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:

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

```
# Installing the libraries with the specified version.
!pip install pandas==2.0.3 numpy==1.25.2 matplotlib==3.7.1 seaborn==0.13.1 scikit-learn==1.2.2 statsmodels==0.14.1 -q --user
```

Note: *After running the above cell, kindly restart the notebook kernel and run all cells sequentially from the start again.*

```
# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)
# setting the precision of floating numbers to 5 decimal points
pd.set_option("display.float_format", lambda x: "%.5f" % x)

# Library to split data
from sklearn.model_selection import train_test_split

# To build model for prediction
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# To tune different models
from sklearn.model_selection import GridSearchCV


# To get diferent metric scores
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    precision_recall_curve,
    roc_curve,
    make_scorer,
)

import warnings
warnings.filterwarnings("ignore")

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

Loading the dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# original data
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/4 - Supervised Learning - Classification/Final Project/INNHotelsGroup.csv')
```

Data Overview

- Observations
- Sanity checks

Loading the dataset

```
df.head()
```

	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	lead_time	arrival_
0	INN00001	2	0	1	2	Meal Plan 1	0	Room_Type 1	224	
1	INN00002	2	0	2	3	Not Selected	0	Room_Type 1	5	
2	INN00003	1	0	2	1	Meal Plan 1	0	Room_Type 1	1	
3	INN00004	2	0	0	2	Meal Plan 1	0	Room_Type 1	211	
4	INN00005	2	0	1	1	Not Selected	0	Room_Type 1	48	

Next steps: Generate code with df View recommended plots New interactive sheet

Shape of the dataset

```
df.shape
```

```
(36275, 19)
```

Observations - There are 36,275 rows and 19 columns in the dataset

Info regarding column datatypes

```
df.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 14 numerical (13 int64 & 1 float64) and 5 object type columns in the dataset

Statistics summary for the numerical columns

```
df.describe()
```

	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	repeated_g
count	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.0
mean	1.84496	0.10528	0.81072	2.20430	0.03099	85.23256	2017.82043	7.42365	15.59700	0.0
std	0.51871	0.40265	0.87064	1.41090	0.17328	85.93082	0.38384	3.06989	8.74045	0.1
min	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	2017.00000	1.00000	1.00000	0.0
25%	2.00000	0.00000	0.00000	1.00000	0.00000	17.00000	2018.00000	5.00000	8.00000	0.0
50%	2.00000	0.00000	1.00000	2.00000	0.00000	57.00000	2018.00000	8.00000	16.00000	0.0
75%	2.00000	0.00000	2.00000	3.00000	0.00000	126.00000	2018.00000	10.00000	23.00000	0.0
max	4.00000	10.00000	7.00000	17.00000	1.00000	443.00000	2018.00000	12.00000	31.00000	1.0

Checking missing values

```
df.isnull().sum()
```

	0
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

Observations - There is no missing values in the data

Check for duplicates in the dataset

```
print("There are",df.duplicated().sum(),"duplicated rows")
```

```
There are 0 duplicated rows
```

Dropping the columns with all unique values

```
df.Booking_ID.nunique()
```

```
36275
```

Observations - the Booking_ID column contains only unique values, so we can drop it

```
#drop the Booking_ID column
df = df.drop(["Booking_ID"], axis=1)
```

```
#get info after dropping the Booking_ID column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 18 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   no_of_adults                          36275 non-null  int64  
 1   no_of_children                        36275 non-null  int64  
 2   no_of_weekend_nights                  36275 non-null  int64  
```

```
 3   no_of_week_nights                     36275 non-null  int64
 4   type_of_meal_plan                     36275 non-null  object
 5   required_car_parking_space            36275 non-null  int64
 6   room_type_reserved                    36275 non-null  object
 7   lead_time                             36275 non-null  int64
 8   arrival_year                          36275 non-null  int64
 9   arrival_month                         36275 non-null  int64
 10  arrival_date                          36275 non-null  int64
 11  market_segment_type                   36275 non-null  object
 12  repeated_guest                        36275 non-null  int64
 13  no_of_previous_cancellations          36275 non-null  int64
 14  no_of_previous_bookings_not_canceled  36275 non-null  int64
 15  avg_price_per_room                    36275 non-null  float64
 16  no_of_special_requests                36275 non-null  int64
 17  booking_status                        36275 non-null  object
dtypes: float64(1), int64(13), object(4)
memory usage: 5.0+ MB
```

Observations - There are 14 numerical (13 int64 & 1 float64) and 4 object type columns in the dataset

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 few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

Statistical summary of the data

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

	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

The below functions are needed to be defined to carry out the EDA

```
# the functions below were copied from "MLS 2 - Decision Tree: Session Notebook - Machine Failure Prediction" to help with the Exploratory Data Analysis (EDA)

# function to create histogram boxplot
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.2, 0.5)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="darkseagreen"
    )  # boxplot will be created and a triangle will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram


# function to create labeled barplots
def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top
```

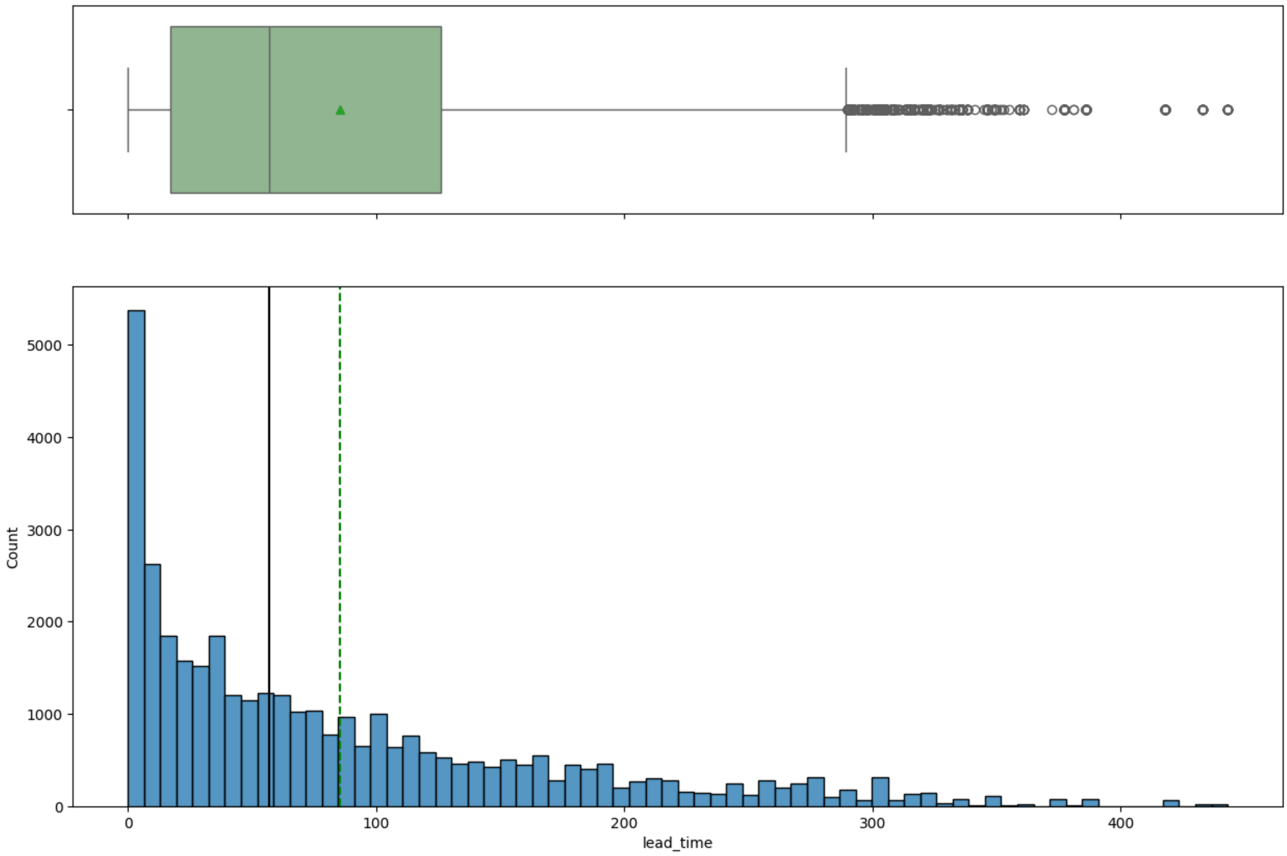
```
    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 2, 6))
    else:
        plt.figure(figsize=(n + 2, 6))

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

    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


# function to plot distributions wrt target
def distribution_plot_wrt_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()


# function to plot stacked barplot
def stacked_barplot(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, 5))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

Univariate Analysis

lead_time field

```
histogram_boxplot(df, "lead_time")
```

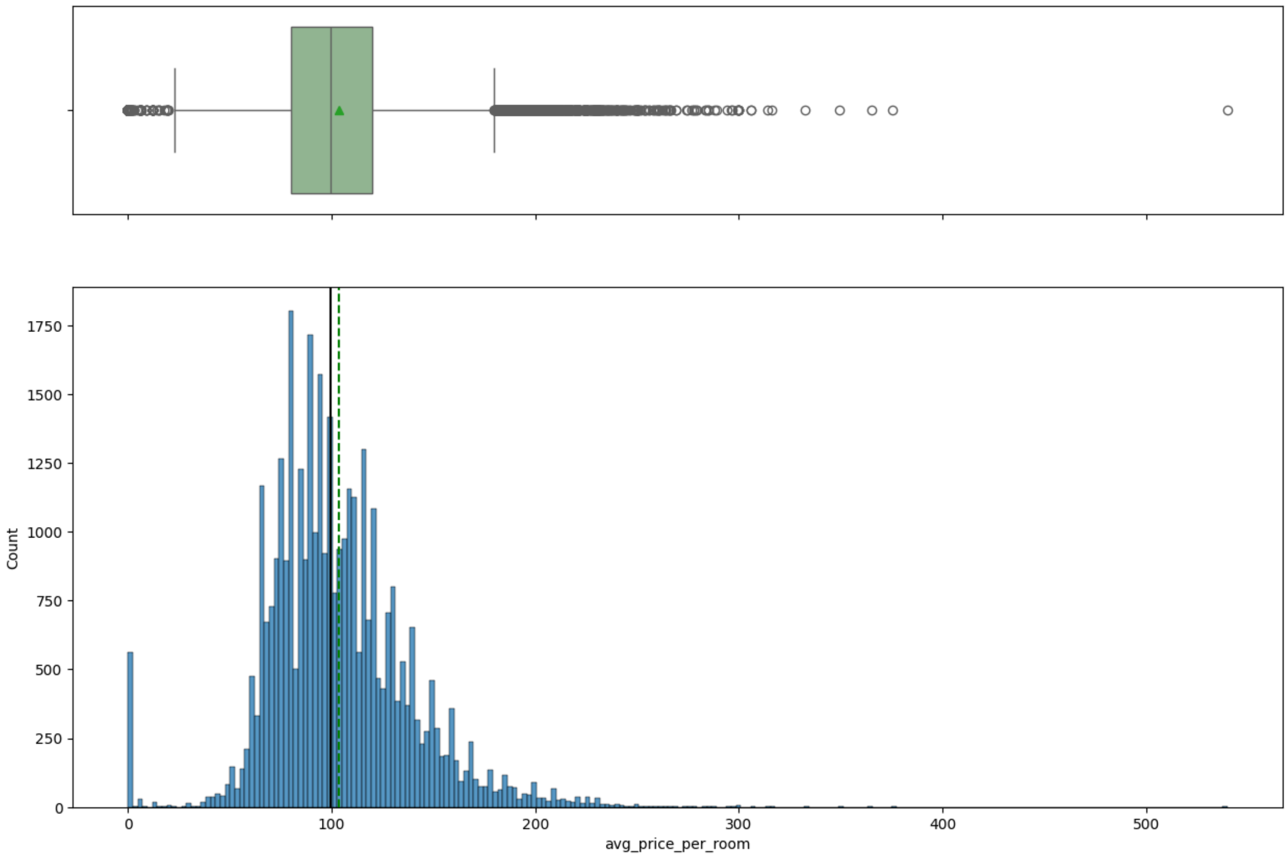


Observations:

- The lead_time distribution is left skewed and it presents outliners

avg_price_per_room field

```
histogram_boxplot(df, "avg_price_per_room")
```



Observations:

- The avg_price_per_room distribution is slightly left skewed and it presents outliners

```
df[df["avg_price_per_room"] == 0]
```

	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	arrival_year	ar
63	1	0	0	1	Meal Plan 1	0	Room_Type 1	2	2017	
145	1	0	0	2	Meal Plan 1	0	Room_Type 1	13	2018	
209	1	0	0	0	Meal Plan 1	0	Room_Type 1	4	2018	
266	1	0	0	2	Meal Plan 1	0	Room_Type 1	1	2017	
267	1	0	2	1	Meal Plan 1	0	Room_Type 1	4	2017	
...	...	...	...	...	...	...	...	...	...	
35983	1	0	0	1	Meal Plan 1	0	Room_Type 7	0	2018	
36080	1	0	1	1	Meal Plan 1	0	Room_Type 7	0	2018	
36114	1	0	0	1	Meal Plan 1	0	Room_Type 1	1	2018	
36217	2	0	2	1	Meal Plan 1	0	Room_Type 2	3	2017	
36250	1	0	0	2	Meal Plan 2	0	Room_Type 1	6	2017	

545 rows × 18 columns

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

market_segment_type	count
Complementary	354
Online	191

dtype: int64

```
Q1 = df["avg_price_per_room"].quantile(0.25)  # 25th quantile
Q3 = df["avg_price_per_room"].quantile(0.75)  # 75th quantile

# Calculating IQR
IQR = Q3 - Q1

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

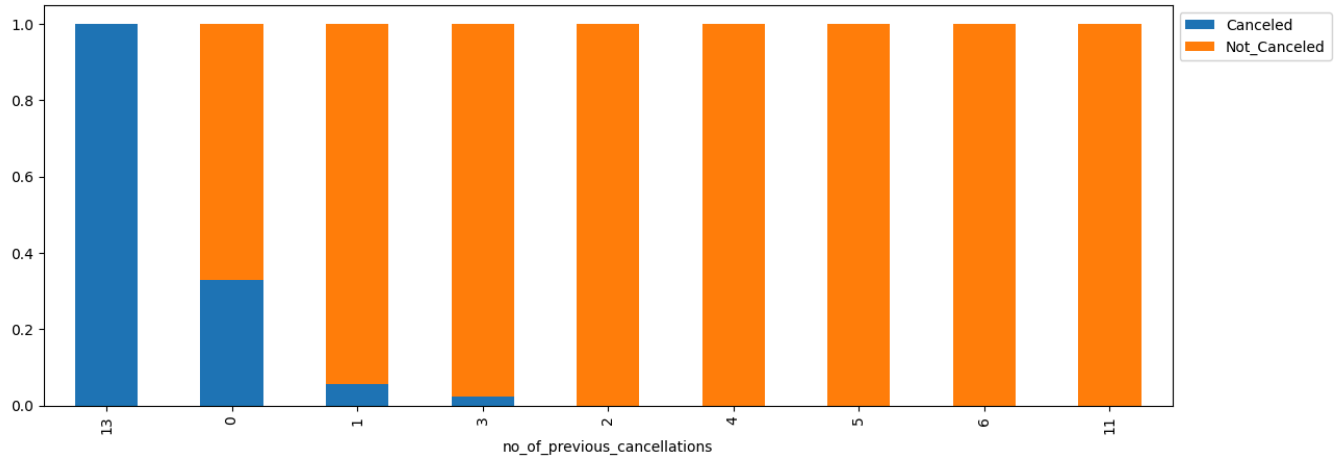
```
179.55
```

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

no_of_previous_cancellations field

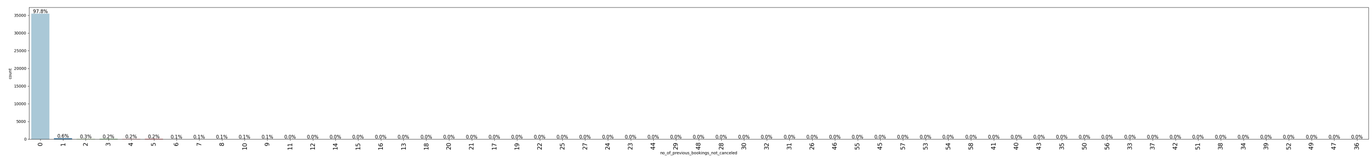
```
stacked_barplot(df, "no_of_previous_cancellations", "booking_status")
```

```
booking_status                Canceled  Not_Canceled    All
no_of_previous_cancellations
All                              11885         24390  36275
0                                11869         24068  35937
1                                   11           187    198
13                                   4             0      4
3                                    1            42     43
2                                    0            46     46
4                                    0            10     10
5                                    0            11     11
6                                    0             1      1
11                                   0            25     25
------------------------------------------------------------------------------------------------------------------------
```

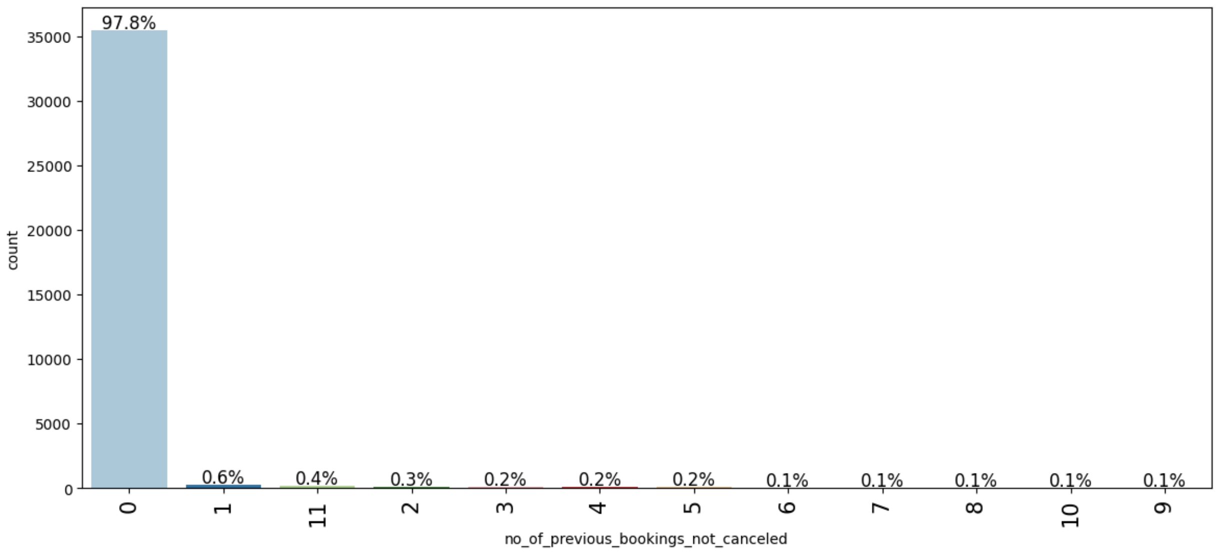


no_of_previous_bookings_not_canceled field

```
labeled_barplot(df, "no_of_previous_bookings_not_canceled", perc=True)
```

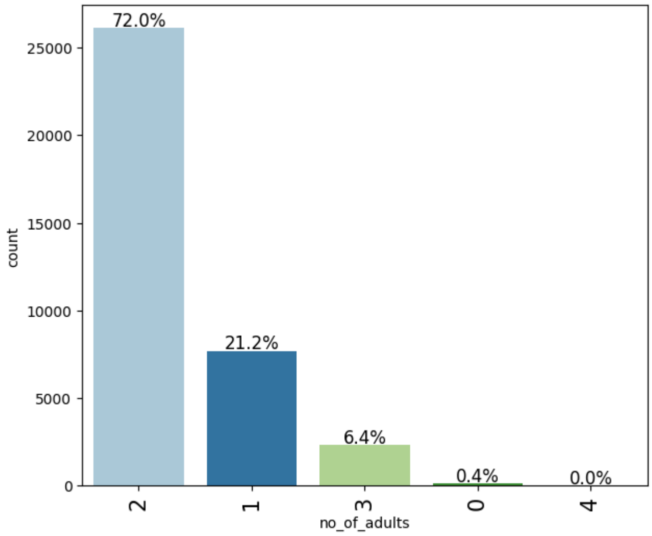


```
#combine greater than 11 into 11 for better analysis
data = df.copy()
data.loc[data['no_of_previous_bookings_not_canceled'] > 11,'no_of_previous_bookings_not_canceled'] = 11
labeled_barplot(data, "no_of_previous_bookings_not_canceled", perc=True)
```



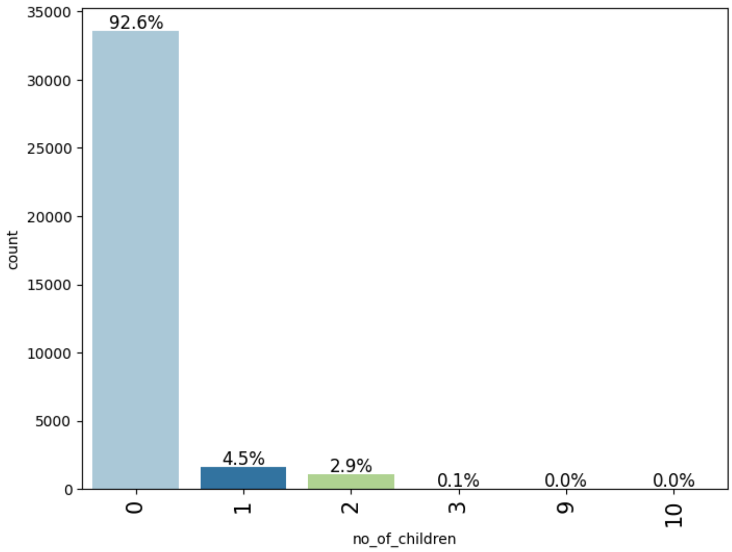
no_of_adults field

```
labeled_barplot(df, "no_of_adults", perc=True)
```

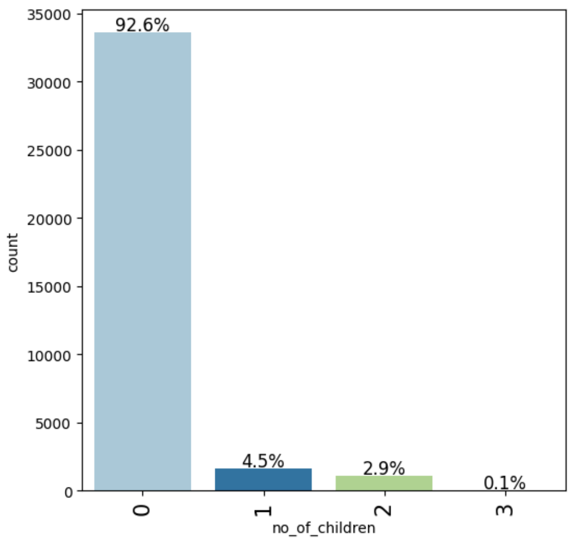


no_of_children field

```
labeled_barplot(df, "no_of_children", perc=True)
```

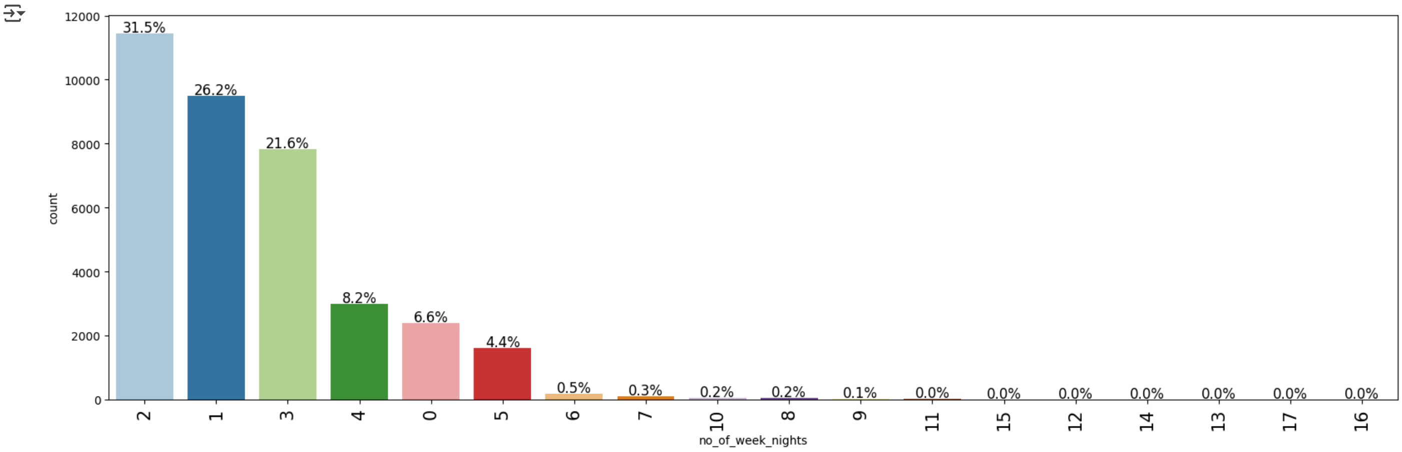



```
#combine 9 & 10 to 3 for better analysis
df["no_of_children"] = df["no_of_children"].replace([9, 10], 3)
labeled_barplot(df, "no_of_children", perc=True)
```



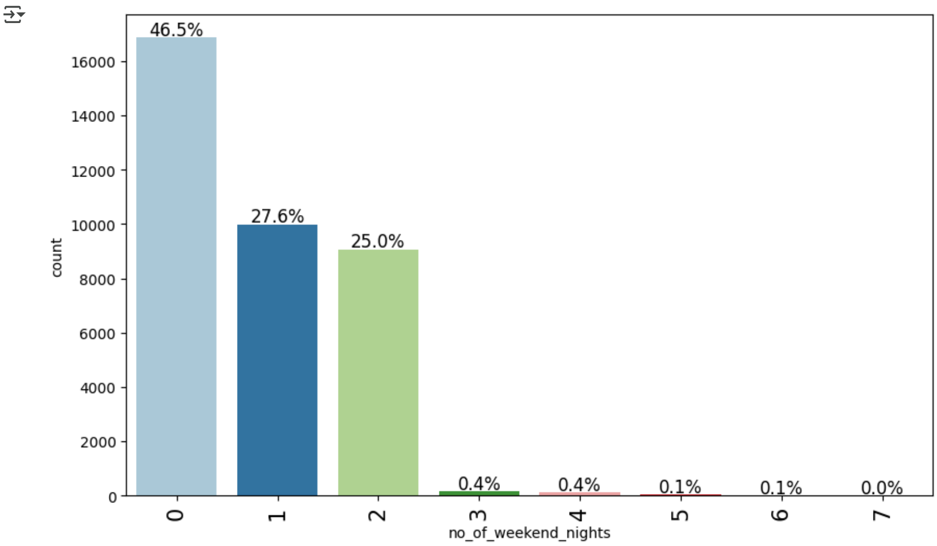
no_of_week_nights field

```
labeled_barplot(df, "no_of_week_nights", perc=True)
```



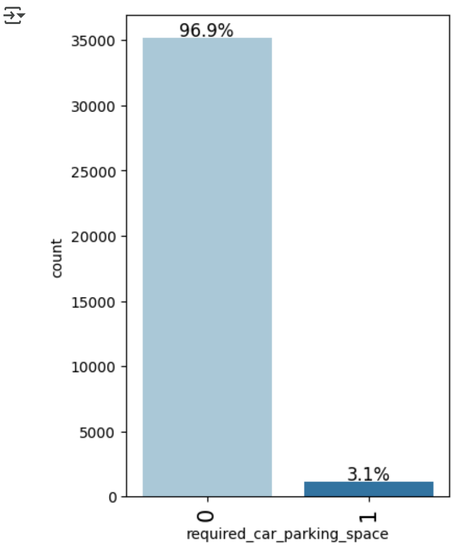
no_of_weekend_nights field

```
labeled_barplot(df, "no_of_weekend_nights", perc=True)
```

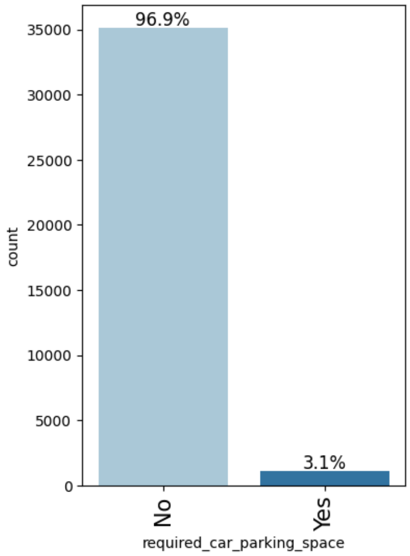


required_car_parking_space field

```
labeled_barplot(df, "required_car_parking_space", perc=True)
```

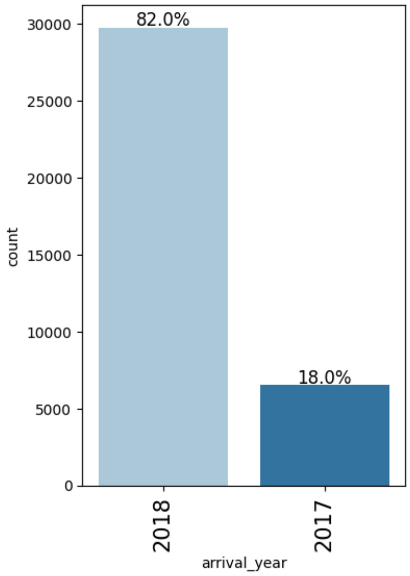



```
#changed label as per data description - (0 - No, 1- Yes)
data = df.copy()
data["required_car_parking_space"] = data["required_car_parking_space"].apply(lambda x: "No" if x == 0 else "Yes")
labeled_barplot(data, "required_car_parking_space", perc=True)
```



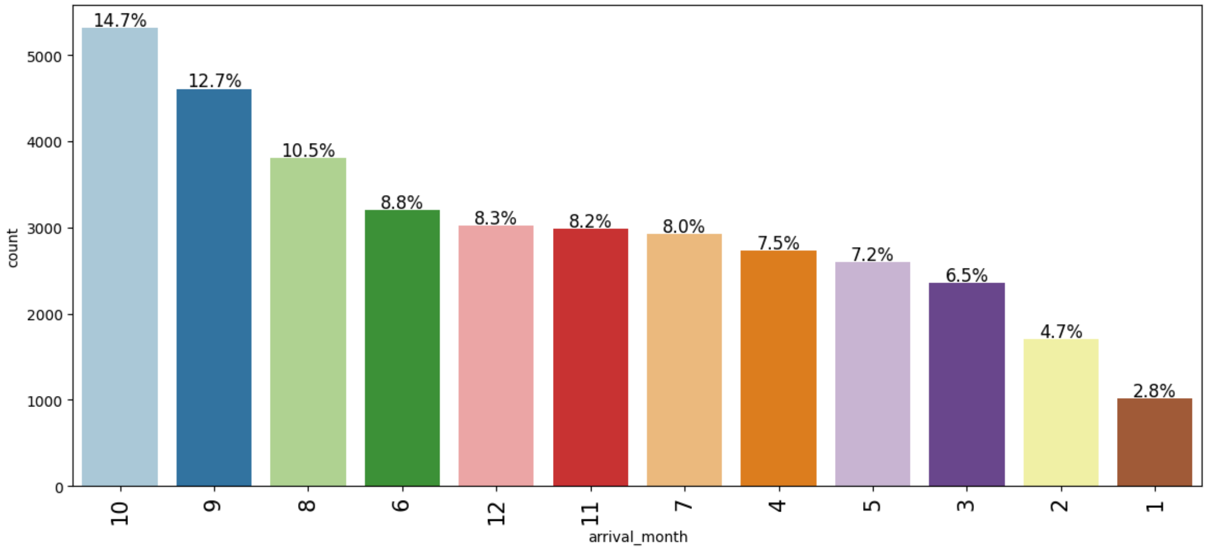
arrival_year field

```
labeled_barplot(df, "arrival_year", perc=True)
```



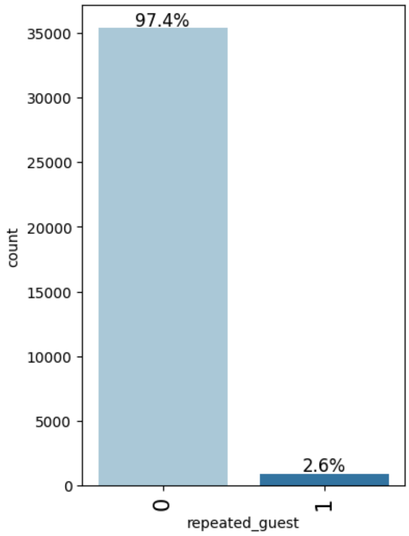
arrival_month field

```
labeled_barplot(df, "arrival_month", perc=True)
```

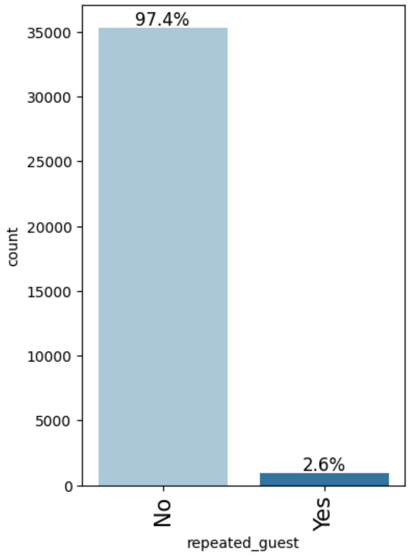


repeated_guest field

```
labeled_barplot(df, "repeated_guest", perc=True)
```

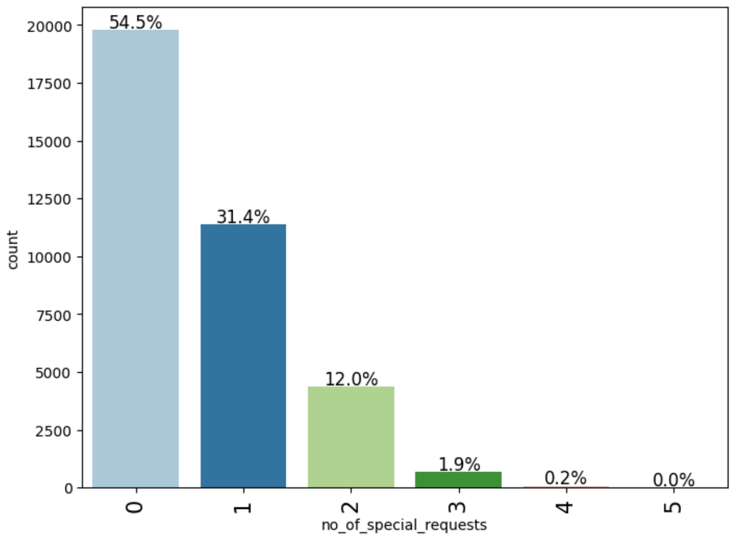



```
#changed label as per data description - (0 - No, 1- Yes)
data = df.copy()
data["repeated_guest"] = data["repeated_guest"].apply(lambda x: "No" if x == 0 else "Yes")
labeled_barplot(data, "repeated_guest", perc=True)
```



no_of_special_requests field

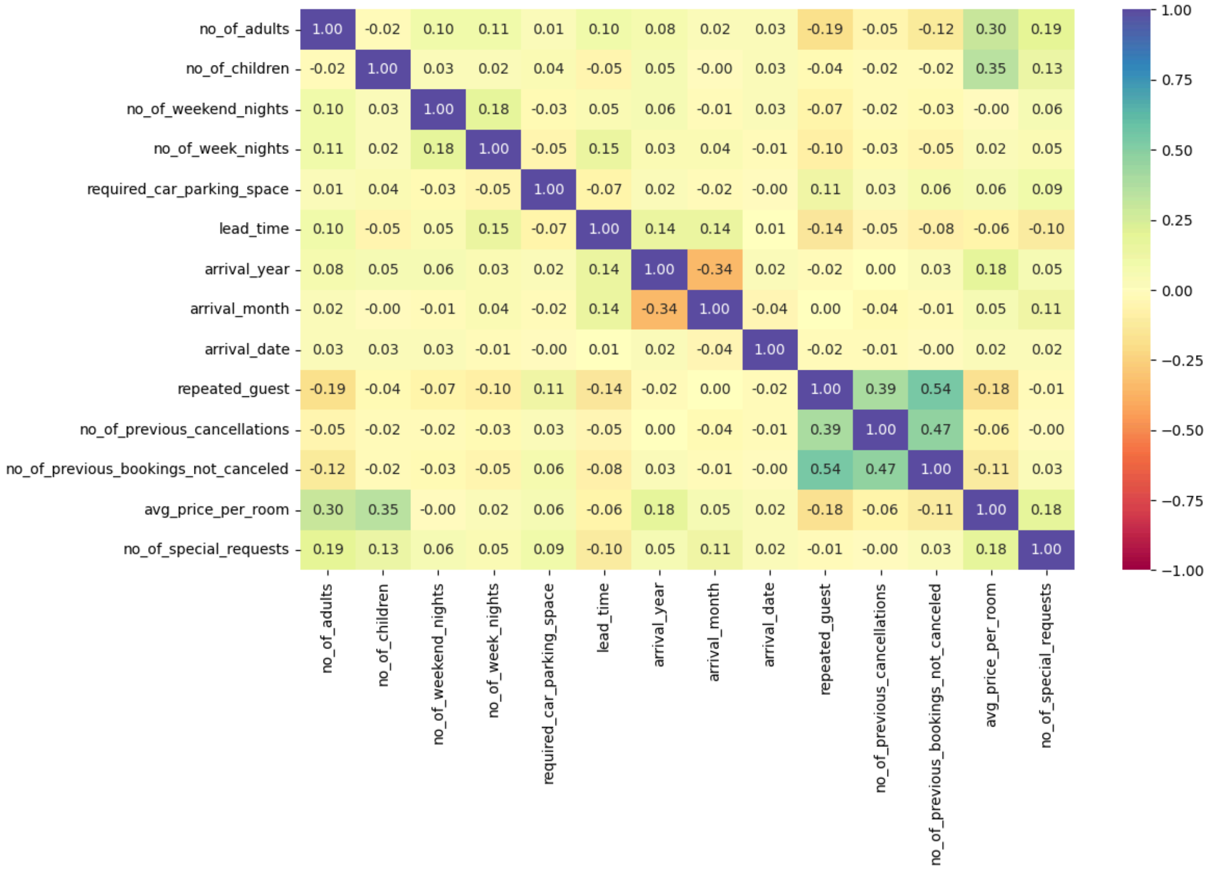
```
labeled_barplot(df, "no_of_special_requests", perc=True)
```



Bivariate Analysis

Heatmap

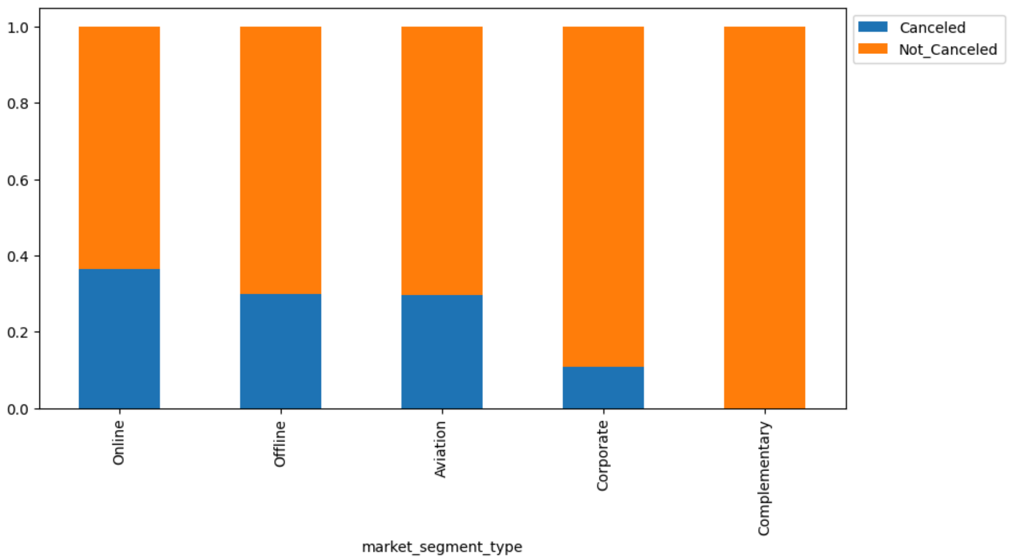
```
cols_list = df.select_dtypes(include=np.number).columns.tolist()

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



market_segment_type vs booking_status

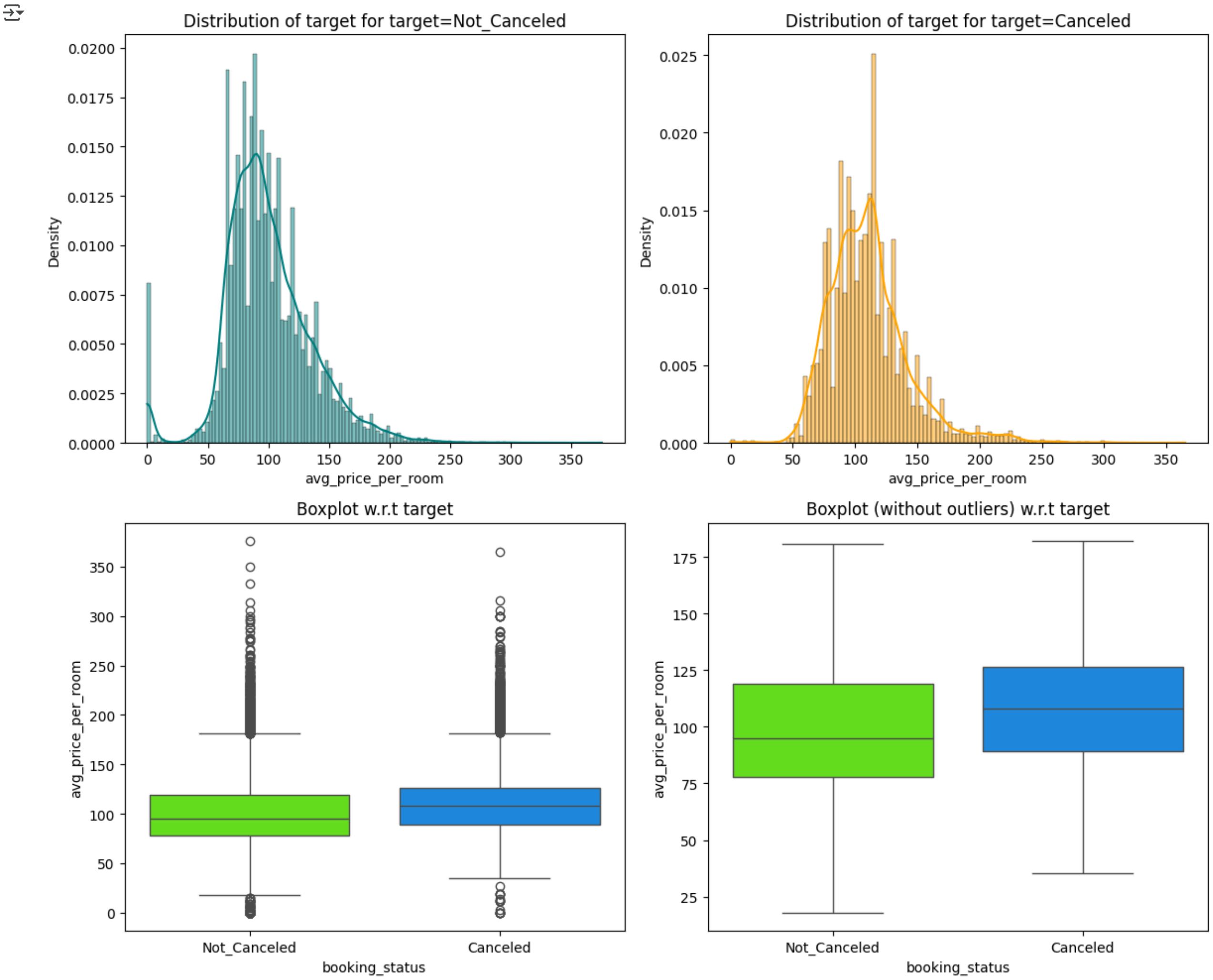
```
stacked_barplot(df, "market_segment_type", "booking_status")
```

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



avg_price_per_room vs booking_status

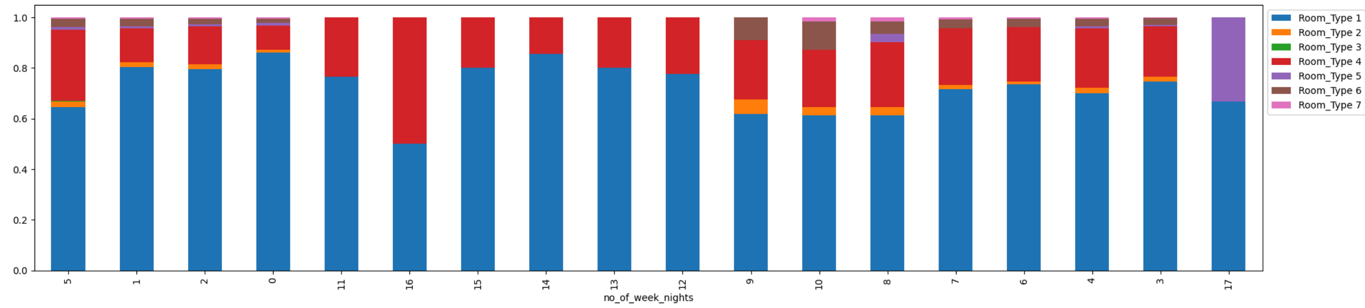
```
distribution_plot_wrt_target(df, "avg_price_per_room", "booking_status")
```



no_of_week_nights vs room_type_reserved

```
stacked_barplot(df, "no_of_week_nights", "room_type_reserved")
```

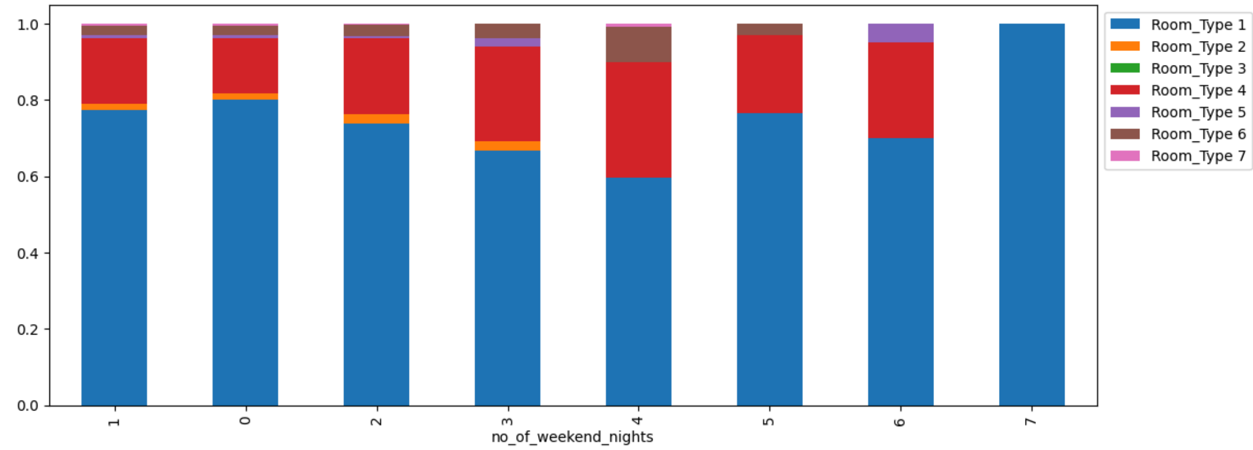
room_type_reserved no_of_week_nights	Room_Type 1	Room_Type 2	Room_Type 3	Room_Type 4	Room_Type 5	Room_Type 6	Room_Type 7	All
All	28130	692	7	6057	265	966	158	36275
2	9111	215	3	1723	79	266	47	11444
1	7620	187	3	1271	68	288	51	9488
5	1044	35	1	457	18	52	7	1614
11	13	0	0	4	0	0	0	17
17	2	0	0	0	1	0	0	3
16	1	0	0	1	0	0	0	2
15	8	0	0	2	0	0	0	10
14	6	0	0	1	0	0	0	7
13	4	0	0	1	0	0	0	5
12	7	0	0	2	0	0	0	9
0	2058	22	0	231	27	39	10	2387
10	38	2	0	14	0	7	1	62
8	38	2	0	16	2	3	1	62
7	81	2	0	25	0	4	1	113
6	139	2	0	41	0	6	1	189
4	2094	62	0	707	23	92	12	2990
3	5845	161	0	1553	47	206	27	7839
9	21	2	0	8	0	3	0	34



no_of_weekend_nights vs room_type_reserved

```
stacked_barplot(df, "no_of_weekend_nights", "room_type_reserved")
```

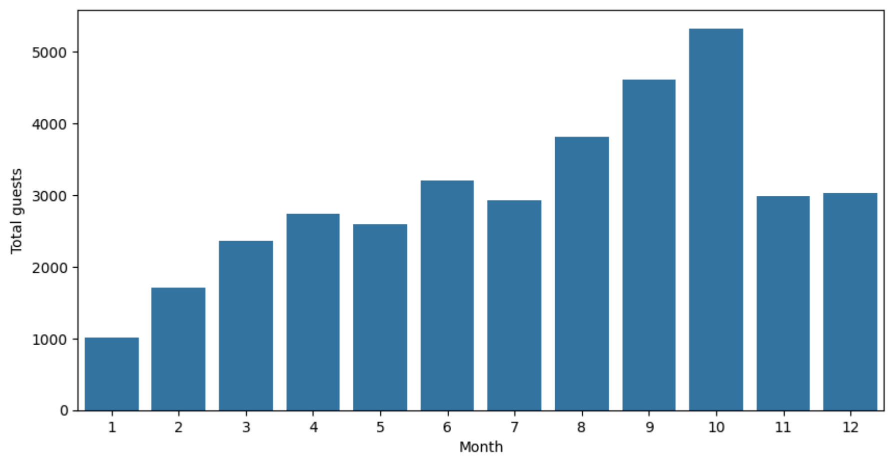
```
room_type_reserved    Room_Type 1  Room_Type 2  Room_Type 3  Room_Type 4  \
no_of_weekend_nights
All                         28130          692            7         6057
1                            7732          173            4         1705
0                           13493          286            3         2456
2                            6685          229            0         1807
3                             102            4            0           38
4                              77            0            0           39
5                              26            0            0            7
6                              14            0            0            5
7                               1            0            0            0

room_type_reserved    Room_Type 5  Room_Type 6  Room_Type 7    All
no_of_weekend_nights
All                           265          966          158  36275
1                              86          248           47   9995
0                             125          432           77  16872
2                              50          267           33   9071
3                               3            6            0    153
4                               0           12            1    129
5                               0            1            0     34
6                               1            0            0     20
7                               0            0            0      1
------------------------------------------------------------------------------------------------------------------------
```



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?

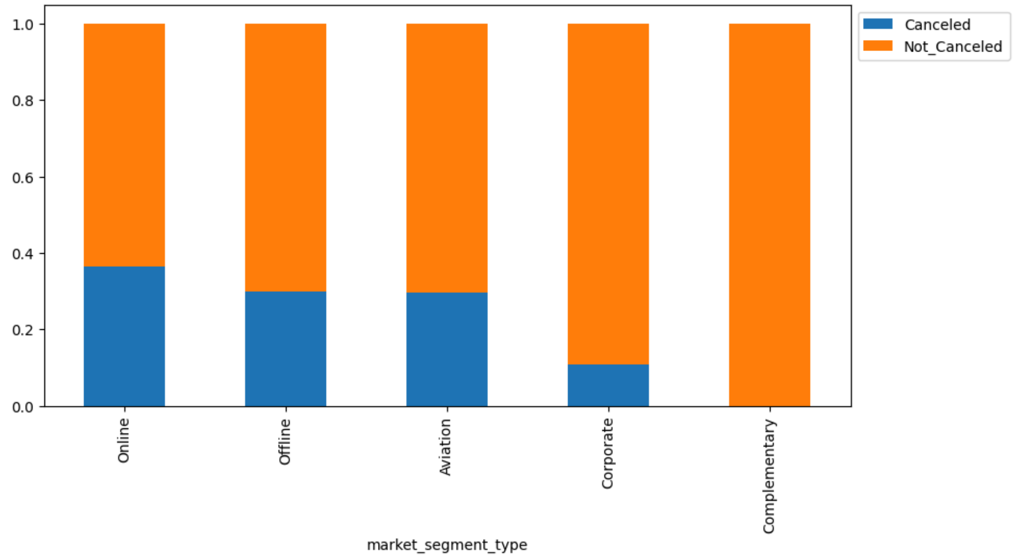
```
#1 - What are the busiest months in the hotel?
data = df.groupby(["arrival_month"])["booking_status"].count()
data = pd.DataFrame({"Month": list(data.index), "Total guests": list(data.values)})

plt.figure(figsize=(10, 5))
sns.barplot(data=data, x="Month", y="Total guests")
plt.show()
```



Observations: October is the busiest month

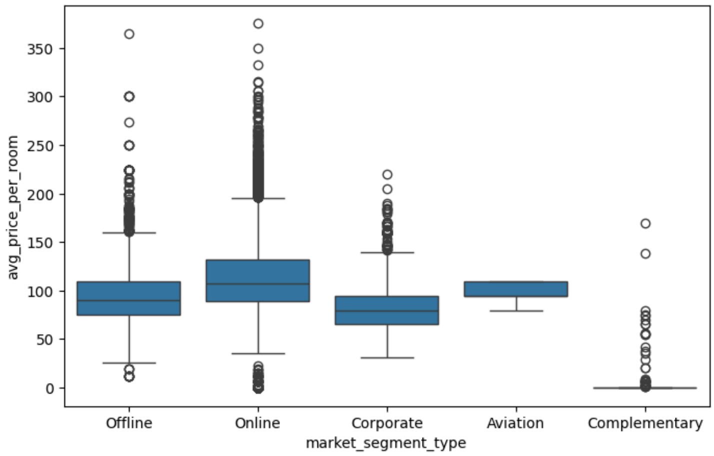
```
#2. Which market segment do most of the guests come from?
stacked_barplot(df, "market_segment_type", "booking_status")
```

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



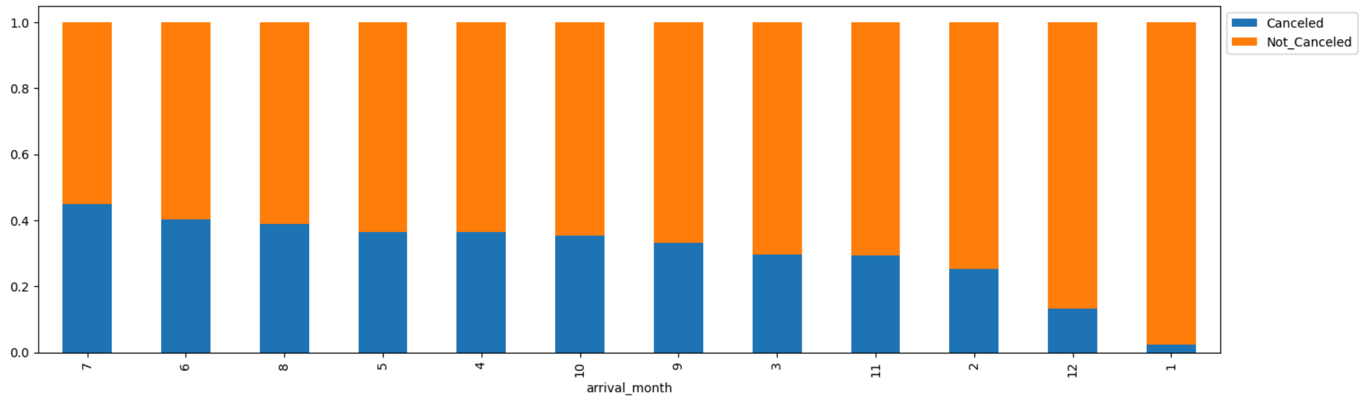
Observations: Online is the market segment most of the guests come from

```
#3. Hotel rates are dynamic and change according to demand and customer demographics. What are the differences in room prices in different market segments?
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x="market_segment_type", y="avg_price_per_room")
plt.show()
```



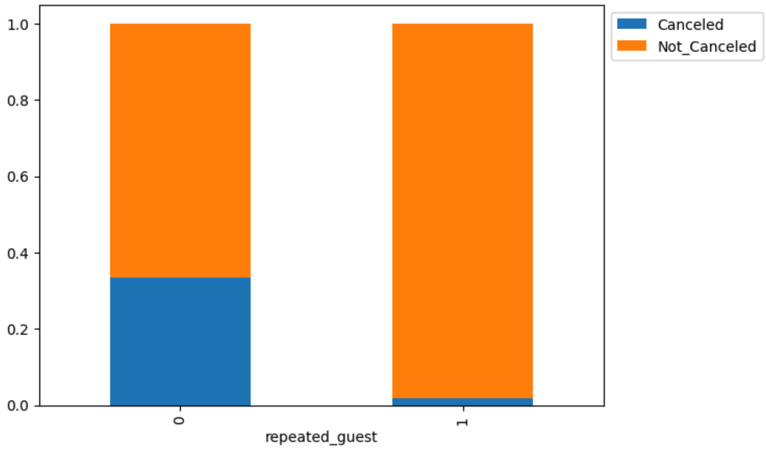

```
#4. What percentage of bookings are canceled?
stacked_barplot(df, "arrival_month", "booking_status")
```

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



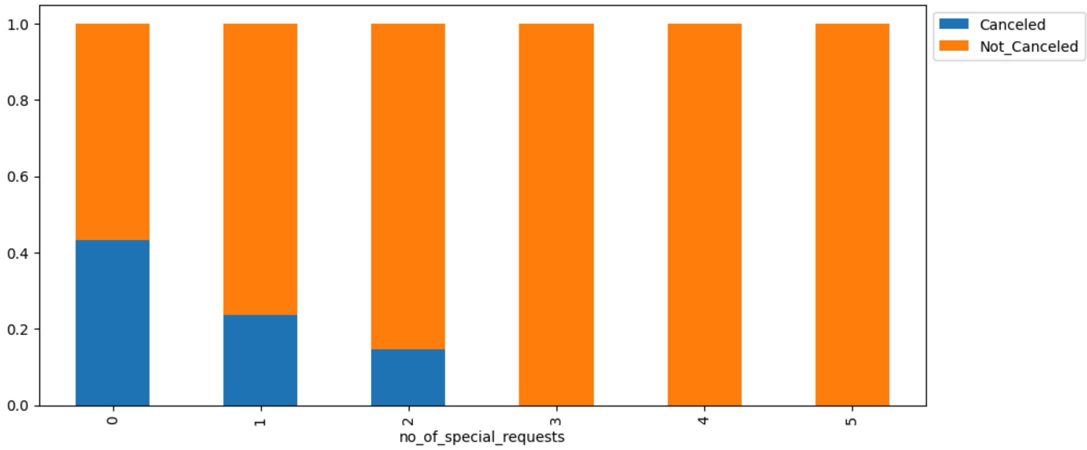

```
#5. Repeating guests are the guests who stay in the hotel often and are important to brand equity. What percentage of repeating guests cancel?
stacked_barplot(df, "repeated_guest", "booking_status")
```

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




```
#6. Many guests have special requirements when booking a hotel room. Do these requirements affect booking cancellation?
stacked_barplot(df, "no_of_special_requests", "booking_status")
```

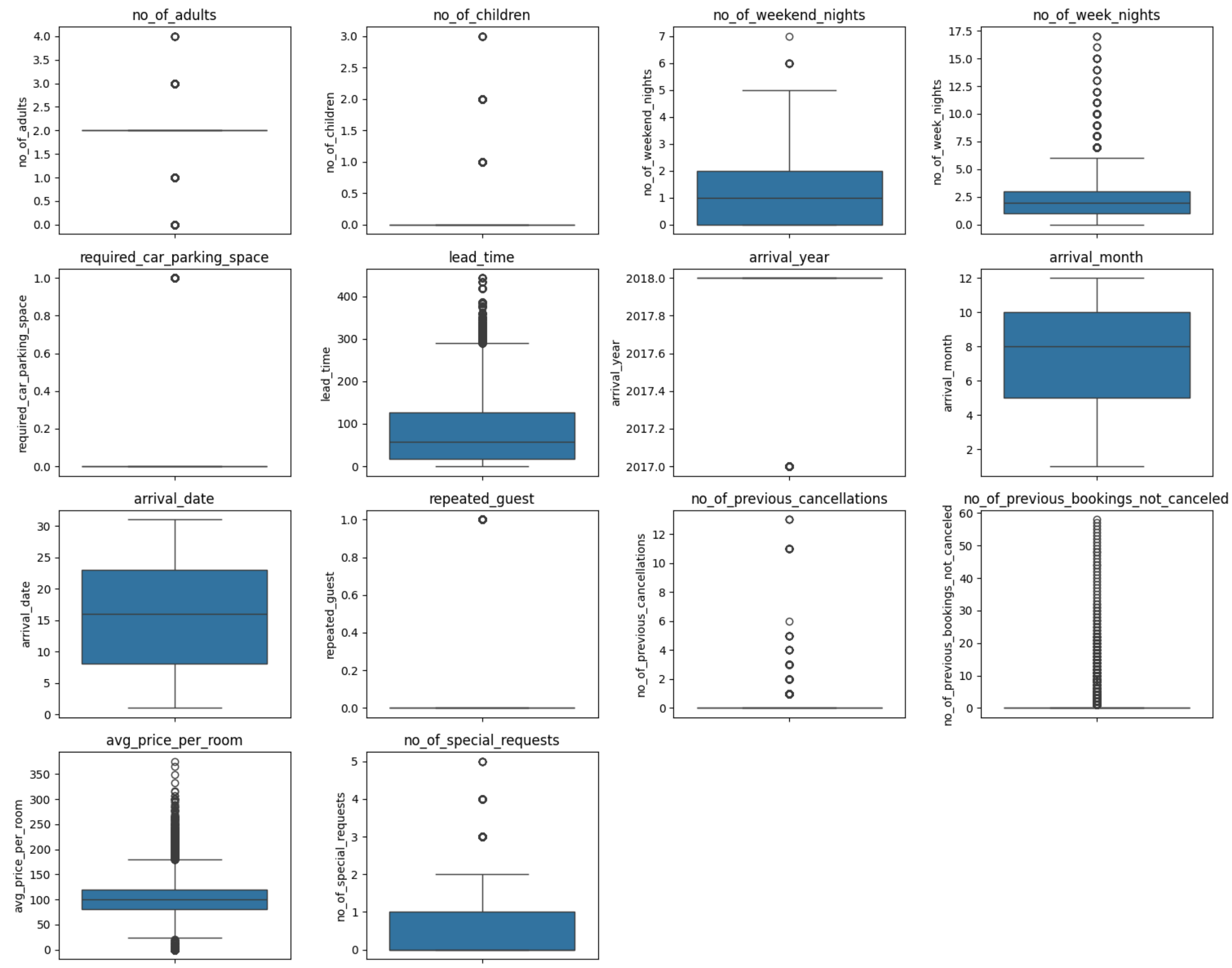
```
booking_status          Canceled  Not_Canceled    All
no_of_special_requests
All                        11885         24390  36275
0                           8545         11232  19777
1                           2703          8670  11373
2                            637          3727   4364
3  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



Data Preprocessing

- Missing value treatment (if needed)
- Feature engineering (if needed)
- Outlier detection and treatment (if needed)
- Preparing data for modeling
- Any other preprocessing steps (if needed)

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


num_cols = df.select_dtypes(include=np.number).columns.tolist()
# drop booking_status
num_cols.remove("booking_status")

plt.figure(figsize=(15, 12))

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

plt.show()
```



EDA

- It is a good idea to explore the data once again after manipulating it.

```
df.shape
```

```
(36275, 18)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 18 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   no_of_adults                          36275 non-null  int64  
 1   no_of_children                        36275 non-null  int64  
 2   no_of_weekend_nights                  36275 non-null  int64  
 3   no_of_week_nights                     36275 non-null  int64  
 4   type_of_meal_plan                     36275 non-null  object 
 5   required_car_parking_space            36275 non-null  int64  
 6   room_type_reserved                    36275 non-null  object 
 7   lead_time                             36275 non-null  int64  
 8   arrival_year                          36275 non-null  int64  
 9   arrival_month                         36275 non-null  int64  
 10  arrival_date                          36275 non-null  int64  
 11  market_segment_type                   36275 non-null  object 
 12  repeated_guest                        36275 non-null  int64  
 13  no_of_previous_cancellations          36275 non-null  int64  
 14  no_of_previous_bookings_not_canceled  36275 non-null  int64  
 15  avg_price_per_room                    36275 non-null  float64
 16  no_of_special_requests                36275 non-null  int64  
 17  booking_status                        36275 non-null  int64  
dtypes: float64(1), int64(14), object(3)
memory usage: 5.0+ MB
```

```
df.head()
```

	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	arrival_year	arriva
0	2	0	1	2	Meal Plan 1	0	Room_Type 1	224	2017	
1	2	0	2	3	Not Selected	0	Room_Type 1	5	2018	
2	1	0	2	1	Meal Plan 1	0	Room_Type 1	1	2018	
3	2	0	0	2	Meal Plan 1	0	Room_Type 1	211	2018	
4	2	0	1	1	Not Selected	0	Room_Type 1	48	2018	

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.describe()
```

	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	repeated_g
count	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.00000	36275.0
mean	1.84496	0.10476	0.81072	2.20430	0.03099	85.23256	2017.82043	7.42365	15.59700	0.0
std	0.51871	0.39466	0.87064	1.41090	0.17328	85.93082	0.38384	3.06989	8.74045	0.1
min	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	2017.00000	1.00000	1.00000	0.0
25%	2.00000	0.00000	0.00000	1.00000	0.00000	17.00000	2018.00000	5.00000	8.00000	0.0
50%	2.00000	0.00000	1.00000	2.00000	0.00000	57.00000	2018.00000	8.00000	16.00000	0.0
75%	2.00000	0.00000	2.00000	3.00000	0.00000	126.00000	2018.00000	10.00000	23.00000	0.0
max	4.00000	3.00000	7.00000	17.00000	1.00000	443.00000	2018.00000	12.00000	31.00000	1.0

```
# check for missing values
df.isnull().sum()
```

	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

Logistic Regression

```
#function copied from "Session Notebook - WHO Case Study" session
# 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
```

```
#function copied from "Session Notebook - WHO Case Study" session
# 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")
```

Data Preparation for Modeling

- We want to predict if bookings will be canceled

```
X = df.select_dtypes(include=['number'])

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

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

# Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30, random_state=1)
```

```
print("Shape of Training set : ", X_train.shape)
print("Shape of test set : ", X_test.shape)
```

```
Shape of Training set :  (25392, 15)
Shape of test set :  (10883, 15)
```

```
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))
```

```
Percentage of classes in training set:
booking_status
0   0.67064
1   0.32936
Name: proportion, dtype: float64
Percentage of classes in test set:
booking_status
0   0.67638
1   0.32362
Name: proportion, dtype: float64
```

Building a Logistic Regression model

```
# 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:                    25377
Method:                           MLE   Df Model:                           14
Date:                Sat, 22 Feb 2025   Pseudo R-squ.:                  0.2737
Time:                        15:05:44   Log-Likelihood:                -11688.
converged:                       True   LL-Null:                       -16091.
Covariance Type:            nonrobust   LLR p-value:                     0.000
========================================================================================================
                                           coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------
const                                -1832.4436    108.550    -16.881      0.000   -2045.197   -1619.690
no_of_adults                             0.1948      0.034      5.717      0.000       0.128       0.262
no_of_children                           0.0384      0.044      0.874      0.382      -0.048       0.125
no_of_weekend_nights                     0.1415      0.019      7.565      0.000       0.105       0.178
no_of_week_nights                        0.0532      0.012      4.578      0.000       0.030       0.076
required_car_parking_space              -1.3107      0.134     -9.806      0.000      -1.573      -1.049
lead_time                                0.0123      0.000     55.746      0.000       0.012       0.013
arrival_year                             0.9063      0.054     16.849      0.000       0.801       1.012
arrival_month                           -0.0275      0.006     -4.544      0.000      -0.039      -0.016
arrival_date                             0.0019      0.002      1.021      0.307      -0.002       0.006
repeated_guest                          -2.2201      0.526     -4.225      0.000      -3.250      -1.190
no_of_previous_cancellations             0.2552      0.092      2.761      0.006       0.074       0.436
no_of_previous_bookings_not_canceled    -0.1978      0.164     -1.204      0.229      -0.520       0.124
avg_price_per_room                       0.0178      0.001     29.875      0.000       0.017       0.019
no_of_special_requests                  -1.1064      0.027    -41.488      0.000      -1.159      -1.054
========================================================================================================
```

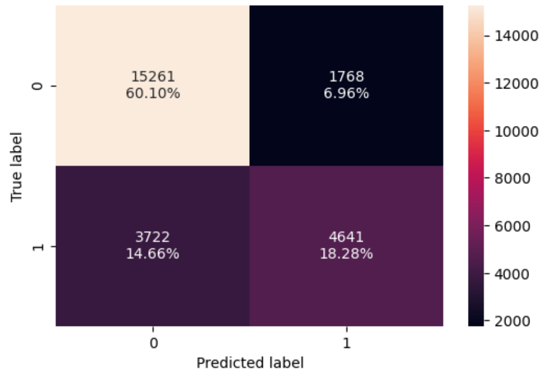
```
print("Training performance:")
model_performance_classification_statsmodels(lg, X_train, y_train)
```

Training performance:

	Accuracy	Recall	Precision	F1
0	0.78379	0.55494	0.72414	0.62835

```
print("Confusion matrix:")
confusion_matrix_statsmodels(lg, X_train, y_train)
```

Confusion matrix:



Checking Multicollinearity

- In order to make statistical inferences from a logistic regression model, it is important to ensure that there is no multicollinearity present in the data.

```
# Function to check VIF
def checking_vif(predictors):
    vif = pd.DataFrame()
    vif["feature"] = predictors.columns

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

```
checking_vif(X_train)
```

	feature	VIF
0	const	34866123.37597
1	no_of_adults	1.21461
2	no_of_children	1.17572
3	no_of_weekend_nights	1.05287
4	no_of_week_nights	1.06931
5	required_car_parking_space	1.03415
6	lead_time	1.15845
7	arrival_year	1.26424
8	arrival_month	1.24097
9	arrival_date	1.00486
10	repeated_guest	1.56324
11	no_of_previous_cancellations	1.37579
12	no_of_previous_bookings_not_canceled	1.63420
13	avg_price_per_room	1.39658
14	no_of_special_requests	1.11675

Drop high p-value

```
# 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_weekend_nights', 'no_of_week_nights', 'required_car_parking_space', 'lead_time', 'arrival_year', 'arrival_month', 'repeated_guest', 'no_of
```

```
X_train1 = X_train[selected_features]
X_test1 = X_test[selected_features]


# fitting logistic regression model
logit1 = sm.Logit(y_train, X_train1.astype(float))
lg1 = logit1.fit(disp=False)

print(lg1.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:         booking_status   No. Observations:                25392
Model:                          Logit   Df Residuals:                    25380
Method:                           MLE   Df Model:                           11
Date:                Sat, 22 Feb 2025   Pseudo R-squ.:                  0.2735
Time:                        15:05:46   Log-Likelihood:                -11691.
converged:                       True   LL-Null:                       -16091.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================================
                                   coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                        -1828.8682    108.606    -16.840      0.000   -2041.732   -1616.005
no_of_adults                     0.1905      0.034      5.682      0.000       0.125       0.256
no_of_weekend_nights             0.1429      0.019      7.656      0.000       0.106       0.179
no_of_week_nights                0.0533      0.012      4.590      0.000       0.031       0.076
required_car_parking_space      -1.3089      0.133     -9.805      0.000      -1.571      -1.047
lead_time                        0.0124      0.000     55.866      0.000       0.012       0.013
arrival_year                     0.9046      0.054     16.807      0.000       0.799       1.010
arrival_month                   -0.0280      0.006     -4.637      0.000      -0.040      -0.016
repeated_guest                  -2.6583      0.460     -5.774      0.000      -3.561      -1.756
no_of_previous_cancellations     0.2101      0.077      2.727      0.006       0.059       0.361
avg_price_per_room               0.0181      0.001     32.838      0.000       0.017       0.019
no_of_special_requests          -1.1046      0.027    -41.551      0.000      -1.157      -1.053
================================================================================================
```

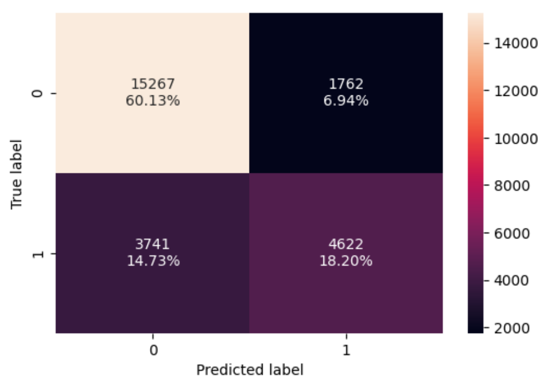
Converting coefficients to odds

```
odds = np.exp(lg1.params)

# finding the percentage change
perc_change_odds = (np.exp(lg1.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_train1.columns).T
```

	const	no_of_adults	no_of_weekend_nights	no_of_week_nights	required_car_parking_space	lead_time	arrival_year	arrival_month	repeated_guest	no_of_p
Odds	0.00000	1.20985	1.15361	1.05479	0.27013	1.01244	2.47090	0.97240	0.07007	
Change_odd%	100.00000	20.98540	15.36140	5.47856	72.98728	1.24405	147.08978	2.75999	92.99294	

Model performance evaluation

```
# creating confusion matrix
confusion_matrix_statsmodels(lg1, X_train1, y_train)
```




```
print("Training performance:")
log_reg_model_train_perf = model_performance_classification_statsmodels(lg1, X_train1, y_train)
log_reg_model_train_perf
```

Training performance:

	Accuracy	Recall	Precision	F1
0	0.78328	0.55267	0.72400	0.62684

```
print("Testing performance:")
log_reg_model_test_perf = model_performance_classification_statsmodels(lg1, X_test1, y_test)
log_reg_model_test_perf
```

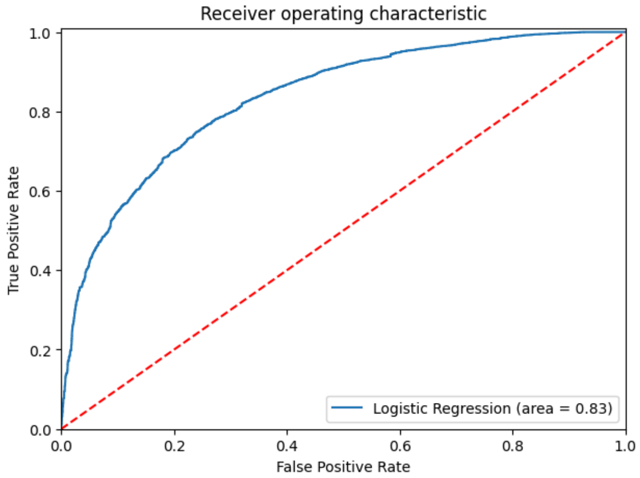
Testing performance:

	Accuracy	Recall	Precision	F1
0	0.79059	0.56360	0.72791	0.63530

ROC Curve and ROC-AUC

ROC-AUC on training set

```
logit_roc_auc_train = roc_auc_score(y_train, lg1.predict(X_train1))
fpr, tpr, thresholds = roc_curve(y_train, lg1.predict(X_train1))
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()
```



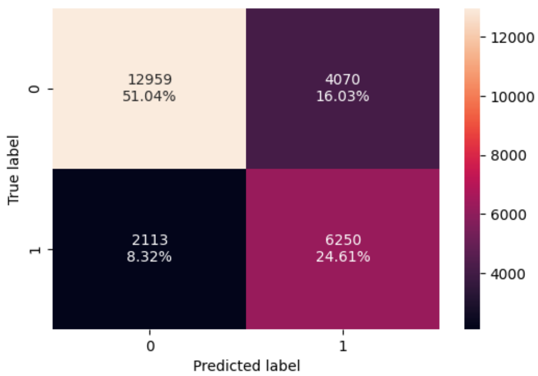
Optimal threshold using AUC-ROC curve

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

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

```
0.33084253659807417
```

Checking model performance on training set

```
# creating confusion matrix
confusion_matrix_statsmodels(
    lg1, X_train1, y_train, threshold=optimal_threshold_auc_roc
)
```



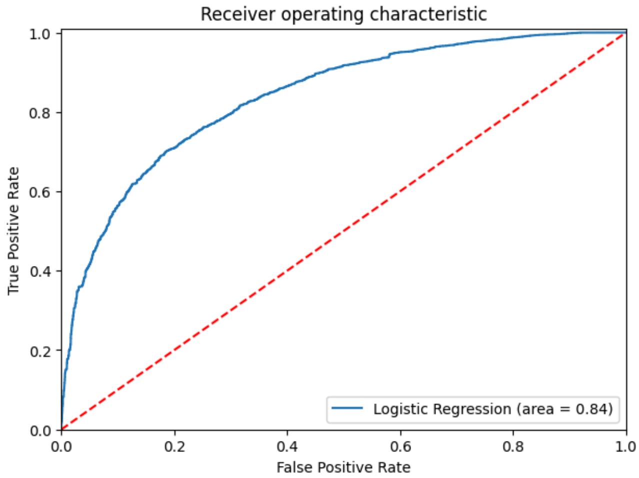

```
# checking model performance for this model
log_reg_model_train_perf_threshold_auc_roc = model_performance_classification_statsmodels(
    lg1, X_train1, y_train, threshold=optimal_threshold_auc_roc
)
print("Training performance:")
log_reg_model_train_perf_threshold_auc_roc
```

Training performance:

	Accuracy	Recall	Precision	F1
0	0.75650	0.74734	0.60562	0.66906

ROC-AUC on test set

```
logit_roc_auc_train = roc_auc_score(y_test, lg1.predict(X_test1))
fpr, tpr, thresholds = roc_curve(y_test, lg1.predict(X_test1))
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()
```



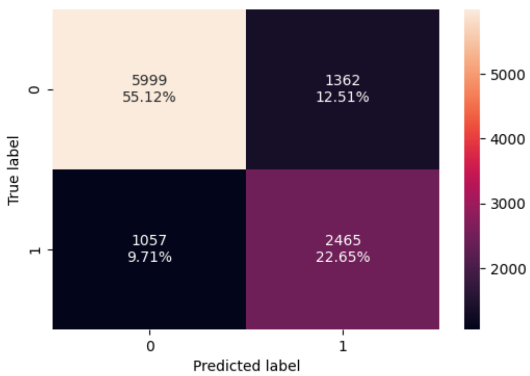
Optimal threshold using AUC-ROC curve

```
# Optimal threshold as per AUC-ROC curve
# The optimal cut off would be where tpr is high and fpr is low
fpr, tpr, thresholds = roc_curve(y_test, lg1.predict(X_test1))

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

```
0.3758934440156464
```

Checking model performance on testing set

```
# creating confusion matrix
confusion_matrix_statsmodels(
    lg1, X_test1, y_test, threshold=optimal_threshold_auc_roc
)
```



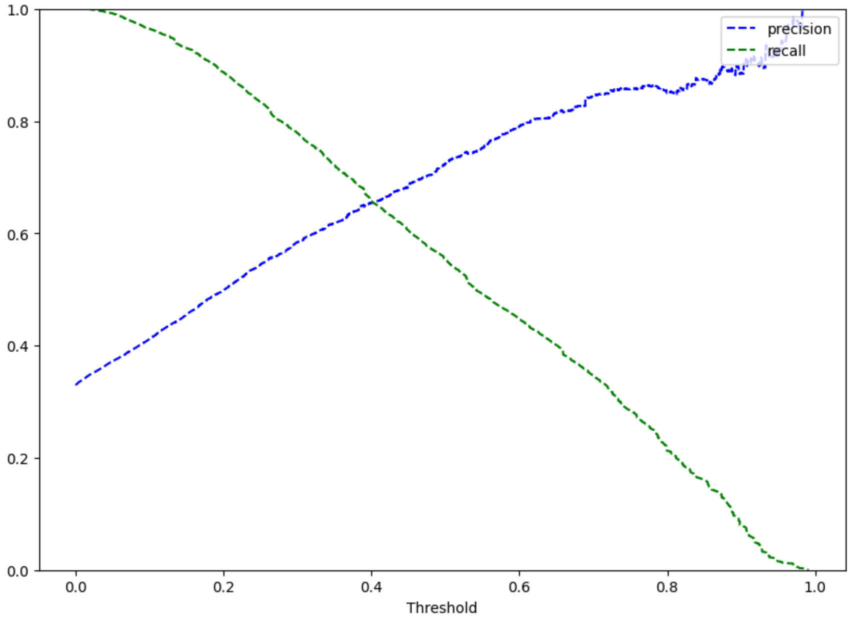

```
# checking model performance for this model
log_reg_model_test_perf_threshold_auc_roc = model_performance_classification_statsmodels(
    lg1, X_test1, y_test, threshold=optimal_threshold_auc_roc
)
print("Testing performance:")
log_reg_model_test_perf_threshold_auc_roc
```

Testing performance:

	Accuracy	Recall	Precision	F1
0	0.77773	0.69989	0.64411	0.67084

Precision-Recall Curve

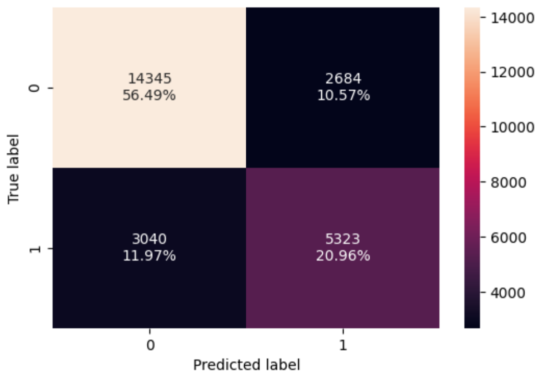
```
y_scores = lg1.predict(X_train1)
prec, rec, tre = precision_recall_curve(y_train, y_scores,)


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 right")
    plt.ylim([0, 1])


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




```
# setting the threshold
optimal_threshold_curve = 0.42
```

Checking model performance on training set

```
# creating confusion matrix
confusion_matrix_statsmodels(lg1, X_train1, y_train, threshold=optimal_threshold_curve)
```



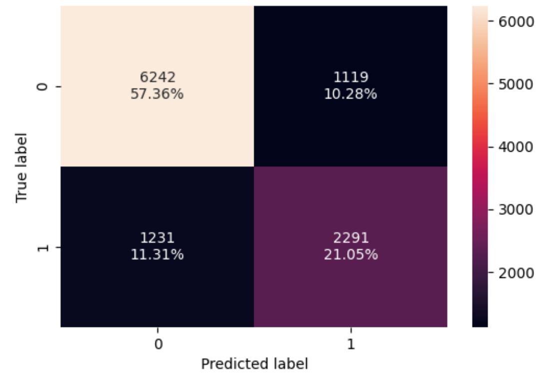

```
log_reg_model_train_perf_threshold_curve = model_performance_classification_statsmodels(
    lg1, X_train1, y_train, threshold=optimal_threshold_curve
)
print("Training performance:")
log_reg_model_train_perf_threshold_curve
```

Training performance:

	Accuracy	Recall	Precision	F1
0	0.77457	0.63649	0.66479	0.65034

Checking model performance on test set

```
# creating confusion matrix
confusion_matrix_statsmodels(lg1, X_test1, y_test, threshold=optimal_threshold_curve)
```




```
log_reg_model_test_perf_threshold_curve = model_performance_classification_statsmodels(
    lg1, X_test1, y_test, threshold=optimal_threshold_curve
)
print("Test performance:")
log_reg_model_test_perf_threshold_curve
```

Test performance:

	Accuracy	Recall	Precision	F1
0	0.78407	0.65048	0.67185	0.66099

Final Model Summary

```
# 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:

	Logistic Regression-default Threshold	Logistic Regression-0.37 Threshold	Logistic Regression-0.42 Threshold
Accuracy	0.78328	0.75650	0.77457
Recall	0.55267	0.74734	0.63649
Precision	0.72400	0.60562	0.66479
F1	0.62684	0.66906	0.65034

Next steps: Generate code with `models_train_comp_df` View recommended plots New interactive sheet

```
# testing 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 set performance comparison:")
models_test_comp_df
```

Test set performance comparison:

	Logistic Regression-default Threshold	Logistic Regression-0.37 Threshold	Logistic Regression-0.42 Threshold
Accuracy	0.79059	0.77773	0.78407
Recall	0.56360	0.69989	0.65048
Precision	0.72791	0.64411	0.67185
F1	0.63530	0.67084	0.66099

Next steps: Generate code with `models_test_comp_df` View recommended plots New interactive sheet

Building a Decision Tree model

Data preparation

```
X = df.select_dtypes(include=['number'])

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

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

# Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30, random_state=1)
```

```
print("Shape of Training set : ", X_train.shape)
print("Shape of test set : ", X_test.shape)
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))
```

```
Shape of Training set :  (25392, 15)
Shape of test set :  (10883, 15)
Percentage of classes in training set:
booking_status
0   0.67064
1   0.32936
Name: proportion, dtype: float64
Percentage of classes in test set:
booking_status
0   0.67638
1   0.32362
Name: proportion, dtype: float64
```

Functions

```
#function copied from "Session Notebook - Machine Failure Prediction" session
# 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
```

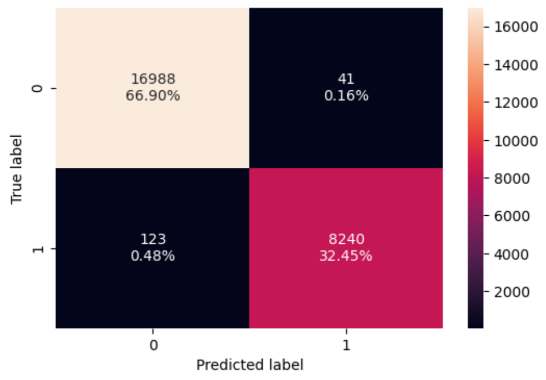
```
#function copied from "Session Notebook - Machine Failure Prediction" session
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    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")
```

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

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

Checking performance on training set

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

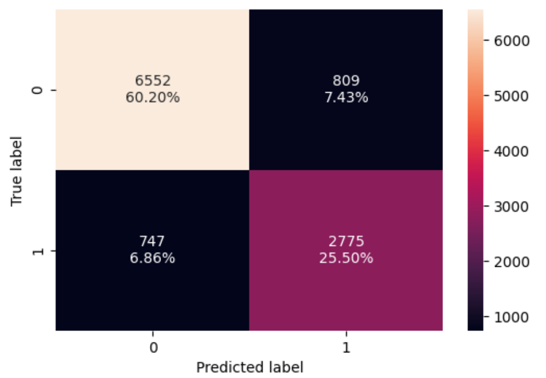



```
decision_tree_perf_train = model_performance_classification_sklearn(
    model, X_train, y_train
)
decision_tree_perf_train
```

	Accuracy	Recall	Precision	F1
0	0.99354	0.98529	0.99505	0.99015

Checking performance on test set

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




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

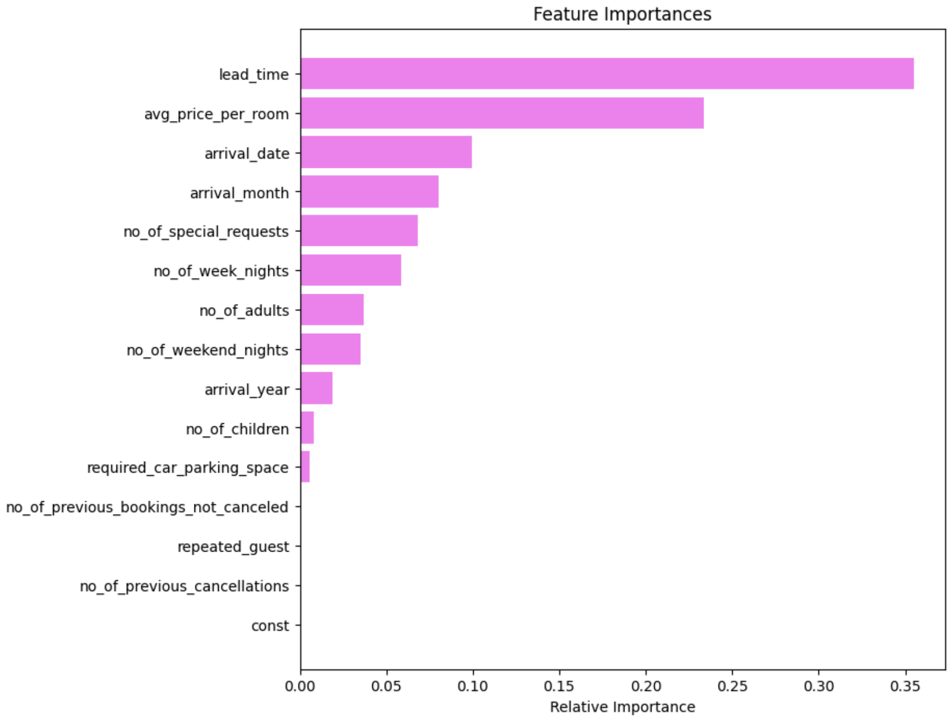
	Accuracy	Recall	Precision	F1
0	0.85702	0.78790	0.77427	0.78103

Checking importance feature

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

```
['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', 'arriv
```

```
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()
```



Do we need to prune the tree?

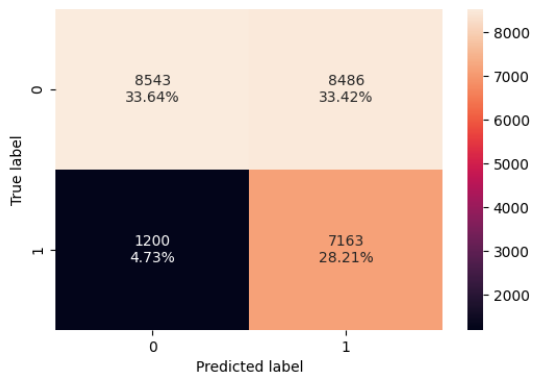
```
# Choose the type of classifier.
estimator = DecisionTreeClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    "class_weight": [None, "balanced"],
    "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(recall_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)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', max_depth=2, max_leaf_nodes=50,
                       min_samples_split=10, random_state=1)
```

Checking performance on training set

Start coding or generate with AI.

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

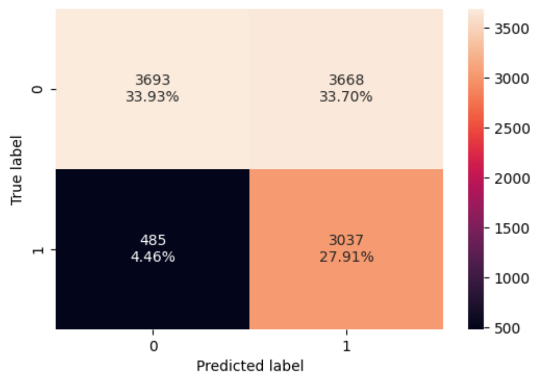



```
decision_tree_tune_perf_train = model_performance_classification_sklearn(estimator, X_train, y_train)
decision_tree_tune_perf_train
```

	Accuracy	Recall	Precision	F1
0	0.61854	0.85651	0.45773	0.59662

Checking performance on test set

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

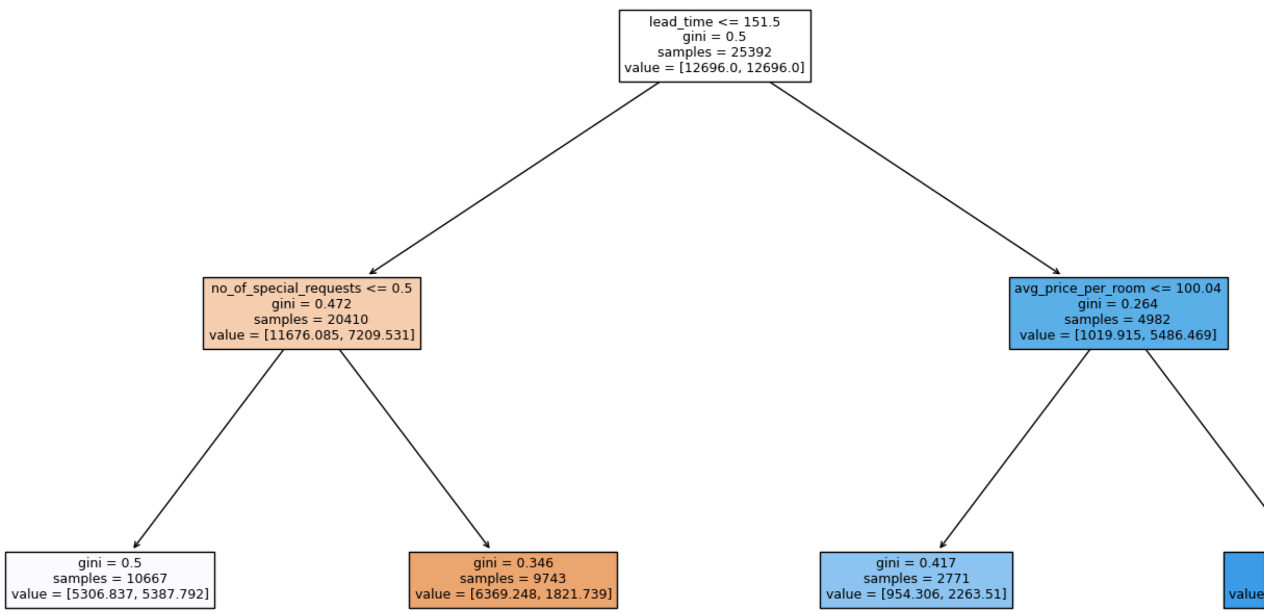



```
decision_tree_tune_perf_test = model_performance_classification_sklearn(estimator, X_test, y_test)
decision_tree_tune_perf_test
```

	Accuracy	Recall	Precision	F1
0	0.61840	0.86229	0.45295	0.59392

Visualizing the Decision Tree

```
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()
```




```
# 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
|   |   |--- weights: [5306.84, 5387.79] class: 1
|   |--- no_of_special_requests >  0.50
|   |   |--- weights: [6369.25, 1821.74] class: 0
|--- lead_time >  151.50
|   |--- avg_price_per_room <= 100.04
|   |   |--- weights: [954.31, 2263.51] class: 1
|   |--- avg_price_per_room >  100.04
|   |   |--- weights: [65.61, 3222.96] class: 1
```

Decision Tree (Post pruning)

Total impurity of leaves vs effective alphas of pruned tree

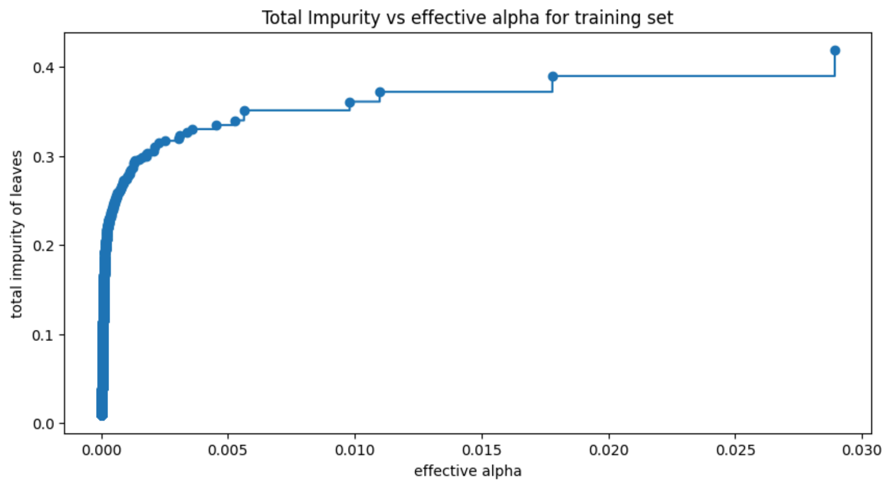
```
clf = DecisionTreeClassifier(random_state=1, class_weight="balanced")
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = abs(path.ccp_alphas), path.impurities
```

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

	ccp_alphas	impurities
0	0.00000	0.00918
1	0.00000	0.00918
2	0.00000	0.00918
3	0.00000	0.00918
4	0.00000	0.00918
...	...	...
2084	0.00980	0.36112
2085	0.01099	0.37210
2086	0.01779	0.38990
2087	0.02893	0.41882
2088	0.08118	0.50000

2089 rows × 2 columns

```
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()
```

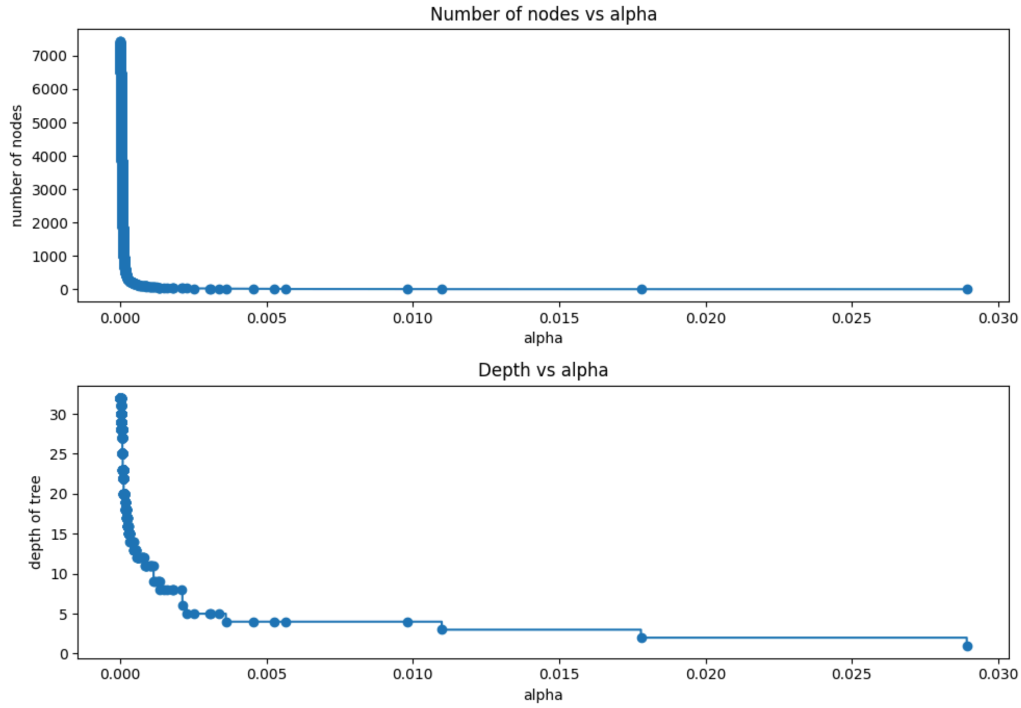



```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(
        random_state=1, ccp_alpha=ccp_alpha, class_weight="balanced"
    )
    clf.fit(X_train, y_train)
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count,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

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.08117914389137049
```

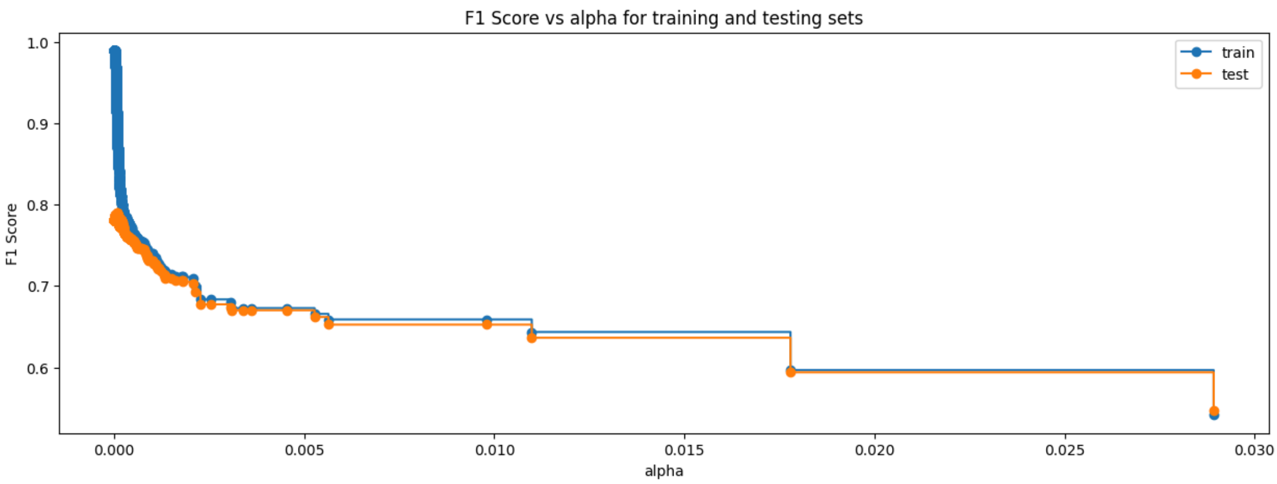
```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

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_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

```
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)


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()
```



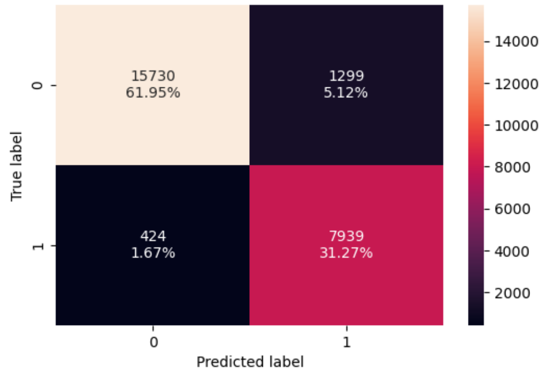

```
index_best_model = np.argmax(f1_test)
best_model = clfs[index_best_model]
print(best_model)
```

```
DecisionTreeClassifier(ccp_alpha=7.329852678870992e-05, class_weight='balanced',
                       random_state=1)
```

Model Performance Comparison and Conclusions

Checking performance on training set

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

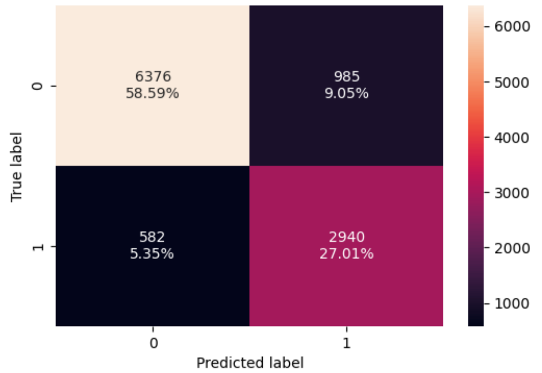



```
decision_tree_post_perf_train = model_performance_classification_sklearn(best_model, X_train, y_train)
decision_tree_post_perf_train
```

	Accuracy	Recall	Precision	F1
0	0.93214	0.94930	0.85939	0.90211

Checking performance on test set

```
confusion_matrix_sklearn(best_model, X_test, y_test)
```




```
decision_tree_post_perf_test = model_performance_classification_sklearn(best_model, X_test, y_test)
decision_tree_post_perf_test
```

	Accuracy	Recall	Precision	F1
0	0.85601	0.83475	0.74904	0.78958

Post-pruned tree

```
#print decision tree with max_depth=3 to accommodate image size
plt.figure(figsize=(20, 10))
out = tree.plot_tree(
    best_model,
    feature_names=feature_names,
    filled=True,
```