Rapport : Projet Data Mining -DA 2021

Réalisé par :

RIALI Mouad

ADDI Kamal

Encadré par :

Pr. ELASRI Ikram

II. Prétraitement des données :

1. pré-traitement des données manquantes

La première étape à faire pour traiter et modéliser une base de données, c'est le traitement et la neutralisation des valeurs manquantes au sein d'une BD, celui-là se fait par deux méthodes :

La première c'est l'élimination des lignes/ colonnes qui ont des valeurs manquantes, alors que la deuxième méthode de remplir ces valeurs-là et les donner certaines valeurs soit numérique : la moyenne, la médiane…etc., soit catégorique ce qui est le cas pour notre base de données, En effet, Après l'importation des bibliothèques suivantes :

```
In [1]: #importing libraries
        import pandas as pd
        import numpy as np
        import matplotlib
        import matplotlib.pyplot as plt
        import seaborn as sns
        import statsmodels.api as sm
        %matplotlib inline
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LinearRegression
        from sklearn.feature_selection import RFE
        from sklearn.linear_model import RidgeCV, LassoCV, Ridge, Lasso
```

Il est temps d'importer notre dataset, et visualiser ses 5 premières lignes et quelques informations qui la décrivent pour bien la comprendre et déterminer l'étape suivante :

```
In [2]: df=pd.read_csv('hotelsBookingDemand.csv')
        df.head()
```

Out[2]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	sta
0	Resort Hotel	0	342	2015	July	27	1	0	
1	Resort Hotel	0	737	2015	July	27	1	0	
2	Resort Hotel	0	7	2015	July	27	1	0	
3	Resort Hotel	0	13	2015	July	27	1	0	
4	Resort Hotel	0	14	2015	July	27	1	0	

5 rows × 32 columns

```
In [69]: df.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
```

D'où notre base de données se compose de 32 colonnes 119390 lignes.

Maintenant, après avoir compris la structure générale de notre dataset, on va commencer dans la phase du “prétraitement”, dans un premier lieu on va essayer d’isoler les valeurs manquantes.

Tout d’abord, il faut savoir quelles sont les variables concernées par ce problème, et à quel point elles sont touchées. Pour cela on va exécuter ces deux commandes :

```
df.isna().sum(axis=0) #nombre des valeurs manquantes pour chaque colonne

df.isna().sum(axis=0)/len(df) * 100 #pourcentage des valeurs manquantes pour chaque colonne
```

Et on va avoir par la suite :

In [4]:
```
df.isna().sum(axis=0)
```

Out[4]:
```
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
```

In [5]:
```
df.isna().sum(axis=0)/len(df) * 100
```

Out[5]:
```
hotel                           0.000000
is_canceled                     0.000000
lead_time                       0.000000
arrival_date_year               0.000000
arrival_date_month              0.000000
arrival_date_week_number        0.000000
arrival_date_day_of_month       0.000000
stays_in_weekend_nights         0.000000
stays_in_week_nights            0.000000
adults                          0.000000
children                        0.003350
babies                          0.000000
meal                            0.000000
country                         0.408744
market_segment                  0.000000
distribution_channel            0.000000
is_repeated_guest               0.000000
previous_cancellations          0.000000
previous_bookings_not_canceled  0.000000
reserved_room_type              0.000000
assigned_room_type              0.000000
booking_changes                 0.000000
deposit_type                    0.000000
agent                          13.686238
company                        94.306893
days_in_waiting_list            0.000000
customer_type                   0.000000
adr                             0.000000
required_car_parking_spaces     0.000000
total_of_special_requests       0.000000
reservation_status              0.000000
reservation_status_date         0.000000
dtype: float64
```

Selon les captures au dessus, on peut citer 4 colonnes qui contiennent des valeurs manquantes qui sont (par ordre descendant) :

- company 94.3 %
- agent 13.7 %
- country 0.4 %
- children 0.0034 %

On va se baser dans le choix des méthodes du traitement des VM sur l'importance de leurs proportions par rapport aux tailles des entières colonnes.

Du coup, on peut classifier les colonnes qui portent des valeurs manquantes en 2 classes :

- la première classe : il contient "country" et "children", leurs proportions sont très petites voire négligeables, alors il vaut mieux de les éliminer pour ne pas influencer - défavorablement - notre modèle.
- La deuxième classe : il se compose de "agent" et "company", les valeurs manquantes de ces dernières colonnes sont très nombreuses, alors on ne peut pas les supprimer tout simplement, en effet, dans le cas de variable "company" on a 94.3% des valeurs sont manquantes/ nulles, alors on ne peut pas éliminer tout ce nombre de lignes – à savoir 112593 - sinon, on n'aura rien à traiter, alors on va essayer de les remplacer par : 0 ou bien "NULL".

"country" & "children" :

Comme on a déjà mentionné on va éliminer les lignes qui portent des valeurs manquantes qui appartiennent aux "country" et "children" :

```
In [70]: #drop rows having missing values in country column

         df_drop = df.dropna(subset=['country','children'])
         df_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 118898 entries, 0 to 119389
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           118898 non-null  object
 1   is_canceled                     118898 non-null  int64
 2   lead_time                       118898 non-null  int64
 3   arrival_date_year               118898 non-null  int64
 4   arrival_date_month              118898 non-null  object
 5   arrival_date_week_number        118898 non-null  int64
 6   arrival_date_day_of_month       118898 non-null  int64
 7   stays_in_weekend_nights         118898 non-null  int64
 8   stays_in_week_nights            118898 non-null  int64
 9   adults                          118898 non-null  int64
 10  children                        118898 non-null  float64
 11  babies                          118898 non-null  int64
 12  meal                            118898 non-null  object
 13  country                         118898 non-null  object
 14  market_segment                  118898 non-null  object
```

“agent” & “company” :

Avant de commencer le code, il vaut mieux de comprendre pourquoi on va remplacer les valeurs manquantes au lieu de les supprimer, D’abord, c’est quoi la signification des données offertes par les colonnes : “agent” et “company” ?

Selon le document qui explique la base de donnees, on trouve :

- “agent” : l’ID de l’agence du voyage qui fait la réservation
- “company” : la société qui fait la réservation/ qui paie pour la réservation

Alors seules les réservations qui sont faites par une agence ou une société ont des valeurs non nulles au niveau des colonnes : “agent” & “company”. On déduit alors que le reste peut-être a fait des réservations directes, cela veut dire que les lignes contenant ces valeurs nulles ne sont en réalité que l’ensemble des personnes qui ont fait les réservations d’une manière directe sans avoir besoin d’une agence de voyages ou bien une société pour payer/ réserver des chambres de l’Hôtel.

Du coup, au lieu de supprimer ces réservations ou bien écarter ces deux colonnes, on va remplacer leurs valeurs manquantes par la valeur: “0”

```
In [72]: df_drop=df_drop.fillna(value={'agent':0, 'company':0})
```

ET voilà !!

```
In [73]: df_drop[['agent','company']]
```

Out[73]:

	agent	company
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	304.0	0.0
4	240.0	0.0
...	...	...
119385	394.0	0.0
119386	9.0	0.0
119387	9.0	0.0
119388	89.0	0.0
119389	9.0	0.0

118898 rows × 2 columns

2. pré-traitement des variables de temps

La base de données sujet de traitement contient en somme jusqu'à 5 variables de temps qui décrivent deux différentes phases, celles-ci sont :

- L'arrivée :
  - arrival_date_year : l'annee d'arrivee
  - arrival_date_month : le mois d'arrivee
  - arrival_date_day_of_month : le jour d'arrivee
  - arrival_date_week_number : la semaine d'arrivee
- La réservation :
  - reservation_status_date : la date de la mise a jour du statut de reservation.

Comme ces variables-là décrivent différents phénomènes, on va essayer de les traiter en première phase individuellement et après, on va conclure la relation qui relie chaque catégorie à l'autre, enfin puisque notre premier et notre dernier intérêt c'est de chercher l'histoire globale que raconte la dataset, on va introduire une additionnelle colonne qui va nous servir le plus en émergeant toutes les variables de temps existant dans la base de données.

Les variables d'arrivée:

On a 4 variables temporelles qui décrivent l'arrivée à l'hôtel, chacune d'elles présente une particularité bien précise, une décrit l'année, l'autre s'intéresse au mois, et ainsi de suite, certes on peut bénéficier de cette diversité pour s'assurer est ce qu'on a par exemple une relation entre le nombre des enfants, le type de chambre réservée et le mois d'arrivée...etc., mais pour notre cas cela va juste nous gêner surtout que la corrélation entre les différentes variables du temps - séparées – et la variable de annulation est faible. Alors on va tenter de les forger en une seule variable "arrival date" qu'est tout simplement la forme canonique de la date d'arrivée à l'hôtel.

La première chose à faire est définir une fonction qui prend 3 variables : "year", "month", "day" et qui retourne la date dans sa forme canonique.

```
In [10]: from datetime import datetime

         def getDatetime(year,month,day):
             datestring = day+"-"+month[:3]+"-"+year
             dt = datetime.strptime(datestring, '%d-%b-%Y')
             if dt.month > 9 :
                 return f'{dt.year}-{dt.month}-{dt.day}'
             elif dt.day>9 :
                 return f'{dt.year}-0{dt.month}-{dt.day}'
             else : return f'{dt.year}-0{dt.month}-0{dt.day}'
```

Exemple :

```
In [11]: print(getDatetime("2015","january","14"))

         2015-01-14
```

Puis, on va concaténer les colonnes de “year”, “month” et “day” dans une seule colonne de la forme suivante : “14-january-2015” pour qu’on puisse par la suite interpréter ses valeurs à l’aide de la fonction déjà définie “getDatetime” et engendrer notre colonne “arriaval_date

Pratiquement, on exécute la commande suivante:

```
df_date['arrival_date_str']= df_date["arrival_date_year"].astype(str)+"-"+
df_date["arrival_date_month"].astype(str)+"-"+df_date['arrival_date_day_of_month'].astype(str)
```

```
In [76]: df_date['arrival_date_str']= df_date["arrival_date_year"].astype(str)+"-"+ df_date["arrival_date_month"].astype(str)+

In [77]: df_date['arrival_date_str']

Out[77]: 0            2015-July-1
         1            2015-July-1
         2            2015-July-1
         3            2015-July-1
         4            2015-July-1
                         ...
         119385    2017-August-30
         119386    2017-August-31
         119387    2017-August-31
         119388    2017-August-31
         119389    2017-August-29
         Name: arrival_date_str, Length: 118898, dtype: object
```

Ensuite, on applique une fonction lambda sur les composantes de la colonne “arrival date str” :

```
df_date['arrival_date']= df_date['arrival_date_str'].apply(lambda x:
getDatetime(x.split('-')[0],x.split('-')[1],x.split('-')[2]))
```

```
In [14]: 'al_date']= df_date['arrival_date_str'].apply(lambda x: getDatetime(x.split('-')[0],x.split('-')[1],x.split('-')[2]))
```

```
<ipython-input-14-2d89cb0fa438>:1: 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/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_date['arrival_date']= df_date['arrival_date_str'].apply(lambda x: getDatetime(x.split('-')[0],x.split('-')[1],
x.split('-')[2]))
```

```
In [15]: df_date
```

Out[15]:

	arrival_date_year	arrival_date_month	arrival_date_day_of_month	arrival_date_str	arrival_date
0	2015	July	1	2015-July-1	2015-07-01
1	2015	July	1	2015-July-1	2015-07-01
2	2015	July	1	2015-July-1	2015-07-01
3	2015	July	1	2015-July-1	2015-07-01
4	2015	July	1	2015-July-1	2015-07-01
...	...	...	...	...	...
119385	2017	August	30	2017-August-30	2017-08-30
119386	2017	August	31	2017-August-31	2017-08-31
119387	2017	August	31	2017-August-31	2017-08-31
119388	2017	August	31	2017-August-31	2017-08-31
119389	2017	August	29	2017-August-29	2017-08-29

118898 rows × 5 columns

Enfin, on ajoute une colonne supplementaire a la base de donnees initiale, on va l'appeler "arrival_date":

```
In [16]: df_drop['arrival_date']=df_date['arrival_date']
```

```
<ipython-input-16-0bb6e2a88710>:1: 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/
ng-a-view-versus-a-copy
  df_drop['arrival_date']=df_date['arrival_date']
```

La variable de réservation:

Avant de commencer, il vaut mieux exposer la variable "reservation_statut_date".

