Importing the libraries required

In [108…

```
# Importing the basic libraries we will require for the project

# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# Libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

# Importing the Machine Learning models we require from Scikit-Learn
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier

# Importing the other functions we may require from Scikit-Learn
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.impute import SimpleImputer

# To get diferent metric scores
from sklearn.metrics import confusion_matrix,classification_report,roc_auc_score,plot_cc

# Code to ignore warnings from function usage
import warnings;
import numpy as np
warnings.filterwarnings('ignore')
```

Loading the 2 datasets from Google Drive

In [109…

```
# Loading the dataset - sheet_name parameter is used if there are Basicple tabs in the e

from google.colab import drive
drive.mount('/content/drive')

# TODO: ANALISAR SE DEVEMOS USAR 2 DATAFRAMES OU AGREGAR PRIMEIRO NUM SÓ

data = pd.read_excel('/content/drive/MyDrive/Outros/GL Hackathon/TravelSurveydata_train.

data_test = pd.read_excel('/content/drive/MyDrive/Outros/GL Hackathon/TravelSurveydata_t
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
t("/content/drive", force_remount=True).
```

Overview of the dataset

View the first and last 5 rows of the dataset

Let's **view the first few rows and last few rows** of the dataset in order to understand its structure a little better.

We will use the head() and tail() methods from Pandas to do this.

In [110…
```
data.head()
```

Out[110]:

	ID	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_Delay_in
0	98800001	Female	Loyal Customer	52.0	NaN	Business	272	
1	98800002	Male	Loyal Customer	48.0	Personal Travel	Eco	2200	
2	98800003	Female	Loyal Customer	43.0	Business Travel	Business	1061	
3	98800004	Female	Loyal Customer	44.0	Business Travel	Business	780	
4	98800005	Female	Loyal Customer	50.0	Business Travel	Business	1981	

In [111…
```
data.tail()
```

Out[111]:

	ID	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_De
94374	98894375	Male	Loyal Customer	32.0	Business Travel	Business	1357	
94375	98894376	Male	Loyal Customer	44.0	Business Travel	Business	592	
94376	98894377	Male	NaN	63.0	Business Travel	Business	2794	
94377	98894378	Male	Loyal Customer	16.0	Personal Travel	Eco	2744	
94378	98894379	Male	Loyal Customer	54.0	NaN	Eco	2107	

Understand the shape of the dataset

In [112…
```
print(data.shape)
print(data_test.shape)
```

```
(94379, 25)
(35602, 24)
```

- The dataset has 94379 rows and 25 columns.

Check the data types of the columns for the dataset

In [113…
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 94379 entries, 0 to 94378
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ID                       94379 non-null  int64  
 1   Gender                   94302 non-null  object 
 2   Customer_Type            85428 non-null  object 
 3   Age                      94346 non-null  float64
 4   Type_Travel              85153 non-null  object 
 5   Travel_Class             94379 non-null  object 
 6   Travel_Distance          94379 non-null  int64  
 7   Departure_Delay_in_Mins  94322 non-null  float64
 8   Arrival_Delay_in_Mins    94022 non-null  float64
 9   Seat_Comfort             94318 non-null  object 
 10  Seat_Class               94379 non-null  object 
 11  Arrival_Time_Convenient  85449 non-null  object 
 12  Catering                 85638 non-null  object 
 13  Platform_Location        94349 non-null  object 
 14  Onboard_Wifi_Service     94349 non-null  object 
 15  Onboard_Entertainment    94361 non-null  object 
 16  Online_Support           94288 non-null  object 
 17  Ease_of_Online_Booking   94306 non-null  object 
 18  Onboard_Service          86778 non-null  object 
 19  Legroom                  94289 non-null  object 
 20  Baggage_Handling         94237 non-null  object 
 21  CheckIn_Service          94302 non-null  object 
 22  Cleanliness              94373 non-null  object 
 23  Online_Boarding          94373 non-null  object 
 24  Overall_Experience       94379 non-null  int64  
dtypes: float64(3), int64(3), object(19)
memory usage: 18.0+ MB
```

Check the percentage of missing values in each column

In [114…

```
pd.DataFrame(data={'% of Missing Values':round(data.isna().sum()/data.isna().count()*100
```

Out[114]:

	% of Missing Values
Type_Travel	9.78
Customer_Type	9.48
Arrival_Time_Convenient	9.46
Catering	9.26
Onboard_Service	8.05
Arrival_Delay_in_Mins	0.38
Baggage_Handling	0.15
Online_Support	0.10
Legroom	0.10
Gender	0.08
Ease_of_Online_Booking	0.08
CheckIn_Service	0.08
Seat_Comfort	0.06
Departure_Delay_in_Mins	0.06
Platform_Location	0.03
Onboard_Wifi_Service	0.03
Age	0.03
Onboard_Entertainment	0.02
Cleanliness	0.01
Online_Boarding	0.01
ID	0.00
Seat_Class	0.00
Travel_Distance	0.00
Travel_Class	0.00
Overall_Experience	0.00

Check the number of unique values in each column

In [115…

```
data.nunique()
```

Out[115]:
```
ID                         94379
Gender                         2
Customer_Type                  2
Age                           75
Type_Travel                    2
Travel_Class                   2
Travel_Distance             5210
Departure_Delay_in_Mins      437
Arrival_Delay_in_Mins        434
Seat_Comfort                   6
Seat_Class                     2
Arrival_Time_Convenient        6
Catering                       6
Platform_Location              6
Onboard_Wifi_Service           6
Onboard_Entertainment          6
Online_Support                 6
Ease_of_Online_Booking         6
Onboard_Service                6
Legroom                        6
Baggage_Handling               5
CheckIn_Service                6
Cleanliness                    6
Online_Boarding                6
Overall_Experience             2
dtype: int64
```

- We can drop the column - CustomerID as it is unique for each customer and will not add value to the model.
- Most of the variables are categorical except - Age, duration of pitch, monthly income, and number of trips of customers.

Dropping the unique values column

In [116…
```
# Dropping CustomerID column
data.drop(columns='ID',inplace=True)
```

Question 1: Check the summary statistics of the dataset and write your observations (2 Marks)

Let's check the statistical summary of the data.

In [117…
```
data.describe().T
```

Out[117]:

	count	mean	std	min	25%	50%	75%	max
Age	94346.0	39.419647	15.116632	7.0	27.0	40.0	51.0	85.0
Travel_Distance	94379.0	1978.888185	1027.961019	50.0	1359.0	1923.0	2538.0	6951.0
Departure_Delay_in_Mins	94322.0	14.647092	38.138781	0.0	0.0	0.0	12.0	1592.0
Arrival_Delay_in_Mins	94022.0	15.005222	38.439409	0.0	0.0	0.0	13.0	1584.0
Overall_Experience	94379.0	0.546658	0.497821	0.0	0.0	1.0	1.0	1.0

Check the count of each unique category in each of the categorical variables.

In [118…
```
# Making a list of all catrgorical variables
# TODO: ALTERAR OS NOMES DAS VARIÁVEIS
```

```
cat_col=[
    'Gender','Customer_Type','Type_Travel','Travel_Class','Seat_Comfort','Seat_Class','A
    'Catering','Platform_Location','Onboard_Wifi_Service','Onboard_Entertainment','Onlin
    'Onboard_Service','Legroom','Baggage_Handling','CheckIn_Service','Cleanliness','Onli

# Printing number of count of each unique value in each column
for column in cat_col:
    print(data[column].value_counts())
    print('-'*50)
```

```
Female    47815
Male      46487
Name: Gender, dtype: int64
--------------------------------------------------
Loyal Customer       69823
Disloyal Customer    15605
Name: Customer_Type, dtype: int64
--------------------------------------------------
Business Travel    58617
Personal Travel    26536
Name: Type_Travel, dtype: int64
--------------------------------------------------
Eco         49342
Business    45037
Name: Travel_Class, dtype: int64
--------------------------------------------------
Acceptable           21158
Needs Improvement    20946
Good                 20595
Poor                 15185
Excellent            12971
Extremely Poor        3463
Name: Seat_Comfort, dtype: int64
--------------------------------------------------
Green Car    47435
Ordinary     46944
Name: Seat_Class, dtype: int64
--------------------------------------------------
Good                 19574
Excellent            17684
Acceptable           15177
Needs Improvement    14990
Poor                 13692
Extremely Poor        4332
Name: Arrival_Time_Convenient, dtype: int64
--------------------------------------------------
Acceptable           18468
Needs Improvement    17978
Good                 17969
Poor                 13858
Excellent            13455
Extremely Poor        3910
Name: Catering, dtype: int64
--------------------------------------------------
Manageable           24173
Convenient           21912
Needs Improvement    17832
Inconvenient         16449
Very Convenient      13981
Very Inconvenient        2
Name: Platform_Location, dtype: int64
--------------------------------------------------
Good                 22835
Excellent            20968
Acceptable           20118
Needs Improvement    19596
Poor                 10741
Extremely Poor          91
Name: Onboard_Wifi_Service, dtype: int64
--------------------------------------------------
Good                 30446
Excellent            21644
Acceptable           17560
Needs Improvement    13926
Poor                  8641
Extremely Poor        2144
Name: Onboard_Entertainment, dtype: int64
```

```
--------------------------------------------------
Good                 30016
Excellent            25894
Acceptable           15702
Needs Improvement    12508
Poor                 10167
Extremely Poor           1
Name: Online_Support, dtype: int64
--------------------------------------------------
Good                 28909
Excellent            24744
Acceptable           16390
Needs Improvement    14479
Poor                  9768
Extremely Poor          16
Name: Ease_of_Online_Booking, dtype: int64
--------------------------------------------------
Good                 27265
Excellent            21272
Acceptable           18071
Needs Improvement    11390
Poor                  8776
Extremely Poor           4
Name: Onboard_Service, dtype: int64
--------------------------------------------------
Good                 28870
Excellent            24832
Acceptable           16384
Needs Improvement    15753
Poor                  8110
Extremely Poor         340
Name: Legroom, dtype: int64
--------------------------------------------------
Good                 34944
Excellent            26003
Acceptable           17767
Needs Improvement     9759
Poor                  5764
Name: Baggage_Handling, dtype: int64
--------------------------------------------------
Good                 26502
Acceptable           25803
Excellent            19641
Needs Improvement    11218
Poor                 11137
Extremely Poor           1
Name: CheckIn_Service, dtype: int64
--------------------------------------------------
Good                 35427
Excellent            26053
Acceptable           17449
Needs Improvement     9806
Poor                  5633
Extremely Poor           5
Name: Cleanliness, dtype: int64
--------------------------------------------------
Good                 25533
Acceptable           22475
Excellent            21742
Needs Improvement    13451
Poor                 11160
Extremely Poor          12
Name: Online_Boarding, dtype: int64
--------------------------------------------------
```

In [119…

```python
# Converting the data type of each categorical variable to 'category'
for column in cat_col:
```

```
    data[column]=data[column].astype('category')
```

In [120…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 94379 entries, 0 to 94378
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype   
---  ------                   --------------  -----   
 0   Gender                   94302 non-null  category
 1   Customer_Type            85428 non-null  category
 2   Age                      94346 non-null  float64 
 3   Type_Travel              85153 non-null  category
 4   Travel_Class             94379 non-null  category
 5   Travel_Distance          94379 non-null  int64   
 6   Departure_Delay_in_Mins  94322 non-null  float64 
 7   Arrival_Delay_in_Mins    94022 non-null  float64 
 8   Seat_Comfort             94318 non-null  category
 9   Seat_Class               94379 non-null  category
 10  Arrival_Time_Convenient  85449 non-null  category
 11  Catering                 85638 non-null  category
 12  Platform_Location        94349 non-null  category
 13  Onboard_Wifi_Service     94349 non-null  category
 14  Onboard_Entertainment    94361 non-null  category
 15  Online_Support           94288 non-null  category
 16  Ease_of_Online_Booking   94306 non-null  category
 17  Onboard_Service          86778 non-null  category
 18  Legroom                  94289 non-null  category
 19  Baggage_Handling         94237 non-null  category
 20  CheckIn_Service          94302 non-null  category
 21  Cleanliness              94373 non-null  category
 22  Online_Boarding          94373 non-null  category
 23  Overall_Experience       94379 non-null  int64   
dtypes: category(19), float64(3), int64(2)
memory usage: 5.3 MB
```

In [121…

```
df = data.copy()
```

Exploratory Data Analysis

Question 2: Univariate Analysis

TODO: Aqui é necessário fazer atualização das variáveis para as do novo dataset

Let's explore these variables in some more depth by observing their distributions.

We will first define a **hist_box() function** that provides both a boxplot and a histogram in the same visual, with which we can perform univariate analysis on the columns of this dataset.

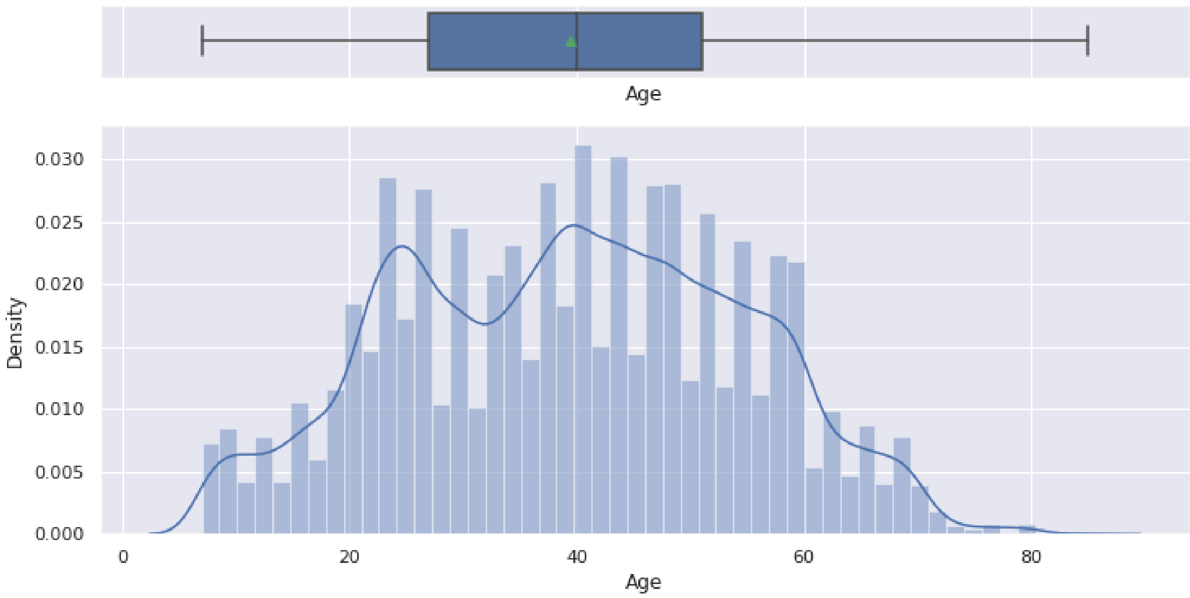
In [122…

```
# Defining the hist_box() function
def hist_box(data,col):
  f, (ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={'height_ratios': (0.1
  # Adding a graph in each part
  sns.boxplot(data[col], ax=ax_box, showmeans=True)
  sns.distplot(data[col], ax=ax_hist)
  plt.show()
```

Question 2.1: Plot the histogram and box plot for the variable `Age` using the hist_box function provided and write your insights. (1 Mark)

In [123…

```
# TODO: ALTERAR TODAS AS VARIÁVEIS DESTE BLOCO
hist_box(df, "Age")
```



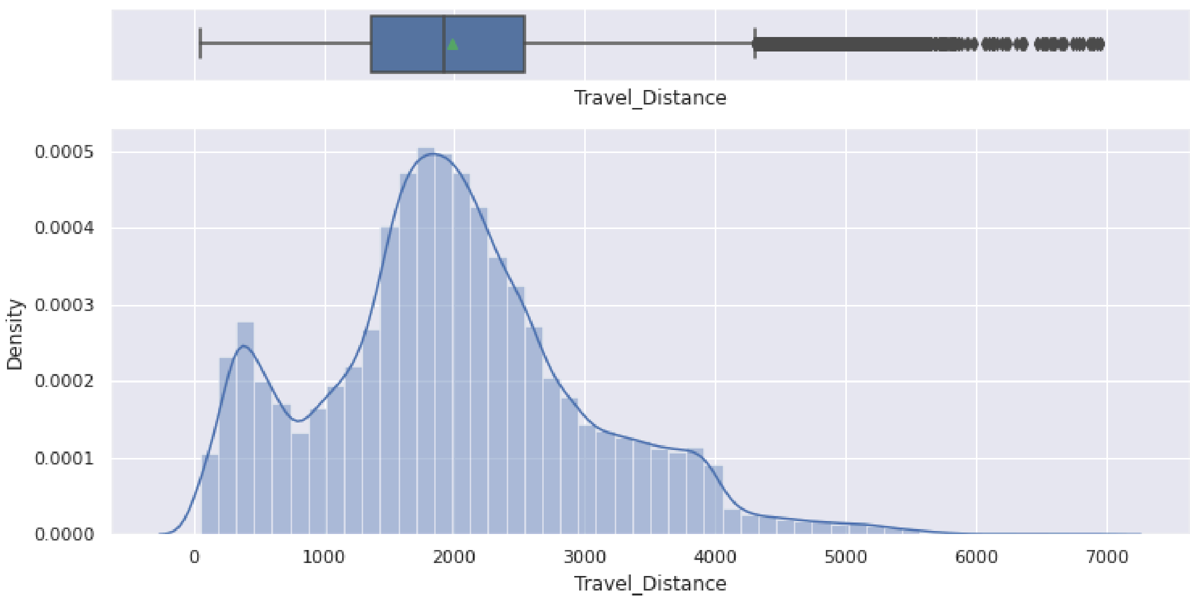
Write your Answer here :

- Age distribution looks approximately normally distributed.
- The boxplot for the age column confirms that there are no outliers for this variable
- Age can be an important variable while targeting customers for the tourism package. We will further explore this in bivariate analysis.

Question 2.2: Plot the histogram and box plot for the variable `Duration of Pitch` using the hist_box function provided and write your insights. (1 Mark)

In [124…

```
hist_box(df, 'Travel_Distance')
```



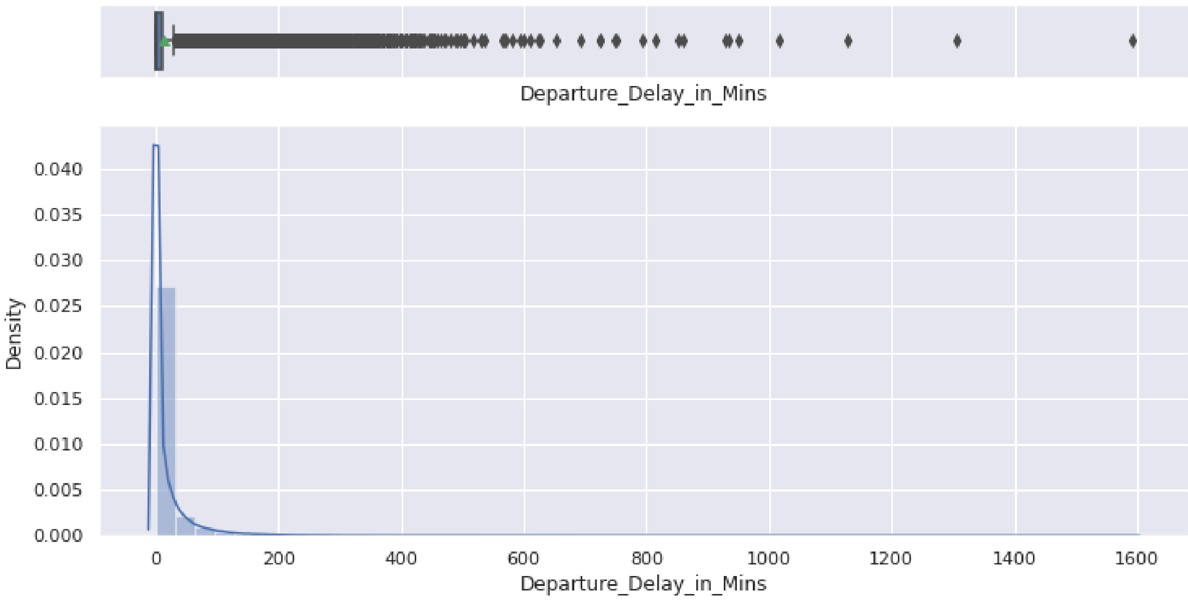
Write your Answer here :

- The distribution for the duration of pitch is right-skewed.
- The duration of the pitch for most of the customers is less than 20 minutes.
- There are some observations that can be considered as outliers as they are very far from the upper whisker in the boxplot. Let's check how many such extreme values are there.

- We can see that there are just two observations which can be considered as outliers.

Lets plot the histogram and box plot for the variable `Monthly Income` using the hist_box function

In [125…
```
hist_box(df, 'Departure_Delay_in_Mins')
```



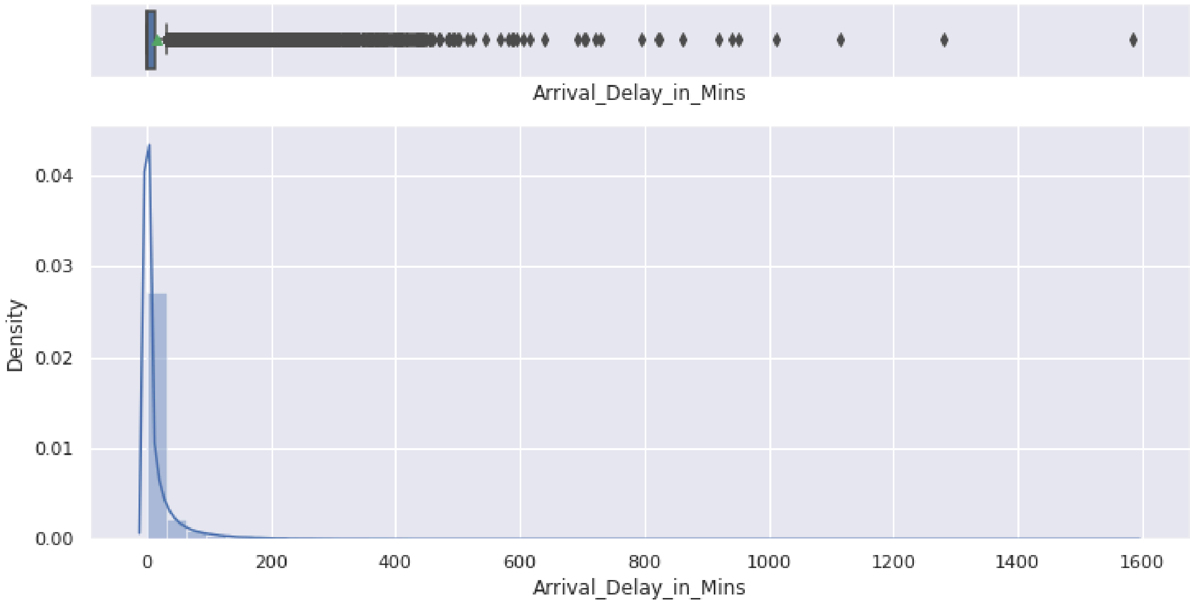
- The distribution for monthly income shows that most of the values lie between 20,000 to 40,000.
- Income is one of the important factors to consider while approaching a customer with a certain package. We can explore this further in bivariate analysis.
- There are some observations on the left and some observations on the right of the boxplot which can be considered as outliers. Let's check how many such extreme values are there.

In [126…
```
# df[(df.MonthlyIncome>40000) | (df.MonthlyIncome<12000)]
```

- There are just four such observations which can be considered as outliers.

Lets plot the histogram and box plot for the variable `Number of Trips` using the hist_box function

In [127…
```
hist_box(df,'Arrival_Delay_in_Mins')
```



- The distribution for the number of trips is right-skewed
- Boxplot shows that the number of trips has some outliers at the right end. Let's check how many such extreme values are there.

:**Removing these outliers form duration of pitch, monthly income, and number of trips.**

In [128…

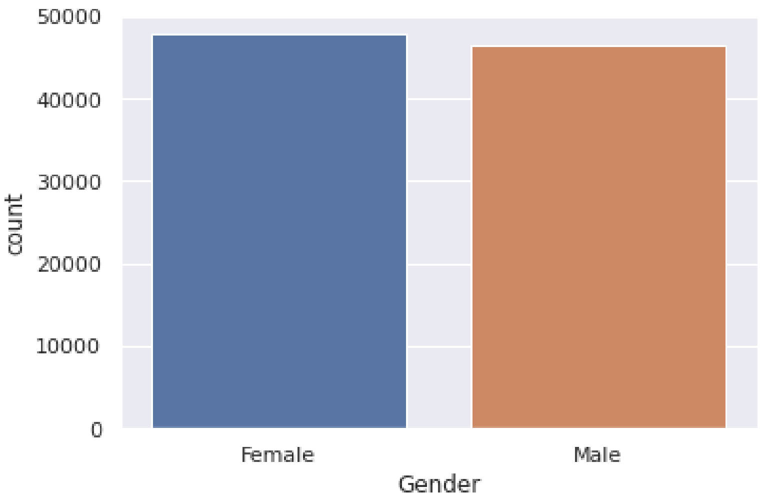
```
## NÃO VAMOS PARA JÁ RETIRAR OS OUTLIERS PORQUE ESTÃO EM VARIÁVEIS MUITO IMPORTANTES PAR

# Dropping observaions with duration of pitch greater than 40. There are just 2 such obs
#df.drop(index=df[df.DurationOfPitch>37].index,inplace=True)

# Dropping observation with monthly income less than 12000 or greater than 40000. There
#df.drop(index=df[(df.MonthlyIncome>40000) | (df.MonthlyIncome<12000)].index,inplace=Tru

# Dropping observations with number of trips greater than 8. There are just 4 such obser
#df.drop(index=df[df.NumberOfTrips>10].index,inplace=True)
```

Let's understand the distribution of the categorical variables

Number of Person Visiting

In [129…

```
sns.countplot(df['Gender'])
plt.show()
```



In [130…

```
df['Gender'].value_counts(normalize=True)
```

Out[130]:

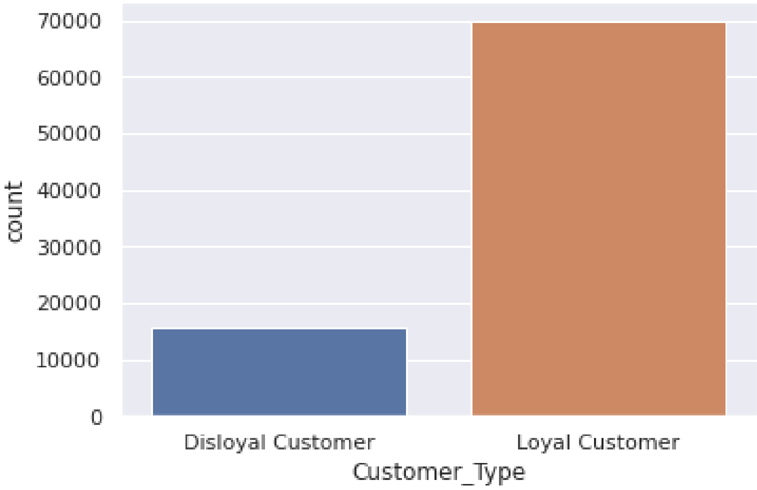
```
Female    0.507041
Male      0.492959
Name: Gender, dtype: float64
```

- Most customers have 3 persons who are visiting with them. This can be because most people like to travel with family.
- As mentioned earlier, there are just 3 observations where the number of persons visiting with the customers are 5 i.e. 0.1%.

Occupation

In [131…

```
sns.countplot(df['Customer_Type'])
plt.show()
```



In [132…

```
df['Customer_Type'].value_counts(normalize=True)
```

Out[132]:

```
Loyal Customer       0.817332
Disloyal Customer    0.182668
Name: Customer_Type, dtype: float64
```

- The majority of customers i.e. 91% are either salaried or owns a small business.
- As mentioned earlier, the freelancer category has only 2 observations.

City Tier

In [133…

```
sns.countplot(df['Type_Travel'])
plt.show()
```

In [134…

```
df['Type_Travel'].value_counts(normalize=True)
```

Out[134]:

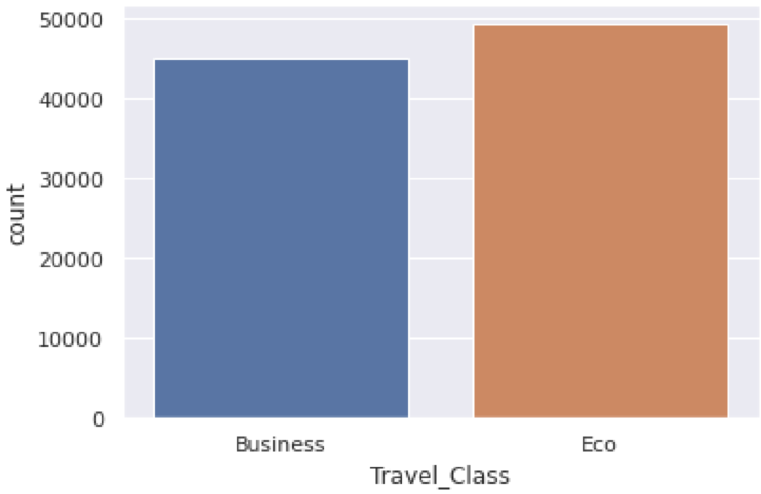
```
Business Travel    0.688373
Personal Travel    0.311627
Name: Type_Travel, dtype: float64
```

- Most of the customers i.e. approx 65% are from tier 1 cities. This can be because of better living standards and exposure as compared to tier 2 and tier 3 cities.
- Surprisingly, tier 3 cities have a much higher count than tier 2 cities. This can be because the company has less marketing in tier 2 cities.

Gender

In [135…

```
sns.countplot(df['Travel_Class'])
plt.show()
```



In [136…

```
df['Travel_Class'].value_counts(normalize=True)
```

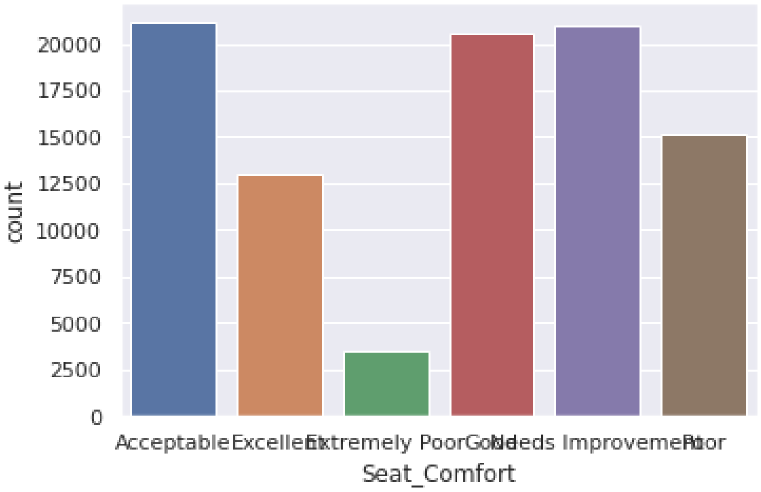
Out[136]:

```
Eco         0.522807
Business    0.477193
Name: Travel_Class, dtype: float64
```

- Male customers are more than the number of female customers
- There are approx 60% male customers as compared to 40% female customers
- This might be because males do the booking/inquiry when traveling with females which imply that males are the direct customers of the company.

Number of Follow ups

In [137…
```
sns.countplot(df['Seat_Comfort'])
plt.show()
```



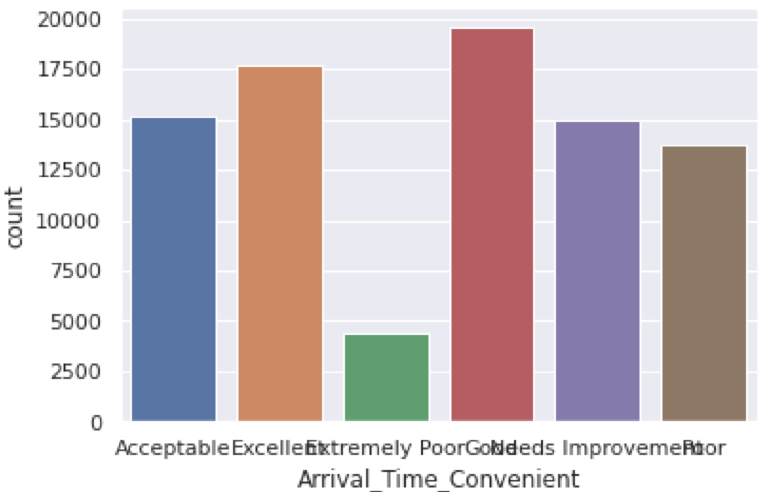
In [138…
```
df['Seat_Comfort'].value_counts(normalize=True)
```

Out[138]:
```
Acceptable           0.224326
Needs Improvement    0.222079
Good                 0.218357
Poor                 0.160998
Excellent            0.137524
Extremely Poor       0.036716
Name: Seat_Comfort, dtype: float64
```

- We can see that company usually follow-ups with 3 or 4 times with their customers
- We can explore this further and observe which number of follow-ups have more customers who buy the product.

Product Pitched

In [139…
```
sns.countplot(df['Arrival_Time_Convenient'])
plt.show()
```



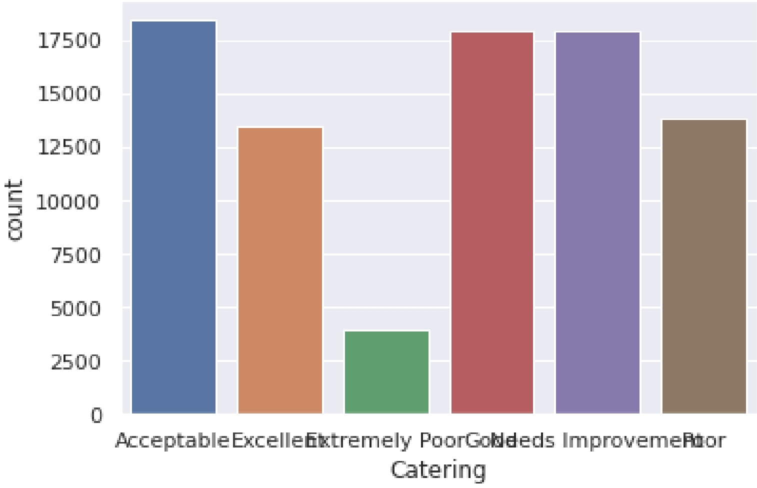
In [140…
```
df['Arrival_Time_Convenient'].value_counts(normalize=True)
```

Out[140]:
```
Good                 0.229072
Excellent            0.206954
Acceptable           0.177615
Needs Improvement    0.175426
Poor                 0.160236
Extremely Poor       0.050697
Name: Arrival_Time_Convenient, dtype: float64
```

- The company pitches Deluxe or Basic packages to their customers more than the other packages.
- This might be because the company makes more profit from Deluxe or Basic packages or these packages are less expensive, so preferred by the majority of the customers.

Type of Contact

In [141…
```
sns.countplot(df['Catering'])
plt.show()
```



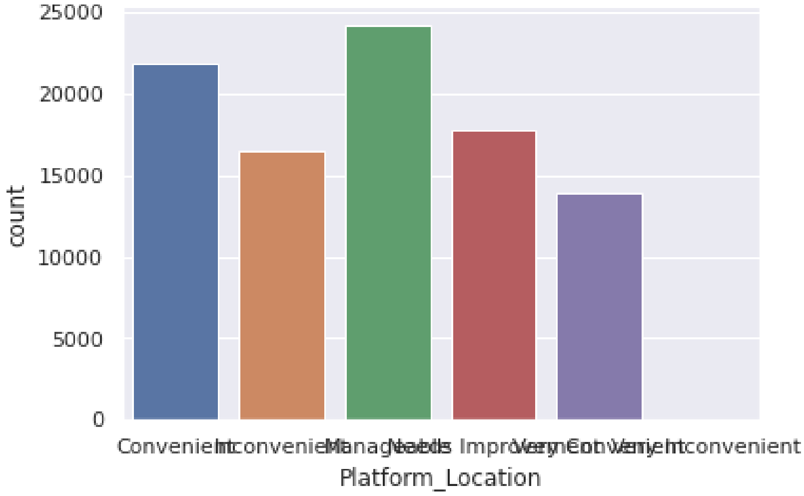
In [142…
```
df['Catering'].value_counts(normalize=True)
```

Out[142]:
```
Acceptable           0.215652
Needs Improvement    0.209930
Good                 0.209825
Poor                 0.161821
Excellent            0.157115
Extremely Poor       0.045657
Name: Catering, dtype: float64
```

- There are approx 70% of customers who reached out to the company first i.e. self-inquiry.
- This shows the positive outreach of the company as most of the inquires are initiated from the customer's end.

Designation

In [143…
```
sns.countplot(df['Platform_Location'])
plt.show()
```



In [144…

```
df['Platform_Location'].value_counts(normalize=True)
```

Out[144]:

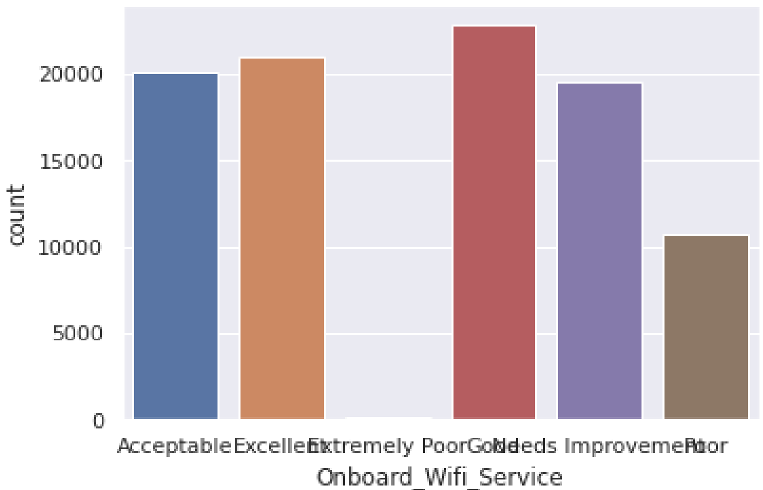
```
Manageable           0.256208
Convenient           0.232244
Needs Improvement    0.189000
Inconvenient         0.174342
Very Convenient      0.148184
Very Inconvenient    0.000021
Name: Platform_Location, dtype: float64
```

- Approx 73% of the customers are at the executive or manager level.
- We can see that the higher the position, the lesser number of observations which makes sense as executives/managers are more common than AVP/VP.

Product Taken

In [145…

```
sns.countplot(df['Onboard_Wifi_Service'])
plt.show()
```



In [146…

```
df['Onboard_Wifi_Service'].value_counts(normalize=True)
```

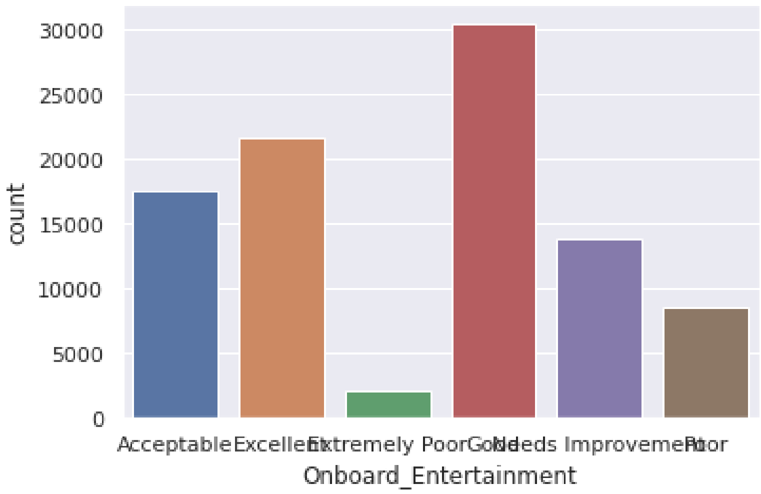
Out[146]:

```
Good                 0.242027
Excellent            0.222239
Acceptable           0.213230
Needs Improvement    0.207697
Poor                 0.113843
Extremely Poor       0.000965
Name: Onboard_Wifi_Service, dtype: float64
```

- This plot shows the distribution of both classes in the target variable is `imbalanced`.
- We only have approx 19% of customers who have purchased the product.

In [147…

```
sns.countplot(df['Onboard_Entertainment'])
plt.show()
```



In [148…

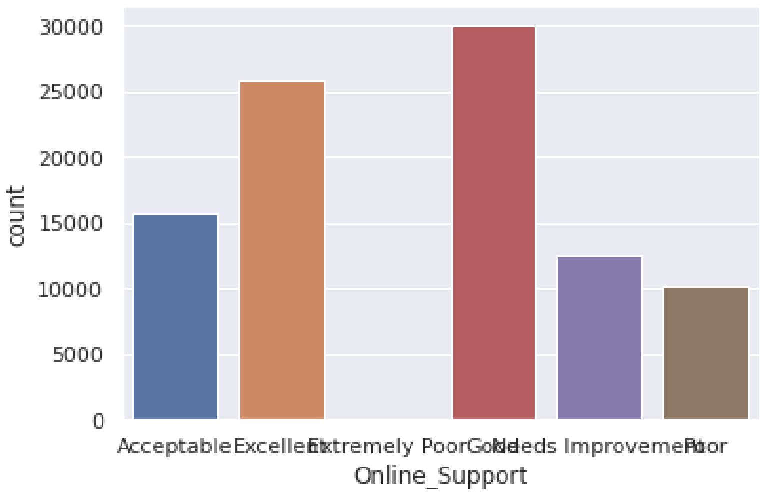
```
df['Onboard_Entertainment'].value_counts(normalize=True)
```

Out[148]:

```
Good                 0.322654
Excellent            0.229374
Acceptable           0.186094
Needs Improvement    0.147582
Poor                 0.091574
Extremely Poor       0.022721
Name: Onboard_Entertainment, dtype: float64
```

In [149…

```
sns.countplot(df['Online_Support'])
plt.show()
```



In [150…

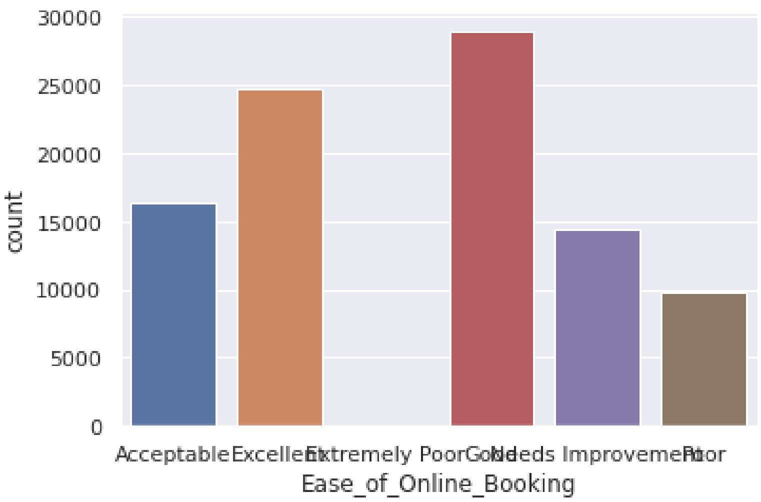
```
df['Online_Support'].value_counts(normalize=True)
```

Out[150]:

```
Good                 0.318344
Excellent            0.274627
Acceptable           0.166532
Needs Improvement    0.132657
Poor                 0.107829
Extremely Poor       0.000011
Name: Online_Support, dtype: float64
```

In [151…

```
sns.countplot(df['Ease_of_Online_Booking'])
plt.show()
```



In [152…

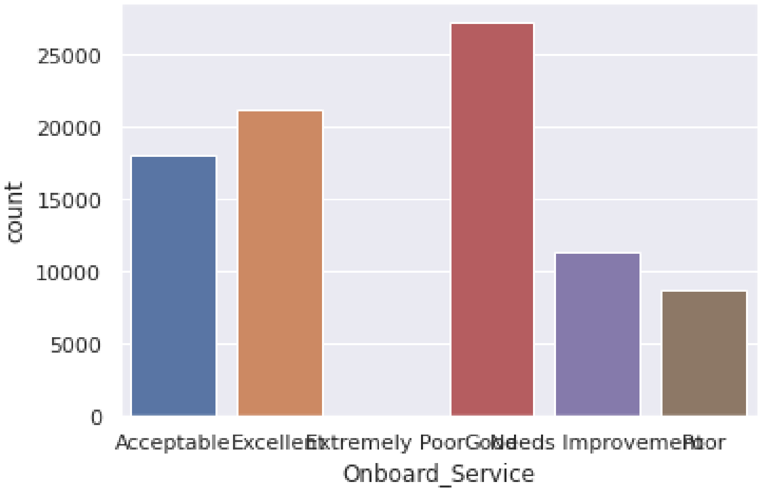
```
df['Ease_of_Online_Booking'].value_counts(normalize=True)
```

Out[152]:

```
Good                 0.306545
Excellent            0.262380
Acceptable           0.173796
Needs Improvement    0.153532
Poor                 0.103578
Extremely Poor       0.000170
Name: Ease_of_Online_Booking, dtype: float64
```

In [153…

```
sns.countplot(df['Onboard_Service'])
plt.show()
```



In [154…

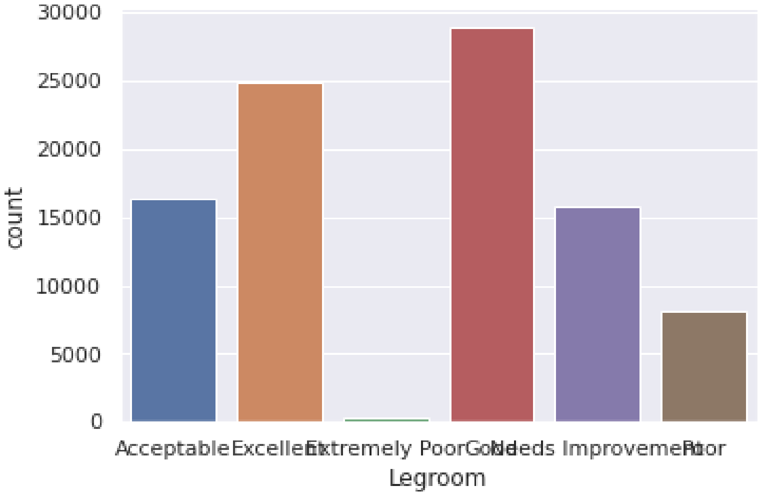
```
df['Onboard_Service'].value_counts(normalize=True)
```

Out[154]:

```
Good                 0.314193
Excellent            0.245131
Acceptable           0.208244
Needs Improvement    0.131254
Poor                 0.101132
Extremely Poor       0.000046
Name: Onboard_Service, dtype: float64
```

In [155…

```
sns.countplot(df['Legroom'])
plt.show()
```



In [156…

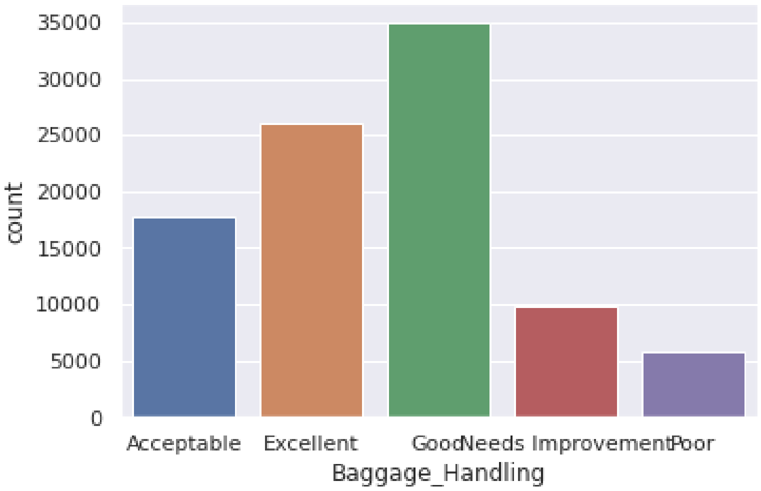
```
df['Legroom'].value_counts(normalize=True)
```

Out[156]:

```
Good                 0.306186
Excellent            0.263361
Acceptable           0.173764
Needs Improvement    0.167071
Poor                 0.086012
Extremely Poor       0.003606
Name: Legroom, dtype: float64
```

In [157…

```
sns.countplot(df['Baggage_Handling'])
plt.show()
```



In [158…

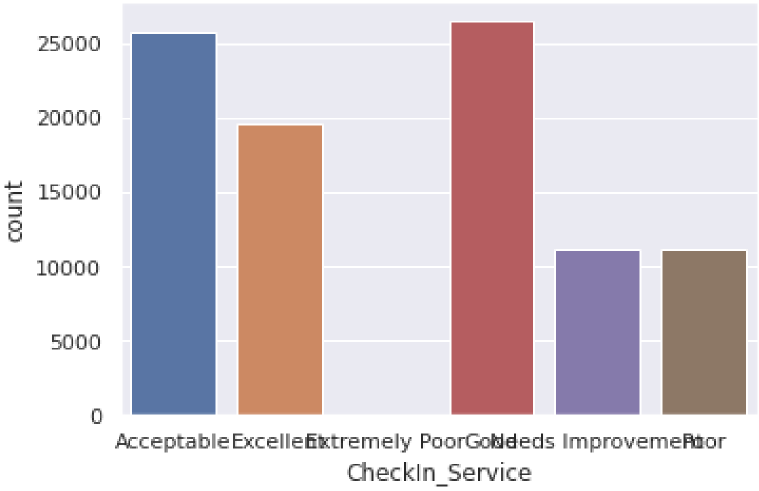
```
df['Baggage_Handling'].value_counts(normalize=True)
```

Out[158]:

```
Good                 0.370810
Excellent            0.275932
Acceptable           0.188535
Needs Improvement    0.103558
Poor                 0.061165
Name: Baggage_Handling, dtype: float64
```

In [159…

```
sns.countplot(df['CheckIn_Service'])
plt.show()
```



In [160…

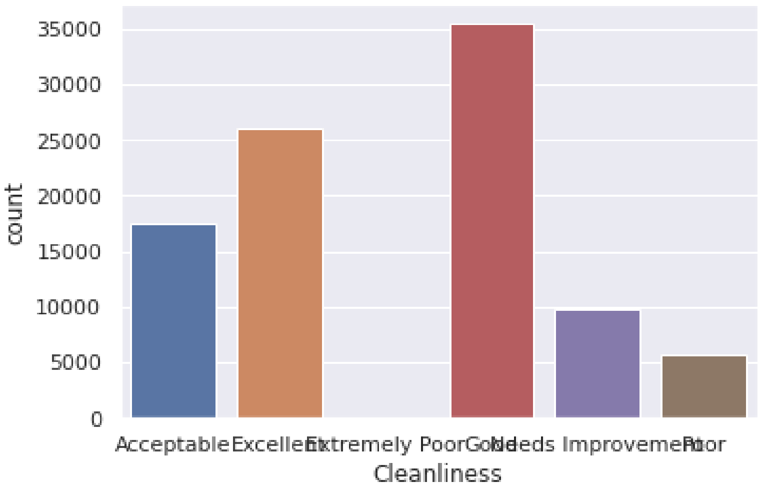
```
df['CheckIn_Service'].value_counts(normalize=True)
```

Out[160]:

```
Good                 0.281033
Acceptable           0.273621
Excellent            0.208278
Needs Improvement    0.118958
Poor                 0.118099
Extremely Poor       0.000011
Name: CheckIn_Service, dtype: float64
```

In [161…

```
sns.countplot(df['Cleanliness'])
plt.show()
```



In [162…

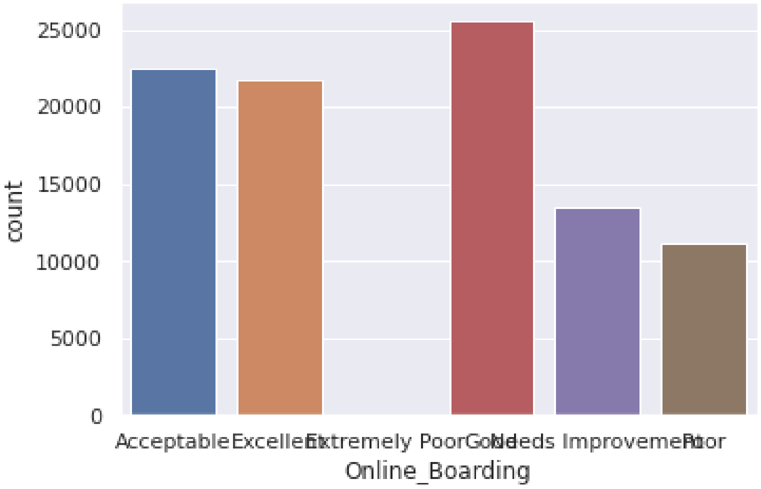
```
df['Cleanliness'].value_counts(normalize=True)
```

Out[162]:

```
Good                 0.375393
Excellent            0.276064
Acceptable           0.184894
Needs Improvement    0.103907
Poor                 0.059689
Extremely Poor       0.000053
Name: Cleanliness, dtype: float64
```

In [163…

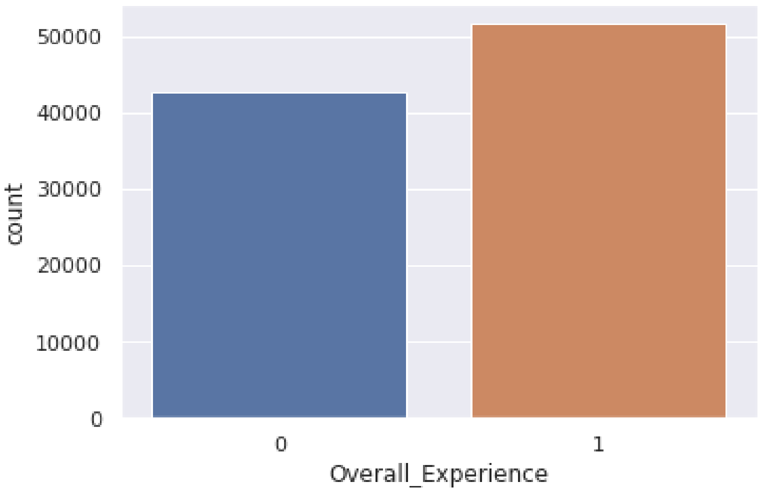
```
sns.countplot(df['Online_Boarding'])
plt.show()
```



In [164…
```
df['Online_Boarding'].value_counts(normalize=True)
```

Out[164]:
```
Good                 0.270554
Acceptable           0.238151
Excellent            0.230384
Needs Improvement    0.142530
Poor                 0.118254
Extremely Poor       0.000127
Name: Online_Boarding, dtype: float64
```

In [165…
```
sns.countplot(df['Overall_Experience'])
plt.show()
```



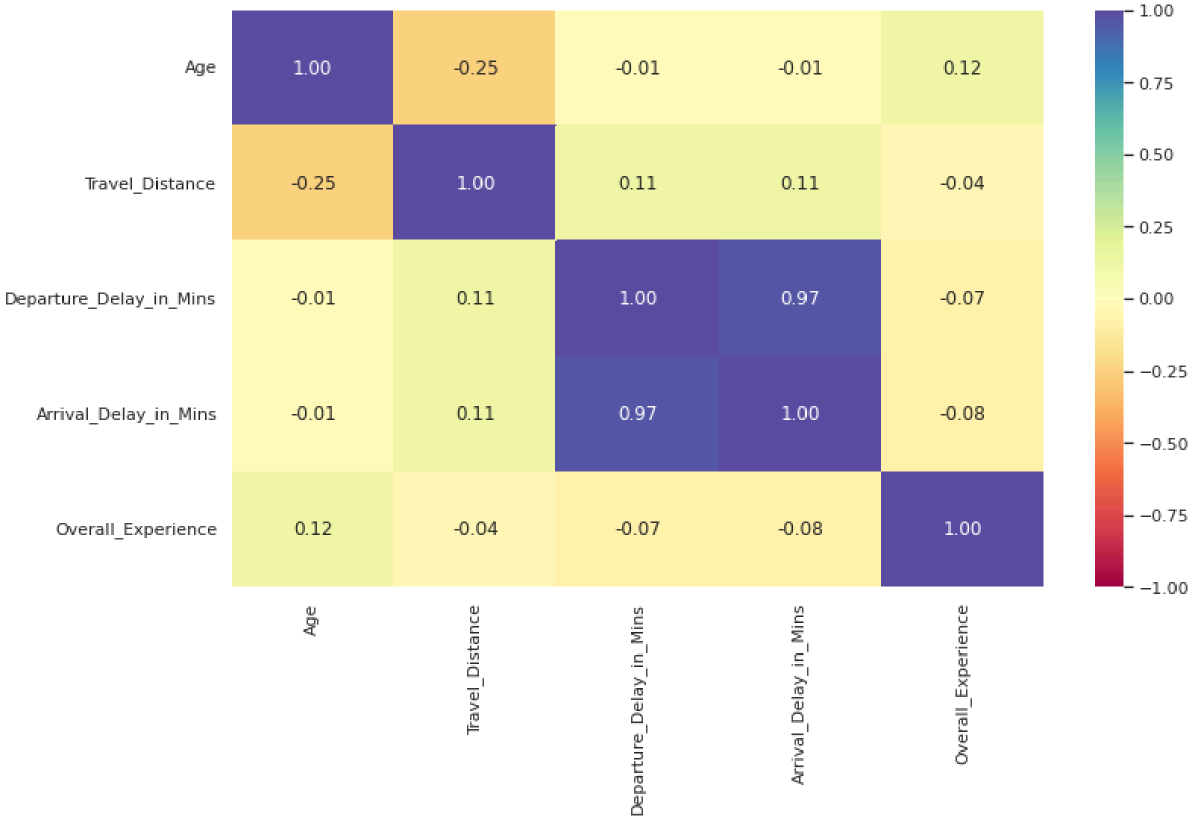
In [166…
```
df['Overall_Experience'].value_counts(normalize=True)
```

Out[166]:
```
1    0.546658
0    0.453342
Name: Overall_Experience, dtype: float64
```

Question 3: Bivariate Analysis

Question 3.1: Find and visualize the correlation matrix using a heatmap and write your observations from the plot. (2 Marks)

In [167…
```
cols_list = df.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(12, 7))
sns.heatmap(data[cols_list].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectr
plt.show()
```



We will define a **stacked barplot()** function to help analyse how the target variable varies across predictor categories.

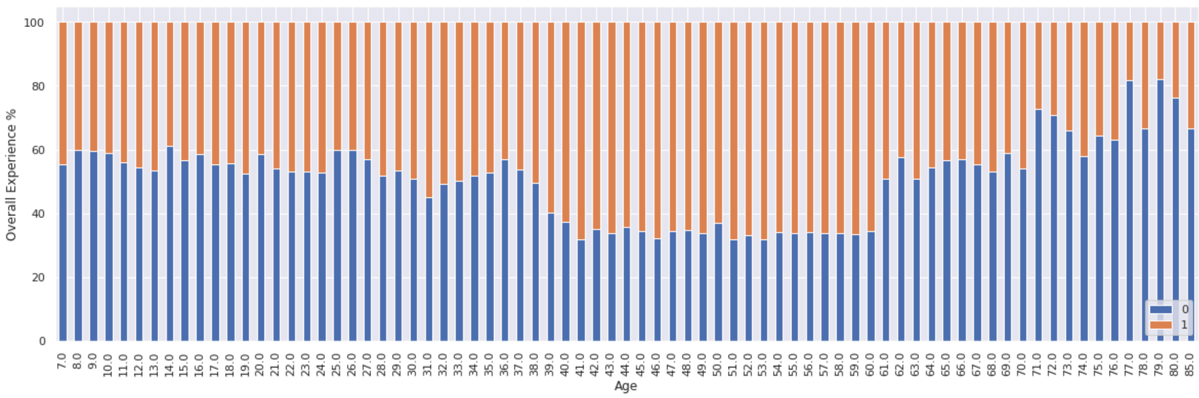
In [168…

```
# Defining the stacked_barplot() function
def stacked_barplot(data,predictor,target,figsize=(20,6)):
  (pd.crosstab(data[predictor],data[target],normalize='index')*100).plot(kind='bar',figs
  plt.legend(loc="lower right")
  plt.ylabel('Overall Experience %')
```

Question 3.2: Plot the stacked barplot for the variable `Marital Status` against the target variable `ProdTaken` using the stacked_barplot function provided and write your insights. (1 Mark)

In [169…

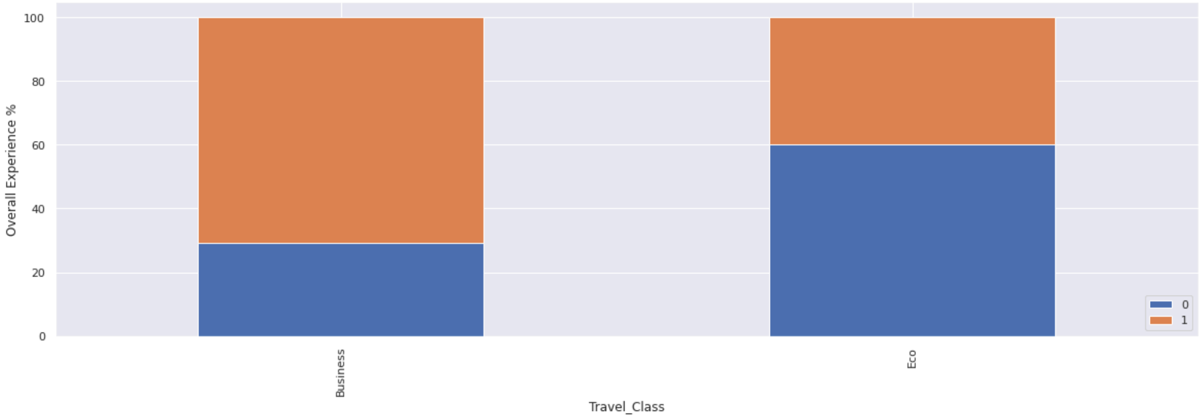
```
stacked_barplot(data, "Age", "Overall_Experience" )
```



Question 3.3: Plot the stacked barplot for the variable `ProductPitched` against the target variable `ProdTaken` using the stacked_barplot function provided and write your insights. (1 Mark)

In [170…

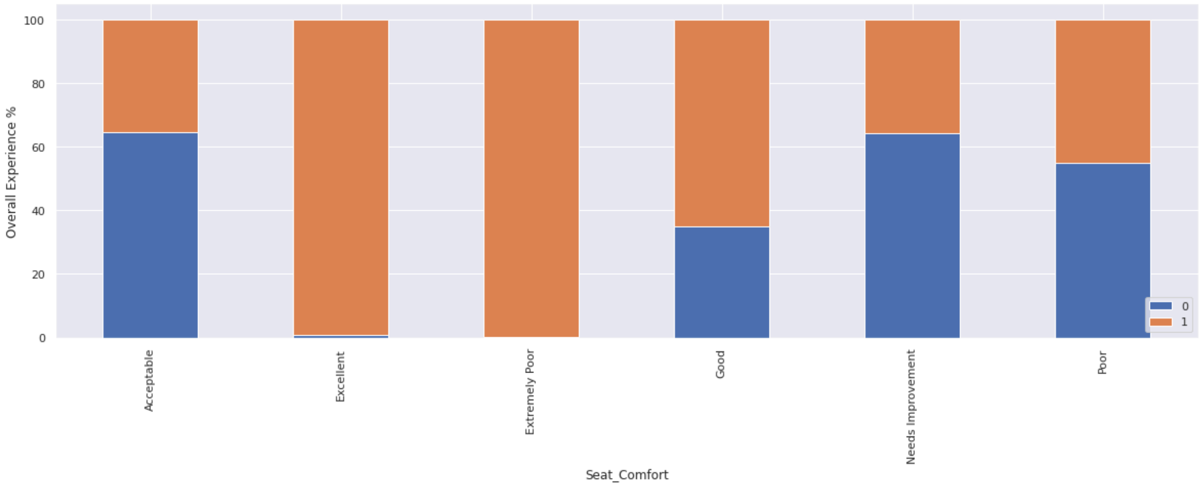
```
stacked_barplot(df, "Travel_Class", "Overall_Experience" )
```



Let's plot the stacked barplot for the variable `Passport` against the target variable `ProdTaken` using the stacked_barplot function.

In [171…

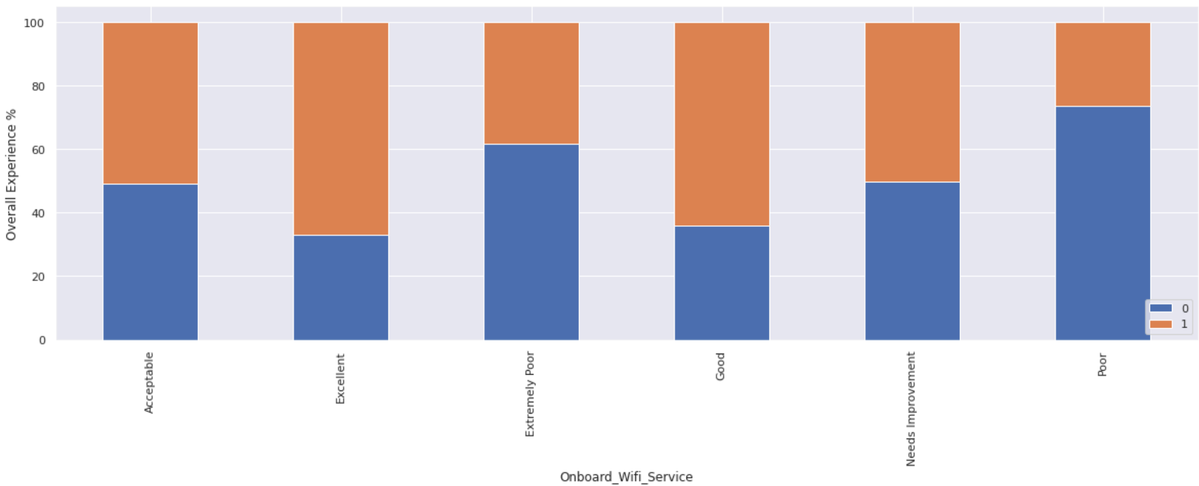
```
stacked_barplot(data, "Seat_Comfort", "Overall_Experience" )
```



Let's plot the stacked barplot for the variable `Designation` against the target variable `ProdTaken` using the stacked_barplot function.

In [172…

```
stacked_barplot(data, "Onboard_Wifi_Service", "Overall_Experience" )
```



Data Preparation for Modeling

TODO: Aqui há muito que alterar, porque depende mto do dataset, mas estão aqui exemplos úteis

Separating the independent variables (X) and the dependent variable (Y)

In [173…

```
# Separating target variable and other variables
X=df.drop(columns='Overall_Experience')
Y=df['Overall_Experience']
```

As we aim to predict customers who are more likely to buy the product, we should drop the columns `DurationOfPitch', 'NumberOfFollowups', 'ProductPitched', 'PitchSatisfactionScore'` as these columns would not be available at the time of prediction for new data.

In [174…

```
## Não vamos para já apagar nenhumas colunas porque todas parecem ser relevantes para o

# Dropping columns
# X.drop(columns=['DurationOfPitch','NumberOfFollowups','ProductPitched','PitchSatisfact
```

Splitting the data into a 70% train and 30% test set

Some classification problems can exhibit a large imbalance in the distribution of the target classes: for instance there could be several times more negative samples than positive samples. In such cases it is recommended to use the stratified sampling technique to ensure that relative class frequencies are approximately preserved in each train and validation fold.

In [175…

```
# Splitting the data into train and test sets

X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.30,random_state=1,stratif
```

In [176…

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(66065, 23)
(28314, 23)
(66065,)
(28314,)
```

As we saw earlier, our data has missing values. We will impute missing values using median for continuous variables and mode for categorical variables. We will use `SimpleImputer` to do this.

The `SimpleImputer` provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median, or most frequent) of each column in which the missing values are located.

In [177…

```
si1=SimpleImputer(strategy='median')

median_imputed_col=['Age','Travel_Distance','Departure_Delay_in_Mins','Arrival_Delay_in_

#X[median_imputed_col]=si1.fit_transform(X[median_imputed_col])

# Fit and transform the train data
X_train[median_imputed_col]=si1.fit_transform(X_train[median_imputed_col])

#Transform the test data i.e. replace missing values with the median calculated using tr
X_test[median_imputed_col]=si1.transform(X_test[median_imputed_col])

# Fit and transform the train data
data_test[median_imputed_col]=si1.fit_transform(data_test[median_imputed_col])
```

```
# Drop ID from the test data
data_test = data_test.drop(columns=['ID'])
```

In [178…

```
X_test.head()
```

Out[178]:

	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_Delay_in_Mins
81488	Female	Loyal Customer	18.0	Personal Travel	Eco	1772.0	18.0
64933	Female	Loyal Customer	28.0	Business Travel	Business	5128.0	0.0
6048	Female	Loyal Customer	39.0	Business Travel	Business	3187.0	0.0
54498	Female	Loyal Customer	32.0	Personal Travel	Eco	2543.0	0.0
45386	Male	Disloyal Customer	40.0	Business Travel	Business	1541.0	4.0

In [179…

```
data_test.head()
```

Out[179]:

	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_Delay_in_Mins	Ar
0	Female	NaN	36.0	Business Travel	Business	532.0	0.0	
1	Female	Disloyal Customer	21.0	Business Travel	Business	1425.0	9.0	
2	Male	Loyal Customer	60.0	Business Travel	Business	2832.0	0.0	
3	Female	Loyal Customer	29.0	Personal Travel	Eco	1352.0	0.0	
4	Male	Disloyal Customer	18.0	Business Travel	Business	1610.0	17.0	

In [180…

```
cleanup_nums = {
 "Gender": {"Female": 0, "Male": 1},
 "Customer_Type": {"Loyal Customer":1, "Disloyal Customer":0},
 "Type_Travel":{"Business Travel":1, "Personal Travel":0},
 "Travel_Class":{"Eco":0, "Business":1},
 "Seat_Comfort":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptable":3, "G
 "Seat_Class":{"Ordinary":0, "Green Car":1},
 "Arrival_Time_Convenient":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Accept
 "Catering":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptable":3, "Good"
 "Platform_Location":{"Very Inconvenient":0, "Inconvenient":1, "Needs Improvement":2, "M
 "Onboard_Wifi_Service":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptabl
 "Onboard_Entertainment":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptab
 "Online_Support":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptable":3,
 "Ease_of_Online_Booking":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Accepta
 "Onboard_Service":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptable":3,
 "Legroom":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptable":3, "Good":
 "Baggage_Handling":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptable":3
 "CheckIn_Service":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptable":3,
 "Cleanliness":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptable":3, "Go
 "Online_Boarding":{"Extremely Poor":0, "Poor":1, "Needs Improvement":2, "Acceptable":3,
}


X_train = X_train.replace(cleanup_nums)
```

```
X_test = X_test.replace(cleanup_nums)

data_test = data_test.replace(cleanup_nums)
```

In [181…

```
pd.set_option('display.max_columns', 500)

X_train.head()
```

Out[181]:

	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_Delay_in_Mins
90112	1.0	1.0	49.0	1.0	1	2023.0	64.0
54258	0.0	1.0	45.0	1.0	1	4879.0	160.0
58136	1.0	1.0	25.0	NaN	1	3779.0	0.0
23288	0.0	0.0	21.0	1.0	0	1928.0	0.0
31834	1.0	0.0	35.0	NaN	1	2331.0	2.0

In [182…

```
X_test.head()
```

Out[182]:

	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_Delay_in_Mins
81488	0.0	1.0	18.0	0.0	0	1772.0	18.0
64933	0.0	1.0	28.0	1.0	1	5128.0	0.0
6048	0.0	1.0	39.0	1.0	1	3187.0	0.0
54498	0.0	1.0	32.0	0.0	0	2543.0	0.0
45386	1.0	0.0	40.0	1.0	1	1541.0	4.0

In [183…

```
data_test.head()
```

Out[183]:

	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_Delay_in_Mins	Ar
0	0.0	NaN	36.0	1.0	1	532.0	0.0	
1	0.0	0.0	21.0	1.0	1	1425.0	9.0	
2	1.0	1.0	60.0	1.0	1	2832.0	0.0	
3	0.0	1.0	29.0	0.0	0	1352.0	0.0	
4	1.0	0.0	18.0	1.0	1	1610.0	17.0	

In [184…

```
# Lets fill in missing values:

si2=SimpleImputer(strategy='most_frequent')

mode_imputed_col=['Gender','Customer_Type','Type_Travel','Travel_Class','Seat_Comfort','
    'Catering','Platform_Location','Onboard_Wifi_Service','Onboard_Entertainment','Onlin
    'Onboard_Service','Legroom','Baggage_Handling','CheckIn_Service','Cleanliness','Onli

#X = si2.fit_transform(X[mode_imputed_col])

# Fit and transform the train data
X_train[mode_imputed_col]=si2.fit_transform(X_train[mode_imputed_col])

# Transform the test data i.e. replace missing values with the mode calculated using tra
X_test[mode_imputed_col]=si2.transform(X_test[mode_imputed_col])
```

```
# Transform the test data i.e. replace missing values with the mode calculated using tra
data_test[mode_imputed_col]=si2.transform(data_test[mode_imputed_col])
```

In [185…

```
X_train.head()
```

Out[185]:

	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_Delay_in_Mins
90112	1.0	1.0	49.0	1.0	1.0	2023.0	64.0
54258	0.0	1.0	45.0	1.0	1.0	4879.0	160.0
58136	1.0	1.0	25.0	1.0	1.0	3779.0	0.0
23288	0.0	0.0	21.0	1.0	0.0	1928.0	0.0
31834	1.0	0.0	35.0	1.0	1.0	2331.0	2.0

In [186…

```
X_test.head()
```

Out[186]:

	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_Delay_in_Mins
81488	0.0	1.0	18.0	0.0	0.0	1772.0	18.0
64933	0.0	1.0	28.0	1.0	1.0	5128.0	0.0
6048	0.0	1.0	39.0	1.0	1.0	3187.0	0.0
54498	0.0	1.0	32.0	0.0	0.0	2543.0	0.0
45386	1.0	0.0	40.0	1.0	1.0	1541.0	4.0

In [187…

```
data_test.head(10)
```

Out[187]:

	Gender	Customer_Type	Age	Type_Travel	Travel_Class	Travel_Distance	Departure_Delay_in_Mins	Ar
0	0.0	1.0	36.0	1.0	1.0	532.0	0.0	
1	0.0	0.0	21.0	1.0	1.0	1425.0	9.0	
2	1.0	1.0	60.0	1.0	1.0	2832.0	0.0	
3	0.0	1.0	29.0	0.0	0.0	1352.0	0.0	
4	1.0	0.0	18.0	1.0	1.0	1610.0	17.0	
5	1.0	1.0	49.0	1.0	1.0	382.0	89.0	
6	1.0	0.0	40.0	1.0	1.0	1761.0	0.0	
7	0.0	1.0	11.0	0.0	0.0	3989.0	0.0	
8	1.0	1.0	57.0	1.0	1.0	2731.0	0.0	
9	0.0	1.0	43.0	1.0	0.0	2645.0	222.0	

In [188…

```
# Checking that no column has missing values in train or test sets
print(X_train.isna().sum())
print('-'*30)
print(X_test.isna().sum())
print('-'*30)
print(data_test.isna().sum())
```

```
Gender                     0
Customer_Type              0
Age                        0
Type_Travel                0
Travel_Class               0
Travel_Distance            0
Departure_Delay_in_Mins    0
Arrival_Delay_in_Mins      0
Seat_Comfort               0
Seat_Class                 0
Arrival_Time_Convenient    0
Catering                   0
Platform_Location          0
Onboard_Wifi_Service       0
Onboard_Entertainment      0
Online_Support             0
Ease_of_Online_Booking     0
Onboard_Service            0
Legroom                    0
Baggage_Handling           0
CheckIn_Service            0
Cleanliness                0
Online_Boarding            0
dtype: int64
------------------------------
Gender                     0
Customer_Type              0
Age                        0
Type_Travel                0
Travel_Class               0
Travel_Distance            0
Departure_Delay_in_Mins    0
Arrival_Delay_in_Mins      0
Seat_Comfort               0
Seat_Class                 0
Arrival_Time_Convenient    0
Catering                   0
Platform_Location          0
Onboard_Wifi_Service       0
Onboard_Entertainment      0
Online_Support             0
Ease_of_Online_Booking     0
Onboard_Service            0
Legroom                    0
Baggage_Handling           0
CheckIn_Service            0
Cleanliness                0
Online_Boarding            0
dtype: int64
------------------------------
Gender                     0
Customer_Type              0
Age                        0
Type_Travel                0
Travel_Class               0
Travel_Distance            0
Departure_Delay_in_Mins    0
Arrival_Delay_in_Mins      0
Seat_Comfort               0
Seat_Class                 0
Arrival_Time_Convenient    0
Catering                   0
Platform_Location          0
Onboard_Wifi_Service       0
Onboard_Entertainment      0
Online_Support             0
Ease_of_Online_Booking     0
```

```
Onboard_Service           0
Legroom                   0
Baggage_Handling          0
CheckIn_Service           0
Cleanliness               0
Online_Boarding           0
dtype: int64
```

Let's create dummy variables for string type variables and convert other column types back to float.

In [189…

```
#converting data types of columns to float
for column in ['Age','Travel_Distance','Departure_Delay_in_Mins','Arrival_Delay_in_Mins'
     X_train[column]=X_train[column].astype('float')
     X_test[column]=X_test[column].astype('float')
     data_test[column]=data_test[column].astype('float')
```

**We won't use dummy variables this time:

In [190…

```
#List of columns to create a dummy variables
#col_dummy=['Gender','Customer_Type','Type_Travel','Travel_Class','Seat_Comfort','Seat_C
#    'Catering','Platform_Location','Onboard_Wifi_Service','Onboard_Entertainment','Onli
#    'Onboard_Service','Legroom','Baggage_Handling','CheckIn_Service','Cleanliness','Onl
```

In [191…

```
#Encoding categorical varaibles
# X=pd.get_dummies(X, columns=col_dummy, drop_first=True)
#X_train=pd.get_dummies(X_train, columns=col_dummy, drop_first=True)
#X_test=pd.get_dummies(X_test, columns=col_dummy, drop_first=True)
```

In [192…

```
print(X_train.shape)
print(X_test.shape)
```

```
(66065, 23)
(28314, 23)
```

**Já nao vamos martelar zeros nas colunas

In [193…

```
# Martelar 0's nas colunas que deixam de existir do df de test:

# X_test['Platform_Location_Very Inconvenient'] = 0
# X_test['Onboard_Service_Extremely Poor'] = 0
# X_test['CheckIn_Service_Extremely Poor'] = 0
# X_test['Online_Support_Extremely Poor'] = 0

# print(X_train.shape)
# print(X_test.shape)
```

Model evaluation criterion:

TODO: O sumo do tema. Aqui podemos dividir o trabalho em 2 e depois comparar resultados

- Andre: RF e SVN
- Valter: Rede neuronal e KNN?

The model can make wrong predictions as:

1. Predicting a customer will buy the product and the customer doesn't buy - Loss of resources
2. Predicting a customer will not buy the product and the customer buys - Loss of opportunity

Which case is more important?

- Predicting that customer will not buy the product but he buys i.e. losing on a potential source of income for the company because that customer will not be targeted by the marketing team when he should be targeted.

How to reduce this loss i.e need to reduce False Negatives?

- The company wants Recall to be maximized, the greater the Recall lesser the chances of false negatives.

Building the model

We will be building 4 different models:

- **Logistic Regression**
- **Support Vector Machine(SVM)**
- **Decision Tree**
- **Random Forest**

Also, let's create a function to calculate and print the classification report and confusion matrix so that we don't have to rewrite the same code repeatedly for each model.

In [194…

```
# Creating metric function
def metrics_score(actual, predicted):
    print(classification_report(actual, predicted))

    cm = confusion_matrix(actual, predicted)
    plt.figure(figsize=(8,5))

    sns.heatmap(cm, annot=True,  fmt='.2f')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()
```

Question 7: Random Forest (4 Marks)

Question 7.1: Build a Random Forest Model (1 Mark)

In [195…

```
rf_estimator = RandomForestClassifier(random_state = 1)

rf_estimator.fit(X_train, y_train)
```

Out[195]:

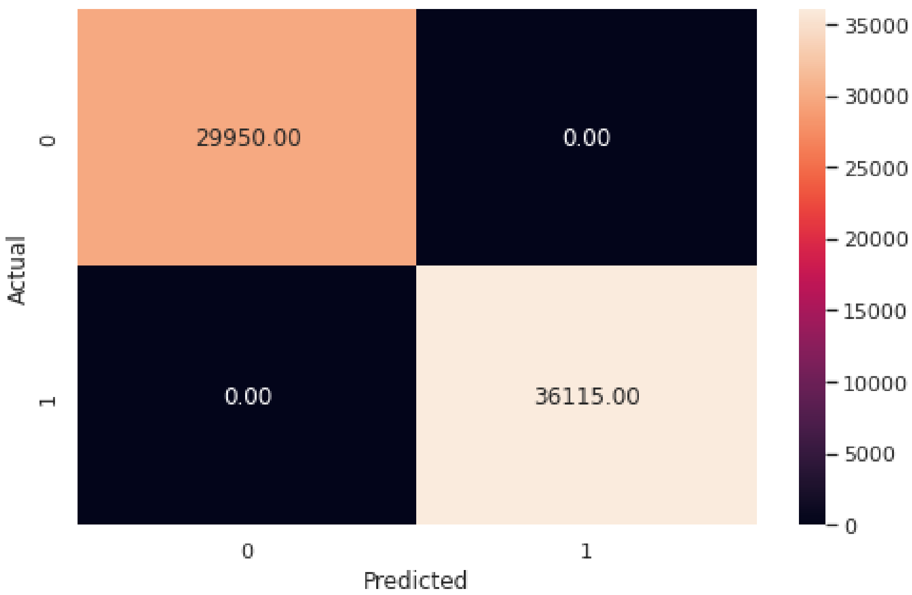
```
RandomForestClassifier(random_state=1)
```

Question 7.2: Check the performance of the model on the train and test data (2 Marks)

In [196…

```
y_pred_train_rf = rf_estimator.predict(X_train)

metrics_score(y_train, y_pred_train_rf)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     29950
           1       1.00      1.00      1.00     36115

    accuracy                           1.00     66065
   macro avg       1.00      1.00      1.00     66065
weighted avg       1.00      1.00      1.00     66065
```



Write your Answer here :

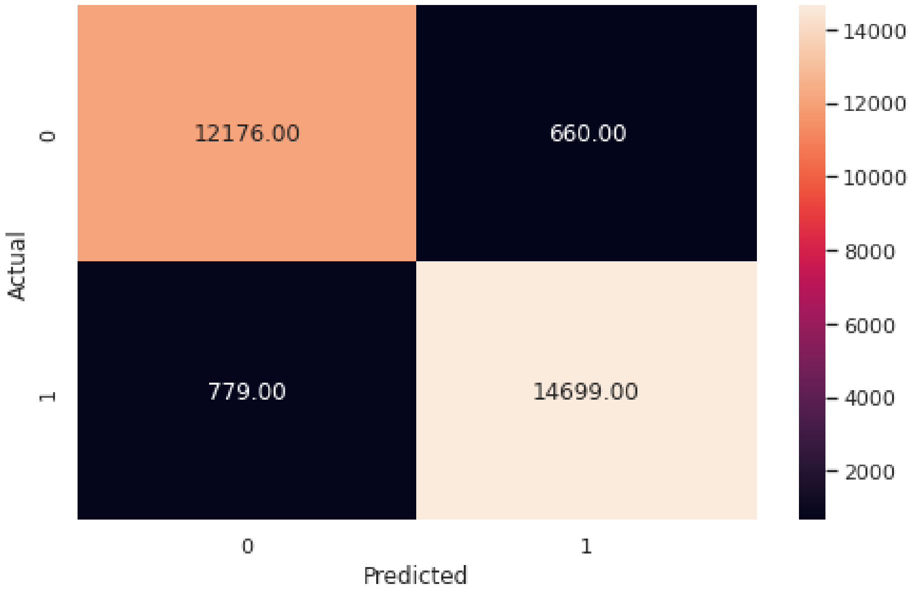
- 0 errors on training set!
- Model has performed very well on the training set.

In [197…
```
X_train.shape
```
Out[197]:
```
(66065, 23)
```

In [198…
```
X_test.shape
```
Out[198]:
```
(28314, 23)
```

In [199…
```
y_pred_test_rf = rf_estimator.predict(X_test)

metrics_score(y_test, y_pred_test_rf)
```

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.94     12836
           1       0.96      0.95      0.95     15478

    accuracy                           0.95     28314
   macro avg       0.95      0.95      0.95     28314
weighted avg       0.95      0.95      0.95     28314
```



Write your Answer here :

- The Random Forest classifier seems good.
- Accuracy is high 95%
- We can reduce overfitting by hyperparameter tuning.

Creat export output for submission based on nontuned RF:

In [200…

```
y_pred_export_rf = rf_estimator.predict(data_test)
pd.DataFrame(y_pred_export_rf).to_csv('/content/drive/MyDrive/Outros/GL Hackathon/output
```

In [201…

```
## Random Forest hyperparameter tuning

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 20)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)

# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
```

```
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter
# Fit the random search model
# rf_random.fit(X_train, y_train)
```

```
{'n_estimators': [200, 294, 389, 484, 578, 673, 768, 863, 957, 1052, 1147, 1242, 1336, 1
431, 1526, 1621, 1715, 1810, 1905, 2000], 'max_features': ['auto', 'sqrt'], 'max_depth':
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
```

In [202…

```
#rf_random.best_params_
```

In [203…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False],
    'max_depth': [20, 30, 40, None],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [3, 5, 7],
    'n_estimators': [1140, 1242, 1280]
}

# Create a based model
rf = RandomForestRegressor(n_estimators = 1242, min_samples_split = 5, min_samples_leaf
# Instantiate the grid search model
# grid_search = GridSearchCV(estimator = rf, param_grid = param_grid, cv = 2, n_jobs = -

# Fit the grid search to the data
# grid_search.fit(X_train, y_train)
# grid_search.best_params_
```

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV

rf_estimator_tunned = RandomForestClassifier(random_state = 1, bootstrap = False, max_de

rf_estimator_tunned.fit(X_train, y_train)

y_pred_test_tunned_rf = rf_estimator_tunned.predict(X_test)

metrics_score(y_test, y_pred_test_tunned_rf)
```

Question 7.3: What are some important features based on the Random Forest? (1 Mark)

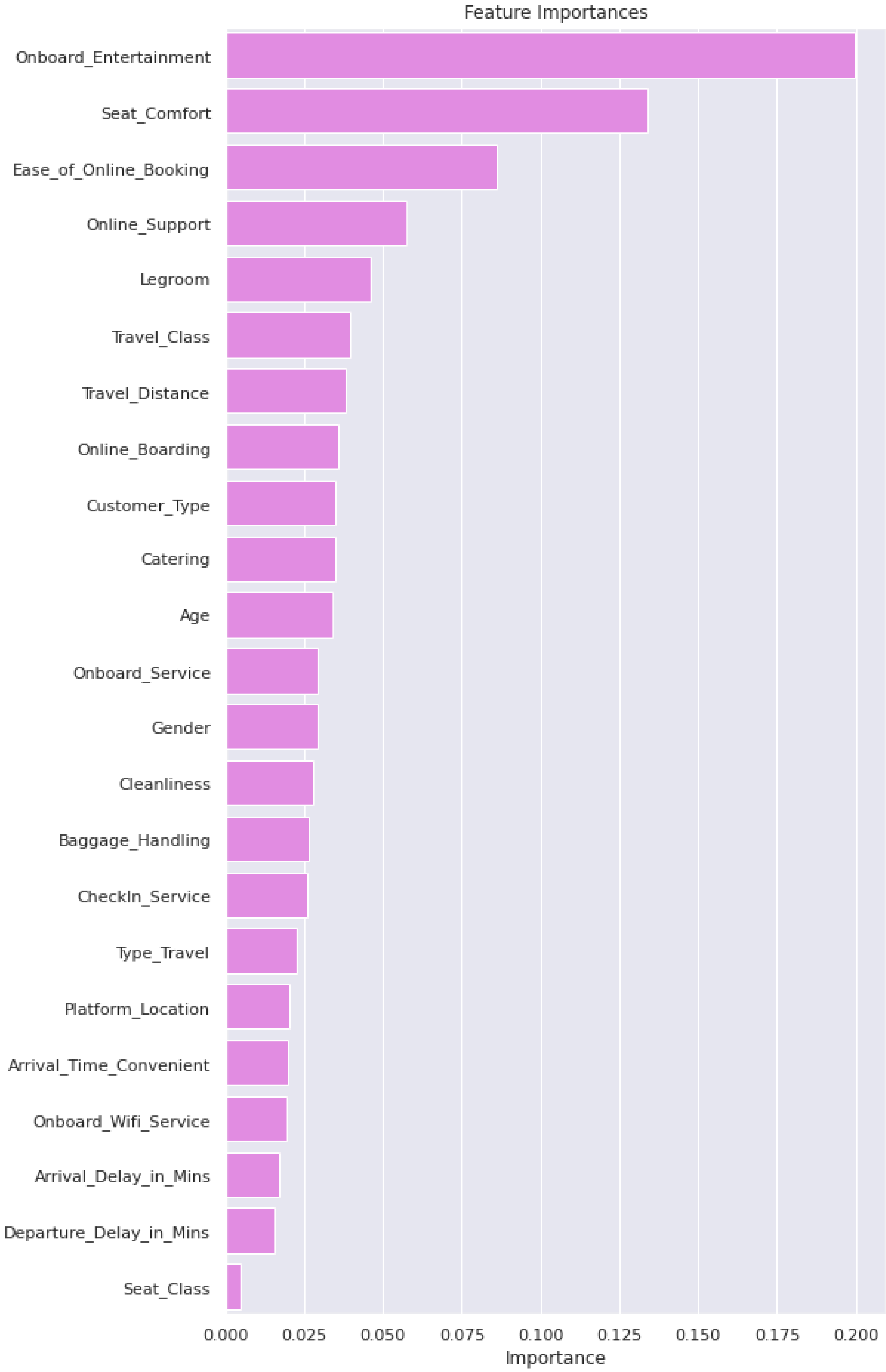
Let's check the feature importance of the Random Forest

In [ ]:

```
importances = rf_estimator.feature_importances_

columns = X_train.columns

importance_df = pd.DataFrame(importances, index = columns, columns = ['Importance']).sor


plt.figure(figsize=(8, 16))
plt.title("Feature Importances")
sns.barplot(importance_df.Importance, importance_df.index,color="violet")
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb60f0b3f40>
```



Question 4: Logistic Regression (6 Marks)

Question 4.1: Build a Logistic Regression model (Use the sklearn library) (1 Mark)

In [ ]:
```
# Fitting logistic regression model
lg = LogisticRegression()
```

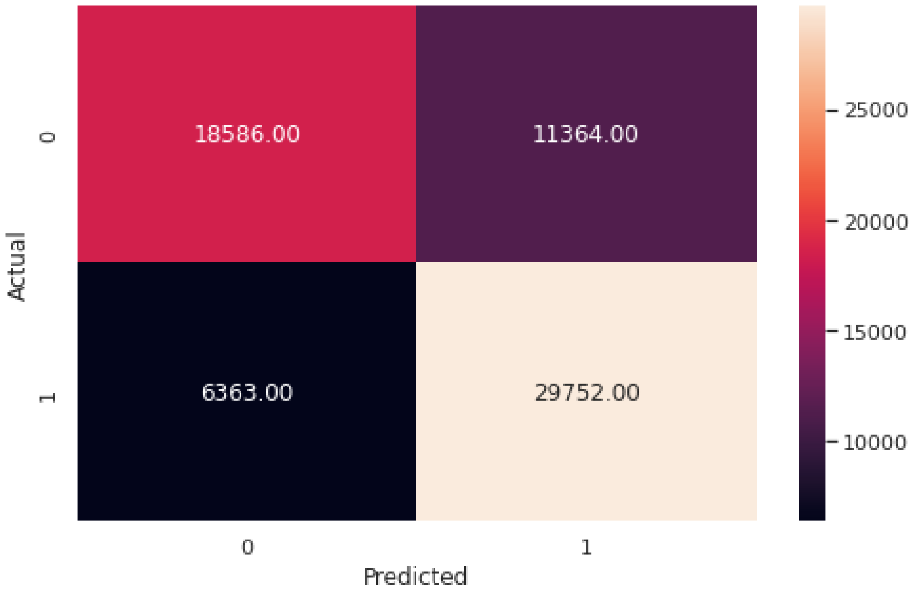
```
lg.fit(X_train,y_train)
```

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

Question 4.2: Check the performance of the model on train and test data (2 Marks)

In [ ]:
```
# Checking the performance on the training data
y_pred_train = lg.predict(X_train)
metrics_score(y_train, y_pred_train)
```

```
              precision    recall  f1-score   support

           0       0.74      0.62      0.68     29950
           1       0.72      0.82      0.77     36115

    accuracy                           0.73     66065
   macro avg       0.73      0.72      0.72     66065
weighted avg       0.73      0.73      0.73     66065
```



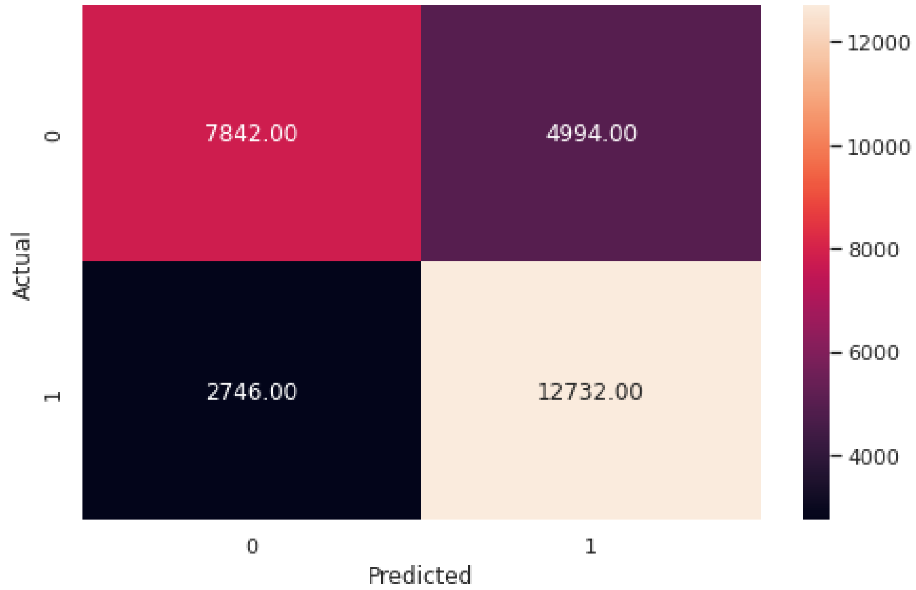
Write your Answer here:

- We have been able to build a predictive model that can be used by the tourist company to predict the customers who are likely to accept the new package with a recall score of 25%.

Let's check the performance on the test set

In [ ]:
```
# Checking the performance on the test dataset
y_pred_test = lg.predict(X_test)
metrics_score(y_test, y_pred_test)
```

```
              precision    recall  f1-score   support

           0       0.74      0.61      0.67     12836
           1       0.72      0.82      0.77     15478

    accuracy                           0.73     28314
   macro avg       0.73      0.72      0.72     28314
weighted avg       0.73      0.73      0.72     28314
```



Write your Answer here:

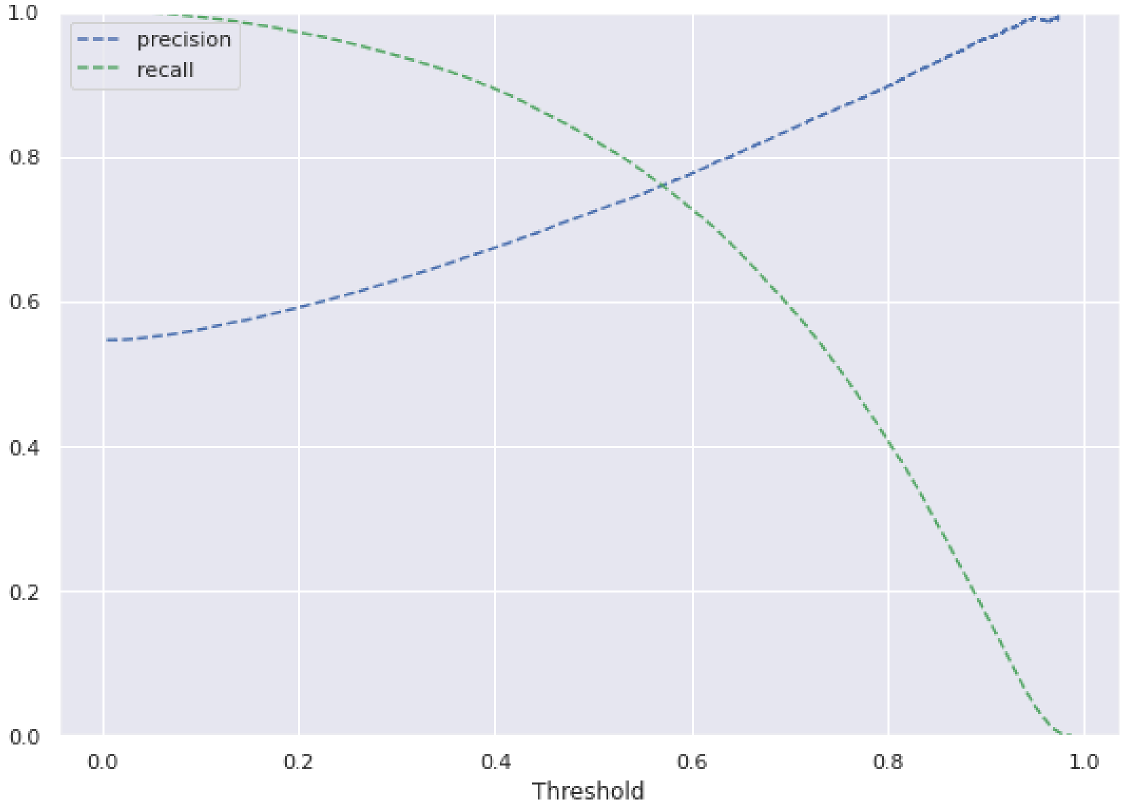
- Using the model with default threshold the model gives a low recall but decent precision score.
- We can’t have both precision and recall high. If you increase precision, it will reduce recall, and vice versa. This is called the precision/recall tradeoff.
- So let's find an optimal threshold where we can balance both the metrics.

Question 4.3: Find the optimal threshold for the model using the Precision-Recall Curve. (1 Mark)

Precision-Recall curve summarizes the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.

Let's use the Precision-Recall curve and see if we can find a **better threshold.**

In [ ]:
```
# Predict_proba gives the probability of each observation belonging to each class
y_scores_lg=lg.predict_proba(X_train)

precisions_lg, recalls_lg, thresholds_lg = precision_recall_curve(y_train, y_scores_lg[:

# Plot values of precisions, recalls, and thresholds
plt.figure(figsize=(10,7))
plt.plot(thresholds_lg, precisions_lg[:-1], 'b--', label='precision')
plt.plot(thresholds_lg, recalls_lg[:-1], 'g--', label = 'recall')
plt.xlabel('Threshold')
plt.legend(loc='upper left')
plt.ylim([0,1])
plt.show()
```



- We want to choose a threshold that has a high recall while also having a small drop in precision. High recall is necessary, simultaneously we also need to be careful not to lose precision too much. So the threshold value of 0.25 should be sufficient because it has good recall and does not cause a significant drop in precision.

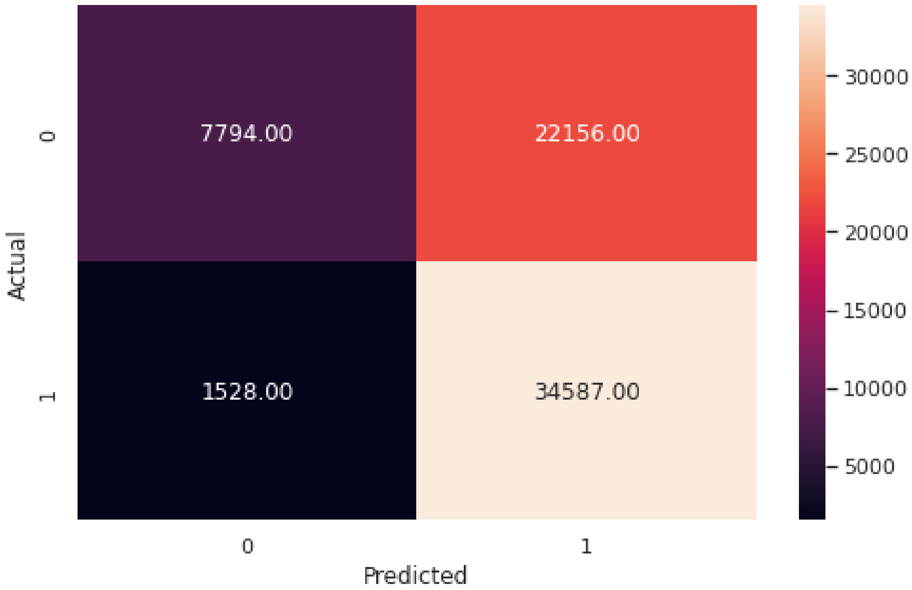
Note: We are attempting to maximise recall because that is our metric of interest. Consider the F1 score as the metric of interest then we must find the threshold that provides balanced precision and recall values. In that case, the theshold value will be 0.30.

In [ ]:
```
# Setting the optimal threshold
optimal_threshold = 0.25
```

Question 4.4: Check the performance of the model on train and test data using the optimal threshold. (2 Marks)

In [ ]:
```
# creating confusion matrix
y_pred_train = lg.predict_proba(X_train)
metrics_score(y_train, y_pred_train[:,1]>optimal_threshold)
```

	precision	recall	f1-score	support
0	0.84	0.26	0.40	29950
1	0.61	0.96	0.74	36115
accuracy			0.64	66065
macro avg	0.72	0.61	0.57	66065
weighted avg	0.71	0.64	0.59	66065



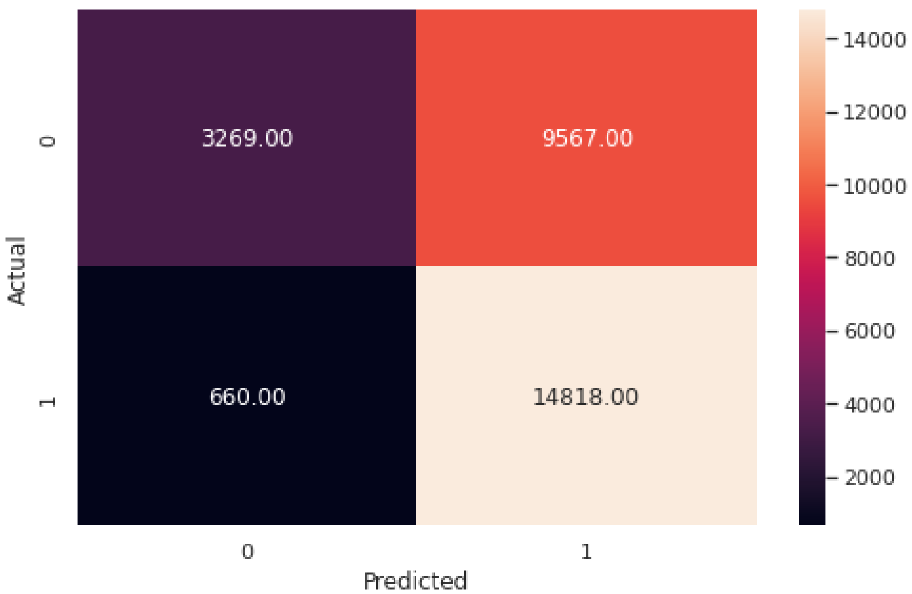
Write your Answer here :

- The model performance has improved as compared to our initial model.The recall has increased by 36%.

Let's check the performance on the test set

```
In [ ]: y_pred_test = lg.predict_proba(X_test)
        metrics_score(y_test, y_pred_test[:,1]>optimal_threshold)
```

```
              precision    recall  f1-score   support

           0       0.83      0.25      0.39     12836
           1       0.61      0.96      0.74     15478

    accuracy                           0.64     28314
   macro avg       0.72      0.61      0.57     28314
weighted avg       0.71      0.64      0.58     28314
```



Write your Answer here :

- Using the model with a threshold of 0.25, the model has achieved a recall of 67% i.e. increase of 44%.

- The precision has dropped compared to inital model but using optimial threshold the model is able to provide the balanced performance.

However the model performance is not good. So let's try building another model.

Question 5: Support Vector Machines (11 Marks)

To accelerate SVM training, let's scale the data for support vector machines.

In [ ]:
```
scaling = MinMaxScaler(feature_range=(-1,1)).fit(X_train)
X_train_scaled = scaling.transform(X_train)
X_test_scaled = scaling.transform(X_test)
```

Let's build the models using the two of the widely used kernel functions:

1. **Linear Kernel**
2. **RBF Kernel**

Question 5.1: Build a Support Vector Machine model using a linear kernel (1 Mark)

In [107…
```
svm = SVC(kernel='linear',probability=True) # Linear kernal or linear decision boundary
model = svm.fit(X= X_train_scaled, y = y_train)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-107-b173052bb445> in <module>
      1 svm = SVC(kernel='linear',probability=True) # Linear kernal or linear decision boundary
----> 2 model = svm.fit(X= X_train_scaled, y = y_train)

/usr/local/lib/python3.8/dist-packages/sklearn/svm/_base.py in fit(self, X, y, sample_weight)
    253 
    254         seed = rnd.randint(np.iinfo("i").max)
--> 255         fit(X, y, sample_weight, solver_type, kernel, random_seed=seed)
    256         # see comment on the other call to np.iinfo in this file
    257 

/usr/local/lib/python3.8/dist-packages/sklearn/svm/_base.py in _dense_fit(self, X, y, sample_weight, solver_type, kernel, random_seed)
    313             self._probB,
    314             self.fit_status_,
--> 315         ) = libsvm.fit(
    316             X,
    317             y,

KeyboardInterrupt: 
```

Question 5.2: Check the performance of the model on train and test data (2 Marks)

In [ ]:
```
y_pred_train_svm = model.predict(X_train_scaled)
metrics_score(y_train, y_pred_train_svm)
```

Write your Answer here :

- This model has completely failed to detect the class 1. The model predicted all the instances as class 0.
- The model has an recall score of 0.

Checking model performance on test set

In [ ]:
```
print("Testing performance:")
y_pred_test_svm = model.predict(X_test_scaled)
metrics_score(y_test, y_pred_test_svm)
```

Write your Answer here:

- As the dataset has an imbalanced class distribution the model almost always predicts 0.
- So for linear kernel the 0.5 threshold doesn't seems to work. So lets find the optimal threshold and check if the model performs well.

Question 5.3: Find the optimal threshold for the model using the Precision-Recall Curve. (1 Mark)

In [ ]:
```
# Predict on train data
y_scores_svm=model.predict_proba(X_train_scaled)

precisions_svm, recalls_svm, thresholds_svm = precision_recall_curve(y_train, y_scores_s

# Plot values of precisions, recalls, and thresholds
plt.figure(figsize=(10,7))
plt.plot(thresholds_svm, precisions_svm[:-1], 'b--', label='precision')
plt.plot(thresholds_svm, recalls_svm[:-1], 'g--', label = 'recall')
plt.xlabel('Threshold')
plt.legend(loc='upper left')
plt.ylim([0,1])
plt.show()
```

- In this case the threshold value of 0.25 seems to be good as it has good recall and there isn't much drop in precision.

In [ ]:
```
optimal_threshold_svm=0.25
```

Question 5.4: Check the performance of the model on train and test data using the optimal threshold. (2 Marks)

In [ ]:
```
print("Training performance:")
y_pred_train_svm = model.predict_proba(X_train_scaled)
metrics_score(y_train, y_pred_train_svm[:,1]>optimal_threshold_svm)
```

Write your Answer here :

- The model performance has improved by selecting the optimal threshold of 0.25.
- The recall has increased from 0 to 56%.

In [ ]:
```
y_pred_test = model.predict_proba(X_test_scaled)
metrics_score(y_test, y_pred_test[:,1]>optimal_threshold_svm)
```

Write your Answer here :

- SVM model with **linear kernel** is not overfitting as the accuracy is around 78% for both train and test dataset
- The model has a **Recall** of 61% which is highest compared to the above models.
- At the optimal threshold of .25, the model performance has improved really well. The F1 score has improved from 0.00 to 0.52.

Lets try using non-linear kernel and check if it can improve the performance.

Question 5.5: Build a Support Vector Machines model using an RBF kernel (1 Mark)

In [ ]:
```
svm_rbf=SVC(kernel='rbf',probability=True)
# Fit the model
svm_rbf.fit(X_train_scaled,y_train)
```

Question 5.6: Check the performance of the model on train and test data (2 Marks)

In [ ]:
```
y_pred_train_svm = svm_rbf.predict(X_train_scaled)
metrics_score(y_train, y_pred_train_svm)
```

Write your Answer here :

- When compared to the baseline svm model with linear kernel, the model's performance on training data has been slightly improved by using an RBF kernel.

Checking model performance on test set

In [ ]:
```
y_pred_test = svm_rbf.predict(X_test_scaled)

metrics_score(y_test, y_pred_test)
```

Write your Answer here :

- When compared to the baseline svm model with linear kernel, the recall score on testing data has increased from 0% to 26%.

In [ ]:
```
# Predict on train data
y_scores_svm=svm_rbf.predict_proba(X_train_scaled)

precisions_svm, recalls_svm, thresholds_svm = precision_recall_curve(y_train, y_scores_s

# Plot values of precisions, recalls, and thresholds
plt.figure(figsize=(10,7))
plt.plot(thresholds_svm, precisions_svm[:-1], 'b--', label='precision')
plt.plot(thresholds_svm, recalls_svm[:-1], 'g--', label = 'recall')
plt.xlabel('Threshold')
plt.legend(loc='upper left')
plt.ylim([0,1])
plt.show()
```

In [ ]:
```
optimal_threshold_svm=0.17
```

Question 5.7: Check the performance of the model on train and test data using the optimal threshold. (2 Marks)

Checking model performance on training set

```
In [ ]: y_pred_train_svm = model.predict_proba(X_train_scaled)
        metrics_score(y_train, y_pred_train_svm[:,1]>optimal_threshold_svm)
```

Write your Answer here :

- SVM model with **RBF kernel** is performing better compared to the linear kernel.
- The model has achieved a recall score of 0.78 but there is a slight drop in the precision value.
- Using the model with a threshold of 0.17, the model gives a better recall score compared to the initial model.

Checking model performance on test set

```
In [ ]: y_pred_test = svm_rbf.predict_proba(X_test_scaled)
        metrics_score(y_test, y_pred_test[:,1]>optimal_threshold_svm)
```

Write your Answer here :

- The **recall score** for the model is around 69%.
- At the optimal threshold of .17, the model performance has improved from 0.26 to 0.69.
- This is the best performing model when compared to SVM with linear kernel and Logistic Regression because it provides good recall with no big drop in precision as well.

Let's build some non-linear models and see if they can outperform linear models.

Question 6: Decision Trees (7 Marks)

Question 6.1: Build a Decision Tree Model (1 Mark)

```
In [ ]: model_dt = DecisionTreeClassifier(random_state=1)
        model_dt.fit(X_train, y_train)
```

Question 6.2: Check the performance of the model on train and test data (2 Marks)

```
In [ ]: # Checking performance on the training dataset
        pred_train_dt = model_dt.predict(X_train)
        metrics_score(y_train, pred_train_dt)
```

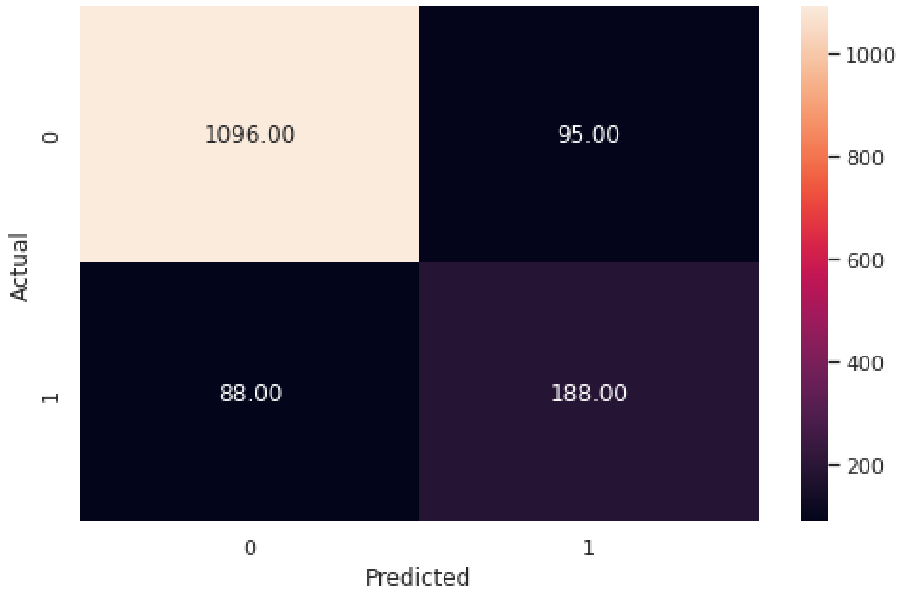
Write your Answer here :

- Almost 0 errors on the training set, each sample has been classified correctly.
- Model has performed very well on the training set.
- As we know a decision tree will continue to grow and classify each data point correctly if no restrictions are applied as the trees will learn all the patterns in the training set.
- Let's check the performance on test data to see if the model is overfitting.

Checking model performance on test set

```
In [ ]: pred_test_dt = model_dt.predict(X_test)
        metrics_score(y_test, pred_test_dt)
```

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92      1191
           1       0.66      0.68      0.67       276

    accuracy                           0.88      1467
   macro avg       0.79      0.80      0.80      1467
weighted avg       0.88      0.88      0.88      1467
```



Write your Answer here :

- The decision tree model is clearly overfitting. However the decision tree has better performance compared to Logistic Regression and SVM models.
- We will have to tune the decision tree to reduce the overfitting.

Question 6.3: Perform hyperparameter tuning for the decision tree model using GridSearch CV (1 Mark)

In [ ]:

```
# Choose the type of classifier.
estimator = DecisionTreeClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    "max_depth": np.arange(1,100,10),
    "max_leaf_nodes": [50, 75, 150, 250],
    "min_samples_split": [10, 30, 50, 70],
}
# Run the grid search
grid_obj = GridSearchCV(estimator, parameters, cv=5,scoring='recall',n_jobs=-1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
estimator.fit(X_train, y_train)
```

Out[ ]:

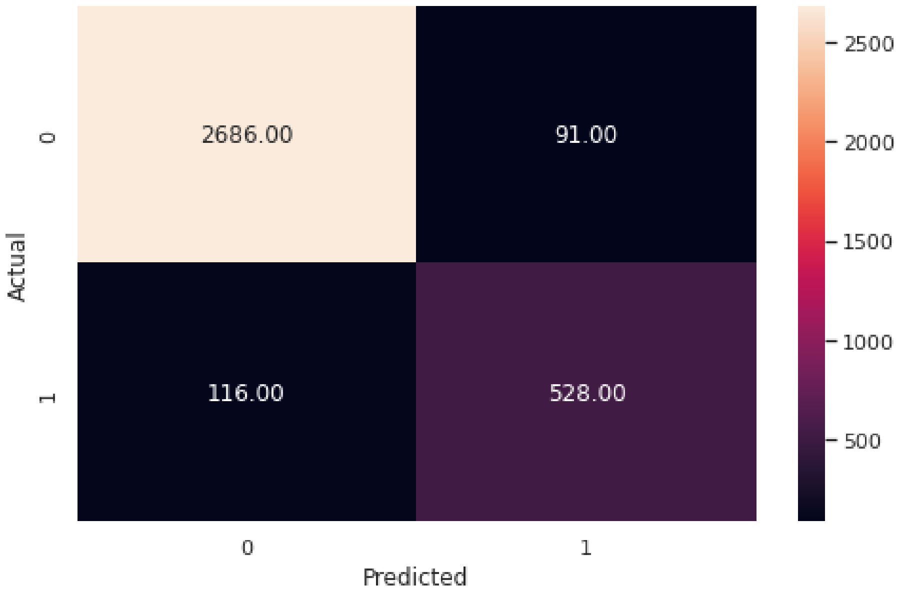
```
DecisionTreeClassifier(max_depth=21, max_leaf_nodes=250, min_samples_split=10,
                       random_state=1)
```

Question 6.4: Check the performance of the model on the train and test data using the tuned model (2 Mark)

Checking performance on the training set

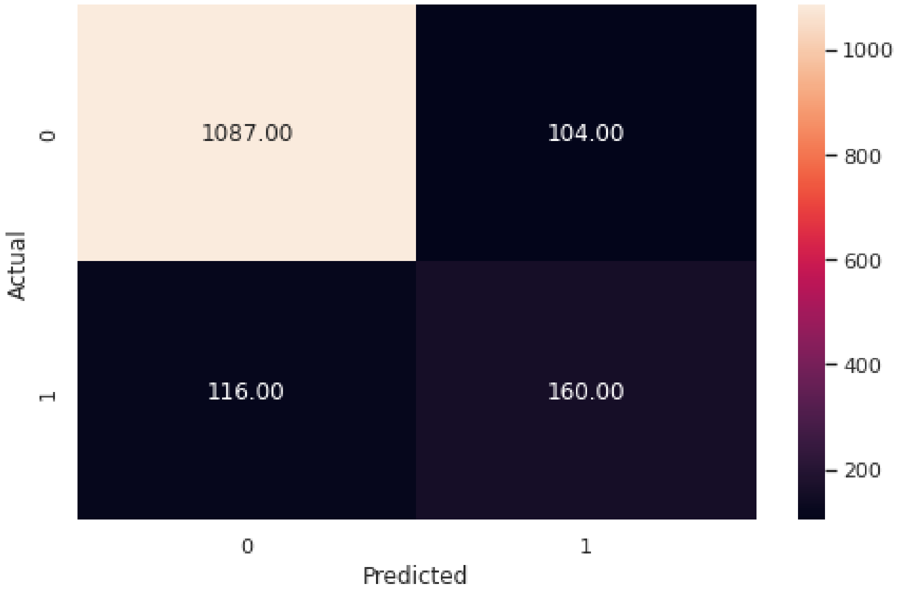
In [ ]:
```
# Checking performance on the training dataset
dt_tuned = estimator.predict(X_train)
metrics_score(y_train,dt_tuned)
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      2777
           1       0.85      0.82      0.84       644

    accuracy                           0.94      3421
   macro avg       0.91      0.89      0.90      3421
weighted avg       0.94      0.94      0.94      3421
```



In [ ]:
```
# Checking performance on the training dataset
y_pred_tuned = estimator.predict(X_test)
metrics_score(y_test,y_pred_tuned)
```

```
              precision    recall  f1-score   support

           0       0.90      0.91      0.91      1191
           1       0.61      0.58      0.59       276

    accuracy                           0.85      1467
   macro avg       0.75      0.75      0.75      1467
weighted avg       0.85      0.85      0.85      1467
```

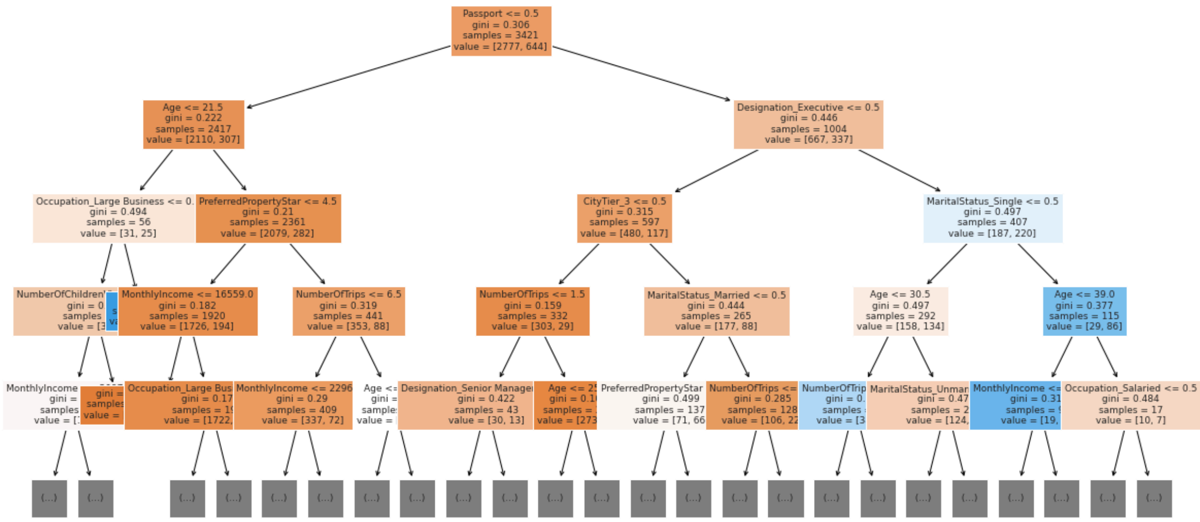


Write your Answer here :

- Decision tree model with default parameters is overfitting the training data and is not able to generalize well.
- Tuned moded has provided a generalised performance with balanced precision and recall values.
- However, there is still some overfitting, and model performance on test data has not significantly improved.

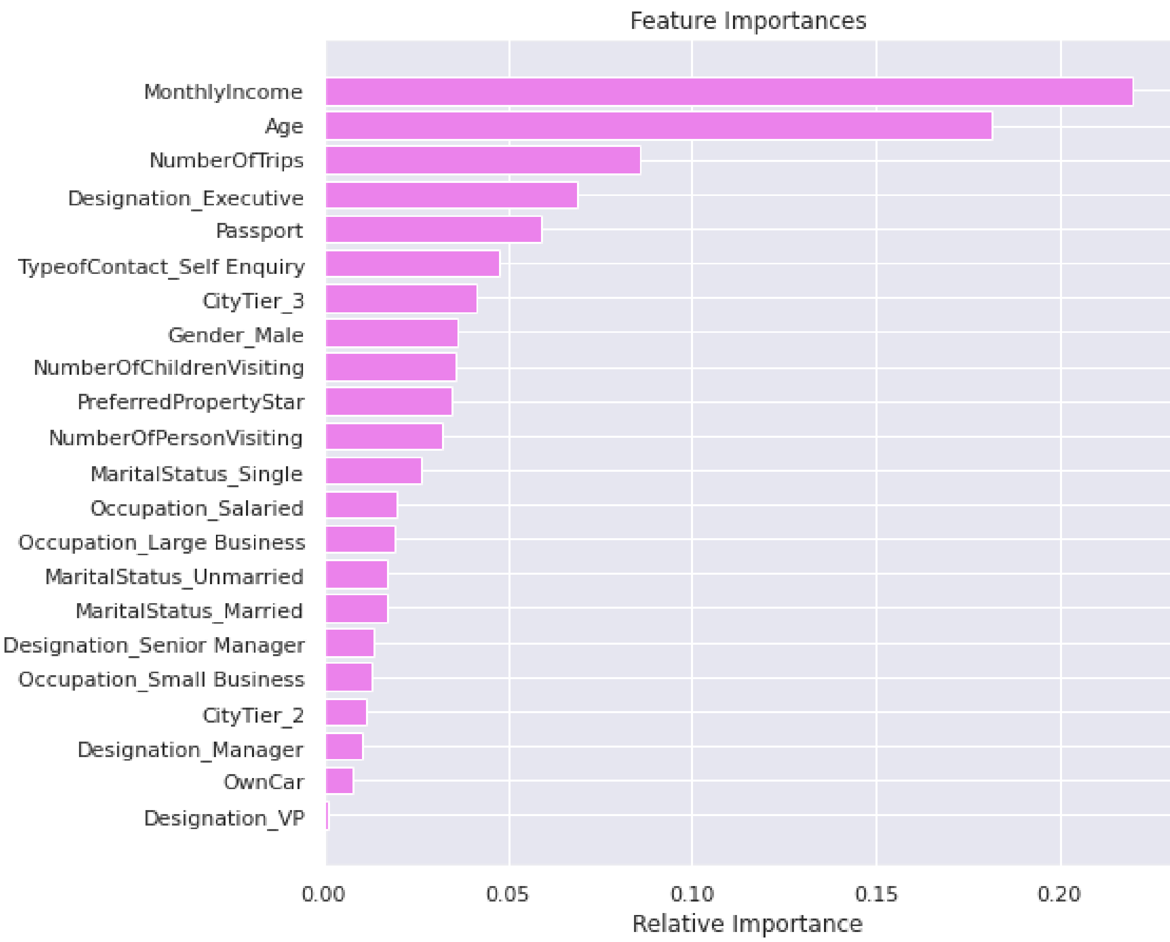
Visualizing the Decision Tree

In [ ]:
```
feature_names = list(X_train.columns)
plt.figure(figsize=(20, 10))
out = tree.plot_tree(
    estimator,
    max_depth=4,
    feature_names=feature_names,
    filled=True,
    fontsize=9,
    node_ids=False,
    class_names=None,
)
# below code will add arrows to the decision tree split if they are missing
for o in out:
    arrow = o.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor("black")
        arrow.set_linewidth(1)
plt.show()
```



Question 6.5: What are some important features based on the tuned decision tree? (1 Mark)

In [ ]:
```python
# Importance of features in the tree building
importances = model_dt.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(8, 8))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Write your Answer here :

- We can see that the tree has become simpler and the rules of the trees are readable.
- The model performance of the model has been generalized.

- We observe that the most important features are:
  - Monthly Income
  - Age
  - Number of trips

Conclusion:

- The SVM with RBF kenel has outperformed other models and provided balanced metrics.
- We have been able to build a predictive model that can be used by the tourist company to predict the customers who are likely to accept the new package with the recall score of 0.69 formulate marketing policies accordingly.

Question 8: Conclude ANY FOUR key takeaways for business recommendations (4 Marks)

Write your Answer here :

- Our analysis shows that very few customers have passports and they are more likely to purchase the travel package. The company should customize more international packages to attract more such customers.
- We have customers from tier 1 and tier 3 cities but very few from tier 2 cities. The company should expand its marketing strategies to increase the number of customers from tier 2 cities.
- We saw in our analysis that people with higher income or at high positions like AVP or VP are less likely to buy the product. The company can offer short-term travel packages and customize the package for higher- income customers with added luxuries to target such customers.
- When implementing a marketing strategy, external factors, such as the number of follow-ups, time of call, should also be carefully considered as our analysis shows that the customers who have been followed up more are the ones buying the package.
- After we identify a potential customer, the company should pitch packages as per the customer's monthly income, for example, do not pitch king packages to a customer with low income and such packages can be pitched more to the higher-income customers.
- We saw in our analysis that young and single people are more likely to buy the offered packages. The company can offer discounts or customize the package to attract more couples, families, and customers above 30 years of age.