Hotel Booking Prediction using Machine Learning in Python

- **Data is provided in csv format**
- **First we read the given data using pandas and have a sneakpeek of its attributes**
- **Further we will start the process of cleaning the data**

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
data=pd.read_csv(r'E:\Project\Project 1\Data\hotel_bookings.csv')
data.head()
```

Out[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	...	deposit_type	agent	com
0	Resort Hotel	0	342	2015	July	27	1	0	0	2	...	No Deposit	NaN	
1	Resort Hotel	0	737	2015	July	27	1	0	0	2	...	No Deposit	NaN	
2	Resort Hotel	0	7	2015	July	27	1	0	1	1	...	No Deposit	NaN	
3	Resort Hotel	0	13	2015	July	27	1	0	1	1	...	No Deposit	304.0	
4	Resort Hotel	0	14	2015	July	27	1	0	2	2	...	No Deposit	240.0	

5 rows × 32 columns

Lets us now find out no. of rows and columns of our data

In [4]:
```
data.shape
```

Out[4]: (119390, 32)

Here we can clearly see that data table has more than 119000 rows and 32 columns, so we have a huge amount of data to analyse

Let us now use '.dtypes' to find out the data type of each column of our data set

In [5]:
```
data.dtypes
```

Out[5]:
```
hotel                           object
is_canceled                      int64
lead_time                        int64
arrival_date_year                int64
arrival_date_month              object
arrival_date_week_number         int64
arrival_date_day_of_month        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
dtype: object
```

Let us now clean the data and find the non available values in the Data Frame

- step 1: finding out the columns with non available data

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
hotel                               0
is_canceled                         0
lead_time                           0
arrival_date_year                   0
arrival_date_month                  0
arrival_date_week_number            0
arrival_date_day_of_month           0
stays_in_weekend_nights             0
stays_in_week_nights                0
adults                              0
children                            4
babies                              0
meal                                0
country                           488
market_segment                      0
distribution_channel                0
is_repeated_guest                   0
previous_cancellations              0
previous_bookings_not_canceled      0
reserved_room_type                  0
assigned_room_type                  0
booking_changes                     0
deposit_type                        0
agent                           16340
company                        112593
days_in_waiting_list                0
customer_type                       0
adr                                 0
required_car_parking_spaces         0
total_of_special_requests           0
reservation_status                  0
reservation_status_date             0
dtype: int64
```

For cleaning the data let us define a function which will fill the 'na' values with 0, this is done so that during ML process the algorithms can better handle 0's rather that 'NULL

In [7]:
```
def data_clean(df):
    df.fillna(0,inplace=True)
    print(df.isnull().sum())
```

In [8]:
```
data_clean(data)
```

```
hotel                           0
is_canceled                     0
lead_time                       0
arrival_date_year               0
arrival_date_month              0
arrival_date_week_number        0
arrival_date_day_of_month       0
stays_in_weekend_nights         0
stays_in_week_nights            0
adults                          0
children                        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
dtype: int64
```

For second part of cleaning let us see important attributes of the DF where customer occupancy/checkin as per adult,children and babies is given

- let us first check the values entered by the receptionist

In [9]:
```
lst=['adults','children','babies']
for i in lst:
    print(f'unique values for {i} in DF: {data[i].sort_values().unique()}')
```

```
unique values for adults in DF: [ 0  1  2  3  4  5  6 10 20 26 27 40 50 55]
unique values for children in DF: [ 0.  1.  2.  3. 10.]
unique values for babies in DF: [ 0  1  2  9 10]
```

- ***Note that here we can clearly see that some bookings has had 0 adults,0 children and/or 0 babies:****

problem with this scenario is that there may be some rows where adults attribute have value = 0 which is highly improbable and also there may be some rows where all three attributes have value=0 which is also improbable because how come receptionist was able to insert values in rest of the attributes/columns when there was no arrival of the guest as suggest by 0s in those attributes

In [10]:
```
filter=(data['adults']==0) & (data['children']==0) & (data['babies']==0) |\
(data['adults']==0) & (data['children']!=0) & (data['babies']!=0)
```

- ***Now let us remove this data using the filter we created above****

In [11]:
```
data=data[~ filter]
data.reset_index(drop=True,inplace=True)
```

In [12]:
```
data
```

Out[12]:

	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	...	deposit_type	agent
0	Resort Hotel	0	342	2015	July	27	1	0	0	2	...	No Deposit	0.0
1	Resort Hotel	0	737	2015	July	27	1	0	0	2	...	No Deposit	0.0
2	Resort Hotel	0	7	2015	July	27	1	0	1	1	...	No Deposit	0.0
3	Resort Hotel	0	13	2015	July	27	1	0	1	1	...	No Deposit	304.0
4	Resort Hotel	0	14	2015	July	27	1	0	2	2	...	No Deposit	240.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
119202	City Hotel	0	23	2017	August	35	30	2	5	2	...	No Deposit	394.0
119203	City Hotel	0	102	2017	August	35	31	2	5	3	...	No Deposit	9.0
119204	City Hotel	0	34	2017	August	35	31	2	5	2	...	No Deposit	9.0
119205	City Hotel	0	109	2017	August	35	31	2	5	2	...	No Deposit	89.0
119206	City Hotel	0	205	2017	August	35	29	2	7	2	...	No Deposit	9.0

119207 rows × 32 columns

Task 1: Where do the guests come from

In [13]:
```
# guest_count_country_wise=data.loc[(data['is_canceled']==0) & (data['country']!=0),['hotel','country']].\
# set_index('hotel').groupby(['hotel','country']).value_counts()
guest_count_country_wise=data[(data['country']!=0) & (data['is_canceled']==0)]['country'].value_counts().reset_index()
guest_count_country_wise.columns=['Country','No of guests']
guest_count_country_wise
```

Out[13]:

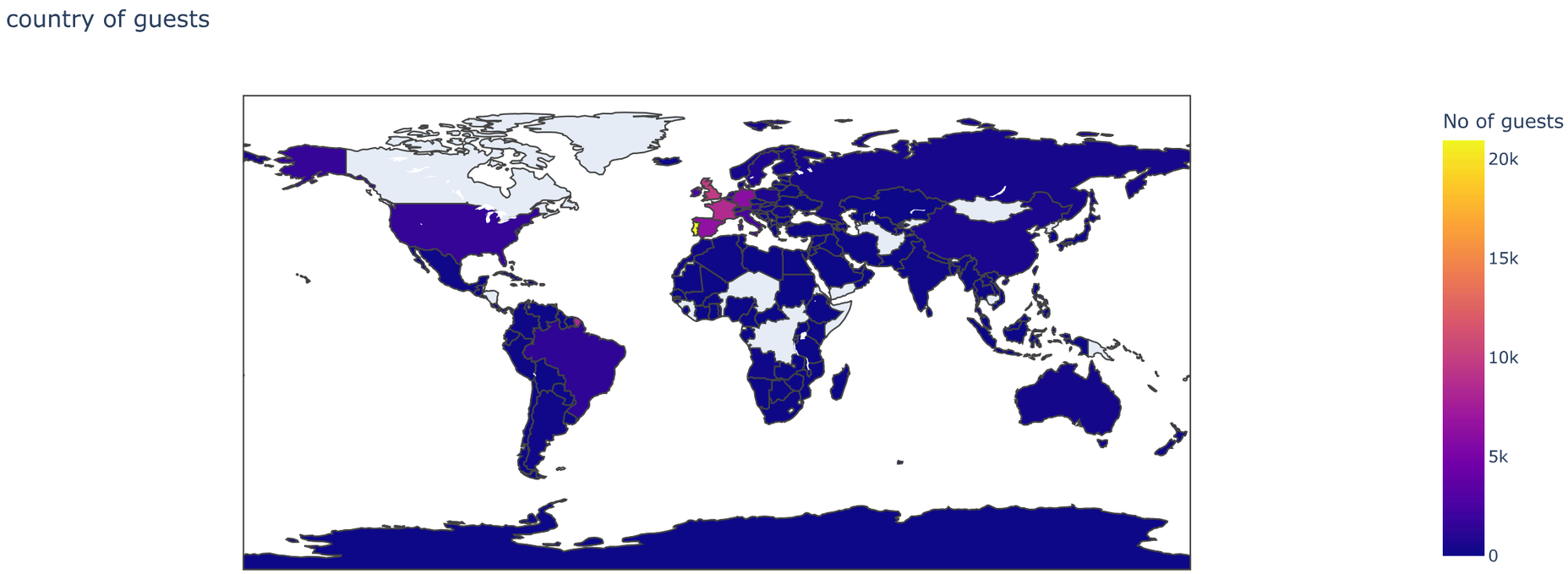
	Country	No of guests
0	PRT	20976
1	GBR	9668
2	FRA	8468
3	ESP	6383
4	DEU	6067
...	...	...
160	BHR	1
161	DJI	1
162	MLI	1
163	NPL	1
164	FRO	1

165 rows × 2 columns

let's create map visualization of the above data

In [14]:
```
import plotly.express as px
```

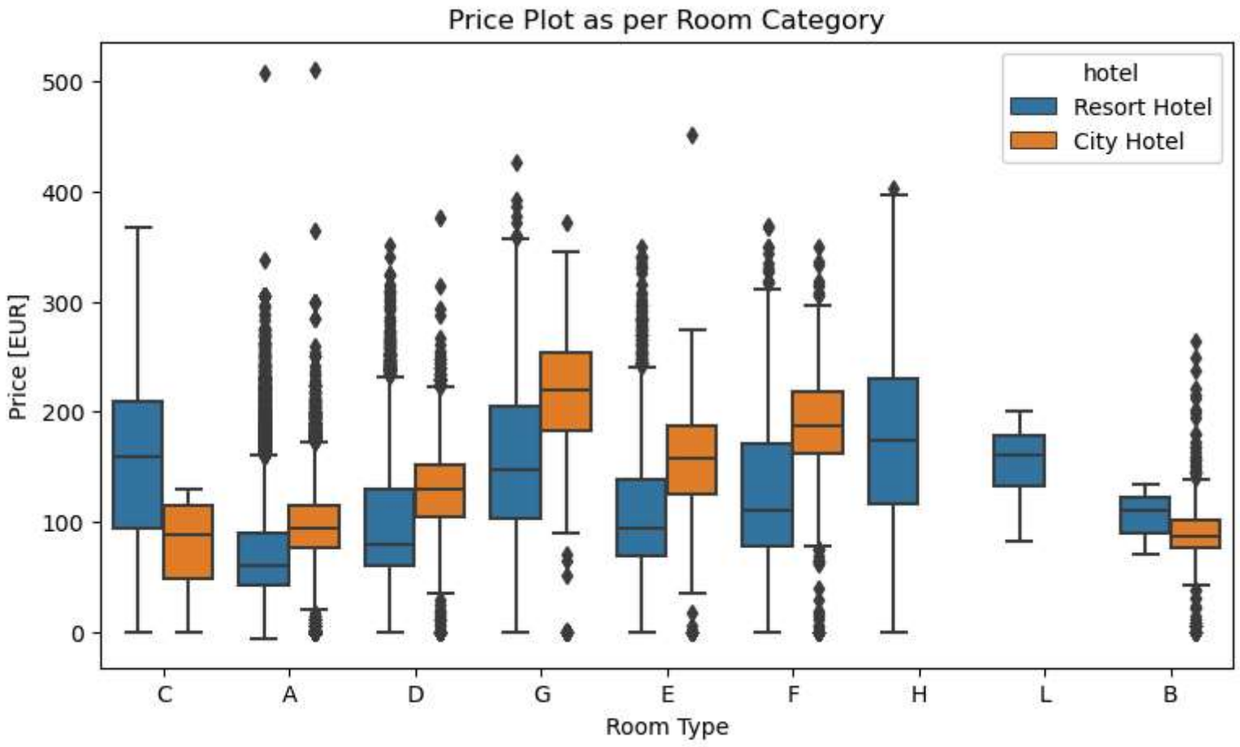
In [15]:
```
plt.figure(figsize=(5,7))
map_guests_country_wise=px.choropleth(guest_count_country_wise,
                          locations=guest_count_country_wise['Country'],
                          color=guest_count_country_wise['No of guests'],
                          hover_name=guest_count_country_wise['Country'],
                          title='country of guests')
map_guests_country_wise.show()
```




```
<Figure size 500x700 with 0 Axes>
```

Let us now see price differences using boxplot as per room category

In [16]:
```
data_bxplt=data[(data['is_canceled']==0) & (data['country']!=0)][['hotel','adr','reserved_room_type']]
data_bxplt
plt.figure(figsize=(9,5))
sns.boxplot(x='reserved_room_type',
            y='adr',
            hue='hotel',
           data=data_bxplt)
plt.xlabel('Room Type')
plt.ylabel('Price [EUR]')
plt.title('Price Plot as per Room Category');
```



In [17]:
```
data.columns
```

Out[17]:
```
Index(['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', 'children', 'babies', 'meal',
       'country', 'market_segment', 'distribution_channel',
       'is_repeated_guest', 'previous_cancellations',
       'previous_bookings_not_canceled', 'reserved_room_type',
       'assigned_room_type', 'booking_changes', 'deposit_type', 'agent',
       'company', 'days_in_waiting_list', 'customer_type', 'adr',
       'required_car_parking_spaces', 'total_of_special_requests',
       'reservation_status', 'reservation_status_date'],
      dtype='object')
```

In [18]:
```
# hotels=list(data['hotel'].unique())
# data_lgraph={}
# count=0
# for i in hotels:
#     data_lgraph[count]=data[(data['is_canceled']==0) & (data['country']!=0) & (data['hotel']==i)][['hotel','arrival_date_month','adr']].\
#     groupby(['hotel','arrival_date_month']).mean().reset_index().sort_values(by='arrival_date_month')
#     count+=1
# data_lgraph[0]
```

let's compare the pricing of resort and city hotels, we will do this by:

- modifying the dataframe to get mean price distribution month wise for both category and sorting if required
- plotting a graph showing trend lines according to months

In [19]:
```
data_lgraph_resort=data[(data['country']!=0) & (data['is_canceled']==0) & (data['hotel']=='Resort Hotel')].groupby('arrival_date_month')['adr'].mean().reset_index()
data_lgraph_city=data[(data['country']!=0) & (data['is_canceled']==0) & (data['hotel']=='City Hotel')].groupby('arrival_date_month')['adr'].mean().reset_index()
data_lgraph_merged=data_lgraph_resort.merge(data_lgraph_city,
                                           on='arrival_date_month')
data_lgraph_merged.rename(columns={'arrival_date_month':'month','adr_x':'price_resort','adr_y':'price_city'},inplace=True)
data_lgraph_merged
```

Out[19]:

	month	price_resort	price_city
0	April	76.176140	111.962267
1	August	181.348715	118.674598
2	December	68.634584	88.406654
3	February	54.468180	86.520062
4	January	48.994018	82.330983
5	July	150.483553	115.818019
6	June	108.133154	117.874678
7	March	57.408308	90.658533
8	May	76.734804	120.669827
9	November	48.882907	86.946178
10	October	61.932777	102.004672
11	September	96.288897	112.776582

Lets us now sort the data here sorting is not straight forward since month col is string type, therefore, we can def a function sort for such case and can use same function afterwards also

In [20]:
```
from calendar import month_name
def sort_months(df,col_name):
    month_dict={j:i for i,j in enumerate(month_name)}
    df['month_num']=df[col_name].apply(lambda x: month_dict[x])
    return df.sort_values(by='month_num').reset_index().drop(columns=['index','month_num'])
```

In [21]:
```
# test=data_lgraph_merged
```

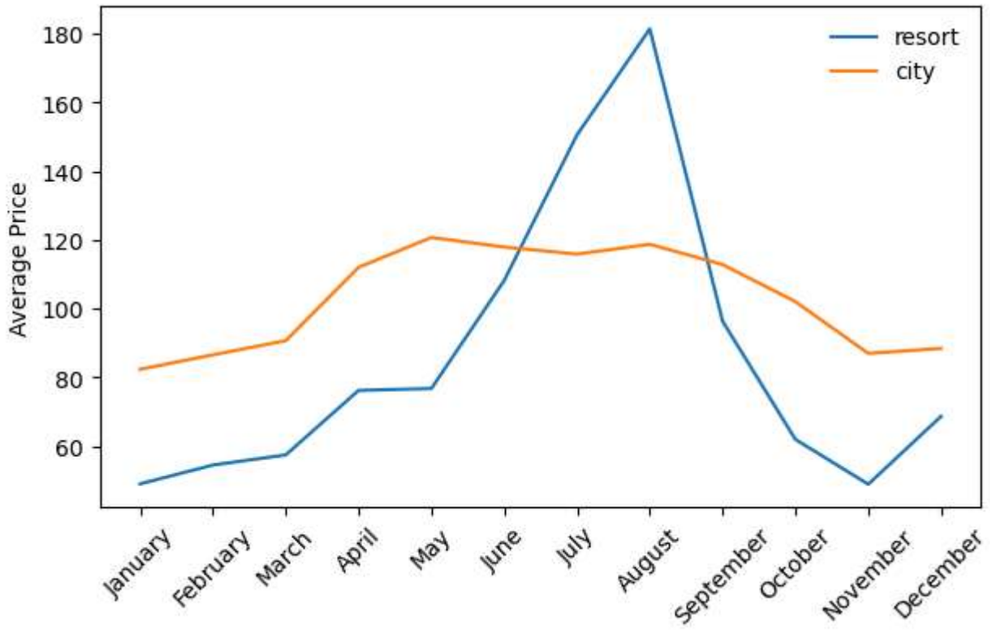
In [22]:
```
data_lgraph_merged_sorted=sort_months(data_lgraph_merged,'month')
data_lgraph_merged_sorted
```

Out[22]:

	month	price_resort	price_city
0	January	48.994018	82.330983
1	February	54.468180	86.520062
2	March	57.408308	90.658533
3	April	76.176140	111.962267
4	May	76.734804	120.669827
5	June	108.133154	117.874678
6	July	150.483553	115.818019
7	August	181.348715	118.674598
8	September	96.288897	112.776582
9	October	61.932777	102.004672
10	November	48.882907	86.946178
11	December	68.634584	88.406654

In [23]:
```
fig,ax=plt.subplots(figsize=(7,4))

ax.plot(data_lgraph_merged_sorted['month'],
        data_lgraph_merged_sorted['price_resort'],
        label='resort')
ax.plot(data_lgraph_merged_sorted['month'],
        data_lgraph_merged_sorted['price_city'],
        label='city')
ax.legend(loc='upper right',frameon=False)
ax.set_ylabel('Average Price')
plt.xticks(rotation=45);
```



Let us now find out the number of total guests visits on monthly basis

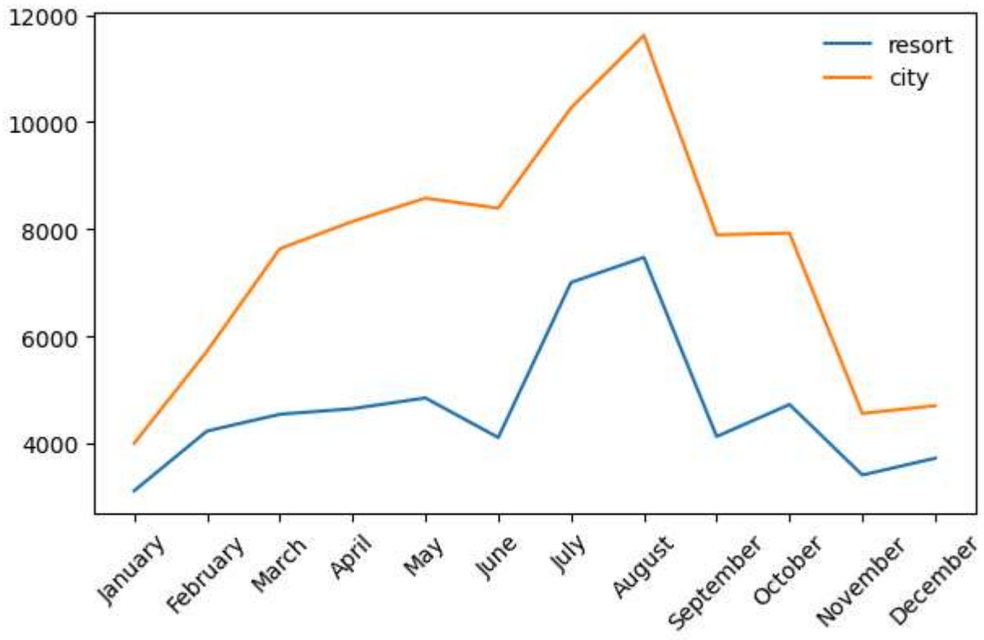
1. modifying DF as per requiremment
2. Sorting by months
3. Plotting the line graph

In [24]:
```
guest_num={}
for i in list(data['hotel'].unique()):
    guest_num[i]=data[(data['country']!=0) & (data['is_canceled']==0) & (data['hotel']==i)][['arrival_date_month','adults', 'children', 'babies']].\
    groupby('arrival_date_month').sum().assign(total=lambda x: x.sum(axis=1)).reset_index()
    guest_num[i].columns.values[0]='month'
```

In [25]:
```
guest_num_sorted_r=sort_months(guest_num['Resort Hotel'],'month')
guest_num_sorted_c=sort_months(guest_num['City Hotel'],'month')
```

In [26]:
```
fig,ax=plt.subplots(figsize=(7,4))

ax.plot(guest_num_sorted_r['month'],
        guest_num_sorted_r['total'],
        label='resort')
ax.plot(guest_num_sorted_c['month'],
        guest_num_sorted_c['total'],
        label='city')
ax.legend(loc='upper right',frameon=False)
plt.xticks(rotation=45);
```



In [27]:
```
data.columns
```

Out[27]:
```
Index(['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', 'children', 'babies', 'meal',
       'country', 'market_segment', 'distribution_channel',
       'is_repeated_guest', 'previous_cancellations',
       'previous_bookings_not_canceled', 'reserved_room_type',
       'assigned_room_type', 'booking_changes', 'deposit_type', 'agent',
       'company', 'days_in_waiting_list', 'customer_type', 'adr',
       'required_car_parking_spaces', 'total_of_special_requests',
       'reservation_status', 'reservation_status_date'],
      dtype='object')
```

Lets us now analyse for how much time a guest stays in a hotel

In [28]:
```
tot_stay=data[(data['is_canceled']==0) & (data['hotel']!=0)][['hotel','stays_in_weekend_nights','stays_in_week_nights']]
tot_stay['total stay']=tot_stay[['stays_in_weekend_nights','stays_in_week_nights']].sum(axis=1)
tot_stay_grouped=tot_stay[['hotel','total stay']].groupby(['hotel','total stay']).value_counts().reset_index()
tot_stay_grouped=tot_stay_grouped.rename(columns={'count':'No of stays'}).loc[:,['hotel','total stay','No of stays']]
tot_stay_grouped
```

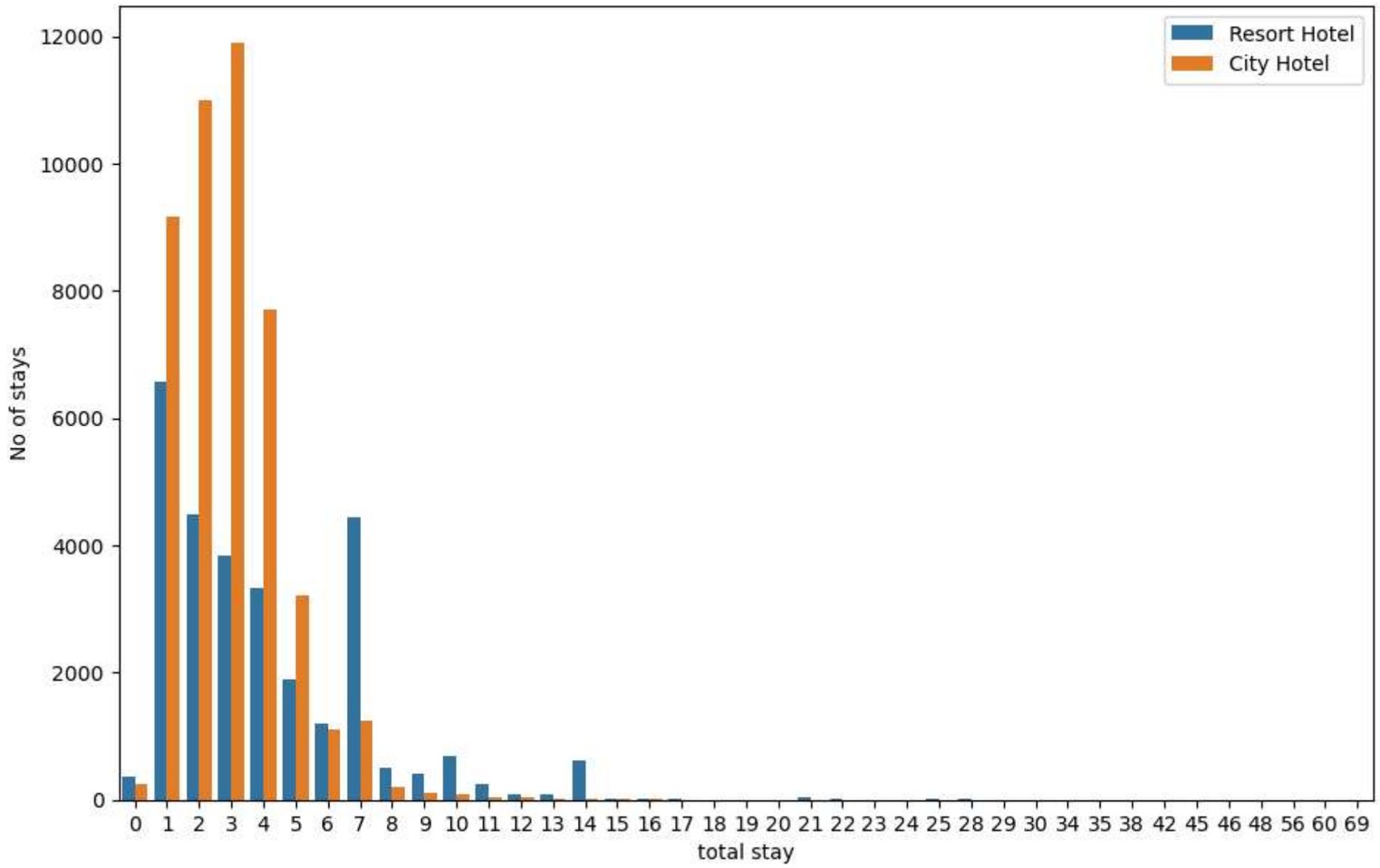
Out[28]:

	hotel	total stay	No of stays
0	City Hotel	0	251
1	City Hotel	1	9154
2	City Hotel	2	10983
3	City Hotel	3	11888
4	City Hotel	4	7694
...	...	...	...
57	Resort Hotel	45	1
58	Resort Hotel	46	1
59	Resort Hotel	56	1
60	Resort Hotel	60	1
61	Resort Hotel	69	1

62 rows × 3 columns

In [29]:
```
plt.figure(figsize=(11,7))
sns.barplot(x='total stay',
            y='No of stays',
            hue='hotel',
            hue_order=['Resort Hotel','City Hotel'],
            data=tot_stay_grouped)
plt.legend(loc='upper right')
```

Out[29]: <matplotlib.legend.Legend at 0x2c1c17ca630>



In [30]:
```
correlation=data.select_dtypes(include=['int64','float64']).corr()  #Correlation cannot be between str and int type datatypes
correlation
```

Out[30]:

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	is_r
is_canceled	1.000000	0.292879	0.016628	0.008307	-0.005943	-0.001336	0.025531	0.058185	0.004861	-0.032592	
lead_time	0.292879	1.000000	0.040311	0.127073	0.002298	0.085973	0.166896	0.117523	-0.037788	-0.020815	
arrival_date_year	0.016628	0.040311	1.000000	-0.540352	-0.000129	0.021681	0.031205	0.030176	0.054866	-0.012912	
arrival_date_week_number	0.008307	0.127073	-0.540352	1.000000	0.066580	0.018640	0.016049	0.026658	0.005430	0.010158	
arrival_date_day_of_month	-0.005943	0.002298	-0.000129	0.066580	1.000000	-0.016229	-0.028354	-0.001790	0.014606	-0.000124	
stays_in_weekend_nights	-0.001336	0.085973	0.021681	0.018640	-0.016229	1.000000	0.494177	0.094724	0.046220	0.018784	
stays_in_week_nights	0.025531	0.166896	0.031205	0.016049	-0.028354	0.494177	1.000000	0.096235	0.044654	0.020377	
adults	0.058185	0.117523	0.030176	0.026658	-0.001790	0.094724	0.096235	1.000000	0.029809	0.018745	
children	0.004861	-0.037788	0.054866	0.005430	0.014606	0.046220	0.044654	0.029809	1.000000	0.022819	
babies	-0.032592	-0.020815	-0.012912	0.010158	-0.000124	0.018784	0.020377	0.018745	0.022819	1.000000	
is_repeated_guest	-0.083747	-0.123214	0.010276	-0.031121	-0.006473	-0.086013	-0.095302	-0.141007	-0.032463	-0.008778	
previous_cancellations	0.110140	0.086024	-0.119911	0.035497	-0.027028	-0.012770	-0.013976	-0.007079	-0.024749	-0.007492	
previous_bookings_not_canceled	-0.057366	-0.073602	0.029232	-0.021007	-0.000307	-0.042860	-0.048873	-0.108878	-0.021073	-0.006537	
booking_changes	-0.144837	0.002255	0.031435	0.006305	0.011294	0.050212	0.080005	-0.041380	0.050870	0.085407	
agent	-0.046772	-0.013127	0.056422	-0.018210	0.000152	0.162404	0.196780	0.023324	0.050547	0.030428	
company	-0.083595	-0.085859	0.033677	-0.032908	0.003665	-0.080786	-0.044437	-0.166220	-0.042542	-0.009387	
days_in_waiting_list	0.054302	0.170007	-0.056353	0.022681	0.022531	-0.054402	-0.002025	-0.008377	-0.033287	-0.010607	
adr	0.046496	-0.065023	0.198422	0.076297	0.030291	0.050670	0.066846	0.224266	0.325176	0.029135	
required_car_parking_spaces	-0.195704	-0.116630	-0.013820	0.001987	0.008567	-0.018524	-0.024933	0.014419	0.056293	0.037504	
total_of_special_requests	-0.234869	-0.095922	0.108651	0.026173	0.003037	0.073155	0.068744	0.123485	0.081608	0.097716	

Eliminating same feataure correlation and finding correlation values w.r.t "is_canceled"

In [31]:
```
correlation_sliced=correlation.iloc[1:,0].abs().sort_values(ascending=False).reset_index() # Eliminating 'is_canceled' row as is_canceled, is_canceled will have corr 1 (not useful)
correlation_sliced
```

Out[31]:

	index	is_canceled
0	lead_time	0.292879
1	total_of_special_requests	0.234869
2	required_car_parking_spaces	0.195704
3	booking_changes	0.144837
4	previous_cancellations	0.110140
5	is_repeated_guest	0.083747
6	company	0.083595
7	adults	0.058185
8	previous_bookings_not_canceled	0.057366
9	days_in_waiting_list	0.054302
10	agent	0.046772
11	adr	0.046496
12	babies	0.032592
13	stays_in_week_nights	0.025531
14	arrival_date_year	0.016628
15	arrival_date_week_number	0.008307
16	arrival_date_day_of_month	0.005943
17	children	0.004861
18	stays_in_weekend_nights	0.001336

Furnishing important numerical features

- eliminating weak correlations
- Finding numerial columns only

In [150…
```
weak_corr_cols=list(correlation_sliced[correlation_sliced['is_canceled']<0.01]['index'])
print(weak_corr_cols,"<---- these will be eliminated from data as these results in very weak coorelation")
```

```
['arrival_date_week_number', 'arrival_date_day_of_month', 'children', 'stays_in_weekend_nights'] <---- these will be eliminated from data as these results in very weak coorelation
```

In [33]:
```
weak_corr_cols1=["days_in_waiting_list", "arrival_date_year"]
num_features=[col for col in data.columns if data[col].dtype!="O" and col not in weak_corr_cols1]
print(num_features)
len(num_features)
```

```
['is_canceled', 'lead_time', 'arrival_date_week_number', 'arrival_date_day_of_month', 'stays_in_weekend_nights', 'stays_in_week_nights', 'adults', 'children', 'babies', 'is_repeated_guest', 'previous_cancellations', 'previous_bookings_not_canceled', 'booking_changes', 'agent', 'company', 'adr', 'required_car_parking_spaces', 'total_of_special_requests']
```

Out[33]: 18

In [34]:
```
num_features_df=data[num_features]
num_features_df.head()
```

Out[34]:

	is_canceled	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	is_repeated_guest	previous_cancellations	previous_bookir
0	0	342	27	1	0	0	2	0.0	0	0	0	
1	0	737	27	1	0	0	2	0.0	0	0	0	
2	0	7	27	1	0	1	1	0.0	0	0	0	
3	0	13	27	1	0	1	1	0.0	0	0	0	
4	0	14	27	1	0	2	2	0.0	0	0	0	

Furnishing important categorical features

In [35]:
```
data.columns
```

Out[35]:
```
Index(['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', 'children', 'babies', 'meal',
       'country', 'market_segment', 'distribution_channel',
       'is_repeated_guest', 'previous_cancellations',
       'previous_bookings_not_canceled', 'reserved_room_type',
       'assigned_room_type', 'booking_changes', 'deposit_type', 'agent',
       'company', 'days_in_waiting_list', 'customer_type', 'adr',
       'required_car_parking_spaces', 'total_of_special_requests',
       'reservation_status', 'reservation_status_date'],
      dtype='object')
```

In [36]:
```
list_not=['country','reservation_status','booking_changes','days_in_waiting_list']
cat_features=[col for col in data.columns if data[col].dtype == "O" and col not in list_not]
cat_features
```

Out[36]:
```
['hotel',
 'arrival_date_month',
 'meal',
 'market_segment',
 'distribution_channel',
 'reserved_room_type',
 'assigned_room_type',
 'deposit_type',
 'customer_type',
 'reservation_status_date']
```

Now Let us explore 'reservation_status_date' column

In [37]:
```
print(data['reservation_status_date'].head())
data['reservation_status_date'].dtypes
```

```
0    7/1/2015
1    7/1/2015
2    7/2/2015
3    7/2/2015
4    7/3/2015
Name: reservation_status_date, dtype: object
```

Out[37]:
```
dtype('O')
```

we can clearly see that this column is of object type and needs to be converted explicitly into 'datetime' type

In [38]:
```
data_cat=data[cat_features]
```

In [39]:
```
data_cat['reservation_status_year']= pd.to_datetime(data_cat['reservation_status_date']).dt.year
data_cat['reservation_status_month']= pd.to_datetime(data_cat['reservation_status_date']).dt.month
data_cat['reservation_status_day']= pd.to_datetime(data_cat['reservation_status_date']).dt.day
data_cat.head()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_13816\358319399.py:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Lenovo\AppData\Local\Temp\ipykernel_13816\358319399.py:2: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Lenovo\AppData\Local\Temp\ipykernel_13816\358319399.py:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[39]:

	hotel	arrival_date_month	meal	market_segment	distribution_channel	reserved_room_type	assigned_room_type	deposit_type	customer_type	reservation_status_date	reservation_status_year	reservation_sta
0	Resort Hotel	July	BB	Direct	Direct	C	C	No Deposit	Transient	7/1/2015	2015	
1	Resort Hotel	July	BB	Direct	Direct	C	C	No Deposit	Transient	7/1/2015	2015	
2	Resort Hotel	July	BB	Direct	Direct	A	C	No Deposit	Transient	7/2/2015	2015	
3	Resort Hotel	July	BB	Corporate	Corporate	A	A	No Deposit	Transient	7/2/2015	2015	
4	Resort Hotel	July	BB	Online TA	TA/TO	A	A	No Deposit	Transient	7/3/2015	2015	

To do away with this annoying warning we can use warnings module

In [40]:
```
import warnings
from warnings import filterwarnings
filterwarnings('ignore')
```

Now lets see if it works, also let us drop the 'reservation_status_date' column as we have no need for it in our model now

In [41]:
```
data_cat['reservation_status_year']= pd.to_datetime(data_cat['reservation_status_date']).dt.year
data_cat['reservation_status_month']= pd.to_datetime(data_cat['reservation_status_date']).dt.month
data_cat['reservation_status_day']= pd.to_datetime(data_cat['reservation_status_date']).dt.day
data_cat.drop(columns=['reservation_status_date'],inplace=True)
data_cat.head()
```

Out[41]:

	hotel	arrival_date_month	meal	market_segment	distribution_channel	reserved_room_type	assigned_room_type	deposit_type	customer_type	reservation_status_year	reservation_status_month	reservation_
0	Resort Hotel	July	BB	Direct	Direct	C	C	No Deposit	Transient	2015	7	
1	Resort Hotel	July	BB	Direct	Direct	C	C	No Deposit	Transient	2015	7	
2	Resort Hotel	July	BB	Direct	Direct	A	C	No Deposit	Transient	2015	7	
3	Resort Hotel	July	BB	Corporate	Corporate	A	A	No Deposit	Transient	2015	7	
4	Resort Hotel	July	BB	Online TA	TA/TO	A	A	No Deposit	Transient	2015	7	

The ML algo works on numerical data only so now since we have important categorical data also, which needs to be included in our analysis we need some technique to convert the categorical (str type) data in its numerical representation, this process is called 'Feature Encoding' there are several techniques, here we will use 'Mean Encoding'

What is mean encoding? lets understand with an example:

1. we will use is_cancelled column to encode categorical columns
2. After that we will use mean encoding

In [42]:
```
data_cat=data_cat.assign(cancellation=data['is_canceled'])
data_cat.head().iloc[:,-5:-1]
```

Out[42]:

	customer_type	reservation_status_year	reservation_status_month	reservation_status_day
0	Transient	2015	7	1
1	Transient	2015	7	1
2	Transient	2015	7	2
3	Transient	2015	7	2
4	Transient	2015	7	3

In [43]:
```
print(data_cat.shape)
```

```
(119207, 13)
```

In [44]:
```
data_enc=data_cat[['hotel','cancellation']].groupby('hotel').agg(count=('cancellation','count'),sum=('cancellation','sum'),mean=('cancellation','mean'))
data_enc
```

Out[44]:

hotel	count	sum	mean
City Hotel	79160	33078	0.417863
Resort Hotel	40047	11120	0.277674

$$\text{Sum of values counts} \quad \Rightarrow \quad 79160 + 40047 = 119207$$

The mean column in the above table will represent hotel column in terms of number, for our further analysis using ML

Lets start with feature encoding (mean) for all the categorical columns

In [45]:
```
# Training test and test data function see chat
```

In [46]:
```
data_cat.columns
```

Out[46]:
```
Index(['hotel', 'arrival_date_month', 'meal', 'market_segment',
       'distribution_channel', 'reserved_room_type', 'assigned_room_type',
       'deposit_type', 'customer_type', 'reservation_status_year',
       'reservation_status_month', 'reservation_status_day', 'cancellation'],
      dtype='object')
```

In [47]:
```
def mean_encode(df,col,mean_col):
    df_dict=df.groupby(col)[mean_col].agg('mean').to_dict()
    df[col]=df[col].map(df_dict)
    return df
# 'replace' method can also be used instaed of 'map'
```

In [48]:
```
df_dict=data_cat.groupby('hotel')['cancellation'].agg('mean').to_dict()
df_dict
```

Out[48]:
```
{'City Hotel': 0.4178625568468924, 'Resort Hotel': 0.27767373336329815}
```

Mean Encoding for categorical columns is as follows:

In [49]:
```
# for col in data_cat.columns[0:9]:

#         data_cat=mean_encode(data_cat,col,'cancellation')
# print(data_cat)
```

In [50]:
```
data_cat.head()
```

Out[50]:

	hotel	arrival_date_month	meal	market_segment	distribution_channel	reserved_room_type	assigned_room_type	deposit_type	customer_type	reservation_status_year	reservation_status_month	reservation_
0	Resort Hotel	July	BB	Direct	Direct	C	C	No Deposit	Transient	2015	7	
1	Resort Hotel	July	BB	Direct	Direct	C	C	No Deposit	Transient	2015	7	
2	Resort Hotel	July	BB	Direct	Direct	A	C	No Deposit	Transient	2015	7	
3	Resort Hotel	July	BB	Corporate	Corporate	A	A	No Deposit	Transient	2015	7	
4	Resort Hotel	July	BB	Online TA	TA/TO	A	A	No Deposit	Transient	2015	7	

A better approach?

In [51]:
```
for col in data_cat.columns:
    if str(data_cat[col].dtype) not in ['int32','int64']:
        data_cat_mean_enc=mean_encode(data_cat,col,'cancellation')
data_cat_mean_enc
```

Out[51]:

	hotel	arrival_date_month	meal	market_segment	distribution_channel	reserved_room_type	assigned_room_type	deposit_type	customer_type	reservation_status_year	reservation_status_month
0	0.277674	0.374644	0.374107	0.153644	0.174812	0.330827	0.188186	0.284018	0.407862	2015	7
1	0.277674	0.374644	0.374107	0.153644	0.174812	0.330827	0.188186	0.284018	0.407862	2015	7
2	0.277674	0.374644	0.374107	0.153644	0.174812	0.391567	0.188186	0.284018	0.407862	2015	7
3	0.277674	0.374644	0.374107	0.187618	0.220568	0.391567	0.445055	0.284018	0.407862	2015	7
4	0.277674	0.374644	0.374107	0.367603	0.410607	0.391567	0.445055	0.284018	0.407862	2015	7
...	...	...	...	...	...	...	...	...	...	...	...
119202	0.417863	0.377823	0.374107	0.343313	0.410607	0.391567	0.445055	0.284018	0.407862	2017	9
119203	0.417863	0.377823	0.374107	0.367603	0.410607	0.292683	0.251603	0.284018	0.407862	2017	9
119204	0.417863	0.377823	0.374107	0.367603	0.410607	0.318108	0.251373	0.284018	0.407862	2017	9
119205	0.417863	0.377823	0.374107	0.367603	0.410607	0.391567	0.445055	0.284018	0.407862	2017	9
119206	0.417863	0.377823	0.344653	0.367603	0.410607	0.391567	0.445055	0.284018	0.407862	2017	9

119207 rows × 13 columns

Merging the categorical and numerical featured columns of the data frame to create a final DF

In [52]:
```
final_df=pd.concat([num_features_df,data_cat_mean_enc],axis=1)
final_df.head(10)
```

Out[52]:

	is_canceled	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	is_repeated_guest	...	market_segment	distribution_chann
0	0	342	27	1	0	0	2	0.0	0	0	...	0.153644	0.1748
1	0	737	27	1	0	0	2	0.0	0	0	...	0.153644	0.1748
2	0	7	27	1	0	1	1	0.0	0	0	...	0.153644	0.1748
3	0	13	27	1	0	1	1	0.0	0	0	...	0.187618	0.2205
4	0	14	27	1	0	2	2	0.0	0	0	...	0.367603	0.4106
5	0	14	27	1	0	2	2	0.0	0	0	...	0.367603	0.4106
6	0	0	27	1	0	2	2	0.0	0	0	...	0.153644	0.1748
7	0	9	27	1	0	2	2	0.0	0	0	...	0.153644	0.1748
8	1	85	27	1	0	3	2	0.0	0	0	...	0.367603	0.4106
9	1	75	27	1	0	3	2	0.0	0	0	...	0.343313	0.4106

10 rows × 31 columns

Are there any outliers?

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

Out[53]:

	is_canceled	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	is_repeated_guest	...	ma
count	119207.000000	119207.000000	119207.000000	119207.000000	119207.000000	119207.000000	119207.000000	119207.000000	119207.000000	119207.000000	...	
mean	0.370767	104.111495	27.163011	15.798812	0.927068	2.499190	1.859253	0.103996	0.007936	0.031500	...	
std	0.483012	106.875834	13.600939	8.781134	0.995122	1.897119	0.575117	0.398727	0.097383	0.174665	...	
min	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
25%	0.000000	18.000000	16.000000	8.000000	0.000000	1.000000	2.000000	0.000000	0.000000	0.000000	...	
50%	0.000000	69.000000	28.000000	16.000000	1.000000	2.000000	2.000000	0.000000	0.000000	0.000000	...	
75%	1.000000	161.000000	38.000000	23.000000	2.000000	3.000000	2.000000	0.000000	0.000000	0.000000	...	
max	1.000000	737.000000	53.000000	31.000000	19.000000	50.000000	55.000000	10.000000	10.000000	1.000000	...	

8 rows × 31 columns

In [54]:
```
final_df.columns
```

Out[54]:
```
Index(['is_canceled', 'lead_time', 'arrival_date_week_number',
       'arrival_date_day_of_month', 'stays_in_weekend_nights',
       'stays_in_week_nights', 'adults', 'children', 'babies',
       'is_repeated_guest', 'previous_cancellations',
       'previous_bookings_not_canceled', 'booking_changes', 'agent', 'company',
       'adr', 'required_car_parking_spaces', 'total_of_special_requests',
       'hotel', 'arrival_date_month', 'meal', 'market_segment',
       'distribution_channel', 'reserved_room_type', 'assigned_room_type',
       'deposit_type', 'customer_type', 'reservation_status_year',
       'reservation_status_month', 'reservation_status_day', 'cancellation'],
      dtype='object')
```

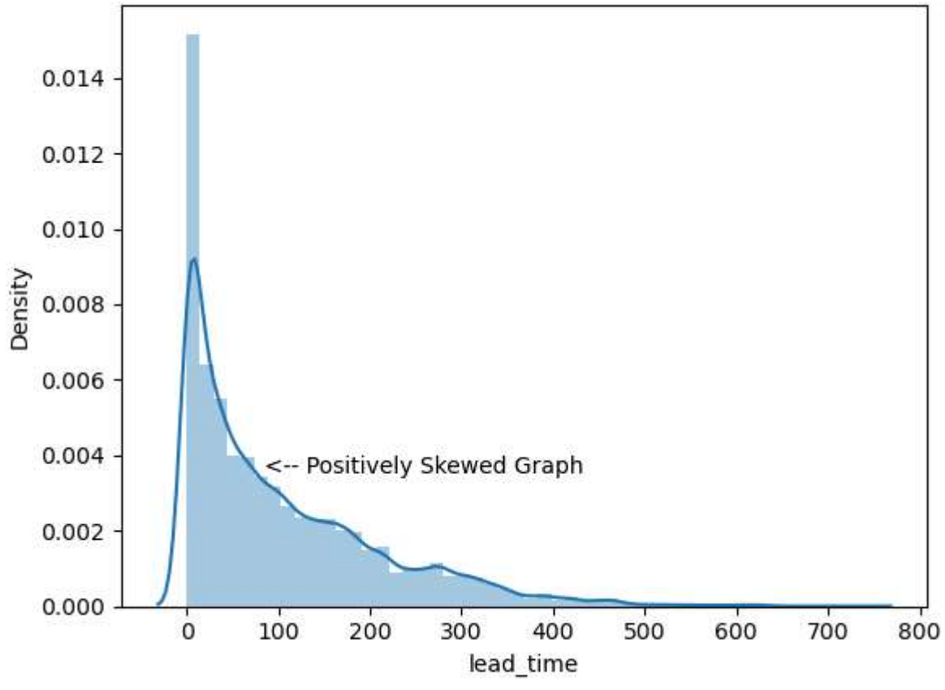
Since lead time column's 75% percentile is only 161 whereas max value is 737 this suggest that there are some outliers present in this column let us explore these outliers

1. **Use of quantile method to (0.5 and 0.95) to get the limits**

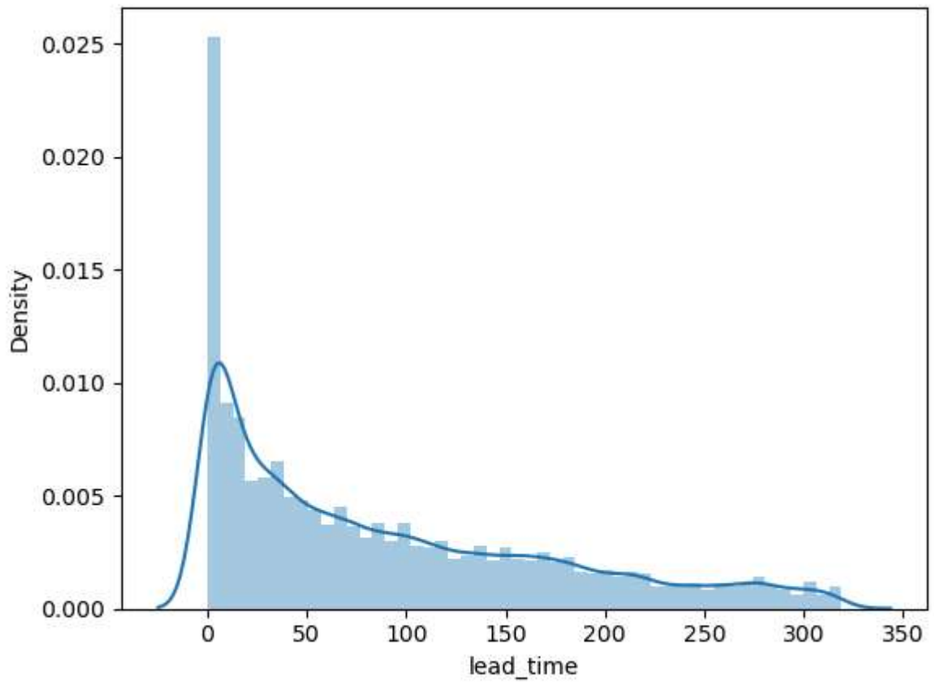
In [55]:
```
lower_bound=final_df['lead_time'].quantile(0.05)
upper_bound=final_df['lead_time'].quantile(0.95)
print(lower_bound)
print(upper_bound)
```

```
0.0
320.0
```

In [56]:
```
sns.distplot(final_df['lead_time'])
plt.text(85,0.0035,"<-- Positively Skewed Graph");
```



In [57]:
```
sns.distplot(final_df[final_df['lead_time']<upper_bound]['lead_time']);
# plt.text(85,0.0035,"<-- Positively Skewed Graph");
```



In [58]:
```
final_df_log=final_df.dropna()
```

2. Use of Logarithmic method

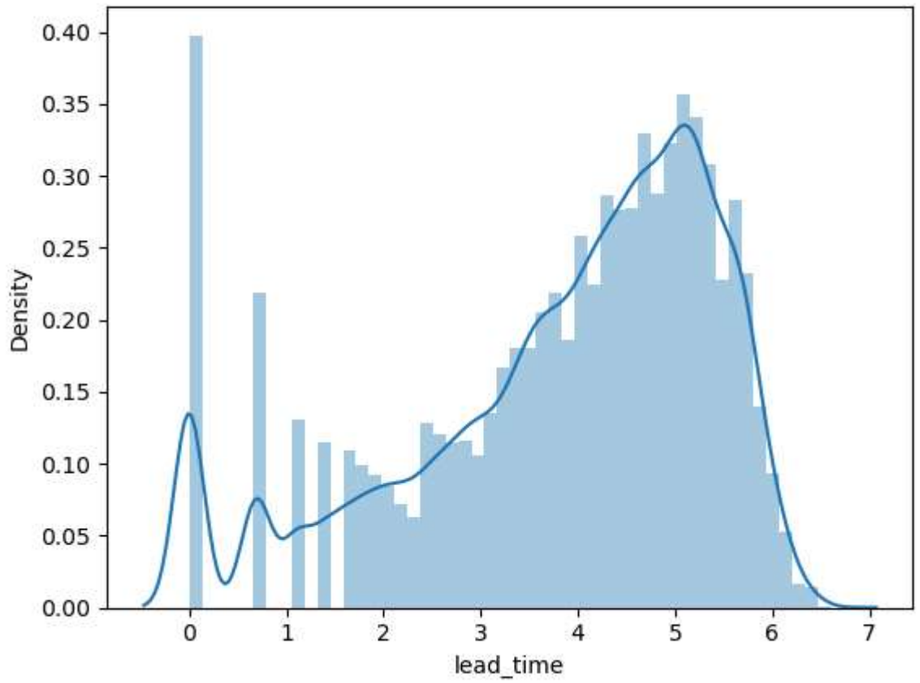
- **leat us create a function to convert original values in a col to a logarithmic values**

In [59]:
```
def change_log(col):
    final_df_log[col]=np.log1p(final_df_log[col])
```

In [60]:
```
def revert_log(col):
    final_df_log[col]=np.log1p(final_df_log[col])
```

In [61]:
```
change_log('lead_time')
```

In [62]:
```
sns.distplot(final_df_log['lead_time'].dropna());
```



In [63]:
```
final_df.adr.describe()
```

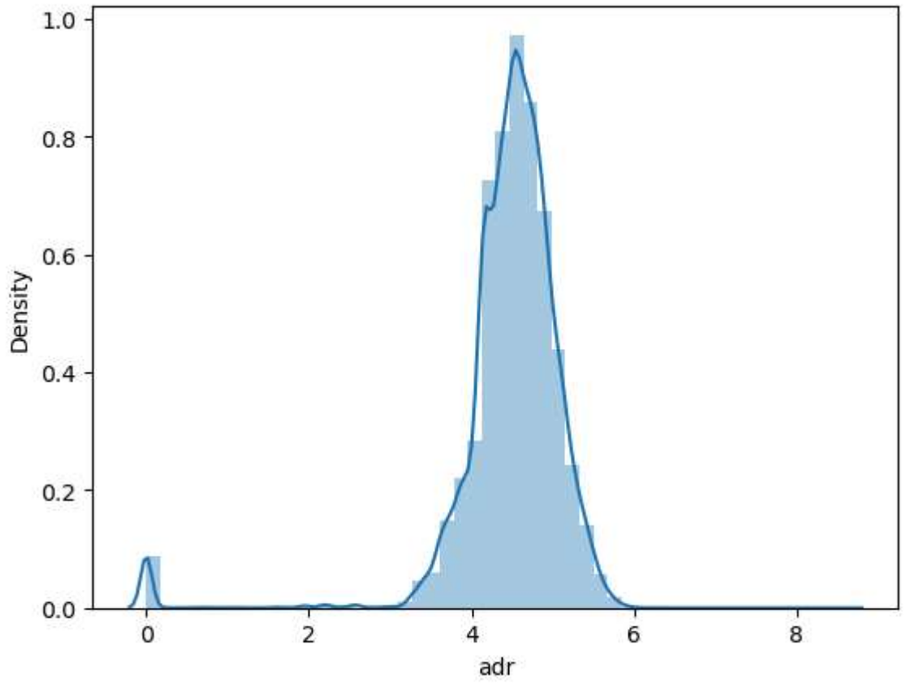
Out[63]:
```
count    119207.000000
mean        101.969357
std          50.434535
min          -6.380000
25%          69.500000
50%          94.950000
75%         126.000000
max        5400.000000
Name: adr, dtype: float64
```

Similarly doing this for 'adr' column we have:

In [64]:
```
change_log('adr')
```

In [65]:
```
sns.distplot(final_df_log['adr'].dropna())
```

Out[65]:
```
<Axes: xlabel='adr', ylabel='Density'>
```



In [66]:
```
final_df_log.isna().sum()
```

Out[66]:
```
is_canceled                     0
lead_time                       0
arrival_date_week_number        0
arrival_date_day_of_month       0
stays_in_weekend_nights         0
stays_in_week_nights            0
adults                          0
children                        0
babies                          0
is_repeated_guest               0
previous_cancellations          0
previous_bookings_not_canceled  0
booking_changes                 0
agent                           0
company                         0
adr                             1
required_car_parking_spaces     0
total_of_special_requests       0
hotel                           0
arrival_date_month              0
meal                            0
market_segment                  0
distribution_channel            0
reserved_room_type              0
assigned_room_type              0
deposit_type                    0
customer_type                   0
reservation_status_year         0
reservation_status_month        0
reservation_status_day          0
cancellation                    0
dtype: int64
```

1. data spliting
2. confusion matrix
3. knn
4. Try changing Algo parameters in KNN (e.g. dist), doc acc impact
5. choose the best algo among the diff cases
6. now tune the neigh for the best paramer from the above and find out the best accuracy
7. Advangtages of KNN : parallelisable code read more
8. dis adv: 1.curse of dimentionality 2. high no of computations (highly calculative)

Lets us now split the dependent and independent features from our final DF

In [67]:
```
final_df_log.dropna(inplace=True)
```

In [68]:
```
y=final_df_log['is_canceled'] # Dependent Variable
print(y.head())
print(y.tail())
print(y.shape)
```

```
0    0
1    0
2    0
3    0
4    0
Name: is_canceled, dtype: int64
119202    0
119203    0
119204    0
119205    0
119206    0
Name: is_canceled, dtype: int64
(119206,)
```

In [69]:
```
x=final_df_log.drop(columns=['is_canceled','cancellation']) # Independent Variable
print(x.head())
print(x.shape)
```

```
   lead_time  arrival_date_week_number  arrival_date_day_of_month  \
0   5.837730                        27                          1
1   6.603944                        27                          1
2   2.079442                        27                          1
3   2.639057                        27                          1
4   2.708050                        27                          1

   stays_in_weekend_nights  stays_in_week_nights  adults  children  babies  \
0                        0                     0       2       0.0       0
1                        0                     0       2       0.0       0
2                        0                     1       1       0.0       0
3                        0                     1       1       0.0       0
4                        0                     2       2       0.0       0

   is_repeated_guest  previous_cancellations  ...      meal  market_segment  \
0                  0                       0  ...  0.374107        0.153644
1                  0                       0  ...  0.374107        0.153644
2                  0                       0  ...  0.374107        0.153644
3                  0                       0  ...  0.374107        0.187618
4                  0                       0  ...  0.374107        0.367603

   distribution_channel  reserved_room_type  assigned_room_type  deposit_type  \
0              0.174812            0.330827            0.188186      0.284018
1              0.174812            0.330827            0.188186      0.284018
2              0.174812            0.391567            0.188186      0.284018
3              0.220568            0.391567            0.445055      0.284018
4              0.410607            0.391567            0.445055      0.284018

   customer_type  reservation_status_year  reservation_status_month  \
0       0.407862                     2015                         7
1       0.407862                     2015                         7
2       0.407862                     2015                         7
3       0.407862                     2015                         7
4       0.407862                     2015                         7

   reservation_status_day
0                       1
1                       1
2                       2
3                       2
4                       3

[5 rows x 29 columns]
(119206, 29)
```

We wil now use liner-model 'lasso' to select only the most important features for ML analysis

1. Importing relevant libraries
2. Creating selection procedure

In [76]:
```
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel
```

In [77]:
```
def ML(alp=None,r_state=None):
    global x
    global y
    return_ori(x)
    if alp is None:
```

```
        alp=float(input('enter alpha value for lasso:'))
    if r_state is None:
        r_state=int(input('enter random_state value for lasso:'))
    fea_sel_model=SelectFromModel(Lasso(alpha=alp,random_state=r_state)).fit(x,y)
    # fea_sel_model.fit(x,y)
    # fea_sel_model.get_support()
    x=x[x.columns[(fea_sel_model.get_support())]]
    return fea_sel_model.get_support(),x.columns,len(x.columns)
```

In [78]:
```
def return_ori(df):
    global x
    df=final_df_log.drop(columns=['is_canceled','cancellation'])
    x=df
```

In [79]:
```
ML() #Give input alpha=0.005,random_state=0
```

Out[79]:
```
(array([ True,  True, False, False, False,  True,  True, False, False,
         True, False,  True, False,  True,  True,  True,  True, False,
        False, False, False, False, False, False,  True, False,  True,
         True,  True]),
 Index(['lead_time', 'arrival_date_week_number', 'adults', 'children',
        'previous_cancellations', 'booking_changes', 'company', 'adr',
        'required_car_parking_spaces', 'total_of_special_requests',
        'deposit_type', 'reservation_status_year', 'reservation_status_month',
        'reservation_status_day'],
       dtype='object'),
 14)
```

What is happening in the above ML function is explained below:

In [73]:
```
fea_sel_model=SelectFromModel(Lasso(alpha=0.005,random_state=0))
```

In [74]:
```
fea_sel_model.fit(x,y)
```

Out[74]:
SelectFromModel
estimator: Lasso
Lasso

In [75]:
```
fea_sel_model.get_support()
```

Out[75]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True])
```

No of features before 'Lasso'

In [76]:
```
len(x.columns)
```

Out[76]:
```
14
```

No of features after 'Lasso'

In [77]:
```
len(x.columns[(fea_sel_model.get_support())])
```

Out[77]: 14

Overriding these important features in the x DF

In [78]:
```
x=x[x.columns[(fea_sel_model.get_support())]]
```

In [79]:
```
x.columns
```

Out[79]:
```
Index(['lead_time', 'arrival_date_week_number', 'adults', 'children',
       'previous_cancellations', 'booking_changes', 'company', 'adr',
       'required_car_parking_spaces', 'total_of_special_requests',
       'deposit_type', 'reservation_status_year', 'reservation_status_month',
       'reservation_status_day'],
      dtype='object')
```

In [80]:
```
len(x.columns) # Checking
```

Out[80]: 14

In [81]:
```
print(x.head())
print(x.shape)
```

```
   lead_time  arrival_date_week_number  adults  children  \
0   5.837730                        27       2       0.0
1   6.603944                        27       2       0.0
2   2.079442                        27       1       0.0
3   2.639057                        27       1       0.0
4   2.708050                        27       2       0.0

   previous_cancellations  booking_changes  company       adr  \
0                       0                3      0.0  0.000000
1                       0                4      0.0  0.000000
2                       0                0      0.0  4.330733
3                       0                0      0.0  4.330733
4                       0                0      0.0  4.595120

   required_car_parking_spaces  total_of_special_requests  deposit_type  \
0                            0                          0      0.284018
1                            0                          0      0.284018
2                            0                          0      0.284018
3                            0                          0      0.284018
4                            0                          1      0.284018

   reservation_status_year  reservation_status_month  reservation_status_day
0                     2015                         7                       1
1                     2015                         7                       1
2                     2015                         7                       2
3                     2015                         7                       2
4                     2015                         7                       3
(119206, 14)
```

In [82]:
```
print(y.head())
print(y.shape)
```

```
0    0
1    0
2    0
3    0
4    0
Name: is_canceled, dtype: int64
(119206,)
```

Splitting the data for modelling purpose

In [80]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.75,random_state=45)
```

In [141…
```
def app_algo(algo_name=None,n_size=None,mt=None,wt=None):
    global x_train
    global y_train
    global x_test
    global y_test
    if algo_name is None:
        algo_name=input('Choose from KNN,Decision Tree, Naive Bias,Random Forest,LR').lower()
    if algo_name=='knn':
        from sklearn.neighbors import KNeighborsClassifier
        n_size=int(input('how many neighbours?'))
        choice=input("'metric' or 'weighted'?")
        if choice=='metric':
            mt=input('Choose your metric [1. Euclidean, 2. Chebyshev, 3. Manhattan] ')
            model=KNeighborsClassifier(n_neighbors=n_size,metric=mt.lower())
        else:
            wt=input('Choose your weight [1. Uniform, 2. Distance] ')
            model=KNeighborsClassifier(n_neighbors=n_size,metric=wt.lower())
    elif algo_name=='decision tree':
        from sklearn.tree import DecisionTreeClassifier
        try:
            cri_dict=dict({'1':'entropy','2':'gini','entropy':'entropy','gini impurity':'gini'})
            cri=input('Choose Criterion: 1. entropy 2. gini impurity')
            max_dp=int(input('Choose max depth (interger no.):'))
            model=DecisionTreeClassifier(criterion=cri_dict[cri],max_depth=max_dp)
        except:
            model=DecisionTreeClassifier()
    elif algo_name=='lr':
        from sklearn.linear_model import LogisticRegression

        sol=input("enter solver to be used ['saga','liblinear']").lower()
        pen=input("Specify the penalty ['l1','l2']").lower()
        if pen=='' or pen=='none' and sol!=liblinear:
            pen=None
        model=LogisticRegression(random_state=45,solver=sol,penalty=pen)
    model.fit(x_train,y_train)
    prediction=model.predict(x_test)
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import precision_score
    return confusion_matrix(prediction,y_test),accuracy_score(prediction,y_test),precision_score(prediction,y_test)
```

Implementing KNN Algorithm

Euclidean with No. of neighbors=3

In [85]:
```
app_algo()
```

Out[85]:
```
(array([[18443,  1255],
        [  267,  9837]], dtype=int64),
 0.9489296020401315,
 0.8868553912729895)
```

Euclidean with No. of neighbors=5

In [86]:
```
app_algo()
```

Out[86]:
```
(array([[18537,  1423],
        [  173,  9669]], dtype=int64),
 0.9464465472115965,
 0.8717093400649116)
```

Euclidean with No. of neighbors=7

In [87]:
```
app_algo()
```

Out[87]:
```
(array([[18570,  1555],
        [  140,  9537]], dtype=int64),
 0.9431246225085564,
 0.8598088712585648)
```

Euclidean with No. of neighbors=9

In [88]:
```
app_algo()
```

Out[88]:
```
(array([[18585,  1672],
        [  125,  9420]], dtype=int64),
 0.9397020334205758,
 0.8492607284529391)
```

Euclidean with No. of neighbors=11

In [89]:
```
app_algo()
```

Out[89]:
```
(array([[18604,  1751],
        [  106,  9341]], dtype=int64),
 0.9376887457217636,
 0.8421384781824739)
```

Chebyshev with No. of neighbors=3

In [90]:
```
app_algo()
```

Out[90]:
```
(array([[17894,  1594],
        [  816,  9498]], dtype=int64),
 0.9191329440977115,
 0.8562928236566895)
```

Chebyshev with No. of neighbors=5

In [91]:
```
app_algo()
```

```
Out[91]: (array([[18077,  1791],
                 [  633,  9301]], dtype=int64),
          0.9186631769679887,
          0.8385322755138839)
```

Chebyshev with No. of neighbors=7

```
In [92]: app_algo()
```

```
Out[92]: (array([[18178,  1969],
                 [  532,  9123]], dtype=int64),
          0.9160794577545132,
          0.8224846736386585)
```

Chebyshev with No. of neighbors=9

```
In [93]: app_algo()
```

```
Out[93]: (array([[18258,  2131],
                 [  452,  8961]], dtype=int64),
          0.9133279645661365,
          0.8078795528308691)
```

Chebyshev with No. of neighbors=11

```
In [94]: app_algo()
```

```
Out[94]: (array([[18303,  2240],
                 [  407,  8852]], dtype=int64),
          0.9111804576874035,
          0.7980526505589615)
```

Manhattan with No. of neighbors=3

```
In [95]: app_algo()
```

```
Out[95]: (array([[18479,  1203],
                 [  231,  9889]], dtype=int64),
          0.9518824239983894,
          0.8915434547421565)
```

Manhattan with No. of neighbors=5

```
In [96]: app_algo()
```

```
Out[96]: (array([[18554,  1366],
                 [  156,  9726]], dtype=int64),
          0.9489296020401315,
          0.8768481788676523)
```

Manhattan with No. of neighbors=7

```
In [97]: app_algo()
```

```
Out[97]: (array([[18601,  1467],
                 [  109,  9625]], dtype=int64),
          0.9471176431112006,
          0.8677425171294627)
```

Manhattan with No. of neighbors=9

In [98]:
```
app_algo()
```

Out[98]:
```
(array([[18615,  1587],
        [   95,  9505]], dtype=int64),
 0.9435608348432991,
 0.8569239091236928)
```

Manhattan with No. of neighbors=11

In [99]:
```
app_algo()
```

Out[99]:
```
(array([[18628,  1671],
        [   82,  9421]], dtype=int64),
 0.9411784443997048,
 0.8493508835196538)
```

SUMMARY-KNN

In [104…
```
table={
       ('Metric','Euclidean'):[0.9489,0.9464,0.9431,0.9397,0.9377],
       ('Metric','Chevyshev'):[0.9191,0.9187,0.9161,0.9133,0.9112],
       ('Metric','Manhatttan'):[0.9519,0.9489,0.9471,0.9436,0.9412]
        }
row_index=[3,5,7,9,11]
summary=pd.DataFrame(table,index=row_index)
summary
```

Out[104…

	Metric		
	Euclidean	**Chevyshev**	**Manhatttan**
3	0.9489	0.9191	0.9519
5	0.9464	0.9187	0.9489
7	0.9431	0.9161	0.9471
9	0.9397	0.9133	0.9436
11	0.9377	0.9112	0.9412

Implementing Decision Tree

Default with no parameters

In [107…
```
app_algo()
```

Out[107…
```
(array([[17916,   750],
        [  794, 10342]], dtype=int64),
 0.9481913965505671,
 0.932383699963938)
```

Entropy with max_depth=5

In [108…
```
app_algo()
```

Out[108…
```
(array([[18668,  6879],
        [   42,  4213]], dtype=int64),
 0.7677672639420173,
 0.3798232960692391)
```

Entropy with max_depth=7

In [109…
```
app_algo()
```

Out[109…
```
(array([[18643,  5273],
        [   67,  5819]], dtype=int64),
 0.8208173948057177,
 0.5246123332131266)
```

Entropy with max_depth=9

In [110…
```
app_algo()
```

Out[110…
```
(array([[18221,  4263],
        [  489,  6829]], dtype=int64),
 0.8405476142540769,
 0.6156689505950235)
```

Gini-Impurity with max_depth=5

In [111…
```
app_algo()
```

Out[111…
```
(array([[18473,  5283],
        [  237,  5809]], dtype=int64),
 0.8147775317092812,
 0.5237107825459791)
```

Gini-Impurity with max_depth=7

In [112…
```
app_algo()
```

Out[112…
```
(array([[18662,  5040],
        [   48,  6052]], dtype=int64),
 0.8292732031407288,
 0.5456184637576632)
```

Gini-Impurity with max_depth=9

In [113…
```
app_algo()
```

Out[113…
```
(array([[18258,  4142],
        [  452,  6950]], dtype=int64),
 0.8458492718609489,
 0.6265777136675081)
```

Gini-Impurity with max_depth=12

In [96]:
```
app_algo()
```

Out[96]:
```
(array([[18504,  3592],
        [  206,  7500]], dtype=int64),
 0.8725588886651903,
 0.6761630003606203)
```

Gini-Impurity with max_depth=18

In [97]:
```
app_algo()
```

Out[97]:
```
(array([[18289,  1626],
        [  421,  9466]], dtype=int64),
 0.9313133346755251,
 0.8534078615218176)
```

SUMMARY-Decision Tree

1. Default with no parameters given by the user

Accuracy Score=0.9482

2. Changing Criterion and Max_depth

In [119…
```
table={

       ('DT','Entropy'):[0.7678,0.8208,0.8405],
       ('DT','Gini Impurity'):[0.8148,0.8293,0.8458],
        }
row_index=[5,7,9]
summary=pd.DataFrame(table,index=row_index)
summary
```

Out[119…

	DT	
	Entropy	**Gini Impurity**
5	0.7678	0.8148
7	0.8208	0.8293
9	0.8405	0.8458

Implementing Logistic Resoning

No Parameters default-lbfgs with l2 penalty

In [105…
```
app_algo()
```

Out[105…
```
(array([[15893,  4337],
        [ 2817,  6755]], dtype=int64),
 0.7599489967116301,
 0.608997475658132)
```

Saga solver with l1 penalty

In [135…
```
app_algo()
```

Out[135…
```
(array([[16906,  6303],
        [ 1804,  4789]], dtype=int64),
 0.7279712770954969,
 0.43175261449693475)
```

Saga solver with l2 penalty

In [134…
```
app_algo()
```

Out[134…
```
(array([[16906,  6303],
        [ 1804,  4789]], dtype=int64),
 0.7279712770954969,
 0.43175261449693475)
```

Liblinear solver with l1 penalty

In [133…
```
app_algo()
```

Out[133…
```
(array([[18123,  5165],
        [  587,  5927]], dtype=int64),
 0.8069928192738742,
 0.5343490804183195)
```

Liblinear solver with l2 penalty

In [132…
```
app_algo()
```

Out[132…
```
(array([[18119,  5190],
        [  591,  5902]], dtype=int64),
 0.8060197302194484,
 0.5320952037504508)
```

SUMMARY- Logistic Regression

1. Default with no parameters given by the user

Accuracy Score=0.7599

2. Changing Solver and Penalty

In [147…
```
table={

       ('LR','Liblinear'):[0.8070,0.8060],
       ('LR','saga'):[0.7280,0.7280],
        }
row_index=['l1','l2']
summary=pd.DataFrame(table,index=row_index)
summary
```

Out[147…

	LR	
	Liblinear	**saga**
l1	0.807	0.728
l2	0.806	0.728

Are these results acceptable? for this we need to find out baseline accuracy of our model

- **let us create a function so that it becomes easier to create baseline accuracy scores with different parameters**

In [93]:
```
def base_acc(strgy=None):
    from sklearn.dummy import DummyClassifier
    from sklearn.metrics import accuracy_score
    strgy=input('which strategy would you like to use ?: 1. most_frequent 2. stratified 3. uniform 4.constant, enter the corresponding no ')
    strgy_dict=dict({'1':'most_frequent','2':'stratified','3':'uniform','4':'constant'})
    if strgy=='1' or strgy=='2' or strgy=='3':
        dummy_clf=DummyClassifier(strategy=strgy_dict[strgy])
    else:
        const_val=int(input('Give a constant value?'))
        dummy_clf=DummyClassifier(strategy=strgy_dict[strgy],constant=const_val)
    dummy_clf.fit(x_train,y_train)
    prediction=dummy_clf.predict(x_test)
    return f' ACCURACY SCORE for {strgy_dict[strgy]} is {accuracy_score(prediction,y_test):.2f}'
```

In [81]:
```
base_acc()
```

Out[81]: ' ACCURACY SCORE for most_frequent is 0.63'

In [82]:
```
base_acc()
```

Out[82]: ' ACCURACY SCORE for stratified is 0.54'

In [83]:
```
base_acc()
```

Out[83]: ' ACCURACY SCORE for uniform is 0.50'

In [94]:
```
base_acc()
```

Out[94]: ' ACCURACY SCORE for constant is 0.63'

In [95]:
```
base_acc()
```

Out[95]: ' ACCURACY SCORE for constant is 0.37'

CONCLUSION

The given dataset was trained with 75% data and was tested on 25% of the data. following models with different parameters via scikit-learn library were used:

- KNN
- Decision Tree
- Logistic Regression\

All the above models resulted in better accuracy score when compared to baseline accuracy calculated above. Analysing the accuracy score of different models the best performing model is KNN model with 3 neighbors and Manhattan metric which gives us 95.19% accuracy score

In [ ]:
```

```