Date à laquelle le dernier statut a été défini. Cette variable peut être utilisée conjointement avec la réservationStatus pour comprendre quand la réservation a été annulée ou quand le client a-t-il quitté l'hôtel.

```
In [18]: df_drop['reservation_status_date']

Out[18]: 0         2015-07-01
         1         2015-07-01
         2         2015-07-02
         3         2015-07-02
         4         2015-07-03
                      ...
         119385    2017-09-06
         119386    2017-09-07
         119387    2017-09-07
         119388    2017-09-07
         119389    2017-09-07
         Name: reservation_status_date, Length: 118898, dtype: object
```

On constate alors que cette variable donne la date en sa forme canonique de la dernière définition du statut de réservation. D'ailleurs, on va introduire une autre variable qui va indiquer est ce que la réservation est valide ou non. Au reste , il faut répondre à la question suivante : C'est quoi une réservation valide?

Évidemment, on peut dire d'une réservation qu'elle est valide si elle survient après ou à la même date que celui d'arrivée.

Eh bien, on exécute la commande suivante pour s'assurer de la validation de la réservation en retournant 1/True si oui, et 0/False sinon :

```
df_drop['reservation_valid']=df_drop['arrival_date'].map(lambda x:datetime.strptime(x, '%Y-%m-%d'))
<= df_drop['reservation_status_date'].map(lambda x:datetime.strptime(x, '%Y-%m-%d'))
```

```
In [20]: df_drop['reservation_valid']=df_drop['arrival_date'].map(lambda x:datetime.strptime(x, '%Y-%m-%d')) <= df_drop['rese
         #datetime.strptime(datestring, '%d-%b-%Y')

         <ipython-input-20-1a32a9a186cf>:1: 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/user_guide/indexing.html#returni
         ng-a-view-versus-a-copy
           df_drop['reservation_valid']=df_drop['arrival_date'].map(lambda x:datetime.strptime(x, '%Y-%m-%d')) <= df_drop['r
         eservation_status_date'].map(lambda x:datetime.strptime(x, '%Y-%m-%d'))

In [21]: df_drop['reservation_valid'].value_counts()

Out[21]: True     76811
         False    42087
         Name: reservation_valid, dtype: int64
```

Alors, parmi 118898 réservations, juste 76811 sont valides.

III. L’analyse exploratoire de données :

Dans cette étape on doit effectuer une sorte d’analyse exploratoire, qui va nous aider par la suite dans la mission de modélisation de données et même dans la narration d’histoire liée à cette base de données.

Voire qu’on a 3 types de variable, on va diviser cette analyse en 3 parties, la première c’est l’analyse des variables temporelles, la deuxième concerne les variables catégorielles, et la troisième met l’accent sur les variables numériques.

1. l’analyse des variables temporelles

Premièrement on va créer une différente base de données qui ne contient que les variables décrivantes le temps ainsi que celle-ci “is_canceled” ;

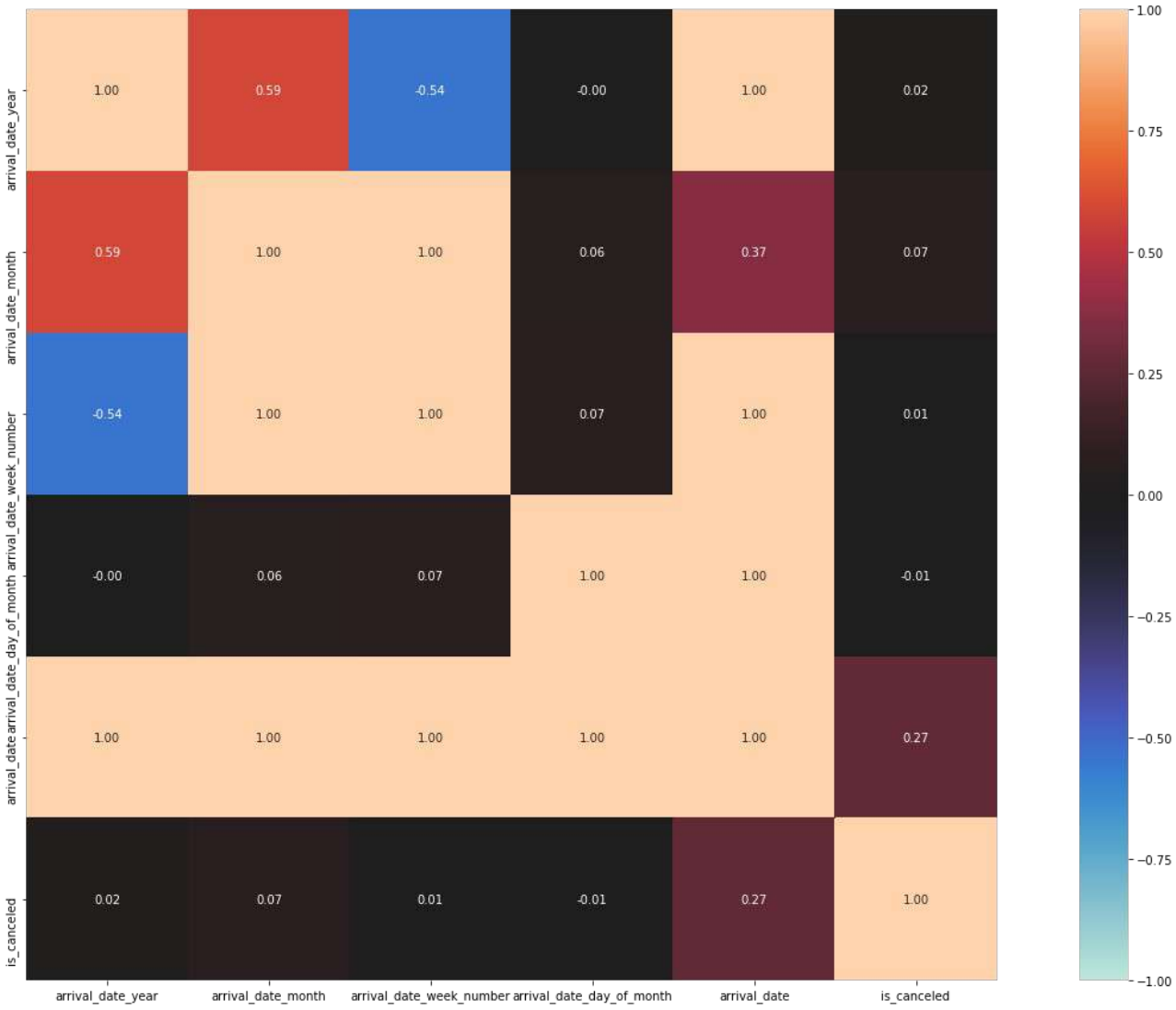
```
In [16]: df_date = df_drop[['arrival_date_year',
                'arrival_date_month', 'arrival_date_week_number',
                'arrival_date_day_of_month','arrival_date','is_canceled']]

         df_date=df_date.sort_values(by=['arrival_date_year','arrival_date_week_number'])

         df_date
```

Out[16]:

	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	arrival_date	is_canceled
0	2015	July	27	1	2015-07-01	0
1	2015	July	27	1	2015-07-01	0
2	2015	July	27	1	2015-07-01	0
3	2015	July	27	1	2015-07-01	0
4	2015	July	27	1	2015-07-01	0
...	...	...	...	...	...	...
119385	2017	August	35	30	2017-08-30	0
119386	2017	August	35	31	2017-08-31	0
119387	2017	August	35	31	2017-08-31	0
119388	2017	August	35	31	2017-08-31	0
119389	2017	August	35	29	2017-08-29	0

118898 rows × 6 columns

On va générer maintenant la matrice de corrélation entre toutes les variables appartenances à “df_date” en utilisant une méthode spéciale s’appelle “Theil U” qui va nous décrire la relation asymétrique entre toutes les variables indépendamment de leurs types ou de leurs formats.

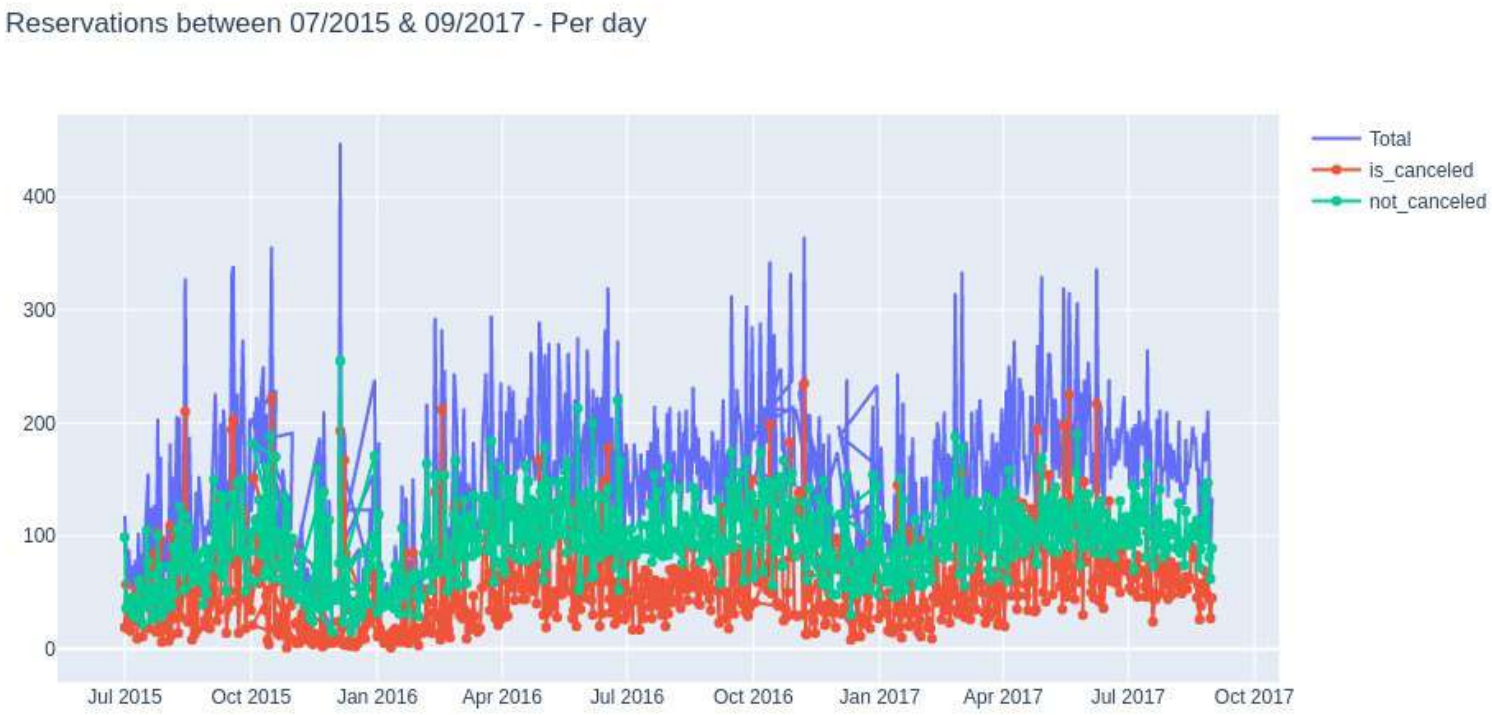


Et le journal statique complet est le suivant :

```
{'corr':                       arrival_date_year  arrival_date_month  \
 arrival_date_year                      1.000000            0.586516
 arrival_date_month                     0.586516            1.000000
 arrival_date_week_number              -0.540493            0.995134
 arrival_date_day_of_month             -0.000590            0.055156
 arrival_date                           1.000000            1.000000
 is_canceled                            0.016412            0.068756

                            arrival_date_week_number  \
 arrival_date_year                         -0.540493
 arrival_date_month                         0.995134
 arrival_date_week_number                   1.000000
 arrival_date_day_of_month                  0.066839
 arrival_date                               1.000000
 is_canceled                                0.007465

                            arrival_date_day_of_month  arrival_date  \
 arrival_date_year                          -0.000590      1.000000
 arrival_date_month                          0.055156      0.372604
```

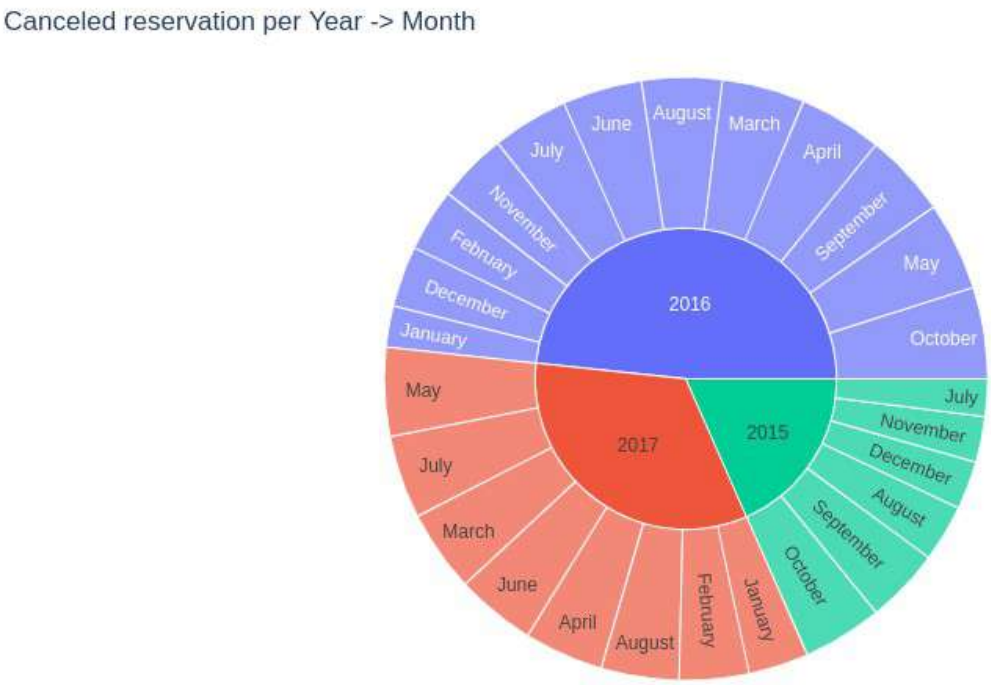
```
arrival_date_week_number                    0.066839      1.000000
arrival_date_day_of_month                   1.000000      1.000000
arrival_date                                1.000000      1.000000
is_canceled                                -0.006127      0.269286

                           is_canceled
arrival_date_year             0.016412
arrival_date_month            0.068756
arrival_date_week_number      0.007465
arrival_date_day_of_month    -0.006127
arrival_date                  0.269286
is_canceled                   1.000000  ,
'ax': <AxesSubplot:>}
```

Comme il est manifeste sur la figure au-dessus, on peut dire que les variables de temps n'influencent pas la annulation des réservations, mais il faut quand même qu'on soit sûr de ce résultat. Alors on va dessiner un graphe qui présente le nombre de réservations (Total, canceled, not canceled) en fonction du temps, depuis le 02/07/2015 jusqu'au 31/08/2017 et voici le résultat :



C'est vrai que le nombre total de réservations varie d'une période à une autre, mais on peut facilement observer que l'aspect général des annulations est le même, en effet, lorsque le nombre de réservations total augmente le taux de annulation augmente aussi, et la même chose pour le taux de validation de réservations, ce qui ne résout pas vraiment le problème posé.

Pour bien comprendre ce point-là on va appeler à d’autres visualisations qui sont plus claires et significatives, comme celle au-dessous:



ce “sunbirst” graphe présente le nombre des annulations en fonction des années puis des mois. On constate qu’il n'y a pas une grande différence entre les nombres d’annulations soit en fonction des années ou en fonction des mois, à noter que 2016 à la part de lion a cause de la période d’étude qui a commencé dans la deuxième moitié de 2015 et ne se termine qu’au mois 8 de l’année 2017, alors toute l’année 2016 était mise en considération, du coup il apparaît que la plupart des annulations dans cette année sont plus importantes que les autres au niveau du taux d'annulation des réservations.

Pour conclure, on peut certainement dire que les variables de temps n’influencent pas - fortement - le taux d’annulation. Alors lors de la modélisation on peut les éliminer pour ne pas gêner le modèle.

2. l’analyse des variables categorielles :

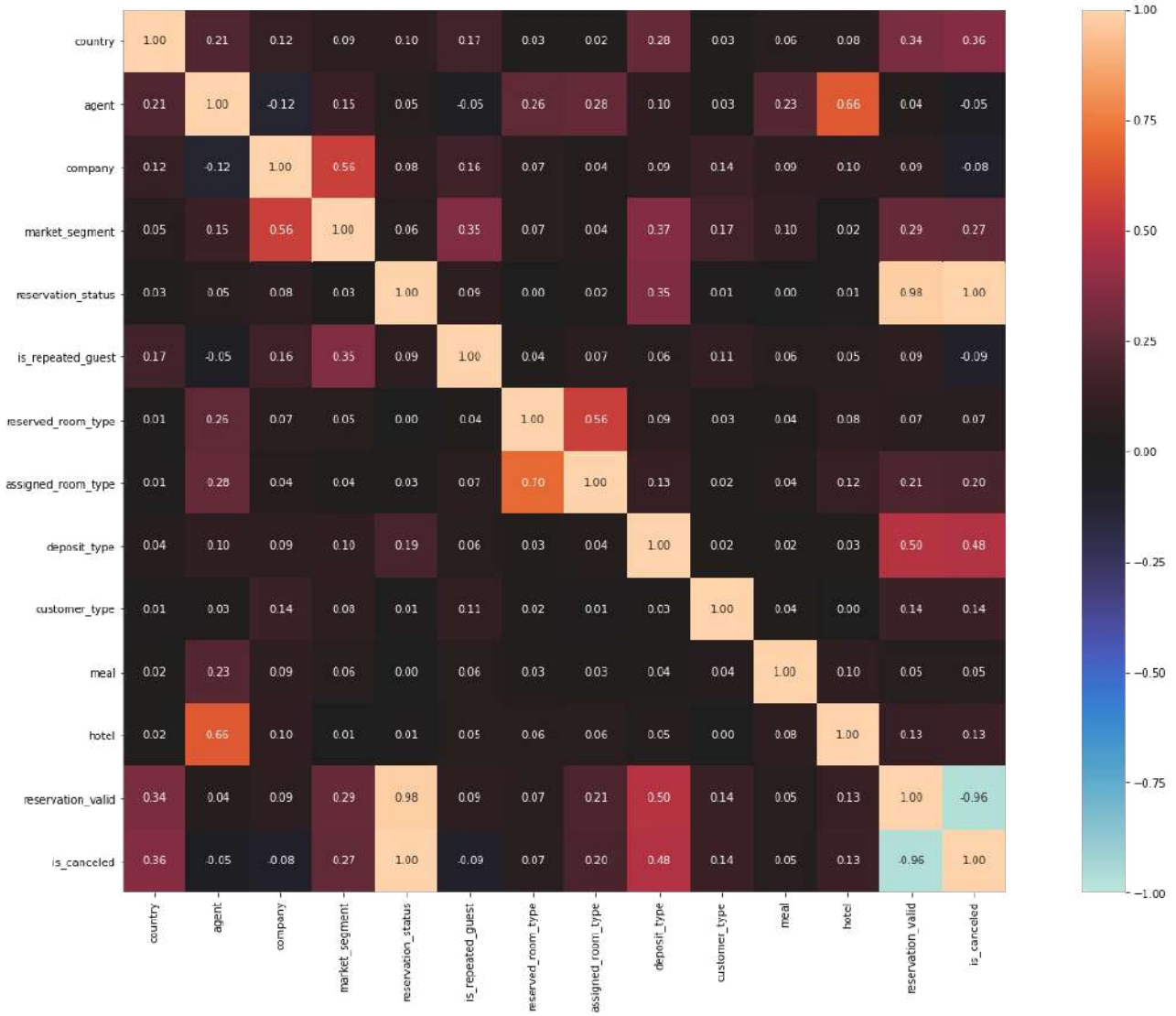
On va commencer par la matrice de corrélation qui représente le journal statistique des différentes variables catégorielles, “is canceled” est inclue :

la définition de la nouvelle base de données :

```
In [26]: df_Categ = df_drop.loc[:, ['country','agent','company','market_segment','reservation_status','is_repeated_guest',
                                    'reserved_room_type','assigned_room_type','deposit_type',
                                    'customer_type','meal','hotel','reservation_valid','is_canceled']]
```

Et comme on a déjà fait avec les variables temporelles, on va générer la matrice de corrélation des variables catégorielles :

```
In [27]: from dython.nominal import associations

         associations(df_Categ, theil_u=True, figsize=(25, 15))
```



Le journal statistique :

```
{'corr':                country   agent   company  market_segment \
 country             1.000000  0.212563  0.119048        0.090096
 agent               0.212563  1.000000 -0.119788        0.145311
 company             0.119048 -0.119788  1.000000        0.559804
 market_segment      0.051588  0.145311  0.559804        1.000000
 reservation_status  0.028544  0.046996  0.083830        0.028213
 is_repeated_guest   0.168453 -0.052489  0.162036        0.352820
 reserved_room_type  0.012732  0.264090  0.070324        0.049837
 assigned_room_type  0.011097  0.278073  0.043611        0.036543
 deposit_type        0.044512  0.095831  0.089833        0.100650
 customer_type       0.008261  0.032278  0.136460        0.081647
 meal                0.017347  0.228114  0.094371        0.056094
 hotel               0.019958  0.655528  0.101368        0.007044
 reservation_valid   0.340277  0.041201  0.088944        0.286815
 is_canceled         0.360261 -0.046543 -0.082306        0.265683

                    reservation_status  is_repeated_guest
reserved_room_type  \
 country                     0.098236           0.168453
0.032229
 agent                       0.046996          -0.052489
0.264090
 company                     0.083830           0.162036
0.070324
 market_segment              0.055598           0.352820
0.072237
 reservation_status          1.000000           0.086393
0.002971
 is_repeated_guest           0.086393           1.000000
0.037998
 reserved_room_type          0.004039           0.037998
1.000000
 assigned_room_type          0.031518           0.071810
0.697498
 deposit_type                0.189817           0.058688
0.034532
 customer_type               0.014261           0.105897
0.021906
 meal                        0.002417           0.060939
0.029656
 hotel                       0.013010           0.051349
0.055309
 reservation_valid           0.984289           0.090147
0.073531
 is_canceled                 1.000000          -0.085179
0.073260

                    assigned_room_type  deposit_type  customer_type
meal  \
 country                     0.022363      0.283030       0.029296
0.056193
```

```
 agent                          0.278073      0.095831       0.032278
0.228114
 company                        0.043611      0.089833       0.136460
0.094371
 market_segment                 0.042167      0.366454       0.165795
0.104046
 reservation_status             0.018456      0.350697       0.014695
0.002275
 is_repeated_guest              0.071810      0.058688       0.105897
0.060939
 reserved_room_type             0.555279      0.086740       0.030689
0.037951
 assigned_room_type             1.000000      0.129044       0.019205
0.041191
 deposit_type                   0.040898      1.000000       0.016056
0.017954
 customer_type                  0.010913      0.028789       1.000000
0.038114
 meal                           0.025625      0.035241       0.041726
1.000000
 hotel                          0.063932      0.046431       0.001902
0.082578
 reservation_valid              0.206356      0.497313       0.135715
0.048772
 is_canceled                    0.201854      0.481372       0.137798
0.050581

                        hotel  reservation_valid  is_canceled
 country             0.076224           0.340277     0.360261
 agent               0.655528           0.041201    -0.046543
 company             0.101368           0.088944    -0.082306
 market_segment      0.015404           0.286815     0.265683
 reservation_status  0.014437           0.984289     1.000000
 is_repeated_guest   0.051349           0.090147    -0.085179
 reserved_room_type  0.083447           0.073531     0.073260
 assigned_room_type  0.121160           0.206356     0.201854
 deposit_type        0.027888           0.497313     0.481372
 customer_type       0.002048           0.135715     0.137798
 meal                0.097357           0.048772     0.050581
 hotel               1.000000           0.132182     0.133964
 reservation_valid   0.132182           1.000000    -0.963104
 is_canceled         0.133964          -0.963104     1.000000  ,
 'ax': <AxesSubplot:>}
```

