

Week 11 - Final Assignment

Author: Jessica Cervi

Expected time = 3.5 hours

Assignment overview

Welcome to the final assignment of this course! At this point, you have learnt about different machine learning algorithms and you have applied each of them to different problems. In this assignment, we will instead use one single data set and try make a prediction by using different algorithms with the goal of deciding which one performs the best.

After reading the data set, we will perform some exploratory data analysis to understand the relationship between some of the features in the data set. Next, we will extensively manipulate the dataframe to prepare our data for our analysis. We will do so by using techniques such as label encoding, one hot encoding and correlation. Finally, we will apply three different machine learning algorithms to solve our problem.

This assignment is designed to help you apply the machine learning algorithms you have learned using packages in Python. Python concepts, instructions, and a starter code are embedded within this Jupyter Notebook to help guide you as you progress through the assignment. Remember to run the code of each code cell prior to submitting the assignment. Upon completing the assignment, we encourage you to compare your work against the solution file to perform a self-assessment.

Learning objectives

- Define the hypothesis to be tested and prediction to be made
- Present the approaches used in the analysis
- Discuss the results obtained using different algorithms
- Discuss the strengths and limitations of each analysis

Index:

Week 11: final assignment

- [Part 1](#) - Importing the data set
- [Part 2](#) - Exploratory data analysis (EDA)
- [Part 3](#) - Preparing our data
- [Part 4](#) - Correlation
- [Part 5](#) - Splitting our data for modelling
- [Part 6](#) - Decision tree model (reservation_status included)
- [Part 7](#) - Decision tree model (reservation_status excluded)
- [Part 8](#) - K-nearest neighbours
- [Part 9](#) - Naïve Bayes
- [Part 10](#) - Conclusion

Final assignment

In the past weeks, we have learnt about different machine learning algorithms that allowed us to analyse and make predictions on different data sets.

Now that we have come to the end of this course, it is time to compare the performances of some of these algorithms, analyse the results, and discuss the strengths and limitations of each of them.

The data set that we will be using in this assignment 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. You can find more information about the data set [here \(https://www.kaggle.com/jessemostipak/hotel-booking-demand\)](https://www.kaggle.com/jessemostipak/hotel-booking-demand).

The aim is to create meaningful estimators from the data set we have and to select the model that predicts the cancellation best by comparing them with the accuracy scores of different machine learning models.

As you can imagine, the cancellation rate for bookings in the online booking industry is quite high. Once the reservation has been cancelled, there is almost nothing to be done. This may create discomfort for many hotels and thus creates a necessity to take precautions. Therefore, predicting reservations that can be cancelled and preventing these cancellations can create a positive value for the hotels.

[Back to top](#)

Part 1 - Importing the data set

As usual, we will begin this assignment by importing the necessary libraries and the data set.

Run the code cell below to import some of the libraries that we will need throughout this assignment.

```
In [11]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

We will then use `pandas` to import the data set. Complete the code cell below by adding the name of the data set, "hotel_bookings.csv" as a `str` to `.read_csv()`. Assign the dataframe to the variable `df`.

```
In [12]: df=pd.read_csv('hotel_bookings.csv')
```

Before building any machine learning algorithms, we should explore the data.

We begin by visualising the first ten rows of the DataFrame `df` using the function `.head()`. By default, `.head()` displays the first five rows of a DataFrame.

Complete the code cell below by passing the desired number of rows as an `int` to the function `.head()`.

```
In [13]: df.head(10)
```

```
Out[13]:
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_num
0	Resort Hotel	0	342	2015	July	
1	Resort Hotel	0	737	2015	July	
2	Resort Hotel	0	7	2015	July	
3	Resort Hotel	0	13	2015	July	
4	Resort Hotel	0	14	2015	July	
5	Resort Hotel	0	14	2015	July	
6	Resort Hotel	0	0	2015	July	
7	Resort Hotel	0	9	2015	July	
8	Resort Hotel	1	85	2015	July	
9	Resort Hotel	1	75	2015	July	

10 rows × 32 columns

Here's a description of what some of the features in the dataframe above represent:

- `hotelHotel` (H1 = Resort Hotel or H2 = City Hotel)
- `is_canceled` : value indicating if the booking was canceled (1) or not (0)
- `lead_time` : number of days that elapsed between the entering date of the booking into the PMS and the 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
- `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
- `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 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
- `market_segment` : market segment designation. In categories, the term “TA” means “Travel Agents” and “TO” means “Tour Operators”.
- `Type of booking` , assuming one of four categories:
 - Contract - when the booking has an allotment or another 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 another transient booking
 - Transient-party – when the booking is transient, but is associated to at least one other transient booking
- `reservation_status_date` : date at which the last status was set.
- `reservation_status` : status to understand when the booking was canceled or when the customer checked-out of the hotel

Finally, it is always useful to get a sense of how many missing values there are in a dataframe.

Run the code cell below to print the number of NaN's for each column in our dataframe.

```
In [14]: print("# of NaN in each columns:", df.isnull().sum(), sep='\n')
```

```
# of NaN in each columns:
hotel                                0
is_canceled                          0
lead_time                            0
arrival_date_year                     0
arrival_date_month                    0
arrival_date_week_number              0
arrival_date_day_of_month              0
stays_in_weekend_nights                0
stays_in_week_nights                  0
adults                                0
children                              4
babies                                0
meal                                  0
country                              488
market_segment                        0
distribution_channel                  0
is_repeated_guest                     0
previous_cancellations                 0
previous_bookings_not_canceled         0
reserved_room_type                    0
assigned_room_type                    0
booking_changes                       0
deposit_type                          0
agent                                16340
company                              112593
days_in_waiting_list                  0
customer_type                          0
adr                                    0
required_car_parking_spaces            0
total_of_special_requests              0
reservation_status                     0
reservation_status_date                0
dtype: int64
```

[Back to top](#)

Part 2 - Exploratory data analysis (EDA)

To make sure we consider all the correct features to make an accurate prediction, it may be useful to create some plots to have a better understanding of our data.

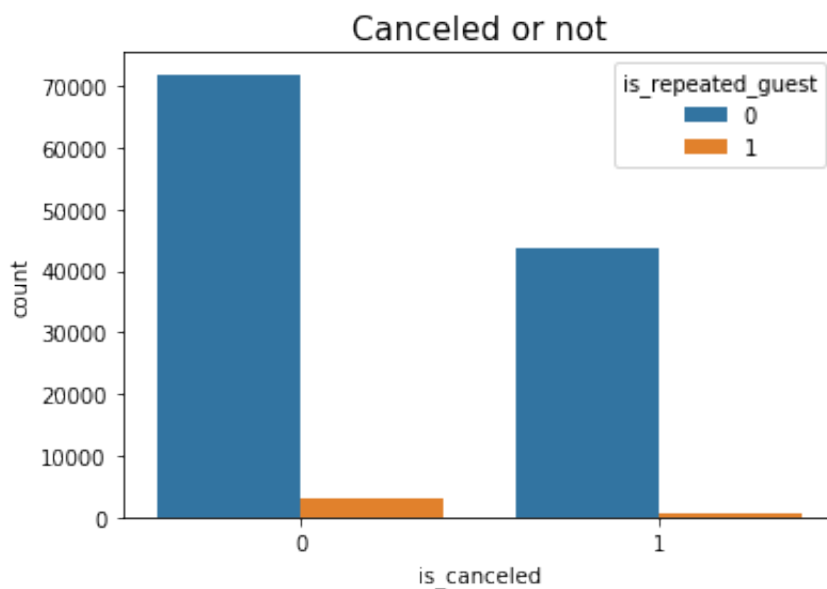
Note: In this part of the assignment, we will be using the Python data visualisation library `seaborn`. Links to the documentation of each function used in this part will be given below.

We could begin by visualising the number of cancellations by repeated guests. A good approach to do so is by using a [count plot](https://seaborn.pydata.org/generated/seaborn.countplot.html) (<https://seaborn.pydata.org/generated/seaborn.countplot.html>), or barplot, that shows the counts of observations in each categorical bin using bars.

Complete the code in the code cell below by setting the following parameters of the function `countplot()` :

- `x = "is_canceled"`,
- `hue = 'is_repeated_guest'`,
- `data = df`

```
In [15]: #plot title
plt.title("Canceled or not", fontdict = {'fontsize': 15})
ax = sns.countplot(x = "is_canceled", hue = 'is_repeated_guest', data = df)
```



What can you infer from the figure above?

DOUBLE CLICK ON THIS CELL TO TYPE YOUR ANSWER

(Correct answer: (not to be shown to students): There is no surprise that repeated guests do not cancel their reservations. Of course, there are some exceptions. Also, most of the customers are not repeated guests.)

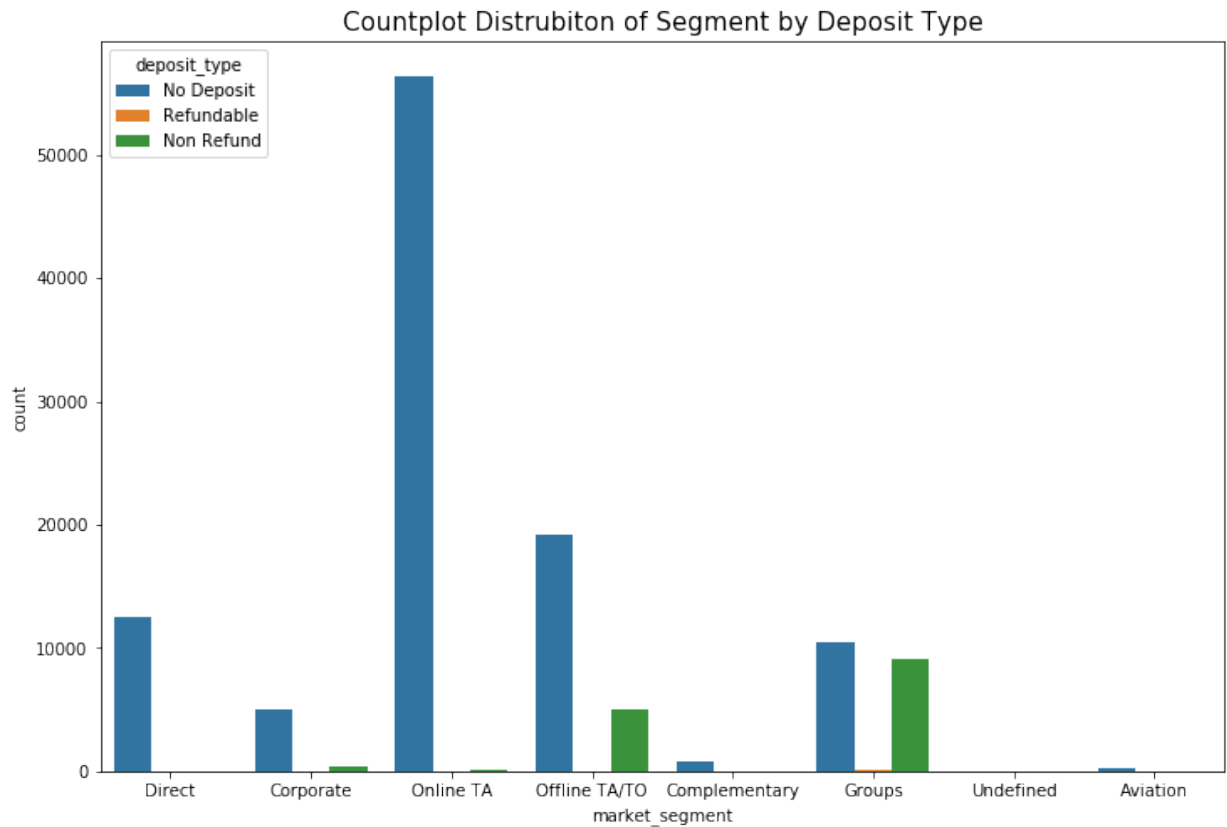
Next, it may be interesting to look at the distribution of market segments by deposit type.

Again, we can use a count plot to visualise this.

Complete the code in the code cell below by setting the following parameters of the function `countplot()` :

- `x = "market_segment"`,
- `hue = 'deposit_type'`,
- `data = df`

```
In [16]: #plot size
plt.figure(figsize = (12,8))
#plot title
plt.title("Countplot Distrubiton of Segment by Deposit Type", fontdict
= {'fontsize':15})
ax = sns.countplot(x = "market_segment", hue = 'deposit_type', data =
df)
```

