|  |  |
| --- | --- |
|  | **DOKUZ EYLÜL UNIVERSITY**  **ENGINEERING FACULTY**  **DEPT. OF COMPUTER ENGINEERING** |

# CME 4403 Machine Learning

# Term Project Report

**2019-2020 SPRING**

**Booking Cancellation Prediction Using Hotel Bookings Data**

**Prepared by: Mustafa ÖZSARAÇ - 2016510086**

**Proposed to: Dr. Özge KART**

# 1-Introduction

In this progress report, I will explain my process for the machine learning term project. This report will contain explanation of what I have done, description of my data set, what techniques I applied in order to prepare the dataset and which machine learning algorithms that I used in order to predict my target feature. Also it will contain test results for different machine learning models to select which one is the best for this dataset. My aim for this term project will be to create a system which can accurately predict if guests are going to cancel their reservation or not.

# 2-Information About Dataset

Reservation cancellation is not uncommon in the hotel industry. Each cancellation means a lost revenue opportunity that can never be recovered. When working at the front desk at an airport hotel, we had to call each guest to confirm if they will show up in the afternoon. If they confirm that they cannot show up, we could try to sell the room again. This practice ensures the revenue to a certain degree. However, it is not sufficient to call each guest in the afternoon since a majority of guests check-in during the afternoon.

Therefore, if we can predict if a guest would cancel a reservation, hotels could contact guests that most likely to cancel to confirm more efficiently and to resell the room to optimize revenues.

This data set contains booking information for a city hotel and a resort hotel, and includes information such as when the booking was made, length of stay, the number of adults, children, and/or babies, and the number of available parking spaces, among other things.

**COLUMN DESCRIPTIONS**:

* **Hotel:** (H1 = Resort Hotel or H2 = City Hotel)
* **lead\_time: #** of days that elapsed between the entering date of the booking into the PMS and the arrival date
* **arrival\_date\_year**: Year of arrival date
* **arrival\_date\_month:** Month of arrival date
* **arrival\_date\_week\_number:** Week number of year for arrival date
* **arrival\_date\_day\_of\_month**: Day of arrival date
* **stays\_in\_weekend\_nights:** Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
* **stays\_in\_week\_nights:** Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel
* **adults:** Number of adults
* **children:** Number of children
* **babies:** Number of babies
* **mealType**: of meal booked. Categories are presented in standard hospitality meal packages: Undefined/SC – no meal package;
  + BB – Bed & Breakfast;
  + HB – Half board (breakfast and one other meal – usually dinner);
  + FB – Full board (breakfast, lunch and dinner)
* **country:** Country of origin. Categories are represented in the ISO 3155–3:2013 format
* **market\_segment:** Market segment designation. In categories, the term “TA” means “Travel Agents” and “TO” means “Tour Operators”
* **distribution\_channel:** Booking distribution channel. The term “TA” means “Travel Agents” and “TO” means “Tour Operators”
* **is\_repeated\_guest:** Value indicating if the booking name was from a repeated guest (1) or not (0)
* **previous\_cancellations:** Number of previous bookings that were cancelled by the customer prior to the current booking
* **previous\_bookings\_not\_canceled:** Number of previous bookings not cancelled by the customer prior to the current booking
* **reserved\_room\_type:** Code of room type reserved. Code is presented instead of designation for anonymity reasons.
* **assigned\_room\_type:** Code for the type of room assigned to the booking.
* **booking\_changes:** Number of changes/amendments made to the booking from the moment the booking was entered on the PMS until the moment of check-in or cancellation
* **deposit\_type:** Indication on if the customer made a deposit to guarantee the booking. This variable can assume three categories:
  + No Deposit – no deposit was made;
  + Non Refund – a deposit was made in the value of the total stay cost;
  + Refundable – a deposit was made with a value under the total cost of stay.
* **agent:** ID of the travel agency that made the booking
* **company:** ID of the company/entity that made the booking or responsible for paying the booking. ID is presented instead of designation for anonymity reasons
* **days\_in\_waiting\_list:** Number of days the booking was in the waiting list before it was confirmed to the customer
* **customer\_type:** Contract - when the booking has an allotment or other type of contract associated to it;
  + Group – when the booking is associated to a group;
  + Transient – when the booking is not part of a group or contract, and is not associated to other transient booking;
  + Transient-party – when the booking is transient, but is associated to at least other transient booking
* **adr:** Average Daily Rate as defined by dividing the sum of all lodging transactions by the total number of staying nights
* **required\_car\_parking\_spaces**: Number of car parking spaces required by the customer
* **total\_of\_special\_requests**: Number of special requests made by the customer (e.g. twin bed or high floor)
* **reservation\_status:** Reservation last status, assuming one of three categories:
  + Canceled – booking was canceled by the customer;
  + Check-Out – customer has checked in but already departed;
  + No-Show – customer did not check-in and did inform the hotel of the reason why
* **reservation\_status\_date:** Date at which the last status was set.

