# Machine Learning in Python - Group Project 2

**Due Friday, April 14th by 16.00 pm. Extended to Friday, April 21st by 16.00 pm.**

*Data Scientist Contractors: Luke Appleby, Adam Dunajski, Tom Birkbeck, Josep Garcia-Reyero Sais*

A consortium operating hotels contracted MLP's data science practice in March 2023 to build a model of hotel booking cancellations. In this report, we put forward a validated model based on provided representative sample data from two hotels in the consortium's chain.

# 1. Introduction

The dataset provided (Antonio et al, 2019) is broad, messy and real-world. It includes a wide variety of variables including week of booking in the year, whether a company or agency made the booking, the number of adults and children in the booking, the lead time on the booking and more. This project will use this dataset, assuming it is a representative sample of the consortium's operations, to provide modelling and insight.

## 1.1. Project objectives

The primary objective of the project is the deliverable for the hotel consortium, our model of booking cancellations. In addition, the consortium have specified that the model must be understandable, validated, accurate and reliable, and should be explainable.

The ultimate goal is to develop a single model, but as per the requests of the client, this report contains details of other models tested as well.

Finally, the client has specified that they would like some business analysis of the model, including the potential losses and gains when considering accurate predictions as well as false positives/false negatives. The client has requested some recommendations and some analysis of whether or not they would be economically viable.

## 1.2. Headline results

A number of models were tested, and full results are available in section 3. The most accurate model found was a neural network, which correctly predicted hotel cancellations on cross-validated test data with 87.8% accuracy.

We make two recommendations for the client.

- To charge an additional cancellation fee for bookings which are assessed to have a high probability of cancellation by our model.
- To overbook up to 50% of the rooms which are likely to be cancelled.

MLP's data science practice recommends that a full investigation into the financial effects of these policies be carried out. However, preliminary analysis based on our modelling indicates these two policy changes could lead to an additional $4,281,620 . This is comprised of 6,296 extra bookings per annum, with an estimated average booking value of $340 (SiteMinder, 2016), equating to $2,140,640 in additional annual revenue; and charging an average cancellation fee of $170 on the 12,594 cancellations, equating to $2,140,980, for a total of $4,281,620 additional annual revenue across comparable operations sizes. This would scale across the client's operations beyond the dataset, so long as the data provided remains a representative sample.

Full results and methodology are contained in Section 4.

## 1.3. Table of contents

## 1.4. General Setup

Necessary modules and the data file are loaded here. The client should be aware that these dependencies are required for future usage of the model and analysis techniques.

In [ ]:

```
pip install keras-tuner --upgrade
```

In [2]:

```
# 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

# sklearn modules
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

import pandas as pd

import warnings
warnings.simplefilter("ignore")

from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import ConfusionMatrixDisplay

import tensorflow as tf
import tensorflow.keras as keras

import keras_tuner as kt
import keras_tuner
```

In [3]:

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

In [4]:

```
# Load data
hotel_df = pd.read_csv("hotel.csv")

# Fix typo in response variable.
hotel_df.rename(columns = {'is_canceled':'is_cancelled'}, inplace = True)
df_explore = hotel_df.copy()
```

# 2. Exploratory Data Analysis and Feature Engineering

## 2.1. Data quality analysis and cleaning

In [5]:

```
# Number of cancellations

print(df_explore['is_cancelled'].sum())

# The timeframe of the data is 3 years and one month
df_explore[['arrival_date_year', 'arrival_date_month']]
```

```
print("Average number of yearly cancellations", np.round((df_explore['is_cancelled'].sum() / (3*12 + 1) * 12),0))
```

```
44224
Average number of yearly cancellations 14343.0
```

We begin by considering the duplicate and missing values in the data set.

In [6]:

```
# Count the number of duplicates
num_duplicates = df_explore.duplicated().sum()

# Print the number of duplicates and the number of NaNs
print(f"Number of duplicates: {num_duplicates}\n")
```

```
Number of duplicates: 32252
```

We first see that we have 32,252 duplicate bookings, which represents a significant 27% of the data set. Exploring the `is_cancelled` outcome for the duplicate bookings, we have the following:

In [8]:

```
print("Percentage of duplicate booking cancelled: ",
      round(df_explore[df_explore.duplicated(keep=False)]['is_cancelled'].sum()
      /df_explore["is_cancelled"].sum() * 100, 2))

print("Percentage of all booking cancelled: ",
      round((df_explore["is_cancelled"].sum()/len(df_explore)) * 100, 2))
```

```
Percentage of duplicate booking cancelled:  53.67
Percentage of all booking cancelled:  37.04
```

In our data we see a larger proportion of cancelations where the booking is a duplicate. In some cases this could be the a data error where the row as been listed twice, however we suspect that this is sometimes customers accidentally booking the reservation twice or more, then cancelling all. Duplicates could also be two group booking a joint holiday and so making two bookings with the same paramaters, e.g. a group of 10 booking two 5 person rooms seperately for the same trip. For this reason we do not drop duplicates, and instead add a categorical variable that states if the booking is a duplicate. If a new data point is added, it should be compared to existing data set to determine if it is a duplicate booking, and the value for "duplicate".

In [9]:

```
df_explore["is_duplicate"] = df_explore.drop(['is_cancelled'], axis=1).duplicated(keep=False)
df_explore.is_duplicate = df_explore.is_duplicate.replace({True: 1, False: 0})
```

We now consider the missing entries in the data.

In [10]:

```
# Find columns with NaN values
nan_columns = df_explore.columns[df_explore.isna().any()].tolist()

# Count the number of NaNs in each column
nan_counts = df_explore[nan_columns].isna().sum().reset_index()
nan_counts.columns = ['column', 'num_nan']

# Add a column for the percentage of NaNs in each column
nan_counts['pct_nan'] = nan_counts['num_nan'] / len(df_explore) * 100

# Add a column for the number of unique values in each column
unique_counts = df_explore[nan_columns].nunique().reset_index()
unique_counts.columns = ['column', 'num_unique']
nan_counts = pd.merge(nan_counts, unique_counts, on='column')

# Calculate the correlations between each NaN column and 'is_cancelled'
correlations = df_explore[nan_columns + ['is_cancelled']].corr().iloc[:-1, -1].reset_index()
correlations.columns = ['column', 'correlation']
nan_counts = pd.merge(nan_counts, correlations, on='column')

print(nan_counts.to_string(index=False))
```

```
  column  num_nan   pct_nan  num_unique  correlation
children        4  0.003350           5     0.005048
   agent    16340 13.686238         333    -0.083114
 company   112593 94.306893         352    -0.020642
```

We see that company has a lot of missing entries, 94.3% of the whole column, and agent also has a significant percentage of 13.7%. We will drop these columns, but we create a binary attribute for the top 10 agents and top 5 companies, since the correlations are lower for companies, that are most correlated to booking cancellation.

In [11]:

```
def create_top_n_binary_features(col_name, n):
    # Find the correlation between 'is_cancelled' and each value in the column
    col_corr = df_explore.groupby(col_name)['is_cancelled'].corr(df_explore[col_name]).reset_index()
    col_corr.columns = [col_name, 'corr']

    # Find the top n values with the highest correlation with 'is_cancelled'
    top_values = col_corr.sort_values(by='corr',
                                      ascending=False).head(n)[col_name].tolist()

    # Create binary features for the top n values
    for value in top_values:
        col = f'{col_name}_{value}'
        df_explore[col] = np.where(df_explore[col_name] == value, 1, 0)

    # Drop the original column
    df_explore.drop(columns=[col_name], inplace=True)

create_top_n_binary_features('agent', 10)
create_top_n_binary_features('company', 5)
```

Finally we drop all the remaining rows with missing values, which will come from the children and country columns but represent a negligible percentage.

In [12]:

```
df_explore = df_explore.dropna()
```

We now create a single variable to capture whether the room time that the user got is not the expected room that was booked. To do so we create a binary attribute which is 1 if `reserved_room_type` != `assigned_room_type`. We then drop both `reserved_room_type` and `assigned_room_type`.

In [13]:

```
df_explore["not_expected_room"] = df_explore["reserved_room_type"] != df_explore["assigned_room_type"]

print("Percentage not_expected_room bookings cancelled: ",
      round(df_explore[df_explore["not_expected_room"] == 1]["is_cancelled"].sum()
      /len(df_explore[df_explore["not_expected_room"] == 1]) *100, 2))

print("Percentage of all bookings cancelled: ",
      round(df_explore["is_cancelled"].sum()/len(df_explore) *100, 2))

df_explore.drop(columns=["reserved_room_type", "assigned_room_type"], inplace=True)
```

```
Percentage not_expected_room bookings cancelled:  5.39
Percentage of all bookings cancelled:  37.14
```

We see that the proportion of bookings cancelled is lower in the cases where the room type is not the desired room type. This might be due to room upgrades or other factors. Further, we will combine the features `previous_cancellations` and `previous_bookings_not_cancelled` into `proportion_previous_cancelled`.

In [14]:

```
df_explore['proportion_previous_cancelled'] = 0

# We use mask to avoid division by 0.
```

```python
mask = (df_explore['previous_cancellations'] + df_explore['previous_bookings_not_canceled']) > 0
df_explore.loc[mask, 'proportion_previous_cancelled'] = (df_explore['previous_cancellations'] /
                                                         (df_explore['previous_cancellations'] +
                                                          df_explore['previous_bookings_not_canceled']))

corr = df_explore['is_cancelled'].corr(df_explore['proportion_previous_cancelled'])
corr_prev_cancel = df_explore['previous_cancellations'].corr(df_explore['is_cancelled'])
corr_prev_not_cancel = df_explore['previous_bookings_not_canceled'].corr(df_explore['is_cancelled'])

print(f"Correlation between previous_cancellations and is_cancelled: {corr_prev_cancel:.2f}")
print(f"Correlation between previous_bookings_not_cancelled and is_cancelled: {corr_prev_not_cancel:.2f}")
print(f"Correlation between proportion_previous_cancelled and is_cancelled: {corr:.2f}")

df_explore.drop(columns=["previous_cancellations", "previous_bookings_not_canceled"], inplace=True)

# df_explore["has_cancelled"] = df_explore["previous_cancellations"] > 0

# print("Percentage of booking cancelled for which customer has previously cancelled: "
#,df_explore[df_explore["has_cancelled"] == 1]["is_cancelled"].sum()/len(df_explore[df_explore["has_cancelled"] == 1]) *100)
# print("Percentage of all booking cancelled: ",df_explore["is_cancelled"].sum()/len(df_explore) *100)
```

```
Correlation between previous_cancellations and is_cancelled: 0.11
Correlation between previous_bookings_not_cancelled and is_cancelled: -0.06
Correlation between proportion_previous_cancelled and is_cancelled: 0.29
```

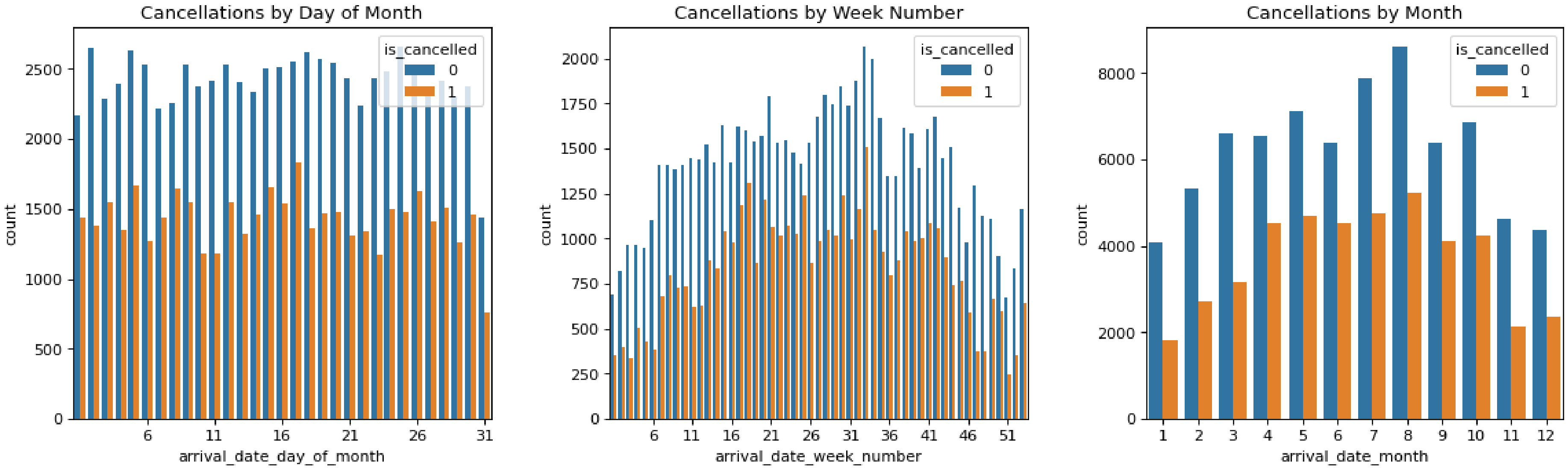
We can see that the new feature `proportion_previous_cancelled` has a much higher correlation with `is_cancelled` than `previous_cancellations` and `previous_bookings_not_cancelled`. Therefore, we drop `previous_cancellations` and `previous_bookings_not_cancelled` to avoid overfitting.

Finally, we will consider the date variables. We will drop `arrival_date_year` straight away because it does not provide any significant information. Then we will turn `arrival_date_month` and plot histograms for `arrival_date_month`, `arrival_date_day_of_month` and `arrival_date_week_number`.

In [16]:

```python
# Drop year and convert month to numerical.
df_explore.drop(columns=["arrival_date_year"], inplace=True)
df_explore['arrival_date_month'] = pd.to_datetime(df_explore['arrival_date_month'], format='%B').dt.month

# Plot histograms.
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15,5))

sns.countplot(x='arrival_date_day_of_month', hue='is_cancelled', data=df_explore, ax=axes[0])
axes[0].set_title('Cancellations by Day of Month')
```

```python
axes[0].set_xticks(range(5, 32, 5))
sns.countplot(x='arrival_date_week_number', hue='is_cancelled', data=df_explore, ax=axes[1])
axes[1].set_title('Cancellations by Week Number')
axes[1].set_xticks(range(5, 53, 5))

sns.countplot(x='arrival_date_month', hue='is_cancelled', data=df_explore, ax=axes[2])
axes[2].set_title('Cancellations by Month')

fig.tight_layout(pad=3)

plt.show()
```


Cancellations by Day of Month

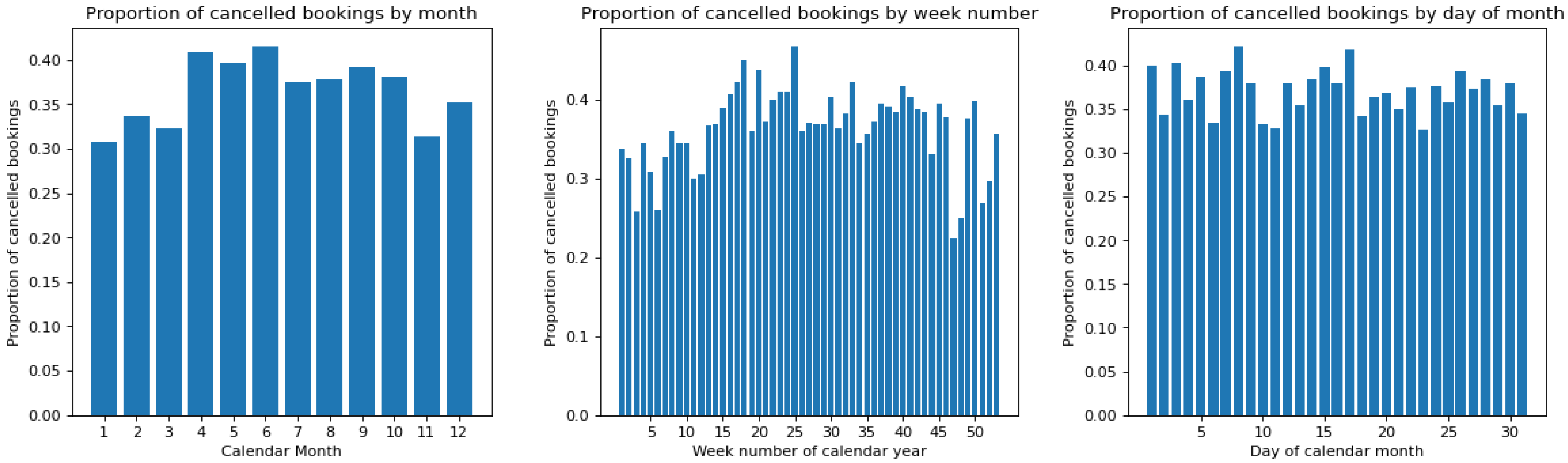
Cancellations by Week Number

Cancellations by Month

In [17]:

```python
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15,5))
# plot proportion of cancelled bookings by month
proportions = []
for i in range(1,13):
  p = df_explore[df_explore["arrival_date_month"] == i][df_explore["is_cancelled"] == 1].shape[0]
       / df_explore[df_explore["arrival_date_month"] == i].shape[0]
  proportions.append(p)
proportions

axes[0].bar(range(1,13),proportions)
axes[0].set_title("Proportion of cancelled bookings by month")
axes[0].set_ylabel("Proportion of cancelled bookings")
axes[0].set_xlabel("Calendar Month")
```

```
axes[0].set_xticks(range(1,13))

# plot proportion of cancelled bookings by month
proportions = []
for i in range(1,54):
  p = df_explore[df_explore["arrival_date_week_number"] == i][df_explore["is_cancelled"] == 1].shape[0]
      / df_explore[df_explore["arrival_date_week_number"] == i].shape[0]
  proportions.append(p)
proportions

axes[1].bar(range(1,54),proportions)
axes[1].set_title("Proportion of cancelled bookings by week number")
axes[1].set_ylabel("Proportion of cancelled bookings")
axes[1].set_xlabel("Week number of calendar year")
axes[1].set_xticks(range(5,54,5))

# plot proportion of cancelled bookings by month
proportions = []
for i in range(1,32):
  p = df_explore[df_explore["arrival_date_day_of_month"] == i][df_explore["is_cancelled"] == 1].shape[0]
       / df_explore[df_explore["arrival_date_day_of_month"] == i].shape[0]
  proportions.append(p)
proportions

axes[2].bar(range(1,32),proportions)
axes[2].set_title("Proportion of cancelled bookings by day of month")
axes[2].set_ylabel("Proportion of cancelled bookings")
axes[2].set_xlabel("Day of calendar month")
axes[2].set_xticks(range(5,32,5))
fig.tight_layout(pad=3)
plt.show()
```



We can see that `arrival_date_day_of_month` is rather uniformly distributed for both cancelled bookings and none-cancelled bookings. We will therefore drop it since this suggests low predictive power. Similarly, `arrival_date_month` and `arrival_date_week_number` follow rather similar distributions. We will thus only keep `arrival_date_week_number` as it may capture more seasonal holiday fluctuations, for instance in short holidays like UK half-terms. By only keeping one date feature we also avoid overfitting.

In [18]:

```
df_explore.drop(columns=["arrival_date_day_of_month", "arrival_date_month"], inplace=True)
```

In the cell below, we look at the potential unique values for each column in the dataset. By developing this understanding, we can get a stronger idea of what we're working with and make more informed analysis and modelling decisions. For numerical variables, the maximum and minimum are shown instead.

In [19]:

```
column_names = df_explore.columns.to_list()
unique_values = [df_explore[str(name)].unique() for name in column_names]
for i in range(0,len(unique_values)):
  if column_names[i]=="country" or column_names[i]=="company" or column_names[i]=="agent":
    print(column_names[i],":",unique_values[i])
  elif len(unique_values[i])>25:
    print(column_names[i]," max: ",max(unique_values[i])," min: ",min(unique_values[i]))
  else:
    print(column_names[i],":",unique_values[i])
```

```
is_cancelled : [0 1]
hotel : ['Resort Hotel' 'City Hotel']
lead_time  max:  737  min:  0
arrival_date_week_number  max:  53  min:  1
stays_in_weekend_nights : [ 0  1  2  4  3  6 13  8  5  7 12  9 16 10 14]
stays_in_week_nights  max:  41  min:  0
adults : [ 2  1  3  4 40 26 50 27 55  0 20  6  5 10]
children : [ 0.  1.  2. 10.  3.]
babies : [ 0  1  2 10  9]
meal : ['BB' 'FB' 'HB' 'SC' 'Undefined']
country : ['PRT' 'GBR' 'USA' 'ESP' 'IRL' 'FRA' 'ROU' 'NOR' 'OMN' 'ARG' 'POL' 'DEU'
 'BEL' 'CHE' 'CN' 'GRC' 'ITA' 'NLD' 'DNK' 'RUS' 'SWE' 'AUS' 'EST' 'CZE'
 'BRA' 'FIN' 'MOZ' 'BWA' 'LUX' 'SVN' 'ALB' 'IND' 'CHN' 'MEX' 'MAR' 'UKR'
 'SMR' 'LVA' 'PRI' 'SRB' 'CHL' 'AUT' 'BLR' 'LTU' 'TUR' 'ZAF' 'AGO' 'ISR'
 'CYM' 'ZMB' 'CPV' 'ZWE' 'DZA' 'KOR' 'CRI' 'HUN' 'ARE' 'TUN' 'JAM' 'HRV'
 'HKG' 'IRN' 'GEO' 'AND' 'GIB' 'URY' 'JEY' 'CAF' 'CYP' 'COL' 'GGY' 'KWT'
 'NGA' 'MDV' 'VEN' 'SVK' 'FJI' 'KAZ' 'PAK' 'IDN' 'LBN' 'PHL' 'SEN' 'SYC'
 'AZE' 'BHR' 'NZL' 'THA' 'DOM' 'MKD' 'MYS' 'ARM' 'JPN' 'LKA' 'CUB' 'CMR'
 'BIH' 'MUS' 'COM' 'SUR' 'UGA' 'BGR' 'CIV' 'JOR' 'SYR' 'SGP' 'BDI' 'SAU'
 'VNM' 'PLW' 'QAT' 'EGY' 'PER' 'MLT' 'MWI' 'ECU' 'MDG' 'ISL' 'UZB' 'NPL'
 'BHS' 'MAC' 'TGO' 'TWN' 'DJI' 'STP' 'KNA' 'ETH' 'IRQ' 'HND' 'RWA' 'KHM'
 'MCO' 'BGD' 'IMN' 'TJK' 'NIC' 'BEN' 'VGB' 'TZA' 'GAB' 'GHA' 'TMP' 'GLP'
 'KEN' 'LIE' 'GNB' 'MNE' 'UMI' 'MYT' 'FRO' 'MMR' 'PAN' 'BFA' 'LBY' 'MLI'
 'NAM' 'BOL' 'PRY' 'BRB' 'ABW' 'AIA' 'SLV' 'DMA' 'PYF' 'GUY' 'LCA' 'ATA'
 'GTM' 'ASM' 'MRT' 'NCL' 'KIR' 'SDN' 'ATF' 'SLE' 'LAO']
market_segment : ['Direct' 'Corporate' 'Online TA' 'Offline TA/TO' 'Complementary' 'Groups'
 'Aviation']
distribution_channel : ['Direct' 'Corporate' 'TA/TO' 'Undefined' 'GDS']
is_repeated_guest : [0 1]
booking_changes : [ 3  4  0  1  2  5 17  6  8  7 10 16  9 13 12 20 14 15 11 21 18]
deposit_type : ['No Deposit' 'Refundable' 'Non Refund']
days_in_waiting_list  max:  391  min:  0
customer_type : ['Transient' 'Contract' 'Transient-Party' 'Group']
adr  max:  5400.0  min:  -6.38
required_car_parking_spaces : [0 1 2 8 3]
total_of_special_requests : [0 1 3 2 4 5]
is_duplicate : [0 1]
agent_1.0 : [0 1]
agent_2.0 : [0 1]
agent_3.0 : [0 1]
agent_4.0 : [0 1]
agent_5.0 : [0 1]
agent_6.0 : [0 1]
agent_7.0 : [0 1]
```

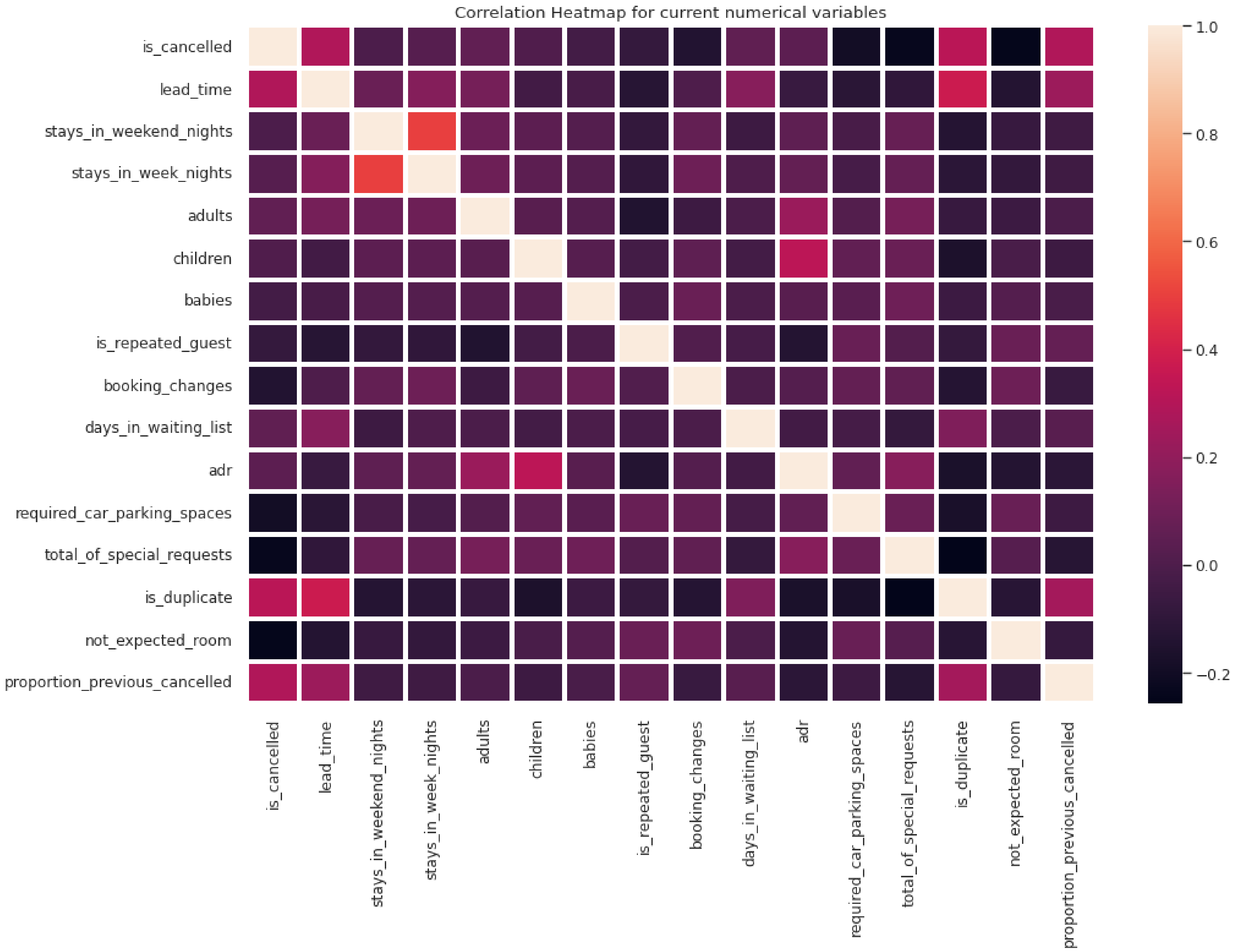
```
agent_8.0 : [0 1]
agent_9.0 : [0 1]
agent_10.0 : [0 1]
company_6.0 : [0 1]
company_8.0 : [0 1]
company_9.0 : [0 1]
company_10.0 : [0 1]
company_11.0 : [0 1]
not_expected_room : [False  True]
proportion_previous_cancelled  max:  1.0  min:  0.0
```

## 2.2. Looking into correlations

We now create a correlation plot of all the variables currently in the dataset, so we can see cross-correlations and correlations with the variable we are trying to predict, `is_cancelled`.

In [21]:

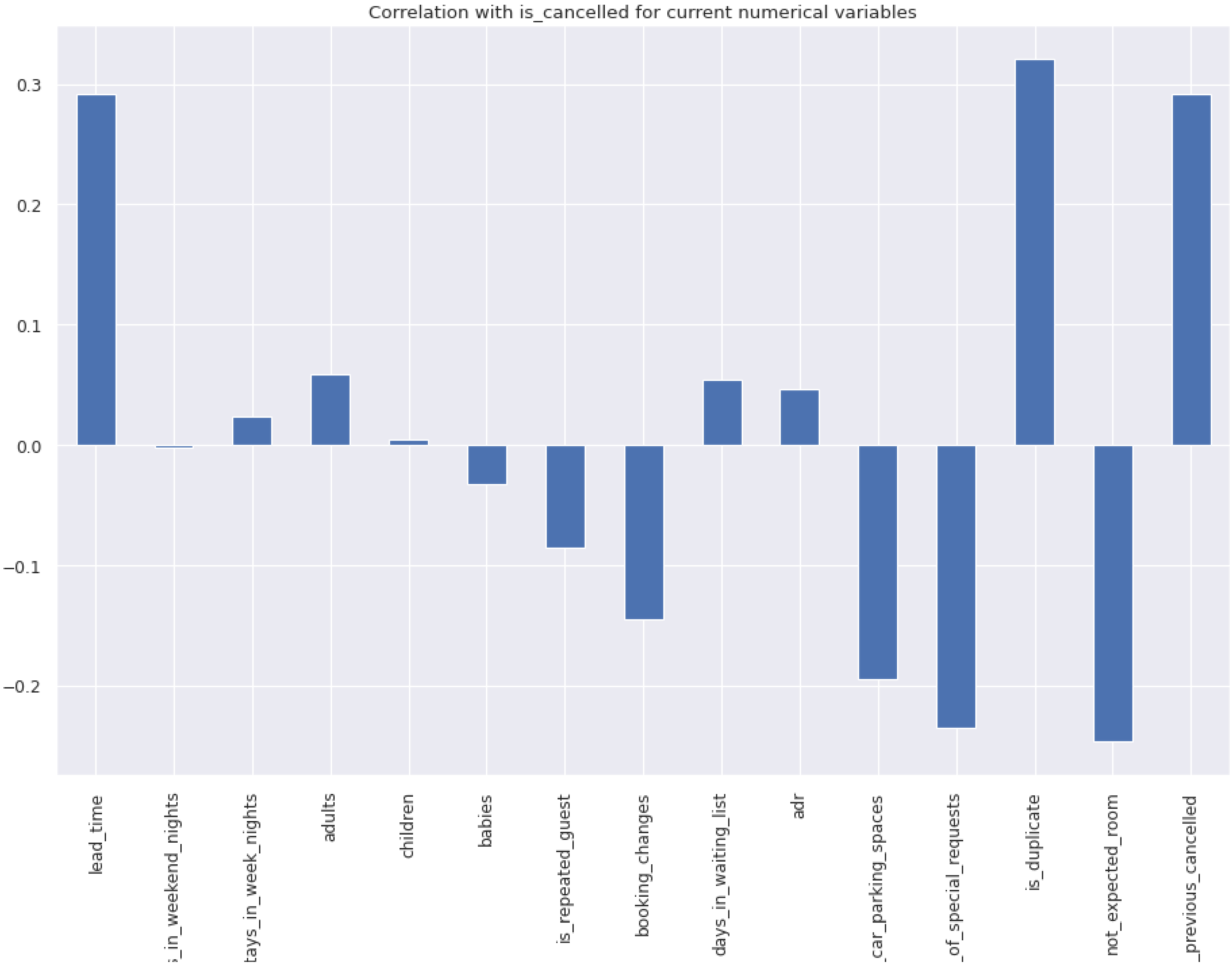
```
column_names = df_explore.columns.to_list()
columns = []
for c in column_names:
  if c[:5] != "agent" and c[:7] != "company" and c[:7] != "arrival":
    columns.append(c)

sns.set(rc={'figure.figsize': (14, 9)})
sns.heatmap(df_explore[columns].corr(), annot = False, fmt = '.1f', linewidths = 2)
plt.title("Correlation Heatmap for current numerical variables")
plt.show()
```


Correlation Heatmap for current numerical variables
is_cancelled
lead_time
stays_in_weekend_nights
stays_in_week_nights
adults
children
babies
is_repeated_guest
booking_changes
days_in_waiting_list
adr
required_car_parking_spaces
total_of_special_requests
is_duplicate
not_expected_room
proportion_previous_cancelled
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

The following bar plot shows the correlations of current numerical variables with is_cancelled. This gives us a strong understanding of which variables might be significant in any model we try to create: for example, **`total_of_special_requests`** has a strong negative correlation, implying that many special requests may reduce the chance of being cancelled. In real world data, we may be seeing things like wheelchair-accessible rooms, in which case cancelling to change hotel or trip details are likely to be more difficult, perhaps explaining some of the negative correlation.

In [23]:

```python
df_explore[columns].corr()["is_cancelled"][1:].plot.bar()
plt.title("Correlation with is_cancelled for current numerical variables")
plt.show()
```

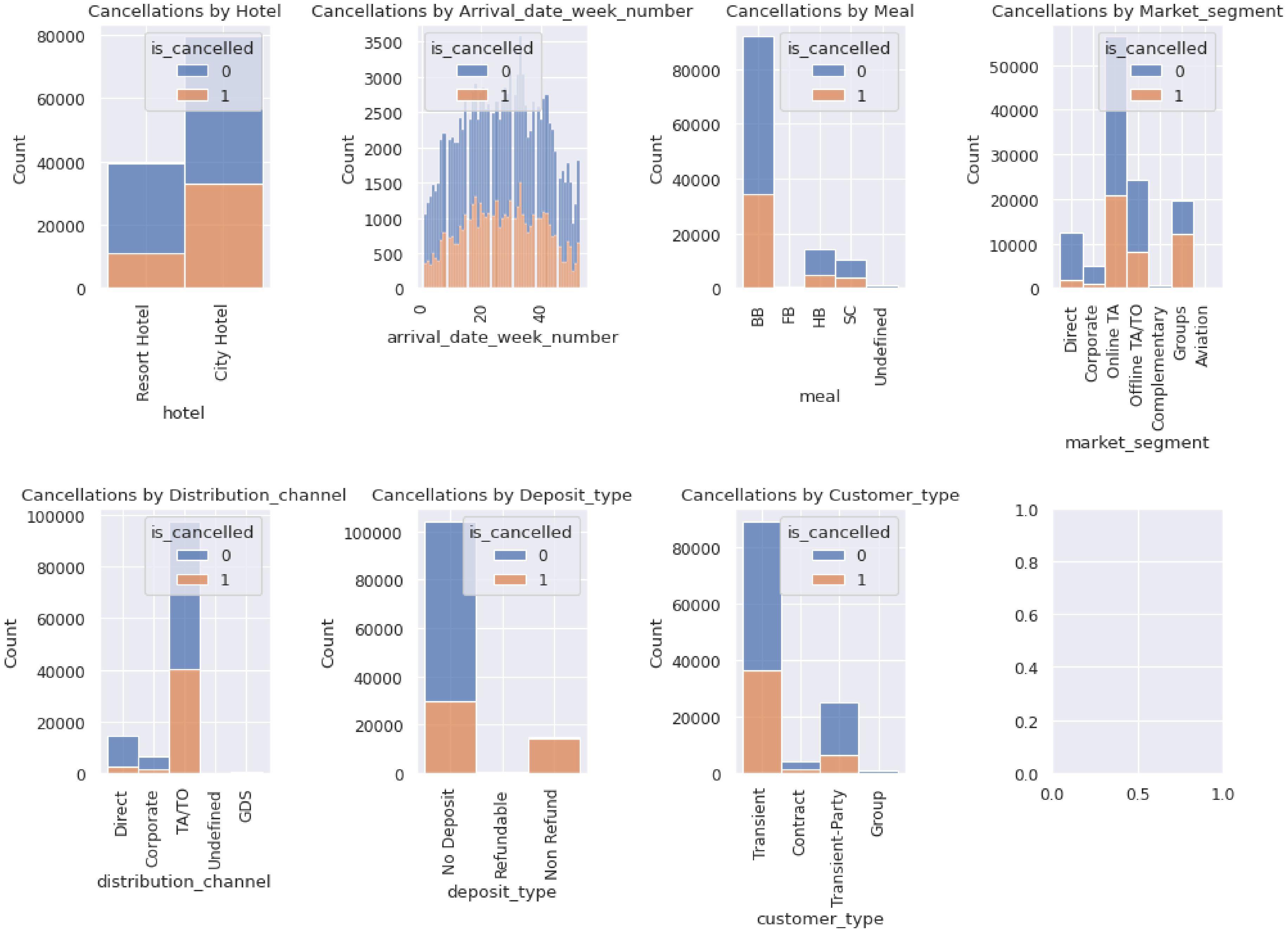

Correlation with is_cancelled for current numerical variables
0.3
0.2
0.1
0.0
−0.1
−0.2
lead_time
s_in_weekend_nights
tays_in_week_nights
adults
children
babies
is_repeated_guest
booking_changes
days_in_waiting_list
adr
_car_parking_spaces
_of_special_requests
is_duplicate
not_expected_room
_previous_cancelled

## 2.3. Dealing with categorical variables

The next cell generates count plots for categorical variables, so we can see how many cancellations there are per categorical variable per category. For example, the proportions of cancellations for `transient` compared with `transient-party` customers are different, implying that `transient` customers may be more likely to cancel.

In [25]:

```
categorical_vars = ['hotel', 'arrival_date_week_number','meal', 'market_segment', 'distribution_channel',
                    'deposit_type', 'customer_type', 'country']

fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(13, 10), sharey=False)

# Loop through the categorical variables and create a countplot for each variable.
# We will not use country as there are too many countries.
for i, cat_var in enumerate(categorical_vars[:-1]):
    row = i // 4
    col = i % 4
    sns.histplot(x=cat_var, hue='is_cancelled', data=df_explore, ax=axes[row, col],multiple="stack")
    axes[row, col].set_title(f'Cancellations by {cat_var.capitalize()}')
    axes[row, col].set_xticklabels(axes[row, col].get_xticklabels(), rotation=90)

# Adjust the spacing between subplots
fig.subplots_adjust(hspace=0.7, wspace=0.5)
fig.tight_layout(pad=3)
plt.show()
```


Cancellations by Hotel
Count
80000
60000
40000
20000
0
is_cancelled
0
1
Resort Hotel
City Hotel
hotel
Cancellations by Arrival_date_week_number
Count
3500
3000
2500
2000
1500
1000
500
0
is_cancelled
0
1
0
20
40
arrival_date_week_number
Cancellations by Meal
Count
80000
60000
40000
20000
0
is_cancelled
0
1
BB
FB
HB
SC
Undefined
meal
Cancellations by Market_segment
Count
50000
40000
30000
20000
10000
0
is_cancelled
0
1
Direct
Corporate
Online TA
Offline TA/TO
Complementary
Groups
Aviation
market_segment
Cancellations by Distribution_channel
Count
100000
80000
60000
40000
20000
0
is_cancelled
0
1
Direct
Corporate
TA/TO
Undefined
GDS
distribution_channel
Cancellations by Deposit_type
Count
100000
80000
60000
40000
20000
0
is_cancelled
0
1
No Deposit
Refundable
Non Refund
deposit_type
Cancellations by Customer_type
Count
80000
60000
40000
20000
0
is_cancelled
0
1
Transient
Contract
Transient-Party
Group
customer_type
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0

In the next cell, we apply some one hot encoding for categorical variables, expanding the categorical columns into binary columns for each category. This is important for building any models, as dealing with categorical data can be difficult.

In [26]:

```
df_explore = pd.get_dummies(df_explore, columns=categorical_vars, drop_first=False)

# Setting the main df to df_explore
hotel_df = df_explore
```

# 3. Model Fitting and Tuning

## 3.1. Table of modelling results

A number of different models were tried. The following table shows the results for the two leading modelling approaches.

| Model | Accuracy on Test Set | AUC |
|---|---|---|
| Logistic Regression | 0.823 | 0.793 |
| Neural Network | 0.878 | 0.864 |

We chose not to implement any tree based models in this report since there are such a large number of features a suitably accurate tree based model would end up being large to the point where it loses its understandability which is one of the main reasons to use such a model. Additionally we did not use rando forest or bagging type models as these are considered to be ‘black box’ methods and are also computationally too demanding for the resources we had available.

Another type of model we chose not to investigate was the Support Vector Machine (SVM). These are a generalisation of Support Vector Classifiers (SVC) which in turn generalise Maximal Margin Classifiers (MMC). The aim of an MMC is to separate two classes of points with a line that leaves the greatest space between the line and points. One can only use an MMC where it is possible to draw a line between two classes completely separating the two. When this is not possible a SVC is used and there is an area in which points can lie on the ‘wrong’ side of the line. In SVCs only these points affect the placement of the line so all other points do not affect the fit of the model. SVMs generalise the concept further by making use of kernels, however, like SVCs, only a small subset of the data determines the fit, these points are known as support vectors. This is problematic for large datasets as we lose significant amounts of data that could be informative on how the model should be fitted. Given that our dataset is very large we decided against making use of SVMs.

Logistic regression using the same dataset as the neural networks had varying accuracies when testing across different parameters. The best one is included in the table.

## 3.2. Chosen model and implementation

In the table above, the neural network model is clearly superior. We therefore choose the neural network as our model, as we believe it will perform better in predicting cancellations.We include this model here, and provide some exposition on how it works and how it was implemented.

**Feature and layer design**

In our final data, there were 287 features, after cleaning and one-hot encoding categorical variables and additional features. However, broadly, these features are all derived from one of the 28 original features.

With this in mind, we designed a neural network with an initial layer of the 287 input variables, and a hidden layer of 28 nodes before a binary output node. The full 287 features in the input layer feed into the hidden layer, reconstructing the effect each feature had.

Testing shows this to be broadly correct once the weightings are applied.

**Neuron activation**

Additionally, we used a `relu` activation for the neurons in the hidden layer. There are so many neurons in the input layer that using the `relu` activation effectively ignores less significant factors allowing the model to be trained faster.

**Tuning process**

Hyper-parameter tuning was conducted to determine the optimal learning rate for producing a high accuracy score. We could have performed hyper-parameter tuning to find an optimal number of neurons for the hidden layer, but our testing confirmed that there were only small variations in the accuracy score as a result of this. On the other hand, using a different number of hidden neurons reduces the understandibility and explainability of the model, and the client has specified a desire for a 'white box' model. Therefore, small and insignificant gains in accuracy were traded off for an understandable model.

The following code cells document the implementation and tuning of this model, with some brief exposition on each cell's mechanical function in the process.

### 3.2.1 Initial model

The following cell splits the data into test and training data with 30% allocation to testing, and creates the layers for the neural network.

In [27]:

```
d_ = hotel_df
d_ = d_.dropna()

cols = list(d_.columns)[1:]
X = d_[cols]
y = d_['is_cancelled']

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.3, random_state=42)

input_layer = keras.layers.Input(shape=(len(cols),))
hidden_layer = keras.layers.Dense(28, activation='relu')
logit_layer = keras.layers.Dense(1, activation='sigmoid')


output = logit_layer(hidden_layer(input_layer))
model_NN = keras.models.Model(input_layer, output)
model_NN.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 286)]             0         
                                                                 
 dense (Dense)               (None, 28)                8036      
                                                                 
 dense_1 (Dense)             (None, 1)                 29        
                                                                 
=================================================================
Total params: 8,065
Trainable params: 8,065
Non-trainable params: 0
_________________________________________________________________
```

The next cells compile and fit the model to the data. 50 epochs were found to enough for further gains in the loss function to be insignificant.

In [28]:

```
model_NN.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3))
```

In [29]:

```
X_train=np.asarray(X_train).astype(np.int)
y_train=np.asarray(y_train).astype(np.int)

history = model_NN.fit(x = X_train, y = y_train, epochs = 50,
                       callbacks = False, shuffle = True,
                       validation_split = 0.1, verbose=0)
```

### Initial model results

The next two cells use accuracy metrics to evaluate the model performance. These are a mixture of imported packages and standard industry procedures to report accuracy scores and confusion matrices.

These are only initial, as the following section includes fine-tuning, but this gives us an idea for the improvement level of the model after tuning.

In [30]:

```
X_test = np.asarray(X_train).astype(np.int)
y_test = np.asarray(y_train).astype(np.int)

y_pred = model_NN.predict(X_test).flatten()
y_pred_lab = y_pred
y_pred_lab[y_pred_lab > 0.5] = 1
y_pred_lab[y_pred_lab <= 0.5] = 0

confmat = confusion_matrix(y_true = y_test, y_pred = y_pred_lab)
# Extracting the quantities from the confusion matrix
FP = confmat[0,1]
TN = confmat[0,0]
TP = confmat[1,1]
FN = confmat[1,0]

# Calculation of FPR, Recall, Precision, F1 Score and Accuracy
FPR = (FP)/(FP+TN)
Recall = (TP)/(TP+FN)
Precision = (TP)/(FP+TP)
F1 = f1_score(y_true=y_test, y_pred=y_pred_lab)
Accuracy = (TN+TP)/(TN+TP+FN+FP)

confmat = confusion_matrix(y_true = y_test, y_pred = y_pred_lab)
print('False Positive Rate (FPR): '+ '%.3f' % FPR)
print('Recall: '+ '%.3f' % Recall)
print('Precision: '+ '%.3f' % Precision)
print('F1 Score: '+ '%.3f' % F1)
print('Accuracy: '+ '%.3f' % Accuracy)


#ConfMatPlot = ConfusionMatrixDisplay(confmat,cmap="PiYG")
#ConfMatPlot.plot()
#plt.show()

print('confusion matrix: ', confmat)
```

```
2601/2601 [==============================] - 4s 2ms/step
False Positive Rate (FPR): 0.061
Recall: 0.760
Precision: 0.880
F1 Score: 0.815
Accuracy: 0.872
confusion matrix:  [[49113  3208]
 [ 7425 23482]]
```

In [31]:

```
accuracy_nn = accuracy_score(y_test, y_pred)
mse_nn = mean_squared_error(y_test, y_pred)
auc_nn = roc_auc_score(y_test, y_pred)

print("Accuracy of the Neural Network model: ", accuracy_nn)
print("Mean Squared Error of the Neural Network model: ", mse_nn)
print("Neural Network AUC: ", auc_nn)
```

```
Accuracy of the Neural Network model:  0.8722425145383765
Mean Squared Error of the Neural Network model:  0.1277574854616235
Neural Network AUC:  0.8492246738198505
```

### 3.2.2 Fine tuning

We now fine tune our neural network model by optimising the hyper parameter learning rate to maximise accuracy. We do so by fitting the model at 50 epochs for different learning rates in order to determine the optimal rate. We then use the optimal rate in our final model.

In [35]:

```
# initially tested rates
learning_rates = [0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,0.00005]

# second set of rates tested
learning_rates_v2 = [10**-(2.3+i/8) for i in range(9)]
```

In [37]:

```
# Model Tuning
```

```
def build_model(hp):
  model = keras.Sequential()
  model.add(keras.layers.Input(len(cols),))
  model.add(keras.layers.Dense(28, activation='relu'))
  model.add(keras.layers.Dense(1, activation='sigmoid'))
  hp_learning_rate = hp.Choice('learning_rate', learning_rates_v2)
  model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                loss='binary_crossentropy',
                metrics=['accuracy'])
  return model
```

In [38]:

```
tuner = keras_tuner.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=9)
```

In [39]:

```
tuner.search(X_train, y_train, epochs=50, validation_data=(X_test, y_test))
best_model = tuner.get_best_models()[0]
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 28)                8036      
                                                                 
 dense_1 (Dense)             (None, 1)                 29        
                                                                 
=================================================================
Total params: 8,065
Trainable params: 8,065
Non-trainable params: 0
_________________________________________________________________
```

In [40]:

```
best_model.summary()
history = best_model.fit(x = X_train, y = y_train, epochs = 50,
                       callbacks = False, shuffle = True,
                       validation_split = 0.1, verbose=0)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 286)]             0

 dense_2 (Dense)             (None, 28)                8036

 dense_3 (Dense)             (None, 1)                 29

=================================================================
Total params: 8,065
Trainable params: 8,065
Non-trainable params: 0
_________________________________________________________________
```

We firstly search accross a wide rand of values for Learning Rate to get a roungh estimate of which gives the best final precision. We find that the default of 0.001 is optimal.

| Learning Rate | Accuracy |
|---|---|
| 0.1 | 0.629 |
| 0.05 | 0.803 |
| 0.01 | 0.855 |
| 0.005 | 0.864 |
| 0.001 | 0.873 |
| 0.0005 | 0.869 |
| 0.0001 | 0.851 |
| 0.00005 | 0.826 |

We then hone in on the value of 0.001 exploring a logarithmically spaced range of Learning Rate centered around 0.001. We conclude that out of these 0.00158 is optimal, hence we use this value in our final model.

| Learning Rate | Accuracy |
|---|---|
| 0.00500 | 0.864 |

| Learning Rate | Accuracy |
|---|---|
| 0.00376 | 0.870 |
| 0.00282 | 0.866 |
| 0.00211 | 0.874 |
| 0.00158 | 0.874 |
| 0.00100 | 0.873 |
| 0.00089 | 0.873 |
| 0.00067 | 0.872 |
| 0.00050 | 0.869 |

In [43]:

```
X_test = np.asarray(X_train).astype(np.int)
y_test = np.asarray(y_train).astype(np.int)

y_pred = best_model.predict(X_test).flatten()
y_pred_lab = y_pred
y_pred_lab[y_pred_lab > 0.5] = 1
y_pred_lab[y_pred_lab <= 0.5] = 0

confmat = confusion_matrix(y_true = y_test, y_pred = y_pred_lab)
# Extracting the quantities from the confusion matrix
FP = confmat[0,1]
TN = confmat[0,0]
TP = confmat[1,1]
FN = confmat[1,0]

# Calculation of FPR, Recall, Precision, F1 Score and Accuracy
FPR = (FP)/(FP+TN)
Recall = (TP)/(TP+FN)
Precision = (TP)/(FP+TP)
F1 = f1_score(y_true=y_test, y_pred=y_pred_lab)
Accuracy = (TN+TP)/(TN+TP+FN+FP)

confmat = confusion_matrix(y_true = y_test, y_pred = y_pred_lab)
print('False Positive Rate (FPR): '+ '%.3f' % FPR)
print('Recall: '+ '%.3f' % Recall)
print('Precision: '+ '%.3f' % Precision)
print('F1 Score: '+ '%.3f' % F1)
print('Accuracy: '+ '%.3f' % Accuracy)

confmat
```

```
2601/2601 [==============================] - 4s 2ms/step
False Positive Rate (FPR): 0.073
Recall: 0.796
Precision: 0.865
F1 Score: 0.829
Accuracy: 0.878
```

```
Out[43]: array([[48482,  3839],
                [ 6303, 24604]])
```

```
In [47]: print ("Accuracy of best model on test set :", round(accuracy_score(y_test, y_pred)*100,3), "%")
```

```
Accuracy of best model on test set : 87.814 %
```

# 4. Discussion & Conclusions

## 4.1. Model discussion

The final model that we chose was a neural network with an input layer of 287 neurons and a hidden layer of 28 neurons. The rational for this as discussed in section 3 is that the 287 features in our cleaned and one-hot encoded data originated from an origonal set of 28 features. The aim is for the neurons the hidden layer to act as the initial variables and testing showed that this works to a degree as intended.

The activation type `relu` was used since there are such a large number of features the activation pattern of `relu` will ignore less significant features.

**Accuracy, model performance and weaknesses**

The model performs with a high degree of accuracy, 0.878 (87.8\%) and from the confusion matrix we can see that there are a roughly equal number of false posatives and false negatives with a slight bias toward more false positives. Despite the high accuracy of the model this bias suggests that we should be somehwat cautious when intentionally overbooking rooms to account for revenue loss from redicted cancellations. The reputation loss and revenue loss from a double booking is far greater than if some rooms were left empty.

Based on these results, we make two policy reccomendations for the client to improve their operational procedures and financial positions.

## 4.2. Policy 1: Charge a cancellation fee

Using our model, the client will be able to determine whether a room is likely to be cancelled. If the booking is likely to be cancelled, they can include a cancellation charge in the booking contract. There is no need to include this for booking that aren't likely to be cancelled. The effect of this would be two-fold:

- Customers may be less likely to cancel their room when a cancellation charge would affect them.
- The client retains some revenue for customers who cancel, rather than losing it entirely.

It is important to note that a cancellation fee could reduce the likelihood of booking, so it should only be tacked onto booking which are expected to be cancelled. Further data collection and modelling would be required to determine whether the psychological effect of a cancellation fee would have a serious impact on the sales process, but the impacts are limited by only charging it to customers who were likely to cancel.

The cancellation fee would vary based on the booking value, but one potential value is half of the booking cost. With an average booking cost of $340 (SiteMinder, 2016), this would retain $170 of revenue. We expect 14,343 cancellations per year, and expect our model to detect 12,593 of these. If all these bookings decide to cancel despite the cancellation fee, this would increase revenue by $2,140,810 per annum.

### 4.3. Policy 2: Rebook 50% of the rooms which are likely to be cancelled

A certain number of rooms are expected to be cancelled, meaning they will not be filled. As a result, some of these can be rebooked. The exact proportion would require further modelling and data collection to determine - we put forward 50% as the number of rooms which, when policy 1 is also applied, are still cancelled. Consideration must also be given to the need for some 'breathing room' in the bookings policy: the client does not want to be in a situation where there are not enough rooms to house all the customers who turn up. Bearing in mind that 50% is only indicative, we continue with the analysis.

We expect to detect 12,593 of the cancellations per year, and by rebooking 50% of these at an average booking cost of $340 (SiteMinder, 2016), we increase annual revenue once again by $2,140,810. This assumes that of the 50% of rebooked rooms, none of these will be cancelled. This is unrealistic, and could be adapted based on expected cancellation ratios at each hotel, using the predictive powers of the model.

Whilst our modelling on these policies is rough and preliminary, Our model enables Policy 1 to be applied in conjunction with Policy 2. If our services are retained, further investigation into the trade off between these policies, the size of the exact cancellation fee, and the proportion of predicted cancelled bookings to be re-booked could be conducted.

# 5. References

- SiteMinder, 2016. *SiteMinder report: Average booking value on direct hotel websites nearly double that on third-party channels*. Accessed 07/04/2023. URL: https://www.siteminder.com/news/average-booking-value-direct-hotel-websites-nearly-double-third-party-channels/
- Antonio, N., Almeida, A. d., Nunes, L., 2019. *Hotel booking demand datasets*. Data in Brief vol 22, February 2019, pp 41-49. Accessed 28/03/2023. Available at: https://www.sciencedirect.com/science/article/pii/S2352340918315191#f0010