In [1]:
```
# Import the Standard libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import plotly.express as px
import plotly.io as pio
import matplotlib.image as mpimg
import plotly.graph_objects as go
warnings.filterwarnings('ignore')
```

In [2]:
```
# Load the data
data=pd.read_csv('C://Users//Admin//Downloads//archive//Passanger_booking_data.csv')
data.head()
```

Out[2]:

	num_passengers	sales_channel	trip_type	purchase_lead	length_of_stay	flight_hour	flight_day	
0	1	Internet	RoundTrip	21	12	6	Tue	A
1	2	Internet	RoundTrip	262	19	7	Sat	A
2	1	Internet	RoundTrip	112	20	3	Sat	A
3	2	Internet	RoundTrip	243	22	17	Wed	A
4	1	Internet	RoundTrip	96	31	4	Sat	A

Data Preprocessing

- In the data preprocessing phase, we first examined the shape of the data to understand its dimensions.
- Next, we checked for null values in the dataset and removed them if any were found.
- Additionally, we performed a check for duplicate values and replaced them to ensure data integrity.
- To gain insights into the relationships between different variables, we visualized the correlation map using a heatmap. This visualization allowed us to identify patterns and dependencies among the features in the dataset, helping us understand the interplay between various attribute

In [3]:
```
data.columns
```

Out[3]:
```
Index(['num_passengers', 'sales_channel', 'trip_type', 'purchase_lead',
       'length_of_stay', 'flight_hour', 'flight_day', 'route',
       'booking_origin', 'wants_extra_baggage', 'wants_preferred_seat',
       'wants_in_flight_meals', 'flight_duration', 'booking_complete'],
      dtype='object')
```

In [4]:
```
#Checking the data shape
print(f'The dataset contains {data.shape[0]} rows and {data.shape[1]} columns')
```

```
The dataset contains 50002 rows and 14 columns
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50002 entries, 0 to 50001
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   num_passengers         50002 non-null  int64  
 1   sales_channel          50002 non-null  object 
 2   trip_type              50002 non-null  object 
 3   purchase_lead          50002 non-null  int64  
 4   length_of_stay         50002 non-null  int64  
 5   flight_hour            50002 non-null  int64  
 6   flight_day             50002 non-null  object 
 7   route                  50002 non-null  object 
 8   booking_origin         50002 non-null  object 
 9   wants_extra_baggage    50002 non-null  int64  
 10  wants_preferred_seat   50002 non-null  int64  
 11  wants_in_flight_meals  50002 non-null  int64  
 12  flight_duration        50002 non-null  float64
 13  booking_complete       50002 non-null  int64  
dtypes: float64(1), int64(8), object(5)
memory usage: 5.3+ MB
```

In [6]:
```
# Stastic about the data set
data.describe().style.background_gradient(cmap='bone_r')
```

Out[6]:

	num_passengers	purchase_lead	length_of_stay	flight_hour	wants_extra_baggage	wants_pref
count	50002.000000	50002.000000	50002.000000	50002.000000	50002.000000	50
mean	1.591256	84.940582	23.044778	9.066277	0.668773	
std	1.020167	90.450548	33.887171	5.412569	0.470659	
min	1.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	21.000000	5.000000	5.000000	0.000000	
50%	1.000000	51.000000	17.000000	9.000000	1.000000	
75%	2.000000	115.000000	28.000000	13.000000	1.000000	
max	9.000000	867.000000	778.000000	23.000000	1.000000	

In [7]:
```
# Checking the null values in the dataset
data.isna().sum()/len(data)*100
```

Out[7]:
```
num_passengers           0.0
sales_channel            0.0
trip_type                0.0
purchase_lead            0.0
length_of_stay           0.0
flight_hour              0.0
flight_day               0.0
route                    0.0
booking_origin           0.0
wants_extra_baggage      0.0
wants_preferred_seat     0.0
wants_in_flight_meals    0.0
flight_duration          0.0
booking_complete         0.0
dtype: float64
```

In [8]:
```
# Checking the duplicate values in the data
duplicate_values=data.duplicated().sum()
print(f'The data contains {duplicate_values} duplicate values')
```

```
The data contains 719 duplicate values
```

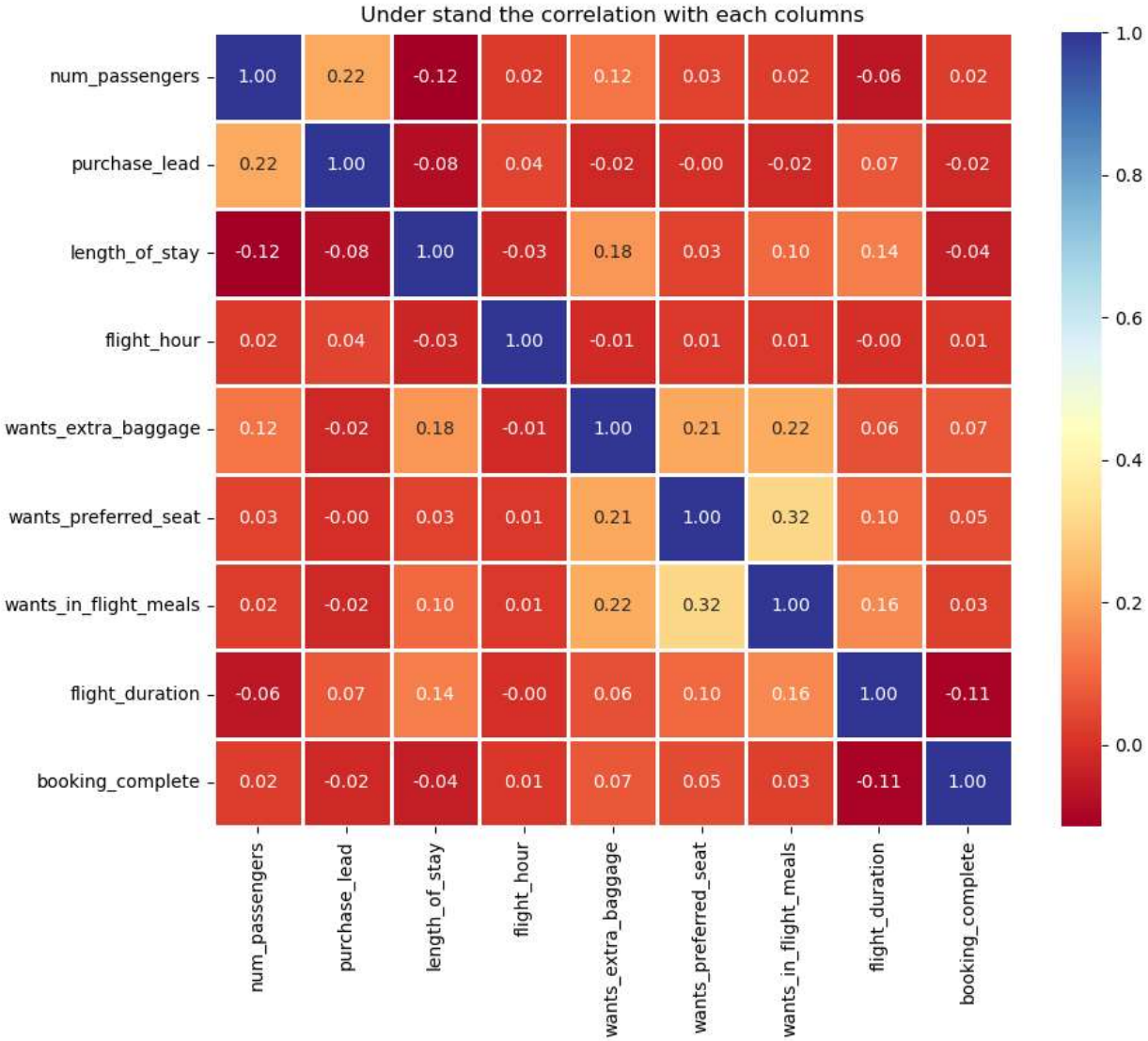
In [9]:
```
# drop the duplicate values in the dataset we using the pandas function
data=data.drop_duplicates()
data.shape
```

Out[9]:
```
(49283, 14)
```

💡About the DataSet

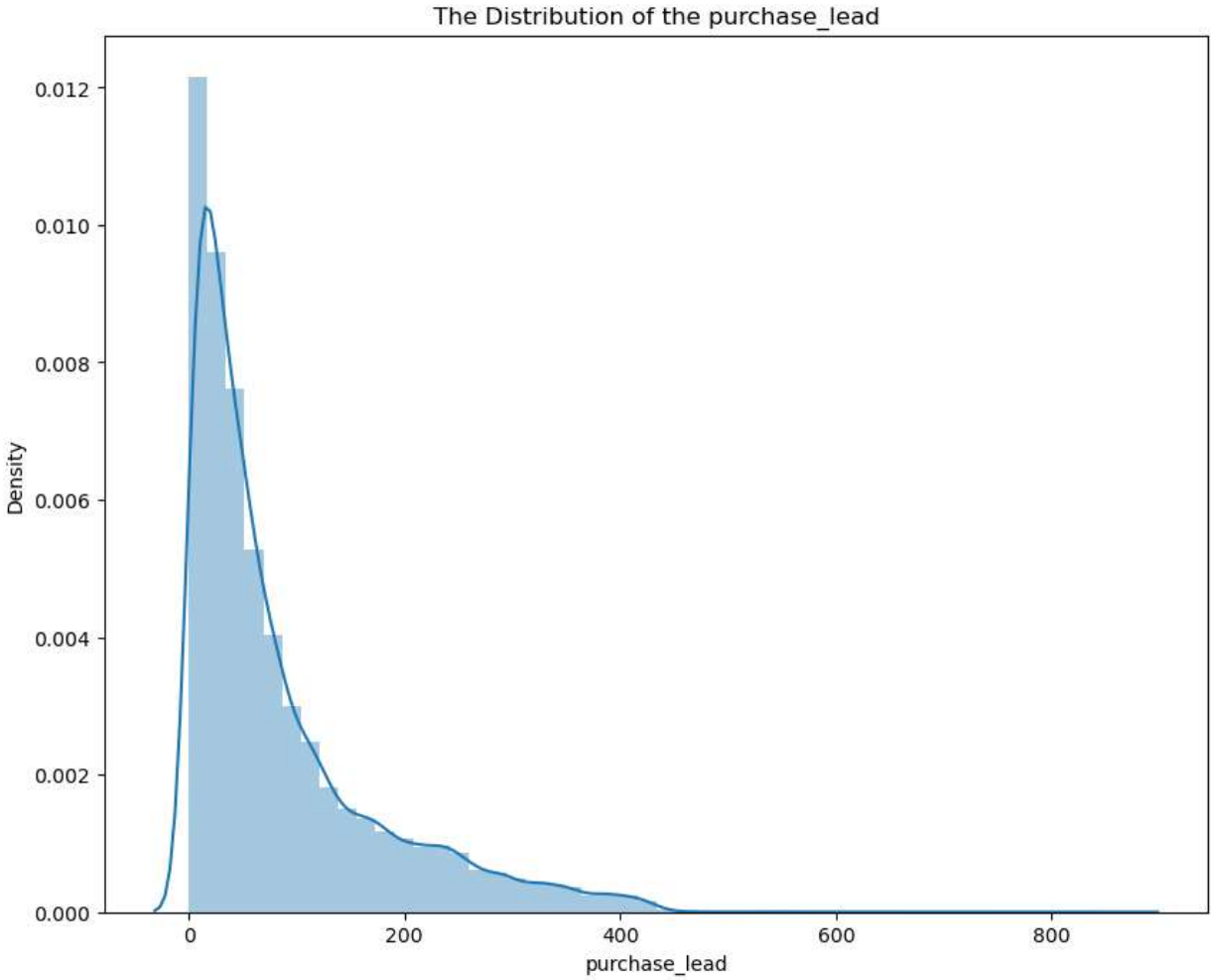
- #### Data Size: The dataset contains 49283 rows and 14 columns.
- #### Data Types: The data contains features with data types int64, Object, Binary and float64.
- #### Missing Values: No column has missing values in the dataset, which is a great sign and simplifies the data cleaning process.
- #### Unique Values: The number of unique values varies among features.
- #### Statistical Details: The 'min', 'max', 'average', and 'standard deviation' values indicate the range and dispersion of data for each column, highlighting potential outliers or anomalies.
- #### Irrelevant Features: All the Features seems important and useful for final evaluation.

In [10]:
```
# Visualize the correlation map
plt.figure(figsize=(10,8))
sns.heatmap(data.corr(),annot=True,cmap='RdYlBu',fmt='.2f',
    annot_kws=None,
    linewidths=1,)
plt.title("Under stand the correlation with each columns")
plt.show()
```



Explore Data Analysis

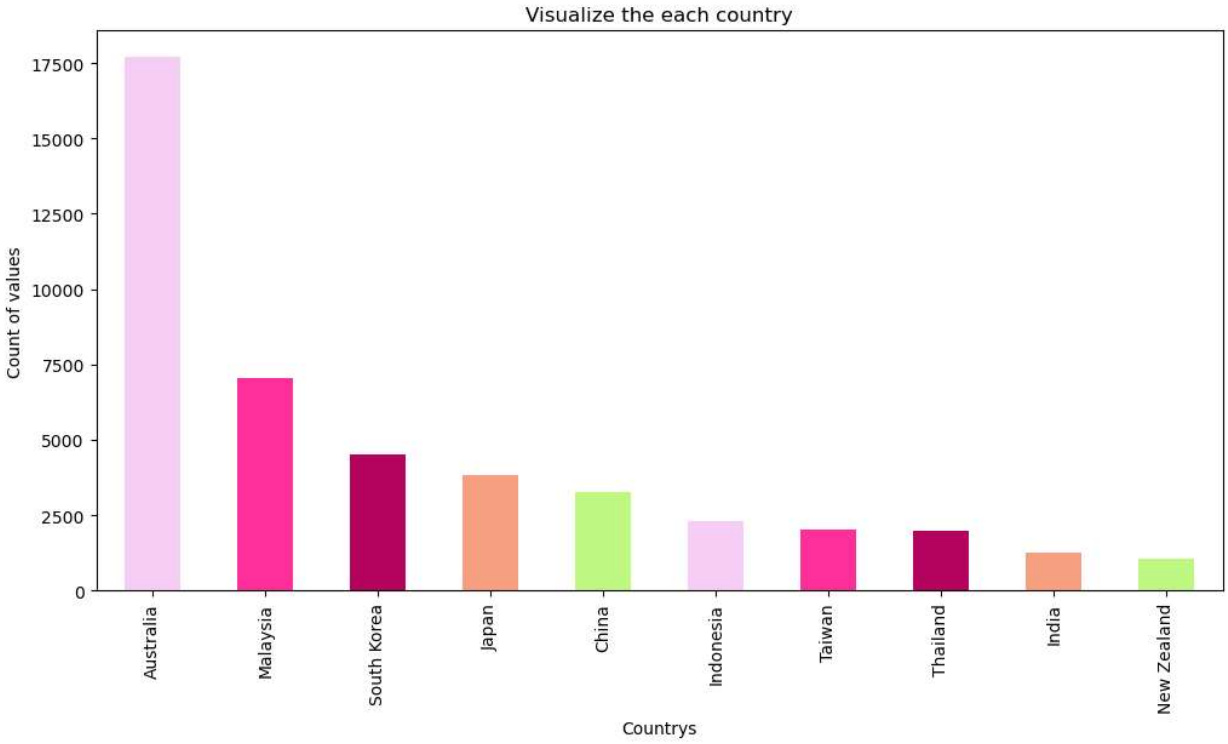
In [11]:
```
# Distribution of the num of passengers in the data
plt.figure(figsize=(10,8))
sns.distplot(data['purchase_lead'],hist=True,bins=50)
plt.title("The Distribution of the purchase_lead")
plt.show()
```



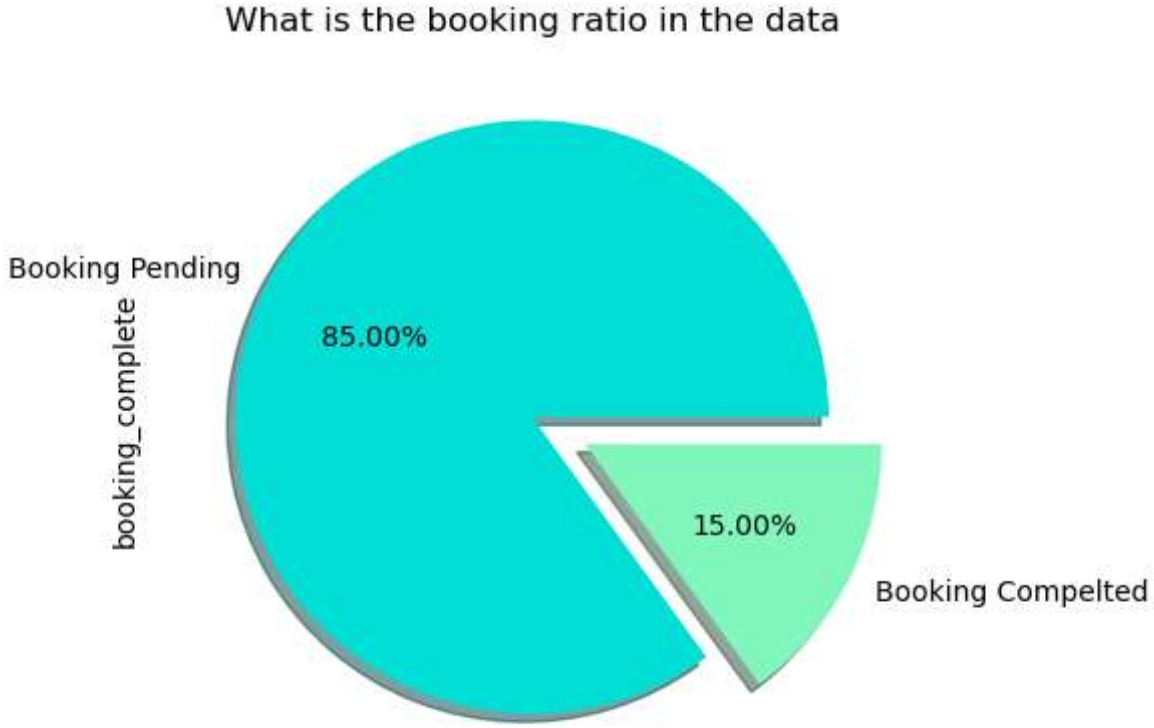
In [12]:
```
total_sales_channel=data['trip_type'].value_counts()/data['trip_type'].value_counts().
internet=data[data['sales_channel']=='Internet'].trip_type.value_counts()
Mobile=data[data['sales_channel']=='Mobile'].trip_type.value_counts()
total_sales=data['trip_type'].value_counts()
sales_done_by_mobile=(total_sales/internet)-.108
total_sales['Color']=np.where(sales_done_by_mobile < 0, 'blue','red')
fig=go.Figure(go.Bar(x=(Mobile/total_sales).index, y= (Mobile/total_sales).values-.108
fig.show()
```

In [13]:
```
# Create a bar plot visualize the top 10 most demanding origin
data['booking_origin'].value_counts().sort_values(ascending=False).nlargest(10).plot(k
plt.title("Visualize the each country")
plt.xlabel("Countrys")
plt.xticks(rotation=90)
plt.ylabel("Count of values")
```

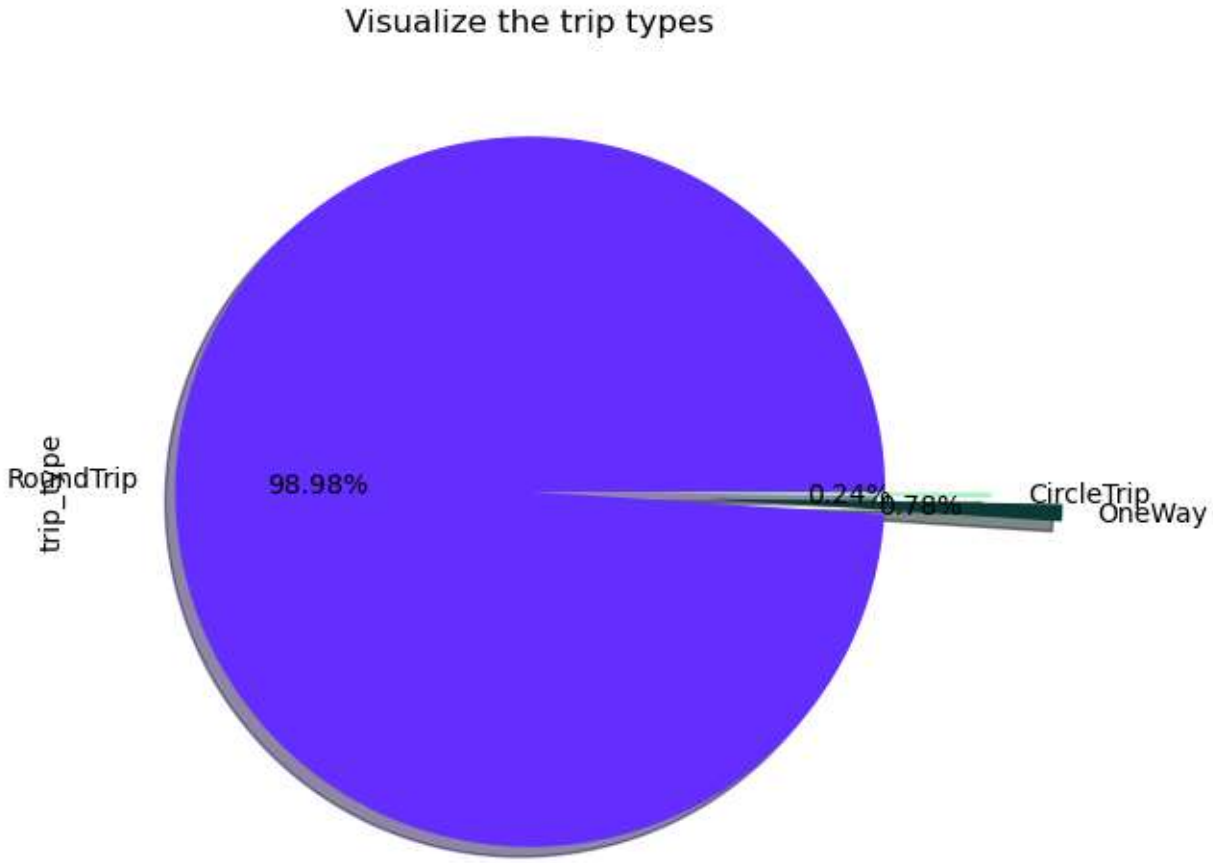
Out[13]:
```
Text(0, 0.5, 'Count of values')
```



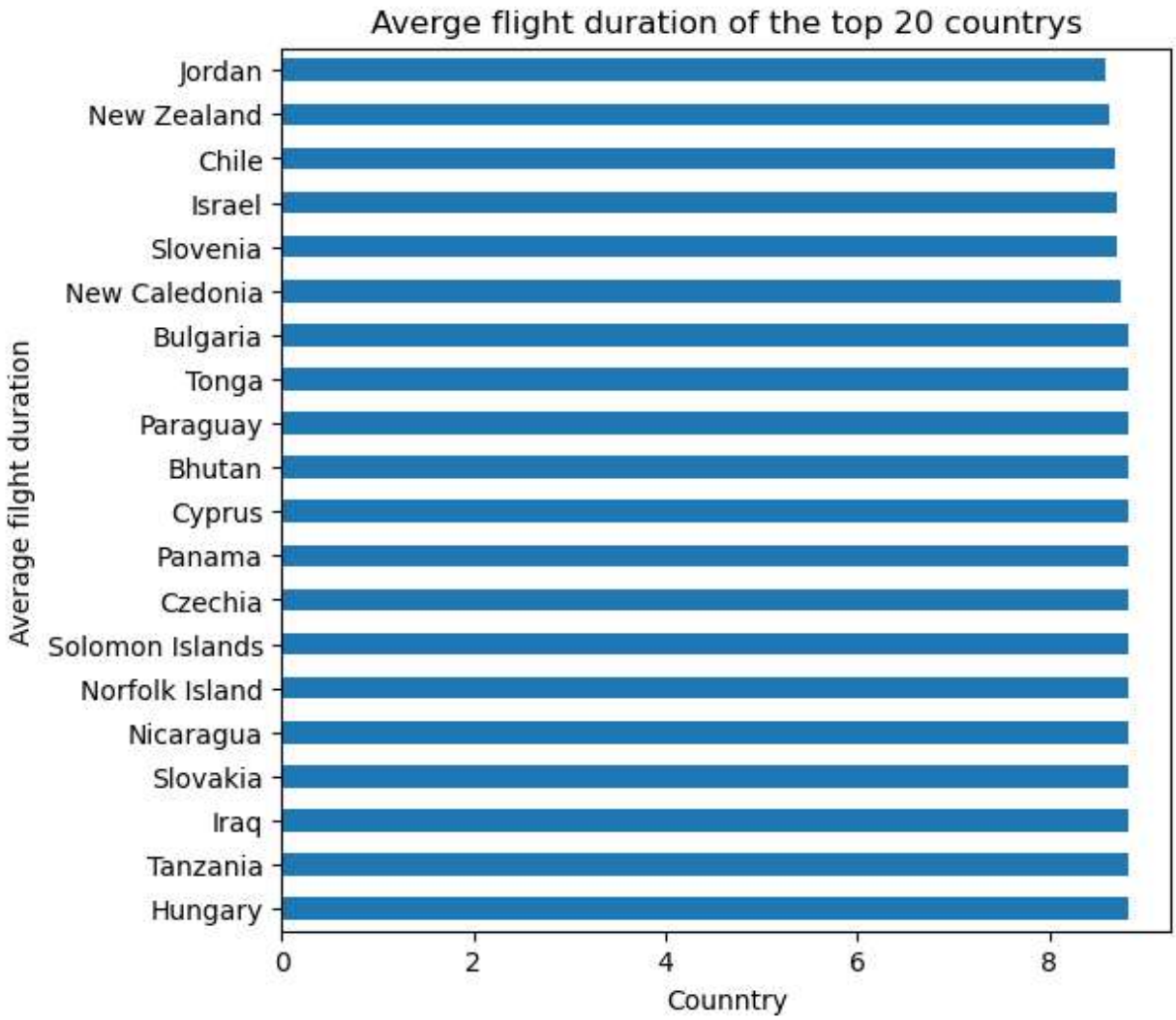
In [14]:
```
# Using the pie chart understand the how much precentage is complete there bookings
data['booking_complete'].value_counts().plot(kind='pie',
    explode=[0,0.2],
    labels=['Booking Pending',"Booking Compelted"],
    colors=['#01DFD7','#81F7BE'],
    autopct='%1.2f%%',
    shadow=True)
plt.title("What is the booking ratio in the data")
plt.show()
```



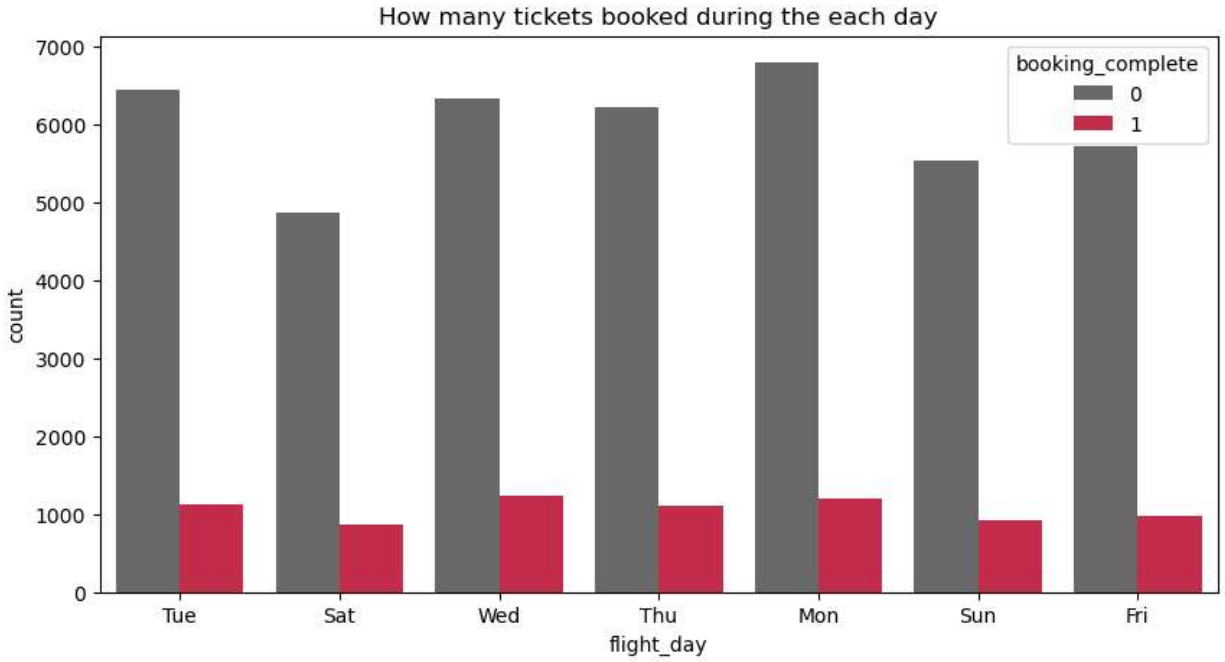
In [15]:
```
# How much percentage people intrested in the trip types
trip_type=['trip_type']
for i,col in enumerate(trip_type):
    plt.figure(figsize=(10,6))
    data[col].value_counts().plot(kind='pie',explode=[0,0.5,0.3],
    labels=['RoundTrip',"OneWay",'CircleTrip'],
    colors=['#642EFE','#0B3B39','#A9F5BC'],
    autopct='%1.2f%%',
    shadow=True)
plt.title("Visualize the trip types")
plt.show()
```



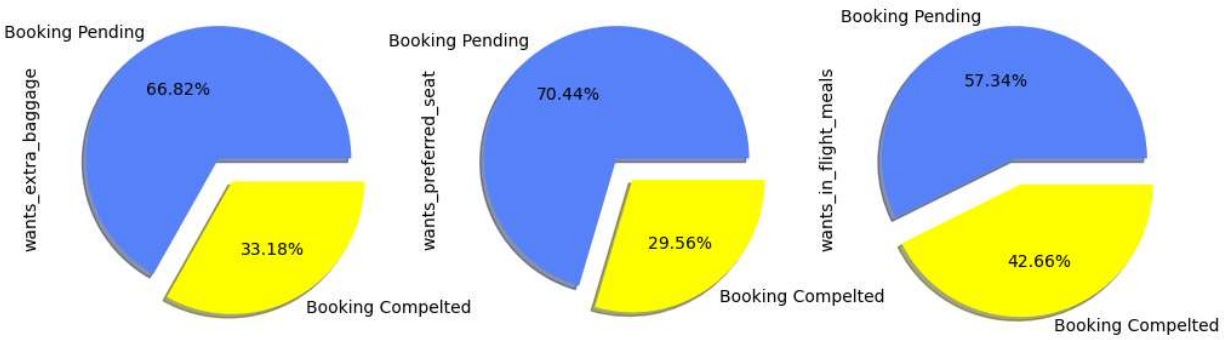
In [16]:
```
# Find the average flight duration of the top 20 destination in the bar charts
data.groupby('booking_origin')['flight_duration'].mean().sort_values(ascending=False).
.plot(kind='barh',figsize=(6,6))
plt.title("Averge flight duration of the top 20 countrys")
plt.xlabel("Counntry")
plt.ylabel("Average filght duration")
plt.show()
```



In [17]:
```
# Create countplot understand the booking status on the flight day
plt.figure(figsize=(10,5))
sns.countplot(data=data,x='flight_day',hue='booking_complete',palette=['dimgrey','crir
plt.title("How many tickets booked during the each day")
plt.show()
```

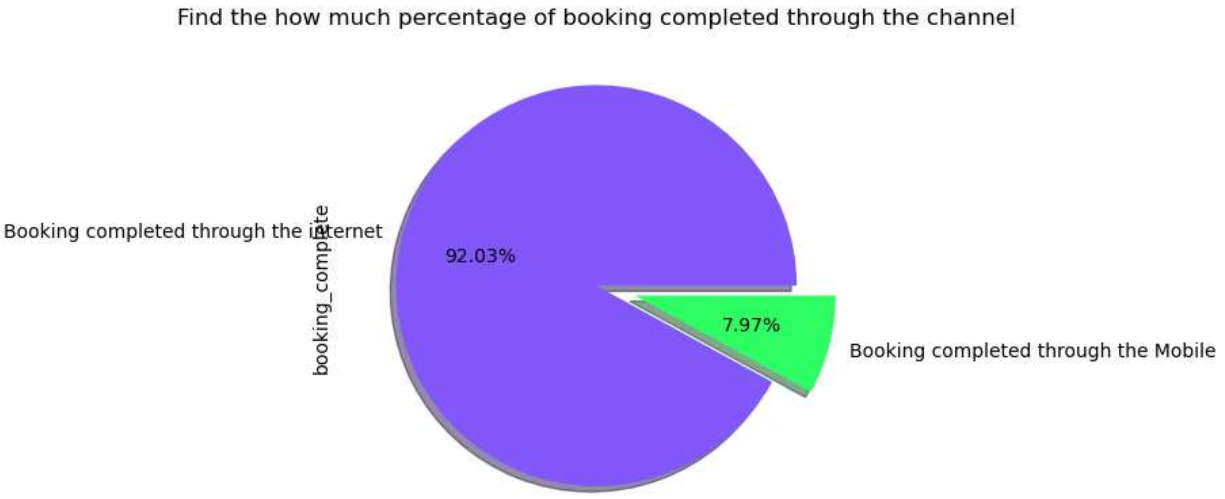


In [18]:
```
# Create a data frame for the extra
df=['wants_extra_baggage', 'wants_preferred_seat',
       'wants_in_flight_meals']
plt.figure(figsize=(12,5))
for i,col in enumerate(df):
    plt.subplot(1,3,i+1)
    data[col].value_counts().plot(kind='pie',explode=[0,0.2],
    labels=['Booking Pending',"Booking Compelted"],
    colors=['#5882FA','#FFFF00'],
    autopct='%1.2f%%',
    shadow=True)
```



In [19]:
```
# Let's Compare How much percentage of the booking complete through mobile and interne
data.groupby('sales_channel')['booking_complete'].sum().plot(kind='pie',
    explode=[0,0.2],
    labels=['Booking completed through the internet',"Booking completed through the Mo
    colors=['#8258FA','#2EFE64'],
    autopct='%1.2f%%',
    shadow=True)
plt.title("Find the how much percentage of booking completed through the channel")
```

Out[19]:
```
Text(0.5, 1.0, 'Find the how much percentage of booking completed through the channe
l')
```

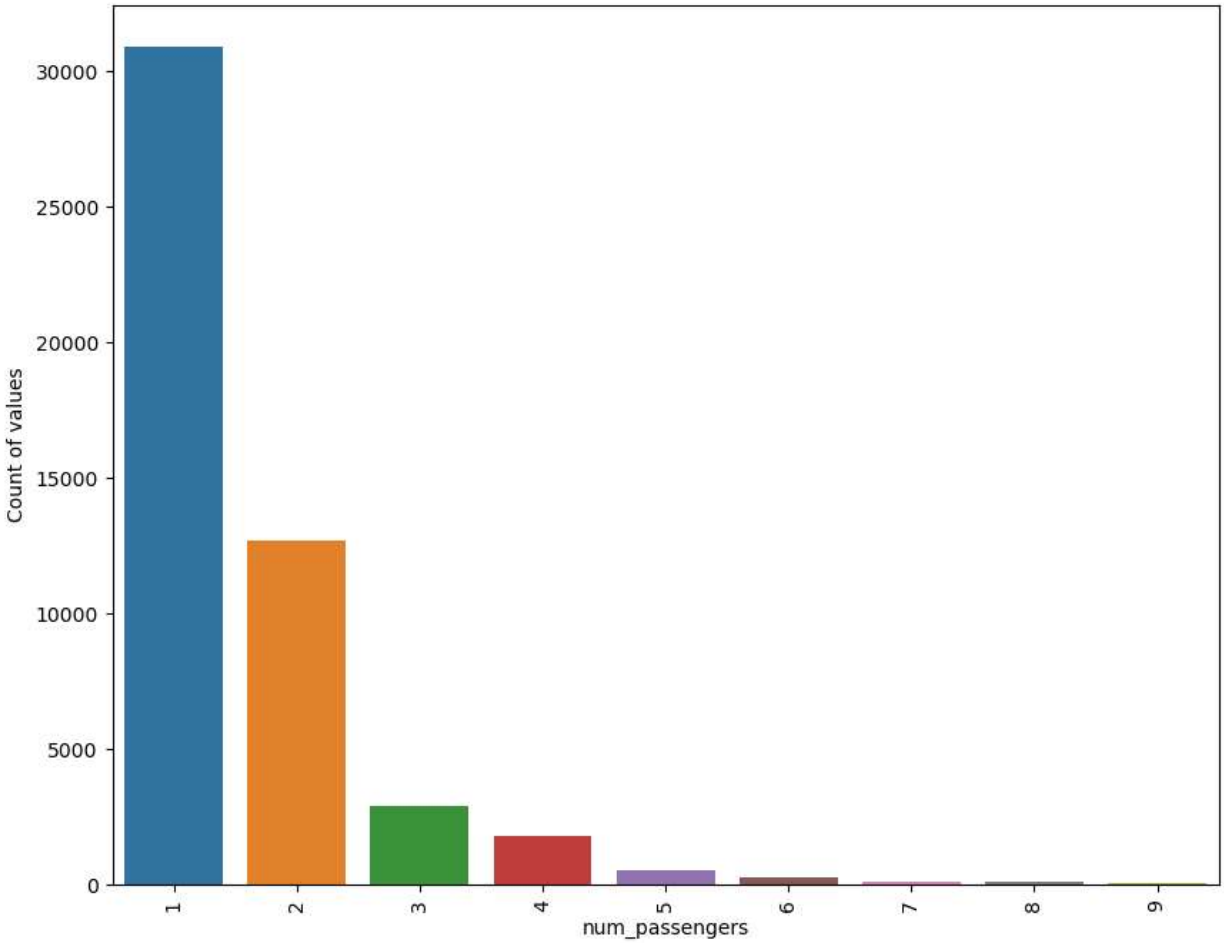


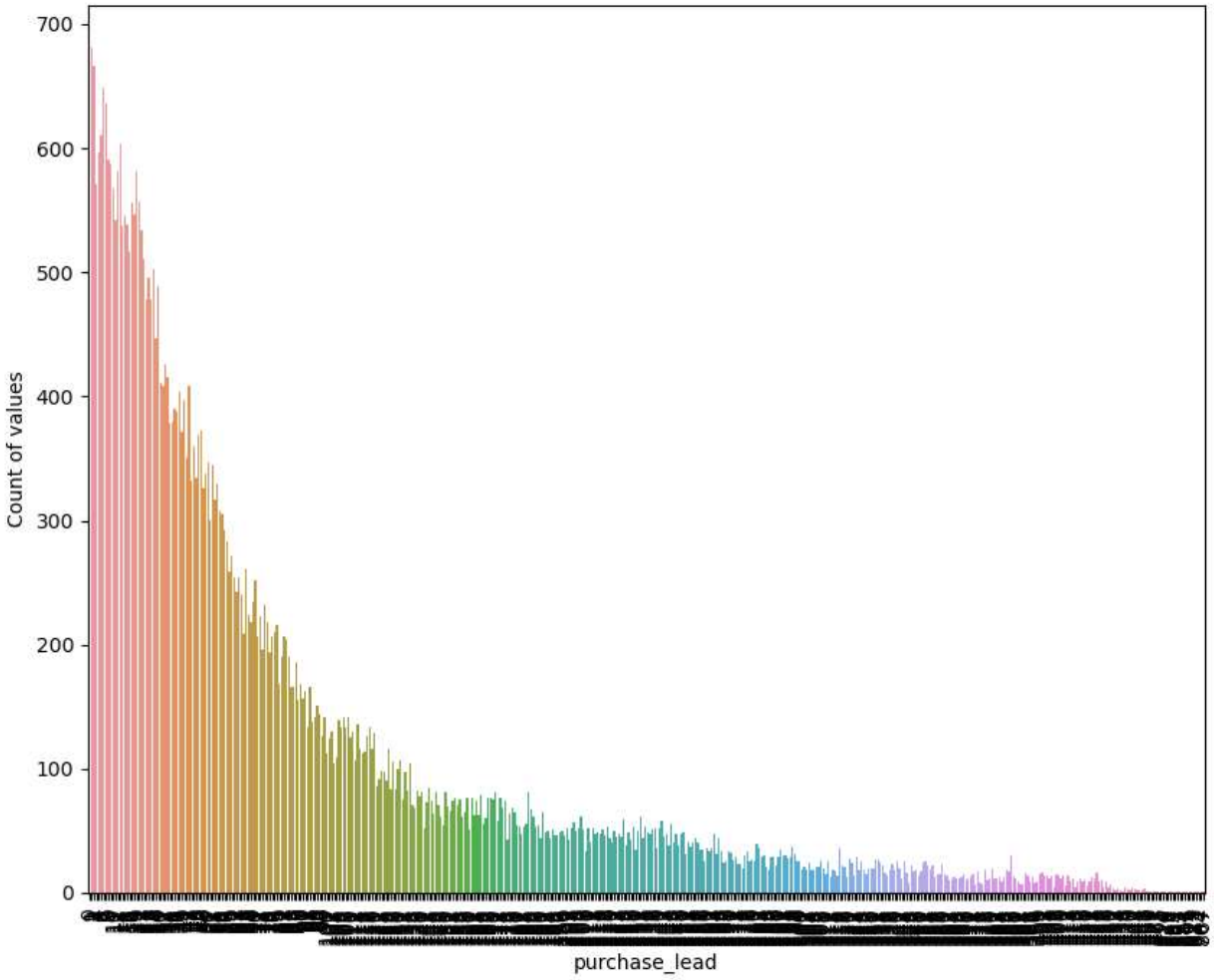
In [20]:
```
# create a separate data frame for Roundtrip and find the which day most of the ticket
roundtrip=data[data['trip_type']=='RoundTrip']
roundtrip.groupby(['flight_day'])[['booking_complete']].value_counts().sort_values(asc
.unstack()\
.style.background_gradient(cmap='PuBuGn_r')
```

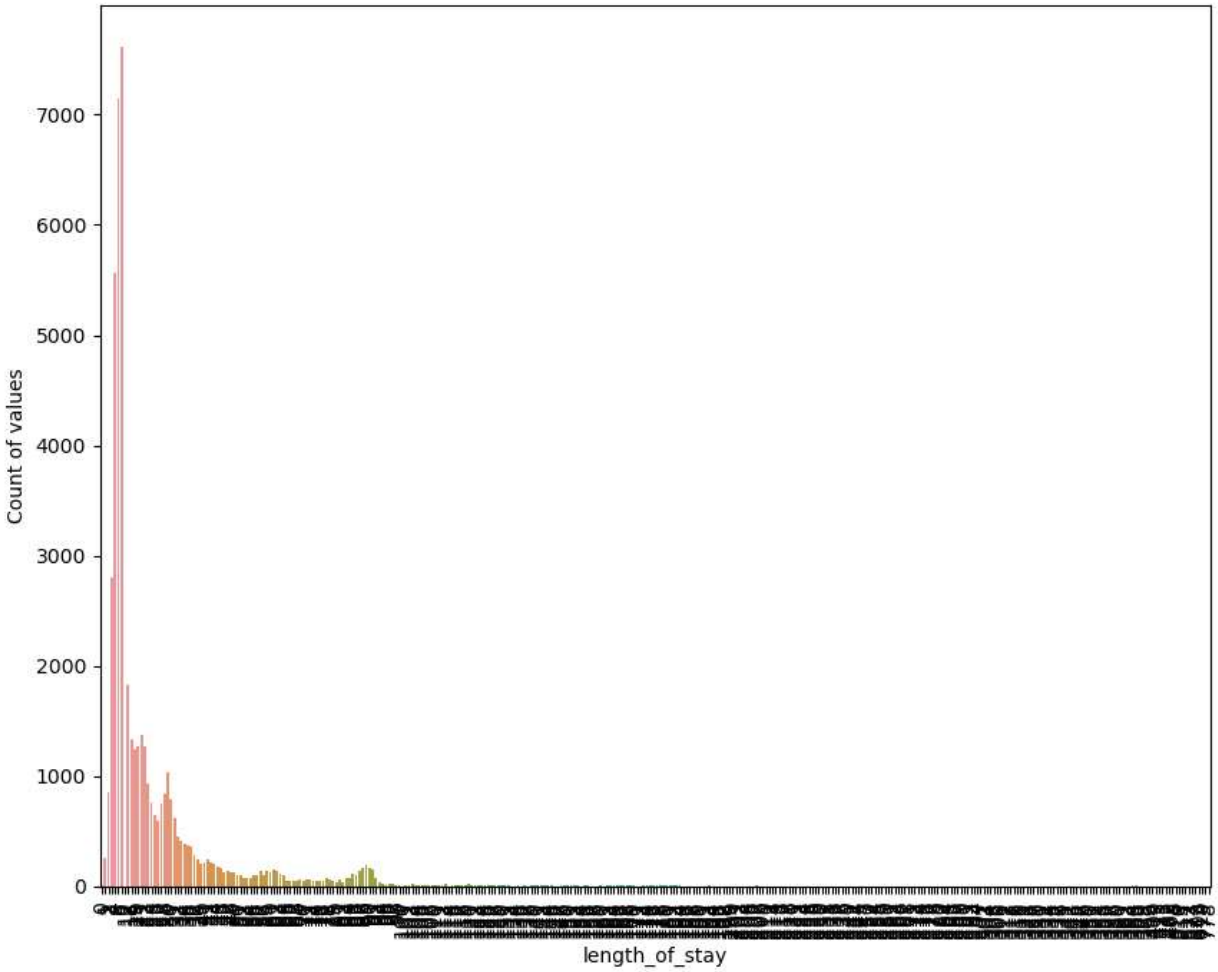
Out[20]:

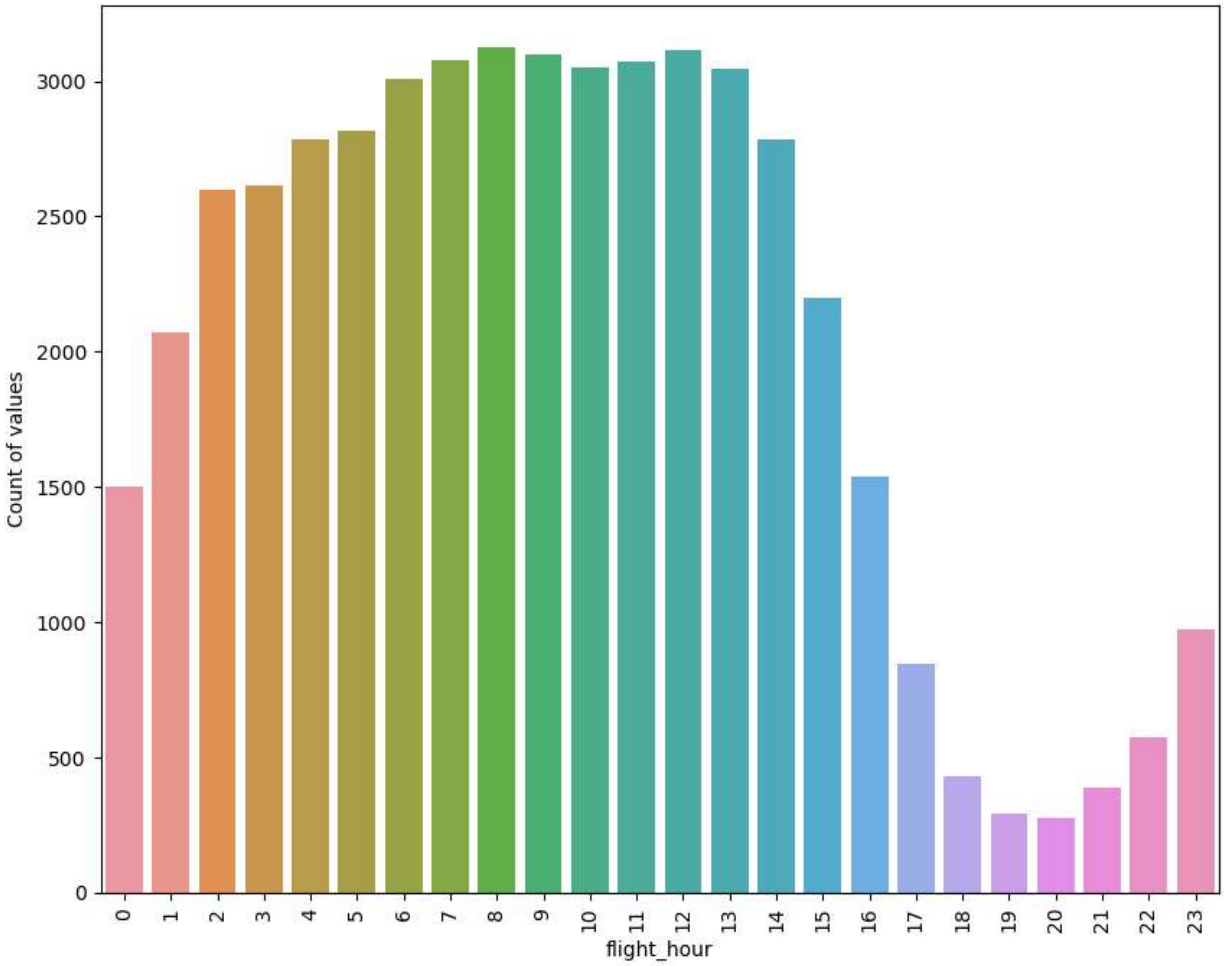
booking_complete	0	1
flight_day		
Fri	5647	971
Mon	6724	1191
Sat	4809	853
Sun	5465	911
Thu	6151	1104
Tue	6350	1115
Wed	6267	1222

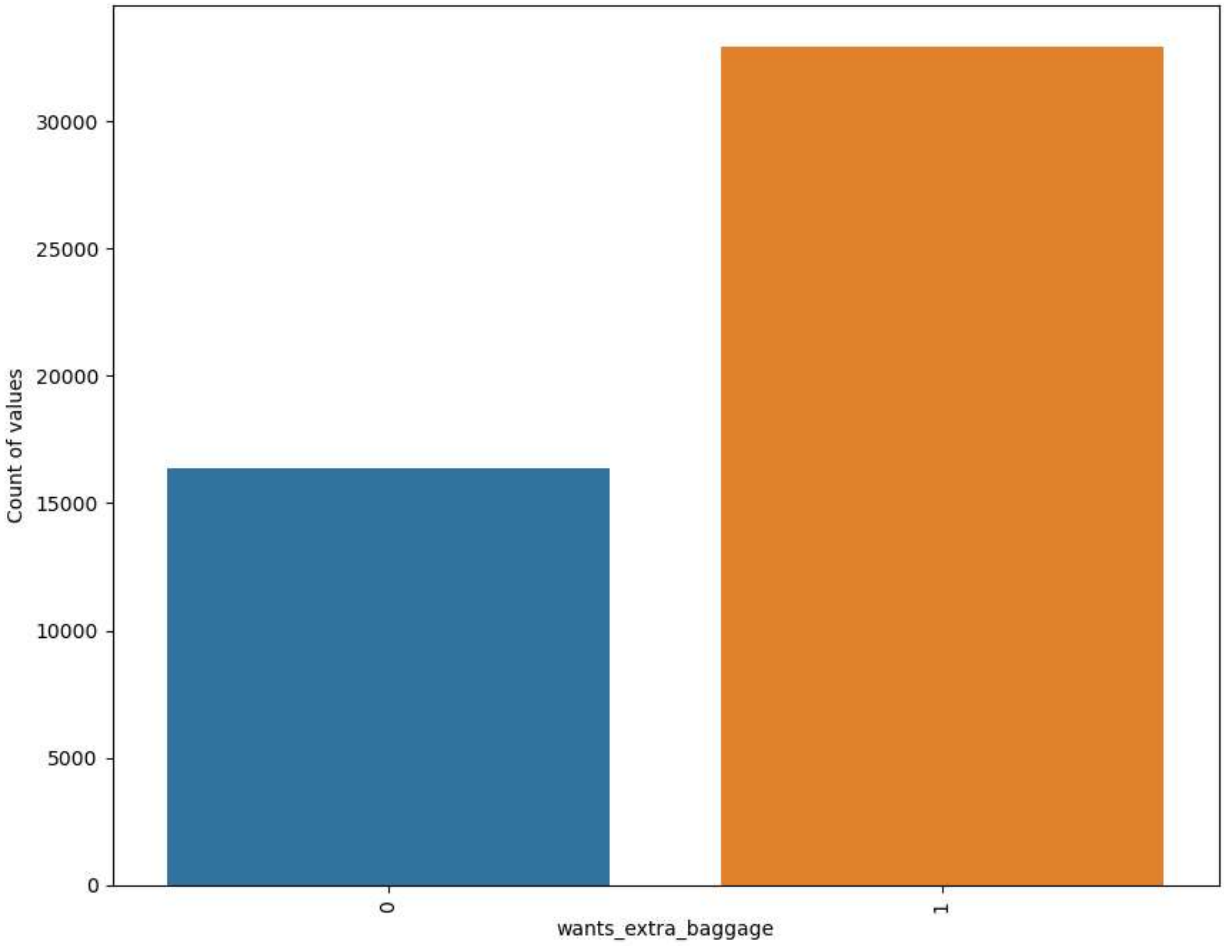
In [21]:
```
# Create a countplot to understand the each columns values in the data
for i in data.select_dtypes(include='int'):
    plt.figure(figsize=(10,8))
    sns.countplot(data=data,x=data[i])
    plt.xlabel(f'{i}')
    plt.ylabel('Count of values')
    plt.xticks(rotation=90)
    plt.show()
```

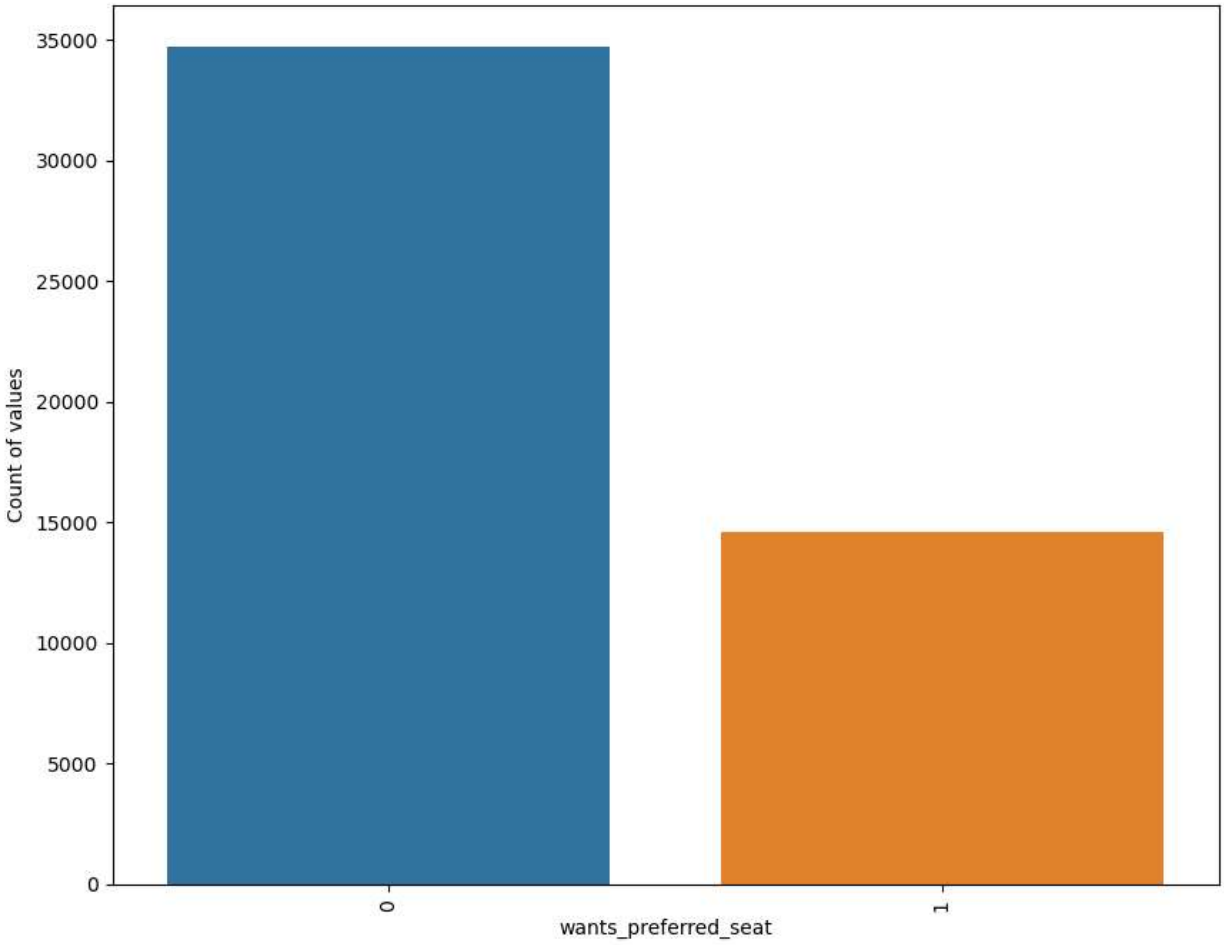

30000
25000
20000
15000
10000
5000
0
Count of values
1
2
3
4
5
6
7
8
9
num_passengers

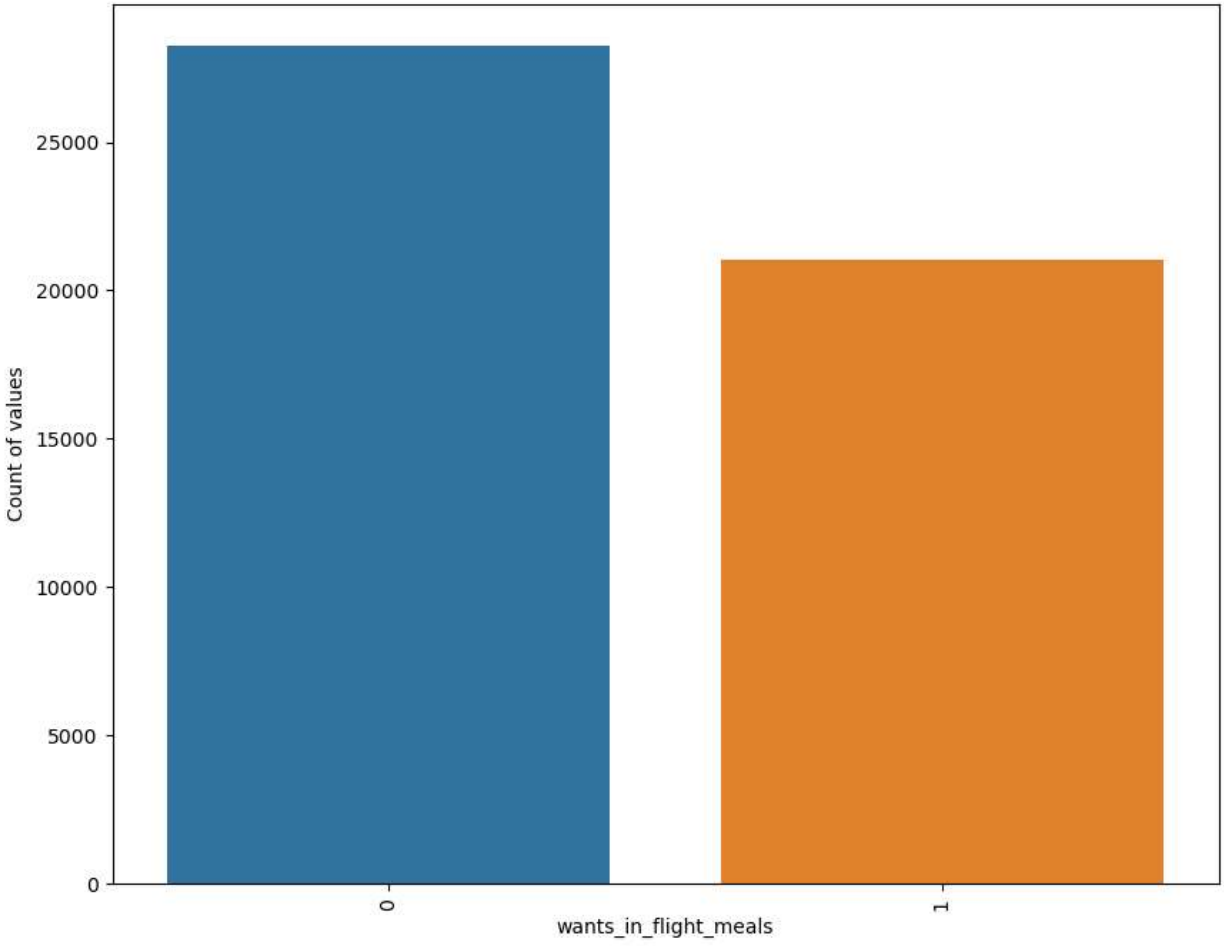

700
600
500
400
300
200
100
0
Count of values
purchase_lead

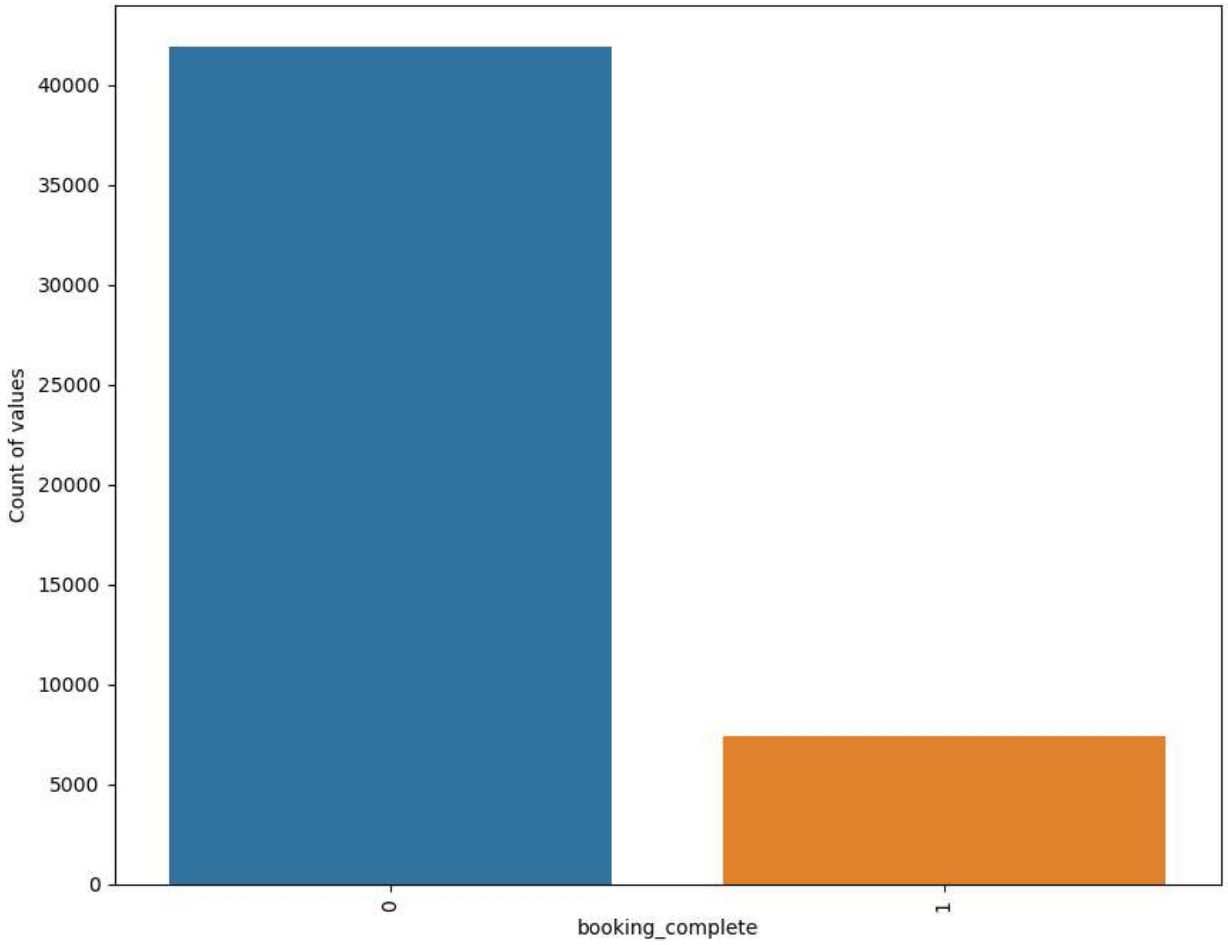

7000
6000
5000
4000
3000
2000
1000
0
Count of values
length_of_stay


Count of values
3000
2500
2000
1500
1000
500
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
flight_hour


Count of values
30000
25000
20000
15000
10000
5000
0
0 1
wants_extra_baggage


35000
30000
25000
20000
15000
10000
5000
0
Count of values
0
1
wants_preferred_seat


25000
20000
15000
10000
5000
0
Count of values
0
1
wants_in_flight_meals



In [22]:
```
# Using the groupby function we find the how many bookings are done in each day trough
data.groupby(['sales_channel'])[['flight_day']].value_counts().sort_values(ascending=F
.unstack()\
.style.background_gradient(cmap='gist_heat_r')
```

Out[22]:

flight_day	Fri	Mon	Sat	Sun	Thu	Tue	Wed
sales_channel							
Internet	5897	7197	4965	5591	6606	6827	6835
Mobile	788	791	758	851	717	732	728

In [23]:
```
#Some intresting question asked from the data
# Find the day which is more bussy and less busy days in the data
print('Which is the busy day in the flight booking counter',data['flight_day'].value_c
print('\nWhich is the less busy day in the flight booking counter',data['flight_day'].
# And the we also do same thing about the booking origin
print('\nWhich day booking origin is bussy',data['booking_origin'].value_counts().idxm
print('\nWhich day booking origin is less busy',data['booking_origin'].value_counts().
```

```
Which is the busy day in the flight booking counter Mon

Which is the less busy day in the flight booking counter Sat

Which day booking origin is bussy Australia

Which day booking origin is less busy Vanuatu
```

In [24]:
```
# Using the pivot table find the each origin and day which day most ticket are booked.
pd.pivot_table(data,index='booking_origin',columns='flight_day',values='booking_comple
.style.background_gradient(cmap='icefire')
```

Out[24]:

flight_day	Fri	Mon	Sat	Sun	Thu	Tue	Wed
booking_origin							
(not set)	0.272727	0.000000	1.000000	0.166667	0.076923	0.076923	0.428571
Afghanistan	nan	nan	nan	nan	0.000000	nan	nan
Algeria	nan	nan	nan	nan	0.000000	nan	nan
Argentina	nan	0.000000	nan	nan	0.000000	0.000000	0.000000
Australia	0.043275	0.050264	0.053197	0.049311	0.054953	0.050225	0.052612
Austria	0.000000	0.000000	nan	0.000000	0.000000	nan	0.000000
Bahrain	nan	nan	0.500000	nan	0.666667	nan	nan
Bangladesh	0.000000	0.000000	0.200000	0.000000	0.000000	0.000000	0.200000
Belarus	0.000000	nan	nan	nan	nan	nan	nan
Belgium	0.000000	nan	0.000000	0.000000	nan	0.000000	1.000000
Bhutan	nan	nan	nan	nan	0.000000	nan	nan
Brazil	0.000000	0.000000	nan	0.000000	0.000000	0.000000	nan
Brunei	0.310345	0.117647	0.117647	0.240000	0.347826	0.266667	0.250000
Bulgaria	nan	nan	nan	nan	nan	0.000000	nan
Cambodia	0.437500	0.166667	0.181818	0.375000	0.200000	0.421053	0.192308
Canada	0.000000	0.000000	0.000000	0.142857	0.000000	0.090909	0.000000
Chile	0.000000	0.000000	nan	0.000000	0.000000	0.200000	0.000000
China	0.214442	0.204710	0.224490	0.176638	0.184426	0.229323	0.215983
Colombia	nan	nan	nan	nan	0.000000	0.000000	0.000000
Croatia	nan	0.000000	nan	nan	nan	nan	0.000000
Cyprus	nan	nan	nan	nan	nan	0.000000	0.000000
Czech Republic	0.000000	0.000000	0.000000	nan	0.000000	nan	0.500000
Czechia	nan	nan	nan	0.000000	nan	nan	nan
Denmark	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	nan
Egypt	nan	nan	0.000000	nan	nan	nan	nan
Estonia	nan	nan	nan	nan	nan	nan	0.000000
Finland	0.000000	0.000000	0.000000	0.000000	nan	0.000000	0.000000
France	0.200000	0.272727	0.000000	0.142857	0.125000	0.333333	0.333333
Germany	0.142857	0.111111	0.333333	0.111111	0.500000	0.428571	0.142857
Ghana	nan	nan	nan	nan	nan	0.000000	nan
Gibraltar	nan	nan	nan	nan	0.000000	nan	nan
Greece	0.000000	1.000000	0.000000	0.000000	nan	0.000000	0.000000

flight_day	Fri	Mon	Sat	Sun	Thu	Tue	Wed
booking_origin							
Guam	nan	nan	nan	0.000000	nan	nan	nan
Guatemala	nan	nan	0.000000	nan	nan	nan	nan
Hong Kong	0.195652	0.243243	0.137931	0.250000	0.244444	0.260870	0.327273
Hungary	0.000000	0.000000	0.000000	nan	nan	nan	nan
India	0.082474	0.113772	0.101124	0.092896	0.124294	0.061453	0.138889
Indonesia	0.270096	0.228947	0.233449	0.236686	0.305296	0.263889	0.306250
Iran	0.000000	0.000000	nan	0.000000	0.000000	0.000000	0.000000
Iraq	nan	nan	nan	nan	nan	1.000000	nan
Ireland	0.000000	0.000000	0.000000	nan	nan	0.000000	0.000000
Israel	nan	0.000000	nan	nan	nan	nan	0.000000
Italy	0.166667	0.111111	0.285714	0.400000	0.083333	0.000000	0.307692
Japan	0.129151	0.150338	0.111984	0.096672	0.115830	0.131970	0.130879
Jordan	nan	nan	nan	nan	nan	nan	0.000000
Kazakhstan	nan	nan	1.000000	0.000000	1.000000	nan	nan
Kenya	0.000000	nan	0.000000	nan	nan	nan	1.000000
Kuwait	0.000000	1.000000	0.000000	0.500000	nan	0.000000	nan
Laos	0.000000	0.000000	0.400000	0.000000	0.250000	0.400000	0.500000
Lebanon	nan	0.000000	nan	nan	nan	nan	nan
Macau	0.367347	0.270833	0.352941	0.407407	0.357143	0.239130	0.243902
Malaysia	0.341969	0.356624	0.335501	0.357223	0.356597	0.320841	0.345700
Maldives	0.000000	nan	nan	0.000000	0.000000	0.000000	0.000000
Malta	nan	nan	nan	nan	0.000000	nan	nan
Mauritius	0.000000	0.000000	0.000000	0.000000	0.300000	0.000000	0.200000
Mexico	0.000000	0.000000	nan	0.000000	0.000000	1.000000	0.000000
Mongolia	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Myanmar (Burma)	0.333333	0.200000	0.250000	0.250000	0.142857	0.000000	0.428571
Nepal	0.142857	0.000000	0.000000	0.000000	0.000000	0.200000	0.200000
Netherlands	0.166667	0.166667	0.000000	0.000000	0.000000	0.000000	0.000000
New Caledonia	nan	0.000000	nan	0.000000	0.000000	0.000000	nan
New Zealand	0.040000	0.057803	0.040000	0.052326	0.062016	0.058480	0.036364
Nicaragua	nan	nan	0.000000	nan	nan	nan	nan
Norfolk Island	nan	0.000000	nan	nan	nan	nan	nan

flight_day	Fri	Mon	Sat	Sun	Thu	Tue	Wed
booking_origin							
Norway	0.500000	0.000000	nan	0.000000	nan	nan	nan
Oman	0.500000	0.000000	0.000000	0.000000	1.000000	1.000000	nan
Pakistan	nan	1.000000	nan	nan	nan	nan	nan
Panama	nan	nan	nan	nan	nan	nan	0.000000
Papua New Guinea	0.000000	nan	nan	nan	nan	nan	nan
Paraguay	0.000000	nan	nan	nan	nan	nan	nan
Peru	nan	0.000000	nan	nan	nan	0.000000	nan
Philippines	0.304348	0.266667	0.322581	0.125000	0.388889	0.305556	0.234043
Poland	0.000000	0.000000	0.000000	0.000000	0.000000	nan	nan
Portugal	0.000000	nan	nan	0.500000	0.000000	nan	0.000000
Qatar	0.600000	0.000000	0.333333	nan	0.000000	0.000000	0.500000
Romania	nan	0.500000	nan	0.000000	nan	1.000000	nan
Russia	0.000000	0.000000	0.500000	0.000000	0.250000	0.000000	0.500000
Réunion	nan	0.000000	0.000000	0.000000	1.000000	nan	0.000000
Saudi Arabia	0.428571	0.333333	0.333333	0.666667	0.200000	0.000000	0.166667
Seychelles	nan	nan	nan	nan	nan	nan	1.000000
Singapore	0.232394	0.268156	0.313043	0.330645	0.250000	0.317073	0.297468
Slovakia	nan	0.000000	nan	nan	nan	nan	nan
Slovenia	0.000000	nan	nan	nan	0.000000	nan	nan
Solomon Islands	0.000000	nan	nan	nan	nan	0.000000	nan
South Africa	nan	0.000000	nan	nan	0.000000	nan	nan
South Korea	0.086735	0.105769	0.115839	0.096539	0.099548	0.089013	0.120879
Spain	0.000000	0.000000	0.000000	0.000000	0.000000	0.166667	0.000000
Sri Lanka	0.000000	0.083333	0.000000	0.000000	0.000000	0.000000	0.000000
Svalbard & Jan Mayen	nan	nan	0.000000	nan	nan	nan	nan
Sweden	0.000000	0.666667	nan	0.000000	0.000000	0.666667	0.500000
Switzerland	0.000000	0.000000	0.000000	nan	0.200000	0.250000	0.250000
Taiwan	0.102837	0.095101	0.134199	0.097959	0.093750	0.093168	0.145763
Tanzania	nan	nan	nan	nan	nan	nan	0.000000
Thailand	0.243802	0.239617	0.220408	0.195195	0.225681	0.226974	0.280936
Timor-Leste	0.000000	nan	nan	nan	nan	nan	0.000000
Tonga	nan	nan	nan	nan	nan	nan	0.000000

flight_day	Fri	Mon	Sat	Sun	Thu	Tue	Wed
booking_origin							
Tunisia	nan	nan	0.000000	0.000000	nan	nan	nan
Turkey	nan	nan	0.000000	0.000000	0.000000	0.500000	0.000000
Ukraine	0.000000	nan	0.000000	0.000000	nan	0.000000	nan
United Arab Emirates	0.000000	0.200000	0.000000	0.000000	0.000000	0.166667	0.166667
United Kingdom	0.190476	0.037037	0.058824	0.125000	0.129032	0.085714	0.230769
United States	0.152542	0.117647	0.157895	0.230769	0.263889	0.179487	0.174419
Vanuatu	nan	1.000000	nan	nan	nan	nan	nan
Vietnam	0.180000	0.376623	0.352941	0.175000	0.301887	0.300000	0.309091

Observations:

- During the Exploratory Data Analysis (EDA) phase, we identified several interesting insights.
- The distribution plot revealed that the majority of purchase leads fall within the range of 200 to 400.
- Australia recorded the highest number of purchase leads, followed by Malaysia in second place.
- Only 15% of the leads resulted in ticket bookings, indicating that 85% did not convert.
- Countries like Jordan and New Zealand had a higher average flight duration compared to others.
- The pie chart showed that 92% of bookings were completed through the internet, while 8% were completed via mobile devices.

Machine Learning Modeling

- Firstly, we utilized the label encoder to convert categorical columns into numerical values, enabling us to work with these features in our machine learning models.
- Next, we divided the data into independent and dependent variables. To ensure uniformity in the data, we applied normalization techniques.
- Subsequently, we split the data into training and testing sets, reserving 25% of the data for testing purposes, thus allowing us to evaluate the model's performance on unseen data.
- We then proceeded to create a function for machine learning modeling. With this function, we could apply various classification algorithms to the data and compare their performance to determine the most suitable model for our task.

In [25]:
```
# Import the all required libraries for machine learning modeling
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
```

In [26]:
```
# Covert the categorical data into the numerical using the labelencoder
for col in data.select_dtypes(include='object').columns:
    label_encoder=LabelEncoder()
    label_encoder.fit(data[col].unique())
    data[col]=label_encoder.transform(data[col])
```

In [27]:
```
#Divided the data into independent and dependent variables
X=data.drop(['booking_complete','purchase_lead','route'],axis=1)
y=data['booking_complete']
# Scalling the data
scaler=StandardScaler()
X=scaler.fit_transform(X)
#Split the data into train and test data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=120)
```

In [28]:
```
# Create a function for machine learning model
def model_bulding(model,X_train,X_test,y_train,y_test):
    print(f'Name of the {model}')
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    score=accuracy_score(y_test,y_pred)
    print(f'\nThe Accuracy_score is {score*100:.2f}')
    print(f'\n{classification_report(y_test,y_pred)}')
    print(f'\n{confusion_matrix(y_test,y_pred)}')
    print('__'*20)
```

In [29]:
```
# Create all model into the dictionary formate
models={
    'logistic':LogisticRegression(),
    'decisiontree':DecisionTreeClassifier(),
    'radnom':RandomForestClassifier(),
    'Kneighbors':KNeighborsClassifier(),
    'xGB':XGBClassifier(),
    'Cat':CatBoostClassifier(iterations=1)

}
```

In [30]:
```
# Then iterating through the each model and visualize the classification and confusior
for i in range(len(models)):
    model=list(models.values())[i]
    name=list(models.keys())[i]
    model_bulding(model,X_train,X_test,y_train,y_test)
```

```
Name of the LogisticRegression()

The Accuracy_score is 84.95

              precision    recall  f1-score   support

           0       0.85      1.00      0.92      8374
           1       0.00      0.00      0.00      1483

    accuracy                           0.85      9857
   macro avg       0.42      0.50      0.46      9857
weighted avg       0.72      0.85      0.78      9857


[[8374    0]
 [1483    0]]
```

```
Name of the DecisionTreeClassifier()

The Accuracy_score is 76.90

              precision    recall  f1-score   support

           0       0.87      0.85      0.86      8374
           1       0.26      0.30      0.28      1483

    accuracy                           0.77      9857
   macro avg       0.57      0.58      0.57      9857
weighted avg       0.78      0.77      0.77      9857


[[7137 1237]
 [1040  443]]
```

```
Name of the RandomForestClassifier()

The Accuracy_score is 83.82

              precision    recall  f1-score   support

           0       0.86      0.97      0.91      8374
           1       0.38      0.12      0.18      1483

    accuracy                           0.84      9857
   macro avg       0.62      0.54      0.54      9857
weighted avg       0.79      0.84      0.80      9857


[[8088  286]
 [1309  174]]
```

```
Name of the KNeighborsClassifier()

The Accuracy_score is 83.44

              precision    recall  f1-score   support

           0       0.86      0.96      0.91      8374
           1       0.35      0.12      0.18      1483
```

```
    accuracy                           0.83      9857
   macro avg       0.61      0.54      0.54      9857
weighted avg       0.78      0.83      0.80      9857


[[8047  327]
 [1305  178]]
```

```
Name of the XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)

The Accuracy_score is 85.06

              precision    recall  f1-score   support

           0       0.86      0.98      0.92      8374
           1       0.52      0.10      0.16      1483

    accuracy                           0.85      9857
   macro avg       0.69      0.54      0.54      9857
weighted avg       0.81      0.85      0.80      9857


[[8239  135]
 [1338  145]]
```

```
Name of the <catboost.core.CatBoostClassifier object at 0x0000015DA8266580>
Learning rate set to 0.5
0:      learn: 0.4976508        total: 161ms    remaining: 0us

The Accuracy_score is 84.95

              precision    recall  f1-score   support

           0       0.85      1.00      0.92      8374
           1       0.00      0.00      0.00      1483

    accuracy                           0.85      9857
   macro avg       0.42      0.50      0.46      9857
weighted avg       0.72      0.85      0.78      9857


[[8374    0]
 [1483    0]]
```

About the project:

- Some interesting insights are observed in the dataset. It appears that the majority of people, approximately 91%, did not book their tickets, while only 9% of the people showed interest in booking. This highlights the need to enhance the quality of extra services such as luggage handling, specific seat selection, and meal options, as these factors seem to have a significant impact on customers' decisions. Additionally, we could consider incorporating online advertisements to attract more bookings.

- Furthermore, it is notable that most of the trips are round trips. To capitalize on this trend, we should focus on promoting and improving the experience for round trips while also considering offering advertising and incentives for one-way and circular trips. By understanding these patterns and preferences, we can tailor our marketing strategies to target specific trip types and attract more customers to book their tickets.