Project - Classification and Hypothesis Testing: Hotel Booking Cancellation Prediction

Marks: 40

Problem Statement

Context

A significant number of hotel bookings are called off due to cancellations or no-shows. 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. This may be 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.

This pattern of cancellations of bookings impacts 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

This 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 this 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 weekday 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 the libraries required

In [1]:
```
# Importing the basic libraries we will require for the project

# 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
sns.set()

# Importing the Machine Learning models we require from Scikit-Learn
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

# Importing the other functions we may require from Scikit-Learn
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
```

```
# To get diferent metric scores
from sklearn.metrics import confusion_matrix,classification_report,roc_auc_score,plot_co

# Code to ignore warnings from function usage
import warnings;
import numpy as np
warnings.filterwarnings('ignore')
```

Loading the dataset

In [2]:
```
hotel = pd.read_csv("INNHotelsGroup.csv")
```

In [3]:
```
# Copying data to another variable to avoid any changes to original data
data = hotel.copy()
```

Overview of the dataset

View the first and last 5 rows of the dataset

Let's **view the first few rows and last few rows** of the dataset in order to understand its structure a little better.

We will use the head() and tail() methods from Pandas to do this.

In [4]:
```
data.head()
```

Out[4]:

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_pla
0	INN00001	2	0	1	2	Meal Plan
1	INN00002	2	0	2	3	Not Selecte
2	INN00003	1	0	2	1	Meal Plan
3	INN00004	2	0	0	2	Meal Plan
4	INN00005	2	0	1	1	Not Selecte

In [5]:
```
data.tail()
```

Out[5]:

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_mea
36270	INN36271	3	0	2	6	Meal
36271	INN36272	2	0	1	3	Meal
36272	INN36273	2	0	2	6	Meal
36273	INN36274	2	0	0	3	Not Se
36274	INN36275	2	0	1	2	Meal

Understand the shape of the dataset

In [6]:
```
data.shape
```

Out[6]:
```
(36275, 19)
```

- The dataset has 36275 rows and 19 columns.

Check the data types of the columns for the dataset

In [7]:
```
data.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
```

- `Booking_ID`, `type_of_meal_plan`, `room_type_reserved`, `market_segment_type`, and `booking_status` are of object type while rest columns are numeric in nature.
- There are no null values in the dataset.

Dropping duplicate values

In [8]:
```
# checking for duplicate values
data.duplicated().sum()
```

Out[8]:
```
0
```

- There are **no duplicate values** in the data.

Dropping the unique values column

Let's drop the Booking_ID column first before we proceed forward, as a column with unique values will have almost no predictive power for the Machine Learning problem at hand.

In [9]:
```
data = data.drop(["Booking_ID"], axis=1)
```

In [10]:
```
data.head()
```

Out[10]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_c
0	2	0	1	2	Meal Plan 1	
1	2	0	2	3	Not Selected	
2	1	0	2	1	Meal Plan 1	
3	2	0	0	2	Meal Plan 1	
4	2	0	1	1	Not Selected	

Question 1: Check the summary statistics of the dataset and write your observations (2 Marks)

Let's check the statistical summary of the data.

In [11]:

```
# Remove _________ and complete the code
data.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%
no_of_adults	36275.0	1.844962	0.518715	0.0	2.0	2.00	2.0
no_of_children	36275.0	0.105279	0.402648	0.0	0.0	0.00	0.0
no_of_weekend_nights	36275.0	0.810724	0.870644	0.0	0.0	1.00	2.0
no_of_week_nights	36275.0	2.204300	1.410905	0.0	1.0	2.00	3.0
required_car_parking_space	36275.0	0.030986	0.173281	0.0	0.0	0.00	0.0
lead_time	36275.0	85.232557	85.930817	0.0	17.0	57.00	126.0
arrival_year	36275.0	2017.820427	0.383836	2017.0	2018.0	2018.00	2018.0
arrival_month	36275.0	7.423653	3.069894	1.0	5.0	8.00	10.0
arrival_date	36275.0	15.596995	8.740447	1.0	8.0	16.00	23.0
repeated_guest	36275.0	0.025637	0.158053	0.0	0.0	0.00	0.0
no_of_previous_cancellations	36275.0	0.023349	0.368331	0.0	0.0	0.00	0.0
no_of_previous_bookings_not_canceled	36275.0	0.153411	1.754171	0.0	0.0	0.00	0.0
avg_price_per_room	36275.0	103.423539	35.089424	0.0	80.3	99.45	120.0
no_of_special_requests	36275.0	0.619655	0.786236	0.0	0.0	0.00	1.0

- The avg_price_per_room has outliers, as the 75th percentile is 120 and the max value is 540.
- The lead_time has also outliers, as the 75th percentile is 126 and the max value is 443.
- The arrival_date is similarly distributed along all the days of the month.
- The number of week_nights and weekend_nights have outliers at the higher end.

Exploratory Data Analysis

Question 2: Univariate Analysis

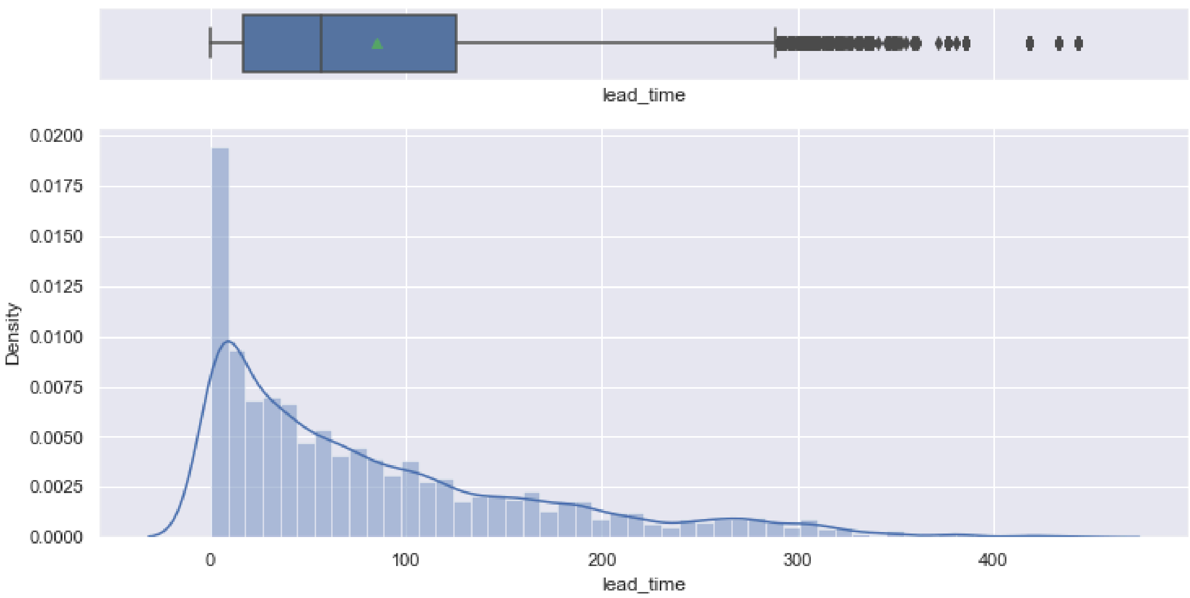
Let's explore these variables in some more depth by observing their distributions.

We will first define a **hist_box() function** that provides both a boxplot and a histogram in the same visual, with which we can perform univariate analysis on the columns of this dataset.

In [12]:
```
# Defining the hist_box() function
def hist_box(data,col):
  f, (ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={'height_ratios': (0.1
  # Adding a graph in each part
  sns.boxplot(data[col], ax=ax_box, showmeans=True)
  sns.distplot(data[col], ax=ax_hist)
  plt.show()
```

Question 2.1: Plot the histogram and box plot for the variable `Lead Time` using the hist_box function provided and write your insights. (1 Mark)

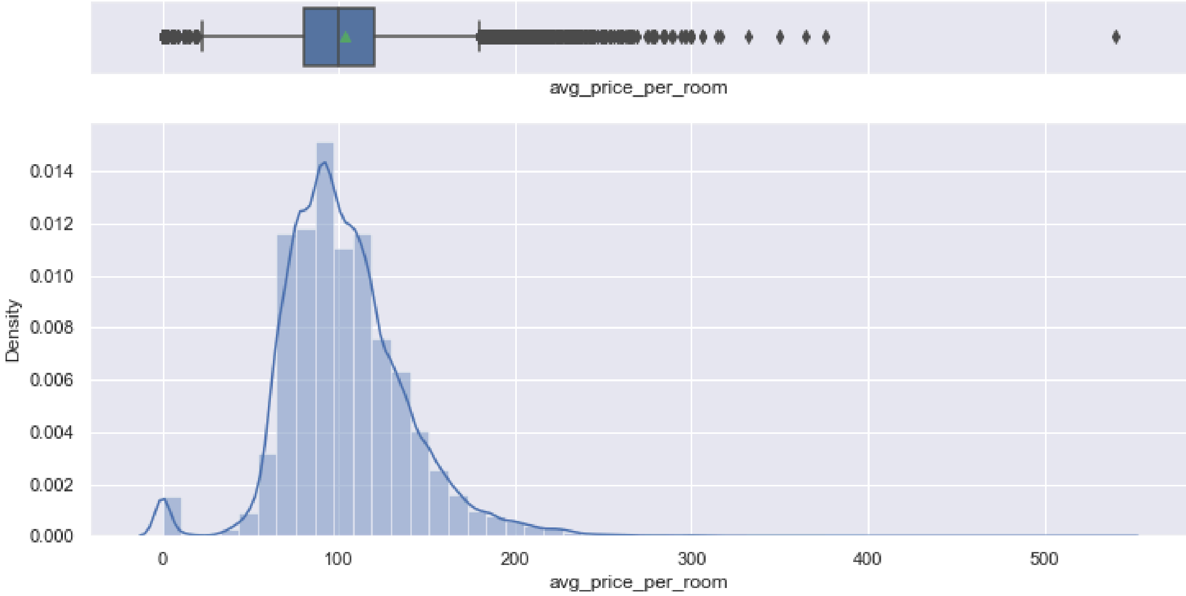
In [13]:
```
# Remove _________ and complete the code
hist_box(data,'lead_time')
```



- As expected from the statistical summarym this variable distribution is right-skewed.
- The are many outliers in the upper region, as shown in the boxplot.
- Most of the guests have a lead time under 200 days.

Question 2.2: Plot the histogram and box plot for the variable `Average Price per Room` using the hist_box function provided and write your insights. (1 Mark)

In [14]:
```
# Remove _________ and complete the code
hist_box(data,'avg_price_per_room')
```



- The average price payed per room is about 100 euros.
- The distribution of the data looks normally distributed.
- There are some outliers, near to zero (maybe some offer from the Hotel), and many outliers in the upper region, with one data point above 500 euros (this is worthy investigating, maybe it´s a mistake, or it could be the presidential suite)

Interestingly some rooms have a price equal to 0. Let's check them.

In [15]:
```
data[data["avg_price_per_room"] == 0]
```

Out[15]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	requir
63	1	0	0	1	Meal Plan 1	
145	1	0	0	2	Meal Plan 1	
209	1	0	0	0	Meal Plan 1	
266	1	0	0	2	Meal Plan 1	
267	1	0	2	1	Meal Plan 1	
...	...	...	...	...	...	
35983	1	0	0	1	Meal Plan 1	
36080	1	0	1	1	Meal Plan 1	
36114	1	0	0	1	Meal Plan 1	
36217	2	0	2	1	Meal Plan 1	
36250	1	0	0	2	Meal Plan 2	

545 rows × 18 columns

- There are quite a few hotel rooms which have a price equal to 0.
- In the market segment column, it looks like many values are complementary.

In [16]:
```
data.loc[data["avg_price_per_room"] == 0, "market_segment_type"].value_counts()
```

Out[16]:
```
Complementary    354
Online           191
Name: market_segment_type, dtype: int64
```

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

In [17]:
```
# Calculating the 25th quantile
Q1 = data["avg_price_per_room"].quantile(0.25)

