TP2: Pandas, data analysis library

Kim ANTUNEZ, Isabelle BERNARD (Group : Mr Denis)

1 Predicting cancellation: Part I – visualization

Question 1. Propose a solution that will re-order the barplot above using standard month ordering. Hint: use `pd.Categorical()` function of pandas.

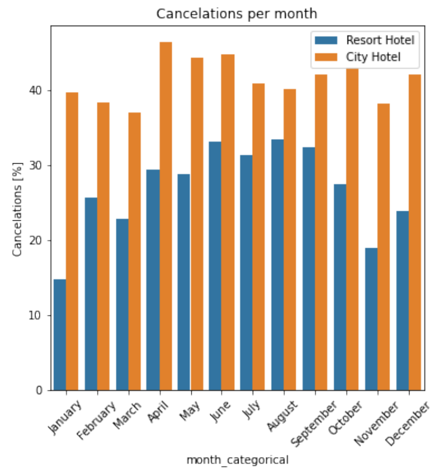
We have to consider the `month` variable as a categorical and ordered variable (as a factor). The following lines allow to do so. And then we just plot the new `month_categorical` variable using the previous code.

```
# create a vector of ordered months
months_ordered_from_july = data['arrival_date_month'].unique()
months_ordered_from_january = [months_ordered_from_july[i] for i in [*range(6, 12), *range(0, 6)]]
print(months_ordered_from_january)

#create a new 'month' variable which is categorical
data_visual["month_categorical"] = pd.Categorical(data_visual["month"], ordered=True,
                    categories=months_ordered_from_january)
```

```
['January', 'February', 'March', 'April', 'May', 'June',
  'July', 'August', 'September', 'October', 'November',
  'December']
```

```
plt.figure(figsize=(6, 6))
sns.barplot(x = "month_categorical", y = "percent_cancel",
hue="hotel", hue_order = ["Resort Hotel",
"City Hotel"], data=data_visual)
plt.title("Cancelations per month")
plt.xticks(rotation=45)
plt.ylabel("Cancelations [%]")
plt.legend()
plt.show()
```



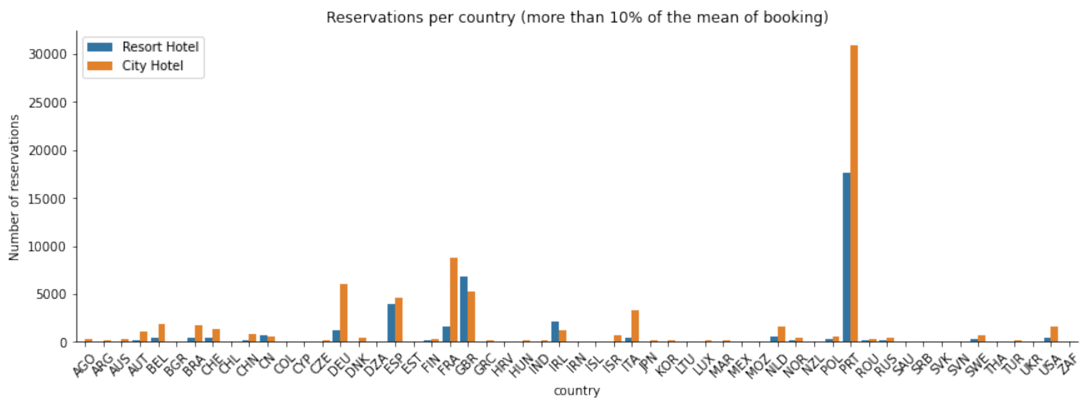
Question 2. Provide interpretation of the above plot.

The cancelations at the City hotel are, in proportion, higher than in the Resort Hotel. For the Resort hotel, the cancelations are in proportion higher in Summer and Spring than in Winter and Fall. On the contrary, this seasonality is less clear for the City hotel. We only observe pics in April-May-June, then in September-October, and finally in December.

Question 3. What is the most and the second most common country of origin for reservations of each hotel?

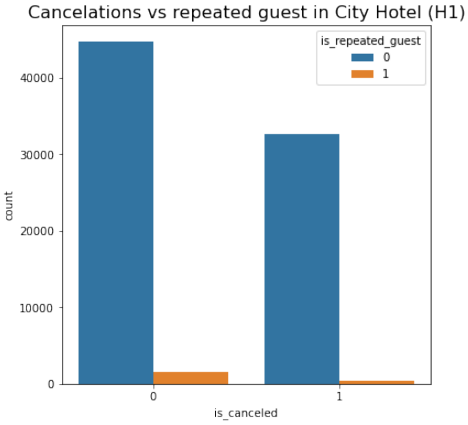
- **Resort Hotel** : the first most common country of origin for reservations is Portugal (PRT) and the second one is Great Britain (GBR)
- **City Hotel** : the first most common country of origin for reservations is Portugal (PRT) and the second one is France (FRA)

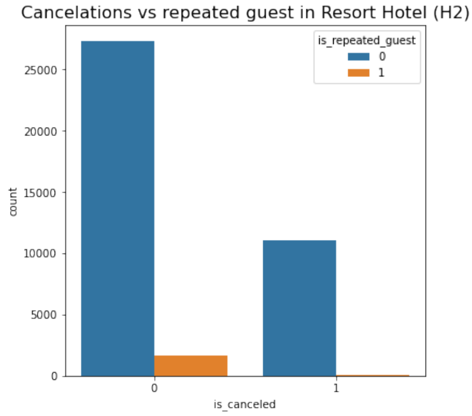
See the plot below and the associated code in Annex A.



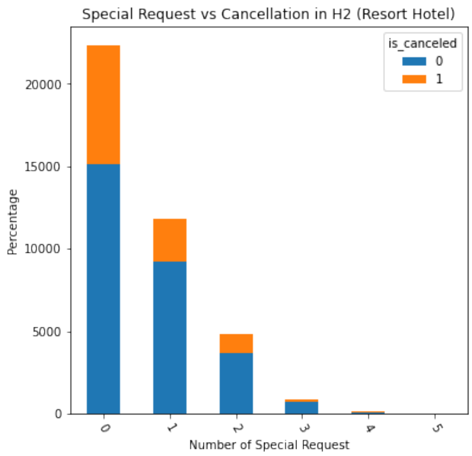
> **Question 4.** Plot the number of cancelations for repeated and not repeated guests for both hotels.

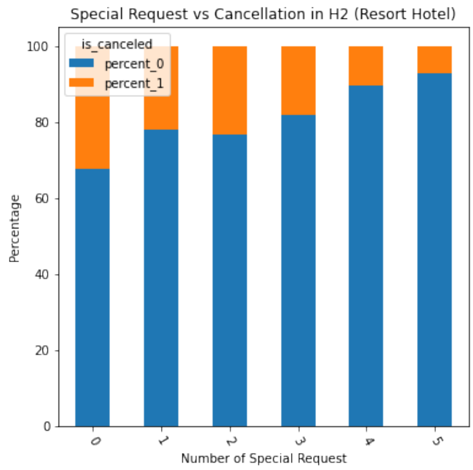
See the 2 following plots and the associated codes in Annex A.





> **Question 5.** Make the same plot for Resort Hotel. Make your conclusions.





Most of the reservations in the Resort Hotel have no special request and the cancelation in this case is almost of 32 %. However, when special requests are made, the cancelation rate is lower (22 % when 1 special request is made, 23 % when 2 special requests are made, 18 % when 3 special request is made, 11 % when 4 special request is made, 7 % when 5 special request is made), but this decrease with the number of special requests is lower than with the City Hotel.

See the associated code in Annex A.

2 Predicting cancellations: Part II – ML

> **Question 1:** What is `OneHotEncoder()`? Why do we use it in our case?

