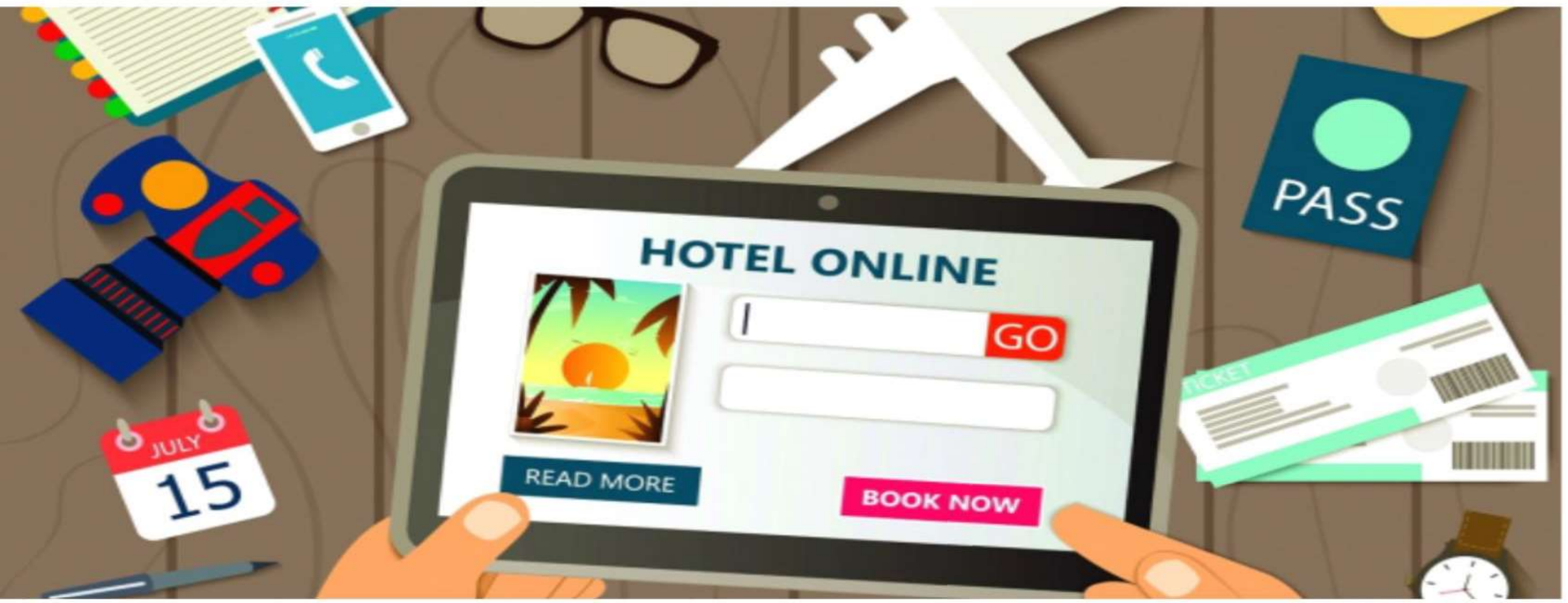
UNIVERSITÀ degli STUDI di CATANIA | DIPARTIMENTO di ECONOMIA e IMPRESA

2022

Hotel Booking



Muhammad Hassan Ali | 1000023619
Syed Muhammad Zaffar | 1000023608

Advance Machine Learning & Knowledge Discovery
Department of Economy and Business
University of Catania

Submitted to:

Prof. Vincenza Carchiolo

7/15/2022

Hotel Booking & Type Analysis

Table Of Content:

DATASET

This data set contains booking information for a city hotel and a resort hotel and includes information 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. All personally identifying information has from the data. We will perform supervised, unsupervised learnings and exploratory data analysis as well with python to get insight from the data.

Attribute Specification

Attribute	Attribute Type	Description
hotel	Categorical	booking information for a city hotel or a resort hotel
is_canceled (Target Variable)	Categorical	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 the arrival date
arrival_date_day_of_month	Integer	Day of the month of the 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 week nights (Monday to Friday) the guest stayed or booked to stay at the hotel
adults	Integer	Number of adults
children	Integer	Number of childern
babies	Integer	Number of babies
meal	Categorical	Type of meal booked. Categories are presented in standard hospitality meal packages: Undefined/SC – no meal package; BB – Bed & Breakfast; HB – Half board (breakfast and one other meal – usually dinner);
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	Categorical	Value indicating if the booking name was from a repeated guest (1) or not (0
previous_cancellations	Integer	Number of previous bookings that were cancelled by the customer prior to the current booking
agent	Categorical	ID of the travel agency that made the 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. Code is presented instead of designation for anonymity reasons
booking_changes	Integer	Number of changes or 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	Indication on if the customer made a deposit to

		guarantee the booking. This variable can assume three categories: No Deposit – no deposit was made; Non Refund – a deposit was made in the value of the total stay cost; Refundable – a deposit was made with a value under the total cost of stay.
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, assuming one of four categories: Contract - when the booking has an allotment or other type of contract associated to it; Group – when the booking is associated to a group; Transient – when the booking is not part of a group or contract, and is not associated to other transient booking; Transient-party – when the booking is transient, but is associated to at least other transient booking
adr	Numeric	Average Daily Rate as defined
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_date	Date	Date at which the last status was set. This variable can be used in conjunction with the Reservation Status to understand when was the booking canceled or when did the customer checked-out of the hotel

The Dataset consists of 14 variables that are categorical type, 16 variables that are of numeric type and 1 variable of Date type.

In [21]:
```
#Importing Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import datetime
from sklearn.preprocessing import LabelEncoder
import plotly.express as px

plt.style.use('fivethirtyeight')
%matplotlib inline
pd.set_option('display.max_columns', 32)
import plotly.express as px


import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
#Importing Data
df = pd.read_csv('hotel_bookings.csv')
```

After the data is loaded, the number of columns and the number of data table rows were viewed. Then, we visualized a few lines of data to get a general idea of this data, and to see if there are any missing values, which will be represented by NaN:

In [53]:
```
df.describe()
```

Out[53]:

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_r
count	119210.000000	119210.000000	119210.000000	119210.000000	119210.000000	119210.0
mean	0.370766	104.109227	2016.156472	27.163376	15.798717	0.9
std	0.483012	106.875450	0.707485	13.601107	8.781070	0.9
min	0.000000	0.000000	2015.000000	1.000000	1.000000	0.0
25%	0.000000	18.000000	2016.000000	16.000000	8.000000	0.0
50%	0.000000	69.000000	2016.000000	28.000000	16.000000	1.0
75%	1.000000	161.000000	2017.000000	38.000000	23.000000	2.0
max	1.000000	737.000000	2017.000000	53.000000	31.000000	19.0

In [54]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 119210 entries, 0 to 119389
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   hotel                           119210 non-null  object 
 1   is_canceled                     119210 non-null  int64  
 2   lead_time                       119210 non-null  int64  
 3   arrival_date_year               119210 non-null  int64  
 4   arrival_date_month              119210 non-null  object 
 5   arrival_date_week_number        119210 non-null  int64  
 6   arrival_date_day_of_month       119210 non-null  int64  
 7   stays_in_weekend_nights         119210 non-null  int64  
 8   stays_in_week_nights            119210 non-null  int64  
 9   adults                          119210 non-null  int64  
 10  children                        119210 non-null  float64
 11  babies                          119210 non-null  int64  
 12  meal                            119210 non-null  object 
 13  country                         119210 non-null  object 
 14  market_segment                  119210 non-null  object 
 15  distribution_channel            119210 non-null  object 
 16  is_repeated_guest               119210 non-null  int64  
 17  previous_cancellations          119210 non-null  int64  
 18  previous_bookings_not_canceled  119210 non-null  int64  
 19  reserved_room_type              119210 non-null  object 
 20  assigned_room_type              119210 non-null  object 
 21  booking_changes                 119210 non-null  int64  
 22  deposit_type                    119210 non-null  object 
 23  agent                           119210 non-null  float64
 24  company                         119210 non-null  float64
 25  days_in_waiting_list            119210 non-null  int64  
 26  customer_type                   119210 non-null  object 
 27  adr                             119210 non-null  float64
 28  required_car_parking_spaces     119210 non-null  int64  
 29  total_of_special_requests       119210 non-null  int64  
 30  reservation_status              119210 non-null  object 
 31  reservation_status_date         119210 non-null  object 
dtypes: float64(4), int64(16), object(12)
memory usage: 30.0+ MB
```

In [18]:
```
# checking for null values

null = pd.DataFrame({'Null Values' : df.isna().sum(), 'Percentage Null Values' : (df.isna().sum()) / (df.
null
```

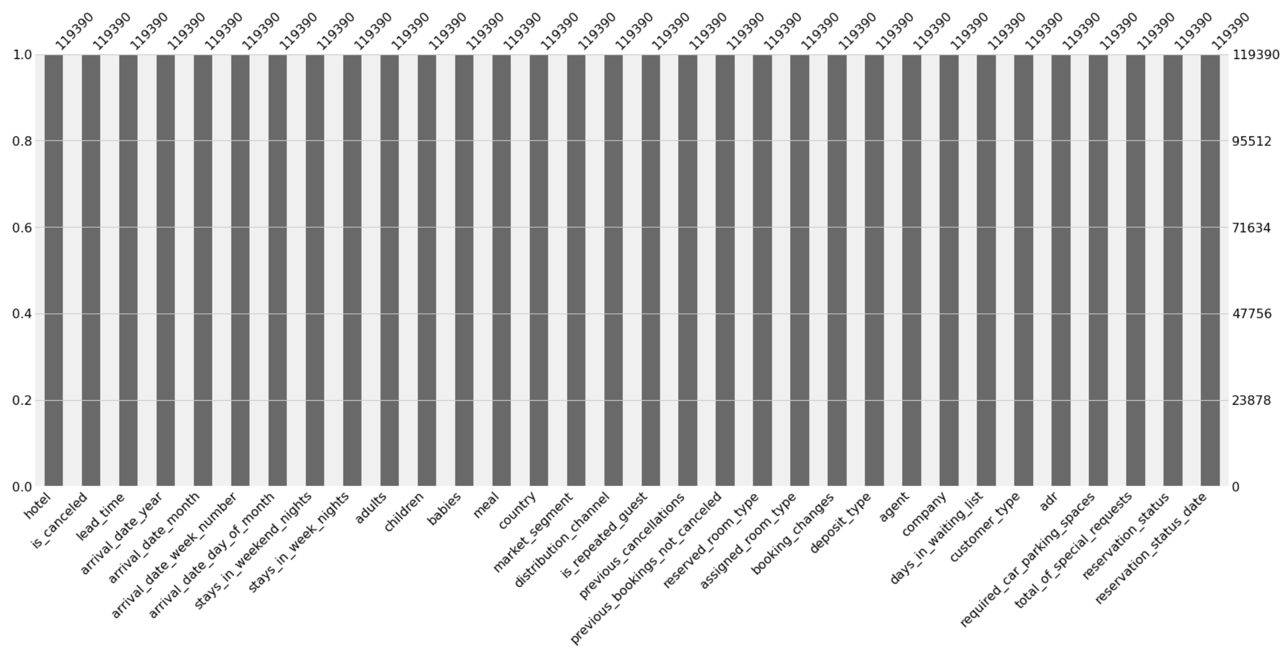
Out[18]:

	Null Values	Percentage Null Values
hotel	0	0.000000
is_canceled	0	0.000000
lead_time	0	0.000000
arrival_date_year	0	0.000000
arrival_date_month	0	0.000000
arrival_date_week_number	0	0.000000
arrival_date_day_of_month	0	0.000000
stays_in_weekend_nights	0	0.000000
stays_in_week_nights	0	0.000000
adults	0	0.000000
children	4	0.003350
babies	0	0.000000
meal	0	0.000000
country	488	0.408744
market_segment	0	0.000000
distribution_channel	0	0.000000
is_repeated_guest	0	0.000000
previous_cancellations	0	0.000000
previous_bookings_not_canceled	0	0.000000
reserved_room_type	0	0.000000
assigned_room_type	0	0.000000
booking_changes	0	0.000000
deposit_type	0	0.000000
agent	16340	13.686238
company	112593	94.306893
days_in_waiting_list	0	0.000000
customer_type	0	0.000000
adr	0	0.000000
required_car_parking_spaces	0	0.000000
total_of_special_requests	0	0.000000
reservation_status	0	0.000000
reservation_status_date	0	0.000000

In [19]:
```
# filling null values with zero

df.fillna(0, inplace = True)
```

In [22]:
```
# visualizing null values
msno.bar(df)
plt.show()
```



In [23]:
```
# adults, babies and children cant be zero at same time, so dropping the rows having all these zero at sa

filter = (df.children == 0) & (df.adults == 0) & (df.babies == 0)
df[filter]
```

Out[23]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_mont
2224	Resort Hotel	0	1	2015	October	41	
2409	Resort Hotel	0	0	2015	October	42	1
3181	Resort Hotel	0	36	2015	November	47	2
3684	Resort Hotel	0	165	2015	December	53	3
3708	Resort Hotel	0	165	2015	December	53	3
...	...	...	...	...	...	...	.
115029	City Hotel	0	107	2017	June	26	2
115091	City Hotel	0	1	2017	June	26	3
116251	City Hotel	0	44	2017	July	28	1
116534	City Hotel	0	2	2017	July	28	1
117087	City Hotel	0	170	2017	July	30	2

180 rows × 32 columns

In [24]:
```
df = df[~filter]
df
```

Out[24]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_mont
0	Resort Hotel	0	342	2015	July	27	
1	Resort Hotel	0	737	2015	July	27	
2	Resort Hotel	0	7	2015	July	27	
3	Resort Hotel	0	13	2015	July	27	
4	Resort Hotel	0	14	2015	July	27	
...	...	...	...	...	...	...	.
119385	City Hotel	0	23	2017	August	35	3
119386	City Hotel	0	102	2017	August	35	3
119387	City Hotel	0	34	2017	August	35	3
119388	City Hotel	0	109	2017	August	35	3
119389	City Hotel	0	205	2017	August	35	2

119210 rows × 32 columns

Exploratory Data Analysis

- From where the most guests are coming ?

In [28]:
```
country_wise_guests = df[df['is_canceled'] == 0]['country'].value_counts().reset_index()
country_wise_guests.columns = ['country', 'No of guests']
country_wise_guests
```

Out[28]:

	country	No of guests
0	PRT	20977
1	GBR	9668
2	FRA	8468
3	ESP	6383
4	DEU	6067
...	...	...
161	KIR	1
162	SDN	1
163	PYF	1
164	ATF	1
165	FRO	1

166 rows × 2 columns

People from all over the world are staying in these two hotels. Most guests are from Portugal and other countries in Europe.

- How much do guests pay for a room per night?

In [30]:
```
df.head()
```

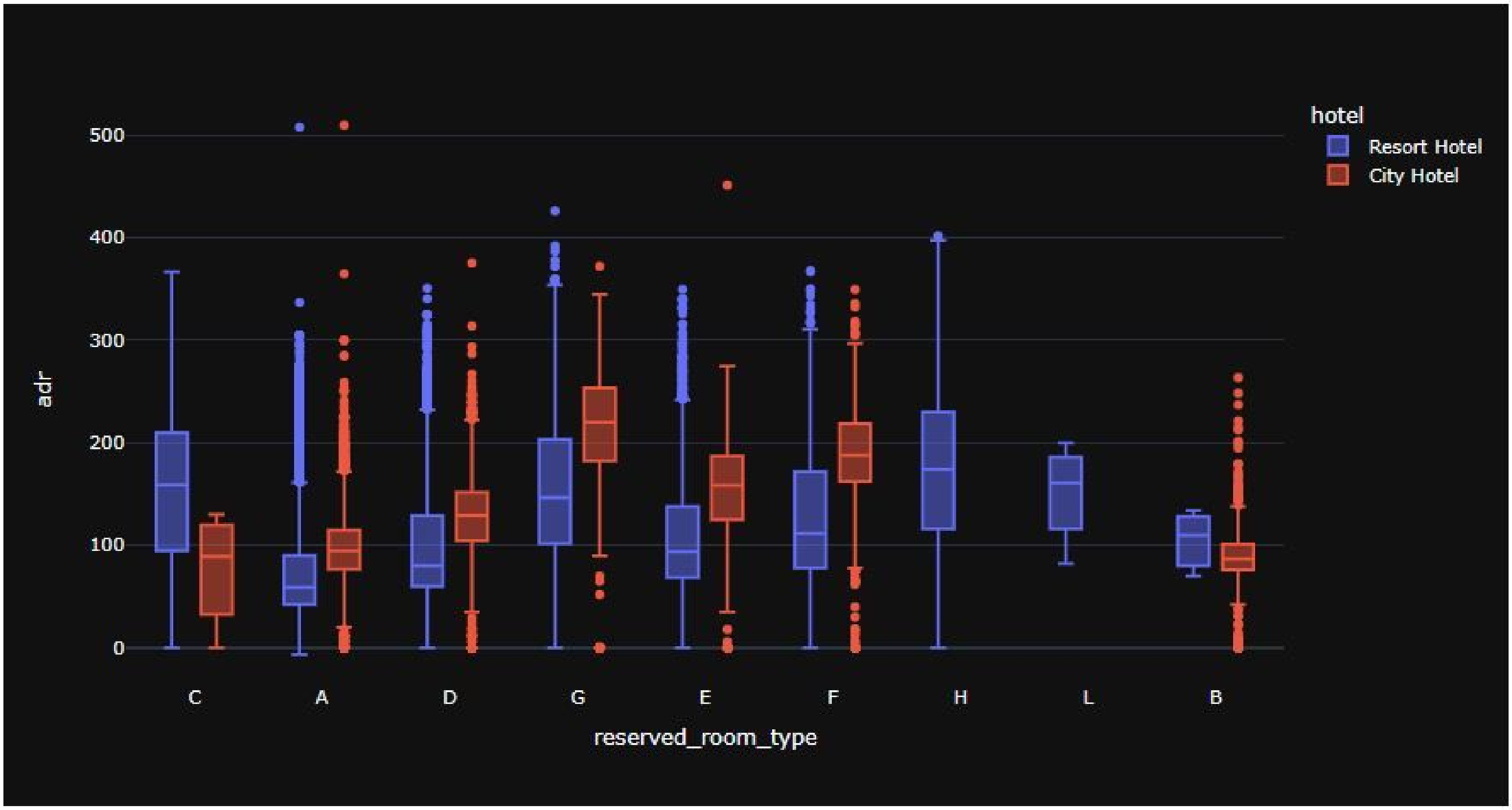
Out[30]:

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

Both hotels have different room types and different meal arrangements.Seasonal factors are also important, So the prices varies a lot.

In [31]:
```
data = df[df['is_canceled'] == 0]

px.box(data_frame = data, x = 'reserved_room_type', y = 'adr', color = 'hotel', template = 'plotly_dark')
```



The figure shows that the average price per room depends on its type and the standard deviation.

- How does the price vary per night over the year?

In [34]:
```
data_resort = df[(df['hotel'] == 'Resort Hotel') & (df['is_canceled'] == 0)]
data_city = df[(df['hotel'] == 'City Hotel') & (df['is_canceled'] == 0)]
```

In [35]:
```
resort_hotel = data_resort.groupby(['arrival_date_month'])['adr'].mean().reset_index()
resort_hotel
```

Out[35]:

	arrival_date_month	adr
0	April	75.867816
1	August	181.205892
2	December	68.410104
3	February	54.147478
4	January	48.761125
5	July	150.122528
6	June	107.974850
7	March	57.056838
8	May	76.657558
9	November	48.706289
10	October	61.775449
11	September	96.416860

In [36]:
```
city_hotel=data_city.groupby(['arrival_date_month'])['adr'].mean().reset_index()
city_hotel
```

Out[36]:

	arrival_date_month	adr
0	April	111.962267
1	August	118.674598
2	December	88.401855
3	February	86.520062
4	January	82.330983
5	July	115.818019
6	June	117.874360
7	March	90.658533
8	May	120.669827
9	November	86.946592
10	October	102.004672
11	September	112.776582

In [37]:
```
final_hotel = resort_hotel.merge(city_hotel, on = 'arrival_date_month')
final_hotel.columns = ['month', 'price_for_resort', 'price_for_city_hotel']
final_hotel
```

Out[37]:

	month	price_for_resort	price_for_city_hotel
0	April	75.867816	111.962267
1	August	181.205892	118.674598
2	December	68.410104	88.401855
3	February	54.147478	86.520062
4	January	48.761125	82.330983
5	July	150.122528	115.818019
6	June	107.974850	117.874360
7	March	57.056838	90.658533
8	May	76.657558	120.669827
9	November	48.706289	86.946592
10	October	61.775449	102.004672
11	September	96.416860	112.776582

Now we observe here that month column is not in order, and if we visualize we will get improper conclusions.

So, first we have to provide right hierarchy to month column.

In [39]:
```
import sort_dataframeby_monthorweek as sd

def sort_month(df, column_name):
    return sd.Sort_Dataframeby_Month(df, column_name)
```

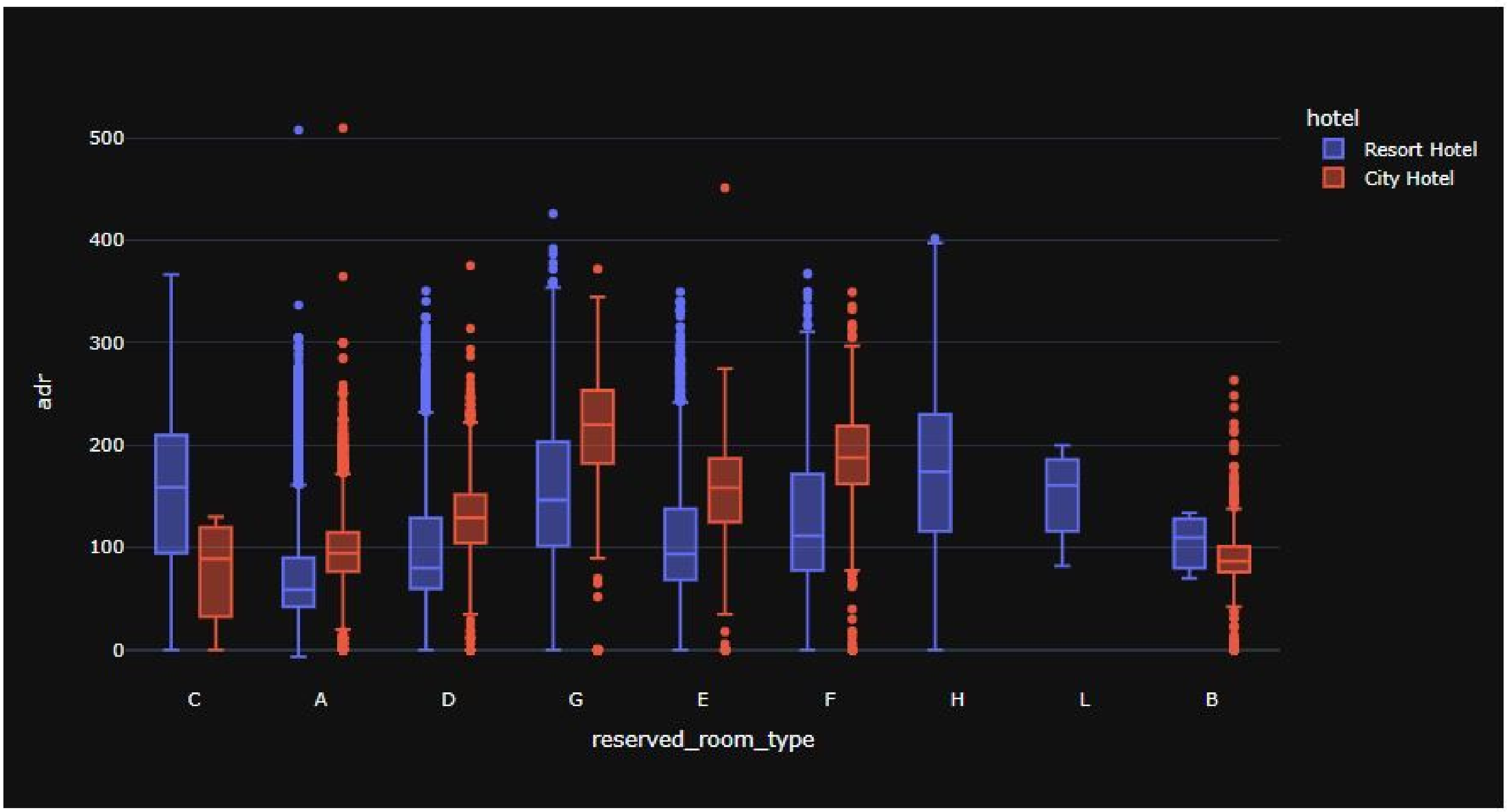
In [40]:
```
final_prices = sort_month(final_hotel, 'month')
final_prices
```

Out[40]:

	month	price_for_resort	price_for_city_hotel
0	January	48.761125	82.330983
1	February	54.147478	86.520062
2	March	57.056838	90.658533
3	April	75.867816	111.962267
4	May	76.657558	120.669827
5	June	107.974850	117.874360
6	July	150.122528	115.818019
7	August	181.205892	118.674598
8	September	96.416860	112.776582
9	October	61.775449	102.004672
10	November	48.706289	86.946592
11	December	68.410104	88.401855

In [41]:
```
plt.figure(figsize = (17, 8))

px.line(final_prices, x = 'month', y = ['price_for_resort','price_for_city_hotel'],
        title = 'Room price per night over the Months', template = 'plotly_dark')
```




```
<Figure size 1224x576 with 0 Axes>
```

This plot clearly shows that prices in the Resort Hotel are much higher during the summer and prices of city hotel varies less and is most expensive during Spring and Autumn .

- Which are the most busy months?

In [43]:
```
resort_guests = data_resort['arrival_date_month'].value_counts().reset_index()
resort_guests.columns=['month','no of guests']
resort_guests
```

Out[43]:

	month	no of guests
0	August	3257
1	July	3137
2	October	2575
3	March	2571
4	April	2550
5	May	2535
6	February	2308
7	September	2102
8	June	2037
9	December	2014
10	November	1975
11	January	1866

In [44]:
```
city_guests = data_city['arrival_date_month'].value_counts().reset_index()
city_guests.columns=['month','no of guests']
city_guests
```

Out[44]:

	month	no of guests
0	August	5367
1	July	4770
2	May	4568
3	June	4358
4	October	4326
5	September	4283
6	March	4049
7	April	4010
8	February	3051
9	November	2676
10	December	2377
11	January	2249

In [45]:
```
final_guests = resort_guests.merge(city_guests,on='month')
final_guests.columns=['month','no of guests in resort','no of guest in city hotel']
final_guests
```

Out[45]:

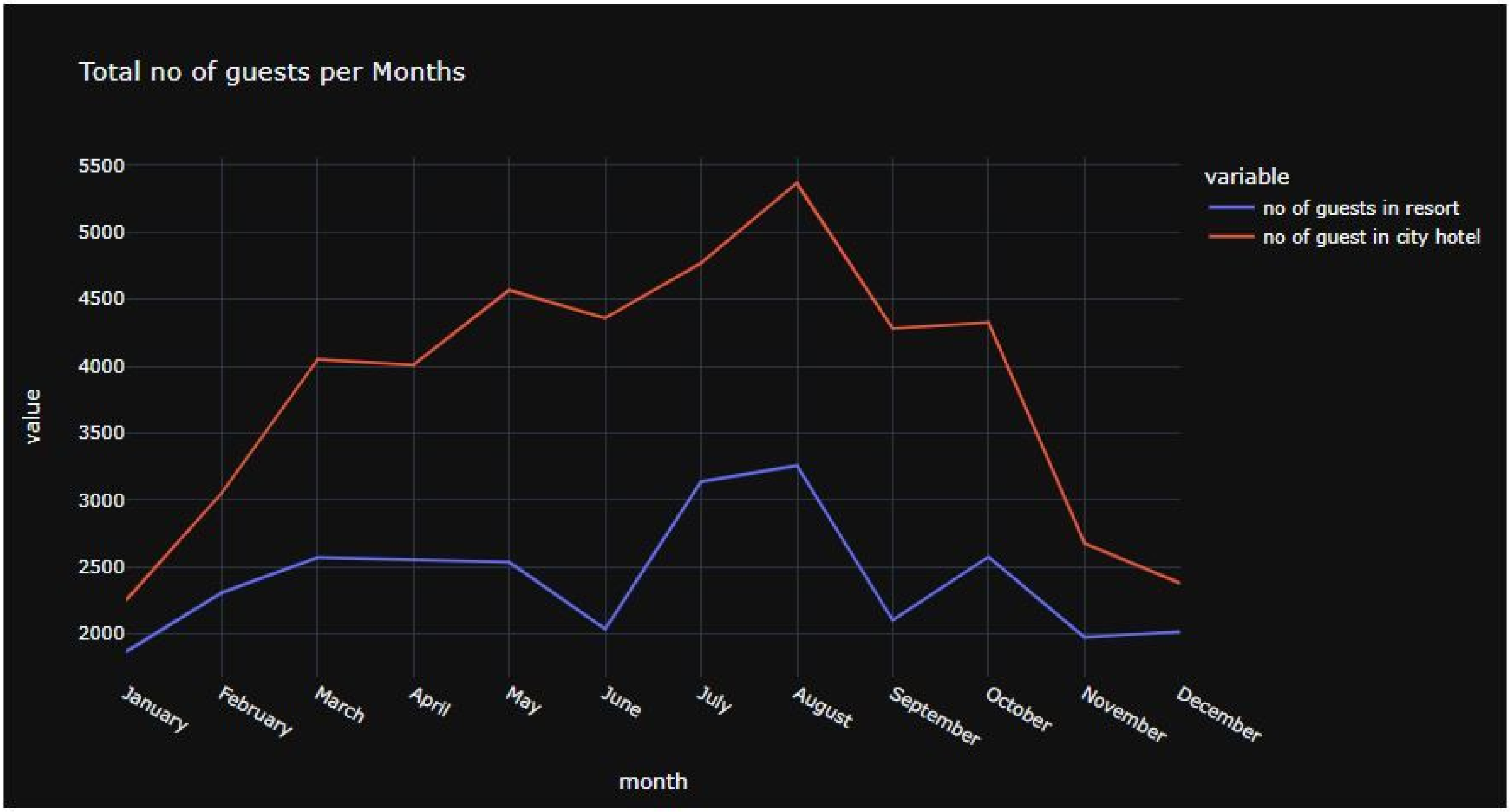
	month	no of guests in resort	no of guest in city hotel
0	August	3257	5367
1	July	3137	4770
2	October	2575	4326
3	March	2571	4049
4	April	2550	4010
5	May	2535	4568
6	February	2308	3051
7	September	2102	4283
8	June	2037	4358
9	December	2014	2377
10	November	1975	2676
11	January	1866	2249

In [46]:
```
final_guests = sort_month(final_guests,'month')
final_guests
```

Out[46]:

	month	no of guests in resort	no of guest in city hotel
0	January	1866	2249
1	February	2308	3051
2	March	2571	4049
3	April	2550	4010
4	May	2535	4568
5	June	2037	4358
6	July	3137	4770
7	August	3257	5367
8	September	2102	4283
9	October	2575	4326
10	November	1975	2676
11	December	2014	2377

In [47]:
```
px.line(final_guests, x = 'month', y = ['no of guests in resort','no of guest in city hotel'],
        title='Total no of guests per Months', template = 'plotly_dark')
```



- The City hotel has more guests during spring and autumn, when the prices are also highest, In July and August there are less visitors, although prices are lower.
- Guest numbers for the Resort hotel go down slighty from June to September, which is also when the prices are highest. Both hotels have the fewest guests during the winter.

- How long do people stay at the hotels?

In [48]:
```
filter = df['is_canceled'] == 0
data = df[filter]
data.head()
```

Out[48]:

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

In [49]:
```
data['total_nights'] = data['stays_in_weekend_nights'] + data['stays_in_week_nights']
data.head()
```

Out[49]:

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

5 rows × 33 columns

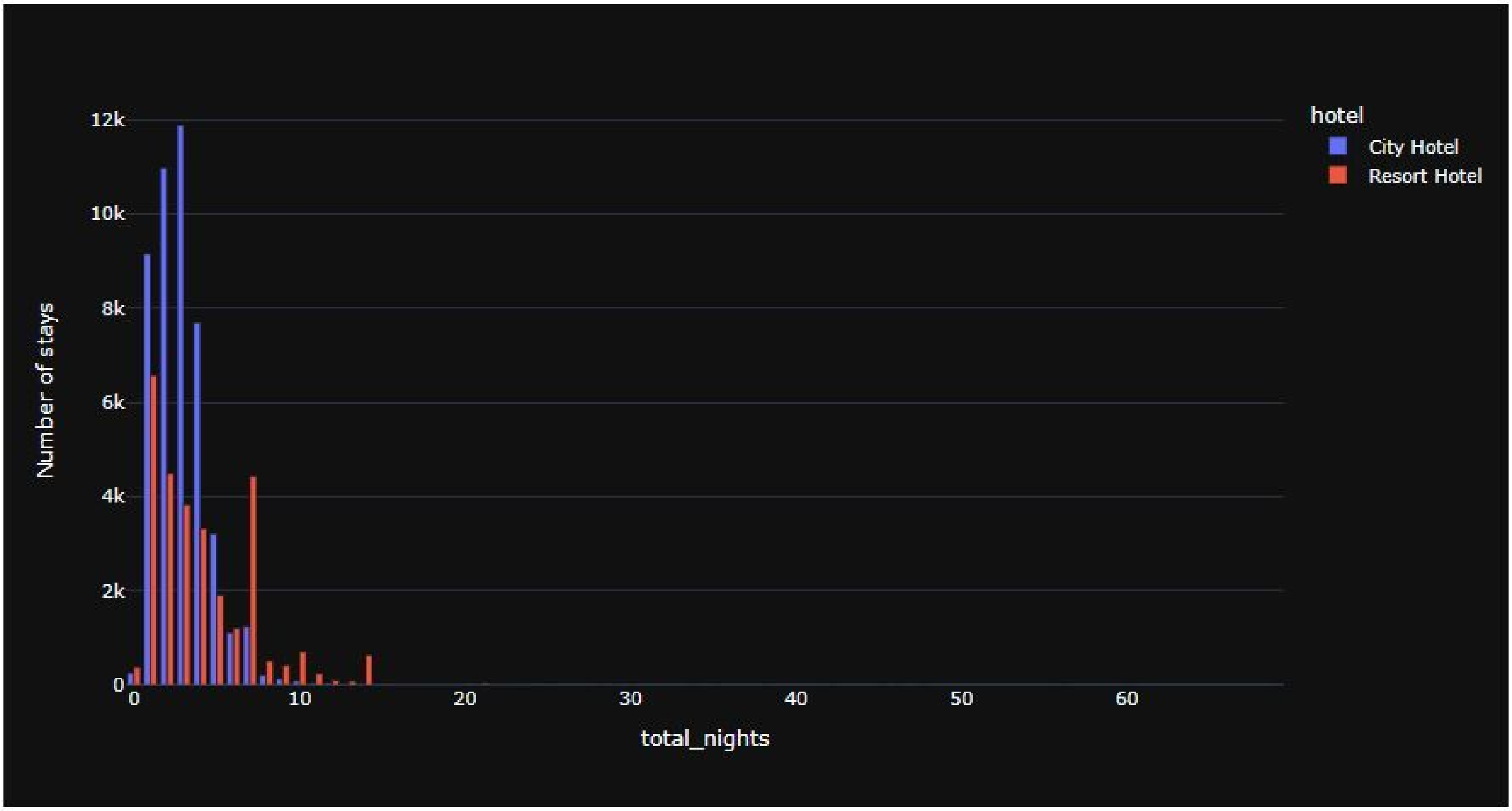
In [50]:
```
stay = data.groupby(['total_nights', 'hotel']).agg('count').reset_index()
stay = stay.iloc[:, :3]
stay = stay.rename(columns={'is_canceled':'Number of stays'})
stay
```

Out[50]:

	total_nights	hotel	Number of stays
0	0	City Hotel	251
1	0	Resort Hotel	371
2	1	City Hotel	9155
3	1	Resort Hotel	6579
4	2	City Hotel	10983
...	...	...	...
57	46	Resort Hotel	1
58	48	City Hotel	1
59	56	Resort Hotel	1
60	60	Resort Hotel	1
61	69	Resort Hotel	1

62 rows × 3 columns

In [51]:
```
px.bar(data_frame = stay, x = 'total_nights', y = 'Number of stays', color = 'hotel', barmode = 'group',
        template = 'plotly_dark')
```



Preprocessing

In [9]:
```
#Dropping the rows that contain missing values except company and agent (except: company and agent)
df = df[df['children'].notna()]
df = df[df['country'].notna()]
```

In [10]:
```
df.shape
```

Out[10]: (118898, 32)

arrival year, month and day to arrival_date

Then we merge the three columns 'arrival_date_year', 'arrival_date_month', 'arrival date day_of_month' into a column called "arrival_date", containing the day, month and year of the client's arrival in datetime form. To do this, we run the following code:

In [11]:
```
#Transforming arrival_date_month to datetime type
df["arrival_date_month"]=pd.to_datetime(df['arrival_date_month'],format='%B').dt.month
```

In [12]:
```
#combine year and month and day in a datetime variable
df["arrival_date"]=pd.to_datetime({"year":df["arrival_date_year"].values,"month":df["arrival_date_month"]
```

In [13]:
```
#Droping the year and month and day columns
df=df.drop(columns=['arrival_date_year','arrival_date_month','arrival_date_day_of_month'])
```

In [14]:
```
#Visualizing the shape of our dataframe (rows,columns) again
df.shape
```

Out[14]: (118898, 30)

In [15]:
```
#Visualizing a sample of 10 rows of our dataframe
df.sample(10)
```

Out[15]:

	hotel	is_canceled	lead_time	arrival_date_week_number	stays_in_weekend_nights	stays_in_week_nights	adults	childre
30372	Resort Hotel	0	2	47	0	1	2	0.
61649	City Hotel	1	101	52	2	5	2	1.
55171	City Hotel	1	189	32	2	5	2	0.
59734	City Hotel	1	166	45	0	3	1	0.
49858	City Hotel	1	265	17	0	4	2	0.
4980	Resort Hotel	1	212	16	2	5	2	0.
80630	City Hotel	1	54	53	1	3	2	0.
41345	City Hotel	0	133	33	2	2	2	0.
55	Resort Hotel	0	1	27	0	1	2	2.
32435	Resort Hotel	0	1	8	0	2	1	0.

10 rows × 30 columns

Verifiying that the timestamp of the variable reservation_status_date must occur after or at the same date as the input variable arrival_date

In [16]:
```
#Visualizing the types of our dataframe's variables
df.dtypes
```

Out[16]:
```
hotel                                     object
is_canceled                                int64
lead_time                                  int64
arrival_date_week_number                   int64
stays_in_weekend_nights                    int64
stays_in_week_nights                       int64
adults                                     int64
children                                 float64
babies                                     int64
meal                                      object
country                                   object
market_segment                            object
distribution_channel                      object
is_repeated_guest                          int64
previous_cancellations                     int64
previous_bookings_not_canceled             int64
reserved_room_type                        object
assigned_room_type                        object
booking_changes                            int64
deposit_type                              object
agent                                    float64
company                                  float64
days_in_waiting_list                       int64
customer_type                             object
adr                                      float64
required_car_parking_spaces                int64
total_of_special_requests                  int64
reservation_status                        object
reservation_status_date                   object
arrival_date                      datetime64[ns]
dtype: object
```

In [17]:
```
#Transforming the reservation_status_date variable type to Datetime
df["reservation_status_date"]=pd.to_datetime(df["reservation_status_date"], format = '%Y-%m-%d')
```

In [18]:
```
#Visualizing the types of our dataframe's variables again
df.dtypes
```

Out[18]:
```
hotel                                   object
is_canceled                              int64
lead_time                                int64
arrival_date_week_number                 int64
stays_in_weekend_nights                  int64
stays_in_week_nights                     int64
adults                                   int64
children                               float64
babies                                   int64
meal                                    object
country                                 object
market_segment                          object
distribution_channel                    object
is_repeated_guest                        int64
previous_cancellations                   int64
previous_bookings_not_canceled           int64
reserved_room_type                      object
assigned_room_type                      object
booking_changes                          int64
deposit_type                            object
agent                                  float64
company                                float64
days_in_waiting_list                     int64
customer_type                           object
adr                                    float64
required_car_parking_spaces              int64
total_of_special_requests                int64
reservation_status                      object
reservation_status_date         datetime64[ns]
arrival_date                    datetime64[ns]
dtype: object
```

Data Cleaning:

We propose to treat the missing values, to use the approach of filling each empty box with the median of the values of the column to which this empty box belongs, and we can extend this solution, by adding another column which will contain two values, True or False, to indicate if the value of the first column is original or it is calculated by the median. We implement this solution for the two columns “agent” and “company” as it is illustrated in the following figure:

In [19]:
```
#Filling null values in these two columns with the mean of values of each column
for column in ['agent','company']:
    df[column] =df[column].fillna(df[column].mean())
```

In [20]:
```
#Vizualizing the sum of missing values in each variable
df.isnull().sum()
```

Out[20]:
```
hotel                               0
is_canceled                         0
lead_time                           0
arrival_date_week_number            0
stays_in_weekend_nights             0
stays_in_week_nights                0
adults                              0
children                            0
babies                              0
meal                                0
country                             0
market_segment                      0
distribution_channel                0
is_repeated_guest                   0
previous_cancellations              0
previous_bookings_not_canceled      0
reserved_room_type                  0
assigned_room_type                  0
booking_changes                     0
deposit_type                        0
agent                               0
company                             0
days_in_waiting_list                0
customer_type                       0
adr                                 0
required_car_parking_spaces         0
total_of_special_requests           0
reservation_status                  0
reservation_status_date             0
arrival_date                      492
dtype: int64
```

In [21]:
```
#Filling null values in these two columns with the mean of values of each column
for column in ['arrival_date']:
    df[column] =df[column].fillna(df[column].mean())
```

In [22]:
```
#Vizualizing the sum of missing values in each variable
df.isnull().sum()
```

Out[22]:
```
hotel                             0
is_canceled                       0
lead_time                         0
arrival_date_week_number          0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          0
babies                            0
meal                              0
country                           0
market_segment                    0
distribution_channel              0
is_repeated_guest                 0
previous_cancellations            0
previous_bookings_not_canceled    0
reserved_room_type                0
assigned_room_type                0
booking_changes                   0
deposit_type                      0
agent                             0
company                           0
days_in_waiting_list              0
customer_type                     0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
reservation_status                0
reservation_status_date           0
arrival_date                      0
dtype: int64
```

We check if there are duplicate lines, if so we opt to delete them, using the following command:

In [23]:
```
#Droping the duplicated values
df.drop_duplicates( inplace = True)
```

Transformation: in order to properly treat categorical variables, we propose the creation of columns among the number of categories for each variable. Each column is filled with the values 0 and 1. The value 0 replaces NULL, and 1 means that the corresponding row has this category. In our case; we first specify the categorical variables; and we transform them as follows:

In [24]:
```
#Transformation of categoriccal variables to numirical variables
categoricalV=["hotel","meal","country","market_segment","distribution_channel","reserved_room_type","assi
df[categoricalV[1:11]]=df[categoricalV[1:11]].astype('category')
```

In [25]:
```
df[categoricalV[1:11]]=df[categoricalV[1:11]].apply(lambda x:LabelEncoder().fit_transform(x))
```

In [26]:
```
df['hotel_Num']=LabelEncoder().fit_transform(df['hotel'])
```

In [27]:
```
df.dtypes
```

Out[27]:
```
hotel                                     object
is_canceled                                int64
lead_time                                  int64
arrival_date_week_number                   int64
stays_in_weekend_nights                    int64
stays_in_week_nights                       int64
adults                                     int64
children                                 float64
babies                                     int64
meal                                       int32
country                                    int32
market_segment                             int32
distribution_channel                       int32
is_repeated_guest                          int64
previous_cancellations                     int64
previous_bookings_not_canceled             int64
reserved_room_type                         int32
assigned_room_type                         int32
booking_changes                            int64
deposit_type                               int32
agent                                    float64
company                                  float64
days_in_waiting_list                       int64
customer_type                              int32
adr                                      float64
required_car_parking_spaces                int64
total_of_special_requests                  int64
reservation_status                        object
reservation_status_date           datetime64[ns]
arrival_date                      datetime64[ns]
hotel_Num                                  int32
dtype: object
```

Modeling

From a demographic perspective, if we have precise data we will predict whether it is a resort hotel or a city hall. Supervised learning techniques will allow us to accomplish such a task, including Logistic Regression, KNN, SVM. In other words, the problem is purely a classification problem, which emphasizes segmentation of individuals based on the target variable hotel. This will help the hotel to divide the guests into groups based on the type of host. Which means a significant increase in profits and relevant revenue management.

Logistic Regrssion

In [32]:
```
#Importing the train_test_split module for spliting data
from sklearn.model_selection import train_test_split
#Importing datetime
import datetime as dt
```

In [33]:
```
#transforming Datetime variables to numerical variables
df['numerical_larrival_date']=df['arrival_date'].map(dt.datetime.toordinal)
df['numerical_reservation_status_date']=df['reservation_status_date'].map(dt.datetime.toordinal)
```

In [34]:
```
#transforming is_canceled to a numerical variable
df["is_canceled"].replace({'not canceled': 0,'canceled':1}, inplace=True)
df["reservation_status"].replace({'Canceled': 0,'Check-Out':1,'No-Show':2}, inplace=True)
```

In [35]:
```
#Defining X (target values) and Y (usefull columns)
usefull_columns = df.columns.difference(['hotel','hotel_Num','arrival_date','reservation_status_date'])
X = df[usefull_columns]
Y = df["hotel_Num"].astype(int)
```

In [36]:
```
#Spliting data to train data and test data
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3,random_state=150)
```

Test_size = 0.3 means that 30% of the initial data is dedicated to model testing, and the 70% is dedicated to model training.
Random_state means the degree of randomness with which we will divide our dataset

In [37]:
```
#Importing some needed metrics for evaluating the models
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import plot_confusion_matrix
```

In [38]:
```
#Training our Logistic Regressing model
logisticR = LogisticRegression()
logisticR.fit(X_train,Y_train)
Y_pred= logisticR.predict(X_test)
Y_train_pred = logisticR.predict(X_train)
acc_logrest= accuracy_score(Y_test,Y_pred)
```

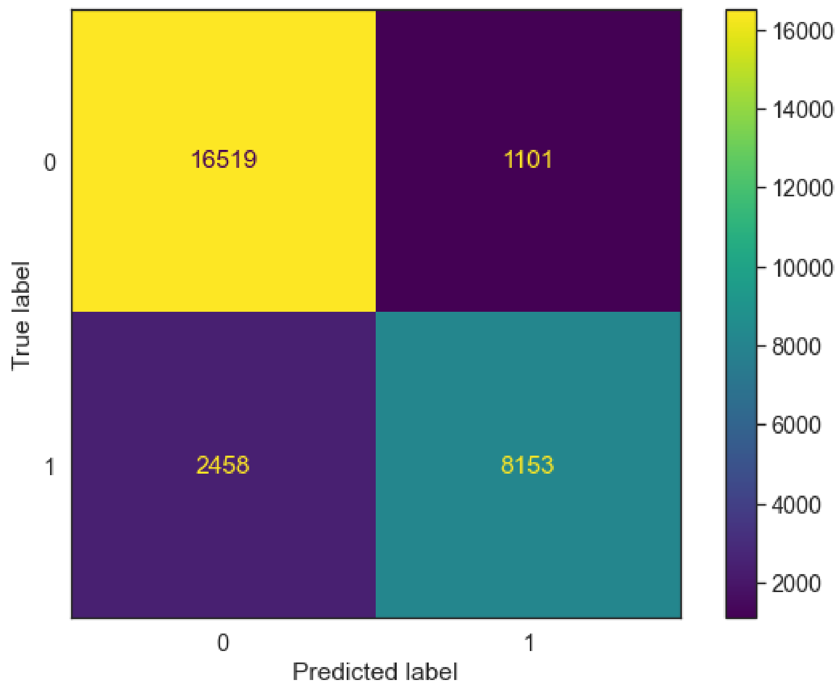
In [39]:
```
#metrics and accuracy score
print('Recall Score :',recall_score(Y_test,Y_pred))
print('Precision Score :',precision_score(Y_test,Y_pred))
print('F1 Score :',f1_score(Y_test,Y_pred))
print('-----------------------------------------------')
print('Accuracy Score :',accuracy_score(Y_test,Y_pred))
```

```
Recall Score : 0.7683535953256055
Precision Score : 0.8810244218716231
F1 Score : 0.8208406745532345
-----------------------------------------------
Accuracy Score : 0.8739329106301583
```

And we can see that our model's accuracy is 87%, which represents a good performance.

In [40]:
```
#Ploting the confusion matrix
plot_confusion_matrix(logisticR,X_test,Y_test)
```

Out[40]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x17700589c70>



SVM - Support Vector Machine

In [41]:
```
#Importing the needed packages for SVM algorithme
from sklearn import svm
from sklearn.svm import SVC
```

In [42]:
```
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler(feature_range=(-1,1)).fit(X_train)
X_train = scaling.transform(X_train)
X_test = scaling.transform(X_test)
```

In [43]:
```
#Defining an SVM classifier
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, Y_train)
```

Out[43]: SVC(kernel='linear')

In [44]:
```
#Training the model
Y_pred = svclassifier.predict(X_test)
acc_svm= accuracy_score(Y_test,Y_pred)
```

In [45]:
```
#metrics and accuracy scores
print('Recall Score :',recall_score(Y_test,Y_pred))
print('Precision Score :',precision_score(Y_test,Y_pred))
print('F1 Score :',f1_score(Y_test,Y_pred))
print('-----------------------------------------------')
print('Accuracy Score :',accuracy_score(Y_test,Y_pred))
```

```
Recall Score : 0.8567524267269815
Precision Score : 0.8922367258808519
F1 Score : 0.8741346153846153
-----------------------------------------------
Accuracy Score : 0.907265063228366
```

We have as a result a classification rate of 90%, considered as a very good precision.

KNN - K-Nearest Neighbors

In [46]:
```
#Importing the needed packages for KNN algorithme
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error
from math import sqrt
```

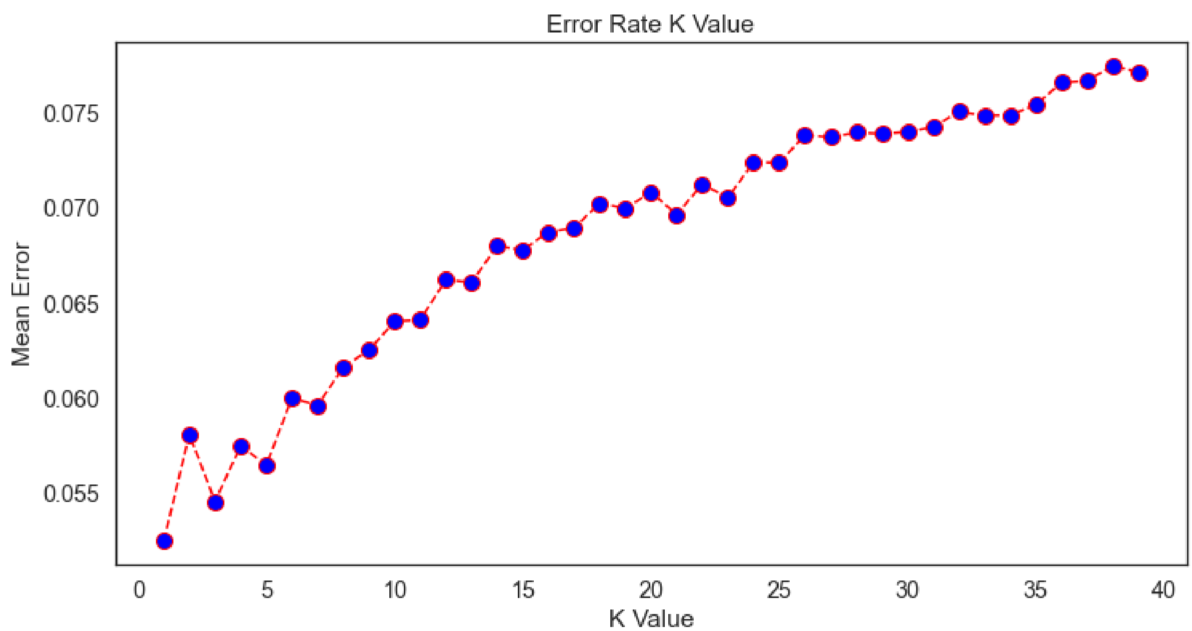
In this section, we will plot the mean error for the predicted values of the test set for all K values between 1 and 40. first the error mean for all predicted values where K is between 1 and 40, In each iteration, the average error for the predicted values of the set of test is calculated and the result is added to the error list:

In [47]:
```
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, Y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != Y_test))
```

In [48]:
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',markerfacecolor='blue', markers
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Out[48]: `Text(0, 0.5, 'Mean Error')`



Note that the use of the value 1 for K is the most optimal. In order to train the KNN algorithm, we rely on the use of Scikit-Learn. The first step is to import the KNeighborsClassifier class from from the sklearn.neighbors library. This class is initiated with a parameter, (n_neigbours). This is basically the value of K. The last step is to make predictions about our test data. To do this, we run the following script:

In [49]:
```
#Defining an KNN classifier and training the model
classifier = KNeighborsClassifier(n_neighbors=1)
classifier.fit(X_train, Y_train)
Y_pred = classifier.predict(X_test)
acc_knn=accuracy_score(Y_test,Y_pred)
```

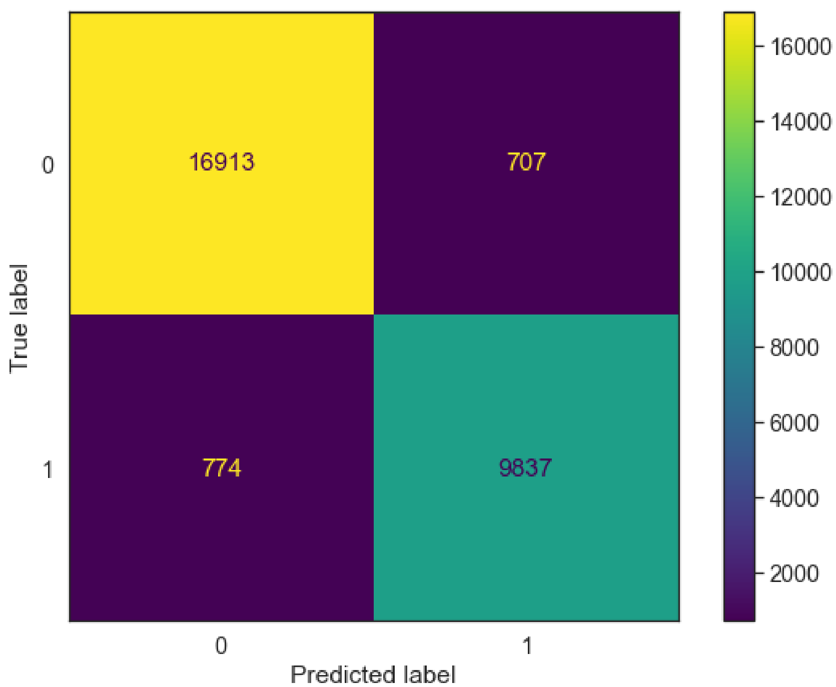
In [50]:
```
#Showing the metrics and accuracy scores
print('Recall Score :',recall_score(Y_test,Y_pred))
print('Precision Score :',precision_score(Y_test,Y_pred))
print('F1 Score :',f1_score(Y_test,Y_pred))
print('-----------------------------------------------')
print('Accuracy Score :',accuracy_score(Y_test,Y_pred))
```

```
Recall Score : 0.9270568278201866
Precision Score : 0.9329476479514416
F1 Score : 0.9299929094776648
-----------------------------------------------
Accuracy Score : 0.9475399383656264
```

The results show that our KNN algorithm was able to rank the test set records with an accuracy of 94%, which is excellent given the high dimensionality of our dataset.

In [51]:
```
#Ploting the confusion matrix
plot_confusion_matrix(classifier,X_test,Y_test)
```

Out[51]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1770070c910>



Models Comparison

Following is the chunk of code by which we drawn models comparison.

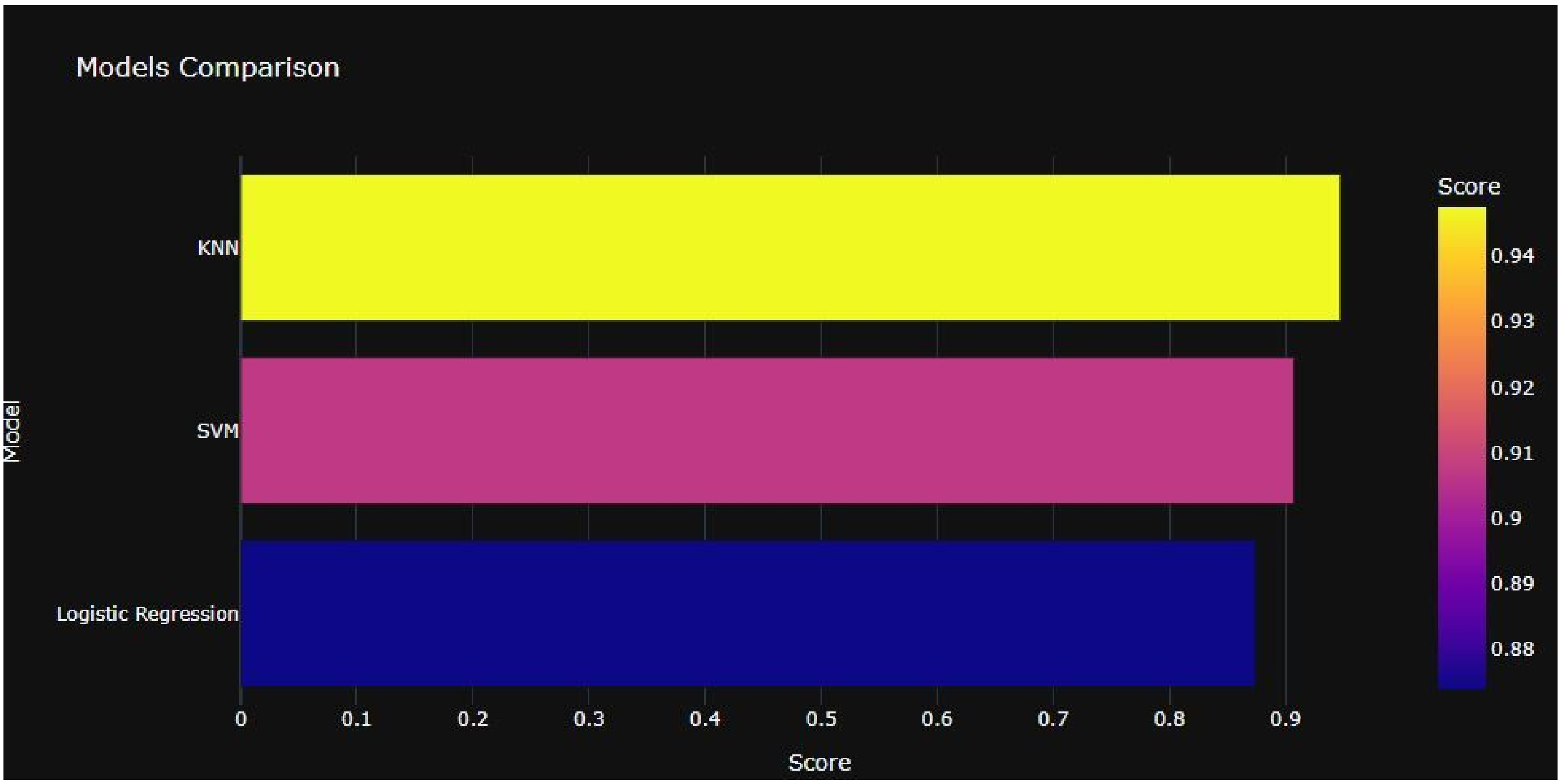
In [52]:
```
models = pd.DataFrame({
    'Model' : ['Logistic Regression', 'SVM', 'KNN'],
    'Score' : [acc_logrest, acc_svm, acc_knn]
})


models.sort_values(by = 'Score', ascending = False)
```

Out[52]:

	Model	Score
2	KNN	0.947540
1	SVM	0.907265
0	Logistic Regression	0.873933

In [53]:
```
import plotly.express as px
px.bar(data_frame = models, x = 'Score', y = 'Model', color = 'Score', template = 'plotly_dark', title =
```



Concerning the accuracy, we had as result 94% for KNN, 90% for SVM, and 86% for Logistic Regression. The greatest value is that of KNN ... In other words with Knn 94% of our predictions will be correct, it therefore represents the best model to adopt. Which is logical, in the literature we find that when the training data is much bigger than the other features, KNN is better than SVM. Besides KNN is easy to implement. Yet KNN is slower in execution time than LR, but not slow enough than SVM. From a more global perspective, KNN and SVM support nonlinear solutions, and they are unparameterized where Parameterized Logistic Regression, deals with linear solutions. SVM is less computationally demanding than kNN and it is easier to interpret but can only identify a limited set of patterns. On the other hand, kNN can find very complex models but its output is more difficult to interpret.

Clustering with K-Means

After we used supervised algorithms in the first part, now we have considered an unsupervised problem, a clustering problem based on K-Means, and we will analyze the results of each cluster to identify the most profitable clients in our data set based on lead time and ADR. The first challenge that we encounter when we want to use clustering with K-means, is to determine the optimal number of clusters that we want to have as results. So first to determine the number of clusters, we used the Elbow method:

In [54]:
```
import sklearn.cluster as cluster
```

In [55]:
```
df_Short = df[['lead_time','adr']]
```

In [56]:
```
K=range(1,12)
wss = []
for k in K:
    kmeans=cluster.KMeans(n_clusters=k,init="k-means++")
    kmeans=kmeans.fit(df_Short)
    wss_iter = kmeans.inertia_
    wss.append(wss_iter)
```

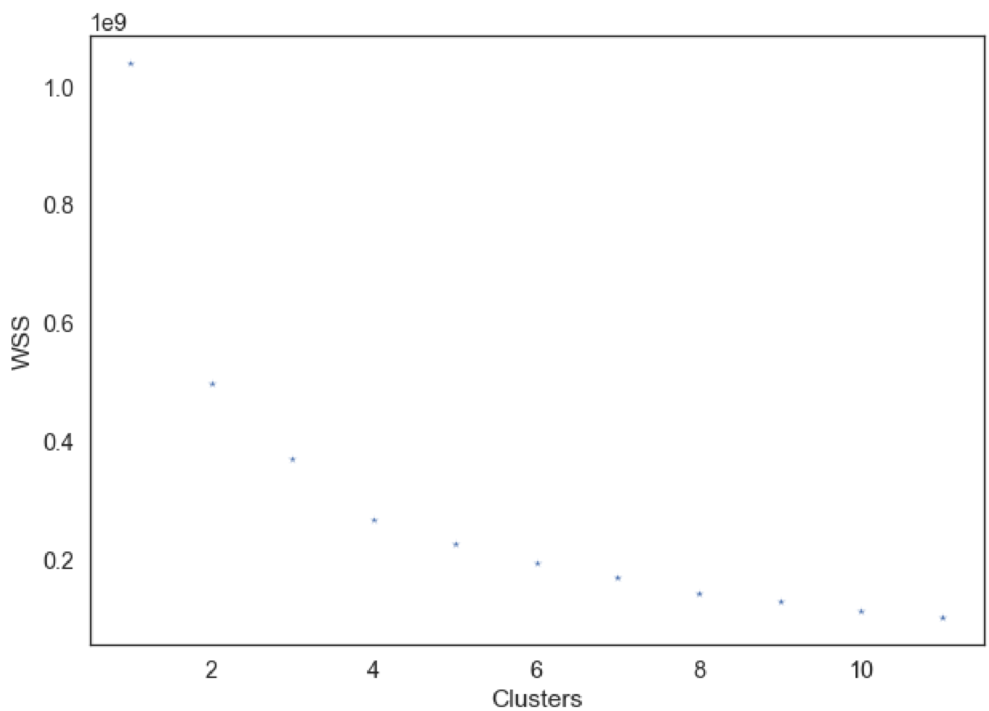
In [57]:
```
mycenters = pd.DataFrame({'Clusters' : K, 'WSS' : wss})
mycenters
```

Out[57]:

	Clusters	WSS
0	1	1.039543e+09
1	2	4.978291e+08
2	3	3.707516e+08
3	4	2.674674e+08
4	5	2.267966e+08
5	6	1.939704e+08
6	7	1.705141e+08
7	8	1.444610e+08
8	9	1.284159e+08
9	10	1.144940e+08
10	11	1.025504e+08

In [58]:
```
sns.scatterplot(x = 'Clusters', y = 'WSS', data = mycenters, marker='*')
```

Out[58]: <AxesSubplot:xlabel='Clusters', ylabel='WSS'>



To determine the optimal number of clusters, one must select the value of k after which the distortion begins to decrease linearly. Thus, we conclude that the optimal number of clusters for the data is 4. So we ran the k-means algorithm based on lead_time and ADR with a number of clusters equal to 4, and we displayed the cluster centers:

In [59]:
```
kmeans = cluster.KMeans(n_clusters=4 ,init="k-means++")
```

In [60]:
```
kmeans = kmeans.fit(df[['lead_time','adr']])
```

In [61]:
```
kmeans.cluster_centers_
```

Out[61]:
```
array([[144.98898383, 107.46222962],
       [ 26.40389407,  74.85986442],
       [ 37.71738588, 170.31215415],
       [286.76676038,  92.19575368]])
```

In [62]:
```
df['Clusters'] = kmeans.labels_
```

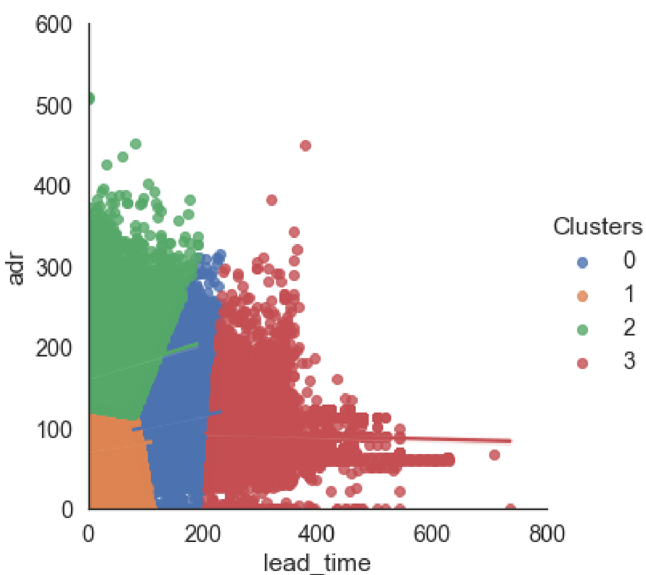
Then we displayed the number of observations belonging to each cluster:

In [63]:
```
df['Clusters'].value_counts()
```

Out[63]:
```
1    41032
0    23871
2    19973
3     9225
Name: Clusters, dtype: int64
```

Finally we have displayed the clusters:

In [64]:
```
sns.lmplot(x="lead_time", y="adr",hue = 'Clusters',  data=df)
plt.ylim(0, 600)
plt.xlim(0, 800)
plt.show()
```



The clients with the lowest lead time and the highest ADR ie the clients that appear in the green cluster are considered to be the most profitable. While the red category shows the lowest ADR and the highest (least profitable) delivery time. With regard to unsupervised learning in general - it is important to remember that this is largely a method of exploratory analysis - the goal is not necessarily to predict but rather to reveal information about data that may not have been taken into account before.

Conclusion

In short, we learned three different ways to classify data using python for the supervised type (KNN, SVM, RL), and one for the unsupervised. We have found that, for the first type, KNN remains the best in terms of performance, for our case. And for the unsupervised, K-means allowed us to visualize the most profitable customers and the least profitable customers, based on the two variables lead_time (Number of days elapsed between the date of entry of the reservation in the PMS and the client arrival date) and adr (Average daily rate as defined by dividing the sum of all accommodation transactions by the total number of nights), and we used the result of this algorithm (which is under graph form) to ask specific questions about the variation in profitability of our customers, in order to give the hotel manager ideas to make their customers more profitable. Overall, the ideal model chosen for machine learning very often depends on the problem. There will be some datasets where KNN could fail miserably, so it is good to implement all the other models, for each problem, in order to judge the performance of each and choose the best model to adopt. It all comes down to specifying the variables to be processed, and choosing the right machine learning model.