Machine Learning in Python - Group Project 2

Due Friday, April 14th by 16.00 pm.

William Jones, Tristan Pang, Aidan Garrity, Elliot Leishman

(Group 15)

1. Introduction

1.1 Executive summary

This report analyses the factors causing hotel booking cancellations. By using a data set with over a 100 thousand datapoints and features such as lead time and country of origin, we performed exploratary data analysis and created a machine learning model using a forest based approach. Our final model succesfully predicted 82% of our test data. This suggests that our model can be useful in predicting cancellations, and can be used to optimise hotel room allocations which will generate more profit in the long run.

Based on our insights, we recommend that customers are encouraged to book closer to the date, and that customers from demographics that are less likely to cancel are targetted. Further work can be done with more data to generate a algorithmic model to minimise empty rooms and improve hotel revenue.

1.2 Background and introduction

According to Avivo.com [1], in the year 2022, almost 20% of hotel bookings were cancelled across the UK. Clearly this is a major issue for hoteliers who risk losing out on business if customers cancel one fith of all their stays, and hence revenue. If one could predict with a reasonable level of confidence that a client will cancel their booking, a hotelier could double book a room to reduce potential loss of revenue. However, this obviously comes with an inherent risk; what if the client doesn't cancel and the hotelier finds themselves double booked? The aim of this report is to produce an accurate model which will use a customer's booking information to predict if they are likely to cancel. We will analyse our results in order to increase the understanding among hoteliers as to which customers are most likely to cancel bookings.

The specifications of the model provided in this report are as follows:

- Model must be an accurate predictor of hotel booking cancellations,
- Model must be easily interpretable to those without a data science background.

Displayed below, we were given a large comprehensive dataset containing booking information for two hotels between the 1st of July 2015 to the 31st of July 2017. Although this dataset was gathered a few years ago now we believe it is still relevant to the hoteliers of today. In fact, even more so than more recent datasets which cover the COVID-19 pandemic. The subsequent lockdowns around the world had a huge impact on hotel bookings which we do not want to influence our model. The two hotels that we have information on are a city hotel and a resort hotel. Having information on two different types of hotel will make the model more robust and increase generalisability because the model is not trained on data from just one type of hotel.

A potential issue we observed with this dataset was the fact that it contained many missing values and erroneous data entries. This is to be expected with datasets gathered from the real world and meant that care had to be taken to clean the data before using it in our models. More information on this process can be found in section 2 of this report.

This type of problem is known to data scientists as a 'classification problem', this means we want to be able to predict a discrete outcome. In this context where we are assuming a booking can be cancelled or not cancelled, moreover we assume these events to be mutually exclusive. The outcome or response in our dataset is called 'is_cancelled' and takes the binary values 0 (not cancelled) and 1 (cancelled).

There are 4 major types of classification models we considered in this report, they were logistic regression, support vector classifiers, decision trees and neural networks. Each type of model has its own advantages and disadvantages, which we discuss now.

Logistic regression is perhaps the simplest classifier. It has many advantages such as the fact it is easily interpretable, easily to implement and is efficient for small datasets. These advantages help explain why logistic regression is very popular in many machine learning settings. However, logistic regression does have some disadvantages, most importantly it assumes a linear relationship between the continuous features and the log-odds of the response, which is not necessarily the case for real-world data. For this reason, we decided against using logistic regression for our model.

Support vector classifiers, usually shortened to (SVCs), work by splitting the feature space with a hyper plane which maximises the margin between the two classes. One advantage of SVCs is we can use a kernel function to manipulate the hyper plane, which is very useful to separate data which are not linearly separable. But SVCs have two major drawbacks for us, first they are computationally expensive to train on big datasets such the one we considered in this report. Furthermore, they are not easily interpretable, it can be very difficult to understand the importance of individual features in the classification. Since a major part of our specification was that the model had to be easily interpretable, we will not further consider SVCs in this report.

Neural networks are currently the most 'on-trend' classifier in the world of machine learning and rightly so, they can provide unparalleled accuracy in prediction compared to conventional classifiers. They work by using layers of interconnected nodes to learn complex patterns in the data and make predictions. However, they suffer from similar disadvantages to SVCs, namely, they can be computationally intensive to train and they are hard to interpret. In fact, neural networks are colloquially referred to as 'black box' models, illustrating just how difficult it can be to understand how it arrived at its predictions. For these reasons, we will not suggest a model based on neural networks.

The final type of classifier we will discuss are decision trees. Decision trees are an easy to implement classifier that can handle nonlinear and missing data. Perhaps their biggest strength is that they can be displayed graphically, this means they can be interpreted easily by someone with zero knowledge of data science. It is for these reasons that the model presented in this report is based on a decision tree model. Decision trees do have some disadvantages such as the fact that they are prone to overfitting, but these disadvantages can be dealt with in the model creation phase. One method we make use of is known as 'ensemble methods' these essentially combine the building blocks of different models to create a better single model. More detail is provided on ensemble methods in section 3.

Now we have discussed the general model we used we now need to discuss how we evaluate the performance of specific models. We use evaluation metrics to do this. The most common evaluation metric for classifiers is known as accuracy. Accuracy is defined as the number of correct classifications divided by the total number of classifications. Clearly this is a very important and interpretable metric as it simply displays the percentage of classifications the model got correct.

Two other useful metrics are known as precision and recall. Recall, also known as the true positive rate, is the number of positive classifications by the model divided by the total number of true positive observations. Whereas precision is the number of true positive cases divided by the number of positive classifications made by the model.

Precision and recall have advantages over accuracy when the 'cost' of misclassifying is not equal. By this we mean the cost faced by the client for a false positive might be greater than the cost faced by a false negative. In our context a false positive occurs when the model doesn't predict a cancellation but the customer ends up cancelling. Obviously, this can lead to loss of revenue for the hotelier as they risk having an empty room for a night. Conversely, a false positive occurs when the model predicts a cancellation which doesn't occur. This could lead to a room being double booked which comes with its own problems such as angry customers and bad reviews which could harm the longer-term revenue of the hotel.

As we can see false positives and false negatives both result in bad outcomes for a hotelier. We believe, however, that false positives are slightly worse in this context so we shall evaluate our models with favour to accuracy and precision.

The main structure of the report is as follows: in section 2 we clean the data, visualise and examine trends in the features. A common machine learning technique for datasets with a large number of features is called Principal Component Analysis (PCA) this transforms the features to allow us to encapsulate move of the data variability in fewer features. We decided against using PCA in this report because it can be hard to understand which features are most important for the classifier, and, as stated earlier, interpretability is a major part of our model choice. In the third section we introduce our model choice, a histogram based gradient boosting classification tree, for this classification problem. We discuss how the model works and also perform tests on the model to show reliability and accuracy. In the final section we review the model and provide an understanding for hoteliers as to what type of clients are most likely to cancel their bookings.

Our final model is a forest-based model using gradient boosting. This results in an accuracy of 82% and precision of 73%. Other models were also tried, but they all gave worse accuracy and were less computationally efficient. The model uses many decision trees and adds the results up. Thus, we can identify important features, which include how far in advance bookings are made, and the country of origin of customers. For example, customers who book further in advance are more likely to cancel than customers who book close to their arrival date. The features identified by our final model are consistent with our data analysis.

1.3 General Setup

In [1]:
```
# Add any additional libraries or submodules below

# Display plots inline
%matplotlib inline

# Data libraries
import pandas as pd
import numpy as np
```

```
# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats

# sklearn modules
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# For country heatmap
import folium
from folium.plugins import HeatMap
import plotly.express as px
```

In [2]:
```
# Plotting defaults
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80
```

In [3]:
```
# Load data
d = pd.read_csv("hotel.csv")
```

In [4]:
```
def proportion_table(df, feat, index, show = True):
    '''
    Creates a proportion table for a feature in dataframe. Takes
    a dataframe `df`, and a feature column `feat` and index row.
    Returns proportion table (prints if show=True).
    '''

    # Create proportion table of entire feature
    prop_table = pd.DataFrame(df[feat].value_counts()).reset_index()
    prop_table = prop_table.rename(columns= {feat:'Total', 'index':index})

    for i in [0,1]: # Denotes if we are considering cancelation or not
        if i == 0:
            cancel_status = 'Not_canceled'
        else:
            cancel_status = 'Canceled'
        df_canc = df[(df['is_canceled'] == i)]
        df_canc = pd.DataFrame(df_canc[feat].value_counts()).reset_index()
        df_canc = df_canc.rename(columns= {feat:cancel_status, 'index':index})
        prop_table = prop_table.merge(df_canc, on = index, how = 'outer')

        # Calculate propoprtions
        prop_table[f'% {cancel_status}'] = prop_table[cancel_status] / prop_table['Total'] * 100

    # fill NaN values with zeros
    prop_table = prop_table.fillna(0)

    # Round proportions to two decimal places
    prop_table = prop_table.round(2)



    # Display DataFrame or return it
    if show:
        display(prop_table)
    else:
        return(prop_table)
```

In [5]:
```
display(d)
```

	is_canceled	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_i
0	0	Resort Hotel	342	2015	July	27	1	
1	0	Resort Hotel	737	2015	July	27	1	
2	0	Resort Hotel	7	2015	July	27	1	
3	0	Resort Hotel	13	2015	July	27	1	
4	0	Resort Hotel	14	2015	July	27	1	
...	...	...	...	...	...	...	...	...
119385	0	City Hotel	23	2017	August	35	30	
119386	0	City Hotel	102	2017	August	35	31	
119387	0	City Hotel	34	2017	August	35	31	
119388	0	City Hotel	109	2017	August	35	31	
119389	0	City Hotel	205	2017	August	35	29	

119390 rows × 30 columns

---

2. Exploratory Data Analysis and Feature Engineering

This section explores the dataset of booking information supplied by the hotel chain, more specifically we clean the data and analyse each of the features. This analysis will inform us when it comes to selecting a model. The structure of this section is as follows:

- Initial data cleaning.
- Collection errors.
- Response variable.
- Feature investigation.
- Data encoding.
- Correlation.

Section 2.4, Feature investigation, is split up into further subsections to increase readability.

2.1 Initial data cleaning

In this section we will perform basic cleaning of the data. In particular, we will:

- Deal with na values.
- Check for duplicate values.
- Examine the structure, shape and type of data features.
- Encode categorical variables with numerical values.

\newline We start by examining any null values that exist in the dataframe.

In [6]:
```
# Copy original dataframe
df = d.copy()

# Get columns which have NaN values
cols_with_nan = d.columns[d.isna().any()].tolist()
print('Features containing NaN values are: ', cols_with_nan)

# Find out how many NaN values are in those columns
for col in cols_with_nan:

    # Count number of na values
    no_nas = d[col].isna().sum()
```

```
    # Calculate the percentage of total values which are nas
    perc_vals_na = 100 * no_nas /119390

    print(f"Column '{col}' contains {no_nas} NaN values, this constitutes {np.round(perc_vals_na,2)}% of values.")
```

```
Features containing NaN values are:  ['children', 'country', 'agent', 'company']
Column 'children' contains 4 NaN values, this constitutes 0.0% of values.
Column 'country' contains 488 NaN values, this constitutes 0.41% of values.
Column 'agent' contains 16340 NaN values, this constitutes 13.69% of values.
Column 'company' contains 112593 NaN values, this constitutes 94.31% of values.
```

From the code segment above we can see that the 'children' and 'country' features have 4 and 488 NaN values, respectively. Because these observations represent such a small percentage (<0.5%) of our total observations we drop the entire observation from the dataframe rather then work to supply a value for them. For the 'agent' feature we assume these NaN values correspond to zero i.e. no agent was used in the booking process, therefore we substitute these values with zero. We make this assumption because there are no observations with an agent value of zero in the raw data. For the company feature we remove the entire feature from our dataset because it contains 94% NaN values.

Furthermore, we drop duplicate observations here. We can do this with confidence because it is very unlikely that two parties will have booked the exact same stay in the exact same hotel on the exact same day. We acknowledge that although unlikely, this is still possible, but we are assume these 'true' duplicates make up a negligable proportion of the total duplicates.

We implement these changes and next consider the datatypes in the dataset.

In [7]:
```
# drop rows which have NaN values in 'children' columns.
df = df.dropna(subset=['children', 'country'])

df = df.drop('company', axis = 1)

# fill the remaining NaN values to be zero
df = df.fillna(0)

# Delete duplicate rows. Note 32215 rows are duplicated rows to delete
df.drop_duplicates(keep='first', inplace=True)
```

In [8]:
```
# Investigating shape and datatypes of dataframe
print(f'The dataframe is of size {df.shape}')
print(df.dtypes)
```

```
The dataframe is of size (86675, 29)
is_canceled                      int64
hotel                           object
lead_time                        int64
arrival_date_year                int64
arrival_date_month              object
arrival_date_week_number         int64
arrival_date_day_of_month        int64
stays_in_weekend_nights          int64
stays_in_week_nights             int64
adults                           int64
children                       float64
babies                           int64
meal                            object
country                         object
market_segment                  object
distribution_channel            object
is_repeated_guest                int64
previous_cancellations           int64
previous_bookings_not_canceled   int64
reserved_room_type              object
assigned_room_type              object
booking_changes                  int64
deposit_type                    object
agent                          float64
days_in_waiting_list             int64
customer_type                   object
adr                            float64
required_car_parking_spaces      int64
total_of_special_requests        int64
dtype: object
```

We observe that the dataframe has 29 features and 119390 observations. Due to overfitting and cost issues some models may be impractical to use with such a large dataset so we will have to consider the size of the dataframe when selecting a model. We may choose to only use a sample of this dataset.