One-Hot-Encoding is a way (among others such as label encoding) to convert categorical values into numerical values because most of the ML algorithms work better with numerical inputs. In this strategy, each category value is converted into a new column and assigned a 1 or 0 (notation for true/false) value to the column. Let's consider the previous example of the names of the hotels (first column) with one-hot encoding (2 last columns).

	hotel	0	1
0	Resort Hotel	0	1
1	Resort Hotel	0	1
2	Resort Hotel	0	1
3	Resort Hotel	0	1
4	Resort Hotel	0	1

We use it in our case because **many variables are categorical** : hotel, arrival_date_month, customer_type, company, deposit_type. . .

Source : see here.

Question 2: In the previous example we again encounter the convergence problem. Of course we can set higher number of iterations, but it is time consuming. As you have seen, proper normalization can resolve the issue. Insert a normalization step in the pipeline. Note that we do not want to normalize the categorical data, it simply does not make sense. Be careful to normalize only the numerical data. Did it resolve the warning?

To normalize the numeric features of the data, we modify the `numeric_transformer`.

```
#numeric_transformer = SimpleImputer(strategy="constant", fill_value=0) # to deal with missing num data
```

We change it into the following pipeline :

```
from sklearn.preprocessing import StandardScaler
numeric_transformer = Pipeline(steps=[
                                    ("imputer", SimpleImputer(strategy="constant", fill_value=0)),
                                    ("scaler", StandardScaler())]) #NEW : rescale the data !
```

And it solved the warning! See the execution of the code in Annex B.

Question 3: As we can see, previous code uses only logistic regression. Modify the above code inserting your favorite ML method.

Below, you can see the same actions but using the **SVM** method instead of logistic regressions. The main differences are:

- This time we use the following scaler : `MinMaxScaler()`
- We use `LinearSVC()` instead of `LogisticRegression()`
- We test the parameter `svc__C` instead of `logreg__C`

```
from sklearn.svm import LinearSVC
from sklearn.preprocessing import MinMaxScaler
numeric_transformer = Pipeline(steps=[
                                    ("imputer", SimpleImputer(strategy="constant", fill_value=0)),
                                    ("scaler", MinMaxScaler())]) #NEW : MinMaxScaler
preproc = ColumnTransformer(transformers=[("num", numeric_transformer, numeric_features),
                                          ("cat", categorical_transformer, categorical_features)])
models = [("logreg", LogisticRegression(max_iter=500)),
          ("svc", LinearSVC(max_iter=700))]
grids = {"logreg" : {'logreg__C': np.logspace(-2, 2, 5, base=2)},
         "svc" : {'svc__C': np.logspace(-2, 2, 5, base=2)}}
for name, model in models:
    pipe = Pipeline(steps=[('preprocessor', preproc), (name, model)])
    clf = GridSearchCV(pipe, grids[name], cv=3)
    clf.fit(X, y)
    print('Results for {}'.format(name))
    #print(clf.cv_results_)
    display(pd.DataFrame(clf.cv_results_)[['params','mean_test_score', 'std_test_score',
                                            'rank_test_score']])
```

```
Results for logreg
```

	params	mean_test_score	std_test_score	rank_test_score
0	{'logreg__C': 0.25}	0.73992	0.0330775	1
1	{'logreg__C': 0.5}	0.739283	0.0336877	2
2	{'logreg__C': 1.0}	0.734693	0.0391517	3
3	{'logreg__C': 2.0}	0.731133	0.0435457	4
4	{'logreg__C': 4.0}	0.72785	0.0476724	5

```
Results for svc
```

	params	mean_test_score	std_test_score	rank_test_score
0	{'svc__C': 0.25}	0.740933	0.0362662	1
1	{'svc__C': 0.5}	0.738647	0.0392476	2
2	{'svc__C': 1.0}	0.737424	0.0408303	3
3	{'svc__C': 2.0}	0.735489	0.0434047	4
4	{'svc__C': 4.0}	0.734057	0.0453227	5

➔ **The results of SVM seem to be a bit better than the ones for the logistic regression : the mean test scores are globally higher.**

3 The homework: Part III

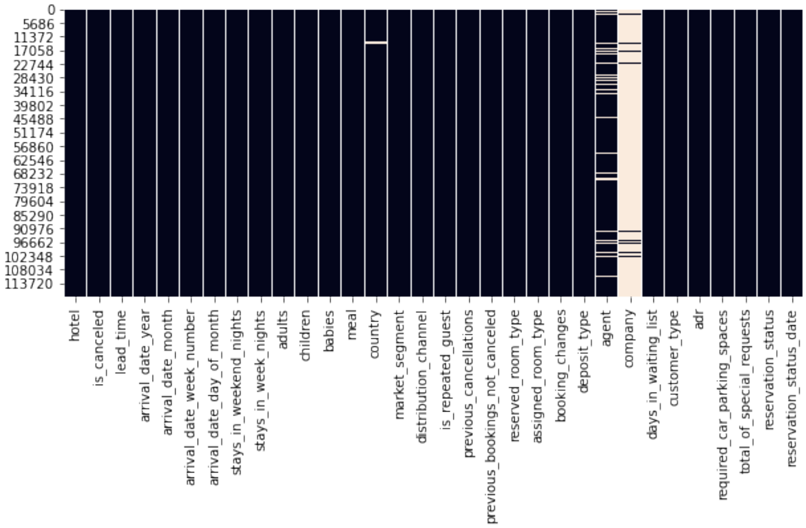
3.1 FIRST STEP : choose the relevant features and build the dataset

See in Annex E the whole code for Part III.

We first **delete** from the dataset **4 features** that cannot be observed at the moment when the SMS is sent :`is_canceled`, `assigned_room_type`, `reservation_status` and `reservation_status_date`. See in Annex C for more details about these variables.

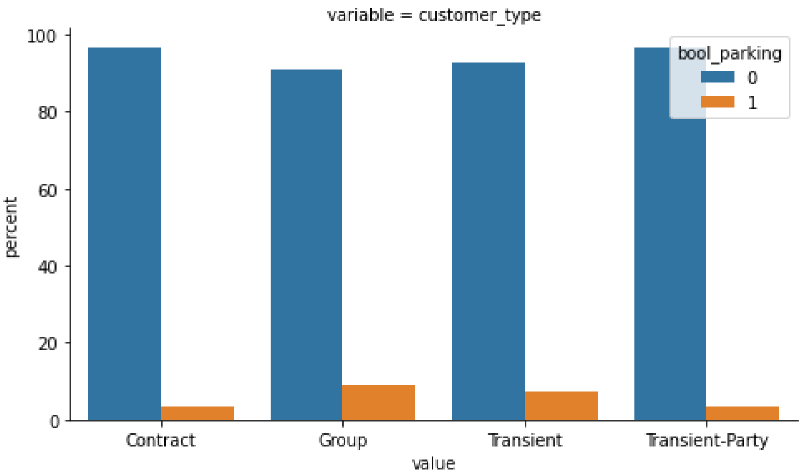
Then, we look at the features containing lots of missing data (see figure on the right). The feature "company" displays a lot of missing data. Therefore we don't use this feature in our predicting models.

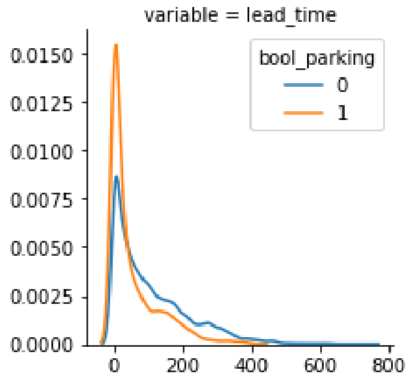
Finally, we create the data.frame we need by creating a binary variable called `bool_parking` whose value is 1 if `required_car_parking_spaces` > 0



3.2 SECOND STEP : Exploratory Data analysis

Now, we do some plots to see what kind of variables seem to be important to predict the outcome `bool_parking`. Two examples of the plots are put below. The rest of them are available in Annex D.





