Case Study: Visit with us | Travel Package Purchase Prediction

Using Ensemble Techniques

Background & Context

- The Policy Maker of the company wants to enable and establish a viable business model to expand the customer base.
- The company in the last campaign contacted the customers at random without looking at the available information and we observed that `18%` of the customers purchased the packages.
- Currently, there are 5 types of packages the company is offering - Basic, Standard, Deluxe, Super Deluxe, King.
- One of the ways to expand the customer base is to introduce a new offering of packages. The company is now planning to launch a new product i.e. `Wellness Tourism Package` .

Objective

- Explore and visualize the dataset.
- Enable and establish a viable business model to expand the customer base
- Generate a set of insights and recommendations that will help the business.
- Predict which customer is more likely to purchase the newly introduced travel package.

Data Dictionary -

Customer details:

- CustomerID: Unique customer ID
- ProdTaken: Whether the customer has purchased a package or not (0: No, 1: Yes)
- Age: Age of customer
- TypeofContact: How customer was contacted (Company Invited or Self Inquiry)
- CityTier: City tier depends on the development of a city, population, facilities, and living standards. The categories are ordered i.e. Tier 1 > Tier 2 > Tier 3
- Occupation: Occupation of customer
- Gender: Gender of customer

- NumberOfPersonVisiting: Total number of persons planning to take the trip with the customer
- PreferredPropertyStar: Preferred hotel property rating by customer
- MaritalStatus: Marital status of customer
- NumberOfTrips: Average number of trips in a year by customer
- Passport: The customer has a passport or not (0: No, 1: Yes)
- OwnCar: Whether the customers own a car or not (0: No, 1: Yes)
- NumberOfChildrenVisiting: Total number of children with age less than 5 planning to take the trip with the customer
- Designation: Designation of the customer in the current organization
- MonthlyIncome: Gross monthly income of the customer

Customer interaction data:

- PitchSatisfactionScore: Sales pitch satisfaction score
- ProductPitched: Product pitched by the salesperson
- NumberOfFollowups: Total number of follow-ups has been done by the salesperson after the sales pitch
- DurationOfPitch: Duration of the pitch by a salesperson to the customer

---

Loading libraries

In [1]:
```
# this will help in making the Python code more structured automatically (goo
%load_ext nb_black
```

In [2]:
```
# silence unnecessary warnings
import warnings

warnings.filterwarnings("ignore")

# Libraries to help with read, manipulation and visualization data
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# To build sklearn / xgboost model
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import (
    BaggingClassifier,
    RandomForestClassifier,
    RandomForestClassifier,
    GradientBoostingClassifier,
    AdaBoostClassifier,
    StackingClassifier,
)
from xgboost import XGBClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.linear_model import LogisticRegression
```

```
# Removes the limit from the number of displayed columns and rows.
# This is so I can see the entire dataframe when I  print it
pd.set_option("display.max_columns", None)
# pd.set_option('display.max_rows', None)
pd.set_option("display.max_rows", 500)
```

Import Dataset

In [3]:
```
tourism = pd.ExcelFile("Tourism.xlsx")
# see all sheet names
print(
    f"There are \033[1m{tourism.sheet_names} \033[m sheet on the excel file,
)
```

```
There are ['Data Dict', 'Tourism']  sheet on the excel file, let's check them to see which one has the data we gonna use.
```

In [4]:
```
tourism.parse("Data Dict").head()  # reading a specific sheet to DataFrame
```

Out[4]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3
0	NaN	Data	Variable	Discerption
1	NaN	Tourism	CustomerID	Unique customer ID
2	NaN	Tourism	ProdTaken	Whether the customer has purchased a package o...
3	NaN	Tourism	Age	Age of customer
4	NaN	Tourism	TypeofContact	How customer was contacted (Company Invited or...

Observation:

- `Data Dict` sheeet is only the summarize of the variables and discerption.

In [5]:
```
tourism.parse("Tourism").head()
```

Out[5]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gende
0	200000	1	41.0	Self Enquiry	3	6.0	Salaried	Femal
1	200001	0	49.0	Company Invited	1	14.0	Salaried	Mal
2	200002	1	37.0	Self Enquiry	1	8.0	Free Lancer	Mal
3	200003	0	33.0	Company Invited	1	9.0	Salaried	Femal
4	200004	0	NaN	Self Enquiry	1	8.0	Small Business	Mal

Observation:

- `Tourism` sheeet has the informations that we need to build our project.

In [6]:
```
# Load the data into pandas dataframe
tourism = pd.read_excel("Tourism.xlsx", sheet_name=1)  # choosing the second

# Coping the data to another variable to avoid any changes to the original da
df = tourism.copy()
```

Overview of the data

In [7]:
```
# Understanding the shape of the data
print(
    f"There are \033[1;4m{df.shape[0]}\033[m rows and \033[1;4m{df.shape[1]}\
)

# Look at 15 random rows
# Setting the random seed via np.random.seed to get the same random results e
np.random.seed(1)
df.sample(n=15)
```

There are **4888** rows and **20** columns.

Out[7]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Ge
3015	203015	0	27.0	Company Invited	1	7.0	Salaried	Fe
1242	201242	0	40.0	Self Enquiry	3	13.0	Small Business	
3073	203073	0	29.0	Self Enquiry	2	15.0	Small Business	
804	200804	0	48.0	Company Invited	1	6.0	Small Business	
3339	203339	0	32.0	Self Enquiry	1	18.0	Small Business	
3080	203080	1	36.0	Company Invited	1	32.0	Salaried	Fe
2851	202851	0	46.0	Self Enquiry	1	17.0	Salaried	
2883	202883	1	32.0	Company Invited	1	27.0	Salaried	
1676	201676	0	22.0	Self Enquiry	1	11.0	Salaried	
1140	201140	0	44.0	Self Enquiry	1	13.0	Small Business	Fe
748	200748	1	26.0	Company Invited	3	35.0	Small Business	
2394	202394	1	NaN	Company Invited	1	8.0	Salaried	Fe
4881	204881	1	41.0	Self Enquiry	2	25.0	Salaried	
3415	203415	0	52.0	Self Enquiry	1	18.0	Large Business	Fe
2253	202253	0	NaN	Self Enquiry	1	13.0	Large Business	Fe

Observation:

- `ProdTaken` : is our target variable.
- Some columns like Age and NumberOfChildrenVisiting has some NaN values, we gonna treat it on our Missing Values steeps.
- `Age` seems to be float, we gonna convert it to int.

In [8]:
```
# Analyzing the % that target variable is distributed on data set.
df["ProdTaken"].value_counts() / len(df["ProdTaken"])
```

Out[8]:
```
0    0.811784
1    0.188216
Name: ProdTaken, dtype: float64
```

> We have an imbalanced data set, with ***81.2%*** of customers that didn't buy a travel package and only ***18.8%*** that bought it.

In [9]:
```
# Checking the data types of the columns for the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   CustomerID                4888 non-null   int64  
 1   ProdTaken                 4888 non-null   int64  
 2   Age                       4662 non-null   float64
 3   TypeofContact             4863 non-null   object 
 4   CityTier                  4888 non-null   int64  
 5   DurationOfPitch           4637 non-null   float64
 6   Occupation                4888 non-null   object 
 7   Gender                    4888 non-null   object 
 8   NumberOfPersonVisiting    4888 non-null   int64  
 9   NumberOfFollowups         4843 non-null   float64
 10  ProductPitched            4888 non-null   object 
 11  PreferredPropertyStar     4862 non-null   float64
 12  MaritalStatus             4888 non-null   object 
 13  NumberOfTrips             4748 non-null   float64
 14  Passport                  4888 non-null   int64  
 15  PitchSatisfactionScore    4888 non-null   int64  
 16  OwnCar                    4888 non-null   int64  
 17  NumberOfChildrenVisiting  4822 non-null   float64
 18  Designation               4888 non-null   object 
 19  MonthlyIncome             4655 non-null   float64
dtypes: float64(7), int64(7), object(6)
memory usage: 763.9+ KB
```

Observation:

- ***Age***: is int not float, we gonna convert it;
- ***TypeofContact, Occupation, Gender, ProductPitched, MaritalStatus, Designation*** represented as object but that we really will want to be as categorical (it reduces the memory usage);

- ***Some columns***: has NaN values that need to be treated (We will check it using isnull on next step).

In [10]:
```
# Converting all columns with data type 'object' to 'category'
cat_col = df.select_dtypes(include=["object"]).columns.tolist()
df[cat_col] = df[cat_col].astype("category")
```

In [11]:
```
# Checking the data types of the columns for the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   CustomerID                4888 non-null   int64   
 1   ProdTaken                 4888 non-null   int64   
 2   Age                       4662 non-null   float64 
 3   TypeofContact             4863 non-null   category
 4   CityTier                  4888 non-null   int64   
 5   DurationOfPitch           4637 non-null   float64 
 6   Occupation                4888 non-null   category
 7   Gender                    4888 non-null   category
 8   NumberOfPersonVisiting    4888 non-null   int64   
 9   NumberOfFollowups         4843 non-null   float64 
 10  ProductPitched            4888 non-null   category
 11  PreferredPropertyStar     4862 non-null   float64 
 12  MaritalStatus             4888 non-null   category
 13  NumberOfTrips             4748 non-null   float64 
 14  Passport                  4888 non-null   int64   
 15  PitchSatisfactionScore    4888 non-null   int64   
 16  OwnCar                    4888 non-null   int64   
 17  NumberOfChildrenVisiting  4822 non-null   float64 
 18  Designation               4888 non-null   category
 19  MonthlyIncome             4655 non-null   float64 
dtypes: category(6), float64(7), int64(7)
memory usage: 564.4 KB
```

In [12]:
```
# looking at which columns have the most missing values
df.isnull().sum().sort_values(ascending=False)
```

Out[12]:
```
DurationOfPitch             251
MonthlyIncome               233
Age                         226
NumberOfTrips               140
NumberOfChildrenVisiting     66
NumberOfFollowups            45
PreferredPropertyStar        26
TypeofContact                25
Passport                      0
MaritalStatus                 0
ProductPitched                0
Designation                   0
NumberOfPersonVisiting        0
Gender                        0
Occupation                    0
PitchSatisfactionScore        0
CityTier                      0
OwnCar                        0
ProdTaken                     0
CustomerID                    0
dtype: int64
```

Summary of the dataset

In [13]:
```
df.describe().T
```

Out[13]:

	count	mean	std	min	25%	50%
CustomerID	4888.0	202443.500000	1411.188388	200000.0	201221.75	202443.5
ProdTaken	4888.0	0.188216	0.390925	0.0	0.00	0.0
Age	4662.0	37.622265	9.316387	18.0	31.00	36.0
CityTier	4888.0	1.654255	0.916583	1.0	1.00	1.0
DurationOfPitch	4637.0	15.490835	8.519643	5.0	9.00	13.0
NumberOfPersonVisiting	4888.0	2.905074	0.724891	1.0	2.00	3.0
NumberOfFollowups	4843.0	3.708445	1.002509	1.0	3.00	4.0
PreferredPropertyStar	4862.0	3.581037	0.798009	3.0	3.00	3.0
NumberOfTrips	4748.0	3.236521	1.849019	1.0	2.00	3.0
Passport	4888.0	0.290917	0.454232	0.0	0.00	0.0
PitchSatisfactionScore	4888.0	3.078151	1.365792	1.0	2.00	3.0
OwnCar	4888.0	0.620295	0.485363	0.0	0.00	1.0
NumberOfChildrenVisiting	4822.0	1.187267	0.857861	0.0	1.00	1.0
MonthlyIncome	4655.0	23619.853491	5380.698361	1000.0	20346.00	22347.0

- CustomerID: The ID attribute does not add any information to our analysis as all the values are unique. There is no association between a person's customer ID and customer more likely to purchase travel packge.
- Age: Average age of customers is 38 years, age of customers has a wide range from 18 to 61 years.
- DurationOfPitch: Average duration is 15.49 but the duration has a wide range from 5 to 127.
- NumberOfPersonVisiting: Min 1 Max 5 with a mean around 3
- NumberOfFollowups: 25% of customers had 3 or less followups while 75% of customers had between 4 to 6.
- PreferredPropertyStar: at least 75% of customers preffers property between 3 and 4 stars.
- NumberOfTrips: mim 1 max 22, lets check this 22 to see it is a outliers or real data, considering is a huge number of trip for a customer per **year**
- Passport: 29.09% of customer has passaport.
- PitchSatisfactionScore: Mean satisfaction is around 3.
- OwnCar: 62.02% of customers has there on car.
- NumberOfChildrenVisiting: usually customers planning in take 1 children with age less than 5. Max is 3 and we gonna check the number to see if it is an outlier.
- MonthlyIncome: Customers income ranges between 1000 to 98678 with mean 23619. Lets take a look closer on this amounts to check for outliers (min and max seems to be

outliers).

`PreferredPropertyStar` , `PitchSatisfactionScore` are Ordinal Categorical we gonna keep it as numerical and procide with `Label Encoding` considering that there is a sense of order on the values.

`Passaport` , `OwnCar` are Binary Categorical, we gonna keep it as numerical and procide with `Label Encoding` .

In [14]:
```
df.describe(include="category").T
```

Out[14]:

	count	unique	top	freq
TypeofContact	4863	2	Self Enquiry	3444
Occupation	4888	4	Salaried	2368
Gender	4888	3	Male	2916
ProductPitched	4888	5	Basic	1842
MaritalStatus	4888	4	Married	2340
Designation	4888	5	Executive	1842

In [15]:
```
for i in cat_col:
    print(f"Unique values in \033[1m{i}\033[m are:")
    print(df[i].value_counts())
    print("\n", "*" * 50, "\n")
```

```
Unique values in TypeofContact are:
Self Enquiry       3444
Company Invited    1419
Name: TypeofContact, dtype: int64

 **************************************************

Unique values in Occupation are:
Salaried          2368
Small Business    2084
Large Business     434
Free Lancer          2
Name: Occupation, dtype: int64

 **************************************************

Unique values in Gender are:
Male       2916
Female     1817
Fe Male     155
Name: Gender, dtype: int64

 **************************************************

Unique values in ProductPitched are:
Basic           1842
Deluxe          1732
Standard         742
Super Deluxe     342
King             230
Name: ProductPitched, dtype: int64

 **************************************************
```

```
Unique values in MaritalStatus are:
Married      2340
Divorced      950
Single        916
Unmarried     682
Name: MaritalStatus, dtype: int64

 **************************************************

Unique values in Designation are:
Executive         1842
Manager           1732
Senior Manager     742
AVP                342
VP                 230
Name: Designation, dtype: int64

 **************************************************
```

- TypeofContact: Most of the customers contacted the company (Self Enquiry).
- Occupation: 91% of the customers are Salaried or have a small busines.
- Gender: 60% of the customers are Male. ***Female category comprises Female and Fe Male, we gonna fix it on next steep***.
- Product Pitched: There are 5 product type and the most pitched was Basic, followed by Deluxe.
- MaritalStatus: There are 4 sub category and most of the customers are Married(48%).
- Designation: Most of the customers are Executive or Manager.

Feature Engineering

Fixing Gender category

In [16]:
```
# Fixing Fe Male subcategory on Gender category
df.Gender = df.Gender.apply(lambda x: "Female" if x == "Fe Male" else x)
```

In [17]:
```
# Checking gender variable
df["Gender"].value_counts()
```

Out[17]:
```
Male      2916
Female    1972
Name: Gender, dtype: int64
```

In [18]:
```
# converting Gender to Category
df["Gender"] = df["Gender"].astype("category")
```

Dropping Customer ID

In [19]:
```
# Dropping CustomerID
df = df.drop(["CustomerID"], axis=1)
```

In [20]:
```
df.head()
```

Out[20]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfI
0	1	41.0	Self Enquiry	3	6.0	Salaried	Female	
1	0	49.0	Company Invited	1	14.0	Salaried	Male	
2	1	37.0	Self Enquiry	1	8.0	Free Lancer	Male	
3	0	33.0	Company Invited	1	9.0	Salaried	Female	
4	0	NaN	Self Enquiry	1	8.0	Small Business	Male	

Checking Duration Of Pitch extreme values

In [21]:
```
# Checking Duration of Pitch mean
df["DurationOfPitch"].mean()
```

Out[21]:
```
15.490834591330602
```

In [22]:
```
# Check Duration Of Pitch extreme values
df.sort_values(by=["DurationOfPitch"], ascending=False).head(5)
```

Out[22]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
3878	0	53.0	Company Invited	3	127.0	Salaried	Male	
1434	0	NaN	Company Invited	3	126.0	Salaried	Male	
2796	0	49.0	Self Enquiry	3	36.0	Small Business	Female	
2868	0	58.0	Self Enquiry	3	36.0	Small Business	Male	
2648	1	39.0	Self Enquiry	1	36.0	Small Business	Male	

In [23]:
```
# Check for type of Contact equal a Company Invited and then Duration Of Pitc
df[df.TypeofContact == "Company Invited"].sort_values(
    by=["DurationOfPitch"], ascending=False
).head(5)
```

Out[23]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
3878	0	53.0	Company Invited	3	127.0	Salaried	Male	
1434	0	NaN	Company Invited	3	126.0	Salaried	Male	
2505	0	39.0	Company Invited	1	36.0	Salaried	Female	
2853	0	43.0	Company Invited	1	36.0	Salaried	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
3200	0	33.0	Company Invited	1	36.0	Small Business	Female	

- It looks like the two extreme values are outliers, problably who typed it, accidently put 1 in front of it.
- It doen't make a lot of sense spend all this time with customers who did not buy the packege and had only 4 and 3 Followups.
- If we compare it, with others customer who the Company Invited, the mean Duration of pitch per follow up is around 9, and on this two extrem value the average is higher than 30.

> We gonna remove the number 1 and keep 27 and 26 respectivily.

In [24]:
```
# Replacing #DurationOfPitch in row index 3878 form 127 to 27
df.loc[3878, "DurationOfPitch"] = 27
```

In [25]:
```
# Replacing #DurationOfPitch in row index 1434 from 126 to 26
df.loc[1434, "DurationOfPitch"] = 26
```

In [26]:
```
# Checking data after replacement
df.sort_values(by=["DurationOfPitch"], ascending=False).head(3)
```

Out[26]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
4662	1	27.0	Company Invited	3	36.0	Small Business	Male	
3200	0	33.0	Company Invited	1	36.0	Small Business	Female	
2869	0	51.0	Self Enquiry	1	36.0	Salaried	Male	

Checking Number of Trips extreme values

In [27]:
```
df.sort_values(by=["NumberOfTrips"], ascending=False).head(5)
```

Out[27]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
3260	0	40.0	Company Invited	1	16.0	Salaried	Male	
816	0	39.0	Company Invited	1	15.0	Salaried	Male	
2829	1	31.0	Company Invited	1	11.0	Large Business	Male	
385	1	30.0	Company Invited	1	10.0	Large Business	Male	
3074	0	23.0	Self Enquiry	1	7.0	Salaried	Male	

- The two first customer travels 22 and 21 times per year (respectively), which is not common for Salaried employees. Unless they work traveling.
- 3rd and 4th customer travel 20 and 19 times per year (respectively), which is ok for Executive of a Large Business, but it still a lot.

> We not gonna treat this informations as outliers. .

In [28]:
```
df.groupby(["Occupation"])["NumberOfTrips"].mean()
```

Out[28]:
```
Occupation
Free Lancer       7.500000
Large Business    3.456019
Salaried          3.221062
Small Business    3.202877
Name: NumberOfTrips, dtype: float64
```

In [29]:
```
df.groupby(["Occupation"])["NumberOfTrips"].max()
```

Out[29]:
```
Occupation
Free Lancer        8.0
Large Business    20.0
Salaried          22.0
Small Business     8.0
Name: NumberOfTrips, dtype: float64
```

Checking Low Monthly Income

In [30]:
```
df.sort_values(by=["MonthlyIncome"]).head(5)
```

Out[30]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
142	0	38.0	Self Enquiry	1	9.0	Large Business	Female	
2586	0	39.0	Self Enquiry	1	10.0	Large Business	Female	
513	1	20.0	Self Enquiry	1	16.0	Small Business	Male	
1983	1	20.0	Self Enquiry	1	16.0	Small Business	Male	
2197	0	18.0	Company Invited	1	11.0	Salaried	Male	

- The two lowest income are from Female customer that run they own `Large Business` , both single and travels 4 and 5 times in a year. So this income is an outlier.

In [31]:
```
# Checking the min Monthly Income for each Occupation
df.groupby(["Occupation"])["MonthlyIncome"].min()
```

Out[31]:
```
Occupation
Free Lancer       17090.0
```

```
Large Business     1000.0
Salaried          16051.0
Small Business    16009.0
Name: MonthlyIncome, dtype: float64
```

In [32]:
```
# Checking the min Monthly Income for Large Business
df[df.Occupation == "Large Business"].sort_values(by="MonthlyIncome").head(5)
```

Out[32]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
142	0	38.0	Self Enquiry	1	9.0	Large Business	Female	
2586	0	39.0	Self Enquiry	1	10.0	Large Business	Female	
1365	1	29.0	Company Invited	3	30.0	Large Business	Male	
1052	0	30.0	Company Invited	1	13.0	Large Business	Male	
977	0	34.0	Company Invited	1	32.0	Large Business	Female	

Considering that the min for Large Business is `16091.0` , we gonna replace the other 2 values with the min for this category.

In [33]:
```
df.MonthlyIncome = df.MonthlyIncome.apply(lambda x: 16091.0 if x < 5000 else
```

In [34]:
```
# Checking the min Monthly Income for each Occupation after replaicing it
df.groupby(["Occupation"])["MonthlyIncome"].min()
```

Out[34]:
```
Occupation
Free Lancer       17090.0
Large Business    16091.0
Salaried          16051.0
Small Business    16009.0
Name: MonthlyIncome, dtype: float64
```

Checking High Monthly Income

In [35]:
```
df.sort_values(by="MonthlyIncome", ascending=False).head(5)
```

Out[35]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2482	0	37.0	Self Enquiry	1	12.0	Salaried	Female	
38	0	36.0	Self Enquiry	1	11.0	Salaried	Female	
4104	0	53.0	Self Enquiry	1	7.0	Salaried	Male	
2634	0	53.0	Self Enquiry	1	7.0	Salaried	Male	
4660	0	42.0	Company Invited	1	14.0	Salaried	Female	

In [36]:
```
df.groupby(["Designation", "Occupation"])["MonthlyIncome"].min()
```

Out[36]:
```
Designation     Occupation
AVP             Free Lancer          NaN
                Large Business    28120.0
                Salaried          21151.0
                Small Business    17705.0
Executive       Free Lancer       17090.0
                Large Business    16091.0
                Salaried          16051.0
                Small Business    16009.0
Manager         Free Lancer          NaN
                Large Business    16091.0
                Salaried          17272.0
                Small Business    17042.0
Senior Manager  Free Lancer          NaN
                Large Business    17372.0
                Salaried          17372.0
                Small Business    17875.0
VP              Free Lancer          NaN
                Large Business    34232.0
                Salaried          33041.0
                Small Business    17517.0
Name: MonthlyIncome, dtype: float64
```

In [37]:
```
df.groupby(["Designation", "Gender"])["MonthlyIncome"].mean()
```

Out[37]:
```
Designation     Gender
AVP             Female    32211.587500
                Male      32266.945055
Executive       Female    20163.681115
                Male      19809.581605
Manager         Female    22527.410606
                Male      22754.277538
Senior Manager  Female    26851.558282
                Male      26470.197115
VP              Female    35572.951220
                Male      36048.486486
Name: MonthlyIncome, dtype: float64
```

- Considering that they are Salaried Executives, we gonna replaced by the mean 20.000

In [38]:
```
# Replacing #High Income in row index 3878 form 127 to 27
df.loc[2482, "MonthlyIncome"] = 20000
df.loc[38, "MonthlyIncome"] = 20000
```

In [39]:
```
df.sort_values(by="MonthlyIncome", ascending=False).head()
```

Out[39]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2634	0	53.0	Self Enquiry	1	7.0	Salaried	Male	
4104	0	53.0	Self Enquiry	1	7.0	Salaried	Male	
3190	0	42.0	Company Invited	1	14.0	Salaried	Female	
4660	0	42.0	Company Invited	1	14.0	Salaried	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
3295	0	57.0	Self Enquiry	1	11.0	Large Business	Female	

EDA

Univariate analysis

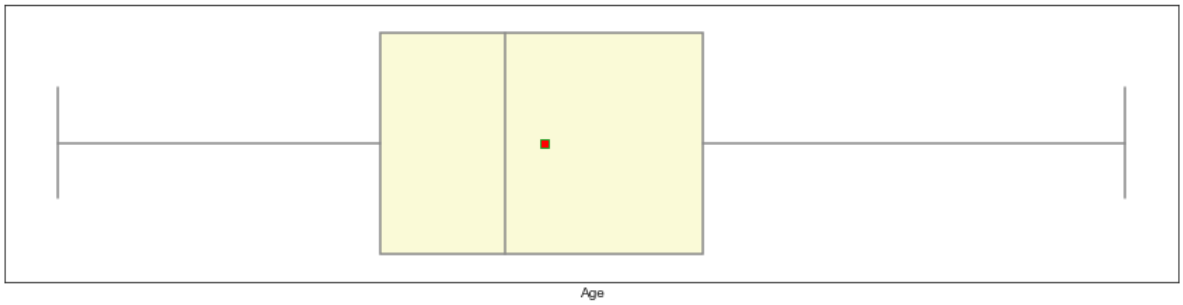
In [40]:
```
# A function to create boxplot and histogram for any input numerical


def hist_boxplot(dataframe, figsize=(15, 8), bins=None):
    """
    This function takes the numerical column as the input and returns the box
    dataframe: 1-d feature array
    figsize: size of fig (default (13,8))
    bins: number of bins (default None / auto)
    """

    # Figure aesthetics
    sns.set_style("white")

    # Creating the 2 subplots
    fig, (ax_box, ax_hist) = plt.subplots(nrows=2, sharex=True, figsize=figsi

    # Boxplot will be created and a red square will indicate the mean value o
    sns.boxplot(
        dataframe,
        ax=ax_box,
        showmeans=True,
        meanprops={"marker": "s", "markerfacecolor": "red"},
        color="xkcd:eggshell",
    )

    # For histogram
    sns.distplot(
        dataframe, kde=F, ax=ax_hist, color="lightblue", bins=bins
    ) if bins else sns.distplot(dataframe, kde=False, ax=ax_hist, color="ligh

    # Add mean to the histogram
    ax_hist.axvline(np.mean(dataframe), color="r", linestyle="dotted")
    # Add median to the histogram
    ax_hist.axvline(np.median(dataframe), color="gray", linestyle="solid")
```

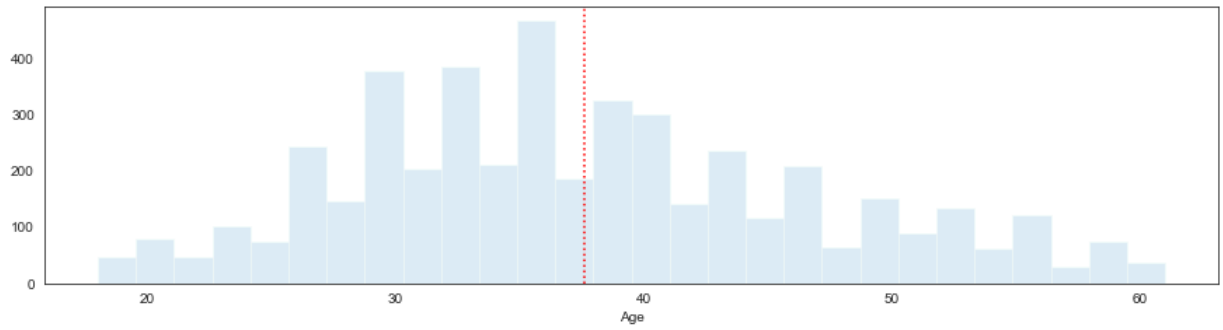
In [41]:
```
# Function to create barplots that indicate percentage for each category.


def perc_on_bar(dataframe):
    """
    plot
    feature: categorical feature
    the function won't work if a column is passed in hue parameter
    """
```

```
    total = len(dataframe)  # length of the column
    plt.figure(figsize=(15, 5))
    ax = sns.countplot(dataframe, palette="Paired")
    for p in ax.patches:
        percentage = "{:.1f}%".format(
            100 * p.get_height() / total
        )  # percentage of each class of the category
        x = p.get_x() + p.get_width() / 2 - 0.05  # width of the plot
        y = p.get_y() + p.get_height()  # hieght of the plot

        ax.annotate(percentage, (x, y), size=12)  # annotate the percantage
    plt.show()  # show the plot
```

Observations on Age

In [42]:
```
hist_boxplot(df.Age)
```

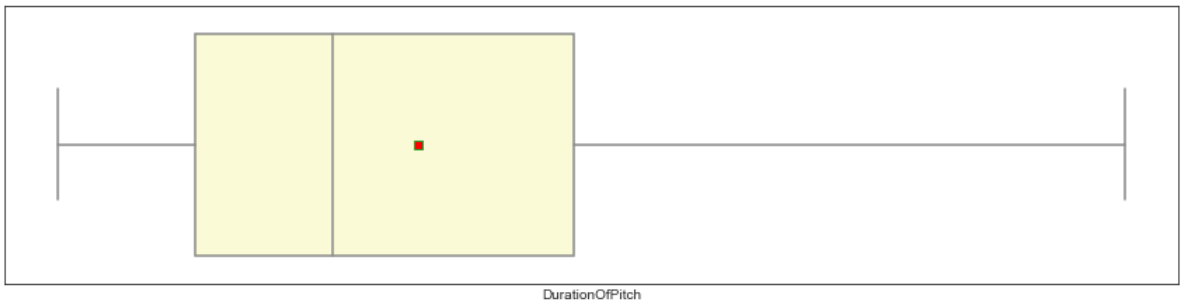


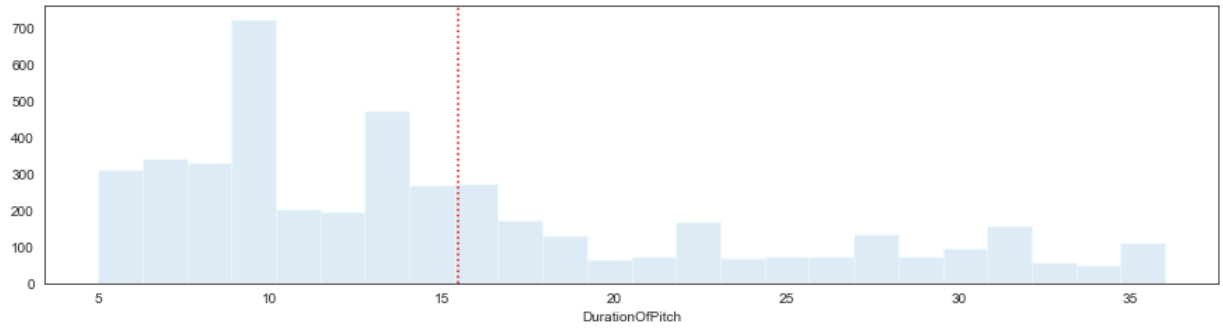


- The distribution of Age is fairly symmetrical about the mean and the median.
- The mean and median age of customers is almost equal to ~37 years.
- Age range between 18 to 61 years.
- There are some miss values

Observations on Duration Of Pitch

In [43]:
```
hist_boxplot(df.DurationOfPitch)
```

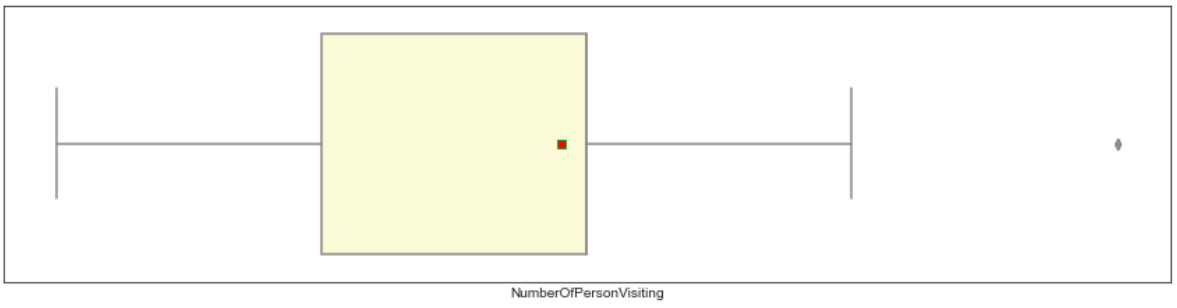


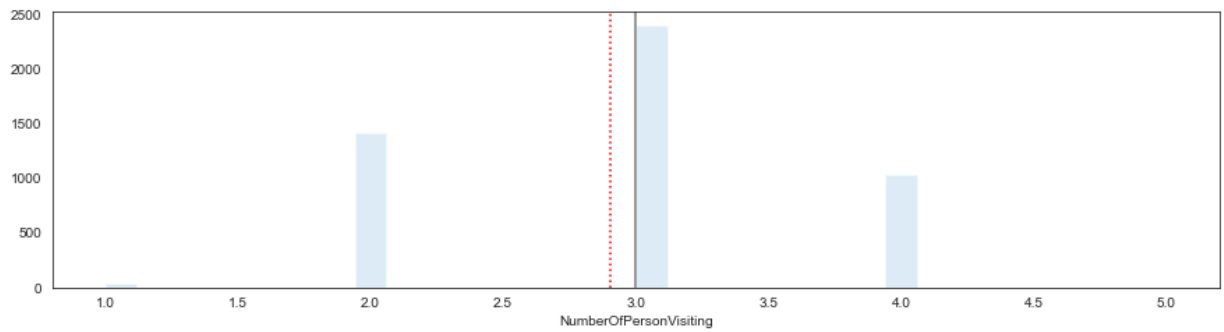


- Min around 5 and max 36 (we already treat the 'outliers')
- Mean of 15.5
- Duration concentrated between 5 to 20
- There are some miss values

Observations on Number Of Person Visiting

In [44]:
```
hist_boxplot(df.NumberOfPersonVisiting)
```

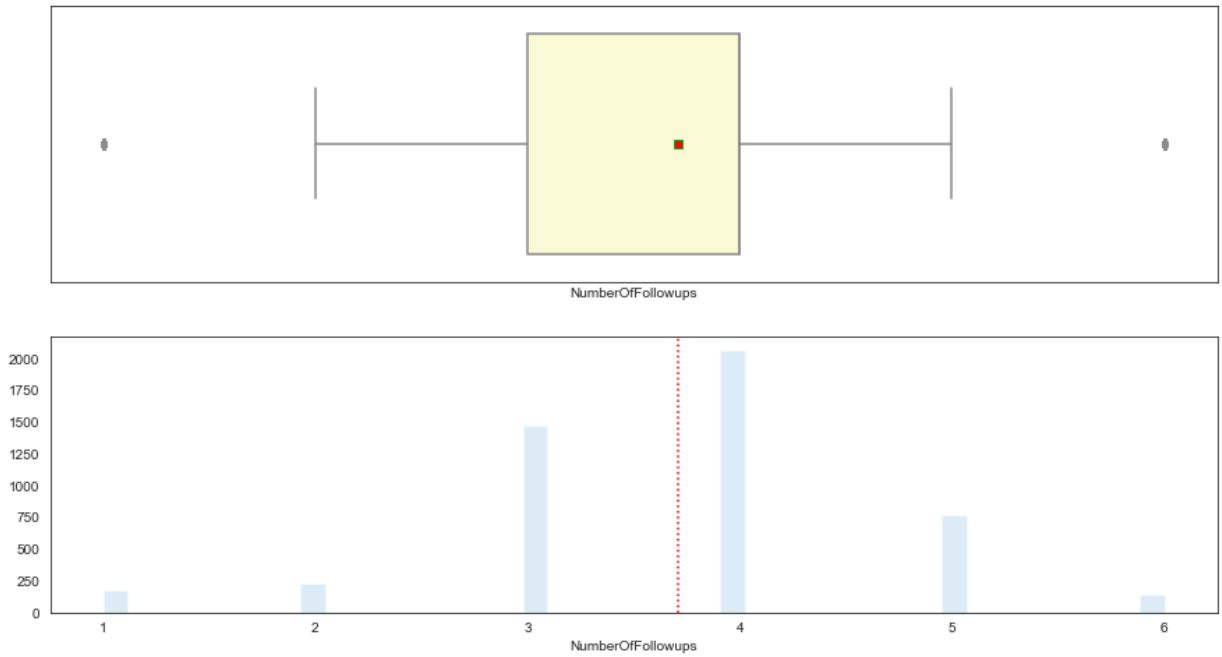




- The distribution of Number of Person Visiting is fairly symmetrical about the mean and the median.
- The mean and median is almost equal to ~ 3
- Min of 1 and max of 5

Observations on Number Of Follow ups

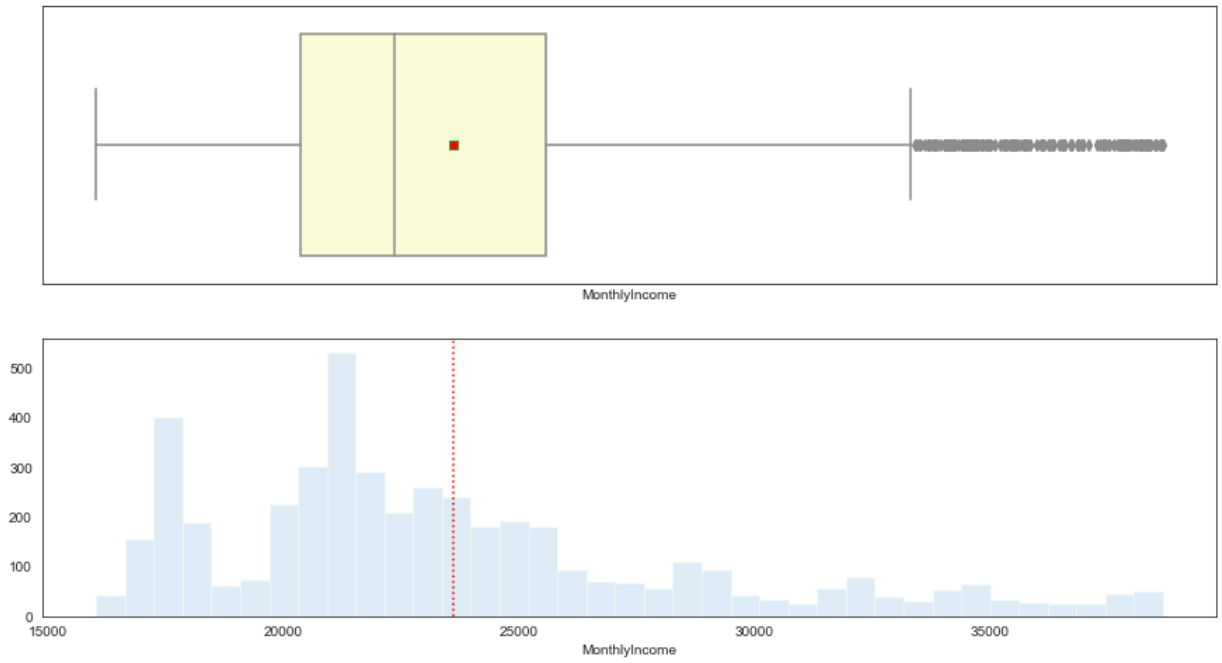
In [45]:
```
hist_boxplot(df.NumberOfFollowups)
```



- Min number of follow ups is 1 and max 6
- Mean and Median around ~ 4
- There are some miss values

Observations on Monthly Income

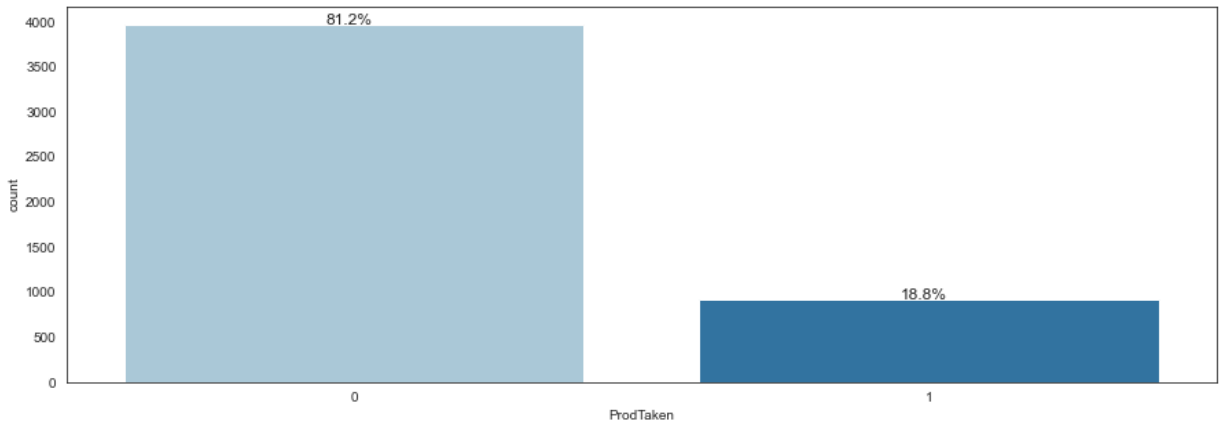
In [46]:
```
hist_boxplot(df.MonthlyIncome)
```



- There are some miss values
- There are some high values that we gonna check if it is an outlier or real information.
- There are a wide range, with min 1000 and ~ 40000
- There are a high frequency of customers with income around 22000

Observations on ProdTaken (target variable)

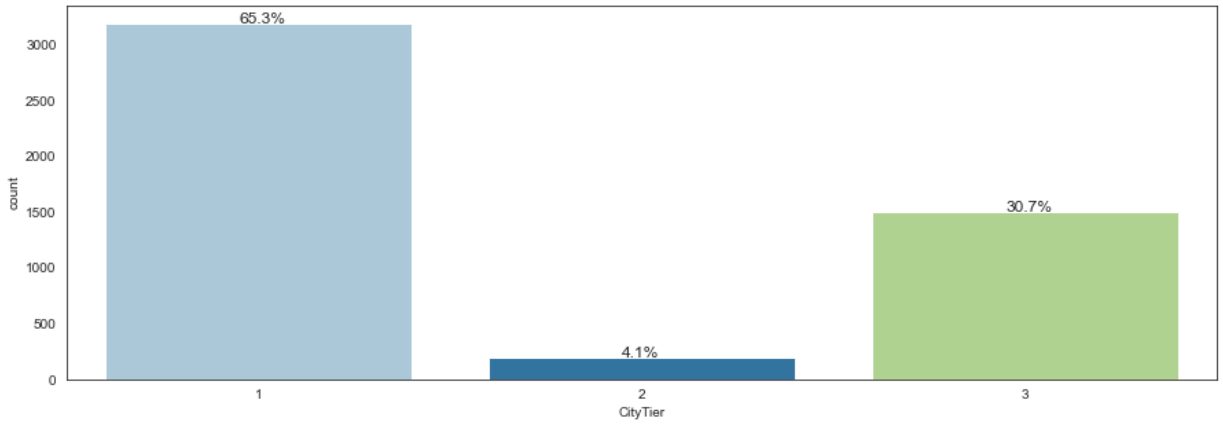
In [47]:
```
perc_on_bar(df.ProdTaken)
```



- On last year, 18.8% of the customers purchased the packages

Observations on CityTier

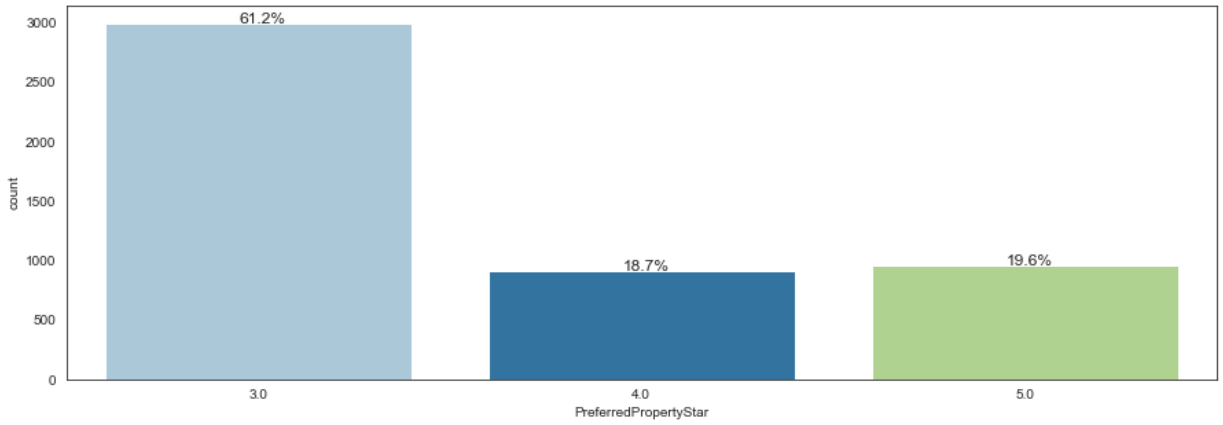
In [48]:
```
perc_on_bar(df.CityTier)
```



- 65.3% of customers belongs to big and developed city
- 30.7% belongs to small and not developed city

Observations on Preferred Property Star

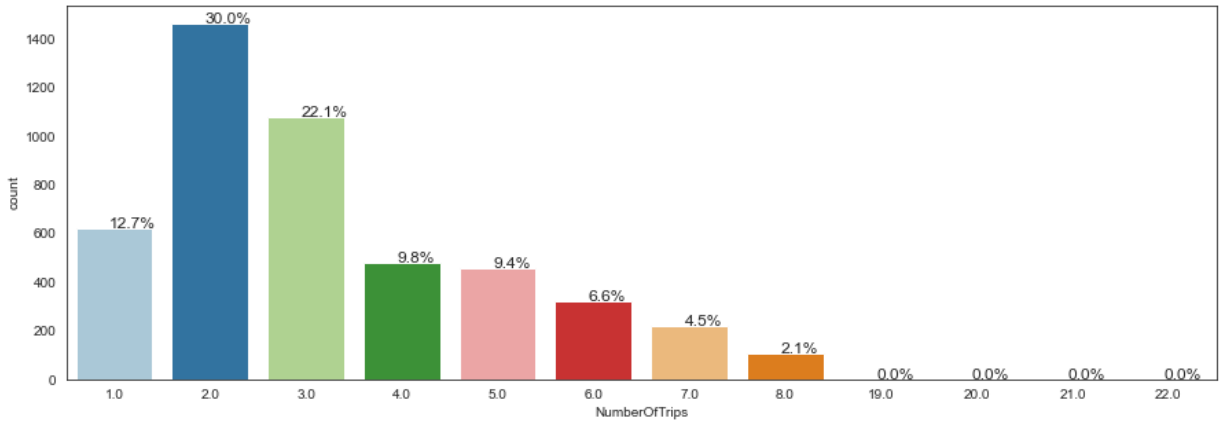
In [49]:
```
perc_on_bar(df.PreferredPropertyStar)
```



- Most of the customers preferred hotel property rating by customer as 3 stars (61.2%).
- 4 and 5 stars has almost the same prefference (18.7% and 19.6% respectivilly).

Observations on Number Of Trips

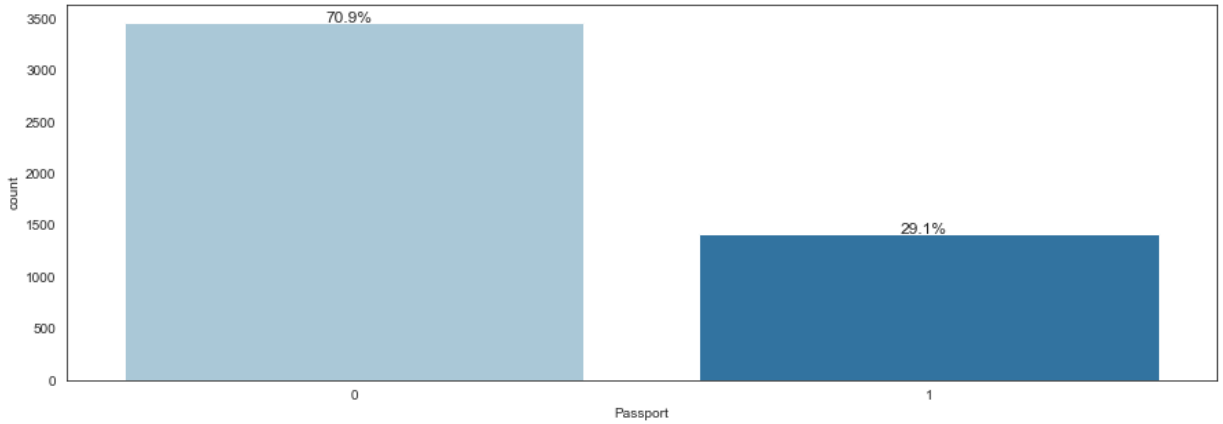
In [50]:
```
perc_on_bar(df.NumberOfTrips)
```



- Wide range between 1 to 22 number of trips (we gonna check for outliers).
- Most os the customers usually travels 2 (30%) and 3 (22.1%) times per year.

Observations on Passport

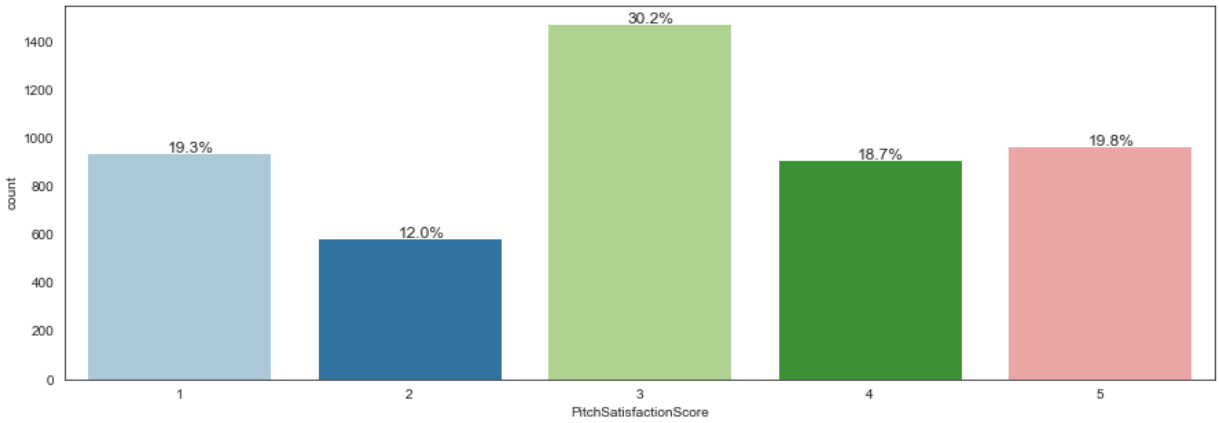
In [51]:
```
perc_on_bar(df.Passport)
```



- 70.9% of customers do not have passaport.
- Only 29.1% of customers has passaport.

