VIETNAM NATIONAL UNIVERSITY - HOCHIMINH CITY
INTERNATIONAL UNIVERSITY
School of Industrial and Engineering Management

DATA MINING IN SUPPLY CHAIN MANAGEMENT

Hotel Bookings Demand

Lecturer: M.Sc Ngo Thi Thao Uyen

No.	Student's name	Student's ID	Contribution
1	Lê Phong Công Thành	IELSIU19266	100%
2	Ngô Trọng Gia	IELSIU19142	100%
3	Nguyễn Hải Triều	IELSIU19293	100%

TABLE OF CONTENT

I. INTRODUCTION

1. Data-mining overview

Data Mining is defined as extracting information from huge sets of data. In other words, we can say that data mining is the procedure of mining knowledge from data. The information or knowledge extracted so can be used for any of the following applications
Data mining is a big area of data sciences, which tries to find patterns and features in data, frequently massive data sets. It encompasses techniques like regression, classification, clustering, anomaly detection, and others. In addition, it involves preprocessing, validating, summarizing, and finally interpreting the data sets.

In the discipline of chemometrics as well as the information technology sector, data mining has gained enormous popularity. This has happened at the same time that chemometrics has expanded into the larger field of chemoinformatics. The increasing availability of very huge volumes of data and the urgent need to transform such data into valuable information and knowledge make data mining vital. The study of information extraction from huge data sets and databases is known as data mining.

2. Problem overview

The hospitality sector is expanding as more and more individuals use their money to travel and engage in leisure activities. The demand for hotel rooms is not evenly spread throughout the year since people only stay in hotels during the holidays or special events. The hotel management frequently used a pricing plan, one of which involved increasing the room rate when demand was strong and implementing a promotion when demand was low. As a result, it is crucial for the pricing strategy to be able to predict future demand accurately. Forecasting becomes more difficult since multiple models may be needed for different consumer segments, whose demand may vary.

3. Data description

1. **hotel** : Hotel (Resort Hotel or City Hotel)
2. **is_canceled** : Value indicating if the booking was canceled (1) or not (0)
3. **lead_time** : Number of days that elapsed between the entering date of the booking into the PMS and the arrival date
4. **arrival_date_year** : Year of arrival date
5. **arrival_date_month** : Month of arrival date
6. **arrival_date_week_number** : Week number of year for arrival date
7. **arrival_date_day_of_month** : Day of arrival date
8. **stays_in_weekend_nights** : Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
9. **stays_in_week_nights** : Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel
10. **adults** : Number of adults
11. **children** : Number of children

12. **babies** : Number of babies
13. **meal** : Type of meal booked. Categories are presented in standard hospitality meal packages:
    a. Undefined/SC – no meal package
    b. BB – Bed & Breakfast
    c. HB – Half board (breakfast and one other meal – usually dinner)
    d. FB – Full board (breakfast, lunch and dinner)
14. **country** : Country of origin. Categories are represented in the ISO 3155–3:2013 format
15. **market_segment** : Market segment designation. In categories, the term "TA" means "Travel Agents" and "TO" means "Tour Operators"
16. **distribution_channel** : Booking distribution channel. The term "TA" means "Travel Agents" and "TO" means "Tour Operators"
17. **is_repeated_guest** : Value indicating if the booking name was from a repeated guest (1) or not (0)
18. **previous_cancellations** : Number of previous bookings that were canceled by the customer prior to the current booking
19. **previous_bookings_not_canceled** : Number of previous bookings not canceled by the customer prior to the current booking
20. **reserved_room_type** : Code of room type reserved. Code is presented instead of designation for anonymity reasons.
21. **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.
22. **booking_changes** : Number of changes/amendments made to the booking from the moment the booking was entered on the PMS until the moment of check-in or cancellation
23. **deposit_type** : Indication on if the customer made a deposit to guarantee the booking. This variable can assume three categories:
    a. No Deposit – no deposit was made
    b. Non Refund * a deposit was made in the value of the total stay cost
    c. Refundable – a deposit was made with a value under the total cost of stay.
24. **agent** : ID of the travel agency that made the booking
25. **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
26. **days_in_waiting_list** : Number of days the booking was in the waiting list before it was confirmed to the customer
27. **customer_type** : Type of booking, assuming one of four categories:
    a. Contract - when the booking has an allotment or other type of contract associated to it
    b. Group – when the booking is associated to a group

   c. Transient – when the booking is not part of a group or contract, and is not associated to other transient booking
   d. Transient-party – when the booking is transient, but is associated to at least other transient booking

28. **adr** : Average Daily Rate as defined by dividing the sum of all lodging transactions by the total number of staying nights
29. **required_car_parking_spaces** : Number of car parking spaces required by the customer
30. **total_of_special_requests** : Number of special requests made by the customer (e.g. twin bed or high floor)
31. **reservation_status** : Reservation last status, assuming one of three categories:
    a. Canceled – booking was canceled by the customer
    b. Check-Out – customer has checked in but already departed
    c. No-Show – customer did not check-in and did inform the hotel of the reason why
32. **reservation_status_date** : Date at which the last status was set. This variable can be used in conjunction with the *ReservationStatus* to understand when was the booking canceled or when did the customer checked-out of the hotel

II. Mathematical model

1. Processing

Data must satisfy data quality, including accuracy, completeness, consistency, timeliness, plausibility, and interpretability before they are mined. We go through the main steps involved in data preprocessing, namely data cleaning, data integration, data reduction and data transformation.

- **Data cleaning** routines attempt to fill in missing values, smooth out noise while identifying outliers,correct inconsistencies, dealing with skewness in the data

- To filling in the missing values, we following methods
  1) Ignore the tuple
  2) Fill in the missing value manually
  3) Use a global constant to fill in the missing value
  4) Use a measure of central tendency for the attribute (e.g., the mean or median) to fill in the missing value
  5) Use the attribute mean or median for all samples belonging to the same class as the given tuple
  6) Use the most probable value to fill in the missing value
- Noise is a random error or variance in a measured variable. We can “smooth” out the data to remove the noise following data smoothing techniques.
  1) Binning
  2) Regression:
  3) Outlier analysis

- **Data integration** combines data from multiple sources including multiple databases, data cubes, or flat files to form a coherent data store. The resolution of semantic heterogeneity, metadata, correlation analysis, tuple duplication detection, and data conflict detection contribute to smooth data integration.
- **Data reduction** techniques obtain a reduced representation of the data while minimizing the loss of information content. These include methods of dimensionality reduction, numerosity reduction, data compression and removing variables

- **Principal Components Analysis (PCA)** is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set. The basic procedure is as follows:
  1) The input data are normalized, so that each attribute falls within the same range. This step helps ensure that attributes with large domains will not dominate attributes with smaller domains.
  2) PCA computes k orthonormal vectors that provide a basis for the normalized input data. These are unit vectors that each point in a direction perpendicular to the others. These vectors are referred to as the principal components. The input data are a linear combination of the principal components

3) The principal components are sorted in order of decreasing “significance” or strength. The principal components essentially serve as a new set of axes for the data providing important information about variance. That is, the sorted axes are such that the first axis shows the most variance among the data, the second axis shows the next highest variance, and so on.
4) Because the components are sorted in decreasing order of “significance,” the data size can be reduced by eliminating the weaker components, that is, those with low variance. Using the strongest principal components, it should be possible to reconstruct a good approximation of the original data.

- **Data transformation** routines convert the data into appropriate forms for mining. . Strategies for data transformation include the following

1) **Smoothing**, which works to remove noise from the data. Techniques include binning, regression, and clustering
2) **Attribute construction** (or feature construction), where new attributes are constructed and added from the given set of attributes to help the mining process.
3) **Aggregation**, where summary or aggregation operations are applied to the data.
4) **Normalization**, where the attribute data are scaled so as to fall within a smaller range, such as -1.0 to 1.0, or 0.0 to 1.0
5) **Discretization**, where the raw values of a numeric attribute (e.g., age) are replaced by interval labels (e.g., 0–10, 11–20, etc.) or conceptual labels (e.g., youth, adult, senior). The labels, in turn, can be recursively organized into higher-level concepts, resulting in a concept hierarchy for the numeric attribute.
6) **Concept hierarchy generation for nominal data,** where attributes such as street can be generalized to higher-level concepts, like city or country. Many hierarchies for nominal attributes are implicit within the database schema and can be automatically defined at the schema definition level.

2. Supervised technique classifications

1) K-Nearest-Neighbor

Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by n attributes. Each tuple represents a point in an n-dimensional space. In this way, all the training tuples are stored in an n-dimensional pattern space. When given an unknown tuple, a k-nearest-neighbor classifier searches the pattern space for the k training tuples that are closest to the unknown tuple. These k training tuples are the k “nearest neighbors” of the unknown tuple.

“Closeness” is defined in terms of a distance metric, such as Euclidean distance. The Euclidean distance between two points or tuples, say

$$\boldsymbol{X_1} = (x_{11}, x_{12}, x_{12}, \ldots, x_{1n})$$

$$\boldsymbol{X_2} = (x_{21}, x_{22}, x_{22}, \ldots, x_{2n})$$

$$dist(\boldsymbol{X_1}, \boldsymbol{X_2}) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

2) Decision trees

Decision tree induction is a top-down recursive tree induction algorithm, which uses an attribute selection measure to select the attribute tested for each nonleaf node in the tree. Tree pruning algorithms attempt to improve accuracy by removing tree branches reflecting noise in the data. Early decision tree algorithms typically assume that the data are memory resident.

The algorithm is called with three parameters: D, attribute list, and Attribute selection method. We refer to D as a data partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter attribute list is a list of attributes describing the tuples. Attribute selection method specifies a heuristic procedure for selecting the attribute that "best" discriminates the given tuples according to class. This procedure employs an attribute selection measure such as information gain or the Gini index. Whether the tree is strictly binary is generally driven by the attribute selection measure. Some attribute selection measures, such as the Gini index, enforce the resulting tree to be binary. Others, like information gain, do not, therein allowing multiway splits (i.e., two or more branches to be grown from a node).

Algorithm: Generate decision tree. Generate a decision tree from the training tuples of data partition, *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

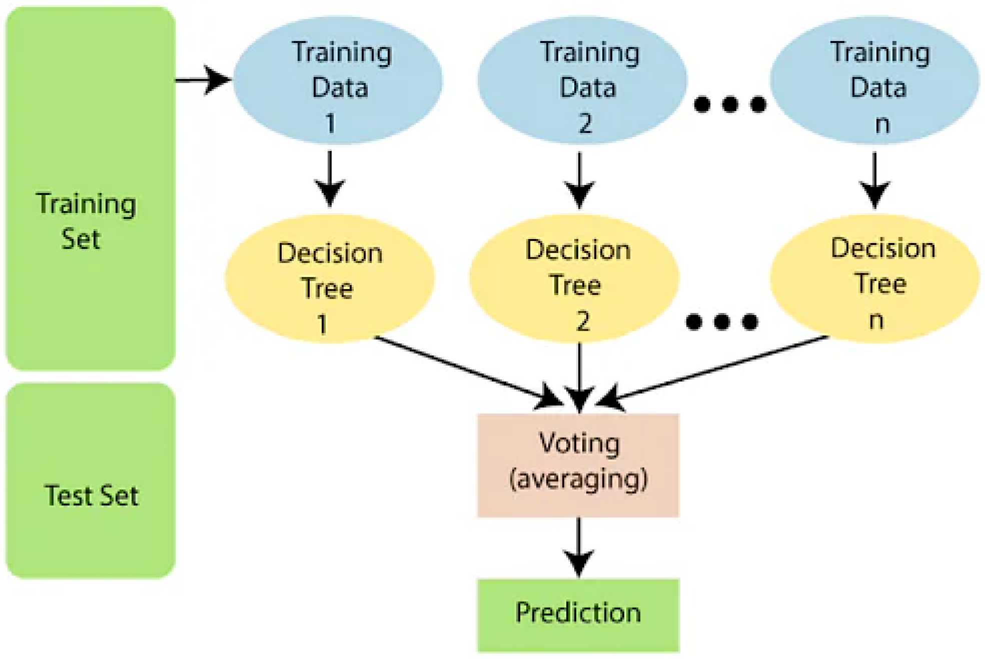
```
(1) create a node N;
(2) if tuples in D are all of the same class, C, then
(3)     return N as a leaf node labeled with the class C;
```

```
(4) if attribute_list is empty then
(5)     return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply Attribute selection method(D, attribute_list) to find the "best" splitting_criterion;
(7) label node N with splitting criterion;
(8) if splitting_attribute is discrete-valued and
multiway splits allowed then // not restricted to binary trees
(9)     attribute_list ← attribute_list - splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting criterion
// partition the tuples and grow subtrees for each partition
(11)    let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)    if Dj is empty then
(13)            attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate decision tree(Dj, attribute_list) to node N; endfor
(15) return N;
```

3) Random forest

Random forests can be built using bagging in tandem with random attribute selection. The following is the general process to construct k decision trees for the ensemble. A training set, Di, of d tuples is sampled with replacement from D for each iteration, i(i = 1, 2,..., k). In other words, each Di is a bootstrap sample of D, allowing some tuples to appear more than once in Di but excluding others. To determine the split at each node, let F be the amount of attributes to be used, where F is substantially less than the total number of attributes. Mi will randomly choose F attributes as candidates for the split at each node in order to build a decision tree classifier. The trees are grown using the CART approach. The trees are left to reach their full growth without being pruned. Forest-RI refers to random forests that are created in this manner using random input selection.

Forest-RC, a different kind of random forest, makes use of random linear combinations of the input attributes. It develops new attributes (or features) that are a linear combination of the current attributes rather than picking a subset of the attributes at random. That is, L, the number of original attributes to be combined, is used to construct an attribute. L characteristics are randomly chosen at each node and appended with coefficients that are uniform random values on the interval [1,1]. The optimal split is then found by searching over the resulting F linear combinations. To lessen the association between individual classifiers when there are few available attributes, this type of random forest is helpful.



4) Support vector machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The SVM algorithm's objective is to establish the best line or decision boundary that can divide n-dimensional space into classes, allowing us to quickly classify fresh data points in the future. A hyperplane is the name given to this optimal decision boundary.

The advantages of support vector machines are:

- Still useful in situations where the number of dimensions exceeds the number of samples.
- It is also memory efficient because it only uses a portion of the training points (known as support vectors) in the decision function.
- Different Kernel functions can be given for the decision function, making it versatile. There are common kernels available, but you can also define your own kernels.

The disadvantages of support vector machines are:

- Avoid over-fitting when selecting Kernel functions and regularization terms if the number of features is significantly more than the number of samples.
- Probability estimates are not directly provided by SVMs; instead, they are computed via an expensive five-fold cross-validation method.

SVM comes in two varieties:

1. Linear SVM: Linear SVM is used for data that can be divided into two classes using a single straight line. This type of data is called linearly separable data, and the classifier employed is known as a Linear SVM classifier.

2. Non-linear SVM: Non-Linear SVM is used for non-linearly separated data. If a dataset cannot be classified using a straight line, it is considered non-linear data, and the classifier employed is referred to as a Non-linear SVM classifier.\

The SVM algorithm uses support vectors and hyperplanes:

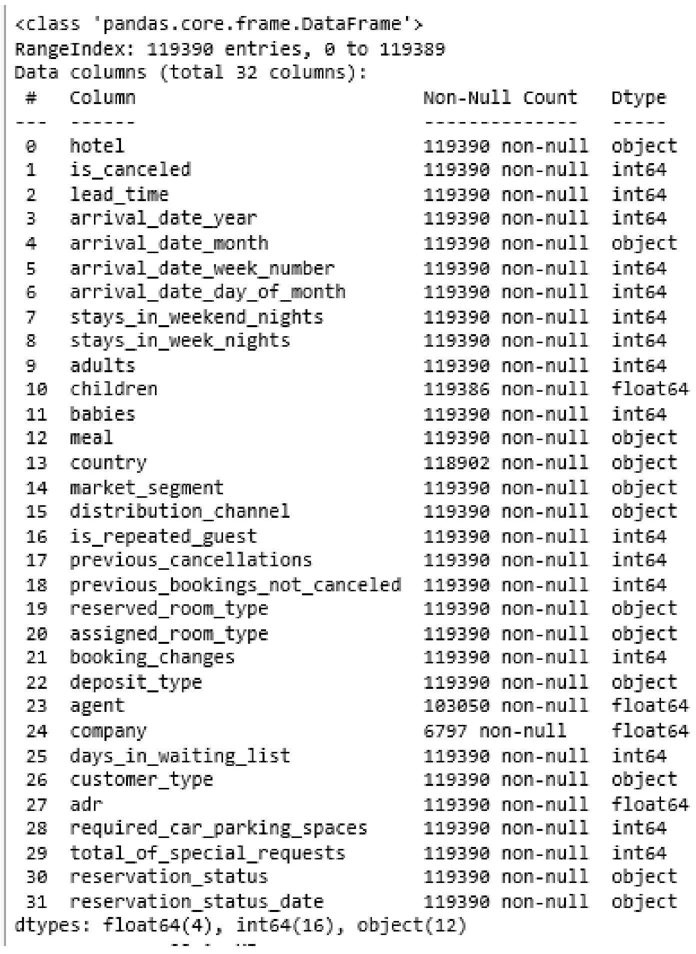
- Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.
- Support Vectors: Support vectors are the data points or vectors that are closest to the hyperplane and have the greatest influence on where the hyperplane is located. These vectors are called support vectors because they support the hyperplane.

III. Data Analysis and Classification Models

With the goal of classify whether customer will cancel booking or not , we need to determine the **'Label'**

1. Exploration Data Analysis

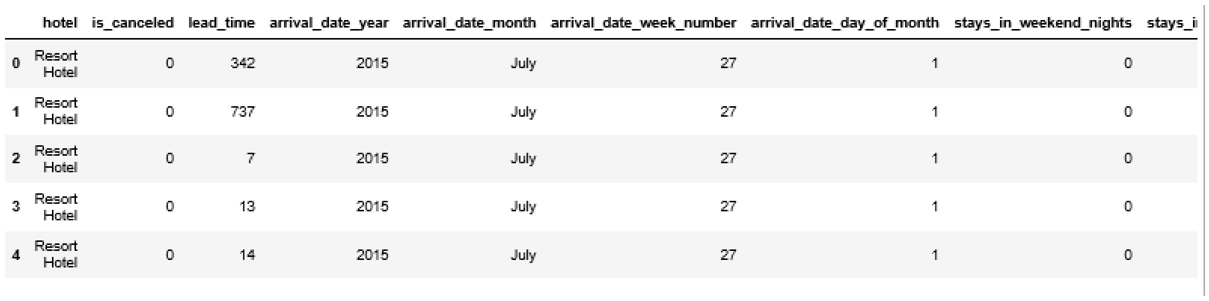
Step 1: Check the relationship between features for consistency



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

Figure: Data Information

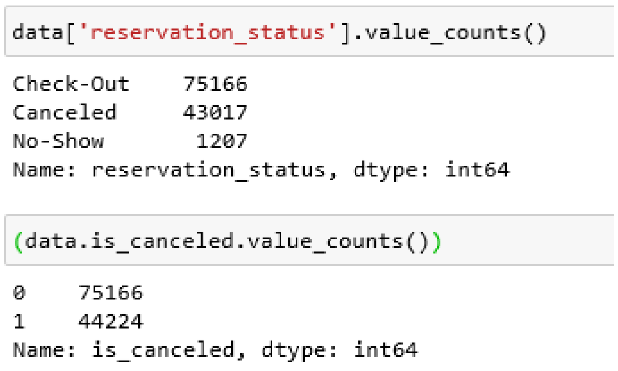
Dataset includes 119390 observations and 32 attributes. As of columns, there are 4 nulled columns: 'children','country','company','agent'.



	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_i
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	

Figure: The first five rows of data set

There are some important points to consider to understand potential relationships between columns whether these columns are dependable or not. **'is_canceled'** and **'reservation_status'** columns are needed to check the relationship between both.



```
data['reservation_status'].value_counts()
```

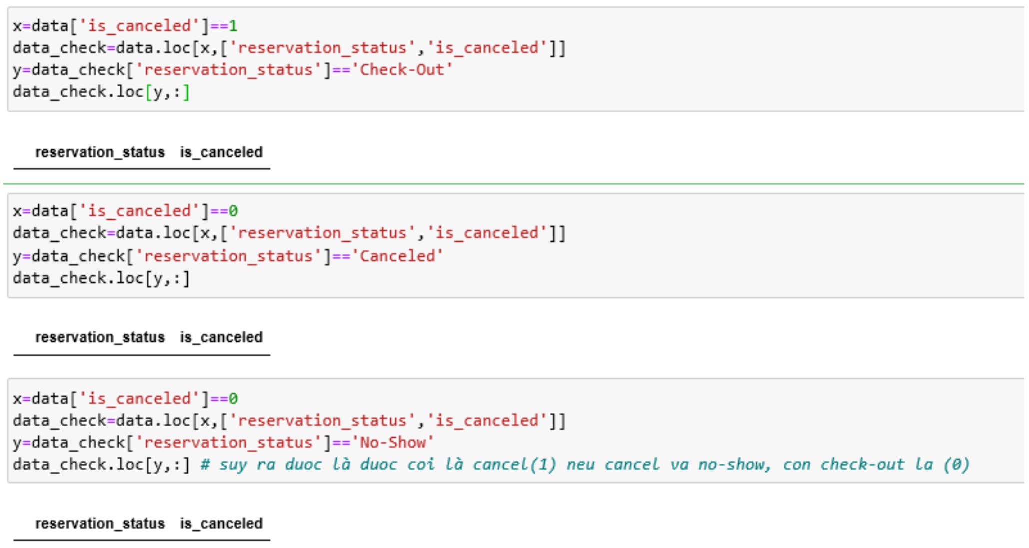
```
Check-Out    75166
Canceled     43017
No-Show       1207
Name: reservation_status, dtype: int64
```

```
(data.is_canceled.value_counts())
```

```
0    75166
1    44224
Name: is_canceled, dtype: int64
```

Figure: Values of 'reservation_status' and 'is_canceled'

The question issued is which types of value 'reservation_status' will return which values of 0 and 1 of 'is_canceled'. The check will be:



```
x=data['is_canceled']==1
data_check=data.loc[x,['reservation_status','is_canceled']]
y=data_check['reservation_status']=='Check-Out'
data_check.loc[y,:]
```

```
reservation_status  is_canceled
```

```
x=data['is_canceled']==0
data_check=data.loc[x,['reservation_status','is_canceled']]
y=data_check['reservation_status']=='Canceled'
data_check.loc[y,:]
```

```
reservation_status  is_canceled
```

```
x=data['is_canceled']==0
data_check=data.loc[x,['reservation_status','is_canceled']]
y=data_check['reservation_status']=='No-Show'
data_check.loc[y,:] # suy ra duoc là duoc coi là cancel(1) neu cancel va no-show, con check-out la (0)
```

```
reservation_status  is_canceled
```

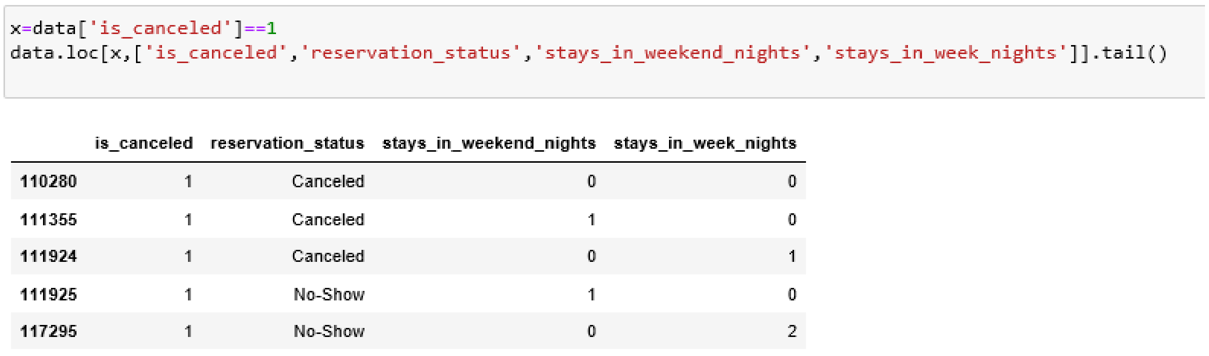
Figure: Check the relationship of 'is_canceled' and "reservation_status'

The results show that if ‘is_canceled’ = 1, ‘reservation_status’ = ‘No-Show’ and ‘Canceled’, and if ‘is_canceled’ = 0, ‘reservation_status’ = ‘Check-Out’ and no outliers. It’s easy to conclude that these 2 columns reflect the same information.

‘is_canceled’	‘reservation_status’
1	‘No-Show’ and ‘Canceled’
0	‘Check-Out’

Table: The relationship of ‘is_canceled’ and ‘reservation_status’

The column ‘is_canceled’ has to be Label, and the column ‘reservation_status’ has to be removed.
Another point needed to consider is that the relationship of ‘is_canceled’ and ‘stays_in_weekend_nights’ and ‘stays_in_week_nights’



```
x=data['is_canceled']==1
data.loc[x,['is_canceled','reservation_status','stays_in_weekend_nights','stays_in_week_nights']].tail()
```

	is_canceled	reservation_status	stays_in_weekend_nights	stays_in_week_nights
110280	1	Canceled	0	0
111355	1	Canceled	1	0
111924	1	Canceled	0	1
111925	1	No-Show	1	0
117295	1	No-Show	0	2

Figure: The relationship of ‘stays_in_weekend_nights’ and ‘stays_in_week_nights’

Although entry is canceled, there are still ‘stays_in_weekend_nights’ and ‘stays_in_week_nights’. The reason for this is that the data of ‘stays_in_weekend_nights’ and ‘stays_in_week_nights’ is recorded early by booking, ‘is_canceled’ and ‘reservation_status’ is recorded then on arrival date and canceled date.

Step 2: Data analysis for features

- For **‘lead_time’** column

```
data['lead_time'].describe()
sns.boxplot(y='lead_time',data=data) #việc đặt trước cá  năm không hẳn là
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2161273880>
```

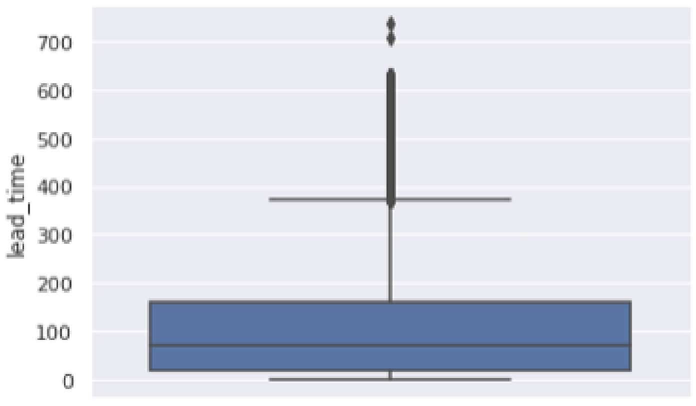
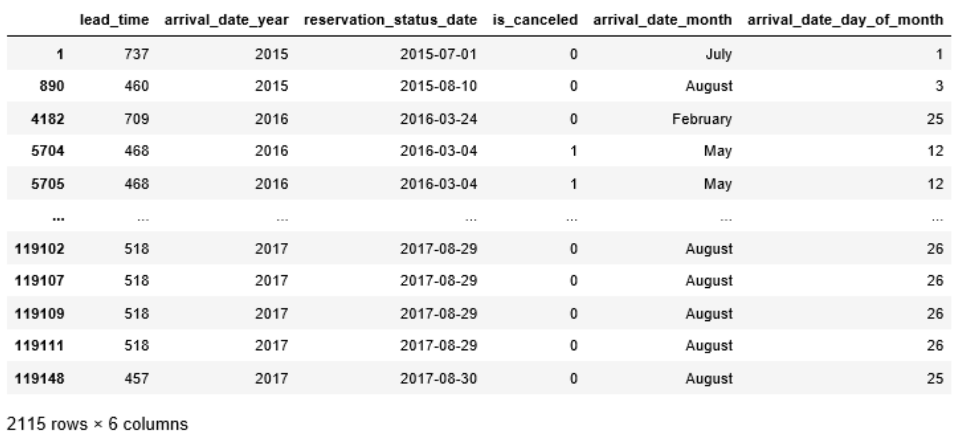


Figure : Boxplot for ‘lead_time’

There are many outliers in ‘lead_time’. Check to see whether outliers affect the decision of the customer.

```
x=data['lead_time']>=400
data.loc[x,['lead_time','arrival_date_year','reservation_status_date','is_canceled','arrival_date_month','arrival_date_day_
```



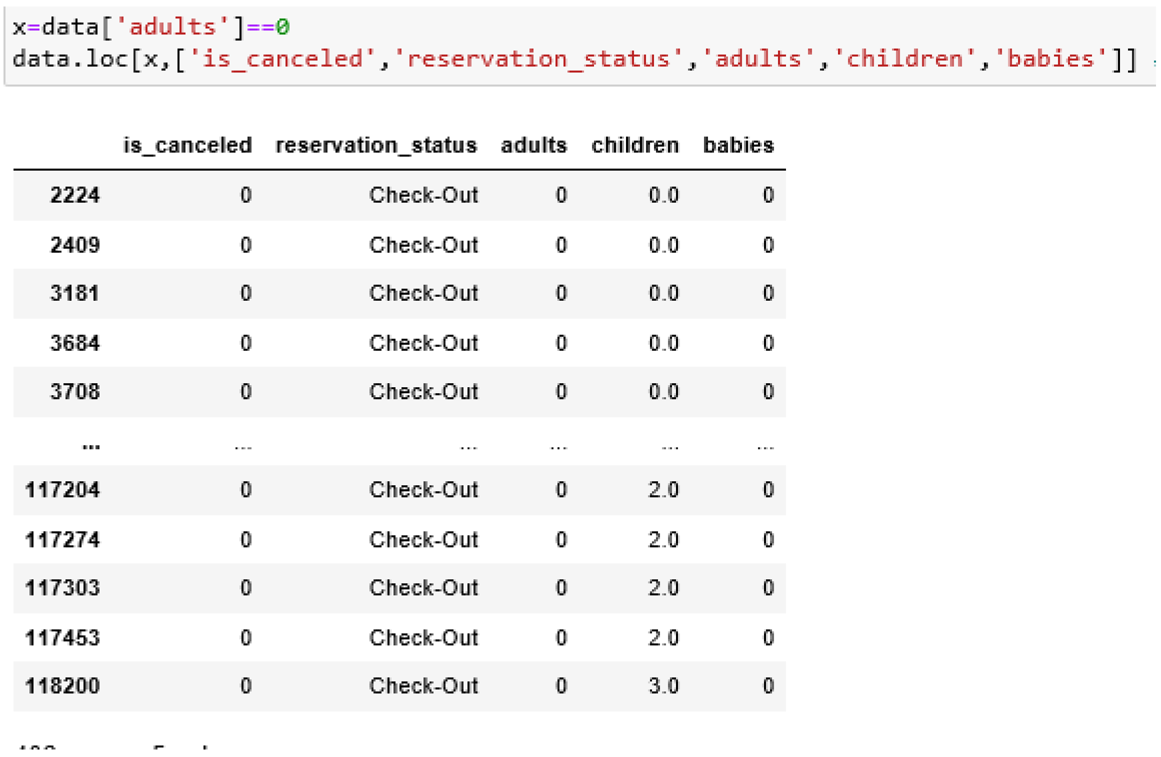
	lead_time	arrival_date_year	reservation_status_date	is_canceled	arrival_date_month	arrival_date_day_of_month
1	737	2015	2015-07-01	0	July	1
890	460	2015	2015-08-10	0	August	3
4182	709	2016	2016-03-24	0	February	25
5704	468	2016	2016-03-04	1	May	12
5705	468	2016	2016-03-04	1	May	12
...	...	...	...	...	...	...
119102	518	2017	2017-08-29	0	August	26
119107	518	2017	2017-08-29	0	August	26
119109	518	2017	2017-08-29	0	August	26
119111	518	2017	2017-08-29	0	August	26
119148	457	2017	2017-08-30	0	August	25

2115 rows × 6 columns

Figure: the outliers of ‘lead_time’

The outliers account for 2115 rows. So, they are still an important factor for whether customers cancel or not. No need to remove them.

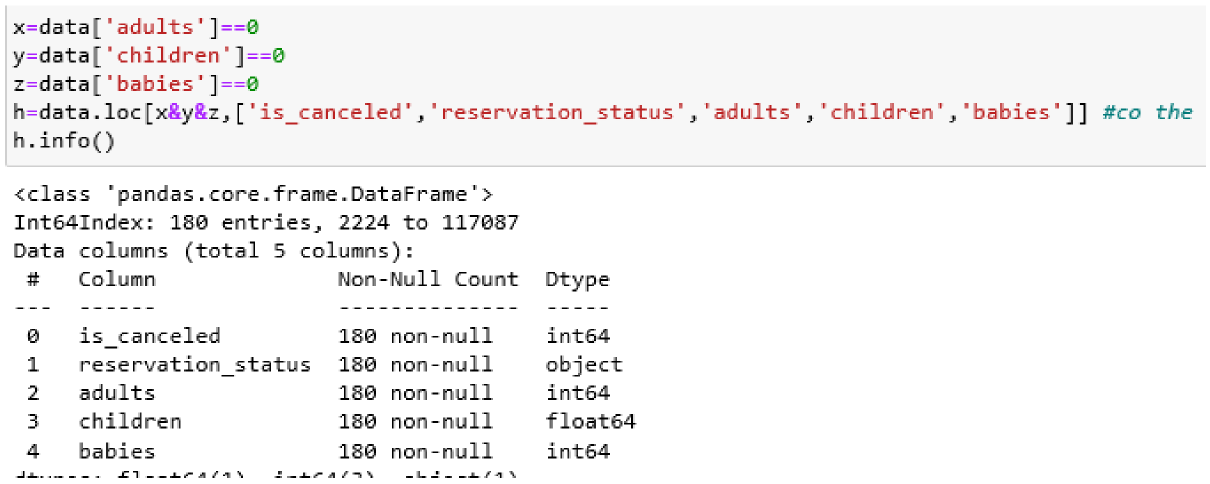
- For **‘adults’** column



```
x=data['adults']==0
data.loc[x,['is_canceled','reservation_status','adults','children','babies']]
```

	is_canceled	reservation_status	adults	children	babies
2224	0	Check-Out	0	0.0	0
2409	0	Check-Out	0	0.0	0
3181	0	Check-Out	0	0.0	0
3684	0	Check-Out	0	0.0	0
3708	0	Check-Out	0	0.0	0
...	...	...	...	...	...
117204	0	Check-Out	0	2.0	0
117274	0	Check-Out	0	2.0	0
117303	0	Check-Out	0	2.0	0
117453	0	Check-Out	0	2.0	0
118200	0	Check-Out	0	3.0	0

Figure: Location of observation with 0 adult



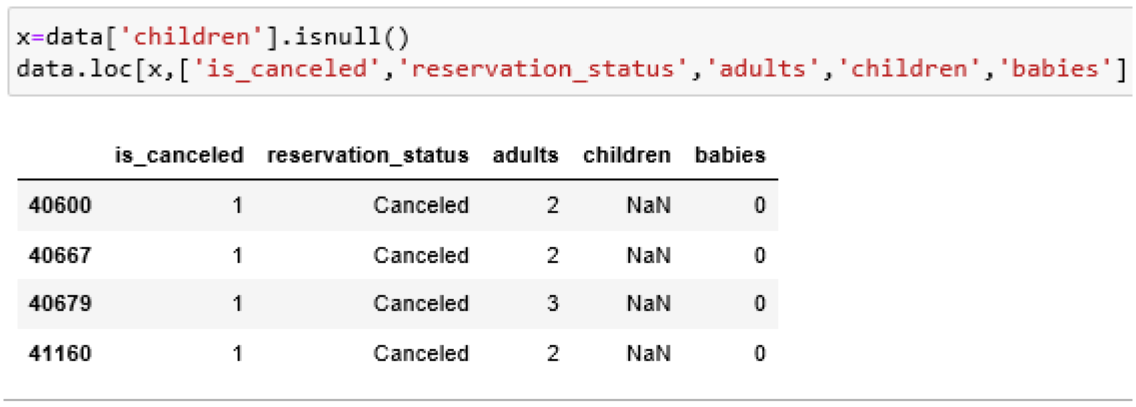
```
x=data['adults']==0
y=data['children']==0
z=data['babies']==0
h=data.loc[x&y&z,['is_canceled','reservation_status','adults','children','babies']] #co the
h.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 180 entries, 2224 to 117087
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   is_canceled         180 non-null    int64  
 1   reservation_status  180 non-null    object 
 2   adults              180 non-null    int64  
 3   children            180 non-null    float64
 4   babies              180 non-null    int64  
```

Figure: 180 entries with zero of adults

3 columns ‘children’.’adults’,’babies’ are equal to 0 at the same time, so we need to fill out one of them, choose ‘adults’ to fill out.

- For **‘children’** column: there are 4 missing values

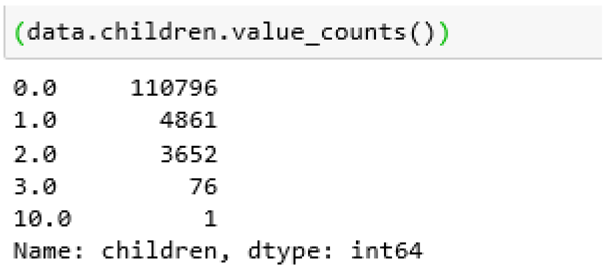


```
x=data['children'].isnull()
data.loc[x,['is_canceled','reservation_status','adults','children','babies']]
```

	is_canceled	reservation_status	adults	children	babies
40600	1	Canceled	2	NaN	0
40667	1	Canceled	2	NaN	0
40679	1	Canceled	3	NaN	0
41160	1	Canceled	2	NaN	0

Figure: Missing Value of children

Replace NaN with 0 because 0 is the most frequency

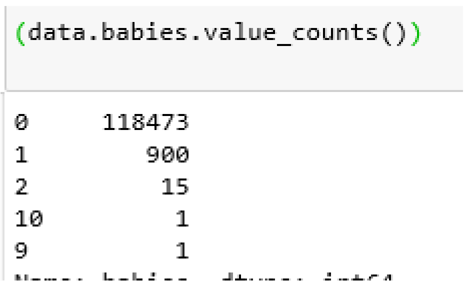


```
(data.children.value_counts())
```

```
0.0     110796
1.0       4861
2.0       3652
3.0         76
10.0         1
Name: children, dtype: int64
```

Figure: Values of children

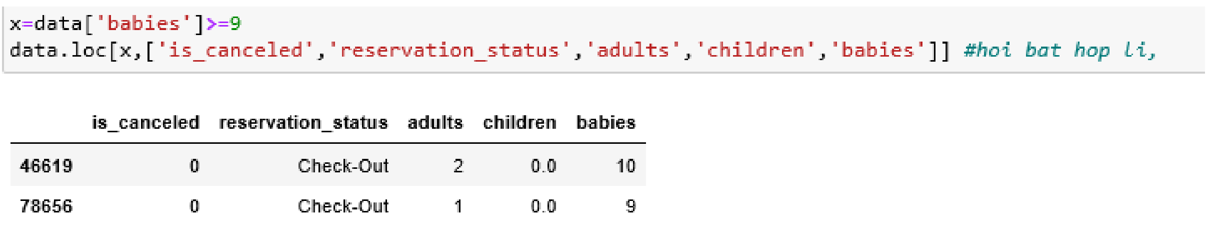
- For the **'babies'** column: Babies 10 and 9 may be outliers.



```
(data.babies.value_counts())
```

```
0     118473
1        900
2         15
10         1
9          1
```

Figure: Values of babies



```
x=data['babies']>=9
data.loc[x,['is_canceled','reservation_status','adults','children','babies']] #hoi bat hop li,
```

	is_canceled	reservation_status	adults	children	babies
46619	0	Check-Out	2	0.0	10
78656	0	Check-Out	1	0.0	9

Figure: 2 entries with babies of 10 and 9

This is not reasonable, with only 2 and 1 adults going with 10 and 9 babies. These babies should be replaced.

```
x=data['babies']>=1
h=data.loc[x,['is_canceled','reservation_status','adults','children','babies']]
```

```
k=h['adults']==0
h.loc[k,:]
```

is_canceled	reservation_status	adults	children	babies

Figure: Location of Adults with Babies more than and equal to 1

With Babies more than and equal to 1, there is no value of 'adults' equal to 0. → It's reasonable because 'adults' have to go with 'babies'

```
data[['country']].isnull().value_counts()
```

```
country
False      118902
True          488
dtype: int64
```

```
data['country'].value_counts().sort_values(ascending=False)
```

```
PRT    48590
GBR    12129
FRA    10415
ESP     8568
DEU     7287
       ...  
AIA        1
NCL        1
SDN        1
KIR        1
NAM        1
Name: country, Length: 177, dtype: int64
```

Figure: Missing values of 'country'

Fill missing values of 'country' with Mode value.

- For **'previous_cancellations'** column

```
data.previous_cancellations.value_counts()
sns.boxplot(y='previous_cancellations',data=data)
```

```
<AxesSubplot:ylabel='previous_cancellations'>
```

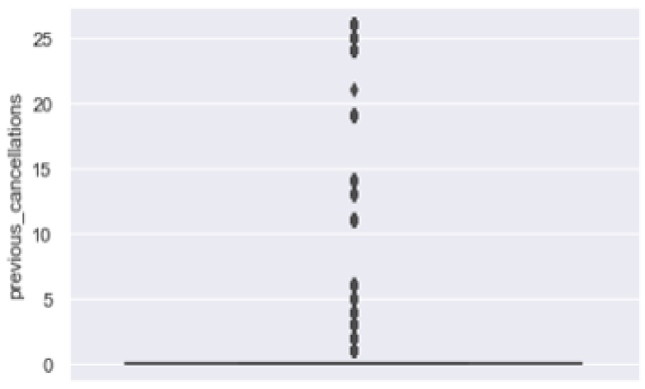
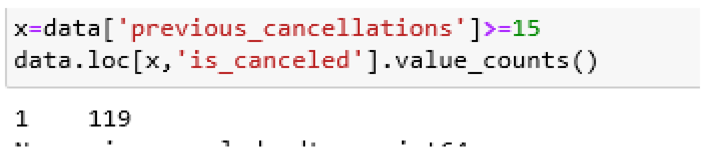


Figure: Boxplot of 'previous_cancellations'

=>There are so many outliers.



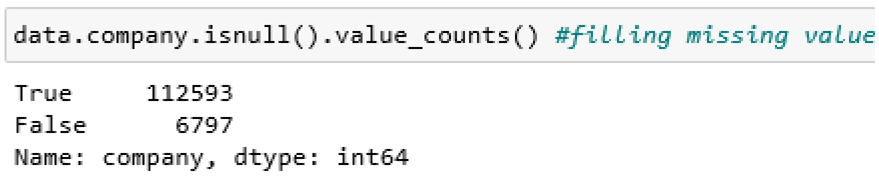
```
x=data['previous_cancellations']>=15
data.loc[x,'is_canceled'].value_counts()
```

```
1    119
```

Figure: Count values of 'is_canceled' with 'previous_cancellations' more than and equal 15

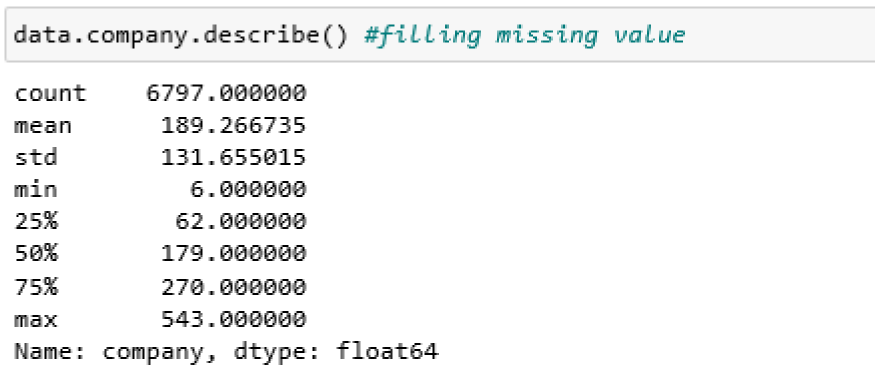
If 'previous_cancellation' of booking is more than and equal to 15, 'is_cancelled' returns to 1. It shows that people who cancel so many times previously, will cancel the next time of booking. That is important characteristics of label 1

- For **'company'** columns:



```
data.company.isnull().value_counts() #filling missing value
```

```
True     112593
False      6797
Name: company, dtype: int64
```



```
data.company.describe() #filling missing value
```

```
count    6797.000000
mean      189.266735
std       131.655015
min         6.000000
25%        62.000000
50%       179.000000
75%       270.000000
max       543.000000
Name: company, dtype: float64
```

Figure: Statistics of "company'

There are so many nulls, we need to fill them out. The best way to do it is replacing null values with 0, because maybe booking is for family and no company.

- For **'adr'** column:

```
x=data['adr']==5400.00
data.loc[x,['is_canceled','adr']]
```

	is_canceled	adr
48515	1	5400.0

```
sns.boxplot(y='adr',data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2160ce7a00>
```

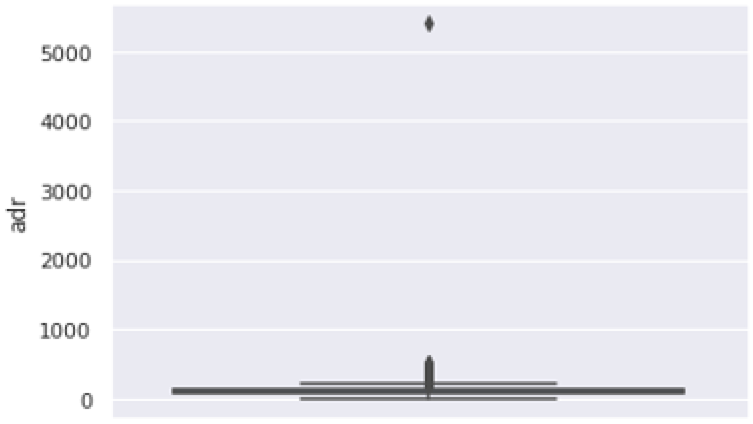
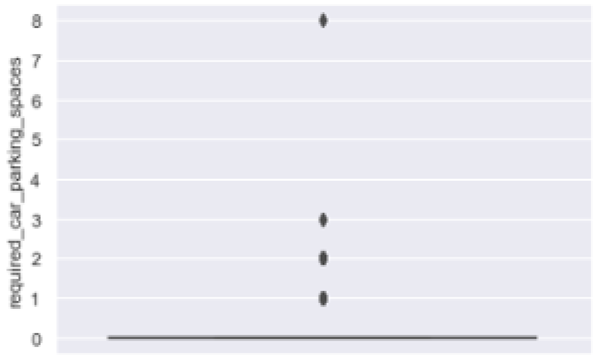


Figure: Outliers of 'adr'

There is an outlier of 5400, but retain this one because that is characteristics for label 1.

- For **'required_car_parking_space'** column



```
sns.boxplot(y='required_car_parking_spaces',data=data)
data.loc[data['required_car_parking_spaces']>=3,['required_car_parking_spaces','is_canceled']]
```

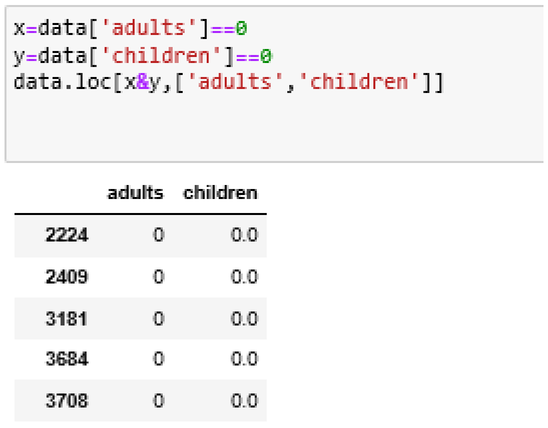
	required_car_parking_spaces	is_canceled
29045	8	0
29046	8	0
38117	3	0
102762	3	0
110812	3	0

Figure: Outliers of 'Required_car_parking_spaces'

Retain this one because that is a characteristics for label 0.

Step 3: Fix Problems

- For **‘adults’ and ‘children’**

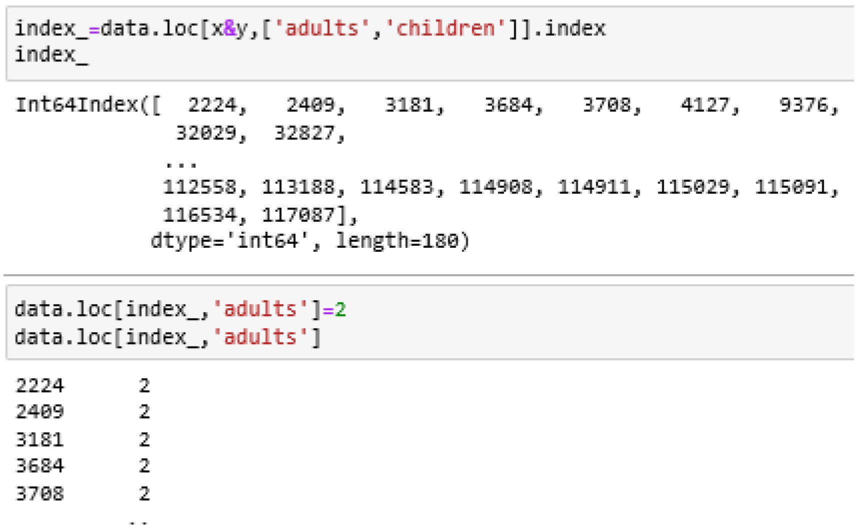


```
x=data['adults']==0
y=data['children']==0
data.loc[x&y,['adults','children']]
```

	adults	children
2224	0	0.0
2409	0	0.0
3181	0	0.0
3684	0	0.0
3708	0	0.0

Figure: Location of ‘adults’ and ‘children’ are equal to 0 together

Replacing the values of ‘adults’ and ‘children’ are equal to 0 together with the median, because both of them should not be 0 at the same time. It's not reasonable for anyone to stay. Deciding to replace one of both, it’s more reasonable for replacing in ‘adults’.



```
index_=data.loc[x&y,['adults','children']].index
index_
```

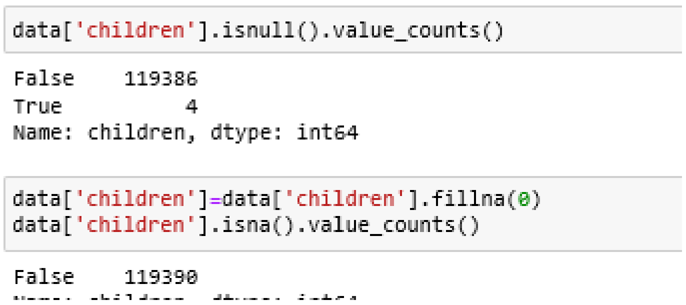
```
Int64Index([  2224,   2409,   3181,   3684,   3708,   4127,   9376,
             32029,  32827,
            ...
            112558, 113188, 114583, 114908, 114911, 115029, 115091,
            116534, 117087],
           dtype='int64', length=180)
```

```
data.loc[index_,'adults']=2
data.loc[index_,'adults']
```

```
2224      2
2409      2
3181      2
3684      2
3708      2
         ..
```

Figure: Replacing with mode of ‘adult’ for ‘adult’

- For **‘children’** column;



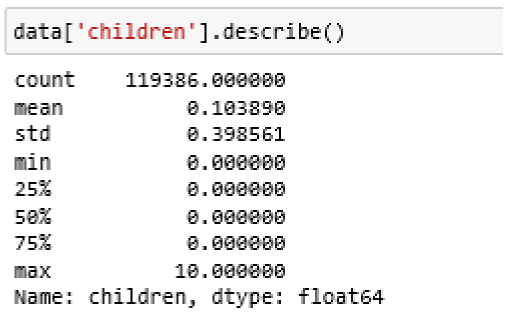
```
data['children'].isnull().value_counts()
```

```
False    119386
True          4
Name: children, dtype: int64
```

```
data['children']=data['children'].fillna(0)
data['children'].isna().value_counts()
```

```
False    119390
```

Figure: Fill in missing values of ‘children’ with mode of ‘children’



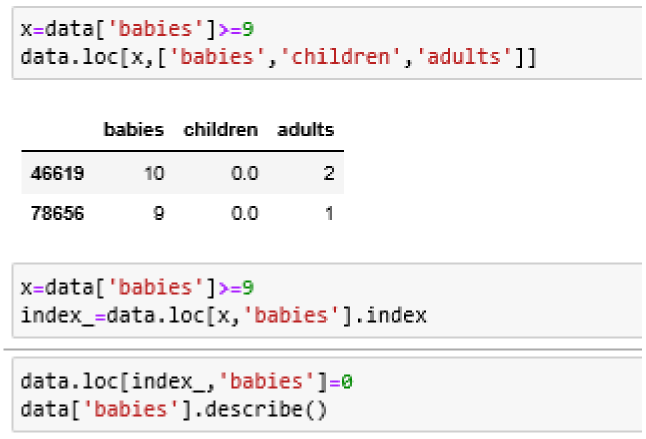
```
data['children'].describe()
```

```
count    119386.000000
mean          0.103890
std           0.398561
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max          10.000000
Name: children, dtype: float64
```

Figure: Statistics of 'children'

Fill mode of 'children' because statistics show that 0 accounts almost for 'children'.

- For **'babies'** column



```
x=data['babies']>=9
data.loc[x,['babies','children','adults']]
```

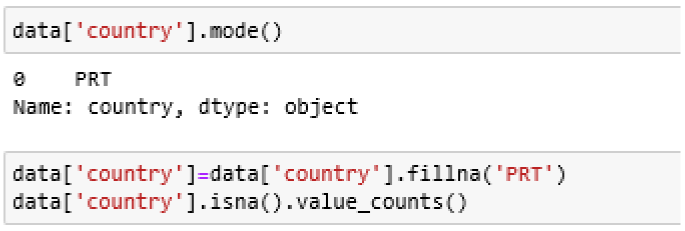
	babies	children	adults
46619	10	0.0	2
78656	9	0.0	1

```
x=data['babies']>=9
index_=data.loc[x,'babies'].index
```

```
data.loc[index_,'babies']=0
data['babies'].describe()
```

Figure: Replacing outliers of babies with 0

- For **'country'** column



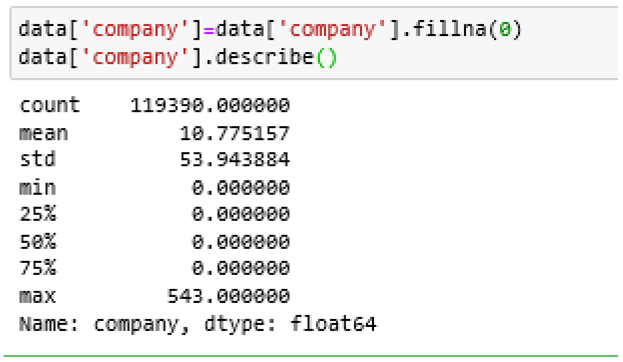
```
data['country'].mode()
```

```
0    PRT
Name: country, dtype: object
```

```
data['country']=data['country'].fillna('PRT')
data['country'].isna().value_counts()
```

Figure: Fill missing values of 'country' with mode

- For **'company'** column



```
data['company']=data['company'].fillna(0)
data['company'].describe()
```

```
count    119390.000000
mean         10.775157
std          53.943884
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max         543.000000
Name: company, dtype: float64
```

Figure: Fill missing values of 'company' with 0

- For **'agent'** column

```
data['agent']=data['agent'].fillna(0)
```

```
data['agent'].describe()
```

```
count    119390.000000
mean         74.828319
std         107.141953
min           0.000000
25%           7.000000
50%           9.000000
75%         152.000000
max         535.000000
Name: agent, dtype: float64
```

Figure: Fill missing values of ‘agent’ with 0

Step 4: Feature engineering

- Creating **‘arrival_date’** column

```
cols=["arrival_date_year","arrival_date_month","arrival_date_day_of_month"]
data['arrival_date'] = data[cols].apply(lambda x: '-'.join(x.values.astype(str)), axis="columns")
data['arrival_date']=pd.to_datetime(data['arrival_date'])
data['reservation_status_date']=pd.to_datetime(data['reservation_status_date'])
data[['arrival_date','reservation_status_date']]
```

	arrival_date	reservation_status_date
0	2015-07-01	2015-07-01
1	2015-07-01	2015-07-01
2	2015-07-01	2015-07-02
3	2015-07-01	2015-07-02
4	2015-07-01	2015-07-03
...	...	...

*Figure: Creating **‘arrival_date’** column*

Creating ‘arrival_date’ with the goal of comparing 'arrival_date' and 'reservation_status_date'

+ With ‘check-out’ , 'arrival_date' is equal and smaller than 'reservation_status_date'

```
 x=data.loc[data['arrival_date'] > data['reservation_status_date'],'reservation_status']=='Check-Out'
x.value_counts() #chứng tỏ ngày khách check out Luôn Lớn hơn hoặc bằng ngày khách đến`
```

```
False    42137
Name: reservation_status, dtype: int64
```

```
x=data.loc[data['arrival_date'] < data['reservation_status_date'],'reservation_status']=='Check-Out'
x.value_counts() #chứng tỏ ngày khách check out Luôn Lớn hơn hoặc bằng ngày khách đến`
```

```
True    74460
Name: reservation_status, dtype: int64
```

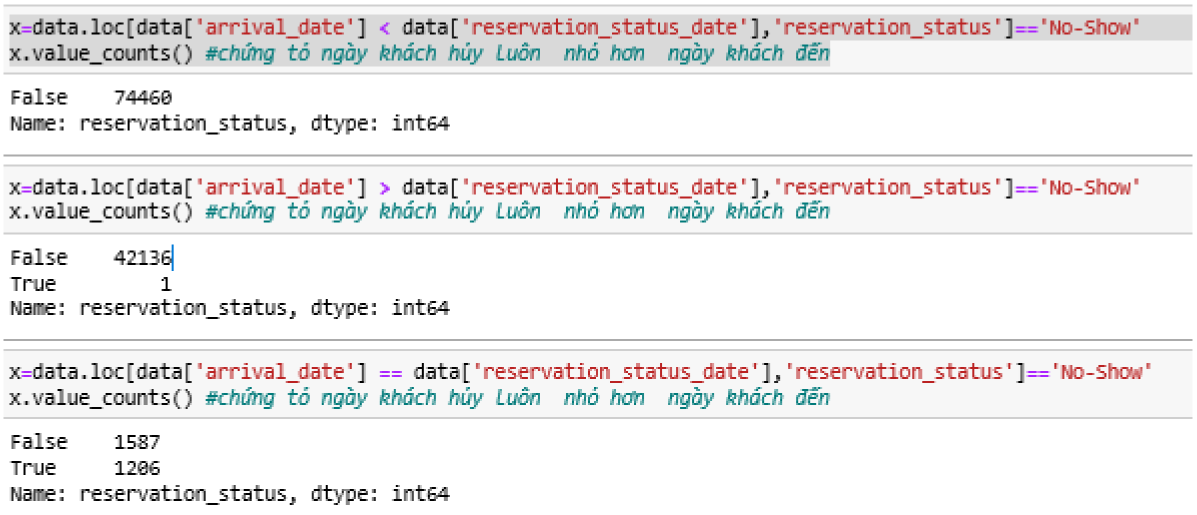
```
y=data.loc[data['arrival_date'] == data['reservation_status_date'],'reservation_status']=='Check-Out'
y.value_counts() #chứng tỏ ngày khách check out Luôn Lớn hơn hoặc bằng ngày khách đến => hợp lí
```

```
False    2087
True      706
Name: reservation_status, dtype: int64
```

Figure: Check 'arrival_date' is equal and smaller than 'reservation_status_date' with ‘check-out’

→ It’s reasonable

+ With ‘No-Show’, 'arrival_date' is equal to 'reservation_status_date'



```
x=data.loc[data['arrival_date'] < data['reservation_status_date'],'reservation_status']=='No-Show'
x.value_counts() #chứng tỏ ngày khách hủy Luôn  nhỏ hơn  ngày khách đến
```

```
False    74460
Name: reservation_status, dtype: int64
```

```
x=data.loc[data['arrival_date'] > data['reservation_status_date'],'reservation_status']=='No-Show'
x.value_counts() #chứng tỏ ngày khách hủy Luôn  nhỏ hơn  ngày khách đến
```

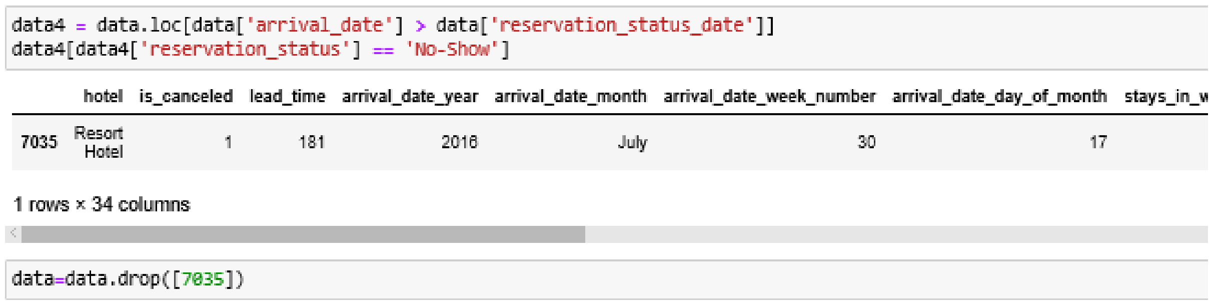
```
False    42136
True         1
Name: reservation_status, dtype: int64
```

```
x=data.loc[data['arrival_date'] == data['reservation_status_date'],'reservation_status']=='No-Show'
x.value_counts() #chứng tỏ ngày khách hủy Luôn  nhỏ hơn  ngày khách đến
```

```
False    1587
True     1206
Name: reservation_status, dtype: int64
```

Figure: Check 'arrival_date' is equal to 'reservation_status_date'

→ There is 1 row 'arrival_date' is larger than 'reservation_status_date', need to remove that row



```
data4 = data.loc[data['arrival_date'] > data['reservation_status_date']]
data4[data4['reservation_status'] == 'No-Show']
```

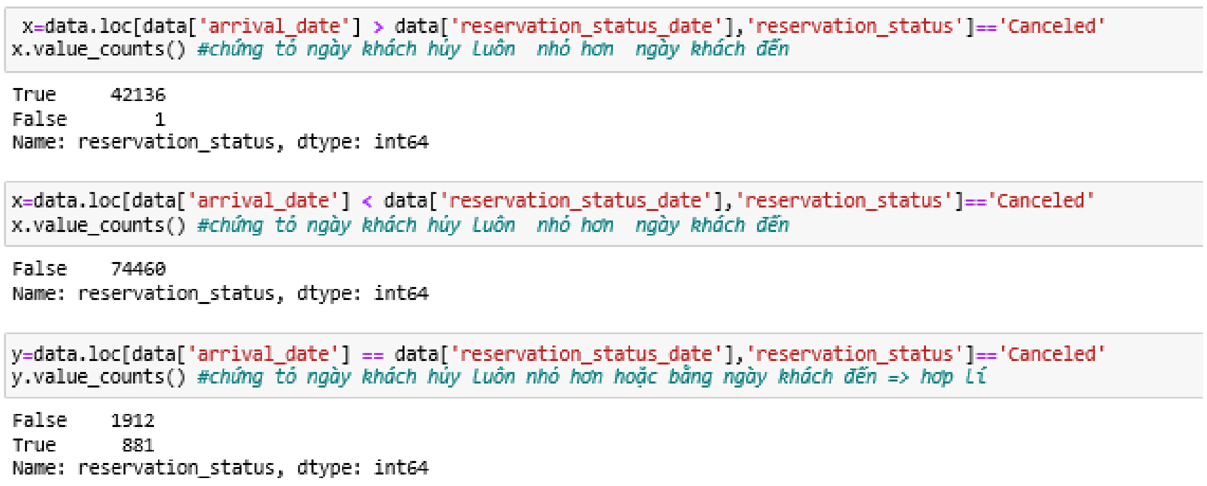
	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_w
7035	Resort Hotel	1	181	2016	July	30	17	

1 rows × 34 columns

```
data=data.drop([7035])
```

Figure: Remove that row

+ With ‘Canceled’, 'arrival_date' is equal and larger than 'reservation_status_date'



```
x=data.loc[data['arrival_date'] > data['reservation_status_date'],'reservation_status']=='Canceled'
x.value_counts() #chứng tỏ ngày khách hủy Luôn  nhỏ hơn  ngày khách đến
```

```
True     42136
False        1
Name: reservation_status, dtype: int64
```

```
x=data.loc[data['arrival_date'] < data['reservation_status_date'],'reservation_status']=='Canceled'
x.value_counts() #chứng tỏ ngày khách hủy Luôn  nhỏ hơn  ngày khách đến
```

```
False    74460
Name: reservation_status, dtype: int64
```

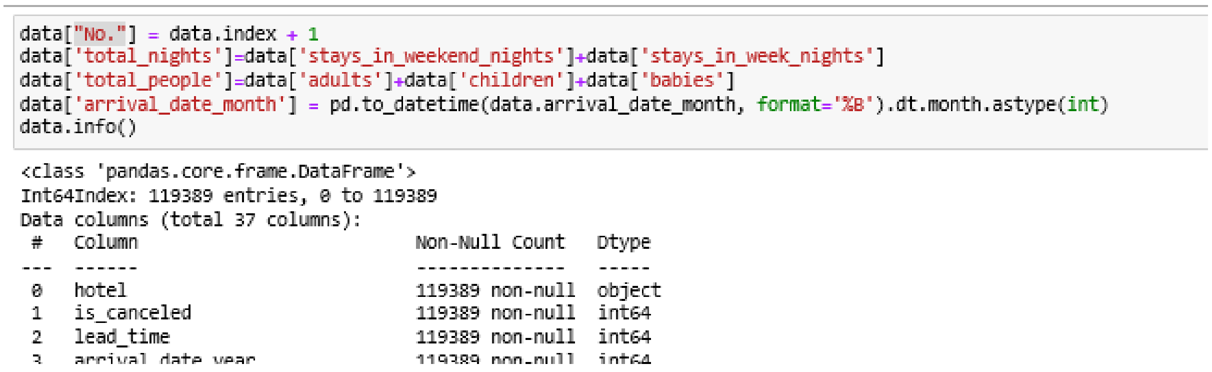
```
y=data.loc[data['arrival_date'] == data['reservation_status_date'],'reservation_status']=='Canceled'
y.value_counts() #chứng tỏ ngày khách hủy Luôn nhỏ hơn hoặc bằng ngày khách đến => hợp lí
```

```
False    1912
True      881
Name: reservation_status, dtype: int64
```

Figure: Check 'arrival_date' is equal and larger than 'reservation_status_date' with 'Canceled'

→ It's reasonable

- Create 'No.', 'total_nights', 'total_people', 'arrival_date_month' columns



```
data["No."] = data.index + 1
data['total_nights']=data['stays_in_weekend_nights']+data['stays_in_week_nights']
data['total_people']=data['adults']+data['children']+data['babies']
data['arrival_date_month'] = pd.to_datetime(data.arrival_date_month, format='%B').dt.month.astype(int)
data.info()
```

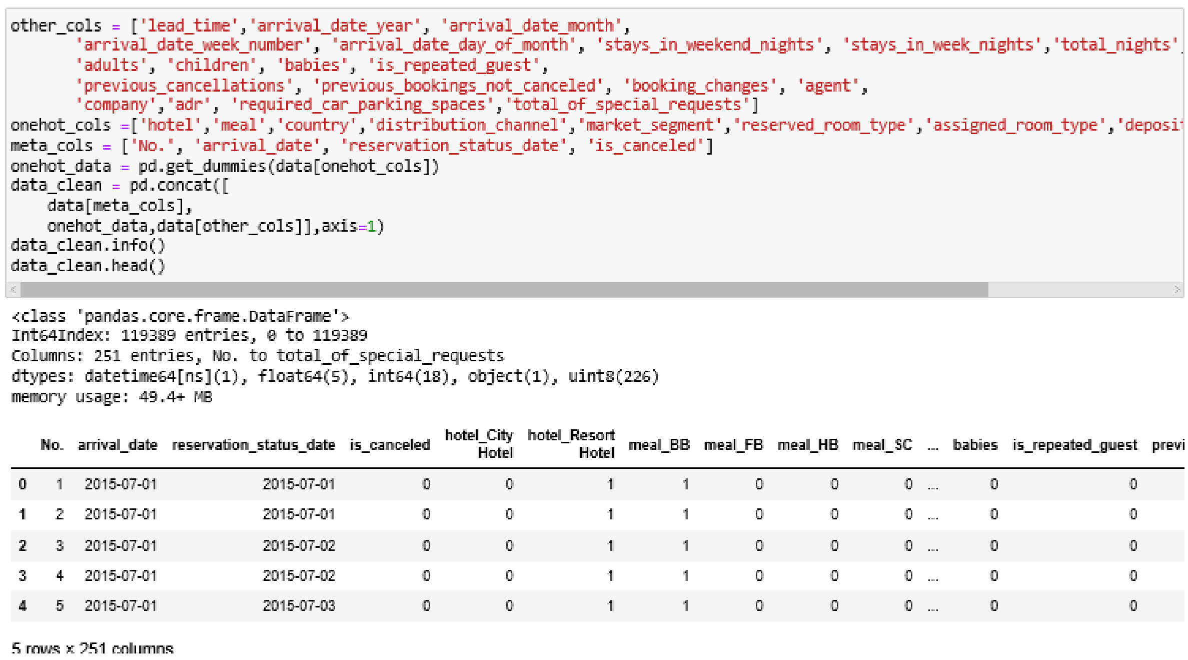
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 119389 entries, 0 to 119389
Data columns (total 37 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           119389 non-null  object
 1   is_canceled                     119389 non-null  int64
 2   lead_time                       119389 non-null  int64
 3   arrival_date_year               119389 non-null  int64
```

Figure: Create features

+ Set data.index +1 into 'No.' column
+ Set sum of'stays_in_weekend_nights'+'stays_in_week_nights' into 'total_nights'
+ Set sum of 'adults' and ' children' into 'total_people'
+ Set dtype of 'arrival_date_month' into 'integer'

2. Modelling

Step 1: One hot encoding



```
other_cols = ['lead_time','arrival_date_year', 'arrival_date_month',
       'arrival_date_week_number', 'arrival_date_day_of_month', 'stays_in_weekend_nights', 'stays_in_week_nights','total_nights'
       'adults', 'children', 'babies', 'is_repeated_guest',
       'previous_cancellations', 'previous_bookings_not_canceled', 'booking_changes', 'agent',
       'company','adr', 'required_car_parking_spaces','total_of_special_requests']
onehot_cols =['hotel','meal','country','distribution_channel','market_segment','reserved_room_type','assigned_room_type','deposi
meta_cols = ['No.', 'arrival_date', 'reservation_status_date', 'is_canceled']
onehot_data = pd.get_dummies(data[onehot_cols])
data_clean = pd.concat([
    data[meta_cols],
    onehot_data,data[other_cols]],axis=1)
data_clean.info()
data_clean.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 119389 entries, 0 to 119389
Columns: 251 entries, No. to total_of_special_requests
dtypes: datetime64[ns](1), float64(5), int64(18), object(1), uint8(226)
memory usage: 49.4+ MB
```

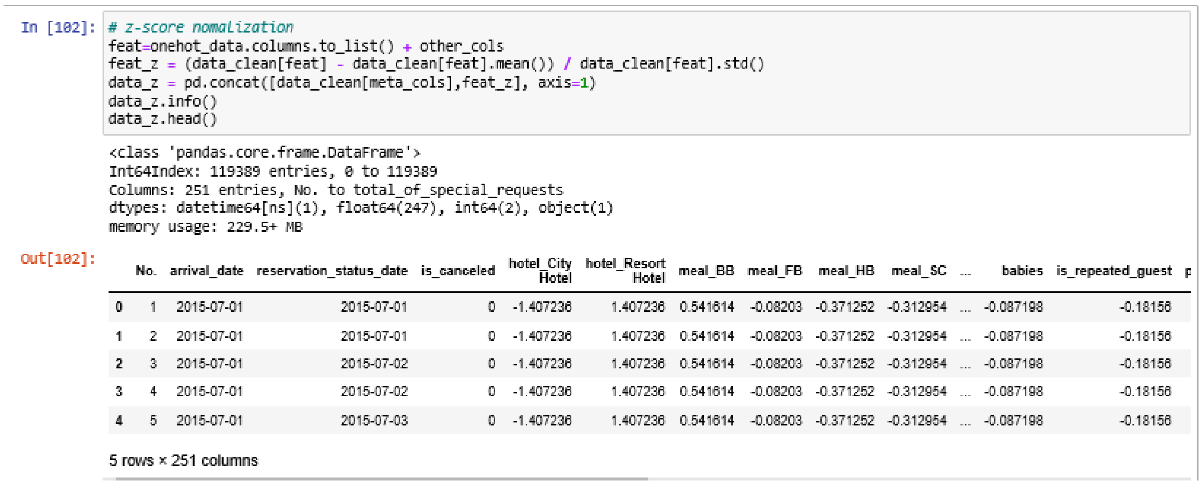
	No.	arrival_date	reservation_status_date	is_canceled	hotel_City Hotel	hotel_Resort Hotel	meal_BB	meal_FB	meal_HB	meal_SC	...	babies	is_repeated_guest	previ
0	1	2015-07-01	2015-07-01	0	0	1	1	0	0	0	...	0	0	
1	2	2015-07-01	2015-07-01	0	0	1	1	0	0	0	...	0	0	
2	3	2015-07-01	2015-07-02	0	0	1	1	0	0	0	...	0	0	
3	4	2015-07-01	2015-07-02	0	0	1	1	0	0	0	...	0	0	
4	5	2015-07-01	2015-07-03	0	0	1	1	0	0	0	...	0	0	

5 rows × 251 columns

Figure: Dataset after one hot encoding

Features are categorical variables that convert into numeric vars by onehot encoding. onehot_cols=['hotel','meal','country','distribution_channel','market_segment','reserved_room_type','assigned_room_type','deposit_type','customer_type']

Step 2: Normalization



```
In [102]: # z-score nomalization
          feat=onehot_data.columns.to_list() + other_cols
          feat_z = (data_clean[feat] - data_clean[feat].mean()) / data_clean[feat].std()
          data_z = pd.concat([data_clean[meta_cols],feat_z], axis=1)
          data_z.info()
          data_z.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 119389 entries, 0 to 119389
Columns: 251 entries, No. to total_of_special_requests
dtypes: datetime64[ns](1), float64(247), int64(2), object(1)
memory usage: 229.5+ MB
```

Out[102]:

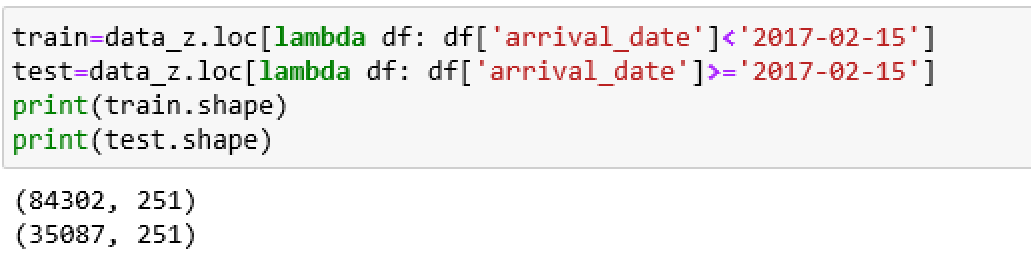
	No.	arrival_date	reservation_status_date	is_canceled	hotel_City Hotel	hotel_Resort Hotel	meal_BB	meal_FB	meal_HB	meal_SC	...	babies	is_repeated_guest	p
0	1	2015-07-01	2015-07-01	0	-1.407236	1.407236	0.541614	-0.08203	-0.371252	-0.312954	...	-0.087198	-0.18156	
1	2	2015-07-01	2015-07-01	0	-1.407236	1.407236	0.541614	-0.08203	-0.371252	-0.312954	...	-0.087198	-0.18156	
2	3	2015-07-01	2015-07-02	0	-1.407236	1.407236	0.541614	-0.08203	-0.371252	-0.312954	...	-0.087198	-0.18156	
3	4	2015-07-01	2015-07-02	0	-1.407236	1.407236	0.541614	-0.08203	-0.371252	-0.312954	...	-0.087198	-0.18156	
4	5	2015-07-01	2015-07-03	0	-1.407236	1.407236	0.541614	-0.08203	-0.371252	-0.312954	...	-0.087198	-0.18156	

5 rows × 251 columns

Figure: Dataset after normalization

Use Z-score normalization to normalize numerical variables.

Step 3: Split data



```
train=data_z.loc[lambda df: df['arrival_date']<'2017-02-15']
test=data_z.loc[lambda df: df['arrival_date']>='2017-02-15']
print(train.shape)
print(test.shape)
```

```
(84302, 251)
(35087, 251)
```

Figure: Split data into train and test data

Split data into train and test data based on ‘arrival_date’ with the ratio 7:3

- Train data: dataset with ‘arrival_date’ < ‘2017-02-15’
- Test data; dataset with ‘arrival_date’ >= ‘2017-02-15’

Step 4: Modeling

Training set is fitted to each model and uses this model to classify the testing set. Compare the classified result with the real result of testing set to evaluate which model is suitable.

- K Nearest Neighbors

```
# Creat a KNN classifier
knn_model = KNeighborsClassifier(n_neighbors=10)

# Train the model using the training set
knn_model.fit(train[feat], train[label_col])

# Predict on test data
knn_predict = knn_model.predict(test[feat])
```

```
knn_cm = confusion_matrix(test[label_col], knn_predict)
knn_cm = pd.DataFrame(knn_cm, index=['True 0', 'True 1'], columns=['Predict 0', 'Predi
knn_cm
```

	Predict 0	Predict 1
True 0	18938	2299
True 1	6492	7358

Figure: Confusion matrix of K Nearest Neighbors

- Support Vector Machine

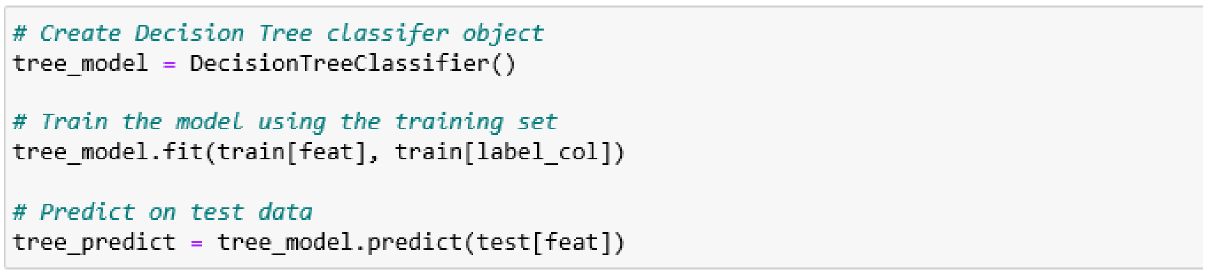
```
# Create a svm Classifier
svm_model = svm.SVC(kernel='linear') # Linear Kernel

# Train the model using the training set
svm_model.fit(train[feat], train[label_col])

# Predict on test data
svm_predict = svm_model.predict(test[feat])
```

```
svm_cm = confusion_matrix(test[label_col], svm_predict)
svm_cm = pd.DataFrame(svm_cm, index=['True 0', 'True 1'], columns=['Predict 0', 'Pr
print(svm_cm)
```

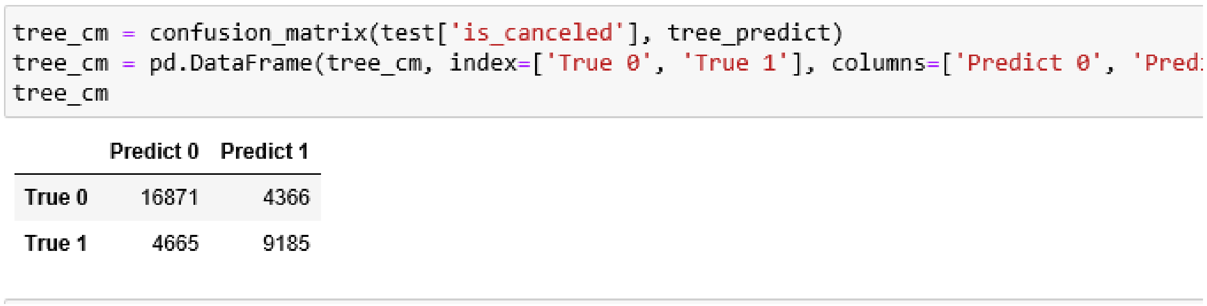
```
        Predict 0  Predict 1
True 0      17240       3997
True 1       3849      10001
```

Figure: Confusion matrix of SVM

- Decision Tree



```
# Create Decision Tree classifer object
tree_model = DecisionTreeClassifier()

# Train the model using the training set
tree_model.fit(train[feat], train[label_col])

# Predict on test data
tree_predict = tree_model.predict(test[feat])
```

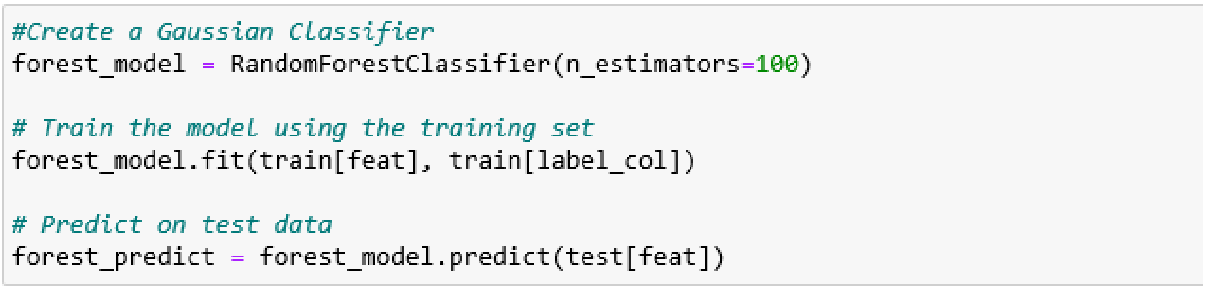


```
tree_cm = confusion_matrix(test['is_canceled'], tree_predict)
tree_cm = pd.DataFrame(tree_cm, index=['True 0', 'True 1'], columns=['Predict 0', 'Pred
tree_cm
```

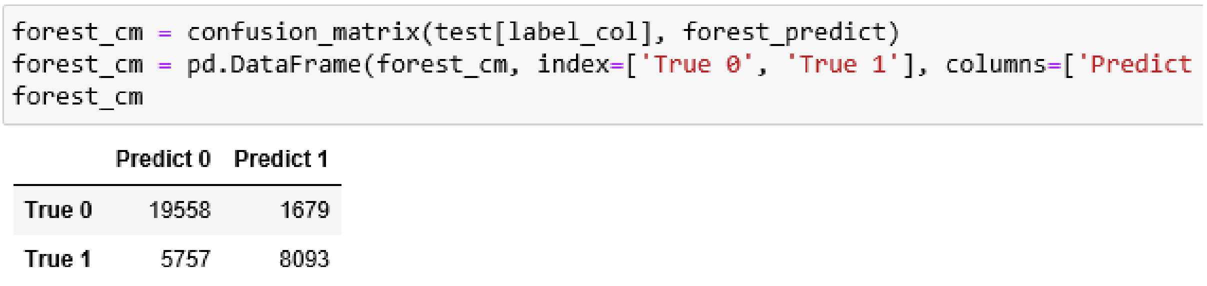
	Predict 0	Predict 1
True 0	16871	4366
True 1	4665	9185

Figure: Confusion matrix of Decision Tree

- Random Forest Classifier



```
#Create a Gaussian Classifier
forest_model = RandomForestClassifier(n_estimators=100)

# Train the model using the training set
forest_model.fit(train[feat], train[label_col])

# Predict on test data
forest_predict = forest_model.predict(test[feat])
```



```
forest_cm = confusion_matrix(test[label_col], forest_predict)
forest_cm = pd.DataFrame(forest_cm, index=['True 0', 'True 1'], columns=['Predict
forest_cm
```

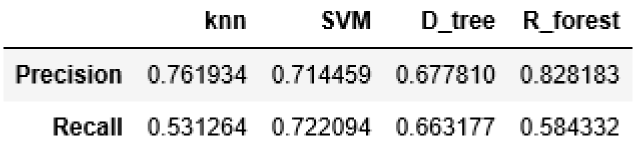
	Predict 0	Predict 1
True 0	19558	1679
True 1	5757	8093

Figure: Confusion matrix of Random Forest

Step 5: Compare Models

Recall and Precision are calculated by Predict 1 and True 1. It means that both of them are calculated by canceled booking. Not choose Predict 0 and True 0 for calculating Recall and Precision, because the ratio 'is_canceled' of '0' and '1' is equal to 2. It shows that booking not canceled is twice as much as canceled booking.

So, the results of 0 are better because there are many observations of '0' to support for classification of '0'. It makes Recall and Precision for '0' always high no matter how model is. This leads to difficulty of effectiveness of those models.



	knn	SVM	D_tree	R_forest
Precision	0.761934	0.714459	0.677810	0.828183
Recall	0.531264	0.722094	0.663177	0.584332

Figure: Precision and Recall of each model

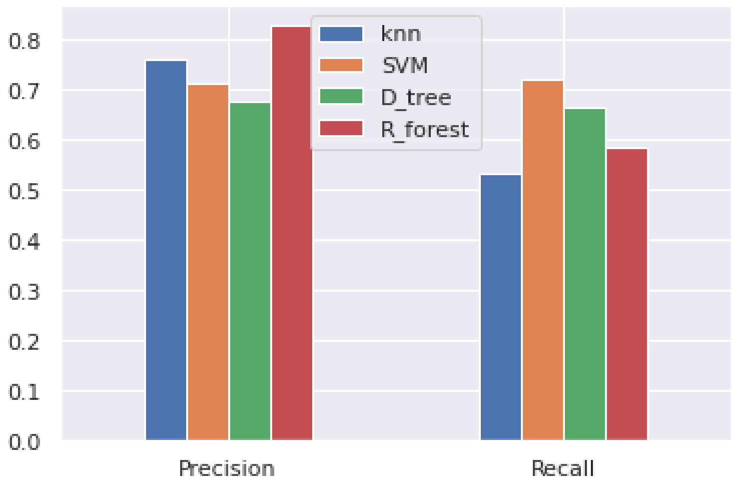
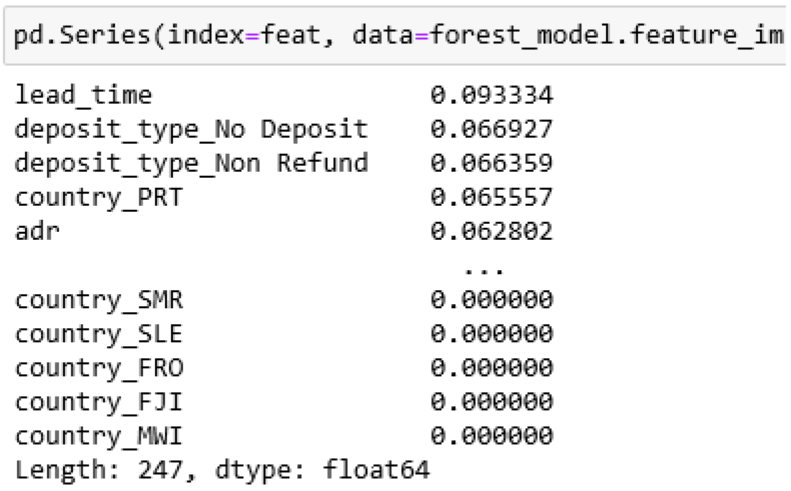


Figure: Compare recall and precision

→ Model SVM with the highest of Recall may be the best model for this dataset

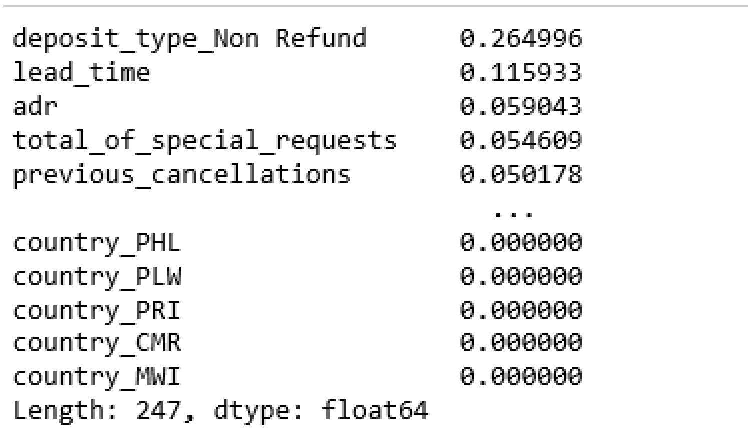
Step 6: Feature importance



```
pd.Series(index=feat, data=forest_model.feature_im
```

```
lead_time                    0.093334
deposit_type_No Deposit      0.066927
deposit_type_Non Refund      0.066359
country_PRT                  0.065557
adr                          0.062802
                               ...
country_SMR                  0.000000
country_SLE                  0.000000
country_FRO                  0.000000
country_FJI                  0.000000
country_MWI                  0.000000
Length: 247, dtype: float64
```

Figure: Feature Importance of Random Forest Model



```
deposit_type_Non Refund      0.264996
lead_time                    0.115933
adr                          0.059043
total_of_special_requests    0.054609
previous_cancellations       0.050178
                               ...
country_PHL                  0.000000
country_PLW                  0.000000
country_PRI                  0.000000
country_CMR                  0.000000
country_MWI                  0.000000
Length: 247, dtype: float64
```

Figure: Feature Importance of Decision Tree model

IV. Conclusion

Through this exercise, we had the opportunity to interact with a large data set and apply the knowledge learned in class to the data. During the course of the test, we learned how to deal with a dataset, how to ask questions when a problem occurs and how to solve it, especially for anomalies in the data set. data. In terms of strength, the dataset has been processed logically and the model has been written based on what we have learned in class and can apply this model to similar datasets. However, the weakness in this test is that we do not have much knowledge about the hotel industry, so the information in the dataset has to be searched online, so the test may have errors in understanding of data. Finally, we hope that in the future we will have the opportunity to be exposed to data sets closer to our field - Logistics & Supply Chain Management.