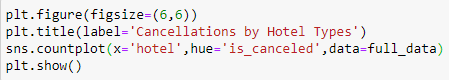
**TARGET VALUE:**

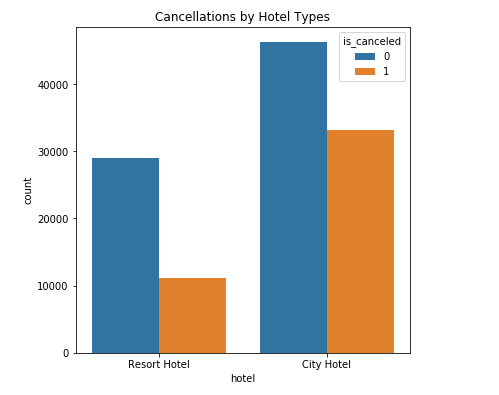
* **is**\_**canceled:** Value indicating if the booking was canceled (1) or not (0)

Now, let’s do some **EDA** **(exploratory data analysis)** to know the data set a little bit better.

**Q1: Which hotel type has more cancelations?**

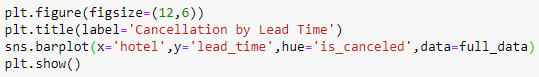
There are two types of hotels in this data set. These are “Resort Hotel” and “City Hotel”.

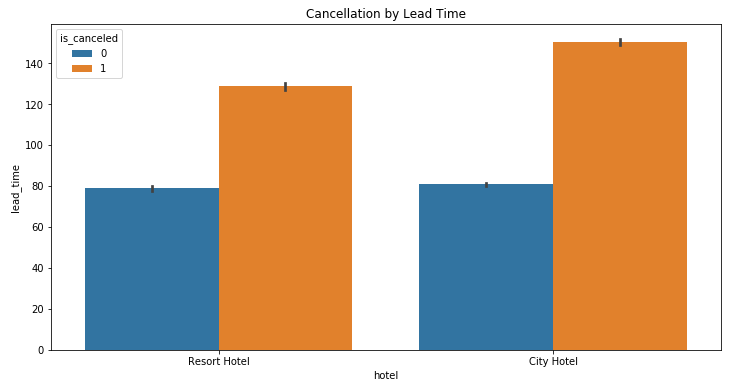




As you can see here, city hotels have much more cancellations.

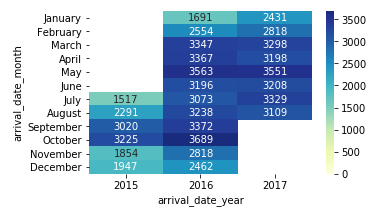
**Q2: Does lead time effect the cancellations?**





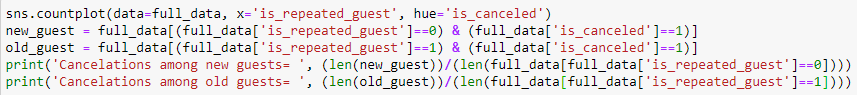
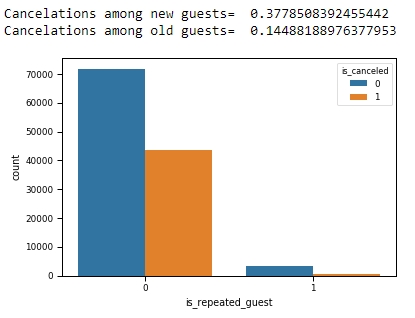
As you can see here, it looks like the longer the lead time, the reservation is more likely to be canceled.

