Projet Data Mining

Hotels Booking Demand DataSet

Conducted By :

- ***Mme Elasri***

Made By :

-Hmidich Aymane

-Slaoui Mehdi

Contents

1 DataSet: 2
- 1.1 Data description: . . . 2
- 1.2 Data visualisation: . . . 2
  - 1.2.1 Hotel types: . . . 2
  - 1.2.2 Canceled Booking: . . . 3
  - 1.2.3 Distribution of lead time for hotel types: . . . 3
  - 1.2.4 Data Type and Missing Values: . . . 4

2 Data preprocessing : 5
- 2.1 Verify missing values : . . . 5
- 2.2 Changing arrival year, month and day feature to datetime format called arrival date : . . . 6
- 2.3 Verify the timestamp between reservation_status _date and arrival_date : 7
- 2.4 Data Processing : . . . 8
  - 2.4.1 Meals Column : . . . 8
  - 2.4.2 Columns Data types : . . . 8
  - 2.4.3 Agent and Company Columns : . . . 9
  - 2.4.4 Senseless Rows: . . . 9
  - 2.4.5 Numerical Data Normalization : . . . 10
  - 2.4.6 Categorical Variables Encoding : . . . 10

3 Exploratory Data Analysis : 12
- 3.1 Create Data Set Summary Statistics : . . . 12
  - 3.1.1 Date Variables : . . . 12
  - 3.1.2 Categorical Variables : . . . 13
  - 3.1.3 Integer and Numeric Variables : . . . 13
- 3.2 The distribution of hotel type for cancellation : . . . 14
- 3.3 Distribution of Cancellation and Number of Adults: . . . 15
- 3.4 Target Variables : . . . 17
  - 3.4.1 Meal: . . . 17
  - 3.4.2 Repeated Guest: . . . 17

4 Modeling : 19
- 4.1 Prepare Data: . . . 19
- 4.2 Models : . . . 20
  - 4.2.1 RandomForestClassifier Algorithm : . . . 20
  - 4.2.2 LogisticRegression Algorithm : . . . 20
  - 4.2.3 Train Data and Compare Results: . . . 20

4.3 Hyperparameter Tuning : 21
4.3.1 Hyperparameter Tuning for Logistic Regression : 21
4.3.2 Hyperparameter Tuning for Random Forrest Classifier : 22

5 Model Evaluation : 24
5.1 Prepare the tools : 24
5.2 Confusion Matrix : 25
5.2.1 Regression Model : 25
5.2.2 Random Forrest Classifier Model : 26
5.2.3 Conclusion : 26
5.3 Classification Report : 27
5.3.1 Regression Model : 27
5.3.2 Random Forrest Classifier Model : 27
5.3.3 Conclusion : 27
5.4 ROC CURVE : 28
5.4.1 Regression Model : 28
5.4.2 Random Forrest Classifier Model : 28
5.4.3 Conclusion : 29
5.5 Using different metrics with cross validation : 29
5.5.1 Regression Model : 29
5.5.2 Random Forrest Classifier Model : 29
5.5.3 Conclusion : 30

Introduction :

The hotel industry almost like any other industry is evolving very quickly due to the digitalization of their services, therefore hotel owners can't afford being left behind, and have no choice but to adopt the new ways of running their business.

One of the most important new features is online booking, people now can book hotel rooms without paying a single penny, which can drive a lot of traffic, although this has been proven very useful it has an obvious downside, especially in the peak seasons: Hotels can't afford turning down bookings if they are not sure that their supposed customers will arrive.

That's where data mining plays an important role, after gathering some data, preprocessing it and training a classification model, we can predict with high accuracy if the customer will cancel his booking or not, and just like that we can save the hotel, hotel owner and the world a lot of money aka value.

Chapter 1

DataSet:

1.1 Data description:

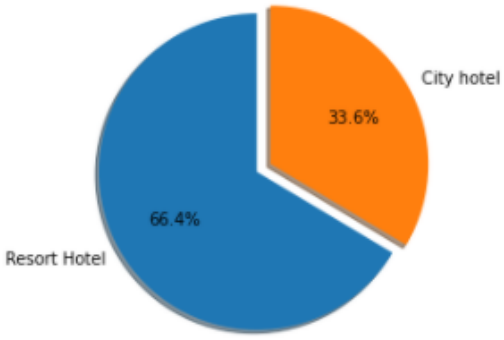
This data describes two datasets with hotel demand data. One of the hotels (H1) is a resort hotel and the other is a city hotel (H2). Both datasets share the same structure, with 31 variables describing the 40,060 observations of H1 and 79,330 observations of H2. Each observation represents a hotel booking. Since this is hotel real data, all data elements pertaining hotel or costumer identification were deleted.

1.2 Data visualisation:

We will start our adventure by visualize attitudes of some attribute to have a general idea about the data Set.

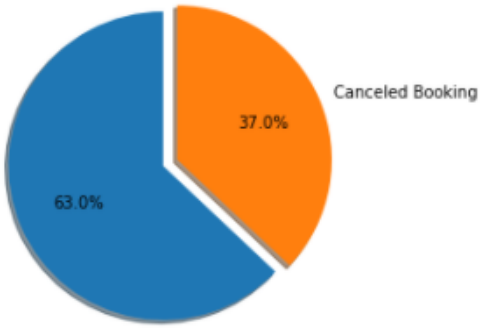
1.2.1 Hotel types:

```
sizes=hotels.groupby('hotel').count()['is_canceled']
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=(0, 0.1), labels=['Resort Hotel','City hotel'], autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
plt.show()
```



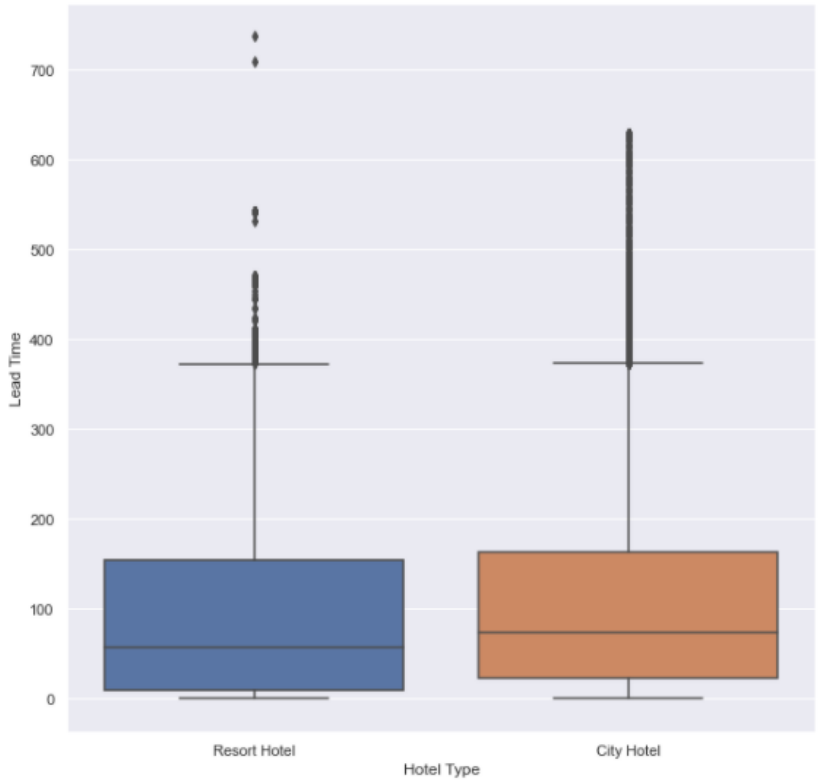
1.2.2 Canceled Booking:

```
sizes=hotels.groupby('is_canceled').count()['hotel']
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=(0, 0.1), labels=['','Canceled Booking'], autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
plt.show()
```



1.2.3 Distribution of lead time for hotel types:

```
sns.boxplot(x = 'hotel', y = 'lead_time', data = hotels)
sns.set(rc={'figure.figsize':(10,10)})
plt.xlabel('Hotel Type')
plt.ylabel('Lead Time')
plt.show()
```



1.2.4 Data Type and Missing Values:

```
hotels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 32 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           119390 non-null  object
 1   is_canceled                     119390 non-null  int64
 2   lead_time                       119390 non-null  int64
 3   arrival_date_year               119390 non-null  int64
 4   arrival_date_month              119390 non-null  object
 5   arrival_date_week_number        119390 non-null  int64
 6   arrival_date_day_of_month       119390 non-null  int64
 7   stays_in_weekend_nights         119390 non-null  int64
 8   stays_in_week_nights            119390 non-null  int64
 9   adults                          119390 non-null  int64
 10  children                        119386 non-null  float64
 11  babies                          119390 non-null  int64
 12  meal                            119390 non-null  object
 13  country                         118902 non-null  object
 14  market_segment                  119390 non-null  object
 15  distribution_channel            119390 non-null  object
 16  is_repeated_guest               119390 non-null  int64
 17  previous_cancellations          119390 non-null  int64
 18  previous_bookings_not_canceled  119390 non-null  int64
 19  reserved_room_type              119390 non-null  object
 20  assigned_room_type              119390 non-null  object
 21  booking_changes                 119390 non-null  int64
 22  deposit_type                    119390 non-null  object
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64
 26  customer_type                   119390 non-null  object
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64
 29  total_of_special_requests       119390 non-null  int64
 30  reservation_status              119390 non-null  object
 31  reservation_status_date         119390 non-null  object
dtypes: float64(4), int64(16), object(12)
memory usage: 29.1+ MB
```

As we can see, we have some missing values that we will to take care of later:

children : 4 / country : 488 / agent : 16340 / company : 112593

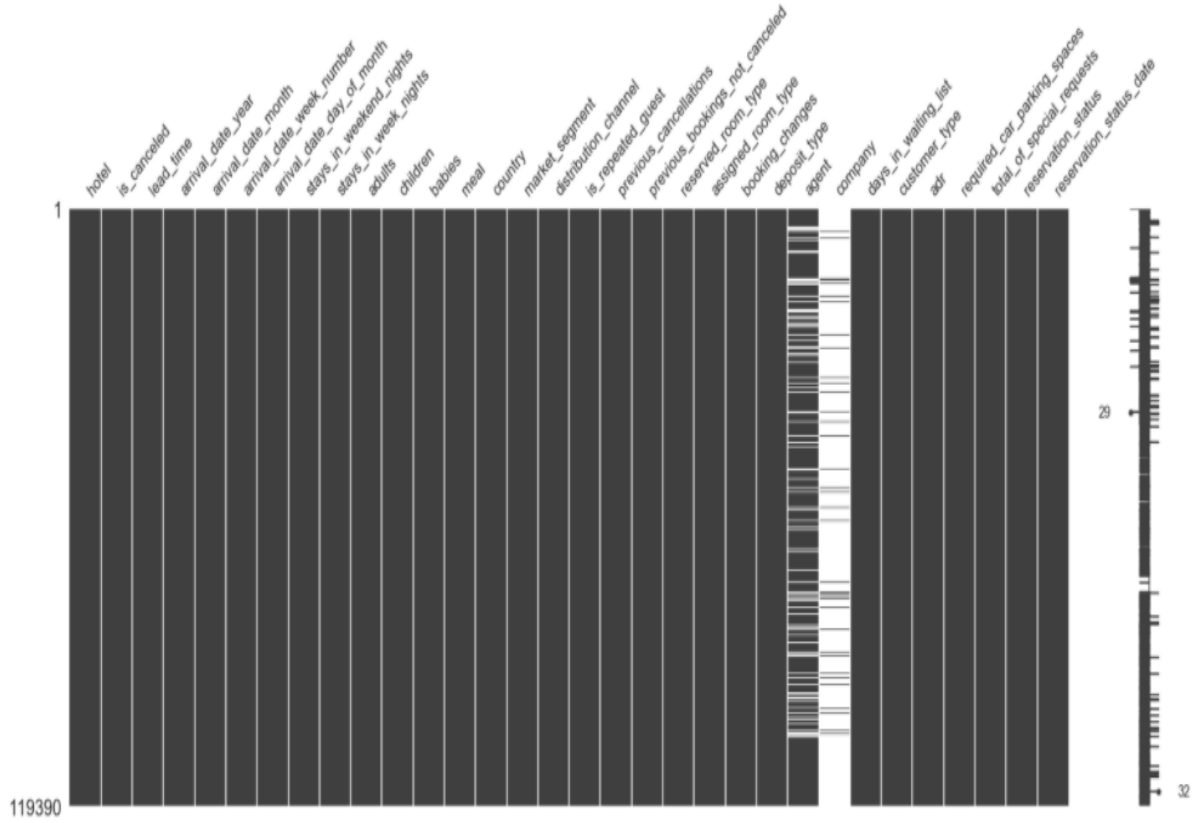
Chapter 2

Data preprocessing :

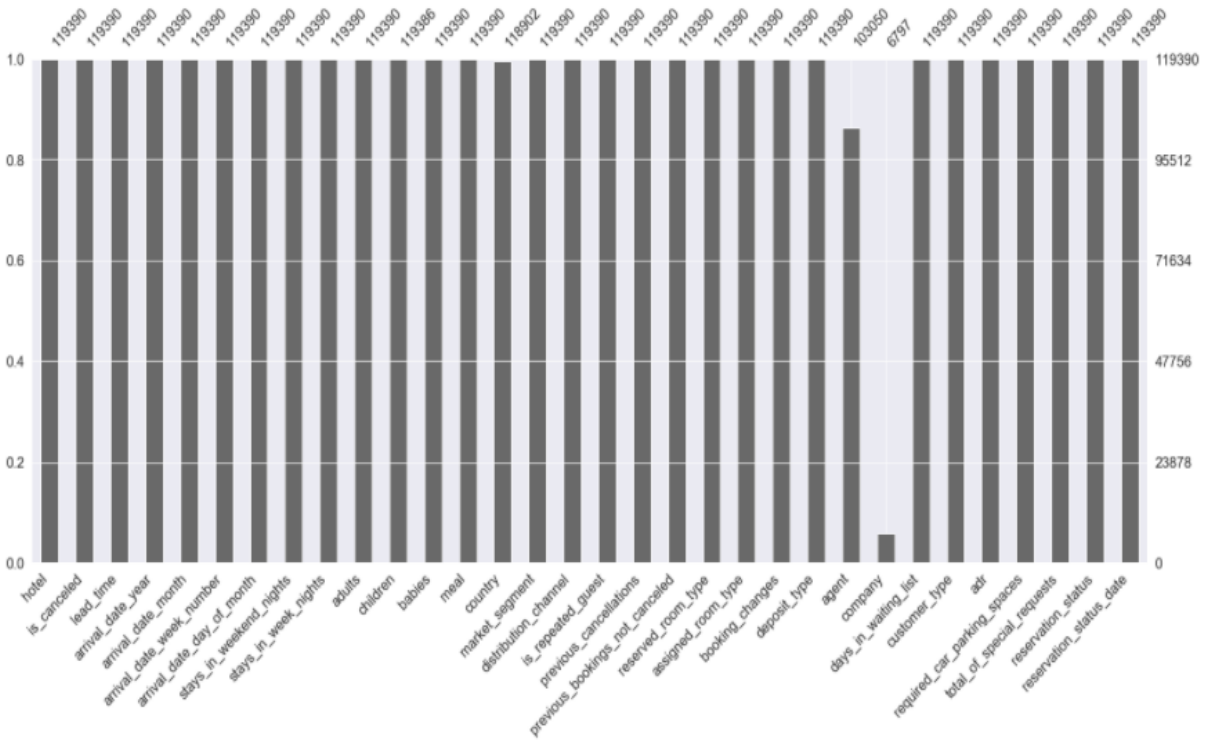
2.1 Verify missing values :

We will use missingno library to visualize our missing data, get a better understanding of it and detect any relation ship between the missing data in the columns.

```
msno.matrix(hotels)
plt.show()
```



```
msno.bar(hotels)
plt.show()
```



Now it's time to start cleaning the Data Set by removing all the missing values from country's and children's columns.

```
hotels.dropna(subset=['country', 'children'], inplace=True,axis='rows')
hotels.reset_index(drop=True, inplace=True)
```

2.2 Changing arrival year, month and day feature to datetime format called arrival date :

To get this done, we need to use pandas.to_datetime function. First, we will create a new column called arrival_date that contain day, month and year of the arrival date, then convert it to a DateTime type.

```
day_Month=hotels['arrival_date_day_of_month'].astype(str)
Month = hotels['arrival_date_month'].astype(str)
Year= hotels['arrival_date_year'].astype(str)
hotels['arrival_date']=day_Month+' '+Month+' '+Year
hotels['arrival_date'] = pd.to_datetime(hotels['arrival_date'], format='%d %B %Y')
```

Result:

```
X=hotels[['arrival_date_year','arrival_date_month','arrival_date_day_of_month','arrival_date']]
print(X.dtypes)
X.head()
```

```
arrival_date_year                    int64
arrival_date_month                  object
arrival_date_day_of_month            int64
arrival_date                datetime64[ns]
dtype: object
```

	arrival_date_year	arrival_date_month	arrival_date_day_of_month	arrival_date
0	2015	July	1	2015-07-01
1	2015	July	1	2015-07-01
2	2015	July	1	2015-07-01
3	2015	July	1	2015-07-01
4	2015	July	1	2015-07-01

We can now drop year, month and day of the arrival date columns from the data set.

```
hotels.drop(['arrival_date_day_of_month', 'arrival_date_month', 'arrival_date_year'],
            axis='columns', inplace=True)
```

2.3 Verify the timestamp between reservation_status _date and arrival_date :

We will start by converting reservation_status_date feature to Datetime :

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

Then compare the timestamp between reservation_status_date and arrival_date. All the variable reservation_status_date must occur after or at the same date as the input variable arrival_date.

```
checkout_index=(hotels['reservation_status_date'][hotels['reservation_status']=='Check-Out']).index
incon_dates=(hotels['arrival_date'][checkout_index])>hotels['reservation_status_date'][checkout_index]
incon_dates.unique()
```

```
array([False])
```

Conclusion : There are no arrival dates after the checkout so we have nothing to change in the data set.

2.4 Data Processing :

2.4.1 Meals Column :

Here we will change 'Undefined' category to 'SC' category, they refer to the same type of meal booked.

```
print(hotels['meal'].unique())
hotels['meal'] = hotels['meal'].replace({'Undefined': 'SC'})
hotels['meal'].unique()
```

```
['BB' 'FB' 'HB' 'SC' 'Undefined']
```

```
array(['BB', 'FB', 'HB', 'SC'], dtype=object)
```

2.4.2 Columns Data types :

```
hotels.dtypes
```

```
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
```

We can see that all column data types are correct except children's column, which we can change into integer.

```
hotels['children']=hotels['children'].astype('int64')
```

Now we can standardize numerical data to help the ML model algorithm.

2.4.3 Agent and Company Columns :

We will impute NULL in agent and category columns with 0, so that we can use the column in modeling.

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.NaN, strategy='constant', fill_value=0)
imputer = imputer.fit(hotels[['agent','company']])
hotels[['agent','company']] = imputer.transform(hotels[['agent','company']])
```

Or we could have just used :

```
hotels = hotels.fillna({'agent':0,'company':0})
```

But simple imputer is way faster.

2.4.4 Senseless Rows:

```
print(hotels['distribution_channel'].unique())
print(hotels[hotels['distribution_channel']=='Undefined'].index)
```

```
['Direct' 'Corporate' 'TA/TO' 'Undefined' 'GDS']
Int64Index([14185], dtype='int64')
```

We only have one row with undefined distribution_channel, we will delete that row.

```
d=hotels[hotels['distribution_channel']=='Undefined'].index
hotels.drop(d, inplace=True)
hotels.reset_index(drop=True, inplace=True)
```

We have also observed that some columns have 0 in adults, children and babies which doesn't make any sense that's why we decided to remove those columns.

```
z=hotels[hotels['adults']==0]
z=z[z['children']==0].index
print(len(z))
hotels.drop(z,inplace=True)
hotels.reset_index(drop=True, inplace=True)
```

```
170
```

2.4.5 Numerical Data Normalization :

```
print(hotels['lead_time'].max())
print(hotels['babies'].max())
```

```
737
10
```

As we observe, numerical data in our data set have different ranges so it is interesting to standardize our data.

```
from sklearn.preprocessing import StandardScaler

columns_to_standarize=['lead_time','arrival_date_week_number', 'stays_in_weekend_nights',
                       'stays_in_week_nights','adults','children','babies','previous_cancellations',
                       'previous_bookings_not_canceled','booking_changes','days_in_waiting_list',
                       'adr','required_car_parking_spaces','total_of_special_requests']

scaled_data=hotels.copy()
scaler = StandardScaler()
scaled_columns = scaler.fit_transform(scaled_data[columns_to_standarize].values)

scaled_data[columns_to_standarize]=scaled_columns
scaled_data.head()
```

	hotel	is_canceled	lead_time	arrival_date_week_number	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	meal	...
0	Resort Hotel	0	2.223407	-0.012256	-0.932429	-1.316808	0.244755	-0.261059	-0.081618	BB	...
1	Resort Hotel	0	5.918350	-0.012256	-0.932429	-1.316808	0.244755	-0.261059	-0.081618	BB	...
2	Resort Hotel	0	-0.910279	-0.012256	-0.932429	-0.790536	-1.483635	-0.261059	-0.081618	BB	...
3	Resort Hotel	0	-0.854153	-0.012256	-0.932429	-0.790536	-1.483635	-0.261059	-0.081618	BB	...
4	Resort Hotel	0	-0.844799	-0.012256	-0.932429	-0.264264	0.244755	-0.261059	-0.081618	BB	...

5 rows × 30 columns

2.4.6 Categorical Variables Encoding :

Label encoding is straight but it has the disadvantage that the numeric values can be misinterpreted by algorithms as having some sort of hierarchy/order in them, but if we chose one hot encoder we will end up with over 200 columns which is very expensive in terms of computing. That's why we will experiment with label encoding first and see ho the satisfying the results are.

```
from sklearn.preprocessing import LabelEncoder
columns_to_label=['hotel','meal','country','market_segment','distribution_channel',
                  'reserved_room_type','assigned_room_type','deposit_type','customer_type',
                  'reservation_status']

labelencoder = LabelEncoder()
for column_to_label in columns_to_label:
    scaled_data[column_to_label] = labelencoder.fit_transform(scaled_data[column_to_label])

scaled_data.head()
```

	hotel	is_canceled	lead_time	arrival_date_week_number	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	meal	...
0	1	0	2.223407	-0.012256	-0.932429	-1.316808	0.244755	-0.261059	-0.081618	0	...
1	1	0	5.918350	-0.012256	-0.932429	-1.316808	0.244755	-0.261059	-0.081618	0	...
2	1	0	-0.910279	-0.012256	-0.932429	-0.790536	-1.483635	-0.261059	-0.081618	0	...
3	1	0	-0.854153	-0.012256	-0.932429	-0.790536	-1.483635	-0.261059	-0.081618	0	...
4	1	0	-0.844799	-0.012256	-0.932429	-0.264264	0.244755	-0.261059	-0.081618	0	...

5 rows × 30 columns

Chapter 3

Exploratory Data Analysis :

3.1 Create Data Set Summary Statistics :

In this section of the project we will display some data set summary statistics. It include mean, median, mode, minimum value, maximum value, range, standard deviation, etc. Just enough information to understand some features.

3.1.1 Date Variables :

```
hotels.describe(datetime_is_numeric=True)[['reservation_status_date','arrival_date']]
```

	reservation_status_date	arrival_date
count	118898	118898
mean	2016-07-30 07:37:53.336756224	2016-08-29 02:40:04.440783360
min	2014-10-17 00:00:00	2015-07-01 00:00:00
25%	2016-02-02 00:00:00	2016-03-14 00:00:00
50%	2016-08-08 00:00:00	2016-09-07 00:00:00
75%	2017-02-09 00:00:00	2017-03-19 00:00:00
max	2017-09-14 00:00:00	2017-08-31 00:00:00
std	NaN	NaN

3.1.2 Categorical Variables :

```
hotels.describe(include=['object'])
```

	hotel	meal	country	market_segment	distribution_channel	reserved_room_type	assigned_room_type	deposit_type	customer_type	reservation_status
count	118898	118898	118898	118898	118898	118898	118898	118898	118898	118898
unique	2	5	177	7	5	10	12	3	4	3
top	City Hotel	BB	PRT	Online TA	TA/TO	A	A	No Deposit	Transient	Check-Out
freq	79302	91863	48586	56402	97730	85601	73863	104163	89174	74745

3.1.3 Integer and Numeric Variables :

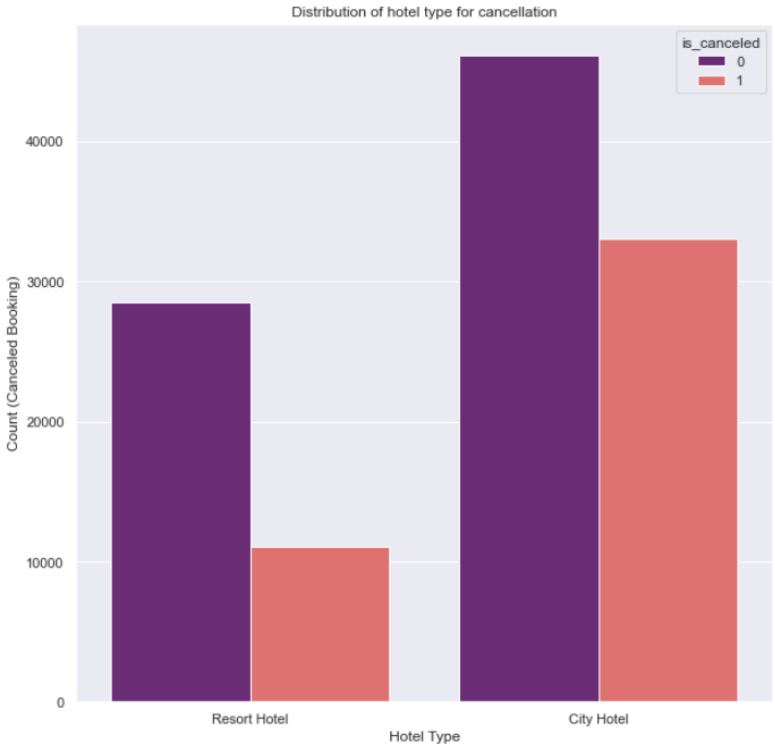
```
hotels.describe(include=['int64','float64'])
```

	is_canceled	lead_time	arrival_date_week_number	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies
count	118898.000000	118898.000000	118898.000000	118898.000000	118898.000000	118898.000000	118898.000000	118898.000000
mean	0.371352	104.311435	27.166555	0.928897	2.502145	1.858391	0.104207	0.007948
std	0.483168	106.903309	13.589971	0.996216	1.900168	0.578576	0.399172	0.097380
min	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	18.000000	16.000000	0.000000	1.000000	2.000000	0.000000	0.000000
50%	0.000000	69.000000	28.000000	1.000000	2.000000	2.000000	0.000000	0.000000
75%	1.000000	161.000000	38.000000	2.000000	3.000000	2.000000	0.000000	0.000000
max	1.000000	737.000000	53.000000	16.000000	41.000000	55.000000	10.000000	10.000000

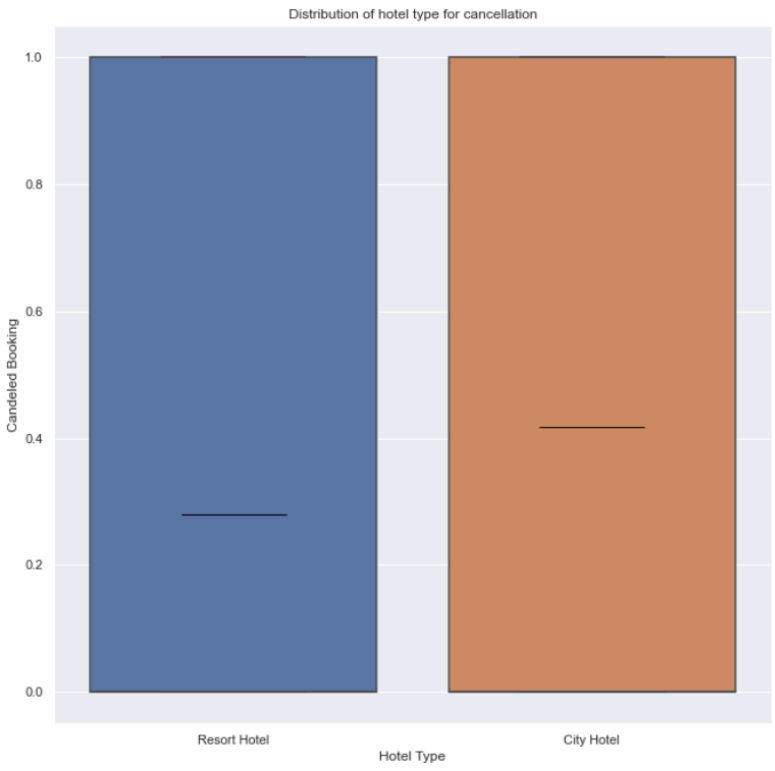
babies	is_repeated_guest	previous_cancellations	previous_bookings_not_canceled	booking_changes	agent	company	days_in_waiting_list
118898.000000	118898.000000	118898.000000	118898.000000	118898.000000	102894.000000	6623.000000	118898.000000
0.007948	0.032011	0.087142	0.131634	0.221181	86.545532	189.624792	2.330754
0.097380	0.176029	0.845869	1.484672	0.652785	110.714259	132.124298	17.630452
0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	6.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	9.000000	62.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	14.000000	179.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	229.000000	270.000000	0.000000
10.000000	1.000000	26.000000	72.000000	21.000000	535.000000	543.000000	391.000000

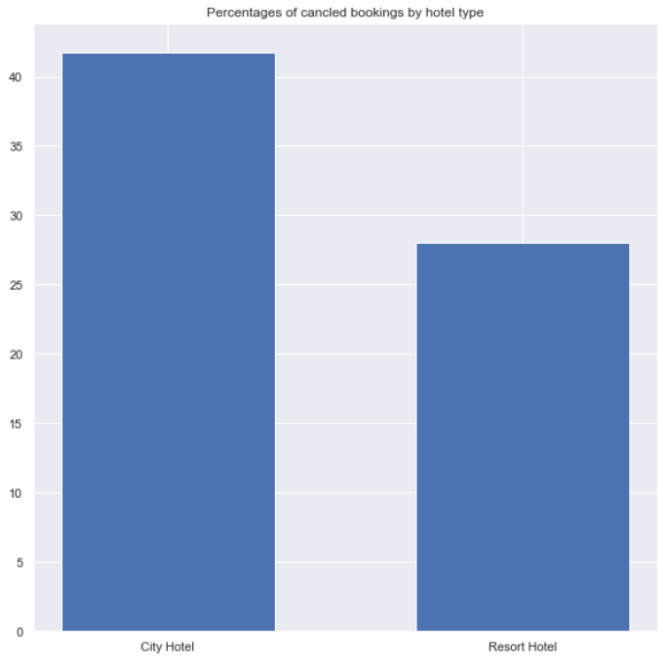
days_in_waiting_list	adr	required_car_parking_spaces	total_of_special_requests
118898.000000	118898.000000	118898.000000	118898.000000
2.330754	102.003243	0.061885	0.571683
17.630452	50.485862	0.244172	0.792678
0.000000	-6.380000	0.000000	0.000000
0.000000	70.000000	0.000000	0.000000
0.000000	95.000000	0.000000	0.000000
0.000000	126.000000	0.000000	1.000000
391.000000	5400.000000	8.000000	5.000000

3.2 The distribution of hotel type for cancellation :



This representation show the distribution of hotel type for cancellation. There is more reservation at cities Hotels than at Resort ones.





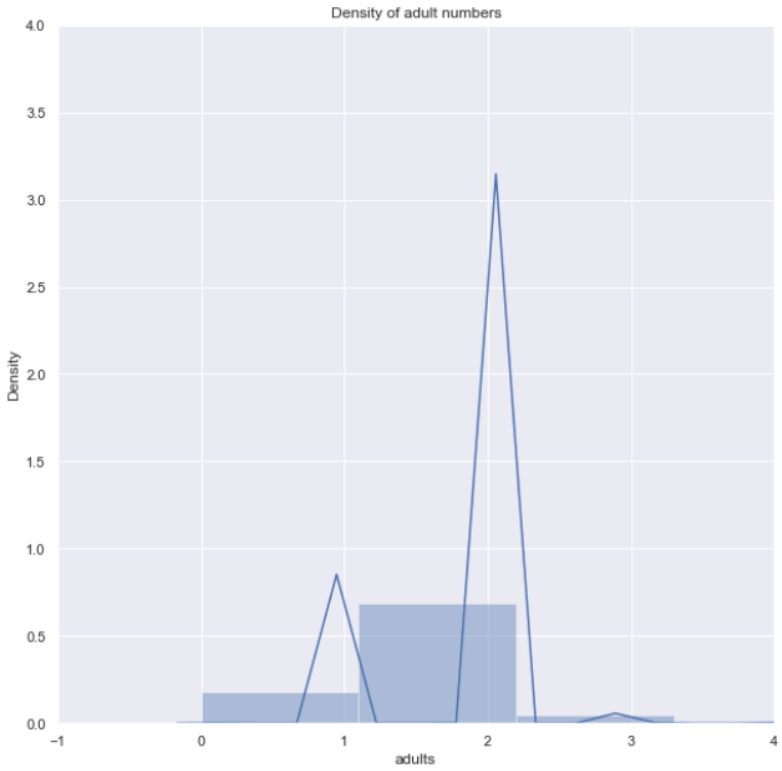
This representation show that there is more canceled booking for the Cities Hotels than it is for the Resort ones.

- 28% Canceled and 72% not for Resort Hotels
- 41% Canceled and 59% not for City Hotels

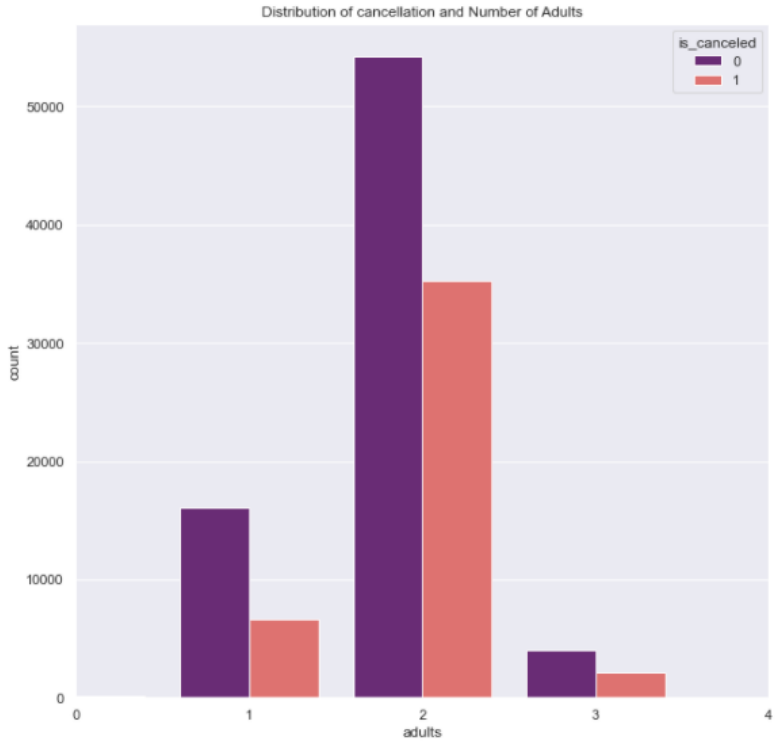
3.3 Distribution of Cancellation and Number of Adults:

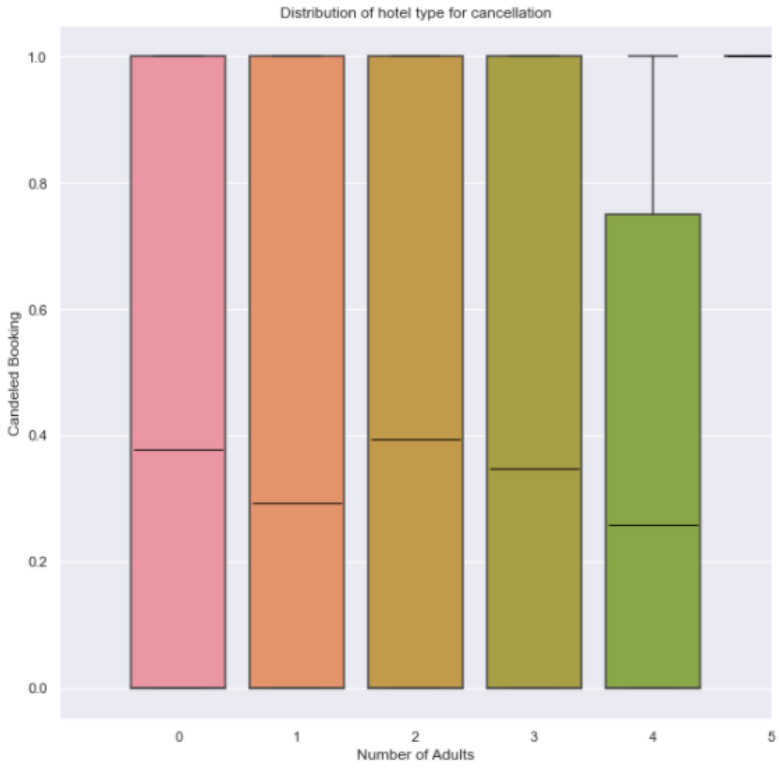
```
pd.DataFrame(hotels.groupby('adults').count()["hotel"])
```

adults	hotel
0	223
1	22735
2	89494
3	6197
4	62
5	2
6	1
10	1
20	2
26	5
27	2
40	1
50	1
55	1



As we can see, the density of adults numbers is mainly concentrated on 1, 2 and 3 adults. We will be focusing on plotting distribution of Cancellation and those Number of Adults.





Those 2 plots shows that for any number of adults the percentage of canceled bookings varies between 30% and 39%. We can therefore conclude that the number of adults doesn't influence the is_canceled variable too much.

3.4 Target Variables :

3.4.1 Meal:

```
hotels['meal'].unique()
```

```
array(['BB', 'FB', 'HB', 'SC'], dtype=object)
```

We can consider the target variable meal and see what the future customers are more likely to order, this is very important in the hotel industry as meal preparation takes a lot of time and money, and by predicting how much and when to cook food, that could save tons of money and time for the hotel restaurant

3.4.2 Repeated Guest:

```
hotels['is_repeated_guest'].unique()
```

```
array([0, 1], dtype=int64)
```

The target variable 'is_repeated_guest' looks very interesting, when a new customer visits the hotel and is about to leave (so that the data is gathered) we can predict if

he is coming back or not, if he's got a good chance in stopping by one more time the hotel marketing team can make special offers for those kind of customers, like: coupon reduction for their next visit, or give them a souvenir to leave a good impression...

This target variable can play huge in the marketing strategy of the hotel, and can readily generate a lot of cash.

Chapter 4

Modeling :

Based on the type of our data mining problem (binary classification) and the size of our DataSet we have chosed 2 classification algorithms to fit, test and score on our hotels data,to be able to chose the best one.

4.1 Prepare Data:

```
scaled_data.columns.values
```

```
array(['hotel', 'is_canceled', 'lead_time', 'arrival_date_week_number',
       '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', 'arrival_date'], dtype=object)
```

```
features=np.delete(scaled_data.columns, (1,3,16,27,28,29))
features
```

```
Index(['hotel', 'lead_time', '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', 'assigned_room_type',
       'booking_changes', 'deposit_type', 'agent', 'company',
       'days_in_waiting_list', 'customer_type', 'adr',
       'required_car_parking_spaces', 'total_of_special_requests'],
      dtype='object')
```

We have deleted some features that couldn't make sense, like:

- Arrival date: if he arrives we don't have to know if he cancels.

- Assigned room: he will be assigned a room when he arrives so it doesn't make sense to leave it.

```
target=scaled_data.columns[1]
target
```

```
'is_canceled'
```

```
X = scaled_data[features].values
y = scaled_data[target].values
```

We split our data into train and test:

```
from sklearn.model_selection import train_test_split
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, random_state=42)
print(X_trainset.shape, X_testset.shape, y_trainset.shape, y_testset.shape)
```

```
(89045, 24) (29682, 24) (89045,) (29682,)
```

4.2 Models :

We decided to use RandomForestClassifier and LogisticRegression to predicts if the booking is likely to be canceled.

4.2.1 RandomForestClassifier Algorithm :

```
from sklearn.ensemble import RandomForestClassifier
```

We have seen in the class the decision tree model, we decided to take it a step forward and work with a more performance and sophisticated algorithm called: Random Forrest Classifier. The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of the observations, splitting nodes in each tree considering a limited number of the features. The final predictions of the random forest are made by averaging the predictions of each individual tree.

4.2.2 LogisticRegression Algorithm :

```
from sklearn.linear_model import LogisticRegression
```

We have chosed logistic regression because it works so well in binary classification and with sizes of data such as the one we have.

4.2.3 Train Data and Compare Results:

```
models={"LogisticRegression":LogisticRegression(random_state=42,max_iter=4000),
        "RandomForestClassifier":RandomForestClassifier(n_estimators=100,
                                 bootstrap = True,
                                 max_features = 'sqrt')
        }

results={}
```

We put our models in our dictionary to speed up our coding using some for loops. We chosed some hyperparameters that we judged are good for our problem, later on we will do some hyper parameter tuning to make sure that we're making the right choices.

```
for model_name, model in models.items():
    model.fit(X_trainset, y_trainset)
    results[model_name]=model.score(X_testset, y_testset)
```

Training of both our models, and storing the score values in the results dictionary.

```
results_df=pd.DataFrame(data=results.values(),index=results.keys())
results_df['accuracy']=results_df[0]
results_df.drop([0],axis=1)
```

	accuracy
LogisticRegression	0.795458
RandomForestClassifier	0.889420

We can see that both the algorithms did a pretty good job, still the RandomForestClassifier is way better with over 88.5% correct answers.

4.3 Hyperparameter Tuning :

Often times, we don't immediately know what the optimal model architecture should be for a given model, and thus we'd like to be able to explore a range of possibilities. We'll ideally ask the machine to perform this exploration and select the optimal model architecture automatically. Parameters which define the model architecture are referred to as hyperparameters and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning.

4.3.1 Hyperparameter Tuning for Logistic Regression :

We will use RandomizedSearchCV to find the best hyperparameters for our logistic regression model:

```
from sklearn.model_selection import RandomizedSearchCV
```

```
log_reg_grid={"C": np.logspace(-4,4,20),
              "solver": ["liblinear"]}

rs_log_reg=RandomizedSearchCV(estimator=models["LogisticRegression"],
                              param_distributions=log_reg_grid,
                              cv=5,
                              n_iter=5,
                              verbose=0)

best_model = rs_log_reg.fit(X_trainset, y_trainset)
c=best_model.best_estimator_.get_params()['C']
print('Best C:', best_model.best_estimator_.get_params()['C'])
print('Best solver:', best_model.best_estimator_.get_params()['solver'])
```

```
Best C: 78.47599703514607
Best solver: liblinear
```

```
best_model.score(X_testset, y_testset)
```

```
0.7954920100925147
```

The new model with the new hyperparameters is doing better than the old one in terms of scoring he's doing approximately 0.002 better.

4.3.2 Hyperparameter Tuning for Random Forrest Classifier :

We will use GridSearchCV to find the best hyperparameters for our RandomForestClassifier model :

```
from sklearn.model_selection import GridSearchCV
```

```
n_estimators = [100, 300, 500]
max_depth = [ 15, 25, 30]
min_samples_split = [2, 5, 10, 15]
min_samples_leaf = [1, 2, 5]

hyperF = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split,
             min_samples_leaf = min_samples_leaf)

gridF = GridSearchCV(RandomForestClassifier(), hyperF, cv = 3, verbose = 1,
                     n_jobs = -1)
bestF = gridF.fit(X_trainset, y_trainset)
```

1. n_estimators: The n_estimators parameter specifies the number of trees in the forest of the model. The default value for this parameter is 10, which means that 10 different decision trees will be constructed in the random forest.

2. max_depth: The max_depth parameter specifies the maximum depth of each tree. The default value for max_depth is None, which means that each tree will expand until every leaf is pure. A pure leaf is one where all of the data on the leaf comes from the same class.

3. min_samples_split: The min_samples_split parameter specifies the minimum number of samples required to split an internal leaf node. The default value for this parameter is 2, which means that an internal node must have at least two samples before it can be split to have a more specific classification.

4. min_samples_leaf: The min_samples_leaf parameter specifies the minimum number of samples required to be at a leaf node. The default value for this parameter is 1, which means that every leaf must have at least 1 sample that it classifies.

```
gridF.best_params_
```

```
{'max_depth': 30,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 300}
```

```
bestF.score(X_testset, y_testset)
```

```
0.9610538373424972
```

The new random forrest classifier is doing extremly well with over 96.1% in accuracy scoring, that is a great exemple of showing the importance of hyperparameter tuning.

Chapter 5

Model Evaluation :

Untill now, we have only used the score() function to evaluate our models, but when dealing with classification it's better to use some other metrics to get better insights.

- Confusion matrix - Compares the predicted values with the true values in a tabular way, if 100% correct, all values in the matrix will be top left to bottom right (diagonal line).
- Cross-validation - Splits your DataSet into multiple parts and train and tests your model on each part and evaluates performance as an average.
- Precision - Proportion of true positives over total number of samples. Higher precision leads to less false positives.
- Recall - Proportion of true positives over total number of true positives and false positives. Higher recall leads to less false negatives.
- F1 score - Combines precision and recall into one metric. 1 is best, 0 is worst.
- Classification report - Sklearn has a built-in function called classification_report() which returns some of the main classification metrics such as precision, recall and f1-score.
- ROC Curve - Receiver Operating Characteristic is a plot of true positive rate versus false positive rate.
- Area Under Curve (AUC) - The area underneath the ROC curve. A perfect model achieves a score of 1.0.

5.1 Prepare the tools :

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import plot_roc_curve
```

We will use our best hyperparameters to create new models:

```
clf=LogisticRegression(C=c,solver='liblinear')
forrest=RandomForestClassifier(max_depth = 30,
                               n_estimators = 300,
                               min_samples_split = 2,
                               min_samples_leaf = 1)
```

```
clf.fit(X_trainset, y_trainset)
forrest.fit(X_trainset, y_trainset)
```

```
RandomForestClassifier(max_depth=30, n_estimators=300)
```

```
y_preds_regression=clf.predict(X_testset)
y_preds_rfc=forrest.predict(X_testset)
```

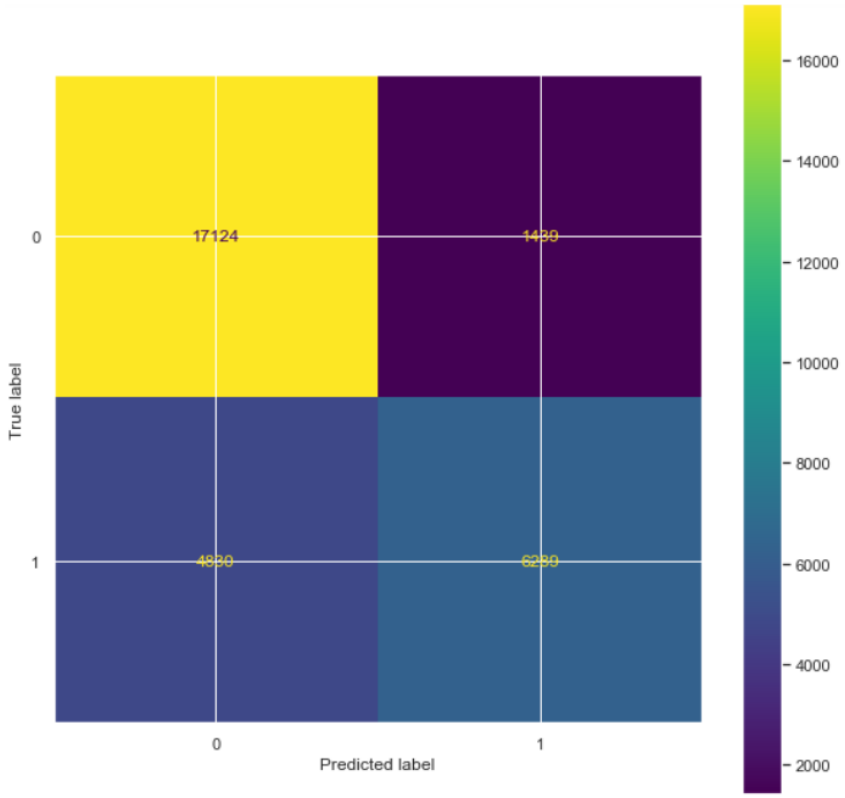
5.2 Confusion Matrix :

5.2.1 Regression Model :

```
conf_mat_regression=confusion_matrix(y_testset, y_preds_regression)
conf_mat_regression
```

```
array([[17266,  1413],
       [ 4728,  6275]], dtype=int64)
```

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf,X_testset, y_testset)
plt.show()
```

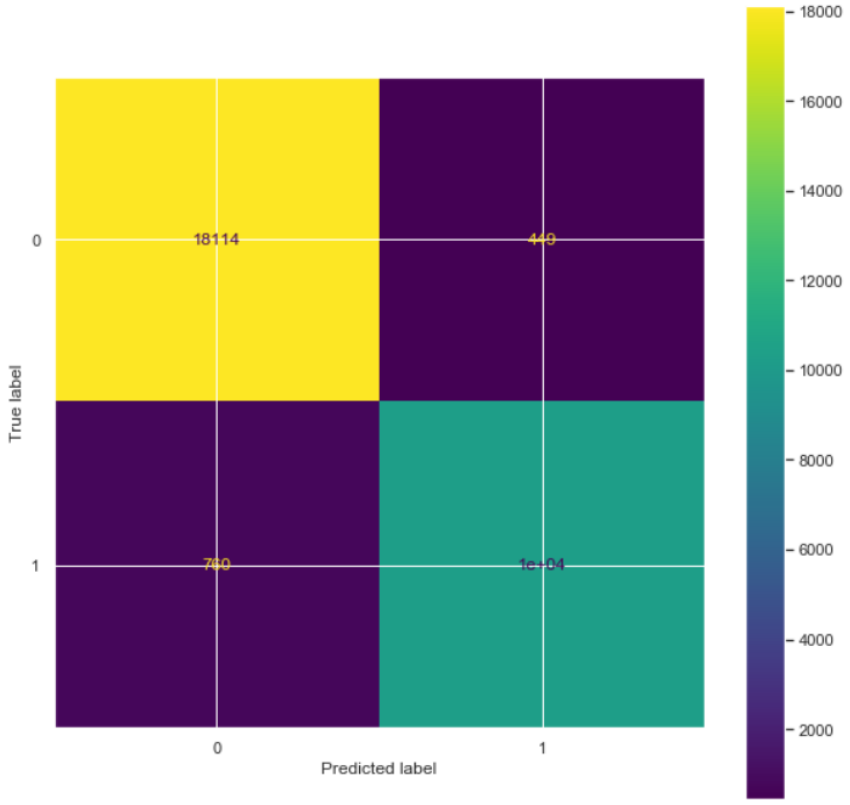


5.2.2 Random Forrest Classifier Model :

```
conf_mat_rfc=confusion_matrix(y_testset, y_preds_rfc)
conf_mat_rfc
```

```
array([[17403,  1276],
       [ 2066,  8937]], dtype=int64)
```

```
plot_confusion_matrix(forrest,X_testset, y_testset)
plt.show()
```



5.2.3 Conclusion :

Both of the algorithms did a better job predicting if the reservation was not going to be canceled '0', than if it will be canceled.(the number of true positives is higher than the number f true negatives) which is good because the hotels will rarely have to deal with problems with customers, this way the hotels will earn more money without risking their reputation.

5.3 Classification Report :

Classification_report() is an sklearn built-in function which returns some of the main classification metrics such as precision, recall and f1-score.

5.3.1 Regression Model :

```
cr_regression=classification_report(y_testset, y_preds_regression)
print('classification report for regression')
print(cr_regression)
```

```
classification report for regression
              precision    recall  f1-score   support

           0       0.78      0.92      0.85     18563
           1       0.82      0.56      0.67     11119

    accuracy                           0.79     29682
   macro avg       0.80      0.74      0.76     29682
weighted avg       0.79      0.79      0.78     29682
```

5.3.2 Random Forrest Classifier Model :

```
cr_rfc=classification_report(y_testset, y_preds_rfc)
print('classification report for random forrest classifier')
print(cr_rfc)
```

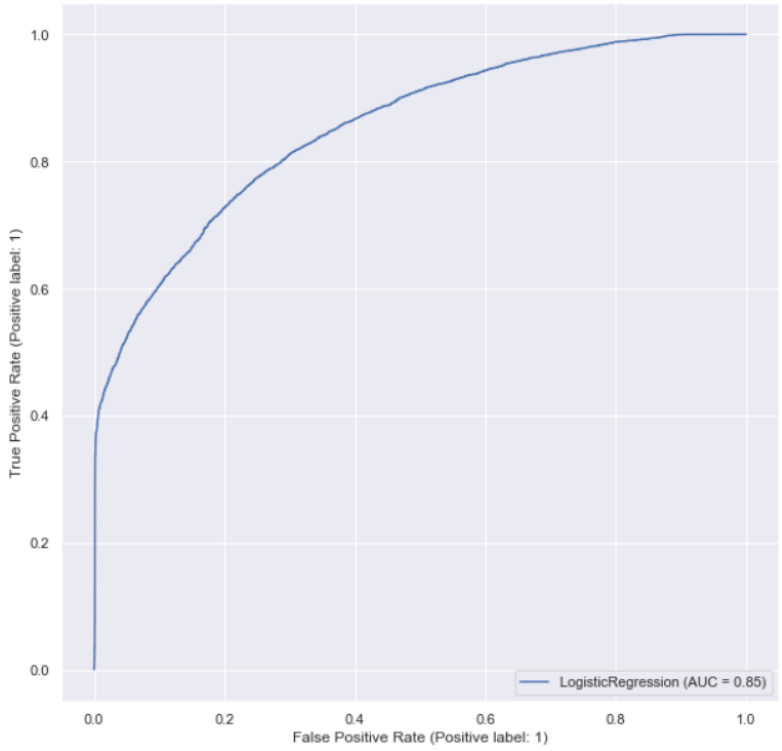
```
classification report for random forrest classifier
              precision    recall  f1-score   support

           0       0.89      0.93      0.91     18679
           1       0.88      0.81      0.84     11003

    accuracy                           0.89     29682
   macro avg       0.88      0.87      0.88     29682
weighted avg       0.89      0.89      0.89     29682
```

5.3.3 Conclusion :

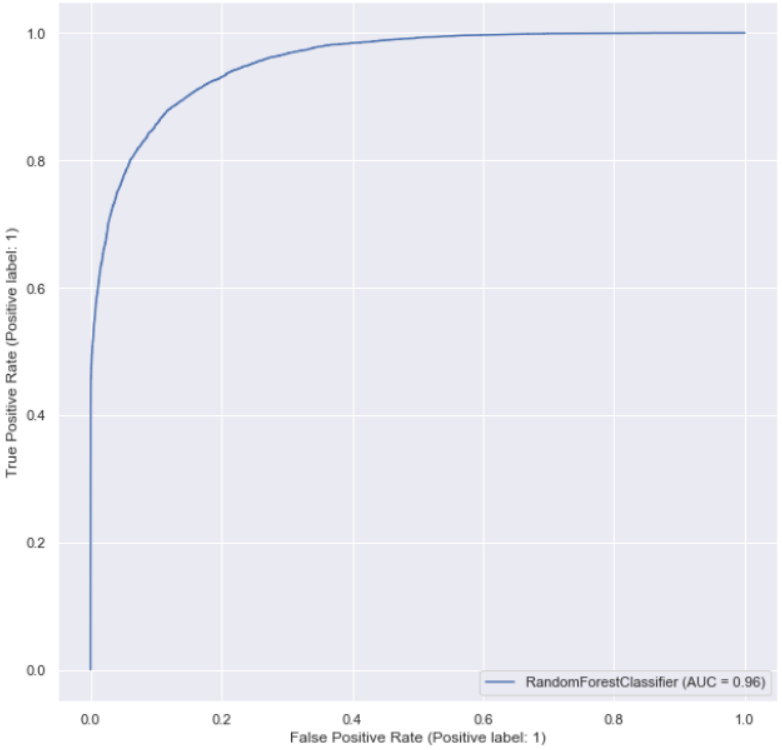
Clearly the Random Forrest Classifier takes the lead here and is better performing according to all metrics.

5.4 ROC CURVE :

5.4.1 Regression Model :



5.4.2 Random Forrest Classifier Model :



5.4.3 Conclusion :

The area under ROC curve confirms our evaluation of the models, at any threshold the true positives of the Random Forrest Classifier are higher than the true positives of the Logistic Regression Classifier.

5.5 Using different metrics with cross validation :

So far we have evaluated our model with only one training set and one testing set, a more sophisticated way of doing the evaluation is by using the cross validation.

```
from sklearn.model_selection import cross_val_score
```

```
from sklearn.metrics import make_scorer
```

5.5.1 Regression Model :

```
cross_val_score(clf, X, y,scoring="accuracy",cv=5)
```

```
array([0.68352565, 0.71877369, 0.78159613, 0.76218151, 0.8068646 ])
```

```
cross_val_acc1=np.mean(cross_val_score(clf, X, y,scoring="accuracy",cv=5))
cross_val_acc1
```

```
0.7505883150192914
```

```
ps_regression=np.mean(cross_val_score(clf,X,y, scoring = make_scorer(precision_score), cv = 5 ))
rs_regression=np.mean(cross_val_score(clf,X,y, scoring = make_scorer(recall_score), cv = 5 ))
f1_regression=np.mean(cross_val_score(clf,X,y, scoring = make_scorer(f1_score), cv = 5 ))
```

5.5.2 Random Forrest Classifier Model :

```
cross_val_score(forrest, X, y,scoring="accuracy",cv=5)
```

```
array([0.6932536 , 0.75557989, 0.75556959, 0.80374816, 0.77810065])
```

```
cross_val_acc2=np.mean(cross_val_score(forrest, X, y,scoring="accuracy",cv=5))
cross_val_acc2
```

```
0.756938751502464
```

```
ps_rfc=np.mean(cross_val_score(forrest,X,y, scoring = make_scorer(precision_score), cv = 5 ))
rs_rfc=np.mean(cross_val_score(forrest,X,y, scoring = make_scorer(recall_score), cv = 5 ))
f1_rfc=np.mean(cross_val_score(forrest,X,y, scoring = make_scorer(f1_score), cv = 5 ))
```

5.5.3 Conclusion :

```
results_cross_val_score=pd.DataFrame(data=[[ps_regression,rs_regression,
                                            f1_regression],[ps_rfc,rs_rfc,f1_rfc]],
                                     columns=['precision_score','recall_score','f1_score'])

results_cross_val_score
```

	precision_score	recall_score	f1_score
0	0.758156	0.520800	0.582065
1	0.740992	0.589927	0.618324

Conclusion :

In one notebook we solved a real life hotel industry problem, and helped saving a lot of money, time and energy, which shows how easily ML is revolutionizing everything. Machine learning and AI are truly the new electricity.