16 features are of datatype int, 4 are floats and 10 are object. These datatypes are indeed suitable for the features we have. For example, integer types are counting number of occurances and hence it makes sense to store these are integers.

For the float features we:

- Convert the 'children' feature into an integer because we cannot have a non-whole number of children.
- Rename 'adr' as 'avg_daily_rate' as this more clearly conveys what it means. We keep this feature as a float because it corresponds to a price and will need to store values after the decimal point.
- Re-encode 'agent' as a binary integer feature. This column has a number encoding the travel agent the hotel was booked through but we do not have access to this encoding. Therefore, we would not be able to reccommend a specific agent to the clients. We will instead investigate whether booking through an agent affects cancellation as opposed to investigating which agents affect cancellations.

Note: For the features of type object, we will later need to encode them with numerical values before fitting our model. We perform this encoding later, to make feature analysis easier.

In [9]:
```
# Convert children column to integer
df['children'] = df['children'].astype('int64')

# Rename the adr column
df = df.rename(columns={"adr": "avg_daily_rate"})

# Encode agent as binary 1 = agent used, 0 = agent not used
agent_used = (df.agent != 0)
df.loc[agent_used, 'agent'] = 1

# Change datatype to integer
df['agent'] = df['agent'].astype('int64')
```

2.2 Collection errors

We have considered NaN values and duplicates but we must also consider observations in our data set that do not make sense. In particular we will look at the following three scenarios:

- A room has been booked but the total number of adults, children and babies staying in that booking is zero.
- A room was stayed in for zero weekday nights and zero weekend nights.
- A customer is marked as a repeat customer but has no previous stays or cancelations recorded.

Any observation which falls into one of these three categories is removed at this stage.

In [10]:
```
# No adults, children or babies
no_people = (df.children == 0) & (df.adults == 0) & (df.babies == 0)
print(f'Number of observations with no people is {df[no_people].shape[0]}')

# No weekend and weekday nights were spent at hotel
no_nights = (df.stays_in_weekend_nights == 0) & (df.stays_in_week_nights == 0)
print(f'Number of observations with no nights is {df[no_nights].shape[0]}')

# Repeat customer with no previous stays recorded
no_prev_stays = (df.is_repeated_guest == 1) & (df.previous_cancellations == 0) & (df.previous_bookings_not_canceled
print(f'Number of observations with incorrect previous stays is {df[no_prev_stays].shape[0]}')

# Delete these anomolous observations from the dataframe
bool_series = no_people|no_nights|no_prev_stays
df = df[~bool_series.reindex(df.index)]

print(f'Number of removed observations is {df[bool_series.reindex(df.index)].shape[0]}')
```

```
Number of observations with no people is 161
Number of observations with no nights is 642
Number of observations with incorrect previous stays is 506
Number of removed observations is 0
```

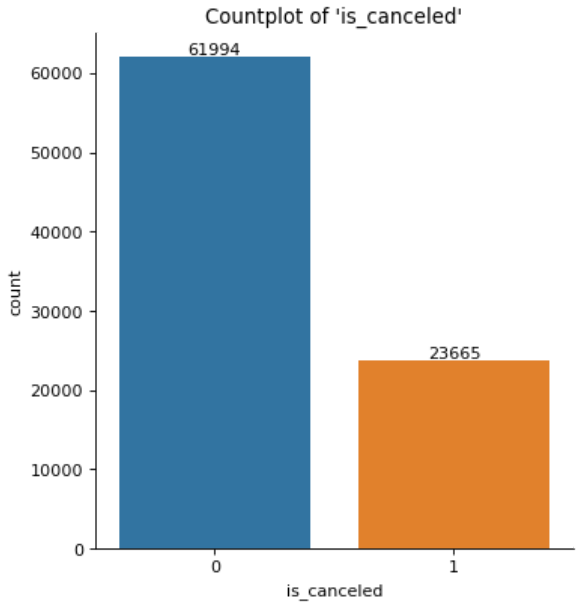
We have removed 913 observations from the dataset because they did not make logical sense. The most common collection error was not recording any nights stayed.

2.3 Response variable

In this section we will examine the response variable 'is_canceled'. We examine the balance and distribution of the classes as this will inform us on our model choice.

In [11]:
```
sns.catplot(data = df, x = 'is_canceled', kind  = 'count') # count plot using seaborn library

# Print the counts on each bar
ax = plt.gca()  # Get the current axes
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',  # Labels
                (p.get_x() + p.get_width() / 2, p.get_height()),  # The x, y coordinates of the text
                ha='center', va='bottom')  # Horizontal and vertical alignment of the text
```

```
plt.title("Countplot of 'is_canceled'")
plt.show()
```



The above plot shows that the ratio of not cancelled to cancelled is approximately 3:1. Therefore, the data response variable does not exhibit a large class imbalance and we do not to be overly cautious when constructing our model. To increase the accuracy of the model we will conduct stratified sampling of the response variable when we doing the test-train split.

2.4 Feature Investigation

In this subsection we examine each remaining feature of the dataset. We will consider seperately the continuous features and the categorical features. For each we suitably visualise the data, examine summary statistics and use our knowledge of the context to establish possible outliers. When considering categorical variables we will further split them up into four groups: date & time, personal information, booking information and other information.

2.4.1 Continuous Features

The only continuous features in our dataset are: 'avg_daily_rate' and 'lead_time'.

In [12]:
```
# continuous features we will consider
continuous_features = ['avg_daily_rate', 'lead_time']

# Print summary statistics for avg_daily_rate
display(df[continuous_features].describe().round(2))
```

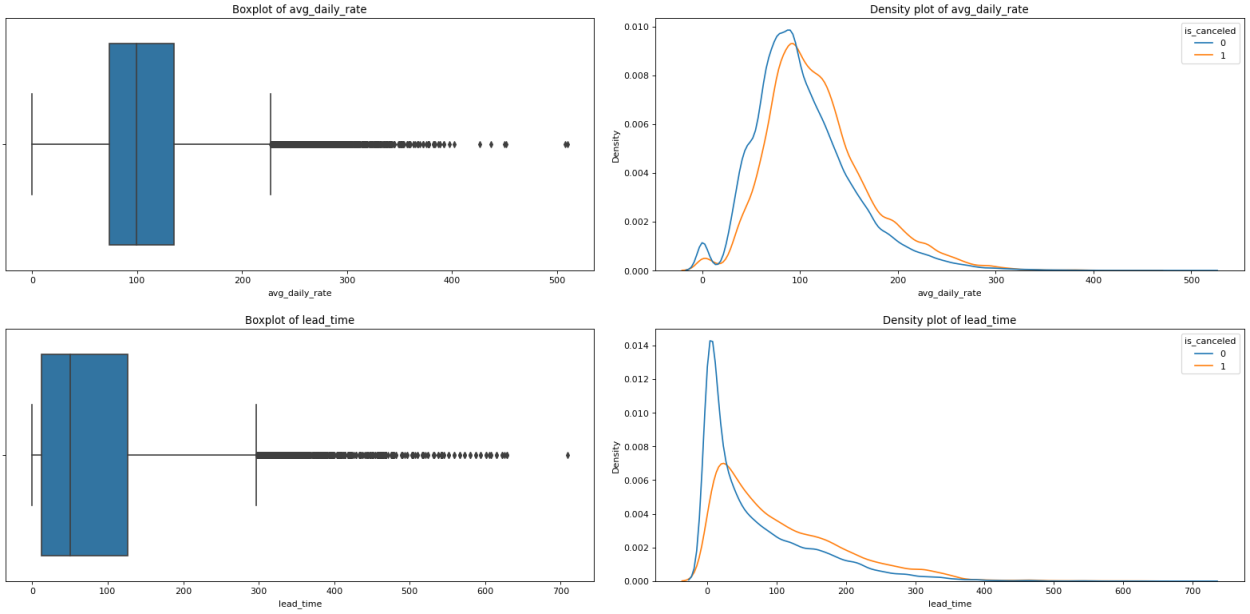
	avg_daily_rate	lead_time
count	85659.00	85659.00
mean	107.55	80.63
std	54.33	86.03
min	-6.38	0.00
25%	73.46	12.00
50%	99.00	50.00
75%	135.00	126.00
max	5400.00	709.00

On review of the summary statistics we can see that the minimum value for 'avg_daily_rate' is negative - this makes no sense as it implies the hotel is paying someone to stay there. To counteract this we will remove all observations that have a negative 'avg_daily_rate'. Another consideration is that the max value is 5400 - this is 98 standard deviations away from the mean so we will consider it an outlier and remove it. This summary statistics of 'lead_time' does not indicate any immediate outliers.

In [13]:
```
# Remove observations with negative average daily rate
neg_adr = (df.avg_daily_rate < 0)
print(f'Number of observations with negative adr is {df[neg_adr].shape[0]}')
df = df[~neg_adr]

# Remove outlier
outlier = (df.avg_daily_rate == 5400)
print(f'Number of outliers with value 5400 is {df[outlier].shape[0]}')
```

```
df = df[~outlier]

fig, axes = plt.subplots(2, 2, figsize = (20,5*len(continuous_features)))

for i, feature in enumerate(continuous_features):
    ## Boxplot ##
    sns.boxplot(data = df, x = feature, ax = axes[i][0])
    axes[i][0].set_title(f'Boxplot of {feature}')

    ## Density Plot ##
    # common_norm = False, sets each curve to have density 1
    sns.kdeplot(data = df, x = feature, hue = 'is_canceled', common_norm = False, ax = axes[i][1])
    axes[i][1].set_title(f'Density plot of {feature}')

plt.tight_layout(pad = 2.0)
```

```
Number of observations with negative adr is 1
Number of outliers with value 5400 is 1
```



The boxplot and density graphs for these two features allows us to make conclusions about both of them:

- '***avg_daily_rate***': Firstly, the boxplot verifies our decision to remove the outlier value of 5400 here. Following on, we see that there are some values close to zero, intuitively these are hard to explain in context, but they could be due to the customer recieving a large discount. If we look at the density plot we can see that these near-zero values don't make up a large proportion of the observations so we will leave them in our dataset. The density plot also shows us that the distribution of daily average rate is roughly the same for cancelled bookings and non-cancelled bookings, therefore, we wouldn't expect avg_daily_rate to have a large impact on cancellations.

- '***lead_time***': We see that the median is substantially smaller than the mean, skewing the distribution to the left, a pattern which is confirmed by the density plot. The boxplot shows us that there are a large number of observations classed as outliers. We will not remove any of these outliers because they are consistent and although unlikely, it is possible for a customer to book a hotel room more than 500 days in advance. For example, a customer could be booking a wedding venue. Furthermore, the difference in distribution of the response variable implies we might expect lead_time to be an important feature in our classification.

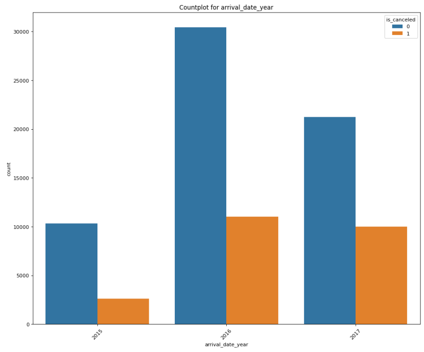
2.4.2 Categorical Features: *Date & Time*

In this subsection we consider 'arrival_date_year', 'arrival_date_month', 'arrival_date_week_number', 'arrival_date_day_of_month', 'stays_in_weekend_nights' and 'stays_in_week_nights'.

In [14]:
```
# Create a list of features we wil consider here
datetime_features = ['arrival_date_year', 'arrival_date_month', 'arrival_date_week_number', 'arrival_date_day_of_mon

fig, axes = plt.subplots(6, 3, figsize = (35,10*len(datetime_features)))

for i, feature in enumerate(datetime_features):
    ## Countplot ##
    sns.countplot(data = df, x = feature, hue = 'is_canceled', ax = axes[i][0])
    axes[i][0].tick_params(axis='x', rotation=45)
    axes[i][0].set_title(f'Countplot for {feature}')

    ## Proportion Table ##
    prop_table = proportion_table(df, feature, feature, show = False)
    axes[i][1].axis('off')
```

```
    table = axes[i][1].table(cellText=prop_table.values, colLabels=prop_table.columns, rowLabels=prop_table.index, b
    table.auto_set_font_size(False)
    table.set_fontsize(9)  # Set the font size to 9
    axes[i][1].set_title(f"Proportion Table for {feature}")

    ## Summary Statistics ##
    sum_stats = df[feature].describe().to_frame()
    axes[i][2].axis('off')
    table = axes[i][2].table(cellText=sum_stats.values, colLabels=sum_stats.columns, rowLabels=sum_stats.index, bbox
    table.auto_set_font_size(False)
    table.set_fontsize(9)  # Set the font size to 9
    axes[i][2].set_title(f"Summary Statistics for {feature}")

plt.tight_layout(pad = 2.0)
```

Countplot for arrival_date_year



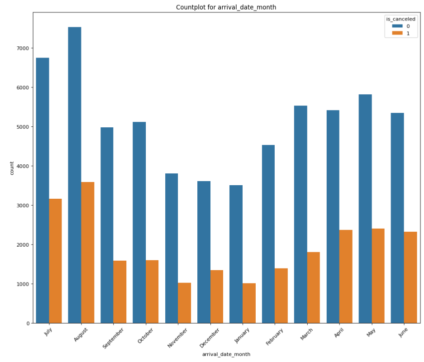
Proportion Table for arrival_date_year

	arrival_date_year	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	2016.0	41483.0	30443.0	73.39	11040.0	26.61
1	2017.0	31253.0	21241.0	67.96	10012.0	32.04
2	2015.0	12921.0	10309.0	79.78	2612.0	20.22

Summary Statistics for arrival_date_year

	arrival_date_year
count	85657.0
mean	2016.2140163676056
std	0.6855001666896033
min	2015.0
25%	2016.0
50%	2016.0
75%	2017.0
max	2017.0

Countplot for arrival_date_month



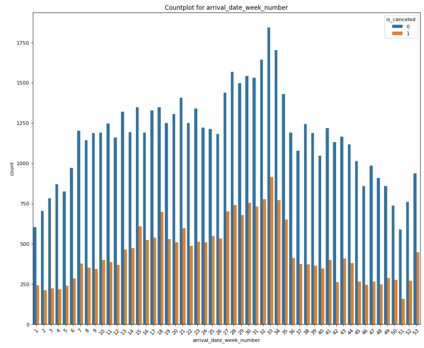
Proportion Table for arrival_date_month

	arrival_date_month	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	August	11132	7537	67.71	3595	32.29
1	July	9918	6749	68.05	3169	31.95
2	May	8228	5826	70.81	2402	29.19
3	April	7790	5421	69.59	2369	30.41
4	June	7675	5346	69.65	2329	30.35
5	March	7345	5536	75.37	1809	24.63
6	October	6728	5122	76.13	1606	23.87
7	September	6579	4986	75.79	1593	24.21
8	February	5928	4538	76.55	1390	23.45
9	December	4962	3610	72.75	1352	27.25
10	November	4839	3809	78.71	1030	21.29
11	January	4533	3513	77.5	1020	22.5

Summary Statistics for arrival_date_month

	arrival_date_month
count	85657
unique	12
top	August
freq	11132

Countplot for arrival_date_week_number



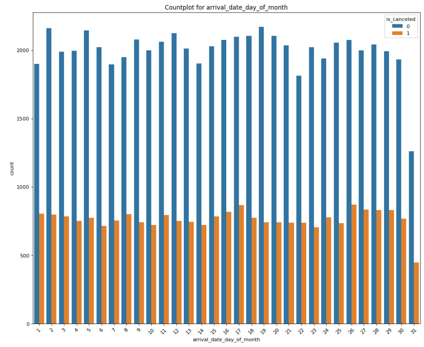
Proportion Table for arrival_date_week_number

	arrival_date_week_number	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	33.0	2760.0	1844.0	66.81	916.0	33.19
1	34.0	2472.0	1702.0	68.85	770.0	31.15
2	32.0	2419.0	1642.0	67.88	777.0	32.12
3	28.0	2308.0	1568.0	67.94	740.0	32.06
4	30.0	2298.0	1543.0	67.15	755.0	32.85
5	31.0	2262.0	1531.0	67.68	731.0	32.32
6	29.0	2174.0	1496.0	68.81	678.0	31.19
7	27.0	2139.0	1439.0	67.27	700.0	32.73
8	35.0	2078.0	1429.0	68.77	649.0	31.23
9	18.0	2046.0	1349.0	65.93	697.0	34.07
10	21.0	2006.0	1408.0	70.19	598.0	29.81
11	15.0	1957.0	1349.0	68.93	608.0	31.07
12	17.0	1864.0	1327.0	71.19	537.0	28.81
13	23.0	1851.0	1339.0	72.34	512.0	27.66
14	20.0	1816.0	1307.0	71.97	509.0	28.03
15	13.0	1785.0	1320.0	73.95	465.0	26.05
16	19.0	1778.0	1250.0	70.3	528.0	29.7
17	25.0	1761.0	1212.0	68.82	549.0	31.18
18	22.0	1737.0	1250.0	71.96	487.0	28.04
19	24.0	1729.0	1220.0	70.56	509.0	29.44
20	16.0	1714.0	1190.0	69.43	524.0	30.57
21	26.0	1714.0	1183.0	69.02	531.0	30.98
22	14.0	1666.0	1193.0	71.61	473.0	28.39
23	11.0	1633.0	1247.0	76.36	386.0	23.64
24	41.0	1618.0	1218.0	75.28	400.0	24.72
25	38.0	1616.0	1243.0	76.92	373.0	23.08
26	36.0	1603.0	1191.0	74.3	412.0	25.7
27	10.0	1591.0	1191.0	74.86	400.0	25.14
28	7.0	1578.0	1202.0	76.17	376.0	23.83
29	43.0	1572.0	1164.0	74.05	408.0	25.95
30	39.0	1551.0	1188.0	76.6	363.0	23.4
31	9.0	1530.0	1187.0	77.58	343.0	22.42
32	12.0	1528.0	1158.0	75.79	370.0	24.21
33	44.0	1497.0	1117.0	74.62	380.0	25.38
34	8.0	1493.0	1142.0	76.49	351.0	23.51
35	37.0	1452.0	1077.0	74.17	375.0	25.83
36	40.0	1394.0	1048.0	75.18	346.0	24.82
37	42.0	1393.0	1132.0	81.26	261.0	18.74
38	53.0	1385.0	937.0	67.65	448.0	32.35
39	45.0	1277.0	1012.0	79.25	265.0	20.75
40	6.0	1257.0	972.0	77.33	285.0	22.67
41	47.0	1249.0	985.0	78.86	264.0	21.14
42	48.0	1157.0	910.0	78.65	247.0	21.35
43	49.0	1143.0	857.0	74.98	286.0	25.02
44	46.0	1102.0	858.0	77.86	244.0	22.14
45	4.0	1085.0	869.0	80.09	216.0	19.91
46	5.0	1064.0	825.0	77.54	239.0	22.46
47	52.0	1030.0	760.0	73.79	270.0	26.21
48	50.0	1012.0	736.0	72.73	276.0	27.27
49	3.0	1006.0	782.0	77.73	224.0	22.27
50	2.0	915.0	704.0	76.94	211.0	23.06
51	1.0	844.0	601.0	71.21	243.0	28.79
52	51.0	748.0	589.0	78.74	159.0	21.26

Summary Statistics for arrival_date_week_number

	arrival_date_week_number
count	85657.0
mean	26.828525397807535
std	13.62149502802063
min	1.0
25%	16.0
50%	27.0
75%	37.0
max	53.0

Countplot for arrival_date_day_of_month



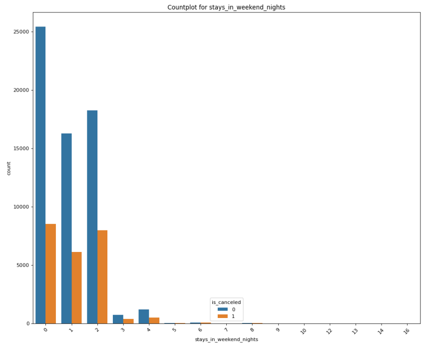
Proportion Table for arrival_date_day_of_month

	arrival_date_day_of_month	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	17.0	2964.0	2098.0	70.78	866.0	29.22
1	2.0	2959.0	2162.0	73.07	797.0	26.93
2	26.0	2947.0	2075.0	70.41	872.0	29.59
3	5.0	2919.0	2145.0	73.48	774.0	26.52
4	19.0	2911.0	2170.0	74.54	741.0	25.46
5	16.0	2896.0	2077.0	71.72	819.0	28.28
6	18.0	2880.0	2105.0	73.09	775.0	26.91
7	12.0	2875.0	2125.0	73.91	750.0	26.09
8	28.0	2872.0	2042.0	71.1	830.0	28.9
9	11.0	2857.0	2062.0	72.17	795.0	27.83
10	20.0	2846.0	2106.0	74.0	740.0	26.0
11	27.0	2833.0	1998.0	70.53	835.0	29.47
12	29.0	2823.0	1992.0	70.56	831.0	29.44
13	9.0	2823.0	2080.0	73.68	743.0	26.32
14	15.0	2811.0	2028.0	72.15	783.0	27.85
15	25.0	2789.0	2055.0	73.68	734.0	26.32
16	21.0	2774.0	2036.0	73.4	738.0	26.6
17	3.0	2774.0	1989.0	71.7	785.0	28.3
18	13.0	2759.0	2014.0	73.0	745.0	27.0
19	8.0	2749.0	1949.0	70.9	800.0	29.1
20	4.0	2748.0	1996.0	72.63	752.0	27.37
21	6.0	2738.0	2022.0	73.85	716.0	26.15
22	23.0	2726.0	2021.0	74.14	705.0	25.86
23	10.0	2723.0	2000.0	73.45	723.0	26.55
24	24.0	2717.0	1939.0	71.37	778.0	28.63
25	1.0	2707.0	1901.0	70.23	806.0	29.77
26	30.0	2701.0	1932.0	71.53	769.0	28.47
27	7.0	2654.0	1898.0	71.51	756.0	28.49
28	14.0	2623.0	1902.0	72.51	721.0	27.49
29	22.0	2553.0	1814.0	71.05	739.0	28.95
30	31.0	1706.0	1260.0	73.86	446.0	26.14

Summary Statistics for arrival_date_day_of_month

	arrival_date_day_of_month
count	85657.0
mean	15.818975682080858
std	8.832849399044187
min	1.0
25%	8.0
50%	16.0
75%	23.0
max	31.0

Countplot for stays_in_weekend_nights



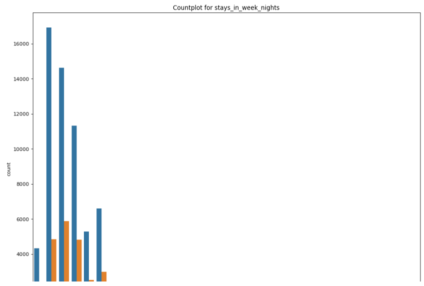
Proportion Table for stays_in_weekend_nights

	stays_in_weekend_nights	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	0.0	33909.0	25400.0	74.91	8509.0	25.09
1	2.0	26241.0	18251.0	69.55	7990.0	30.45
2	1.0	22390.0	16269.0	72.66	6121.0	27.34
3	4.0	1710.0	1193.0	69.77	517.0	30.23
4	3.0	1133.0	749.0	66.11	384.0	33.89
5	6.0	111.0	56.0	50.45	55.0	49.55
6	5.0	66.0	31.0	46.97	35.0	53.03
7	8.0	57.0	24.0	42.11	33.0	57.89
8	7.0	15.0	4.0	26.67	11.0	73.33
9	9.0	8.0	3.0	37.5	5.0	62.5
10	10.0	7.0	5.0	71.43	2.0	28.57
11	12.0	5.0	4.0	80.0	1.0	20.0
12	13.0	2.0	2.0	100.0	0.0	0.0
13	16.0	2.0	1.0	50.0	1.0	50.0
14	14.0	1.0	1.0	100.0	0.0	0.0

Summary Statistics for stays_in_weekend_nights

	stays_in_weekend_nights
count	85657.0
mean	1.01500169279802
std	1.0239481069387257
min	0.0
25%	0.0
50%	1.0
75%	2.0
max	16.0

Countplot for stays_in_week_nights



Proportion Table for stays_in_week_nights

	stays_in_week_nights	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	1.0	21764.0	16918.0	77.73	4846.0	22.27
1	2.0	20499.0	14631.0	71.37	5868.0	28.63
2	3.0	16135.0	11327.0	70.2	4808.0	29.8
3	5.0	9605.0	6612.0	68.84	2993.0	31.16
4	4.0	7821.0	5287.0	67.6	2534.0	32.4
5	0.0	5413.0	4323.0	79.86	1090.0	20.14
6	6.0	1363.0	844.0	61.92	519.0	38.08
7	10.0	955.0	697.0	72.98	258.0	27.02
8	7.0	954.0	660.0	69.18	294.0	30.82
9	8.0	608.0	410.0	67.43	198.0	32.57
10	9.0	216.0	132.0	61.11	84.0	38.89
11	15.0	67.0	39.0	58.21	28.0	41.79
12	11.0	49.0	21.0	42.86	28.0	57.14
13	12.0	40.0	18.0	45.0	22.0	55.0
14	20.0	39.0	22.0	56.41	17.0	43.59
15	13.0	27.0	16.0	59.26	11.0	40.74
16	14.0	23.0	4.0	17.39	19.0	82.61
17	19.0	20.0	7.0	35.0	13.0	65.0
18	21.0	14.0	4.0	28.57	10.0	71.43
19	16.0	11.0	4.0	36.36	7.0	63.64
20	22.0	6.0	0.0	0.0	6.0	100.0
21	25.0	6.0	6.0	100.0	0.0	0.0
22	18.0	6.0	1.0	16.67	5.0	83.33
23	30.0	4.0	3.0	75.0	1.0	25.0
24	17.0	4.0	2.0	50.0	2.0	50.0
25	24.0	2.0	0.0	0.0	2.0	100.0
26	40.0	2.0	1.0	50.0	1.0	50.0

Summary Statistics for stays_in_week_nights

	stays_in_week_nights
count	85657.0
mean	2.648329967194742
std	2.022815262654507
min	0.0
25%	1.0
50%	2.0
75%	4.0

There are no obvious issues which arise from the summary statistics for the date & time features. Issues we are looking for include years outside the range; week numbers greater than 53 and less than 1; days of the month outside the 1-31 range; and a negative number of weekend or weekday nights.

