$$Study of the Hotel B \infty k \in gdem \text{ and } Dataset$$

Contents

This dataset contain information of hotel booking, We will perform exploratory data analysis to get insight from the data.

- Data loading
- Data analysis

We will try to answer the following Questions

1. How Many Booking Were Cancelled?
2. What is the booking ratio between Resort Hotel and City Hotel?
3. What is the percentage of booking for each year?
4. Which is the most busy month for hotel?
5. From which country most guest come?
6. How Long People Stay in the hotel?
7. Which was the most booked accommodation type (Single, Couple, Family)?

After that we will make predictive model to predict whether the booking will be cancelled or not

We will:

- Perform the Feature Engineering to make new featuers
- Perform the Data Selection to select only relevant features
- Tranform the Data (Categorial to Numerical)
- Split the data (Train Test Split)
- Model the data (Fit the Data)
- And finally Evaluate our model

Packages to import

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from itertools import cycle, islice
import seaborn as sns
import pycountry as pc

pd.options.display.max_columns = None
```

In [2]:
```
hotel= pd.read_csv("hotel_bookings.csv", sep=';')
```

In [3]:
```
hotel
```

Out[3]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number
0	Resort Hotel	0	342	2015	July	27

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number
1	Resort Hotel	0	737	2015	July	27
2	Resort Hotel	0	7	2015	July	27
3	Resort Hotel	0	13	2015	July	27
4	Resort Hotel	0	14	2015	July	27
...	...	...	...	...	...	...
9385	City Hotel	0	23	2017	August	35
9386	City Hotel	0	102	2017	August	35
9387	City Hotel	0	34	2017	August	35
9388	City Hotel	0	109	2017	August	35
9389	City Hotel	0	205	2017	August	35

)390 rows × 32 columns

In [5]:
```
hotel.shape
```

Out[5]:
```
(119390, 32)
```

Notre jeu de données contient 119390 observations et 32 variables qui sont: 1. La première colonne est nommée Hotel et elle contient deux modalités H1 = Resort Hotel et H2 = City Hotel). 2. La deuxième colonne est nommée is_canceled. C'est une variable qui nous indique si la réservation a été annulée. Et elle contient deux modalités (1) si la réservation a été annulé et (0) si elle est conservée. 3. La troisième colonne est nommée lead_time. Cette variable nous indique le Nombre de jours qui se sont écoulés entre la date d'entrée de la réservation dans le PMS et la date d'arrivée. 4. La quatrième colonne est nommée arrival_date_year. Cette variable nous indique l'Année de la date d'arrivée du client. 5. La cinquième colonne est nommée arrival_date_month. Cette variable nous indique le Mois de la date d'arrivée du client. 6. La sixième colonne est nommée arrival_date_week_number. Cette variable nous indique le Numéro de semaine de l'année pour la date d'arrivée du client. 7. La septième colonne est nommée arrival_date_day_of_month. Cette variable nous indique le Jour de la date d'arrivée du client. 8. La huitième colonne est nommée stays_in_weekend_nights. Cette variable nous indique le Nombre de nuits de week-end (samedi ou dimanche) où le client a séjourné ou réservé son séjour à l'hôtel. 9. La neuvième colonne est nommée stays_in_week_nights. Cette variable nous indique le nombre de nuits en semaine (du lundi au vendredi) où le client a séjourné ou réservé son séjour à l'hôtel. 10. La dixième colonne est nommée adults. Cette variable nous indique le nombre de personnes adultes qui sont concernée par la réservation. 11. La onzième colonne est nommée children. Elle nous indique le Nombre d'enfants concernée par la réservation. 12. La douzième colonne es nommée babies. Cette variable nous indique le nombre de bébés inclus dans la réservation. 13. La treizième colonne est nommée meal. Cette variable nous indique le Type de repas réservé. Les catégories sont présentées selon les forfaits repas standard de l'hôtellerie : Undefined/SC - aucun forfait repas ; BB - Bed & Breakfast ; HB (Half board) - Demi-pension (petit-déjeuner et un autre repas - généralement le dîner) ; FB (Full board) - Pension complète (petit-déjeuner, déjeuner et dîner). 14. La quatorzième colonne est nommée country. Cette variable nous indique le pays d'origine du client. Les catégories sont représentées dans le format ISO 3155-3:2013 15. La Quinzième colonne est nommée market_segment. Cette variable nous indique la Désignation du segment de marché. Dans les catégories, le terme "TA" (Travel Agents) qui signifie "agents de voyage" et "TO" (Tour Operators) qui signifie "tour-opérateurs". 16. La seizième colonne est nommée

distribution_channel. Cette variable nous indique le Canal de distribution des réservations. le terme "TA" (Travel Agents) qui signifie "agents de voyage" et "TO" (Tour Operators) qui signifie "tour-opérateurs". 17. La dix-septième colonne est nommée is_repeated_guest. Cette variable nous présente une Valeur indiquant si le nom de la réservation provient d'un invité répété. Et elle comporte deux modalités (1) si c'est oui et (0) si c'est non. 18. La dix-huitième colonne est nommée previous_cancellations. Cette variable nous indique le nombre de réservations précédentes qui ont été annulées par le client avant la réservation actuelle. 19. La dix-neuvième colonne est nommée previous_bookings_not_canceled. Cette variable nous indique le nombre de réservations précédentes qui n'ont pas été annulées par le client avant la réservation actuelle. 20. La vingtième colonne est nommée reserved_room_type. Cette variable nous indique le code du type de chambre réservé. Le code est présenté à la place de la désignation pour des raisons d'anonymat. 21. La Vingt-unième colonne est nommée assigned_room_type. Cette variable nous indique le code pour le type de chambre attribué à la réservation. Il arrive que le type de chambre attribué diffère du type de chambre réservé pour des raisons de fonctionnement de l'hôtel (par exemple, surréservation) ou à la demande du client. Le code est présenté à la place de la désignation pour des raisons d'anonymat. 22. La vingt-deuxième colonne est nommée booking_changes. Cette variable nous indique le nombre de changements/modifications apportés à la réservation depuis le moment où la réservation a été saisie dans le PMS jusqu'au moment de l'enregistrement ou de l'annulation. 23. La vingt-troisième colonne est nommée deposit_type. Cette variable nous indique si le client a effectué un dépôt pour garantir la réservation. Cette variable peut être classée en trois catégories : No Deposit - aucun dépôt n'a été effectué ; Non Refund - un dépôt a été effectué dans la valeur du coût total du séjour ; Refundable - un dépôt a été effectué avec une valeur inférieure au coût total du séjour. 24. La vingt-quatrième colonne est nommée agent. Cette variable nous indique le ID de l'agence de voyage qui a effectué la réservation. 25. La vingt-cinquième colonne est nommée company. Cette variable nous indique le ID de la société/entité qui a effectué la réservation ou qui est responsable du paiement de la réservation. L'ID est présenté au lieu de la désignation pour des raisons d'anonymat. 26. La vingt-sixième colonne est nommée days_in_waiting_list. Cette variable nous indique le nombre de jours où la réservation est restée sur la liste d'attente avant d'être confirmée au client. 27. La vingt-septième colonne est nommée customer_type. Cette variable nous indique le Type de réservation, en assumant l'une des quatre catégories suivantes : Contract (contrat) : lorsque la réservation est associée à un allotissement ou à un autre type de contrat ; Group (Groupe) : lorsque la réservation est associée à un groupe ; Transient (Transitoire) : lorsque la réservation ne fait pas partie d'un groupe ou d'un contrat, et n'est pas associée à une autre réservation transitoire ; Transient-party (Transitoire-partie) - lorsque la réservation est transitoire, mais est associée à au moins une autre réservation transitoire. 28. La vingt-huitième colonne est nommée adr. Cette variable nous indique le tarif journalier moyen et est défini en divisant la somme de toutes les transactions d'hébergement par le nombre total de nuits d'hébergement. 29. La vingt-neuvième colonne est nommée required_car_parking_spaces. Cette variable nous indique le nombre d'emplacements de parking requis par le client. 30. La trentième colonne est nommée total_of_special_requests. Cette variable nous indique le nombre de demandes spéciales faites par le client (par exemple, lit double ou étage élevé). 31. La trente-unième colonne est nommée reservation_status. Cette variable nous indique le dernier statut de la réservation, en supposant l'une des trois catégories suivantes : Canceled : la réservation a été annulée par le client ; Check-Out : le client a terminé son séjour et a déjà quitté l'hôtel; No-Show - le client ne s'est pas présenté et a informé l'hôtel de la raison de son absence. 32. La trente-deuxième colonne est nommée reservation_status_date. Cette variable nous indique la Date à laquelle le dernier statut a été défini. Cette variable peut être utilisée en conjonction avec reservation_status pour comprendre quand la réservation a été annulée ou quand le client a quitté l'hôtel.

In [ ]:

Data analysis

Display the dataset to view all the data and see which ones are missing.

Management of missing values

In [6]:
```
data = hotel.copy()
```

In [7]:
```
data.describe()
```

Out[7]:

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of
count	119390.000000	119390.000000	119390.000000	119390.000000	11939(

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_o
mean	0.370416	104.011416	2016.156554	27.165173	1
std	0.482918	106.863097	0.707476	13.605138	
min	0.000000	0.000000	2015.000000	1.000000	
25%	0.000000	18.000000	2016.000000	16.000000	
50%	0.000000	69.000000	2016.000000	28.000000	1
75%	1.000000	160.000000	2017.000000	38.000000	2
max	1.000000	737.000000	2017.000000	53.000000	3

Interprétation : - Pour la première variable is_canceled : c'est une variable à deux modalités 0 et 1. Donc il est difficile de l'interpréter. - Pour la variable lead_time : on peut constater que le nombre d'observations est de : 119.390 dont les valeurs sont comprises entre 0 et 737 et la moyenne est de 104.0114, la médiane est de 69 qui est nettement inférieure à la moyenne. Ceci explique qu'on a beaucoup de données aberrantes pour cette variable. - Pour la variable arrival_date_year : On a également aussi 119.390 observations dont les valeurs sont comprises entre 2015 et 2017 avec une moyenne de 2016 et la médiane est de 2016 également, ce qui montre qu'on a pas vraiment de valeurs aberrantes pour cette variable et que les données sont mieux distribuées. - Pour la variable arrival_date_week_number nous avons aussi 119.390 observations dont les valeurs sont réparties entre 1 et 53 qui correspondent aux numéros de semaines à laquelle le client s'est présenté à l'hôtel. On a une moyenne de 27 et la médiane est 28, ce qui montre qu'on a pas beaucoup de valeurs aberrantes pour cette variable et que les données sont mieux distribuées.

In [8]:
```
data.info()
```

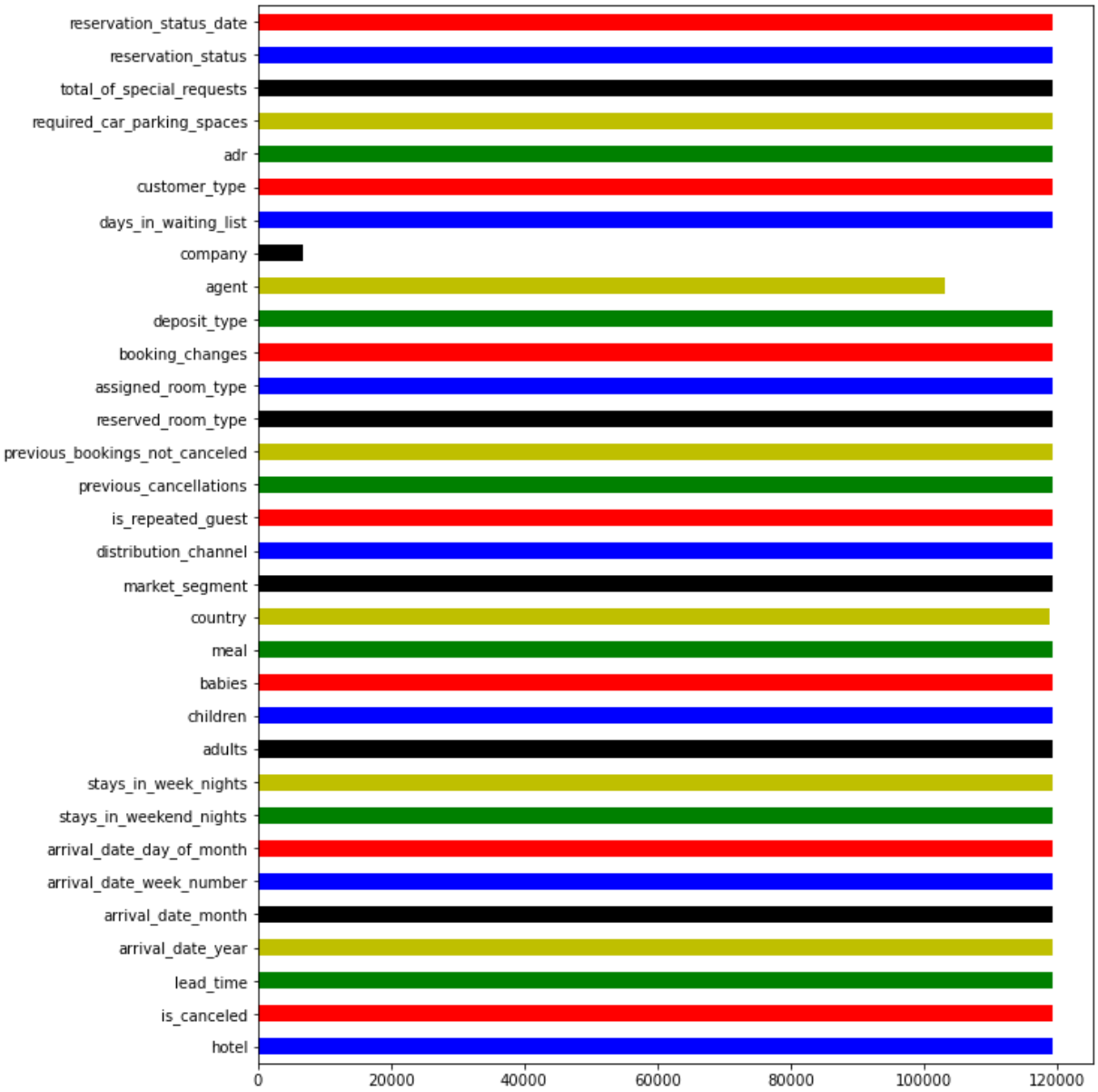
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   hotel                           119390 non-null  object 
 1   is_canceled                     119390 non-null  int64  
 2   lead_time                       119390 non-null  int64  
 3   arrival_date_year               119390 non-null  int64  
 4   arrival_date_month              119390 non-null  object 
 5   arrival_date_week_number        119390 non-null  int64  
 6   arrival_date_day_of_month       119390 non-null  int64  
 7   stays_in_weekend_nights         119390 non-null  int64  
 8   stays_in_week_nights            119390 non-null  int64  
 9   adults                          119390 non-null  int64  
 10  children                        119386 non-null  float64
 11  babies                          119390 non-null  int64  
 12  meal                            119390 non-null  object 
 13  country                         118902 non-null  object 
 14  market_segment                  119390 non-null  object 
 15  distribution_channel            119390 non-null  object 
 16  is_repeated_guest               119390 non-null  int64  
 17  previous_cancellations          119390 non-null  int64  
 18  previous_bookings_not_canceled  119390 non-null  int64  
 19  reserved_room_type              119390 non-null  object 
 20  assigned_room_type              119390 non-null  object 
 21  booking_changes                 119390 non-null  int64  
 22  deposit_type                    119390 non-null  object 
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64  
 26  customer_type                   119390 non-null  object 
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64  
 29  total_of_special_requests       119390 non-null  int64  
 30  reservation_status              119390 non-null  object 
```

```
 31  reservation_status_date         119390 non-null  object
dtypes: float64(4), int64(16), object(12)
memory usage: 29.1+ MB
```

