HOTEL RESERVATIONS

PROJECT PHASE – 1

Done By,

Sai Teja Sankarneni (saitejas)

Nishant varma Indukuri (nindukur)

PROBLEM STATEMENT

(A)

The Online Hotel Reservation system has changed the landscape of hotel booking. Customers no longer need to worry about the availability of rooms and services until they reach the front desk. They can check and plan accordingly beforehand through Online Reservation Systems. However, this has also led to a few more challenges for the hotels beyond meeting customer expectations. It has led to a significant increase in cancellations and no-shows.

Although cancellations are common occurrences and could be due to various reasons like a change of plans, weather, scheduling conflicts, etc., they have caused negative implications for the hotels. These hotels started to overbook or under book in some instances, which ultimately led to an impact on the Hotels' revenue.

So, to deal with these kinds of situations, we decided to come up with a model that can help predict whether the customer will cancel the reservation or not. The contents of the table are given below:

- Booking_ID: Unique booking code for each booking
- No_of_adults: Number of adults
- No_of_children: Number of children
- No_of_weekend_nights: Number of Saturday/Sunday nights the guests stayed
- No_of_week_nights: Number of weekday nights the guests stayed
- Type_of_meal_plan: Meal plan chosen by the guest
- Required_car_parking_space: Whether the guest require car parking or not
- Room_type_reserved: Room type selected by the customer
- Lead_time: Number of days between the date of booking and date of arrival
- Arrival_year: Year of the arrival date
- Arrival_month: Month of the arrival date
- Arrival_date: Date of the month of arrival
- Market_segment_type: Designation of market segment
- Repeated_guest: Is the guest a frequent or returning customer

- No_of_previous_cancellations: Number of previous bookings cancelled before arrival date
- No_of_previous_bookings_not_cancelled: Number of previous bookings attended by the customer.
- Avg_price_per_room: Average price of room per day of reservation
- No_of_special_requests: Number of special requests made by the customer
- Booking_status: Booking indication whether it is cancelled or not

(B)

This project has the potential to revolutionize how hotels manage their booking, increase revenue, and enhance customer satisfaction. With the help of this model, we can use historical data on hotel bookings to optimize prices, predict future demand, and suggest choices and services to customers. Through this, hotels could be well-functioning and aim for profits.

DATA SOURCES

The data file is also attached in the folder as Hotel_reservations_data.csv

And has been taken from Kaggle. It consists of 36275 rows and 19 columns.

https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset?datasetId=2783627&sortBy=voteCount

DATA INITIALIZATION

Firstly, we import all the required libraries and also read the csv file.

In [6]:
```
# Importing all required libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
```

In [8]:
```
# Reading the HotelReservationsDataset and displaying top 10 rows of the dataset
df = pd.read_csv("DIC.csv")
print("Total Number of Rows:",df.shape[0])
print("Total Number of Columns:",df.shape[1])
df
```

```
Total Number of Rows: 36275
Total Number of Columns: 19
```

Out[8]:

	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_reserve
0	INN00001	2	0	1	2	Meal Plan 1	0	Room_Type
1	INN00002	2	0	2	3	Not Selected	0	Room_Type
2	INN00003	1	0	2	1	Meal Plan 1	0	Room_Type
3	INN00004	2	0	0	2	Meal Plan 1	0	Room_Type
4	INN00005	2	0	1	1	Not Selected	0	Room_Type
...	...	...	...	...	...	...	...	.
36270	INN36271	3	0	2	6	Meal Plan 1	0	Room_Type
36271	INN36272	2	0	1	3	Meal Plan 1	0	Room_Type
36272	INN36273	2	0	2	6	NaN	0	Room_Type
36273	INN36274	2	0	0	3	Not Selected	0	Room_Type
36274	INN36275	2	0	1	2	Meal Plan 1	0	Room_Type

DATA CLEANING

To get the best predictions from the models we need to have a clean dataset for which we need to perform Data Cleaning operations on it.

(1) CHECKING FOR NULL VALUES AND REMOVING THEM

```
In [263]: # Data Cleaning and Pre-Processing
          # Checking for Null Values in the Dataset
          df.isnull().sum()
          # We can Clearly see that we have some null values in "required_car_parking_space" and "room_type_reserved" columns.

Out[263]: 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                       169
          required_car_parking_space                0
          room_type_reserved                       91
          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
```

As, we can see there are many NULL values present in the data. We now drop those rows as they might affect the accuracy of the model.

```
In [264]: # Step-1 (DataCleaning)
          # As there are 36275 rows in our dataset removing the rows containing null values will not have much impact on the dataset.
          # Therefore removing the rows containing null values from the dataset.
          df = df.dropna()
          print("Total Number of Rows:",df.shape[0])
          print("Total Number of Columns:",df.shape[1])

          Total Number of Rows: 36050
          Total Number of Columns: 19
```

(2) CHECKING THE DATATYPES OF ALL COLUMNS

```
In [265]: # Checking the datatypes of all the columns and verify whether the columns are having correct datatype.
          print(df.dtypes)

          Booking_ID                               object
          no_of_adults                              int64
          no_of_children                            int64
          no_of_weekend_nights                      int64
          no_of_week_nights                         int64
          type_of_meal_plan                        object
          required_car_parking_space               object
          room_type_reserved                       object
          lead_time                                 int64
          arrival_year                              int64
          arrival_month                             int64
          arrival_date                              int64
          market_segment_type                      object
          repeated_guest                            int64
          no_of_previous_cancellations              int64
          no_of_previous_bookings_not_canceled      int64
          avg_price_per_room                      float64
          no_of_special_requests                    int64
          booking_status                           object
          dtype: object
```

We can derive that required_car_parking_space is an integer type column but has been shown as object. So, it means that the data has a mix of two data types. So now we need to remove those unwanted data values and replace them with 0(Zero).

In [267]:
```python
# Step-2 (DataCleaning)
# Converting the "required_car_parking_space" column to int datatype and replacing the unwanted data with null values.
df['required_car_parking_space'] = pd.to_numeric(df['required_car_parking_space'], errors='coerce').astype('Int64')
print(df['required_car_parking_space'].head(15))
```

```
0        0
1        0
2        0
3        0
4        0
6     <NA>
7        0
8        0
9        0
10       0
11       0
12       0
14       0
15       0
16       0
Name: required_car_parking_space, dtype: Int64
```

In [268]:
```python
# Now the 'required_car_parking_space' Column has only three types of data that are 0, 1 and Null.
print("'required_car_parking_space' column Data Categories:", df['required_car_parking_space'].drop_duplicates().to_list())
print(" ")
print("Occurences of 0's and 1's:")
print(df['required_car_parking_space'].value_counts())
```

```
'required_car_parking_space' column Data Categories: [0, <NA>, 1]

Occurences of 0's and 1's:
0    34793
1     1115
Name: required_car_parking_space, dtype: Int64
```

(3) REPLACING NULL VALUES WITH DEFAULT VALUE

In [10]:
```python
# Step-3 (DataCleaning)
# As we have more significantly 0's than 1's in the 'required_car_parking_space' column Data.
# we are considring 0 as the default value for this column.
# And replacing the Null values with the default value that is 0.
df['required_car_parking_space'] = df['required_car_parking_space'].fillna(0)
print(df['required_car_parking_space'].dtype)
```

```
int64
```

As the default value is 0 and there are NULL as a value in few columns we convert these NULL values into 0. This step is necessary as the unwanted data should be converted into the required data type of the feature.

(4) DROPPING UNWANTED COLUMNS

Now, we can remove the unwanted column which is “Booking_ID” as it is not significant for our model and further steps.

In [22]:
```python
# Step-4 (DataCleaning)
# Dropping the column "Booking_ID" as it is just column that has data which uniquely identifies every row.
# It does not add any significance while training any ML model therfore we are removing this column.
df = df.drop("Booking_ID", axis=1)
df
```

Out[22]:

	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
0	2	0	1	2	Meal Plan 1	0	Room_Type 1	224
1	2	0	2	3	Not Selected	0	Room_Type 1	5
2	1	0	2	1	Meal Plan 1	0	Room_Type 1	1
3	2	0	0	2	Meal Plan 1	0	Room_Type 1	211
4	2	0	1	1	Not Selected	0	Room_Type 1	48
...	...	...	...	...	...	...	...	...
36269	2	2	0	1	Meal Plan 1	0	Room_Type 6	0
36270	3	0	2	6	Meal Plan 1	0	Room_Type 4	85
36271	2	0	1	3	Meal Plan 1	0	Room_Type 1	228
36273	2	0	0	3	Not Selected	0	Room_Type 1	63
36274	2	0	1	2	Meal Plan 1	0	Room_Type 1	207

36050 rows × 18 columns

(5) MERGING SIMILAR COLUMNS

We can merge a few similar columns as they would not have any significance for prediction when apart.

In [24]:
```python
# Step-5 (DataCleaning)
# Merging the "arrival_year","arrival_month","arrival_date" columns into a single column named "arrival_date"
c = ["arrival_year","arrival_month","arrival_date"]
df['arrival_date'] = df[c].apply(lambda x: '-'.join(x.values.astype(str)), axis="columns")
df
```

Out[24]:

	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
0	2	0	1	2	Meal Plan 1	0	Room_Type 1	224
1	2	0	2	3	Not Selected	0	Room_Type 1	5
2	1	0	2	1	Meal Plan 1	0	Room_Type 1	1
3	2	0	0	2	Meal Plan 1	0	Room_Type 1	211
4	2	0	1	1	Not Selected	0	Room_Type 1	48
...	...	...	...	...	...	...	...	...
36269	2	2	0	1	Meal Plan 1	0	Room_Type 6	0
36270	3	0	2	6	Meal Plan 1	0	Room_Type 4	85
36271	2	0	1	3	Meal Plan 1	0	Room_Type 1	228
36273	2	0	0	3	Not Selected	0	Room_Type 1	63
36274	2	0	1	2	Meal Plan 1	0	Room_Type 1	207

36050 rows × 18 columns

(6) DROPPING UNWANTED COLUMNS AND CONVERTING TO APPROPRIATE DATATYPE

```
In [40]: # Step-6 (DataCleaning)
         # Now Dropping the columns "arrival_month", "arrival_year" as we have merged them into "arrival_date" column.
         # Changing the datatype of the "arrivaldate" into datetime datatype.
         df = df.drop(labels=["arrival_month", "arrival_year"], axis=1)
         df['arrival_date'] = pd.to_datetime(df['arrival_date'], errors='coerce')
         df
         print(df.dtypes)

         no_of_adults                                   int64
         no_of_children                                 int64
         no_of_weekend_nights                           int64
         no_of_week_nights                              int64
         type_of_meal_plan                             object
         required_car_parking_space                     Int64
         room_type_reserved                            object
         lead_time                                      int64
         arrival_date                          datetime64[ns]
         market_segment_type                           object
         repeated_guest                                 int64
         no_of_previous_cancellations                   int64
         no_of_previous_bookings_not_canceled           int64
         avg_price_per_room                           float64
         no_of_special_requests                         int64
         booking_status                                object
         dtype: object
```

Here we drop “arrival_month” and “arrival_year” as we have already merged them into “arrival_date” column. Then we convert “arrival_date” into datetime datatype. This step is essential for future purposes as data should be in appropriate format and only contain required information.

Now we check for any NULL values present in the “arrival_date” and remove them from the dataset.

```
In [73]: # Cheching null values for the newly created column "arrival_date" and removing them from the dataset
         df["arrival_date"].isnull().sum()

Out[73]: 36

In [74]: # Dropping the rows with Null values in the "arrival_date" column
         df = df.dropna()
         print("Total Number of Rows:",df.shape[0])
         print("Total Number of Columns:",df.shape[1])

         Total Number of Rows: 36014
         Total Number of Columns: 16
```

(7) DROPPING DUPLICATE ROWS

```
In [75]: # Step-7 (DataCleaning)
         # Dropping duplicate rows
         df = df.drop_duplicates()
         print("Total Number of Rows:",df.shape[0])
         print("Total Number of Columns:",df.shape[1])

         Total Number of Rows: 25827
         Total Number of Columns: 16

In [76]: # Copied the cleaned data before encoding into another df named "Cleaned_Raw_df" to perform EDA.
         Cleaned_Raw_df = df.copy()
```

The rows which are similar have been removed as they don’t have any high significance and Duplicate rows make data model less reliable for use. By removing them we can obtain an effective model.

(8) ENCODING THE DATA

In [277]:
```
# Step-8 (DataCleaning)
# The columns "type_of_meal_plan", "room_type_reserved", "market_segment_type", "booking_status" can be converted into catregoric
print("Before Encoding:")
print("type_of_meal_plan Categories:", df["type_of_meal_plan"].drop_duplicates().to_list())
print("room_type_reserved Categories:", df["room_type_reserved"].drop_duplicates().to_list())
print("market_segment_type Categories:", df["market_segment_type"].drop_duplicates().to_list())
print("booking_status Categories:", df["booking_status"].drop_duplicates().to_list())
```

```
Before Encoding:
type_of_meal_plan Categories: ['Meal Plan 1', 'Not Selected', 'Meal Plan 2', 'Meal Plan 3']
room_type_reserved Categories: ['Room_Type 1', 'Room_Type 4', 'Room_Type 2', 'Room_Type 6', 'Room_Type 5', 'Room_Type 7', 'Room
_Type 3']
market_segment_type Categories: ['Offline', 'Online', 'Corporate', 'Aviation', 'Complementary']
booking_status Categories: ['Not_Canceled', 'Canceled']
```