**Q3: In which period of the year the number of bookings peak?**



So, this matrix shows that the number of fulfilled reservations peak each year at the months of May and October.

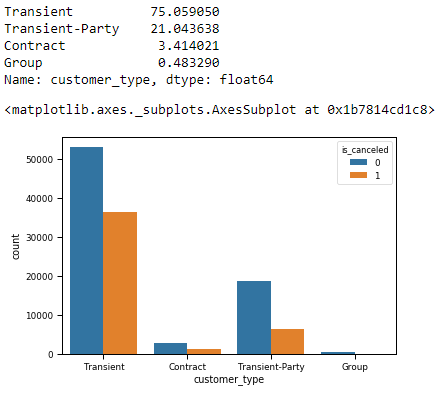
**Q4: Does being an old guest has an effect on cancellations?**



As seen in the table, the above graph bolsters the evidence that maximum customers are new comers and they are more likely to cancel their current booking. Old guests are less likely to cancel the booking (14%).

**Q5: Effect of cancellations by customer types:**

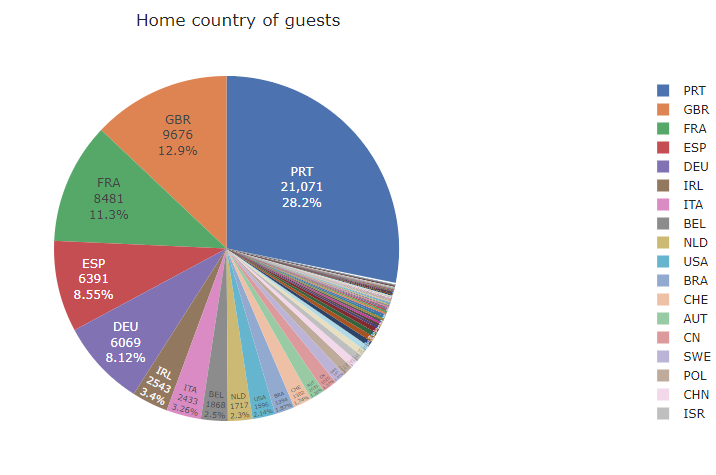




75% bookings occur in Transient category of customers. It also sees the highest cancellation among all the categories.

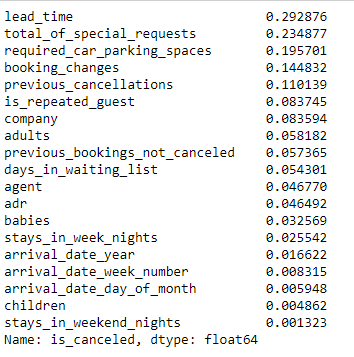
**Q6: Where do the guests come from?**





People from all over the world are staying in these two hotels. Most guests are from Portugal and other countries in Europe. It’s mainly because these both hotels are located in Portugal ("Hotel1 at the resort region of Algarve and Hotel2 at the city of Lisbon").

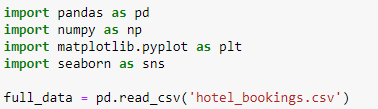
**Q7: Which numerical features are most important?**



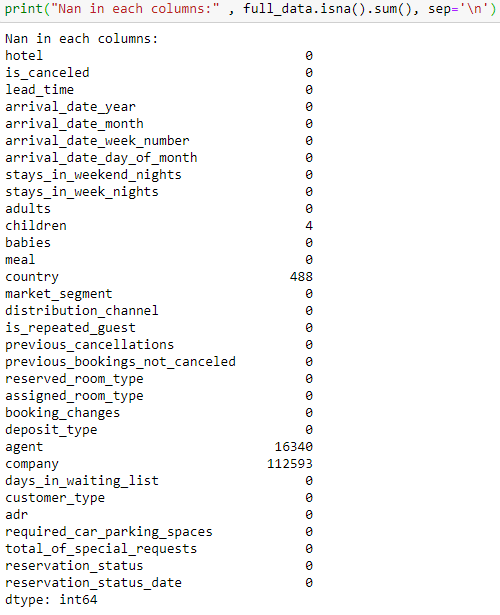
From this list, we can say that lead\_time, total\_of\_special\_requests, required\_car\_parking\_spaces, booking\_changes, previous cancellations are the five most important features.

# 3-Data Preparation Techniques

I used **Python** with **Jupyter Notebook** and “**pandas**” library to load this data as data frame. To read the data, I used the code shown below:



First, I print the list of NA values in data frame:



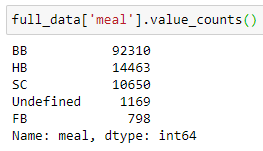
To replace these missing values:

* **Agent**: If no agent is given, then probably there is not one. So I filled them with 0’s.
* **Company**: If no company is given, it’s probably private.
* **Country**: If non given, it should be tagged as “Unknown”.
* **Children**: If non given, it’s probably zero.

I used the code shown below:



Also, there are some Undefined values in feature called “**meal**”.

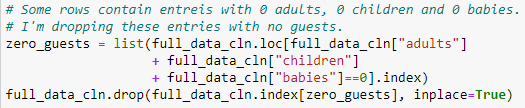


So I replaced “**Undefined**” with “**SC**” which is equal.



There some zeros in **Adult, children and babies** categories, which makes no sense. So I need to drop those rows.

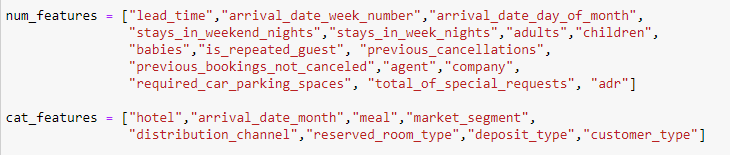




Now, data is clean and I’m storing that on variable called “full\_data\_cln”. We can move on.

# 4-Organizing the Data, Model Implementation & Results

First of all, I need to categorize my features as numerical and categorical. Then I’ll preprocess those data, and label them if they’re categorical with simple imputer and one hot encoding.



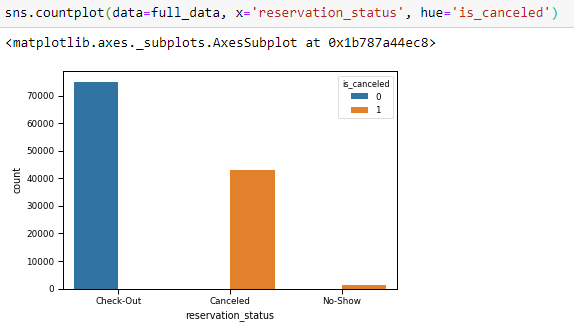
As you can see above, I categorized my features as numerical and categorical.

I choose them manually, because I exclude some columns to make the model more general and also prevent the data leakage.

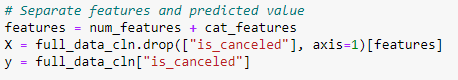
I excluded: **arrival\_date\_year, assigned\_room\_type, booking\_changes, reservation\_status, country, days\_in\_waiting\_list,** **reservation\_status\_date.**

I could have included “**country**”, but I’m trying to make a general model and it would make the model more local.

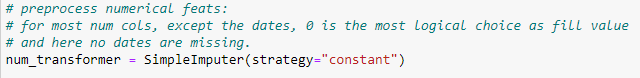
As you can see the graph below, the **reservation\_status, booking changes** list are cheating. Because they are complete match with is\_canceled column so this is why I excluded them.



Now, I can separate target and train features.