In [9]:
```
my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None, len(hotel)))
```

In [10]:
```
data.count().plot(kind="barh", stacked=True, color=my_colors, figsize=(10, 13))
```

Out[10]: <AxesSubplot:>



In [11]:
```
data.isnull().sum().sort_values(ascending=False)[:]
```

Out[11]:
```
company                         112593
agent                            16340
country                            488
children                             4
lead_time                            0
arrival_date_year                    0
arrival_date_month                   0
arrival_date_week_number             0
is_canceled                          0
market_segment                       0
arrival_date_day_of_month            0
stays_in_weekend_nights              0
stays_in_week_nights                 0
adults                               0
babies                               0
```

```
meal                            0
reservation_status_date         0
distribution_channel            0
reservation_status              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
days_in_waiting_list            0
customer_type                   0
adr                             0
required_car_parking_spaces     0
total_of_special_requests       0
hotel                           0
dtype: int64
```

Ici nous pouvons voir exactement toutes les colonnes où il manque de données. On constate qu'il y'a au total quatre colonnes qui ont des valeurs manquantes. Il s'agit des colonnes suivantes : - La colonne "company" qui contient 6797 dont il manque 112593 lignes manquantes. - La colonne "agent" qui contient 103050 lignes dont il manque 16340 lignes. - La colonne "country" qui contient 118902 lignes dont il manque 488 lignes. - La colonne "children" qui contient également 119386 lignes dont il manque 4 lignes.

In [12]:
```
list_quanti= ['lead_time', 'stays_in_weekend_nights', 'stays_in_weekend_nights', 'ad
```

In [13]:
```
data[list_quanti].corr(method ='pearson')
```

Out[13]:

	lead_time	stays_in_weekend_nights	stays_in_weekend_nights	adults
lead_time	1.000000	0.085671	0.085671	0.119519
stays_in_weekend_nights	0.085671	1.000000	1.000000	0.091871
stays_in_weekend_nights	0.085671	1.000000	1.000000	0.091871
adults	0.119519	0.091871	0.091871	1.000000
children	-0.037622	0.045793	0.045793	0.030447
babies	-0.020915	0.018483	0.018483	0.018146
booking_changes	0.000149	0.063281	0.063281	-0.051673
days_in_waiting_list	0.170084	-0.054151	-0.054151	-0.008283
adr	-0.063077	0.049342	0.049342	0.230641
required_car_parking_spaces	-0.116451	-0.018554	-0.018554	0.014785
total_of_special_requests	-0.095712	0.072671	0.072671	0.122884

In [ ]:
```

```

Data PreProcessing

1. Drop Rows where there is no adult, baby and child

In [14]:
```
data = data.drop(data[(data.adults+data.babies+data.children)==0].index)
```

In [15]:
```
data.shape
```

Out[15]:
```
(119210, 32)
```

2. If there is no id of agent or the company is null, we will just replace it with 0

In [16]:
```
data[['agent','company']] = data[['agent','company']].fillna(0.0)
```

3. For the missing values in the country column, we will replace it with the mode (value that appears most often)

In [17]:
```
data['country'].fillna(data.country.mode().to_string(), inplace=True)
```

4. For missing children value, replace it with rounded mean value

In [18]:
```
data['children'].fillna(round(data.children.mean(), 3), inplace=True)
```

In [19]:
```
data.isnull().sum().sort_values(ascending=False)[:]
```

Out[19]:
```
reservation_status_date           0
reservation_status                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                          0
babies                            0
meal                              0
country                           0
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                             0
company                           0
days_in_waiting_list              0
customer_type                     0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
hotel                             0
dtype: int64
```

2. Converting Datatype

Convert datatype of the columns: children, comapny and agent from float to integer

In [20]:
```
data[['children', 'company', 'agent', 'adr']] = data[['children', 'company', 'agent'
```

Definition of some methods

In [21]:
```
def display(x, y, label_x=None,label_y=None, title=None, figsize=(7,5), type='bar'):

    '''
    INPUT:
        x:        Array containing values for x-axis
        y:        Array containing values for y-axis
```

```
        lable_x:  String value for x-axis label
        lable_y:  String value for y-axis label
        title:    String value for plot title
        figsize:  tuple value, for figure size
        type:     type of plot (default is bar plot)

    OUTPUT:
        Display the plot
    '''

    sns.set_style('darkgrid')

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

    if label_x != None:
        ax.set_xlabel(label_x)

    if label_y != None:
        ax.set_ylabel(label_y)

    if title != None:
        ax.set_title(title)

    if type == 'bar':
        sns.barplot(x,y, ax = ax)
    elif type == 'line':
        sns.lineplot(x,y, ax = ax)


    plt.show()
```

In [22]:
```
def get_value(data, limit=None):

    '''
    INPUT:
        series: Pandas Series (Single Column from DataFrame)
        limit:  If value given, limit the output value to first limit samples.
    OUTPUT:
        x = Unique values
        y = Count of unique values
    '''

    if limit != None:
        data = data.value_counts()[:limit]
    else:
        data = data.value_counts()

    x = data.index
    y = data/data.sum()*100

    return x.values,y.values
```

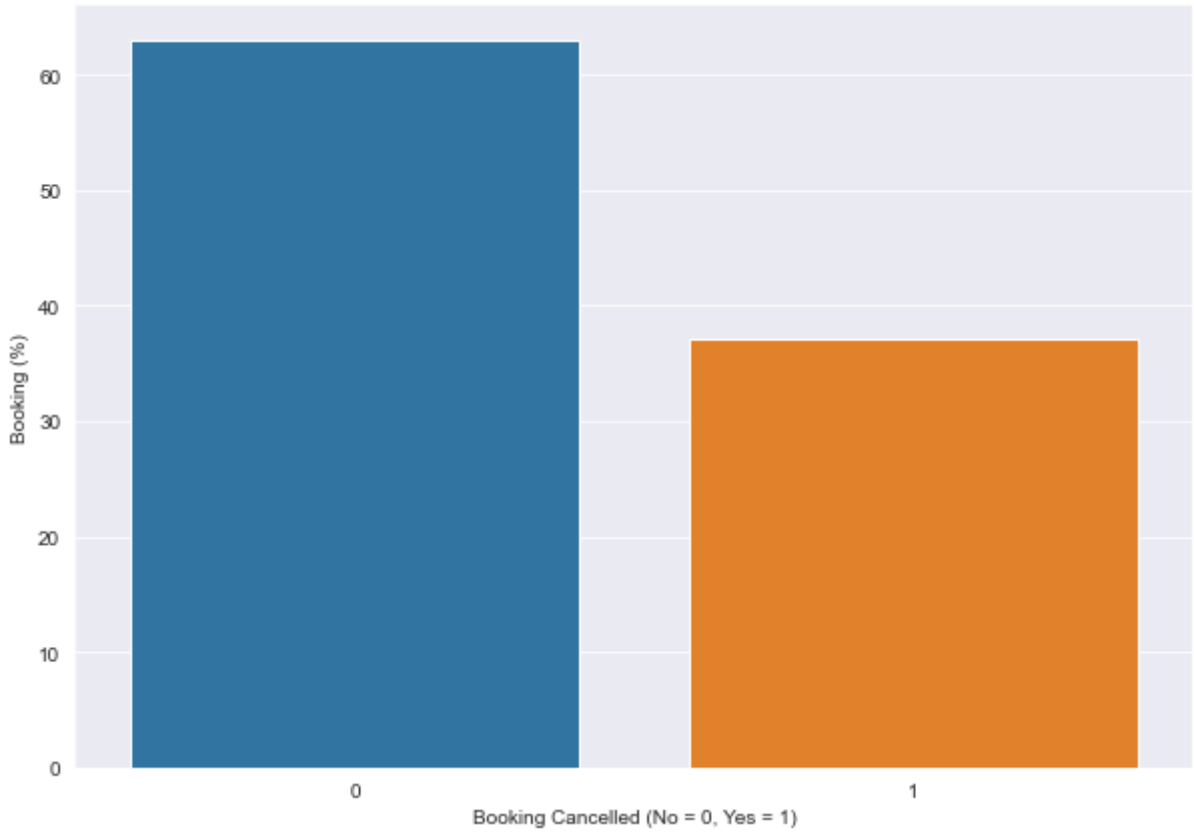
Let's start answering our questions

1. How many reservations were cancelled?

In [23]:
```
x,y = get_value(data['is_canceled'])
display(x,y, label_x='Booking Cancelled (No = 0, Yes = 1)', label_y='Booking (%)', f
```

```
B:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
lowing variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword wil
```

```
l result in an error or misinterpretation.
  warnings.warn(
```



For a more in-depth analysis, let's select only those reservations that have not been cancelled.

In [24]:
```
data_not_canceled = data[data['is_canceled'] == 0]
```

In [25]:
```
data_not_canceled
```

Out[25]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_numb
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	

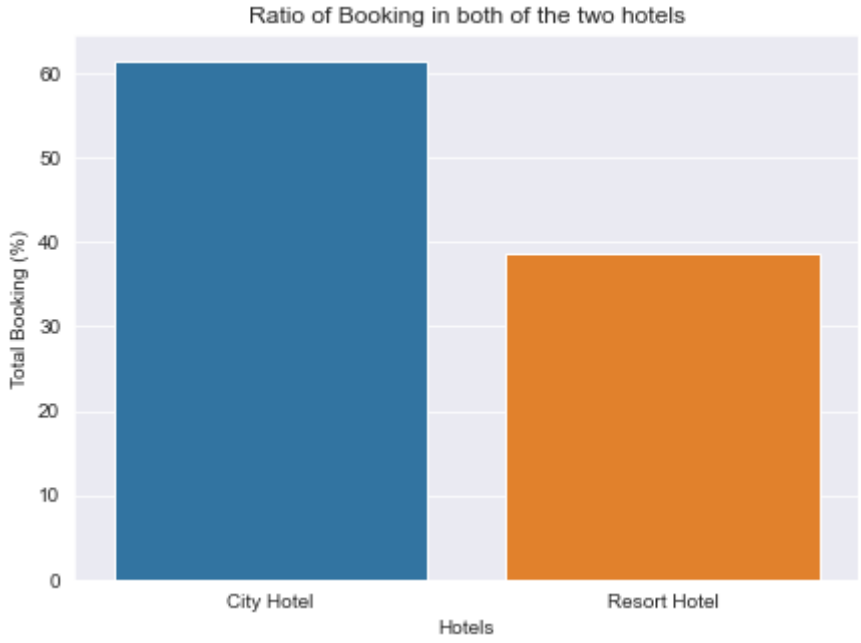
	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_numb
119388	City Hotel	0	109	2017	August	
119389	City Hotel	0	205	2017	August	

75011 rows × 32 columns

2. What is the booking ratio between Resort Hotel and City Hotel?

In [26]:
```
x,y = get_value(data_not_canceled['hotel'])
display(x,y, label_x='Hotels', label_y='Total Booking (%)', title='Ratio of Booking
```

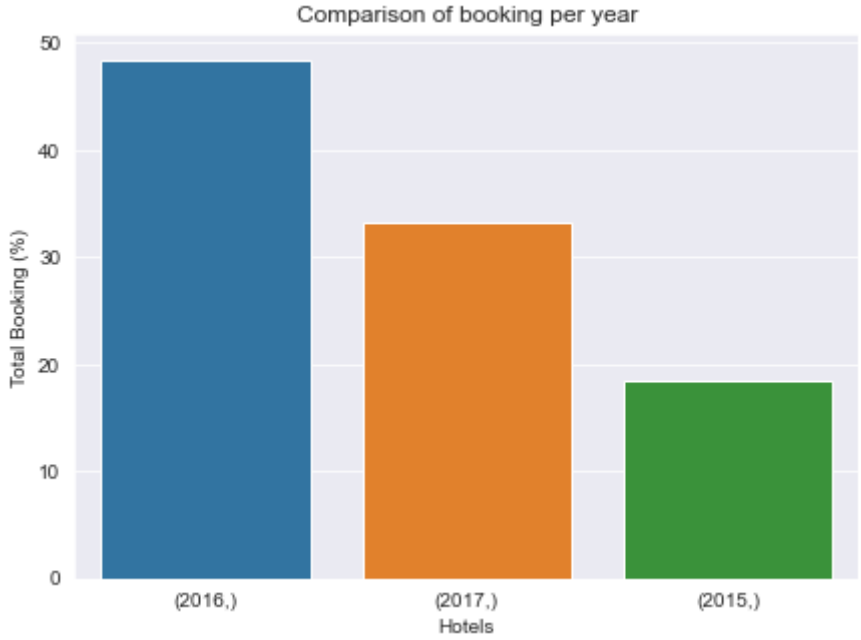
```
B:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
lowing variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword wil
l result in an error or misinterpretation.
  warnings.warn(
```



3. What is the percentage of booking for each year per hotel?

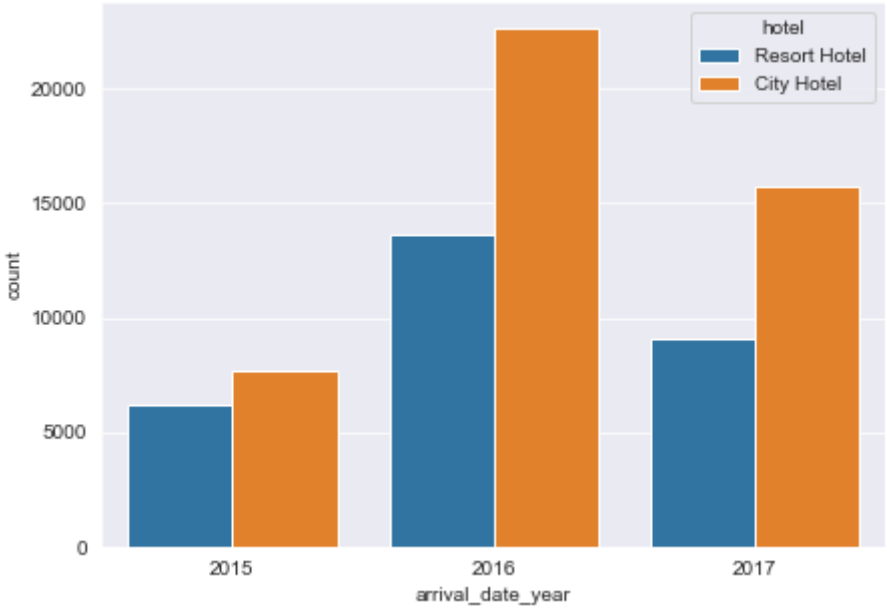
In [27]:
```
x,y = get_value(data_not_canceled[['arrival_date_year']])
display(x,y, label_x='Hotels', label_y='Total Booking (%)', title='Comparison of boo
```

```
B:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
lowing variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword wil
l result in an error or misinterpretation.
  warnings.warn(
```



Let's separte it by hotel

In [28]:
```
plt.subplots(figsize=(7,5))
sns.countplot(x='arrival_date_year', hue='hotel',  data=data_not_canceled);
```

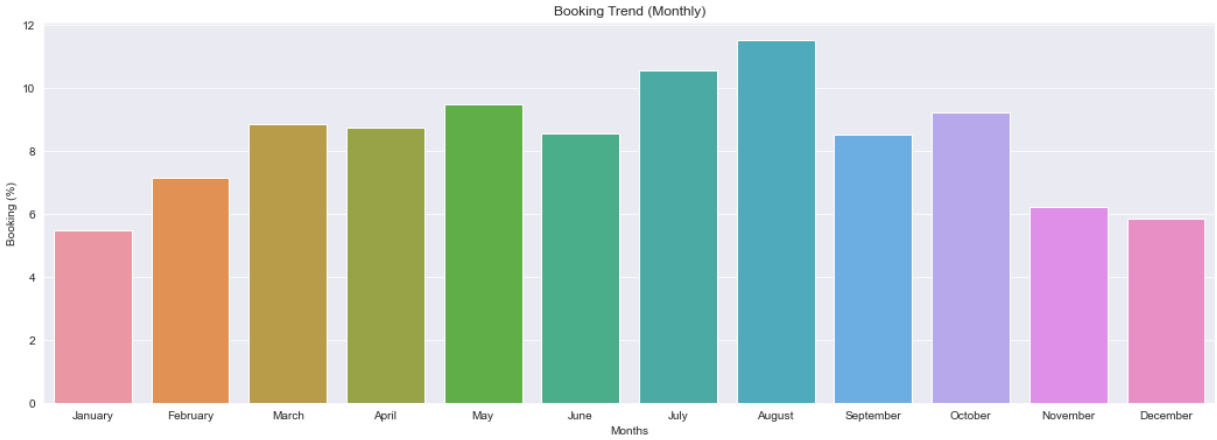


4. Which is the most busy month for hotel?

In [29]:
```
new_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August

sorted_months = data_not_canceled['arrival_date_month'].value_counts().reindex(new_o

x = sorted_months.index
y = sorted_months/sorted_months.sum()*100


#sns.lineplot(x, y.values)
display(x, y.values, label_x='Months', label_y='Booking (%)', title='Booking Trend (
```

```
B:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
lowing variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword wil
```

