      0.000   -1159.653    -686.000
no_of_adults                             0.1137      0.038      3.019      0.003       0.040       0.188
no_of_children                           0.1580      0.062      2.544      0.011       0.036       0.280
no_of_weekend_nights                     0.1067      0.020      5.395      0.000       0.068       0.145
no_of_week_nights                        0.0397      0.012      3.235      0.001       0.016       0.064
required_car_parking_space              -1.5943      0.138    -11.565      0.000      -1.865      -1.324
lead_time                                0.0157      0.000     58.863      0.000       0.015       0.016
arrival_year                             0.4561      0.060      7.617      0.000       0.339       0.573
arrival_month                           -0.0417      0.006     -6.441      0.000      -0.054      -0.029
arrival_date                             0.0005      0.002      0.259      0.796      -0.003       0.004
repeated_guest                          -2.3472      0.617     -3.806      0.000      -3.556      -1.139
no_of_previous_cancellations             0.2664      0.086      3.108      0.002       0.098       0.434
no_of_previous_bookings_not_canceled    -0.1727      0.153     -1.131      0.258      -0.472       0.127
avg_price_per_room                       0.0188      0.001     25.396      0.000       0.017       0.020
no_of_special_requests                  -1.4689      0.030    -48.782      0.000      -1.528      -1.410
type_of_meal_plan_Meal Plan 2            0.1756      0.067      2.636      0.008       0.045       0.306
type_of_meal_plan_Meal Plan 3           17.3584   3987.813      0.004      0.997   -7798.612    7833.329
type_of_meal_plan_Not Selected           0.2784      0.053      5.247      0.000       0.174       0.382
room_type_reserved_Room_Type 2          -0.3605      0.131     -2.748      0.006      -0.618      -0.103
room_type_reserved_Room_Type 3          -0.0012      1.310     -0.001      0.999      -2.568       2.566
room_type_reserved_Room_Type 4          -0.2823      0.053     -5.304      0.000      -0.387      -0.178
room_type_reserved_Room_Type 5          -0.7189      0.209     -3.438      0.001      -1.129      -0.309
room_type_reserved_Room_Type 6          -0.9501      0.151     -6.274      0.000      -1.247      -0.653
room_type_reserved_Room_Type 7          -1.4003      0.294     -4.770      0.000      -1.976      -0.825
market_segment_type_Complementary      -40.5975   5.65e+05  -7.19e-05      1.000   -1.11e+06    1.11e+06
market_segment_type_Corporate           -1.1924      0.266     -4.483      0.000      -1.714      -0.671
market_segment_type_Offline             -2.1946      0.255     -8.621      0.000      -2.694      -1.696
market_segment_type_Online              -0.3995      0.251     -1.590      0.112      -0.892       0.093
========================================================================================================
```

In [84]:

```
print("Training performance:")
model_performance_classification_statsmodels(lg, X_train, y_train)
```

Training performance:

Out[84]:

	Accuracy	Recall	Precision	F1
0	0.80600	0.63410	0.73971	0.68285

Observations:

- Negative values of the coefficient show that the probability of customers canceling the booking decreases with the increase of the corresponding attribute value.

- Positive values of the coefficient show that the probability of customer canceling increases with the increase of corresponding attribute value.
- p-value of a variable indicates if the variable is significant or not. If we consider the significance level to be 0.05 (5%), then any variable with a p-value less than 0.05 would be considered significant.
- But these variables might contain multicollinearity, which will affect the p-values.
- We will have to remove multicollinearity from the data to get reliable coefficients and p-values.
- There are different ways of detecting (or testing) multi-collinearity, one such way is the Variation Inflation Factor.

Checking Multicolinearity

In [85]:

```
# we will define a function to check VIF
def checking_vif(predictors):
    vif = pd.DataFrame()
    vif["feature"] = predictors.columns

    # calculating VIF for each feature
    vif["VIF"] = [
        variance_inflation_factor(predictors.values, i)
        for i in range(len(predictors.columns))
    ]
    return vif
```

In [86]:

```
checking_vif(X_train)
```

Out[86]:

	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

	feature	VIF
12	no_of_previous_bookings_not_canceled	1.65200
13	avg_price_per_room	2.06860
14	no_of_special_requests	1.24798
15	type_of_meal_plan_Meal Plan 2	1.27328
16	type_of_meal_plan_Meal Plan 3	1.02526
17	type_of_meal_plan_Not Selected	1.27306
18	room_type_reserved_Room_Type 2	1.10595
19	room_type_reserved_Room_Type 3	1.00330
20	room_type_reserved_Room_Type 4	1.36361
21	room_type_reserved_Room_Type 5	1.02800
22	room_type_reserved_Room_Type 6	2.05614
23	room_type_reserved_Room_Type 7	1.11816
24	market_segment_type_Complementary	4.50276
25	market_segment_type_Corporate	16.92829
26	market_segment_type_Offline	64.11564
27	market_segment_type_Online	71.18026

- Ignore multicolinearity of dummy variables. For numeric variables, no multicolinearity present.

Drop High p-value variables

- We will drop the predictor variables having a p-value greater than 0.05 as they do not significantly impact the target variable.
- But sometimes p-values change after dropping a variable. So, we'll not drop all variables at once.
- Instead, we will do the following:
  - Build a model, check the p-values of the variables, and drop the column with the highest p-value.
  - Create a new model without the dropped feature, check the p-values of the variables, and drop the column with the highest p-value.
  - Repeat the above two steps till there are no columns with p-value > 0.05.
- The above process can also be done manually by picking one variable at a time that has a high p-value, dropping it, and building a model again. But that might be a little tedious and using a loop will be more efficient.

In [87]:
```
# initial list of columns
cols = X_train.columns.tolist()

# setting an initial max p-value
max_p_value = 1
```

```
while len(cols) > 0:
    # defining the train set
    X_train_aux = X_train[cols]

    # fitting the model
    model = sm.Logit(y_train, X_train_aux).fit(disp=False)

    # getting the p-values and the maximum p-value
    p_values = model.pvalues
    max_p_value = max(p_values)

    # name of the variable with maximum p-value
    feature_with_p_max = p_values.idxmax()

    if max_p_value > 0.05:
        cols.remove(feature_with_p_max)
    else:
        break

selected_features = cols
print(selected_features)
```

```
['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', 'no_of_previous_cancellations', 'avg_price_per_room', 'no_of_special_requests', 'type_of_meal_plan_Meal Plan 2', 'type_of_meal_plan_Not Selected', 'room_type_reserved_Room_Type 2', 'room_type_reserved_Room_Type 4', 'room_type_reserved_Room_Type 5', 'room_type_reserved_Room_Type 6', 'room_type_reserved_Room_Type 7', 'market_segment_type_Corporate', 'market_segment_type_Offline']
```

In [88]:

```
X_train2 = X_train[selected_features]
X_test2 = X_test[selected_features]
```

In [89]:

```
logit2 = sm.Logit(y_train, X_train2.astype(float))
lg2 = logit2.fit(disp=False)

print(lg2.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:         booking_status   No. Observations:                25392
Model:                          Logit   Df Residuals:                    25370
Method:                           MLE   Df Model:                           21
Date:                Fri, 21 Jan 2022   Pseudo R-squ.:                  0.3282
Time:                        01:06:19   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
==================================================================================================
```

In [90]:

```python
print("Training performance:")
model_performance_classification_statsmodels(lg2, X_train2, y_train)
```

Training performance:

Out[90]:

	Accuracy	Recall	Precision	F1
0	0.80545	0.63267	0.73907	0.68174

Now no feature has p-value greater than 0.05, so we'll consider the features in *X_train2* as the final ones and *lg2* as final model. Performance on training data is similar to before dropping high p-value variables.

Coefficient interpretations

- Coefficients of required_car_parking_space, arrival_month, repeated_guest, no_of_special_requests, room types, and market segment types are negative, an increase in these will lead to a decrease in chances of a customer canceling their booking.
- Coefficients of no_of_adults, no_of_children, no_of_weekend_nights, no_of_week_nights, lead_time, avg_price_per_room, meal plan 2 and not selected, and some others are positive, an increase in these will lead to a increase in the chances of a customer canceling their booking.

Converting coefficients to odds

- The coefficients of the logistic regression model are in terms of log(odd), to find the odds we have to take the exponential of the coefficients.
- Therefore, **odds = exp(b)**
- The percentage change in odds is given as **odds = (exp(b) - 1) * 100**

In [91]:

```python
# converting coefficients to odds
odds = np.exp(lg2.params)

# finding the percentage change
```

```
perc_change_odds = (np.exp(lg2.params) - 1) * 100

# removing limit from number of columns to display
pd.set_option("display.max_columns", None)

# adding the odds to a dataframe
pd.DataFrame({"Odds": odds, "Change_odd%": perc_change_odds}, index=X_train2.columns).T
```

Out[91]:

	const	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	required_car_parking_space	lead_time	arrival_year	arrival_
Odds	0.00000	1.11491	1.16546	1.11470	1.04258	0.20296	1.01583	1.57195	0
Change_odd%	-100.00000	11.49096	16.54593	11.46966	4.25841	-79.70395	1.58331	57.19508	-(

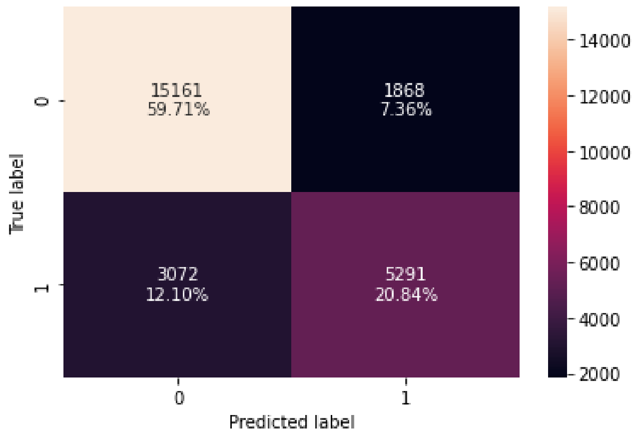
Coefficient interpretations

- `no_of_adults` : Holding all other features constant a 1 unit change in the number of children will increase the odds of a booking getting cancelled by 1.11 times or a 11.49% increase in the odds of a booking getting cancelled.
- `no_of_children` : Holding all other features constant a 1 unit change in the number of children will increase the odds of a booking getting cancelled by 1.16 times or a 16.54% increase in the odds of a booking getting cancelled.
- `no_of_weekend_nights` : Holding all other features constant a 1 unit change in the number of weeknights a customer stays at the hotel will increase the odds of a booking getting cancelled by 1.11 times or a 11.46% increase in the odds of a booking getting cancelled.
- `no_of_week_nights` : Holding all other features constant a 1 unit change in the number of weeknights a customer stays at the hotel will increase the odds of a booking getting cancelled by 1.04 times or a 4.25% increase in the odds of a booking getting cancelled.
- `required_car_parking_space` : The odds of a customer who requires a car parking space are 0.2 times less than a customer who doesn't require a car parking space or a 79.70% fewer odds of a customer canceling their booking.
- `lead_time` : Holding all other features constant a 1 unit change in the lead time will increase the odds of a booking getting cancelled by 1.01 times or a 1.58% increase in the odds of a booking getting cancelled.
- `no_of_special_requests` : Holding all other features constant a 1 unit change in the number of special requests made by the customer will decrease the odds of a booking getting cancelled by 0.22 times or a 77% decrease in the odds of a booking getting cancelled.
- `avg_price_per_room` : Holding all other features constant a 1 unit change in the lead time will increase the odds of a booking getting cancelled by 1.01 times or a 1.93% increase in the odds of a booking getting cancelled.
- `type_of_meal_plan_Not Selected` : The odds of a customer who has not selected any meal plan cancelling the booking are 1.33 times more than a customer who has selected a meal plan or a 33.10% higher odds of a booking getting cancelled if a meal plan is not selected. [keeping all the other meal plan types as reference]

`Interpretation for other attributes follows the same pattern.`

Model Performance Evaluation

In [92]:
```
# creating confusion matrix
confusion_matrix_statsmodels(lg2, X_train2, y_train)
```



In [93]:
```
print("Training performance:")
log_reg_model_train_perf = model_performance_classification_statsmodels(
    lg2, X_train2, y_train
)
log_reg_model_train_perf
```

Training performance:

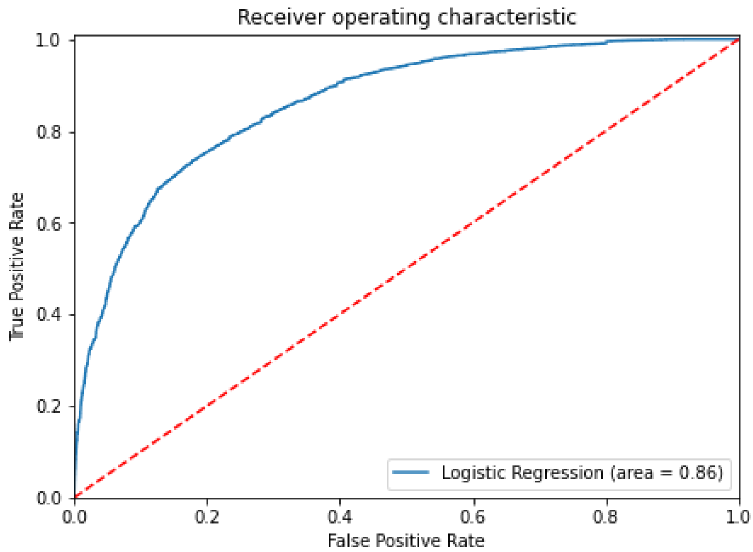
Out[93]:

	Accuracy	Recall	Precision	F1
0	0.80545	0.63267	0.73907	0.68174

ROC-AUC

In [94]:
```
# ROC-AUC on training set