From these countplots, proportion tables and summary statistics we draw the following conclusions about each feature:

- ***'arrival_date_year'***: The countplot shows that the number of hotel bookings in our dataset peaked in 2016 - this is likely because the data spans July 2015 to August 2017, hence 2016 is the only year we have complete data for. From the proportion table we see that the percentage of cancelled bookings steadily rose between 2015 and 2017. This could imply that the arrival year might have a large impact in whether a booking is cancelled.

- ***'arrival_date_month'***: The count plot for arrival month shows a seasonal trend in the number of hotel bookings. As we might expect, during the summer months the number of hotel bookings peaks and then drop to its lowest levels in December and January. At this point, we note again that the dataset we have starts at the start of July and ends at the end of August. Therefore, we have three years of data for these two months, and the largest bins in the histogram should correspond to July and August. However, we argue that this does not mean our observation of a seasonal trend is invalid because we can see the trend appear in the months surrounding these. The same seasonal trend does not appear in the percentage of booking that are canceled, implying that there might not be a strong relationship between cancellations and month of arrival.

- ***'arrival_date_week_number'***: We can draw the same conclusions about the week number as we did from month of arrival. We would not expect week number of arrival to be an important feature in our classification.

- ***'arrival_date_day_of_month'***: The constant level of the countplot implies there does not appear to be any relationship between day of arrival and both cancelled and not cancelled bookings. One interesting observation we can make from the plot is that booking numbers tail off on the 31st day of the month. This is to be expected because only seven months a year have a 31st day.

- ***'stays_in_weekend_nights'***: The countplot for number of weekend nights stayed in the hotel peaks at 0 nights, drops for 1 night, and then form a secondary lower peak on 2 nights. This could be explainied by customers preferring to stay for the whole weekend if they can. After 2 nights the distribution falls to close to zero for the remaining number of weekend nights. There does not appear to be a large difference in the proportions of bookings that are cancelled a the number of weekend nights stayed, again this implies that this feature will not have a large effect on cancellations.

- ***'stays_in_week_nights'***: Finally, the countplot for weekday nights stayed shows that the most common number of week nights stayed in the hotel was 2, with the number of bookings quickly tailing off after this. There is a secondary peak at 5 week days most likely to those who stay for a whole week. On inspection of the proportion table, we see that the proportion of bookings cancelled peaks after 10 weekday nights are stayed, with values exceeding 80%. The number of observations with these number of weekday nights is very low, all much less than 100 unique bookings. Therefore, we would expect these small number of observations to skew our model. Therefore we decide to remove observations where the number of weekday nights stayed is greater than or to 10.

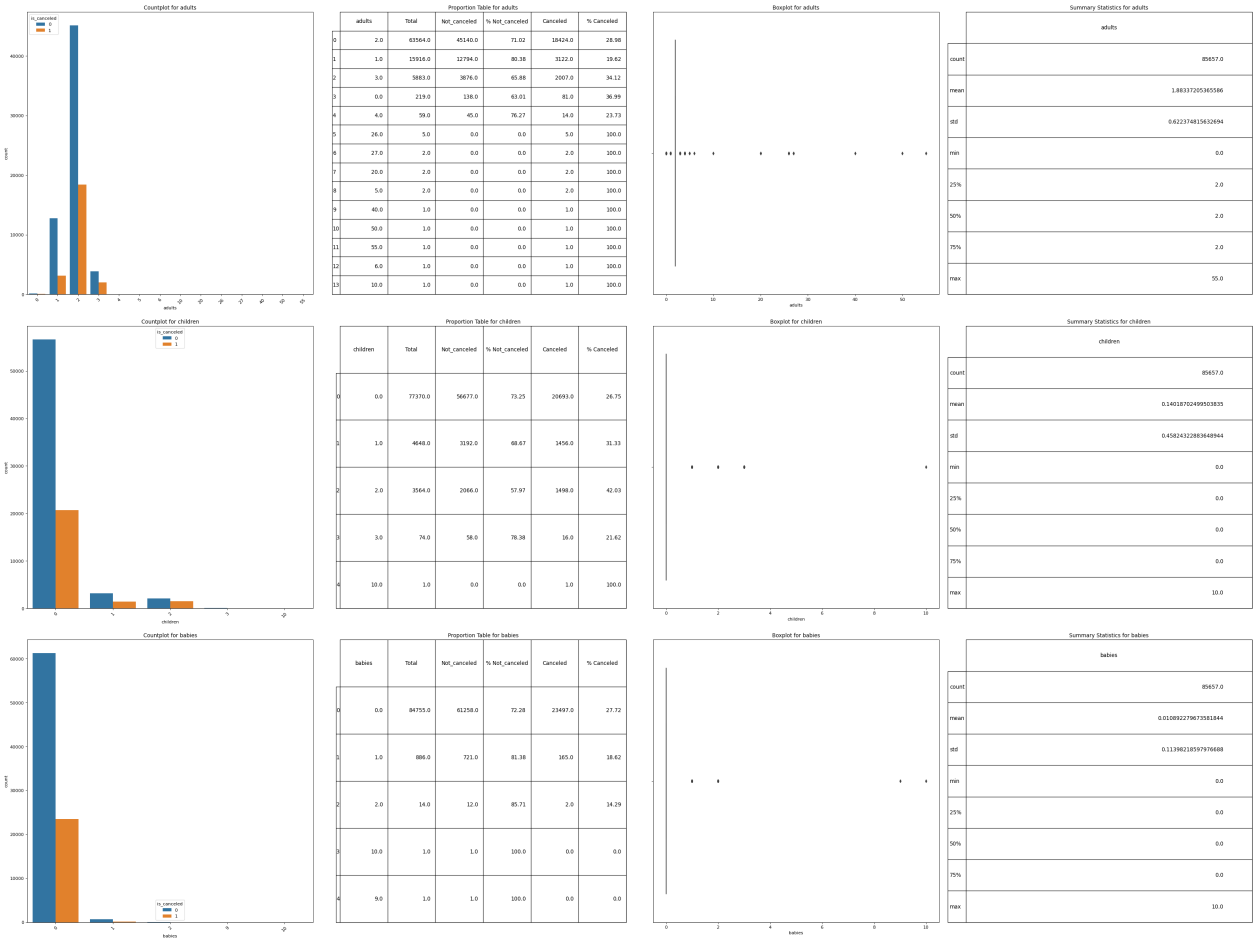
In [15]:
```
# Week nights stayed more than 10
week_nights = (df.stays_in_week_nights > 10)
```

2.4.3 Categorical Features: *Personal Information*

Here we consider the features 'adults', 'children' and 'babies'.

In [16]:
```
# Personal information
personal_features = ['adults', 'children','babies']

fig, axes = plt.subplots(len(personal_features), 4, figsize = (40,10*len(personal_features)))

for i, feature in enumerate(personal_features):
    ## Countplot ##
    sns.countplot(data = df, x = feature, hue = 'is_canceled', ax = axes[i][0])
    axes[i][0].tick_params(axis='x', rotation=45)
    axes[i][0].set_title(f'Countplot for {feature}')

    ## Proportion Table ##
    prop_table = proportion_table(df, feature, feature, show = False)
    axes[i][1].axis('off')
    table = axes[i][1].table(cellText=prop_table.values, colLabels=prop_table.columns, rowLabels=prop_table.index, b
    table.auto_set_font_size(False)
    table.set_fontsize(12)  # Set the font size to 9
    axes[i][1].set_title(f"Proportion Table for {feature}")

    ## Boxplot ##
```

```
    sns.boxplot(data = df, x = feature, ax = axes[i][2])
    axes[i][2].set_title(f"Boxplot for {feature}")

    ## Summary Statistics ##
    sum_stats = df[feature].describe().to_frame()
    axes[i][3].axis('off')
    table = axes[i][3].table(cellText=sum_stats.values, colLabels=sum_stats.columns, rowLabels=sum_stats.index, bbox
    table.auto_set_font_size(False)
    table.set_fontsize(12)  # Set the font size to 9
    axes[i][3].set_title(f"Summary Statistics for {feature}")

plt.tight_layout(pad = 2.0)
```



From these countplots, proportion tables, boxplots and summary statistics we draw the following conclusions about each feature:

- ***'adults':*** We see that perhaps unsurprisingly the most common number of adults in a booking is 2, with respect to both cancelled and not cancelled bookings. This is corroborated by the summary statistics and the boxplot which show that the median number of adults is 2 as well as first and third quartile. The summary statistics also show that the max value for number if adults is 55 this sounds like an erroneous outlier. Indeed, on inspection of the boxplot and the proportion table we see that there are multiple outliers. For this reason we decide to remove observations with 10 or more adults as they may have negative consequences on our model. These outliers that we remove only account for 13 observations out of a total of around 80000.

- ***'children'***: For the 'children' feature we see that almost all observations have between 0 and 3 children apart from one observation with 10 children. We believe this observation to be an outlier because it is over 20 standard deviations away from the mean, hence this observation will also be removed.

- ***'babies'***: Finally, we examine the 'babies' feature. The barchart shows that the majority of bookings, both cancelled and not cancelled, have 0 babies. The boxplot does show that there are two observations with 9 and 10 observations respectively. We treat these as outliers and remove them from the dataset.

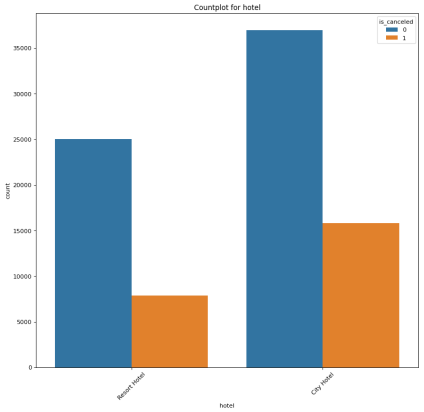
In [17]:
```
# 10 or more adults
adults = (df.adults > 10)

# 10 children
children = (df.children == 10)

# 9 or more babies
babies = (df.babies >= 9)
```

2.4.4 Categorical Features: *Booking Information*

The features we consider here are 'hotel', 'meal', 'market_segment', 'distribution_channel', 'is_repeated_guest', 'reserved_room_type', 'assigned_room_type', 'booking_changes', 'deposit_type', 'agent', 'customer_type' and 'total_of_special_requests'.