Observations on Pitch Satisfaction Score

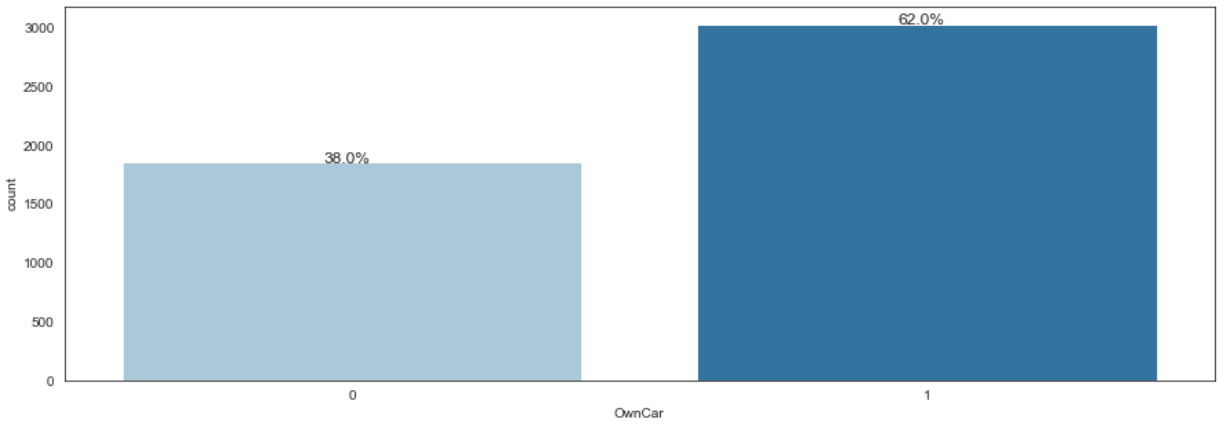
In [52]:
```
perc_on_bar(df.PitchSatisfactionScore)
```



- 31.3% of customers pitch 2 or less as a satusfactuib Score.
- 30.2% of customers pitch 3 and,
- 38.5% of customers pitch 4 or 5.

Observations on Own Car

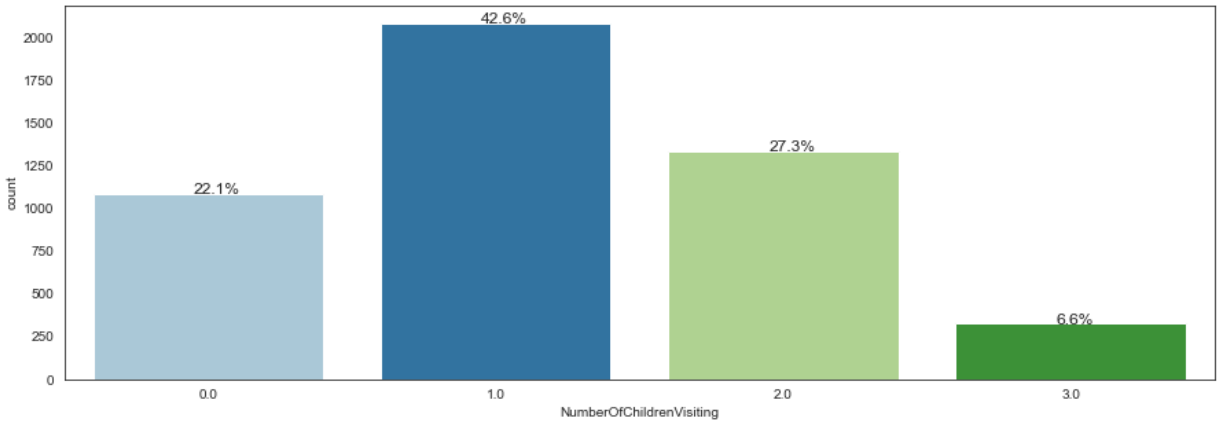
In [53]:
```
perc_on_bar(df.OwnCar)
```



- 62% of customers own a car

Observations on Number Of Children Visiting

In [54]:
```
perc_on_bar(df.NumberOfChildrenVisiting)
```

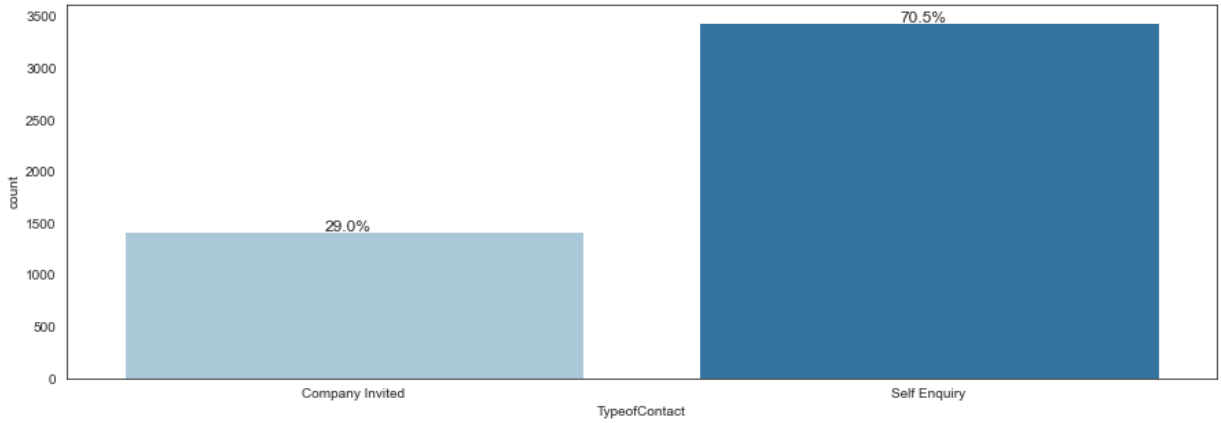


- The most common is 1 children under 5 years traveling with the customer (42.6%).

- Second most common is 2 children (27.3%)
- Travelling with no children were chosen for 22.1% of our customers.

Observations on Type of Contact

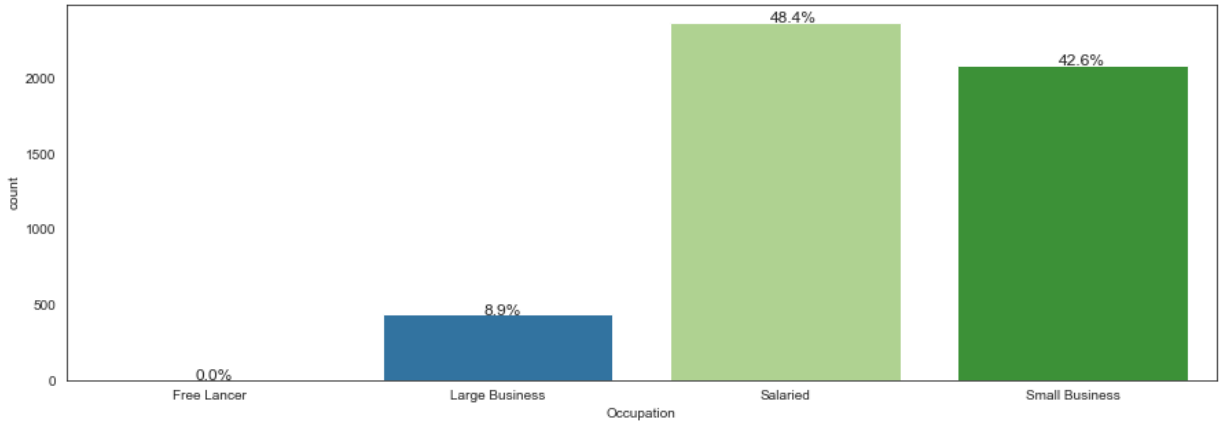
In [55]:
```
perc_on_bar(df.TypeofContact)
```



- 70.5% of customer self enquiry
- Only 29% of customer were contacted by the company

Observations on Occupation

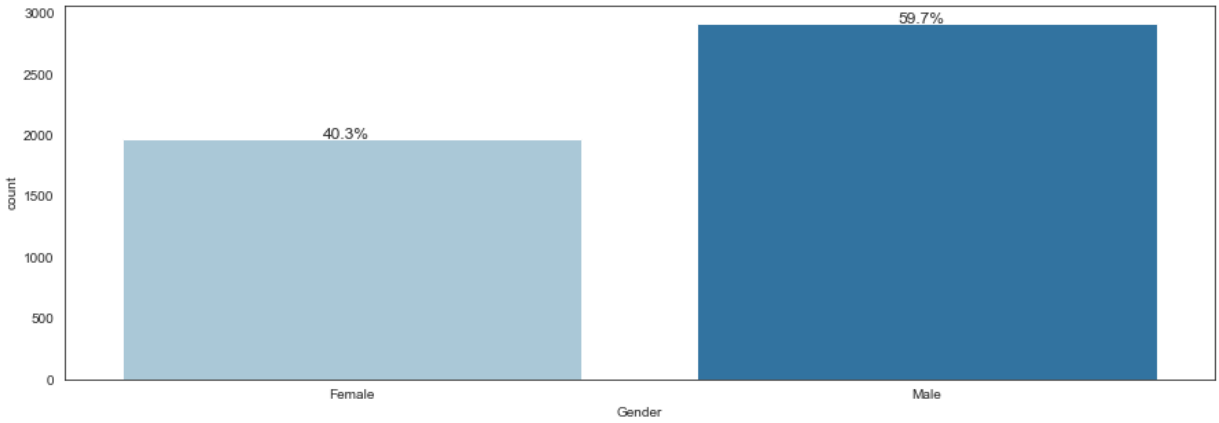
In [56]:
```
perc_on_bar(df.Occupation)
```



- 48.4% of customers are Salaried
- 42.6% have a Small Business

Observations on Gender

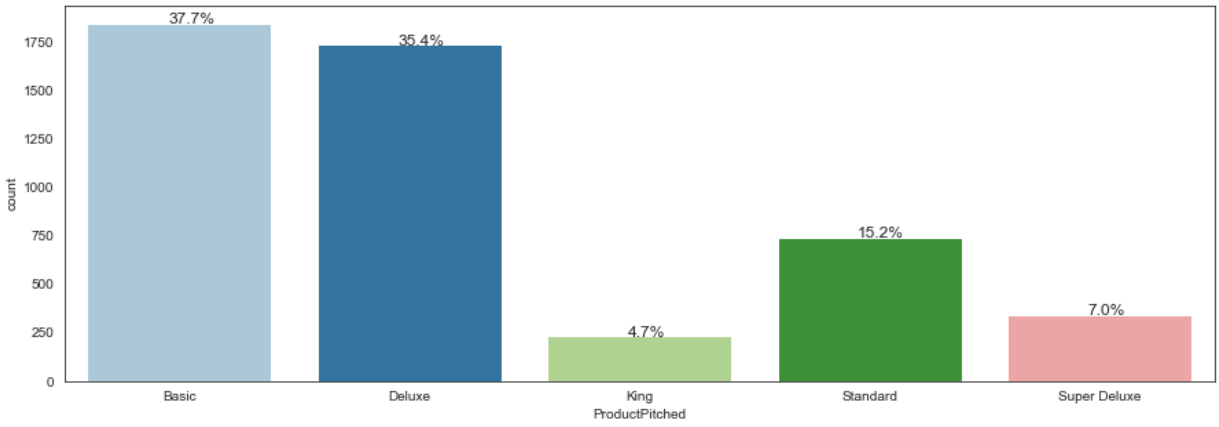
In [57]:
```
perc_on_bar(df.Gender)
```



- 59.7% of customers are Male
- 40.3% of customers are Female

Observations on Product Pitched

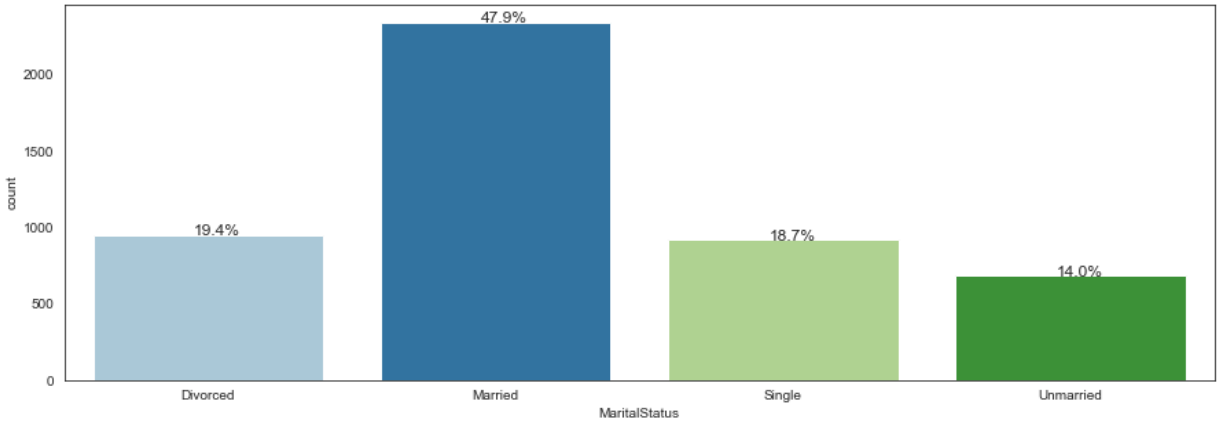
In [58]:
```
perc_on_bar(df.ProductPitched)
```



- 73.1% of Product pitched by the salesperson were Basic or Deluxe travel package.
- 15.2% were Stamdard.
- Less than 10% were Super Deluxe (7%).
- Less than 5% were King (4.7%).

Observations on Marital Status

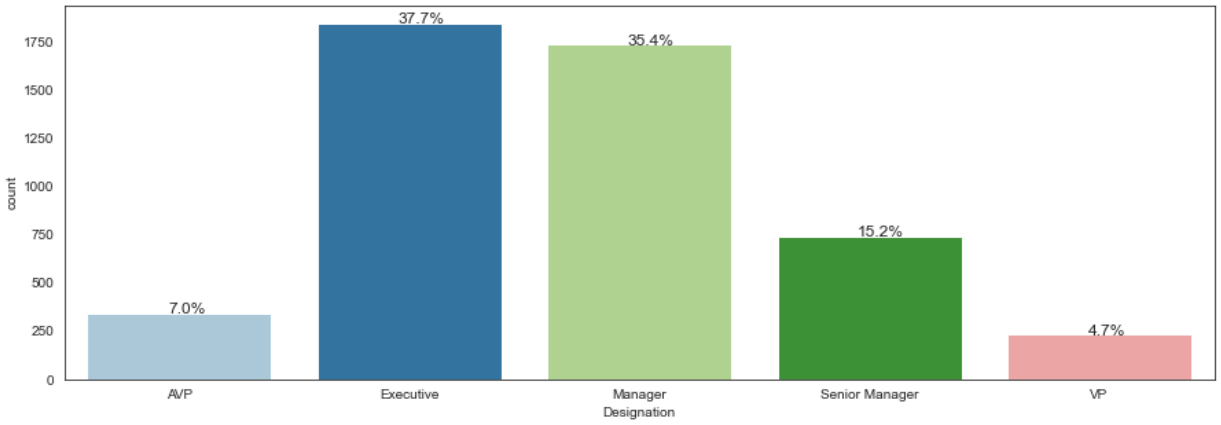
In [59]:
```
perc_on_bar(df.MaritalStatus)
```