---

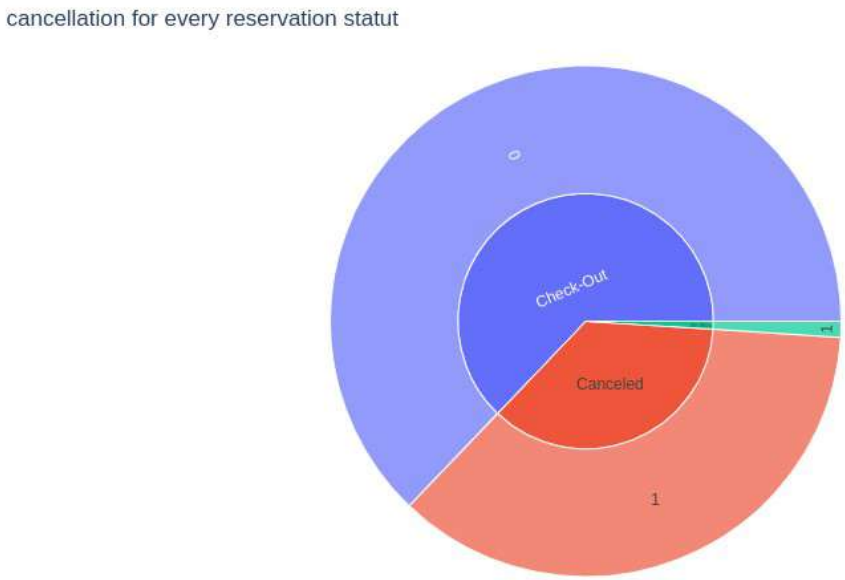
d’après le journal statistique, on peut bien dire que les variables suivantes sont celles-ci catégorielles qui influencent le plus le taux d’annulation des réservations :

- reservation_statut : 1
- reservation_valid : 0.963104
- deposit_type : 0.481372
- country : 0.360261

“reservation_statut” :

On a le taux de “corr” entre “réservations statut” et “is canceled” est égal à 1, ce qui n’est pas logique sauf si la première variable est une copie de la deuxième variable qui n’est pas le cas -, ou bien si elle donne les mêmes informations et les renseignements que la variable “is canceled” sans qu'elle la copie. Pour tester tous ces hypothèses, voyons la distribution de la variable “réservation statut” par rapport celle de la deuxième :

on va dessiner le graphe “Sunburst” suivant :



Dans la figure au-dessus, on peut bien déterminer ou existe l’anomalie pour laquelle le Coeff. “corr” soit égal à 1, en effet, on a 3 valeurs possibles de la variable “réservation statut” qui sont :

- Check-out : pour cette option “is_canceled”=0
- Canceled : pour cette option “is_canceled”=1
- No-Show : pour cette option “is_canceled”=1

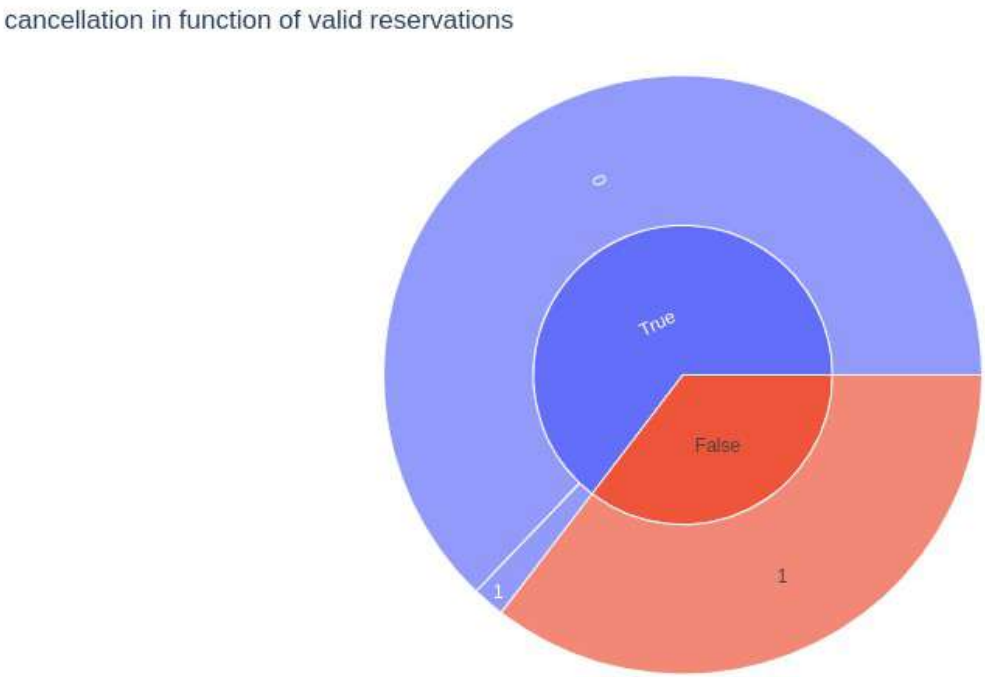
Alors, on a une redondance d’information, Autrement dit, c’est comme si on a la colonne “is canceled” 2 fois répétée au niveau de la base de données.

Bref, lors de la modélisation on va éliminer cette variable-là pour ne pas tomber dans un “if else” problème, et se bénéficier de la capacité totale des algorithmes de la machine learning.

“reservation_valid” :

On a ‘corr’ = -0.963104, Alors cette variable est fortement corrélée avec “is canceled” ce qui va nous servir lors de l’entrainement et la prediction de notre modèle. En effet, ces deux variables sont inversement proportionnelles, si la réservation est valide, la probabilité de l’annuler est faible, et vis versa.

Graphiquement on peut visualiser cela sous la forme suivante :

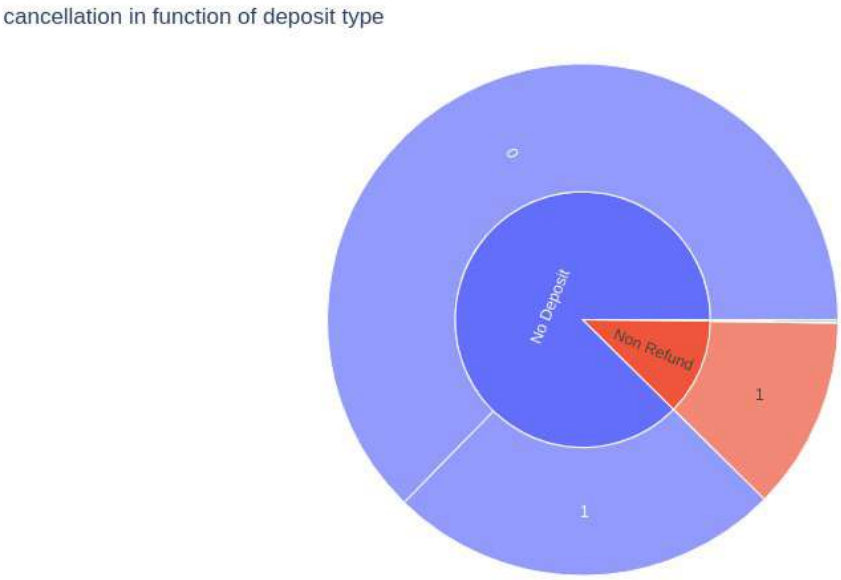


tel que : True/False sont associees a “reservation_valid”

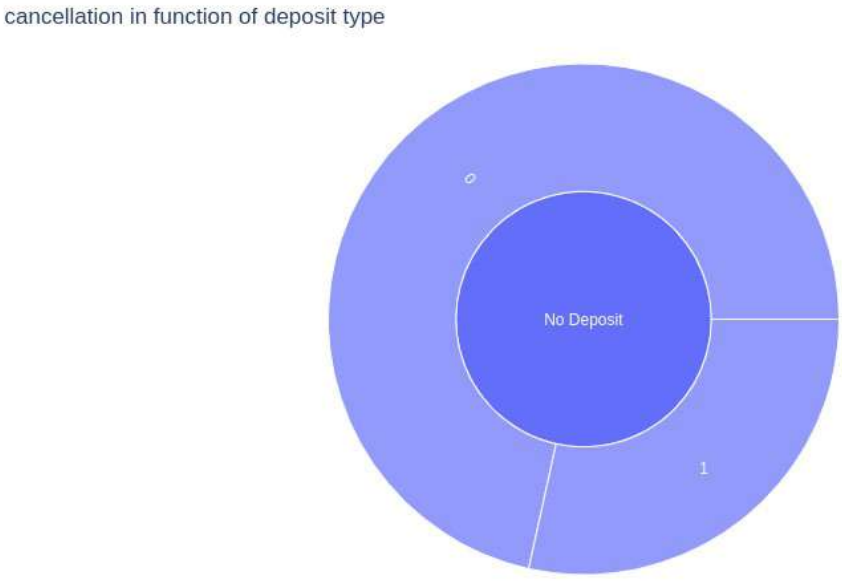
et 0/1 sont associees a “is_canceled”

“deposit_type” :

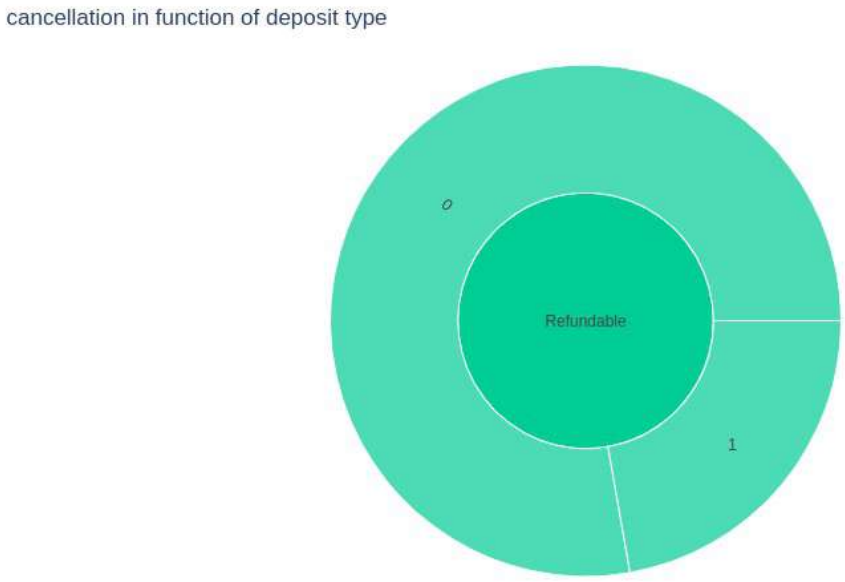
De meme on a pour “deposit_type” :



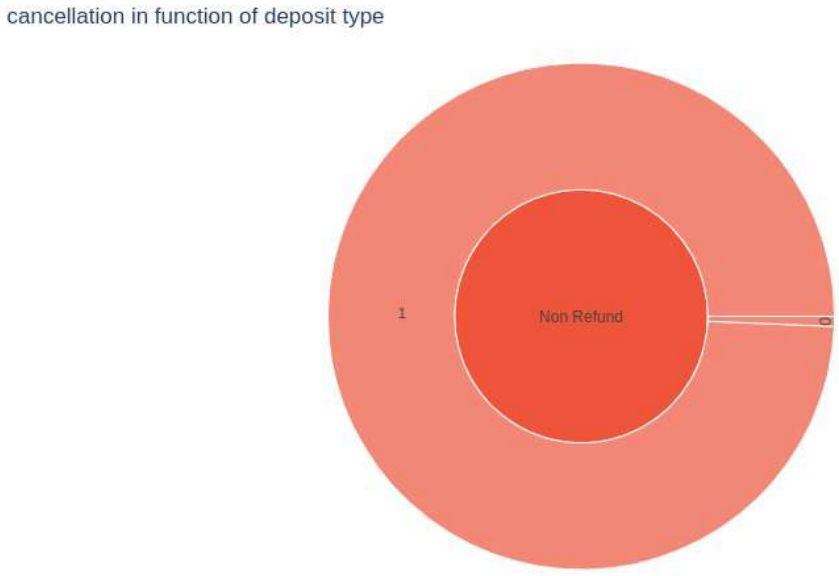
Quand le client n’a pas de dépôt, la probabilité de ne pas annuler sa réservation augmente, effectivement :



Quand un client fait un dépôt d’une valeur inférieure du cout de séjour, il a une chance faible pour qu’il annule sa réservation, graphiquement on a :

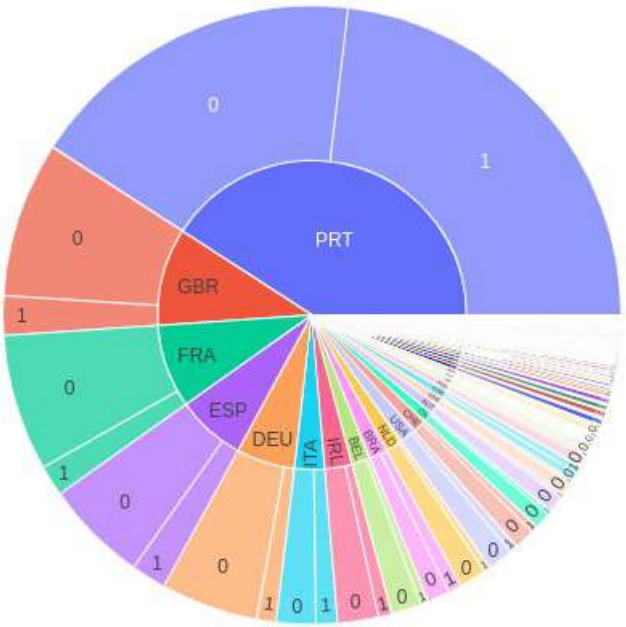


Dans le cas restant où les clients font des dépôts égaux du cout total du séjour, on observe qu’une importante partie d’eux annule les réservations, en effet ;



“country” :

Pour la variable “country”,On a le coeff de corrélation est égal à 0.3, même si cette valeur n’apparaît pas assez grande pour engendrer une véritable différence, mais si on la comparait aux autres variables, on constate qu’elle mérite d’être traitée, d’abord on va commencer par un graphe simple dans lequel, la distribution des pays par rapport les annulations manifeste, Effectivement, on a :



le Portugal est le premier pays soit au niveau d’annulation ou bien de réservation jusqu’au bout, suivi par le Royaume-Uni (GBR) puis la France, l’Espagne, l’Allemagne, l’Italie…etc.

Tous ces pays mentionnés avant se situent à l'Europe et en mettant en considération que le pays qui a le plus important nombre de réservations est le Portugal, alors on peut établir plusieurs hypothèses sur les emplacements ou se trouve la majorité des hôtels sujets de l’étude.

La plus intéressante hypothèse c’est que la grande partie de ces hôtels se trouvent en Europe, précisément en Portugal, surtout si on connaît que pendant les années 2015 jusqu’à 2017 le Portugal a cassé son record au niveau des internationales touristes.

3. l’analyse des variables continues(numériques) :

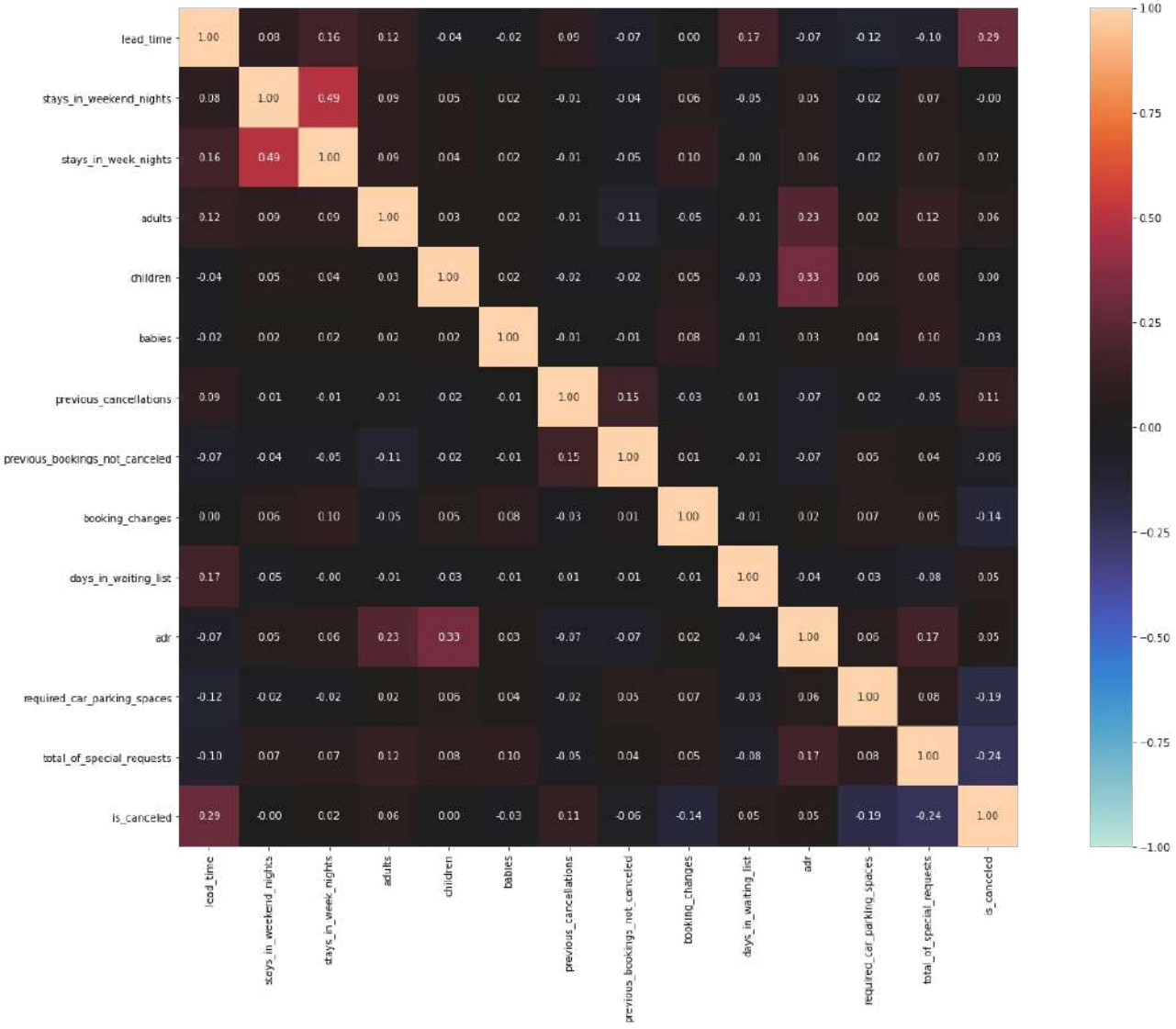
De meme on genere la matrice de correlation de la base de donnees contenante juste les variables numeriques e plus du target : “is_canceled”.

```
In [35]: df_num = df_drop.loc[:, ['lead_time','stays_in_weekend_nights','stays_in_week_nights','adults','children',
                                  'babies','previous_cancellations','previous_bookings_not_canceled','booking_changes',
                                  'days_in_waiting_list','adr','required_car_parking_spaces','total_of_special_requests',
                                  'is_canceled']]
         df_num.head()
```

Out[35]:

	lead_time	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	previous_cancellations	previous_bookings_not_canceled	booking_cha
0	342	0	0	2	0.0	0	0	0	
1	737	0	0	2	0.0	0	0	0	
2	7	0	1	1	0.0	0	0	0	
3	13	0	1	1	0.0	0	0	0	
4	14	0	2	2	0.0	0	0	0	

Alors on a :



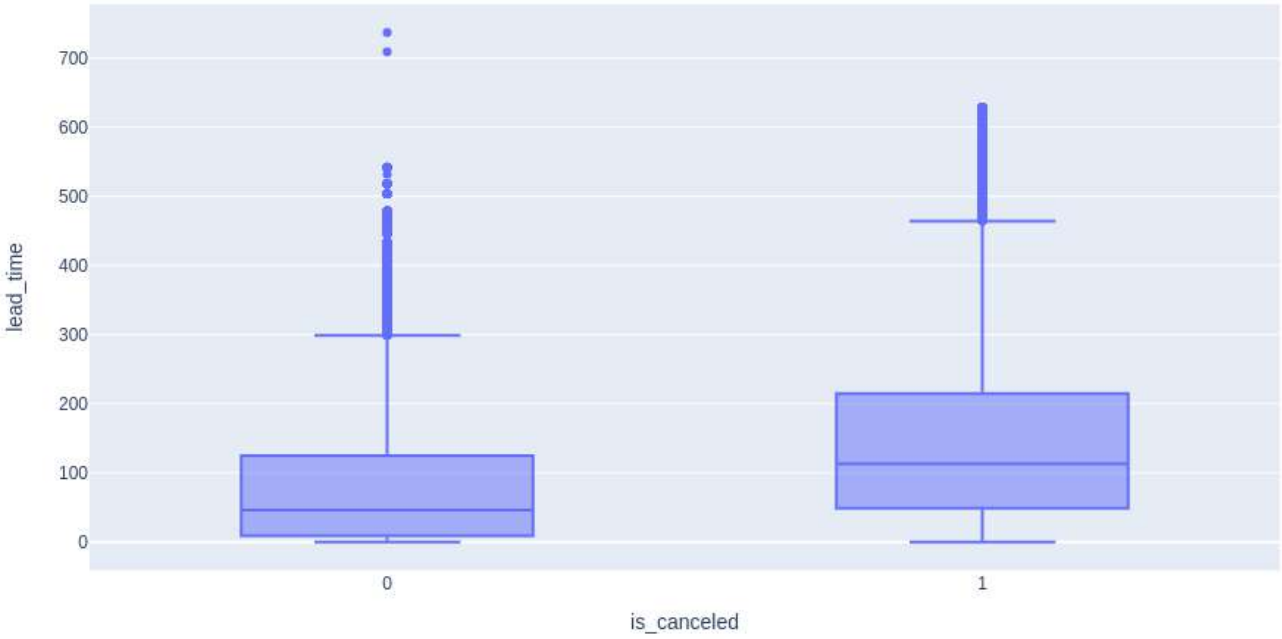
Journal statistique :

```
{'corr':                                lead_time  stays_in_weekend_nights  \
 lead_time                       1.000000                 0.083984
 stays_in_weekend_nights         0.083984                 1.000000
 stays_in_week_nights            0.164783                 0.494888
 adults                          0.116799                 0.090410
 children                       -0.038335                 0.045430
 babies                         -0.021149                 0.018396
 previous_cancellations          0.085961                -0.013008
 previous_bookings_not_canceled -0.071128                -0.040597
 booking_changes                 0.000004                 0.062401
 days_in_waiting_list            0.170007                -0.054568
 adr                            -0.066381                 0.047300
 required_car_parking_spaces    -0.115561                -0.018147
 total_of_special_requests      -0.096536                 0.071669
 is_canceled                     0.291994                -0.002631

                                stays_in_week_nights    adults  children  \
 lead_time                                  0.164783  0.116799 -0.038335
 stays_in_weekend_nights                    0.494888  0.090410  0.045430
 stays_in_week_nights                       1.000000  0.091999  0.044259
 adults                                     0.091999  1.000000  0.029590
 children                                   0.044259  0.029590  1.000000
 babies                                     0.020157  0.017887  0.024131
 previous_cancellations                    -0.014274 -0.006974 -0.024752
 previous_bookings_not_canceled            -0.047367 -0.105028 -0.020364
 booking_changes                            0.095665 -0.052420  0.048660
 days_in_waiting_list                      -0.002161 -0.008765 -0.033396
 adr                                        0.063628  0.227480  0.325034
 required_car_parking_spaces               -0.024378  0.016370  0.057060
 total_of_special_requests                  0.066785  0.121815  0.081786
 is_canceled                                0.024110  0.058381  0.004751

                                  babies  previous_cancellations  \
 lead_time                      -0.021149                0.085961
 stays_in_weekend_nights         0.018396               -0.013008
 stays_in_week_nights            0.020157               -0.014274
 adults                          0.017887               -0.006974
 children                        0.024131               -0.024752
 babies                          1.000000               -0.007489
 previous_cancellations         -0.007489                1.000000
 previous_bookings_not_canceled -0.006306                0.154285
 booking_changes                 0.083220               -0.027092
 days_in_waiting_list           -0.010648                0.005927
 adr                             0.028591               -0.065930
 required_car_parking_spaces     0.036971               -0.018455
 total_of_special_requests       0.097601               -0.048585
 is_canceled                    -0.032521                0.109922

                                 previous_bookings_not_canceled  \
 lead_time                                            -0.071128
 stays_in_weekend_nights                              -0.040597
 stays_in_week_nights                                 -0.047367
 adults                                               -0.105028
```

```
children                                        -0.020364
babies                                          -0.006306
previous_cancellations                           0.154285
previous_bookings_not_canceled                   1.000000
booking_changes                                  0.011970
days_in_waiting_list                            -0.009011
adr                                             -0.069638
required_car_parking_spaces                      0.046945
total_of_special_requests                        0.037595
is_canceled                                     -0.055493

                                booking_changes  days_in_waiting_list  \
lead_time                             0.000004              0.170007
stays_in_weekend_nights               0.062401             -0.054568
stays_in_week_nights                  0.095665             -0.002161
adults                               -0.052420             -0.008765
children                              0.048660             -0.033396
babies                                0.083220             -0.010648
previous_cancellations               -0.027092              0.005927
previous_bookings_not_canceled        0.011970             -0.009011
booking_changes                       1.000000             -0.011661
days_in_waiting_list                 -0.011661              1.000000
adr                                   0.019201             -0.041333
required_car_parking_spaces           0.065724             -0.030462
total_of_special_requests             0.052434             -0.082970
is_canceled                          -0.144659              0.054016

                                     adr  required_car_parking_spaces  \
lead_time                       -0.066381                    -0.115561
stays_in_weekend_nights          0.047300                    -0.018147
stays_in_week_nights             0.063628                    -0.024378
adults                           0.227480                     0.016370
children                         0.325034                     0.057060
babies                           0.028591                     0.036971
previous_cancellations          -0.065930                    -0.018455
previous_bookings_not_canceled  -0.069638                     0.046945
booking_changes                  0.019201                     0.065724
days_in_waiting_list            -0.041333                    -0.030462
adr                              1.000000                     0.058053
required_car_parking_spaces      0.058053                     1.000000
total_of_special_requests        0.171458                     0.082675
is_canceled                      0.046199                    -0.194796

                                total_of_special_requests  is_canceled
lead_time                                       -0.096536     0.291994
stays_in_weekend_nights                          0.071669    -0.002631
stays_in_week_nights                             0.066785     0.024110
adults                                           0.121815     0.058381
children                                         0.081786     0.004751
babies                                           0.097601    -0.032521
previous_cancellations                          -0.048585     0.109922
previous_bookings_not_canceled                   0.037595    -0.055493
booking_changes                                  0.052434    -0.144659
days_in_waiting_list                            -0.082970     0.054016
adr                                              0.171458     0.046199
required_car_parking_spaces                      0.082675    -0.194796
total_of_special_requests                        1.000000    -0.235643
is_canceled                                     -0.235643     1.000000  ,
'ax': <AxesSubplot:>}
```

Premierement, on va decrire la variable “lead_time” en ce qui concerne se relation avec celle-ci “is_canceled”. En effet, elle donne le nombre de jours écoulés entre la date d'entrée de la réservation dans le PMS et la date d'arrivée.

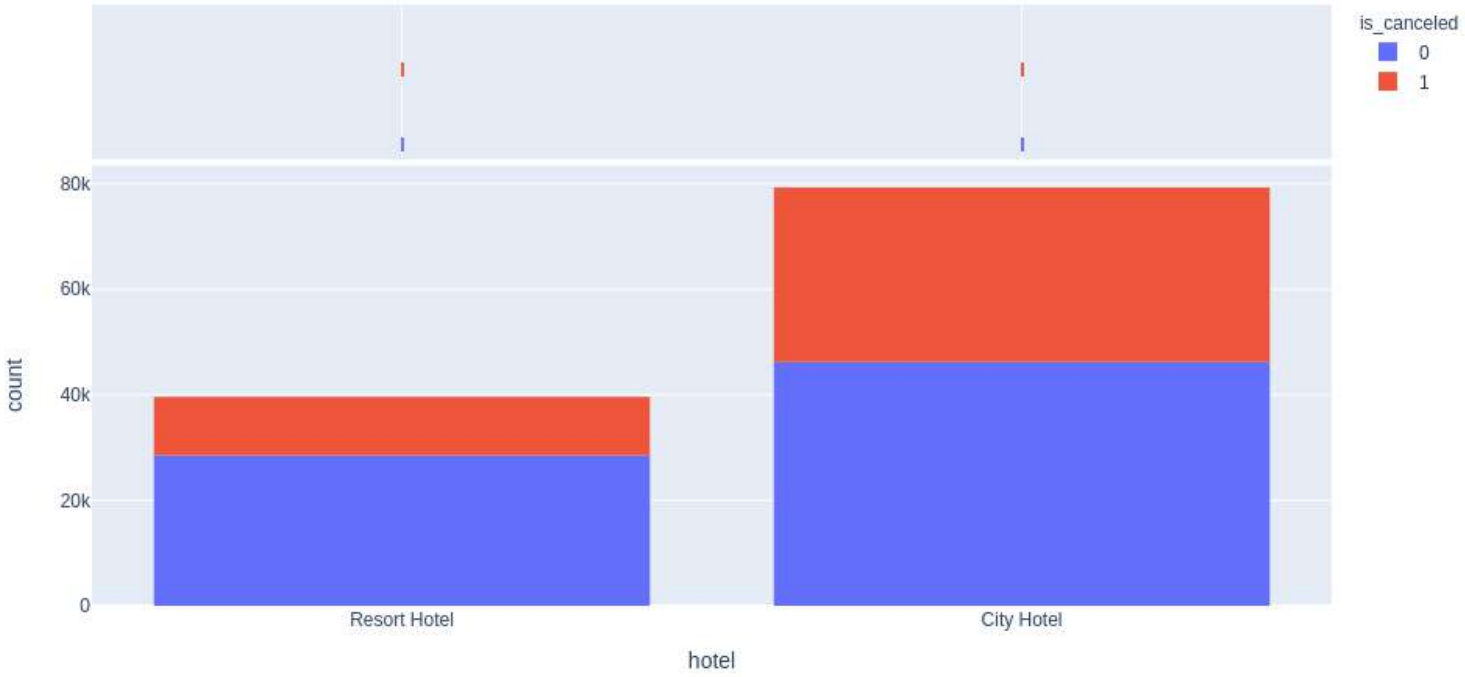
Construisons d’abord les boxplots suivants :



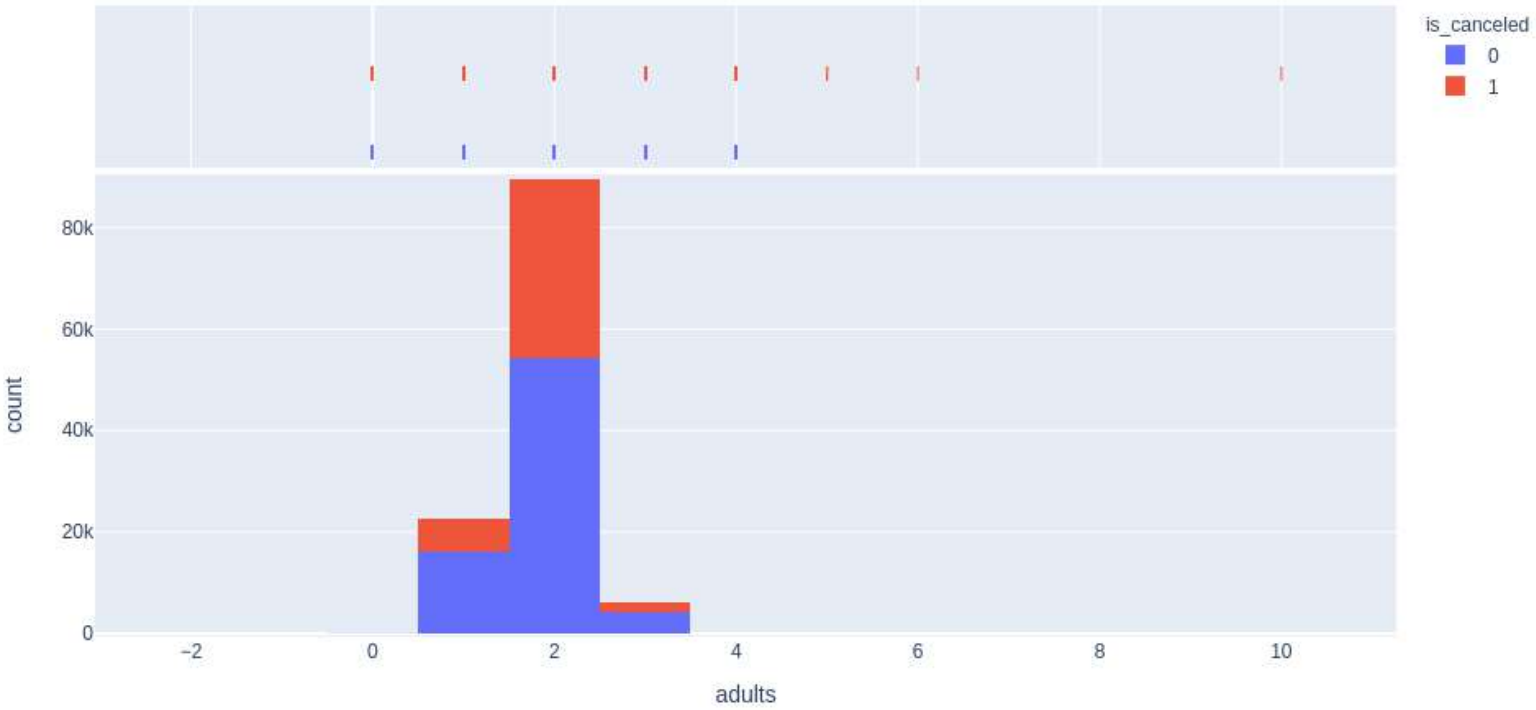
Par conséquent, plus le nombre de jours entre la date d'entrée de la réservation dans le PMS et la date d'arrivée est petit, plus il est probable que le client n'annule pas sa réservation.

Pour les autres variables, elles sont soit corrélé avec “lead time” soit elles ont des faibles coeff de corrélation. C’est pourquoi on va stopper ici en ce qui concerne l’analyse des variables continues / numériques.

Distribution du “hotel” & “is_canceled” :



Distribution du “Adults” & “is_canceled” :



Finalement, il faut admettre que les hôtels qui font partie de cette étude ont des immenses chances pour bien se développer et augmenter leurs revenues. En effet, ils peuvent se concentrer sur les clients de différentes nationalités comme ils le font avec les portugaises, ainsi il faut mettre en valeur les “ressort Hôtel” surtout que le taux d’annulation des reservations dans ce type d’hôtels est très inférieur que celui des “City Hôtel”, sans oublier qu’ils peuvent encourager les gens à choisir les modes “No deposit” et “Refundable” au lieu du celui de “not Refund” lors de la réservation sur le PMS

IV-Modeling :

Dans cette partie, nous allons :

1. **Préparer les données**
2. **Préparer un modèle d'apprentissage automatique**
3. **Évaluer les prédictions du modèle**
4. **Expérimenter différents modèles de classification**
5. **Hyperparameter Tuning : KNN avec RandomzedSearchCV**
6. **LGBMClassifier**

Nous allons travailler avec la librairie Scikit-learn.

Scikit-learn est une librairie pour Python spécialisée dans le machine learning (apprentissage automatique). Elle propose dans son framework de nombreuses algorithmes.

1. Préparer les données :

Nous commencerons par l'élimination des colonnes qu'on va pas utiliser dans notre modélisation :

```
In [28]: main_cols = df_pre.columns.difference(['children', 'meal', 'reservation_status',
                                                'reservation_status_date', 'arrival_date']).tolist()
         df_pre = df_pre[main_cols]
         df_pre.head()
```

Out[28]:

	adr	adults	agent	assigned_room_type	babies	booking_changes	customer_type	days_in_waiting_list	deposit_type	distribution_channel	...	lead_time
0	0.0	2	0.0	C	0	3	Transient	0	No Deposit	Direct	...	342
1	0.0	2	0.0	C	0	4	Transient	0	No Deposit	Direct	...	737
2	75.0	1	0.0	C	0	0	Transient	0	No Deposit	Direct	...	7
3	75.0	1	304.0	A	0	0	Transient	0	No Deposit	Corporate	...	13
4	98.0	2	240.0	A	0	0	Transient	0	No Deposit	TA/TO	...	14

5 rows × 23 columns

Les colonnes de type « String » :

```
In [29]: df_pre_object = df_pre.select_dtypes(include=['object']).copy()
         df_pre_object.head()
```

Out[29]:

	assigned_room_type	customer_type	deposit_type	distribution_channel	hotel	market_segment	reserved_room_type
0	C	Transient	No Deposit	Direct	Resort Hotel	Direct	C
1	C	Transient	No Deposit	Direct	Resort Hotel	Direct	C
2	C	Transient	No Deposit	Direct	Resort Hotel	Direct	A
3	A	Transient	No Deposit	Corporate	Resort Hotel	Corporate	A
4	A	Transient	No Deposit	TA/TO	Resort Hotel	Online TA	A

In [76]:
```
print(df_pre["assigned_room_type"].value_counts())
print(df_pre["customer_type"].value_counts())
print(df_pre["deposit_type"].value_counts())
print(df_pre["distribution_channel"].value_counts())
print(df_pre["hotel"].value_counts())
print(df_pre["market_segment"].value_counts())
print(df_pre["reserved_room_type"].value_counts())
```

```
A    73863
D    25166
E     7738
F     3732
G     2539
C     2354
B     2159
H      708
I      357
K      279
P        2
L        1
Name: assigned_room_type, dtype: int64
Transient          89174
Transient-Party    25078
Contract            4076
Group                570
Name: customer_type, dtype: int64
No Deposit    104163
Non Refund     14573
Refundable       162
Name: deposit_type, dtype: int64
TA/TO        97730
Direct       14483
Corporate     6491
GDS            193
Undefined        1
Name: distribution_channel, dtype: int64
City Hotel      79302
Resort Hotel    39596
Name: hotel, dtype: int64
Online TA        56402
Offline TA/TO    24160
Groups           19806
Direct           12448
Corporate         5111
Complementary      734
Aviation           237
Name: market_segment, dtype: int64
A    85601
D    19173
E     6497
F     2890
G     2083
B     1114
C      931
H      601
L        6
P        2
Name: reserved_room_type, dtype: int64
```

On doit transformer ces colonnes en types « intiger », sinon le modèle ça ne va pas marche :

On utilise la technique **« one hot encoding »**

In [31]:
```
# One hot encoding
df_pre = pd.get_dummies(df_pre, columns = ['hotel', 'market_segment', 'distribution_channel', 'assigned_room_type',
                                           'reserved_room_type', 'deposit_type', 'customer_type'])
```

In [32]:
```
df_pre.head()
```

Out[32]:

	adr	adults	agent	babies	booking_changes	days_in_waiting_list	is_canceled	is_repeated_guest	lead_time	previous_bookings_not_canceled	...	reserve
0	0.0	2	0.0	0	3	0	0	0	342	0	...	
1	0.0	2	0.0	0	4	0	0	0	737	0	...	
2	75.0	1	0.0	0	0	0	0	0	7	0	...	
3	75.0	1	304.0	0	0	0	0	0	13	0	...	
4	98.0	2	240.0	0	0	0	0	0	14	0	...	

5 rows × 59 columns

Notre objectif ici est de créer un modèle d'apprentissage automatique sur toutes les colonnes restantes dans le dataframe, à l'exception des targets pour prédire les targets **« is_canceled »**.

En substance, la colonne « is_canceled » est notre variable cible (également appelée y ou label) et le reste des autres colonnes sont nos variables indépendantes (également appelées caractéristiques ou X).

In [33]:
```
X = df_pre.drop('is_canceled', axis = 1)
y = df_pre['is_canceled']
```

Maintenant que nous avons divisé nos données en X et y, nous utiliserons Scikit-Learn pour les diviser en ensembles d'entraînement et de test (80% de training et 20% de test).

In [34]:
```
# Splitting Data
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

2. Préparer un modèle d'apprentissage automatique :

Étant donné que nos données sont maintenant dans des ensembles d'entraînement et de test, nous allons créer un modèle d'apprentissage automatique pour adapter les modèles dans les données d'entraînement, puis effectuer des prédictions sur les données de test.

Pour ce faire commencent d'abord par l'importation des modèles qu'on va utiliser et les métriques pour évaluer les performances de ces modèles :

In [35]:
```
# Libs
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from sklearn.model_selection import StratifiedKFold, RandomizedSearchCV

from sklearn.metrics import accuracy_score, precision_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

np.random.seed(100)
```

Dans cette partie nous allons appliquer une régression logistique en utilisant le classifier `LogisiticRegression` de la librerie `scikit-learn` :

La régression logistique porte assez mal son nom car il ne s'agit pas à proprement parler d'une régression au sens classique du terme (on essaye pas d'expliquer une variable quantitative mais de classer des individus dans deux catégories).

Cette méthode présente depuis de nombreuses années est la méthode la plus utilisée aujourd'hui en production pour construire des modèles de classification.

Nous pouvons lancer la régression maintenant. Après appel du constructeur de la classe LogisticRegression() où nous passons les données, nous faisons appel à la fonction fit() qui génère un objet résultat doté de propriétés et méthodes qui nous seront très utiles par la suite.

Using Logistic Regression :

```
In [36]: # training
         model_logReg = LogisticRegression()
         model_logReg.fit(X_train, y_train)

Out[36]: LogisticRegression()
```

3. Évaluer les prédictions du modèle :

Évaluer les prédictions est très important. Vérifions comment notre modèle en appelant la méthode score() et en lui passant les données d'entraînement (X_train, y_train) et de test (X_test, y_test).

```
In [37]: # performance on training data
         model_logReg.score(X_train, y_train)

Out[37]: 0.981643852898505

In [38]: # performance on test data
         model_logReg.score(X_test, y_test)

Out[38]: 0.981623212783852
```

Faisons quelques prédictions sur les données de test en utilisant notre dernier modèle et sauvegardons-les dans y_pred :

```
In [39]: # make prediction
         y_pred = model_logReg.predict(X_test)
```

Il est temps d'utiliser les prédictions que notre modèle a trouvé pour l'évaluer :

a) Le Rapport de Classification :

Les colonnes de ce rapport de classification sont :

• **Précision** - Indique la proportion d'identifications positives (classe 1 prédite par le modèle) qui étaient réellement correctes. Un modèle qui ne produit aucun faux positif a une précision de 1,0.

• **Rappel** - Indique la proportion de positifs réels qui ont été correctement classés. Un modèle qui ne produit aucun faux négatif a un rappel de 1,0.

• **Score F1** - Une combinaison de précision et de rappel. Un modèle parfait obtient un score F1 de 1,0.

• **Support** - Le nombre d'échantillons sur lequel chaque métrique a été calculée.

• **Précision** - La précision du modèle sous forme décimale. La précision parfaite est égale à 1,0.

• **Macro moyenne** - Abréviation de macro moyenne, la précision moyenne, le rappel et le score F1 entre les classes. La macro moyenne ne classe pas le déséquilibre en effort, donc si vous avez des déséquilibres de classe, faites attention à cette métrique.

• **Moyenne pondérée** - Abréviation de moyenne pondérée, précision moyenne pondérée, rappel et score F1 entre les classes. Pondéré signifie que chaque métrique est calculée par rapport au nombre d'échantillons dans chaque classe. Cette métrique favorisera la classe majoritaire (par exemple, donnera une valeur élevée lorsqu'une classe surpassera une autre en raison du plus grand nombre d'échantillons).

```
In [40]: cl_report = classification_report(y_test, y_pred)
         print(cl_report)
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99     15034
           1       1.00      0.95      0.97      8746

    accuracy                           0.98     23780
   macro avg       0.99      0.98      0.98     23780
weighted avg       0.98      0.98      0.98     23780
```

on remarque que le rapport de classification pour ce modele est de 1 pour le cas “is_canceled” et 0.97 pour predire la classe not “is_canceled”

b) La Matrice de confusion :

Une **Confusion Matrix** (matrice de confusion) ou tableau de contingence est un outil permettant de mesurer les performances d'un modèle de Machine Learning en vérifiant notamment à quelle fréquence ses prédictions sont exactes par rapport à la réalité dans des problèmes de classification.

Pour bien comprendre le fonctionnement d'une matrice de confusion, il convient de bien comprendre les quatre terminologies principales : TP, TN, FP et FN. Voici la définition précise de chacun de ces termes :

- **TP (True Positives) :** les cas où la prédiction est positive, et où la valeur réelle est effectivement positive. Exemple : le médecin vous annonce que vous êtes enceinte, et vous êtes bel et bien enceinte.
- **TN (True Negatives) :** les cas où la prédiction est négative, et où la valeur réelle est effectivement négative. Exemple : le médecin vous annonce que vous n'êtes pas enceinte, et vous n'êtes effectivement pas enceinte.
- **FP (False Positive) :** les cas où la prédiction est positive, mais où la valeur réelle est négative. Exemple : le médecin vous annonce que vous êtes enceinte, mais vous n'êtes pas enceinte.
- **FN (False Negative) :** les cas où la prédiction est négative, mais où la valeur réelle est positive. Exemple : le médecin vous annonce que vous n'êtes pas enceinte, mais vous êtes enceinte.

	Class 1 Predicted	*Class 2 Predicted*
Class 1 Actual	**TP**	**FN**
Class 2 Actual	**FP**	**TN**

Calcule de la matrice de confusion en python :

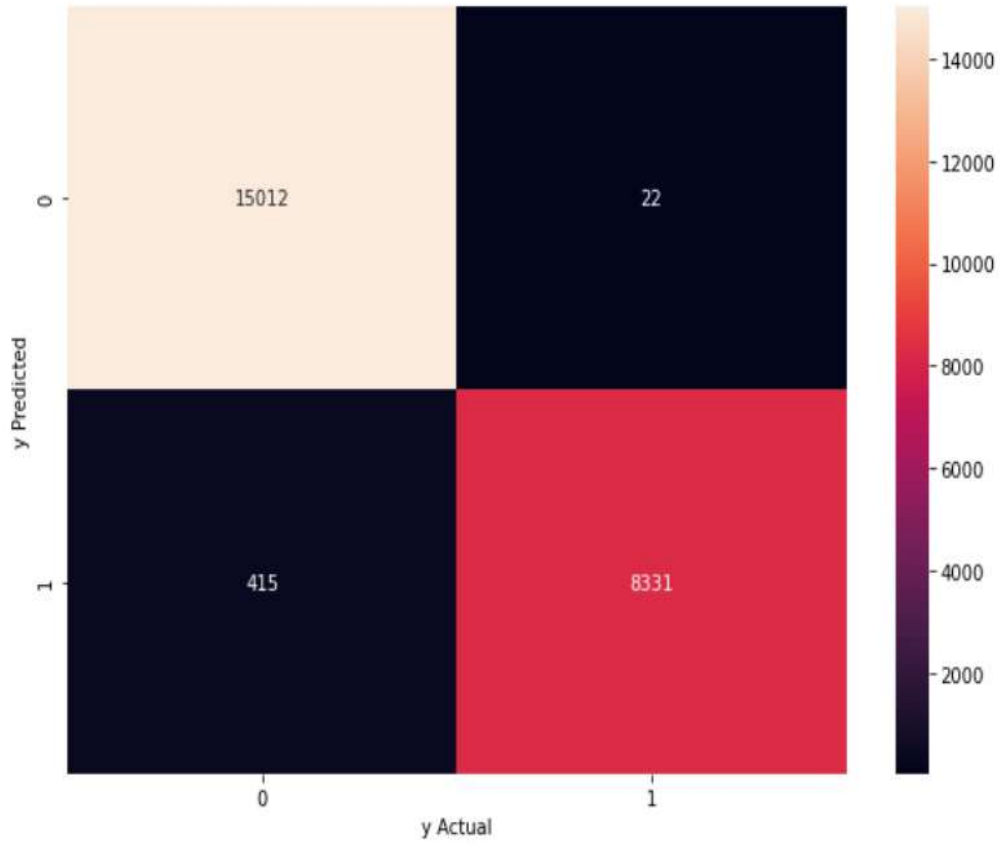
```
In [41]: cnf_matrix = confusion_matrix(y_test, y_pred)
         print(cnf_matrix)
```

```
[[15012    22]
 [  415  8331]]
```

Pour mieux voir cette matrice de confusion on peut la visualiser a l'aide de la bibliotheque seaborn et matplotlib.

Cette visualisation permet d'analyser rapidement la matrice de confusion, ainsi analyser l'efficacité du modèle.

```
In [42]: f, ax = plt.subplots(figsize=(10,7))
         sns.heatmap(cnf_matrix, annot=True, fmt='.0f', ax=ax)
         plt.xlabel('y Actual')
         plt.ylabel('y Predicted')
         plt.show()
```



4. Expérimenter différents modèles de classification :

Nous allons maintenant essayer une série de différents modèles d'apprentissage automatique et voir lequel obtient les meilleurs résultats sur notre ensemble de données

En parcourant la documentation de Scikit-Learn, nous voyons qu'il existe un certain nombre de modèles de classification différents que nous pouvons essayer.

Pour notre cas, les modèles que nous allons essayer de comparer sont :

- **LogisticRegression**
- **DecisionTreeClassifier**
- **RandomForestClassifier**
- **KNeighborsClassifier**
- **SVC**

Nous suivrons le même workflow que nous avons utilisé ci-dessus (sauf cette fois pour plusieurs modèles):

1. **Importer un modèle d'apprentissage automatique**
2. **Préparez-le**
3. **Ajustez-le aux données et faites des prédictions**
4. **Évaluer le modèle ajusté**

Grâce à la cohérence de la conception de l'API de Scikit-Learn, nous pouvons utiliser pratiquement le même code pour ajuster, évaluer et faire des prédictions avec chacun de nos modèles.

Pour voir quel modèle fonctionne le mieux, nous allons procéder comme suit :

1. **Instancier chaque modèle dans un dictionnaire**
2. **Créer un dictionnaire de résultats vide**
3. **Ajustez chaque modèle sur les données d'entraînement**
4. **Evaluer chaque modèle sur les données de test**
5. **Vérifiez les résultats**

Étant donné que chaque modèle que nous utilisons a les mêmes fonctions fit () et score (), nous pouvons parcourir notre dictionnaire de modèles et appeler fit () sur les données d'entraînement, puis appeler score () avec les données de test :

In [44]:
```
%%time

models = {
    "LogisticRegression" :LogisticRegression() ,
    "DecisionTreeClassifier" : DecisionTreeClassifier(),
    "RandomForestClassifier" : RandomForestClassifier(),
    "KNN" : KNeighborsClassifier(),
    "SVC" : SVC(),
}
results = {}

for model_name, model in models.items():
    model.fit(X_train, y_train)
    results[model_name] = model.score(X_test, y_test)

sort_results = sorted(results.items(), key=lambda x: x[1], reverse=True)
sort_results
```

```
Wall time: 23min 36s
```

Out[44]:
```
[('RandomForestClassifier', 0.9843566021867115),
 ('LogisticRegression', 0.981623212783852),
 ('DecisionTreeClassifier', 0.9727922624053826),
 ('KNN', 0.8222035323801514),
 ('SVC', 0.8043734230445753)]
```

Le modèle qui fonctionne le mieux pour ce probleme est RandomForestClassifier avec un score, sur les donnees de test, de 98% ,la regression logistique lui meme a un score de 98%.

Mais il est toujours possible d’améliorer la performance des autres algorithmes, en choisissant les meilleur parametres d’apprentissage, prenons par exemple le KNN.

5. Hyperparameter Tuning : KNN & RandomzedSearchCV :

Pour trouver les meilleurs hyper-parametres il suffit de créer deux listes Python **k_range** et **weight_options**, avec les hyper-paramètres.

Puisque nous avons un ensemble d'hyperparamètres, nous pouvons importer RandomizedSearchCV, lui transmettre le classifier et nos listes d'hyperparamètres et le laisser rechercher la meilleure combinaison de ces hyperparamètres :

KNN with RandomizedSearchCV

In [45]:
```
# instantiate model
knn = KNeighborsClassifier(n_neighbors=5)

k_range = list(range(1, 10))
weight_options = ['uniform', 'distance']
param_dist = dict(n_neighbors=k_range,
                  weights=weight_options)

rand = RandomizedSearchCV(knn,
                          param_dist,
                          cv=10,
                          scoring='accuracy',
                          n_iter=10,
                          random_state=5)

# fit the grid with data
rand.fit(X_train, y_train)
```

Out[45]:
```
RandomizedSearchCV(cv=10, estimator=KNeighborsClassifier(),
                   param_distributions={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8,
                                                        9],
                                        'weights': ['uniform', 'distance']},
                   random_state=5, scoring='accuracy')
```

L'objet rand va conserver le bon parametrage et peut directement faire appel à la fonction **fit()** et **predict().**

Nous pouvons aussi regarder quel paramétrage a été élu via les propriétés **best_score_**, **best_params_** et **best_estimator_**.

```
In [46]: # examine the best model
         print(rand.best_score_)
         print(rand.best_params_)
         print(rand.best_estimator_)
```

```
0.8410815894729126
{'weights': 'distance', 'n_neighbors': 6}
KNeighborsClassifier(n_neighbors=6, weights='distance')
```

Cela nous donne le meilleur score obtenu avec la meilleure combinaison.

Nous avons essayé de trouver les meilleurs hyperparamètres sur notre modèle en utilisant RandomizedSearchCV.

Maintenant nous allons instancier une nouvelle instance de notre modèle en utilisant les meilleurs hyperparamètres trouvés par RandomizedSearchCV :

{'weights': **'distance'**, 'n_neighbors': **6**}

Using the best parameters to make predictions

```
In [53]: # train your model using all data and the best known parameters

         # instantiate model with best parameters
         knn = KNeighborsClassifier(n_neighbors=6, weights='distance')
         # train the model
         knn.fit(X_train, y_train)
```

```
Out[53]: KNeighborsClassifier(n_neighbors=6, weights='distance')
```

Après le processus de réglage des hyperparamètres, le score augmente de plus de 2% :

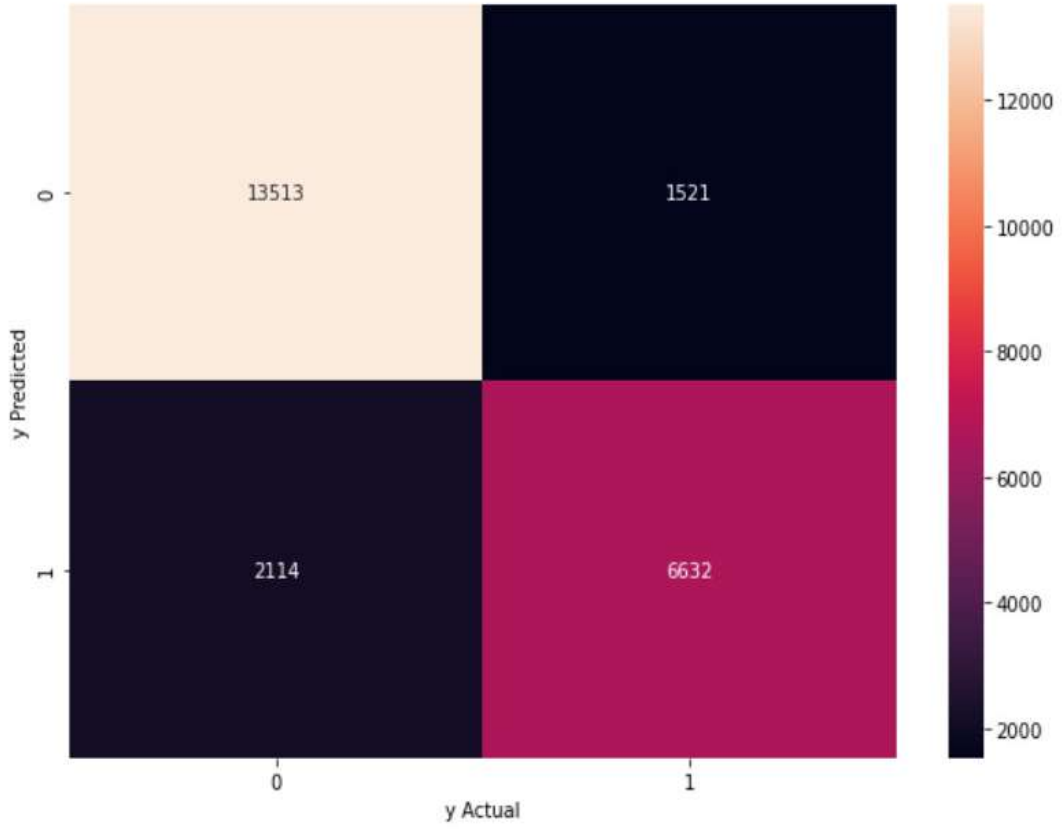
```
In [54]: knn.score(X_test, y_test)

Out[54]: 0.8471404541631623
```

Faisons quelques prédictions sur les données de test en utilisant notre dernier modèle et sauvegardons-les dans y_pred.

```
In [55]: # make prediction
         y_pred = knn.predict(X_test)
```

Il est temps d'utiliser les prédictions que notre modèle a trouvé pour l'évaluer en utilisent la matrice de confusion :

```
In [56]: cnf_matrix = confusion_matrix(y_test, y_pred)
         f, ax = plt.subplots(figsize=(10,7))
         sns.heatmap(cnf_matrix, annot=True, fmt='.0f', ax=ax)
         plt.xlabel('y Actual')
         plt.ylabel('y Predicted')
         plt.show()
```



6. LGBMClassifier :

Maintenant on va travailler avec une autre series d'algorithmes, performants et rapides :

LightGBM est une bibliothèque qui utilise des algorithmes d'apprentissage basés sur les arbres. Il est conçu pour être distribué et efficace par rapport aux autres algorithmes. Un modèle qui peut être utilisé à des fins de comparaison est XGBoost, qui est également une méthode de stimulation et il fonctionne exceptionnellement bien par rapport aux autres algorithmes. Lightgbm peut gérer une grande quantité de données, moins d'utilisation de la mémoire, un apprentissage parallèle et **GPU**, une bonne précision, **une vitesse d'entraînement plus rapide et une efficacité plus grande** par rapport aux autres algorithmes.

Installons d'abord la librairie **lightgbm** :

```
In [57]: pip install lightgbm

Requirement already satisfied: lightgbm in c:\programdata\anaconda3\lib\site-packages (3.2.1)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from lightgbm) (1.18.5)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from lightgbm) (1.5.0)
Requirement already satisfied: wheel in c:\programdata\anaconda3\lib\site-packages (from lightgbm) (0.34.2)
Requirement already satisfied: scikit-learn!=0.22.0 in c:\programdata\anaconda3\lib\site-packages (from lightgbm) (0.23.1)
Requirement already satisfied: joblib>=0.11 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn!=0.22.0->lightgbm)
(0.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn!=0.22.0->l
ightgbm) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

Le principe est le meme :

on commence par :

1. Importer le modèle d'apprentissage
2. Instanciez-le
3. Ajustez-le aux données et faites des prédictions
4. Évaluer le modèle ajusté

```
In [58]: %time
         from lightgbm import LGBMClassifier
         # Train a model
         model = LGBMClassifier(random_state=31)
         model.fit(X_train, y_train)

         # Make predictions
         y_pred = model.predict(X_test)

         # Check score
         accuracy_score(y_test, y_pred)

         Wall time: 0 ns

Out[58]: 0.9842304457527334
```

Ce modèle donne un score de 98% sur les données de test, et le temps d’exécution est presque nul d’ordre de nano-secondes .

Ce modèle a deux avantages par rapport aux autres modèles :

1. **Il donne une meilleure performance**
2. **Il est très rapide en terme de temps d’exécution**