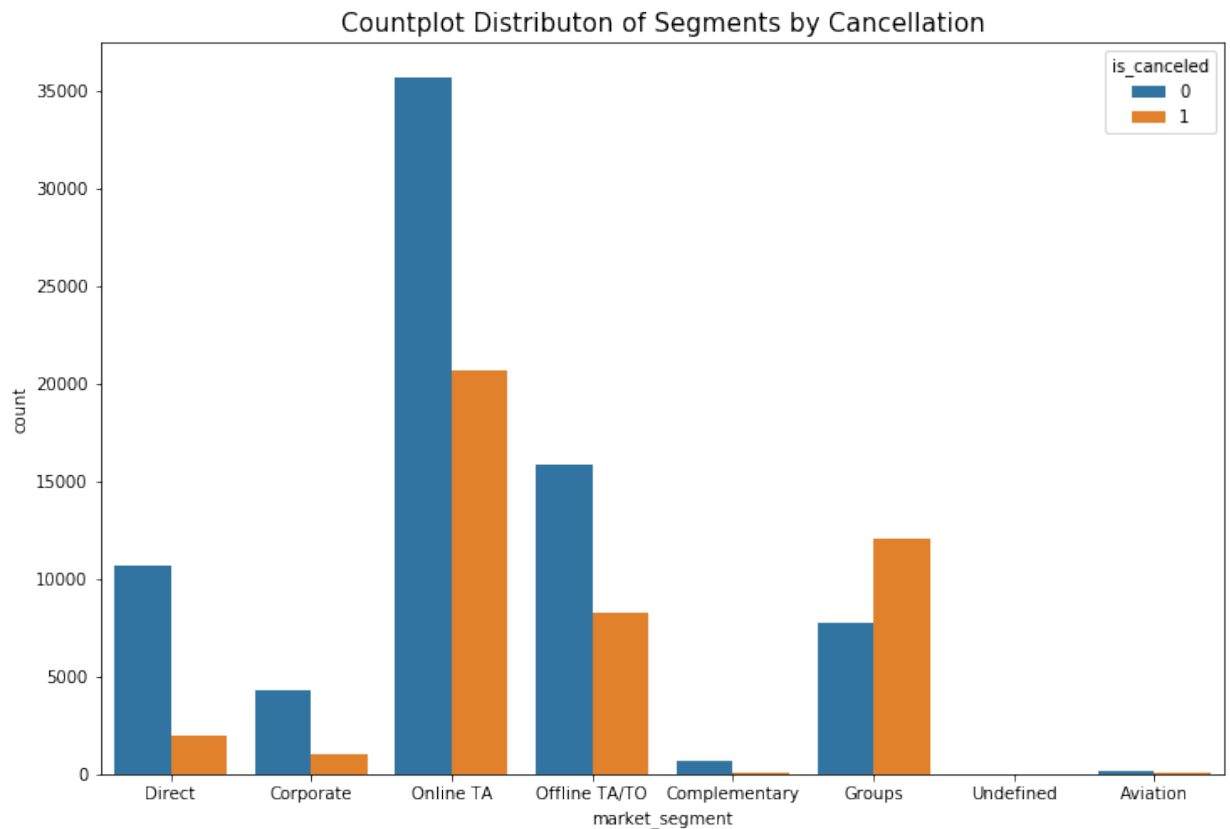



Next, it may be interesting to look at the distribution of market segments by cancellation.

Complete the code in the code cell below by setting the following parameters of the function `countplot()` :

- `x = "market_segment"`,
- `hue = 'is_canceled'`,
- `data = df`

```
In [17]: plt.figure(figsize = (12,8))
plt.title("Countplot Distributon of Segments by Cancellation", fontdic
t = {'fontsize':15})
ax = sns.countplot(x = "market_segment", hue = 'is_canceled', data = d
f)
```



Looking at Offline TA/TO and Groups, the situations where the deposit was received were only in the scenarios where the groups came. It is quite logical to apply a deposit for a large number of customers who will fill important amount of the hotel capacity.

As a first thought, we expected the cancellation rate to be lower in the market segments where deposits are applied. But when we look at the cancellations by market segment, it seems that this is not the case. Deposits were applied to Offline TA/TO and Groups reservations, and not for Direct reservations:

- Groups segment has a cancellation rate more than 50%.
- Offline TA/TO (Travel Agents/Tour Operators) and Online TA has a cancellation rate more than 33%.
- Direct segment has a cancellation rate less than 20%.

It is surprising that the cancellation rate in the first two segments is high despite the application of a deposit. The fact that cancellations are made collectively, like reservations, may explain this situation a bit.

Another insight that looks interesting is that the cancellation rate in the direct segment is very low. We might hypothesise that a mutual trust relationship has been established in the cases where people are communicating one to one.

[Back to top](#)

Part 3 - Preparing our data

In Part 1, we saw that the columns `Company`, `Agent` and `Country` had missing values. Before deciding whether it is a good idea to drop these columns or keep some of them and try to fill the values using imputation, it is always useful to have an understanding of the percentage of missing values.

In the code cell below, define a function called `perc_mv`. Your function should take, as input, a dataframe `x` and a series `y`. Your function should compute the percentage of missing values in the series `y` by using the formula:

$$perc = \frac{\sum(\text{Null values in } y)}{\text{len}(x)} * 100$$

Your function should return the variable `perc`.

HINT: you can compute the sum of null values in `y` by using the same code we used in Part 1 to print the number of missing values for each column. Next, you can sum these null values using the function `sum()`.

```
In [18]: def perc_mv(x, y):  
         perc = y.isnull().sum() / len(x) * 100  
         return perc
```

Run the code cell below to print the percentage of missing values in each column:

```
In [19]: print('Missing value ratios:\nCompany: {}\nAgent: {}\nCountry: {}\nChildren: {}'.format(
    perc_mv(df, df['company']),
    perc_mv(df, df['agent']),
    perc_mv(df, df['country']),
    perc_mv(df, df['children'])))
```

```
Missing value ratios:
Company: 94.30689337465449
Agent: 13.686238378423655
Country: 0.40874445095904177
Children: 0.003350364352123293
```

As we can see, 94.3% of Company column has missing values. Therefore, we do not have enough values to fill the rows of the company column via any imputation method. The best option is to drop the Company column.

In the code cell below, use the function `drop()` to drop the column `Company` from the dataframe. Set the argument `axis = 1`.

```
In [20]: df = df.drop(['company'], axis = 1)
```

We have also four missing values in the `children` column. If there is no information about children, we might infer that those customers do not have any children.

Complete the code cell below to fill the missing values in the column `children` with 0. Use the function `fillna()`.

```
In [21]: df['children'] = df['children'].fillna(0)
```

In [22]: df

Out[22]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week
0	Resort Hotel	0	342	2015	July	
1	Resort Hotel	0	737	2015	July	
2	Resort Hotel	0	7	2015	July	
3	Resort Hotel	0	13	2015	July	
4	Resort Hotel	0	14	2015	July	
...
119385	City Hotel	0	23	2017	August	
119386	City Hotel	0	102	2017	August	
119387	City Hotel	0	34	2017	August	
119388	City Hotel	0	109	2017	August	
119389	City Hotel	0	205	2017	August	

119390 rows × 31 columns

Next, we may want to check the features to create some more meaningful variables and reduce the number of features.