These plots make us draw some first conclusions about the interesting predictors of the outcome. People more susceptible to use a parking seem to be people :

- coming in groups and transient travellers : guests who are predominantly on-the-move and seek short hotel-stays (`customer_type = Group` or `Transient`)
- going to the Resort Hotel (`hotel = Resort Hotel`) ➔ It may mean that we should do 2 different predictive models, one for each hotel
- who book directly at the hotel (`market_segment = direct`)
- who book a certain kind of room (`reserved_room_type = H`) but for anonymity reasons we don't know what it represents
- coming in the summer (`arrival_date_week_number` around 25)
- who don't book in advance (`lead_time` close to 0)
- rich (`adr` is `high`)
- who have children (`children` is `high`)

3.3 THIRD STEP : Modelisation

Now that the data-processing is done, we apply some ML models to predict the variable `bool_parking`.

3.3.1 Models including all the variables

After preparing, the test and training set, we run the models (SVM and logistic regressions) using Cross-validation. Finally, we run the models with the best parameters and evaluate them with different scores. Below are the results.

```
Results for logreg                                     Results for svc
Returned hyperparameter: {'logreg__C': 0.25}           Returned hyperparameter: {'svc__C': 0.25}
Best classification accuracy in train is: 0.9374483437555715Best classification accuracy in train is: 0.9374483482455481
Classification accuracy on test is: 0.9391584025730367      Classification accuracy on test is: 0.9391584025730367
```

	params	mean_test_score	std_test_score	rank_test_score	params	mean_test_score	std_test_score	rank_test_score
0	{'logreg__C': 0.25}	0.937996	1.53278e-05	0 2	{'svc__C': 0.25}	0.93804	1.48151e-05	1
1	{'logreg__C': 0.5}	0.938018	9.78925e-07	1 1	{'svc__C': 0.5}	0.93804	1.48151e-05	1
2	{'logreg__C': 1.0}	0.937996	3.20887e-05	2 3	{'svc__C': 1.0}	0.93804	1.48151e-05	1
3	{'logreg__C': 2.0}	0.937996	3.20887e-05	3 3	{'svc__C': 2.0}	0.93804	1.48151e-05	1
4	{'logreg__C': 4.0}	0.937984	4.78791e-05	4 5	{'svc__C': 4.0}	0.93804	1.48151e-05	1

```
MSE logreg:  0.25110316775715696                       MSE svc:  0.25210185856501033
R2 logreg:  -0.08354580290869595                       R2 svc:  -0.06839376846142531
accuracy logreg :  0.936947199142321                   accuracy svc :  0.9364446529080676
```

➔ Both SVM and LogisticRegression WITHOUT selection of features present really performent results. The accuracy is nearly 94%. Indeed, the number of individuals is +100 000 (n) and the number of features (p) is around 30. Therefore, we have n >> p which prevents from overfitting

3.3.2 Models with best predictive features chosen MANUALLY

Now we try to pick up **MANUALLY** features with high predicting powers to see if it can reduce over-fitting and show better results. We filter columns with the 8 pre-selected variables (4 numeric features : "arrival_date_week_number","lead_time", "adr","children" and 4 categorical features "customer_type","hotel", "market_segment", "reserved_room_type") and we do the same process as before.

```
Results for logreg                                     Results for svc
Returned hyperparameter: {'logreg__C': 2.0}            Returned hyperparameter: {'svc__C': 0.25}
Best classification accuracy in train is: 0.9379620748954963Best classification accuracy in train is: 0.9379844099095207
Classification accuracy on test is: 0.9375502546234253      Classification accuracy on test is: 0.9375502546234253
```

	params	mean_test_score	std_test_score	rank_test_score	params	mean_test_score	std_test_score	rank_test_score
0	{'logreg__C': 0.25}	0.937783	2.82109e-05	0 1	{'svc__C': 0.25}	0.937795	1.53264e-05	1
1	{'logreg__C': 0.5}	0.937772	1.63076e-05	1 3	{'svc__C': 0.5}	0.937795	1.53264e-05	1
2	{'logreg__C': 1.0}	0.937783	2.82109e-05	2 1	{'svc__C': 1.0}	0.937795	1.53264e-05	1
3	{'logreg__C': 2.0}	0.937772	4.25343e-05	3 4	{'svc__C': 2.0}	0.937795	1.53264e-05	1
4	{'logreg__C': 4.0}	0.937772	4.25343e-05	4 4	{'svc__C': 4.0}	0.937795	1.53264e-05	1

```
MSE logreg:  0.246864551270153                         MSE svc:  0.246864551270153
R2 logreg:  -0.06660949113779324                       R2 svc:  -0.06489707089086316
accuracy logreg :  0.939057893326186                   accuracy svc :  0.939057893326186
```

➔ Both SVM and LogisticRegression WITH MANUAL selection of features still present really good results. Even if the different scores are a bit less promising than the ones of the previous method, they are still very good. For example, the accuracy is here still nearly 94%.

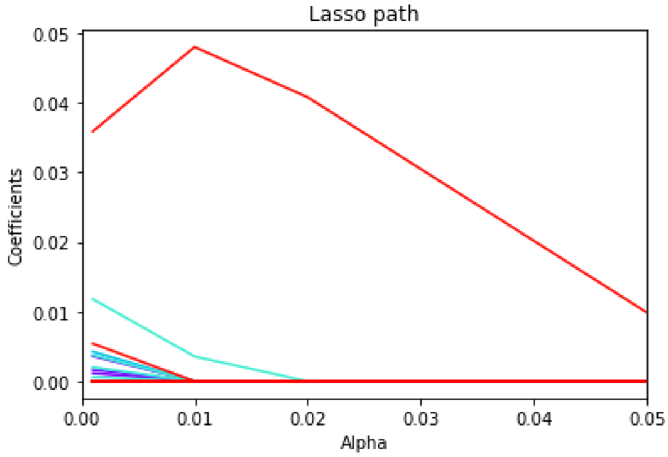
3.3.3 Models with best predictive features chosen with REGULARIZED REGRESSIONS

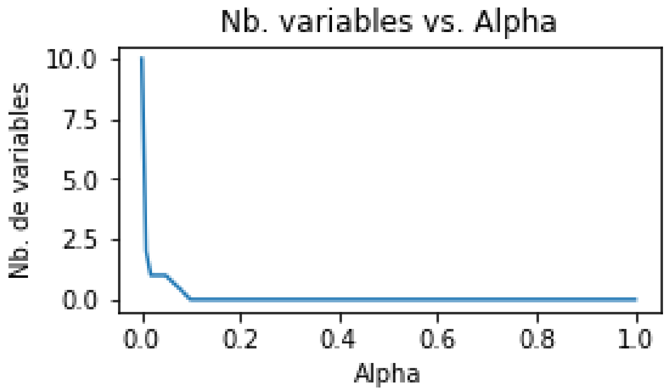
We try to improve the models using a polynomial regression with a regularization called LASSO. This method only selects the features that best predict the outcome. Indeed, as we explained earlier, subsetting the features may prevent from overfitting. LASSO regression is such as $\hat{f} = \underset{f \in \mathcal{F}_n^{\text{poly}}}{argmin}\{\frac{1}{2m}\sum_{i=1}^{m}(Y_i - f(X_i))^2 + \alpha\sum_{k=1}^{n}|a_k|\}$ where $\alpha > 0$ is a parameter to chose.

- If α is too large, all the coefficients of the regression equal to zero
- If α is too small and close to 0, we get the coefficients of the usual linear regression.

Thus, we must find a compromise, by choosing alpha by cross-validation.

The "lasso path" can also help us. This is a graph that relates the value of α to the estimated coefficients.





The best alpha is the lowest (0.001). It is logical because, as we said earlier, the number of features is low (p=30) compared to the number of individuals (n=+100 000). Therefore, regularized regressions are not necessary in our case. Finally, we run the lasso with the best α and evaluate it with different scores.

```
MSE lasso:  0.23160623528346608
R2 lasso:  -0.06815575882526304
```

➔ We can see that the MSE obtained on the test set (0.23) is lower (and thus better) than the one of the previous models (0.25), which means that the regularization may have improved a little bit the model !

A Code of Part I

A.1 Question 3

```
# Lets create a database following the same model than before but using countries instead of months
#First Hotel
n2_reserv_H1 = data.loc[(data["hotel"] == "Resort Hotel")].groupby("country")["hotel"].count()
data2_visualH1 = pd.DataFrame({"hotel": "Resort Hotel",
                                "country": list(n2_reserv_H1.index),
                                "n_booking": list(n2_reserv_H1.values)})
data2_visualH1.sort_values(by=['n_booking'], ascending=False)
```

	hotel	country	n_booking
95	Resort Hotel	PRT	17630
45	Resort Hotel	GBR	6814
40	Resort Hotel	ESP	3957
55	Resort Hotel	IRL	2166
44	Resort Hotel	FRA	1611

```
#Second Hotel
n2_reserv_H2 = data.loc[(data["hotel"] == "City Hotel")].groupby("country")["hotel"].count()
data2_visualH2 = pd.DataFrame({"hotel": "City Hotel",
                                "country": list(n2_reserv_H2.index),
                                "n_booking": list(n2_reserv_H2.values)})
data2_visualH2.sort_values(by=['n_booking'], ascending=False)
```

	hotel	country	n_booking
125	City Hotel	PRT	30960
50	City Hotel	FRA	8804
39	City Hotel	DEU	6084
53	City Hotel	GBR	5315
46	City Hotel	ESP	4611

```
data2_visual = pd.concat([data2_visualH1, data2_visualH2], ignore_index=True)
data2_visual = data2_visual.sort_values(by=['country'], ascending=True)

plt.figure(figsize=(6, 6))
#sns.barplot(x = "country", y = "n_booking" , hue="hotel",
#            hue_order = ["Resort Hotel", "City Hotel"], data=data2_visual)
sns.catplot(x = "country", y = "n_booking" , hue="hotel", hue_order = ["Resort Hotel", "City Hotel"],
            data=data2_visual[data2_visual["n_booking"]>(0.1*np.mean(data2_visual["n_booking"]))],
            legend_out=False, kind="bar", height = 4, aspect=3)
plt.title("Reservations per country (more than 10% of the mean of booking)")
plt.xticks(rotation=45)
plt.ylabel("Number of reservations")
plt.legend()
plt.show()
```

A.2 Question 4

```
# The same only with City Hotel (H1)
plt.figure(figsize=(6, 6))
sns.countplot(x="is_canceled", hue='is_repeated_guest', data=data[(data['hotel'] == 'City Hotel')])
plt.title("Cancelations vs repeated guest in City Hotel (H1)", fontsize=16)
plt.plot()
```

```
# The same only with Resort Hotel (H2)
plt.figure(figsize=(6, 6))
sns.countplot(x="is_canceled", hue='is_repeated_guest', data=data[(data['hotel'] == 'Resort Hotel')])
plt.title("Cancelations vs repeated guest in Resort Hotel (H2)", fontsize=16)
plt.plot()
```

A.3 Question 5

```
data_req2 = data[(data['hotel'] == 'Resort Hotel')].groupby(['total_of_special_requests',
                                                              'is_canceled']).size().unstack(level=1)
data_req2.plot(kind='bar', stacked=True, figsize=(6,6))
plt.title('Special Request vs Cancellation in H2 (Resort Hotel)')
plt.xlabel('Number of Special Request', fontsize=10)
plt.xticks(rotation=300)
plt.ylabel('Percentage', fontsize=10)
```

```
# From raw value to percentage
total = [i+j for i,j in zip(data_req2[0], data_req2[1])]
data_req2['percent_0'] = [i / j * 100 for i,j in zip(data_req2[0], total)]
data_req2['percent_1'] = [i / j * 100 for i,j in zip(data_req2[1], total)]
data_req2.iloc[:, 2:4]
```

total_of_special_requests	0	1	percent_0	percent_1
0	15145	7216	67.7295	32.2705
1	9209	2597	78.0027	21.9973
2	3700	1127	76.6522	23.3478
3	744	166	81.7582	18.2418
4	127	15	89.4366	10.5634
5	13	1	92.8571	7.14286

```
data_req2.iloc[:, 2:4].plot(kind='bar', stacked=True, figsize=(6,6))
plt.title('Special Request vs Cancellation in H1 (Resort Hotel)')
plt.xlabel('Number of Special Request', fontsize=10)
plt.xticks(rotation=300)
plt.ylabel('Percentage', fontsize=10)
```

B Code of Part II

B.1 Question 1

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
# creating instance of one-hot-encoder
enc = OneHotEncoder(handle_unknown='ignore')
# passing hotel-types-cat column (label encoded values of bridge_types)
enc_df = pd.DataFrame(enc.fit_transform(data[['hotel']]).toarray())
# merge with main dataframe on key values
hotel_df = data.iloc[:, 0:1].join(enc_df)
hotel_df.head()
```

B.2 Question 2

```
from sklearn.preprocessing import StandardScaler
#numeric_transformer = SimpleImputer(strategy="constant", fill_value=0) # to deal with missing numeric dat
numeric_transformer = Pipeline(steps=[
                                    ("imputer", SimpleImputer(strategy="constant", fill_value=0)),
                                    ("scaler", StandardScaler())]) #NEW : rescale the data !
preproc = ColumnTransformer(transformers=[("num", numeric_transformer, numeric_features),
                                          ("cat", categorical_transformer, categorical_features)])
```

```
models = [("logreg", LogisticRegression(max_iter=500))]
grids = {"logreg" : {'logreg__C': np.logspace(-2, 2, 5, base=2)}}
for name, model in models:
    pipe = Pipeline(steps=[('preprocessor', preproc), (name, model)])
    clf = GridSearchCV(pipe, grids[name], cv=3)
    clf.fit(X, y)
    print('Results for {}'.format(name))
    print(clf.cv_results_)
```

```
Results for logreg
{'mean_fit_time': array([1.07707651, 1.06985847, 1.26302139, 0.94357912, 1.00347384]),
'std_fit_time': array([0.22217166, 0.14027325, 0.44600286, 0.12203947, 0.05911086]),
'mean_score_time': array([0.0600067 , 0.0611678 , 0.05684272, 0.0552206 ,
0.06061093]), 'std_score_time': array([0.00130701, 0.0016936 , 0.00354926, 0.00281719,
0.00184487]), 'param_logreg__C': masked_array(data=[0.25, 0.5, 1.0, 2.0, 4.0],
             mask=[False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'logreg__C': 0.25}, {'logreg__C': 0.5},
            {'logreg__C': 1.0}, {'logreg__C': 2.0}, {'logreg__C': 4.0}],
            'split0_test_score': array([0.70128402, 0.7000779 , 0.68552906,
            0.67786517, 0.66600498]), 'split1_test_score': array([0.78262181,
            0.78244591, 0.78232028, 0.78221977, 0.78219464]),
            'split2_test_score': array([0.73585285, 0.73582772, 0.73582772,
            0.736079  , 0.736079  ]), 'mean_test_score': array([0.73991956,
            0.73945051, 0.73455902, 0.73205464, 0.72809287]),
            'std_test_score': array([0.03333029, 0.03372404, 0.03952503,
            0.04269752, 0.04776919]),
            'rank_test_score': array([1, 2, 3, 4, 5])}
```

C Variables observed when sending the SMS

The variables deleted from the dataset because they cannot be observed at the moment when the SMS is sent are in a strikethrough font below with their definitions.

-**hotel** : Hotel (H1 = Resort Hotel or H2 = City Hotel)

~~-**is_canceled** : Value indicating if the booking was canceled (1) or not (0)~~

-**lead_time** : Number of days that elapsed between the entering date of the booking into the PMS and the arrival date

-**arrival_date_year** : Year of arrival date

-**arrival_date_month** : Month of arrival date

-**arrival_date_week_number** : Week number of year for arrival date

-**arrival_date_day_of_month** : Day of arrival date

-**stays_in_weekend_nights** : Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel

-**stays_in_week_nights** : Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel

-**adults** : Number of adults

-**children** : Number of children

-**babies** : Number of babies

-**meal** : Type of meal booked. Categories are presented in standard hospitality meal packages: Undefined/SC – no meal package; BB – Bed & Breakfast; HB – Half board (breakfast and one other meal – usually dinner); FB – Full board (breakfast, lunch and dinner)

-**country** : Country of origin. Categories are represented in the ISO 3155–3:2013 format

-**market_segment** : Market segment designation. In categories, the term “TA” means “Travel Agents” and “TO” means “Tour Operators”

-**distribution_channel** : Booking distribution channel. The term “TA” means “Travel Agents” and “TO” means “Tour Operators”

-**is_repeated_guest** : Value indicating if the booking name was from a repeated guest (1) or not (0)

-**previous_cancellations** : Number of previous bookings that were cancelled by the customer prior to the current booking

-**previous_bookings_not_canceled** : Number of previous bookings not cancelled by the customer prior to the current booking

-**reserved_room_type** : Code of room type reserved. Code is presented instead of designation for anonymity reasons.

~~-**assigned_room_type** : Code for the type of room assigned to the booking. Sometimes the assigned room type differs from the reserved room type due to hotel operation reasons (e.g. overbooking) or by customer request. Code is presented instead of designation for anonymity reasons.~~

-**booking_changes** : Number of changes/amendments made to the booking from the moment the booking was entered on the PMS

-**deposit_type** : Indication on if the customer made a deposit to guarantee the booking. This variable can assume three categories: No Deposit – no deposit was made; Non Refund – a deposit was made in the value of the total stay cost; Refundable – a deposit was made with a value under the total cost of stay.

-**agent** : ID of the travel agency that made the booking

-**company** : ID of the company/entity that made the booking or responsible for paying the booking. ID is presented instead of designation for anonymity reasons

-**days_in_waiting_list** : Number of days the booking was in the waiting list before it was confirmed to the customer

-**customer_type** : Type of booking, assuming one of four categories: Contract - when the booking has an allotment or other type of contract associated to it; Group – when the booking is associated to a group; Transient – when the booking is not part of a group or contract, and is not associated to other transient booking; Transient-party – when the booking is transient, but is associated to at least other transient booking

-**adr** : Average Daily Rate as defined by dividing the sum of all lodging transactions by the total number of staying nights

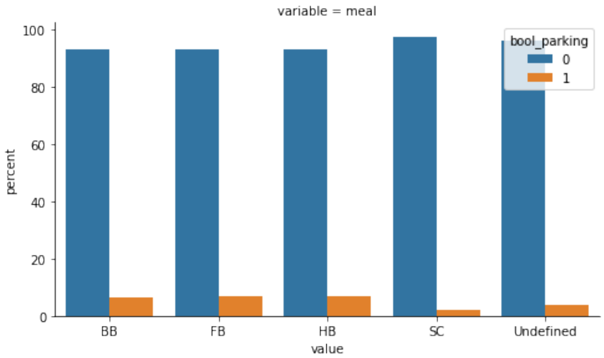
-**required_car_parking_spaces** : Number of car parking spaces required by the customer

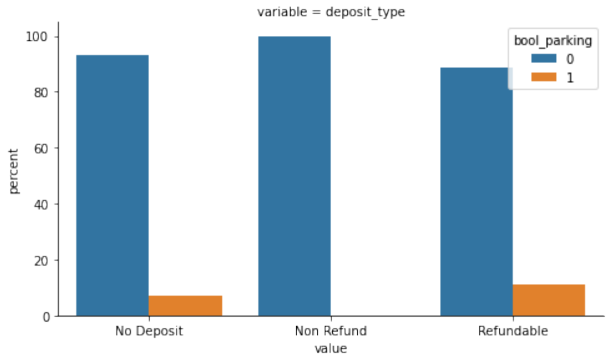
-**total_of_special_requests** : Number of special requests made by the customer (e.g. twin bed or high floor)

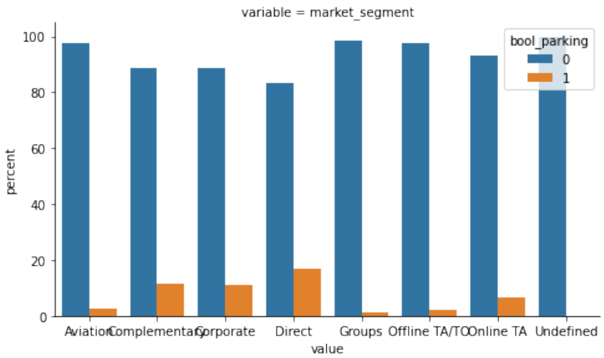
~~-**reservation_status** : Reservation last status, assuming one of three categories: Canceled – booking was canceled by the customer; Check-Out – customer has checked in but already departed; No-Show – customer did not check-in and did inform the hotel of the reason why~~

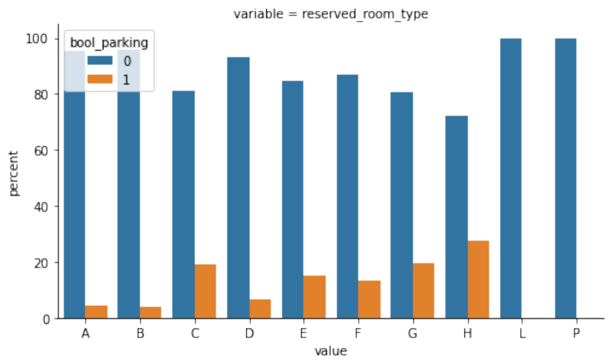
~~-**reservation_status_date** : Date at which the last status was set. This variable can be used in conjunction with the ReservationStatus~~

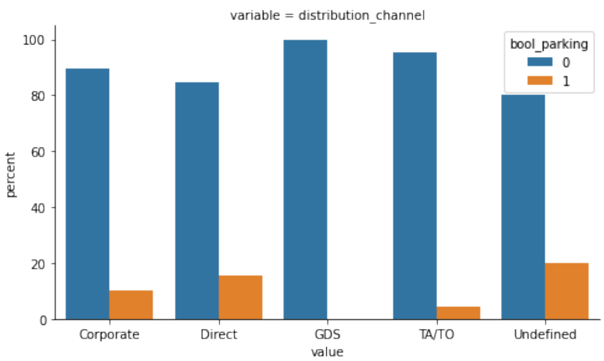
D Plots of Exploratory Data analysis (Part III)

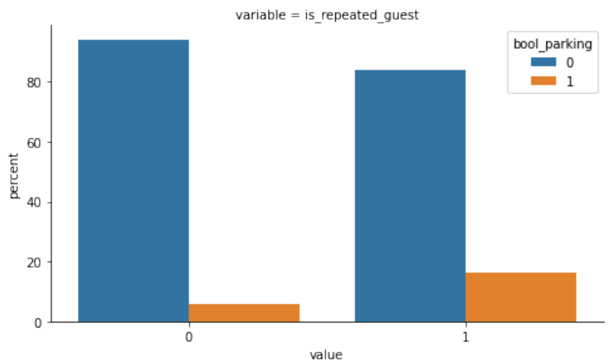


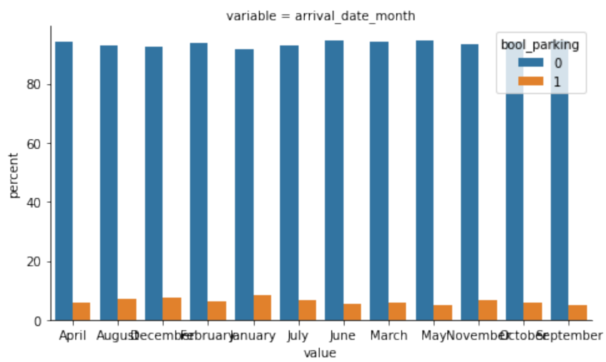


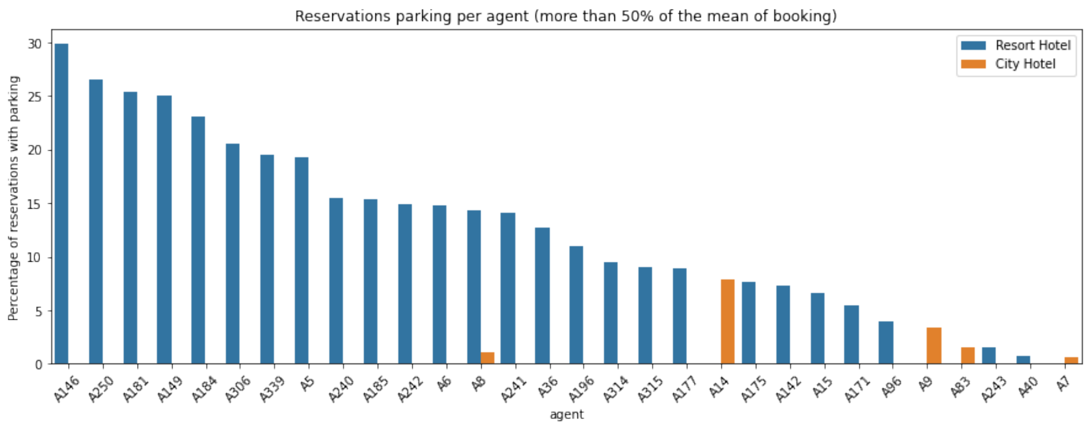


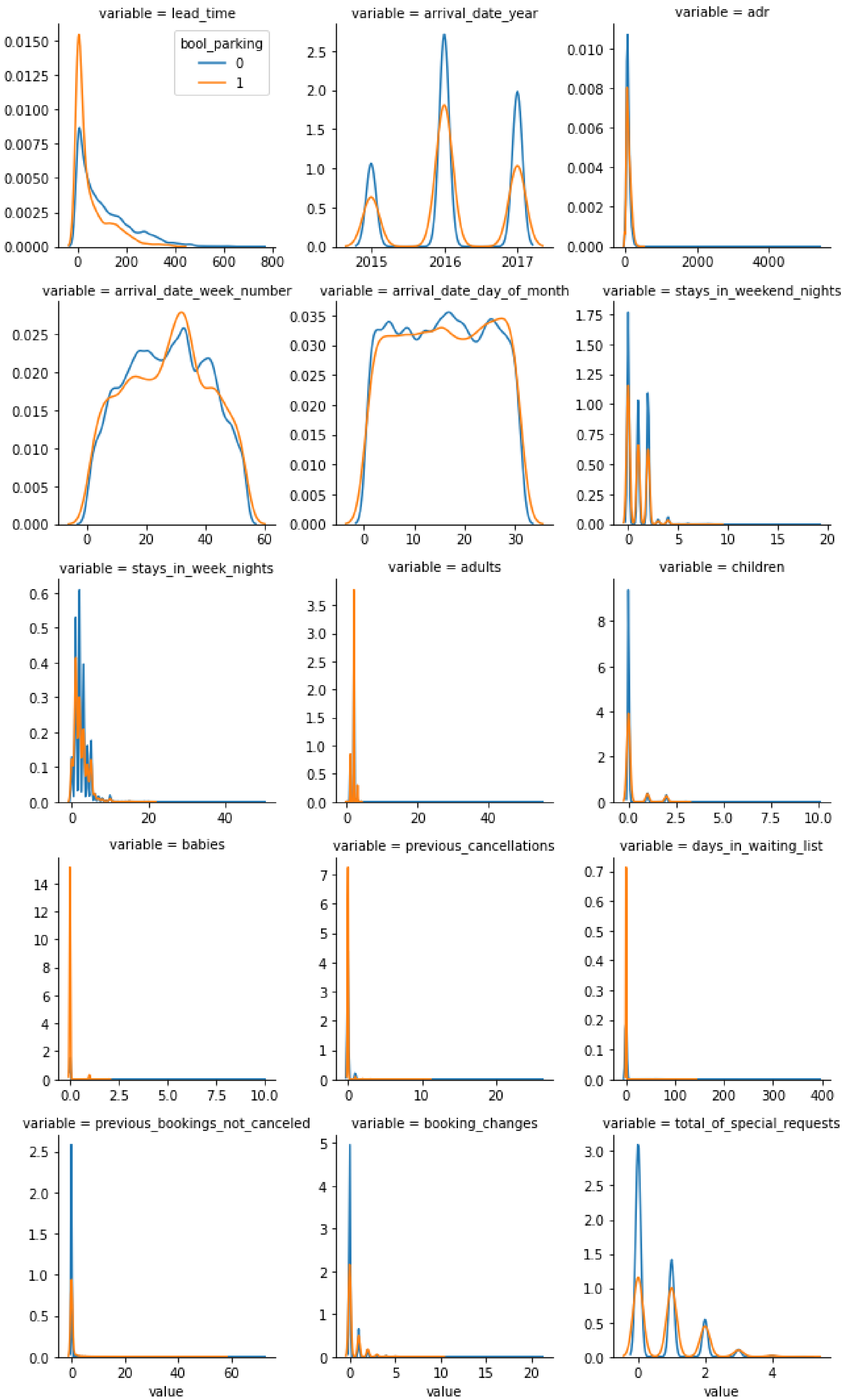











variable = lead_time
bool_parking
0
1
variable = arrival_date_year
variable = adr
variable = arrival_date_week_number
variable = arrival_date_day_of_month
variable = stays_in_weekend_nights
variable = stays_in_week_nights
variable = adults
variable = children
variable = babies
variable = previous_cancellations
variable = days_in_waiting_list
variable = previous_bookings_not_canceled
variable = booking_changes
variable = total_of_special_requests
value
value
value

E Code of part III

```
#import packages for part III
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.svm import LinearSVC
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

E.1 FIRST STEP : choose the relevant features and build the dataset

E.1.1 Remove features with too many missing data

```
#Check Missing Data
plt.figure(figsize=(10,4))
sns.heatmap(data.isna(),cbar=False)
```

E.1.2 Creation of dataset

```
data2 = data.copy()

# New variable : boolean of value 1 if required_car_parking_spaces > 0
data2["bool_parking"] = data2["required_car_parking_spaces"] > 0
data2["bool_parking"] = data2["bool_parking"].astype(int)

#We modify agent which is a strange variable (mix of numeric and categorical) by adding the prefix "A"
data2["agent"] = data2["agent"].map(lambda x:x if np.isnan(x) else ("A" + str(int(x))))

#We distinguish numerical and categorical features
numeric_features = ["lead_time", "arrival_date_year",
                    "adr",
                    "arrival_date_week_number",
                    "arrival_date_day_of_month", "stays_in_weekend_nights", "stays_in_week_nights",
                    "adults", "children",
                    "babies", "previous_cancellations",
                    "days_in_waiting_list", "previous_bookings_not_canceled", "booking_changes",
                    "total_of_special_requests"]
categorical_features = ["meal", "country", "market_segment", "distribution_channel",
                        "hotel", "reserved_room_type", "deposit_type","agent",
                       "customer_type","is_repeated_guest","arrival_date_month"]
features = numeric_features + categorical_features
```

E.2 SECOND STEP : Exploratory Data analysis

```
#------ For executing the code Javascript IPython in JupyterLab -----------
%matplotlib inline
#----------------------------------------------------------
############# categorical variables
variables = categorical_features.copy()
```

```
variables.append("bool_parking")
variables.pop(7) #variable agent not very intersting to plot (to many of them)
variables.pop(1) #variable country not very intersting to plot (to many of them)
df = data2.filter(variables)
df = pd.melt(df, df.columns[-1], df.columns[:-1])
#df.head()
df = df.groupby(['variable','value'])['bool_parking'].value_counts(normalize=True)
df = df.mul(100)
df = df.rename('percent').reset_index()

for vExpl in list(variables):
    if vExpl != 'bool_parking':
        data_graph = df[(df['variable'] == vExpl)]
        g = sns.catplot(x="value", y="percent", hue="bool_parking", col="variable", data=data_graph,
              col_wrap=3, kind="bar", sharex=False, sharey=True,legend_out=False, height=4,aspect=1.6)
```

```
#-----------------------------------------------------------------------------------------
#Agent : Percentage of reservations with parking for agent with more than 50% mean of bookings
#        (exclude percentage with few reservations)
#-----------------------------------------------------------------------------------------
n_reserv_byagent_H1 = data2.loc[
    (data["hotel"] == "Resort Hotel")].groupby("agent")["hotel"].count()
n_bparking_byagent_H1 = data2.loc[
    (data2["hotel"] == "Resort Hotel")].groupby("agent")["bool_parking"].sum()
data_agent_visualH1 = pd.DataFrame({"hotel": "Resort Hotel",
                                "agent": list(n_reserv_byagent_H1.index),
                                "n_booking": list(n_reserv_byagent_H1.values),
                                "n_bool_parking": list(n_bparking_byagent_H1.values)})
data_agent_visualH1.sort_values(by=['n_bool_parking'], ascending=False)
n_reserv_byagent_H2 = data2.loc[
    (data["hotel"] == "City Hotel")].groupby("agent")["hotel"].count()
n_bparking_byagent_H2 = data2.loc[
    (data2["hotel"] == "City Hotel")].groupby("agent")["bool_parking"].sum()
data_agent_visualH2 = pd.DataFrame({"hotel": "City Hotel",
                                "agent": list(n_reserv_byagent_H2.index),
                                "n_booking": list(n_reserv_byagent_H2.values),
                                "n_bool_parking": list(n_bparking_byagent_H2.values)})
data_agent_visualH2.sort_values(by=['n_bool_parking'], ascending=False)
data_agent_visual = pd.concat([data_agent_visualH1, data_agent_visualH2], ignore_index=True)
data_agent_visual["percent_bParking"] = data_agent_visual[
    "n_bool_parking"] / data_agent_visual["n_booking"] * 100 # percent of parking
data_agent_visual = data_agent_visual.sort_values(by=['percent_bParking'], ascending=False)
plt.figure(figsize=(15, 5))
sns.barplot(x = "agent", y = "percent_bParking" , hue="hotel",
            hue_order = ["Resort Hotel", "City Hotel"],
            data=data_agent_visual[
                data_agent_visual["n_bool_parking"]>0.5*np.mean(data_agent_visual["n_bool_parking"])])
plt.title("Reservations parking per agent (more than 50% of the mean of booking)")
plt.xticks(rotation=45)
plt.ylabel("Percentage of reservations with parking")
plt.legend()
plt.show()
```

```
############# numeric variables
sns.distributions._has_statsmodels = False
variables = numeric_features.copy()
variables.append("bool_parking")
df = data2.filter(variables)
df = pd.melt(df, df.columns[-1], df.columns[:-1])
g = sns.FacetGrid(df, col="variable", hue="bool_parking",
                  col_wrap=3,sharex=False, sharey=False,legend_out=False)
g.map(sns.kdeplot, "value") #, shade=True
```

```
g.add_legend()
```

E.3 THIRD STEP : Modelisation

E.3.1 Models including all the variables

Preparing the test and training sets

```
#-----------------------------------------------------------------------------
# Feature Engineering : Process of converting raw data into a structured format
# Making the data ready to use for model training (transformers)
# Creating the test and training sets
#-----------------------------------------------------------------------------
numeric_transformer = Pipeline(steps=[
                                    ("imputer", SimpleImputer(strategy="constant", fill_value=0)),
                                    ("scaler", MinMaxScaler())])
categorical_transformer = Pipeline(steps=[
                                    ("imputer", SimpleImputer(strategy="constant",
                                    fill_value="Not defined")),
                                    ("onehot", OneHotEncoder(handle_unknown='ignore'))]) #
preproc = ColumnTransformer(transformers=[("num", numeric_transformer, numeric_features),
                                          ("cat", categorical_transformer, categorical_features)])

X = data2.drop(["is_canceled","assigned_room_type","reservation_status",
"reservation_status_date","company"],
              axis=1)[features]
y = data2["bool_parking"]

X, y = shuffle(X,y)
X_train, X_test, y_train, y_test = train_test_split(X, y)

X_preproc = preproc.fit_transform(X)
X_train_preproc, X_test_preproc, y_train_preproc, y_test_preproc = train_test_split(X_preproc, y)
```

Running the models using Cross-validation

```
models = [("logreg", LogisticRegression(max_iter=700)),
          ("svc", LinearSVC(max_iter=800))]
grids = {"logreg" : {'logreg__C': np.logspace(-2, 2, 5, base=2)},
         "svc" : {'svc__C': np.logspace(-2, 2, 5, base=2)}}
X_train, X_test, y_train, y_test = train_test_split(X, y)

for name, model in models:
    pipe = Pipeline(steps=[('preprocessor', preproc), (name, model)])
    clf = GridSearchCV(pipe, grids[name], cv=3)
    clf.fit(X_train, y_train)
    print('Results for {}'.format(name))
    print('Returned hyperparameter: {}'.format(clf.best_params_))
    print('Best classification accuracy in train is: {}'.format(clf.best_score_))
    print('Classification accuracy on test is: {}'.format(clf.score(X_test, y_test)))
    display(pd.DataFrame(clf.cv_results_)[['params','mean_test_score', 'std_test_score',
                                           'rank_test_score']])
```

Running the models with the best parameters and evaluating with different scores

```
logreg = LogisticRegression(max_iter=700, C=0.25) #best parameter logreg : C=0.25
logreg.fit(X_train_preproc,y_train_preproc)
y_pred = logreg.predict(X_test_preproc)
print("MSE logreg: ", mean_squared_error(y_test_preproc, y_pred, squared=False))
print("R2 logreg: ", r2_score(y_test, y_pred))
print("accuracy logreg : ", accuracy_score(y_test_preproc, y_pred)) #OK because Y_pred is binary
```

```
print("\n")

svc = LinearSVC(max_iter=800, C=0.25) #best parameter svc : C=0.25
svc.fit(X_train_preproc,y_train_preproc)
y_pred = svc.predict(X_test_preproc)
print("MSE svc: ", mean_squared_error(y_test_preproc, y_pred, squared=False))
print("R2 svc: ", r2_score(y_test_preproc, y_pred))
print("accuracy svc : ", accuracy_score(y_test_preproc, y_pred)) #OK because Y_pred is binary
```

E.3.2 Models with best predictive features chosen MANUALLY

Preparing the test and training sets

```
dataHW = data2.copy()

#-----------------------------------------------------------------------------
# Feature Selection : Picking up the most predictive features
#-----------------------------------------------------------------------------
numeric_features = ["arrival_date_week_number","lead_time", "adr","children"]
categorical_features = ["customer_type","hotel", "market_segment", "reserved_room_type"]
features = numeric_features + categorical_features

#-----------------------------------------------------------------------------
# Feature Engineering : Process of converting raw data into a structured format
# Making the data ready to use for model training (transformers)
# Creating the test and training sets
#-----------------------------------------------------------------------------
numeric_transformer = Pipeline(steps=[
                                    ("imputer", SimpleImputer(strategy="constant", fill_value=0)),
                                    ("scaler", MinMaxScaler())])
categorical_transformer = Pipeline(steps=[
                                    ("imputer", SimpleImputer(strategy="constant",
                                    fill_value="Not defined")),
                                    ("onehot", OneHotEncoder(handle_unknown='ignore'))])
preproc = ColumnTransformer(transformers=[("num", numeric_transformer, numeric_features),
                                          ("cat", categorical_transformer, categorical_features)])

X = dataHW[features]
y = dataHW["bool_parking"]

X, y = shuffle(X,y)
X_train, X_test, y_train, y_test = train_test_split(X, y)

X_preproc = preproc.fit_transform(X)
X_train_preproc, X_test_preproc, y_train_preproc, y_test_preproc = train_test_split(X_preproc, y)
```

Running the models using Cross-validation

```
models = [("logreg", LogisticRegression(max_iter=700)),
          ("svc", LinearSVC(max_iter=800))]
grids = {"logreg" : {'logreg__C': np.logspace(-2, 2, 5, base=2)},
         "svc" : {'svc__C': np.logspace(-2, 2, 5, base=2)}}
for name, model in models:
    pipe = Pipeline(steps=[('preprocessor', preproc), (name, model)])
    clf = GridSearchCV(pipe, grids[name], cv=3)
    clf.fit(X_train, y_train)
    print('Results for {}'.format(name))
    print('Returned hyperparameter: {}'.format(clf.best_params_))
    print('Best classification accuracy in train is: {}'.format(clf.best_score_))
    print('Classification accuracy on test is: {}'.format(clf.score(X_test, y_test)))
```

```
    display(pd.DataFrame(clf.cv_results_)[['params','mean_test_score', 'std_test_score',
                                           'rank_test_score']])
```

Running the models with the best parameters and evaluating with different scores

```
logreg = LogisticRegression(max_iter=700, C=0.25) #best parameter logreg : C=0.25
logreg.fit(X_train_preproc,y_train_preproc)
y_pred = logreg.predict(X_test_preproc)
print("MSE logreg: ", mean_squared_error(y_test_preproc, y_pred, squared=False))
print("R2 logreg: ", r2_score(y_test, y_pred))
print("accuracy logreg : ", accuracy_score(y_test_preproc, y_pred)) #OK because Y_pred is binary

print("\n")

svc = LinearSVC(max_iter=800, C=0.25) #best parameter svc : C=0.25
svc.fit(X_train_preproc,y_train_preproc)
y_pred = svc.predict(X_test_preproc)
print("MSE svc: ", mean_squared_error(y_test_preproc, y_pred, squared=False))
print("R2 svc: ", r2_score(y_test_preproc, y_pred))
print("accuracy svc : ", accuracy_score(y_test_preproc, y_pred)) #OK because Y_pred is binary
```

E.3.3 Models with best predictive features chosen with REGULARIZED REGRESSIONS

Preparing the test and training sets

```
#--------------------------------------------------------------------------------
# Feature Selection : Picking up the most predictive features
#--------------------------------------------------------------------------------
#We distinguish numerical and categorical features
numeric_features = ["lead_time", "arrival_date_year",
                    "adr",
                    "arrival_date_week_number",
                    "arrival_date_day_of_month", "stays_in_weekend_nights", "stays_in_week_nights",
                    "adults", "children",
                    "babies", "previous_cancellations",
                    "days_in_waiting_list", "previous_bookings_not_canceled", "booking_changes",
                    "total_of_special_requests"]
categorical_features = ["meal", "country", "market_segment", "distribution_channel",
                        "hotel", "reserved_room_type", "deposit_type","agent",
                       "customer_type","is_repeated_guest","arrival_date_month"]
features = numeric_features + categorical_features

#--------------------------------------------------------------------------------
# Feature Engineering : Process of converting raw data into a structured format
# Making the data ready to use for model training (transformers)
# Creating the test and training sets
#--------------------------------------------------------------------------------

numeric_transformer = Pipeline(steps=[
                                    ("imputer", SimpleImputer(strategy="constant", fill_value=0)),
                                    ("scaler", MinMaxScaler())])
categorical_transformer = Pipeline(steps=[
                                    ("imputer", SimpleImputer(strategy="constant",
                                    fill_value="Not defined")),
                                    ("onehot", OneHotEncoder(handle_unknown='ignore'))]) #
preproc = ColumnTransformer(transformers=[("num", numeric_transformer, numeric_features),
                                          ("cat", categorical_transformer, categorical_features)])

X = data2.drop(["is_canceled","assigned_room_type","reservation_status",
"reservation_status_date", "company"],
              axis=1)[features]
```

```
y = data2["bool_parking"]

X, y = shuffle(X,y)
X_train, X_test, y_train, y_test = train_test_split(X, y)

X_preproc = preproc.fit_transform(X)
X_train_preproc, X_test_preproc, y_train_preproc, y_test_preproc = train_test_split(X_preproc, y)
```

Running the model using Cross-validation

```
#lasso path
from sklearn.linear_model import lasso_path
my_alphas = np.array([0.001,0.01,0.02,0.025,0.05,0.1,0.25,0.5,0.8,1.0])
alpha_for_path, coefs_lasso, _ = lasso_path(X_train_preproc,y,alphas=my_alphas)
nb_coeff = coefs_lasso.shape[0] #578
nb_alpha = coefs_lasso.shape[1]
import matplotlib.cm as cm
couleurs = cm.rainbow(np.linspace(0,1,nb_coeff))

#lasso path plot(one curve per variable)
plt.figure(figsize=(3, 3))
fig, ax1 = plt.subplots()
for i in range(nb_coeff):
    ax1.plot(alpha_for_path,coefs_lasso[i,:],c=couleurs[i])
plt.xlabel('Alpha')
plt.xlim(0,0.05)
plt.ylabel('Coefficients')
plt.title('Lasso path')
plt.show()
```

```
#number of non-zero coefficient(s) for each alpha
nbNonZero = np.apply_along_axis(func1d=np.count_nonzero,arr=coefs_lasso,axis=0)
plt.figure(figsize=(4, 2))
plt.plot(alpha_for_path,nbNonZero)
plt.xlabel('Alpha')
plt.ylabel('Nb. de variables')
plt.title('Nb. variables vs. Alpha')
plt.show()
```

```
#function to create labels for one-hot vectors
# inspiration : https://stackoverflow.com/questions/41987743/merge-two-multiindex-levels-into-one-in-panda
def labels_one_hot(data, categorical, numeric):
    enc_cols=[]
    for var in categorical:
        uniq_val = X[var].unique().astype(str).tolist()
        indice_nan = [i for i,x in enumerate(uniq_val) if x == 'nan']
        for index in indice_nan:
            uniq_val[index] = "undefined"
        uniq_val = [var + "_" + sub for sub in uniq_val]
        enc_cols = enc_cols + uniq_val
    cols = numeric + enc_cols
    return(cols)
```

```
nom_var = labels_one_hot(X,categorical_features,numeric_features)
#print the non-zero coefficients for alpha = 0.001
coeff001 = pd.DataFrame({'Variables':nom_var,'Coefficients':coefs_lasso[:,9]}) #alpha = 0.001
coeff001[coeff001['Coefficients']>0]
```

```
#Choose the best alpha using cros-validation
from sklearn.linear_model import LassoCV
lcv = LassoCV(alphas=my_alphas,normalize=False,fit_intercept=False,random_state=0,cv=5)
```

```
lcv.fit(X_train_preproc,y_train_preproc)
#mean of MSE for each alpha
avg_mse = np.mean(lcv.mse_path_,axis=1)
#alphas and MSE
print(pd.DataFrame({'alpha':lcv.alphas_,'MSE':avg_mse}))
#best alpha
print("best alpha:", lcv.alpha_) #0.001
```

Running the models with the best alpha and evaluating with different scores

```
from sklearn.linear_model import Lasso
lasso = Lasso(fit_intercept=False,normalize=False, alpha=0.001) #best parameter lasso : alpha=0.001
lasso.fit(X_train_preproc,y_train_preproc)
#print(lasso.coef_)
y_pred = lasso.predict(X_test_preproc)
print("MSE lasso: ", mean_squared_error(y_test_preproc, y_pred, squared=False))
print("R2 lasso: ", r2_score(y_test, y_pred))
```