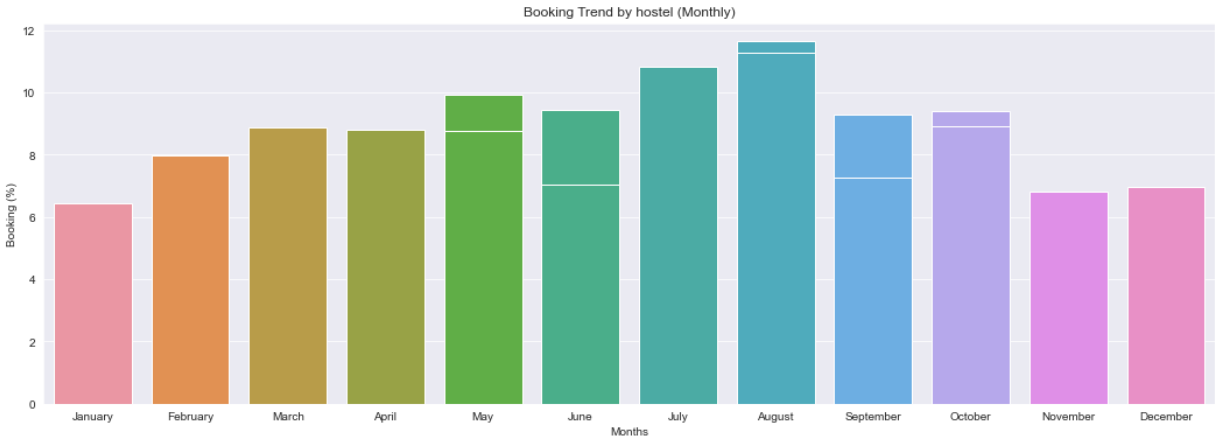
```
l result in an error or misinterpretation.
  warnings.warn(
```



In [30]:

```
## Order of months
new_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August


## Select only City Hotel
sorted_months = data_not_canceled.loc[data.hotel=='City Hotel' ,'arrival_date_month'

x1 = sorted_months.index
y1 = sorted_months/sorted_months.sum()*100


## Select only Resort Hotel
sorted_months = data_not_canceled.loc[data.hotel=='Resort Hotel' ,'arrival_date_mont

x2 = sorted_months.index
y2 = sorted_months/sorted_months.sum()*100

## Draw the line plot

fig, ax = plt.subplots(figsize=(18,6))

ax.set_xlabel('Months')
ax.set_ylabel('Booking (%)')
ax.set_title('Booking Trend by hostel (Monthly)')

sns.barplot(x1, y1.values, label='City Hotel')
sns.barplot(x2, y2.values, label='Resort Hotel')

plt.show()
```

```
B:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
lowing variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword wil
l result in an error or misinterpretation.
  warnings.warn(
B:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
lowing variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword wil
l result in an error or misinterpretation.
  warnings.warn(
```



5. From which country most guest come?

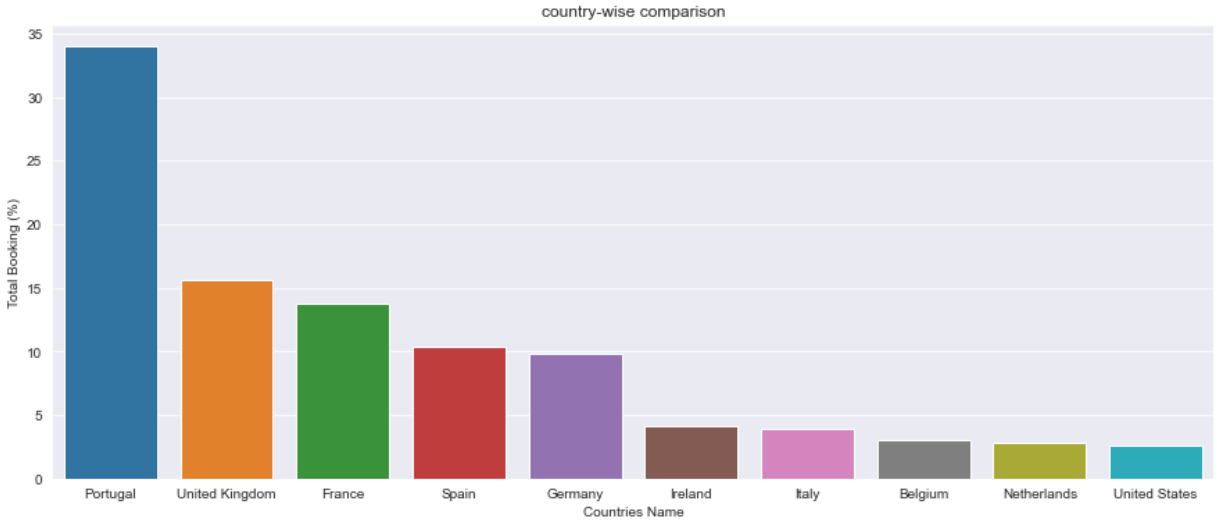
pycountry is very useful python package.

We will use this package to get country names from country codes

- https://github.com/flyingcircusio/pycountry
- https://pypi.org/project/pycountry/

In [31]:
```
import pycountry as pc

x,y = get_value(data_not_canceled['country'], limit=10)

## For each country code select the country name
country_name = [pc.countries.get(alpha_3=name).name for name in x]

display(country_name,y, label_x='Countries Name', label_y='Total Booking (%)', title
```

```
B:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
lowing variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword wil
l result in an error or misinterpretation.
  warnings.warn(
```

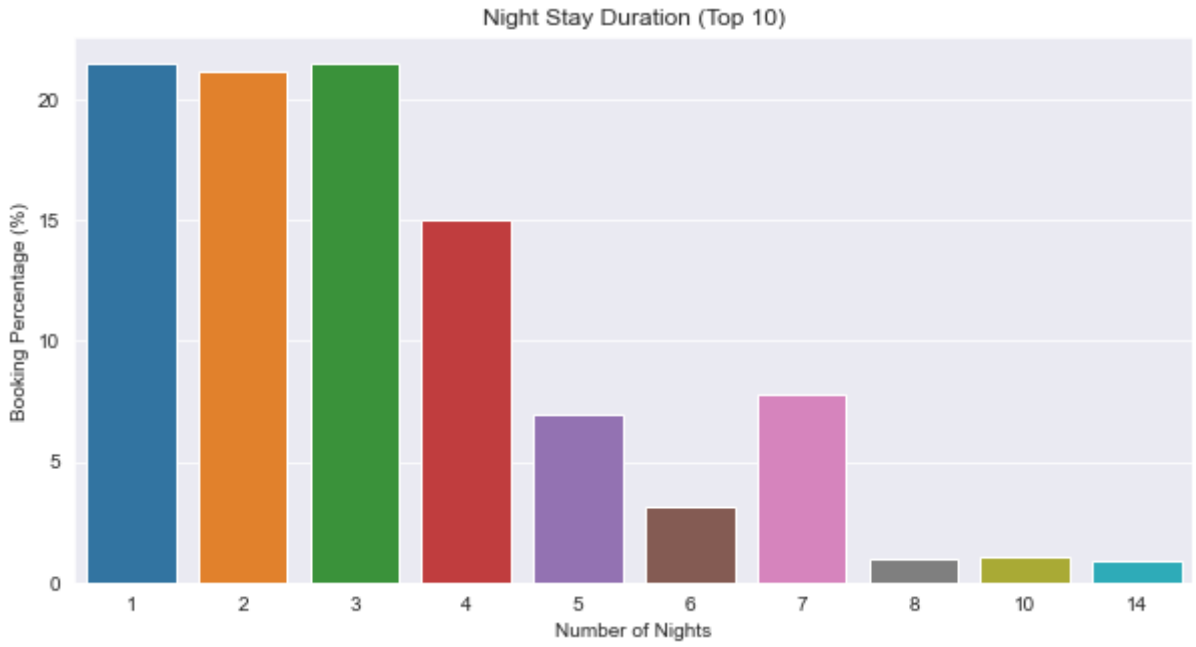


6. How Long People Stay in the hotel?

In [32]:
```
total_nights = data_not_canceled['stays_in_weekend_nights']+ data_not_canceled['stay
x,y = get_value(total_nights, limit=10)

display(x,y, label_x='Number of Nights', label_y='Booking Percentage (%)', title='Ni
```

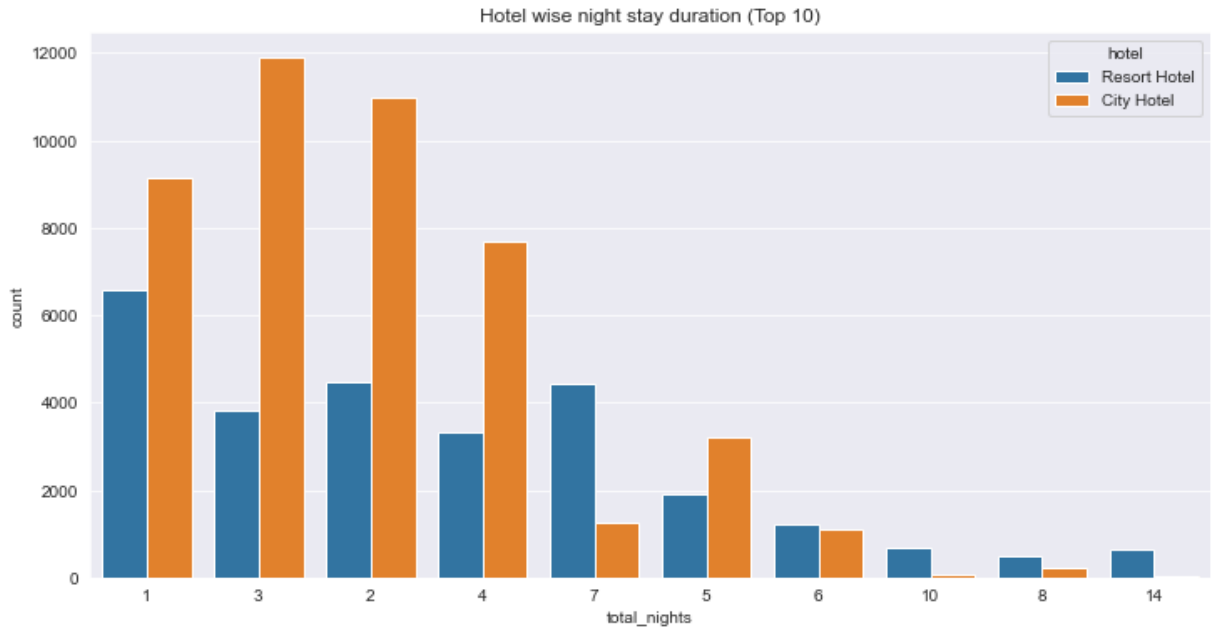
```
B:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
```

```
lowing variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword wil
l result in an error or misinterpretation.
  warnings.warn(
```



In [33]:
```
data_not_canceled.loc[:,'total_nights'] = data_not_canceled['stays_in_weekend_nights

fig, ax = plt.subplots(figsize=(12,6))
ax.set_xlabel('Number of Nights')
ax.set_ylabel('Number of Nights')
ax.set_title('Hotel wise night stay duration (Top 10)')
sns.countplot(x='total_nights', hue='hotel', data=data_not_canceled,
              order = data_not_canceled.total_nights.value_counts().iloc[:10].index,
```

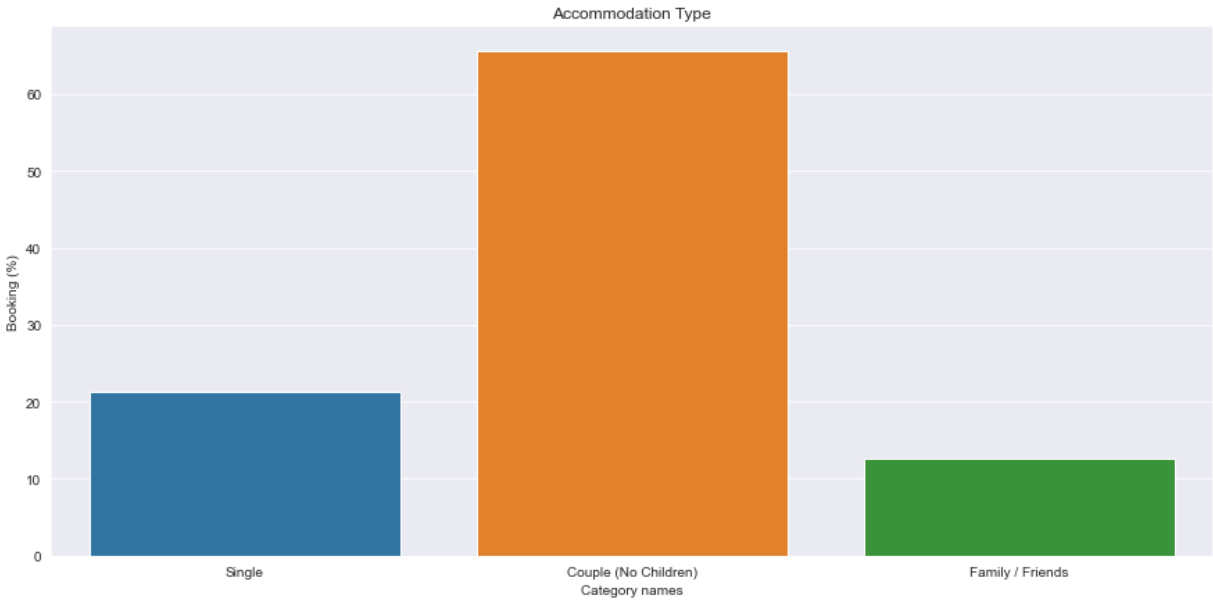
```
B:\Anaconda\lib\site-packages\pandas\core\indexing.py:1596: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[key] = _infer_fill_value(value)
B:\Anaconda\lib\site-packages\pandas\core\indexing.py:1745: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(ilocs[0], value)
```



7.Which was the most booked accommodation type (Single, Couple, Family)?

In [34]:
```
## Select single, couple, multiple adults and family
single   = data_not_canceled[(data_not_canceled.adults==1) & (data_not_canceled.chil
couple   = data_not_canceled[(data_not_canceled.adults==2) & (data_not_canceled.chil
#n_adults = data_not_canceled[(data_not_canceled.adults>2) & (data_not_canceled.chil
family   = data_not_canceled[data_not_canceled.adults + data_not_canceled.children +

## Make the list of Category names, and their total percentage
names = ['Single', 'Couple (No Children)', 'Family / Friends']
count = [single.shape[0],couple.shape[0], family.shape[0]]
count_percent = [x/data_not_canceled.shape[0]*100 for x in count]

## Draw the curve
display(names,count_percent,  label_x='Category names', label_y='Booking (%)', title
```

```
B:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the fol
lowing variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword wil
l result in an error or misinterpretation.
  warnings.warn(
```



In [ ]:

Feature Selection and Feature Engineering

In [35]:
```
### Recopy the dataframe
data1 = data.copy()
```

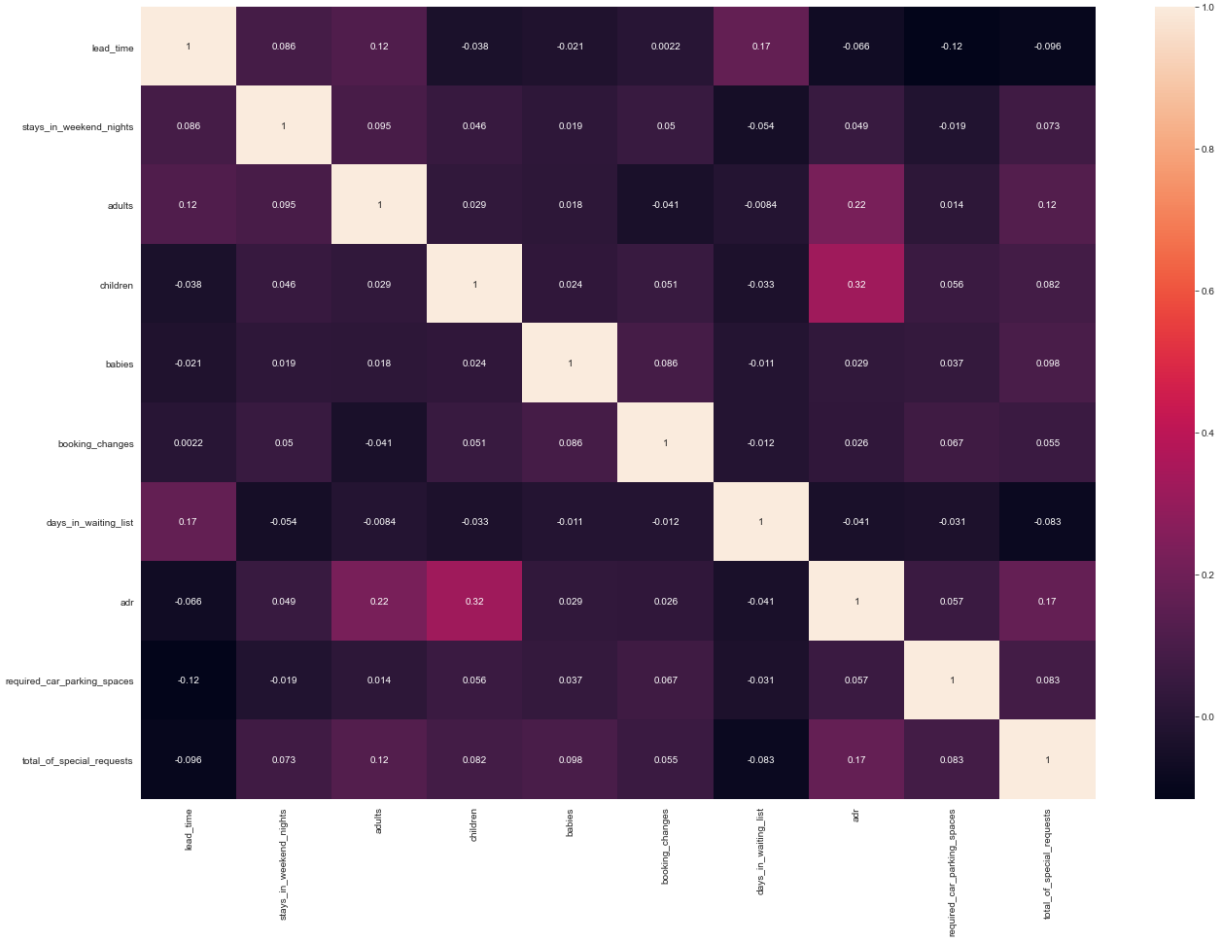
In [36]:
```
## Create a new column which contain 1 if guest received the same room which was res
data1['Room'] = 0
data1.loc[ data1['reserved_room_type'] == data1['assigned_room_type'] , 'Room'] = 1


## Make the new column which contain 1 if the guest has cancelled more booking in th
## than the number of booking, otherwise 0

data1['not_cancelled'] = 0
data1.loc[ data1['previous_cancellations'] > data1['previous_bookings_not_canceled']
```

In [ ]:

Pour regarder la corrélation entre les variables, on doit sélectionner les variables quantitatives.

In [37]:
```
list_quanti1=['lead_time', 'stays_in_weekend_nights', 'adults', 'children', 'babies'
```

In [38]:
```
## Plot the heatmap to see correlation with columns
fig, ax = plt.subplots(figsize=(22,15))
sns.heatmap(data1[list_quanti1].corr(), annot=True, ax=ax);
```



On remarque que dans l'ensemble la corrélation est très faible.

In [39]:
```
data1
```

Out[39]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_numb

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_numb
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	

119210 rows × 34 columns

In [ ]:

In [40]:
```
data1.dtypes
```

Out[40]:
```
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                         int64
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                            int64
company                          int64
days_in_waiting_list             int64
```

```
customer_type                   object
adr                              int64
required_car_parking_spaces      int64
total_of_special_requests        int64
reservation_status              object
reservation_status_date         object
Room                             int64
not_cancelled                    int64
dtype: object
```

1. Converting Categorical variables to Numerical

In [41]:
```
def transform(dataframe):


    ## Import LabelEncoder from sklearn
    from sklearn.preprocessing import LabelEncoder

    le = LabelEncoder()


    ## Select all categorcial features
    categorical_features = list(dataframe.columns[dataframe.dtypes == object])


    ## Apply Label Encoding on all categorical features
    return dataframe[categorical_features].apply(lambda x: le.fit_transform(x.astype

data2 = transform(data1)
```

In [42]:
```
data2.columns
```

Out[42]:
```
Index(['hotel', 'arrival_date_month', 'meal', 'country', 'market_segment',
       'distribution_channel', 'reserved_room_type', 'assigned_room_type',
       'deposit_type', 'customer_type', 'reservation_status',
       'reservation_status_date'],
      dtype='object')
```

Donc après conversion des variables catégorielles en numériques de notre jeu de données. Et comme le résultat de la transformation n'a pas été directement stocké dans notre jeu de données initial, nous allons devoir supprimer les colonnes non converties de notre jeu de données pour les remplacer par celles qui ont été convertit. Et pour cela nous avons fait appel à la méthode drop() de python pour supprimer toutes ces colonnes catégorielles

In [43]:
```
data1 = data1.drop(['hotel', 'arrival_date_month', 'meal', 'country', 'market_segmen
       'distribution_channel', 'reserved_room_type', 'assigned_room_type',
       'deposit_type', 'customer_type', 'reservation_status',
       'reservation_status_date'], axis=1)
```

Nous allons ensuite faire appel à la méthode join() de python pour concaténer notre jeu de données initial sans les colonnes catégorielles avec le résultat de notre conversion qui a été stocké dans la variable data2.

In [44]:
```
# df = df_train.join(one_hot)

df_ml = data1.join(data2)
df_ml
```

Out[44]:

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_mon
0	0	342	2015	27	
1	0	737	2015	27	
2	0	7	2015	27	
3	0	13	2015	27	

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_mon
4	0	14	2015	27	
...	...	...	...	...	
119385	0	23	2017	35	
119386	0	102	2017	35	
119387	0	34	2017	35	
119388	0	109	2017	35	
119389	0	205	2017	35	

119210 rows × 34 columns

In [45]:
```
df_ml.dtypes
```

Out[45]:
```
is_canceled                     int64
lead_time                       int64
arrival_date_year               int64
arrival_date_week_number        int64
arrival_date_day_of_month       int64
stays_in_weekend_nights         int64
stays_in_week_nights            int64
adults                          int64
children                        int64
babies                          int64
is_repeated_guest               int64
previous_cancellations          int64
previous_bookings_not_canceled  int64
booking_changes                 int64
agent                           int64
company                         int64
days_in_waiting_list            int64
adr                             int64
required_car_parking_spaces     int64
total_of_special_requests       int64
Room                            int64
not_cancelled                   int64
hotel                           int32
arrival_date_month              int32
meal                            int32
country                         int32
market_segment                  int32
distribution_channel            int32
reserved_room_type              int32
assigned_room_type              int32
deposit_type                    int32
customer_type                   int32
reservation_status              int32
reservation_status_date         int32
dtype: object
```

Modeling

CHoix des variables à choisir pour notre modèle

Les variables comme: 'arrival_date_year', 'arrival_date_month', 'arrival_date_week_number', 'arrival_date_day_of_month', 'stays_in_weekend_nights', 'stays_in_week_nights', 'reservation_status', 'reservation_status_date' ne seront pas retenus dans notre modèle du fait qu'elles sont très corrélées avec la variable que nous voulons prédire car il est évident qu'un client qui annule sa réservation n'aura pas de date d'arrivée dans l'hôtel, ni de mois d'arrivée moins de jours d'arrivée dans l'hôtel. Par conséquent nous allons les

supprimer de notre modèle.Egalement les variables comme: 'meal', 'agent', 'company', 'distribution_channel' et 'market_segment' n'ont aucun lien aussi avec la variable que nous voulons prédire. En d'autres termes elle ne nous apporte aucune information concernant la variable que nous voulons prédire.

In [46]:
```
df_ml = df_ml.drop(['arrival_date_year', 'arrival_date_month', 'arrival_date_week_nu
                    'arrival_date_day_of_month','stays_in_weekend_nights', 'stays_in
                    'reservation_status', 'reservation_status_date'], axis=1)
```

In [47]:
```
df_ml = df_ml.drop(['meal', 'agent', 'company', 'distribution_channel', 'market_segm
```

In [48]:
```
df_ml.columns
```

Out[48]:
```
Index(['is_canceled', 'lead_time', 'adults', 'children', 'babies',
       'is_repeated_guest', 'previous_cancellations',
       'previous_bookings_not_canceled', 'booking_changes',
       'days_in_waiting_list', 'adr', 'required_car_parking_spaces',
       'total_of_special_requests', 'Room', 'not_cancelled', 'hotel',
       'country', 'reserved_room_type', 'assigned_room_type', 'deposit_type',
       'customer_type'],
      dtype='object')
```

Donc au finale nous allons retenir seulement les variables ci-après dans le cadre de la mise en place de notre modèle: 'is_canceled', 'lead_time', 'adults', 'children', 'babies', 'is_repeated_guest', 'previous_cancellations', 'previous_bookings_not_canceled', 'booking_changes','days_in_waiting_list', 'adr', 'required_car_parking_spaces', 'total_of_special_requests', 'Room', 'not_cancelled', 'hotel', 'country', 'reserved_room_type', 'assigned_room_type', 'deposit_type', 'customer_type'.

In [49]:
```
list_col_X=['lead_time', 'adults', 'children', 'babies', 'is_repeated_guest', 'previ
       'previous_bookings_not_canceled', 'booking_changes','days_in_waiting_list', '
       'total_of_special_requests', 'Room', 'not_cancelled', 'hotel','country', 'res
       'deposit_type', 'customer_type']
Col_Y= 'is_canceled'
```

2.Train Test Split

In [50]:
```
def data_split(df, label):

    from sklearn.model_selection import train_test_split

    X = df.drop(label, axis=1)
    Y = df[label]

    x_train, x_test, y_train, y_test = train_test_split(X,Y,random_state=0)

    return x_train, x_test, y_train, y_test



x_train, x_test, y_train, y_test = data_split(df_ml, 'is_canceled')
```

In [51]:
```
df_ml
```

Out[51]:

	is_canceled	lead_time	adults	children	babies	is_repeated_guest	previous_cancellations	p
0	0	342	2	0	0	0	0	
1	0	737	2	0	0	0	0	
2	0	7	1	0	0	0	0	
3	0	13	1	0	0	0	0	
4	0	14	2	0	0	0	0	

	is_canceled	lead_time	adults	children	babies	is_repeated_guest	previous_cancellations	p
...	...	...	...	...	...	...	...	
119385	0	23	2	0	0	0	0	
119386	0	102	3	0	0	0	0	
119387	0	34	2	0	0	0	0	
119388	0	109	2	0	0	0	0	
119389	0	205	2	0	0	0	0	

119210 rows × 21 columns

In [52]:
```
x_train.shape,x_test.shape
```

Out[52]:
```
((89407, 20), (29803, 20))
```

In [53]:
```
y_train.mean(), y_test.mean()
```

Out[53]:
```
(0.3693223125706041, 0.37509646679864445)
```

Machine Learning

Faire un model simple avec Logistic Regression

Model 1: Logistic Regression

In [54]:
```
# importer un modèle
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=4000)
```

In [55]:
```
model.fit(X = x_train, y= y_train)
```

Out[55]:
```
LogisticRegression(max_iter=4000)
```

In [56]:
```
model.predict(x_train)
```

Out[56]:
```
array([0, 0, 0, ..., 0, 0, 1], dtype=int64)
```

In [57]:
```
md=model.predict_proba(x_train)[:,1]
md.shape
```

Out[57]:
```
(89407,)
```

In [58]:
```
df_viz = x_train.copy()
```

In [59]:
```
# Ajouter la prediction à al table
df_viz["prediction"] = model.predict_proba(x_train)[:,1]
```

In [60]:
```
df_viz
```

Out[60]:

	lead_time	adults	children	babies	is_repeated_guest	previous_cancellations	previous_book
10128	14	2	0	0	0	0	

	lead_time	adults	children	babies	is_repeated_guest	previous_cancellations	previous_book
20083	0	2	0	0	0	0	
13694	90	2	0	0	0	0	
3112	105	2	0	0	0	0	
62685	74	2	0	0	0	0	
...	...	...	...	...	...	...	
45912	84	1	0	0	0	0	
118132	87	2	0	0	0	0	
42632	11	2	1	0	0	0	
43588	13	1	0	0	0	0	
68311	164	1	0	0	0	0	

89407 rows × 21 columns

Evaluation modèle

In [61]:
```
df_pred = x_test.copy()
df_pred["is_canceled"]= y_test
```

In [62]:
```
# x_test
```

In [63]:
```
# ajout de la prediction à la table
df_pred["pred"]= model.predict(x_test)
df_pred
```

Out[63]:

	lead_time	adults	children	babies	is_repeated_guest	previous_cancellations	previous_book
44440	97	1	0	0	0	0	
118337	54	2	0	0	0	0	
43786	86	1	0	0	0	0	
116566	22	1	0	0	0	0	
58187	165	3	0	0	0	0	
...	...	...	...	...	...	...	
20008	1	1	0	0	0	0	
64570	52	1	0	0	0	0	
37718	273	2	0	0	0	0	
6629	167	2	0	0	0	0	
92732	177	2	0	0	0	0	

29803 rows × 22 columns

In [64]:
```
# ajout de la prediction de la proba à la table
df_pred["pred_proba"]= model.predict_proba(x_test)[:,1]
```

```
df_pred.head(15)
```

Out[64]:

	lead_time	adults	children	babies	is_repeated_guest	previous_cancellations	previous_book
44440	97	1	0	0	0	0	
118337	54	2	0	0	0	0	
43786	86	1	0	0	0	0	
116566	22	1	0	0	0	0	
58187	165	3	0	0	0	0	
116511	168	2	0	0	0	0	
117624	146	2	0	0	0	0	
89527	7	2	0	0	0	0	
115842	114	3	1	0	0	0	
98024	238	2	0	0	0	0	
48091	69	2	0	0	0	0	
111669	84	3	0	0	0	0	
17383	216	2	0	0	0	0	
34766	37	2	0	0	0	0	
62487	38	1	0	0	0	0	

In [65]:

```
df_pred[["is_canceled","pred","pred_proba"]].head(20)
```

Out[65]:

	is_canceled	pred	pred_proba
44440	1	1	0.974959
118337	0	0	0.124025
43786	0	0	0.167264
116566	0	0	0.274736
58187	1	1	0.688659
116511	0	0	0.477904
117624	0	0	0.307460
89527	0	0	0.216584
115842	0	1	0.597134
98024	0	0	0.213159
48091	1	0	0.088705
111669	0	1	0.587030
17383	0	0	0.002150
34766	0	0	0.042288
62487	1	1	0.956214
64774	1	1	0.993964

	is_canceled	pred	pred_proba
6478	1	1	0.985632
80674	0	0	0.318826
10324	1	0	0.411491
109804	0	0	0.242583

In [66]:
```
df_pred["error"]= np.abs(df_pred["is_canceled"] - df_pred["pred"])
df_pred["error"].sum()
```

Out[66]: 6499

In [67]:
```
df_pred.shape
```

Out[67]: (29803, 24)

Sur les 29803 observations gardées pour le test, on predit mal 6499 indivudus.

In [68]:
```
# proba d'erreur qu'on commet
df_pred["error"].sum() / df_pred.shape[0]
```

Out[68]: 0.21806529544005637

In [69]:
```
# proba des individus qu'on predit bien
1 - df_pred["error"].sum() / df_pred.shape[0]
```

Out[69]: 0.7819347045599436

Calcule AUC modèle

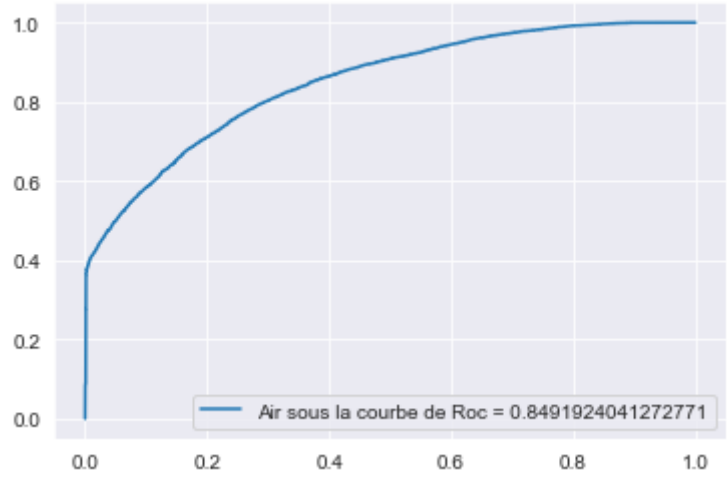
In [70]:
```
from sklearn.metrics import roc_auc_score
```

In [71]:
```
# calcule de l'air sur la courbe de ROC
round(roc_auc_score(df_pred[Col_Y], df_pred["pred_proba"]),3)
```

Out[71]: 0.849

Tracée de la courbe de Roc

In [77]:
```
from sklearn import metrics
y_pred_prob = df_pred["pred_proba"]
fpr,tpr , _ = metrics.roc_curve(y_test,  y_pred_prob)
auc = metrics.roc_auc_score(y_test, y_pred_prob)
plt.plot(fpr, tpr, label="Air sous la courbe de Roc = "+str(auc))
plt.legend(loc=4)
plt.show()
```



cross validation pour évaluer le modèle

In [78]:
```
X= df_ml.copy()
```

In [79]:
```
from sklearn.model_selection import cross_val_score
```

In [83]:
```
score = cross_val_score(model, X[list_col_X], X[Col_Y], cv=5, scoring="roc_auc")
score
```

Out[83]:
```
array([0.81935042, 0.75880495, 0.83260714, 0.80929998, 0.90016531])
```

In [84]:
```
np.mean(score)
```

Out[84]:
```
0.8240455603757428
```

In [ ]:
```

```

Make the best model possible

2. Model (Decision Tree)

In [85]:
```
def train(x_train, y_train):
    from sklearn.tree import DecisionTreeClassifier

    clf = DecisionTreeClassifier(random_state=20,max_depth=10)
    clf.fit(x_train,y_train)

    return clf

model1 = train(x_train, y_train)
```

In [86]:
```
def Score(model1,x_train,y_train,x_test,y_test):
    train_score = model1.score(x_train,y_train)
    test_score = model1.score(x_test,y_test)

    print("========================================")
    print(f'Training Accuracy of our model is: {train_score}')
    print(f'Test Accuracy of our model is: {test_score}')
    print("========================================")


Score(model1,x_train,y_train,x_test,y_test)
```

```
========================================
```

```
Training Accuracy of our model is: 0.8190969387184449
Test Accuracy of our model is: 0.8181726671811562
========================================
```

Evaluation cross validation

In [87]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model1, X[list_col_X], X[Col_Y], cv=20, scoring='roc_auc')
np.mean(scores)
```

Out[87]:
```
0.8719713582291853
```

In [88]:
```
scores
```

Out[88]:
```
array([0.88288456, 0.89893778, 0.84332799, 0.87015637, 0.85569839,
       0.88265699, 0.81818351, 0.87504852, 0.89264136, 0.84984252,
       0.90457616, 0.86061581, 0.79399252, 0.84943403, 0.89116513,
       0.8927895 , 0.85641544, 0.83657792, 0.99287831, 0.89160434])
```

In [89]:
```
#permet de determiner la variable la plus importante pour expliquer la variable cibl
model1.feature_importances_
```

Out[89]:
```
array([6.78982919e-02, 5.88531249e-03, 1.41601055e-03, 0.00000000e+00,
       5.26641563e-04, 3.96656957e-04, 3.28886827e-03, 1.42689310e-02,
       0.00000000e+00, 4.12739925e-02, 4.83278544e-02, 4.19309539e-02,
       6.29195025e-02, 8.30273669e-02, 9.36256675e-03, 5.30910366e-02,
       6.86400658e-04, 1.21256897e-03, 4.84926415e-01, 7.95606290e-02])
```

feature importance

In [90]:
```
pd.DataFrame([model1.feature_importances_, list_col_X]).T.sort_values(0,ascending=0)
```

Out[90]:

	0	1
18	0.484926	deposit_type
13	0.0830274	not_cancelled
19	0.0795606	customer_type
0	0.0678983	lead_time
12	0.0629195	Room
15	0.053091	country
10	0.0483279	required_car_parking_spaces
11	0.041931	total_of_special_requests
9	0.041274	adr
7	0.0142689	booking_changes
14	0.00936257	hotel
1	0.00588531	adults
6	0.00328887	previous_bookings_not_canceled
2	0.00141601	children
17	0.00121257	assigned_room_type
16	0.000686401	reserved_room_type
4	0.000526642	is_repeated_guest
5	0.000396657	previous_cancellations

	0	1
8	0	days_in_waiting_list
3	0	babies

Nous avons la composante 'deposit_type' qui représente à elle seule près de la moitié des informations disponibles (48,5%) des informations de notre jeu de données. Elle explique au mieux notre variable de prédiction. Cela peut s'expliquer par le fait qu'un individu qui n'a pas effectué un dépôt de garantie serait plus susceptible d'annuler sa réservation qu'un autre individu qui l'aurait fait sachant qu'il ne serait pas remboursé. Les variables 'days_in_waiting_list' et 'babies' ne représentent que 0% des informations de notre jeu de données, elles seront par conséquent enlevé de notre modèle.

In [91]:
```
X= X.drop(['days_in_waiting_list', 'babies'], axis=1)
```

In [92]:
```
list_col_X = ['lead_time', 'adults', 'children', 'is_repeated_guest',
       'previous_cancellations', 'previous_bookings_not_canceled',
       'booking_changes', 'adr', 'required_car_parking_spaces',
       'total_of_special_requests', 'Room', 'not_cancelled', 'hotel',
       'country', 'reserved_room_type', 'assigned_room_type', 'deposit_type',
       'customer_type']
Col_Y= 'is_canceled'
```

Nous allons appeler notre fonction data_split que nous avons définit plus haut

In [93]:
```
x_train, x_test, y_train, y_test = data_split(X, 'is_canceled')
```

In [94]:
```
model1 = train(x_train, y_train)
```

one time evaluation

In [96]:
```
X_predi = x_test.copy()
X_predi["is_canceled"] = y_test
```

In [97]:
```
X_predi["pred"] = model1.predict(x_test)
```

In [98]:
```
X_pred["pred_proba"] = model1.predict_proba(x_test)[:,1]
```

In [99]:
```
X_pred["error"] = np.abs(df_pred["is_canceled"] - df_pred["pred"])
```

In [100…
```
X_pred.shape
```

Out[100…
```
(29803, 20)
```

In [101…
```
X_pred[["error"]].sum()
```

Out[101…
```
error    6499
dtype: int64
```

Sur les 29803 observations gardées pour le test, on a predi mal 6499 individus

In [102…
```
# proba d'erreur qu'on commet
X_pred["error"].sum() / X_pred.shape[0]
```

Out[102…
```
0.21806529544005637
```

In [103…
```
1 - X_pred["error"].sum() / X_pred.shape[0]
```

Out[103…
```
0.7819347045599436
```

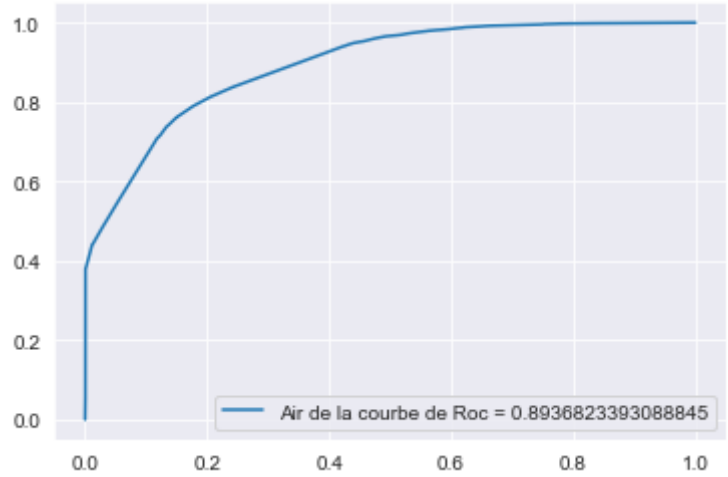
Calcul AUC de notre modèle

In [104…
```
from sklearn.metrics import roc_auc_score
round(roc_auc_score(df_pred["is_canceled"], df_pred["pred_proba"]),4)
```

Out[104… 0.8492

Tracée de la courbe de Roc de notre modèle

In [105…
```
from sklearn import metrics
y_pred_prob = X_pred["pred_proba"]
fpr,tpr , _ = metrics.roc_curve(y_test,  y_pred_prob)
auc = metrics.roc_auc_score(y_test, y_pred_prob)
plt.plot(fpr, tpr, label="Air de la courbe de Roc = "+str(auc))
plt.legend(loc=4)
plt.show()
```



Visualiser l'overfitting d'un arbre de décision

In [108…
```
from sklearn.tree import DecisionTreeClassifier
list_val_train = []
list_val_test = []

for max_depth in range(1,30,1):

    model1 = DecisionTreeClassifier(random_state=42,max_depth=max_depth)
    model1.fit(X = x_train, y= y_train)

    ### auc sur le train
    df_predict = x_train.copy()
    df_predict["is_canceled"] = y_train
    df_predict["pred_proba"] = model1.predict_proba(x_train)[:,1]
    auc_train = roc_auc_score(df_predict[Col_Y], df_predict["pred_proba"])


    #auc_test avec cross validation
    auc_test = np.mean(cross_val_score(model1, X[list_col_X], X[Col_Y], cv=10, scori

    list_val_train.append(auc_train)

    list_val_test.append(auc_test)

    print("MODEL", max_depth, "train:",np.round(auc_train,3), " test by cross valid:
```

```
MODEL 1 train: 0.663  test by cross valid: 0.663
MODEL 2 train: 0.688  test by cross valid: 0.672
MODEL 3 train: 0.738  test by cross valid: 0.723
MODEL 4 train: 0.78  test by cross valid: 0.766
```

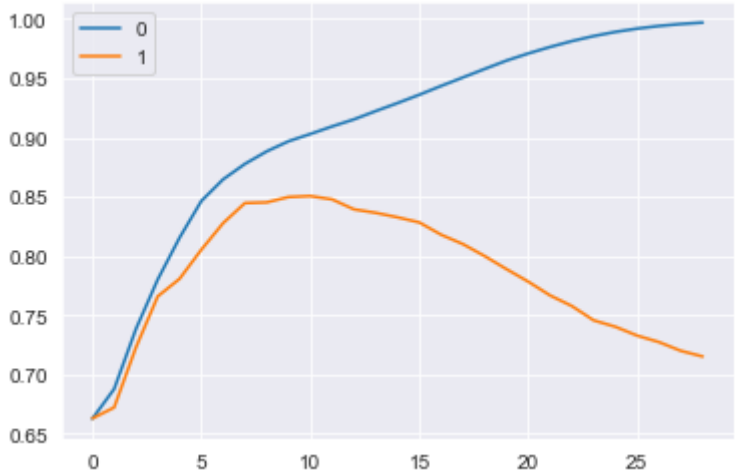
```
MODEL 5 train: 0.815  test by cross valid: 0.781
MODEL 6 train: 0.846  test by cross valid: 0.805
MODEL 7 train: 0.865  test by cross valid: 0.828
MODEL 8 train: 0.878  test by cross valid: 0.845
MODEL 9 train: 0.888  test by cross valid: 0.845
MODEL 10 train: 0.897  test by cross valid: 0.85
MODEL 11 train: 0.903  test by cross valid: 0.85
MODEL 12 train: 0.909  test by cross valid: 0.848
MODEL 13 train: 0.915  test by cross valid: 0.839
MODEL 14 train: 0.922  test by cross valid: 0.836
MODEL 15 train: 0.929  test by cross valid: 0.833
MODEL 16 train: 0.936  test by cross valid: 0.828
MODEL 17 train: 0.943  test by cross valid: 0.818
MODEL 18 train: 0.95  test by cross valid: 0.81
MODEL 19 train: 0.958  test by cross valid: 0.8
MODEL 20 train: 0.965  test by cross valid: 0.789
MODEL 21 train: 0.971  test by cross valid: 0.778
MODEL 22 train: 0.976  test by cross valid: 0.767
MODEL 23 train: 0.981  test by cross valid: 0.758
MODEL 24 train: 0.985  test by cross valid: 0.746
MODEL 25 train: 0.989  test by cross valid: 0.74
MODEL 26 train: 0.992  test by cross valid: 0.733
MODEL 27 train: 0.994  test by cross valid: 0.727
MODEL 28 train: 0.996  test by cross valid: 0.72
MODEL 29 train: 0.997  test by cross valid: 0.715
```

In [ ]:

In [ ]:

In [109…
```
pd.DataFrame([list_val_train,list_val_test]).T.plot()
plt.legend()
```

Out[109… `<matplotlib.legend.Legend at 0x14b941016d0>`



le meilleur arbre de décision est un arbre de max_depth = 10

In [110…
```
model1 = DecisionTreeClassifier(random_state=20,max_depth=10)
model1.fit(X = x_train, y= y_train)
```

Out[110… `DecisionTreeClassifier(max_depth=10, random_state=20)`

In [112…
```
### juste un test  pour voir l'AUC sur le train
df_predi = x_train.copy()
df_predi["is_canceled"] = y_train
df_predi["pred_proba"] = model1.predict_proba(x_train)[:,1]
roc_auc_score(df_pred[Col_Y], df_pred["pred_proba"])
```

Out[112… `0.8491924041272771`

In [113...
```
df_predi["pred"] = model1.predict(x_train)
```

In [114...
```
df_predi["error"] = np.abs(df_predi["is_canceled"] - df_predi["pred"])
```

In [115...
```
df_predi["error"].sum()
```

Out[115... 16175

In [116...
```
df_predi["error"].sum() / df_predi.shape[0]
```

Out[116... 0.1809142460881139

In [117...
```
#### accuracy
1 - df_predi["error"].sum() / df_predi.shape[0]
```

Out[117... 0.8190857539118861

In [118...
```
df_predi[["is_canceled","pred","pred_proba","error"]]
```

Out[118...

	is_canceled	pred	pred_proba	error
10128	1	1	0.741463	0
20083	0	0	0.018158	0
13694	1	0	0.266899	1
3112	0	0	0.024744	0
62685	1	1	0.607915	0
...	...	...	...	...
45912	0	0	0.172348	0
118132	0	0	0.266899	0
42632	1	0	0.391761	1
43588	0	0	0.391761	0
68311	1	1	0.999621	0

89407 rows × 4 columns

In [ ]:
```
```

Model 3 Randomforest

In [119...
```
from sklearn.ensemble import RandomForestClassifier
```

In [122...
```
model3=RandomForestClassifier(random_state=20,max_depth=10)
model3.fit(X = x_train, y= y_train)
```

Out[122... RandomForestClassifier(max_depth=10, random_state=20)

In [160...
```
Score(model3,x_train,y_train,x_test,y_test)
```