In [18]:
```
# Booking Information
booking_features = ['hotel', 'meal', 'market_segment', 'distribution_channel', 'is_repeated_guest', 'reserved_room_t
                    'deposit_type', 'agent', 'customer_type', 'total_of_special_requests']

fig, axes = plt.subplots(len(booking_features), 3, figsize = (30,10*len(booking_features)))

for i, feature in enumerate(booking_features):
    ## Countplot ##
    sns.countplot(data = df, x = feature, hue = 'is_canceled', ax = axes[i][0])
    axes[i][0].tick_params(axis='x', rotation=45)
    axes[i][0].set_title(f'Countplot for {feature}')

    ## Proportion Table ##
    prop_table = proportion_table(df, feature, feature, show = False)
    axes[i][1].axis('off')
    table = axes[i][1].table(cellText=prop_table.values, colLabels=prop_table.columns, rowLabels=prop_table.index, b
    table.auto_set_font_size(False)
    table.set_fontsize(9)  # Set the font size to 9
    axes[i][1].set_title(f"Proportion Table for {feature}")

    ## Summary Statistics ##
    sum_stats = df[feature].describe().to_frame()
    axes[i][2].axis('off')
    table = axes[i][2].table(cellText=sum_stats.values, colLabels=sum_stats.columns, rowLabels=sum_stats.index, bbox
    table.auto_set_font_size(False)
    table.set_fontsize(9)  # Set the font size to 9
    axes[i][2].set_title(f"Summary Statistics for {feature}")

plt.tight_layout(pad = 2.0)
```

Countplot for hotel



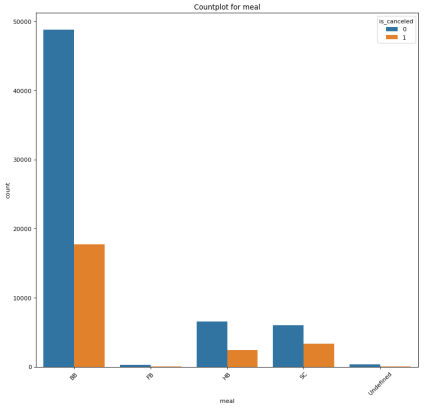
Proportion Table for hotel

	hotel	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	City Hotel	52775	36961	70.04	15814	29.96
1	Resort Hotel	32882	25032	76.13	7850	23.87

Summary Statistics for hotel

	hotel
count	85657
unique	2
top	City Hotel
freq	52775

Countplot for meal



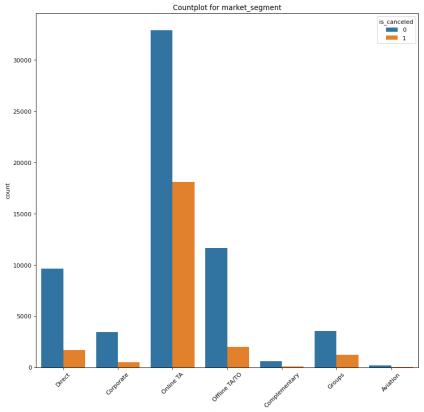
Proportion Table for meal

	meal	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	BB	66567	48796	73.3	17771	26.7
1	SC	9317	6001	64.41	3316	35.59
2	HB	8951	6541	73.08	2410	26.92
3	Undefined	469	395	84.22	74	15.78
4	FB	353	260	73.65	93	26.35

Summary Statistics for meal

	meal
count	85657
unique	5
top	BB
freq	66567

Countplot for market_segment



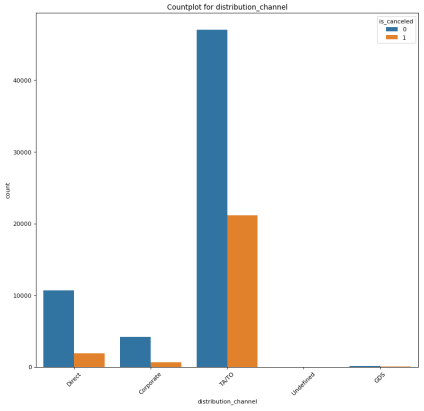
Proportion Table for market_segment

	market_segment	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	Online TA	51020	32915	64.51	18105	35.49
1	Offline TA/TO	13643	11642	85.33	2001	14.67
2	Direct	11343	9646	85.04	1697	14.96
3	Groups	4799	3546	73.89	1253	26.11
4	Corporate	3953	3471	87.81	482	12.19
5	Complementary	679	597	87.92	82	12.08
6	Aviation	220	176	80.0	44	20.0

Summary Statistics for market_segment

	market_segment
count	85657
unique	7
top	Online TA
freq	51020

Countplot for distribution_channel



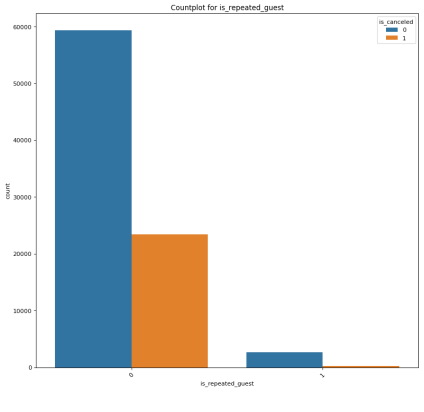
Proportion Table for distribution_channel

	distribution_channel	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	TA/TO	68175	47032	68.99	21143.0	31.01
1	Direct	12516	10642	85.03	1874.0	14.97
2	Corporate	4790	4177	87.2	613.0	12.8
3	GDS	175	141	80.57	34.0	19.43
4	Undefined	1	1	100.0	0.0	0.0

Summary Statistics for distribution_channel

	distribution_channel
count	85657
unique	5
top	TA/TO
freq	68175

Countplot for is_repeated_guest



Proportion Table for is_repeated_guest

	is_repeated_guest	Total	Not_canceled	% Not_canceled	Canceled	% Canceled
0	0.0	82786.0	59350.0	71.69	23436.0	28.31
1	1.0	2871.0	2643.0	92.06	228.0	7.94

Summary Statistics for is_repeated_guest

	is_repeated_guest
count	85657.0
mean	0.033517402796 19879
std	0.179984340505 0949
min	0.0
25%	0.0
50%	0.0
75%	0.0
max	1.0

From these countplots, proportion tables, boxplots and summary statistics we draw the following conclusions about each feature:

- ***'hotel'***: The barchart shows that over the period of interest more bookings were made to the city hotel then the resort hotel. The proportion table shows us that the city hotel had a higher percentage of cancellations with 29.7% of bookings being cancelled compared to the resort hotel where only 23.4% of bookings were cancelled. In this feature there is no indication of erroneous entries or outliers, all observations were made to one of the two types of hotel.

- ***'meal'***: This feature is split into 5 categories, these are:

    - Undefined/ SC – 'self catered' no meal package provided
    - BB – 'bed and breakfast', one meal (breakfast) has been provided
    - HB – 'half board', two meals provided, usually breakfast and dinner
    - FB – 'full board', three meals provided.

    Since undefined and SC correspond to the same category we will change all observations with undefined meals to SC as this will stop us having two labels corresponding to the same thing. The proportion table shows that the vast majority of bookings are for bed and breakfast. The meal category with the highest cancellation percentage is self catered with 35% of booking cancelled even when we combine with the undefined category it will still have greater than 30% cancellations which is much more than all the other categories in the feature, which are all around 26%. Therefore, we would expect meal to be a useful feature for our classification model.

- ***'market_segment'***: There are 7 categories: online TA, offline TA/TO, Direct, Groups, Corporate, Complementary and Aviation. We know that TA and TO refer to travel agent and tour operator respectively. We see that the more than half of all bookings are through online travel agents which isn't surprising considering that our data was gathered between 2015 and 2017 when online travel agents were well-established and mainstream. Online travel agents also have a cancellation rate of 35% which is considerably higher than all other categories. For this reason we would expect 'market_segment' to be an important feature in our classification.

- ***'distribution_channel'***: The barchart shows the most common distribution channel is TA/TO, this is also the category with the greatest proportion of cancellations at 30%. All other distribution channels have cancellation rates between 12-18%. Other points of interest in this feature are the 'GDS' feature which stands for 'Global Distribution System' and refers to an online network that connect travel agents directly with the central reservations systems for airlines and hotels. Furthermore, the proportion table shows that there is one value with undefined distribution channel. We will remove this observation as we lack enough information to robustly impute it.

- ***'is_repeated_guest'***: Examining the barchart we see that most of the bookings made are from non-repeat customers. The proportions show us that 28% of non-repeat customers cancel their bookings but only 7% of repeat customers cancel their

bookings. This disparity in cancelling proportion implies that this feature might be useful for our classification. We can also see from the table that all values in this feature are either 0 or 1 so there are no erroneous values we must consider.

- ***'reserved_room_type'*** & ***''assigned_room_type'***: We see that room type A is the most common room type reserved and assigned. For anonymity purposes we do not know what the letters correspond to so there is not many more conclusions we can draw from these features. We can note that neither feature has values that do not fit into the standard categories, so no additional data cleaning is necessary.

- ***'deposit_type'***: Most bookings are made with no deposit paid. Interestingly, the deposit type with the largest percentage of cancellations is non-refund with 94%.

- ***'agent'***: This feature shows a clear difference in the proportion of cancellation when booked through an agent compared to those not booked through an agent. Surprisingly 29% of bookings made through an agent were cancelled whereas only 12% of those not made through an agent were cancelled. There are no erroneous values in this feature - all observations are either 0 or 1.

- ***'customer_type'***: For 'customer_type' by far the greatest, both in terms of absolute number of bookings and proportion of cancellations, is the transient category. The other three categories only have between around 10-16% cancellation rates. Again there are no erroneous values in this feature to consider.

- ***'total_of_special_requests'***: We can see a clear trend that as the number of sepcial requests increases the number of bookings in that category decreases as well as the proportion of cancellations. Perhaps implying that this is a useful feature for classifying our data. Again we have that there are no erroneous values in this feature to consider.

In [19]:
```
# Rename undefined meals to SC
df['meal'].replace({'Undefined': 'SC'}, inplace=True)

# Undefined distribution channel
undef_dc = (df.distribution_channel == 'undefined')
```

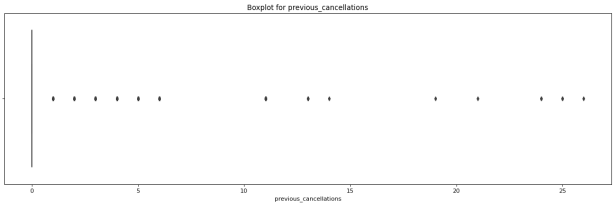
2.4.3 Categorical Features: *Other Information*

In this section we will consider the remaining features: 'previous_cancellations', 'previous_bookings_not_canceled', 'booking_changes', 'days_in_waiting_list', 'required_car_parking_spaces' and 'country'. These are the features we have identified as requiring a closer examination, especially in regard to outliers.

In [20]:
```
other_features = ['previous_cancellations', 'previous_bookings_not_canceled', 'booking_changes', 'days_in_waiting_li

#, 'days_in_waiting_list'
fig, axes = plt.subplots(len(other_features), 2, figsize = (30,5*len(other_features)))

for i, feature in enumerate(other_features):

    ## Boxplot ##
    sns.boxplot(data = df, x = feature, ax = axes[i][0])
    axes[i][0].set_title(f"Boxplot for {feature}")

    ## Summary Statistics ##
    sum_stats = df[feature].describe().to_frame()
    axes[i][1].axis('off')
    table = axes[i][1].table(cellText=sum_stats.values, colLabels=sum_stats.columns, rowLabels=sum_stats.index, bbox
    table.auto_set_font_size(False)
    table.set_fontsize(9)  # Set the font size to 9
    axes[i][1].set_title(f"Summary Statistics for {feature}")

plt.tight_layout(pad = 2.0)

display((df[['days_in_waiting_list']]))
```

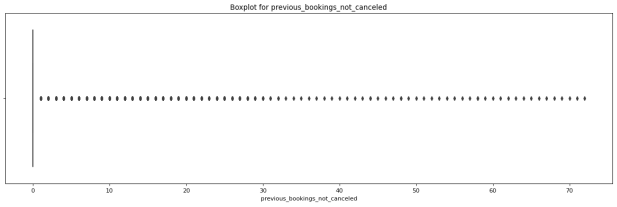
	days_in_waiting_list
2	0
3	0
4	0
6	0
7	0
...	...
119385	0
119386	0
119387	0
119388	0
119389	0

85657 rows × 1 columns



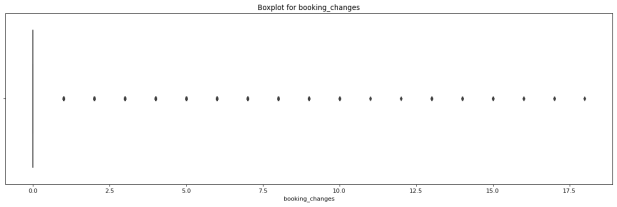
Summary Statistics for previous_cancellations

	previous_cancellations
count	85657.0
mean	0.030295247323627 96
std	0.371763984009505775
min	0.0
25%	0.0
50%	0.0
75%	0.0
max	26.0



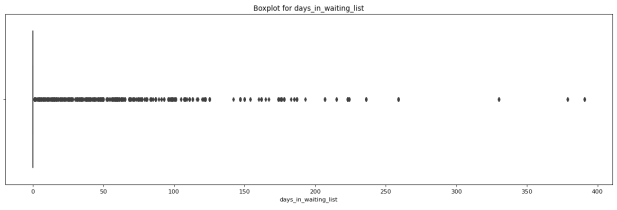
Summary Statistics for previous_bookings_not_canceled

	previous_bookings_not_canceled
count	85657.0
mean	0.17819909639609138
std	1.728359806202 9015
min	0.0
25%	0.0
50%	0.0
75%	0.0
max	72.0



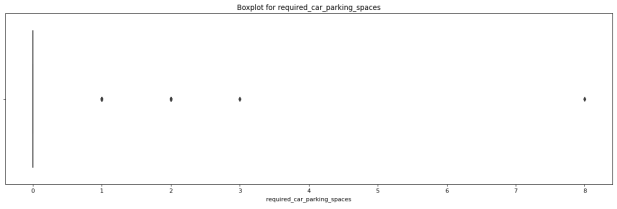
Summary Statistics for booking_changes

	booking_changes
count	85657.0
mean	0.26841939362807476
std	0.7110295104070028
min	0.0
25%	0.0
50%	0.0
75%	0.0
max	18.0



Summary Statistics for days_in_waiting_list

	days_in_waiting_list
count	85657.0
mean	0.7384802176121041
std	9.919198572434926
min	0.0
25%	0.0
50%	0.0
75%	0.0
max	391.0



Summary Statistics for required_car_parking_spaces

	required_car_parking_spaces
count	85657.0
mean	0.08305771157056633
std	0.2809411355801440 5
min	0.0
25%	0.0
50%	0.0
75%	0.0
max	8.0

