proj2

April 15, 2021

1 Machine Learning in Python - Project 2

Finlay Young, Rachel Dance, Silvia Cecilia Hernandez Vargas

1.1 0. Setup

[1]:
```
# Install required packages
!pip install -q -r requirements.txt
```

[165]:
```
# Add any additional libraries or submodules below

# Display plots inline
%matplotlib inline

# Data libraries
import pandas as pd
import numpy as np
import geopy.distance as gpy
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from datetime import datetime
import pycountry as pyc
import ccy

#Web Scraping Requirement
import datapackage

#Import file
import external_functions

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

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

```
# sklearn modules
import sklearn
from sklearn import metrics
from sklearn.model_selection import GridSearchCV, KFold, cross_val_score,␣
↪train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, precision_recall_curve, roc_auc_score
from sklearn.metrics import confusion_matrix,classification_report,␣
↪precision_score
from sklearn.ensemble import RandomForestClassifier
```

[86]:
```
# Load data
d = pd.read_csv("hotel.csv")
n_observations, n_features = d.shape[0], d.shape[1]
print(f'Data contains {n_features} features and {n_observations} observations')
```

```
Data contains 30 features and 119390 observations
```

[168]:
```
d['is_canceled'].sum()/ 119390
```

[168]:
```
is_canceled                     119390
hotel                           119390
lead_time                       119390
arrival_date_year               119390
arrival_date_month              119390
arrival_date_week_number        119390
arrival_date_day_of_month       119390
stays_in_weekend_nights         119390
stays_in_week_nights            119390
adults                          119390
children                        119386
babies                          119390
meal                            119390
country                         118902
market_segment                  119390
distribution_channel            119390
is_repeated_guest               119390
previous_cancellations          119390
previous_bookings_not_canceled  119390
reserved_room_type              119390
assigned_room_type              119390
```

```
booking_changes                 119390
deposit_type                    119390
agent                           103050
company                           6797
days_in_waiting_list            119390
customer_type                   119390
adr                             119390
required_car_parking_spaces     119390
total_of_special_requests       119390
dtype: int64
```

1.2 1. Introduction

In this project we analyse hotel booking data as given in the file 'hotel.csv', which was collected by Antonio, Almeida and Nunes, 2019. This data was gathered for hotels located in Portugal: H1 at the resort region of Algarve and H2 at the city of Lisbon. There are 119,390 samples between July 1st, 2015 through August 31st 2017.

In order to enhance our analysis and to further improve the model provided for predicting a cancellation, additional datasets were introduced. A full description of these datasetsis is given in Section 2b, but briefly they consist of: - 'currency_exchange_rates_02-01-1995_-_02-05-2018.csv' - exchange rates between Jan 1995 and May 2018 - 'ISO 3155' - country three letter codes - 'countries_coords' - central coordinates of countries in dataset - 'curr_codes' - codes depicting country currencies

While cleaning and anlysing the data, we discovered the advantange of creating and removing features from the original data….Our data cleaning approach is to… Before modelling, we must clean the data to reduce as much as possible

Our modeling approach is to test 4 different models: i)Logistic Regression, ii) Support Vector Machines, iii) Classifier trees and iv) Random Classifier trees. Given that we are concerned about false positives in cancellations, we decided to stay with the Random Forest Classifier as it has a % of false positives cancellations. In all cases we split the data into two sets, where 80% (89600 points) of the data is used for training, and 20% (22400 points) for testing in all models.

Our key conslusions are … We obtain the best modeling solution with the random forest model. We have chosen this as we not only want to observe good performance in terms of number of correct predictions, but we also would like to minimise the number of false positive cases which this model accomplishes. We achieve a X% true positive prediction and X% true negative prediction, with only X% false positive and X% false negative outcomes. We appreciate there is a cost associated with both false cases, but we have prioritised minimising false positive as not only is this a loss of business, but there is an associated cost. We discovered that the top three features that contribute to a booking cancellation are: - feature 1 - feature 2 - feature 3 This is not an exhaustive list and more details are given in the Discussion.

Our limitations are… computational limitation have made us be quite specific with data cleaning to make models tractable. We have had to make many assumptions which are detailed below. we do not know the full cost of a false positive cancellation which would allow us to to a more thorough analysis of the costs. Key assumptions: - data features that are sparsely seen, or very unbalanced in the data are not predictively meaningful and are rejected -duplicated information is deleted, e.g.

1.3 2. Exploratory Data Analysis and Feature Engineering

1.3.1 Data Cleaning

Core issues The following observations and actions were taken from initial analysis of the source data provided by the client:

`Children` : nan' valued children were replaced with zeros - this makes the assumption of an error in data entry, i.e. it was omitted when filling this field in for no children.

`company` & `agent` :94% of the company feature is null, which we might expect is an error from filling this field. In the same way, 13% of the `agent` feature is null. However, given that many `agent` values would correspond to a specific company, this still allows for the booking to be attributed to a known agent with more granularity than `company`. Therefore, even both features are sparsly populated we will retain for further analysis.

`country` : Null (415 instances) country codes have been assumed by default to PRT (Portugal), given that the hotels are in Portugal and make up 32% of all bookings are from Portugal. On the other hand, this variable should follow a 3 character code (ISO 3155); nevertheless, there is an alpha-2 code for "CN", which corresponds to China. This has been changed from "CN" to "CNH".

`adr` : There exists data where the feature 'adr' contains zero as a value, which we might expect to be an error in the data as it is not possible to have no expenses during the accomodataion. Our first attempt was to replace this 'zero' values with the average expenses given the reserved room type, assigned room type and type of meal. However, the percentage of the data with no value is low in comparison with the whole data (1.6%). Therefore, we decided to delete the rows which this characteristic from the data.

In the source dataset there are ~32,500 duplicate rows of data. However, for the porpouse of this task as we are investigating what will indicate cancellations, and not the overall booking, mainting this data will be useful for achieving this goal. Furthermore, it would be fair to assume these were all "double bookings" where multiple groups of friends (or coincidental) booking for the same holiday in different transactions, or for a popular event - for this reason we will not be dropping these duplicates. In fact, 90% of the duplicates are reservations for the City hotel, and thus by removing this data we might get rid of potentially useful data.

[4]:
```
#Changing nan children to zero childrem
d['children'] = np.where(d['children'].isnull(),0,d['children'])

# Update China CN-> CNH. & Adding PRT to nan Countries.
d['country'] = d['country'].replace(np.nan, 'PRT')
d['country'] = d['country'].replace('CN', 'CNH')
```

1.3.2 Feature Analysis

Booking type We investigate the relationship between features `distribution_channel`, `market_segment`, `company` and `agent`, in order to determine if these features are dependent on one other, and not provide add value to the model. In the following first graph we can see how `market_segment` can be considered as a child of `distribution_channel`. Therefore, keeping only `market_segment` will provide a more meaningful description without loss of information. We also see that five types of `market_segment` are dominant over the others (see bullet 2 below).

In the second graph we analyse the interaction between `market_segment`, `company` and `agent`. We find the majority of the data provided for `company` is null (>90%). Additionally, there are agents in which the same agent appears in multiple companies, thus agents can not be related to a single company.

From this, the following decision were taken:

- Drop `distribution channel` field and only retian the `market segment` as our feature for describing where bookings originate from.
- Retain the market segment for: 'Online TA', 'Offline TA/TO', 'Groups', 'Direct' and 'Corporate'; and relabeled the rest of the market segments as "other".
- Drop `agents` and `company` fields, as they add complexity to the data but not necessarily adding much in the way of insight.
- Company & Agent were both altered to include a "Company ID" & "Agent ID" prefix for easier reading.

[5]:
```
# Form datafram for plotting
d_cat_cut = d.groupby(['distribution_channel', 'market_segment']).agg({'meal':
 ↪'count'}).reset_index()
d_cat_cut.rename(columns = {'meal' : 'Counts'}, inplace = True)

fig = px.parallel_categories(d_cat_cut, color="Counts",␣
 ↪color_continuous_scale=px.colors.sequential.Inferno,
    title = 'Relation between distribution channel and market segment')
fig.show()
```

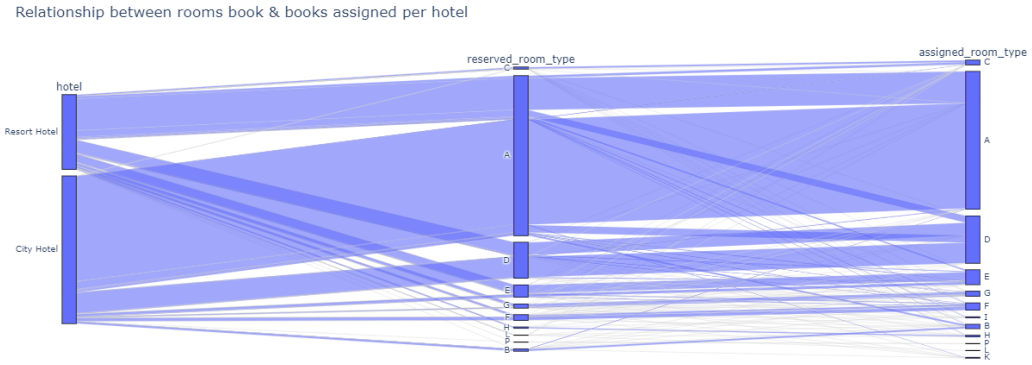
[6]:
```
# Add "Agent ID & "Company ID" prefixes
d['company'] = np.where((d.company.isnull()),np.nan,'Company ID: '+␣
 ↪d["company"].map(str))
d['agent'] = np.where((d.agent.isnull()),np.nan,'agent ID: '+ d["agent"].
 ↪map(str))

# Table data showing the market sector, company and agent dependency
# Visualisation of table data:
booking_means = d.groupby(['market_segment',"company", "agent",]).agg({'meal':
 ↪'count'}).reset_index()
booking_means.rename(columns = {'meal' : "Counts"}, inplace = True)
fig = px.treemap(booking_means, path=['market_segment',"company", "agent",],␣
 ↪values='Counts',
            title = 'Relation between market segment, company and agent feature.
 ↪')
fig.show()
```

[7]:
```
market_retain = ['Online TA', 'Offline TA/TO', 'Groups', 'Direct', 'Corporate']
d['market_segment'] = np.where( d['market_segment'].isin(market_retain) ,␣
 ↪d['market_segment'], 'Other')
```

Reserved room types In this case, we make an analysis of the interaction between `reserved_room_type` and `assigned_room_type`. From the following plot, it can be seen that for both the resort hotel and city hotel, in the vast majority of the cases, the room reserved is the same as the room assigned. Therefore, we assume information in these two features might be equally the same. We expect that, in order to get a better prediction, a more important feature than any individual room type is whether any customer gets what they asked for - i.e. the reserved room matches the room recieved.

From the previous analysis, the following decision were taken:

- Retain the the feature `reserved_room_type` as this might be relevant for predicting a cancellation. For example, this feature can have an interaction with the feature `deposit_type` as a guest might be less probable to cancel a reservation given that he made a `Non Refund` deposit for an expensive type of room.
- Create a new feature `room_granted` with a boolean type, where if true then the reserved room type is the same as the room assigned.



[116]:
```
fig = px.
 ↪parallel_categories(d[['hotel','reserved_room_type','assigned_room_type' ]],␣
 ↪color_continuous_scale=px.colors.sequential.Inferno,  title = 'Relationship␣
 ↪between rooms book & books assigned per hotel ')
fig.show()
```

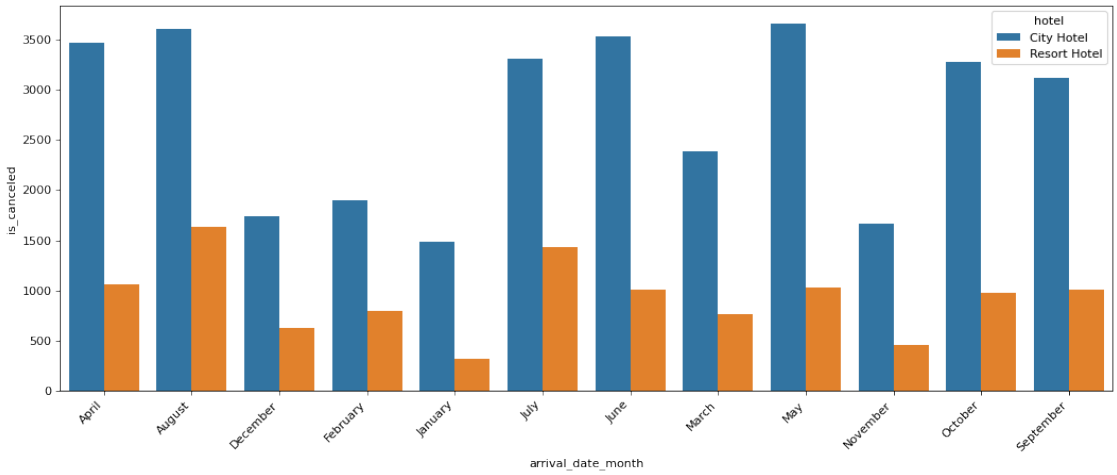
[9]:
```
d['room_granted'] = np.where(d["reserved_room_type"] ==␣
 ↪d["assigned_room_type"], True, False)
```

Booking date The date of the booking and of the hotel stay is kept in the dataframe, but arrival year & week number are omitted. Year is non-sensical as described above, and the week number is a duplication of information we can more meaningfully obtain from the month of the booking. The month feature is more than adequate feature to identify seasonal changes. The dataset consists of 24 months of data spanning three years, and as such we only have one complete year, so we cannot do a comparison of a set month over the three years, but we will keep month as a feature.

The following plot shows that for both the city and resort hotel, from November to January, number

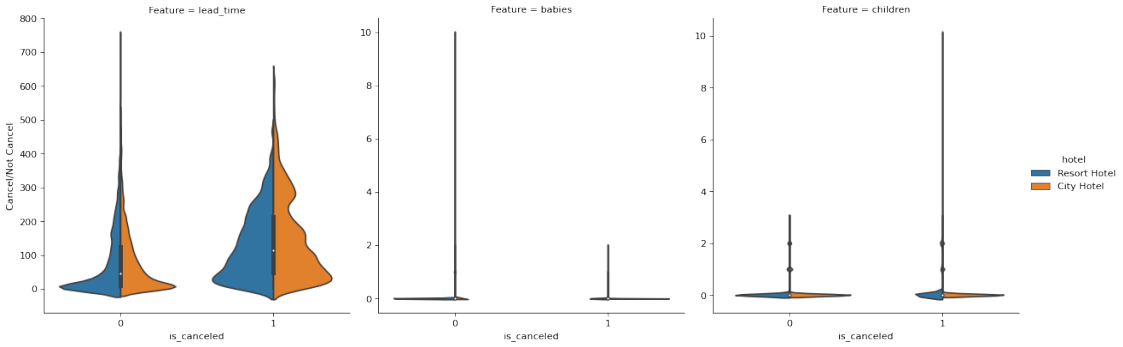
of cancellations register the lowest number of occurrence. Eventhough, for the rest of the months both type of hotels does not necessarilly have the same correlation their behaviour is merely the same.

[10]:
```
months_cancel = d.groupby(['arrival_date_month','hotel']).agg({'is_canceled':
↪'sum'}).reset_index()

fig, ax = plt.subplots(figsize=(16, 6))

#sns.barplot(x ='arrival_date_month', y = 'adults',hue = "is_canceled", data =␣
↪d)
sns.barplot(x ='arrival_date_month', y = 'is_canceled',hue = 'hotel', data =␣
↪months_cancel , )

ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
↪horizontalalignment='right')
plt.show()
```



Lead time, babies and children All Violin plots are scaled by area to make the comparable. We see right away that very few bookings have children, and fewer still have babies. This is marginally lower numbers in the City hotel than in the Resort which is not surprising. For lead time, in both hotels the frequency of is canceled tapers off with lead time - i.e. are people more likely to cancel the later they book.

[136]:
```
g = sns.catplot( data = d.melt(id_vars = ['is_canceled','hotel'],value_vars =␣
↪['lead_time','babies','children'],var_name = 'Feature',value_name = 'Cancel/
↪Not Cancel'),
                    x = 'is_canceled', y = 'Cancel/Not Cancel', hue = 'hotel',␣
↪kind = 'violin', col = 'Feature' , scale = 'area', sharey= False, split=True)
```



1.3.3 Additional Data sources & pre-processing

Aside from cleaning & preprocessing the main hotels data, there was considerable processing of data which was joinned to give additional features. The summary of these additional datsets is:

cur_code - Dataset Info: This is a Google Dataset which maps 3-Alpha Currency code & Name to a 3-Alpha Country code. - **Decription of Preprocessing**: This dataset was used to enrich fx_rates with the Aphabetic Currency code. The preprocessing here was an iterative task, and involved renaming many of the currencies within fx_rates to ensure a match, which would then allow for the enriched fx_rates .

coords - Dataset Info: This is also a Google Developers dataset. which provides a 2-Alpha character code mapping to Londitude & Latidude of listed countries. - **Decription of Preprocessing**: Using the Longditude & Latitude we were able to calculate the Orthodoric (spherical) distance between each country of the world to Portugal, where both the hotel resorts are. This distance in kilometers was then able to be joined onto the main 'Hotels.csv' dataset. This enrichment will be able to give further depth & understanding on each guests decision to cancel if we assume that a greater distance implies a longer transportation time, and more expensive means of transportation.

comp_countries - Dataset Info: This DataFrame hold geographical data which allows for better description of a country, as well as holding the Countries Currency code, and 2&3-Alpha character codes. This is a published dataset on Datahub - **Decription of Preprocessing**: The majority dataset was not used, the data in this dataset was for mapping data between 'hotels.csv', coords and fx_date to ensure that there are consistent country & currency codes to join on.

fx_rates - Dataset Info: This is a dataframe of daily currency Exchange rates from 1995 - 2018, for 50+ currencies, with an exchange rate with respect to US Dollars. This is an available Kaggle dataset. - **Decription of Preprocessing**: Given that there are exchange rates from 1995 - 2018, this data was needed to be cut down to start at the earliest booking date from 'hotels.csv' up to the last arrival date.The rationale for this was to ensure we could enrich the main dataset with an exchange rate from the customers home currency at the point of booking their holiday (booking date) and compare this with an exchange rate percentage on their arrival (arrival date) of their holiday. The data required to first be calculated from US Dollar exchange rates into Euro exchange rates. The data also had many missing dates, which were sundays, which meant that many arrival & booking dates were not within the fx_rates dataset. To fix this, all dates were added between the first booking date & arrival date, and the missing exchange rates were linearly interpolated.

To summarise, the additional data sources shall allow for the dataset to hold how far away from the booking country is, and the relative difference in currency strength between the time of the guest booking to arriving.

[11]:
```
cur_code = pd.read_csv("curr_codes.csv")
coords = pd.read_csv("countries_coords.csv")
fx_rates = pd.read_csv("/work/currency_exchange_rates_02-01-1995_-_02-05-2018.
 ↪csv")

data_url = 'https://datahub.io/core/country-codes/datapackage.json'
# to load Data Package into storage
package = datapackage.Package(data_url)
# to load only tabular data
resources = package.resources
for resource in resources:
    if resource.tabular:
        comp_countries = pd.read_csv(resource.descriptor['path'])
```

[12]:
```
#format arrival date
d['month'] = pd.to_datetime(d.arrival_date_month, format='%B').dt.month
d['day'] = pd.to_datetime(d.arrival_date_day_of_month, format='%d').dt.day
d['Year'] = pd.to_datetime(d.arrival_date_year, format='%Y').dt.year
d['arrival_date']  = pd.to_datetime(d[['Year','month','day']], format =␣
 ↪'%Y%m%d')#.dt.date
d['booking_date'] = d['arrival_date'] - pd.to_timedelta(d['lead_time'],␣
 ↪unit='d')

min_date = d['arrival_date'].min()
max_date = d['arrival_date'].max()
min_booking_date = d['booking_date'].min()

# Join comp_countries data to d (main)
comp_countries = comp_countries.loc[:,␣
 ↪['ISO3166-1-Alpha-3','ISO3166-1-Alpha-2','ISO4217-currency_country_name','ISO4217-currency_
 ↪Name', 'Continent']]
d = pd.merge(d,comp_countries,left_on='country', right_on='ISO3166-1-Alpha-3',␣
 ↪how = 'left')
d.rename(columns = {'ISO4217-currency_alphabetic_code' : 'Currency_code'},␣
 ↪inplace = True)

#Reform Co-Ordinates into list within DF
coords['co_ords'] =  coords[['latitude', 'longitude']].values.tolist()
coords = coords.dropna()

# Set Portugal as basis
portugal = coords['co_ords'].loc[coords["name"] == 'Portugal'].values.tolist()
```

```
#Compute the distance in KM from all countries to Portugal
coords['distance(km)'] = coords.apply(lambda coords: gpy.
 ↪great_circle(portugal,coords['co_ords']).km, axis = 1).round(decimals=2)
# Join to main Dataframe
coords_cut = coords[['country','distance(km)']]
coords_cut.columns = ['ISO3166-1-Alpha-2','distance(km)']
d = pd.merge(d,coords_cut ,on='ISO3166-1-Alpha-2', how = 'left')

#Rename the columns within FX Rates Column
#Select Columns from cur_code
cur_code = cur_code.loc[:, ['country Entity','Currency','AlphabeticCode']]

# Get all FX Rate currency codes (don't include Date field)
fx_codes = pd.DataFrame(fx_rates.columns[1:], )

#Test the results of the merge codes which match well currently
cur_mapping_test = pd.merge(fx_codes,cur_code,left_on=0, right_on='Currency',␣
 ↪how = 'left')

#check for non matching
#sorted(cur_mapping_test[0][cur_mapping_test['Currency'].isna()])
# Create Update Dict:
null_curr_map = sorted(cur_mapping_test[0][cur_mapping_test['Currency'].isna()])
new_curr_map = ('Bahraini Dinar','Pula','Yuan Renminbi','Forint','Iceland␣
 ↪Krona','Rupiah','New Israeli Sheqel',
                'Yen','Tenge','Won', 'Mauritius Rupee','Pakistan␣
 ↪Rupee','Zloty','Qatari Rial','Saudi Riyal',
                'Rand','Baht','Trinidad and Tobago Dollar','UAE Dirham','Pound␣
 ↪Sterling','US Dollar')

# Create Dataframe to update old values within FX Code to (i) join to Main and␣
 ↪(ii) update fxrate columns
fx_rename = dict(zip(null_curr_map, new_curr_map))

# Update FX Rates - ready for updateing fx rates
fx_rates.rename(columns = fx_rename, inplace = True)

# You need this mapping for when the FX Rate data is processed & pivoted.
cur_mapping =  pd.merge(fx_codes,cur_code,left_on=0, right_on='Currency', how =␣
 ↪'left')
```

Exchange rates for all common currencies have been sourced from 1995 - 2018. This data is used to investigate the strength of a guests currency from the time of booking, to when the holiday is close by. This requires some preprocessing of the data for it to be added to the main dataset.

There are a number of unobtainable currency exchange rates where the country's currency data is not readily available (for free). For this reason, we have assumed a 0 value to populate in all the

`FX_delta` engineered feature, the FX difference between the booking & arrival date. This is not a perfect assumption, as these currencies are likely to be more volatile for smaller countries, however there are only ~4000 instances out of the ~119,000 reservations. The majority of these countries are also being re-labelled as "other" as they are not within the top 5% of visting countries. Also, there is a majority of these currencies which are European countries which we would expect to have a currencies tracking the Euro.

[13]:
```
#Reformat Date column
fx_rates['Date'] = pd.to_datetime(fx_rates['Date'], format='%Y-%m-%d')
#fx_rates['Date']  = pd.to_datetime(fx_rates['Date']).dt.date

#Reform into Euro conversion rates
fx_rates.iloc[:,1:] = fx_rates.iloc[:,1:].div(fx_rates.Euro, axis=0)

#Cut Dates so that theres only the date from the earliest booking to the last␣
 ↪booking
fx_rates = fx_rates[fx_rates['Date'].between(min_booking_date,max_date)]

# Create rows full of "nan" for the missing dates within range
fx_rates = pd.merge(pd.DataFrame({"Date" : pd.date_range(min_booking_date,␣
 ↪max_date, freq='D')}),fx_rates ,left_on='Date', right_on='Date', how =␣
 ↪'outer')

# Interpolate to remove missing true null values  and newly created values for␣
 ↪dates
fx_rates = fx_rates.interpolate()

#Pivot Data vertically
fx_rates = fx_rates.melt(id_vars = ['Date'],var_name='Currency',␣
 ↪value_name='FX_Rates')

# Join to get currency code
d.rename(columns = {'ISO4217-currency_alphabetic_code' : 'Currency_code'},␣
 ↪inplace = True)
cur_code.rename(columns = {'AlphabeticCode' : 'Currency_code'}, inplace = True)
fx_rates = pd.merge(fx_rates, cur_code,on='Currency', how = 'left')
fx_rates = fx_rates[['Date','Currency','Currency_code','FX_Rates']]

fx_rates = fx_rates.drop_duplicates()

# get FX rate as at arrival date
d = pd.merge(d, fx_rates,left_on=['Currency_code','arrival_date'], right_on =␣
 ↪['Currency_code','Date'] ,how = 'left' )
d.rename(columns = {'FX_Rates' : 'FX_Rates_on_arrival'}, inplace = True)
# get FX rate as at booking date
d = pd.merge(d, fx_rates,left_on=['Currency_code','booking_date' ], right_on =␣
 ↪['Currency_code','Date'],how = 'left' )
```

```
d.rename(columns = {'FX_Rates' : 'FX_Rates_on_booking'}, inplace = True)
d.drop(['Currency_x'],axis =1, inplace=True )
# Drop useless columns from merges
d.drop(['Date_x','Date_y'],axis =1, inplace=True )
# Calculate the % Loss between Booking & Arrival Dates
d['FX_Delta'] = np.where(((d.Currency_code =='EUR')|(d['FX_Rates_on_arrival'] -␣
↪d['FX_Rates_on_booking'] ==0)),0, (d['FX_Rates_on_arrival'] -␣
↪d['FX_Rates_on_booking'])/ d['FX_Rates_on_booking'])
```

Cancellations by country analysis Here we look to see proportionally whether most cancellations are originated from any countries in particular. We plot a map which visualises data by year. We can see by moving the slider that this varies from year to year, while there are other countries as Russia which maintain the same proportion each year. We do not include the year of booking in the modeling as, for example, saying that a booking made in 2015 is more likely to be canceled is meaningless for prediction.

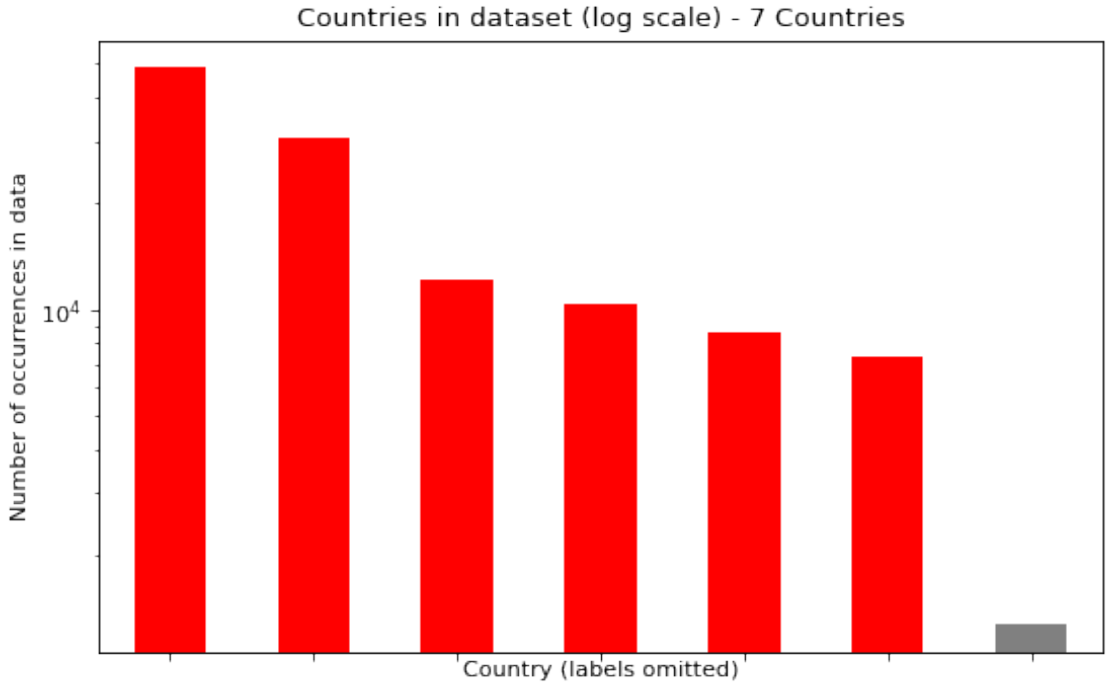
[14]:
```
map_data = d.groupby(['country', 'hotel', 'arrival_date_year',]).
↪agg({'arrival_date_month':'count', 'is_canceled':'mean'}).reset_index()
map_data.rename(columns = {'arrival_date_month': 'Total_booking', 'is_canceled':
↪ 'Cancellation_proportion'}, inplace = True)
map_data.sort_values(by=['arrival_date_year'], inplace=True)
fig = px.choropleth(map_data, locations="country",␣
↪color="Cancellation_proportion", hover_name="Total_booking",
                    animation_frame="arrival_date_year", range_color=[0,1],
                    title ='Proportion of cancellations per country years 2015,␣
↪2016 & 2017' )
fig.show()
```

As seen below there are 177 unique countries in this data. Here we illustrate that some of them only appear very sparsely in the data, and we aim to identify and aggregate them into a single 'other' category, which can be used more meaningfully for our porpouse. Therefore, to train our model we retain the original label of the five most repeated countries in the dataset.

[162]:
```
num_countries = d['country'].nunique()
print(f'There are {num_countries} countries in the original data, compared with␣
↪{n_observations} observations.')
''' Plot the counts of individual countries on a log-lin scale '''
VV = pd.DataFrame(d['country'].value_counts().reset_index())
VV['clrs'] = np.where( (VV['index'].
↪isin(['PRT','GBR','OTHER','ESP','FRA','DEU'])) , 'red', 'grey')
var_count_df = d.country.value_counts()

var_count_df.plot.bar(var_count_df, logy=True, color = VV['clrs'])
plt.tick_params(labelbottom=False)
plt.title(f'Countries in dataset (log scale) - {num_countries} Countries')
plt.ylabel('Number of occurrences in data')
plt.xlabel('Country (labels omitted)')
```

```
percent_to_keep = 0.05 # Choose to keep countries apprering in at least␣
 ↪'percent_to_keep' observations
print(f'We are keeping {percent_to_keep*100}% of {len(var_count_df)} countries.␣
 ↪Threshold = {percent_to_keep * n_observations}')#\n These are:')
var_count_df[var_count_df > (percent_to_keep*len(d['country']))]

# Create list of countries to remove (compliment of previous line)
var_count_df = var_count_df[var_count_df < (percent_to_keep*len(d['country']))]
# Get indices of the ones to keep
indices_ = []
for i in range(len(var_count_df)):
    indices_.append(var_count_df.index[i])
#indices_
#print('Countries to drop: \n',list(indices_))
d['country'] = np.where(d['country'].isin(indices_),'OTHER',d['country'])
print('Named Countries Retained:' ,d['country'].unique())

#Replace the countries...
# Look in d for entries
#for i in range(len(d['country'])):
 #   if d['country'][i] in var_count_df:
  #      d['country'][i] = 'OTHER'
```

```
There are 7 countries in the original data, compared with 119390 observations.
We are keeping 5.0% of 7 countries. Threshold = 5969.5
Named Countries Retained: ['PRT' 'GBR' 'OTHER' 'ESP' 'FRA' nan 'DEU']
```



Correlation 'matrix': The following is givin in place of a basic correlation matrix. The numeric values here show that `lead_time` and `total_of_special_requests` fields are most correlated with the `is_canceled` field. This gives us an initial rough guess what the most important field we might expect to see from our modelling. Note that values close to 0 are not important for interpreting cancellations.

[20]:
```
''' Compute pairwise correlation of columns, excluding NA/null values (docs)'''
d.corr()["is_canceled"].sort_values(ascending=False)[1:]
```

[20]:
```
lead_time                        0.293123
room_granted                     0.247770
previous_cancellations           0.110133
adults                           0.060017
days_in_waiting_list             0.054186
adr                              0.047557
FX_Delta                         0.037970
stays_in_week_nights             0.024765
arrival_date_year                0.016660
Year                             0.016660
month                            0.011022
arrival_date_week_number         0.008148
children                         0.005036
stays_in_weekend_nights         -0.001791
FX_Rates_on_booking             -0.002421
FX_Rates_on_arrival             -0.002496
arrival_date_day_of_month       -0.006130
day                             -0.006130
babies                          -0.032491
previous_bookings_not_canceled  -0.057358
is_repeated_guest               -0.084793
distance(km)                    -0.133363
booking_changes                 -0.144381
required_car_parking_spaces     -0.195498
total_of_special_requests       -0.234658
Name: is_canceled, dtype: float64
```

Required Car parking spaces We look initially to see how balanced the data is with regard to `required_car_parking_spaces` feature.

[21]:
```
d['required_car_parking_spaces'].value_counts()
```

[21]:
```
0    111974
1      7383
2        28
```

```
3         3
8         2
Name: required_car_parking_spaces, dtype: int64
```

[22]:
```
percent_needing_spaces = d['required_car_parking_spaces'].value_counts()[0]/
↪len(d['required_car_parking_spaces'])
print(f'Shows that {round(percent_needing_spaces,2)*100}% of bookings required␣
↪no parking at all.')
```

```
Shows that 94.0% of bookings required no parking at all.
```

This brief grouby table also briefly highlights the impact - all of the 44224 cancellations occur in bookings with 0 parking spaces, therefore the feature is not going to be able to enhance model predictions and will be removed.

[71]:
```
pd.DataFrame(d.groupby(['required_car_parking_spaces', 'is_canceled']).
↪agg({'is_canceled':'sum'}))
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-71-0e9a3bf661b4> in <module>
----> 1 pd.DataFrame(d.groupby(['required_car_parking_spaces', 'is_canceled']).
 ↪agg({'is_canceled':'sum'}))

/usr/local/lib/python3.7/site-packages/pandas/core/frame.py in groupby(self, by␣
 ↪axis, level, as_index, sort, group_keys, squeeze, observed, dropna)
   6722             squeeze=squeeze,
   6723             observed=observed,
-> 6724             dropna=dropna,
   6725         )
   6726

/usr/local/lib/python3.7/site-packages/pandas/core/groupby/groupby.py in␣
 ↪__init__(self, obj, keys, axis, level, grouper, exclusions, selection,␣
 ↪as_index, sort, group_keys, squeeze, observed, mutated, dropna)
    566                 observed=observed,
    567                 mutated=self.mutated,
--> 568                 dropna=self.dropna,
    569             )
    570

/usr/local/lib/python3.7/site-packages/pandas/core/groupby/grouper.py in␣
 ↪get_grouper(obj, key, axis, level, sort, observed, mutated, validate, dropna)
    809                 in_axis, name, level, gpr = False, None, gpr, None
    810             else:
--> 811                 raise KeyError(gpr)
    812         elif isinstance(gpr, Grouper) and gpr.key is not None:
    813             # Add key to exclusions
```

```
KeyError: 'required_car_parking_spaces'
```

Dropping variables from the dataframe Here we drop the variables discussed that we do not need from the dataframe, and any additional fields that have arisen naturally as part of engineering. Only those that were part of the original dataframe have been discussed. **guys did we actually discuss all of these???**

[24]:
```
# Dropping dependant fields not required for models. Full Dataset is stored in␣
↪d_FULL for data analysis at the end of the models.
d_FULL = d.copy()
d.drop(['ISO3166-1-Alpha-3',
        'ISO3166-1-Alpha-2',
        'ISO4217-currency_country_name',
        'Region Name',
        'FX_Rates_on_booking',
        'FX_Rates_on_arrival',
        'Continent',
        'Currency_code' , 'distance(km)','arrival_date' , 'booking_date',
        'arrival_date_week_number','arrival_date_day_of_month',
        'Currency_y', 'Currency_code','agent','company','day',
        'Year','month','arrival_date_year','assigned_room_type',
        'required_car_parking_spaces','distribution_channel'],axis =1,␣
↪inplace=True )

d['FX_Delta'] = d['FX_Delta'].replace(np.nan, 0)
```

1.4 3. Model Fitting and Tuning

In this section you should detail your choice of model and describe the process used to refine and fit that model. You are strongly encouraged to explore many different modeling methods (e.g. logistic regression, classification trees, SVC, etc.) but you should not include a detailed narrative of all of these attempts. At most this section should mention the methods explored and why they were rejected - most of your effort should go into describing the model you are using and your process for tuning and validatin it.

This section should also include the full implementation of your final model, including all necessary validation. As with figures, any included code must also be addressed in the text of the document.

General train and test splitting for all models so that we can directly compare the output probabilities.

[25]:
```
#Code for onehotting
d_onehot = d.copy()
d_onehot = pd.get_dummies(d_onehot)
d_onehot_labels = d_onehot.columns

#initially work on a sample to make life faster :)
```

```
#d2_sample = d_onehot.sample(frac=0.01, random_state=1234)
#X = d2_sample.drop('is_canceled', axis = 1)
#y = d2_sample['is_canceled']

X = d_onehot.drop('is_canceled', axis = 1)
y = d_onehot['is_canceled']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)

print("Dimension of Train set",X_train.shape)
print("Dimension of Test set",X_test.shape,"\n")
# Split Dataset into Continous and Categorical features in training set
num_cols = X_train._get_numeric_data().columns
print("Numerical Features in Dataset:",num_cols.size)
print("Categorical Features in Dataset:",(len(X_train.columns) - num_cols.size))
```

```
Dimension of Train set (95512, 63)
Dimension of Test set (23878, 63)

Numerical Features in Dataset: 63
Categorical Features in Dataset: 0
```

1.4.1 Functions for modeling

[26]:
```
truth = pd.Categorical.from_codes(y, categories = ('not␣
↪cancellation','cancellation'))

def confusion_plot(truth, probs, threshold=0.5, plot=True):
    '''Function to produce the confusion matrix data and a stripplot to
    visualise the data of the confusion matrix for a given threshold
    Input: truth
           probs
           threshold: default 0.5
           plot: default True, whether or not to produce plot as stdout
    Output: confusion matrix data from sklearn.metrics.confusion_matrix for␣
↪data y
     '''
    d = pd.DataFrame(
        data = {'spam': y, 'truth': truth, 'probs': probs}
    )

    # Create a column called outcome that contains the labeling outcome
    # for the given threshold
    d['outcome'] = 'other'
    d.loc[(d.spam == 1) & (d.probs >= threshold), 'outcome'] = 'true positive'
    d.loc[(d.spam == 0) & (d.probs >= threshold), 'outcome'] = 'false positive'
    d.loc[(d.spam == 1) & (d.probs <  threshold), 'outcome'] = 'false negative'
```

```
    d.loc[(d.spam == 0) & (d.probs <  threshold), 'outcome'] = 'true negative'

    if plot == True:
        # Create plot and color according to outcome
        plt.figure(figsize=(12,4))
        plt.xlim((-0.05,1.05))
        sns.stripplot(y='truth', x='probs', hue='outcome', data=d)
        plt.axvline(x=threshold, linestyle='dashed', color='black', alpha=0.5)
        plt.title("threshold = %.2f" % threshold)
        plt.show()

    return sklearn.metrics.confusion_matrix(y_true=d.spam, y_pred=d.probs >=␣
↪threshold)
```

[27]:
```
def true_false_evolution(probs, thresholds_input = np.linspace(0,1,10)):
    '''Plots the evolution of tp, fp, fn, tn with threshold. Calls the␣
↪confusion_matrix
    function to calculate the tp, fp, fn, tn for each input thresholds
    Input: array of threshhold_inputs.
    Output: Plot visualisation to allow decision making on threshold '''
    # initialise thresholds
    thresholds_plot = np.empty((4,len(thresholds_input))) #dont laugh at my␣
↪coding too hard...
    #Create matrix of tp,fp,fn,tn data over thresholds
    for i in range(len(thresholds_input)):
        conf_matrix = confusion_plot(truth, probs,␣
↪threshold=thresholds_input[i], plot=False)
        tp, fp, fn, tn = conf_matrix[0][0], conf_matrix[0][1],␣
↪conf_matrix[1][0], conf_matrix[1][1]
        thresholds_plot[0][i] = tp
        thresholds_plot[1][i] = fp
        thresholds_plot[2][i] = fn
        thresholds_plot[3][i] = tn
    categories = ['tp','fp','fn', 'tn']
    for i in range(0,4):
        plt.plot(thresholds_input, thresholds_plot[i], label=categories[i])
        plt.xlabel('Threshold Value in [0,1]'), plt.ylabel('Count')
        plt.title('Evolution of TP, FP, FN, TN with Threshold.'), plt.legend()
```

[28]:
```
def probs_histogram(model_used):
    '''Short tool for assessing balance of data.
    Input: the model we wish to evaluate
    Output: Prints the number of 0 and 1 in the original data to show balance
            Plots probability distribution of the model output (predictions) '''
    print('Count of labels 0 and 1 in the data to indicate balance:
↪\n',d['is_canceled'].value_counts())
```

```
    plt.hist(model_used.predict_proba(X_test)[:,1].reshape(-1))
    plt.xlabel('Probability (binned)'), plt.ylabel('Count')
    plt.title('Distribution of probability amongst observations.'), plt.show()
```

[70]:
```
def roc_plot(y_true, y_pred, plot = True):
    """ Draw an ROC curve and report AUC
    Input: true y data, predicted y data
    Output: (to stdout) display ROC curve
    returns dataframe containing roc curve information
    """
    roc = pd.DataFrame( data = np.c_[sklearn.metrics.roc_curve(y_true, y_pred)],
        columns = ('fpr', 'tpr', 'threshold'))

    if plot == True:
        sns.lineplot(x='fpr', y='tpr', data=roc, ci=None)
        plt.plot([0,1],[0,1], 'k--', alpha=0.5) # 0-1 line
        plt.title("ROC curve (auc = %.4f)" % sklearn.metrics.
↪roc_auc_score(y_true, y_pred))
        plt.show()

    return roc, sklearn.metrics.roc_auc_score(y_true, y_pred)
```

[30]:
```
def plot_feature_weights(importances, Tolerance = 0.5, plot=True):
    ''' Plot the feature importances predicted by a model.
    Input: importances from the model (ndarray), plot: boolean to display plot
           Tolerance, decimal (>0) above which we consider features significant.
    Output: Displays  feature importances as bar chart if plot = True
            returns dataframe of model coefficients for further analysis. '''

    model_Coefs = pd.DataFrame(importances, columns = {'coefficients'})
    model_Coefs['Feature'] = d_onehot_labels[1:]
    model_Coefs = model_Coefs.sort_values(by = 'coefficients' )
    model_Coefs["Coefficients Correlation"] = np.
↪where(model_Coefs["coefficients"]<0, 'Negative Weights', 'Positive Weights')

    #model_Coefs.drop(model_Coefs[(model_Coefs['coefficients'].eq(0)) |␣
↪(abs(model_Coefs['coefficients']) < Tolerance )].index, inplace = True)
    palette = {'Negative Weights': "#ff0000", 'Positive Weights': "#00ff00"}

    if plot == True:
            fig, ax = plt.subplots(figsize=(20, 7))
            feature_plot = sns.barplot(data = model_Coefs.
↪drop(model_Coefs[(model_Coefs['coefficients'].eq(0)) |␣
↪(abs(model_Coefs['coefficients']) < Tolerance )].index),
                    y='coefficients',x = 'Feature' , hue = 'Coefficients␣
↪Correlation' , ax=ax , palette=palette )
```

```
            feature_plot.set_xticklabels(feature_plot.get_xticklabels(),␣
↪rotation=45, horizontalalignment='right' )
            #ax.set_xlabel('Features within the Lasso Model')
            ax.set_ylabel('Coefficient values')
            plt.title(f'Coefficients of the Features')# \n (Coefficients >␣
↪absolute value {Tolerance})')
            plt.show()

    return model_Coefs
```

[31]:
```
def display_results_report(model_fitted, X_train, X_test, y_train, y_test,␣
↪y_hat):
    ''' Displays the '''
    # View the accuracy score
    print(f'_____ GridSearchCV Model Results_____')
    print("Best Random forest params: ", model_fitted.best_params_)
    print("Training set score for Random Forest: %f" % model_fitted.
↪score(X_train , y_train))
    print("Testing  set score for Random Forest: %f" % model_fitted.
↪score(X_test  , y_test ))

    # Now inspect the final model
    final_model = model_fitted.best_estimator_
    #Y_pred_sum = model_fitted.predict(X_test)

    print('_____ Final Random Forest Model Results_____')
    print('_____ Confusion Matrix_____ : \n',confusion_matrix(y_hat,y_test))
    print('_____ Classification Report_____ :␣
↪\n',classification_report(y_test,y_hat))
    print('% False positive for no␣
↪cancellation',round(confusion_matrix(y_hat,y_test)[0,1]/␣
↪sum(confusion_matrix(y_hat,y_test)[0,:]),5)*100)
    print('% False positive for␣
↪cancellation',round(confusion_matrix(y_hat,y_test)[1,0]/␣
↪sum(confusion_matrix(y_hat,y_test)[1,:]),5)*100)
```

1.4.2 Logistic regression

We trial the logistic model here

Results from Logistic Regression model:

[172]:
```
logreg_model = external_functions.logistic_pipeline(X_train, y_train)

Y_hat_logreg = logreg_model.predict(X_test)
print('Confusion Matrix : \n',confusion_matrix(Y_hat_logreg,y_test))
print("Accuracy:",logreg_model.best_score_,'\n')
```

```
print('% False positive for no␣
 ↪cancellation',round(confusion_matrix(Y_hat_logreg,y_test)[0,1]/␣
 ↪sum(confusion_matrix(Y_hat_logreg,y_test)[0,:]),5)*100)
print('% False positive for␣
 ↪cancellation',round(confusion_matrix(Y_hat_logreg,y_test)[1,0]/␣
 ↪sum(confusion_matrix(Y_hat_logreg,y_test)[1,:]),5)*100)

# Get ROC plot and associated AUC measurement
test_set_predictions = logreg_model.predict_proba(X_test)[:,1].reshape(-1)
roc_df, roc_auc = roc_plot(y_test, test_set_predictions, plot = False)
print(f'This model gives and AUC of: {round(roc_auc,5)}')
print("precision:",precision_score(y_test, Y_hat_logreg),'\n')
```

```
Confusion Matrix :
 [[13605  3336]
 [ 1329  5608]]
Accuracy: 0.8091234320810715

% False positive for no cancellation 19.692
% False positive for cancellation 19.158
This model gives and AUC of: 0.88136
precision: 0.8084186247657489
```

For this model we calculate a ROC curve AUC of 0.88. As an AUC of 0.5 indicates that it is random whether the model classifies correctly, and 1.0 indicates perfect separability, we have a good separation here. We do however still have a significant number of false positives (shown above: 3454) which is undesirable in our hotel context as it constitutes the double booking scenario.

1.4.3 Support vector machines

SVC and NuSVC implement the "one-versus-one" approach for multi-class classification. In total, `n_classes * (n_classes - 1) / 2` classifiers are constructed and each one trains data from two classes (binary classification).

Summary We have discovered that in the current context, SVM takes a prohibitively long time to run, even for the simple parameter spaces defined here. We also note that the SVM does not scale well to large datasets. Specifically that the SVM model uses a kernel matrix which is $n \times n$, and in this case this equates to 10^{10} elements, which is a very large amount of memory and renders it intractible as a solution here.

[https://stats.stackexchange.com/questions/314329/can-support-vector-machine-be-used-in-large-data]

1.4.4 Decision tree

scikit-learn implementation does not support categorical variables for now

Standarization might be useful if you intend to compare performance with other data or other methods like SVM.

Information gain being high when the sum of the impurity of the child nodes is low. Gini Impurity is an alternative measurement, which minimises the probabilty of misclassification. Since computing square is cheaper than logarithmic function we prefer Gini impurity over entropy.

Decision trees allow us assess the importance of each feature for classifying the data,

if decision trees are not pruned, they have a high risk of overfitting to the training data. An a priori limit on nodes, or tree depth, is often set to avoid overfitting due to a deep tree.

Important features mean the features that are more closely related with dependent variable and contribute more for variation of the dependent variable. We can not directly interpret them as how much change in Y is caused due to unit change in X(j), keeping all other features constant

[39]:
```
pipe_dec_tree = make_pipeline(
            DecisionTreeClassifier(
            random_state=42,
            criterion='gini')
        )
parameters = {'decisiontreeclassifier__max_depth': list(range(1,20))}

kf = KFold(n_splits=5, shuffle=True, random_state=0)
models_tree = GridSearchCV(pipe_dec_tree, param_grid = parameters,␣
 ↪scoring="accuracy", cv=kf,return_train_score=True)
models_tree.fit(X_train, y_train)

y_hat_dectree = models_tree.predict(X_test)
```

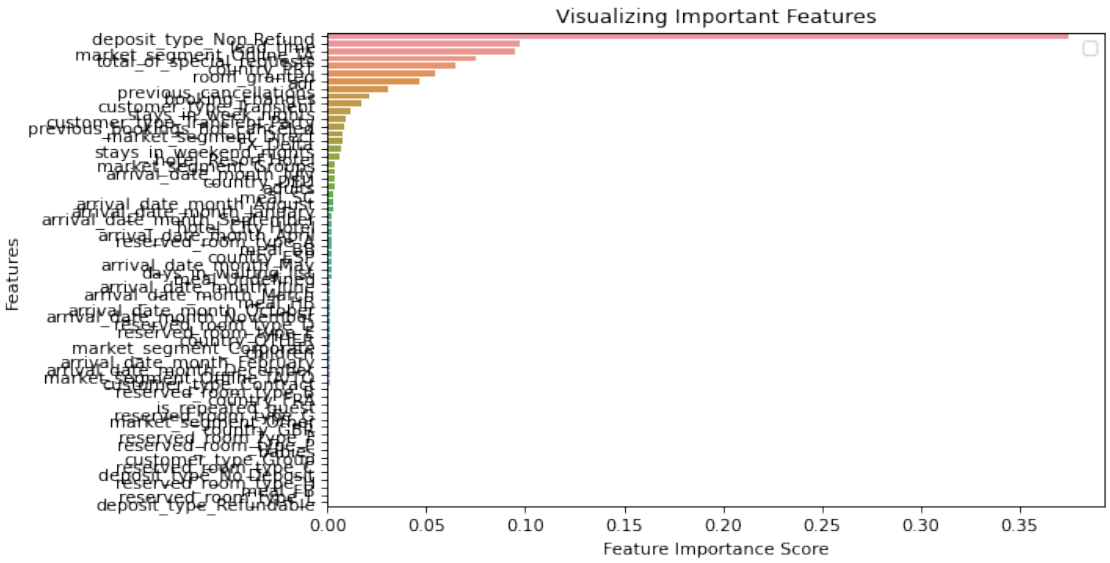
[40]:
```
probs_tree = models_tree.predict_proba(X_test)
display_results_report(models_tree, X_train, X_test, y_train, y_test,␣
 ↪y_hat_dectree)
```

```
_____ GridSearchCV Model Results_____
Best Random forest params:  {'decisiontreeclassifier__max_depth': 15}
Training set score for Random Forest: 0.872519
Testing  set score for Random Forest: 0.842994
_____ Final Random Forest Model Results_____
_____ Confusion Matrix_____ :
 [[13308  2123]
 [ 1626  6821]]
_____ Classification Report_____ :
              precision    recall  f1-score   support

           0       0.86      0.89      0.88     14934
           1       0.81      0.76      0.78      8944

    accuracy                           0.84     23878
   macro avg       0.83      0.83      0.83     23878
```

```
weighted avg       0.84      0.84      0.84     23878

% False positive for no cancellation 13.758000000000001
% False positive for cancellation 19.249
```

[41]:
```
# Extract feature importances
importance_tree = models_tree.best_estimator_._final_estimator.
 ↪feature_importances_
coefficients_tree_dict = dict(zip(X.columns, importance_tree))
# Colate graph data
feature_imp = pd.Series(importance_tree,index=X.columns).
 ↪sort_values(ascending=False)
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```



[42]:
```
pd.DataFrame(feature_imp)
```

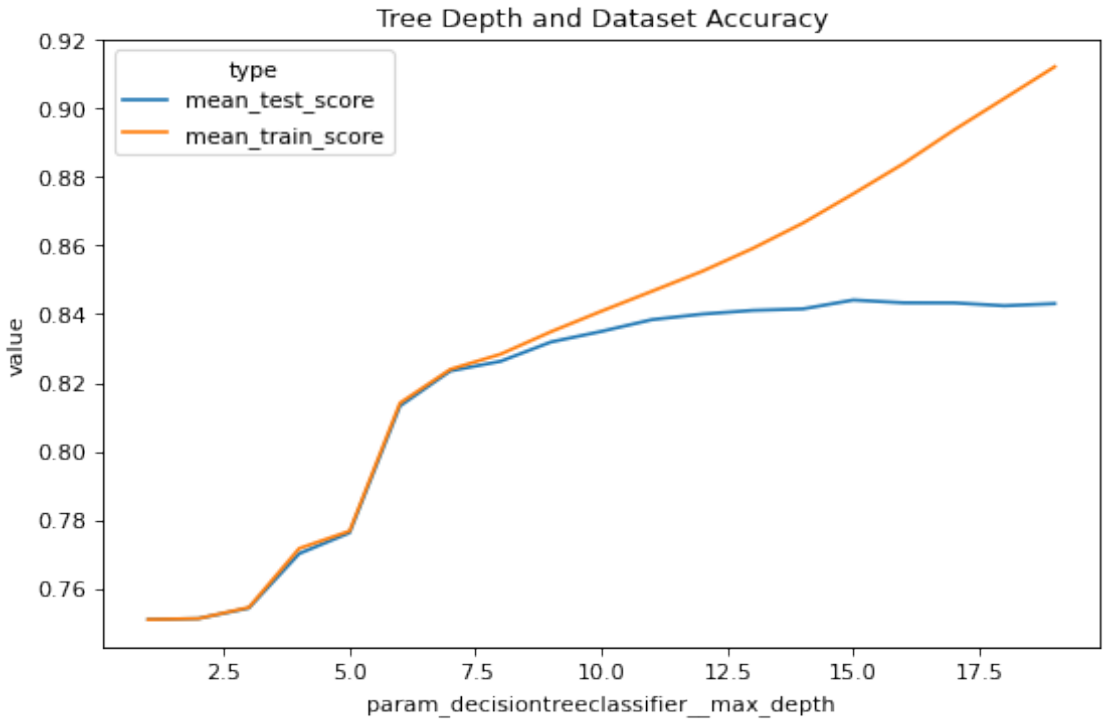
[42]:

	0
deposit_type_Non Refund	0.374577
lead_time	0.097635
market_segment_Online TA	0.094486

```
total_of_special_requests  0.075037
country_PRT                0.064830
…                             …
deposit_type_No Deposit    0.000116
reserved_room_type_H       0.000113
meal_FB                    0.000059
reserved_room_type_L       0.000000
deposit_type_Refundable    0.000000

[63 rows x 1 columns]
```

[43]:
```
cols = ["param_decisiontreeclassifier__max_depth", "mean_test_score",␣
↪"mean_train_score"]

scores_df = pd.DataFrame(models_tree.cv_results_).
↪sort_values("param_decisiontreeclassifier__max_depth")[cols]

scores_df = pd.melt(scores_df,␣
↪value_vars=['mean_test_score','mean_train_score'],
                    id_vars=['param_decisiontreeclassifier__max_depth'],␣
↪var_name = ['type'])

sns.lineplot(data = scores_df, x='param_decisiontreeclassifier__max_depth', y =␣
↪'value', hue = 'type')
plt.title('Tree Depth and Dataset Accuracy')

#pd.DataFrame(models.cv_results_).
↪sort_values("rank_test_score")[["param_decisiontreeclassifier__max_depth",␣
↪"mean_test_score", "std_test_score"]].head()
```

[43]:
```
Text(0.5, 1.0, 'Tree Depth and Dataset Accuracy')
```



From the above plot we can see that even the best depth established by the model was 17, the optimal depth is with 7, as after this the mean test score and the mean train score start to diverge from each other.

1.4.5 Random forest trees

For now we will just examine the effect of `n_estimators` which corresponds to the number of trees and `max_depth` which is the maximum depth of the trees.

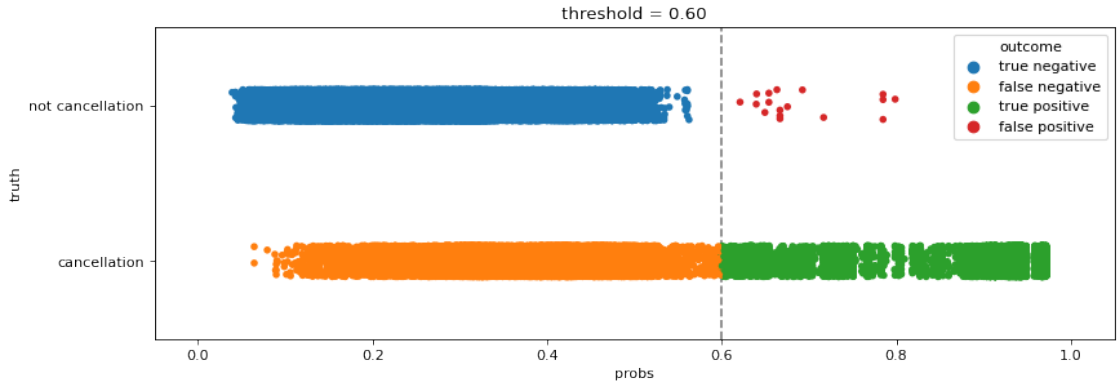
[44]:
```
pipe_random_forest = make_pipeline(
            RandomForestClassifier(
            random_state=42,
            criterion='gini'
        )) #max_depth = max_depth,

parameters = { 'randomforestclassifier__max_depth': list(range(1,7)),#[4, 7],␣
↪#1,20
               'randomforestclassifier__n_estimators': [20, 50, 100] }␣
↪#list(range(1,100))

kf = KFold(n_splits=5, shuffle=True, random_state=0)
models_tree_random = GridSearchCV(pipe_random_forest, param_grid = parameters,␣
↪scoring="accuracy", cv=kf,return_train_score=True)
models_tree_random.fit(X_train, y_train)
```

```
#n_iter = len(list(itertools.product(*list(iter(params.values())))))

y_hat_random_forest = models_tree_random.predict(X_test)
#print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

[45]:
```
display_results_report(models_tree_random, X_train, X_test, y_train, y_test,␣
↪y_hat_random_forest)
```

```
_____ GridSearchCV Model Results_____
Best Random forest params:  {'randomforestclassifier__max_depth': 6,
'randomforestclassifier__n_estimators': 50}
Training set score for Random Forest: 0.774154
Testing  set score for Random Forest: 0.770123
_____ Final Random Forest Model Results_____
_____ Confusion Matrix_____ :
 [[14858  5413]
 [   76  3531]]
_____ Classification Report_____ :
              precision    recall  f1-score   support

           0       0.73      0.99      0.84     14934
           1       0.98      0.39      0.56      8944

    accuracy                           0.77     23878
   macro avg       0.86      0.69      0.70     23878
weighted avg       0.83      0.77      0.74     23878

% False positive for no cancellation 26.703
% False positive for cancellation 2.1069999999999998
```

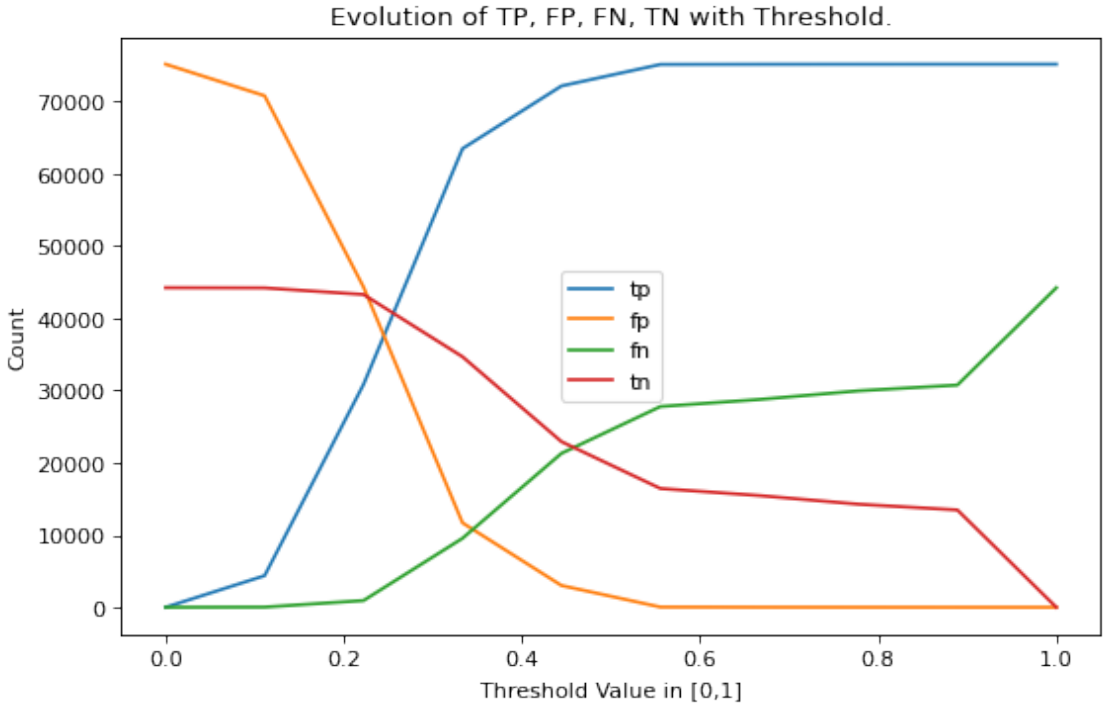
[46]:
```
# This transformation is necessary so that seaborn behaves correctly when␣
↪plotting the data horizontally
probs_random_forest = models_tree_random.predict_proba(X)[:,1]

conf_matrix = confusion_plot(truth, probs_random_forest, threshold=0.6, plot =␣
↪True)
tp, fp, fn, tn = conf_matrix[0][0], conf_matrix[0][1], conf_matrix[1][0],␣
↪conf_matrix[1][1]
print(f'tp: {tp}, fp: {fp}, fn: {fn}, tn: {tn}')
conf_matrix
```




```
tp: 75149, fp: 17, fn: 27932, tn: 16292
```

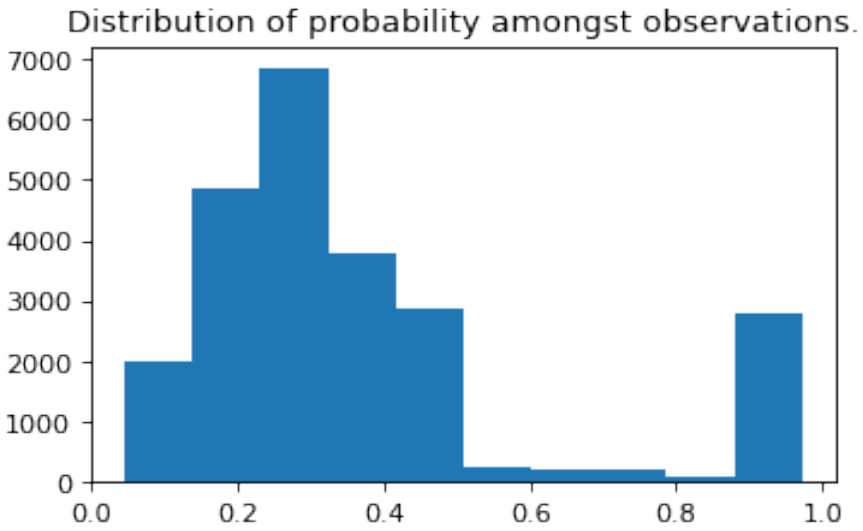
[46]:
```
array([[75149,    17],
       [27932, 16292]])
```

[47]:
```
# How do the tp, fp, tn, fn vary with the threshold we set? Does it matter?
#This shows us what thresholds we want to set to minimise fpr
true_false_evolution(probs_random_forest, thresholds_input = np.
↪linspace(0,1,10))
```



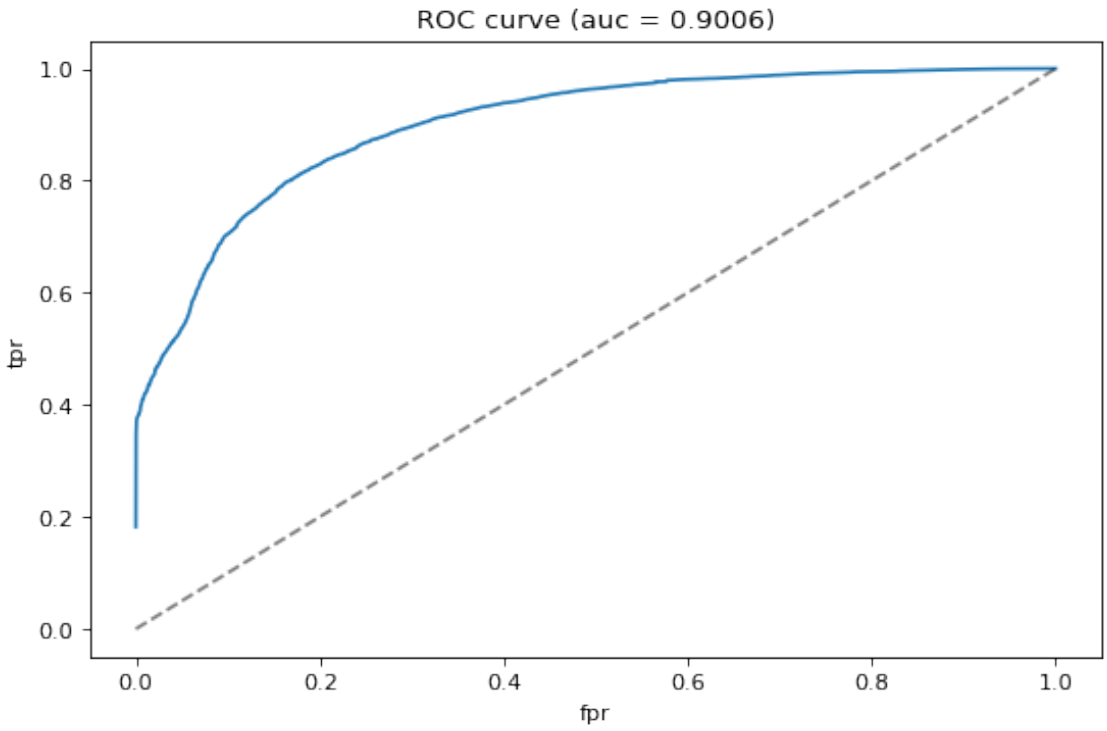
[48]:
```
# How balanced is the distribution of the data labels, as if this is bad it␣
 ↪might degrade the model:
print('Count of labels 0 and 1 in the data to indicate balance:
 ↪\n',d['is_canceled'].value_counts())
fig_hist = plt.figure(figsize=(5,3))
fig_hist = plt.hist(models_tree_random.predict_proba(X_test)[:,1].reshape(-1))
plt.title('Distribution of probability amongst observations.')
plt.show()
```

```
Count of labels 0 and 1 in the data to indicate balance:
 0    75166
1    44224
Name: is_canceled, dtype: int64
```



[109]:
```
test_set_predictions = models_tree_random.predict_proba(X_test)[:,1].reshape(-1)
print(y_hat_random_forest)
roc_df = roc_plot(y_test, test_set_predictions, plot=True)#Y_hat_logreg)
type(roc_df[0])
```

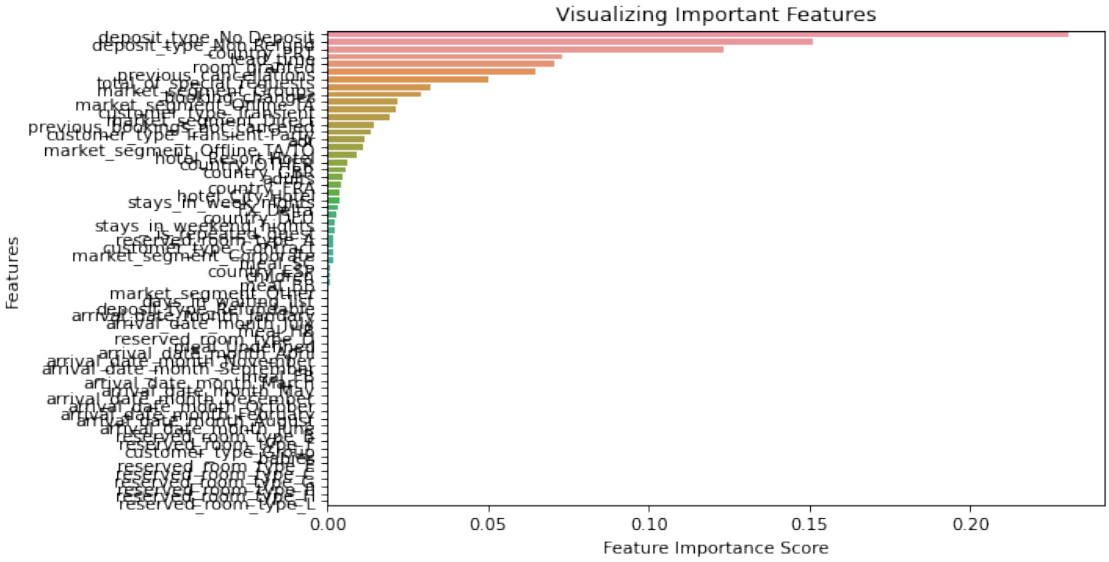
```
[0 0 1 … 0 0 0]
```



[109]:
```
pandas.core.frame.DataFrame
```

[50]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_hat_random_forest))
importance_tree_random = models_tree_random.best_estimator_._final_estimator.
 ↪feature_importances_

feature_imp_tree_random = pd.Series(importance_tree_random,index=X.columns).
 ↪sort_values(ascending=False)
sns.barplot(x=feature_imp_tree_random, y=feature_imp_tree_random.index)

# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.show()

# pd.DataFrame(models_tree_random.cv_results_).
 ↪sort_values('param_randomforestclassifier__max_depth')[cols]
```

```
Accuracy: 0.7701231258899406
```

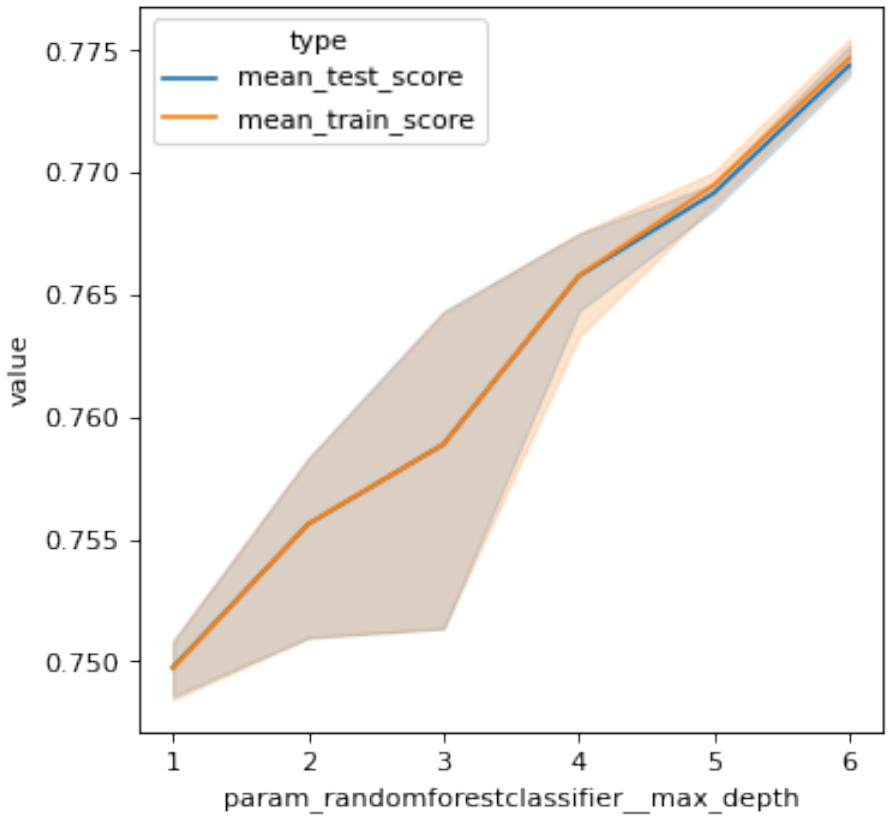


[51]:
```
print('% False positive for no␣
 ↪cancellation',round(confusion_matrix(y_hat_random_forest,y_test)[0,1]/␣
 ↪sum(confusion_matrix(Y_hat_logreg,y_test)[0,:]),5)*100)
print('% False positive for␣
 ↪cancellation',round(confusion_matrix(y_hat_random_forest,y_test)[1,0]/␣
 ↪sum(confusion_matrix(Y_hat_logreg,y_test)[1,:]),5)*100)
```

```
% False positive for no cancellation 31.952
% False positive for cancellation 1.0959999999999999
```

[52]:
```
cols = ['param_randomforestclassifier__max_depth', "mean_test_score",␣
 ↪"mean_train_score"]
scores_df = pd.DataFrame(models_tree_random.cv_results_).
 ↪sort_values('param_randomforestclassifier__max_depth')[cols]
scores_df = pd.melt(scores_df,␣
 ↪value_vars=['mean_test_score','mean_train_score'],
                     id_vars=['param_randomforestclassifier__max_depth'],␣
 ↪var_name = ['type'])

fig = plt.figure(figsize = (5,5))
sns.lineplot(data = scores_df, x='param_randomforestclassifier__max_depth', y =␣
 ↪'value', hue = 'type')
```

[52]:
```
<AxesSubplot:xlabel='param_randomforestclassifier__max_depth', ylabel='value'>
```



1.5 4. Discussion & Conclusions

1.5.1 Model Performance

1.5.2 Explanation of Results

[53]:
```
RandTree_Imp =  pd.DataFrame(feature_imp_tree_random).reset_index()
RandTree_Imp.columns = (['Feature', 'coefficients'])
RandTree_Imp
```

[54]:
```
logreg_Coefs
```

[55]:
```
#Merg_coef = pd.merge(SVM_Coefs.drop(['Coefficients Correlation'], axis = 1),␣
↪logReg_Coefs.drop(['Coefficients Correlation'], axis = 1), on = 'Feature')
#Merg_coef.columns = ['SVM_Coefs', 'Feature','LogReg_Coefs' ]
```

[56]:
```
def categorical_data_viewer(DataFrame, StartsWith = '', Tolerance = 0 ,␣
↪DeleteZeros = 'Y'):
    DataFrame = DataFrame[DataFrame['Feature'].str.match(StartsWith)].copy()
    DataFrame.loc[:,'Feature'] = DataFrame.loc[:,'Feature'].str.
↪replace((StartsWith+'_'),'')
```

```
    if DeleteZeros == 'Y':
        DataFrame.drop(DataFrame[(DataFrame['coefficients'].eq(0)) |␣
↪(abs(DataFrame['coefficients']) < Tolerance )].index, inplace = True)
    else:
        DataFrame.drop(DataFrame[(abs(DataFrame['coefficients']) < Tolerance )].
↪index, inplace = True)

    return DataFrame
```

Country Discussion The only conclusion from the analysis is that domestic travel (guests from Portugal) have a higher likleihood of cancellation.

The decision to group countires into "Other" for countires which consitute for less that 5% of all bookings has allowed the results to show that these countries are unlikely to cancel compared to any of the countires with many visitors. Countries like Fuji or Uzbeckistan only had 1 visitor to either of the hotels,

[58]:
```
country_coefs
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-58-368b02129f9a> in <module>
----> 1 country_coefs

NameError: name 'country_coefs' is not defined
```

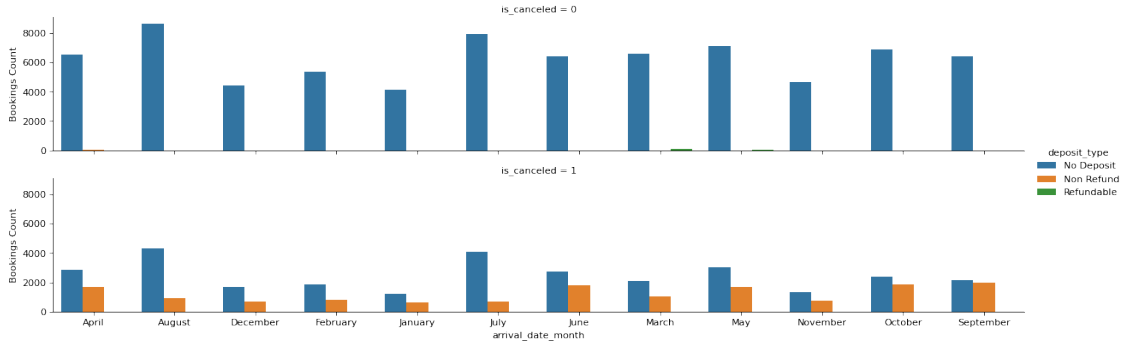
[ ]:
```
RandTree_Imp =  pd.DataFrame(feature_imp_tree_random).reset_index()
```

[ ]:
```
country_coefs = categorical_data_viewer(RandTree_Imp, StartsWith = 'country',␣
↪Tolerance = 0.0 , DeleteZeros = 'N' )
country_coefs = pd.merge(country_coefs, d_FULL[['country', 'distance(km)',␣
↪'Region Name', 'ISO4217-currency_country_name']].drop_duplicates(), left_on␣
↪= 'Feature', right_on='country', how = 'left')
fig = px.scatter(country_coefs, x="distance(km)", y="coefficients",␣
↪color="country", hover_data=['ISO4217-currency_country_name','Region Name'␣
↪], marginal_y = "box")
fig.show()
```

[ ]:
```
#categorical_data_viewer(logReg_Coefs, StartsWith = 'market_segment', Tolerance␣
↪= 0.05 , DeleteZeros = 'N' )
pd.concat((categorical_data_viewer(RandTree_Imp, StartsWith = 'deposit_type',␣
↪Tolerance = 0.0 , DeleteZeros = 'N' ),
categorical_data_viewer(RandTree_Imp, StartsWith = 'adr', Tolerance = 0.0 ,␣
↪DeleteZeros = 'N' )),axis=0)
```

[ ]:
```
        Feature  coefficients
0    No Deposit      0.297654
1    Non Refund      0.174646
32   Refundable      0.000867
14          adr      0.010887
```

[68]:
```
deposits =  d_FULL.groupby(['deposit_type', 'arrival_date_month',␣
 ↪'is_canceled']).agg({'adr':'count'}).reset_index()
deposits.rename(columns = {'adr':'Bookings Count'}, inplace = True)

p = sns.catplot(data = deposits, x = 'arrival_date_month' , y = 'Bookings␣
 ↪Count', kind = 'bar' , hue = 'deposit_type' , row = 'is_canceled' , height =␣
 ↪2.5, aspect = 6)
```



[ ]:
```
d.FULL.groupby(['deposit_type', 'arrival_date_month', 'is_canceled'].
 ↪agg{'hotel'})
```

In this section you should provide a general overview of your final model, its ***performance (economic viability, false positives)****, and* ***reliability (ROC,AUC, data balance)****. You should discuss what the implications of your model are in terms of the included features, predictive performance, and anything else you think is relevant.*

This should be written with a target audience of the client who is with the hotel data and university level mathematics but not necessarily someone who has taken a postgraduate statistical modeling course. Your goal should be to convince this audience that your model is both accurate and useful.

Keep in mind that a negative result, i.e. a model that does not work well predictively, that is well explained and justified in terms of why it failed will likely receive higher marks than a model with strong predictive performance but with poor or incorrect explinations / justifications.

1.6 5. Convert Document

[69]:
```
# Run the following to render to PDF
!jupyter nbconvert --to pdf proj2.ipynb
```

```
[NbConvertApp] Converting notebook proj2.ipynb to pdf
```

```
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/nbconvert/filters/datatypefilter.py:41: UserWarning: Your element with
mimetype(s) dict_keys(['text/html']) is not able to be represented.
  mimetypes=output.keys())
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/nbconvert/filters/datatypefilter.py:41: UserWarning: Your element with
mimetype(s) dict_keys(['text/html']) is not able to be represented.
  mimetypes=output.keys())
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/nbconvert/filters/datatypefilter.py:41: UserWarning: Your element with
mimetype(s) dict_keys(['text/html']) is not able to be represented.
  mimetypes=output.keys())
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/nbconvert/filters/datatypefilter.py:41: UserWarning: Your element with
mimetype(s) dict_keys(['text/html']) is not able to be represented.
  mimetypes=output.keys())
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/nbconvert/filters/datatypefilter.py:41: UserWarning: Your element with
mimetype(s) dict_keys(['text/html']) is not able to be represented.
  mimetypes=output.keys())
[NbConvertApp] Support files will be in proj2_files/
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Making directory ./proj2_files
[NbConvertApp] Writing 184343 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 544888 bytes to proj2.pdf
```

Created in Deepnote