Problem Statement: Hotel Reservation Prediction

To predict if Customer is going to cancel the reservation or not

Importing the Libraries

In [93]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier,VotingClassifier,RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB,BernoulliNB,GaussianNB
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report,roc_auc_score,roc_curve
from sklearn.linear_model import Ridge,Lasso
```

importing data

In [2]:
```
df=pd.read_csv('Hotel Reservations.csv')
```

In [3]:
```
df.head() #reading top 5 data
```

Out[3]:

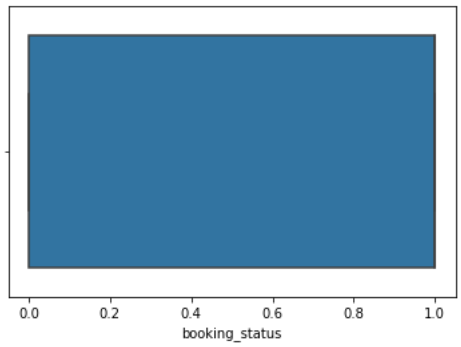
	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	le
0	INN00001	2	0	1	2	Meal Plan 1	0	Room_Type 1	
1	INN00002	2	0	2	3	Not Selected	0	Room_Type 1	
2	INN00003	1	0	2	1	Meal Plan 1	0	Room_Type 1	
3	INN00004	2	0	0	2	Meal Plan 1	0	Room_Type 1	
4	INN00005	2	0	1	1	Not Selected	0	Room_Type 1	

Exploratory Data Analysis

In [4]:
```
#1.Checking information about data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 19 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Booking_ID                            36275 non-null  object
 1   no_of_adults                          36275 non-null  int64
 2   no_of_children                        36275 non-null  int64
 3   no_of_weekend_nights                  36275 non-null  int64
 4   no_of_week_nights                     36275 non-null  int64
 5   type_of_meal_plan                     36275 non-null  object
 6   required_car_parking_space            36275 non-null  int64
 7   room_type_reserved                    36275 non-null  object
 8   lead_time                             36275 non-null  int64
 9   arrival_year                          36275 non-null  int64
 10  arrival_month                         36275 non-null  int64
 11  arrival_date                          36275 non-null  int64
 12  market_segment_type                   36275 non-null  object
 13  repeated_guest                        36275 non-null  int64
 14  no_of_previous_cancellations          36275 non-null  int64
 15  no_of_previous_bookings_not_canceled  36275 non-null  int64
 16  avg_price_per_room                    36275 non-null  float64
 17  no_of_special_requests                36275 non-null  int64
 18  booking_status                        36275 non-null  object
dtypes: float64(1), int64(13), object(5)
memory usage: 5.3+ MB
```

In [115]:
```
#2 Checking outliers
sns.boxplot(df.booking_status)
plt.show()
```



In [79]:
```
#3 checking null values
df.isnull().sum()
```

Out[79]:
```
no_of_adults                          0
no_of_children                        0
no_of_weekend_nights                  0
no_of_week_nights                     0
type_of_meal_plan                     0
required_car_parking_space            0
room_type_reserved                    0
lead_time                             0
arrival_year                          0
arrival_month                         0
arrival_date                          0
market_segment_type                   0
repeated_guest                        0
no_of_previous_cancellations          0
no_of_previous_bookings_not_canceled  0
avg_price_per_room                    0
no_of_special_requests                0
booking_status                        0
dtype: int64
```

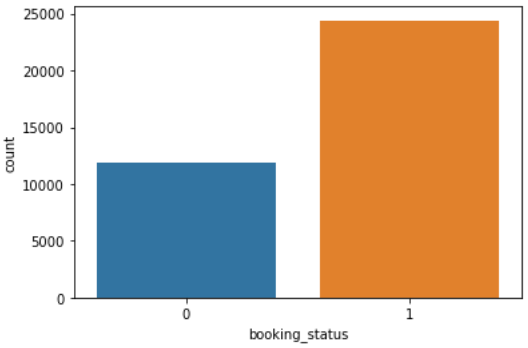
In [5]:
```
#4.Label encoding to convert object into int
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [6]:
```
df['type_of_meal_plan']=le.fit_transform(df['type_of_meal_plan'])
df['room_type_reserved']=le.fit_transform(df['room_type_reserved'])
df['market_segment_type']=le.fit_transform(df['market_segment_type'])
df['booking_status']=le.fit_transform(df['booking_status'])
```

In [7]:
```
#5 Dropping Column Booking ID as it is unique and not required for prediction
df.drop(columns='Booking_ID',inplace=True)
```

In [8]:
```
#6 Counting values of Booking status
sns.countplot(x='booking_status', data=df)
```

Out[8]:
```
<AxesSubplot:xlabel='booking_status', ylabel='count'>
```



We can see that Data is imbalance

In [9]:
```
df['booking_status'].value_counts()
```

Out[9]:
```
1    24390
0    11885
Name: booking_status, dtype: int64
```

Descriptive data

In [10]:
```
df.describe()
```

Out[10]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	lead_ti
count	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000(
mean	1.844962	0.105279	0.810724	2.204300	0.515644	0.030986	0.708890	85.232
std	0.518715	0.402648	0.870644	1.410905	1.048131	0.173281	1.399851	85.930
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000(
25%	2.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	17.000(
50%	2.000000	0.000000	1.000000	2.000000	0.000000	0.000000	0.000000	57.000(
75%	2.000000	0.000000	2.000000	3.000000	0.000000	0.000000	0.000000	126.000(
max	4.000000	10.000000	7.000000	17.000000	3.000000	1.000000	6.000000	443.000(

In [11]:
```
df.corr()
```

Out[11]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space
no_of_adults	1.000000	-0.019787	0.103316	0.105622	0.025555	0.011429
no_of_children	-0.019787	1.000000	0.029478	0.024398	-0.086764	0.034244
no_of_weekend_nights	0.103316	0.029478	1.000000	0.179577	-0.027327	-0.031111
no_of_week_nights	0.105622	0.024398	0.179577	1.000000	-0.083431	-0.048784
type_of_meal_plan	0.025555	-0.086764	-0.027327	-0.083431	1.000000	-0.012991
required_car_parking_space	0.011429	0.034244	-0.031111	-0.048784	-0.012991	1.000000
room_type_reserved	0.270348	0.364073	0.057368	0.094125	-0.209176	0.038778
lead_time	0.097287	-0.047091	0.046595	0.149650	-0.060271	-0.066445
arrival_year	0.076719	0.045983	0.055357	0.032672	0.071396	0.015684
arrival_month	0.021841	-0.003076	-0.009894	0.037376	0.008564	-0.015505
arrival_date	0.026338	0.025482	0.027304	-0.009305	0.004833	-0.000037
market_segment_type	0.314103	0.130618	0.129069	0.112952	0.203361	-0.003723
repeated_guest	-0.192277	-0.036348	-0.067107	-0.099764	-0.062995	0.110909
no_of_previous_cancellations	-0.047426	-0.016390	-0.020690	-0.030080	-0.011622	0.027106
no_of_previous_bookings_not_canceled	-0.119166	-0.021189	-0.026312	-0.049344	-0.038183	0.063810
avg_price_per_room	0.296886	0.337728	-0.004525	0.022753	-0.069257	0.061304
no_of_special_requests	0.189401	0.124486	0.060593	0.045994	0.022091	0.087922
booking_status	-0.086920	-0.033078	-0.061563	-0.092996	-0.026706	0.086185

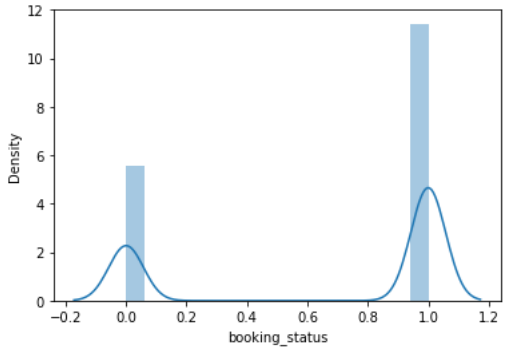
In [12]:
```
df.head()
```

Out[12]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	lead_time	arri
0	2	0	1	2	0	0	0	224	
1	2	0	2	3	3	0	0	5	
2	1	0	2	1	0	0	0	1	
3	2	0	0	2	0	0	0	211	
4	2	0	1	1	3	0	0	48	

In [13]:
```
sns.distplot(df['booking_status']) #shows distribution of status
```

Out[13]:
```
<AxesSubplot:xlabel='booking_status', ylabel='Density'>
```



Separating the data into x and y

In [14]:
```
x=df.iloc[:,:-1]
x
```

Out[14]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	lead_time
0	2	0	1	2	0	0	0	224
1	2	0	2	3	3	0	0	5
2	1	0	2	1	0	0	0	1
3	2	0	0	2	0	0	0	211
4	2	0	1	1	3	0	0	48
...	...	...	...	...	...	...	...	...
36270	3	0	2	6	0	0	3	85
36271	2	0	1	3	0	0	0	228
36272	2	0	2	6	0	0	0	148
36273	2	0	0	3	3	0	0	63
36274	2	0	1	2	0	0	0	207

36275 rows × 17 columns

In [15]:
```
y=df['booking_status']
y
```

Out[15]:
```
0        1
1        1
2        0
3        0
4        0
        ..
36270    1
36271    0
36272    1
36273    0
36274    1
Name: booking_status, Length: 36275, dtype: int32
```

In [16]:
```
# Splitting data in training and testing
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=1)
```

Function to create object and implement classification algorithm

In [17]:
```
def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)
    print(classification_report(ytest,ypred))
    return model
```

In [18]:
```
lr=LogisticRegression()
dtc=DecisionTreeClassifier()
rf=RandomForestClassifier()
svm=SVC()
knn=KNeighborsClassifier(n_neighbors=5)
mnb=MultinomialNB()
```

Classification report of all algorithm without hypertuning

In [19]:
```
mymodel(lr)
```

```
              precision    recall  f1-score   support

           0       0.74      0.58      0.65      2958
           1       0.81      0.90      0.85      6111

    accuracy                           0.79      9069
   macro avg       0.78      0.74      0.75      9069
weighted avg       0.79      0.79      0.79      9069
```

Out[19]:
```
LogisticRegression()
```

In [20]:
```
mymodel(dtc)
```

```
              precision    recall  f1-score   support

           0       0.79      0.80      0.80      2958
           1       0.90      0.90      0.90      6111

    accuracy                           0.87      9069
   macro avg       0.85      0.85      0.85      9069
weighted avg       0.87      0.87      0.87      9069
```

Out[20]:
```
DecisionTreeClassifier()
```

In [21]:
```
mymodel(rf)
```

```
              precision    recall  f1-score   support

           0       0.88      0.81      0.85      2958
           1       0.91      0.95      0.93      6111

    accuracy                           0.90      9069
   macro avg       0.90      0.88      0.89      9069
weighted avg       0.90      0.90      0.90      9069
```

Out[21]:
```
RandomForestClassifier()
```

In [22]:
```
mymodel(svm)
```

```
              precision    recall  f1-score   support

           0       0.76      0.42      0.54      2958
           1       0.77      0.94      0.84      6111

    accuracy                           0.77      9069
   macro avg       0.76      0.68      0.69      9069
weighted avg       0.76      0.77      0.74      9069
```

Out[22]:
```
SVC()
```

In [23]:
```
mymodel(knn)
```

```
              precision    recall  f1-score   support

           0       0.74      0.63      0.68      2958
           1       0.83      0.89      0.86      6111

    accuracy                           0.81      9069
   macro avg       0.79      0.76      0.77      9069
weighted avg       0.80      0.81      0.80      9069
```

Out[23]:
```
KNeighborsClassifier()
```

In [24]:
```
mymodel(mnb)
```

```
              precision    recall  f1-score   support

           0       0.58      0.61      0.60      2958
           1       0.81      0.78      0.80      6111

    accuracy                           0.73      9069
   macro avg       0.69      0.70      0.70      9069
weighted avg       0.73      0.73      0.73      9069
```

Out[24]:
```
MultinomialNB()
```

Without hypertuning it can be found that Random forest has given highest accuracy

Calculating training and testing accuracy for different algorithms

In [25]:
```
print("Train Accuracy logistic:",lr.score(xtrain, ytrain))
print("Test Accuracy logistic:",lr.score(xtest, ytest))
print("Train Accuracy decisiontree:",dtc.score(xtrain, ytrain))
print("Test Accuracy decisiontree:",dtc.score(xtest, ytest))
print("Train Accuracy randomforest:",rf.score(xtrain, ytrain))
print("Test Accuracy randomforest:",rf.score(xtest, ytest))
print("Train Accuracy supportvector:",svm.score(xtrain, ytrain))
print("Test Accuracy supportvector:",svm.score(xtest, ytest))
print("Train Accuracy kneighbors:",knn.score(xtrain, ytrain))
print("Test Accuracy kneighbors:",knn.score(xtest, ytest))
print("Train Accuracy naivebayes:",mnb.score(xtrain, ytrain))
print("Test Accuracy naivebayes:",mnb.score(xtest, ytest))
```

```
Train Accuracy logistic: 0.79056090568257
Test Accuracy logistic: 0.7943543940897563
Train Accuracy decisiontree: 0.9939719179592736
Test Accuracy decisiontree: 0.8683427059212703
Train Accuracy randomforest: 0.9939719179592736
Test Accuracy randomforest: 0.9043996030433344
Train Accuracy supportvector: 0.7627361611409248
Test Accuracy supportvector: 0.765905833057669
Train Accuracy kneighbors: 0.86267735058443
Test Accuracy kneighbors: 0.8062630940566766
Train Accuracy naivebayes: 0.7248768653973389
Test Accuracy naivebayes: 0.7291873414929981
```

In training and test score we can see that DecisionTree,RandomForest,Kneighbors have high training accuracy compared to testing accuracy which result in overfit to maintain we can use hypertuning techniques

In [26]:
```
#1 Decision Tree
for i in range (1,50):
    dtc1=DecisionTreeClassifier(max_depth=i)
    dtc1.fit(xtrain,ytrain)
    ypred=dtc1.predict(xtest)
    print(f'{i}={accuracy_score(ytest,ypred)}')
```

```
1=0.7632594552872423
2=0.7632594552872423
3=0.793031205204543
4=0.8207079060535891
5=0.8348219208291984
6=0.8423199911787408
7=0.8479435439408975
8=0.8599625096482523
9=0.869996692027787
10=0.8740765244238615
11=0.8792590142242805
12=0.8782666225603705
13=0.8789282170029772
14=0.8801411401477561
15=0.8778255595986327
16=0.8750689160877715
17=0.877163965156026
18=0.8731943985003859
19=0.8704377549895248
```

In [27]:
```
dtc1=DecisionTreeClassifier(max_depth=8)
mymodel(dtc1)
```

```
              precision    recall  f1-score   support

           0       0.80      0.76      0.78      2958
           1       0.89      0.91      0.90      6111

    accuracy                           0.86      9069
   macro avg       0.84      0.84      0.84      9069
weighted avg       0.86      0.86      0.86      9069
```

Out[27]:
```
DecisionTreeClassifier(max_depth=8)
```

In [28]:
```
print("Train Accuracy decisiontree:",dtc1.score(xtrain, ytrain))
print("Test Accuracy decisiontree:",dtc1.score(xtest, ytest))
```

```
Train Accuracy decisiontree: 0.866647063147835
Test Accuracy decisiontree: 0.8598522439078179
```

In [29]:
```
for i in range (1,75):
    dtc1=DecisionTreeClassifier(min_samples_leaf=i)
    dtc1.fit(xtrain,ytrain)
    ypred=dtc1.predict(xtest)
    print(f'{i}={accuracy_score(ytest,ypred)}')
```

```
1=0.8685632374021391
2=0.8612856985334657
3=0.867791377219098
4=0.8630499503804168
5=0.8665784540743191
6=0.8647039364869336
7=0.8669092512956225
8=0.8676811114786636
9=0.8661373911125814
10=0.866798985555188
11=0.8695556290660492
12=0.8698864262873525
13=0.8685632374021391
14=0.8681221744404014
15=0.866798985555188
16=0.8674605799977947
17=0.8659168596317124
18=0.8648142022273679
19=0.8647039364869336
20=0.8630499503804168
21=0.8630499503804168
22=0.8632704818612857
23=0.863601279082589
24=0.8620575587165068
25=0.8621678244569412
26=0.8629396846399824
27=0.8628294188995479
28=0.8644834050060646
29=0.8647039364869336
30=0.8641526077847613
31=0.8661373911125814
32=0.8644834050060646
33=0.8650347337082368
34=0.8634910133421546
35=0.8631602161208513
36=0.8645936707464991
37=0.8631602161208513
38=0.8618370272356379
39=0.8610651670525967
40=0.8598522439078179
41=0.858639320763039
42=0.8578674605799977
43=0.8584187892821701
44=0.8584187892821701
45=0.8583085235417356
46=0.8577571948395634
47=0.8580879920608667
48=0.8564340059543499
49=0.8565442716947844
50=0.8557724115117433
51=0.8559929429926122
52=0.8559929429926122
53=0.8563237402139156
54=0.8555518800308745
55=0.8567648031756533
56=0.8567648031756533
57=0.8573161318778255
58=0.8583085235417356
59=0.8583085235417356
60=0.8548902855882677
61=0.8548902855882677
62=0.8548902855882677
63=0.8533465652221854
64=0.8531260337413166
65=0.8523541735582755
66=0.852243907817841
67=0.8534568309626199
68=0.8534568309626199
69=0.8536773624434888
70=0.8543389568860955
71=0.8538978939243577
72=0.8530157680008821
73=0.8515823133752343
74=0.8515823133752343
```

In [60]:
```
dtc1=DecisionTreeClassifier(max_depth=8,min_samples_leaf=60)
mymodel(dtc1)
```

```
              precision    recall  f1-score   support

           0       0.80      0.72      0.76      2958
           1       0.87      0.91      0.89      6111

    accuracy                           0.85      9069
   macro avg       0.83      0.82      0.83      9069
weighted avg       0.85      0.85      0.85      9069
```

Out[60]:
```
DecisionTreeClassifier(max_depth=8, min_samples_leaf=60)
```

In [61]:
```
#Final Decision Tree Model
print("Train Accuracy decisiontree:",dtc1.score(xtrain, ytrain))
print("Test Accuracy decisiontree:",dtc1.score(xtest, ytest))
```

```
Train Accuracy decisiontree: 0.8536719841211498
Test Accuracy decisiontree: 0.8500385930091521
```

After tuning with depth, leaf and criterion the overfitting of the model has been reduced and proper accuracy is also achieved

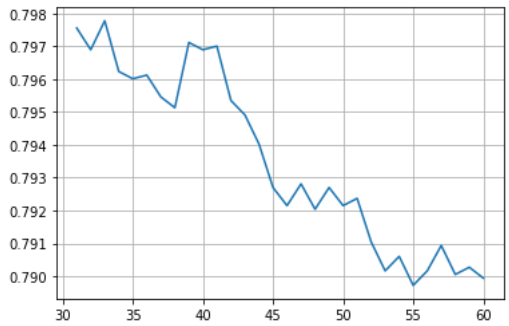
In [74]:
```
#2 KNeighborsClassifier
ac_list=[]
for i in range(1,31):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(xtrain,ytrain)
    ypred=knn.predict(xtest)
    ac=accuracy_score(ytest,ypred)
    ac_list.append(ac)
```

In [75]:
```
ac_list=[]
for i in range(31,61):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(xtrain,ytrain)
    ypred=knn.predict(xtest)
    ac=accuracy_score(ytest,ypred)
    ac_list.append(ac)
```

In [37]:
```
ac_list
```

. . .

In [39]:
```
plt.plot(range(31,61),ac_list)
plt.grid(True)
plt.show()
```



In [136]:
```
#K-Elbow method used to find value of k
knn=KNeighborsClassifier(n_neighbors=41)
```

In [129]:
```
mymodel(knn)
```

```
              precision    recall  f1-score   support

           0       0.80      0.50      0.62      2958
           1       0.80      0.94      0.86      6111

    accuracy                           0.80      9069
   macro avg       0.80      0.72      0.74      9069
weighted avg       0.80      0.80      0.78      9069
```

Out[129]:
```
KNeighborsClassifier(n_neighbors=41)
```

In [130]:
```
print("Train Accuracy kneighbors:",knn.score(xtrain, ytrain))
print("Test Accuracy kneighbors:",knn.score(xtest, ytest))
```

```
Train Accuracy kneighbors: 0.7978019554510034
Test Accuracy kneighbors: 0.797000771860183
```

In [132]:
```
#Final KNN model
knn=KNeighborsClassifier(n_neighbors=41)
```

In [135]:
```
mymodel(knn)
```

```
              precision    recall  f1-score   support

           0       0.80      0.50      0.62      2958
           1       0.80      0.94      0.86      6111

    accuracy                           0.80      9069
   macro avg       0.80      0.72      0.74      9069
weighted avg       0.80      0.80      0.78      9069
```

Out[135]:
```
KNeighborsClassifier(n_neighbors=41)
```

After hypertuning it can be observed that training and testing score is almost same for knn model and accuracy is 80 % and also helps to reduce overfitting of the model

In [88]:
```
lr=LogisticRegression(solver='newton-cg')
mymodel(lr)
```

```
              precision    recall  f1-score   support

           0       0.75      0.61      0.68      2958
           1       0.83      0.90      0.86      6111

    accuracy                           0.81      9069
   macro avg       0.79      0.76      0.77      9069
weighted avg       0.80      0.81      0.80      9069
```

Out[88]:
```
LogisticRegression(solver='newton-cg')
```

The use of hypertuning in Logistic was to increase accuracy from 79 to 81%

In [137]:
```
# Importing Boosting algorithm
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
```

In [82]:
```
ab=AdaBoostClassifier()
ab.fit(xtrain,ytrain)
ypred=ab.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.68      0.71      2958
           1       0.85      0.89      0.87      6111

    accuracy                           0.82      9069
   macro avg       0.79      0.78      0.79      9069
weighted avg       0.81      0.82      0.81      9069
```

In [83]:
```
gb=GradientBoostingClassifier()
gb.fit(xtrain,ytrain)
ypred=gb.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.71      0.76      2958
           1       0.87      0.93      0.90      6111

    accuracy                           0.86      9069
   macro avg       0.85      0.82      0.83      9069
weighted avg       0.85      0.86      0.85      9069
```

In [85]:
```
from xgboost import XGBClassifier
```

In [86]:
```
xb=XGBClassifier()
xb.fit(xtrain,ytrain)
ypred=xb.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.81      0.83      2958
           1       0.91      0.94      0.92      6111

    accuracy                           0.90      9069
   macro avg       0.89      0.87      0.88      9069
weighted avg       0.89      0.90      0.89      9069
```

XGBoost gives highest accuracy of around 90 % along with recall and precision

In [91]:
```
mnb1=BernoulliNB()
```

In [92]:
```
mymodel(mnb1)
```

```
              precision    recall  f1-score   support

           0       0.52      0.20      0.29      2958
           1       0.70      0.91      0.79      6111

    accuracy                           0.68      9069
   macro avg       0.61      0.55      0.54      9069
weighted avg       0.64      0.68      0.63      9069
```

Out[92]:
```
BernoulliNB()
```

In [94]:
```
mnb2=GaussianNB()
```

In [95]:
```
mymodel(mnb2)
```

```
              precision    recall  f1-score   support

           0       0.36      0.98      0.53      2958
           1       0.94      0.15      0.26      6111

    accuracy                           0.42      9069
   macro avg       0.65      0.57      0.40      9069
weighted avg       0.75      0.42      0.35      9069
```

Out[95]:
```
GaussianNB()
```

IN Naivebayes algorithm after using different method for finding accuracy highest accuracy was found for MultinomialNB

In [96]:
```
svm1=SVC(kernel='linear')
```

In [97]:
```
mymodel(svm1)
```

```
              precision    recall  f1-score   support

           0       0.73      0.63      0.68      2958
           1       0.83      0.88      0.86      6111

    accuracy                           0.80      9069
   macro avg       0.78      0.76      0.77      9069
weighted avg       0.80      0.80      0.80      9069
```

Out[97]:
```
SVC(kernel='linear')
```

In [98]:
```
print("Train Accuracy kneighbors:",svm1.score(xtrain, ytrain))
print("Test Accuracy kneighbors:",svm1.score(xtest, ytest))
```

```
Train Accuracy kneighbors: 0.7996397853414688
Test Accuracy kneighbors: 0.8024037931414709
```

In [103]:
```
svm2=SVC(kernel='poly')
mymodel(svm2)
```

```
              precision    recall  f1-score   support

           0       0.76      0.42      0.54      2958
           1       0.77      0.94      0.84      6111

    accuracy                           0.77      9069
   macro avg       0.76      0.68      0.69      9069
weighted avg       0.77      0.77      0.74      9069
```

Out[103]:
```
SVC(kernel='poly')
```

In [102]:
```
print("Train Accuracy kneighbors:",svm2.score(xtrain, ytrain))
print("Test Accuracy kneighbors:",svm2.score(xtest, ytest))
```

```
Train Accuracy kneighbors: 0.7625523781518783
Test Accuracy kneighbors: 0.766898224721579
```

In [106]:
```
svm3=SVC(kernel='sigmoid')
mymodel(svm3)
```

```
              precision    recall  f1-score   support

           0       0.76      0.40      0.52      2958
           1       0.76      0.94      0.84      6111

    accuracy                           0.76      9069
   macro avg       0.76      0.67      0.68      9069
weighted avg       0.76      0.76      0.74      9069
```

Out[106]:
```
SVC(kernel='sigmoid')
```

In [107]:
```
print("Train Accuracy kneighbors:",svm3.score(xtrain, ytrain))
print("Test Accuracy kneighbors:",svm3.score(xtest, ytest))
```

```
Train Accuracy kneighbors: 0.7586561787840918
Test Accuracy kneighbors: 0.7634799867681111
```

In [108]:
```
svm4=SVC(kernel='rbf')
mymodel(svm4)
```

```
              precision    recall  f1-score   support

           0       0.76      0.42      0.54      2958
           1       0.77      0.94      0.84      6111

    accuracy                           0.77      9069
   macro avg       0.76      0.68      0.69      9069
weighted avg       0.76      0.77      0.74      9069
```

Out[108]:
```
SVC()
```

In [109]:
```
print("Train Accuracy kneighbors:",svm4.score(xtrain, ytrain))
print("Test Accuracy kneighbors:",svm4.score(xtest, ytest))
```

```
Train Accuracy kneighbors: 0.7627361611409248
Test Accuracy kneighbors: 0.765905833057669
```

SVM is difficult to run and interpret result but by using different kernel technique for linear kernel highest accuracy was found

In [138]:
```
#Using voting classifier
models=[]
models.append(('logistic',LogisticRegression()))
models.append(('Dt',DecisionTreeClassifier()))
models.append(('SVM',SVC()))
models.append(('rf',RandomForestClassifier()))
models.append(('knn',KNeighborsClassifier()))
models.append(('nb',MultinomialNB()))
```

In [112]:
```
vg=VotingClassifier(models)
vg.fit(xtrain,ytrain)
ypred=vg.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.72      0.76      2958
           1       0.87      0.91      0.89      6111

    accuracy                           0.85      9069
   macro avg       0.83      0.82      0.82      9069
weighted avg       0.85      0.85      0.85      9069
```

This Classifier increases predictive power of ML algorithm after using and interpreting every algorithm it has got accuracy of 85 %

Handling imbalance data by Oversampling

In [139]:
```
!pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.10.1-py3-none-any.whl (226 kB)
Requirement already satisfied: numpy>=1.17.3 in c:\users\admin\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(1.21.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\anaconda3\lib\site-packages (from imbalanced-learn->imble
arn) (2.2.0)
Collecting joblib>=1.1.1
  Downloading joblib-1.2.0-py3-none-any.whl (297 kB)
Requirement already satisfied: scipy>=1.3.2 in c:\users\admin\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.
7.3)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\admin\anaconda3\lib\site-packages (from imbalanced-learn->imblea
rn) (1.0.2)
Installing collected packages: joblib, imbalanced-learn, imblearn
  Attempting uninstall: joblib
    Found existing installation: joblib 1.1.0
    Uninstalling joblib-1.1.0:
      Successfully uninstalled joblib-1.1.0
Successfully installed imbalanced-learn-0.10.1 imblearn-0.0 joblib-1.2.0
```

In [140]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [141]:
```
ros= RandomOverSampler(random_state=1)
```

In [142]:
```
x,y=ros.fit_resample(xtrain,ytrain)
```

In [144]:
```
#Balancing data
pd.Series(y).value_counts()
```

Out[144]:
```
1    18279
0    18279
Name: booking_status, dtype: int64
```

In [164]:
```
pd.Series(ytrain).value_counts()
```

Out[164]:
```
1    13733
0    13685
Name: booking_status, dtype: int64
```

In [145]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=1)
```

In [146]:
```
def mymodel1(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)
    print(classification_report(ytest,ypred))
    return model
```

In [147]:
```
lr=LogisticRegression()
dtc=DecisionTreeClassifier()
rf=RandomForestClassifier()
svm=SVC()
knn=KNeighborsClassifier(n_neighbors=5)
mnb=MultinomialNB()
```

In [148]:
```
mymodel1(lr)
```

```
              precision    recall  f1-score   support

           0       0.76      0.75      0.75      4594
           1       0.75      0.76      0.75      4546

    accuracy                           0.75      9140
   macro avg       0.75      0.75      0.75      9140
weighted avg       0.75      0.75      0.75      9140
```

Out[148]:
```
LogisticRegression()
```

In [151]:
```
mymodel1(dtc)
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.92      4594
           1       0.94      0.88      0.91      4546

    accuracy                           0.91      9140
   macro avg       0.91      0.91      0.91      9140
weighted avg       0.91      0.91      0.91      9140
```

Out[151]:
```
DecisionTreeClassifier()
```

In [152]:
```
mymodel1(rf)
```

```
              precision    recall  f1-score   support

           0       0.92      0.95      0.94      4594
           1       0.95      0.92      0.93      4546

    accuracy                           0.93      9140
   macro avg       0.93      0.93      0.93      9140
weighted avg       0.93      0.93      0.93      9140
```

Out[152]:
```
RandomForestClassifier()
```

In [153]:
```
mymodel1(svm)
```

```
              precision    recall  f1-score   support

           0       0.74      0.59      0.65      4594
           1       0.65      0.79      0.71      4546

    accuracy                           0.69      9140
   macro avg       0.70      0.69      0.68      9140
weighted avg       0.70      0.69      0.68      9140
```

Out[153]:
```
SVC()
```

In [154]:
```
mymodel1(knn)
```

```
              precision    recall  f1-score   support

           0       0.79      0.81      0.80      4594
           1       0.80      0.78      0.79      4546

    accuracy                           0.79      9140
   macro avg       0.79      0.79      0.79      9140
weighted avg       0.79      0.79      0.79      9140
```

Out[154]:
```
KNeighborsClassifier()
```

In [155]:
```
mymodel1(mnb)
```

```
              precision    recall  f1-score   support

           0       0.73      0.61      0.66      4594
           1       0.66      0.77      0.71      4546

    accuracy                           0.69      9140
   macro avg       0.70      0.69      0.69      9140
weighted avg       0.70      0.69      0.69      9140
```

Out[155]:
```
MultinomialNB()
```

It is the classification report of all algorithms after Balancing the dataset, RandomForest has highest accuracy

In [ ]: