Booking Battles- Reservations vs Cancellations

September 14, 2023

[33]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,␣
↪roc_auc_score,precision_score, recall_score
```

1 Import Data

[34]:
```
hr=pd.read_csv('Hotel Reservations.csv')
hr.head()
```

[34]:
```
  Booking_ID  no_of_adults  no_of_children  no_of_weekend_nights  \
0   INN00001             2               0                     1
1   INN00002             2               0                     2
2   INN00003             1               0                     2
3   INN00004             2               0                     0
4   INN00005             2               0                     1

   no_of_week_nights type_of_meal_plan  required_car_parking_space  \
0                  2       Meal Plan 1                           0
1                  3      Not Selected                           0
2                  1       Meal Plan 1                           0
3                  2       Meal Plan 1                           0
4                  1      Not Selected                           0

  room_type_reserved  lead_time  arrival_year  arrival_month  arrival_date  \
0        Room_Type 1        224          2017             10             2
1        Room_Type 1          5          2018             11             6
2        Room_Type 1          1          2018              2            28
```

```
3        Room_Type 1        211          2018              5            20
4        Room_Type 1         48          2018              4            11

  market_segment_type  repeated_guest  no_of_previous_cancellations  \
0             Offline               0                             0
1              Online               0                             0
2              Online               0                             0
3              Online               0                             0
4              Online               0                             0

   no_of_previous_bookings_not_canceled  avg_price_per_room  \
0                                     0               65.00
1                                     0              106.68
2                                     0               60.00
3                                     0              100.00
4                                     0               94.50

   no_of_special_requests booking_status
0                       0   Not_Canceled
1                       1   Not_Canceled
2                       0       Canceled
3                       0       Canceled
4                       0       Canceled
```

[35]:

```python
hr.columns
```

[35]:

```
Index(['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_year', 'arrival_month',
       'arrival_date', 'market_segment_type', 'repeated_guest',
       'no_of_previous_cancellations', 'no_of_previous_bookings_not_canceled',
       'avg_price_per_room', 'no_of_special_requests', 'booking_status'],
      dtype='object')
```

[36]:

```python
hr.shape
```

[36]:

```
(36275, 19)
```

2 Data Cleaning

Check for null values

[37]:

```python
missing_data = hr.isnull().sum()
missing_data
```

[37]:

```
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
```

Number of distinct values in each column

[38]:
```
hr.nunique()
```

[38]:
```
Booking_ID                              36275
no_of_adults                                5
no_of_children                              6
no_of_weekend_nights                        8
no_of_week_nights                          18
type_of_meal_plan                           4
required_car_parking_space                  2
room_type_reserved                          7
lead_time                                 352
arrival_year                                2
arrival_month                              12
arrival_date                               31
market_segment_type                         5
repeated_guest                              2
no_of_previous_cancellations                9
no_of_previous_bookings_not_canceled       59
avg_price_per_room                       3930
no_of_special_requests                      6
booking_status                              2
dtype: int64
```

Unique values for specific columns

[39]:
```
cols1=['no_of_adults','no_of_children','type_of_meal_plan','required_car_parking_space','room_
d = {}
```

```
for c in cols1:
    d[c] = hr[c].unique()
d
```

[39]:
```
{'no_of_adults': array([2, 1, 3, 0, 4]),
 'no_of_children': array([ 0,  2,  1,  3, 10,  9]),
 'type_of_meal_plan': array(['Meal Plan 1', 'Not Selected', 'Meal Plan 2', 'Meal
Plan 3'],
       dtype=object),
 'required_car_parking_space': array([0, 1]),
 'room_type_reserved': array(['Room_Type 1', 'Room_Type 4', 'Room_Type 2',
'Room_Type 6',
        'Room_Type 5', 'Room_Type 7', 'Room_Type 3'], dtype=object),
 'market_segment_type': array(['Offline', 'Online', 'Corporate', 'Aviation',
'Complementary'],
       dtype=object),
 'repeated_guest': array([0, 1]),
 'booking_status': array(['Not_Canceled', 'Canceled'], dtype=object)}
```

3 Feature Engineering

Summary stats for each variable

[40]:
```
cols=['lead_time','no_of_previous_cancellations','no_of_previous_bookings_not_canceled','avg_p
summary = hr[cols].describe()

# Display the summary
summary
```

[40]:
```
           lead_time  no_of_previous_cancellations  \
count  36275.000000                  36275.000000
mean      85.232557                      0.023349
std       85.930817                      0.368331
min        0.000000                      0.000000
25%       17.000000                      0.000000
50%       57.000000                      0.000000
75%      126.000000                      0.000000
max      443.000000                     13.000000

       no_of_previous_bookings_not_canceled  avg_price_per_room  \
count                          36275.000000        36275.000000
mean                               0.153411          103.423539
std                                1.754171           35.089424
min                                0.000000            0.000000
25%                                0.000000           80.300000
50%                                0.000000           99.450000
75%                                0.000000          120.000000
```

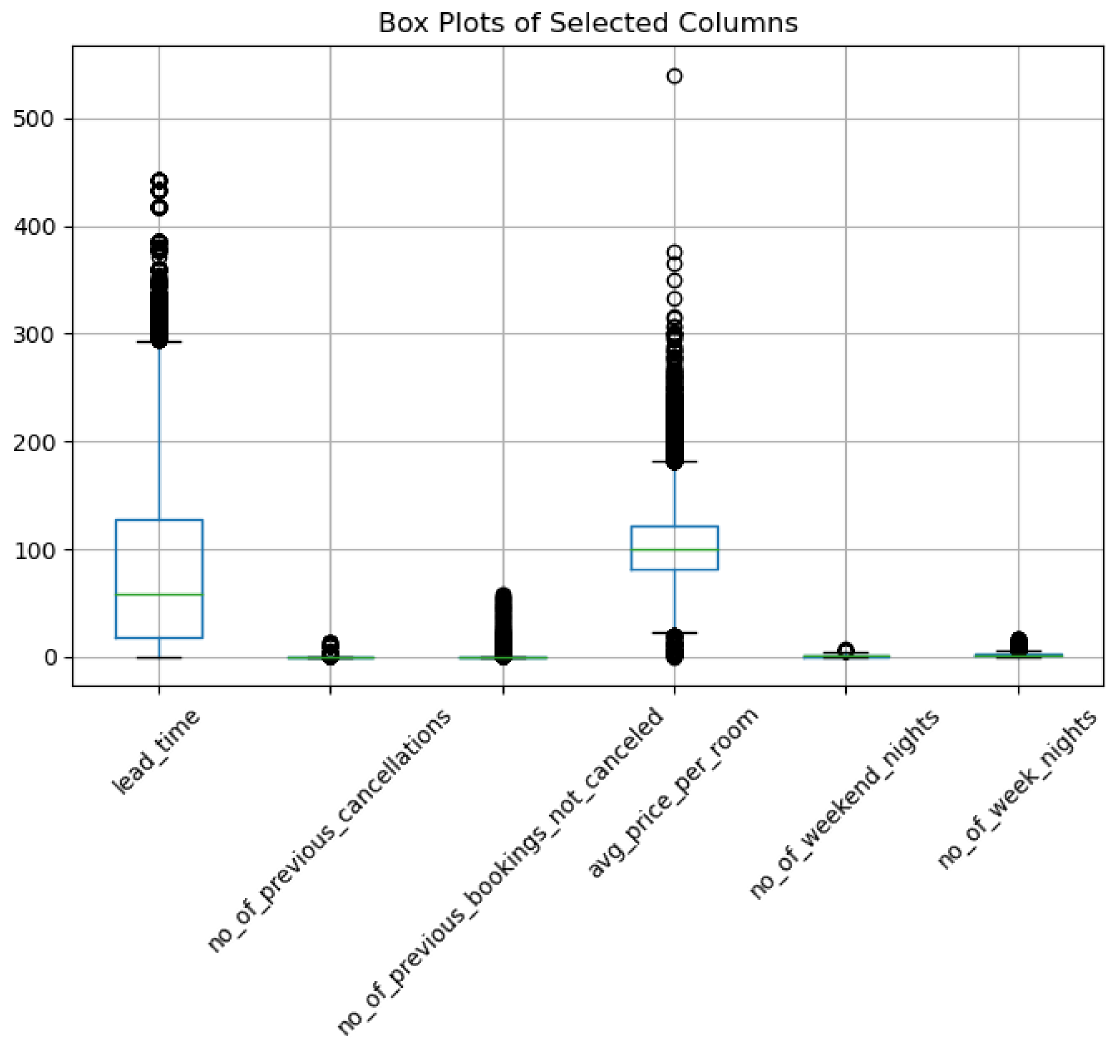
```
max                                 58.000000          540.000000

       no_of_weekend_nights  no_of_week_nights
count          36275.000000       36275.000000
mean               0.810724           2.204300
std                0.870644           1.410905
min                0.000000           0.000000
25%                0.000000           1.000000
50%                1.000000           2.000000
75%                2.000000           3.000000
max                7.000000          17.000000
```

Average price per room cannot be zero so we remove those rows

[41]:
```
# Remove those rows
hr=hr[hr['avg_price_per_room']>0]
```

Create box plots for specific columns

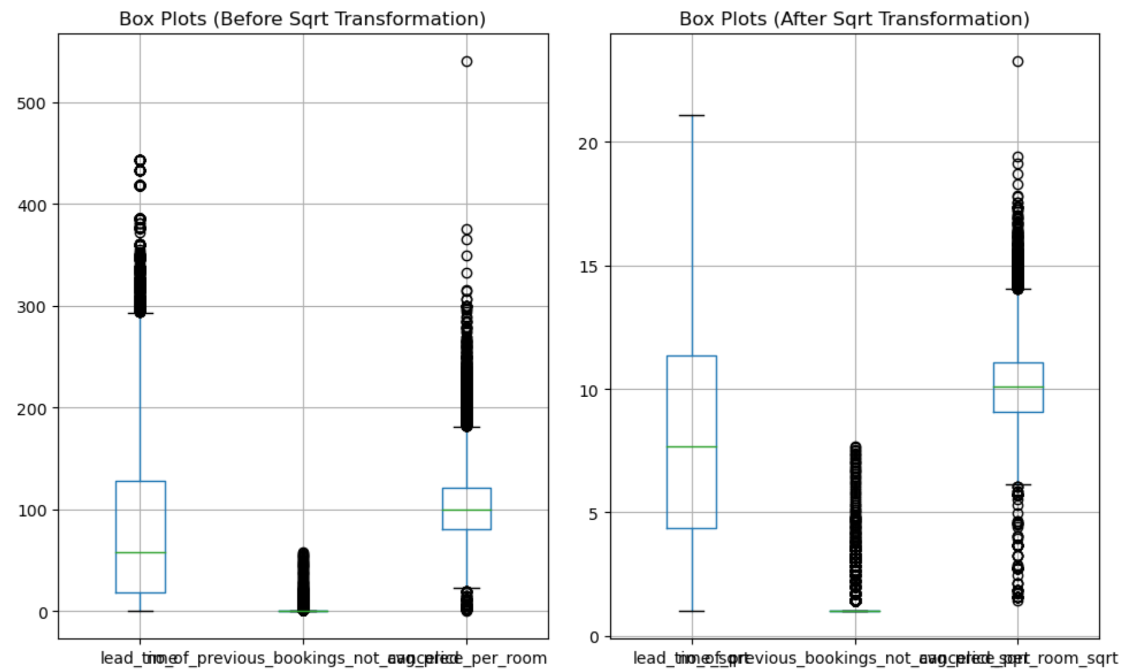
[42]:
```
plt.figure(figsize=(8, 5))
hr[cols].boxplot()
plt.title('Box Plots of Selected Columns')
plt.xticks(rotation=45)
plt.show()
```



Looks like lead time, no of previous bookings not cancelled, average price per room have outliers To deal with outliers, we follow these steps: 1. We write a function to count the number of outliers 2. We transform the data to handle outliers 3. We gauge the effectiveness of the transformation by comparing the number of outliers before and after the transformation

[43]:
```
# Columns with outliers
outlier_cols = ['lead_time', 'no_of_previous_bookings_not_canceled',␣
↪'avg_price_per_room']

# Function to count outliers using IQR method
def count_outliers(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
```

```python
    upper_bound = Q3 + 1.5 * IQR
    return data[(data < lower_bound) | (data > upper_bound)].count()

# Count outliers before sqrt transformation
outliers_before = hr[outlier_cols].apply(count_outliers)

# Apply sqrt transformation with an offset to each column
for col in outlier_cols:
    offset = hr[col].min() + 1  # Add 1 or a small positive value to handle␣
 ↪zeros
    hr[f'{col}_sqrt'] = np.sqrt(hr[col] + offset)

# Count outliers after sqrt transformation
outliers_after = hr[[f'{col}_sqrt' for col in outlier_cols]].
 ↪apply(count_outliers)

# Plot box plots before and after sqrt transformation
plt.figure(figsize=(10, 6))

# Before sqrt transformation
plt.subplot(1, 2, 1)
hr[outlier_cols].boxplot()
plt.title('Box Plots (Before Sqrt Transformation)')

# After sqrt transformation
plt.subplot(1, 2, 2)
hr[[f'{col}_sqrt' for col in outlier_cols]].boxplot()
plt.title('Box Plots (After Sqrt Transformation)')

plt.tight_layout()
plt.show()

# Display the number of outliers before and after transformation
print("Number of Outliers (Before Sqrt Transformation):")
print(outliers_before)

print("\nNumber of Outliers (After Sqrt Transformation):")
print(outliers_after)
```




```
Number of Outliers (Before Sqrt Transformation):
lead_time                                  1184
no_of_previous_bookings_not_canceled        690
avg_price_per_room                         1101
dtype: int64

Number of Outliers (After Sqrt Transformation):
lead_time_sqrt                                0
no_of_previous_bookings_not_canceled_sqrt   690
avg_price_per_room_sqrt                     711
dtype: int64
```

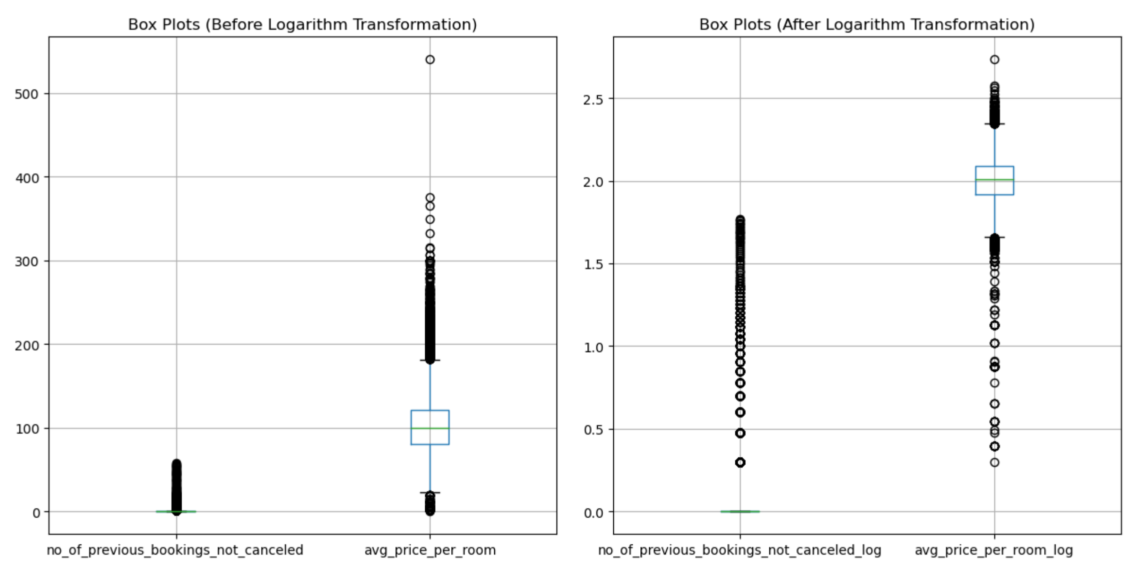
Clearly the sqrt transformation worked to handle outliers from lead time but not the other variables
So we drop the sqrt columns for the other two variables

[44]:
```python
hr=hr.
↪drop(columns=['no_of_previous_bookings_not_canceled_sqrt','avg_price_per_room_sqrt'],axis=1
```

We now apply the logarithmic transformation to the remaining two variables

[45]:
```python
outlier_cols_1 = ['no_of_previous_bookings_not_canceled', 'avg_price_per_room']

outliers_before = hr[outlier_cols_1].apply(count_outliers)

# Apply logarithm transformation with an offset to each column
for col in outlier_cols_1:
```

```
    offset = hr[col].min() + 1  # Add 1 or a small positive value to handle␣
↪zeros
    hr[f'{col}_log'] = np.log10(hr[col] + offset)

# Count outliers after logarithm transformation
outliers_after = hr[[f'{col}_log' for col in outlier_cols_1]].
↪apply(count_outliers)

# Plot box plots before and after logarithm transformation
plt.figure(figsize=(12, 6))

# Before logarithm transformation
plt.subplot(1, 2, 1)
hr[outlier_cols_1].boxplot()
plt.title('Box Plots (Before Logarithm Transformation)')

# After logarithm transformation
plt.subplot(1, 2, 2)
hr[[f'{col}_log' for col in outlier_cols_1]].boxplot()
plt.title('Box Plots (After Logarithm Transformation)')

plt.tight_layout()
plt.show()

# Display the number of outliers before and after transformation
print("Number of Outliers (Before Logarithm Transformation):")
print(outliers_before)

print("\nNumber of Outliers (After Logarithm Transformation):")
print(outliers_after)
```



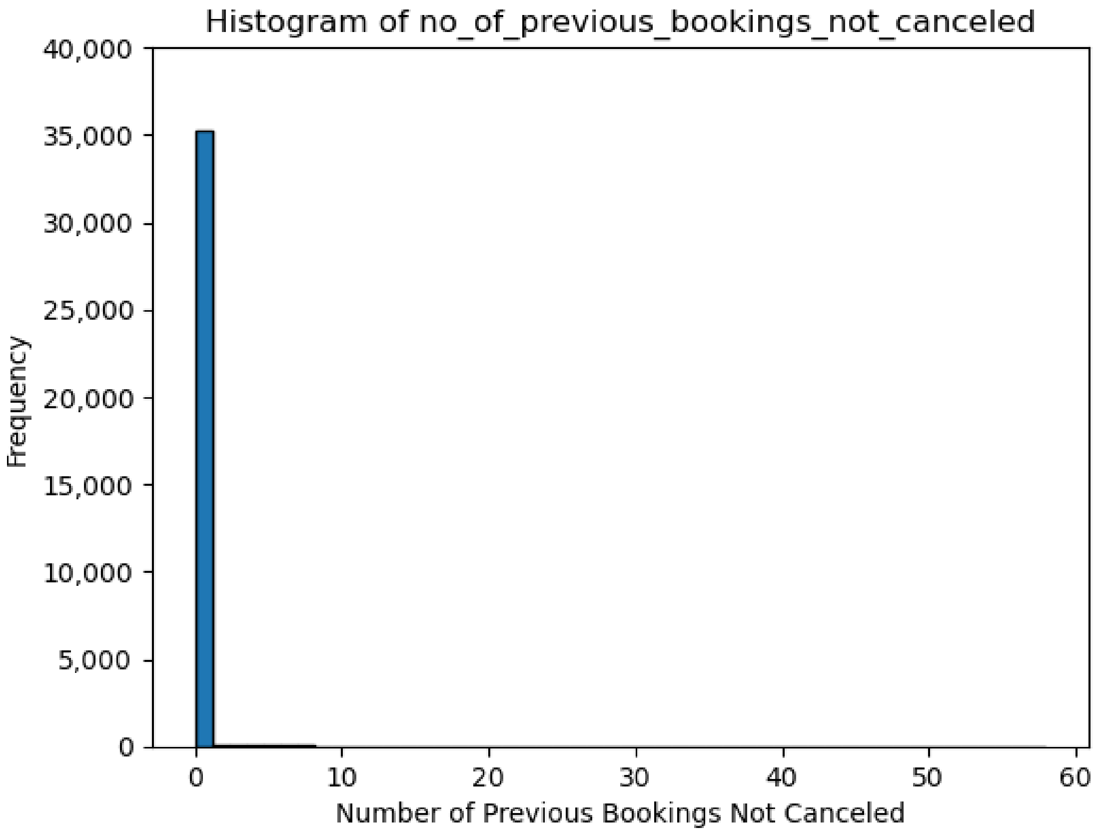
```
Number of Outliers (Before Logarithm Transformation):
no_of_previous_bookings_not_canceled      690
avg_price_per_room                       1101
dtype: int64

Number of Outliers (After Logarithm Transformation):
no_of_previous_bookings_not_canceled_log    690
avg_price_per_room_log                      451
dtype: int64
```

The log transformation significantly reduced the outliers for avg_price_per_room_log by ~60% but did not help with no_of_previous_bookings_not_canceled

[46]:
```python
# Not removing the original avg_price_per_room variable because we need it for␣
 ↪EDA
hr=hr.drop(columns=['no_of_previous_bookings_not_canceled_log'],axis=1)
```

To further understand no_of_previous_bookings_not_canceled, we plot its histogram

[47]:
```python
# Plot histogram for 'no_of_previous_bookings_not_canceled'
plt.hist(hr['no_of_previous_bookings_not_canceled'], bins=50, edgecolor='black')
plt.xlabel('Number of Previous Bookings Not Canceled')
plt.ylabel('Frequency')
plt.title('Histogram of no_of_previous_bookings_not_canceled')

# Get current y-axis ticks and labels
yticks, ylabels = plt.yticks()

# Increase the spacing between y-axis ticks and labels
plt.yticks(yticks, [f'{int(y):,}' for y in yticks])

plt.show()
```



Even though none of our transformations were effective in handling the outliers in no_of_previous_bookings_not_canceled, we choose not remove the outliers for this variable because we might be losing valuable data

Now we move on to checking if the data is imbalanced In case of imbalanced data, we need to apply techniques like SMOTE, etc. to correct the imbalance before running the model

[48]:
```
# Check the class distribution for the 'booking_status' column
booking_status_counts = hr['booking_status'].value_counts()

# Calculate the percentage of each class
total_bookings = len(hr)
class_proportions = booking_status_counts / total_bookings * 100

# Print class distribution
print("Class Distribution:")
print(booking_status_counts)
print("\nClass Proportions (%):")
print(class_proportions)
```

```
Class Distribution:
Not_Canceled    23851
```

```
Canceled        11879
Name: booking_status, dtype: int64

Class Proportions (%):
Not_Canceled    66.753428
Canceled        33.246572
Name: booking_status, dtype: float64
```

The data shows a healthy balance and thus, there is no need for synthetic over-sampling

We combine the date columns and convert to datetime We need this so we can track bookings over time for EDA We also remove rows with invalid dates e.g., February 29 in non-leap years)

[49]:
```
# Remove rows with invalid dates
hr = hr[~((hr['arrival_month'] == 2) & (hr['arrival_date'] == 29) &␣
 ↪~(hr['arrival_year'] % 4 == 0))]

# Combine 'arrival_year', 'arrival_month', and 'arrival_date' with a custom␣
 ↪separator
hr['arrival_date_combined'] = hr['arrival_year'].astype(str) + '-' +␣
 ↪hr['arrival_month'].astype(str) + '-' + hr['arrival_date'].astype(str)

# Convert the concatenated date column to a datetime object
hr['arrival_date_combined'] = pd.to_datetime(hr['arrival_date_combined'])

# Drop the original 'arrival_date', 'arrival_month', and 'arrival_year' columns
hr.drop(['arrival_date', 'arrival_month', 'arrival_year'], axis=1, inplace=True)
```

Convert 'booking_status' into dummy variables

[50]:
```
hr=pd.get_dummies(hr,columns=['booking_status'],drop_first=False)
hr=hr.drop('booking_status_Not_Canceled', axis=1)
```

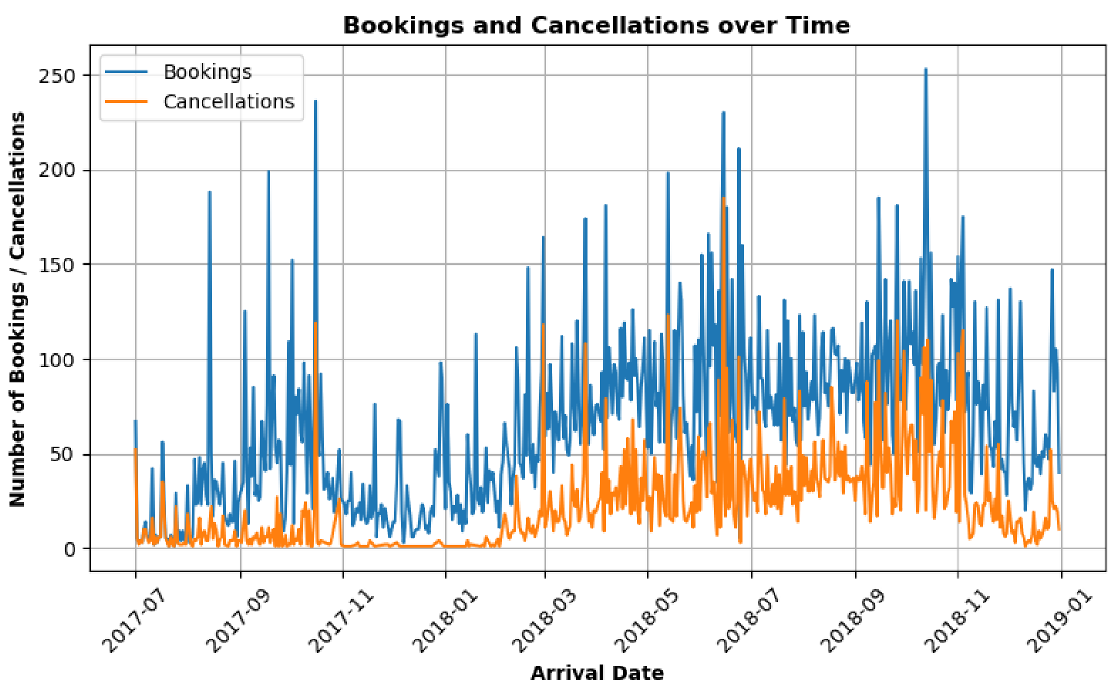
We cannot use the dates as is for features and need to extract day, week, quarter, etc. from it

[51]:
```
def feature_engineering_dates(df):
    df['year'] = df['arrival_date_combined'].dt.year
    df['month'] = df['arrival_date_combined'].dt.month
    df['day'] = df['arrival_date_combined'].dt.day
    df['week'] = df['arrival_date_combined'].dt.isocalendar().week.astype(float)
    df['dayofweek'] = df['arrival_date_combined'].dt.dayofweek
    df['quarter'] = df['arrival_date_combined'].dt.quarter
    df['dayofyear'] = df['arrival_date_combined'].dt.dayofyear

    return df

# Call the function to create date-related features
hr = feature_engineering_dates(hr)
```

4 Exploratory Data Analysis

Let's look at the distribution of Bookings and Cancellations over time

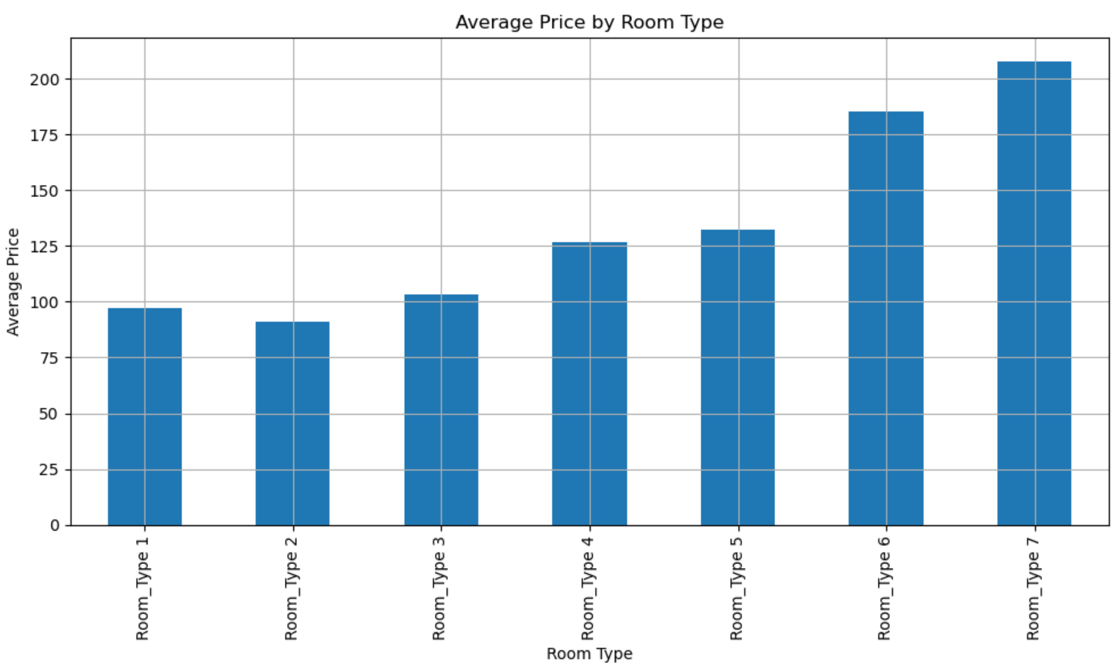
[52]:
```
# Group the data by arrival dates and count the total bookings for each date
bookings_by_date = hr.groupby('arrival_date_combined').size()

# Group the data by arrival dates and count the cancellations for each date
cancellations_by_date = hr[hr['booking_status_Canceled'] == 1].
↪groupby('arrival_date_combined').size()

# Plot the distribution of bookings and cancellations over time
plt.figure(figsize=(8, 5))
plt.plot(bookings_by_date.index, bookings_by_date.values, label='Bookings')
plt.plot(cancellations_by_date.index, cancellations_by_date.values,␣
↪label='Cancellations')
plt.xlabel('Arrival Date', fontdict={'weight': 'bold'})
plt.ylabel('Number of Bookings / Cancellations', fontdict={'weight': 'bold'})
plt.title('Bookings and Cancellations over Time', fontdict={'weight': 'bold',␣
↪'size': 12})
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()  # Add some space between title and plot
plt.show()
```



Looks like there are mostly bulk bookings and cancellations

Look at the average price of each type of room

[53]:
```
# Group data by 'room_type_reserved' and calculate average price for each group
average_price_by_room_type = hr.
 ↪groupby('room_type_reserved')['avg_price_per_room'].mean()

# Plot the data
plt.figure(figsize=(10, 6))
average_price_by_room_type.plot(kind='bar')
plt.xlabel('Room Type')
plt.ylabel('Average Price')
plt.title('Average Price by Room Type')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Looks like Room Type 1 is the cheapest and Room Type 7 is the most expensive

What is the profile of the guests making the reservations?

[54]:
```
# List of profile features to analyze
profile_features = ['type_of_meal_plan', 'room_type_reserved',␣
 ↪'market_segment_type',
                    'no_of_adults', 'no_of_children', 'repeated_guest',␣
 ↪'no_of_special_requests','required_car_parking_space',
```

```
                    'year','month','day','week','dayofweek','quarter']

# Create a function to display table for each profile feature
def display_tables(feature):
    cancellation_counts = hr.groupby([feature, 'booking_status_Canceled']).
 ↪size().unstack()

    # Calculate percentages
    cancellation_counts['Total'] = cancellation_counts.sum(axis=1)
    cancellation_counts['Not Canceled %'] = ((cancellation_counts[0] /␣
 ↪cancellation_counts['Total'])*100).round(0)
    cancellation_counts['Canceled %'] = ((cancellation_counts[1] /␣
 ↪cancellation_counts['Total'])*100).round(0)

    # Display table
    print(f"Table for {feature}:\n")
    print(cancellation_counts)

# Display table for each profile feature
for feature in profile_features:
    display_tables(feature)
    print("\n")
```

```
Table for type_of_meal_plan:

booking_status_Canceled        0       1    Total  Not Canceled %  Canceled %
type_of_meal_plan
Meal Plan 1              18683.0  8669.0  27352.0            68.0        32.0
Meal Plan 2               1739.0  1504.0   3243.0            54.0        46.0
Meal Plan 3                  NaN     1.0      1.0             NaN       100.0
Not Selected              3399.0  1698.0   5097.0            67.0        33.0


Table for room_type_reserved:

booking_status_Canceled      0     1  Total  Not Canceled %  Canceled %
room_type_reserved
Room_Type 1              18657  9060  27717            67.0        33.0
Room_Type 2                439   228    667            66.0        34.0
Room_Type 3                  3     2      5            60.0        40.0
Room_Type 4               3924  2068   5992            65.0        35.0
Room_Type 5                174    72    246            71.0        29.0
Room_Type 6                542   406    948            57.0        43.0
Room_Type 7                 82    36    118            69.0        31.0


Table for market_segment_type:
```

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
market_segment_type					
Aviation	88.0	37.0	125.0	70.0	30.0
Complementary	36.0	NaN	36.0	100.0	NaN
Corporate	1791.0	220.0	2011.0	89.0	11.0
Offline	7366.0	3152.0	10518.0	70.0	30.0
Online	14540.0	8463.0	23003.0	63.0	37.0

Table for no_of_adults:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
no_of_adults					
0	91	44	135	67.0	33.0
1	5528	1850	7378	75.0	25.0
2	16742	9112	25854	65.0	35.0
3	1448	863	2311	63.0	37.0
4	12	3	15	80.0	20.0

Table for no_of_children:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
no_of_children					
0	22163.0	10871.0	33034.0	67.0	33.0
1	1057.0	538.0	1595.0	66.0	34.0
2	587.0	457.0	1044.0	56.0	44.0
3	12.0	5.0	17.0	71.0	29.0
9	1.0	1.0	2.0	50.0	50.0
10	1.0	NaN	1.0	100.0	NaN

Table for repeated_guest:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
repeated_guest					
0	23037	11857	34894	66.0	34.0
1	784	15	799	98.0	2.0

Table for no_of_special_requests:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
no_of_special_requests					
0	10948.0	8534.0	19482.0	56.0	44.0
1	8479.0	2701.0	11180.0	76.0	24.0
2	3665.0	637.0	4302.0	85.0	15.0

3	655.0	NaN	655.0	100.0	NaN
4	66.0	NaN	66.0	100.0	NaN
5	8.0	NaN	8.0	100.0	NaN

Table for required_car_parking_space:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
required_car_parking_space					
0	22841	11758	34599	66.0	34.0
1	980	114	1094	90.0	10.0

Table for year:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
year					
2017	5323	958	6281	85.0	15.0
2018	18498	10914	29412	63.0	37.0

Table for month:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
month					
1	959	24	983	98.0	2.0
2	1219	423	1642	74.0	26.0
3	1637	700	2337	70.0	30.0
4	1717	995	2712	63.0	37.0
5	1620	948	2568	63.0	37.0
6	1883	1290	3173	59.0	41.0
7	1580	1314	2894	55.0	45.0
8	2267	1487	3754	60.0	40.0
9	2999	1537	4536	66.0	34.0
10	3356	1877	5233	64.0	36.0
11	2055	875	2930	70.0	30.0
12	2529	402	2931	86.0	14.0

Table for day:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
day					
1	649	465	1114	58.0	42.0
2	1006	308	1314	77.0	23.0
3	683	403	1086	63.0	37.0
4	836	473	1309	64.0	36.0
5	805	328	1133	71.0	29.0

6	814	444	1258	65.0	35.0
7	730	364	1094	67.0	33.0
8	826	355	1181	70.0	30.0
9	816	294	1110	74.0	26.0
10	715	318	1033	69.0	31.0
11	746	330	1076	69.0	31.0
12	726	460	1186	61.0	39.0
13	935	407	1342	70.0	30.0
14	902	326	1228	73.0	27.0
15	722	538	1260	57.0	43.0
16	815	473	1288	63.0	37.0
17	871	447	1318	66.0	34.0
18	881	365	1246	71.0	29.0
19	902	413	1315	69.0	31.0
20	848	413	1261	67.0	33.0
21	752	376	1128	67.0	33.0
22	658	351	1009	65.0	35.0
23	631	341	972	65.0	35.0
24	718	372	1090	66.0	34.0
25	739	395	1134	65.0	35.0
26	706	425	1131	62.0	38.0
27	728	313	1041	70.0	30.0
28	712	405	1117	64.0	36.0
29	817	327	1144	71.0	29.0
30	743	465	1208	62.0	38.0
31	389	178	567	69.0	31.0

Table for week:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
week					
1.0	247	11	258	96.0	4.0
2.0	160	6	166	96.0	4.0
3.0	274	3	277	99.0	1.0
4.0	222	11	233	95.0	5.0
5.0	183	27	210	87.0	13.0
6.0	247	71	318	78.0	22.0
7.0	351	99	450	78.0	22.0
8.0	348	74	422	82.0	18.0
9.0	399	233	632	63.0	37.0
10.0	366	128	494	74.0	26.0
11.0	341	129	470	73.0	27.0
12.0	411	256	667	62.0	38.0
13.0	350	131	481	73.0	27.0
14.0	399	240	639	62.0	38.0
15.0	407	201	608	67.0	33.0
16.0	404	276	680	59.0	41.0

17.0	409	224	633	65.0	35.0
18.0	369	194	563	66.0	34.0
19.0	396	276	672	59.0	41.0
20.0	380	222	602	63.0	37.0
21.0	316	166	482	66.0	34.0
22.0	410	239	649	63.0	37.0
23.0	469	288	757	62.0	38.0
24.0	469	432	901	52.0	48.0
25.0	404	269	673	60.0	40.0
26.0	413	288	701	59.0	41.0
27.0	357	296	653	55.0	45.0
28.0	368	242	610	60.0	40.0
29.0	335	325	660	51.0	49.0
30.0	364	237	601	61.0	39.0
31.0	403	330	733	55.0	45.0
32.0	515	318	833	62.0	38.0
33.0	647	410	1057	61.0	39.0
34.0	472	339	811	58.0	42.0
35.0	518	270	788	66.0	34.0
36.0	672	338	1010	67.0	33.0
37.0	684	360	1044	66.0	34.0
38.0	857	360	1217	70.0	30.0
39.0	617	411	1028	60.0	40.0
40.0	854	435	1289	66.0	34.0
41.0	872	530	1402	62.0	38.0
42.0	771	437	1208	64.0	36.0
43.0	568	298	866	66.0	34.0
44.0	590	513	1103	53.0	47.0
45.0	529	117	646	82.0	18.0
46.0	536	179	715	75.0	25.0
47.0	463	186	649	71.0	29.0
48.0	408	101	509	80.0	20.0
49.0	718	76	794	90.0	10.0
50.0	314	45	359	87.0	13.0
51.0	394	56	450	88.0	12.0
52.0	851	169	1020	83.0	17.0

Table for dayofweek:

booking_status_Canceled	0	1	Total	Not Canceled %	Canceled %
dayofweek					
0	3650	1654	5304	69.0	31.0
1	3272	1585	4857	67.0	33.0
2	3378	1679	5057	67.0	33.0
3	3048	1429	4477	68.0	32.0
4	3150	1507	4657	68.0	32.0
5	3587	1728	5315	67.0	33.0

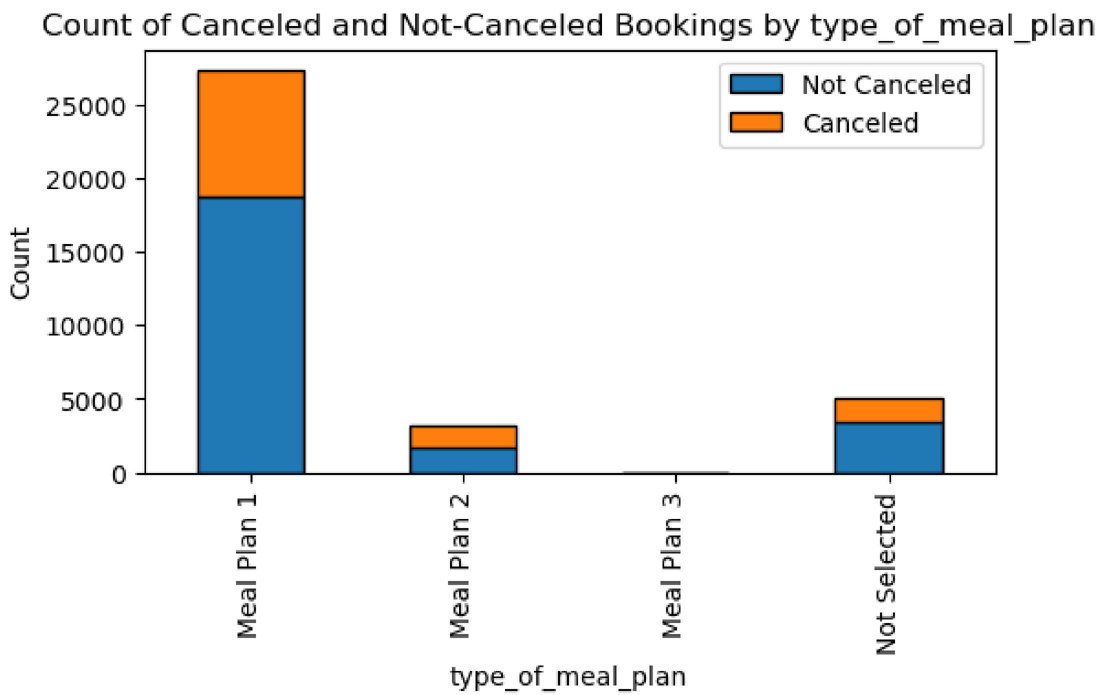
```
6                         3736  2290   6026            62.0         38.0


Table for quarter:

booking_status_Canceled      0     1  Total  Not Canceled %  Canceled %
quarter
1                         3815  1147   4962            77.0         23.0
2                         5220  3233   8453            62.0         38.0
3                         6846  4338  11184            61.0         39.0
4                         7940  3154  11094            72.0         28.0
```

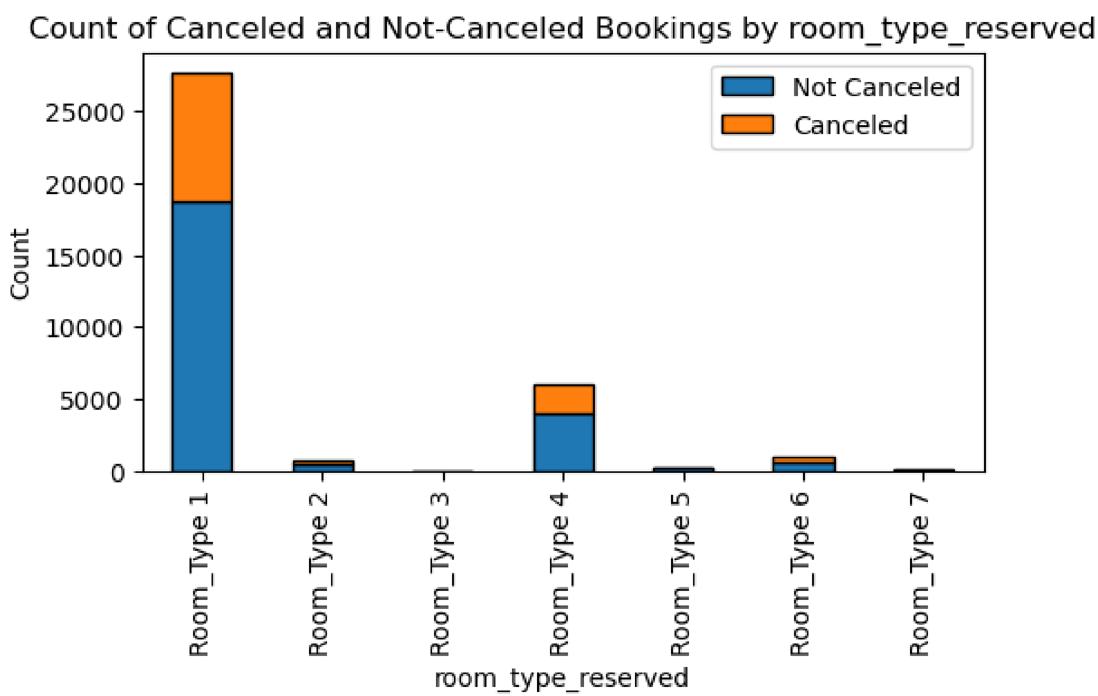
Insights:

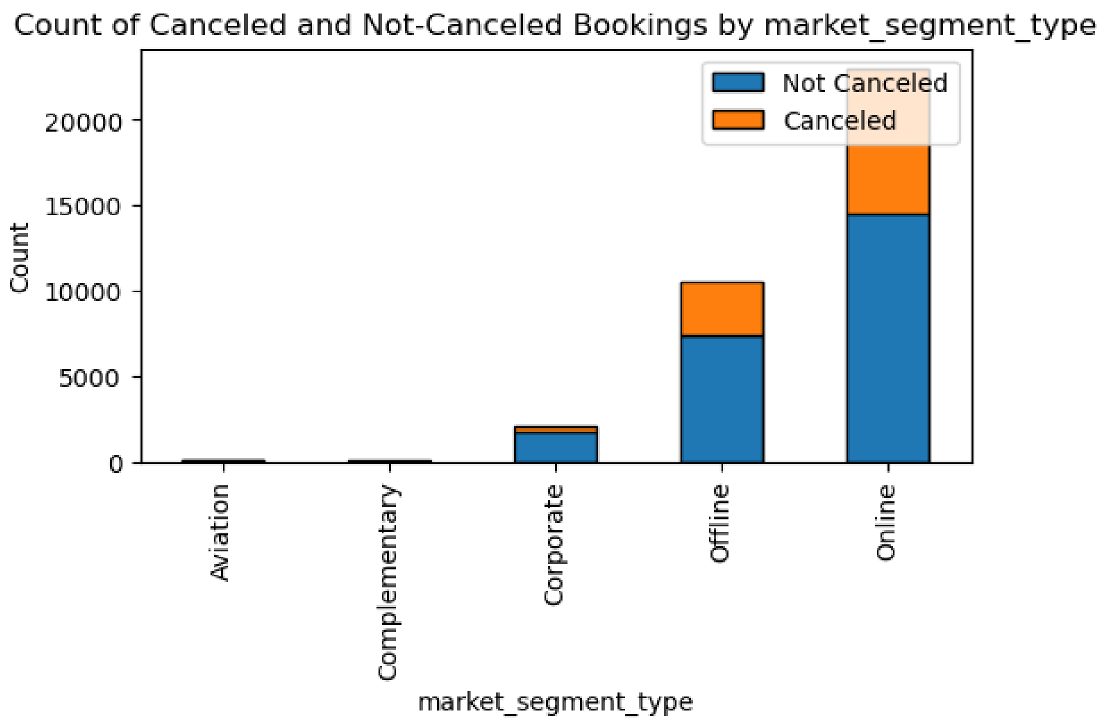
- Almost no one chooses meal plan 3. It had one booking and that was cancelled.
- Coporate and Complementary bookings have the lowest cancellation rates
- Repeated guests have much lower cancellations, as do people with special requests and those who need parking
- Cancellation % has more than doubled from 2017 to 2018
- Cancellation % are the lowest in December and January
- Bookings go up consistently from the first to the third quarter

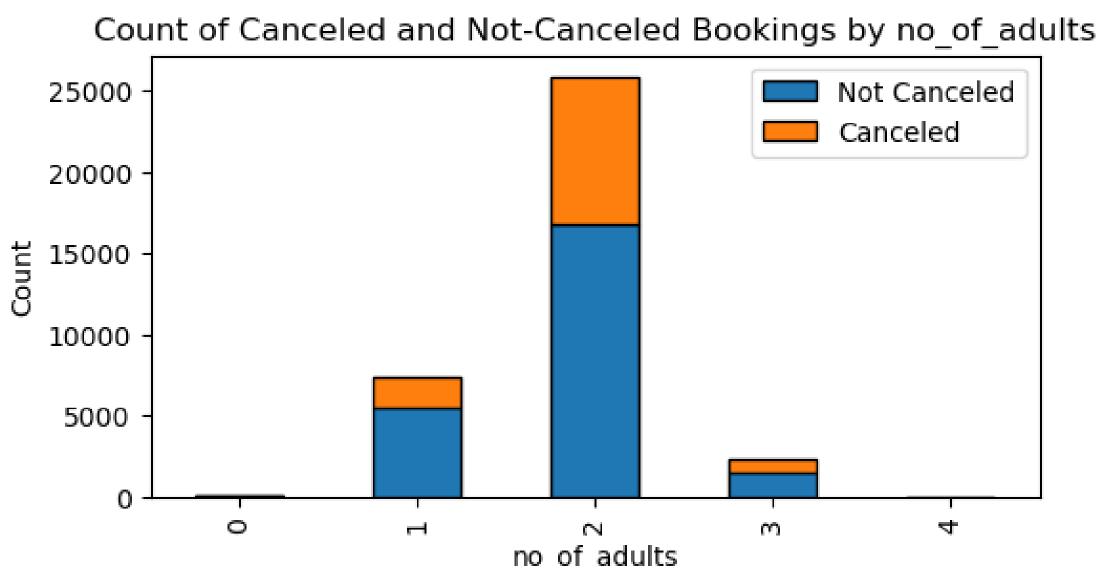
[55]:

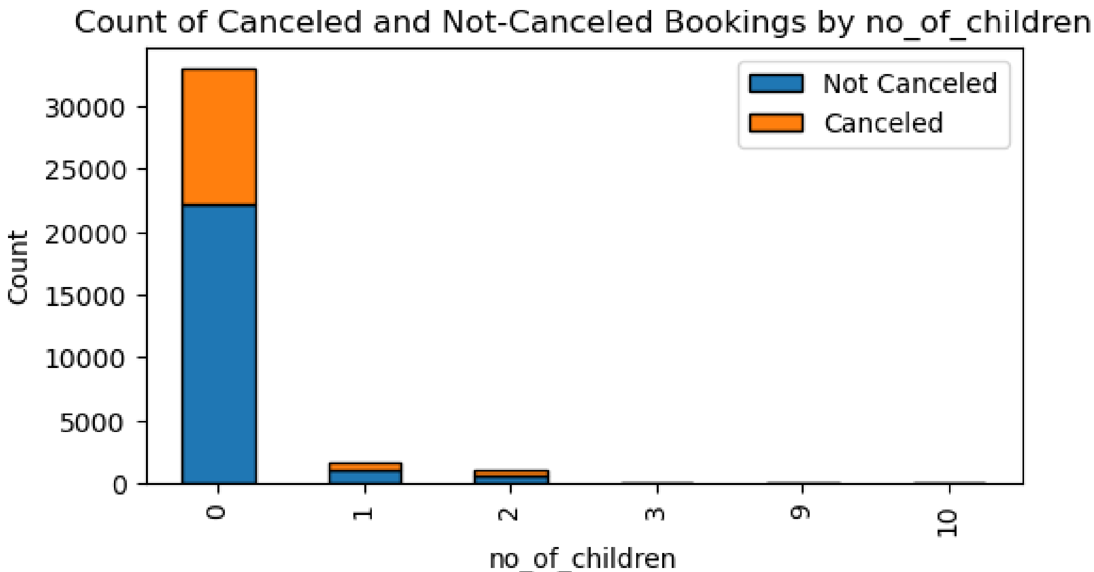
```python
# Create a function to plot stacked bars
def plot_stacked_bars(feature):
    cancellation_counts = hr.groupby([feature, 'booking_status_Canceled']).
 ↪size().unstack()
    cancellation_counts.plot(kind='bar', stacked=True, edgecolor='black',␣
 ↪figsize=(6, 3))
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.title(f'Count of Canceled and Not-Canceled Bookings by {feature}')
    plt.legend(['Not Canceled', 'Canceled'], loc='upper right')
    plt.show()

# Plot stacked bars for each profile feature
for feature in profile_features:
    plot_stacked_bars(feature)
```

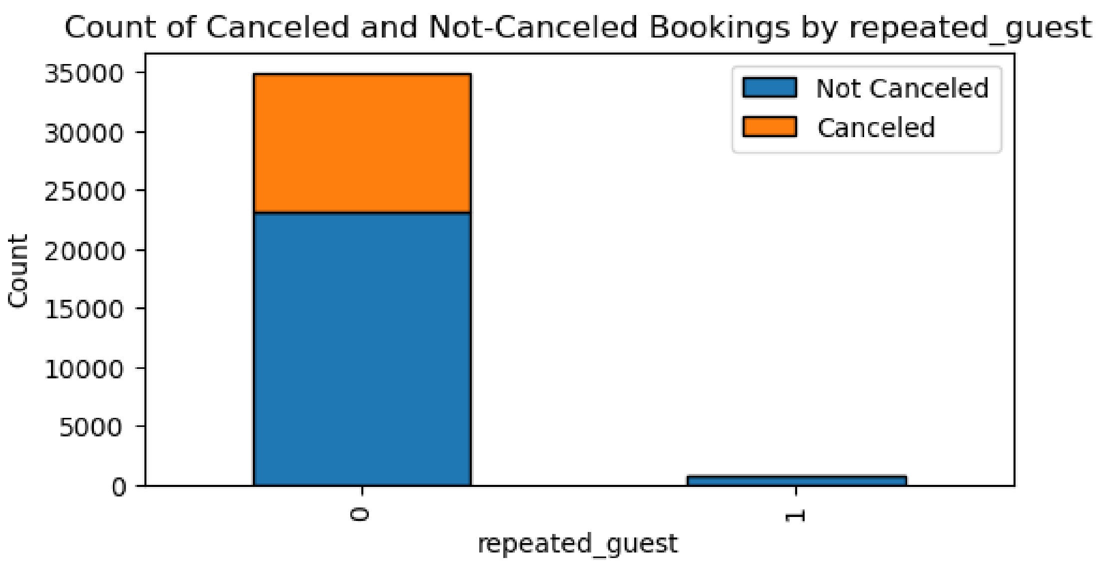

Count of Canceled and Not-Canceled Bookings by type_of_meal_plan
Not Canceled
Canceled
Count
0
5000
10000
15000
20000
25000
Meal Plan 1
Meal Plan 2
Meal Plan 3
Not Selected
type_of_meal_plan

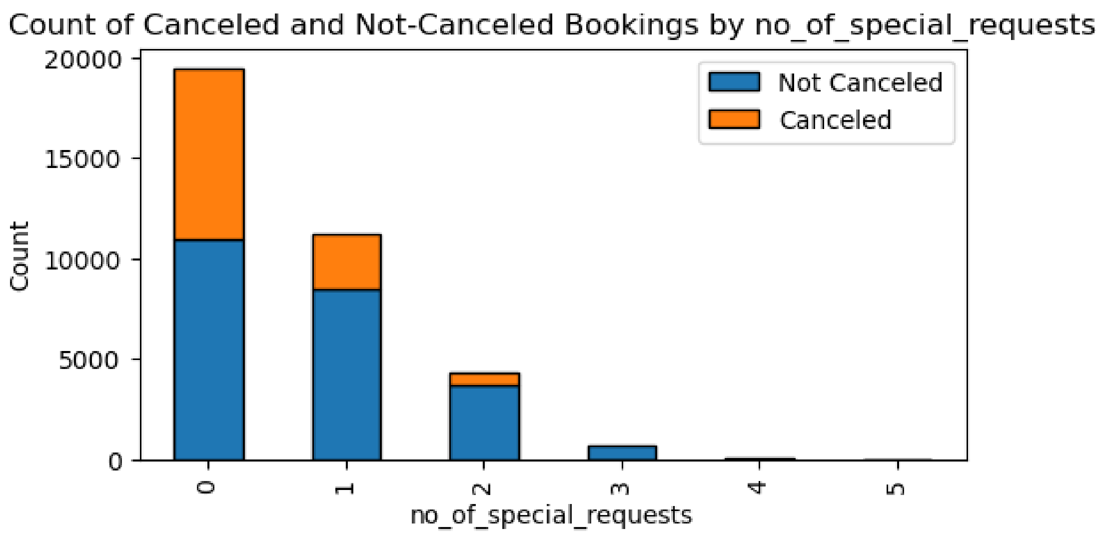

Count of Canceled and Not-Canceled Bookings by room_type_reserved
Not Canceled
Canceled
Count
0
5000
10000
15000
20000
25000
Room_Type 1
Room_Type 2
Room_Type 3
Room_Type 4
Room_Type 5
Room_Type 6
Room_Type 7
room_type_reserved

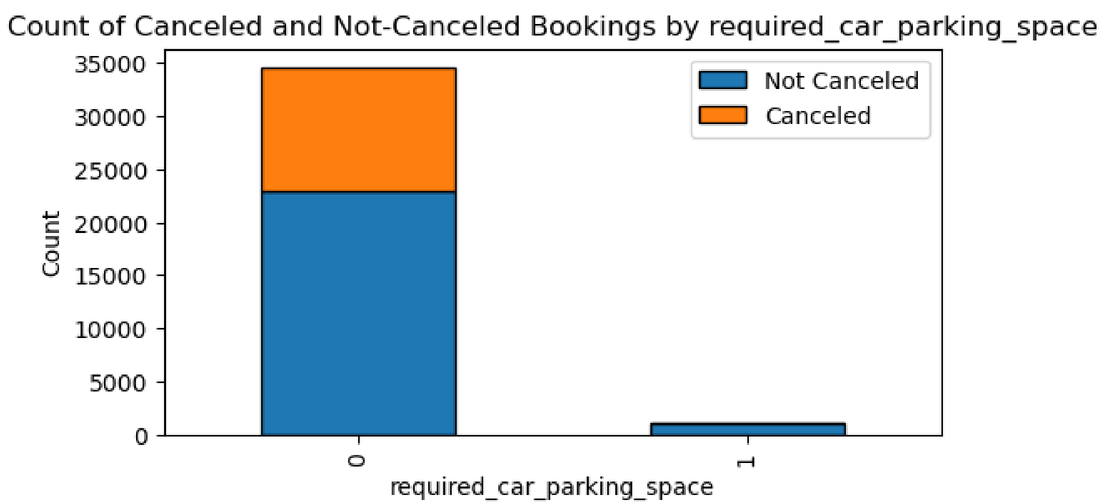

Count of Canceled and Not-Canceled Bookings by market_segment_type
Not Canceled
Canceled
Count
0
5000
10000
15000
20000
Aviation
Complementary
Corporate
Offline
Online
market_segment_type

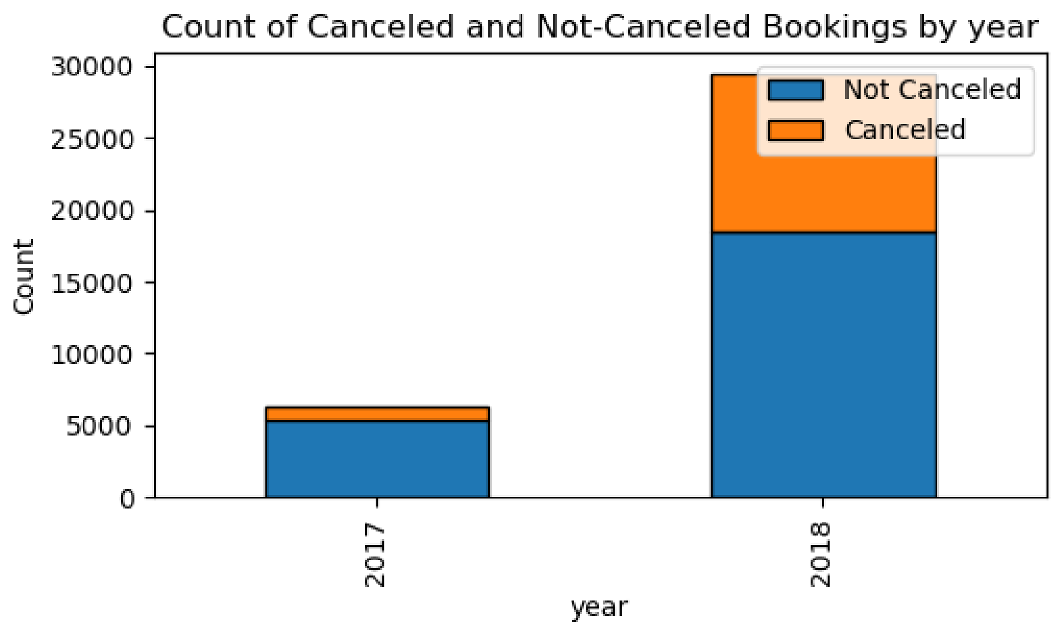

Count of Canceled and Not-Canceled Bookings by no_of_adults
Not Canceled
Canceled
Count
0
5000
10000
15000
20000
25000
0
1
2
3
4
no_of_adults


Count of Canceled and Not-Canceled Bookings by no_of_children
Not Canceled
Canceled
Count
0
5000
10000
15000
20000
25000
30000
0
1
2
3
9
10
no_of_children

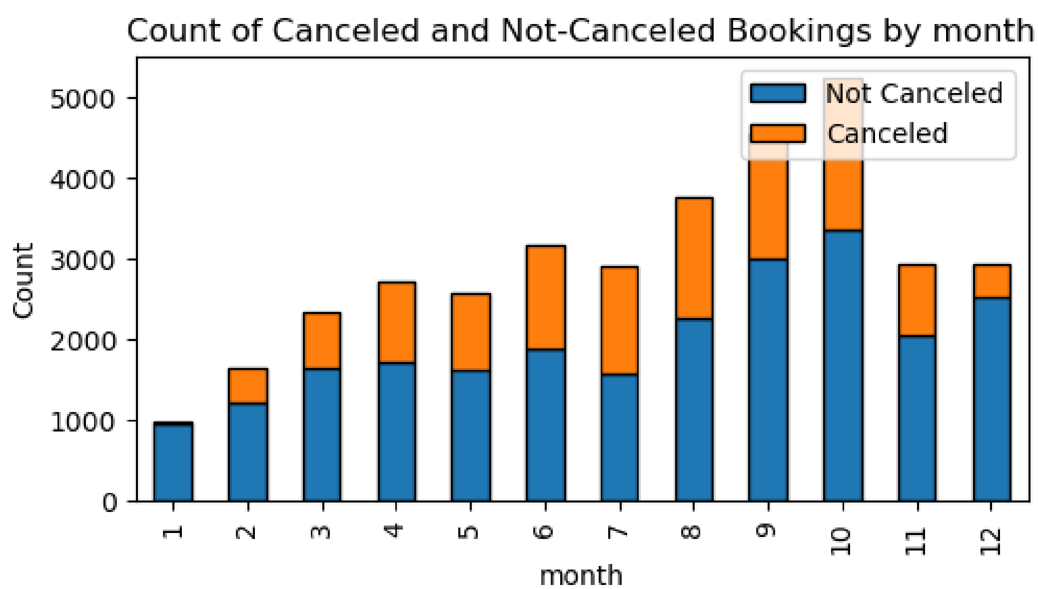

Count of Canceled and Not-Canceled Bookings by repeated_guest
Not Canceled
Canceled
Count
0
5000
10000
15000
20000
25000
30000
35000
0
1
repeated_guest


Count of Canceled and Not-Canceled Bookings by no_of_special_requests
Not Canceled
Canceled
Count
20000
15000
10000
5000
0
0
1
2
3
4
5
no_of_special_requests

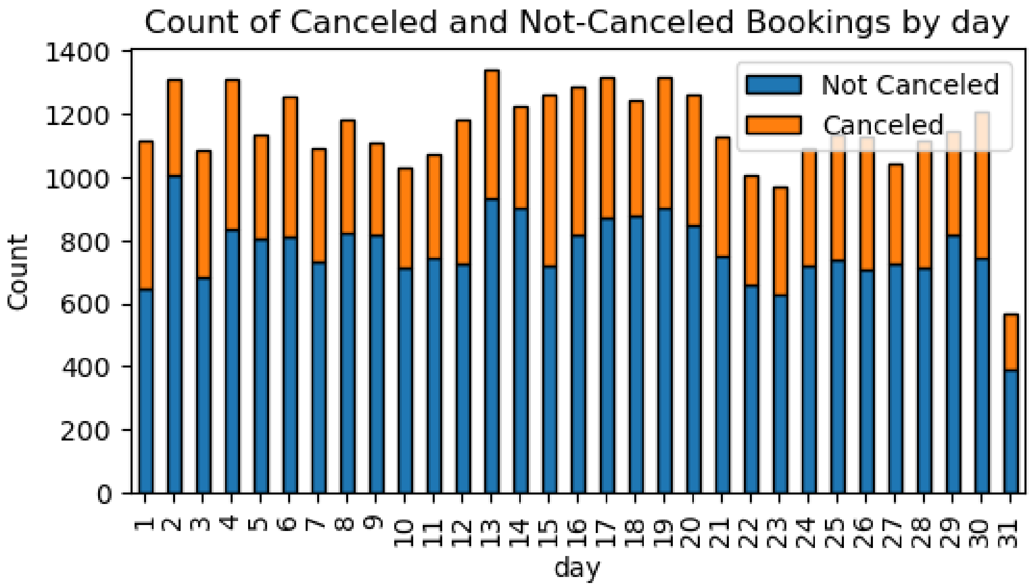

Count of Canceled and Not-Canceled Bookings by required_car_parking_space
Not Canceled
Canceled
Count
35000
30000
25000
20000
15000
10000
5000
0
0
1
required_car_parking_space

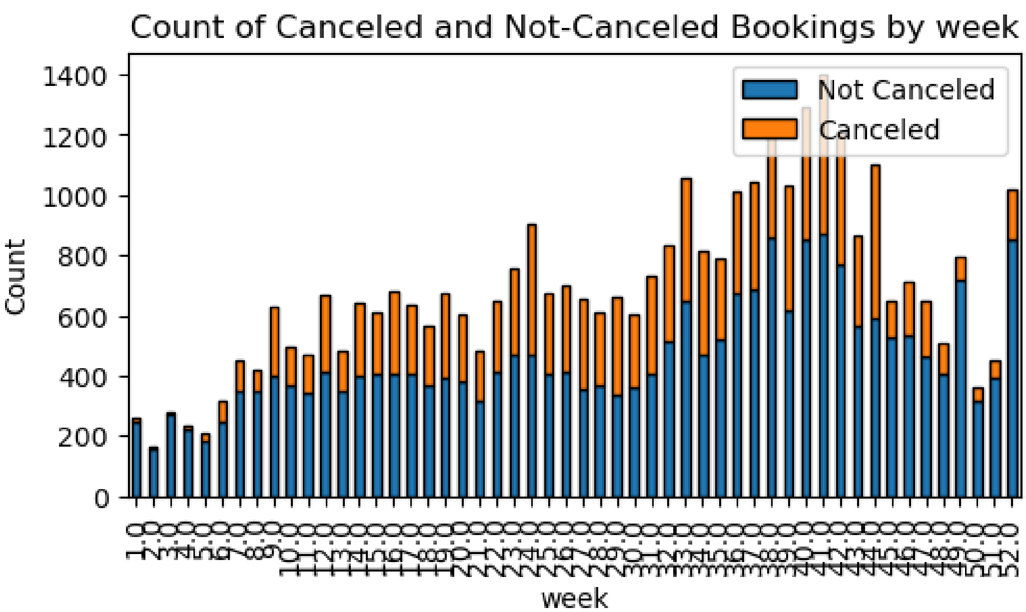


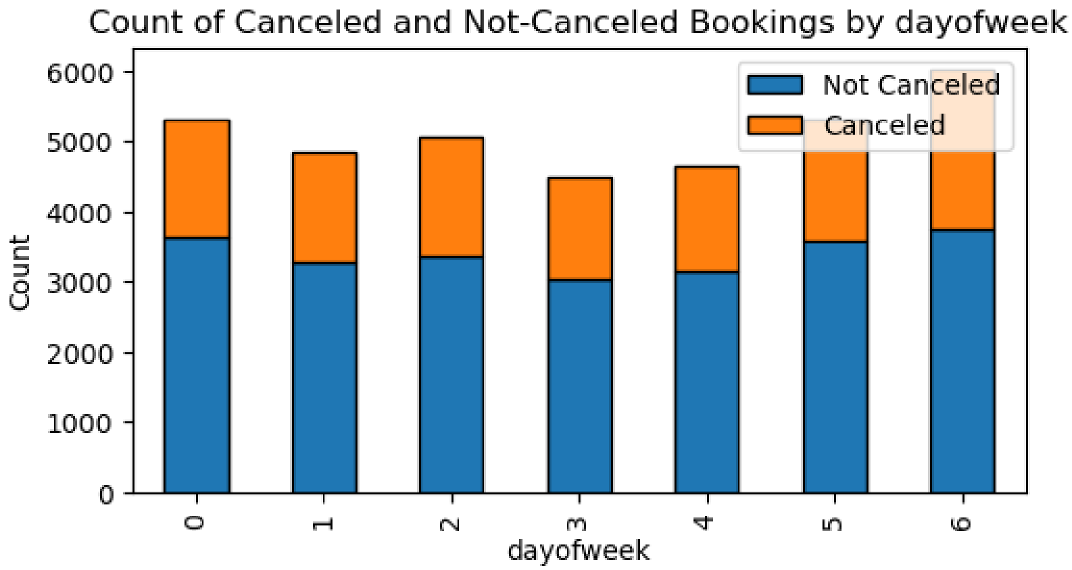
Count of Canceled and Not-Canceled Bookings by year
Not Canceled
Canceled
Count
0
5000
10000
15000
20000
25000
30000
2017
2018
year



Count of Canceled and Not-Canceled Bookings by month
Not Canceled
Canceled
Count
0
1000
2000
3000
4000
5000
1
2
3
4
5
6
7
8
9
10
11
12
month


Count of Canceled and Not-Canceled Bookings by day
Not Canceled
Canceled
Count
0
200
400
600
800
1000
1200
1400
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
day


Count of Canceled and Not-Canceled Bookings by week
Not Canceled
Canceled
Count
0
200
400
600
800
1000
1200
1400
week

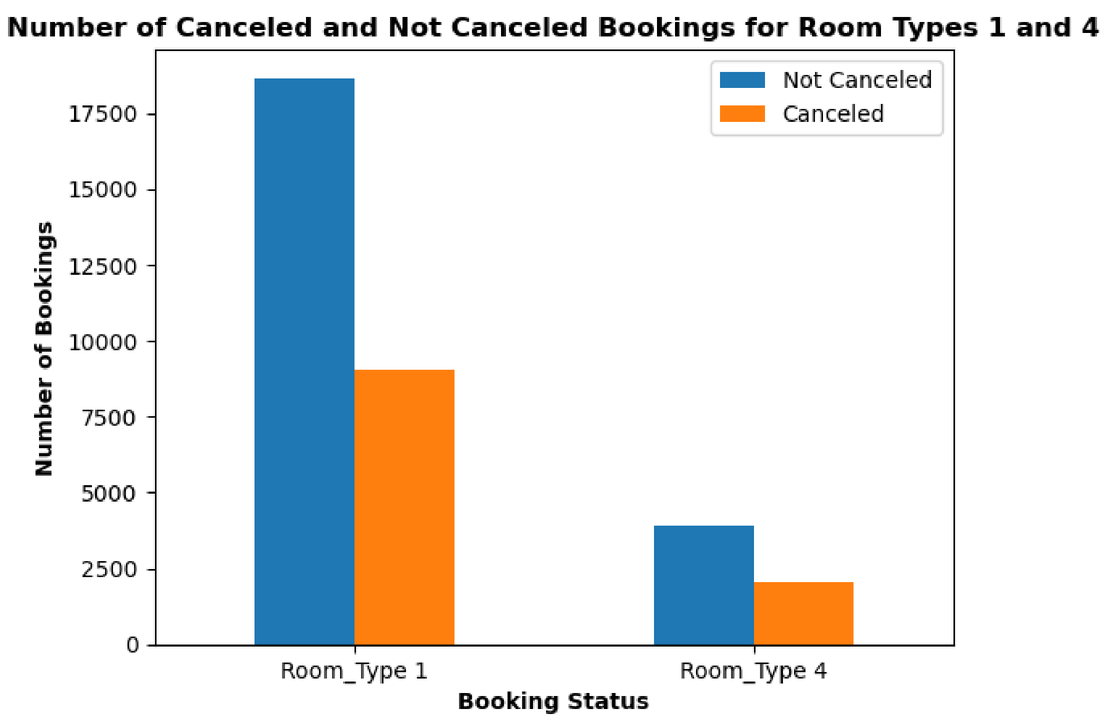


Insights:

- Longer trips have lower cancellations - Longer trips tend to be more planned
- As we get closer to the trip, cancellations increase
- Cancellations are the lowest for very cheap and very expensive rooms

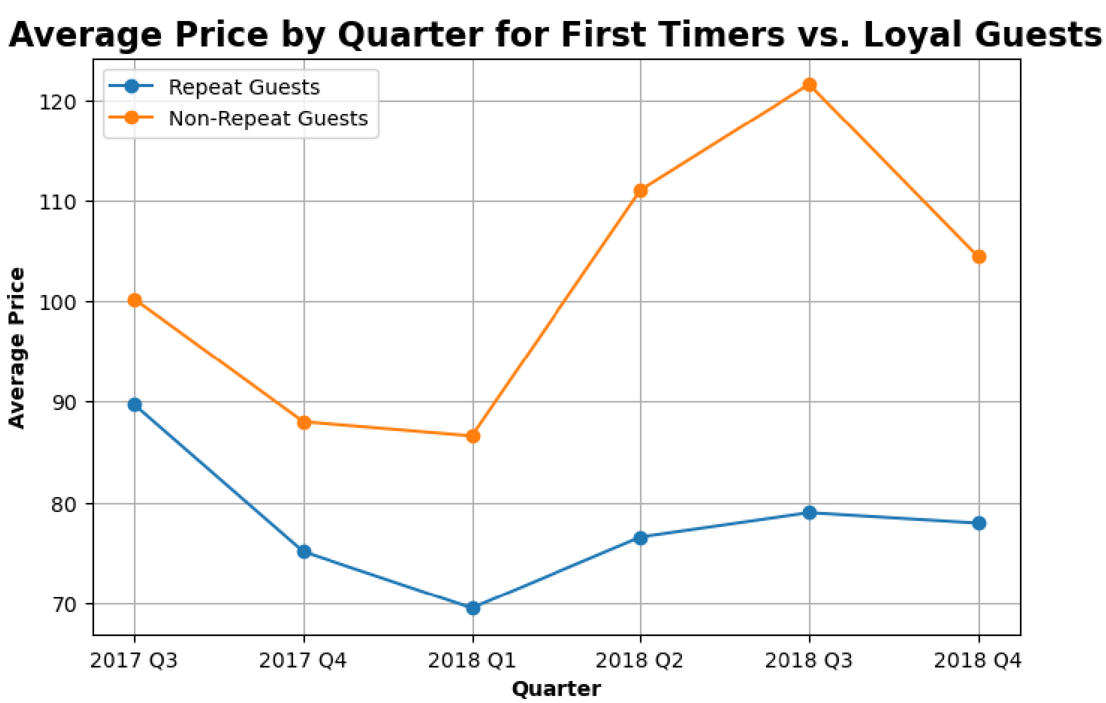
For the two room types with highest occupancy i.e. Room Types 1 and 4, find the number of cancelled vs non cancelled bookings

[56]:
```python
# Filter the DataFrame based on room type 1 and 4
filtered_df = hr[hr['room_type_reserved'].isin(['Room_Type 1', 'Room_Type 4'])]

# Group the data by 'room_type_reserved' and 'booking_status_Canceled', and␣
↪calculate the counts
cancellation_counts = filtered_df.groupby(['room_type_reserved',␣
↪'booking_status_Canceled']).size().unstack()

# Plot the data

ax = cancellation_counts.plot(kind='bar')

# Configure the plot
ax.set_xlabel('Booking Status', fontdict={'weight': 'bold'})
ax.set_ylabel('Number of Bookings', fontdict={'weight': 'bold'})
ax.set_title('Number of Canceled and Not Canceled Bookings for Room Types 1 and␣
↪4', fontdict={'weight': 'bold', 'size': 12})
ax.set_xticklabels(['Room_Type 1', 'Room_Type 4'], rotation=0)
ax.legend(['Not Canceled', 'Canceled'])

# Show the plot
plt.show()
```



How much are First-Timers paying for rooms as compared to Loyal (or repeat) customers?

[57]:
```
# Create a new column 'quarter_year' combining 'quarter' and 'year'
hr['quarter_year'] = hr['year'].astype(str) + ' Q' +hr['quarter'].astype(str)

# Separate repeat guests from non-repeat guests
repeat_guests = hr[hr['repeated_guest'] == 1]
non_repeat_guests = hr[hr['repeated_guest'] == 0]

repeat_guests_avg_price=repeat_guests.
 ↪groupby('quarter_year')['avg_price_per_room'].mean()
non_repeat_guests_avg_price=non_repeat_guests.
 ↪groupby('quarter_year')['avg_price_per_room'].mean()

# Plot the data
quarters = sorted(hr['quarter_year'].unique())  # Get unique quarters in
 ↪ascending order
plt.figure(figsize=(8, 5))
plt.plot(quarters, repeat_guests_avg_price, marker='o', label='Repeat Guests')
plt.plot(quarters, non_repeat_guests_avg_price, marker='o', label='Non-Repeat
 ↪Guests')
plt.xlabel('Quarter',fontdict={'weight': 'bold'})
plt.ylabel('Average Price',fontdict={'weight': 'bold'})
plt.title('Average Price by Quarter for First Timers vs. Loyal
 ↪Guests',fontdict={'weight': 'bold', 'size': 16})
plt.legend()
plt.grid(True)
plt.show()
```



- The hotel is charging first timers a lot more than repeat guests
- This discourages them and may contribute to the higher cancellations for them

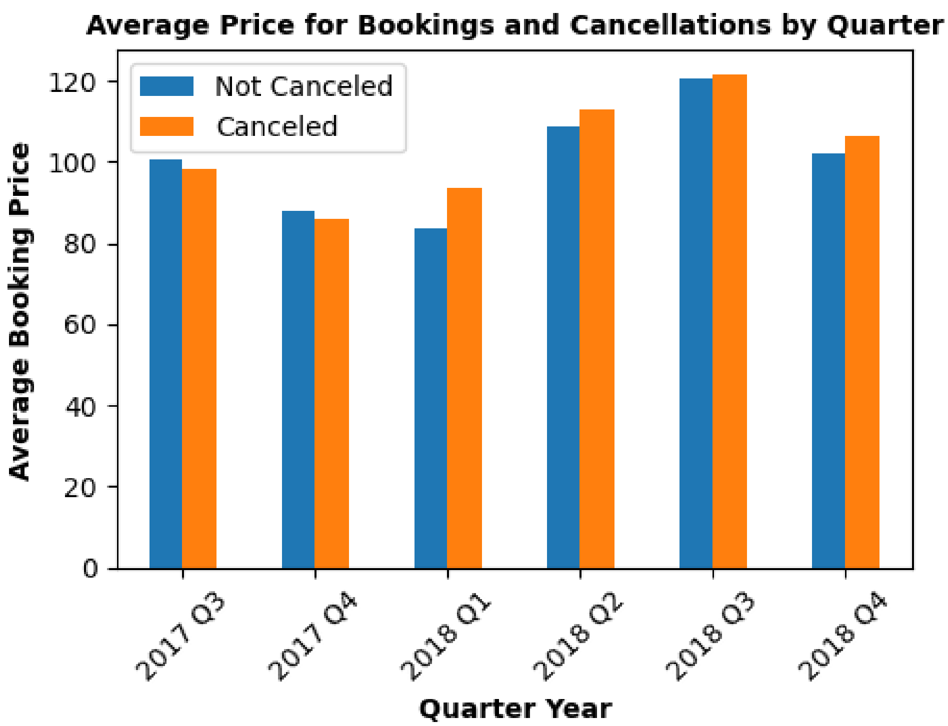
[58]:
```
hr.head()
```

[58]:
```
  Booking_ID  no_of_adults  no_of_children  no_of_weekend_nights  \
0   INN00001             2               0                     1
1   INN00002             2               0                     2
2   INN00003             1               0                     2
3   INN00004             2               0                     0
4   INN00005             2               0                     1

   no_of_week_nights type_of_meal_plan  required_car_parking_space  \
0                  2       Meal Plan 1                           0
1                  3      Not Selected                           0
2                  1       Meal Plan 1                           0
3                  2       Meal Plan 1                           0
4                  1      Not Selected                           0

  room_type_reserved  lead_time market_segment_type  …  \
0        Room_Type 1        224             Offline  …
1        Room_Type 1          5              Online  …
2        Room_Type 1          1              Online  …
3        Room_Type 1        211              Online  …
4        Room_Type 1         48              Online  …

   arrival_date_combined  booking_status_Canceled  year  month  day  week  \
0             2017-10-02                        0  2017     10    2  40.0
1             2018-11-06                        0  2018     11    6  45.0
2             2018-02-28                        1  2018      2   28   9.0
3             2018-05-20                        1  2018      5   20  20.0
4             2018-04-11                        1  2018      4   11  15.0

   dayofweek quarter  dayofyear  quarter_year
0          0       4        275       2017 Q4
1          1       4        310       2018 Q4
2          2       1         59       2018 Q1
3          6       2        140       2018 Q2
4          2       2        101       2018 Q2

[5 rows x 27 columns]
```

How has the average price of rooms evolved over time (split by bookings and cancellations)?

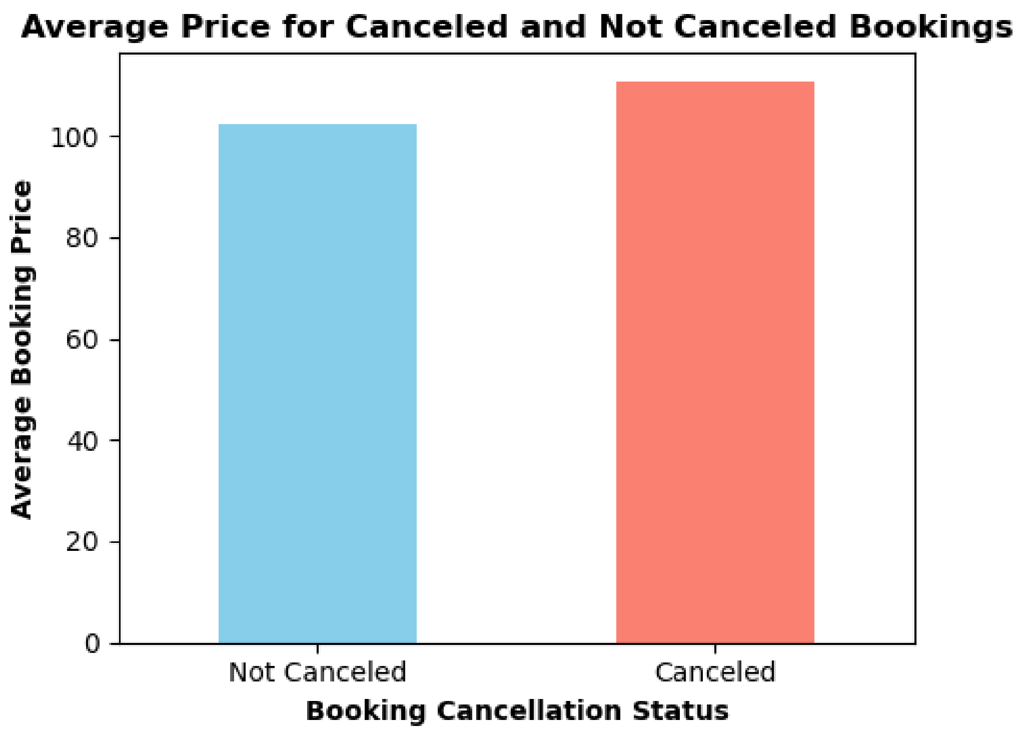
[59]:

```python
# Group the data by 'quarter_year' and 'booking_status_Canceled', and calculate␣
↪the average booking price
avg_price_by_quarter = hr.groupby(['quarter_year',␣
↪'booking_status_Canceled'])['avg_price_per_room'].mean().unstack()

# Plot the data
fig, ax = plt.subplots(figsize=(5, 4))  # Adjust the figure size here (width,␣
↪height)

avg_price_by_quarter.plot(kind='bar', ax=ax)

# Configure the plot
ax.set_xlabel('Quarter Year', fontdict={'weight': 'bold'})
ax.set_ylabel('Average Booking Price', fontdict={'weight': 'bold'})
ax.set_title('Average Price for Bookings and Cancellations by Quarter',␣
↪fontdict={'weight': 'bold', 'size': 10})
ax.set_xticklabels(avg_price_by_quarter.index, rotation=45)
ax.legend(['Not Canceled', 'Canceled'])

# Show the plot
plt.tight_layout()
plt.show()
```



Were cancelled bookings more expensive than non-cancelled bookings?

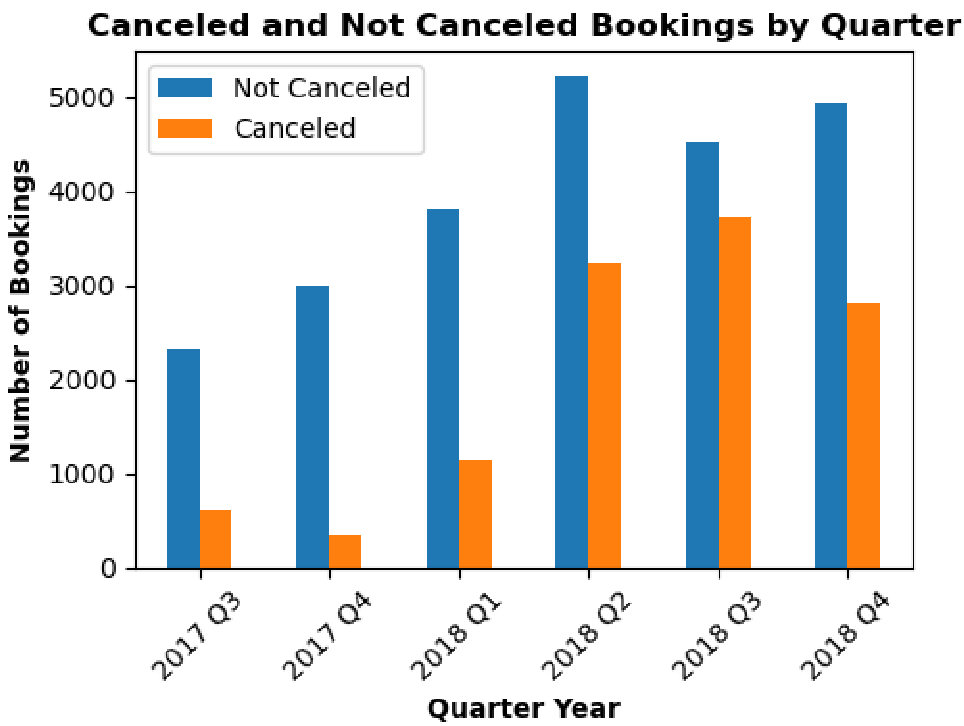
[60]:
```
# Group the data by 'booking_status_Canceled' and calculate the average booking␣
↪price
avg_price_by_canceled = hr.
↪groupby('booking_status_Canceled')['avg_price_per_room'].mean()
# Specify custom colors for the bars
colors = ['skyblue', 'salmon']
# Plot the data
fig, ax = plt.subplots(figsize=(5, 4))  # Adjust the figure size here (width,␣
↪height)

avg_price_by_canceled.plot(kind='bar', ax=ax, color=colors)

# Configure the plot
ax.set_xlabel('Booking Cancellation Status', fontdict={'weight': 'bold'})
ax.set_ylabel('Average Booking Price', fontdict={'weight': 'bold'})
ax.set_title('Average Price for Canceled and Not Canceled Bookings',␣
↪fontdict={'weight': 'bold', 'size': 12})
ax.set_xticklabels(['Not Canceled', 'Canceled'], rotation=0)

# Remove the line ax.legend(['Not Canceled', 'Canceled']) and let Matplotlib␣
↪generate the legend automatically

# Show the plot
plt.tight_layout()
plt.show()
```



How has the mix of cancelled vs non-cancelled bookings evolved over time?

[61]:
```
# Group the data by 'quarter_year' and 'booking_status_Canceled', and calculate␣
↪the counts
cancellation_counts_by_quarter = hr.groupby(['quarter_year',␣
↪'booking_status_Canceled']).size().unstack()

# Plot the data
fig, ax = plt.subplots(figsize=(5, 4))  # Adjust the figure size here (width,␣
↪height)

cancellation_counts_by_quarter.plot(kind='bar', ax=ax)

# Configure the plot
ax.set_xlabel('Quarter Year', fontdict={'weight': 'bold'})
ax.set_ylabel('Number of Bookings', fontdict={'weight': 'bold'})
ax.set_title('Canceled and Not Canceled Bookings by Quarter',␣
↪fontdict={'weight': 'bold', 'size': 12})
ax.set_xticklabels(cancellation_counts_by_quarter.index, rotation=45)
ax.legend(['Not Canceled', 'Canceled'], loc='upper left')

# Show the plot
plt.tight_layout()
```

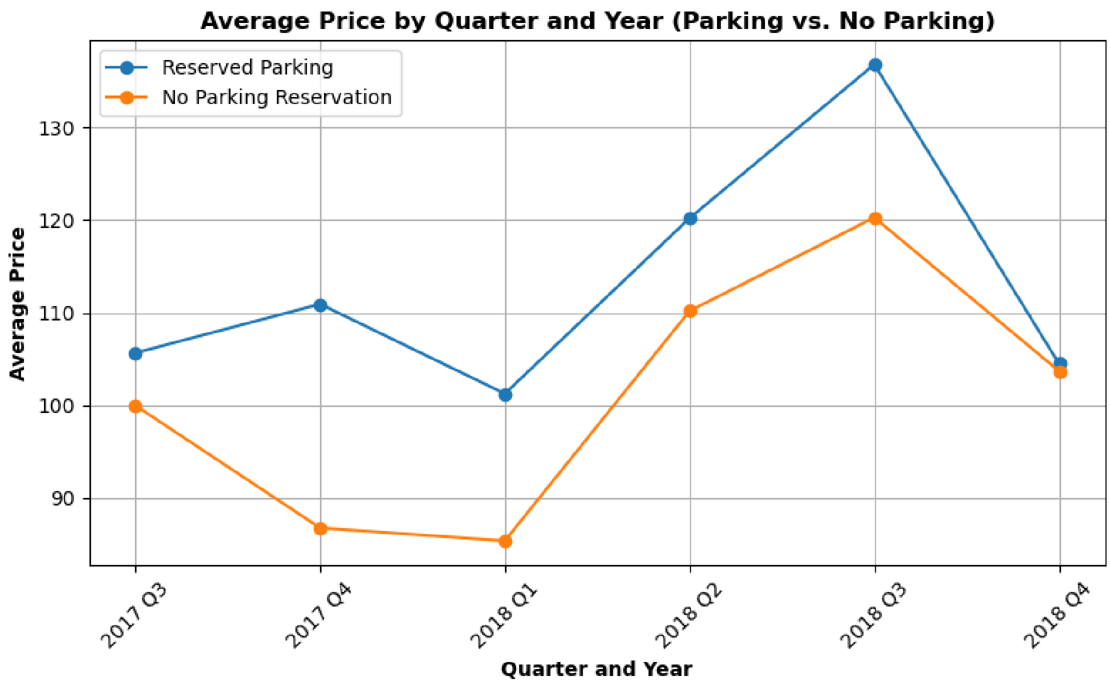
```
plt.show()
```



How much do average prices differ for people who ask for reserved parking vs those who do not?

[62]:
```
# Separate guests who reserved a parking spot from those who did not
reserved_parking = hr[hr['required_car_parking_space'] == 1]
no_parking_reservation = hr[hr['required_car_parking_space'] == 0]

# Group data by quarter_year and calculate average price for each group
reserved_parking_avg_price = reserved_parking.
  ↪groupby('quarter_year')['avg_price_per_room'].mean()
no_parking_reservation_avg_price = no_parking_reservation.
  ↪groupby('quarter_year')['avg_price_per_room'].mean()

# Plot the data
quarters_years = sorted(hr['quarter_year'].unique())  # Get unique␣
  ↪quarters_years in ascending order
plt.figure(figsize=(8, 5))
plt.plot(quarters_years, reserved_parking_avg_price, marker='o',␣
  ↪label='Reserved Parking')
plt.plot(quarters_years, no_parking_reservation_avg_price, marker='o',␣
  ↪label='No Parking Reservation')
plt.xlabel('Quarter and Year',fontdict={'weight': 'bold','size': 10})
```

```
plt.ylabel('Average Price',fontdict={'weight': 'bold'})
plt.title('Average Price by Quarter and Year (Parking vs. No␣
↪Parking)',fontdict={'weight': 'bold', 'size': 12})
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

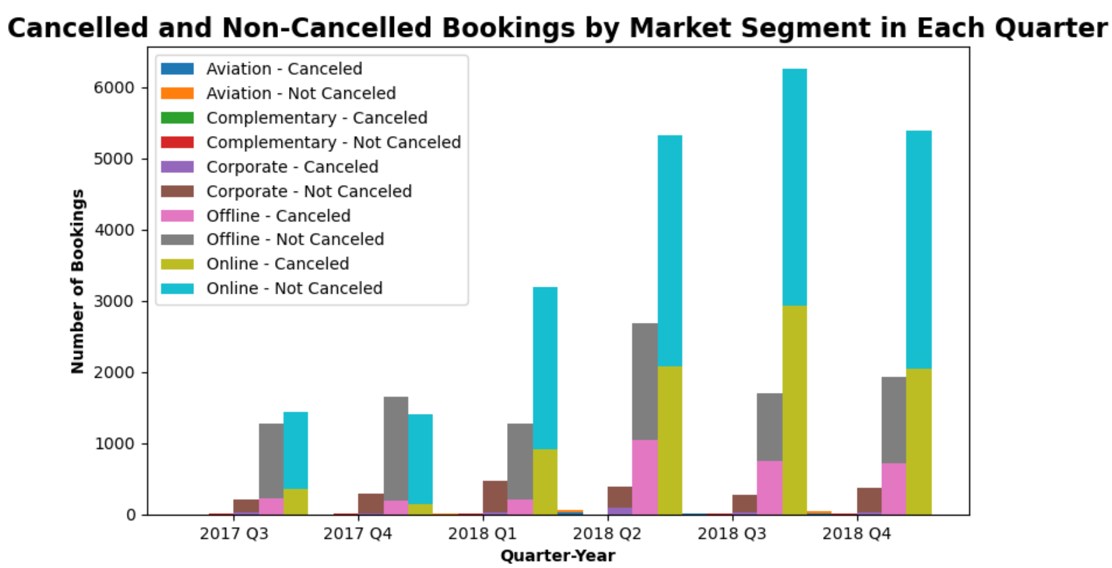


- People who want reserved parking pay more on an average as expected
- But the gap has reduced over time and in Q4 2028, they converged
- This may be a response to their consistently high cancellation rates from the hotel

How do cancelled and non-cancelled bookings vary for various market segment over time?

[63]:
```
# Group the data by quarter and market segment
grouped_data = hr.groupby(['quarter_year', 'market_segment_type'])

# Count the number of canceled and not-canceled bookings for each group
cancelled_counts = grouped_data['booking_status_Canceled'].sum()
not_cancelled_counts = grouped_data['booking_status_Canceled'].count() -␣
↪cancelled_counts

# Create a bar chart
quarters = sorted(hr['quarter_year'].unique())
```

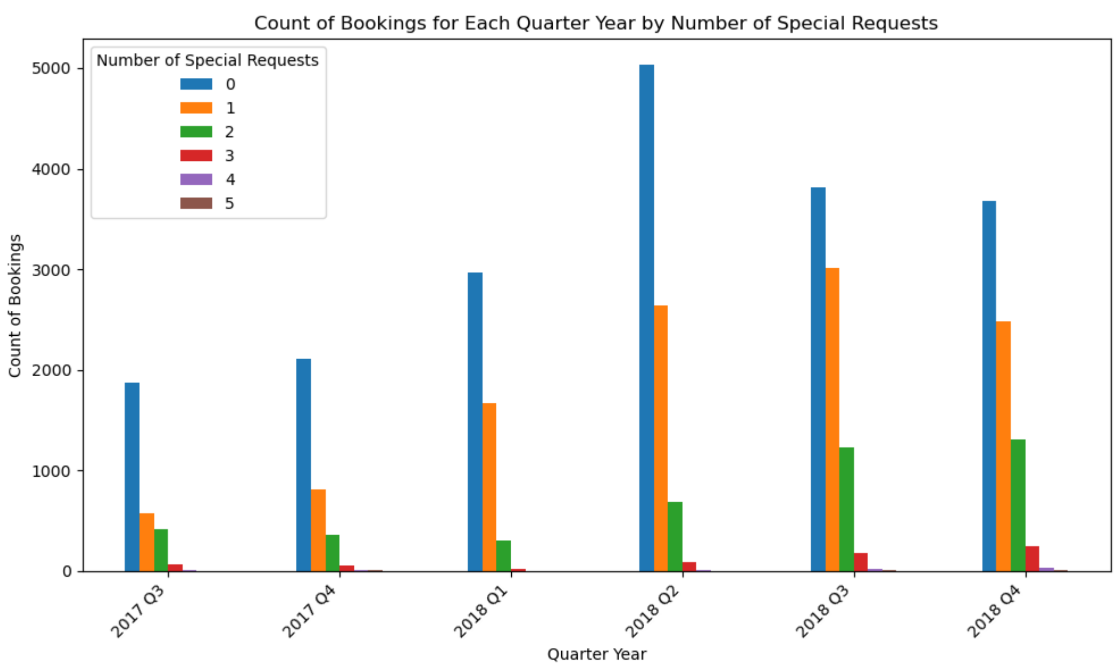
```
market_segments = sorted(hr['market_segment_type'].unique())

width = 0.2  # Width of each bar
x = range(len(quarters))

# Plot the bars
fig, ax = plt.subplots(figsize=(8, 5))
for i, market_segment in enumerate(market_segments):
    canceled_vals = [cancelled_counts.get((quarter, market_segment), 0) for␣
↪quarter in quarters]
    not_cancelled_vals = [not_cancelled_counts.get((quarter, market_segment),␣
↪0) for quarter in quarters]

    ax.bar([pos + i * width for pos in x], canceled_vals, width,␣
↪label=f"{market_segment} - Canceled")
    ax.bar([pos + i * width for pos in x], not_cancelled_vals, width,␣
↪bottom=canceled_vals, label=f"{market_segment} - Not Canceled")

ax.set_xticks([pos + 1.5 * width for pos in x])
ax.set_xticklabels(quarters)
ax.set_xlabel("Quarter-Year",fontdict={'weight': 'bold'})
ax.set_ylabel("Number of Bookings",fontdict={'weight': 'bold'})
ax.set_title("Cancelled and Non-Cancelled Bookings by Market Segment in Each␣
↪Quarter",fontdict={'weight': 'bold', 'size': 16})
ax.legend()

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



- Online bookings have skyrocketed over time
- But close to half of them get cancelled
- Knowing this hotels can comfortably overbook rooms in order to reduce the hit from last minute cancellations

Is there a trend in the number of special requests over time?

[64]:
```
# Group by 'quarter_year' and 'no_of_special_requests', then calculate the
↪count of bookings for each group
booking_counts = hr.groupby(['no_of_special_requests', 'quarter_year']).size().
↪reset_index(name='count')

# Pivot the data to make 'no_of_special_requests' values as columns and
↪'quarter_year' as rows
pivot_hr = booking_counts.pivot(index='quarter_year',
↪columns='no_of_special_requests', values='count')

# Plot the data using a bar plot
ax = pivot_hr.plot(kind='bar', figsize=(10, 6))
ax.set_xlabel('Quarter Year')
ax.set_ylabel('Count of Bookings')
ax.set_title('Count of Bookings for Each Quarter Year by Number of Special
↪Requests')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Number of Special Requests')
plt.tight_layout()
plt.show()
```



We want to recommend a cancellation charge framework on the basis of lead time and find the projected amount that we can recoup from cancellations. For this analysis, we focus only on Room Type 1 as it has the highest occupancy rates. The steps are as follows :

- We split the lead time values into quartiles and the cancellation charge increases as we move closer to the check-in date
- When the lead time is in the fourth quartile, we charge 10% of booking amount for cancellation. For the third, second and first quartiles we charge 15%, 20% and 30% respectively

[67]:
```
# Step 1: Calculate quartiles for lead time
lead_time_quartiles = hr['lead_time'].quantile([0.25, 0.5, 0.75, 1.0])

def get_lead_time_quartile(lead_time):
    if lead_time >= lead_time_quartiles[0.75]:
        return 'Q4'
    elif lead_time >= lead_time_quartiles[0.5]:
        return 'Q3'
    elif lead_time >= lead_time_quartiles[0.25]:
        return 'Q2'
    else:
        return 'Q1'

hr['lead_time_quartile'] = hr['lead_time'].apply(get_lead_time_quartile)

# Step 2: Filter data for Room Type 1 and cancelled
room_type_1_data = hr[(hr['room_type_reserved'] == 'Room_Type 1')␣
↪&(hr['booking_status_Canceled'] == 1)]

# Step 3: Create the cancellation_charge column
def calculate_cancellation_charge(row):
    lead_time = row['lead_time']
    avg_price_per_room = row['avg_price_per_room']
    lead_time_quartile=row['lead_time_quartile']
    if lead_time_quartile=='Q4':
        return 0.10 * avg_price_per_room
    elif lead_time_quartile=='Q3':
        return 0.15 * avg_price_per_room
    elif lead_time_quartile=='Q2':
        return 0.20 * avg_price_per_room
    else:
        return 0.30 * avg_price_per_room

room_type_1_data['cancellation_charge'] = room_type_1_data.
↪apply(calculate_cancellation_charge, axis=1)

# Print the updated DataFrame
```

```python
room_type_1_data.head(20)
```

```
/var/folders/vl/q57w49ys18v2dbmv_5lg2wth0000gn/T/ipykernel_96583/212960769.py:33
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  room_type_1_data['cancellation_charge'] =
room_type_1_data.apply(calculate_cancellation_charge, axis=1)
```

[67]:

```
    Booking_ID  no_of_adults  no_of_children  no_of_weekend_nights  \
2     INN00003             1               0                     2
3     INN00004             2               0                     0
4     INN00005             2               0                     1
5     INN00006             2               0                     0
12    INN00013             2               0                     2
13    INN00014             1               0                     2
15    INN00016             2               0                     0
18    INN00019             2               0                     2
20    INN00021             2               0                     2
28    INN00029             1               0                     1
36    INN00037             1               0                     2
38    INN00039             2               0                     2
43    INN00044             2               0                     1
51    INN00052             2               0                     2
73    INN00074             2               0                     0
87    INN00088             1               0                     0
94    INN00095             2               0                     2
96    INN00097             2               0                     0
110   INN00111             2               0                     0
118   INN00119             2               0                     0

     no_of_week_nights type_of_meal_plan  required_car_parking_space  \
2                    1       Meal Plan 1                           0
3                    2       Meal Plan 1                           0
4                    1      Not Selected                           0
5                    2       Meal Plan 2                           0
12                   1      Not Selected                           0
13                   0       Meal Plan 1                           0
15                   2       Meal Plan 2                           0
18                   2       Meal Plan 1                           0
20                   2       Meal Plan 1                           0
28                   2       Meal Plan 1                           0
36                   1       Meal Plan 1                           0
38                   3      Not Selected                           0
```

```
43                   1      Not Selected                           0
51                   2       Meal Plan 2                           0
73                   2       Meal Plan 1                           0
87                   2       Meal Plan 1                           0
94                   5       Meal Plan 1                           0
96                   2       Meal Plan 2                           0
110                  2       Meal Plan 1                           0
118                  1       Meal Plan 1                           0

    room_type_reserved  lead_time market_segment_type  …  year  month  day  \
2          Room_Type 1          1              Online  …  2018      2   28
3          Room_Type 1        211              Online  …  2018      5   20
4          Room_Type 1         48              Online  …  2018      4   11
5          Room_Type 1        346              Online  …  2018      9   13
12         Room_Type 1         30              Online  …  2018     11   26
13         Room_Type 1         95              Online  …  2018     11   20
15         Room_Type 1        256              Online  …  2018      6   15
18         Room_Type 1         99              Online  …  2017     10   30
20         Room_Type 1         99              Online  …  2017     10   30
28         Room_Type 1         37              Online  …  2017     11    6
36         Room_Type 1         34              Online  …  2018      6   19
38         Room_Type 1        247              Online  …  2018     11   19
43         Room_Type 1         41              Online  …  2018      6   27
51         Room_Type 1        169              Online  …  2018      4   22
73         Room_Type 1        177              Online  …  2018      6    3
87         Room_Type 1        188              Online  …  2018      6   15
94         Room_Type 1        171              Online  …  2018      8   30
96         Room_Type 1        320              Online  …  2018      8   18
110        Room_Type 1        182              Online  …  2018      9   30
118        Room_Type 1        443              Online  …  2018      4   29

     week  dayofweek  quarter  dayofyear quarter_year  lead_time_quartile  \
2     9.0          2        1         59      2018 Q1                  Q1
3    20.0          6        2        140      2018 Q2                  Q4
4    15.0          2        2        101      2018 Q2                  Q2
5    37.0          3        3        256      2018 Q3                  Q4
12   48.0          0        4        330      2018 Q4                  Q2
13   47.0          1        4        324      2018 Q4                  Q3
15   24.0          4        2        166      2018 Q2                  Q4
18   44.0          0        4        303      2017 Q4                  Q3
20   44.0          0        4        303      2017 Q4                  Q3
28   45.0          0        4        310      2017 Q4                  Q2
36   25.0          1        2        170      2018 Q2                  Q2
38   47.0          0        4        323      2018 Q4                  Q4
43   26.0          2        2        178      2018 Q2                  Q2
51   16.0          6        2        112      2018 Q2                  Q4
73   22.0          6        2        154      2018 Q2                  Q4
```

```
87   24.0          4        2        166      2018 Q2                    Q4
94   35.0          3        3        242      2018 Q3                    Q4
96   33.0          5        3        230      2018 Q3                    Q4
110  39.0          6        3        273      2018 Q3                    Q4
118  17.0          6        2        119      2018 Q2                    Q4

     cancellation_charge
2                 18.000
3                 10.000
4                 18.900
5                 11.500
12                17.600
13                13.500
15                11.500
18                 9.750
20                 9.750
28                 7.466
36                16.200
38                 6.375
43                19.620
51                10.600
73                10.000
87                13.000
94                11.159
96                11.500
110               11.790
118                6.500

[20 rows x 29 columns]
```

[73]:
```python
result = room_type_1_data.groupby('lead_time_quartile').agg(
    total_cancellation_charge=pd.NamedAgg(column='cancellation_charge',␣
 ↪aggfunc='sum'),
    total_cancellations=pd.NamedAgg(column='booking_status_Canceled',␣
 ↪aggfunc='sum')
)

print(result)
```

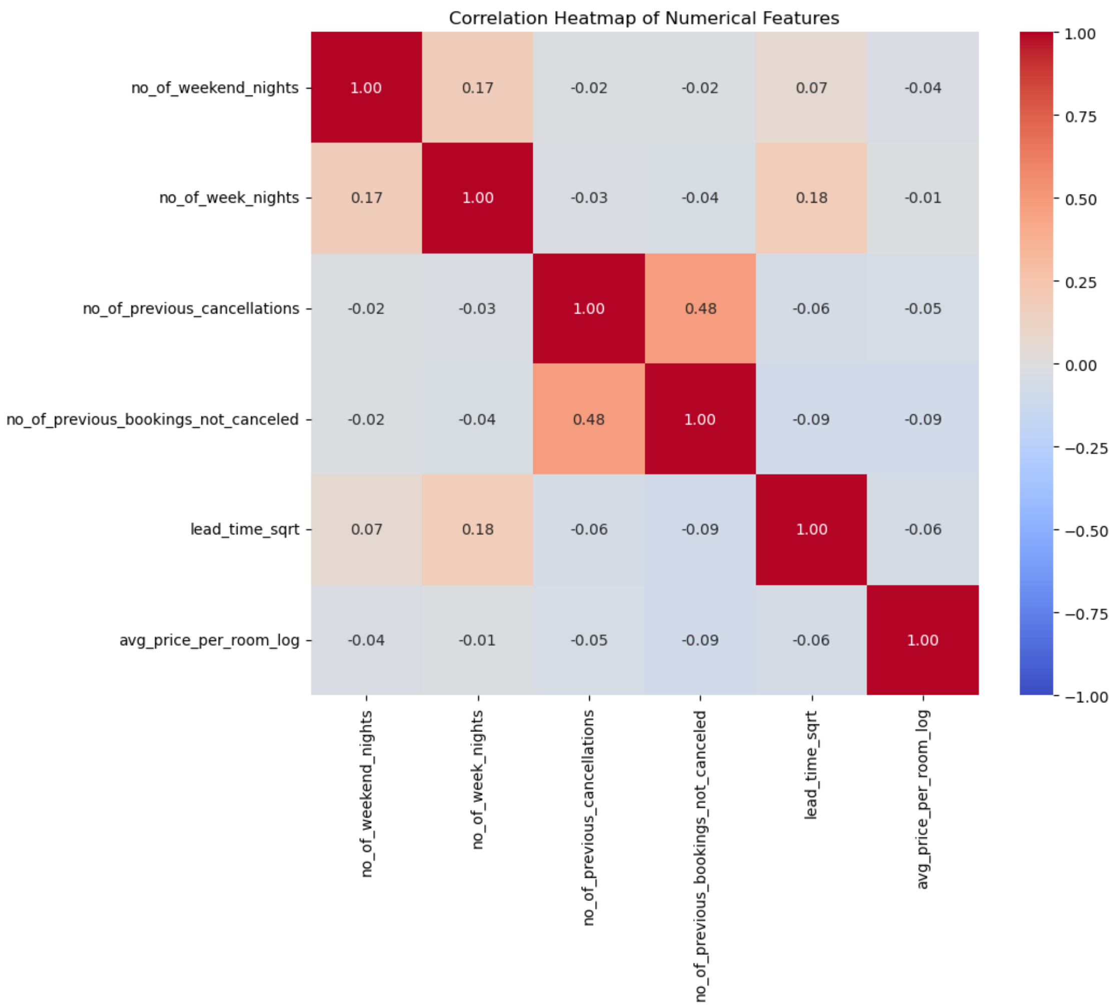
```
                    total_cancellation_charge  total_cancellations
lead_time_quartile
Q1                                  24003.297                  758
Q2                                  28173.876                 1329
Q3                                  31248.852                 2147
Q4                                  49298.945                 4826
```

- By implementing the cancellation charge scheme outlined earlier for Room Type 1, we can potentially recover around 133k euros, which accounts for about 15% of the total revenue lost

due to cancellations.
- The cancellation charges can be adjusted incrementally to strike a balance. We need to fine-tune the charges until the point where it doesn't negatively impact the number of bookings.

Correlation Matrix

[74]:
```
# Define the numerical features to analyze
numerical_features_new = ['no_of_weekend_nights', 'no_of_week_nights',␣
↪'no_of_previous_cancellations',
                          'no_of_previous_bookings_not_canceled',␣
↪'lead_time_sqrt', 'avg_price_per_room_log']

# Assuming you have a DataFrame 'hr' containing the booking data
correlation_matrix = hr[numerical_features_new].corr()

# Create the correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
↪vmin=-1, vmax=1, center=0)
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```



5 Model

[54]:
```
# Prep data further for model
# One hot encoding the other categorical variables
hr_encoded = pd.get_dummies(hr, columns=['type_of_meal_plan',␣
↪'room_type_reserved','market_segment_type'], drop_first=True)

# Drop the original 'arrival_date_combined' column
hr_encoded.
↪drop(columns=['arrival_date_combined','avg_price_per_room','lead_time','quarter_year'],␣
↪inplace=True)
hr_encoded.columns
```

[54]:
```
Index(['Booking_ID', 'no_of_adults', 'no_of_children', 'no_of_weekend_nights',
       'no_of_week_nights', 'required_car_parking_space', 'repeated_guest',
```

```
       'no_of_previous_cancellations', 'no_of_previous_bookings_not_canceled',
       'no_of_special_requests', 'lead_time_sqrt', 'avg_price_per_room_log',
       'no_of_previous_bookings_not_canceled_log', 'booking_status_Canceled',
       'year', 'month', 'day', 'week', 'dayofweek', 'quarter', 'dayofyear',
       'lead_time_quartile', 'type_of_meal_plan_Meal Plan 2',
       'type_of_meal_plan_Meal Plan 3', 'type_of_meal_plan_Not Selected',
       'room_type_reserved_Room_Type 2', 'room_type_reserved_Room_Type 3',
       'room_type_reserved_Room_Type 4', 'room_type_reserved_Room_Type 5',
       'room_type_reserved_Room_Type 6', 'room_type_reserved_Room_Type 7',
       'market_segment_type_Complementary', 'market_segment_type_Corporate',
       'market_segment_type_Offline', 'market_segment_type_Online'],
      dtype='object')
```

[59]:
```
# Separate the target variable from the features
X = hr_encoded.
 ↪drop(columns=['booking_status_Canceled','Booking_ID','lead_time_quartile'])
y = hr_encoded['booking_status_Canceled']

# Split the data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
X_train
```

[59]:
```
       no_of_adults  no_of_children  no_of_weekend_nights  no_of_week_nights  \
18348             2               0                     2                  4
5193              2               0                     0                  1
22960             1               0                     1                  2
7032              2               0                     2                  1
24063             1               0                     0                  1
…               …             …                   …                 …
17110             1               0                     0                  1
6352              2               1                     2                  4
11463             2               0                     1                  2
874               2               0                     0                  2
16042             2               0                     0                  1

       required_car_parking_space  repeated_guest  \
18348                           0               0
5193                            0               0
22960                           0               0
7032                            0               0
24063                           0               0
…                             …             …
17110                           0               0
6352                            0               0
11463                           0               0
874                             0               0
```

```
16042                          0              0

       no_of_previous_cancellations  no_of_previous_bookings_not_canceled  \
18348                             0                                     0
5193                              0                                     0
22960                             0                                     0
7032                              0                                     0
24063                             0                                     0
…                               …                                   …
17110                             0                                     0
6352                              0                                     0
11463                             0                                     0
874                               0                                     0
16042                             0                                     0

       no_of_special_requests  lead_time_sqrt  …  \
18348                       0        9.110434  …
5193                        0       13.490738  …
22960                       1        2.645751  …
7032                        2        3.872983  …
24063                       0        1.000000  …
…                         …             …  …
17110                       0        7.000000  …
6352                        1        9.797959  …
11463                       1        9.165151  …
874                         0        8.366600  …
16042                       0       10.295630  …

       room_type_reserved_Room_Type 2  room_type_reserved_Room_Type 3  \
18348                               0                               0
5193                                0                               0
22960                               0                               0
7032                                0                               0
24063                               0                               0
…                                 …                             …
17110                               0                               0
6352                                0                               0
11463                               0                               0
874                                 0                               0
16042                               0                               0

       room_type_reserved_Room_Type 4  room_type_reserved_Room_Type 5  \
18348                               0                               0
5193                                0                               1
22960                               0                               0
7032                                0                               0
24063                               0                               0
```

```
…                                …                             …
17110                             0                             0
6352                              1                             0
11463                             0                             0
874                               0                             0
16042                             0                             0

       room_type_reserved_Room_Type 6  room_type_reserved_Room_Type 7  \
18348                             0                             0
5193                              0                             0
22960                             0                             0
7032                              0                             0
24063                             0                             0
…                                …                             …
17110                             0                             0
6352                              0                             0
11463                             0                             0
874                               0                             0
16042                             0                             0

       market_segment_type_Complementary  market_segment_type_Corporate  \
18348                                0                             0
5193                                 0                             0
22960                                0                             0
7032                                 0                             0
24063                                0                             0
…                                   …                             …
17110                                0                             0
6352                                 0                             0
11463                                0                             0
874                                  0                             0
16042                                0                             0

       market_segment_type_Offline  market_segment_type_Online
18348                          1                           0
5193                           0                           1
22960                          0                           1
7032                           0                           1
24063                          0                           1
…                             …                           …
17110                          1                           0
6352                           0                           1
11463                          0                           1
874                            1                           0
16042                          1                           0

[28554 rows x 32 columns]
```

5.1 Logistic Regression

RFE, GridSearchCV, Model and Evaluation Metrics (Accuracy, Precision and Recall)

[60]:
```
# Recursive Feature Elimination with Logistic Regression
logistic_model = LogisticRegression()  # Choose the model you want to use for␣
↪RFE
rfe_logistic = RFE(logistic_model, n_features_to_select=10)  # Choose the␣
↪desired number of features
fit_logistic = rfe_logistic.fit(X_train, y_train)

# Selected features
selected_features_logistic = X_train.columns[fit_logistic.support_]
print("Selected Features (Logistic Regression):", selected_features_logistic)

# GridSearchCV to find the best hyperparameters for Logistic Regression
param_grid_logistic = {
    'penalty': ['l1', 'l2'],
    'C': [0.01, 0.1, 1.0, 10.0]
}

logistic_model = LogisticRegression()
grid_search_logistic = GridSearchCV(logistic_model, param_grid_logistic, cv=5,␣
↪n_jobs=-1)
grid_search_logistic.fit(X_train[selected_features_logistic], y_train)

# Best hyperparameters for Logistic Regression
best_params_logistic = grid_search_logistic.best_params_
print("Best Hyperparameters (Logistic Regression):", best_params_logistic)

# Train the Logistic Regression model with optimized hyperparameters
best_model_logistic = LogisticRegression(**best_params_logistic)
best_model_logistic.fit(X_train[selected_features_logistic], y_train)

# Evaluate the model on the test set
y_pred_logistic = best_model_logistic.
↪predict(X_test[selected_features_logistic])

# Calculate accuracy, precision and recall
accuracy_logistic = best_model_logistic.
↪score(X_test[selected_features_logistic], y_test)
print("Accuracy on Test Set (Logistic Regression):", accuracy_logistic)

precision_logistic = precision_score(y_test, y_pred_logistic)
print("Precision:", precision_logistic)

recall_logistic = recall_score(y_test, y_pred_logistic)
```

```python
print("Recall:", recall_logistic)
```

```
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
```

```
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
```

```
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Selected Features (Logistic Regression): Index(['no_of_children',
'required_car_parking_space', 'repeated_guest',
       'no_of_special_requests', 'no_of_previous_bookings_not_canceled_log',
       'type_of_meal_plan_Meal Plan 2', 'room_type_reserved_Room_Type 2',
       'room_type_reserved_Room_Type 7', 'market_segment_type_Corporate',
       'market_segment_type_Offline'],
```

```
      dtype='object')
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

Best Hyperparameters (Logistic Regression): {'C': 0.01, 'penalty': 'l2'}
Accuracy on Test Set (Logistic Regression): 0.7337162067516458
Precision: 0.6225274725274725
Recall: 0.4827439284192586

/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/model_selection/_validation.py:425: FitFailedWarning:
20 fits failed out of a total of 40.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
20 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/model_selection/_validation.py", line 732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/barnana/anaconda3/lib/python3.10/site-packages/sklearn/base.py",
line 1151, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py", line 1168, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/linear_model/_logistic.py", line 56, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/barnana/anaconda3/lib/python3.10/site-
packages/sklearn/model_selection/_search.py:976: UserWarning: One or more of the
test scores are non-finite: [       nan 0.72739379        nan 0.71790271
```

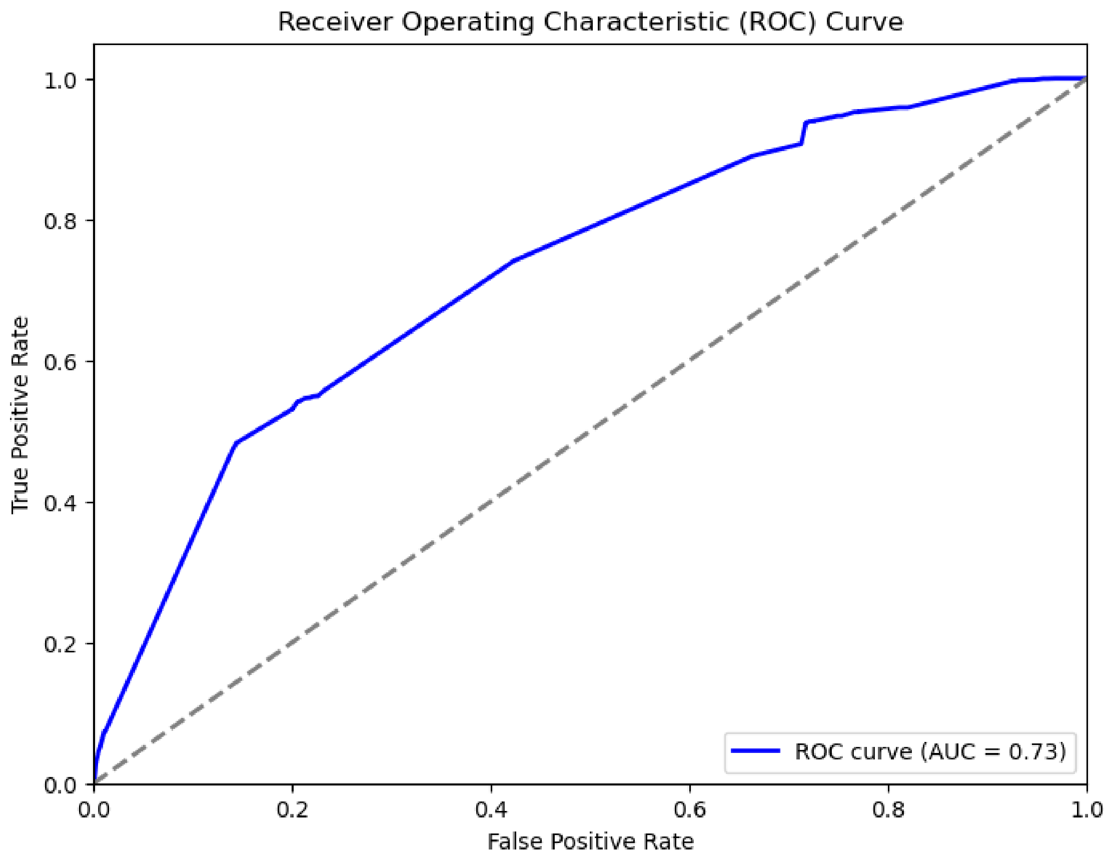
```
nan 0.7152762
        nan 0.7152762 ]
  warnings.warn(
```

Confusion Matrix

[ ]:
```
# Compute confusion matrix
cm_logistic = confusion_matrix(y_test, y_pred_logistic)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(4, 3))
sns.heatmap(cm_logistic, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label',fontdict={'weight': 'bold'})
plt.ylabel('True Label',fontdict={'weight': 'bold'})
plt.title('Confusion Matrix - Logistic Regression\n',fontdict={'weight':␣
↪'bold', 'size': 10})
plt.show()
```

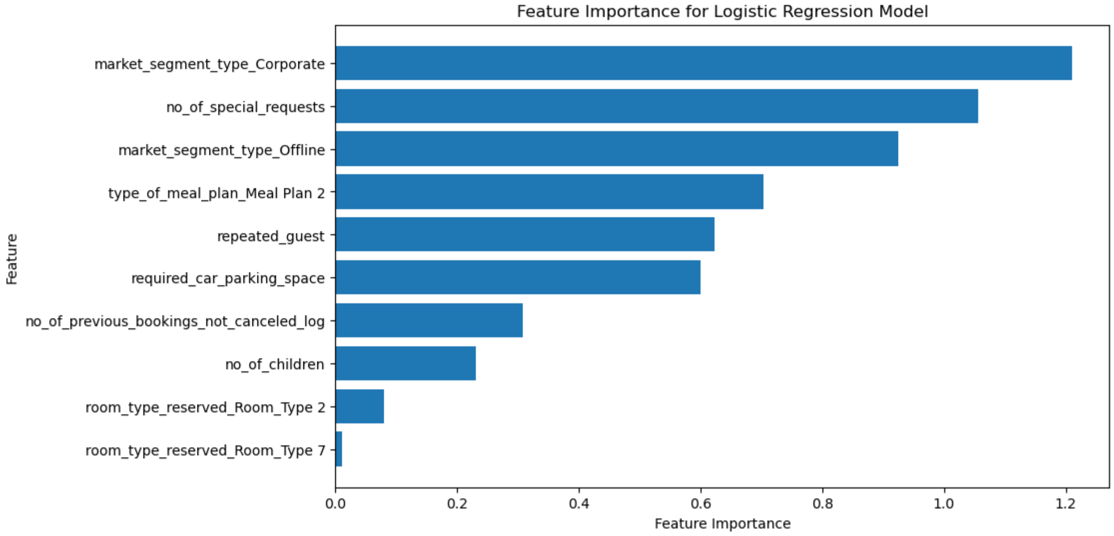
ROC curve and ROC AUC Score

[62]:
```
# Compute ROC curve and ROC AUC score
probs_logistic = best_model_logistic.
↪predict_proba(X_test[selected_features_logistic])[:, 1]
fpr_logistic, tpr_logistic, thresholds_logistic = roc_curve(y_test,␣
↪probs_logistic)
roc_auc_logistic = roc_auc_score(y_test, probs_logistic)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_logistic, tpr_logistic, color='blue', lw=2, label=f'ROC curve (AUC␣
↪= {roc_auc_logistic:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Feature Importance

[64]:
```
# Get feature importance for Logistic Regression model
feature_importance_logistic = best_model_logistic.coef_[0]
feature_importance_df_logistic = pd.DataFrame({'Feature':␣
↪selected_features_logistic, 'Importance': np.
↪abs(feature_importance_logistic)})
feature_importance_df_logistic = feature_importance_df_logistic.
↪sort_values(by='Importance', ascending=True)

# Plot feature importance
plt.figure(figsize=(8, 4))
plt.barh(feature_importance_df_logistic['Feature'],␣
↪feature_importance_df_logistic['Importance'])
plt.xlabel('Feature Importance',fontdict={'weight': 'bold'})
plt.ylabel('Feature',fontdict={'weight': 'bold'})
plt.title('Feature Importance for Logistic Regression Model',fontdict={'weight':
↪ 'bold', 'size': 10})
plt.show()
```



5.2 Random Forest

RFE, GridSearchCV, Model and Evaluation Metrics (Accuracy, Precision and Recall)

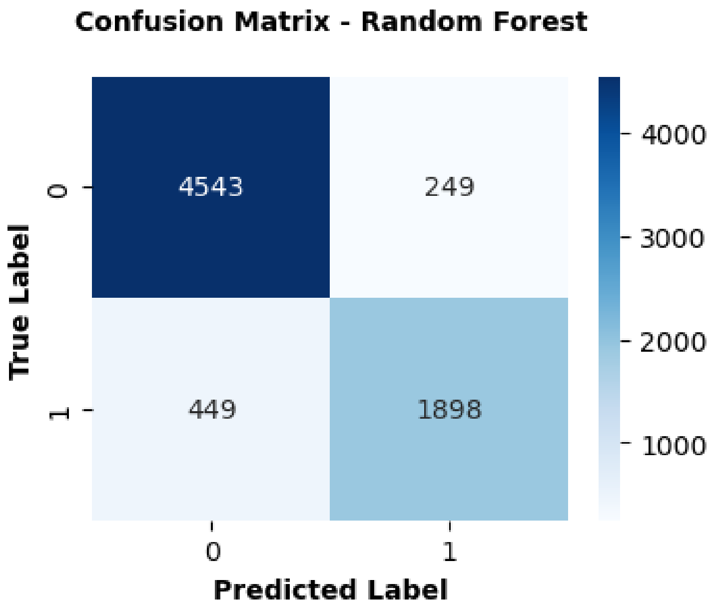
[65]:
```
# Recursive Feature Elimination with Random Forest
RandomForest_model = RandomForestClassifier()  # Choose the model you want to␣
 ↪use for RFE
rfe_RandomForest = RFE(RandomForest_model, n_features_to_select=10)  # Choose␣
 ↪the desired number of features
fit_RandomForest = rfe_RandomForest.fit(X_train, y_train)

# Selected features
selected_features_RandomForest = X_train.columns[fit_RandomForest.support_]
print("Selected Features:", selected_features_RandomForest)

# GridSearchCV to find the best hyperparameters
param_grid_RandomForest = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

RandomForest_model = RandomForestClassifier()
grid_search_RandomForest = GridSearchCV(RandomForest_model,␣
 ↪param_grid_RandomForest, cv=5, n_jobs=-1)
grid_search_RandomForest.fit(X_train[selected_features_RandomForest], y_train)

# Best hyperparameters
```

```
best_params_RandomForest = grid_search_RandomForest.best_params_
print("Best Hyperparameters:", best_params_RandomForest)

# Train the RandomForestClassifier with optimized hyperparameters
best_model_RandomForest = RandomForestClassifier(**best_params_RandomForest)
best_model_RandomForest.fit(X_train[selected_features_RandomForest], y_train)

# Evaluate the model on the test set
y_pred_RandomForest = best_model_RandomForest.
 ↪predict(X_test[selected_features_RandomForest])

# Calculate accuracy, precision and recall
accuracy_RandomForest = best_model_RandomForest.
 ↪score(X_test[selected_features_RandomForest], y_test)
print("Accuracy on Test Set:", accuracy_RandomForest)

precision_RandomForest = precision_score(y_test, y_pred_RandomForest)
print("Precision:", precision_RandomForest)

recall_RandomForest = recall_score(y_test, y_pred_RandomForest)
print("Recall:", recall_RandomForest)
```

```
Selected Features: Index(['no_of_weekend_nights', 'no_of_week_nights',
'no_of_special_requests',
       'lead_time_sqrt', 'avg_price_per_room_log', 'day', 'week', 'dayofweek',
       'dayofyear', 'market_segment_type_Online'],
      dtype='object')
Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100}
Accuracy on Test Set: 0.9025073539711445
Precision: 0.8877407233442931
Recall: 0.8052833404345974
```

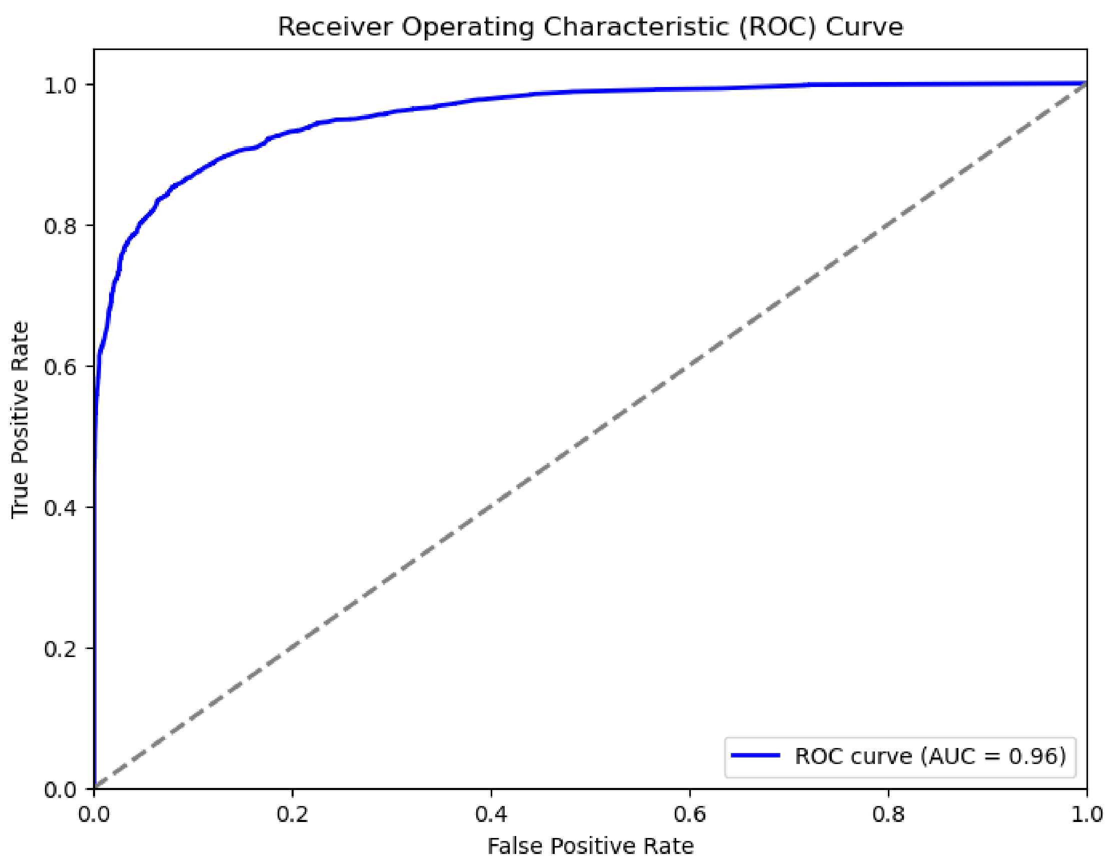
Confusion Matrix

[59]:
```
# Compute confusion matrix
cm_RandomForest = confusion_matrix(y_test, y_pred_RandomForest)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(4, 3))
sns.heatmap(cm_RandomForest, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label',fontdict={'weight': 'bold'})
plt.ylabel('True Label',fontdict={'weight': 'bold'})
plt.title('Confusion Matrix - Random Forest\n',fontdict={'weight': 'bold',␣
 ↪'size': 10})
plt.show()
```



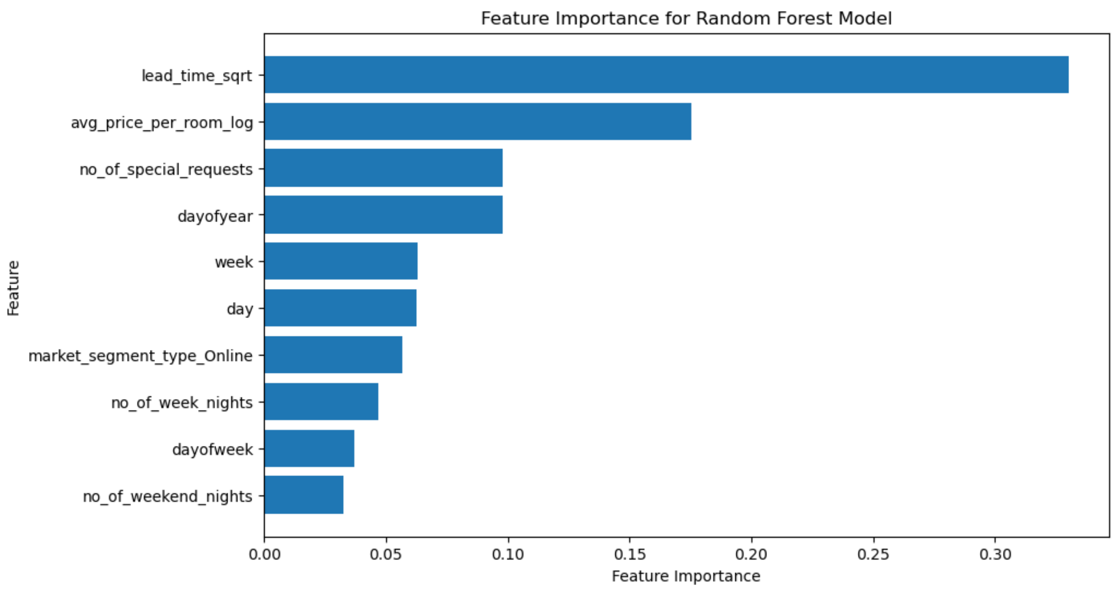
ROC curve and ROC AUC Score

[60]:
```
# Compute ROC curve and ROC AUC score
probs_RandomForest = best_model_RandomForest.
 ↪predict_proba(X_test[selected_features_RandomForest])[:, 1]
fpr_RandomForest, tpr_RandomForest, thresholds_RandomForest = roc_curve(y_test,␣
 ↪probs_RandomForest)
roc_auc_RandomForest = roc_auc_score(y_test, probs_RandomForest)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_RandomForest, tpr_RandomForest, color='blue', lw=2, label=f'ROC␣
 ↪curve (AUC = {roc_auc_RandomForest:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Feature Importance

[61]:
```
# Get feature importance for Random Forest model
feature_importance_randomforest = best_model_RandomForest.feature_importances_
feature_importance_df_randomforest = pd.DataFrame({'Feature':
 ↪selected_features_RandomForest, 'Importance':
 ↪feature_importance_randomforest})
feature_importance_df_randomforest = feature_importance_df_randomforest.
 ↪sort_values(by='Importance', ascending=True)

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df_randomforest['Feature'],
 ↪feature_importance_df_randomforest['Importance'])
plt.xlabel('Feature Importance',fontdict={'weight': 'bold'})
plt.ylabel('Feature',fontdict={'weight': 'bold'})
plt.title('Feature Importance for Random Forest Model',fontdict={'weight':
 ↪'bold', 'size': 10})
plt.show()
```



5.3 XGBoost

RFE, GridSearchCV, Model and Evaluation Metrics (Accuracy, Precision and Recall)

[63]:

```
# Create the XGBoost model
model_xgboost = xgb.XGBClassifier()

# Initialize RFE with the XGBoost model and the desired number of features to
↪select
rfe_xgboost = RFE(model_xgboost, n_features_to_select=10)

# Fit the RFE to the training data
fit_xgboost = rfe_xgboost.fit(X_train, y_train)

# Selected features
selected_features_xgboost = X_train.columns[fit_xgboost.support_]
print("Selected Features:", selected_features_xgboost)

# GridSearchCV to find the best hyperparameters for XGBoost
param_grid_xgboost = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.1, 0.01, 0.001],
    'gamma': [0, 0.1, 0.01]
}

xgb_model = xgb.XGBClassifier()
```

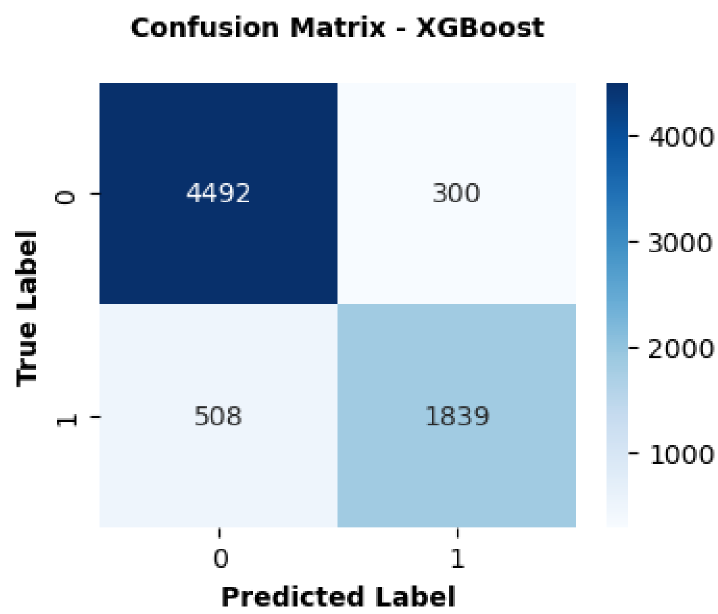
```
grid_search_xgboost = GridSearchCV(xgb_model, param_grid_xgboost, cv=5,␣
 ↪n_jobs=-1)
grid_search_xgboost.fit(X_train[selected_features_xgboost], y_train)

# Best hyperparameters
best_params_xgboost = grid_search_xgboost.best_params_
print("Best Hyperparameters:", best_params_xgboost)

# Train the XGBoost model with the selected features and best hyperparameters
selected_model_xgboost = xgb.XGBClassifier(**best_params_xgboost)
selected_model_xgboost.fit(X_train[selected_features_xgboost], y_train)

# Make predictions on the test set
y_pred_xgboost = selected_model_xgboost.
 ↪predict(X_test[selected_features_xgboost])

# Calculate the accuracy,precision and recall of the model
accuracy_xgboost = accuracy_score(y_test, y_pred_xgboost)
print("Accuracy on Test Set:", accuracy_xgboost)

precision_xgboost = precision_score(y_test, y_pred_xgboost)
print("Precision:", precision_xgboost)

recall_xgboost = recall_score(y_test, y_pred_xgboost)
print("Recall:", recall_xgboost)
```

```
Selected Features: Index(['no_of_adults', 'required_car_parking_space',
'repeated_guest',
       'no_of_special_requests', 'lead_time_sqrt', 'avg_price_per_room_log',
       'year', 'month', 'market_segment_type_Offline',
       'market_segment_type_Online'],
      dtype='object')
Best Hyperparameters: {'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 5,
'n_estimators': 300}
Accuracy on Test Set: 0.886818882196386
Precision: 0.8597475455820477
Recall: 0.7835534725181083
```

Confusion Matrix

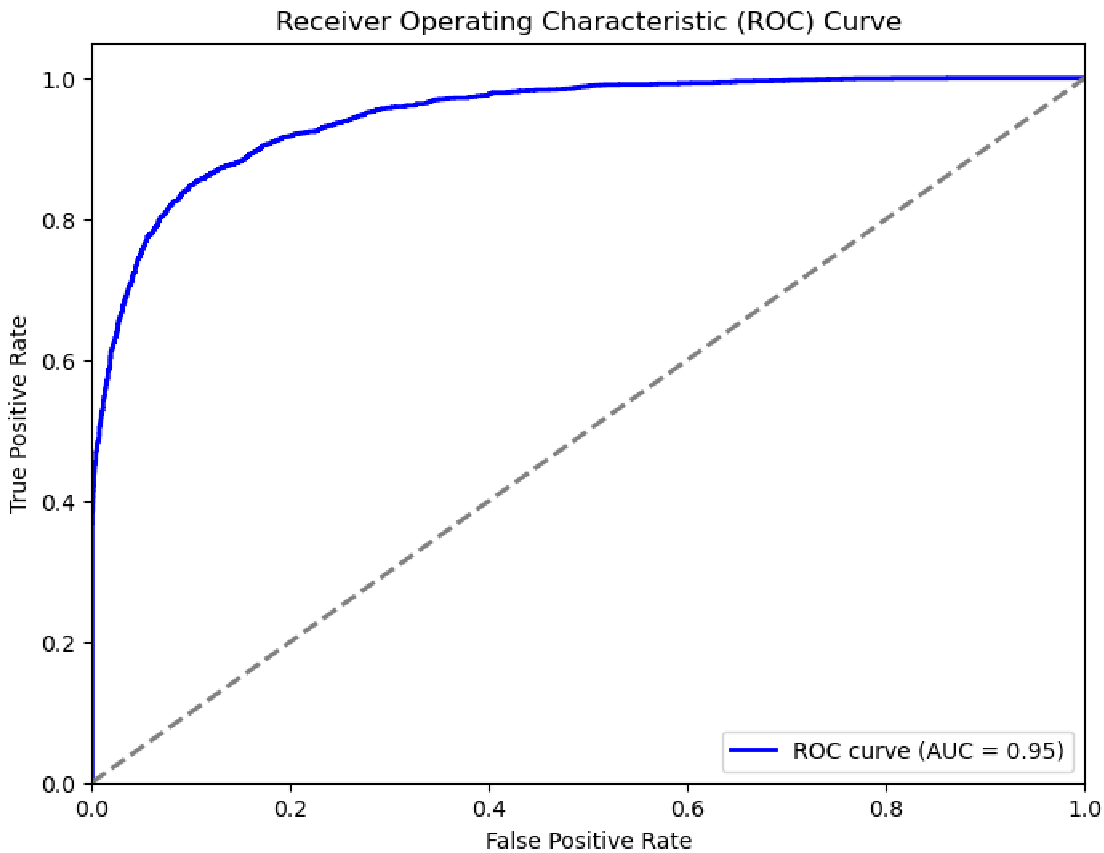
[67]:
```
# Compute confusion matrix
cm_xgboost = confusion_matrix(y_test, y_pred_xgboost)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(4, 3))
sns.heatmap(cm_xgboost, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label',fontdict={'weight': 'bold'})
plt.ylabel('True Label',fontdict={'weight': 'bold'})
```

```
plt.title('Confusion Matrix - XGBoost\n',fontdict={'weight': 'bold', 'size':␣
 ↪10})
plt.show()
```



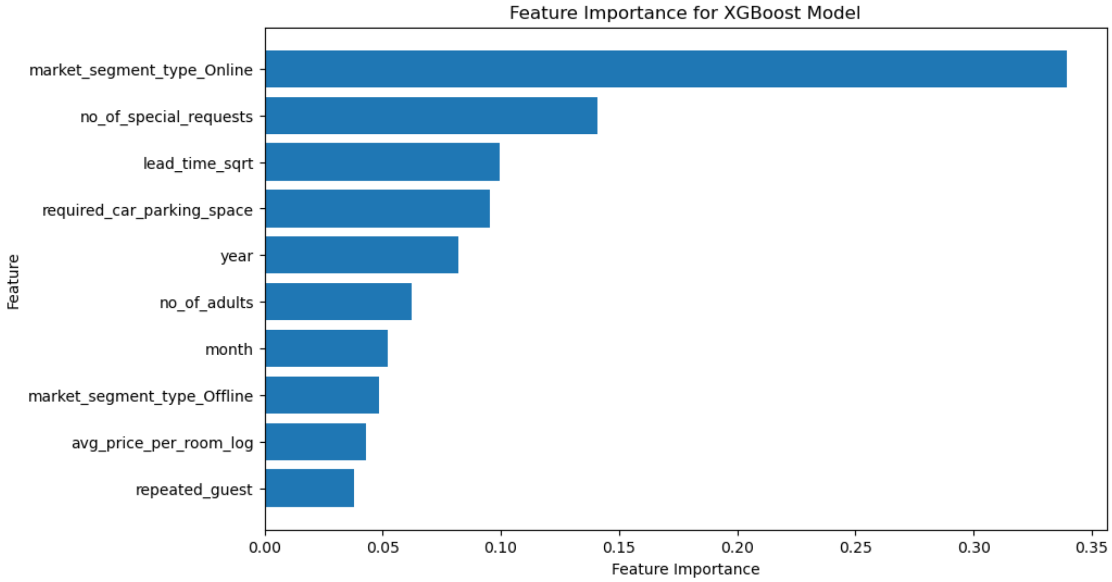
ROC curve and ROC AUC Score

[68]:
```
# Compute ROC curve and ROC AUC score
probs_xgboost = selected_model_xgboost.
 ↪predict_proba(X_test[selected_features_xgboost])[:, 1]
fpr_xgboost, tpr_xgboost, thresholds_xgboost = roc_curve(y_test, probs_xgboost)
roc_auc_xgboost = roc_auc_score(y_test, probs_xgboost)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_xgboost, tpr_xgboost, color='blue', lw=2, label=f'ROC curve (AUC =␣
 ↪{roc_auc_xgboost:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Feature Importance

[66]:
```
# Show feature importance for XGBoost model
feature_importance = selected_model_xgboost.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': selected_features_xgboost,␣
↪'Importance': feature_importance})
feature_importance_df = feature_importance_df.sort_values(by='Importance',␣
↪ascending=True)

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xlabel('Feature Importance',fontdict={'weight': 'bold'})
plt.ylabel('Feature',fontdict={'weight': 'bold'})
plt.title('Feature Importance for XGBoost Model',fontdict={'weight': 'bold',␣
↪'size': 10})
plt.show()
```


Feature Importance for XGBoost Model
market_segment_type_Online
no_of_special_requests
lead_time_sqrt
required_car_parking_space
year
no_of_adults
month
market_segment_type_Offline
avg_price_per_room_log
repeated_guest
Feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Feature Importance

```
[ ]:
```