logit_roc_auc_train = roc_auc_score(y_train, lg2.predict(X_train2))
fpr, tpr, thresholds = roc_curve(y_train, lg2.predict(X_train2))
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, label="Logistic Regression (area = %0.2f)" % logit_roc_auc_train)
plt.plot([0, 1], [0, 1], "r--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.01])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
```

```
plt.legend(loc="lower right")
plt.show()
```



- Logistic Regression model is giving a generalized performance on training and test set.
- ROC-AUC score of 0.86 on training is quite good.

Model Performance Improvement

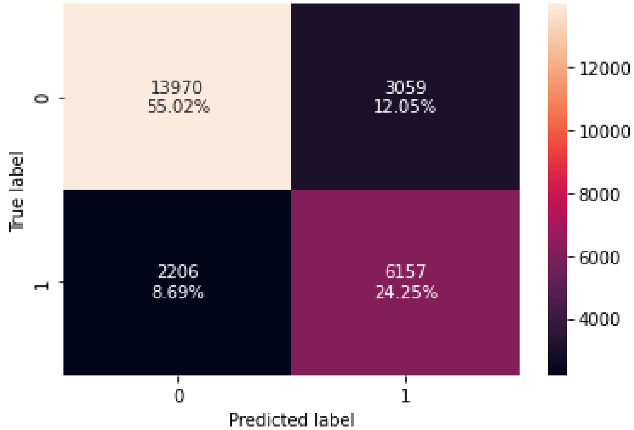
- Let's see if the f1 and recall score can be improved further, by changing the model threshold using AUC-ROC Curve.

Optimal threshold using AUC-ROC curve

In [95]:
```
# Optimal cut off would be where tpr is high and fpr is low

fpr, tpr, thresholds = roc_curve(y_train, lg2.predict(X_train2))

optimal_idx = np.argmax(tpr - fpr)
optimal_threshold_auc_roc = thresholds[optimal_idx]
print(optimal_threshold_auc_roc)
```

```
0.37005225587082935
```

In [96]:
```
# creating confusion matrix
confusion_matrix_statsmodels(
    lg2, X_train2, y_train, threshold=optimal_threshold_auc_roc
)
```



In [97]:
```
# checking model performance for this model
log_reg_model_train_perf_threshold_auc_roc = model_performance_classification_statsmodels(
    lg2, X_train2, y_train, threshold=optimal_threshold_auc_roc
)
print("Training performance:")
log_reg_model_train_perf_threshold_auc_roc
```

```
Training performance:
```

Out[97]:

	Accuracy	Recall	Precision	F1
0	0.79265	0.73622	0.66808	0.70049

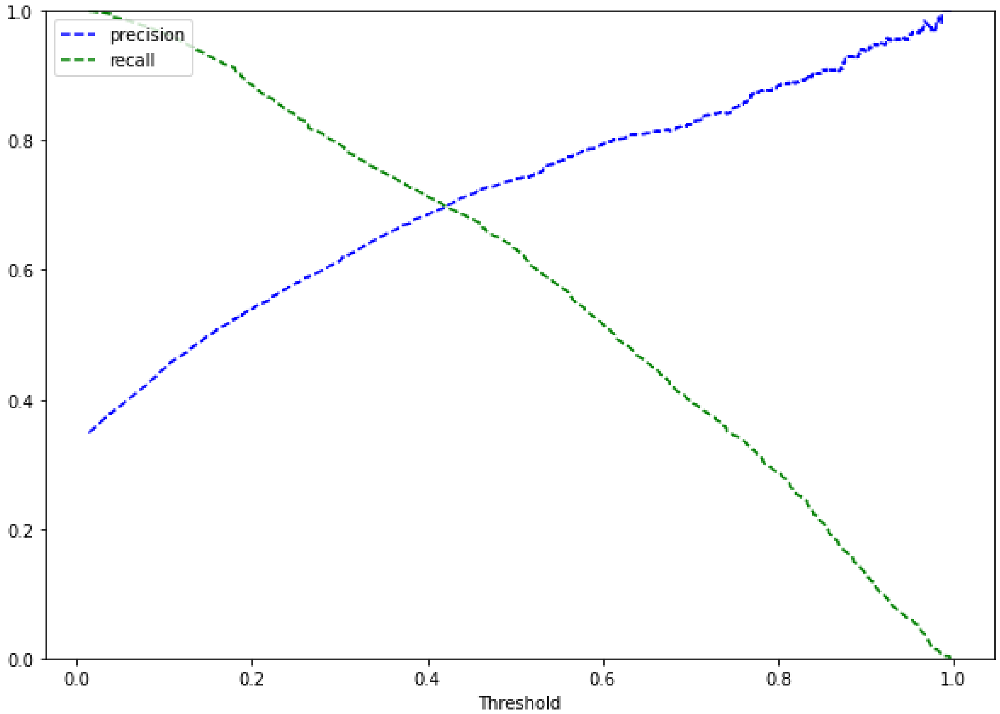
- Recall and f1 have improved, other metrics have reduced slightly.
- Model still giving good performance.

- As we will decrease the threshold value, Recall will keep on increasing and the Precision will decrease, but this is not right, we need to choose an optimal balance between recall and precision.

Possible Better threshold using Precision-Recall curve

In [98]:
```
y_scores = lg2.predict(X_train2)
prec, rec, tre = precision_recall_curve(y_train, y_scores,)


def plot_prec_recall_vs_tresh(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="precision")
    plt.plot(thresholds, recalls[:-1], "g--", label="recall")
    plt.xlabel("Threshold")
    plt.legend(loc="upper left")
    plt.ylim([0, 1])
```

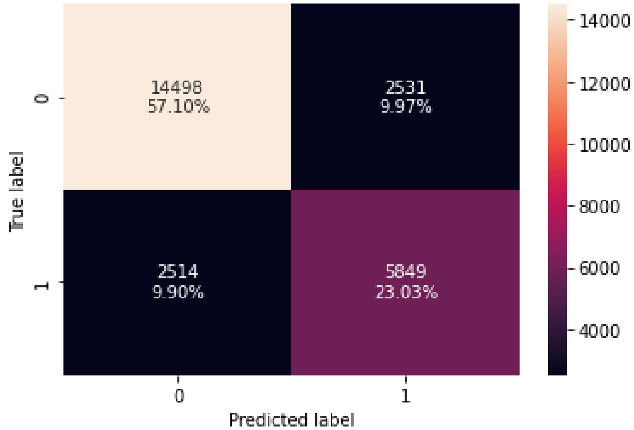
```
plt.figure(figsize=(10, 7))
plot_prec_recall_vs_tresh(prec, rec, tre)
plt.show()
```



- At 0.42 threshold we get a balanced precision and recall.

In [99]:
```
# setting the threshold
optimal_threshold_curve = 0.42
```

In [100…
```
# creating confusion matrix
confusion_matrix_statsmodels(lg2, X_train2, y_train, threshold=optimal_threshold_curve)
```



In [101…

```
log_reg_model_train_perf_threshold_curve = model_performance_classification_statsmodels(
    lg2, X_train2, y_train, threshold=optimal_threshold_curve
)
print("Training performance:")
log_reg_model_train_perf_threshold_curve
```

```
Training performance:
```

Out[101…

	Accuracy	Recall	Precision	F1
0	0.80132	0.69939	0.69797	0.69868

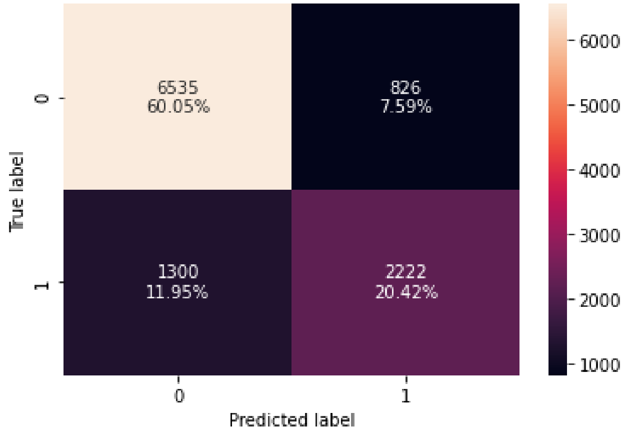
- Model performance has improved as compared to our initial model.
- Model has given a balanced performance in terms of precision and recall.

Let's check the performance on the test set

Using model with default threshold

In [102…

```
# creating confusion matrix
confusion_matrix_statsmodels(lg2, X_test2, y_test)
```



In [103…

```
log_reg_model_test_perf = model_performance_classification_statsmodels(
    lg2, X_test2, y_test
)

print("Test performance:")
log_reg_model_test_perf
```

Test performance:

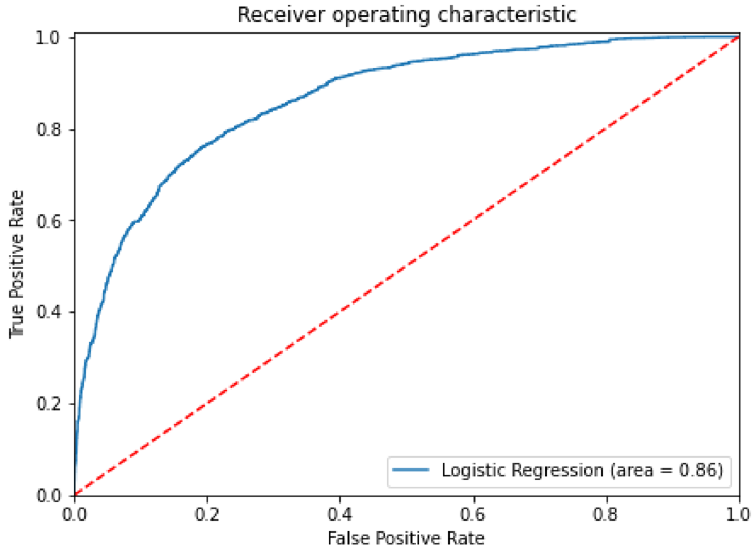
Out[103…

	Accuracy	Recall	Precision	F1
0	0.80465	0.63089	0.72900	0.67641

ROC curve on test set

In [104…

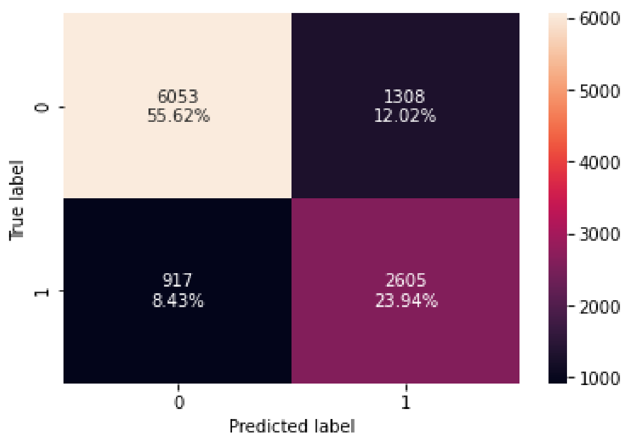
```
logit_roc_auc_train = roc_auc_score(y_test, lg2.predict(X_test2))
fpr, tpr, thresholds = roc_curve(y_test, lg2.predict(X_test2))
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, label="Logistic Regression (area = %0.2f)" % logit_roc_auc_train)
plt.plot([0, 1], [0, 1], "r--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.01])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")
plt.show()
```



Using model with threshold=0.37

In [105…

```
# creating confusion matrix
confusion_matrix_statsmodels(lg2, X_test2, y_test, threshold=optimal_threshold_auc_roc)
```



In [106…

```
# checking model performance for this model
log_reg_model_test_perf_threshold_auc_roc = model_performance_classification_statsmodels(
    lg2, X_test2, y_test, threshold=optimal_threshold_auc_roc
)
print("Test performance:")
log_reg_model_test_perf_threshold_auc_roc
```

```
Test performance:
```

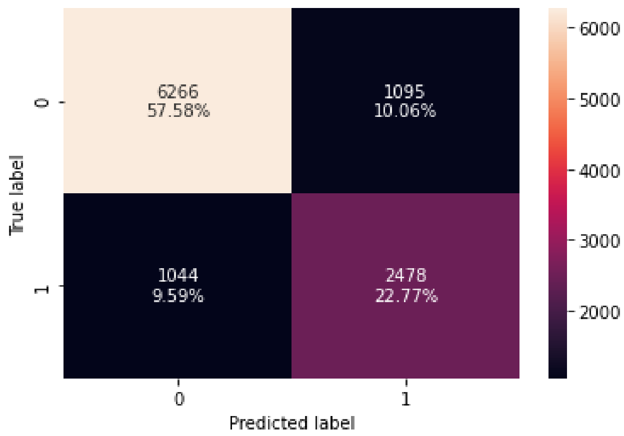
Out[106…

	Accuracy	Recall	Precision	F1
0	0.79555	0.73964	0.66573	0.70074

Using model with threshold=0.42

In [107…

```
# creating confusion matrix
confusion_matrix_statsmodels(lg2, X_test2, y_test, threshold=optimal_threshold_curve)
```



In [108…

```
log_reg_model_test_perf_threshold_curve = model_performance_classification_statsmodels(
    lg2, X_test2, y_test, threshold=optimal_threshold_curve
)
print("Test performance:")
log_reg_model_test_perf_threshold_curve
```

```
Test performance:
```

Out[108…

	Accuracy	Recall	Precision	F1
0	0.80345	0.70358	0.69353	0.69852

Model Performance Summary

In [109…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        log_reg_model_train_perf.T,
        log_reg_model_train_perf_threshold_auc_roc.T,
        log_reg_model_train_perf_threshold_curve.T,
    ],
    axis=1,
)
```

```
models_train_comp_df.columns = [
    "Logistic Regression-default Threshold",
    "Logistic Regression-0.37 Threshold",
    "Logistic Regression-0.42 Threshold",
]

print("Training performance comparison:")
models_train_comp_df
```

Training performance comparison:

Out[109…

	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

In [110…

```
# 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("Test performance comparison:")
models_test_comp_df
```

Test performance comparison:

Out[110…

	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

Observations from Logistic Regression model

- We have been able to build a predictive model that can be used by the hotel to predict which bookings are likely to be cancelled with an F1 score of 0.69 on the training set and formulate marketing policies accordingly.
- The logistic regression models are giving a generalized performance on training and test set.
- **Using the model with default threshold** the model will give a low recall but good precision score - The hotel will be able to predict which bookings will not be cancelled and will be able to provide satisfactory services to those customers which help in maintaining the brand equity but will lose on resources.
- **Using the model with a 0.37 threshold** the model will give a high recall but low precision score - The hotel will be able to save resources by correctly predicting the bookings which are likely to be cancelled but might damage the brand equity.
- **Using the model with a 0.42 threshold** the model will give a balance recall and precision score - The hotel will be able to maintain a balance between resources and brand equity.
- Coefficients of required_car_parking_space, arrival_month, repeated_guest, no_of_special_requests, room types, and market segment types are negative, an increase in these will lead to a decrease in chances of a customer canceling their booking.
- Coefficients of no_of_adults, no_of_children, no_of_weekend_nights, no_of_week_nights, lead_time, avg_price_per_room, meal plan 2 and not selected, and some others are positive, an increase in these will lead to a increase in the chances of a customer canceling their booking.

Decision Tree

In [111…

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

X = pd.get_dummies(X, columns=X.select_dtypes(include=["object", "category"]).columns.tolist(), drop_first=True)
```

In [112…

```
# splitting data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30, random_state=1)
```

First, let's create functions to calculate different metrics and confusion matrix so that we don't have to use the same code repeatedly for each model.

- The model_performance_classification_sklearn function will be used to check the model performance of models.
- The confusion_matrix_sklearnfunction will be used to plot the confusion matrix.

In [113…

```
# defining a function to compute different metrics to check performance of a classification model built using sklearn
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
```

```
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
        index=[0],
    )

    return df_perf
```

In [114…

```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Building a Decision Tree Model

In [115…

```
model = DecisionTreeClassifier(random_state=1)
model.fit(X_train, y_train)
```

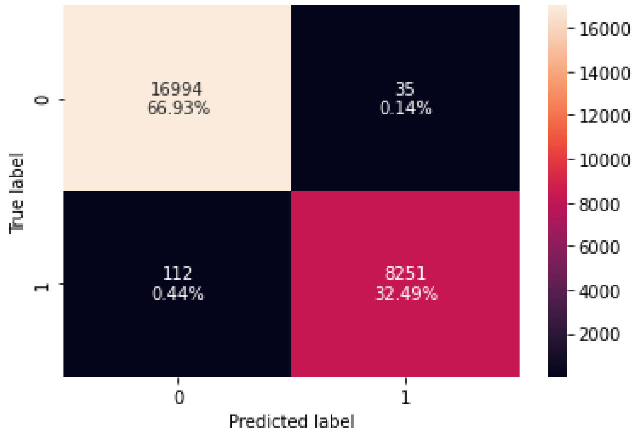
Out[115…

```
DecisionTreeClassifier(random_state=1)
```

Check model performance on training set

In [116…

```
confusion_matrix_sklearn(model, X_train, y_train)
```



In [117…

```
decision_tree_perf_train = model_performance_classification_sklearn(
    model, X_train, y_train
)
decision_tree_perf_train
```

Out[117…

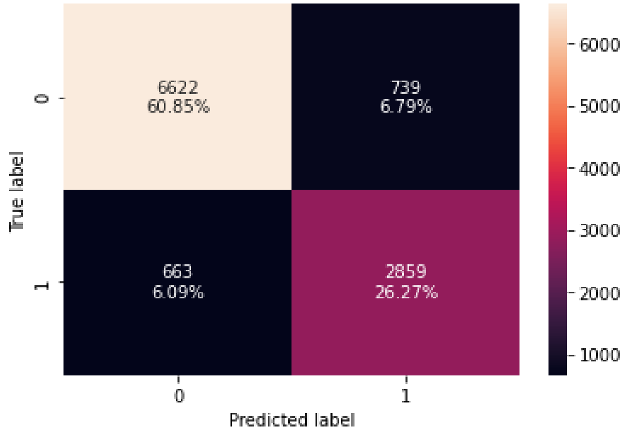
	Accuracy	Recall	Precision	F1
0	0.99421	0.98661	0.99578	0.99117

- Almost 0 errors on the training set, each sample has been classified correctly.
- Model has performed very well on the training set.
- As we know a decision tree will continue to grow and classify each data point correctly if no restrictions are applied as the trees will learn all the patterns in the training set.
- Let's check the performance on test data to see if the model is overfitting.

Check model performance on test set

In [118…

```
confusion_matrix_sklearn(model, X_test, y_test)
```



In [119…

```
decision_tree_perf_test = model_performance_classification_sklearn(
    model, X_test, y_test
)
decision_tree_perf_test
```

Out[119…

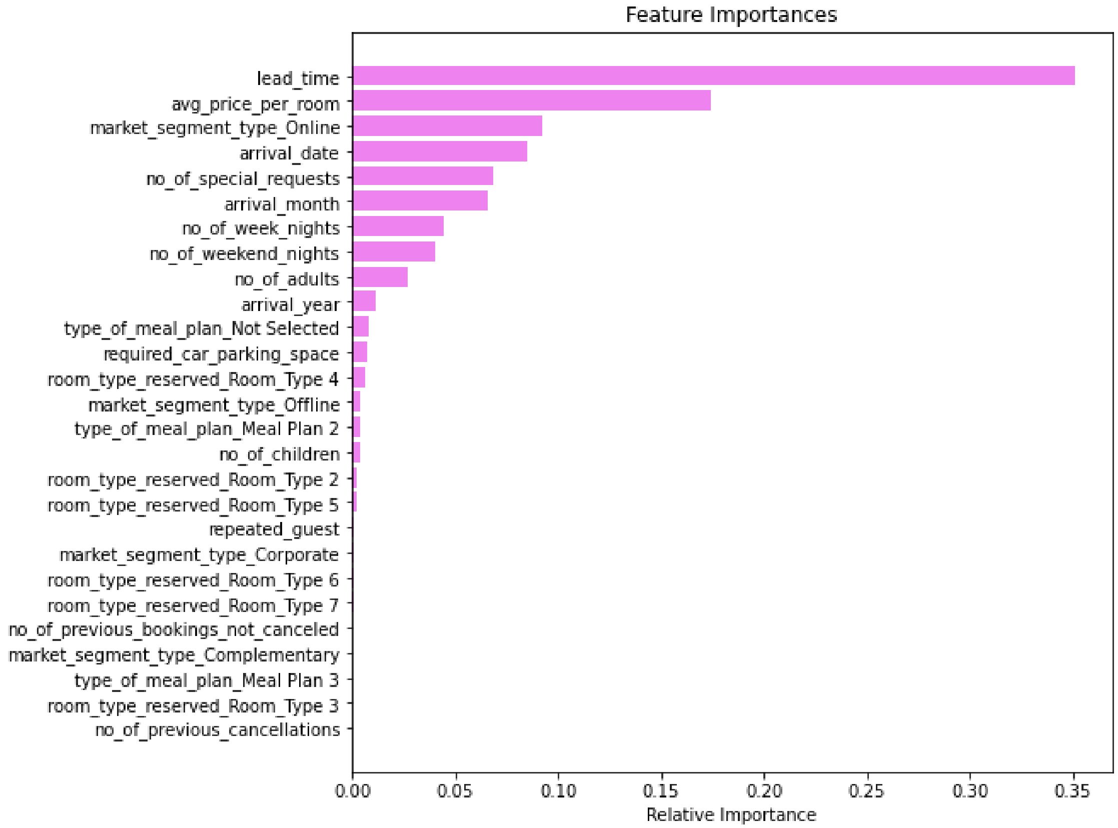
	Accuracy	Recall	Precision	F1
0	0.87118	0.81175	0.79461	0.80309

- The decision tree model is overfitting the data as expected and not able to generalize well on the test set.
- We will have to prune the decision tree.

Before pruning, let's check the important features.

In [120…

```
feature_names = list(X_train.columns)
importances = model.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(8, 8))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Lead time is the most important feature followed by average price per room.
- Now let's prune the tree to see if we can reduce the complexity.

Pruning the Tree

Pre-Pruning

In [121…

```
# 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)
```

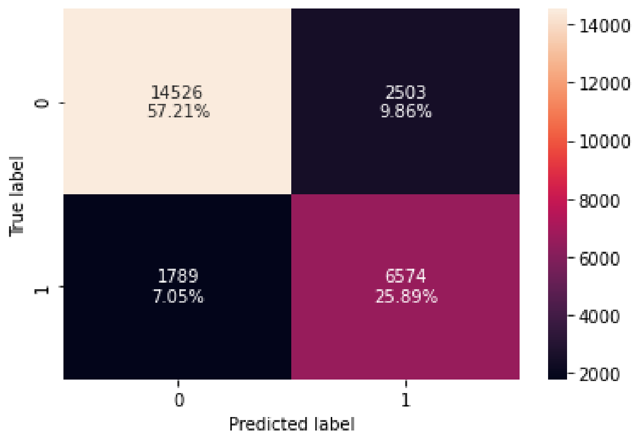
Out[121…

```
DecisionTreeClassifier(class_weight='balanced', max_depth=6, max_leaf_nodes=50,
                       min_samples_split=10, random_state=1)
```

Checking performance on training set

In [122…

```
confusion_matrix_sklearn(estimator, X_train, y_train)
```



In [123…

```
decision_tree_tune_perf_train = model_performance_classification_sklearn(
    estimator, X_train, y_train
)
decision_tree_tune_perf_train
```

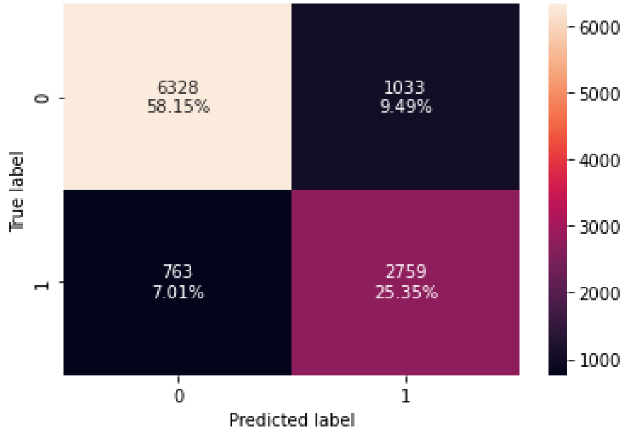
Out[123…

	Accuracy	Recall	Precision	F1
0	0.83097	0.78608	0.72425	0.75390

Checkinng performance on test set

In [124…

```
confusion_matrix_sklearn(estimator, X_test, y_test)
```



In [125…

```
decision_tree_tune_perf_test = model_performance_classification_sklearn(
    estimator, X_test, y_test
)
decision_tree_tune_perf_test
```

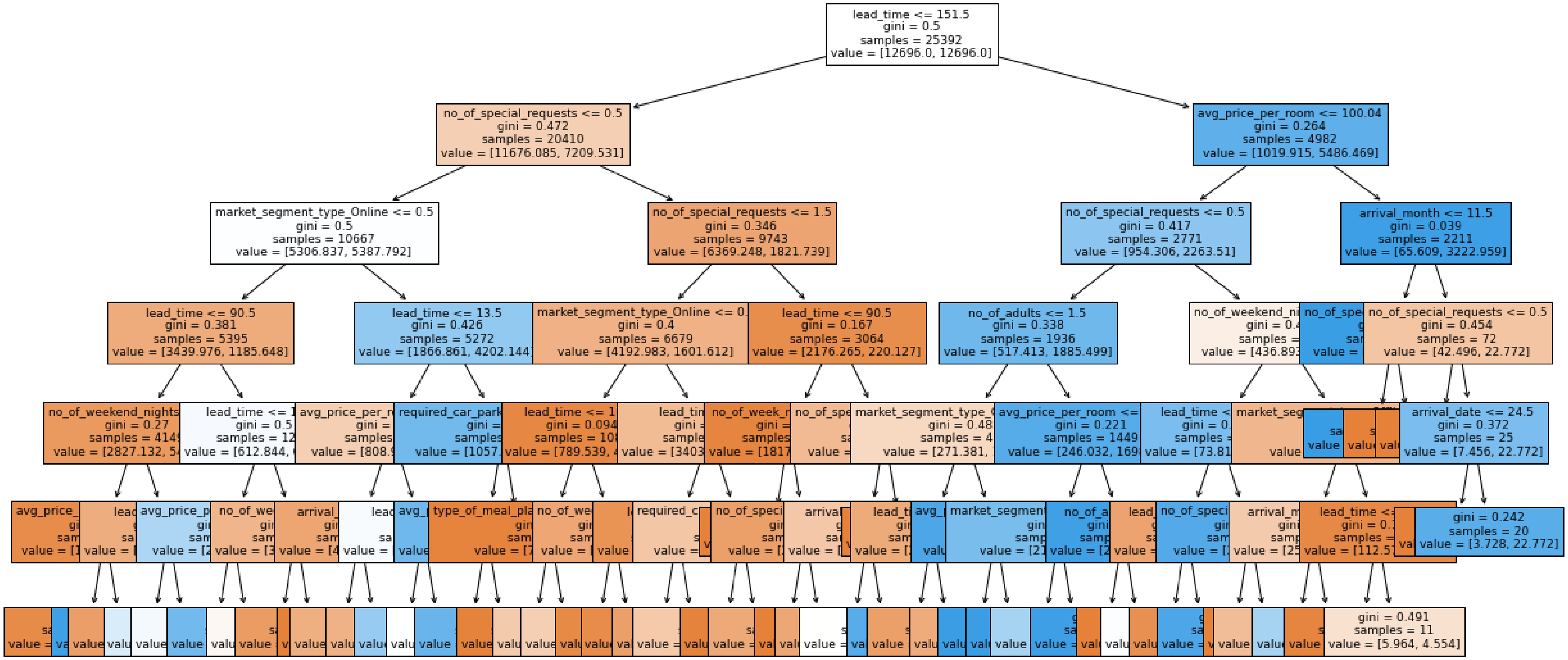
Out[125…

	Accuracy	Recall	Precision	F1
0	0.83497	0.78336	0.72758	0.75444

Visualizing the Decision Tree

In [126…

```
plt.figure(figsize=(20, 10))
out = tree.plot_tree(
    estimator,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
# below code will add arrows to the decision tree split if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```



In [127…

```python
# Text report showing the rules of a decision tree -
print(tree.export_text(estimator, feature_names=feature_names, show_weights=True))
```

```
|--- lead_time <= 151.50
|   |--- no_of_special_requests <= 0.50
|   |   |--- market_segment_type_Online <= 0.50
|   |   |   |--- lead_time <= 90.50
|   |   |   |   |--- no_of_weekend_nights <= 0.50
|   |   |   |   |   |--- avg_price_per_room <= 196.50
|   |   |   |   |   |   |--- weights: [1736.39, 133.59] class: 0
|   |   |   |   |   |--- avg_price_per_room >  196.50
|   |   |   |   |   |   |--- weights: [0.75, 24.29] class: 1
|   |   |   |   |--- no_of_weekend_nights >  0.50
|   |   |   |   |   |--- lead_time <= 68.50
|   |   |   |   |   |   |--- weights: [960.27, 223.16] class: 0
|   |   |   |   |   |--- lead_time >  68.50
|   |   |   |   |   |   |--- weights: [129.73, 160.92] class: 1
|   |   |   |--- lead_time >  90.50
|   |   |   |   |--- lead_time <= 117.50
|   |   |   |   |   |--- avg_price_per_room <= 93.58
|   |   |   |   |   |   |--- weights: [214.72, 227.72] class: 1
|   |   |   |   |   |--- avg_price_per_room >  93.58
|   |   |   |   |   |   |--- weights: [82.76, 285.41] class: 1
```

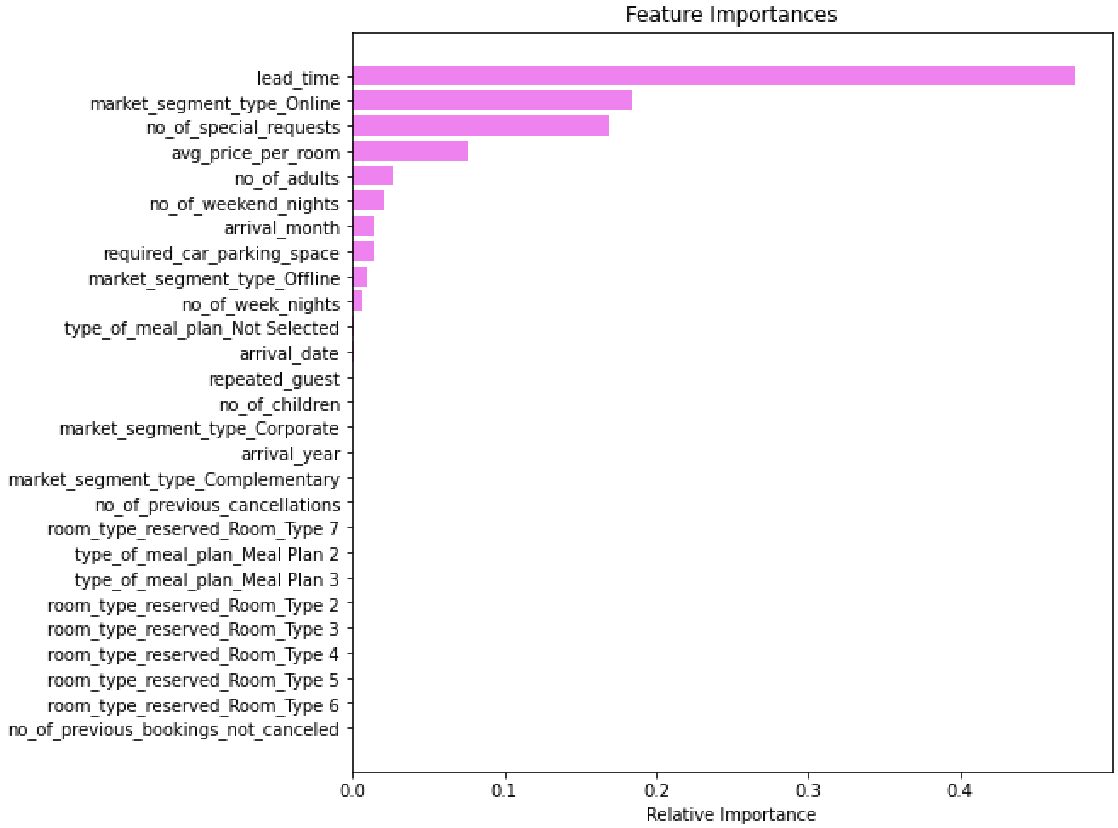
```
|   |   |   |   |--- lead_time >  117.50
|   |   |   |   |   |--- no_of_week_nights <= 1.50
|   |   |   |   |   |   |--- weights: [87.23, 81.98] class: 0
|   |   |   |   |   |--- no_of_week_nights >  1.50
|   |   |   |   |   |   |--- weights: [228.14, 48.58] class: 0
|   |   |--- market_segment_type_Online >  0.50
|   |   |   |--- lead_time <= 13.50
|   |   |   |   |--- avg_price_per_room <= 99.44
|   |   |   |   |   |--- arrival_month <= 1.50
|   |   |   |   |   |   |--- weights: [92.45, 0.00] class: 0
|   |   |   |   |   |--- arrival_month >  1.50
|   |   |   |   |   |   |--- weights: [363.83, 132.08] class: 0
|   |   |   |   |--- avg_price_per_room >  99.44
|   |   |   |   |   |--- lead_time <= 3.50
|   |   |   |   |   |   |--- weights: [219.94, 85.01] class: 0
|   |   |   |   |   |--- lead_time >  3.50
|   |   |   |   |   |   |--- weights: [132.71, 280.85] class: 1
|   |   |   |--- lead_time >  13.50
|   |   |   |   |--- required_car_parking_space <= 0.50
|   |   |   |   |   |--- avg_price_per_room <= 71.92
|   |   |   |   |   |   |--- weights: [158.80, 159.40] class: 1
|   |   |   |   |   |--- avg_price_per_room >  71.92
|   |   |   |   |   |   |--- weights: [850.67, 3543.28] class: 1
|   |   |   |   |--- required_car_parking_space >  0.50
|   |   |   |   |   |--- weights: [48.46, 1.52] class: 0
|   |--- no_of_special_requests >  0.50
|   |   |--- no_of_special_requests <= 1.50
|   |   |   |--- market_segment_type_Online <= 0.50
|   |   |   |   |--- lead_time <= 102.50
|   |   |   |   |   |--- type_of_meal_plan_Not Selected <= 0.50
|   |   |   |   |   |   |--- weights: [697.09, 9.11] class: 0
|   |   |   |   |   |--- type_of_meal_plan_Not Selected >  0.50
|   |   |   |   |   |   |--- weights: [15.66, 9.11] class: 0
|   |   |   |   |--- lead_time >  102.50
|   |   |   |   |   |--- no_of_week_nights <= 2.50
|   |   |   |   |   |   |--- weights: [32.06, 19.74] class: 0
|   |   |   |   |   |--- no_of_week_nights >  2.50
|   |   |   |   |   |   |--- weights: [44.73, 3.04] class: 0
|   |   |   |--- market_segment_type_Online >  0.50
|   |   |   |   |--- lead_time <= 8.50
|   |   |   |   |   |--- lead_time <= 4.50
|   |   |   |   |   |   |--- weights: [498.03, 44.03] class: 0
|   |   |   |   |   |--- lead_time >  4.50
|   |   |   |   |   |   |--- weights: [258.71, 63.76] class: 0
|   |   |   |   |--- lead_time >  8.50
|   |   |   |   |   |--- required_car_parking_space <= 0.50
|   |   |   |   |   |   |--- weights: [2512.51, 1451.32] class: 0
|   |   |   |   |   |--- required_car_parking_space >  0.50
|   |   |   |   |   |   |--- weights: [134.20, 1.52] class: 0
|   |   |--- no_of_special_requests >  1.50
|   |   |   |--- lead_time <= 90.50
|   |   |   |   |--- no_of_week_nights <= 3.50
|   |   |   |   |   |--- weights: [1585.04, 0.00] class: 0
|   |   |   |   |--- no_of_week_nights >  3.50
|   |   |   |   |   |--- no_of_special_requests <= 2.50
|   |   |   |   |   |   |--- weights: [180.42, 57.69] class: 0
|   |   |   |   |   |--- no_of_special_requests >  2.50
|   |   |   |   |   |   |--- weights: [52.19, 0.00] class: 0
```

```
|   |   |   |--- lead_time >  90.50
|   |   |   |   |--- no_of_special_requests <= 2.50
|   |   |   |   |   |--- arrival_month <= 8.50
|   |   |   |   |   |   |--- weights: [184.90, 56.17] class: 0
|   |   |   |   |   |--- arrival_month >  8.50
|   |   |   |   |   |   |--- weights: [106.61, 106.27] class: 0
|   |   |   |   |--- no_of_special_requests >  2.50
|   |   |   |   |   |--- weights: [67.10, 0.00] class: 0
|--- lead_time >  151.50
|   |--- avg_price_per_room <= 100.04
|   |   |--- no_of_special_requests <= 0.50
|   |   |   |--- no_of_adults <= 1.50
|   |   |   |   |--- market_segment_type_Online <= 0.50
|   |   |   |   |   |--- lead_time <= 163.50
|   |   |   |   |   |   |--- weights: [3.73, 24.29] class: 1
|   |   |   |   |   |--- lead_time >  163.50
|   |   |   |   |   |   |--- weights: [257.96, 62.24] class: 0
|   |   |   |   |--- market_segment_type_Online >  0.50
|   |   |   |   |   |--- avg_price_per_room <= 2.50
|   |   |   |   |   |   |--- weights: [8.95, 3.04] class: 0
|   |   |   |   |   |--- avg_price_per_room >  2.50
|   |   |   |   |   |   |--- weights: [0.75, 97.16] class: 1
|   |   |   |--- no_of_adults >  1.50
|   |   |   |   |--- avg_price_per_room <= 82.47
|   |   |   |   |   |--- market_segment_type_Offline <= 0.50
|   |   |   |   |   |   |--- weights: [2.98, 282.37] class: 1
|   |   |   |   |   |--- market_segment_type_Offline >  0.50
|   |   |   |   |   |   |--- weights: [213.97, 385.60] class: 1
|   |   |   |   |--- avg_price_per_room >  82.47
|   |   |   |   |   |--- no_of_adults <= 2.50
|   |   |   |   |   |   |--- weights: [23.86, 1030.80] class: 1
|   |   |   |   |   |--- no_of_adults >  2.50
|   |   |   |   |   |   |--- weights: [5.22, 0.00] class: 0
|   |   |--- no_of_special_requests >  0.50
|   |   |   |--- no_of_weekend_nights <= 0.50
|   |   |   |   |--- lead_time <= 180.50
|   |   |   |   |   |--- lead_time <= 159.50
|   |   |   |   |   |   |--- weights: [7.46, 7.59] class: 1
|   |   |   |   |   |--- lead_time >  159.50
|   |   |   |   |   |   |--- weights: [37.28, 4.55] class: 0
|   |   |   |   |--- lead_time >  180.50
|   |   |   |   |   |--- no_of_special_requests <= 2.50
|   |   |   |   |   |   |--- weights: [20.13, 212.54] class: 1
|   |   |   |   |   |--- no_of_special_requests >  2.50
|   |   |   |   |   |   |--- weights: [8.95, 0.00] class: 0
|   |   |   |--- no_of_weekend_nights >  0.50
|   |   |   |   |--- market_segment_type_Offline <= 0.50
|   |   |   |   |   |--- arrival_month <= 11.50
|   |   |   |   |   |   |--- weights: [231.12, 110.82] class: 0
|   |   |   |   |   |--- arrival_month >  11.50
|   |   |   |   |   |   |--- weights: [19.38, 34.92] class: 1
|   |   |   |   |--- market_segment_type_Offline >  0.50
|   |   |   |   |   |--- lead_time <= 348.50
|   |   |   |   |   |   |--- weights: [106.61, 3.04] class: 0
|   |   |   |   |   |--- lead_time >  348.50
|   |   |   |   |   |   |--- weights: [5.96, 4.55] class: 0
|   |--- avg_price_per_room >  100.04
|   |   |--- arrival_month <= 11.50
```

```
|   |   |   |   |--- no_of_special_requests <= 2.50
|   |   |   |   |   |--- weights: [0.00, 3200.19] class: 1
|   |   |   |   |--- no_of_special_requests >  2.50
|   |   |   |   |   |--- weights: [23.11, 0.00] class: 0
|   |   |   |--- arrival_month >  11.50
|   |   |   |   |--- no_of_special_requests <= 0.50
|   |   |   |   |   |--- weights: [35.04, 0.00] class: 0
|   |   |   |   |--- no_of_special_requests >  0.50
|   |   |   |   |   |--- arrival_date <= 24.50
|   |   |   |   |   |   |--- weights: [3.73, 0.00] class: 0
|   |   |   |   |   |--- arrival_date >  24.50
|   |   |   |   |   |   |--- weights: [3.73, 22.77] class: 1
```

In [128…

```
# importance of features in the tree building

importances = estimator.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(8, 8))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Observations from decision tree

- We can see that the tree has become simpler and the rules of the trees are readable.
- The model performance of the model has been generalized.
- We observe that the most important features are:
  - Lead Time
  - Market Segment - Online
  - Number of special requests
  - Average price per room

The rules obtained from the decision tree can be interpreted as:

- The rules show that lead time plays a key role in identifying if a booking will be cancelled or not. 151 days has been considered as a threshold value by the model to make the first split.

Bookings made more than 151 days before the date of arrival:

- If the average price per room is greater than 100 euros and the arrival month is December, then the the booking is less likely to be cancelled.

- If the average price per room is less than or equal to 100 euros and the number of special request is 0, then the booking is likely to get canceled.

Bookings made under 151 days before the date of arrival:

- If a customer has at least 1 special request the booking is less likely to be cancelled.
- If the customer didn't make any special requests and the booking was done Online it is more likely to get canceled, if the booking was not done online, it is less likely to be canceled.

`If we want more complex then we can go in more depth of the tree`

In [129…

```
for feature in X_train.columns: # Loop through all columns in the dataframe
    if X_train[feature].dtype == 'object': # Only apply for columns with categorical strings
        X_train[feature] = pd.Categorical(X_train[feature])# Replace strings with an integer
X_train.head(5)
```

Out[129…

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	required_car_parking_space	lead_time	arrival_year	arrival_month	arrival_date
13662	1	0	0	1	0	163	2018	10	15
26641	2	0	0	3	0	113	2018	3	31
17835	2	0	2	3	0	359	2018	10	14
21485	2	0	0	3	0	136	2018	6	29
5670	2	0	1	2	0	21	2018	8	15

In [130…

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25392 entries, 13662 to 33003
Data columns (total 27 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   no_of_adults                          25392 non-null  int64
 1   no_of_children                        25392 non-null  int64
 2   no_of_weekend_nights                  25392 non-null  int64
 3   no_of_week_nights                     25392 non-null  int64
 4   required_car_parking_space            25392 non-null  int64
 5   lead_time                             25392 non-null  int64
 6   arrival_year                          25392 non-null  int64
 7   arrival_month                         25392 non-null  int64
 8   arrival_date                          25392 non-null  int64
 9   repeated_guest                        25392 non-null  int64
 10  no_of_previous_cancellations          25392 non-null  int64
 11  no_of_previous_bookings_not_canceled  25392 non-null  int64
 12  avg_price_per_room                    25392 non-null  float64
 13  no_of_special_requests                25392 non-null  int64
 14  type_of_meal_plan_Meal Plan 2         25392 non-null  uint8
 15  type_of_meal_plan_Meal Plan 3         25392 non-null  uint8
 16  type_of_meal_plan_Not Selected        25392 non-null  uint8
```

```
 17  room_type_reserved_Room_Type 2        25392 non-null  uint8
 18  room_type_reserved_Room_Type 3        25392 non-null  uint8
 19  room_type_reserved_Room_Type 4        25392 non-null  uint8
 20  room_type_reserved_Room_Type 5        25392 non-null  uint8
 21  room_type_reserved_Room_Type 6        25392 non-null  uint8
 22  room_type_reserved_Room_Type 7        25392 non-null  uint8
 23  market_segment_type_Complementary     25392 non-null  uint8
 24  market_segment_type_Corporate         25392 non-null  uint8
 25  market_segment_type_Offline           25392 non-null  uint8
 26  market_segment_type_Online            25392 non-null  uint8
dtypes: float64(1), int64(13), uint8(13)
memory usage: 3.2 MB
```

Cost Complexity Pruning

In [131…

```
clf = DecisionTreeClassifier(random_state=1, class_weight="balanced")
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = abs(path.ccp_alphas), path.impurities
```

In [132…

```
pd.DataFrame(path)
```

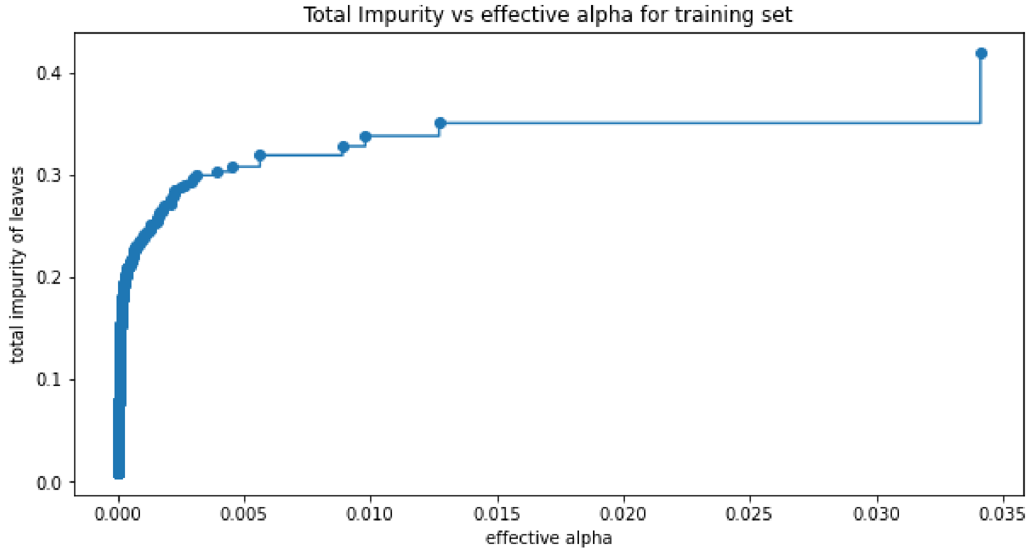
Out[132…

	ccp_alphas	impurities
0	0.00000	0.00838
1	0.00000	0.00838
2	0.00000	0.00838
3	0.00000	0.00838
4	0.00000	0.00838
...	...	...
1889	0.00890	0.32806
1890	0.00980	0.33786
1891	0.01272	0.35058
1892	0.03412	0.41882
1893	0.08118	0.50000

1894 rows × 2 columns

In [133…

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o", drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
plt.show()
```



Next, we train a decision tree using the 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.

In [134…

```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(
       random_state=1, max_depth=5, ccp_alpha=ccp_alpha, class_weight="balanced"
    )
    clf.fit(
        X_train, y_train
    )
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

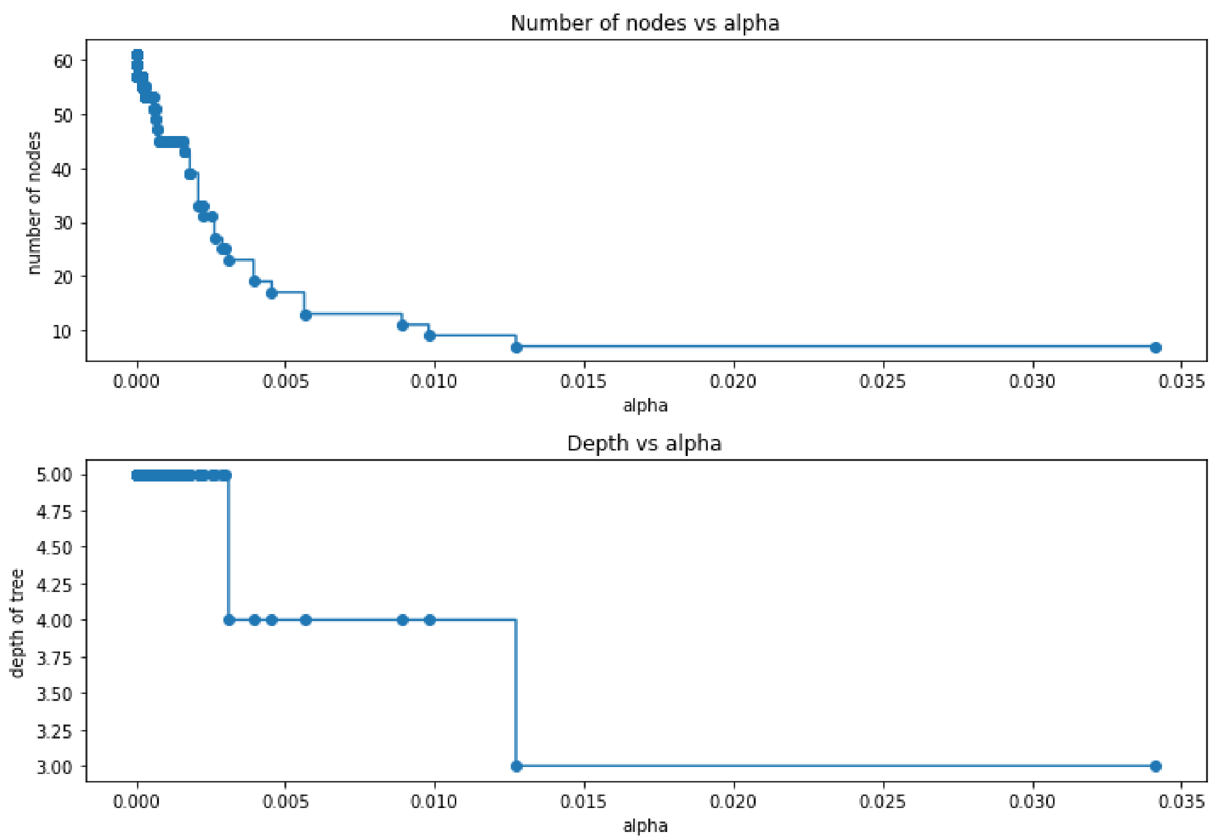
```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.08117914389136954
```

For the remainder, we remove the last element in `clfs` and `ccp_alphas`, because it is the trivial tree with only one node. Here we show that the number of nodes and tree depth decreases as alpha increases.

In [135…

```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]
fig, ax = plt.subplots(2, 1, figsize=(10, 7))
ax[0].plot(ccp_alphas, node_counts, marker="o", drawstyle="steps-post")
```

```
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("number of nodes")
ax[0].set_title("Number of nodes vs alpha")
ax[1].plot(ccp_alphas, depth, marker="o", drawstyle="steps-post")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
fig.tight_layout()
```



F1 score vs alpha for training and testing sets¶

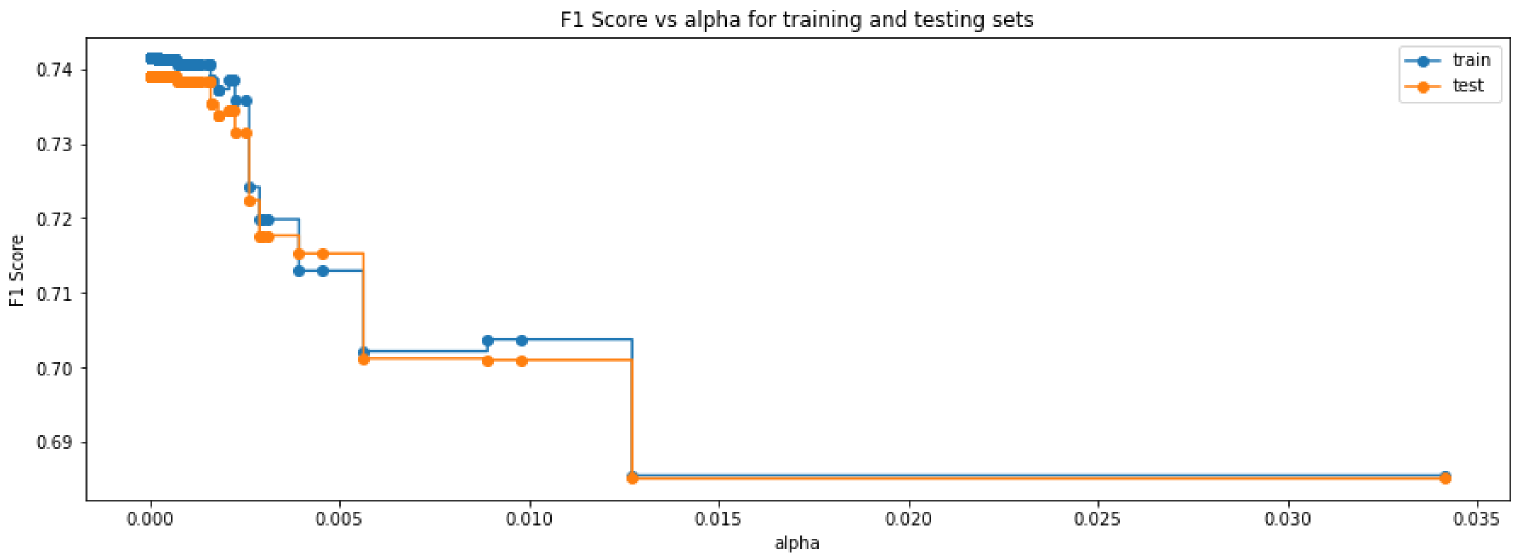
In [136…

```
f1_train = []
for clf in clfs:
    pred_train = clf.predict(X_train)
    values_train = f1_score(y_train, pred_train)
    f1_train.append(values_train)

f1_test = []
for clf in clfs:
    pred_test = clf.predict(X_test)
```

```
    values_test = f1_score(y_test, pred_test)
    f1_test.append(values_test)
```

In [137…

```
fig, ax = plt.subplots(figsize=(15, 5))
ax.set_xlabel("alpha")
ax.set_ylabel("F1 Score")
ax.set_title("F1 Score vs alpha for training and testing sets")
ax.plot(ccp_alphas, f1_train, marker="o", label="train", drawstyle="steps-post")
ax.plot(ccp_alphas, f1_test, marker="o", label="test", drawstyle="steps-post")
ax.legend()
plt.show()
```



In [138…

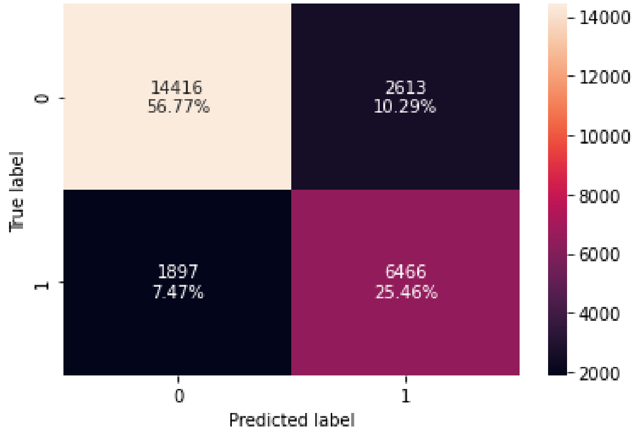
```
index_best_model = np.argmax(f1_test)
best_model = clfs[index_best_model]
print(best_model)
```

```
DecisionTreeClassifier(class_weight='balanced', max_depth=5, random_state=1)
```

Checking performance on training set

In [141…

```
confusion_matrix_sklearn(best_model, X_train, y_train)
```



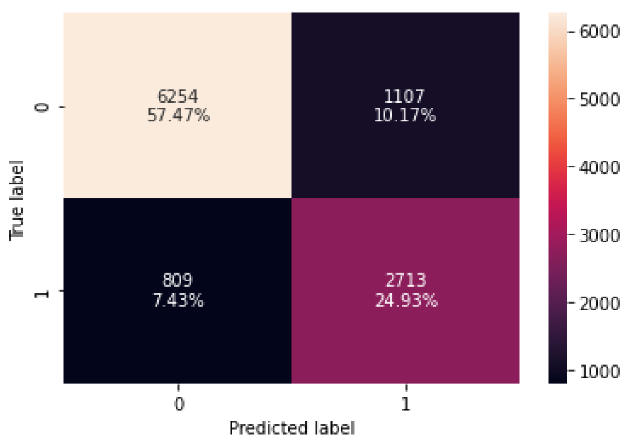
In [142…
```
decision_tree_post_perf_train = model_performance_classification_sklearn(
    best_model, X_train, y_train
)
decision_tree_post_perf_train
```

Out[142…

	Accuracy	Recall	Precision	F1
0	0.82239	0.77317	0.71219	0.74143

Checking performance on test set

In [143…
```
confusion_matrix_sklearn(best_model, X_test, y_test)
```



In [144…
```
decision_tree_post_test = model_performance_classification_sklearn(
    best_model, X_test, y_test
```

```
)
decision_tree_post_test
```

Out[144…

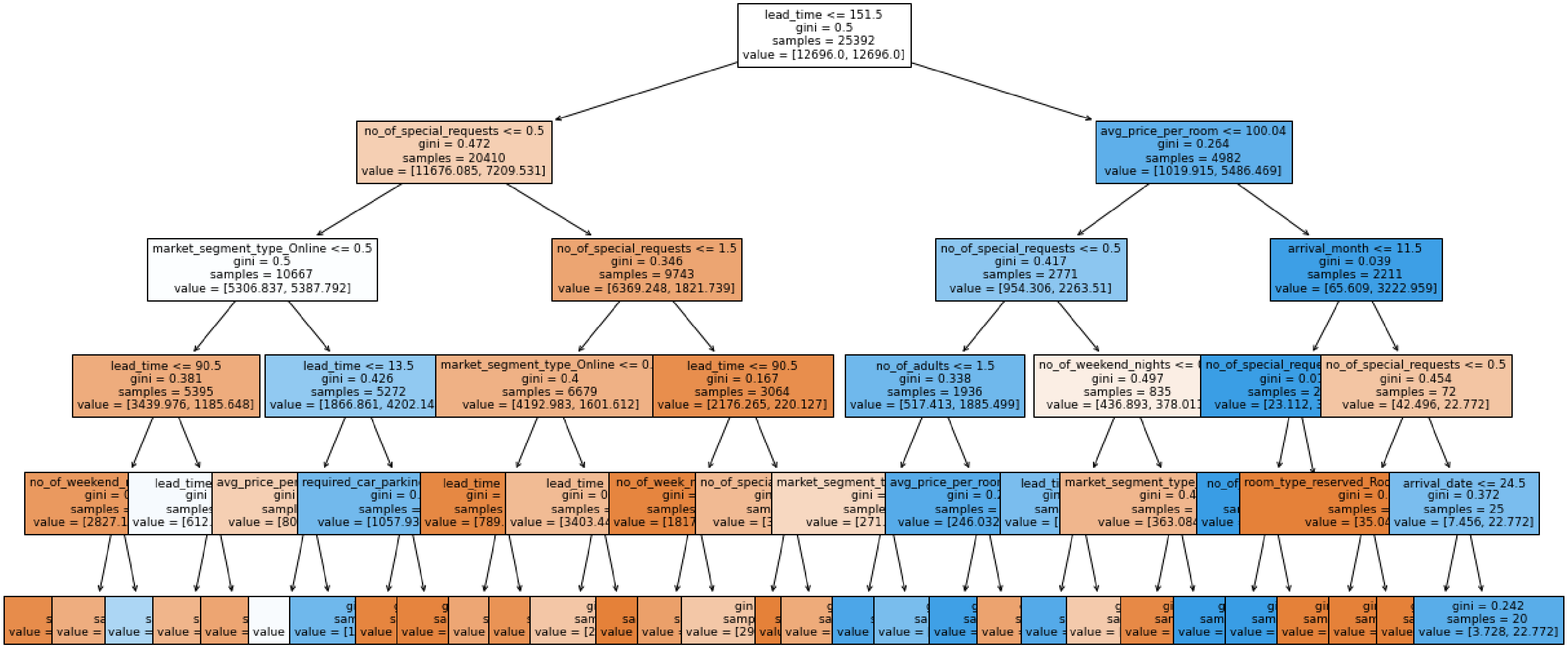
	Accuracy	Recall	Precision	F1
0	0.82395	0.77030	0.71021	0.73904

Observations:

- After post pruning the decision tree the performance has generalized on training and test set.
- The difference between recall and precision has increased.

In [145…

```
plt.figure(figsize=(20, 10))

out = tree.plot_tree(
    best_model,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```




```python
# Text report showing the rules of a decision tree -

print(tree.export_text(best_model, feature_names=feature_names, show_weights=True))
```

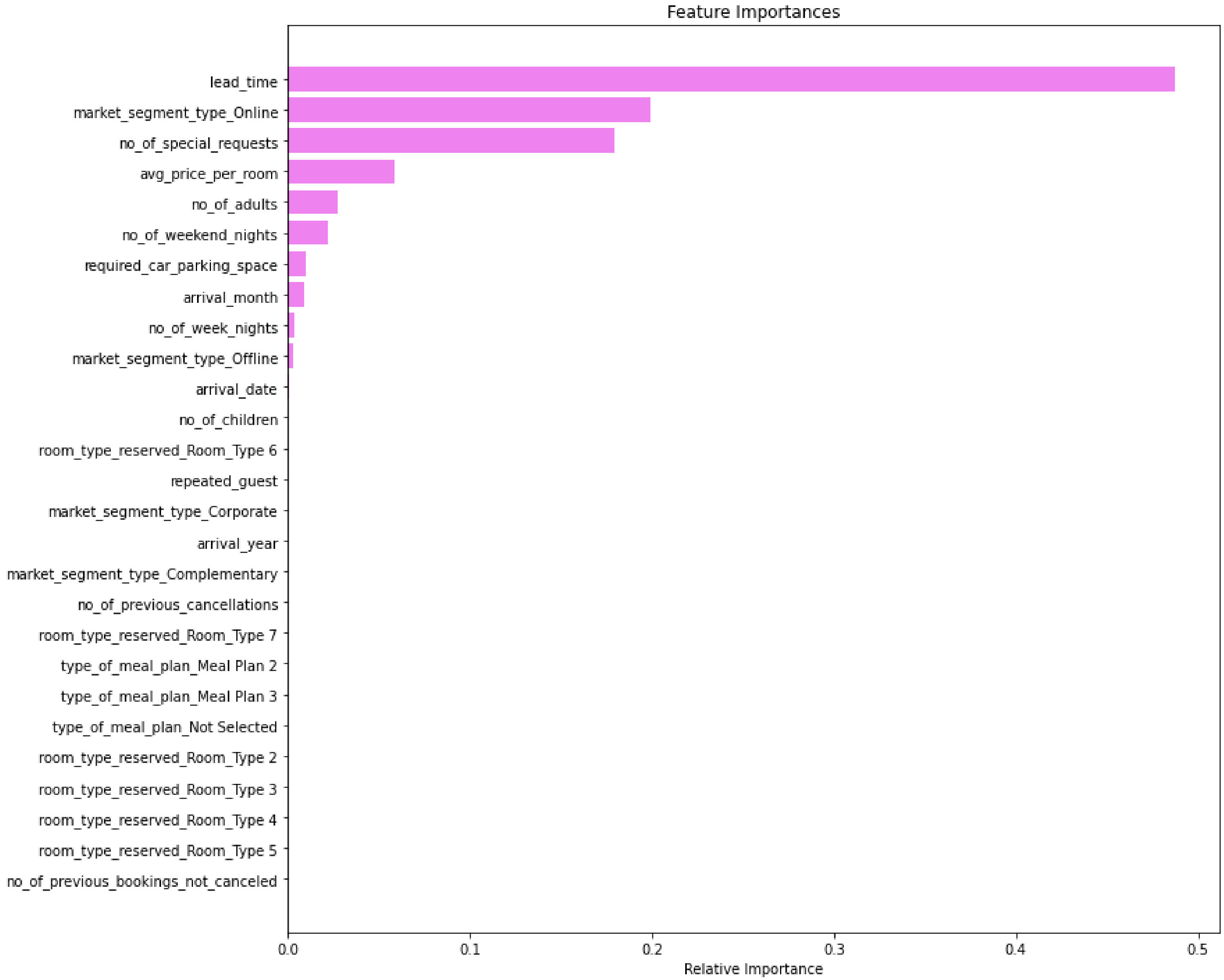
```
|--- lead_time <= 151.50
|   |--- no_of_special_requests <= 0.50
|   |   |--- market_segment_type_Online <= 0.50
|   |   |   |--- lead_time <= 90.50
|   |   |   |   |--- no_of_weekend_nights <= 0.50
|   |   |   |   |   |--- weights: [1737.14, 157.88] class: 0
|   |   |   |   |--- no_of_weekend_nights >  0.50
|   |   |   |   |   |--- weights: [1090.00, 384.08] class: 0
|   |   |   |--- lead_time >  90.50
|   |   |   |   |--- lead_time <= 117.50
|   |   |   |   |   |--- weights: [297.48, 513.12] class: 1
|   |   |   |   |--- lead_time >  117.50
|   |   |   |   |   |--- weights: [315.37, 130.56] class: 0
|   |   |--- market_segment_type_Online >  0.50
|   |   |   |--- lead_time <= 13.50
|   |   |   |   |--- avg_price_per_room <= 99.44
|   |   |   |   |   |--- weights: [456.28, 132.08] class: 0
|   |   |   |   |--- avg_price_per_room >  99.44
|   |   |   |   |   |--- weights: [352.65, 365.87] class: 1
```

```
|   |   |   |--- lead_time >  13.50
|   |   |   |   |--- required_car_parking_space <= 0.50
|   |   |   |   |   |--- weights: [1009.48, 3702.68] class: 1
|   |   |   |   |--- required_car_parking_space >  0.50
|   |   |   |   |   |--- weights: [48.46, 1.52] class: 0
|   |--- no_of_special_requests >  0.50
|   |   |--- no_of_special_requests <= 1.50
|   |   |   |--- market_segment_type_Online <= 0.50
|   |   |   |   |--- lead_time <= 102.50
|   |   |   |   |   |--- weights: [712.75, 18.22] class: 0
|   |   |   |   |--- lead_time >  102.50
|   |   |   |   |   |--- weights: [76.79, 22.77] class: 0
|   |   |   |--- market_segment_type_Online >  0.50
|   |   |   |   |--- lead_time <= 8.50
|   |   |   |   |   |--- weights: [756.73, 107.79] class: 0
|   |   |   |   |--- lead_time >  8.50
|   |   |   |   |   |--- weights: [2646.71, 1452.84] class: 0
|   |   |--- no_of_special_requests >  1.50
|   |   |   |--- lead_time <= 90.50
|   |   |   |   |--- no_of_week_nights <= 3.50
|   |   |   |   |   |--- weights: [1585.04, 0.00] class: 0
|   |   |   |   |--- no_of_week_nights >  3.50
|   |   |   |   |   |--- weights: [232.61, 57.69] class: 0
|   |   |   |--- lead_time >  90.50
|   |   |   |   |--- no_of_special_requests <= 2.50
|   |   |   |   |   |--- weights: [291.51, 162.44] class: 0
|   |   |   |   |--- no_of_special_requests >  2.50
|   |   |   |   |   |--- weights: [67.10, 0.00] class: 0
|--- lead_time >  151.50
|   |--- avg_price_per_room <= 100.04
|   |   |--- no_of_special_requests <= 0.50
|   |   |   |--- no_of_adults <= 1.50
|   |   |   |   |--- market_segment_type_Online <= 0.50
|   |   |   |   |   |--- weights: [261.69, 86.53] class: 0
|   |   |   |   |--- market_segment_type_Online >  0.50
|   |   |   |   |   |--- weights: [9.69, 100.20] class: 1
|   |   |   |--- no_of_adults >  1.50
|   |   |   |   |--- avg_price_per_room <= 82.47
|   |   |   |   |   |--- weights: [216.96, 667.97] class: 1
|   |   |   |   |--- avg_price_per_room >  82.47
|   |   |   |   |   |--- weights: [29.08, 1030.80] class: 1
|   |   |--- no_of_special_requests >  0.50
|   |   |   |--- no_of_weekend_nights <= 0.50
|   |   |   |   |--- lead_time <= 180.50
|   |   |   |   |   |--- weights: [44.73, 12.14] class: 0
|   |   |   |   |--- lead_time >  180.50
|   |   |   |   |   |--- weights: [29.08, 212.54] class: 1
|   |   |   |--- no_of_weekend_nights >  0.50
|   |   |   |   |--- market_segment_type_Offline <= 0.50
|   |   |   |   |   |--- weights: [250.51, 145.74] class: 0
|   |   |   |   |--- market_segment_type_Offline >  0.50
|   |   |   |   |   |--- weights: [112.58, 7.59] class: 0
|   |--- avg_price_per_room >  100.04
|   |   |--- arrival_month <= 11.50
|   |   |   |--- no_of_special_requests <= 2.50
|   |   |   |   |--- no_of_children <= 0.50
|   |   |   |   |   |--- weights: [0.00, 2917.82] class: 1
|   |   |   |   |--- no_of_children >  0.50
```

```
|   |   |   |   |   |--- weights: [0.00, 282.37] class: 1
|   |   |   |--- no_of_special_requests >  2.50
|   |   |   |   |--- weights: [23.11, 0.00] class: 0
|   |   |--- arrival_month >  11.50
|   |   |   |--- no_of_special_requests <= 0.50
|   |   |   |   |--- room_type_reserved_Room_Type 6 <= 0.50
|   |   |   |   |   |--- weights: [30.57, 0.00] class: 0
|   |   |   |   |--- room_type_reserved_Room_Type 6 >  0.50
|   |   |   |   |   |--- weights: [4.47, 0.00] class: 0
|   |   |   |--- no_of_special_requests >  0.50
|   |   |   |   |--- arrival_date <= 24.50
|   |   |   |   |   |--- weights: [3.73, 0.00] class: 0
|   |   |   |   |--- arrival_date >  24.50
|   |   |   |   |   |--- weights: [3.73, 22.77] class: 1
```

In [147…

```
importances = best_model.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Observations:

- The tree is quite complex as compared to the pre-pruned tree.
- The feature importance is same as we got in pre-pruned tree.

Comparing Decision Tree Models

In [148…

```python
# training performance comparison

models_train_comp_df = pd.concat(
    [
        decision_tree_perf_train.T,
        decision_tree_tune_perf_train.T,
        decision_tree_post_perf_train.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree sklearn",
    "Decision Tree (Pre-Pruning)",
    "Decision Tree (Post-Pruning)",
]
print("Training performance comparison:")
models_train_comp_df
```

Training performance comparison:

Out[148...

	Decision Tree sklearn	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Accuracy	0.99421	0.83097	0.82239
Recall	0.98661	0.78608	0.77317
Precision	0.99578	0.72425	0.71219
F1	0.99117	0.75390	0.74143

In [149...

```python
# testing performance comparison

models_test_comp_df = pd.concat(
    [
        decision_tree_perf_test.T,
        decision_tree_tune_perf_test.T,
        decision_tree_post_test.T,
    ],
    axis=1,
)
models_test_comp_df.columns = [
    "Decision Tree sklearn",
    "Decision Tree (Pre-Pruning)",
    "Decision Tree (Post-Pruning)",
]
print("Test set performance comparison:")
models_test_comp_df
```

Test set performance comparison:

Out[149...

	Decision Tree sklearn	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Accuracy	0.87118	0.83497	0.82395

	Decision Tree sklearn	Decision Tree (Pre-Pruning)	Decision Tree (Post-Pruning)
Recall	0.81175	0.78336	0.77030
Precision	0.79461	0.72758	0.71021
F1	0.80309	0.75444	0.73904

Observations:

- Decision tree model with default parameters is overfitting the training data and is not able to generalize well.
- Pre-pruned tree has given a generalized performance with balanced values of precision and recall.
- Post-pruned tree is giving a high F1 score as compared to other models but the difference between precision and recall is high.
- The hotel will be able to maintain a balance between resources and brand equity using the pre-pruned decision tree model.

Conclusion

- Overall we can see that the Decision Tree model performs better on the dataset.
- Looking at important variables based on p-values in Logistic regression and feature importance in the Decision Tree model
- Lead Time, Number of special requests, Average price per room are important in both model
- From the Logistic Regression model we observe that Lead Time, and Average price per room have a positive relation with bookings getting canclled. And the number of special requests has negative relation with bookings getting cancelled.

Actionable Insights and Recommendations

- The lead time and the number of special requests made by the customer play a key role in identifying if a booking will be cancelled or not. Bookings where a customer has made a special request and the booking was done under 151 days to the date of arrival are less likely to be canceled.

  - Using this information, the hotel can take the following actions:
    - Set up a system that can send a prompt like an automated email to the customers before the arrival date asking for a re-confirmation of their booking and any changes they would like to make in their bookings.
    - Remind guests about imminent deadlines.

The response given by the customer will give the hotel ample time to re-sell the room or make preparations for the customers' requests.

- Adopt stricter cancellation policies.

  - The bookings where the average price per room is high, and there were special requests associated should not get a full refund as the loss of resources will be high in these cases.
  - Ideally the cancellation policies should be consistent across all market segments but as noticed in our analysis high percentage of bookings done online are cancelled. The booking cancelled online should yield less percentage of refund to the customers.

The refunds, cancellation fee, etc should be highlighted on the website/app before a customer confirms their booking to safeguard guests' interest.

- The length of stay at the hotel can be restricted.
    - We saw in our analysis that bookings, where the total length of stay was more than 5 days, had higher chances of getting cancelled.
    - Hotel can allow bookings up to 5 days only and then customers should be asked to re-book if they wish to stay longer. These policies can be relaxed for corporate and Aviation market segments. For other market segments, the process should be fairly easy to not hamper their experience with the hotel.

Such restrictions can be strategized by the hotel to generate additional revenue.

- In the months of December and January cancellation to non-cancellation ratio is low. Customers might travel to celebrate Christmas and New Year. The hotel should ensure that enough human resources are available to cater to the needs of the guests.

- October and September saw the highest number of bookings but also high number of cancellations. This should be investigated further by the hotel.

- Post-booking interactions can be initiated with the customers.

    - Post-booking interactions will show the guests the level of attention and care they would receive at the hotel.
    - To give guests a personalized experience, information about local events, nearby places to explore, etc can be shared from time to time.
- Improving the experience of repeated customers.

    - Our analysis shows that there are very few repeated customers and the cancellation among them is very less which is a good indication as repeat customers are important for the hospitality industry as they can help in spreading the word of mouth.
        - A loyal guest is usually more profitable for the business because they are more familiar with offerings from the hotel they have visited before.
        - Attracting new customers is tedious and costs more as compared to a repeated guest.
        - A loyalty program that offers - special discounts, access to services in hotels, etc for these customers can help in improving their experience.