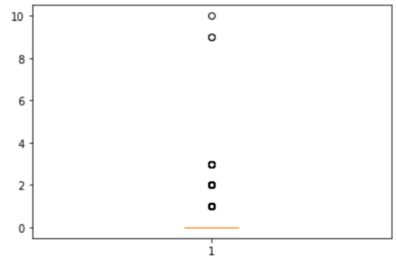
In [78]:
```
# Encoding different categories of these columns to different integer values using label encoder
le = LabelEncoder()
df['type_of_meal_plan']= le.fit_transform(df['type_of_meal_plan'])
df['room_type_reserved']= le.fit_transform(df['room_type_reserved'])
df['market_segment_type']= le.fit_transform(df['market_segment_type'])
df['booking_status']= le.fit_transform(df['booking_status'])

print("After Encoding:")
print("type_of_meal_plan Categories:", df["type_of_meal_plan"].drop_duplicates().to_list())
print("room_type_reserved Categories:", df["room_type_reserved"].drop_duplicates().to_list())
print("market_segment_type Categories:", df["market_segment_type"].drop_duplicates().to_list())
print("booking_status Categories:", df["booking_status"].drop_duplicates().to_list())

# Converting them to Category Datatypes
df['type_of_meal_plan'] = df['type_of_meal_plan'].astype('category')
df['room_type_reserved'] = df['room_type_reserved'].astype('category')
df['market_segment_type'] = df['market_segment_type'].astype('category')
df['booking_status'] = df['booking_status'].astype('category')
df['required_car_parking_space'] = df['required_car_parking_space'].astype('category')
print(df.dtypes)
```

```
After Encoding:
type_of_meal_plan Categories: [0, 3, 1, 2]
room_type_reserved Categories: [0, 3, 1, 5, 4, 6, 2]
market_segment_type Categories: [3, 4, 2, 0, 1]
booking_status Categories: [1, 0]
no_of_adults                                int64
no_of_children                              int64
no_of_weekend_nights                        int64
no_of_week_nights                           int64
type_of_meal_plan                        category
required_car_parking_space               category
room_type_reserved                       category
lead_time                                   int64
arrival_date                       datetime64[ns]
market_segment_type                      category
repeated_guest                              int64
no_of_previous_cancellations                int64
no_of_previous_bookings_not_canceled        int64
avg_price_per_room                        float64
no_of_special_requests                      int64
booking_status                           category
dtype: object
```

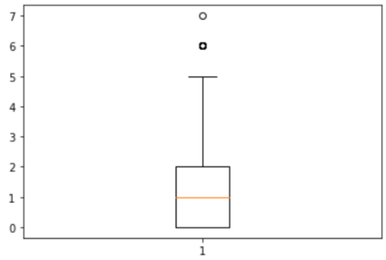
This step helps while fitting data into the model as the data type integer is better suited while fitting into a model than strings. We used label encoder for encoding the columns.

(9) IDENTIFYING OUTLIERS

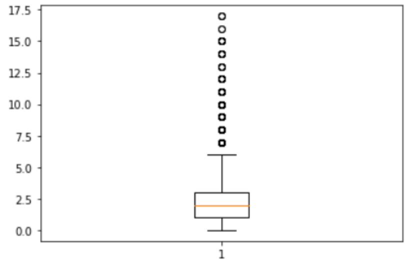
In [79]:
```
# Step-9 (DataCleaning)
# Using Boxplots of some columns tzo identify the outliers.

# Boxplot for 'no_of_children' column
plt.boxplot(df['no_of_children'])
plt.show()
```



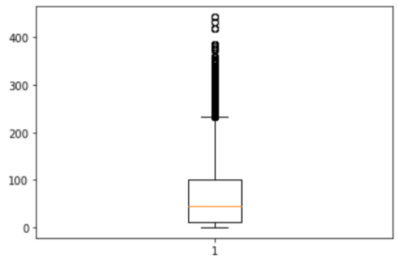
In [80]:
```
# Boxplot for 'no_of_children' column
plt.boxplot(df['no_of_weekend_nights'])
plt.show()
```



In [81]:
```
# Boxplot for 'no_of_week_nights' column
plt.boxplot(df['no_of_week_nights'])
plt.show()
```



In [82]:
```
# Boxplot for 'lead_time' column
plt.boxplot(df['lead_time'])
plt.show()
```



Outliers hinder the accuracy when we use it to train the model. So it is optimal to identify these outliers.

(10) FILTERING OUT ROWS BASED ON VALUES

In [83]:
```
# Step-10 (DataCleaning)
# Filtering out rows based on values and removing outliers using this technique

# From the boxplot we can clearly observe that 10, 8 are outliers and we are considering no. of children above 2 as outliers and
df=df[df['no_of_children']<=2]
# From the boxplot we can clearly observe that values above 6 are ouliers for this column therefore these are removed.
df=df[df['no_of_weekend_nights']<6]
# From the boxplot we can clearly observe that values above 6 are ouliers for this column therefore these are removed.
df=df[df['no_of_week_nights']<6]
# From the boxplot of no_of_week_nights can clearly observe that values above 220 are ouliers for this column therefore these are
df=df[df['lead_time']<220]
```

In this step we use the method of filtering based on values to remove the outliers we obtained. These outliers affect the accuracy prediction.

(11) REINDEXING THE DATA

In [107]:
```
# Step-11 (DataCleaning)
# Re-indexing the rows of the dataset beacuse we have removed several columns with creates missing index values.
df = df.reset_index()
df = df.drop("index", axis=1)
df
```

Out[107]:

	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
0	2	0	2	3	3	0	0	5
1	1	0	2	1	0	0	0	1
2	2	0	0	2	0	0	0	211
3	2	0	1	1	3	0	0	48
4	2	0	1	3	0	0	0	34
...	...	...	...	...	...	...	...	...
24330	2	0	0	2	0	0	3	187
24331	2	0	1	3	0	0	0	15
24332	2	0	2	2	0	0	1	8
24333	2	2	0	1	0	0	5	0
24334	2	0	0	3	3	0	0	63

24335 rows × 16 columns

As there have been many cleaning process performed on the dataset it is important to reindex it to obtain a clean and ordered dataset.

EXPLORATORY DATA ANALYSIS {EDA}

(1) DESCRIBING THE DATA

In [28]:
```
# step-1
# Finding statistics like mean, standard deviation, min, max etc., for all numeric features present in our dataset.
df.describe()
# Our dataset features has different scales for different features.
```

Out[28]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	lead_time	no_of_previous_cancellations	no_of_previous_bookings_not_cance
count	24335.000000	24335.000000	24335.000000	24335.000000	24335.000000	24335.000000	24335.000
mean	1.891720	0.139306	0.838463	2.138155	57.714732	0.029587	0.218
std	0.529389	0.446890	0.832628	1.274219	55.249023	0.418931	2.053
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	2.000000	0.000000	0.000000	1.000000	11.000000	0.000000	0.000
50%	2.000000	0.000000	1.000000	2.000000	40.000000	0.000000	0.000
75%	2.000000	0.000000	2.000000	3.000000	90.000000	0.000000	0.000
max	4.000000	2.000000	4.000000	5.000000	219.000000	13.000000	58.000

Finding statistics like mean, standard deviation, min, max etc., for all numeric features present in our dataset.

(2) CATEGORICAL DATATYPES

In [30]:
```
# Step-2
# Insight on category datatype columns
c = ["type_of_meal_plan","room_type_reserved","market_segment_type","booking_status","required_car_parking_space",'repeated_guest
for col in Cleaned_Raw_df.columns:
    if col in c:
        print(f"Feature: '{col}'")
        print(Cleaned_Raw_df[col].value_counts())
        print("        ")
```

```
Feature: 'type_of_meal_plan'
Meal Plan 1     20242
Not Selected     4452
Meal Plan 2      1128
Meal Plan 3         5
Name: type_of_meal_plan, dtype: int64

Feature: 'required_car_parking_space'
0    24742
1     1085
Name: required_car_parking_space, dtype: Int64

Feature: 'room_type_reserved'
Room_Type 1    18522
Room_Type 4     5387
Room_Type 6      935
Room_Type 2      595
Room_Type 5      227
Room_Type 7      155
```

This analysis helps us understand the categorical data types in a better way. It shows all the categories data in that column and the number of records present in each category.

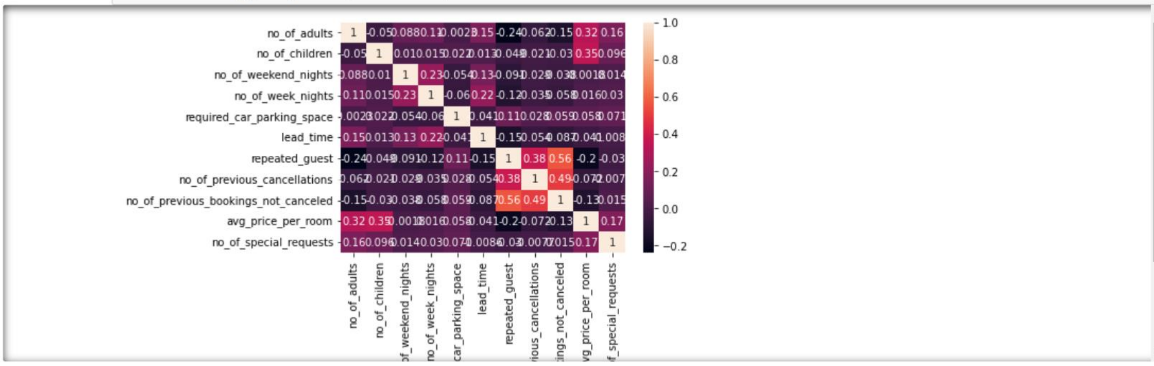
(3) CORRELATION MATRIX

In [31]:
```
# Step-3
# Correlation Matrix
df_corr = pd.DataFrame(Cleaned_Raw_df)
corr_matrix = df_corr.corr()
corr_matrix
```

Out[31]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	required_car_parking_space	lead_time	repeate
no_of_adults	1.000000	-0.050257	0.088385	0.110213	-0.002339	0.153400	-0
no_of_children	-0.050257	1.000000	0.010164	0.015489	0.021609	0.012585	-0
no_of_weekend_nights	0.088385	0.010164	1.000000	0.233935	-0.054081	0.134373	-0
no_of_week_nights	0.110213	0.015489	0.233935	1.000000	-0.059763	0.223469	-0
required_car_parking_space	-0.002339	0.021609	-0.054081	-0.059763	1.000000	-0.040765	0
lead_time	0.153400	0.012585	0.134373	0.223469	-0.040765	1.000000	-0
repeated_guest	-0.239530	-0.049290	-0.090560	-0.120156	0.108707	-0.148775	1
no_of_previous_cancellations	-0.062359	-0.020962	-0.029177	-0.034887	0.027503	-0.053765	0
no_of_previous_bookings_not_canceled	-0.145691	-0.029617	-0.038088	-0.057829	0.059460	-0.087075	0
avg_price_per_room	0.324807	0.349949	-0.001785	0.015601	0.057513	-0.041053	-0
no_of_special_requests	0.156699	0.096281	0.014092	0.030168	0.070874	-0.008619	-0

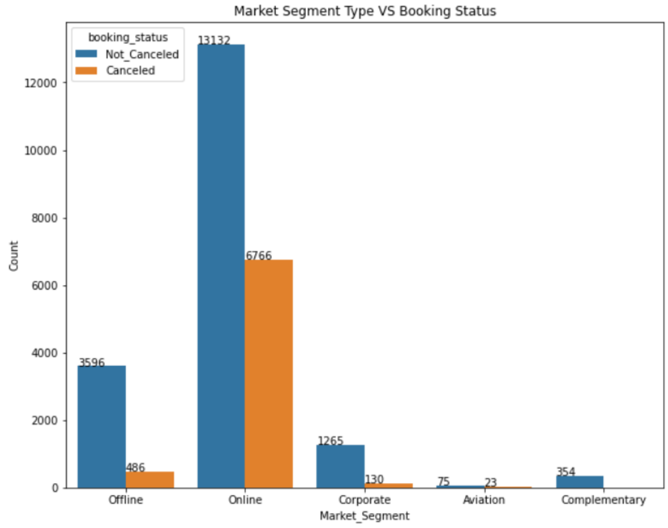
In [32]:
```
# Plotting the correlation matrix
ax = sns.heatmap(corr_matrix, annot=True)
```



A correlation matirx is a table which shows the correlation coefficients for different variables present in the dataset and can be compared. The correlation of a feature when measured by itself gives 1. That's the reson we see 1 in all the diagonal elements.

(4) PLOT BETWEEN MARKET SEGMENT AND BOOKING STATUS

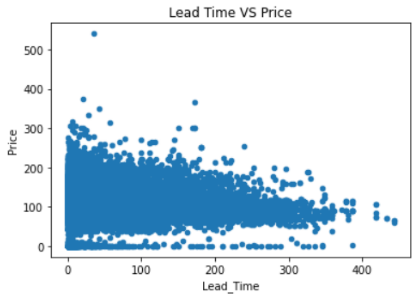
In [33]:
```
# Step-4
# Plot to know how the market segment is effecting the booking status

plt.figure(figsize=(10,8))
ax=sns.countplot(data=Cleaned_Raw_df,x='market_segment_type',hue='booking_status')
plt.title('Market Segment Type VS Booking Status')
plt.xlabel('Market_Segment')
plt.ylabel('Count')
for g in ax.patches:
        ax.annotate(f'{g.get_height():.0f}', (g.get_x(), g.get_height()))
```



This plot was made to analyze how the market segment is effecting the booking status and can derive that online booking have the highest bookings made and also the highest number of non cancellations and cancellations. Aviation on the other hand has the least booking with almost 75% of non cancellations. if any one get complementry then there is no possibility that they will cancel it. Aviation and corporate people cancel the least.

(5) SCATTERPLOT BETWEEN LEAD TIME AND PRICE

In [34]:
```
# Step-5
# Scatter Plot to know how the lead time affects the price
df = pd.DataFrame(Cleaned_Raw_df, columns=["lead_time", "avg_price_per_room"])
df.plot(x="lead_time", y="avg_price_per_room", kind='scatter')
plt.title('Lead Time VS Price')
plt.xlabel('Lead_Time')
plt.ylabel('Price')
plt.show()
```

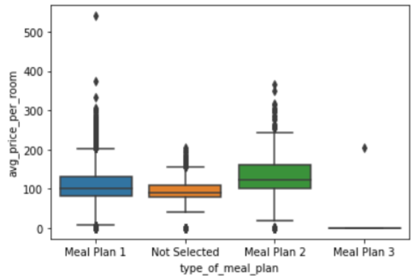


Here we analyze the relation between time of booking and how it affects the pricing of the hotel. And we can derive that the closer the arrival date is to lead time the price gets higer on an average.

(6) BOXPLOT BETWEEN MEAL PLAN TYPE AND AVERAGE PRICE OF ROOM

In [35]:
```
# Step-6
sns.boxplot(x = Cleaned_Raw_df['type_of_meal_plan'], y = Cleaned_Raw_df['avg_price_per_room'])
```

Out[35]:
```
<AxesSubplot:xlabel='type_of_meal_plan', ylabel='avg_price_per_room'>
```



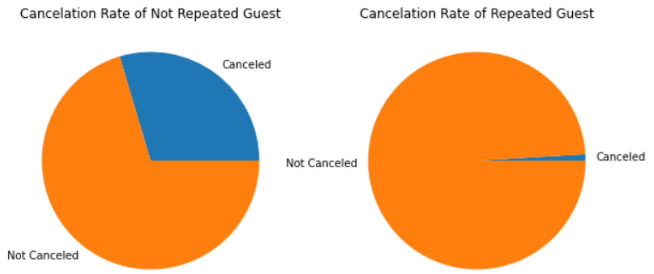
This boxplot shows us the relationship between the type of meal plan selected and the average price of the room. And we can derive that that the higher the price increases, the customer prefers Meal Plan 2.

(7) PIECHART TO SHOW CANCELATION BY REPEATED AND NON REPEATED CUSTOMERS

In [36]:
```
# Step-7
# Grouped booking_status with respect to repeated guests to know the cancellation trend of a repeated guest vs non-repeated guest
Cleaned_Raw_df.groupby("repeated_guest")["booking_status"].value_counts()
```

Out[36]:
```
repeated_guest  booking_status
0               Not_Canceled      17592
                Canceled           7397
1               Not_Canceled        830
                Canceled              8
Name: booking_status, dtype: int64
```

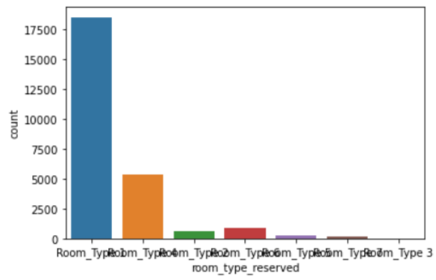
In [37]:
```
# Plotting a pie chart from the obtained data
fig, ax = plt.subplots(1, 2, figsize=(10, 5), subplot_kw={'aspect': 'equal'})
ax[0].pie([7397,17592], labels=["Canceled","Not Canceled"])
ax[0].set_title("Cancelation Rate of Not Repeated Guest")
ax[1].pie([8,830], labels=["Canceled","Not Canceled"])
ax[1].set_title("Cancelation Rate of Repeated Guest");
```



This piechart shows us the relation between cancellations and whether the guest is a repeated guest or not. It shows that Most of the repeated guests preffer to keep their reservation.

(8) COUNTPLOT FOR BOOKING OF EACH TYPES OF ROOMS IN THIS PERIOD

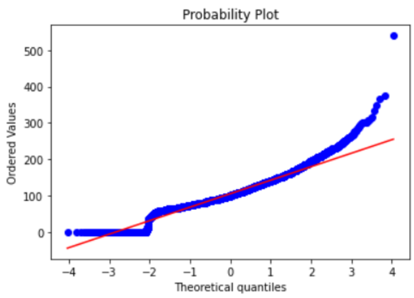
```
In [38]: # Step-8
         # Count plot of room_type to know the room preseference of the customers
         sns.countplot(Cleaned_Raw_df.room_type_reserved)

C:\Users\inish\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(

Out[38]: <AxesSubplot:xlabel='room_type_reserved', ylabel='count'>
```



This countplot is made to understand the preference of customers. This data can be used to understand the demand of each type of room so that the management can decide to take steps based on the bookings so that they can underbook or overbook.

(9) PROBABILITYPLOT OF AVERAGE PRICE PER ROOM

```
In [39]: # Step-9
         # Probability Plot of avg_price_per_room
         from scipy import stats
         stats.probplot(Cleaned_Raw_df.avg_price_per_room, dist='norm', plot = plt)

Out[39]: ((array([-4.039018  , -3.82591326, -3.70943118, ...,  3.70943118,
                   3.82591326,  4.039018  ]),
           array([  0. ,   0. ,   0. , ..., 365. , 375.5, 540. ])),
          (37.07821384730515, 105.75196073876178, 0.9785087666355398))
```



Here we try to analzye the distribution of a single column with the normal distribution. So we use this plot to analyze the price along with the normal distribution of the column.

(10) SKEWNESS OF AVERAGE PRICE PER ROOM

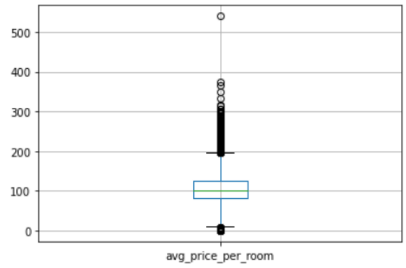
In [566]:
```
# Step-10
print("avg_price_per_room Skewness:", Cleaned_Raw_df["avg_price_per_room"].skew(axis = 0))
print("lead_time Skewness:", Cleaned_Raw_df["lead_time"].skew(axis = 0))

print("avg_price_per_room kutosis:", stats.kurtosis(Cleaned_Raw_df.avg_price_per_room, bias=True))
print("lead_time kurtosis:", stats.kurtosis(Cleaned_Raw_df.lead_time, bias=True))
```

```
avg_price_per_room Skewness: 0.5884097838674266
lead_time Skewness: 1.4069956394477487
avg_price_per_room kutosis: 2.615770813126737
lead_time kurtosis: 1.8712380360556624
```

In [46]:
```
print("Finding the spread and skewness of the above two columns using boxplots")
Cleaned_Raw_df.boxplot(column=['avg_price_per_room'])
```

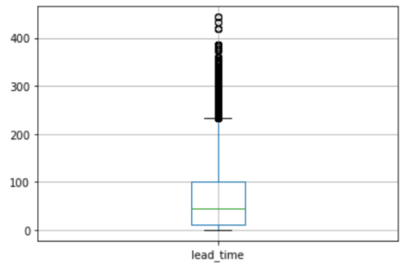
```
Finding the spread and skewness of the above two columns using boxplots
```

Out[46]: <AxesSubplot:>



In [47]:
```
Cleaned_Raw_df.boxplot(column=['lead_time'])
```

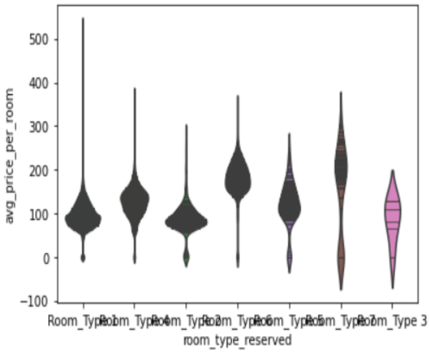
Out[47]: <AxesSubplot:>



Here we define skewness which defines the symmetry of the distribution of the column average price per room and lead_time. Using the boxplots we can see the spread and skewness of the features.

(11) VIOLIN GRAPH TO SHOW RELATION BETWEEN ROOM TYPE AND AVERAGE PRICE PER ROOM

In [49]:
```
# Step-11
# Violin Graph
sns.violinplot(x = 'room_type_reserved', y = "avg_price_per_room" ,data = Cleaned_Raw_df, inner="stick")
plt.show()
```



This graph shows us the reltion between the room type reserved by the customer and the average price of that room. We can also derive the summary statistics and the density of each variable with respect to the other.

NOTE:

The dataset used for entire EDA is before performing Encoding and Outliers. We have done that in the final steps of Data Cleaning.