- 47.9% of customer are Married
- Followed by 19.4% Divorced and 18.7% Single

Observations on Designation

In [60]:
```
perc_on_bar(df.Designation)
```



- 37.7% of customer are Executive
- 35.4% Manager
- 26.9% other Designation

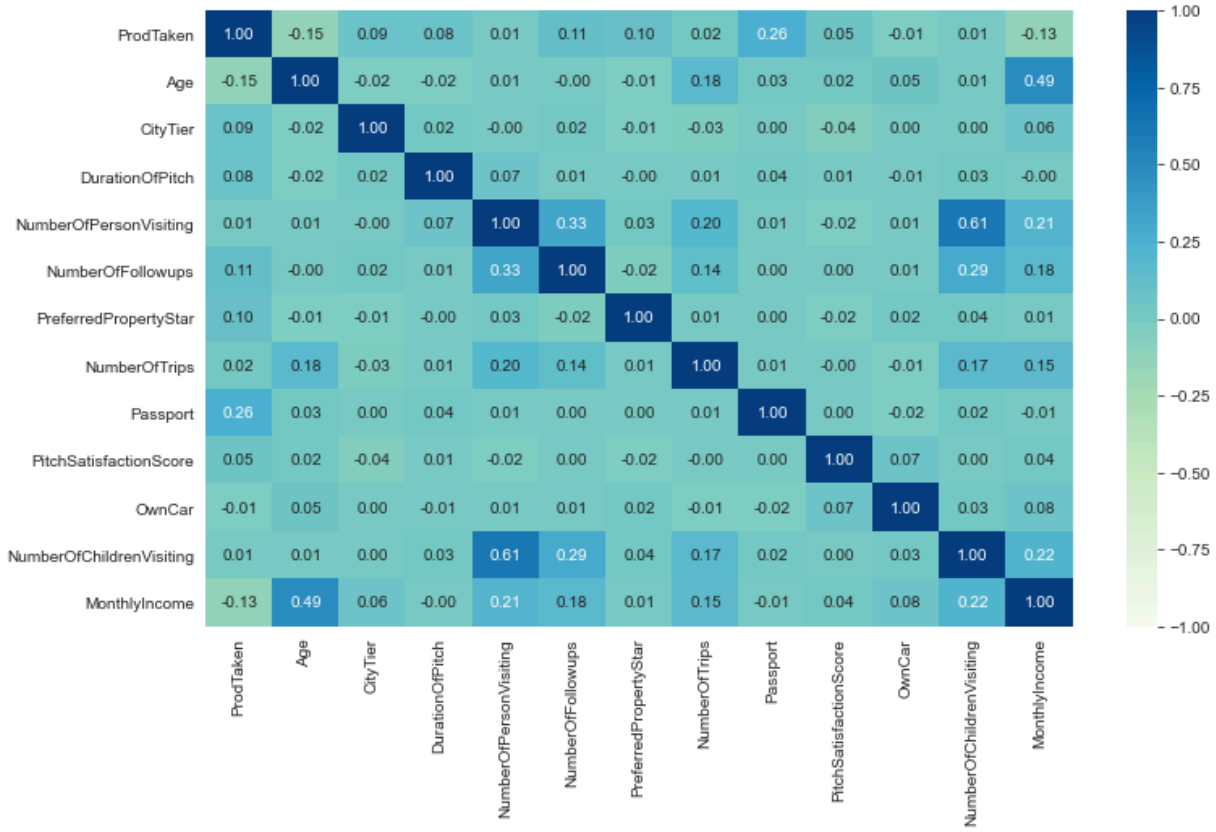
Bivariate Analysis

In [61]:
```
df.corr()["ProdTaken"].sort_values(ascending=False)
```

Out[61]:
```
ProdTaken                 1.000000
Passport                  0.260844
NumberOfFollowups         0.112171
PreferredPropertyStar     0.099577
CityTier                  0.086852
DurationOfPitch           0.083796
PitchSatisfactionScore    0.051394
NumberOfTrips             0.018898
NumberOfPersonVisiting    0.009627
NumberOfChildrenVisiting  0.007421
OwnCar                   -0.011508
MonthlyIncome            -0.133944
Age                      -0.147254
Name: ProdTaken, dtype: float64
```

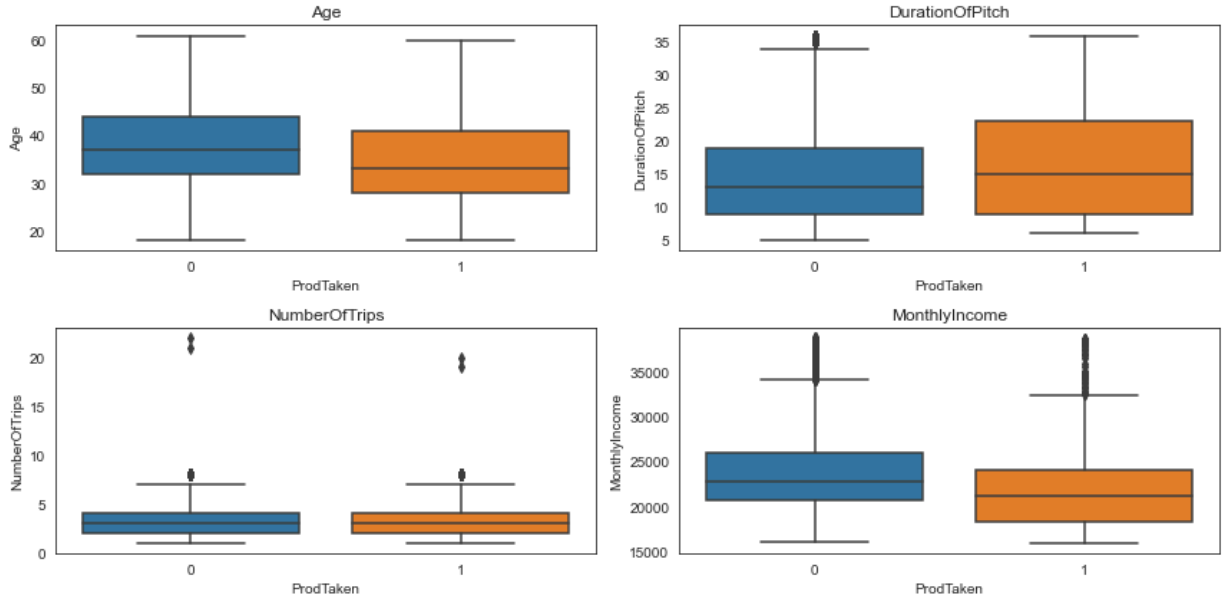
In [62]:
```
plt.figure(figsize=(12, 7))
sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="GnBu")
plt.show
```

Out[62]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



- `OwnCar` , `PitchSatisfactionScore` , `PreferredPropertyStar` , `DurationOfPitch` , `CityTier` , does not show correlation with any of the others variables.
- `ProdTaken` has a 0.26 correlation with `Passaport` , meaning that customer that have passaport are more likehood to buy a travel package.
- `ProdTaken` has negative correlation with `MonthlyIncome` -0. 13 and -0.15 with `Age` , meaning that young customers, that usually has less income, are more likehood to buy a travel package.
- `Age` and `MonthlyIncome` has 0.49 correlation

In [63]:
```
# Creating a list with numerical variables
num_col = []
for col in df.columns:
    if (
        df[col].nunique() > 10
    ):  # filter only numerical variables with more than 10 unique values
        num_col.append(col)

# Boxplot for numerical variables
plt.figure(figsize=(12, 6))
for i, variable in enumerate(num_col):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(df["ProdTaken"], df[variable])
    plt.tight_layout()
    plt.title(variable)
plt.show()
```



Observations:

1. Age vs ProdTaken -> Younger Customer tend to buy travel package.
2. DurationOfPitch vs ProdTaken -> Duration of pitch mean is greather for customer who buy travel package.
3. NumberOfTrips vs ProdTaken -> Does not seems to make the difference between customer who buy or not the travel package.
4. MonthlyIncome vs ProdTaken -> Customer with lower income tend to buy travel package (Correlated with age)

Numerical variables Vs ProdTaken

In [64]:
```
# Creating list for diferent kind of categorical variables wich we gonna trea

cat2_col = []  # list of ordinal variables
catBi_col = []  # list of binary variables

for col in df.columns:
    not_in_list = col not in cat_col
    if not_in_list == True:
        if df[col].nunique() < 10 and df[col].nunique() > 2:
            cat2_col.append(col)
        elif df[col].nunique() == 2 and col != "ProdTaken":
            catBi_col.append(col)
```

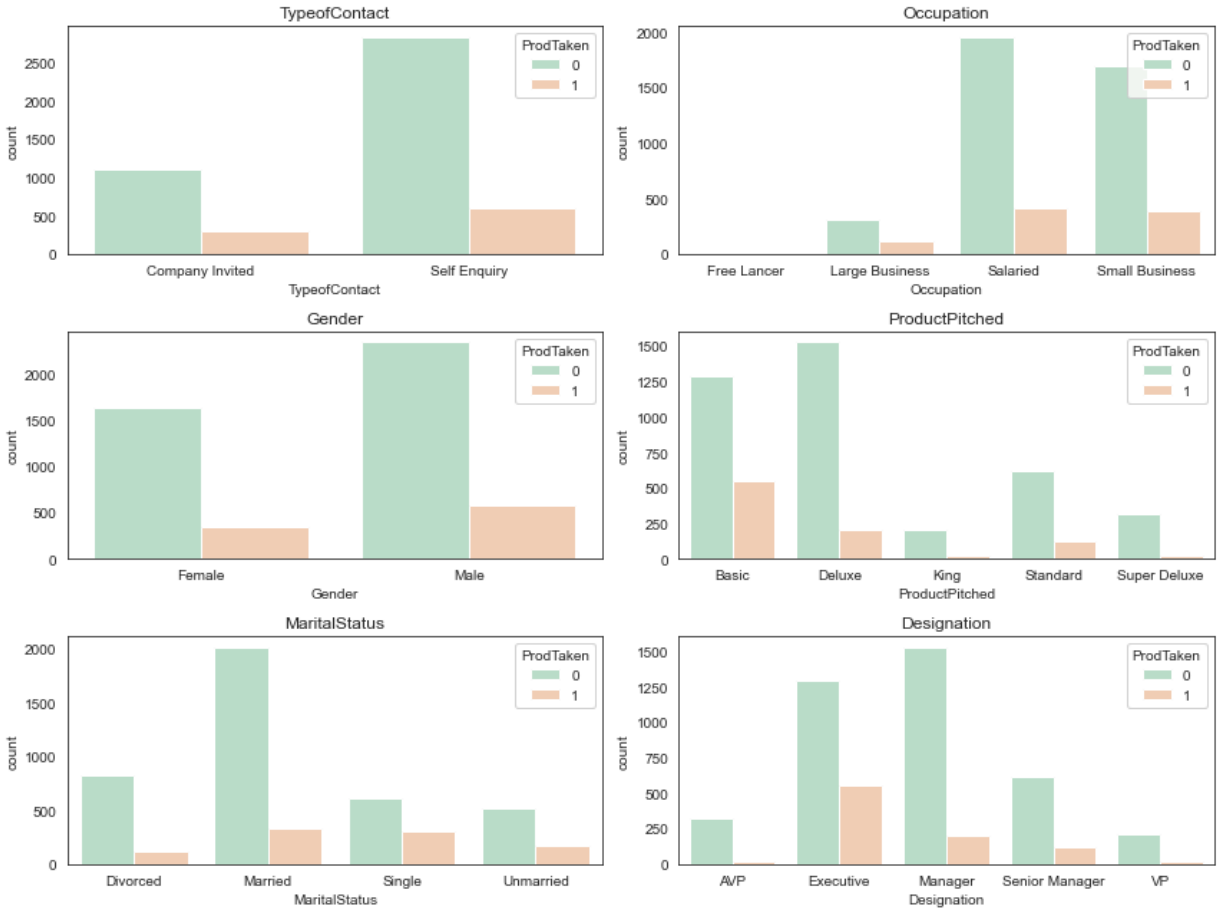
In [65]:
```
# Boxplot for categotical variables


def cat_plot(x):
    """
    plot
    feature: categorical feature
    plot subplot for variables on list x
    """
    y = 2
    plt.figure(figsize=(12, (len(x) / y) * 3))

    for i, variable in enumerate(x):
```

```
        plt.subplot(len(x) / y, y, i + 1)
        sns.countplot(x=df[variable], hue=df["ProdTaken"], palette="Pastel2")
        plt.tight_layout()
        plt.title(variable)
    plt.show()
```

Categorical variables Vs ProdTaken

> Dummy variables

In [66]:
```
cat_plot(cat_col)
```



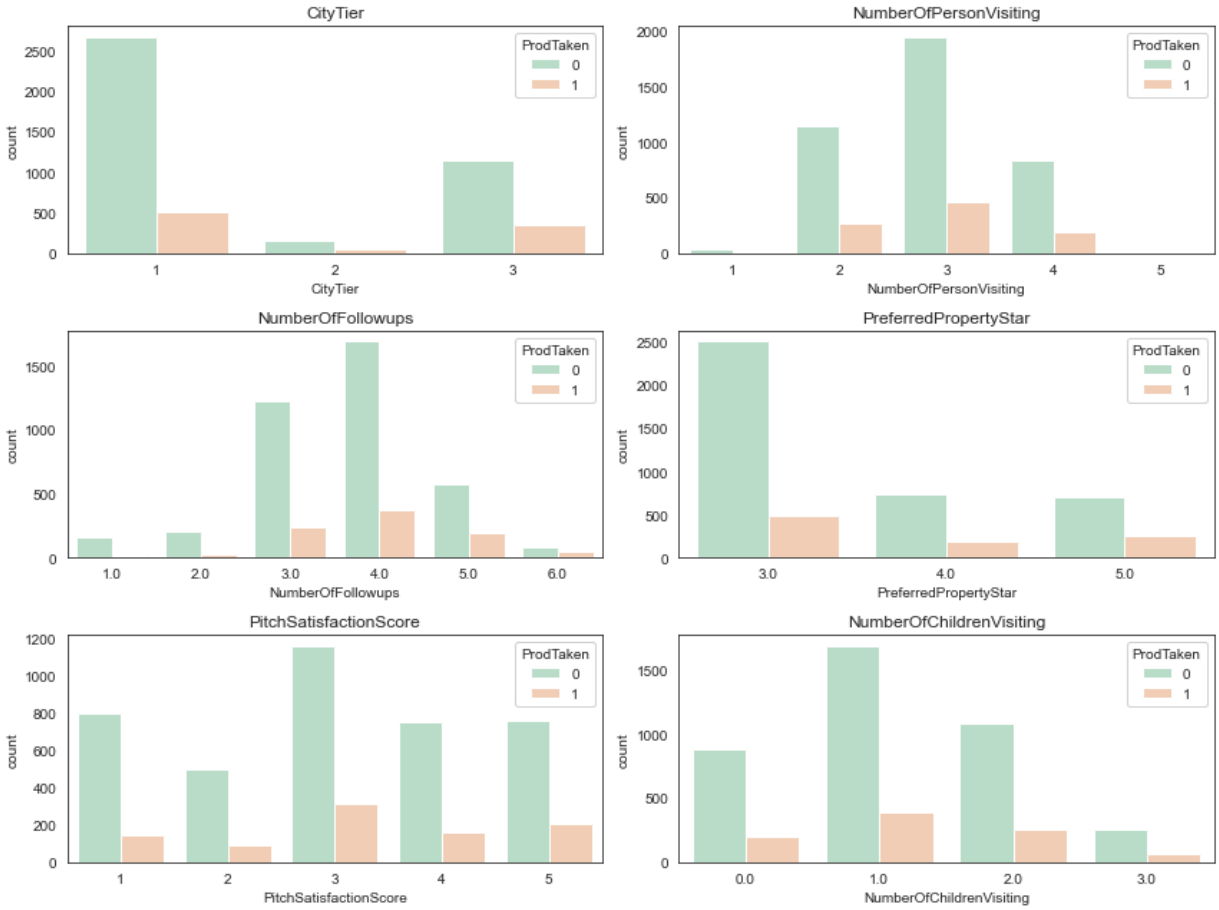
Observations:

1. TypeOfContact vs ProdTaken -> Customer who Self Enquiry is more likehood to buy the travel package.
2. Occupation vs ProdTaken -> Small Bussines and Salaried is more likehood to buy the travel package.
3. Gender vs ProdTaken -> We have more Male customer looking for travel package and buying it.
4. ProductPitched vs ProdTaken -> Basic is the most comum package buy followed by Delux.
5. MaritalStatus vs ProdTaken -> Silgle and Married are the customer who usually buy the travel package.
6. Designation vs ProdTaken -> Executives are more likehood to buy a travel package followed by Managers.

Categorical Ordinal variables Vs ProdTaken

> Label Encoding

In [67]:
```
cat_plot(cat2_col)
```

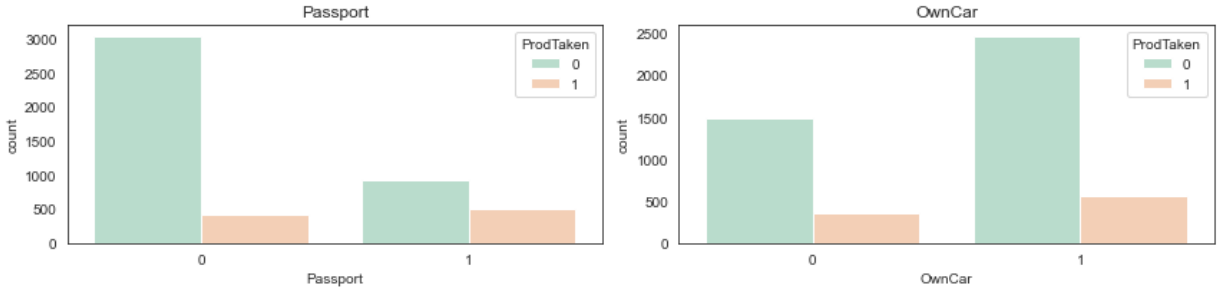


Observations:

1. CityTier vs ProdTaken -> Customer from developed city is more likehood to buy travel package
2. NumberOfPersonVisiting vs ProdTaken -> Usually 3 is the number of person visiting, followed by 2.
3. NumberOfFollowups vs ProdTaken -> Usually customer have 4 followups to decided if they want to or not.
4. PreferredPropertyStar vs ProdTaken -> Customer preffers 3 stars Property follow by 5 Stars.
5. PitchSatisfactionScore vs ProdTaken -> 3 Score is the most comum Stisfaction Score, from customer who bought or not our product.
6. NumberOfChildrenVisiting vs ProdTaken -> Usually customer will take 1 Children, followed by 2 for customer who bought or not our travel package.

Binary variables Vs ProdTaken

In [68]:
```
cat_plot(catBi_col)
```



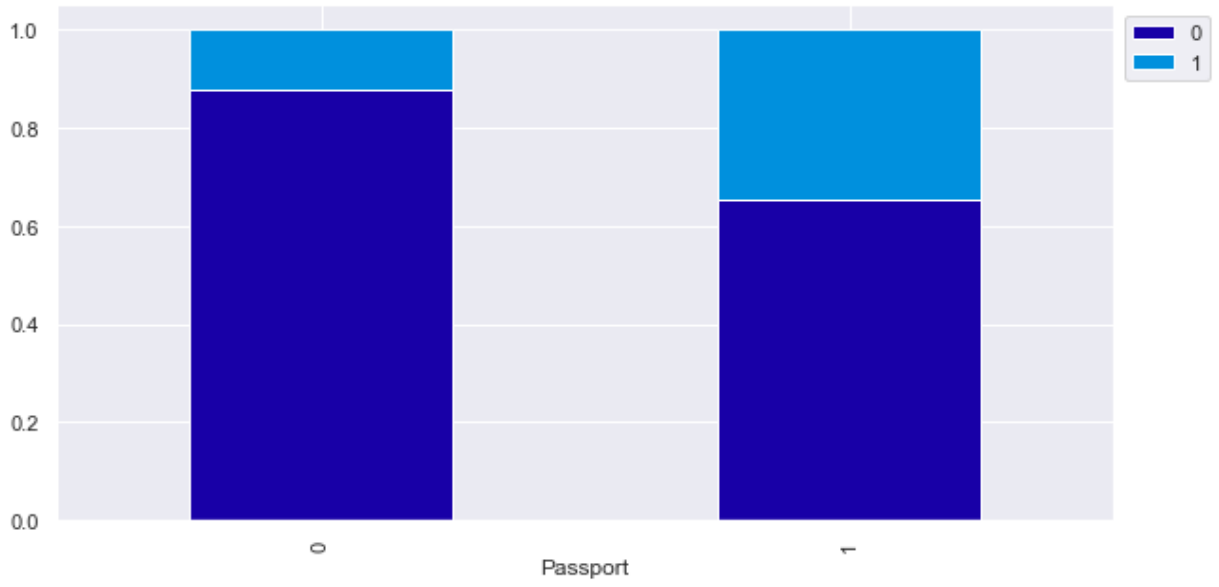
Observations:

1. Passport vs ProdTaken -> Customer with passport bought more travel package, the total of customer that have passport is less than without passport.
2. OwnCar vs ProdTaken -> 62% of customer has a car and from this 62%, only 18.5% bought travel package. On the other hand, 38% of customer does not have a car and from those, 19.4% bought a travel package. So OwnCar, does not seem to be a important feature.

In [69]:
```
def stacked_plot(x):
    sns.set(palette="nipy_spectral")
    tab1 = pd.crosstab(x, df["ProdTaken"], margins=True)
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(x, df["ProdTaken"], normalize="index")
    tab.plot(kind="bar", stacked=True, figsize=(10, 5))
    plt.legend(loc="lower left", frameon=False)
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

In [70]:
```
stacked_plot(df["Passport"])
```

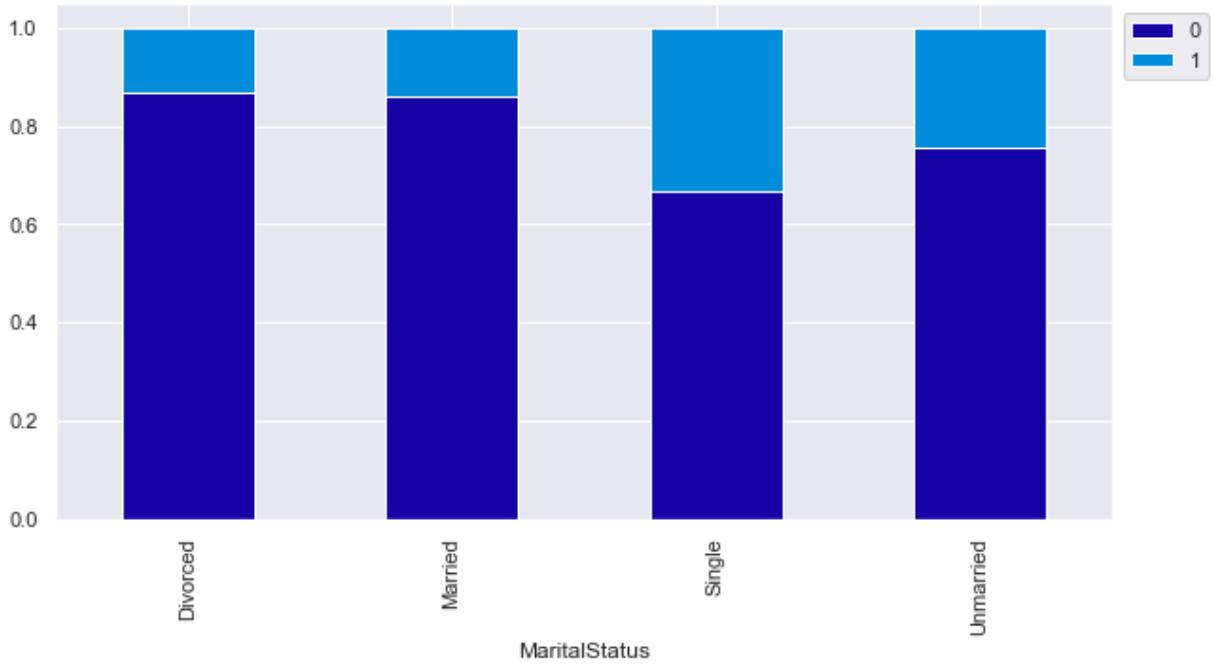
```
ProdTaken     0    1   All
Passport
0          3040  426  3466
1           928  494  1422
All        3968  920  4888
------------------------------------------------------------------------------------------
```



- Customer with passport are more likehood to buy a travel package.

In [71]:
```
stacked_plot(df["MaritalStatus"])
```

```
ProdTaken         0    1   All
MaritalStatus
Divorced        826  124   950
Married        2014  326  2340
Single          612  304   916
Unmarried       516  166   682
All            3968  920  4888
------------------------------------------------------------------------
----------------------------------------
```

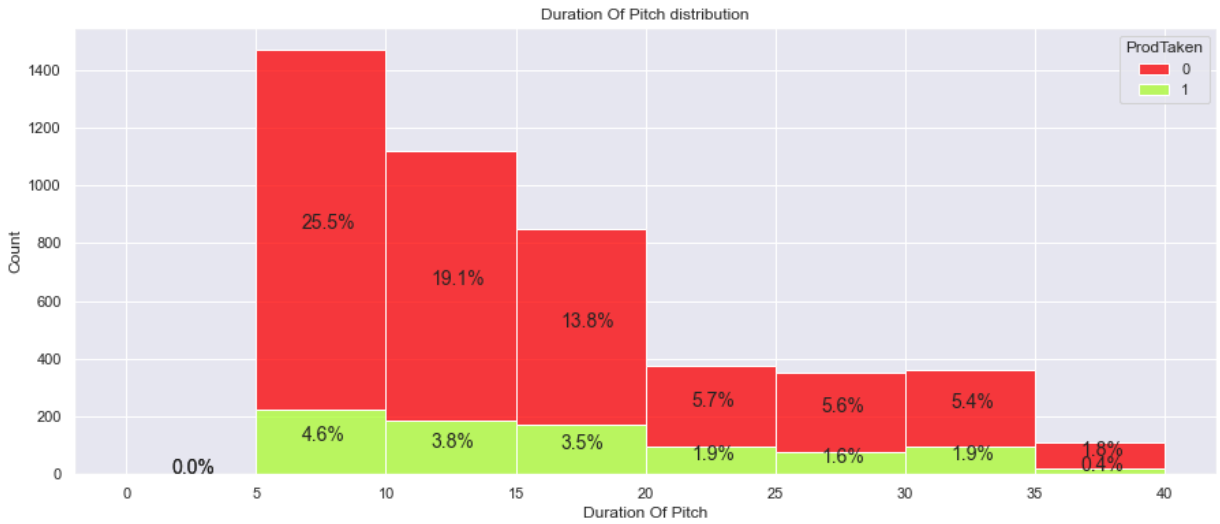


- Single and Unmarried customer are more likehood to buy the travel package.

Multivariate Analysis

In [72]:
```
# analyzing DurationOfPitch distribution

plt.figure(figsize=(15, 6))

# Plot informations
colors = ["red", "greenyellow"]
bins = np.linspace(0, 40, 9)  # creating bins on evenly spaced number on rang

# Plotting stacked histogram
plot_t = sns.histplot(
    data=df,
    x="DurationOfPitch",
    bins=bins,
    hue="ProdTaken",
    multiple="stack",
    palette=colors,
)
plt.xlabel("Duration Of Pitch")
plt.ylabel("Count")
plt.xticks(
    [0, 5, 10, 15, 20, 25, 30, 35, 40]
)  # specifying ticks via the xticks function.
```

```
plt.title("Duration Of Pitch distribution")

# Calculating the length of the column
total = len(df["DurationOfPitch"])

# Looping to calculate percentage of bins and annotate the percentage
for t in plot_t.patches:
    percentage = "{:.1f}%".format(100 * t.get_height() / total)
    x = t.get_x() + t.get_width() / 2 - 0.75
    y = t.get_y() + t.get_height() / 2
    plot_t.annotate(percentage, (x, y), size=14)
```

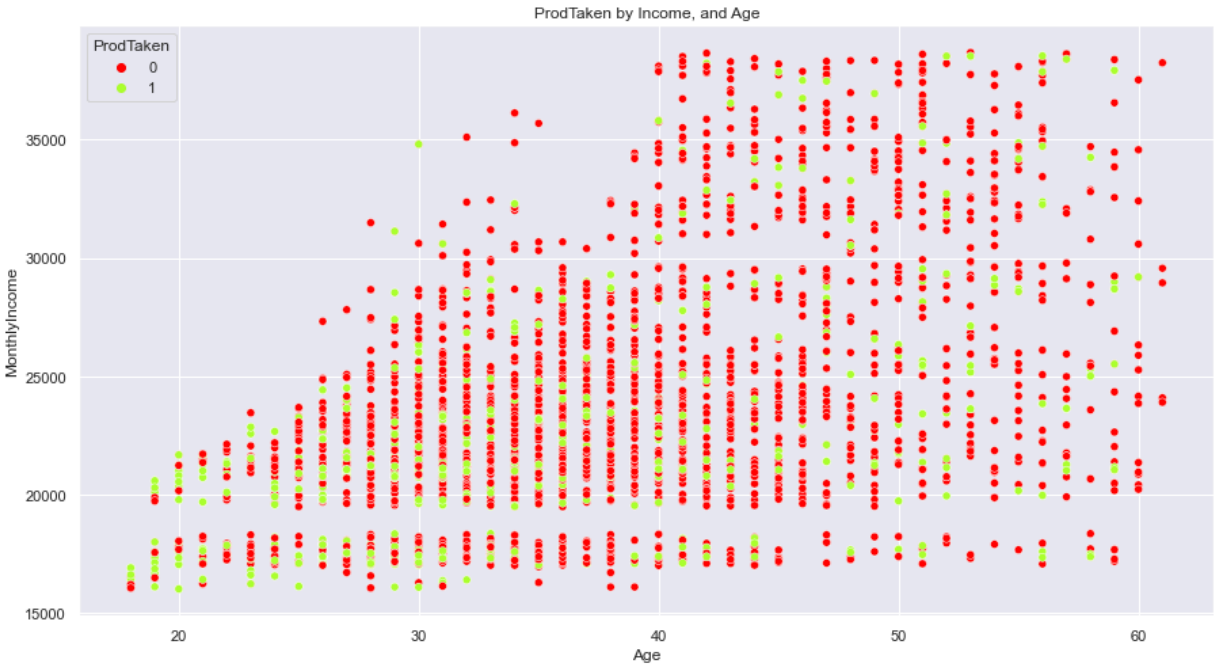


Observations:

30% of customer has a duration of pitch between 5 and 10, this is the time our team has to convince the customer about buying our package or make them listening us for more time.

In [73]:
```
# List of colors to use for the different ProdTaken
colors = ["red", "greenyellow"]

# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.scatterplot(
    df["Age"],
    df["MonthlyIncome"],
    hue=df["ProdTaken"],
    palette=colors,
)

plt.title("ProdTaken by Income, and Age")
```

Out[73]:
```
Text(0.5, 1.0, 'ProdTaken by Income, and Age')
```



- There are a concentration of custumer who bought travel packages between age os 18 to 30 and consequently low income

In [74]:
```
sns.catplot(x="Designation", y="MonthlyIncome", hue="Gender", kind="swarm", d
```

Out[74]:
```
<seaborn.axisgrid.FacetGrid at 0x1508c5fd370>
```

Observations:

- In general, Income is correlated with Designation
- Gender does not make much difference.

In [75]:
```
# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))
```

```
sns.scatterplot(df["Age"], df["MonthlyIncome"], hue=df["Designation"])

plt.title("Age by Income, and Designation")
```

Out[75]:
```
Text(0.5, 1.0, 'Age by Income, and Designation')
```

Observation:

We can see a pattern here.

- Customer with age greather than 40 and Monthly Income greather than 33000 usually are VP or AVP designation
- Income less than 32000 and greatger than 28000 Senior Manager or Manager.
- Low Income is associated with Executive Designation

In [76]:
```
# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.scatterplot(df["Age"], df["MonthlyIncome"], hue=df["Occupation"])

plt.title("Age by Income, and Occupation")
```

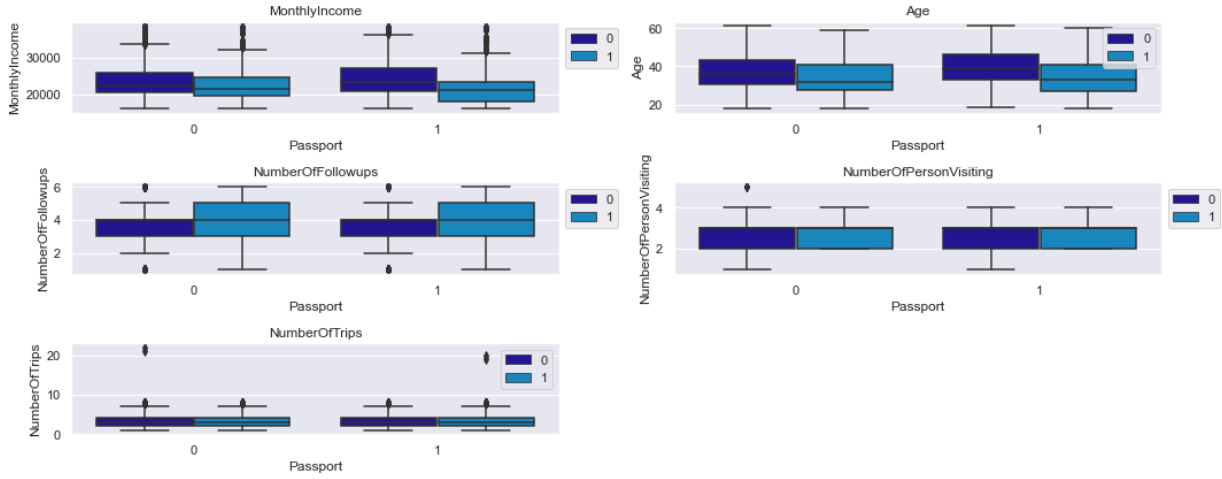
Out[76]:
```
Text(0.5, 1.0, 'Age by Income, and Occupation')
```

Observation:

- Seems not to have a patter between Age, Monthly Income and Occupation, on the other hand, we can see that Salaried and Small Business are more comum between the customers.

Passport

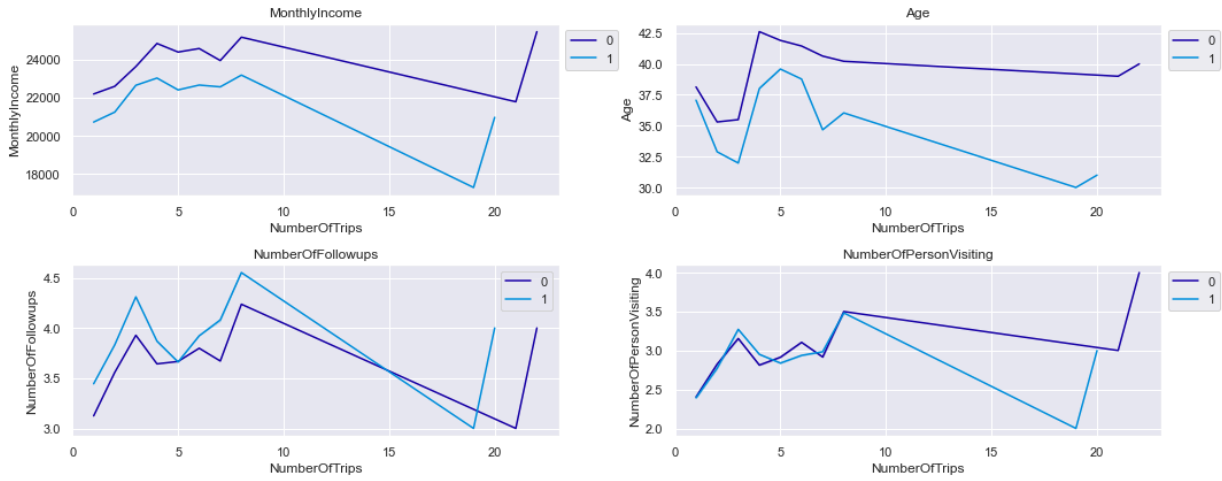
In [77]:
```
cols = df[
    [
        "MonthlyIncome",
        "Age",
        "NumberOfFollowups",
        "NumberOfPersonVisiting",
        "NumberOfTrips",
    ]
].columns.tolist()
plt.figure(figsize=(15, 6))
for i, variable in enumerate(cols):
    plt.subplot(3, 2, i + 1)
    sns.boxplot(df["Passport"], df[variable], hue=df["ProdTaken"])
    plt.tight_layout()
    plt.title(variable)
    plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



Customer with or without passaport seems to have almost the same behavier considering Monthly Income, Age, Number of Folowwups, Number Of Person Visiting and Number of Trips. Further we'll check the behavier only for customer who bought the travel package.

NumeberOfTrip

In [78]:
```
cols = df[
    ["MonthlyIncome", "Age", "NumberOfFollowups", "NumberOfPersonVisiting"]
].columns.tolist()
plt.figure(figsize=(15, 6))
for i, variable in enumerate(cols):
    plt.subplot(2, 2, i + 1)
    sns.lineplot(df["NumberOfTrips"], df[variable], hue=df["ProdTaken"], ci=0
    plt.tight_layout()
    plt.title(variable)
    plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



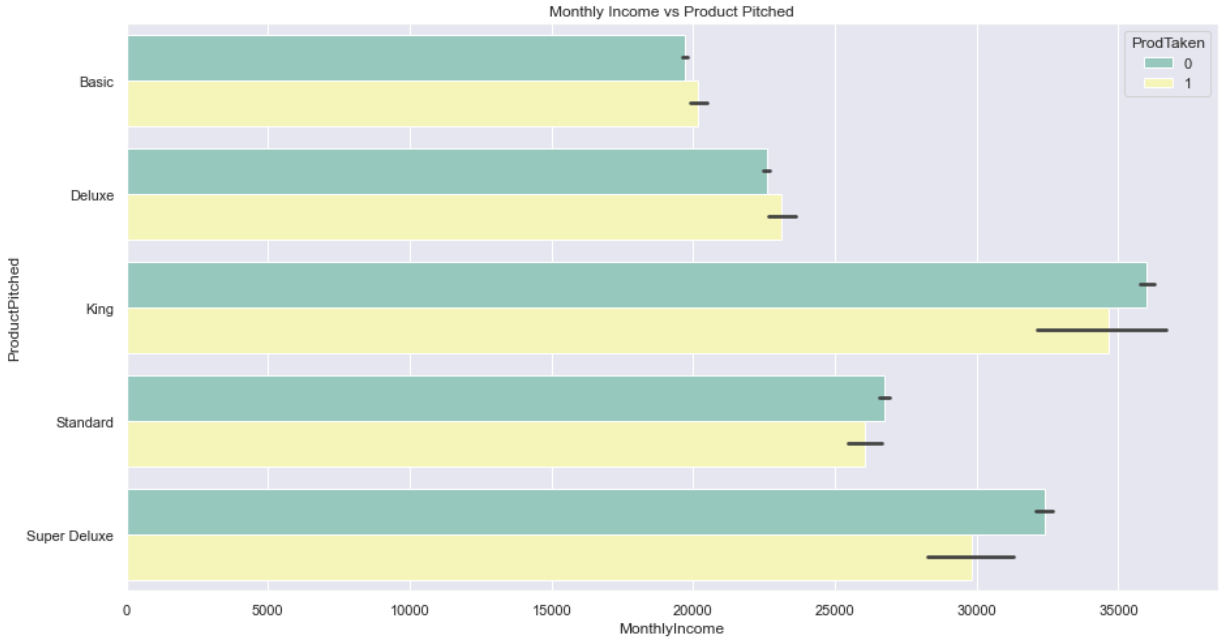
- Customer who bought the travel package has low income, are yonger, does more followups and Number of person Visiting does not change much compared with the one whoi did not buy it.

Monthly Income vs Product Pitched

In [79]:
```
# Ploting countplot for analysis between Monthly Income, Product Pitched and
plt.figure(figsize=(15, 8))

sns.barplot(
```

```
    x=df["MonthlyIncome"], y=df["ProductPitched"], hue=df["ProdTaken"], palet
)

plt.title("Monthly Income vs Product Pitched")
```

Out[79]:
```
Text(0.5, 1.0, 'Monthly Income vs Product Pitched')
```



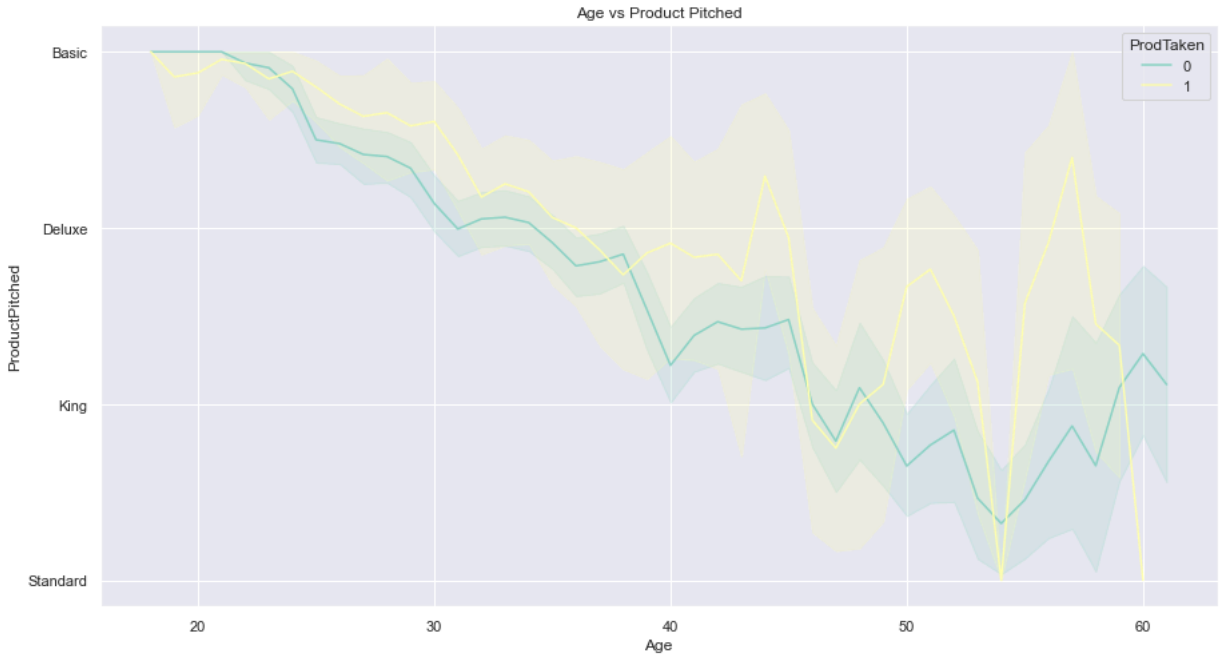
We can see a patter here where customers with:

- Income less than 21000 choose Basic Package;
- Income less than 24000 choose Delux Package;
- Income less than 27000 choose Standard Package;
- Income less than 30000 choose Super Delux Package;
- Income less than 35000 choose King Package;

Age vs Product Pitched

In [80]:
```
# Ploting countplot for analysis between Monthly Income, Product Pitched and
plt.figure(figsize=(15, 8))

sns.lineplot(x=df["Age"], y=df["ProductPitched"], hue=df["ProdTaken"], palett

plt.title("Age vs Product Pitched")
```

Out[80]:
```
Text(0.5, 1.0, 'Age vs Product Pitched')
```

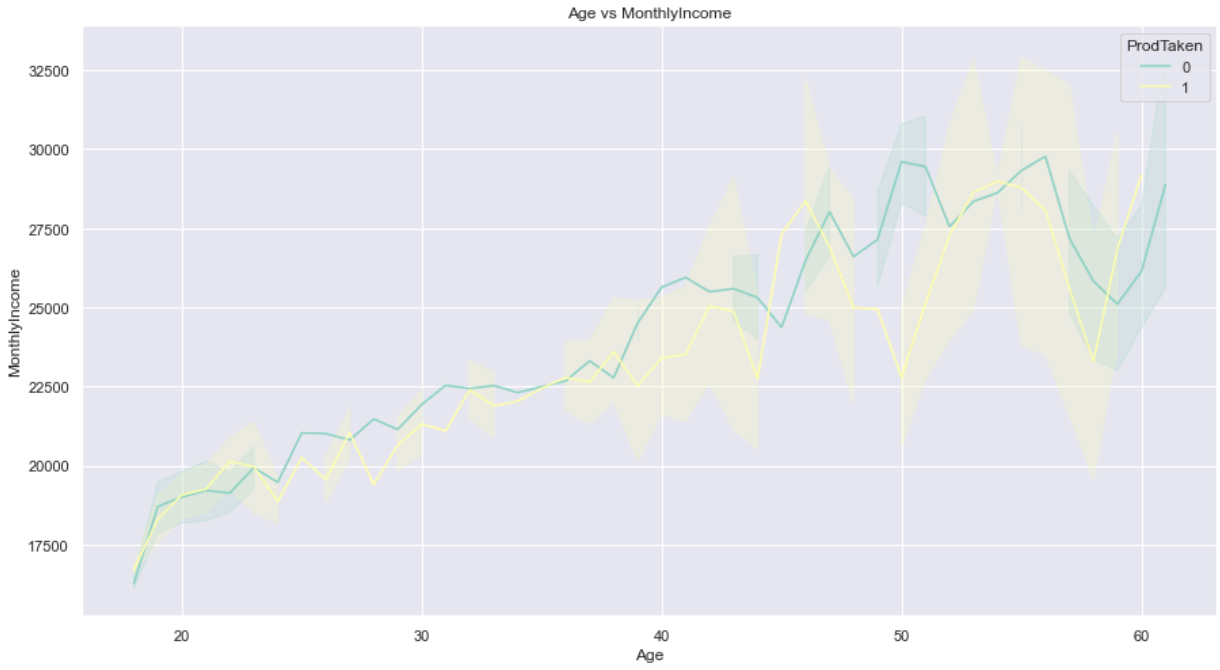


Here we can see clearly the correlation between Age and Product Pitched. Younger the customer basic is the package.

Age vs MonthlyIncome

In [81]:
```
# Ploting countplot for analysis between Monthly Income, Product Pitched and
plt.figure(figsize=(15, 8))

sns.lineplot(x=df["Age"], y=df["MonthlyIncome"], hue=df["ProdTaken"], palette

plt.title("Age vs MonthlyIncome")
```

Out[81]:
```
Text(0.5, 1.0, 'Age vs MonthlyIncome')
```



Here we can see clearly the correlation between Age and Monthly Income. How it was expected, older the customer, higher usually is the Income.

In [82]:
```
# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))
```

```
sns.scatterplot(df["Age"], df["MonthlyIncome"], hue=df["ProductPitched"])