From the bocplots and summary statistics we can make the following conclusions:

- '***previous_cancellations***': The summary statistics show that the mean is roughly 0.03 with standard deviation around 0.4. However, the maximum value is 26 which this suggests the presence of outliers. The boxplot clearly shows that there are indeed outliers. Of note is the fact that the outliers greater than 10 are more spaced out then the ones less then 10. We therefore decide that 10 is reasonable cutoff for our outliers and remove observations with 10 or more previous cancellations.
- '***previous_bookings_not_canceled***' & '***booking_changes***': The boxplot shows that the outlier are evenly spaced, appearing to cover all values between 1 and 72. Therefore, we decide not to remove any observations based on this feature because there is no obvious break in the outliers. We draw the exact same conclusions for the 'booking_changes' feature.
- '***days_in_waiting_list***': There are a large number of observations with 0 days in the waiting list, indicating they got their booking was confirmed on the same day. Yet there are also a large number of observations where the number of days in the

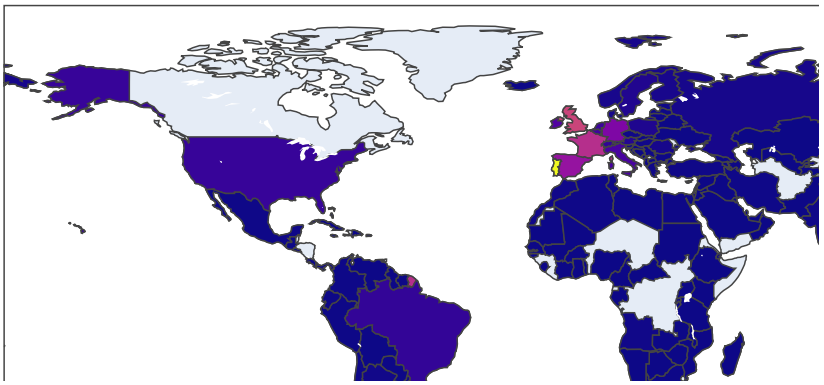
waiting list is much larger. We do not remove any of these values because although it seems unlikely someone would be in the waiting list for 400 days it is possible.

- ***'required_car_parking_spaces'***: Finally, for 'required_car_parking_spaces' we can clearly see an outlying value at 8 which we will remove from the dataset.

In [21]:
```
# Previous cancellations greater then 10
cancellations = (df.previous_cancellations >= 10)

# 8 Parking spaces
car_spaces = (df.required_car_parking_spaces == 8)
```

Our final visualisation in this section is for the ***'country'*** feature. Here we use a heatmap imposed over a geographical map of the wolrd to illustrate where guests are coming from.

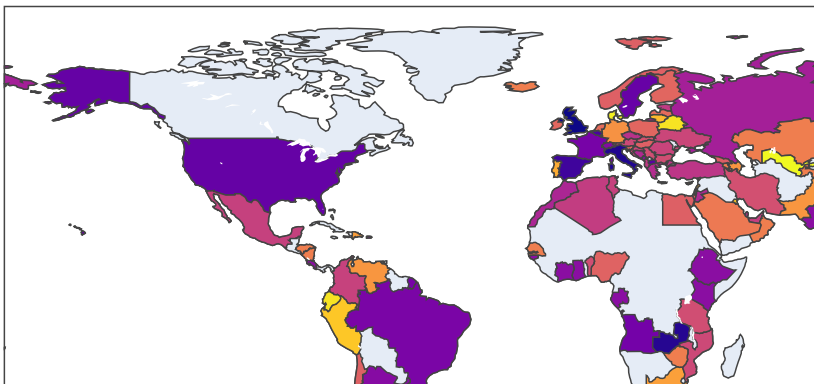
In [22]:
```
# get number of bookings from each country
guest_city = df[df['is_canceled'] == 0]['country'].value_counts().reset_index()
guest_city.columns = ['Country', 'No of guests']
guest_city

basemap = folium.Map()
# generate choropleth map
guests_map = px.choropleth(guest_city, locations = guest_city['Country'],
                           color = guest_city['No of guests'], hover_name = guest_city['Country'])
guests_map.show()
```



We can see that most of the countries are either light blue, indicating no one from that country stayed in the hotel, or dark blue indicating very few people did. Most of the light blue countries are generally considered not wealthy, for example these include Yemen, Somalia and Afghanistan. An interesting outlier in this observation is Canada, it is the only G20 country from which no one booked either hotel. Instead, most of the bookings are concentrated around the wealthier countries of western Europe, with almost 17000 bookings coming from Portugal and 8400 coming from the UK.

Below, we map the proportion of customers who cancel for each country.

In [23]:
```
# get proportion of cancellation from each country

guest_city = df[df['is_canceled'] == 1]['country'].value_counts().reset_index()
guest_city['country'] = guest_city['country']/df['country'].value_counts().reset_index()['country']
guest_city.columns = ['Country', 'Prop cancel']
guest_city

basemap = folium.Map()
# generate choropleth map
guests_map = px.choropleth(guest_city, locations = guest_city['Country'],
                           color = guest_city['Prop cancel'], hover_name = guest_city['Country'])
guests_map.show()
```



We now remove all the outliers that we have discussed in this section in the following code block:

In [24]:
```
# Delete these anomolous observations from the dataframe
bool_series = week_nights|adults|children|babies|undef_dc|cancellations|car_spaces
df = df[~bool_series.reindex(df.index)]
```

2.5 Data Encoding

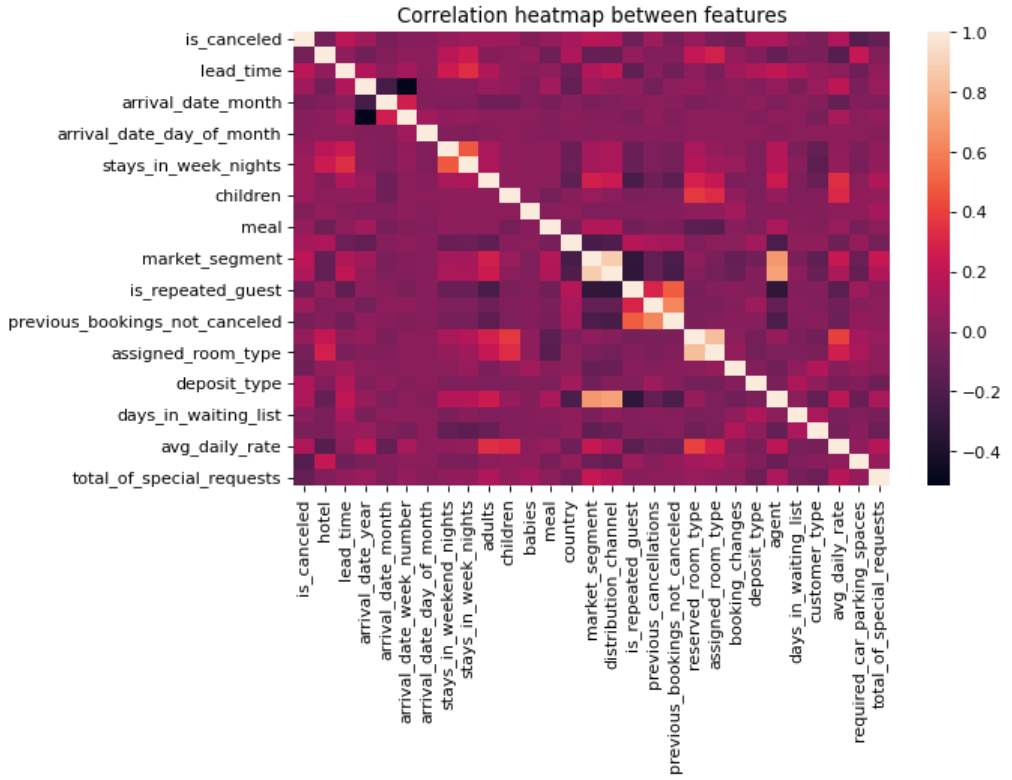
To be able to effectively use the dataset in a model we need to encode varibles which are strings as numbers. For example could look to encode a city hotel as 0 and resort hotel as 1. Since we have quite a few features still in the dataframe we will use a Label Encoder rather than a one hot encoding, and make note of the encoder for inversing encoded variables after using a model.

In [25]:
```
from sklearn.preprocessing import LabelEncoder
from collections import defaultdict

le_dict = defaultdict(LabelEncoder)

# apply label encoder to all columns
encoded_df = df.apply(lambda x: le_dict[x.name].fit_transform(x))

# inverse label encoder, for later use
df = encoded_df.apply(lambda x: le_dict[x.name].inverse_transform(x))

# https://stackoverflow.com/questions/24458645/label-encoding-across-multiple-columns-in-scikit-learn
```

2.6 Correlation

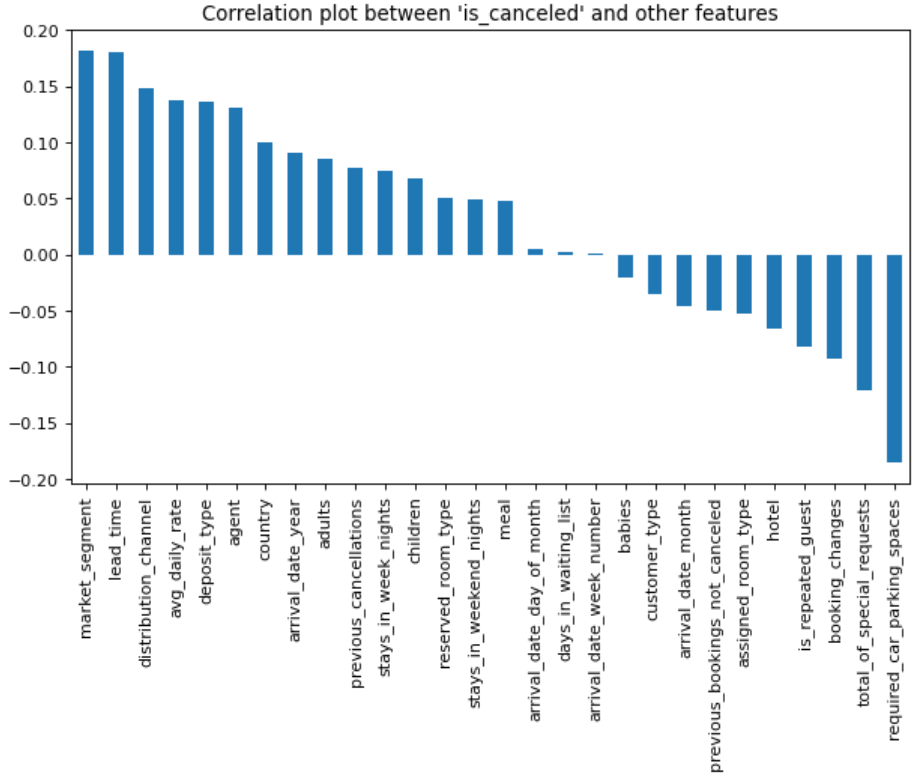
In this section we will examine any correlation between features as well as identifying the features with the largest correlation to the response variable.

In [26]:
```
# Correlation heatmap
corr = encoded_df.corr()
plt.figure()
sns.heatmap(corr)
plt.title('Correlation heatmap between features')
plt.show()
```



In general, there is very little correlation between all the features in our dataset, and noteably no obvious, glaring, correlation between features and the response variable 'is_canceled'. This gives us motivation to examine, in more detail, the correlation between our features and our response variable.

In [27]:
```
# correlation plot between reponse variable and features, as a bar graph
fig, ax = plt.subplots(figsize=(9,5))
encoded_df.corr()["is_canceled"].sort_values(ascending=False)[1:].plot(kind="bar", ax=ax)
plt.title("Correlation plot between 'is_canceled' and other features")
plt.show()
```



The features with the strongest positive correltations to the response are: 'market_segment', 'lead_time' and 'distribution_channel'. The features with the strongest negative correlations to the response are: 'required_car_parking_spaces', 'total_of_special_requests' and 'booking_changes'. We would thus expect these features to play an important part in our classification model.

Note the features 'arrival_date_day_of_month', 'days_in_waiting_list' and 'days_in_waiting_list' have almost zero correlation with the response variable. We suspect that it is possible to drop these features from our dataset. Before we do so we need to justify our

decision with a hypothesis test. We use the 'pearsonr' function from scipy, this returns the Pearson correlation coefficient between the response and a feature. It also returns the p-value associated with a test of the null hypothesis that the distributions underlying the samples are uncorrelated and normally distributed. The p-value roughly indicates the probability of an uncorrelated system producing datasets that have a Pearson correlation at least as extreme as the one computed from these datasets. Therefore we will remove features with p-values above a decided threshold. The threshold we chose is 0.05 as this is a standard choice in statistics.

In [28]:
```
from scipy.stats import pearsonr
# Create empty list to store features to drop
drop_features = []

# Iterate over all features and perform hypothesis test
for feature in encoded_df.columns:
    cor, p_value = pearsonr(encoded_df[feature], encoded_df['is_canceled'])
    #print(feature, cor, p_value)
    if p_value > 0.05:
        drop_features.append(feature)

# Drop features with low correlation to response variable in both Dataframes
encoded_df = encoded_df.drop(drop_features, axis=1)
df = df.drop(drop_features, axis=1)
print(f'The features we will drop are: {drop_features}')
```

```
The features we will drop are: ['arrival_date_week_number', 'arrival_date_day_of_month', 'days_in_waiting_list']
```

We see that, as expected, the features that are insignificantly correlated with our response are 'arrival_date_week_number', 'arrival_date_day_of_month' and 'days_in_waiting_list'. These features have been dropped from both the encoded and unencoded dataframes.

---

3. Model Fitting and Tuning

Now that we have explored our data, we turn to creating a model to predict hotel booking cancellations. Several different models were tested, but the best was a gradient boosting forest method, which will be presented in this section.

In [29]:
```
# Model Evaluation Function

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def eval(y_true, y_pred, y_prob, confusion=True, AUC = True, ROCplot=True, precision=True, f1 = True, recall = True,
    ''' Function to evaluate a model. Takes inputs true y, predicted y and probabilities
    and prints evaluation metrics. Metrics are confusion matrix, AUC curve, ROC plot,
    precision value, f1 value, recall value and accuracy value. These can be individually
    toggled off manually by setting them to False.'''
    if accuracy:
        print("Accuracy:", accuracy_score(y_true, y_pred))
    if precision:
        print("Precision:", precision_score(y_true, y_pred))
    if recall:
        print("Recall:", recall_score(y_true, y_pred))
    if f1:
        print("F1 Score:", f1_score(y_true, y_pred))
    if confusion:
        print('Confusion Matrix: \n', sklearn.metrics.confusion_matrix(y_true, y_pred))
    if AUC:
        print('AUC: ', sklearn.metrics.roc_auc_score(y_true, y_pred))
    if ROCplot:
        fpr, tpr, thresholds = sklearn.metrics.roc_curve(y_true, y_prob)
        # calculate AUC score
        roc_auc = sklearn.metrics.auc(fpr, tpr)

        # plot ROC curve
        plt.figure(figsize=(5,5))
        plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' % roc_auc)
        plt.plot([0, 1], [0, 1], 'k--')
        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')
        plt.legend(loc="lower right")
        plt.show()
```

In [30]:
```
from IPython.display import clear_output
import ipywidgets as widgets

def button(function):
    '''Function that makes button to run function'''
    print('Warning, cell takes a wee while to run. Click to proceed.')
    def on_click(arg):
```

```
        clear_output(wait=False)
        print(f'Running {function.__name__}...')
    def after_click(arg):
        print(f'Finished {function.__name__}...')
    the_button = widgets.Button(description = f'Click to run {function.__name__}.', layout = {'width':'auto'})
    the_button.on_click(on_click)
    the_button.on_click(function)
    the_button.on_click(after_click)
    display(the_button)
```

Here, we define a function to evaluate candidate models for convenience. We used it extensively to narrow down our models before settling on an ensemble approach, specifially Gradient Boosting Classifier. The function takes the test and predicted cancellation data for a model, and returns several of the most commonly used metrics for evaluating a binary model, including AUC and confusion matrices.

3.1 Training & Test Data

As discussed in section 2, we have a very large number of data points which may lead to overfitting and cost issues. To counteract this we will only use a sample of the overall dataset. Even though we have a 3:1 split of not cancelled to cancelled datapoints we stratify over the response variable to ensure fairness. Because model selection and hyperparameter tuning (especially cross validation) can be quite expensive, we find the best model and hyperparameters using a representative sample of the data. Later, we return to the full dataset to train and test our chosen model.

In [31]:
```
from sklearn.model_selection import train_test_split

sample = encoded_df.sample(frac=0.1, random_state=1)

X = sample.drop('is_canceled', axis = 1)
y = sample['is_canceled']

# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state=42, stratify = y)
```

3.2 Gradient boosting 🌲

A crucial part of model selection is hyperparameter tuning. GridSearchCV lets us specify a metric, and then fit several models to determine which combination from the predefined parameter ranges yield the best value for that metric. Below is the parameter cross validation for the HistGradientBoostingClassifer model. We chose the parameters to search over based on conventional choices in literature and documentation, except for max_depth. Because we would like to have high interpretability, we limit the depth of the trees in the model. This very slightly impacts the performance of the model but the conclusions remain the same, and the model is more approachable.

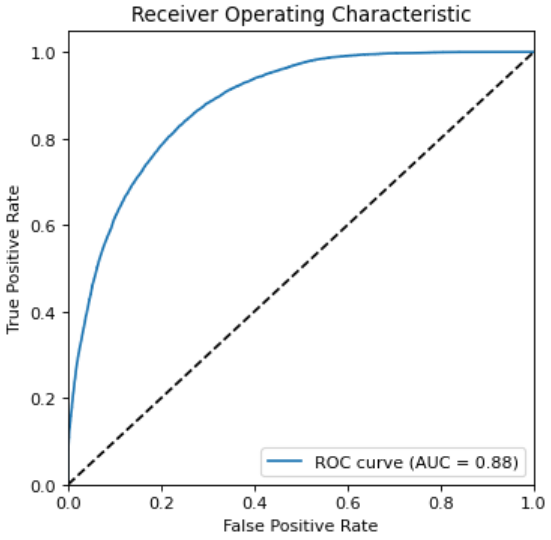
As discussed previously, we want to prioritize precision in the model to limit the financial strain to the hotel incurred by double booking a room. However, only considering precision motivates the model to misclassify most observations to minimize false positives. Scoring by AUC and precision allows the model to prioritize precision without tanking the overall performance of the model.

In [32]:
```
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

def do_grid_seach_HGBCT(arg):
    global grid_search

    param_grid = {
        'learning_rate': [0.01, 0.1, 0.5],
        'max_iter': [50, 100, 200, 300],
        'max_depth': [2, 4, 6],
        'min_samples_leaf': [1, 5, 10],
    }

    rf = HistGradientBoostingClassifier()

    grid_search = GridSearchCV(rf, param_grid=param_grid, cv=5, scoring=['roc_auc','precision'], refit='roc_auc')

    grid_search.fit(X_train, y_train)

    print("Best parameters: ", grid_search.best_params_)
    print("Best score: ", grid_search.best_score_)

    # Make predictions on the testing data
    y_pred = grid_search.predict(X_test)

    eval(y_test,y_pred)


button(do_grid_seach_HGBCT)
```

```
Warning, cell takes a wee while to run. Click to proceed.
Button(description='Click to run do_grid_seach_HGBCT.', layout=Layout(width='auto'), style=ButtonStyle())
```

In [33]:
```
from sklearn.model_selection import train_test_split

sample = encoded_df.sample(frac=1, random_state=1)

X = sample.drop('is_canceled', axis = 1)
y = sample['is_canceled']

# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state=42, stratify = y)

HGBCT = HistGradientBoostingClassifier(learning_rate=0.1, max_depth= 4,max_iter=100, min_samples_leaf= 5)
HGBCT.fit(X_train, y_train)
y_pred = HGBCT.predict(X_test)
y_prob = HGBCT.predict_proba(X_test)[:,1]
eval(y_test,y_pred,y_prob)
```

```
Accuracy: 0.8211880591014172
Precision: 0.7218108900200525
Recall: 0.5697674418604651
F1 Score: 0.636839956450735
Confusion Matrix:
 [[39661  3607]
 [ 7067  9359]]
AUC:  0.7432016464178909
```



The model selected is the Gradient Boosting Classification Tree, closely related to Random Forest classification. In gradient boosting, one iteratively combines many weak learners into a stronger one, and is closely related to gradient descent. The algorithm will start by making a tree with a poor performance. Subsequent trees will consider the residuals of that tree to improve performance. The end result will be a forest of decision trees, each with a weight based on its performance. To predict using new data, the appropriate leaf is chosen from each of the decision trees, and a weighted average is taken to yield the probability of being in either class (in this case, cancellation or not.)

Notice that we use a Histogram-based Gradient Boosting Classifier tree (HGBCT) in the grid search cross validation. HGBCT is derived from the Gradient Boosting classification, with its main difference being that it discretizes continuous features into histograms and only considers a subset of possible splits. As a result, it is significantly faster, especially for larger datasets. This makes cross validation and hyperparameter tuning much more efficient on a manageable timescale.

Other regression tree paradigms were used, and ultimately GradientBoosting was chosen due to its efficiency for data of this scale and high accuracy [3]. Other notable models tested include AdaBoost, where weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases and is well suited to binary classification. Ultimately however, it fell short of HGBCT's performance.

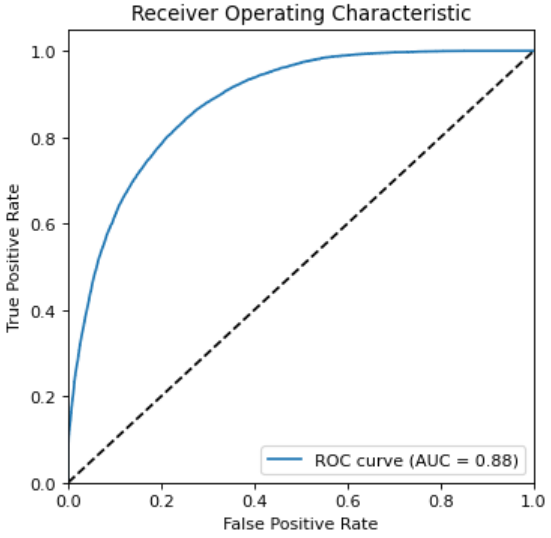
The model could be marginally improved by increasing the maximum depth of the decision tree, but the choice was made to limit this parameter to improve the interpretability of the model.

Since sklearn does not provide a method to create graphical trees for gradient boosting, we use the dependency lightgbm [4]. The functionality is the same.

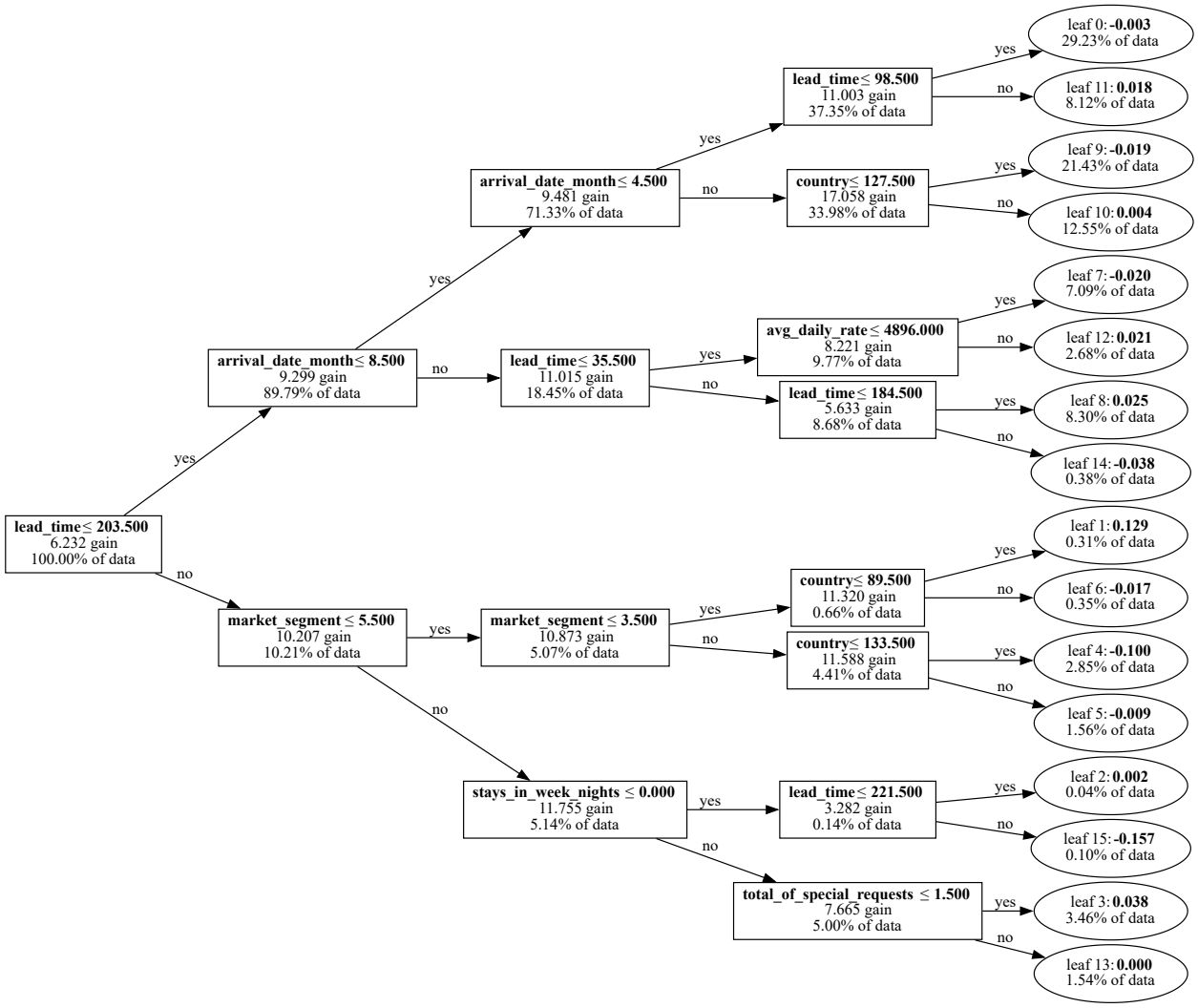
In [34]:
```
import lightgbm as lgb
from sklearn.tree import export_graphviz
import os
import warnings
warnings.filterwarnings("ignore")
```

```python
from pathlib import Path
downloads_path = str(Path.home() / "Downloads") # https://stackoverflow.com/questions/35851281/python-finding-the-us
os.environ["PATH"] += os.pathsep + downloads_path + r'\windows_10_msbuild_Release_graphviz-7.1.0-win32\Graphviz\bin'

# Make lgb model
lgb_HGBCT = lgb.LGBMClassifier(learning_rate=0.1, max_depth= 4,max_iter=100, min_samples_leaf= 5,random_state=612)
lgb_HGBCT.fit(X_train, y_train)
y_pred = lgb_HGBCT.predict(X_test)
y_prob = lgb_HGBCT.predict_proba(X_test)[:,1]

# Evalute model
eval(y_test,y_pred, y_prob)


# Convert the trained model to a LightGBM Booster object
booster = lgb_HGBCT.booster_

# Generate the graphical tree using the LightGBM plot_tree() function
lgb.create_tree_digraph(booster, tree_index=-1, show_info=['split_gain','data_percentage'],orientation='horizontal')
```

```
[LightGBM] [Warning] min_data_in_leaf is set with min_child_samples=20, will be overridden by min_samples_leaf=5. Cu
rrent value: min_data_in_leaf=5
[LightGBM] [Warning] num_iterations is set=100, max_iter=100 will be ignored. Current value: num_iterations=100
Accuracy: 0.8220591684256374
Precision: 0.7256609642301711
Recall: 0.56812370631925
F1 Score: 0.6373010995014684
Confusion Matrix:
 [[39740  3528]
 [ 7094  9332]]
AUC:  0.7432926935035282
```

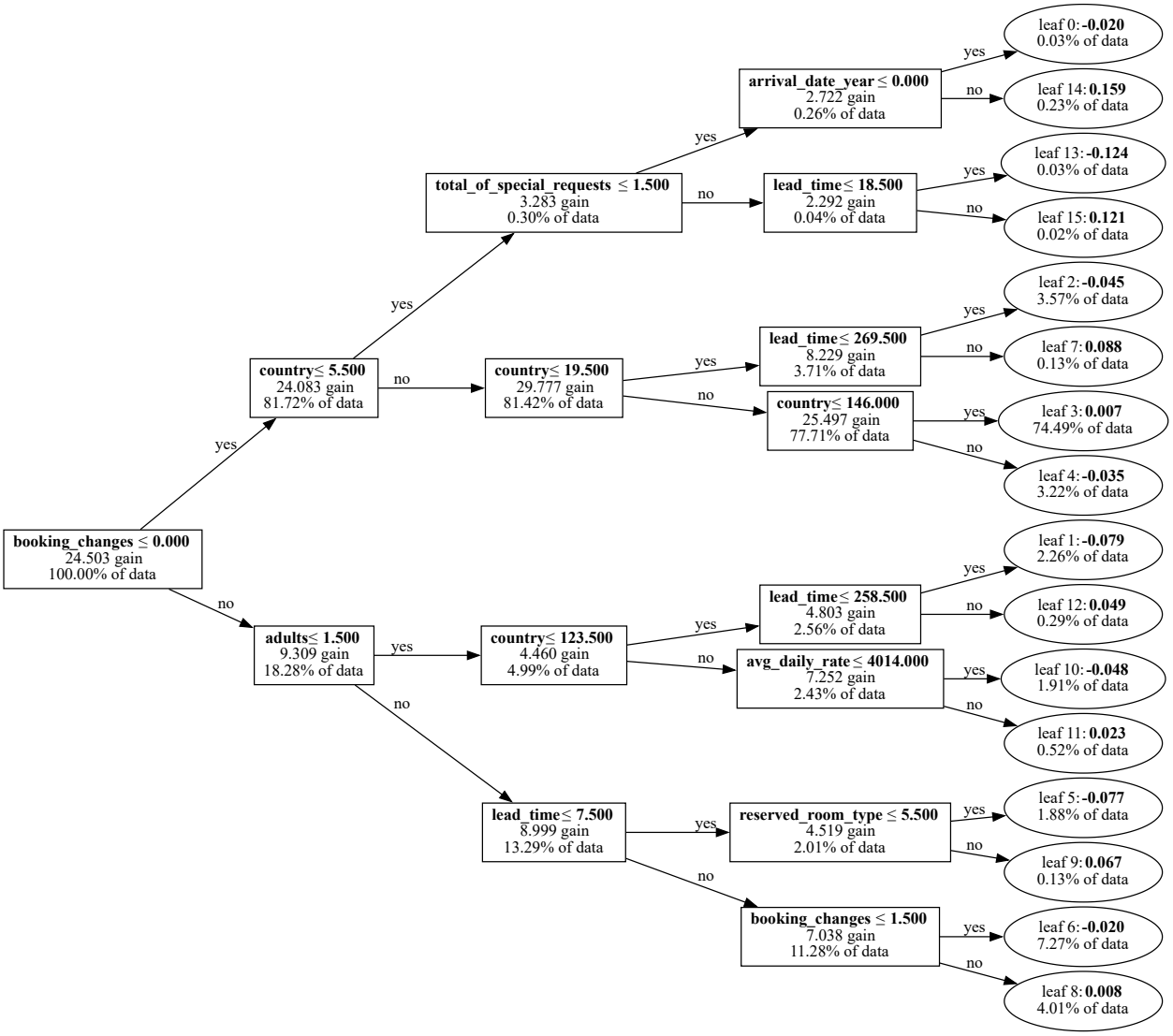


Out[34]:



In [35]:
```
lgb.create_tree_digraph(booster, tree_index=50, show_info=['split_gain','data_percentage'],orientation='horizontal')
```

Out[35]:



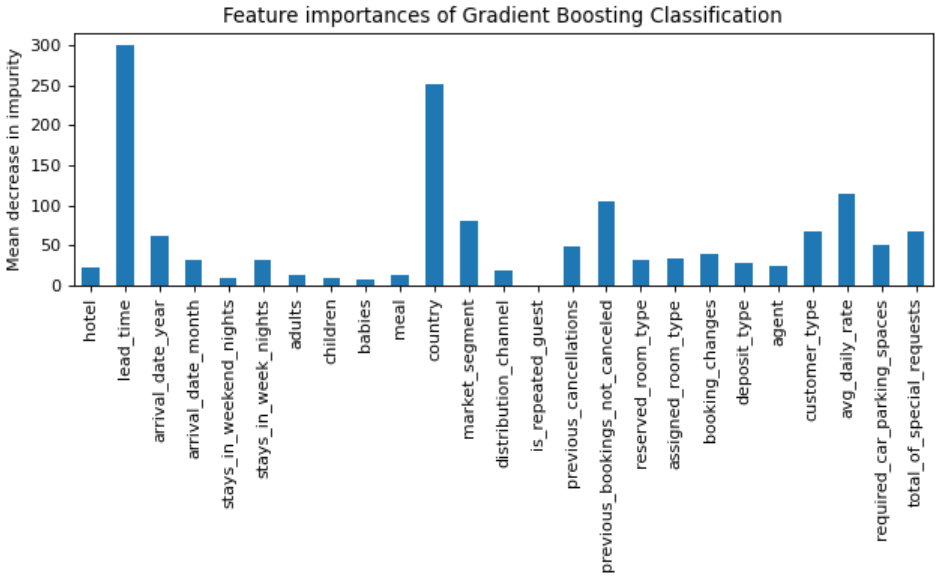
One tree instance of the forest is shown above. Many trees are created from the training set. A single tree is created using a subset of the data. Each tree has an associated weight based on the prediction score for that subset of data. For each data point we want to predict, we follow the decision rules of every tree in the forest. At the end, we reach a left in every tree. These leaves have an associated value, based on the weights of the tree. These values are then summed to give us our final prediction for that point.

We can immediately draw some conclusions even from this small sample of the forest. Note that the values in the leaves are not probabilities themselves but will affect the classification. Negative values indicate that a reservation satistfying the conditions for that node is less likely to be cancelled, proportional to the value. As the algorithm runs, the values of the nodes diminish as the forest hones its predictions.

For example, we can see from the second tree shown that for a booking with more than zero booking changes, 2 or more adults, and a lead time less than 7.5 days, the room type will have a relatively large effect on whether the customer is likely to cancel (leaves 5 and 9.) Also, leaf 15 on the first tree indicates that reservations with no weekday nights (so weekend trips) made hundreds of days in advance are quite unlikely to be cancelled. The aggregation of all of these decisions yields a model with high accuracy.

3.4 Feature importance

In [36]:
```
importances = lgb_HGBCT.feature_importances_
forest_importances = pd.Series(importances, index=X_train.columns)

fig, ax = plt.subplots()
forest_importances.plot.bar(ax=ax)
ax.set_title("Feature importances of Gradient Boosting Classification")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
plt.show()
```



Based on the feature importance plot, we saw that lead time and country of origin were the most important features in our model. This means that having differing values for these features would result in large noticable changes in the prediction. This observation agrees with the correlations noted in the EDA section -- both lead time and country were among the features most correlated with the cancellation status.

The importance of lead time, which is a measure of how long in advance a booking was made, indicates that the longer the lead time, the more likely the boking is to be cancelled. This makes sense as a longer lead time creates more opportunities for things to happen that would make a client cancel.

Similarly, the country of origin of the booking could be an important predictor because different countries may have different cultural norms, economic conditions or polical conditions that influence whether a booking is cancelled.

It's worth noting that while lead time and country were the most important features, other features such as the average daily rate and requiring car parking spaces also had a significant affect on the cancellation status. This highlights the importance of considering multiple factors when predicting whether a booking is likely to be cancelled or not.

Overall, our model provides valuable insights into the factors that influence booking cancellations. By understanding the different factors that impact the cancellation status, more data collection on the important features can be done to improve the model.

3.5 Recommendations

Based on the insights gathered by analysing our model, there are several recommendations that can be made to decrease the cancellation rate of bookings:

- As lead time was identified to be important, a limit on the maximum lead time would encourage customers to book closer to their arrival date. Alternatively, an incentive to book closer to the arrival date will also work.
- Target specific customers from demographics less likely to cancel by running targetted ads or offers.
- Be wary of customers who have previously cancelled bookings, and reward customers who have stayed without cancelling.
- Offer competetive rates for customer retention.

Besides changing booking behaviours, a double-booking scheme can be made, where bookings that have a high probability of being cancelled as predicted by our model should be double-booked. This avoids having empty rooms at the cost of the risk of overbooking. A further model could be created by balancing the risk of overbooking with underbooking.

3.6 Other models

Apart from the tree based models presented above, other models were tried, but deemed less effective than the presented models. Logistic regression did not yield a high accuracy. Similarly, linear support vector classification also did not give a good accuracy score. This is because the data is not divided linearly. Higher order support vector classification might work, but requires much more computing power than we had available, since our dataset is very big. Neural nets also needs a lot of computing power. We chose not to pursure a neural net model as tweaking the types and number of layers is a time consuming black-box process.

---

4. Discussion & Conclusions

In conclusion, we have developed a machine learning model that predicts the likelihood of a hotel booking being cancelled using a gradient boosting forest method. The model was trained on a large dataset containing information about bookings such as lead time, customer demographics, and room type. Through exploratory data analysis and feature engineering, we were able to identify likely factors that contribute to the likelihood of a booking being cancelled.

After training, our model achieved a high accuracy score of 82% on the test data set, which indicates that it can reliably predict booking cancellations. Our model is also robust as cross validation suggests consistent results. Other models were tried, but did not yield results better than gradient boosting forest. We also found that our model gives results consistent with the exploratory data analysis. Several recommendations for reducing cancellation rates were presented based on our findings.

Given these observations, it would be economically viable to implement this model as a method of optimising hotel room allocations to minimise empty rooms. Our model has a false positive rate (predicting cancellation when customer does not cancel) of 5.9% and a false negative rate (predicting not cancelled for a cancelled booking) of 11.9%. If implemeting double booking, false positives cause overbooking, and false negatives cause underbooking. Since the false negative rate is higher than false positives, on average, rooms will be underbooked, and the risk of overbooking in the long run is low.

Overall, our model provides valuable insights for hotel managers to better understand the factors that impact booking cancellations, and take actions to reduce them. Further improvements can be made by incorporating additional data sources, such as local and global events or a detailed breakdown of demographics, to enhance the accuracy of the model.

5. References

[1] https://www.avvio.com/2022-cancellation-rate-trends/

[2] https://stackoverflow.com/questions/24458645/label-encoding-across-multiple-columns-in-scikit-learn

[3] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 30.

[4] https://lightgbm.readthedocs.io/en/latest/index.html