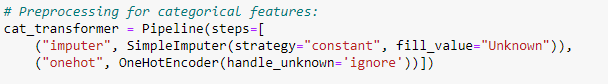


Preprocessing of numerical data:



It’s just filling with 0.

Preprocessing of categorical data:

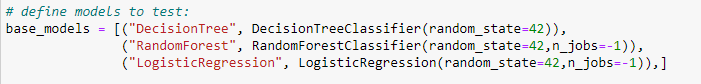


Filling with parameter called “**Unknown**” and then using “**One Hot Encoding**” to make more sense for computer in categorical data.

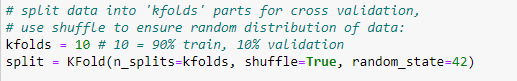
Then, I’m creating a Pipeline for these two preprocessing steps (which I’ll use later):



Now, I can define my machine learning models. I have chosen Decision Tree, Linear Regression and Random Forest. You can see the code shown below:

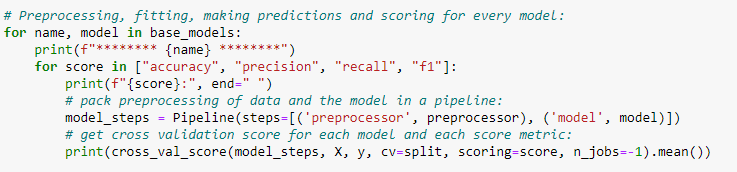


After that, I will create a cross validation to test my algorithms. As you can see:



I choose 10 for folding, shuffle = True to ensure the random distribution of data. Random\_state = 42 is for ensuring same results will occur for the same splitting.

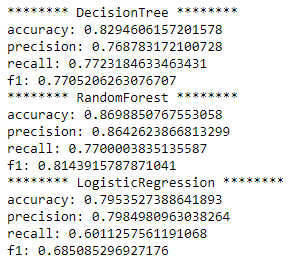
After that, I can use cross fold validation to test algorithms, and print the results.



This codes here do these as listed below:

* Firstly, gets the model from base\_models array and prints the name of model that will be used.
* Then, it chooses scoring metric from the array of accuracy, recall, precision and f1-score.
* After that, with the help of Pipeline it will preprocess the data as prepared, and then it will fit the data into the model.
* Then, it will get the score of wanted metric and wanted model in order. Finally, it will print these results to the console.

The output of the results is shown below:



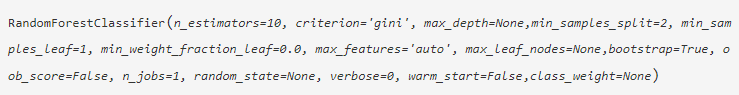
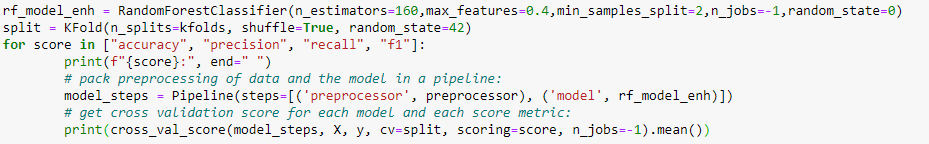
As you can see from the scores, **Random Forest** performs best for this data set.

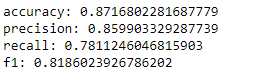
Now I can make parameter tuning for Random Forest to get more accuracy from it.

# 5-Parameter Tuning and Feature Importance

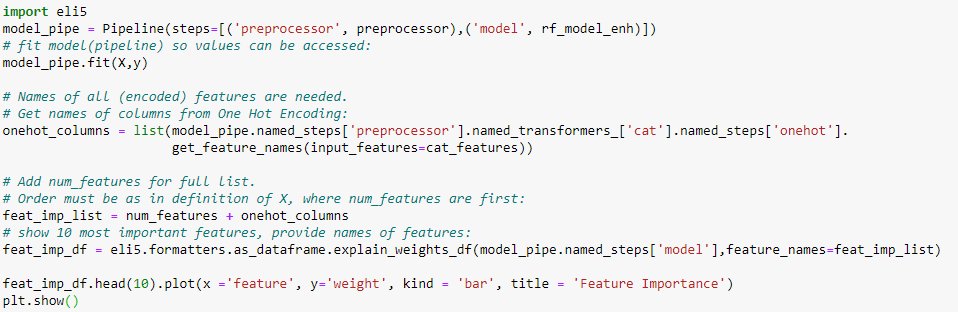
The objective of parameter tuning is to find the optimum value for each parameter to improve the accuracy of the model. To tune these parameters, you must have a good understanding of these meaning and their individual impact on model. You can repeat this process with a number of well performing models.

For example: In random forest, we have various parameters like max\_features, number\_trees, random\_state, oob\_score and others. Intuitive optimization of these parameter values will result in better and more accurate models.

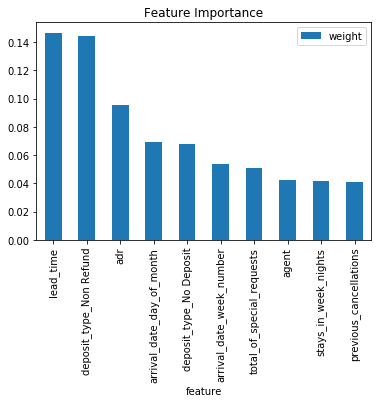
Below is random forest scikit learn algorithm with list of all parameters:Enhanced RF model with the best parameters I found:And the results are as follows:



As you can see here, even with the improved parameters, the accuracy, precision, recall and f1-score increase is minimal.

**Feature Importance:**

With the help of this code, I created a list called feat\_imp\_list which stores feature importance values. Then, I print top ten important features on a plot as shown below:

****

**As you can see from the graph, most important feature is lead\_time.**

# 6-Conclusion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Recall | Precision | F1-score |
| Decision Tree | 0.8294 | 0.7723 | 0.7687 | 0.7705 |
| Random Forest | 0.8698 | 0.7700 | 0.8642 | 0.8143 |
| Logistic Regression | 0.7953 | 0.6011 | 0.7984 | 0.6850 |
| Enhanced Random Forest | 0.8716 | 0.7811 | 0.8599 | 0.8186 |

As in conclusion table, “Enhanced Random Forest” model provided the best accuracy, recall and F1-score; second highest precision. Thus, it is best model for this data set. Other than Random Forest algorithms, my second choice would be Decision Tree with second at accuracy, recall, and F1-score.

# 7-Source Code

#!/usr/bin/env python

# coding: utf-8

#importing necessary libraries:

# common:

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

#for machine learning:

from sklearn.model\_selection import train\_test\_split, KFold, cross\_validate, cross\_val\_score

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

import eli5

#read csv file

full\_data = pd.read\_csv('hotel\_bookings.csv')

full\_data.head()

#print missing values

print("Nan in each columns:" , full\_data.isna().sum(), sep='\n')

#------------- EDA starts from here---------

#plot to show hotel types and cancellations.

plt.figure(figsize=(6,6))

plt.title(label='Cancellations by Hotel Types')

sns.countplot(x='hotel',hue='is\_canceled',data=full\_data)

plt.show()

#plot to show cancellations by lead time

plt.figure(figsize=(12,6))

plt.title(label='Cancellation by Lead Time')

sns.barplot(x='hotel',y='lead\_time',hue='is\_canceled',data=full\_data)

plt.show()

#plot to show peaking month by booking

bookings = full\_data[full\_data['is\_canceled']==0].pivot\_table(index='arrival\_date\_month', columns='arrival\_date\_year', values='hotel', aggfunc=len, fill\_value=0)

bookings.index = pd.CategoricalIndex(bookings.index, categories=['January', 'February', 'March', 'April','May','June','July', 'August','September', 'October', 'November', 'December'], ordered=True)

bookings = bookings.sort\_index()

mask = np.array([[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1]])

f, ax = plt.subplots(figsize=(5, 3))

sns.heatmap(bookings, center=2000, annot=True, mask=mask, fmt="d", ax=ax, cmap="YlGnBu")

sns.set\_context('paper')

plt.show()

#plot to show effect of being old customer on cancelling.

sns.countplot(data=full\_data, x='is\_repeated\_guest', hue='is\_canceled')

new\_guest = full\_data[(full\_data['is\_repeated\_guest']==0) & (full\_data['is\_canceled']==1)]

old\_guest = full\_data[(full\_data['is\_repeated\_guest']==1) & (full\_data['is\_canceled']==1)]

print('Cancelations among new guests= ', (len(new\_guest))/(len(full\_data[full\_data['is\_repeated\_guest']==0])))

print('Cancelations among old guests= ', (len(old\_guest))/(len(full\_data[full\_data['is\_repeated\_guest']==1])))

#customer types and plot to show them.

print(full\_data['customer\_type'].value\_counts(normalize=True)\*100)

sns.countplot(data=full\_data, x='customer\_type', hue='is\_canceled')

# plot to show pie chart of countries.

import plotly.express as px

# get number of acutal guests by country

country\_data = pd.DataFrame(full\_data.loc[full\_data["is\_canceled"] == 0]["country"].value\_counts())

#country\_data.index.name = "country"

country\_data.rename(columns={"country": "Number of Guests"}, inplace=True)

total\_guests = country\_data["Number of Guests"].sum()

country\_data["Guests in %"] = round(country\_data["Number of Guests"] / total\_guests \* 100, 2)

country\_data["country"] = country\_data.index

#country\_data.loc[country\_data["Guests in %"] < 2, "country"] = "Other"

# pie plot

fig = px.pie(country\_data,

values="Number of Guests",

names="country",

title="Home country of guests",

template="seaborn")

fig.update\_traces(textposition="inside", textinfo="value+percent+label")

fig.show()

#showing meal has some undefined values.

full\_data['meal'].value\_counts()

#showing there are some rows that contains zero adults zero children and babies

len(full\_data[(full\_data['adults']==0) & (full\_data['children']==0) & (full\_data['babies']==0)])

#showing a plot that, reservation status directly effects is\_cancelled.

sns.countplot(data=full\_data, x='reservation\_status', hue='is\_canceled')

#-------------EDA ends--------------

#--------DATA CLEANING PROCESS--------------

# Replacing missing values:

# agent: If no agency is given, booking was most likely made without one.

# company: If none given, it was most likely private.

# country: If non given, it should be tagged as “Unknown”.

# children: If non given, it’s probably zero.

# meal: contains values "Undefined", which is equal to SC.

nan\_replacements = {"children:": 0.0,"country": "Unknown", "agent": 0, "company": 0}

full\_data\_cln = full\_data.fillna(nan\_replacements)

full\_data\_cln["meal"].replace("Undefined", "SC", inplace=True)

# Some rows contain entreis with 0 adults, 0 children and 0 babies.

# It does not make sense, so I'm dropping these entries with no guests.

zero\_guests = list(full\_data\_cln.loc[full\_data\_cln["adults"]

+ full\_data\_cln["children"]

+ full\_data\_cln["babies"]==0].index)

full\_data\_cln.drop(full\_data\_cln.index[zero\_guests], inplace=True)

#---------DATA CLEANING ENDS HERE-------------

#print correlation's with is\_canceled column

cancel\_corr = full\_data.corr()["is\_canceled"]

cancel\_corr.abs().sort\_values(ascending=False)[1:]

#---------------DATA PREPROCESSING STEPS---------------

# some columns are excluded to make the model more general and to prevent leakage

# I excluded: (arrival\_date\_year, assigned\_room\_type, booking\_changes,

# reservation\_status, country, days\_in\_waiting\_list, reservation\_status\_date.)

# including the country would increase accuracy, but it may also make the model less general.

num\_features = ["lead\_time","arrival\_date\_week\_number","arrival\_date\_day\_of\_month",

"stays\_in\_weekend\_nights","stays\_in\_week\_nights","adults","children",

"babies","is\_repeated\_guest", "previous\_cancellations",

"previous\_bookings\_not\_canceled","agent","company",

"required\_car\_parking\_spaces", "total\_of\_special\_requests", "adr"]

cat\_features = ["hotel","arrival\_date\_month","meal","market\_segment",

"distribution\_channel","reserved\_room\_type","deposit\_type","customer\_type"]

# Separate features and predicted value

features = num\_features + cat\_features

X = full\_data\_cln.drop(["is\_canceled"], axis=1)[features]

y = full\_data\_cln["is\_canceled"]

# preprocess numerical feats:

# for most num cols, except the dates, 0 is the most logical choice as fill value

# and here no dates are missing.

num\_transformer = SimpleImputer(strategy="constant")

# Preprocessing for categorical features:

cat\_transformer = Pipeline(steps=[

("imputer", SimpleImputer(strategy="constant", fill\_value="Unknown")),

("onehot", OneHotEncoder(handle\_unknown='ignore'))])

# Bundle preprocessing for numerical and categorical features:

preprocessor = ColumnTransformer(transformers=[("num", num\_transformer, num\_features),

("cat", cat\_transformer, cat\_features)])

#---------------DATA PREPROCESSING STEPS ENDS HERE---------------

#---------------DEFINING MODELS AND IMPLEMENTING-----------------

# define models to test:

base\_models = [("DecisionTree", DecisionTreeClassifier(random\_state=42)),

("RandomForest", RandomForestClassifier(random\_state=42,n\_jobs=-1)),

("LogisticRegression", LogisticRegression(random\_state=42,n\_jobs=-1)),]

# split data into 'kfolds' parts for cross validation,

# use shuffle to ensure random distribution of data:

kfolds = 10 # 10 = 90% train, 10% validation

split = KFold(n\_splits=kfolds, shuffle=True, random\_state=42)

# Preprocessing, fitting, making predictions and scoring for every model:

for name, model in base\_models:

print(f"\*\*\*\*\*\*\*\* {name} \*\*\*\*\*\*\*\*")

for score in ["accuracy", "precision", "recall", "f1"]:

print(f"{score}:", end=" ")

# pack preprocessing of data and the model in a pipeline:

model\_steps = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])

# get cross validation score for each model and each score metric:

print(cross\_val\_score(model\_steps, X, y, cv=split, scoring=score, n\_jobs=-1).mean())

#--------------------ENHANCED RF-------------------

#enhanced random forest model with new parameters.

rf\_model\_enh = RandomForestClassifier(n\_estimators=160,max\_features=0.4,min\_samples\_split=2,n\_jobs=-1,random\_state=0)

split = KFold(n\_splits=kfolds, shuffle=True, random\_state=42)

for score in ["accuracy", "precision", "recall", "f1"]:

print(f"{score}:", end=" ")

# pack preprocessing of data and the model in a pipeline:

model\_steps = Pipeline(steps=[('preprocessor', preprocessor), ('model', rf\_model\_enh)])

# get cross validation score for each model and each score metric:

print(cross\_val\_score(model\_steps, X, y, cv=split, scoring=score, n\_jobs=-1).mean())

#-------------------Feature Importance----------------

model\_pipe = Pipeline(steps=[('preprocessor', preprocessor),('model', rf\_model\_enh)])

# fit model(pipeline) so values can be accessed:

model\_pipe.fit(X,y)

# Names of all (encoded) features are needed.

# Get names of columns from One Hot Encoding:

onehot\_columns = list(model\_pipe.named\_steps['preprocessor'].named\_transformers\_['cat'].named\_steps['onehot'].

get\_feature\_names(input\_features=cat\_features))

# Add num\_features for full list.

# Order must be as in definition of X, where num\_features are first:

feat\_imp\_list = num\_features + onehot\_columns

# show 10 most important features, provide names of features:

feat\_imp\_df = eli5.formatters.as\_dataframe.explain\_weights\_df(model\_pipe.named\_steps['model'],feature\_names=feat\_imp\_list)

#print the most important features plot.

feat\_imp\_df.head(10).plot(x ='feature', y='weight', kind = 'bar', title = 'Feature Importance')

plt.show()

#------END------