```
========================================
Training Accuracy of our model is: 0.8181238605478318
Test Accuracy of our model is: 0.8175351474683756
========================================
```

feature importance

In [123…
```
#permet de determiner la variable la plus import
model3.feature_importances_
```

Out[123…
```
array([0.12702411, 0.00718041, 0.00282896, 0.00451263, 0.04580742,
       0.00865947, 0.02706952, 0.03586359, 0.04332948, 0.06119216,
       0.07530633, 0.05440452, 0.00815482, 0.13988279, 0.00561758,
       0.01470933, 0.28757847, 0.0508784 ])
```

Expliquez les variables qui semblent importantes pour prédire un risque d'annulation de réservation.

In [124…
```
pd.DataFrame([model3.feature_importances_, list_col_X]).T.sort_values(0,ascending=0)
```

Out[124…

	0	1
16	0.287578	deposit_type
13	0.139883	country
0	0.127024	lead_time
10	0.0753063	Room
9	0.0611922	total_of_special_requests
11	0.0544045	not_cancelled
17	0.0508784	customer_type
4	0.0458074	previous_cancellations
8	0.0433295	required_car_parking_spaces
7	0.0358636	adr
6	0.0270695	booking_changes
15	0.0147093	assigned_room_type
5	0.00865947	previous_bookings_not_canceled
12	0.00815482	hotel
1	0.00718041	adults
14	0.00561758	reserved_room_type
3	0.00451263	is_repeated_guest
2	0.00282896	children

Les variables qui paraissent importantes pour prédire un risque d'annulation de réservation sont: 1. deposit_type et représente 28,75% d'informations disponibles. Son fort taux de pourcentage s'explique par le faite que c'est la variable qui détermine si le client a fait un dépôt de garanti lors de sa réservation. Donc il est bien évident qu'un client qui a fait un dépôt de garanti ne va pas vouloir par la suite procéder à son annulation sachant qu'il pourrait perdre son dépôt de garanti. 2. country et représente 13,98% d'informations disponibles. 3. lead_time et représente 12,70% d'informations disponibles. 4. Room et représente 7,53% d'informations disponibles. 5. total_of_special_requests et représente 6,11% d'informations disponibles. 6. not_cancelled et représente 5,44% d'informations disponibles. 7. customer_type et représente 5,08% d'informations disponibles. 8. previous_cancellations et représente 4,58% d'informations disponibles. 9. required_car_parking_spaces et représente 4,33% d'informations disponibles. 10. adr et représente 3,58% d'informations disponibles. 11. booking_changes et représente 2,71% d'informations disponibles. 12. assigned_room_type et représente 1,47% d'informations disponibles. De ce fait toute variables qui ne nous apporte pas plus d'1% d'information sera

supprimé de notre modèle. Donc les variables comme: previous_bookings_not_canceled, hotel, adults, reserved_room_type, is_repeated_guest et children seront carrément supprimées de notre modèle

In [125…
```
X=X.drop(['previous_bookings_not_canceled', 'hotel', 'adults', 'reserved_room_type',
```

In [126…
```
list_col_X = ['lead_time', 'previous_cancellations', 'booking_changes', 'adr', 'requ
       'total_of_special_requests', 'Room', 'not_cancelled', 'country', 'assigned_ro
Col_Y= 'is_canceled'

x_train, x_test, y_train, y_test = data_split(X, 'is_canceled')
model3 = train(x_train, y_train)
```

Evaluation du modèle

In [129…
```
df_pre = x_test.copy()
df_pre["is_canceled"]= y_test
```

In [130…
```
df_pre["pred_proba3"]= model3.predict_proba(x_test)[:,1]
df_pre
```

Out[130…

	lead_time	previous_cancellations	booking_changes	adr	required_car_parking_spaces	total_
44440	97	0	0	140	0	
118337	54	0	1	152	0	
43786	86	0	1	87	0	
116566	22	0	0	134	0	
58187	165	0	0	151	0	
...	...	...	...	...	...	
20008	1	0	0	31	1	
64570	52	0	0	80	0	
37718	273	0	0	56	0	
6629	167	0	0	54	0	
92732	177	0	0	80	0	

29803 rows × 14 columns

In [134…
```
# ajout de la prediction à la table
df_pre["pred"]= model3.predict(x_test)
df_pre
```

Out[134…

	lead_time	previous_cancellations	booking_changes	adr	required_car_parking_spaces	total_
44440	97	0	0	140	0	
118337	54	0	1	152	0	
43786	86	0	1	87	0	
116566	22	0	0	134	0	
58187	165	0	0	151	0	
...	...	...	...	...	...	
20008	1	0	0	31	1	

	lead_time	previous_cancellations	booking_changes	adr	required_car_parking_spaces	total_
64570	52	0	0	80	0	
37718	273	0	0	56	0	
6629	167	0	0	54	0	
92732	177	0	0	80	0	

29803 rows × 15 columns

In [143…
```
df_pre["error"] = np.abs(df_pre["is_canceled"] - df_pre["pred"])
```

In [150…
```
df_pre.shape
```

Out[150… (29803, 16)

In [144…
```
df_pre["error"].sum()
```

Out[144… 5438

In [148…
```
# Probabilité d'erreur

df_pre["error"].sum() / df_pre.shape[0]
```

Out[148… 0.18246485253162434

In [149…
```
#### Probabilité de succès

1 - df_pre["error"].sum() / df_pre.shape[0]
```

Out[149… 0.8175351474683756

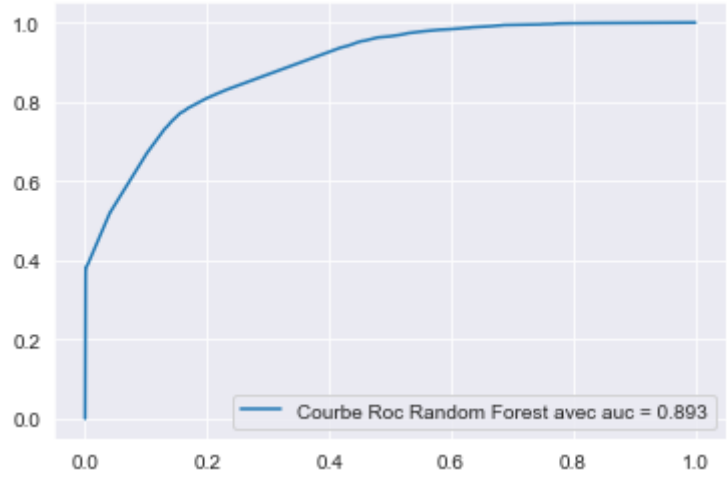
AUC pour notre modèle

In [135…
```
roc_auc_score(df_pre[Col_Y], df_pre["pred_proba3"])
```

Out[135… 0.8932573778338067

Courbe de Roc pour notre Modèle RandomForest

In [136…
```
from sklearn import metrics
y_pred_prob = df_pre["pred_proba3"]
fpr,tpr , _ = metrics.roc_curve(y_test,  y_pred_prob)
auc = round(metrics.roc_auc_score(y_test, y_pred_prob),3)
plt.plot(fpr, tpr, label="Courbe Roc Random Forest avec auc = "+str(auc))
plt.legend(loc=4)
plt.show()
```



In [139…
```
df_pre[["is_canceled","pred_proba3","pred"]]
```

Out[139…

	is_canceled	pred_proba3	pred
44440	1	1.000000	1
118337	0	0.266899	0
43786	0	0.024739	0
116566	0	0.592004	1
58187	1	0.592004	1
...	...	...	...
20008	0	0.000000	0
64570	1	1.000000	1
37718	0	0.164275	0
6629	1	0.666398	1
92732	0	0.339096	0

29803 rows × 3 columns

Cross validation pour évaluer le modèle 3 Random Forest

In [158…
```
from sklearn.model_selection import cross_val_score
score=cross_val_score(model3, X[list_col_X], X[Col_Y], cv=60, scoring="roc_auc")
score
```

Out[158…
```
array([0.84586407, 0.92687535, 0.90712864, 0.88675201, 0.862693  ,
       0.95280623, 0.94126166, 0.8681495 , 0.80298696, 0.8711281 ,
       0.8568341 , 0.86552836, 0.77624098, 0.86223175, 0.97479186,
       0.79396472, 0.94251506, 0.97462958, 0.81449769, 0.84128195,
       0.91437449, 0.90430502, 0.89378182, 0.89956906, 0.92396906,
       0.90449607, 0.8673194 , 0.78102578, 0.88606024, 0.90187843,
       0.87904369, 0.89138073, 0.92452429, 0.88595658, 0.91894111,
       0.7711251 , 0.77304912, 0.86795007, 0.82205482, 0.85431045,
       0.88888033, 0.86923691, 0.87623501, 0.91766893, 0.89070882,
       0.89265563, 0.88150773, 0.92179213, 0.91569932, 0.86781275,
       0.83655326, 0.75024565, 0.79223967, 1.        , 1.        ,
       0.99864783, 0.98124674, 0.9324125 , 0.9689413 , 0.92538424])
```

In [159…
```
#score moyen
np.mean(score)
```

Out[159… 0.8856857610768306

In [ ]:

Le meilleur modèle est celui avec le random forest avec un AUC=89,32%.

In [ ]: