VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

SCHOOL OF INDUSTRIAL ENGINEERING & MANAGEMENT

PROJECT REPORT

Course: Data Mining

HOTEL BOOKING DEMAND

Lecturer: M.Sc. Ngo Thi Thao Uyen

Group: 03

Table of contents

Abstract

A highly accurate demand prediction is fundamental to the success of every revenue management model. Customers who book through the online website cancel the reservation a few days before their scheduled arrival. This problem frequently arises when a hotel offers a free cancellation service, which is viewed as a hotel policy to improve customer service and focus on consumers when booking comfortably without charge. This policy in favor of customers leads to the disadvantage of the hotel when the customer completes the booking and cancels before the official time. The hotel will suffer a loss when it is forced to keep the room unoccupied and not utilize the resources. This paper will analyze the hotel booking dataset and study machine learning methods to investigate which sorts of consumers and what traits of customers frequently cancel rooms. Machine learning is applied for the booking cancellation prediction problem so that hoteliers manage bookings, classify a hotel booking's likelihood to be canceled, and determine how many customers can cancel. From there, come up with a solution that can generate more bookings on internet platforms to reduce the loss and generate sustainable revenue from available resources.

I. Introduction

Hotel Booking Demand Dataset is published in Data in Brief, Volume 22, February 2019 by Nuno Antonio, Ana Almeida, and Luis Nunes. This dataset comes from two hotels, a city hotel, and a resort hotel, with each observation represents a hotel booking. It includes totally 119,390 observations following with 31 variables such as when the booking was made, length of stay, the number of adults, children, and/or babies, and the number of available parking spaces, among other things from July 2015 to August 2017.

1.1 Data

The data set contains the following variables:

Table 1: Description of dataset

Variable	Type	Description
Hotel	Categorical	Type of Hotel whether Resort Hotel or City Hotel
is_canceled	Binary	Value indicating if the booking was canceled (1) or not (0)
lead_time	Integer	Number of days that elapsed between the entering date of the booking into the PMS and the arrival date
arrival_date_year	Integer	Year of arrival date
arrival_date_month	Categorical	Month of arrival date with 12 categories: “January” to “December”
arrival_date_week_number	Integer	Week number of year for arrival date
arrival_date_day_of_month	Integer	Day of arrival date
stays_in_weekend_nights	Integer	Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
stays_in_week_nights	Integer	Number of weeknights (Monday to Friday) the guest stayed or booked to stay at the hotel
adults	Integer	Number of adults
children	Integer	Number of children
babies	Integer	Number of babies
meal	Categorical	Type of meal booked. 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	Categorical	Country of origin.
market_segment	Categorical	Market segment designation. In categories, the term “TA” means “Travel Agents” and “TO” means “Tour Operators”

distribution_channel	Categorical	Booking distribution channel. The term “TA” means “Travel Agents” and “TO” means “Tour Operators”
is_repeated_guest	Binary	Value indicating if the booking name was from a repeated guest (1) or not (0)
previous_cancellations	Categorical	Number of previous bookings that were cancelled by the customer prior to the current booking
previous_bookings_ not_canceled	Integer	Number of previous bookings not cancelled by the customer prior to the current booking
reserved_room_type	Categorical	Code of room type reserved. Code is presented instead of designation for anonymity reasons
assigned_room_type	Categorical	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.
booking_changes	Integer	Number of changes/amendments made to the booking from the moment the booking was entered on the PMS until the moment of check-in or cancellation
deposit_type	Categorical	Type of deposit made for booking: 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	Categorical	ID of the travel agency that made the booking
company	Categorical	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	Integer	Number of days the booking was in the waiting list before it was confirmed to the customer
customer_type	Categorical	Type of booking: 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	Numerical	Average Daily Rate as defined by dividing the sum of all lodging transactions by the total number of staying nights
required_car_parking_ spaces	Integer	Number of car parking spaces required by the customer

total_of_special_requests	Integer	Number of special requests made by the customer (e.g., twin bed or high floor)
reservation_status	Categorical	Reservation last status: 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	Date at which the last status was set. This variable can be used in conjunction with the Reservation Status to understand when the booking was canceled or when did the customer checked-out of the hotel

1.2 Methodology

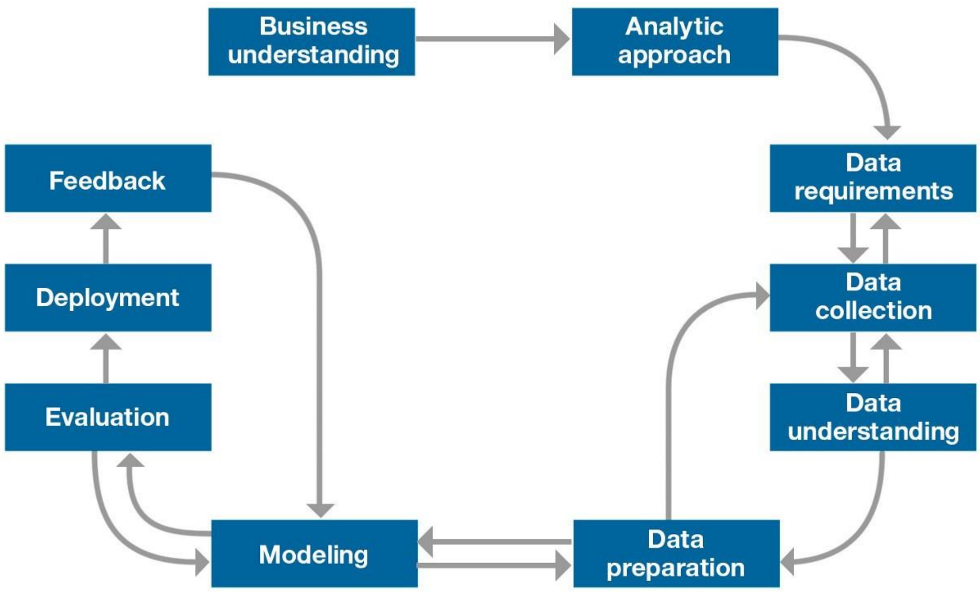


Figure 1.2 The process of methodology

1.3 Import package and data

Using these packages down here to do the analytics for the data

+ *Package for processing analyze data*

- Numpy
- Pandas
- Mathplotlib.pyplot
- Seaborn
- Datetime

+ *Package for prediction*

- From dateutil.relativedelta import relativedelta

- From sklearn.neighbors import KNeighborsClassifier
- From sklearn import svm
- From sklearn.tree import DecisionTreeClassifier
- From sklearn.ensemble import RandomForestClassifier
- From sklearn.metrics import confusion_matrix

\+ *Package for clustering*

- From sklearn.cluster import KMeans
- From scipy.cluster.hierarchy import dendrogram, linkage, fcluster

```
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 1.3: Dataset describe

II. Exploratory data analysis (EDA)

Check data information and describing to know how many columns are numeric, categories and other type. The result gives:

Table 2: Categorize Numerical data and Categorical data

Numeric columns	**Categories columns**
lead_time	hotel
arrival_date_year	is_cancled
arrival_date_week_number	arrival_date_month
arrival_date_day_of_month	meal
stays_in_weekend_nights	country
stays_in_week_nights	market_segment
adults	distribution_channel
Children	is_repeated_guest
Babies	reserved_room_type
previous_cancellations	assigned_room_type
previous_bookings_not_canceled	deposit_type
Agent	company
days_in_waiting_list	customer_type
adr	reservation_status
required_car_parking_spaces	
total_of_special_requests	
booking_changes	

Convert month from category to numeric for easy comparison. Then, combine the day month - year that the guest came in in the format %y-%m-%d.

Convert month type: STR -> INT

```
hotel['arrival_date_month'] = hotel['arrival_date_month'].apply(datetime.strptime,args = ("%B",) )
hotel['arrival_date_month']
```

```
0         1900-07-01
1         1900-07-01
2         1900-07-01
3         1900-07-01
4         1900-07-01
             ...
119385    1900-08-01
119386    1900-08-01
119387    1900-08-01
119388    1900-08-01
119389    1900-08-01
Name: arrival_date_month, Length: 119390, dtype: datetime64[ns]
```

Figure 2.1: Changing from string to integer form of "arrival_date_month"

```
hotel
```

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

119390 rows × 32 columns

Figure 2.2: Final transform "arrival_date_month" to integer

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

```
hotel['arrival_date']
```

```
0         2015-07-01
1         2015-07-01
2         2015-07-01
3         2015-07-01
4         2015-07-01
             ...
119385    2017-08-30
119386    2017-08-31
119387    2017-08-31
119388    2017-08-31
119389    2017-08-29
Name: arrival_date, Length: 119390, dtype: datetime64[ns]
```

Figure 2.3: Transform to "%y - %m - %d" format

Similarly, separate the date-month-year of reservation-status-date.

```
hotel['reservation_status_date'].describe
```

```
<bound method NDFrame.describe of 0         01/07/2015
1         01/07/2015
2         02/07/2015
3         02/07/2015
4         03/07/2015
             ...
119385    06/09/2017
119386    07/09/2017
119387    07/09/2017
119388    07/09/2017
119389    07/09/2017
Name: reservation_status_date, Length: 119390, dtype: object>
```

```
hotel['reservation_status_date'] = pd.to_datetime(hotel['reservation_status_date'], format = '%d/%m/%Y')
hotel['reservation_status_date']
```

```
0        2015-07-01
1        2015-07-01
2        2015-07-02
3        2015-07-02
4        2015-07-03
            ...
119385   2017-09-06
119386   2017-09-07
119387   2017-09-07
119388   2017-09-07
119389   2017-09-07
Name: reservation_status_date, Length: 119390, dtype: datetime64[ns]
```

```
hotel['reservation_month'] = hotel['reservation_status_date'].dt.month
hotel['reservation_day'] = hotel['reservation_status_date'].dt.day
hotel['reservation_year'] = hotel['reservation_status_date'].dt.year
```

Figure 2.4: Separate "reservation_status_date" to "day", "month", "year" invidually columns

Finally, perform new data information with 36 columns adding 4 columns such as arrival_date, reservation_month, reservation_day, reservation_year.

```
 #   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  int64
 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  datetime64[ns]
 32  arrival_date                    119390 non-null  datetime64[ns]
 33  reservation_month               119390 non-null  int64
 34  reservation_day                 119390 non-null  int64
 35  reservation_year                119390 non-null  int64
dtypes: datetime64[ns](2), float64(4), int64(20), object(10)
memory usage: 32.8+ MB
```

Figure 2.5: Final dataset describe

2.1 Missing Data

First, check the total number of null values of the columns, then give the result:

```
adults                               0
children                             4
babies                               0
meal                                 0
country                            488
market_segment                       0
distribution_channel                 0
is_repeated_guest                    0
previous_cancellations               0
previous_bookings_not_canceled       0
reserved_room_type                   0
assigned_room_type                   0
booking_changes                      0
deposit_type                         0
agent                            16340
company                         112593
days_in_waiting_list                 0
```

Figure 2.1.1: Sum of missing value of missing data

Summarize Table:

Columns	Total null
children	4
country	488
agent	16340
company	112593

The first is about the "children" column. After describing value counts, we decided to fill missing value with median of "0" to avoid calculating bias.

```
[ ] children_mean = hotel['children'].mean()
    children_mean

    0.10388990333874994

    children_mode = hotel['children'].mode()
    children_mode

    0    0.0
    dtype: float64

[ ] children_median = hotel['children'].median()
    children_median

    0.0

[ ] hotel["children"].replace(np.nan, children_median, inplace=True)
    hotel['children'].unique()
```

Figure 2.1.2: Mean, Mode, Median of "Children", and replace "NaN"

Secondly, it's about "agent". Similar to "children" we also decided to fill missing value equal to median.

```
agent_mean = hotel['agent'].mean()
agent_mean
```
```
86.69338185346919
```
```
agent_mode = hotel['agent'].mode()
agent_mode
```
```
0    9.0
dtype: float64
```
```
[ ] agent_median = hotel['agent'].median()
    agent_median
```
```
14.0
```
```
[ ] hotel['agent'] = hotel['agent'].fillna(agent_median)
    hotel['agent'].unique()
```

Figure 2.1.3: Mean, Mode. Median of "Agent", and fill the missing value

Thirdly, it's about "country". Since it is a character format, it is impossible to rely on mode or mean because there is no basis, so we decided to fill the missing value by using the previous value as the standard to fill in.

```
hotel['country'].fillna(method='ffill',inplace=True)
hotel['country'].unique()
```

Figure 2.1.4: Fill "Country" column by ffill method

Especially here is the column " company " the total number of null values makes up 112593/119390 percent of the values, making up the majority of the data, so we decided not to manually fill the column with values instead. value "NaN" to value "0" for convenience in normalization step

```
hotel['company'].replace(np.nan, 0, inplace = True)
hotel['company'].unique()
```

Figure 2.1.5: Replace missing value of "Company" column

2.2 Numeric Statistics

	is_canceled	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
count	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000
mean	0.370416	104.011416	2016.156554	6.552483	27.165173	15.798241	0.927599	2.500302
std	0.482918	106.863097	0.707476	3.090619	13.605138	8.780829	0.998613	1.908286
min	0.000000	0.000000	2015.000000	1.000000	1.000000	1.000000	0.000000	0.000000
25%	0.000000	18.000000	2016.000000	4.000000	16.000000	8.000000	0.000000	1.000000
50%	0.000000	69.000000	2016.000000	7.000000	28.000000	16.000000	1.000000	2.000000
75%	1.000000	160.000000	2017.000000	9.000000	38.000000	23.000000	2.000000	3.000000
max	1.000000	737.000000	2017.000000	12.000000	53.000000	31.000000	19.000000	50.000000

adults	children	...	booking_changes	agent	company	days_in_waiting_list	adr	required_car_parking_spaces	total_of_special_requests
119390.000000	119386.000000	...	119390.000000	103050.000000	6797.000000	119390.000000	119390.000000	119390.000000	119390.000000
1.856403	0.103890	...	0.221124	86.693382	189.266735	2.321149	101.831122	0.062518	0.571363
0.579261	0.398561	...	0.652306	110.774548	131.655015	17.594721	50.535790	0.245291	0.792798
0.000000	0.000000	...	0.000000	1.000000	6.000000	0.000000	-6.380000	0.000000	0.000000
2.000000	0.000000	...	0.000000	9.000000	62.000000	0.000000	69.290000	0.000000	0.000000
2.000000	0.000000	...	0.000000	14.000000	179.000000	0.000000	94.575000	0.000000	0.000000
2.000000	0.000000	...	0.000000	229.000000	270.000000	0.000000	126.000000	0.000000	1.000000
55.000000	10.000000	...	21.000000	535.000000	543.000000	391.000000	5400.000000	8.000000	5.000000

Figure 2.2.1: Outlier of "Adr" and "lead_time"

reservation_month	reservation_day	reservation_year
119390.000000	119390.000000	119390.000000
6.334123	15.666639	2016.093743
3.346352	8.778432	0.715306
1.000000	1.000000	2014.000000
3.000000	8.000000	2016.000000
6.000000	16.000000	2016.000000
9.000000	23.000000	2017.000000
12.000000	31.000000	2017.000000

Figure 2.2.2: Outlier of reservation_year

2.3 Outlier Detection

2.3.1 lead_time

```
hotel['lead_time'].describe()
```

```
count    119390.000000
mean        104.011416
std         106.863097
min           0.000000
25%          18.000000
50%          69.000000
75%         160.000000
max         737.000000
Name: lead_time, dtype: float64
```

```
hotel['lead_time'].unique()
```

```
array([342, 737,   7,  13,  14,   0,   9,  85,  75,  23,  35,  68,  18,
        37,  12,  72, 127,  78,  48,  60,  77,  99, 118,  95,  96,  69,
        45,  40,  15,  36,  43,  70,  16, 107,  47, 113,  90,  50,  93,
        76,   3,   1,  10,   5,  17,  51,  71,  63,  62, 101,   2,  81,
       368, 364, 324,  79,  21, 109, 102,   4,  98,  92,  26,  73, 115,
        86,  52,  29,  30,  33,  32,   8, 100,  44,  80,  97,  64,  39,
        34,  27,  82,  94, 110, 111,  84,  66, 104,  28, 258, 112,  65,
        67,  55,  88,  54, 292,  83, 105, 280, 394,  24, 103, 366, 249,
        22,  91,  11, 108, 106,  31,  87,  41, 304, 117,  59,  53,  58,
       116,  42, 321,  38,  56,  49, 317,   6,  57,  19,  25, 315, 123,
        46,  89,  61, 312, 299, 130,  74, 298, 119,  20, 286, 136, 129,
       124, 327, 131, 460, 140, 114, 139, 122, 137, 126, 120, 128, 135,
       150, 143, 151, 132, 125, 157, 147, 138, 156, 164, 346, 159, 160,
       161, 333, 381, 149, 154, 297, 163, 314, 155, 323, 340, 356, 142,
       328, 144, 336, 248, 302, 175, 344, 382, 146, 170, 166, 338, 167,
       310, 148, 165, 172, 171, 145, 121, 178, 305, 173, 152, 354, 347,
       158, 185, 349, 183, 352, 177, 200, 192, 361, 207, 174, 330, 134,
       350, 334, 283, 153, 197, 133, 241, 193, 235, 194, 261, 260, 216,
       169, 209, 238, 215, 141, 189, 187, 223, 284, 214, 202, 211, 168,
       230, 203, 188, 232, 709, 219, 162, 196, 190, 259, 228, 176, 250,
       201, 186, 199, 180, 206, 205, 224, 222, 182, 210, 275, 212, 229,
```

Figure 2.3.1.1: Detection Outlier of "lead_time"

```
sns.histplot(hotel['lead_time'])
```

```
<AxesSubplot:xlabel='lead_time', ylabel='Count'>
```

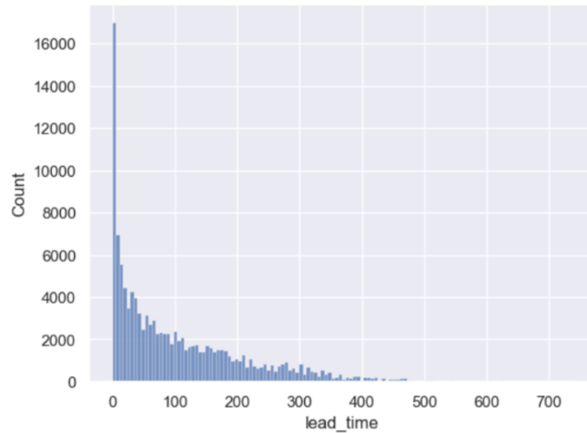


Figure 2.3.1.2: Visualize of "lead_time" column

Lead time booking is unusual here that there are lead time up to more than 1 year.

```
hotel['lead_time'].value_counts()
```

```
0      6345
1      3460
2      2069
3      1816
4      1715
       ...
400       1
370       1
532       1
371       1
463       1
Name: lead_time, Length: 479, dtype: int64
```

Figure 2.3.1.3: Value_counts of "lead_time"

Unusually, when 600-620 is continuous, it jumps to 700. Filter "lead_time" greater than 650 days

```
hotel.loc[lambda df: df['lead_time'] > 650]
```

	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_in_week_nights	adults
1	Resort Hotel	0	737	2015	7	27	1	0	0	2
4182	Resort Hotel	0	709	2016	2	9	25	8	20	2

Figure 2.3.1.4: "lead_time" > 650

2.3.2 arrival_date_year

```
hotel['arrival_date_year'].value_counts()
```

```
2016    56707
2017    40687
2015    21996
Name: arrival_date_year, dtype: int64
```

```
hotel['arrival_date_year'].value_counts(normalize=True)
```

```
2016    0.474973
2017    0.340791
2015    0.184237
Name: arrival_date_year, dtype: float64
```

Figure 2.3.2.1: Value_counts of "arrival_date_year"

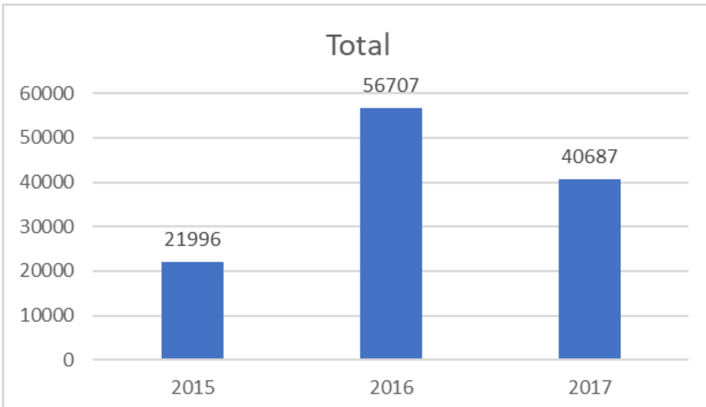


Figure 2.3.2.2: Visualize of "arrival_date_year"

The number of bookings over the years has fluctuated. The 2016 increase compared to 2015 and they in 2017 is less than 2016. However, because the period is from July 2015 to August 2017, only the year of 2016 is full of all booking information in 1 year.

2.3.3 arrival_date_month

```
hotel['arrival_date_month'].value_counts()
```

```
August       13877
July         12661
May          11791
October      11160
April        11089
June         10939
September    10508
March         9794
February      8068
November      6794
December      6780
January       5929
Name: arrival_date_month, dtype: int64
```

```
hotel['arrival_date_month'].value_counts(normalize=True)
```

```
August       0.116233
July         0.106047
May          0.098760
October      0.093475
April        0.092880
June         0.091624
September    0.088014
March        0.082034
February     0.067577
November     0.056906
December     0.056789
January      0.049661
```

Figure 2.3.3.1: Value_counts of "arrival_date_month"

```
sns.histplot(hotel['arrival_date_month'])
```

```
<AxesSubplot:xlabel='arrival_date_month', ylabel='Count'>
```

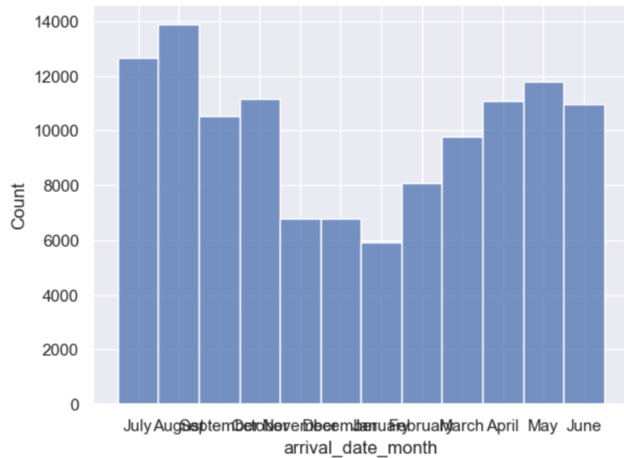
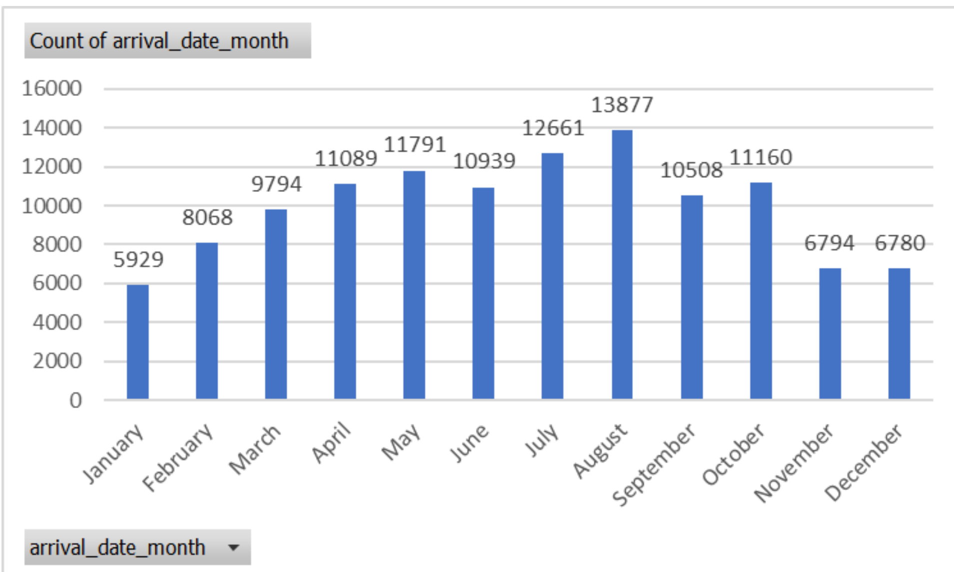


Figure 2.3.3.2: Visualize of "arrival_date_month"



According to the monthly booking chart, it can be seen that many customers book hotels at the end of the second quarter and the beginning of the third quarter of the year, it is considered summer and people tend to travel a lot.

2.3.4 adult

```
hotel['adults'].describe()

count    119390.000000
mean          1.856403
std           0.579261
min           0.000000
25%           2.000000
50%           2.000000
75%           2.000000
max          55.000000
Name: adults, dtype: float64
```

```
hotel['adults'].value_counts()

2     89680
1     23027
3      6202
0       403
4        62
26        5
27        2
20        2
5         2
40        1
50        1
55        1
6         1
10        1
Name: adults, dtype: int64
```

```
hotel['adults'].unique()

array([ 2,  1,  3,  4, 40, 26, 50, 27, 55,  0, 20,  6,  5, 10],
      dtype=int64)
```

Figure 2.3.4.1: Value_counts and describe of "adults"

```
hotel.loc[lambda df: df['adults'] > 39]
```

	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_in_week_nights	adults
1539	Resort Hotel	1	304	2015	9	36	3	0	3	40
1643	Resort Hotel	1	336	2015	9	37	7	1	2	50
2173	Resort Hotel	1	338	2015	10	41	4	2	0	55

Figure 2.3.4.2: "Adults" > 3o

"Adults" represent the normal case when it is possible to be a company group or a tour for more than 40 people.

2.3.5 children

Checking in the "children" column, there is a maximum of 10 anomalies detected. For a better visualization of the data, the image below clearly shows it.

Children

```
hotel['children'].value_counts().sort_index()

0.0     110800
1.0       4861
2.0       3652
3.0         76
10.0         1
Name: children, dtype: int64
```

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

count    119390.000000
mean          0.103886
std           0.398555
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max          10.000000
Name: children, dtype: float64
```

```
hotel.loc[lambda df: df['children'] >3]
```

	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_in_week_nights	adults
328	Resort Hotel	1	55	2015	7	29	12	4	10	2

Figure 2.3.5.1: Value_counts, describe of" children", and" children" >3

"Children" has a case of 10 children but only 2 adults accompany them.

Visualization:

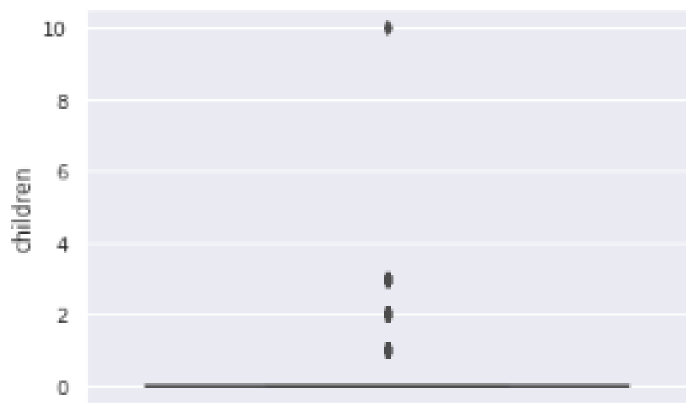


Figure 2.3.5.2: Scatterplot of "children"

2.3.6 babies

Checking in the "Babies" column, the anomaly is detected up to a maximum of 10. To better visualize the data, the following figure clearly shows it.

```
hotel['babies'].value_counts()
```

```
0     118473
1        900
2         15
10         1
9          1
Name: babies, dtype: int64
```

```
hotel.loc[lambda df: df['babies']>2] #bất thường
```

	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_in_week_nights	adults
46619	City Hotel	0	37	2016	1	3	12	0	2	2
78656	City Hotel	0	11	2015	10	42	11	2	1	1

Figure 2.3.6.1: Value_counts of "babies" , and "babies" >2

There are unusual cases when there are 9 and 10 babies while there are only 1 and 2 adults. It is possible that this is a case of a mistyped error.

Visualization:

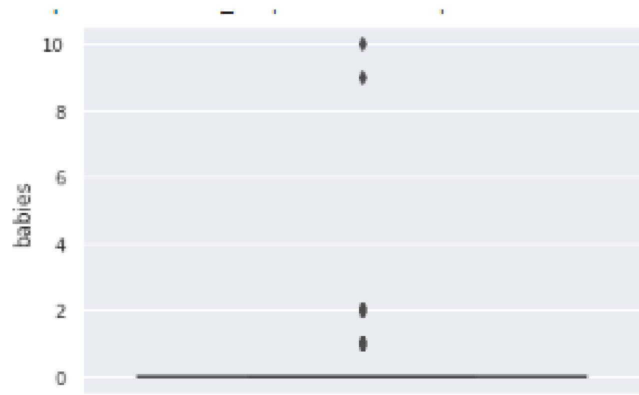


Figure 2.3.6.2: Scatter plot of "babies"

2.3.7 days_in_waiting_list

```python
hotel['days_in_waiting_list'].describe()
```

```
count    119390.000000
mean          2.321149
std          17.594721
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max         391.000000
Name: days_in_waiting_list, dtype: float64
```

```python
hotel['days_in_waiting_list'].value_counts()
```

```
0      115692
39        227
58        164
44        141
31        127
        ...  
116         1
109         1
37          1
89          1
36          1
Name: days_in_waiting_list, Length: 128, dtype: int64
```

Figure 2.3.7.1: Describe and value_counts of "day_in_waiting_list"

```python
sns.histplot(hotel['days_in_waiting_list'])
```

```
<AxesSubplot:xlabel='days_in_waiting_list', ylabel='Count'>
```

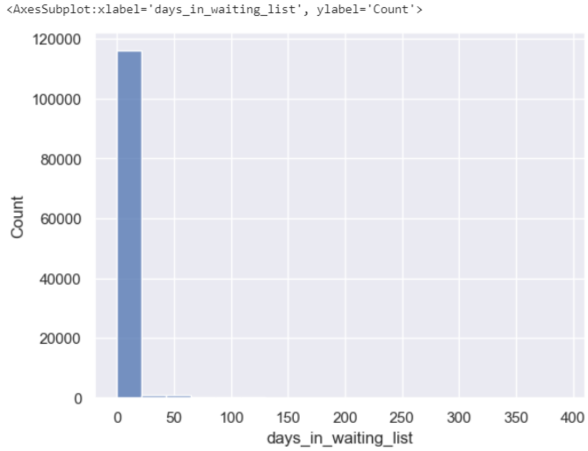


Figure 2.3.7.2: Histogram of "day_in_waiting_list"

```
hotel.loc[lambda df: df['days_in_waiting_list'] > 300]#['is_canceled'].value_counts()
```

	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_in_week_nights	ac
56957	City Hotel	1	422	2016	9	38	16	0	2	
56958	City Hotel	1	422	2016	9	38	16	0	2	
56959	City Hotel	0	422	2016	9	38	16	0	2	
56960	City Hotel	0	422	2016	9	38	16	0	2	
56961	City Hotel	1	422	2016	9	38	16	0	2	
...	...	...	...	...	...	...	...	...	...	
59434	City Hotel	1	464	2016	10	44	28	0	2	
59435	City Hotel	1	464	2016	10	44	28	0	2	
59444	City Hotel	1	464	2016	10	44	28	0	2	
59450	City Hotel	1	464	2016	10	44	28	0	2	
59454	City Hotel	1	464	2016	10	44	28	0	2	

75 rows × 49 columns

Figure 2.3.7.3: "day_in_waiting_list" >300

```
hotel.loc[lambda df: df['days_in_waiting_list'] > 300]['is_canceled'].value_counts()

1    55
0    20
Name: is_canceled, dtype: int64
```

Figure 2.3.7.4: "day_in waiting_list" and "is_canceled" value_counts

The "days_in_waiting_list" column, after checking, can be considered as normal cases and can also be considered the characteristics of guests who cancel rooms that are waiting too long.

2.3.8 adr

Checking in the " adr " column, there is a maximum of 10 anomalies detected. Testing with conditions negative value and value is greater than 600, 2 observations appear. To get a better idea of the data, there is a picture below that clearly shows it.

```
sns.histplot(hotel['adr'])
```

<AxesSubplot:xlabel='adr', ylabel='Count'>

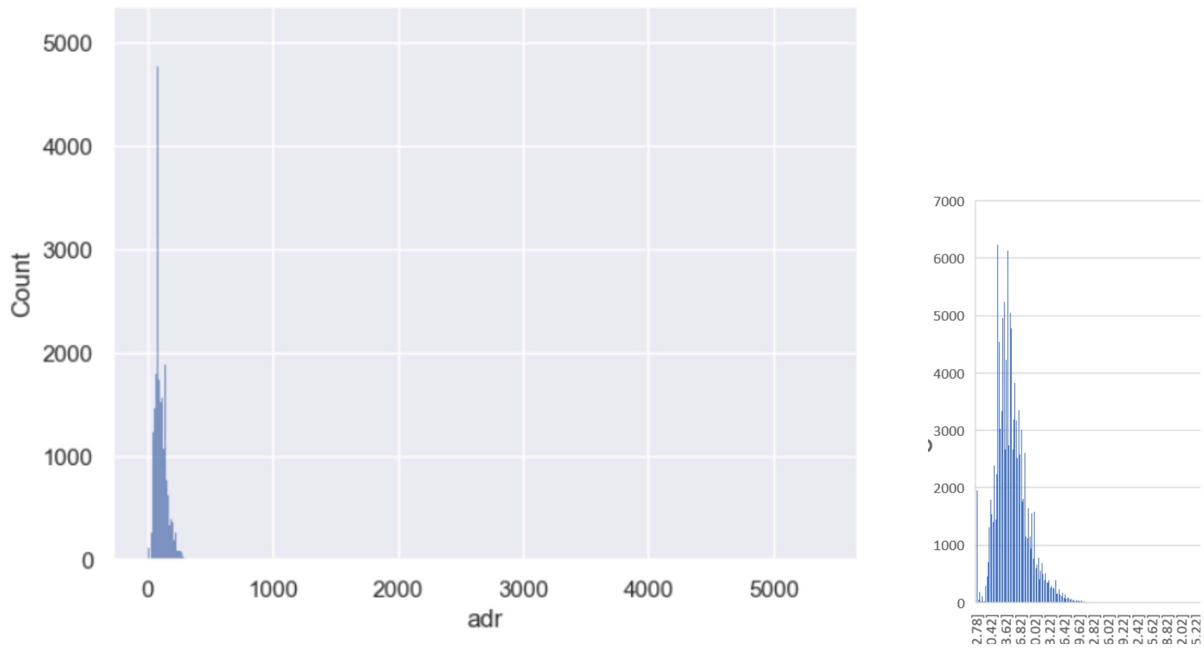


Figure 2.3.8.1: Histogram of "adr"

Hotels provide a variety of accommodation types and dining options. Prices vary widely because a variety of seasonal factors have a role.

```
hotel.loc[lambda df: df['adr'] < 0]
```

	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_in_week_nights
14969	Resort Hotel	0	195	2017	3	10	5	4	6

1 rows × 36 columns

```
hotel.loc[lambda df: df['adr'] > 600]
```

	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_in_week_nights	a
48515	City Hotel	1	35	2016	3	13	25	0	1	

Figure 2.3.8.2: "adr" > 600

"adr" cannot be negative so it will be detected. One case is that the "adr" is too high 5400 while there are only 2 adults and only "stays_in_week_nights" is 1 day which is considered abnormal so it will be removed.

Visualization:

Figure 2.3.8.3: Scatter plot of "adr"

2.3.9 required_car_parking_spaces

Required_car_parking_spaces

```
hotel['required_car_parking_spaces'].describe()
```

```
count    119390.000000
mean          0.062518
std           0.245291
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           8.000000
Name: required_car_parking_spaces, dtype: float64
```

```
hotel.loc[lambda df: df['required_car_parking_spaces'] > 4] #unsual 2 adults but 8 car parking place
```

arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	...	customer_type	adr	required_car_parking_spaces	total_of_special_requests	r
11	14	0	5	2	...	Transient-Party	40.0	8	1	
12	19	2	2	2	...	Transient-Party	80.0	8	0	

Figure 2.3.9.1: "required_car_parking_spaces" > 4

Checking in the column "requirecd_car_parking_spaces" found an anomaly of maximum 8. For a better visualization of the data, the following figure clearly shows it.

Visualization:

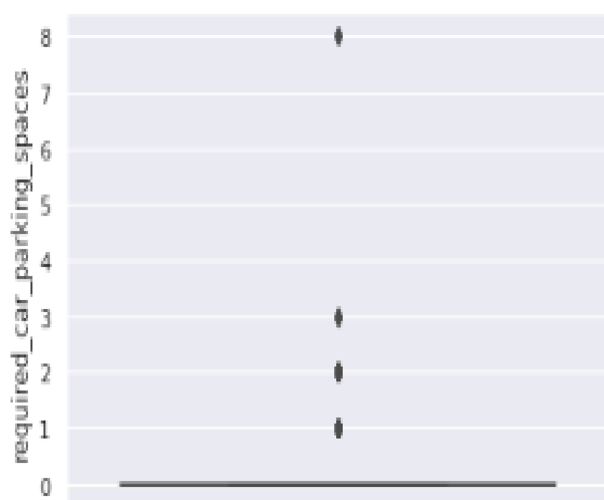


Figure 2.3.9.2: Scatter plot of "required_car_parking_spaces"

There are 2 unusual cases with “required_car_parking_spaces” of 8 seats while there are only 2 adults, so these 2 bookings will be removed.

2.3.10 reservation_status_date

reservation_status_date is the date at which the last status was set. This variable can be used in conjunction with the Reservation Status to understand when the booking was canceled or when the customer checked-out of the hotel.

```
hotel.loc[lambda s: s['reservation_year'] <2015]
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_
1545	Resort Hotel	1	297	2015	9	
73714	City Hotel	1	265	2015	7	
73715	City Hotel	1	258	2015	7	
73716	City Hotel	1	258	2015	7	
73717	City Hotel	1	258	2015	7	
...	...	...	...	...	...	
73890	City Hotel	1	321	2015	9	
73891	City Hotel	1	321	2015	9	
73892	City Hotel	1	321	2015	9	
73893	City Hotel	1	321	2015	9	
73894	City Hotel	1	321	2015	9	

181 rows × 36 columns

Figure 2.3.10.1: “reservation_year” <2015

```
hotel.loc[lambda s: s['reservation_year'] <2015]['reservation_status'].value_counts()

Canceled    181
Name: reservation_status, dtype: int64
```

Figure 2.3.10.2: “reservation_year < 2015” and “reservation_status” value counts

All cases of reservation_status_date is canceled status, so this is considered normal because the customer in 2014 made a reservation for 2015 and made a cancellation in 2014.

2.3.11 reservation_year

After checking all "arrival_year" there are no data lines < 2015 but when checking "reservation_year" there is an amount of data < 2015, but arrival year > reservation year, so I decided This is an anomaly of the dataset

2.3.12 consistency between the arrival date and the reservation status date

After completing the check of the available columns of the data set, we move on to the next step of creating new labels to check the consistency of the arrival - departure - stay dates, or the number of rented car parks compared to number of adults to hire, will depend on the construction logic of the dataset how we will add new labels.

The labels we created:

- "Valid_Check_Out": This label will return a binary value of 0 and 1 under the condition that the required arrival date is less than the check-out date.
- "InValid_Check_Out": This label will be based on the "0" value of "Valid_Check_Out" to find out the wrong cases to remove.
- "number_of_day_stays": This label will be the sum up of "Stays_in_weekend_nights" and "Stays_in_week_nights" to finalize the real validity of check out data.
- "validity": This label will be based on the condition "number_of_day_stays" = '"number_day_in_month_of_reservation_date" and must add the condition "Valid_Check_Out" = 1. This will ensure that the Check-Out data is completely logical and logically correct.
- "Valid_Canceled", "Invalid_Canceled": the same to the "Valid_Check_Out"
- "Valid_No_Show", "Invalid_No_Show": the same to the "Valid_Check_Out"

Adding New Label - Check Validity: arrival datetime < reservation datetime

```
conditions = [
    (hotel['reservation_status'] == "Check-Out") & (hotel['arrival_date_year'] == hotel['reservation_year']) & (hotel['arrival_
]
letters = ['1']
```

```
hotel['Valid_Check_Out'] = np.select(conditions,letters)
hotel['Valid_Check_Out']
```

```
0         0
1         0
2         1
3         1
4         1
         ..
119385    0
119386    0
119387    0
119388    0
119389    0
Name: Valid_Check_Out, Length: 119390, dtype: object
```

Figure 2.3.12.1: Label "Valid_Check_Out"

conditions = [(hotel['reservation_status'] == "Check-Out") & (hotel['arrival_date_year'] == hotel['reservation_year']) & (hotel['arrival_date_month'] <= hotel['reservation_month']) & (hotel['arrival_date_day_of_month'] < hotel['reservation_day'])]

```
hotel['Valid_Check_Out'] = hotel['Valid_Check_Out'].astype(int)
```

```
hotel['Valid_Check_Out'].value_counts()
```

```
1    66273
0    53117
Name: Valid_Check_Out, dtype: int64
```

Invalid_Checkout Label

```
condition = [
         (hotel['reservation_status'] == "Check-Out") & (hotel['Valid_Check_Out'] == 0)

]
letter = ['1']
```

```
hotel['InValid_Check_Out'] = np.select(condition,letter)
```

```
hotel['InValid_Check_Out'] = hotel['InValid_Check_Out'].astype(int)
```

```
hotel['InValid_Check_Out'].value_counts()
```

```
0    110497
1      8893
Name: InValid_Check_Out, dtype: int64
```

Figure 2.3.12.2: Label "InValid_Check_Out"

Valid_Canceled Label

```
condition2 = [
     (hotel['reservation_status'] == "Canceled") & (hotel['arrival_date_year'] == hotel['reservation_year']) & (hotel['
]
letter2 = ['1']
```

```
hotel['Valid_Canceled'] = np.select(condition2,letter2)
```

```
hotel['Valid_Canceled'] = hotel['Valid_Canceled'].astype(int)
```

```
hotel['Valid_Canceled'].value_counts()
```

```
0    84436
1    34954
Name: Valid_Canceled, dtype: int64
```

Invalid_Canceled Label

```
condition3 = [
    (hotel['reservation_status'] == 'Canceled') & (hotel['Valid_Canceled'] == 0)
]
letter3 = ['1']
```

```
hotel['Invalid_Canceled'] = np.select(condition3, letter3)
```

```
hotel['Invalid_Canceled'] = hotel['Invalid_Canceled'].astype(int)
```

```
hotel['Invalid_Canceled'].value_counts()
```

```
0    111327
1      8063
```

Figure 2.3.12.3: Label "Valid_Canceled: and "Invalid_Canceled"

condition2 = [(hotel['reservation_status'] == "Canceled") & (hotel['arrival_date_year'] == hotel['reservation_year']) & (hotel['arrival_date_month'] >= hotel['reservation_month'] & (hotel['arrival_date_day_of_month'] >= hotel['reservation_day']))]

condition3 = [(hotel['reservation_status'] == 'Canceled') & (hotel['Valid_Canceled'] == 0)]

2.3.13 Number of adults compare to the require car parking spaces

The number of adults compared to the enquired parking spaces: We will consider this condition further because there are cases where the parking lot is rented more than the number of adults renting the room. As far as I know, the price of parking in foreign countries is relatively high, so renting an additional parking space doesn't make any sense, so I consider this an anomaly in the data set and decided to be conditional. the number of rented parking spaces must be less than or equal to the number of adults renting the room.

```
x = hotel['required_car_parking_spaces'] >= 0
hotel.loc[x,['required_car_parking_spaces','adults']]
```

	required_car_parking_spaces	adults
0	0	2
1	0	2
2	0	1
3	0	1
4	0	2
...	...	...
119385	0	2
119386	0	3
119387	0	2
119388	0	2
119389	0	2

119390 rows × 2 columns

Figure 2.3.13.1: "Require_car_parking_spaces" vs "adults"

```
hotel.loc[lambda df: df['required_car_parking_spaces'] <= hotel['adults']]
```

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

119374 rows × 49 columns

Figure 2.3.13.2: "Require_car_parking_spaces" <= "adults"

2.3.14 Final detection – Perform new data

Here, we will remove the outliers and will keep the reasonable values to complete the final, and most accurate data set.

Here is the filter condition:

```
hotel_new =(hotel
.loc[lambda df: df['children'] <3]
.loc[lambda df: df['required_car_parking_spaces'] < 8]
.loc[lambda df: df['lead_time'] < 650]
.loc[lambda df: df['adr'] <= 600]
.loc[lambda df: df['adr'] > 0]
.loc[lambda df: df['babies'] <= 2]
.loc[lambda df: df['reservation_year'] > 2014]
.loc[lambda df: df['Invalid_Canceled'] == 0]
.loc[lambda df: df['InValid_Check_Out'] == 0]
.loc[lambda df: df['Invalid_No_Show'] == 0]
)
```

```
hotel_new = hotel_new.loc[lambda df: df['required_car_parking_spaces'] <= hotel_new['adults']]
hotel_new
```

Figure 2.3.14.1: Remove Outlier

The last data set is completed, the number of rows is 101161 and the number of columns is increased to 49

```
hotel_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101161 entries, 2 to 119177
Data columns (total 49 columns):
 #   Column                               Non-Null Count   Dtype
---  ------                               --------------   -----
 0   hotel                                101161 non-null  object
 1   is_canceled                          101161 non-null  int64
 2   lead_time                            101161 non-null  int64
 3   arrival_date_year                    101161 non-null  int64
 4   arrival_date_month                   101161 non-null  int64
 5   arrival_date_week_number             101161 non-null  int64
 6   arrival_date_day_of_month            101161 non-null  int64
 7   stays_in_weekend_nights              101161 non-null  int64
 8   stays_in_week_nights                 101161 non-null  int64
 9   adults                               101161 non-null  int64
 10  children                             101161 non-null  float64
 11  babies                               101161 non-null  int64
 12  meal                                 101161 non-null  object
 13  country                              101161 non-null  object
 14  market_segment                       101161 non-null  object
 15  distribution_channel                 101161 non-null  object
 16  is_repeated_guest                    101161 non-null  int64
 17  previous_cancellations               101161 non-null  int64
 18  previous_bookings_not_canceled       101161 non-null  int64
 19  reserved_room_type                   101161 non-null  object
 20  assigned_room_type                   101161 non-null  object
 21  booking_changes                      101161 non-null  int64
 22  deposit_type                         101161 non-null  object
 23  agent                                101161 non-null  float64
 24  company                              101161 non-null  float64
 25  days_in_waiting_list                 101161 non-null  int64
 26  customer_type                        101161 non-null  object
 27  adr                                  101161 non-null  float64
 28  required_car_parking_spaces          101161 non-null  int64
 29  total_of_special_requests            101161 non-null  int64
 30  reservation_status                   101161 non-null  object
 31  reservation_status_date              101161 non-null  datetime64[ns]
 32  arrival_date                         101161 non-null  datetime64[ns]
 33  reservation_month                    101161 non-null  int64
 34  reservation_day                      101161 non-null  int64
 35  reservation_year                     101161 non-null  int64
 36  Valid_Check_Out                      101161 non-null  int32
 37  InValid_Check_Out                    101161 non-null  int32
 38  Valid_Canceled                       101161 non-null  int32
 39  Invalid_Canceled                     101161 non-null  int32
 40  Valid_No_Show                        101161 non-null  int32
 41  Invalid_No_Show                      101161 non-null  int32
 42  Total_Number_Visitors                101161 non-null  int32
 43  No.#                                 101161 non-null  int64
 44  number_of_day_stays                  101161 non-null  int64
 45  number_day_in_month_of_arrival_date  101161 non-null  int64
 46  number_day_in_month_of_reservation_date  101161 non-null  int64
 47  real_stay_days                       101161 non-null  int64
 48  validity                             101161 non-null  int32
dtypes: datetime64[ns](2), float64(4), int32(8), int64(25), object(10)
```

Figure 2.3.14.2: Hotel_new dataset describe

2.4 Categories Statistics

We also look at all categorical data including hotel, meal, market_segment, distribution_channel, is_repeated_guest, reserved_room_type, assigned_room_type, deposit_type, customer_type, is_repeated_guest, reservation_status

For undefined values, we decided to keep it as there is no basis to remove it and no basis to add another value. So, keeping "Undefined" is reasonable in this case with catergorical data.

2.4.1 hotel

```
hotel['hotel'].value_counts()
```

```
City Hotel      79330
Resort Hotel    40060
Name: hotel, dtype: int64
```

```
hotel['hotel'].value_counts(normalize=True)
```

```
City Hotel      0.664461
Resort Hotel    0.335539
Name: hotel, dtype: float64
```

```
sns.histplot(hotel['hotel'])
```

```
<AxesSubplot:xlabel='hotel', ylabel='Count'>
```

Figure 2.4.1.1: Column "Hotel" value counts and visualization

Customers prefer "City Hotel" over "Resort Hotel".

2.4.2 is_canceled

```
hotel['is_canceled'].value_counts()
```

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

```
hotel['is_canceled'].value_counts(normalize=True)
```

```
0    0.629584
1    0.370416
Name: is_canceled, dtype: float64
```

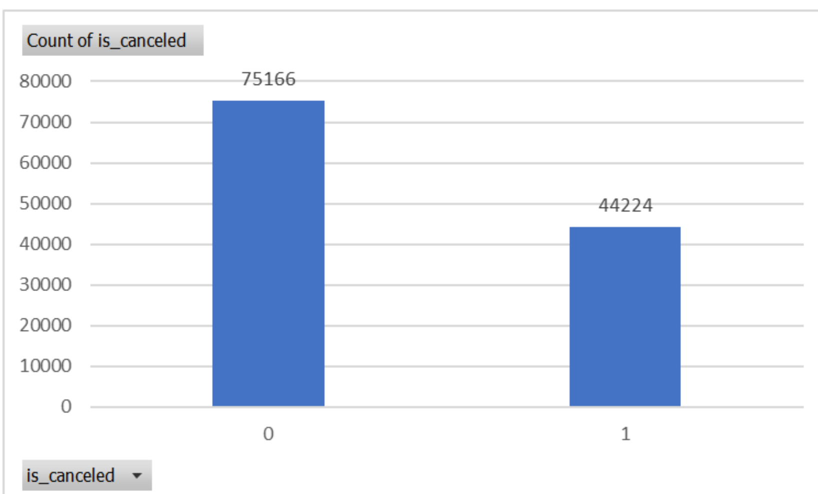


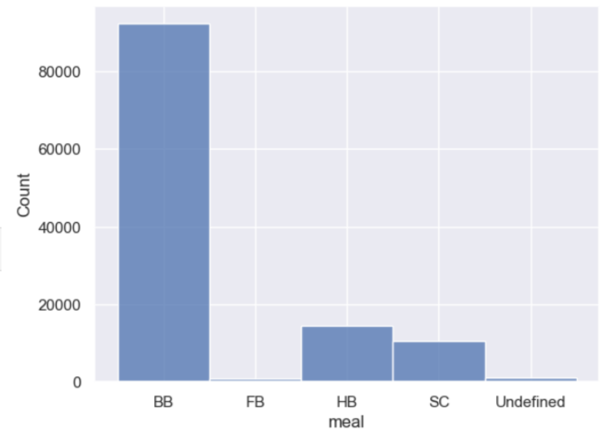
Figure 2.4.2.1: Describe "is_canceled" column

Approximately 37.04% cancellations of the total number of online bookings are quite high. This needs to be addressed to minimize costs and help hoteliers get a good source of revenue.

2.4.3 meal

```
sns.histplot(hotel['meal'])
```

```
<AxesSubplot:xlabel='meal', ylabel='Count'>
```




```
hotel_new['meal'].value_counts()
```

```
BB           77978
HB           12119
SC            9446
Undefined      886
FB             732
Name: meal, dtype: int64
```

Figure 2.4.3.1: Describe" meal" column

Everyone used the BB - Bed & Breakfast meal the most, and the SC - no meal package the least.

2.4.4 country

```
hotel['country'].value_counts()
```

```
PRT    48590
GBR    12129
FRA    10415
ESP     8568
DEU     7287
       ...
DJI        1
BWA        1
HND        1
VGB        1
NAM        1
Name: country, Length: 177, dtype: int64
```

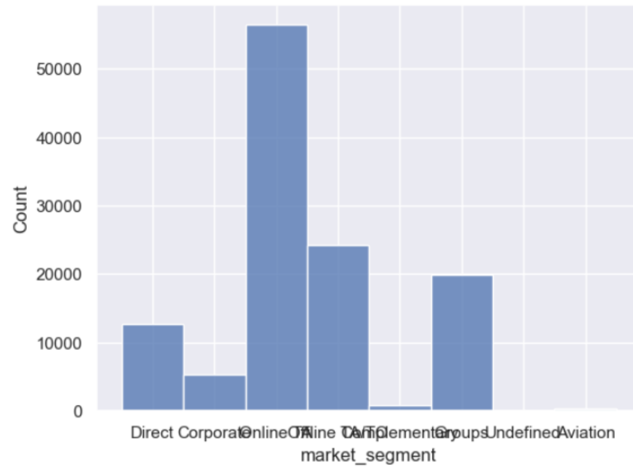
Figure 2.4.4.1: Describe country column

Customers come from different territories and countries.

2.4.5 market_segment

```
sns.histplot(hotel['market_segment'])
```

```
<AxesSubplot:xlabel='market_segment', ylabel='Count'>
```




```
hotel_new['market_segment'].value_counts()
```

```
Online TA        49709
Offline TA/TO    20488
Groups           14752
Direct           11128
Corporate         4811
Aviation           215
Complementary       56
Undefined            2
Name: market_segment, dtype: int64
```

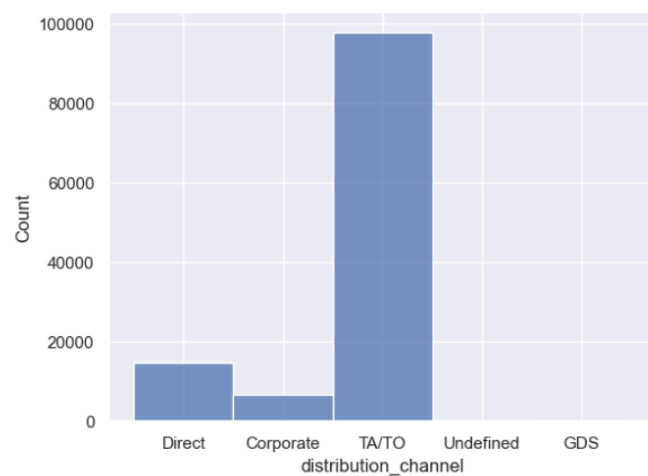
Figure 2.4.5.1: Describe "country" column

Customers prefer to book online more.

2.4.6 distribution_channel

```
sns.histplot(hotel['distribution_channel'])
```

```
<AxesSubplot:xlabel='distribution_channel', ylabel='Count'>
```




```
hotel_new['distribution_channel'].value_counts()
```

```
TA/TO        82780
Direct       12207
Corporate     5986
GDS            183
Undefined        5
Name: distribution_channel, dtype: int64
```

Figure 2.4.6.1: Describe "distribution_channel" column

Large numbers of customers are booked through Travel Agents and Tour Operators. This can prove to be normal for adults up to 40 people.

2.4.7 is_repeated_guest

```
hotel_new['is_repeated_guest'].value_counts()

0    98310
1     2851
Name: is_repeated_guest, dtype: int64
```

```
hotel['is_repeated_guest'].value_counts(normalize=True)

0    0.968088
1    0.031912
Name: is_repeated_guest, dtype: float64
```

Figure 2.4.7.1: Describe "is_repeated_guest" column

It can be seen that there are very few returning customers, only 0.03%. The hotel should check and upgrade the service to be able to attract old customers.

2.4.8 reserved_room_type and assigned_room_type

```
hotel_new['assigned_room_type'].value_counts().sort_index()

A    61978
B     1882
C     2062
D    22297
E     6688
F     3261
G     2116
H      608
I      138
K      130
L        1
Name: assigned_room_type, dtype: int64
```

```
hotel_new['reserved_room_type'].value_counts().sort_index()

A    72545
B      927
C      818
D    16667
E     5465
F     2513
G     1711
H      509
L        6
Name: reserved_room_type, dtype: int64
```

Figure 2.4.8.1: Describe "reserved_room_type" and "assigned_room_type" column

The hotel offers many different types of rooms at different prices.

2.4.9 booking_changes

```
hotel['booking_changes'].value_counts()

0     101314
1      12701
2       3805
3        927
4        376
5        118
6         63
7         31
8         17
9          8
10         6
13         5
14         5
15         3
16         2
17         2
12         2
11         2
20         1
21         1
18         1
Name: booking_changes, dtype: int64
```

```
sns.histplot(hotel['booking_changes'])

<AxesSubplot:xlabel='booking_changes', ylabel='Count'>
```

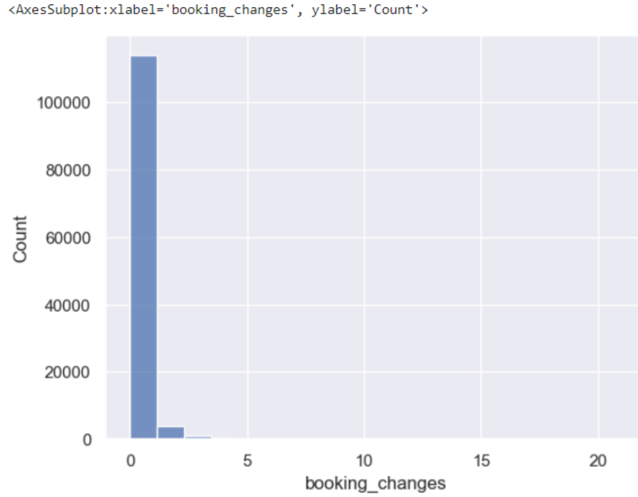
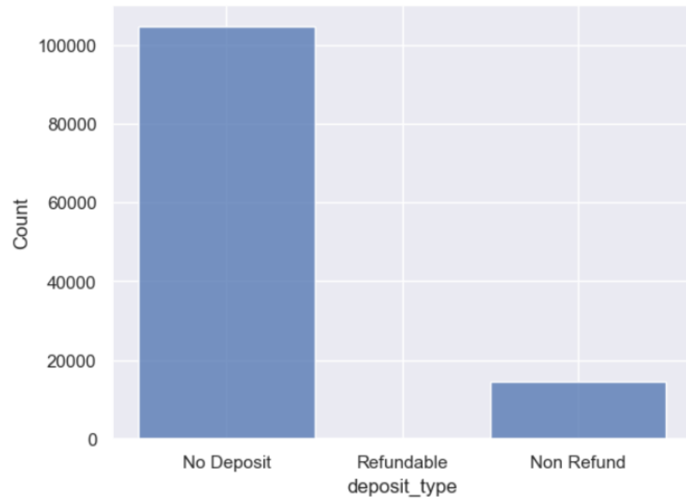


Figure 2.4.9.1: Describe "booking_changes" column

2.4.10 deposit_type

```
sns.histplot(hotel['deposit_type'])
```

```
<AxesSubplot:xlabel='deposit_type', ylabel='Count'>
```




```
hotel_new['deposit_type'].value_counts()
```

```
No Deposit    91327
Non Refund     9750
Refundable       84
Name: deposit_type, dtype: int64
```

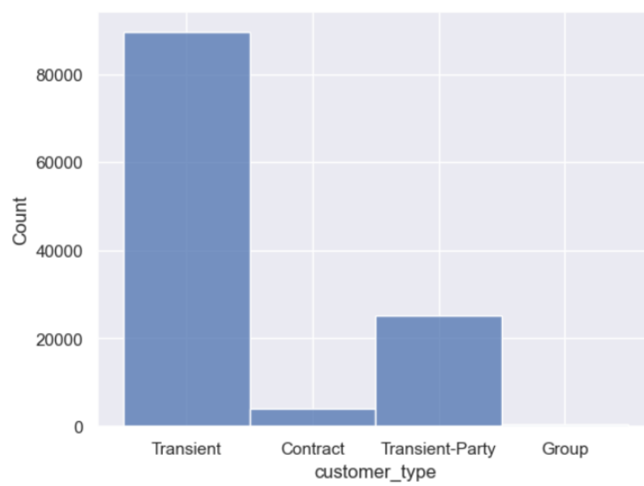
Figure 2.4.10.1: Describe "deposit_type" column

Customers prefer booking without deposit.

2.4.11 customer_type

```
sns.histplot(hotel['customer_type'])
```

```
<AxesSubplot:xlabel='customer_type', ylabel='Count'>
```




```
hotel_new['customer_type'].value_counts()
```

```
Transient          74761
Transient-Party    22412
Contract            3496
Group                492
Name: customer_type, dtype: int64
```

Figure 2.4.11.1: Describe "customer_type" column

Transient customers are the largest and Contract customers are the least.

2.4.12 reservation_status

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

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

```
sns.histplot(hotel['reservation_status'])
```

```
<AxesSubplot:xlabel='reservation_status', ylabel='Count'>
```

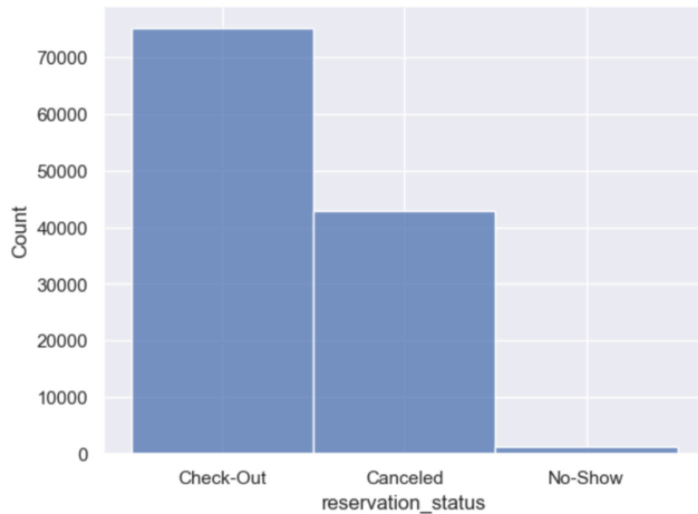


Figure 2.4.12.1: Describe "reservation_status" column

The situation of cancellation before the date of arrival is quite a lot.

III. Modeling

3.1 Featuring Engineering

First to be able to do Z-Normalize, we need to convert all the data to integers. At this point, the new data set can be normalized. To do this, we need one more step called "Feature Engineering".

To do this step we have to select the columns containing the categories that have not been transformed (I set it to onehot_cols, because I will use one_hot coding to convert from categories to dummies values, which is 0 and 1), the columns that you do not. want to transform for formatting and some comparison reasons (I call it meta_cols), and the rest is other_cols. Then we have a completely new dataset, then we will create a separate set consisting of onehot_cols, other_cols and Label is 1 column in "meta_cols"

Featuring Engineering

```
other_cols = ['is_canceled', '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', 'adults', 'children', 'babies','previous_cancellations',
       'previous_bookings_not_canceled','booking_changes', 'agent',
       'company', 'days_in_waiting_list', 'adr',
       'required_car_parking_spaces', 'total_of_special_requests','reservation_month',
       'reservation_day', 'reservation_year','Total_Number_Visitors',
       'number_of_day_stays', 'number_day_in_month_of_arrival_date',
       'number_day_in_month_of_reservation_date', 'real_stay_days', 'is_repeated_guest'
       ]
onehot_cols =['hotel','meal','country','distribution_channel','market_segment','reserved_room_type','assigned_room_type','deposit_type','customer_type
meta_cols = ['No.#', 'arrival_date','reservation_status_date','Valid_Check_Out','Valid_Canceled', 'Invalid_Canceled',
       'Valid_No_Show', 'Invalid_No_Show','InValid_Check_Out', 'validity']
onehot_hotel = pd.get_dummies(hotel_new[onehot_cols])
hotel_clean = pd.concat([
    hotel_new[meta_cols],
    onehot_hotel,hotel_new[other_cols]],axis=1)
hotel_clean.info()
hotel_clean.head()
```

Figure 3.1.1: Feature Engineering

Then normalize with "feat" (the set of onehot_cols and other_cols) according to the formula z-normalize

3.2 Z - Normalization

Z - Normalization

```
hotel_clean[feat].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101161 entries, 2 to 119177
Columns: 251 entries, hotel_City Hotel to is_repeated_guest
dtypes: float64(4), int32(1), int64(24), uint8(222)
memory usage: 44.2 MB
```

```
hotel_clean[feat].columns
```

```
Index(['hotel_City Hotel', 'hotel_Resort Hotel', 'meal_BB', 'meal_FB',
       'meal_HB', 'meal_SC', 'meal_Undefined', 'country_ABW', 'country_AGO',
       'country_AIA',
       ...
       'total_of_special_requests', 'reservation_month', 'reservation_day',
       'reservation_year', 'Total_Number_Visitors', 'number_of_day_stays',
       'number_day_in_month_of_arrival_date',
       'number_day_in_month_of_reservation_date', 'real_stay_days',
       'is_repeated_guest'],
      dtype='object', length=251)
```

```
feat_z = (hotel_clean[feat] - hotel_clean[feat].mean()) / hotel_clean[feat].std()
hotel_z = pd.concat([hotel_clean[meta_cols],feat_z], axis=1)
hotel_z.info()
hotel_z.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101161 entries, 2 to 119177
Columns: 261 entries, No.# to is_repeated_guest
dtypes: datetime64[ns](2), float64(251), int32(7), int64(1)
memory usage: 199.5 MB
```

Figure 3.2.1: Hotel_clean[feat] and hotel_z

```
feat_z = (hotel_clean[feat] - hotel_clean[feat].mean()) / hotel_clean[feat].std()
hotel_z = pd.concat([hotel_clean[meta_cols],feat_z], axis=1)
hotel_z.info()
hotel_z.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101161 entries, 2 to 119177
Columns: 261 entries, No.# to is_repeated_guest
dtypes: datetime64[ns](2), float64(251), int32(7), int64(1)
memory usage: 199.5 MB
```

	No.#	arrival_date	reservation_status_date	Valid_Check_Out	Valid_Canceled	Invalid_Cancel
2	3	2015-07-01	2015-07-02	1	0	
3	4	2015-07-01	2015-07-02	1	0	
4	5	2015-07-01	2015-07-03	1	0	
5	6	2015-07-01	2015-07-03	1	0	
6	7	2015-07-01	2015-07-03	1	0	

5 rows × 261 columns

Figure 3.2.2: hotel_z information

```
hotel_z.describe()
```

	No.#	Valid_Check_Out	Valid_Canceled	Invalid_Canceled	Valid_No_Show	Invali
count	101161.000000	101161.000000	101161.000000	101161.0	101161.000000	
mean	59851.486571	0.644745	0.343601	0.0	0.011655	
std	34781.274899	0.478593	0.474912	0.0	0.107326	
min	3.000000	0.000000	0.000000	0.0	0.000000	
25%	29854.000000	0.000000	0.000000	0.0	0.000000	
50%	58750.000000	1.000000	0.000000	0.0	0.000000	
75%	90720.000000	1.000000	1.000000	0.0	0.000000	
max	119178.000000	1.000000	1.000000	0.0	1.000000	

8 rows × 259 columns

```
hotel_z.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101161 entries, 2 to 119177
Columns: 261 entries, No.# to is_repeated_guest
dtypes: datetime64[ns](2), float64(251), int32(7), int64(1)
memory usage: 199.5 MB
```

Figure 3.2.3: hotel_z describes

IV. Principal component analysis (PCA)

Due to too many columns, we decided to do PCA to find the column containing the most data and important information. Because of the normalization above, the data set has been homogenized above, and we can continue to use the new data set to continue doing PCA.

Then we continue with the next step which is to do the covariance matrix to find out the correlation between the columns of the data set

Covariance matrix: quantity that reflects the degree of linear correlation between two variables and is calculated using the formula. For the first line of code, we decided to keep the columns with formats other than int and float

```
feats = [col for col in hotel_z.columns if col not in ['No.#','reservation_status_date','arrival_date']]
feats
```

```
['Valid_Check_Out',
 'Valid_Canceled',
 'Invalid_Canceled',
 'Valid_No_Show',
 'Invalid_No_Show',
 'InValid_Check_Out',
 'validity',
 'hotel_City Hotel',
 'hotel_Resort Hotel',
 'meal_BB',
 'meal_FB',
 'meal_HB',
 'meal_SC',
 'meal_Undefined',
 'country_ABW',
 'country_AGO',
 'country_AIA',
 'country_ALB',
 'country_AND',
 'country_ARE',
 'country_ARG',
 'country_ARM',
 'country_ASM',
 'country_ATA',
```

Figure 4.1: exclude columns in PCA

```
cov_matrix = np.cov(hotel_z[feats], rowvar=False)
cov_matrix
```

```
array([[ 0.22905129, -0.22153692,  0.        , ..., -0.04140178,
         0.13028356,  0.04257873],
       [-0.22153692,  0.22554152,  0.        , ...,  0.04498904,
        -0.12768355, -0.04310273],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [-0.04140178,  0.04498904,  0.        , ...,  1.        ,
         0.01399933, -0.04573402],
       [ 0.13028356, -0.12768355,  0.        , ...,  0.01399933,
         1.        ,  0.00179307],
       [ 0.04257873, -0.04310273,  0.        , ..., -0.04573402,
         0.00179307,  1.        ]])
```

eigen value and eigen vector

```
eigen_values , eigen_vectors = np.linalg.eig(cov_matrix)
eigen_values = eigen_values.real
eigen_vectors = eigen_vectors.real
```

Figure 4.2: Covariance matrix and create eigen values, eigen vectors

Eigen_values:

```
eigen_values
```

```
array([ 6.40878782e+00,  5.40059191e+00,  4.96623927e+00,  3.89133624e+00,
        3.38490261e+00,  2.71357072e+00,  2.47326178e+00,  2.31136294e+00,
        2.14562225e+00,  1.94332727e+00,  1.90714488e+00,  1.83630068e+00,
        1.81869925e+00,  1.76424945e+00,  1.63633541e+00,  1.62645629e+00,
        1.59933560e+00,  1.54905607e+00,  9.00480104e-02,  9.54650246e-02,
        1.05593239e-01,  1.16240621e-01,  1.63379733e-01,  1.87425093e-01,
        2.23658062e-01,  2.88959149e-01,  3.24804304e-01,  3.05059993e-01,
        3.85695854e-01,  1.43241171e+00,  1.40756587e+00,  1.39294893e+00,
        4.08509765e-01,  3.67307364e-01,  4.43595077e-01,  1.96428687e-03,
        3.18452005e-04,  1.07052091e-04,  4.94517846e-01,  5.05752487e-01,
        6.62221605e-01,  6.30388136e-01,  5.37190924e-01,  5.56910080e-01,
        5.93505851e-01,  5.71302812e-01, -3.22152968e-15, -3.82621507e-15,
        2.05812196e-15,  6.13324733e-16,  6.13324733e-16,  8.19773680e-16,
        4.62069536e-16,  4.62069536e-16, -3.46650369e-17, -2.11066704e-15,
       -1.72383850e-15, -7.95216818e-16, -9.38176226e-16,  1.29376039e+00,
        1.24107552e+00,  7.40342384e-01,  7.93403543e-01,  1.18996070e+00,
        1.15896500e+00,  1.16551724e+00,  1.12270756e+00,  8.15086585e-01,
```

Figure 4.3: Eigen_values

Eigen_vectors:

```
eigen_vectors
```

```
array([[-0.10340321,  0.06211508,  0.12318903, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.10293703, -0.05982522, -0.12218461, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  1.        ,
         0.        ,  0.        ],
       ...,
       [ 0.21297988,  0.33983649, -0.01420188, ...,  0.        ,
         0.        ,  0.        ],
       [-0.09448978,  0.08046541,  0.05348877, ...,  0.        ,
         0.        ,  0.        ],
       [-0.02197417, -0.02395135,  0.15282243, ...,  0.        ,
         0.        ,  0.        ]])
```

Figure 4.4: Eigen_vectors

Sorting eigen_values and eigen_vectores in descending order

```
# sort the eigenvalues in descending order
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalues = eigen_values[sorted_index]

# similarly sort the eigenvectors
sorted_eigenvectors = eigen_vectors[:, sorted_index]
```

```
sorted_index
```

```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  29,  30,  31,  59,  60,  63,  65,  64,
        66,  72,  73,  78,  79,  81,  82,  83,  85,  89,  92,  93,  96,
        97,  99, 100, 102, 104, 105, 106, 107, 108, 114, 115, 116, 118,
       119, 120, 121, 130, 135, 136, 137, 138, 139, 140, 142, 143, 144,
       146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158,
       159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 171, 170,
       172, 173, 176, 177, 175, 174, 179, 180, 181, 182, 183, 184, 185,
       186, 187, 188, 189, 192, 193, 191, 190, 194, 195, 196, 197, 198,
       199, 200, 201, 202, 203, 204, 205, 209, 210, 208, 207, 206, 211,
       212, 213, 214, 215, 216, 218, 217, 219, 220, 224, 221, 225, 222,
       223, 226, 227, 228, 229, 230, 231, 232, 233, 234, 236, 237, 238,
       235, 240, 239, 241, 251, 254, 253, 252, 242, 250, 248, 249, 247,
       243, 244, 246, 245, 178, 145, 141, 134, 133, 132, 131, 129, 128,
       117, 113, 112, 111, 110, 109, 103, 101,  98,  95,  94,  91,  90,
        88,  87,  86,  84,  80,  77,  76,  75,  74,  71,  70,  69,  68,
        67,  62,  61,  40,  41,  44,  45,  43,  42,  39,  38,  34,  32,
        28,  33,  26,  27,  25,  24,  23,  22,  21,  20,  19,  18,  35,
        36,  37,  48,  51,  49,  50,  52,  53, 122, 257, 255, 256,  54,
       126, 125, 124, 127, 123,  57,  58,  56,  55,  46,  47], dtype=int64)
```

Figure 4.5: Sorting in descending order

Plot cumulative explained variance

```
# plot Cumulative Explained Variance
explained_variances_cum = pd.Series(explained_variances).cumsum()
explained_variances_cum.index = ['PC' + str(x+1) for x in explained_variances_cum.index]
explained_variances_cum.plot(kind='bar')
plt.title('Cumulative Explained Variance', size=15);
```

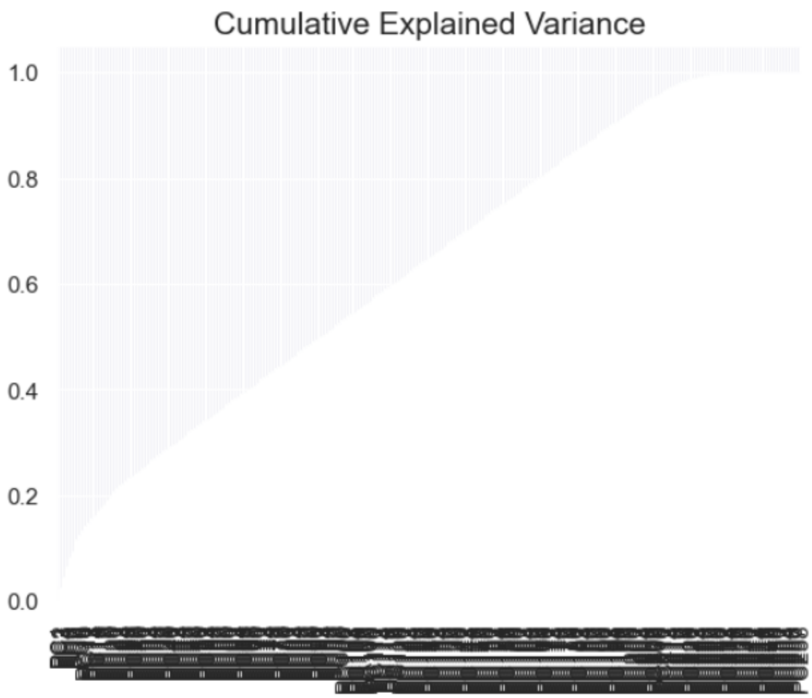


Figure 4.6: Visualization of Cumulative Explained Variance

We find out PCA dataframe

```
# PCA data frame
hotel_pca = hotel_z[feats].dot(sorted_eigenvectors)  # project original data on the principal components
hotel_pca.columns = ['PC' + str(x+1) for x in hotel_pca.columns]  # rename columns
hotel_pca[['No.#']] = hotel_z[['No.#']]
hotel_pca
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	...	PC250	PC251	PC
2	-1.043882	1.743644	3.597238	2.355438	1.008215	1.535136	0.257465	2.598177	-1.112684	-1.507343	...	-0.042547	0.049463	-0.00
3	0.173164	0.793564	5.163778	1.916626	0.726760	-1.895088	2.306118	-1.583642	-0.188132	-0.128052	...	-0.042547	0.049463	-0.00
4	-0.554890	1.491581	0.862608	-0.554016	-0.813348	-0.266346	1.561268	-0.260356	-1.844859	-0.394756	...	-0.042547	0.049463	-0.00
5	-0.554890	1.491581	0.862608	-0.554016	-0.813348	-0.266346	1.561268	-0.260356	-1.844859	-0.394756	...	-0.042547	0.049463	-0.00
6	-1.856173	2.722819	1.799955	4.166288	1.587880	3.045634	0.076141	1.665194	-1.395372	-2.489742	...	-0.042547	0.049463	-0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
119173	-1.622450	0.887257	-0.464528	-1.552317	0.753610	-0.949159	0.228376	0.736078	-1.365142	2.066305	...	-0.042547	0.049463	-0.00
119174	-2.951054	2.190851	-2.040585	-1.147071	1.059136	-0.711767	-1.399646	-1.489437	1.536708	-0.626804	...	-0.042547	0.049463	-0.00
119175	-0.395026	-0.348096	1.122747	-3.769275	0.554105	0.430621	2.255310	0.782229	0.038391	0.525010	...	-0.042547	0.049463	-0.00
119176	1.071397	-0.167565	2.971307	-1.190196	-1.746124	2.207781	-0.390983	-0.581171	0.184908	1.070521	...	-0.042547	0.049463	-0.00
119177	-0.752354	-0.051244	0.601833	-3.946849	0.804496	-0.179920	2.869273	0.777805	-0.162467	0.743718	...	-0.042547	0.049463	-0.00

101161 rows × 259 columns

Figure 4.7: PCA Data frame

Visualization:

About PC1 and PC2

```
sns.scatterplot(
    data= hotel_pca,
    x='PC1',
    y='PC2',
);
```

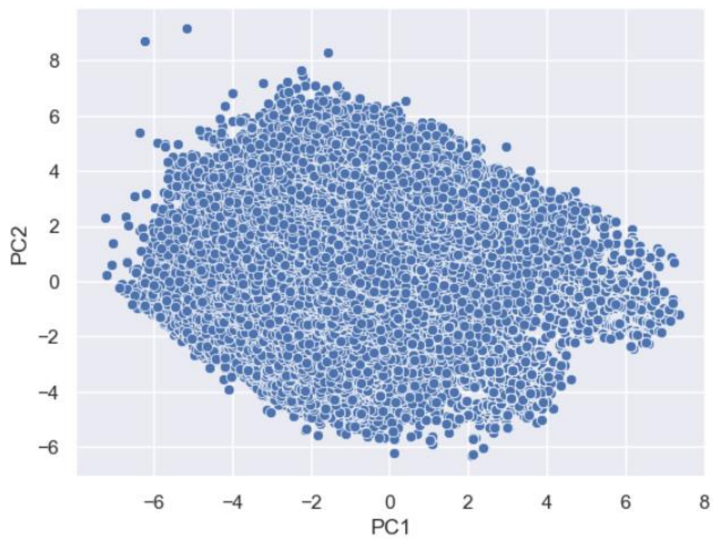


Figure 4.8: Scatter plot of PC1 and PC2

About the "validity"

```
sns.scatterplot(
    data= hotel_pca,
    x='PC1',
    y='PC2',
    hue=hotel_clean['validity']
);
```

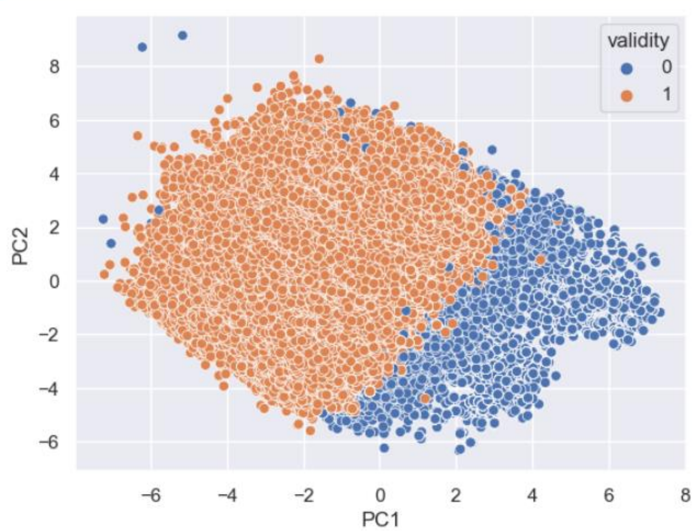


Figure 4.9: "Validity" visualization

Regarding the "validity" column calculation. This column shows the number of customers who have booked and will come to experience the room. According to the chart, more than 60% of customers who book a room will come to experience and check out. This is a good signal of the data set

About "is_repeated_guest"

```
sns.scatterplot(
    data= hotel_pca,
    x='PC1',
    y='PC2',
    hue=hotel_clean['is_repeated_guest']
);
```

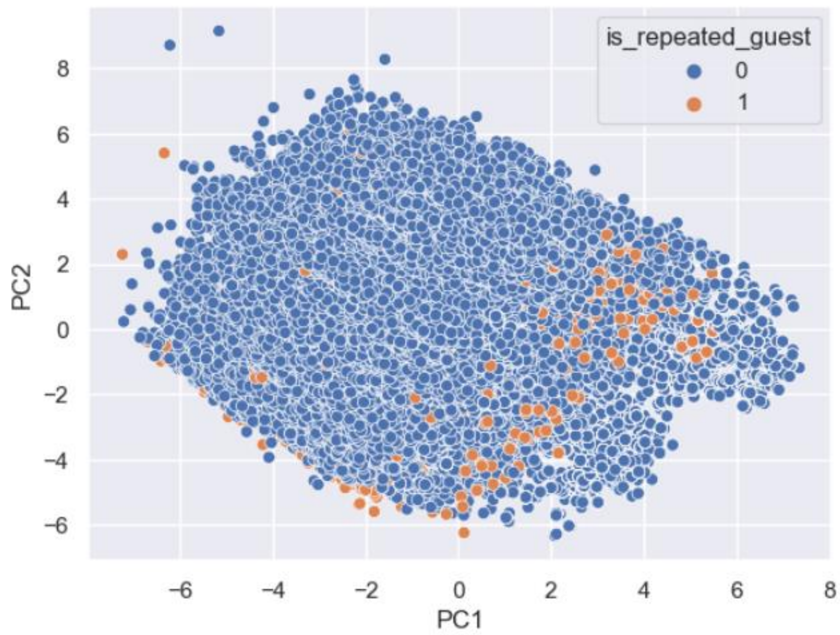


Figure 4.10: "is_repeated_guest" visualization

About calculating column " is_repeated_guest". This column shows the number of loyal customers. As the chart shows, very few points show that loyal customers return to book. This is both a good sign and a bad one. The good sign here is that the hotel has more new customer files, expands the customer file, and the revenue is not stagnant. The bad signal is that the number of loyal customers is too small, leading to customers who only come to experience once and never return.

V. Clustering

We will use the PCA to clustering, and see the visualization through the scatter plot and histogram plot

5.1 K – Mean

K-means clustering is a vector quantization method used to classify given data points into different clusters.

```
# select subset of data
hotel_subset = hotel_pca.set_index('No.#').loc[:, ['PC1', 'PC2']]
hotel_subset
```

No.#	PC1	PC2
3	-1.043882	1.743644
4	0.173164	0.793564
5	-0.554890	1.491581
6	-0.554890	1.491581
7	-1.856173	2.722819
...	...	...
119174	-1.622450	0.887257
119175	-2.951054	2.190851
119176	-0.395026	-0.348096
119177	1.071397	-0.167565
119178	-0.752354	-0.051244

101161 rows × 2 columns

Figure 4.1.1: The subset of PC1 and PC2

Visualization:

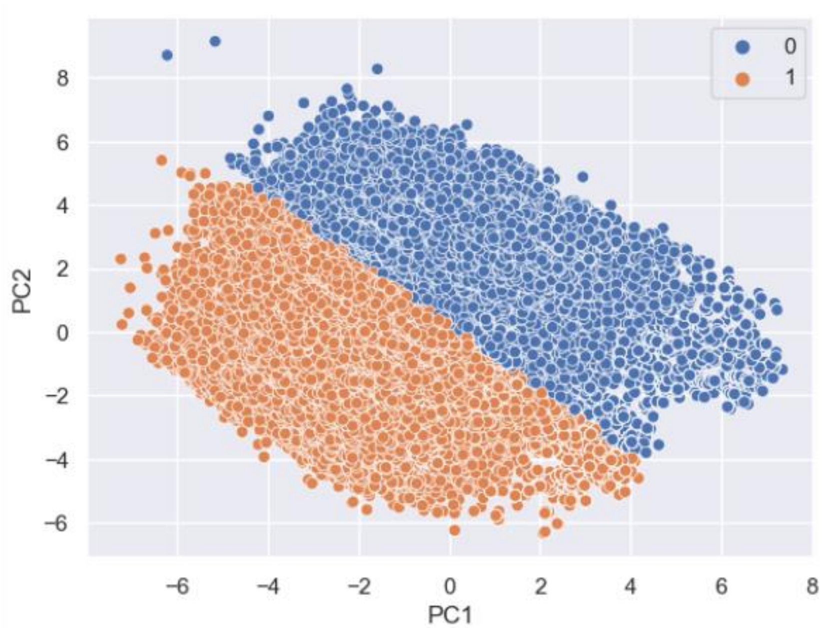


Figure 5.1.2: Visualization of K-Mean when clustering

Parabola figure:

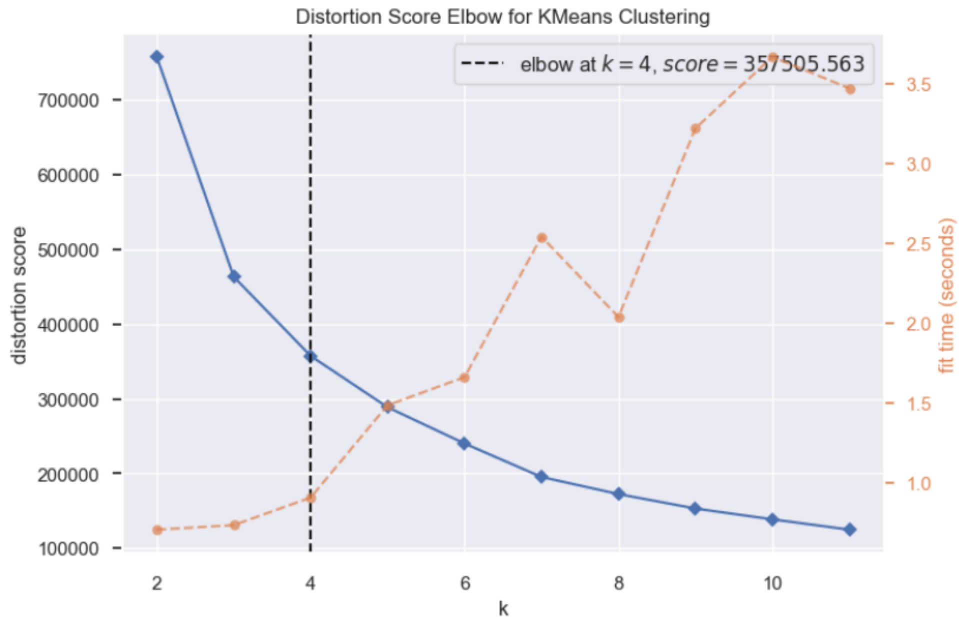


Figure 5.1.3: Parabola of K-Mean Method

With the parabola we will make the decision to choose k on the curvature of the parabola, which is k = 4 (k is the number of cluster).

```
kmeans_fit = KMeans(n_clusters=4, random_state=0)
clr = kmeans_fit.fit_predict(hotel_subset)
# fit model to market_pca
hotel_subset['Cluster No.'] = clr
# fit model to market_clean
hotel_subset['Cluster No.'] = clr
hotel_subset.head()
```

	PC1	PC2	Cluster No.
No.#			
3	-1.043882	1.743644	1
4	0.173164	0.793564	3
5	-0.554890	1.491581	3
6	-0.554890	1.491581	3
7	-1.856173	2.722819	1

Figure 5.1.4: Parabola of K-Mean Method

Classify data in cluster and see the distribute:

```
plt.axes().set_facecolor("white")
pl = sns.countplot(
    x=hotel_subset['Cluster No.'],
    palette = ['firebrick']
)
pl.set_title("Distribution Of The Clusters", fontsize = 15)
pl.set_xlabel("Cluster")
pl.set_ylabel("Count")
plt.savefig('save.png', bbox_inches='tight')
plt.show()
```

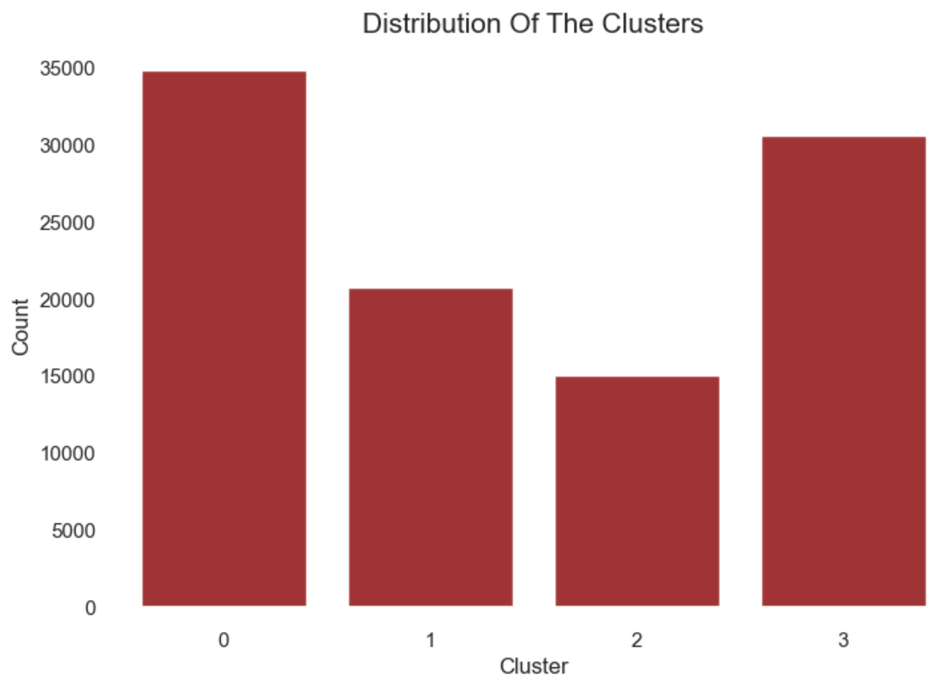


Figure 5.1.5: Parabola of K-Mean Method

We can see clearly that the distribution is quite equal in cluster 0 and 3, in cluster 1 and 2, but when compared 1 and 2 are smaller than the others.

5.2 Hierarchical

In data mining and statistics, hierarchical clustering is a cluster analysis method that aims to build a hierarchy of clusters.

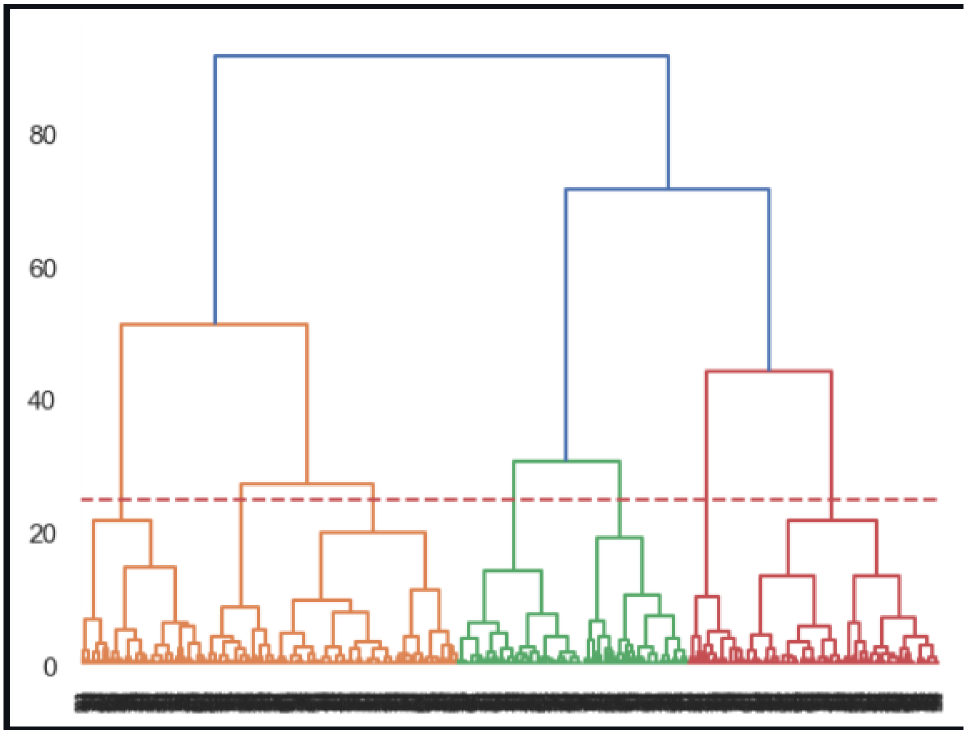


Figure 5.2.1: Hierarchical method plotting with y = 25

Visualization:

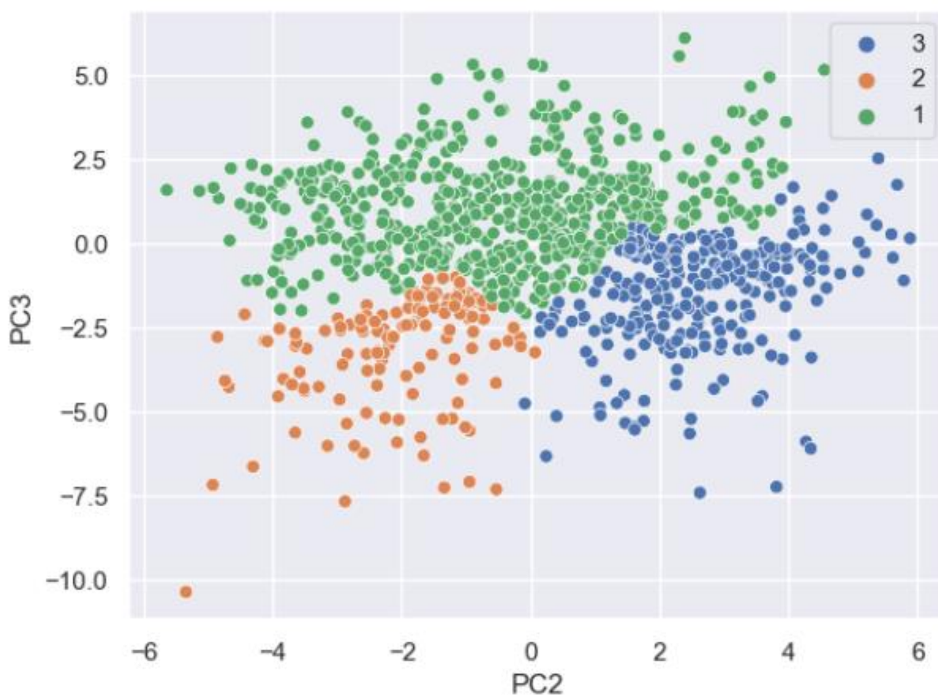


Figure 5.2.2: Scatter plot of subset PC1 and PC2 after clustering

Similar to k-mean, hierarchical divide the data into 3 clusters.

VI. Predicting

For the output of our analysis and prediction, we want to predict the percentage of a customer who makes a reservation, will come to check in and check out (i.e., the percentage that won't cancel the room).

We decided to split the data set into 2 parts:

- *Training data:* is the data set with the number of customers arriving less than December 31, 2016
- *Testing data:* is the data set with the number of daily arrivals greater than or equal to December 31, 2016

Split Data

```
train=hotel_z.loc[lambda df: df['arrival_date']<'2016-12-31']
test=hotel_z.loc[lambda df: df['arrival_date']>='2016-12-31']
print(train.shape)
print(test.shape)
```

```
(68322, 261)
(32839, 261)
```

Figure 6: Split Data

Below are the methods we use for training as well as testing so that we can choose the method with the highest predictability to apply.

Model:

6.1 KNN method

The abbreviation KNN stands for “K-Nearest Neighbour”. It is a supervised machine learning algorithm. The algorithm can be used to solve both classification and regression problem statements. The number of nearest Neighbours to a new unknown variable that has to be predicted or classified is denoted by the symbol 'K'.

Applying in this dataset, and bring out the result in the figure below:

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

✓ 0.4s

	Predict 0	Predict 1
True 0	9029	12149
True 1	4681	33762

Figure 6.1: The result of KNN Method

We will calculate the recall, precision, and accuracy score of KNN Method

```
recall = recall_score(test[label_col], knn_predict)
precision = precision_score(test[label_col], knn_predict)
accuracy = accuracy_score(test[label_col], knn_predict)
print(f"Recal: {recall}")
print(f"Precision: {precision}")
print(f"Accuracy: {accuracy}")
✓ 0.1s

Recal: 0.8782353094191401
Precision: 0.7353793208599246
Accuracy: 0.7177169118263699
```

=> With a high RMSE score, we cannot consider this method for the prediction work.

6.2 SVM method

SVM method is a concept in statistics and computer science for a set of interrelated supervised learning methods for classification and regression analysis. The standard form SVM takes input data and classifies it into two different classes.

Applying in this dataset, and bring out the result in the figure below:

```
svm_cm = confusion_matrix(test[label_col], svm_predict)
svm_cm = pd.DataFrame(svm_cm, index=['True 0', 'True 1'], columns=['Predict 0', 'Predict 1'])
svm_cm
✓ 0.1s
```

	Predict 0	Predict 1
True 0	21175	3
True 1	0	38443

Figure 6.2: The result of SVM Method

We will calculate the recall, precision, and accuracy of SVM Method

```
recall = recall_score(test[label_col], svm_predict)
precision = precision_score(test[label_col], svm_predict)
accuracy = accuracy_score(test[label_col], svm_predict)
print(f"Recal: {recall}")
print(f"Precision: {precision}")
print(f"Accuracy: {accuracy}")
✓ 0.2s

Recal: 1.0
Precision: 0.999921968475264
Accuracy: 0.9999496821589708
```

=> With a very small RMSE score, we can consider this method for the prediction work, and put it on the list. We continue to calculate RMSE and R2 score of all remain methods

6.3 Tree Decision

It is a decision support tool that uses decision tree models and their possible consequences, including chance event outcomes, resource and utility costs. It's a way to show an algorithm that contains only conditional control statements.

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

✓ 0.1s

	Predict 0	Predict 1
True 0	21178	0
True 1	7	38436

Figure 6.3: The result of Tree Decision

We will calculate the recall, precision, and accuracy score of Decision Tree Method

```
recall = recall_score(test[label_col], tree_predict)
precision = precision_score(test[label_col], tree_predict)
accuracy = accuracy_score(test[label_col], tree_predict)
print(f"Recal: {recall}")
print(f"Precision: {precision}")
print(f"Accuracy: {accuracy}")
```

✓ 0.2s

```
Recal: 0.9998179122336966
Precision: 1.0
Accuracy: 0.9998825917042653
```

=> Decision Tree Method has an excellent score in RMSE, which is equal to 0. This mean Decision Tree Method has no wrong predict in prediction work, and we can high consider applying it in the model prediction customer behavior in Check_Out percent.

6.4 Forest Random

A synthetic learning method for classification, regression and other tasks that works by building an infinite number of decision trees at the time of training. For classification tasks, the output of the random forest is the one chosen by most trees.

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

✓ 0.1s

	Predict 0	Predict 1
True 0	11115	6
True 1	0	21718

Figure 6.4: The result of Forest Random

The final one is Forest Method, with

```
recall = recall_score(test[label_col], forest_predict)
precision = precision_score(test[label_col], forest_predict)
accuracy = accuracy_score(test[label_col], forest_predict)
print(f"Recal: {recall}")
print(f"Precision: {precision}")
print(f"Accuracy: {accuracy}")
```

```
✓ 0.3s
```

```
Recal: 1.0
Precision: 0.9997238077702081
Accuracy: 0.9998172904168824
```

=> Forest Method like the SVM Method but has higher 0.005 error

6.5 Summary

Method	KNN	SVM	Tree Decision	Forest Random
Recal	0.8782	1.0	0.9998	1.0
Precision	0.7354	0.9999	1.0	0.9997
Accuracy	0.7177	0.9999	0.9999	0.9998

After the summarize, we will high consider in SVM Method, because it gives highest accuracy, precision and recal.

VII. Feature Importance

We just consider the Tree Decision because we will use it to predict the label we want. In machine learning and statistics, feature extraction is a process of selecting a subset of related attributes for use in model building, and the most important source of information to predict the number of "validity" customers will book and come, is "reservation_status_Check_Out" with a rate of up to 2.862104%

```
pd.Series(index=feat, data=svm_model.coef_[0]).sort_values(ascending=False)  # svm
```

```
reservation_status_Check-Out     2.862104
reservation_month                0.633509
reserved_room_type_F             0.249917
assigned_room_type_F             0.249917
country_FIN                      0.238175
                                   ...
reservation_year                -0.262391
arrival_date_year               -0.262391
reservation_status_No-Show      -1.058653
reservation_status_Canceled     -1.803451
is_canceled                     -2.862104
Length: 251, dtype: float64
```

Figure 7.1: The feature importance of Tree Decision

VIII. Conclusion

Through this dataset, we have gained a more sensitive EDA ability, understand more about the hotel management industry, and in parallel can apply the model to predict customers booking rooms, to produce results. Results what percentage they will come or cancel the room. Although the dataset has a few spots we still have not been able to work through, here's everything we considered and accomplished with the goal of being a "Clean data".

Besides, with this data hotel booking demand, each person will have a different approach, such as forecasting the number of guests in which winter, and which types of customers have high or low-price sensitivity. Our team decided to approach this dataset with a case study of hotel cancellation problems and apply machine learning to solve the problem.

To improve this situation, the hotel first checks the service quality of the room, whether it is clean or not, whether the interior is comfortable or not. Then, hotel owners should pay attention to the price offered for each type of customer coming from many different sources, but still balancing between the levels of service guests choose. Summer is the tourist season, so visitors must focus on this peak time. These things will help businesses build customer trust and satisfy customers, thereby having more customers return. Hotel owners can sell more bookings on online platforms to avoid unexpected cancellations. Besides, we also work with tourist offices to come up with reasonable terms on the contract.

IX. References

1. ANTONIO, Nuno; DE ALMEIDA, Ana; NUNES, Luis. Hotel booking demand datasets. *Data in brief*, 2019, 22: 41-49. https://doi.org/10.1016/j.dib.2018.11.126
2. ANTONIO, Nuno; DE ALMEIDA, Ana; NUNES, Luis. An automated machine learning based decision support system to predict hotel booking cancellations. *An automated machine learning based decision support system to predict hotel booking cancellations*, 2019, 1: 1-20. http://doi.org/10.5334/dsj-2019-032
3. CHEN, Yiying, et al. Comparison and Analysis of Machine Learning Models to Predict Hotel Booking Cancellation. In: *2022 7th International Conference on Financial Innovation and Economic Development (ICFIED 2022)*. Atlantis Press, 2022. p. 1363-1370. https://doi.org/10.2991/aebmr.k.220307.225
4. SATU, Md Shahriare; AHAMMED, Khair; ABEDIN, Mohammad Zoynul. Performance Analysis of Machine Learning Techniques to Predict Hotel booking Cancellations in Hospitality Industry. In: *2020 23rd International Conference on Computer and Information Technology (ICCIT)*. IEEE, 2020. p. 1-6.

https://doi.org/10.1109/ICCIT51783.2020.9392648sss

5. HAENSEL, Alwin; KOOLE, Ger. Booking horizon forecasting with dynamic updating: A case study of hotel reservation data. *International Journal of Forecasting*, 2011, 27.3: 942-960. https://doi.org/10.1016/j.ijforecast.2010.10.004
6. AGAMI, Nedaa, et al. A Futures Studies Tool to Anticipate the Impacts of Wildcards on the Future of the Tourism Industry in Egypt. Int. J. Artif. Intell. Mach. Learn, 2008, 9-14.

Group members

No.	Student's name	Student's ID	Contribution
1	Đào Ngọc Thùy Linh	IELSIU19187	100%
2	Phạm Ngọc Huy	IELSIU19166	100%
3	Tiêu Trí Tịnh	IELSIU19285	80%