In the code cell below, use the attribute `dtype` on `df` to retrieve the data type for each column.

```
In [23]: df.dtypes
```

```
Out[23]: hotel                object
is_canceled                int64
lead_time                  int64
arrival_date_year          int64
arrival_date_month         object
arrival_date_week_number   int64
arrival_date_day_of_month  int64
stays_in_weekend_nights    int64
stays_in_week_nights       int64
adults                     int64
children                   float64
babies                     int64
meal                       object
country                   object
market_segment             object
distribution_channel        object
is_repeated_guest          int64
previous_cancellations      int64
previous_bookings_not_canceled int64
reserved_room_type         object
assigned_room_type         object
booking_changes            int64
deposit_type               object
agent                      float64
days_in_waiting_list       int64
customer_type              object
adr                        float64
required_car_parking_spaces int64
total_of_special_requests   int64
reservation_status         object
reservation_status_date     object
dtype: object
```

Let's begin with the column `hotel` .

Redefine this column so that it has value 1 if the entry is `Resort Hotel` and value 0 if the entry is `City Hotel` .

HINT: You can accomplish this in a number of ways, such as using a list comprehension and/or by using a lambda function. Here's a pseudocode to help you achieve this:

```
df['hotel'] = [1 if x is equal to "Resort Hotel" or 0 otherwise for x in
df['hotel']]
```

```
In [24]: df['hotel'] = [1 if x == "Resort Hotel" else 0 for x in df['hotel']]
df
```

Out[24]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_
0	1	0	342	2015	July	
1	1	0	737	2015	July	
2	1	0	7	2015	July	
3	1	0	13	2015	July	
4	1	0	14	2015	July	
...
119385	0	0	23	2017	August	
119386	0	0	102	2017	August	
119387	0	0	34	2017	August	
119388	0	0	109	2017	August	
119389	0	0	205	2017	August	

119390 rows × 31 columns

Next, we will take of the column `arrival_date_month`.

Here, we want to map the name of each month to the corresponding calendar number (January = 1, February = 2 and so on). Complete the code cell below by filling the remaining months inside the function `map()`.

```
In [25]: df['arrival_date_month'] = df['arrival_date_month'].map({'January':1,
'February': 2, 'March':3, 'April':4, 'May':5, 'June':6, 'July':7,
'August':8
, 'September':9, 'October':10, 'November':11, 'December':12})

df
```

Out[25]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_
0	1	0	342	2015	7	
1	1	0	737	2015	7	
2	1	0	7	2015	7	
3	1	0	13	2015	7	
4	1	0	14	2015	7	
...
119385	0	0	23	2017	8	
119386	0	0	102	2017	8	
119387	0	0	34	2017	8	
119388	0	0	109	2017	8	
119389	0	0	205	2017	8	

119390 rows × 31 columns

Next, let's try to summarise some features in a smart way.

It may be interesting to include in our analysis whether a reservation was made by a family or not. We don't really have this information yet, but we can assume that if the total number of adults and the number of babies or children are greater than zero, then the reservation was made for a family.

In the code cell below, complete the definition of the function `family`. This function takes, as input, a dataframe `data` and returns a value `val` defined in the following way:

```
def family(data):  
    if number of adults > 0 and number of children > 0:  
        val = 1  
    elif number of adults > 0 and number of babies > 0:  
        val = 1  
    else:  
        val = 0  
    return val
```

Your function should return `val`.

```
In [26]: def family(data):  
    if ((data['adults'] > 0) & (data['children'] > 0)):  
        val = 1  
    elif ((data['adults'] > 0) & (data['babies'] > 0)):  
        val = 1  
    else:  
        val = 0  
    return val
```

Next, we can try to somehow encode some information about the deposit.

In the code cell below, complete the definition of the function `deposit` that takes, as input, a dataframe `data`. Your function should return 0 if no deposit was paid or if the deposit type was refundable and 1 otherwise. Note that this information can be extracted from the column `deposit_type`.

```
def deposit(data):  
    if no deposit or refundable deposit:  
        return 0  
    else:  
        return 1
```

```
In [27]: def deposit(data):  
         if ((data['deposit_type'] == 'No Deposit') | (data['deposit_type']  
         == 'Refundable')):  
             return 0  
         else:  
             return 1
```

In the code cell below, we have defined a function `feature()` that takes a dataframe as input and applies the functions defined above to that dataframe.

```
In [28]: def feature(data):  
         data["is_family"] = data.apply(family, axis = 1)  
         data["total_customer"] = data["adults"] + data["children"] + data[  
         "babies"]  
         data["deposit_given"] = data.apply(deposit, axis=1)  
         data["total_nights"] = data["stays_in_weekend_nights"]+ data["stay  
         s_in_week_nights"]  
         return data
```

Complete the code cell below to apply the function `feature` to our dataframe `df`.

```
In [31]: df = feature(df)
```

Run the code cell below to visualise the new dataframe.

In [32]: df

Out[32]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week
0	1	0	342	2015	7	
1	1	0	737	2015	7	
2	1	0	7	2015	7	
3	1	0	13	2015	7	
4	1	0	14	2015	7	
...
119385	0	0	23	2017	8	
119386	0	0	102	2017	8	
119387	0	0	34	2017	8	
119388	0	0	109	2017	8	
119389	0	0	205	2017	8	

119390 rows × 35 columns

Because of the way we modified our dataframe, some columns like `adults` , `children` , `babies` , `deposit_type` and `reservation_status_update` can be dropped.

In the code cell below, use the function `drop()` to eliminate the columns listed above.

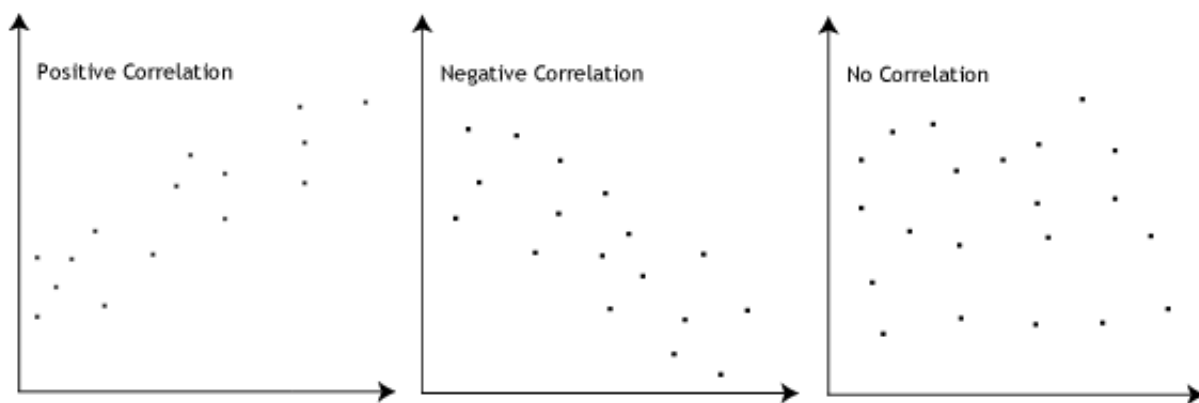
```
In [ ]: df = df.drop(columns = ['adults', 'babies', 'children', 'deposit_type',
                                'reservation_status_date'])
```

[Back to top](#)

Part 4 - Correlation

One way to decide which features are going to be more impactful when running an analysis is to look at the correlation between them.

Pearson's correlations are single value numerical summaries that represent the strength and direction of a linear relationship. Correlation values range from -1 to 1. Values further away from 0 represent stronger relationships, and the sign of the correlation (positive or negative) represents the direction of the relationship. The graphs below depict a visual representation of Pearson correlations.



Run the code cell below to copy the data to check the correlation between variables.

```
In [ ]: cor_df = df.copy()
```

If we observe all of our features, we may assume that some, such as `reservation_status`, might have a higher correlation to `is_cancelled` than others. We can verify this by using label encoding on `reservation_status` and then compute it's correlation value to `is_cancelled`.

Remember, **label encoding** is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering. Label encoding can be implemented in Python using the `sklearn` library.

Complete the code cell below to import `LabelEncoder` from `sklearn.preprocessing`.

```
In [ ]: from sklearn.preprocessing import LabelEncoder
```

Next, use `LabelEncoder()` to define a transform `le`. Finally, use the function `fit_transform()` on the column `reservation_status` to encode the feature as desired. You can find the documentation [here](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>).

```
In [ ]: le = LabelEncoder()
cor_df['reservation_status'] = le.fit_transform(cor_df['reservation_status'])
```

Next, we will visualise the correlation between all of the features.

In the code cell below, use the function `corr()` on `cor_df` to visualise a table with all the correlation values between variables.

```
In [ ]: cor_df.corr()
```

Because we want to try to predict the likelihood of a reservation being cancelled, it is useful to visualise the sorted correlation values for the column `is_canceled`.

In the code cell below, select the column `is_canceled` from the correlation table and apply `sort_values()` to visualise the correlation values in a descending order.

```
In [ ]: cor_df.corr()["is_canceled"].sort_values()
```

What is the most impactful feature?

DOUBLE CLICK ON THIS CELL TO TYPE YOUR ANSWER

(Correct answer: (not to be shown to students): reservation_status)

From the table above, we notice that `arrival_date_week_number`, `stays_in_weekend_nights` and `arrival_date_day_of_month` are not very important when predicting cancellations.

Also, in Part 3, we saw we had some missing values in the `agent` column. It is relevant to predicting cancellation but since the missing values are equal to 13% of the total data, it is better to drop that column. Because the different values in this column represent the ID of the travel agency that made the booking, trying to fill this columns may misguide the predictions.

In the code cell below, use the function `drop()` to eliminate all of the columns listed above from `df`.

```
In [ ]: df = df.drop(columns = [ 'arrival_date_week_number', 'stays_in_weekend_nights', 'arrival_date_month', 'agent'], axis = 1)
```

We can also decide to delete the NA row of the `country` column.

Run the code cell below.

```
In [ ]: indices = df.loc[pd.isna(df["country"]), :].index
df = df.drop(df.index[indices])
```

We are almost ready for modelling! The last thing to do is to make sure that all the variables are converted into a form that could be provided to machine learning algorithms to do a better job in prediction.

Sometimes, in data sets, we encounter columns that contain numbers of no specific order of preference. The data in the columns usually denotes a category or value of the category, also, having the data label encoded in the columns can confuse the machine learning model. To avoid this, the data in the columns should be one-hot encoded.

One-hot encoding refers to splitting the column which contains numerical categorical data to many columns, depending on the number of categories present in that column. Each column contains “0” or “1” corresponding to the column it has been placed. This process can be done using the `pandas` function `get_dummies()`. (You can find the documentation [here \(https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html\)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html)).

In the code cell below, apply the function `get_dummies()` to the columns `meal`, `market_segment`, `distribution_channel`, `reserved_room_type`, `assigned_room_type`, `customer_type`, `reservation_status`.

```
In [ ]: df = pd.get_dummies(data = df, columns = ['meal', 'market_segment', 'distribution_channel', 'reserved_room_type', 'assigned_room_type', 'customer_type', 'reservation_status'])
```

Finally, we need to take care of the column `country`. There are more than 300 classes here, so using label encoding might be more appropriate in this case.

Use `LabelEncoder()` to define a transformer `le`. Finally, use the function `fit_transform()` on the columns `country` and `reservation_status` to encode the feature as desired. You can find the documentation [here \(https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html\)](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html).

```
In [ ]: le = LabelEncoder()  
df['country'] = le.fit_transform(df['country'])
```

[Back to top](#)

Part 5 - Splitting our data for modelling

In this section, we prepare our `x` dataframe and `y` series, separating the features we intend to use from our outcome variable.

Splitting the `x` and the `y` is an important step when pre-processing data. Usually, `x` is our predictors matrix, containing all of the features except for the one we are trying to predict, whereas `y` contains the outcome variable we are interested in.

Complete the code cell below to split our data into `x` and `y`. `x` should contain all features except the outcome variable `is_canceled`.

`y` should contain only the outcome variable `is_canceled` as a pandas series.

```
In [ ]: X = df.drop(['is_canceled'], axis = 'columns')  
y = df['is_canceled']
```

Let's now perform a standard split into a training and test set. We'll use 70% of our data as a training set, and 30% as a test set.

In the code cell below, import the module `train_test_split` from `skikit-learn` and split our `x` and `y` data into training and test sets (`X_train`, `X_test`, `y_train`, `y_test`). Use 30% of the data as a test set, and the remaining 70% for the training set. For reproducibility, set a random state of 42.

```
In [ ]: from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.  
30, random_state=42)
```

[Back to top](#)

Part 6 - Decision tree model (reservation_status included)

Decision trees are a non-parametric supervised learning method used for classification and regression. Decision trees learn from data to approximate a curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules. Decision trees build classification or regression models in the form of trees.

In the code cell below, import the `DecisionTreeClassifier` class from `sklearn.tree` and instantiate the class as `cart`. We'll use `max_depth = 12`.

```
In [ ]: from sklearn.tree import DecisionTreeClassifier
        cart = DecisionTreeClassifier(max_depth = 12)
```

In the code cell below, use the `fit()` method to train the model on the training `x` and `y` sets.

Finally, predict the test set results from the trained `cart_model` object using `X_test` and save them to `y_pred_cart`.

```
In [ ]: cart_model = cart.fit(X_train, y_train)
        y_pred_cart = cart_model.predict(X_test)
```


Let's have a look at the performance of this model! We can do so by printing the accuracy score, the confusion matrix and the AUC score. Remember:

- The *accuracy score* is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- The *confusion matrix* is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarised with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but, more importantly, the types of errors that are being made.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

- The *AUC score* provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.

Run the code cell below to visualise the performance of this model.

```
In [ ]: from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve,
confusion_matrix, auc
print('Decision Tree Model - reservation_status included')

print('Accuracy Score: {}\n\nConfusion Matrix:\n {}\n\nAUC Score: {}'
      .format(accuracy_score(y_test,y_pred_cart), confusion_matrix(y_test,y_pred_cart), roc_auc_score(y_test,y_pred_cart)))
```

Could you say why this model is doing so well?

DOUBLE CLICK ON THIS CELL TO TYPE YOUR ANSWER

(Correct answer: (not to be shown to students): In the correlation, we have seen the impact of reservation status. Reservation status dominates all other features. By keeping reservation_status in the data, it is possible to achieve 100% accuracy rate because that feature is direct way to predict cancellations. However, including reservation_status is cheating, since that information is not known ahead of time.)

In the code cell below, use the function `drop()` to eliminate the columns `reservation_status_Canceled`, `reservation_status_Check-Out`, `reservation_status_No-Show` from `df`.

```
In [ ]: df = df.drop(columns = ['reservation_status_Canceled', 'reservation_status_Check-Out', 'reservation_status_No-Show'], axis = 1)
```

Before going ahead, we need to split our dataframe into an `x` and a `y` set again in the same way as we did Part 5.

Complete the code cell below to achieve so.

```
In [ ]: y = df["is_canceled"]
X = df.drop(["is_canceled"], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

[Back to top](#)

Part 7 - Decision tree model (reservation_status excluded)

In this part, we will recompute the performance of the decision tree model after excluding the reservation status feature.

Complete the code cell below to instantiate the class as `dt`. Again, we'll use `max_depth = 12`.

```
In [ ]: dt = DecisionTreeClassifier(max_depth = 12)
```

In the code cell below, use the `fit()` method to train the model on the training `x` and `y` sets.

Finally, predict the test set results from the trained `dt_model` object using `x_test` and save them to `y_pred_dt`.

```
In [ ]: dt_model = dt.fit(X_train, y_train)
        y_pred_dt = dt_model.predict(X_test)
```

Following the code syntax we used in Part 6, complete the print statement in the code cell below to print the accuracy score, the confusion matrix, and the AUC score.

```
In [ ]: print('Decision Tree Model - reservation_status excluded')

        print('Accuracy Score: {}\n\nConfusion Matrix:\n {} \n\nAUC Score: {}'
              .format(accuracy_score(y_test, y_pred_dt), confusion_matrix(y_test, y_pred_dt),
                      roc_auc_score(y_test, y_pred_dt)))
```

How did this model perform compared to the one used in Part 6?

DOUBLE CLICK ON THIS CELL TO TYPE YOUR ANSWER

(Correct answer: (not to be shown to students): Although the scores of this model are lower than the ones obtained in Part 6, we are confident that this model makes a more accurate prediction because we are not cheating using the `reservation_status` column.)

[Back to top](#)

Part 8 - K-nearest neighbours

The K-nearest neighbours algorithm is an easy, straight-forward algorithm that's easy to implement using Python. It simply works by calculating the distance between new data points and classifying them based on how close it is to existing, labeled data from the training set. Thus, it's considered a *lazy* algorithm.

K-nearest neighbours only requires that you specify two parameters in `scikit-learn` -- the value of `K` (the number of neighbours, `n_neighbors` in Python), and the distance metric used (i.e., Euclidean distance or Manhattan distance, specified by the parameter `p`).

In the code cell below, import the `KNeighborsClassifier` class from `sklearn.neighbors` and instantiate the class as `knn`. We'll use `n_neighbors = 10` and `p = 1` to represent the Manhattan Distance algorithm.

```
In [ ]: from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 10, p = 1)
```

In the code cell below, use the `fit()` method to train the model on the training `x` and `y` sets.

Finally, predict the test set results from the trained `knn_model` object using `x_test` and save them to `y_pred_knn`.

```
In [ ]: knn_model = knn.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
```

Following the code syntax we used in Part 7, complete the print statement in the code cell below to print the accuracy score, the confusion matrix and the AUC score to screen for the K-nearest neighbours algorithm.

```
In [ ]: print('K-Nearest Neighbors')

print('Accuracy Score: {}\n\nConfusion Matrix:\n {} \n\nAUC Score: {}'
      .format(accuracy_score(y_test,y_pred_knn), confusion_matrix(y_test,y_pred_knn), roc_auc_score(y_test,y_pred_knn)))
```

How do the values of the confusion matrix above compare to the ones obtained in Part 7? What can you infer from this?

DOUBLE CLICK ON THIS CELL TO TYPE YOUR ANSWER

[Back to top](#)

Part 9 - Naïve Bayes

Naïve Bayes is a classification technique based on Bayes' theorem with an assumption of independence among predictors.

In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

In the code cell below, import the `GaussianNB` class from `sklearn.naive_bayes` and instantiate the class as `nb`.

```
In [ ]: from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
```

In the code cell below, use the `fit()` method to train the model on the training `x` and `y` sets.

Finally, predict the test set results from the trained `nb_model` object using `x_test` and save them to `y_pred_nb`.

```
In [ ]: nb_model = nb.fit(X_train, y_train)
        y_pred_nb = nb_model.predict(X_test)
```

Following the code syntax we used in Part 7, complete the print statement in the code cell below to print the accuracy score, the confusion matrix and the AUC score to screen for the Naïve Bayes algorithm.

```
In [ ]: print('Naïve Bayes')

        print('Accuracy Score: {} \n \n Confusion Matrix: \n {} \n \n AUC Score: {}'
              .format(accuracy_score(y_test, y_pred_nb), confusion_matrix(y_test, y_pred_nb), roc_auc_score(y_test, y_pred_nb)))
```

Why did the Naïve Bayes classifier perform poorly compared to other models?

DOUBLE CLICK ON THIS CELL TO TYPE YOUR ANSWER

[Back to top](#)

Part 10 - Conclusion

Based on all the results obtained above, which algorithm seems to make the most accurate prediction?

DOUBLE CLICK ON THIS CELL TO TYPE YOUR ANSWER

Based on all the results obtained above, which algorithm seems to make the least accurate prediction?

DOUBLE CLICK ON THIS CELL TO TYPE YOUR ANSWER

(Correct answer: (not to be shown to students): Naïve Bayes)

CONGRATULATIONS ON COMPLETING THE FINAL ASSIGNMENT!

```
In [ ]:
```