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

# Calculating IQR
IQR = Q3 - Q1

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

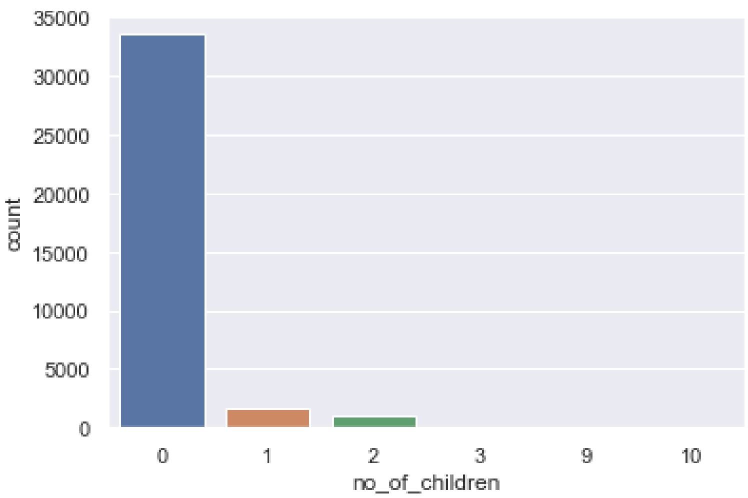
Out[17]:
```
179.55
```

In [18]:
```
# assigning the outliers the value of upper whisker
data.loc[data["avg_price_per_room"] >= 500, "avg_price_per_room"] = Upper_Whisker
```

Let's understand the distribution of the categorical variables

Number of Children

In [19]:
```
sns.countplot(data['no_of_children'])
plt.show()
```



In [20]:
```
data['no_of_children'].value_counts(normalize=True)
```

Out[20]:
```
0     0.925624
1     0.044604
2     0.029166
3     0.000524
9     0.000055
10    0.000028
Name: no_of_children, dtype: float64
```

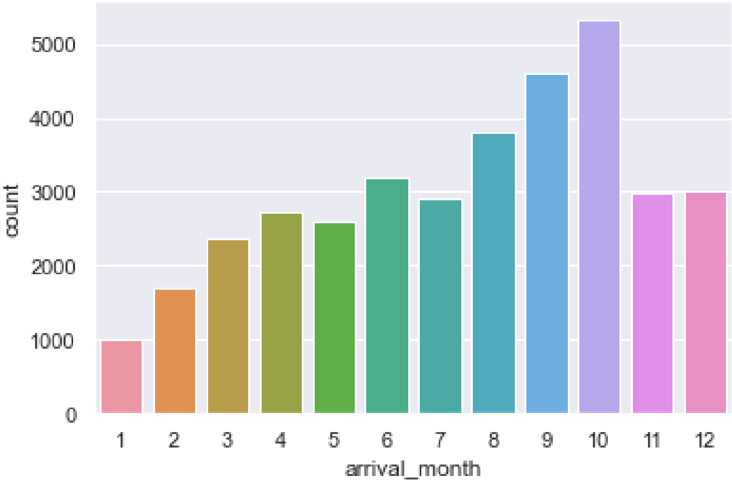
- Customers were not travelling with children in 93% of cases.

- 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 [21]:
```
# replacing 9, and 10 children with 3
data["no_of_children"] = data["no_of_children"].replace([9, 10], 3)
```

Arrival Month

In [22]:
```
sns.countplot(data["arrival_month"])
plt.show()
```



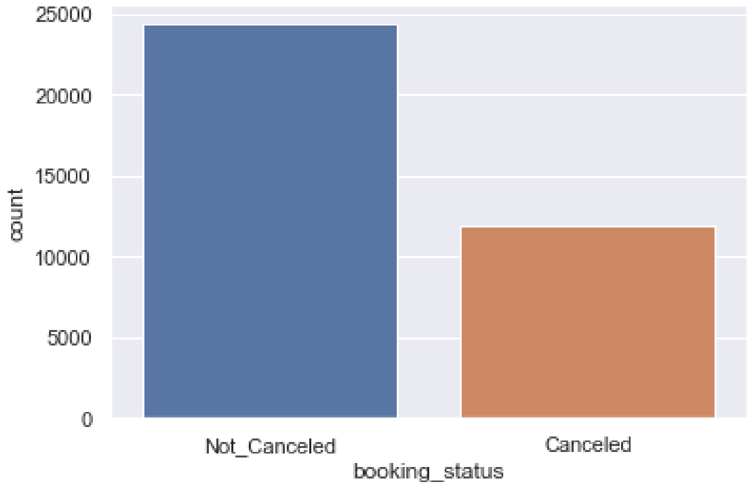
In [23]:
```
data['arrival_month'].value_counts(normalize=True)
```

Out[23]:
```
10    0.146575
9     0.127112
8     0.105114
6     0.088298
12    0.083280
11    0.082150
7     0.080496
4     0.075424
5     0.071620
3     0.065003
2     0.046975
1     0.027953
Name: arrival_month, dtype: float64
```

- October is the busiest month for hotel arrivals followed by September and August. **Over 35% of all bookings**, as we see in the above table, were for one of these three months.
- Around 14.7% of the bookings were made for an October arrival.

Booking Status

In [24]:
```
sns.countplot(data["booking_status"])
plt.show()
```



In [25]:
```
data['booking_status'].value_counts(normalize=True)
```

Out[25]:
```
Not_Canceled    0.672364
Canceled        0.327636
Name: booking_status, dtype: float64
```

- 32.8% of the bookings were canceled by the customers.

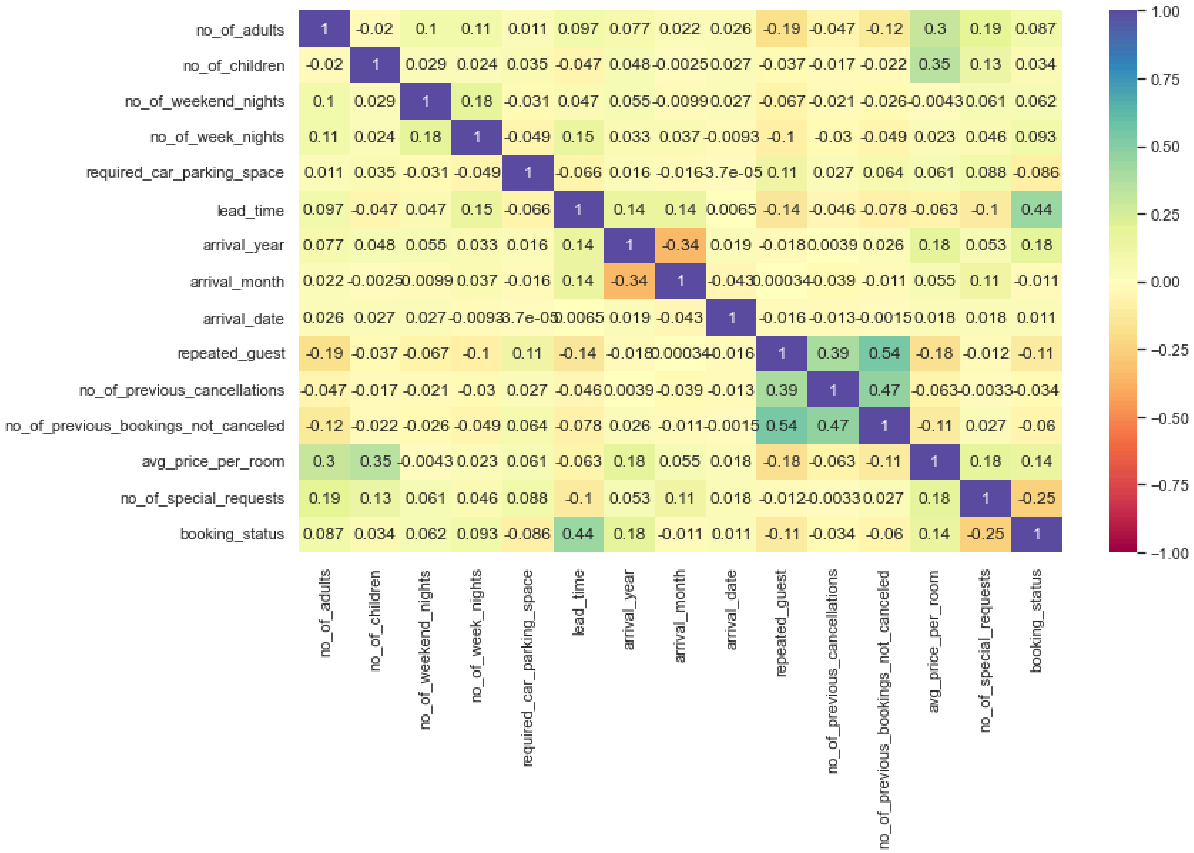
Let's encode Canceled bookings to 1 and Not_Canceled as 0 for further analysis

In [26]:
```
data["booking_status"] = data["booking_status"].apply(
    lambda x: 1 if x == "Canceled" else 0
)
```

Question 3: Bivariate Analysis

Question 3.1: Find and visualize the correlation matrix using a heatmap and write your observations from the plot. (2 Marks)

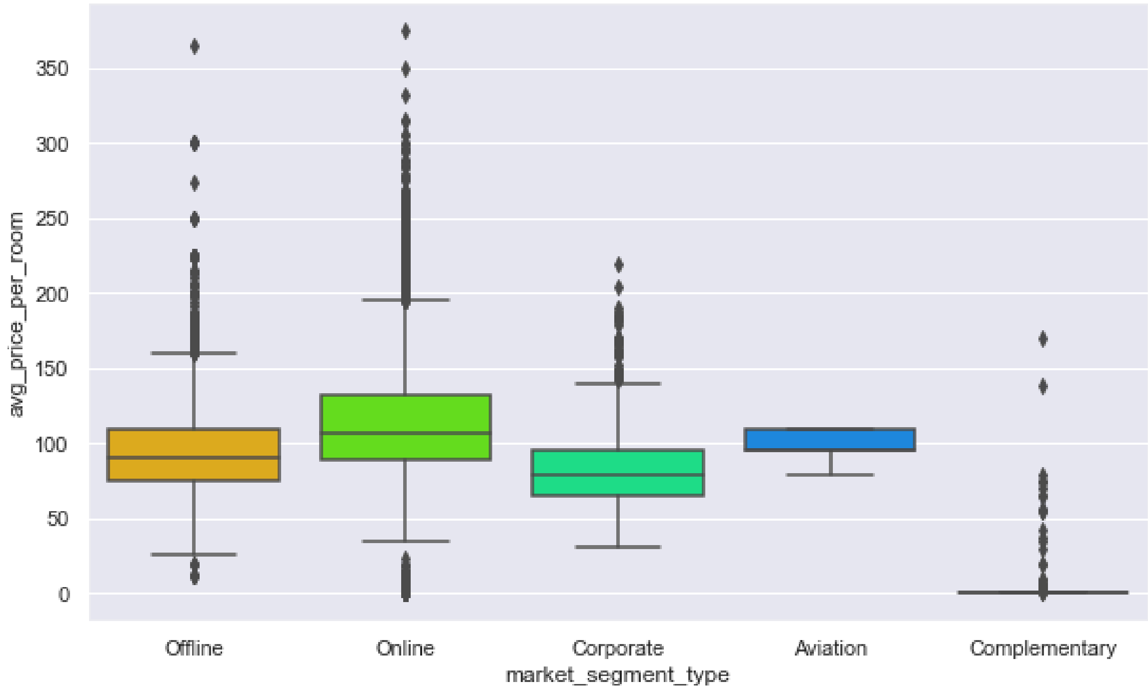
In [27]:
```
# Remove _________ and complete the code
cols_list = data.select_dtypes(include=np.number).columns.tolist()

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



- The repeated_guest is strongly positively correlated (R=0.44) with no_of_previous_bookings_not_canceled, wich makes sense, the frequent guests usually don´t cancel the reservations.
- The booking status is positively correlated with the lead_time, wich is interesting. The more in advance the room is booked, more the chances of cancelation.
- Guests with more children pay more for the rooms (R=0.35); the families stay in more expensive rooms.
- There is a negative correlation (R=-0.25) between cancelations and no_of_special_requests.

Hotel rates are dynamic and change according to demand and customer demographics. Let's see how prices vary across different market segments

In [28]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(
    data=data, x="market_segment_type", y="avg_price_per_room", palette="gist_rainbow"
)
plt.show()
```



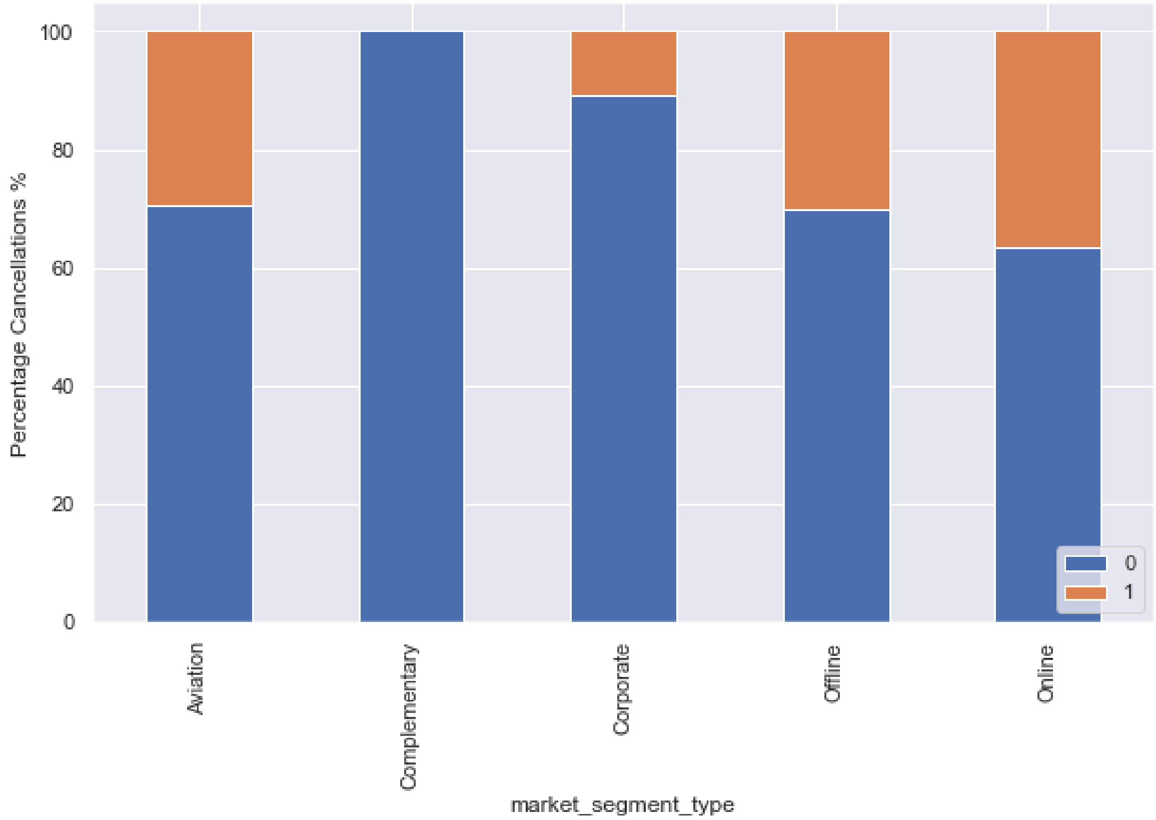
- Rooms booked online have high variations in prices.
- The offline and corporate room prices are almost similar.
- Complementary market segment gets the rooms at very low prices, which makes sense.

We will define a **stacked barplot()** function to help analyse how the target variable varies across predictor categories.

In [29]:
```
# Defining the stacked_barplot() function
def stacked_barplot(data,predictor,target,figsize=(10,6)):
  (pd.crosstab(data[predictor],data[target],normalize='index')*100).plot(kind='bar',figs
  plt.legend(loc="lower right")
  plt.ylabel('Percentage Cancellations %')
```

Question 3.2: Plot the stacked barplot for the variable `Market Segment Type` against the target variable `Booking Status` using the stacked_barplot function provided and write your insights. (1 Mark)

In [30]:
```
# Remove _________ and complete the code
stacked_barplot(data,'market_segment_type','booking_status')
```

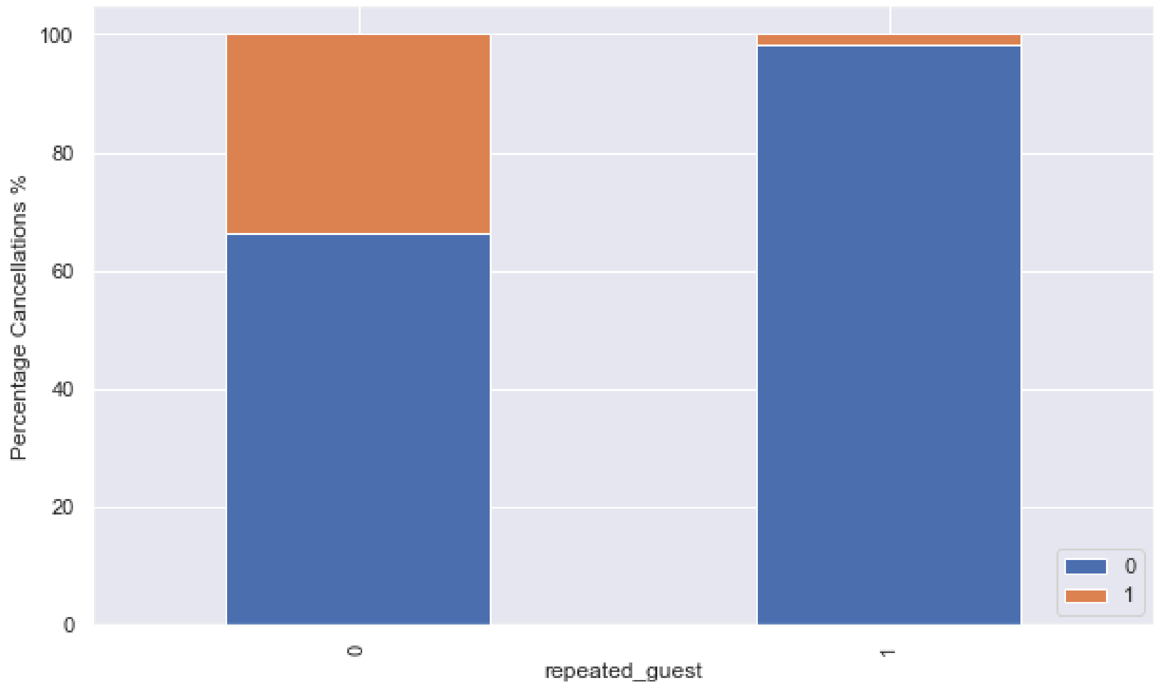


- The Online segment is the one with the most cancelations (almost 40%), and the corporate the one with least booking cancelations (around 10%).
- The focus should be in understanding why the aviation and the online segments cancel so frequently, and develop new offers.

Question 3.3: Plot the stacked barplot for the variable `Repeated Guest` against the target variable `Booking Status` using the stacked_barplot function provided and write your insights. (1 Mark)

Repeating guests are the guests who stay in the hotel often and are important to brand equity.

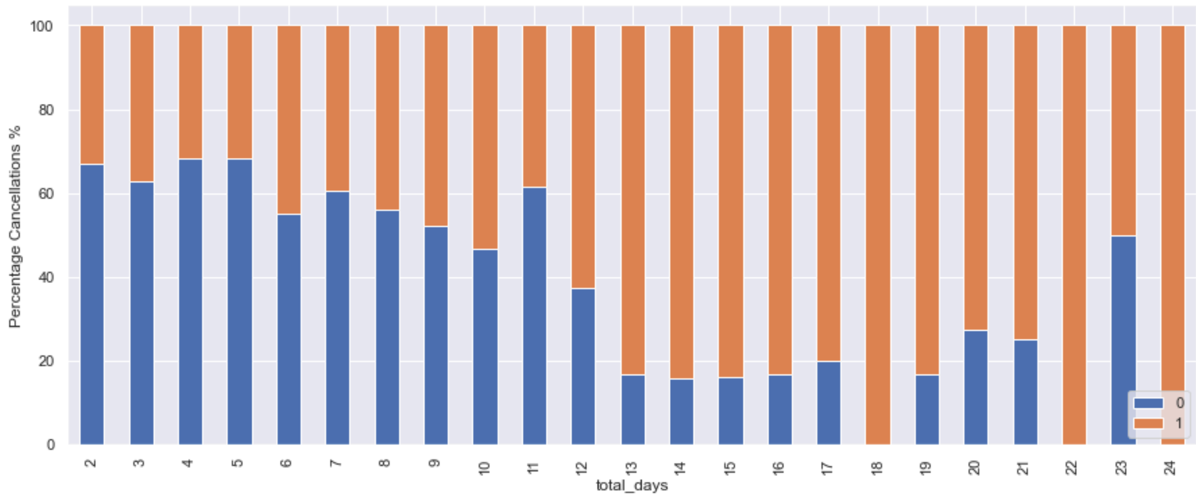
In [31]:
```
# Remove _________ and complete the code
stacked_barplot(data,'repeated_guest','booking_status')
```



- Practically none of the repeated guests of the hotel cancel the reservations.
- More than 30% of the new guests cancel the rooms. This is clearly where we must put our efforts, to understand why and try to solve it.

Let's analyze the customer who stayed for at least a day at the hotel.

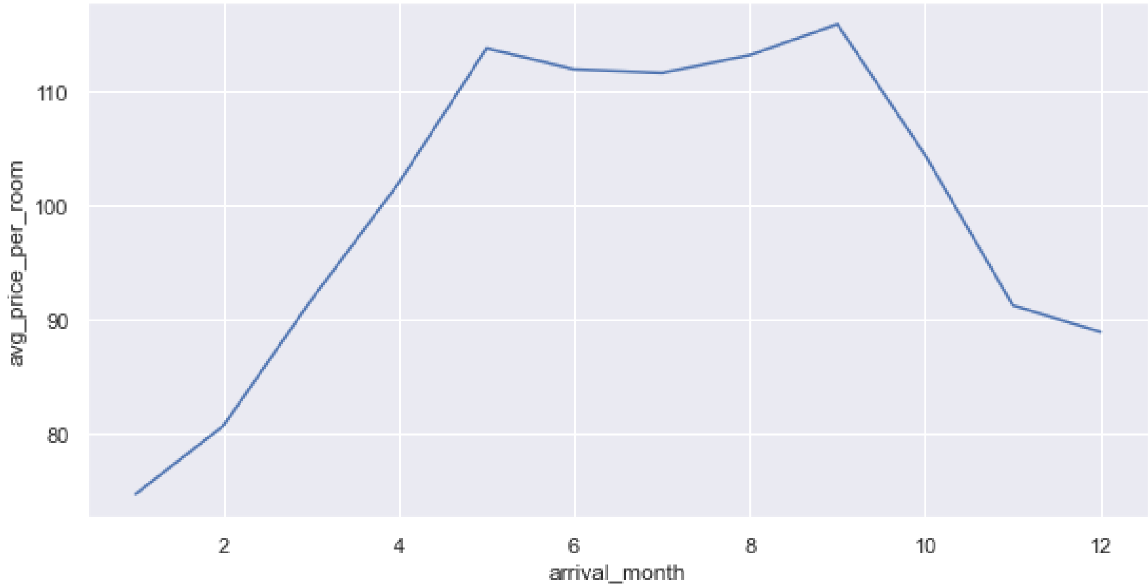
In [32]:
```
stay_data = data[(data["no_of_week_nights"] > 0) & (data["no_of_weekend_nights"] > 0)]
stay_data["total_days"] = (stay_data["no_of_week_nights"] + stay_data["no_of_weekend_nig

stacked_barplot(stay_data, "total_days", "booking_status",figsize=(15,6))
```



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

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

In [33]:
```
plt.figure(figsize=(10, 5))
sns.lineplot(y=data["avg_price_per_room"], x=data["arrival_month"], ci=None)
plt.show()
```



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

Data Preparation for Modeling

- We want to predict which bookings will be canceled.
- Before we proceed to build a model, we'll have to encode categorical features.
- We'll split the data into train and test to be able to evaluate the model that we build on the train data.

Separating the independent variables (X) and the dependent variable (Y)

In [34]:
```
X = data.drop(["booking_status"], axis=1)
Y = data["booking_status"]

X = pd.get_dummies(X, drop_first=True) # Encoding the Categorical features
```

Splitting the data into a 70% train and 30% test set

Some classification problems can exhibit a large imbalance in the distribution of the target classes: for instance there could be several times more negative samples than positive samples. In such cases it is recommended to use the **stratified sampling** technique to ensure that relative class frequencies are approximately preserved in each train and validation fold.

In [35]:
```
# Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30,stratify=Y, rar
```

In [36]:
```
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, 27)
Shape of test set :  (10883, 27)
Percentage of classes in training set:
0    0.672377
1    0.327623
Name: booking_status, dtype: float64
Percentage of classes in test set:
0    0.672333
1    0.327667
Name: booking_status, dtype: float64
```

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 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 the hotel might not be able to provide satisfactory services to the customer by assuming that this booking will be canceled. This might damage brand equity.

How to reduce the losses?

- The hotel would want the `F1 Score` to be maximized, the greater the F1 score, the higher the chances of minimizing False Negatives and False Positives.

Also, let's create a function to calculate and print the classification report and confusion matrix so that we don't have to rewrite the same code repeatedly for each model.

In [37]:
```
# Creating metric function
def metrics_score(actual, predicted):
    print(classification_report(actual, predicted))

    cm = confusion_matrix(actual, predicted)
    plt.figure(figsize=(8,5))

    sns.heatmap(cm, annot=True,  fmt='.2f', xticklabels=['Not Cancelled', 'Cancelled'],
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
```

Building the model

We will be building 4 different models:

- **Logistic Regression**
- **Support Vector Machine (SVM)**
- **Decision Tree**

- **Random Forest**

Question 4: Logistic Regression (6 Marks)

Question 4.1: Build a Logistic Regression model (Use the sklearn library) (1 Mark)

In [38]:
```
# Remove _________ and complete the code

# Fitting logistic regression model
lg = LogisticRegression()
lg.fit(X_train,y_train)
```

Out[38]:
```
LogisticRegression()
```

Question 4.2: Check the performance of the model on train and test data (2 Marks)

In [39]:
```
# Remove _________ and complete the code

# Checking the performance on the training data
y_pred_train = lg.predict(X_train)
metrics_score(y_train,y_pred_train)
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86     17073
           1       0.74      0.59      0.66      8319

    accuracy                           0.80     25392
   macro avg       0.78      0.74      0.76     25392
weighted avg       0.79      0.80      0.79     25392
```

- We obtained a model with a recall of 59% and a precision of 74%, the f1-score being 66%.

Let's check the performance on the test set

In [40]:
```
# Remove _________ and complete the code

# Checking the performance on the test dataset
```

```
y_pred_test = lg.predict(X_test)
metrics_score(y_test,y_pred_test )
```

```
              precision    recall  f1-score   support

           0       0.81      0.89      0.85      7317
           1       0.73      0.58      0.65      3566

    accuracy                           0.79     10883
   macro avg       0.77      0.74      0.75     10883
weighted avg       0.79      0.79      0.79     10883
```

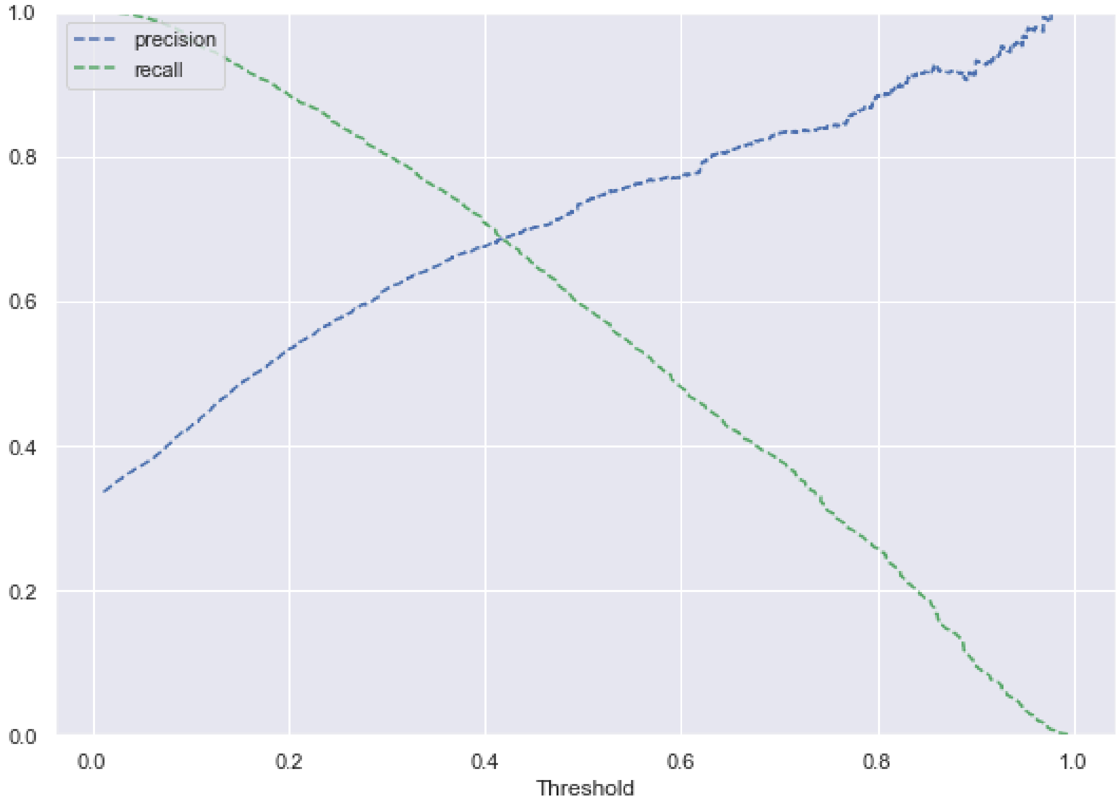
- The model is not overfitting, as the accuracy for the test set is 79%, and 80% for the trainig set.
- The f1-scores are almost identical. Maybe if we change the threshold value, we can improve the balance precision/recall and increase the f1-score.

Question 4.3: Find the optimal threshold for the model using the Precision-Recall Curve. (1 Mark)

Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.

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

In [41]:
```
# Remove _________ and complete the code

# Predict_proba gives the probability of each observation belonging to each class
y_scores_lg=lg.predict_proba(X_train)

precisions_lg, recalls_lg, thresholds_lg = precision_recall_curve(y_train,y_scores_lg[:,

# Plot values of precisions, recalls, and thresholds
plt.figure(figsize=(10,7))
plt.plot(thresholds_lg, precisions_lg[:-1], 'b--', label='precision')
plt.plot(thresholds_lg, recalls_lg[:-1], 'g--', label = 'recall')
plt.xlabel('Threshold')
plt.legend(loc='upper left')
plt.ylim([0,1])
plt.show()
```



- As we want f1-score to be maximized, both precision and recall should as high as possible.
- The point where the two lines cross is the optimum, around 0.41.

In [42]:
```
# Setting the optimal threshold
optimal_threshold = 0.41
```

Question 4.4: Check the performance of the model on train and test data using the optimal threshold. (2 Marks)

In [43]:
```
# Remove _________ and complete the code

# Creating confusion matrix
y_pred_train = lg.predict_proba(X_train)
metrics_score(y_train,y_pred_train[:,1]>optimal_threshold)
```

```
              precision    recall  f1-score   support

           0       0.85      0.84      0.85     17073
           1       0.68      0.70      0.69      8319

    accuracy                           0.79     25392
   macro avg       0.77      0.77      0.77     25392
weighted avg       0.80      0.79      0.80     25392
```

- With the optimal threshold, the f1-score has slightly improved, form 66% to 69%.

Let's check the performance on the test set

In [44]:
```
# Remove _________ and complete the code

y_pred_test = lg.predict_proba(X_test)
metrics_score(y_test,y_pred_test[:,1]>optimal_threshold)
```

```
              precision    recall  f1-score   support

           0       0.84      0.84      0.84      7317
           1       0.67      0.69      0.68      3566

    accuracy                           0.79     10883
   macro avg       0.76      0.76      0.76     10883
weighted avg       0.79      0.79      0.79     10883
```

- Although the overall accuracy stayed the same oround 79%, the f1-score incresaed from 65% to 68%.
- It´s a resonably good model, but we might be ablo to do better with other models.

Question 5: Support Vector Machines (11 Marks)

To accelerate SVM training, let's scale the data for support vector machines.

In [45]:
```
scaling = MinMaxScaler(feature_range=(-1,1)).fit(X_train)
X_train_scaled = scaling.transform(X_train)
X_test_scaled = scaling.transform(X_test)
```

Let's build the models using the two of the widely used kernel functions:

1. **Linear Kernel**
2. **RBF Kernel**

Question 5.1: Build a Support Vector Machine model using a linear kernel (1 Mark)

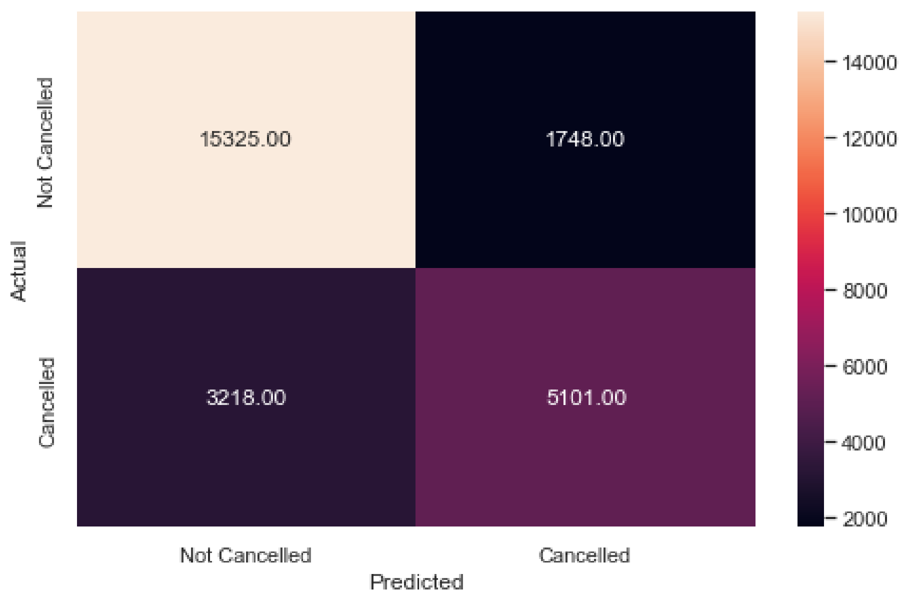
Note: Please use the scaled data for modeling Support Vector Machine

In [46]:
```
# Remove _________ and complete the code

svm = SVC(kernel='linear',probability=True) # Linear kernal or linear decision boundary
model = svm.fit(X_train_scaled,y_train)
```

Question 5.2: Check the performance of the model on train and test data (2 Marks)

In [47]:
```
# Remove _________ and complete the code

y_pred_train_svm = model.predict(X_train_scaled)
metrics_score(y_train,y_pred_train_svm)
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86     17073
           1       0.74      0.61      0.67      8319

    accuracy                           0.80     25392
   macro avg       0.79      0.76      0.77     25392
weighted avg       0.80      0.80      0.80     25392
```



- The f1-score has dropped 2%, to 67%, and the accuracy remains at 80%.

Checking model performance on test set

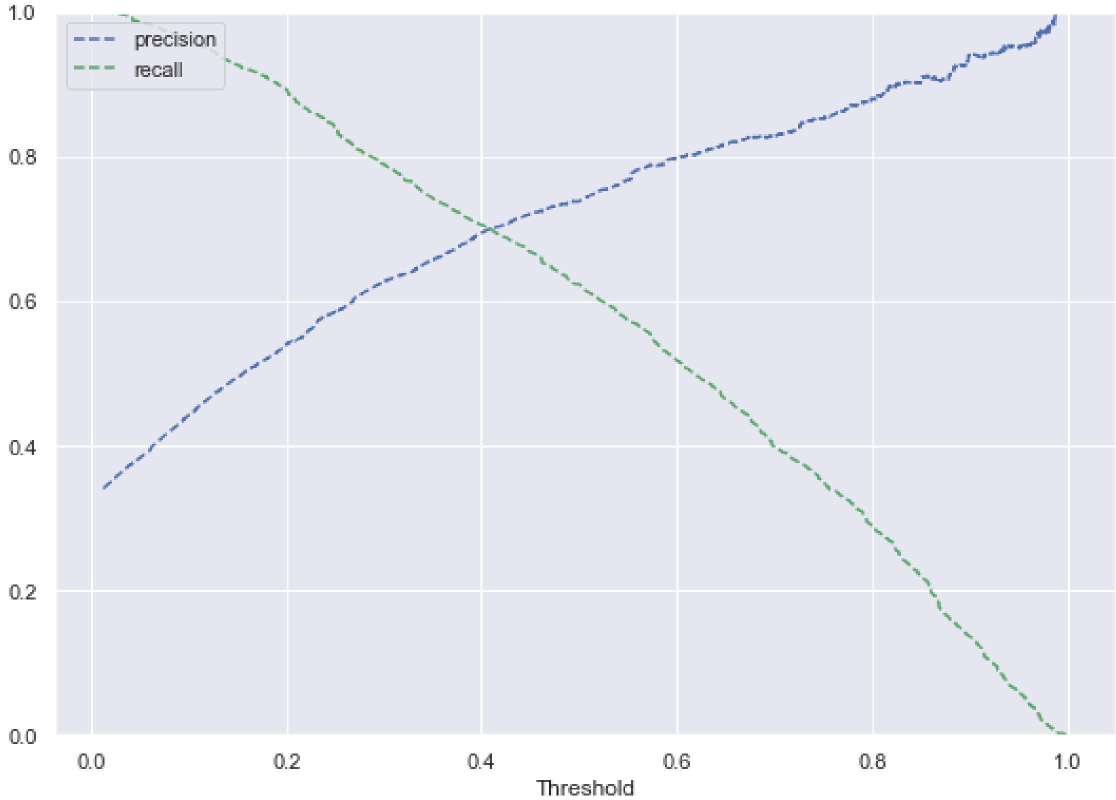
In [48]:
```
# Remove _________ and complete the code

y_pred_test_svm = model.predict(X_test_scaled)
metrics_score(y_test,y_pred_test_svm)
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      7317
           1       0.74      0.61      0.67      3566

    accuracy                           0.80     10883
   macro avg       0.78      0.75      0.76     10883
weighted avg       0.80      0.80      0.80     10883
```

- The performance of the svm model with linear kernel is similar for the train and test datasets, so there is no overfitting of the model.

Question 5.3: Find the optimal threshold for the model using the Precision-Recall Curve. (1 Mark)

In [49]:
```
# Remove _________ and complete the code

# Predict on train data
y_scores_svm=model.predict_proba(X_train_scaled)

precisions_svm, recalls_svm, thresholds_svm = precision_recall_curve(y_train,y_scores_sv

# Plot values of precisions, recalls, and thresholds
plt.figure(figsize=(10,7))
plt.plot(thresholds_svm, precisions_svm[:-1], 'b--', label='precision')
plt.plot(thresholds_svm, recalls_svm[:-1], 'g--', label = 'recall')
plt.xlabel('Threshold')
plt.legend(loc='upper left')
plt.ylim([0,1])
plt.show()
```



- The best compromise between precision and recall is achieved at 0.41, wich will potentiate the maximum f1-score possible.

In [50]:
```
optimal_threshold_svm=0.41
```

Question 5.4: Check the performance of the model on train and test data using the optimal threshold. (2 Marks)

In [51]:
```
# Remove _________ and complete the code

y_pred_train_svm = model.predict_proba(X_train_scaled)
metrics_score(y_train,y_pred_train[:,1]>optimal_threshold_svm)
```

```
              precision    recall  f1-score   support

           0       0.85      0.84      0.85     17073
           1       0.68      0.70      0.69      8319

    accuracy                           0.79     25392
   macro avg       0.77      0.77      0.77     25392
weighted avg       0.80      0.79      0.80     25392
```

- The f1-score with the adjusted threshold value has increased from 67% to 69%

In [52]:
```
# Remove _________ and complete the code

y_pred_test = model.predict_proba(X_test_scaled)
metrics_score(y_test,y_pred_test[:,1]>optimal_threshold_svm)
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85      7317
           1       0.69      0.70      0.69      3566

    accuracy                           0.80     10883
   macro avg       0.77      0.77      0.77     10883
weighted avg       0.80      0.80      0.80     10883
```

- The f1-score with the adjusted threshold value has also increased from 67% to 69%, with no overfitting issues.

Question 5.5: Build a Support Vector Machines model using an RBF kernel (1 Mark)

In [53]:
```
# Remove _________ and complete the code

svm_rbf=SVC(kernel='rbf',probability=True)
svm_rbf.fit(X_train_scaled,y_train)
```

Out[53]:
```
SVC(probability=True)
```

Question 5.6: Check the performance of the model on train and test data (2 Marks)

In [54]:
```
# Remove _________ and complete the code

y_pred_train_svm = svm_rbf.predict(X_train_scaled)
metrics_score(y_train,y_pred_train_svm)
```

```
              precision    recall  f1-score   support

           0       0.84      0.91      0.88     17073
           1       0.79      0.65      0.71      8319

    accuracy                           0.83     25392
   macro avg       0.81      0.78      0.80     25392
weighted avg       0.82      0.83      0.82     25392
```

- The f1-score of the rfb kernel is higher (71%) than the one with the linear kernel (67%).
- Model performance is a little bit better.

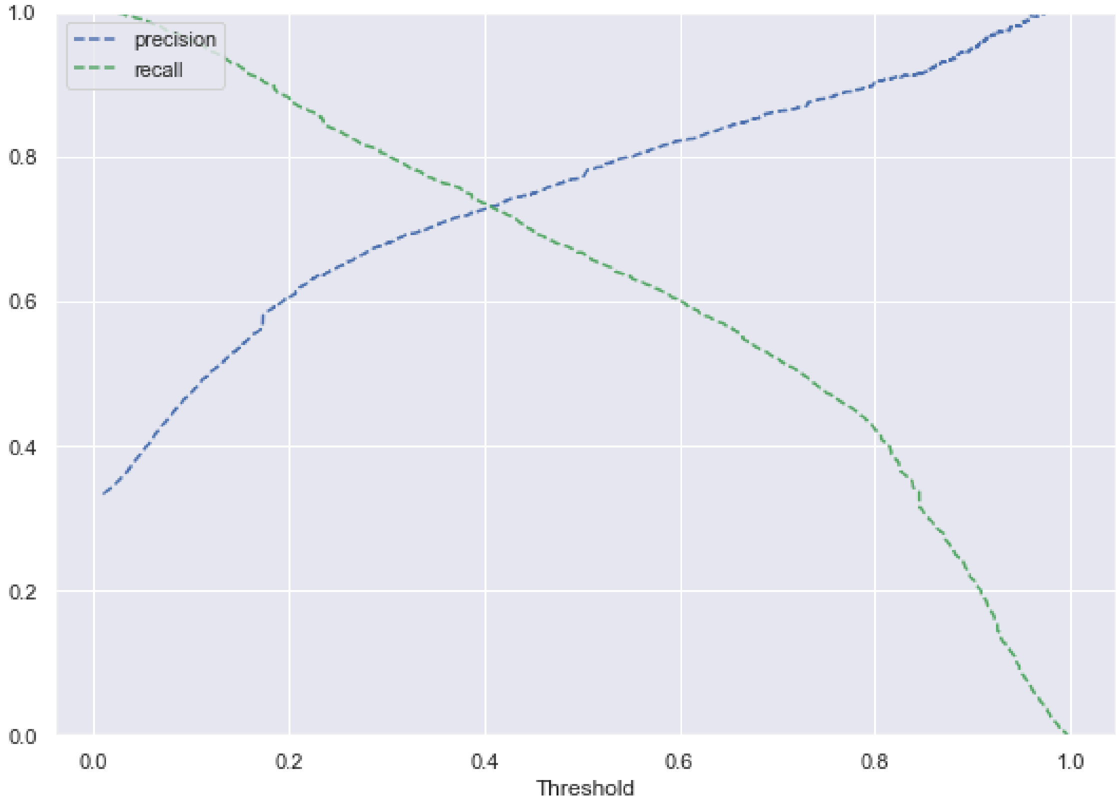
Checking model performance on test set

In [55]:
```
# Remove _________ and complete the code

y_pred_test = svm_rbf.predict(X_test_scaled)

metrics_score(y_test,y_pred_test)
```

```
              precision    recall  f1-score   support

           0       0.84      0.91      0.87      7317
           1       0.78      0.63      0.70      3566

    accuracy                           0.82     10883
   macro avg       0.81      0.77      0.78     10883
weighted avg       0.82      0.82      0.81     10883
```

- There is no overfitting of this model, and the f1-score at 70% is better than the linear kernel.

In [56]:
```
# Predict on train data
y_scores_svm=svm_rbf.predict_proba(X_train_scaled)

precisions_svm, recalls_svm, thresholds_svm = precision_recall_curve(y_train, y_scores_s

# Plot values of precisions, recalls, and thresholds
plt.figure(figsize=(10,7))
plt.plot(thresholds_svm, precisions_svm[:-1], 'b--', label='precision')
plt.plot(thresholds_svm, recalls_svm[:-1], 'g--', label = 'recall')
plt.xlabel('Threshold')
plt.legend(loc='upper left')
plt.ylim([0,1])
plt.show()
```



In [57]:
```
optimal_threshold_svm=0.41
```

Question 5.7: Check the performance of the model on train and test data using the optimal threshold. (2 Marks)

In [58]:
```
# Remove _________ and complete the code

y_pred_train_svm = model.predict_proba(X_train_scaled)
metrics_score(y_train,y_pred_train_svm[:,1]>optimal_threshold_svm)
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85     17073
           1       0.70      0.70      0.70      8319

    accuracy                           0.80     25392
   macro avg       0.78      0.78      0.78     25392
weighted avg       0.80      0.80      0.80     25392
```

- The performance hasn´t improved with the new threshold value (0.71 to 0.70)

In [59]:
```
# Remove _________ and complete the code

y_pred_test = svm_rbf.predict_proba(X_test_scaled)
metrics_score(y_test,y_pred_test[:,1]>optimal_threshold_svm)
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86      7317
           1       0.72      0.72      0.72      3566

    accuracy                           0.82     10883
   macro avg       0.79      0.79      0.79     10883
weighted avg       0.82      0.82      0.82     10883
```

-The f1-score increased slightly from 70% to 72%, without overfitting, ans a good accuracy of 82%.
-This one is the best of the linear models, although they all present similar results; let´s compare with non-linear models like Decision Trees and Random Forest.

Question 6: Decision Trees (7 Marks)

Question 6.1: Build a Decision Tree Model (1 Mark)

In [60]:
```
# Remove _________ and complete the code

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

Out[60]:
```
DecisionTreeClassifier(random_state=1)
```

Question 6.2: Check the performance of the model on train and test data (2 Marks)

In [61]:
```
# Remove _________ and complete the code

# Checking performance on the training dataset
pred_train_dt = model_dt.predict(X_train)
metrics_score(y_train,pred_train_dt)
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     17073
           1       1.00      0.99      0.99      8319

    accuracy                           0.99     25392
   macro avg       1.00      0.99      0.99     25392
weighted avg       0.99      0.99      0.99     25392
```

- As expected, the decision tree classifier adapts itself very well to the training data, with almost perfect scores.
- It's probably overfitting, we will investigate next.

Checking model performance on test set

In [62]:
```python
pred_test_dt = model_dt.predict(X_test)
metrics_score(y_test,pred_test_dt)
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      7317
           1       0.79      0.79      0.79      3566

    accuracy                           0.87     10883
   macro avg       0.85      0.85      0.85     10883
weighted avg       0.87      0.87      0.87     10883
```

- The decision tree is overfitting (0,99 to 0,87 accuracy), but the f1-score is the best so far (0.79).
- Let´s tune the hyperparameters to try reduce overfitting.

Question 6.3: Perform hyperparameter tuning for the decision tree model using GridSearch CV (1 Mark)

Note: Please use the following hyperparameters provided for tuning the Decision Tree. In general, you can experiment with various hyperparameters to tune the decision tree, but for this project, we recommend sticking to the parameters provided.

In [65]:
```
# Remove _________ and complete the code

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

# 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],
}


# Run the grid search
grid_obj = GridSearchCV(estimator,parameters,cv=5,scoring='f1',n_jobs=1)
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)
```

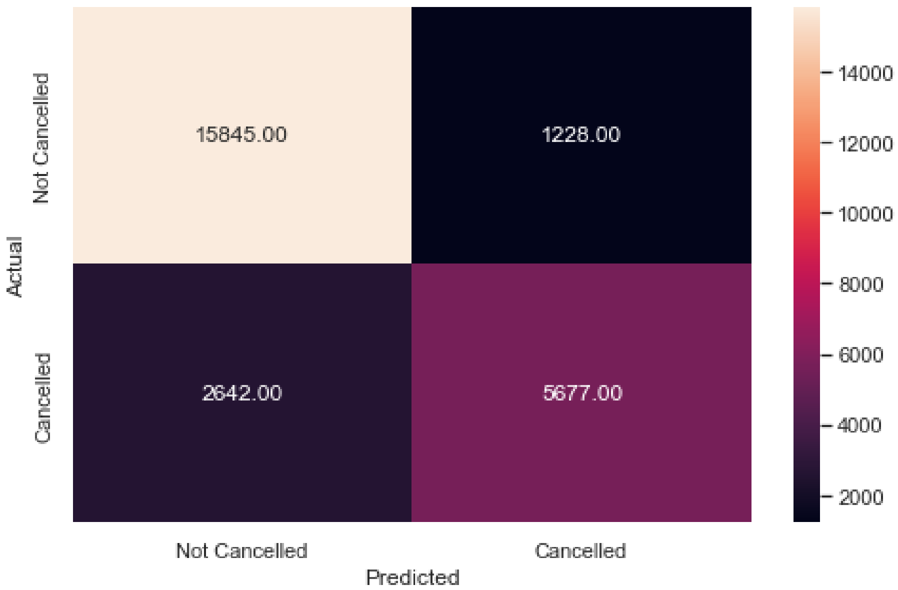
Out[65]:
```
DecisionTreeClassifier(max_depth=6, max_leaf_nodes=50, min_samples_split=10,
                       random_state=1)
```

Question 6.4: Check the performance of the model on the train and test data using the tuned model (2 Mark)

Checking performance on the training set

In [66]:
```
# Remove _________ and complete the code

# Checking performance on the training dataset
dt_tuned = estimator.predict(X_train)
metrics_score(y_train,dt_tuned)
```

```
              precision    recall  f1-score   support

           0       0.86      0.93      0.89     17073
           1       0.82      0.68      0.75      8319

    accuracy                           0.85     25392
   macro avg       0.84      0.81      0.82     25392
weighted avg       0.85      0.85      0.84     25392
```



- The f1-score has decreased with the tuned model.

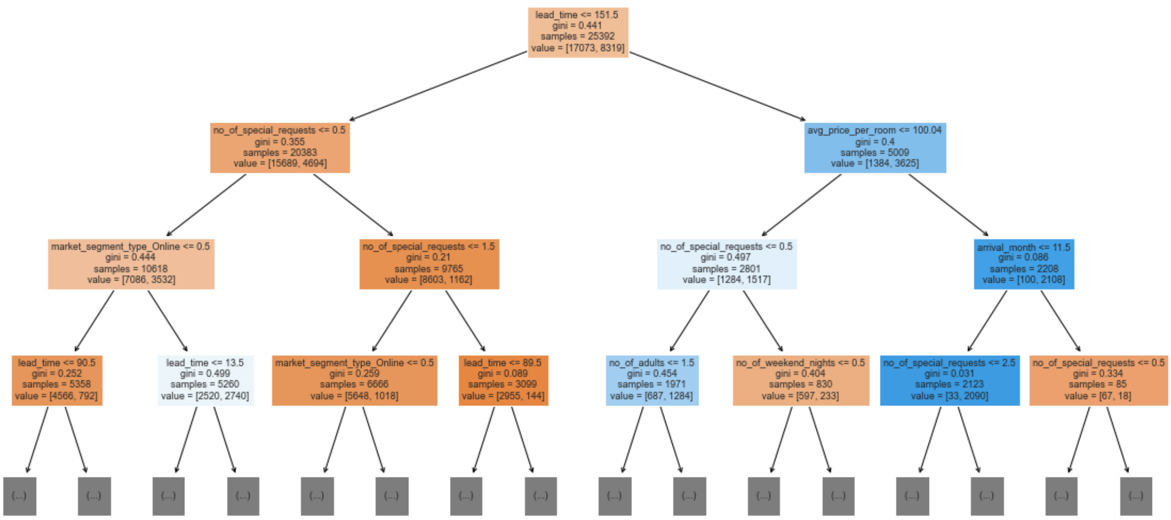
In [68]:
```
# Remove _________ and complete the code

# Checking performance on the training dataset
y_pred_tuned = estimator.predict(X_test)
metrics_score(y_test,y_pred_tuned)
```

```
              precision    recall  f1-score   support

           0       0.85      0.93      0.89      7317
           1       0.82      0.67      0.74      3566

    accuracy                           0.84     10883
   macro avg       0.84      0.80      0.81     10883
weighted avg       0.84      0.84      0.84     10883
```

- In this model there is no overfitting, but the f1-score (0.74) is below the one without tunning (0.79).

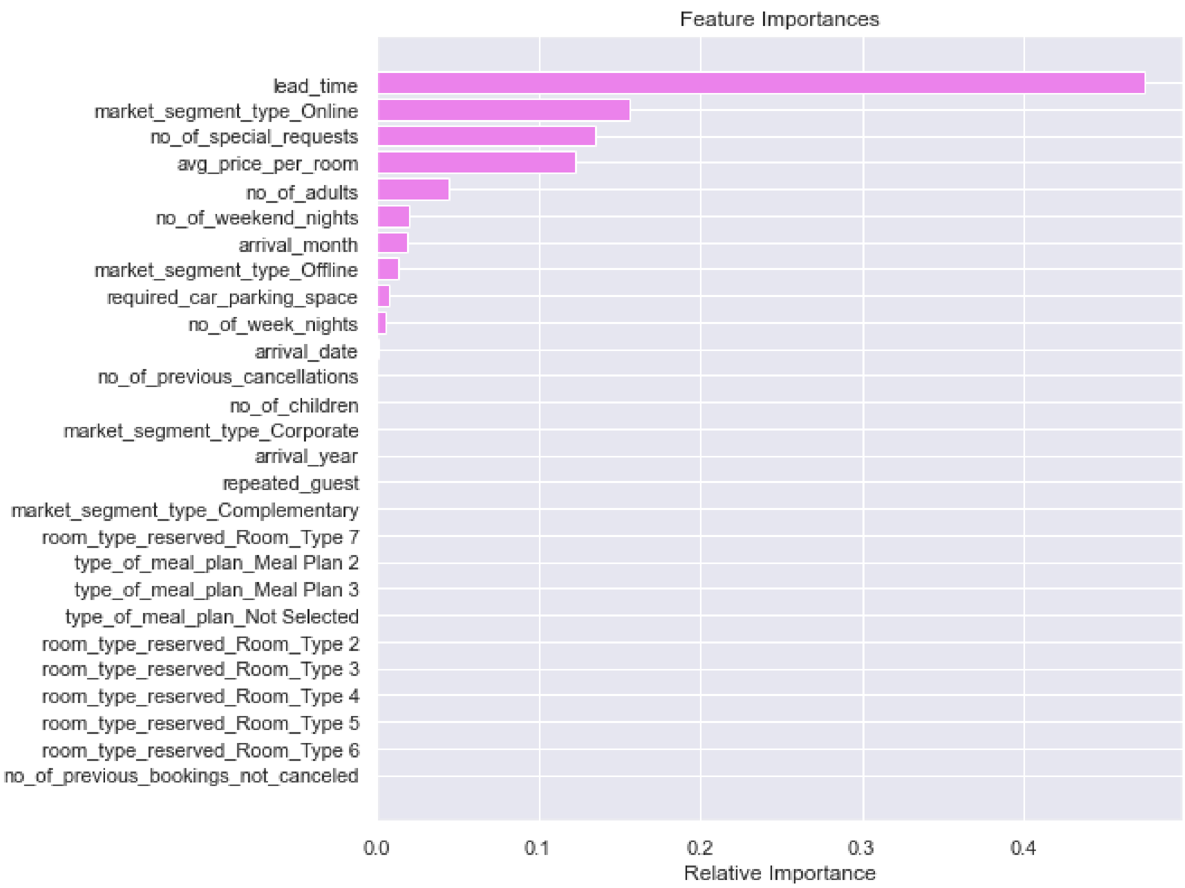
Visualizing the Decision Tree

In [69]:
```
feature_names = list(X_train.columns)
plt.figure(figsize=(20, 10))
out = tree.plot_tree(
    estimator,max_depth=3,
    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()
```



Question 6.5: What are some important features based on the tuned decision tree? (1 Mark)

In [70]:
```
# Remove _________ and complete the code

# 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()
```



- The tree has been reduced to 10 classifiers, and the most important features are: 1) lead time 2) market_segment_type_Online 3) no_of_special_requests

---

Question 7: Random Forest (4 Marks)

Question 7.1: Build a Random Forest Model (1 Mark)

In [71]:
```
# Remove _________ and complete the code

rf_estimator = RandomForestClassifier(random_state=1)

rf_estimator.fit(X_train,y_train)
```

Out[71]:
```
RandomForestClassifier(random_state=1)
```

Question 7.2: Check the performance of the model on the train and test data (2 Marks)

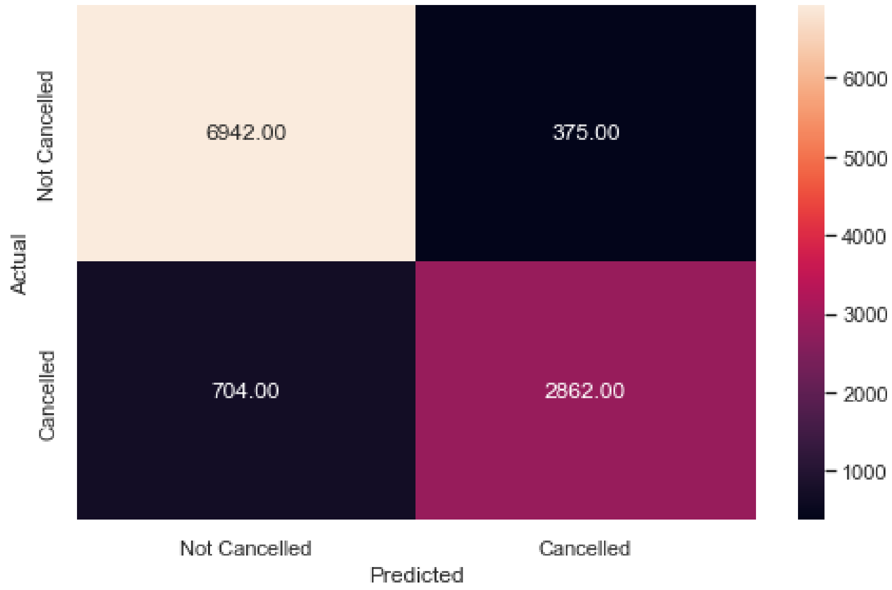
In [72]:
```python
# Remove _________ and complete the code

y_pred_train_rf = rf_estimator.predict(X_train)

metrics_score(y_train,y_pred_train_rf)
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     17073
           1       1.00      0.99      0.99      8319

    accuracy                           0.99     25392
   macro avg       0.99      0.99      0.99     25392
weighted avg       0.99      0.99      0.99     25392
```

- Almost all points well classified, very ggod model on the trainig data.

In [73]:
```python
# Remove _________ and complete the code

y_pred_test_rf = rf_estimator.predict(X_test)

metrics_score(y_test,y_pred_test_rf)
```

```
              precision    recall  f1-score   support

           0       0.91      0.95      0.93      7317
           1       0.88      0.80      0.84      3566

    accuracy                           0.90     10883
   macro avg       0.90      0.88      0.88     10883
weighted avg       0.90      0.90      0.90     10883
```



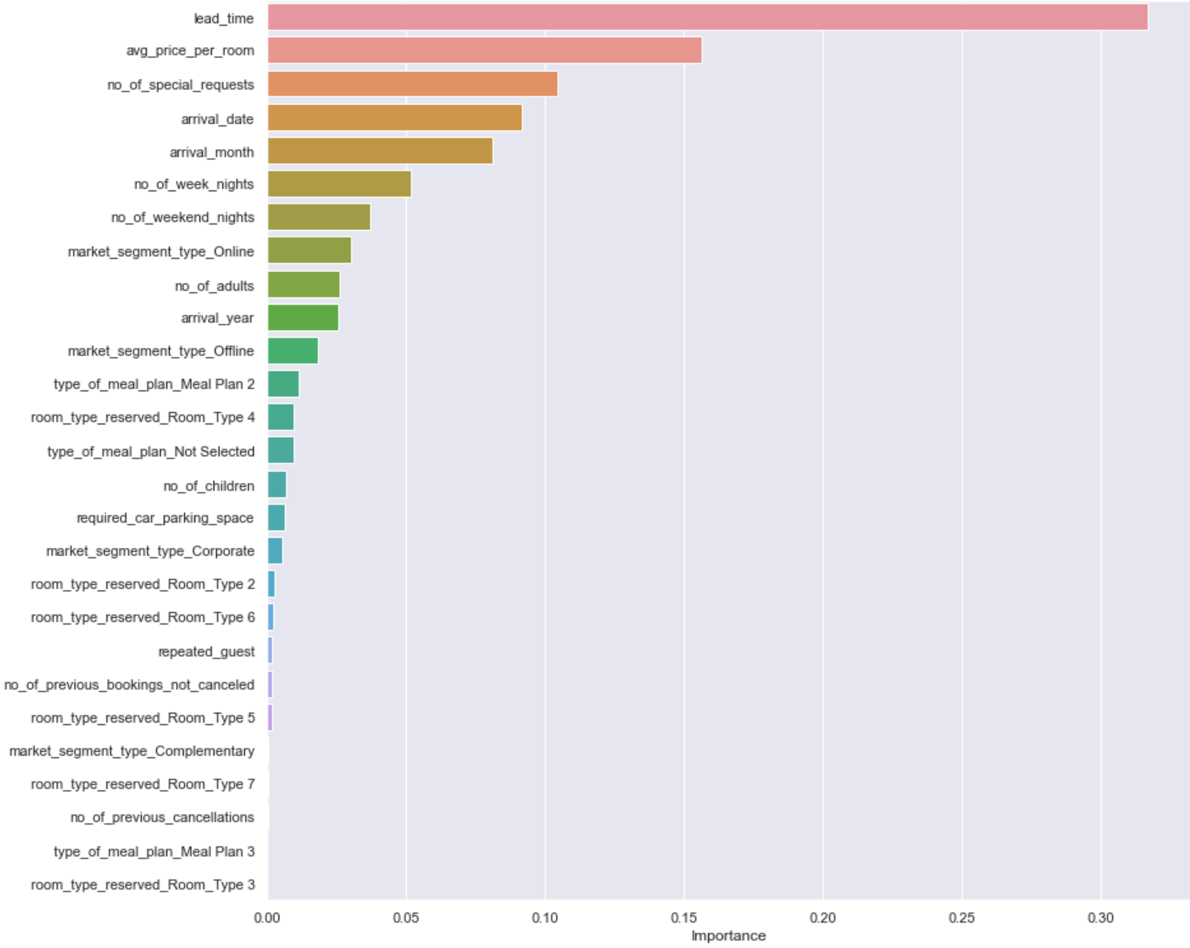
- This is by far the best classifying model so far, with f1-score=84%, and accuracy of 90%.
- However, it's still overfitting on the training data.
- We can try to ajust the hyperparameters to reduce this condition.

Question 7.3: What are some important features based on the Random Forest? (1 Mark)

Let's check the feature importance of the Random Forest

In [75]:
```
# Remove _________ and complete the code

importances = rf_estimator.feature_importances_

columns = X.columns

importance_df = pd.DataFrame(importances,index=columns,columns=['Importance']).sort_valu

plt.figure(figsize = (13, 13))

sns.barplot(importance_df.Importance, importance_df.index)
```

Out[75]:
```
<AxesSubplot:xlabel='Importance'>
```



- With the Random Forest Classifier, the most important features are not entirely the same as in Decision Tree. 1) lead_time 2) average_price_per_room 3) no_of_special_requests
- So, the lead time and the price per room are the two main featureas that contribute to booking cancelations.

Question 8: Conclude ANY FOUR key takeaways for business recommendations (4 Marks)

33% of the bookings were canceled by the customers.

The Online segment is the one with the most cancelations (almost 40%), and the corporate the one with least booking cancelations (around 10%). The focus should be in understanding why the aviation and the online segments cancel so frequently, and develop new offers.

Practically none of the repeated guests of the hotel cancel the reservations. More than 30% of the new guests cancel the rooms. The hotel could present a more apealing package to the firts-time guests.

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

As the no_of_special_requests impacts the cancelations, the hotel should improve this feature.

The price per room is the second cause of cancelations, so the hotel could try a different distribution of prices troughout the year.

Happy Learning!

```
In [ ]:
```