plt.title("Age by Income, and ProdPitched")
```

Out[82]: Text(0.5, 1.0, 'Age by Income, and ProdPitched')

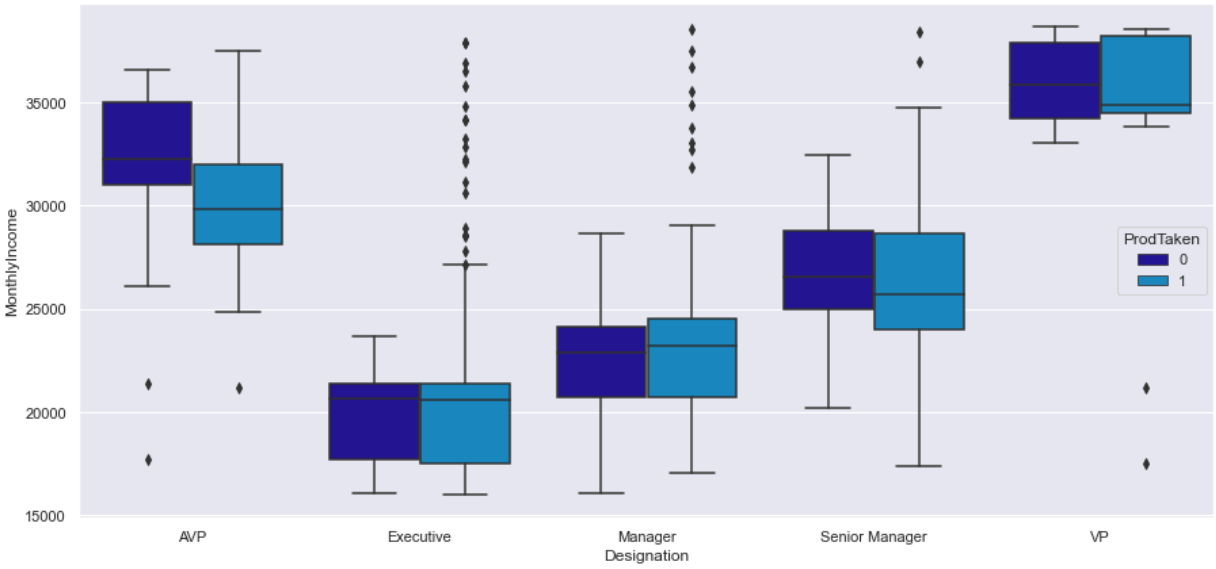
Putting all togheter, We can see kind of a pattern between Age, Income and ProdPitched.

- Younger customers with Income Lower than 21000, with Basic package.
- Customer older than 40, with high income (>32000) with King package.

Designation per Monthly Income by ProdTaken

In [83]:
```
plt.figure(figsize=(15, 7))
sns.boxplot(x="Designation", y="MonthlyIncome", hue="ProdTaken", data=df)
```

Out[83]: <AxesSubplot:xlabel='Designation', ylabel='MonthlyIncome'>

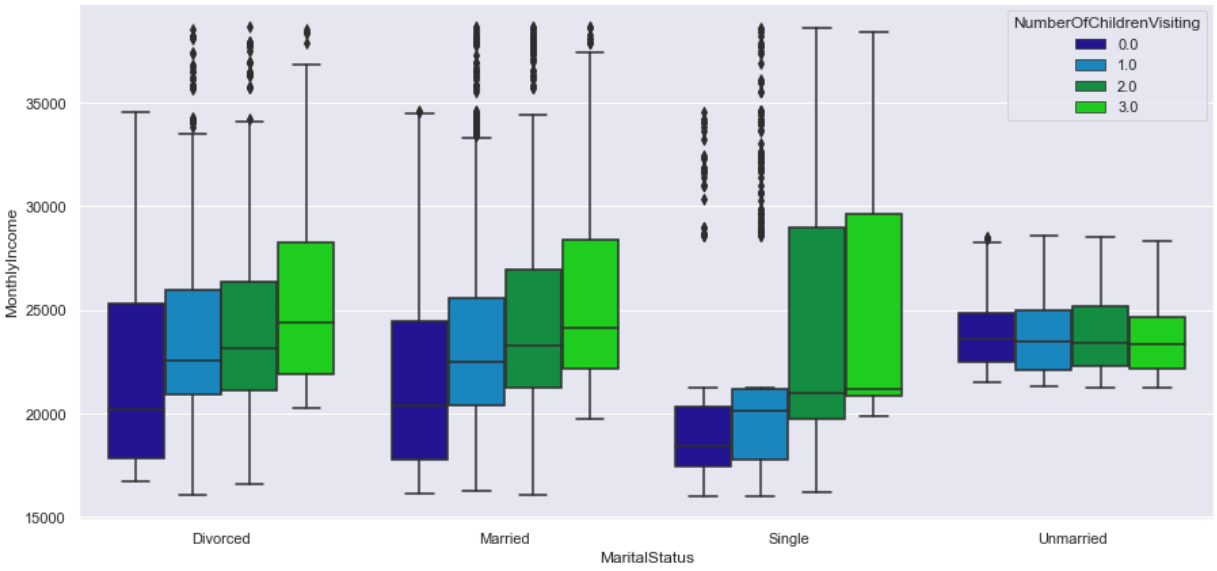


- Monthly Income is correlated with Designation.
- There are no much difference between customer who bought or not the package.

NumberOfChildrenVisiting per Monthly Income and MaritalStatus

In [84]:
```
plt.figure(figsize=(15, 7))
sns.boxplot(
    x="MaritalStatus", y="MonthlyIncome", hue="NumberOfChildrenVisiting", dat
)
```

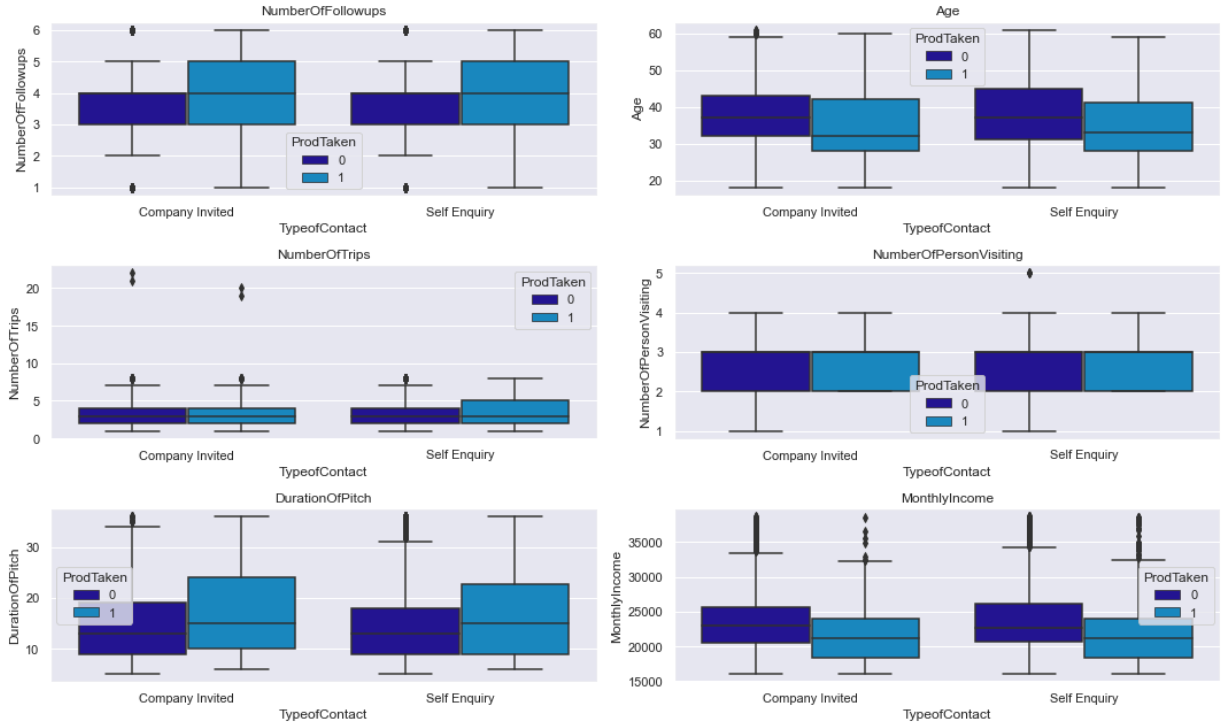
Out[84]:
```
<AxesSubplot:xlabel='MaritalStatus', ylabel='MonthlyIncome'>
```



- Usually Higher Income is associated with more Children going togheter.

TypeofContact

In [85]:
```
cols = df[
    [
        "NumberOfFollowups",
        "Age",
        "NumberOfTrips",
        "NumberOfPersonVisiting",
        "DurationOfPitch",
        "MonthlyIncome",
    ]
].columns.tolist()
plt.figure(figsize=(15, 9))
for i, variable in enumerate(cols):
    plt.subplot(3, 2, i + 1)
    sns.boxplot(x="TypeofContact", y=variable, hue="ProdTaken", data=df)
    plt.tight_layout()
    plt.title(variable)
plt.show()
```



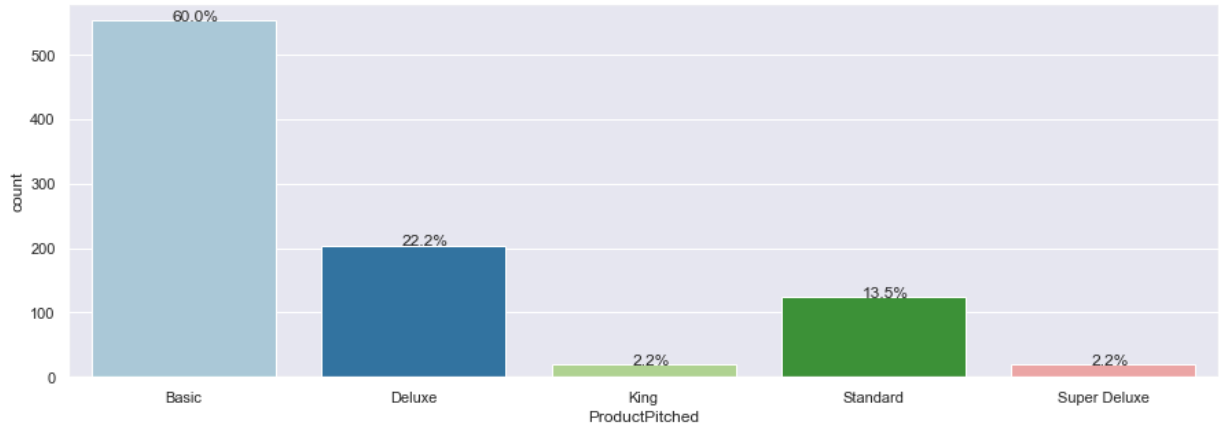
- Comparing customer who bought or not the package considering Company Invited VS Self Enquiry, does not have much difference on behavier.
- The difference between buy or not is more correlated with Number Of Followups, Yonger customers, Higher Duration Of Pitche and low Income.

Customer Profile

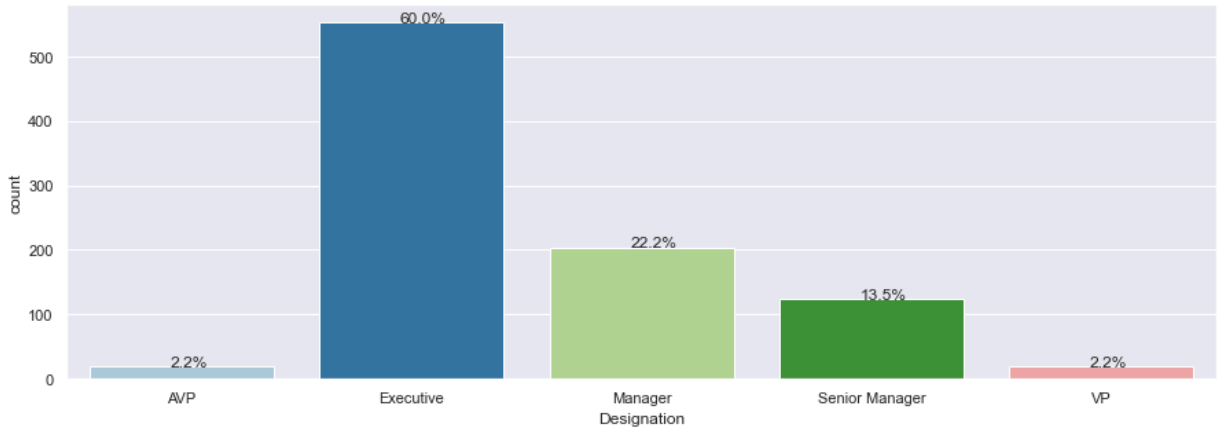
In [86]:
```
# Lets check only customer who bought the travel package
df_temp = df[df["ProdTaken"] == 1]
```

In [87]:
```
# Lets check only customer who DID NOT bought the travel package
df_tempNO = df[df["ProdTaken"] == 0]
```

In [88]:
```
perc_on_bar(df_temp.ProductPitched)
```

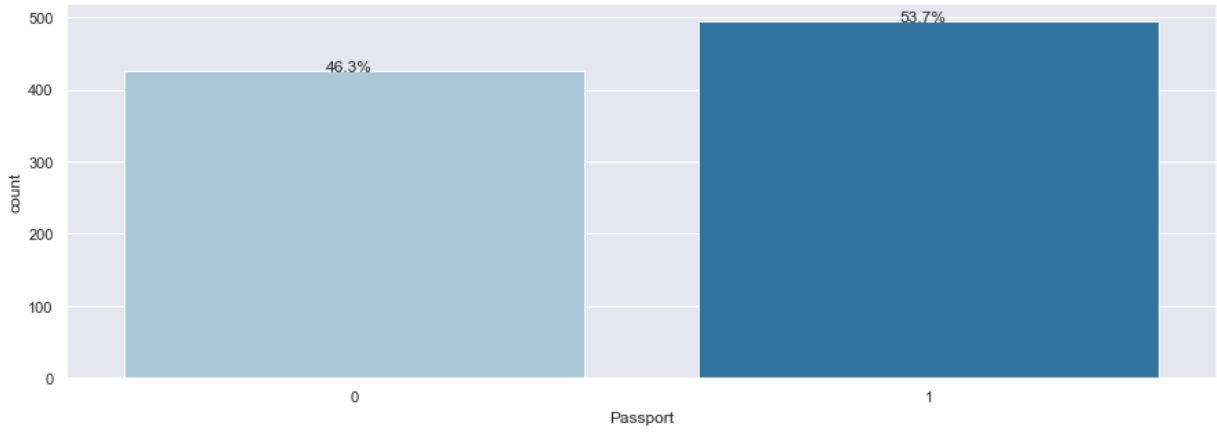


In [89]:
```
perc_on_bar(df_temp.Designation)
```



- Analyzing both graphcs, we can see a relation between Designation and ProdPitched.
- 60% of buyers work as Executives and chose Basic Package.
- 22.2% choose Deluxe and works as Managers.
- VP and AVP are customers who less bouth the package, considering they prefference on Super Deluxe and King, we should analyze this packages and why they are not convencing our customer to buy it.

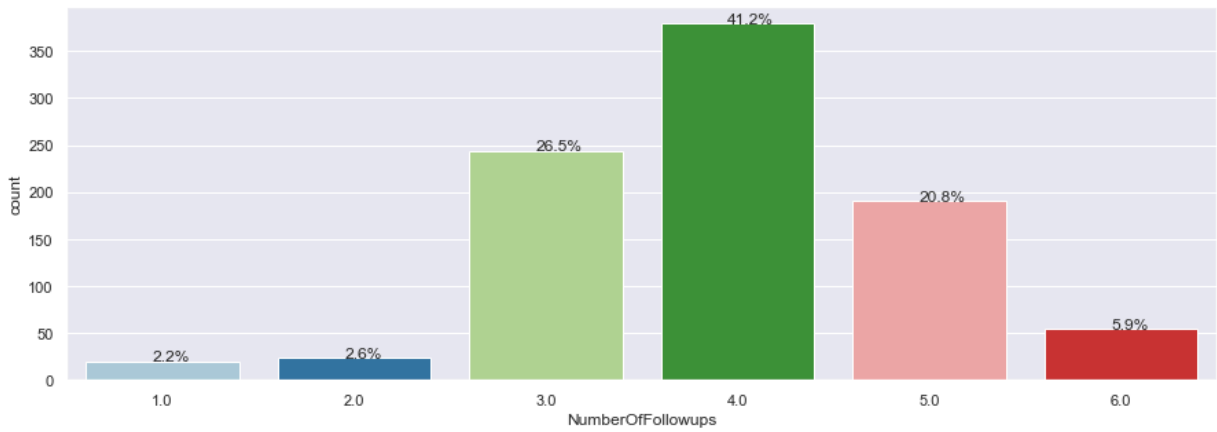
In [90]:
```
perc_on_bar(df_temp.Passport)
```



- 53.7% of buyers do have a passport
- 46.3% do not have it.

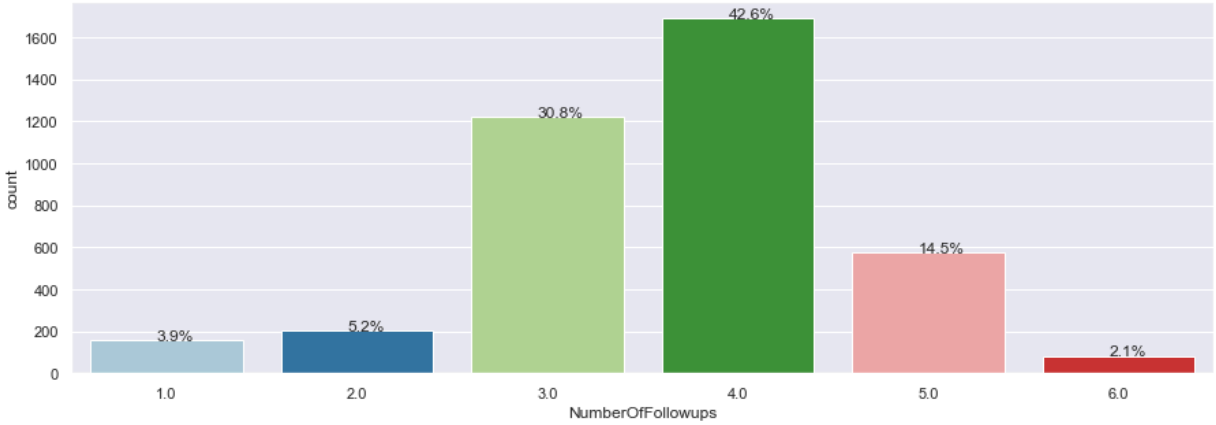
Number Of Followups Customers who DID buy the package

In [91]:
```
perc_on_bar(df_temp.NumberOfFollowups)
```



Number Of Followups Customers who DID NOT buy the package

In [92]:
```
perc_on_bar(df_tempNO.NumberOfFollowups)
```



- we can see that ~ 30% of our customers decide to buy or not on the 3rd follow up and ~ 41% decided it on the 4th.
- Our company need to think the best estrategies to convince our customer on the 3rd followup, using the best of the time to show the benefits of our packages

In [93]:
```
df_temp.groupby(["Designation"])["MonthlyIncome"].mean()
```

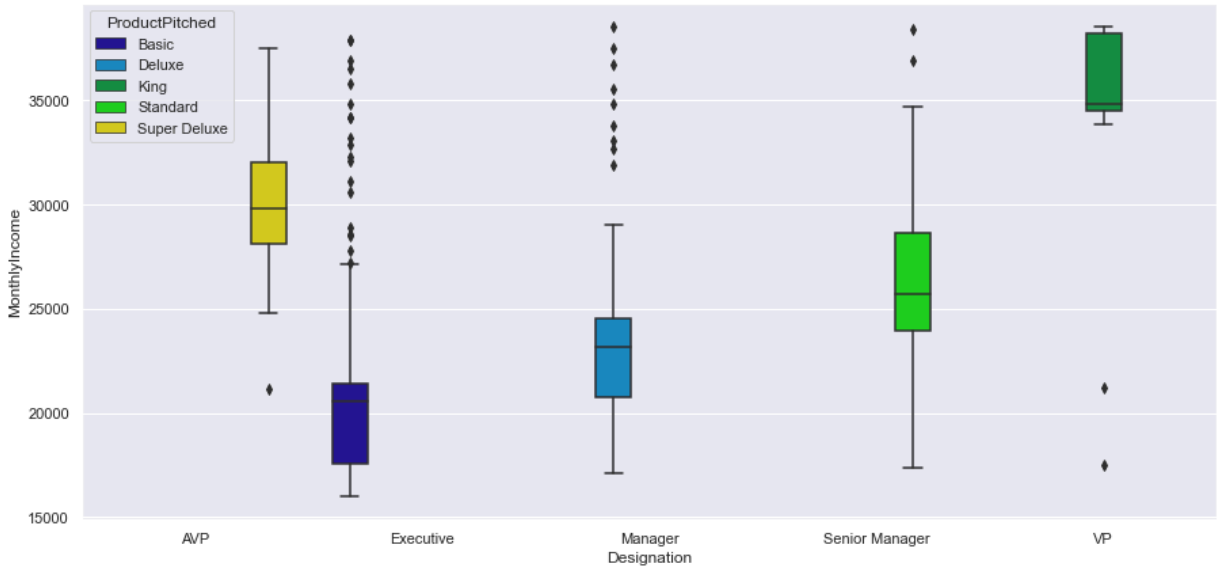
Out[93]:
```
Designation
AVP               29823.800000
Executive         20161.529301
Manager           23106.215385
Senior Manager    26035.419355
VP                34672.100000
Name: MonthlyIncome, dtype: float64
```

In [94]:
```
df_temp.groupby(["Designation"])["Age"].mean()
```

Out[94]:
```
Designation
AVP               43.500000
Executive         31.289320
Manager           37.641414
Senior Manager    41.008130
VP                48.900000
Name: Age, dtype: float64
```

In [95]:
```
plt.figure(figsize=(15, 7))
sns.boxplot(x="Designation", y="MonthlyIncome", hue="ProductPitched", data=df
```

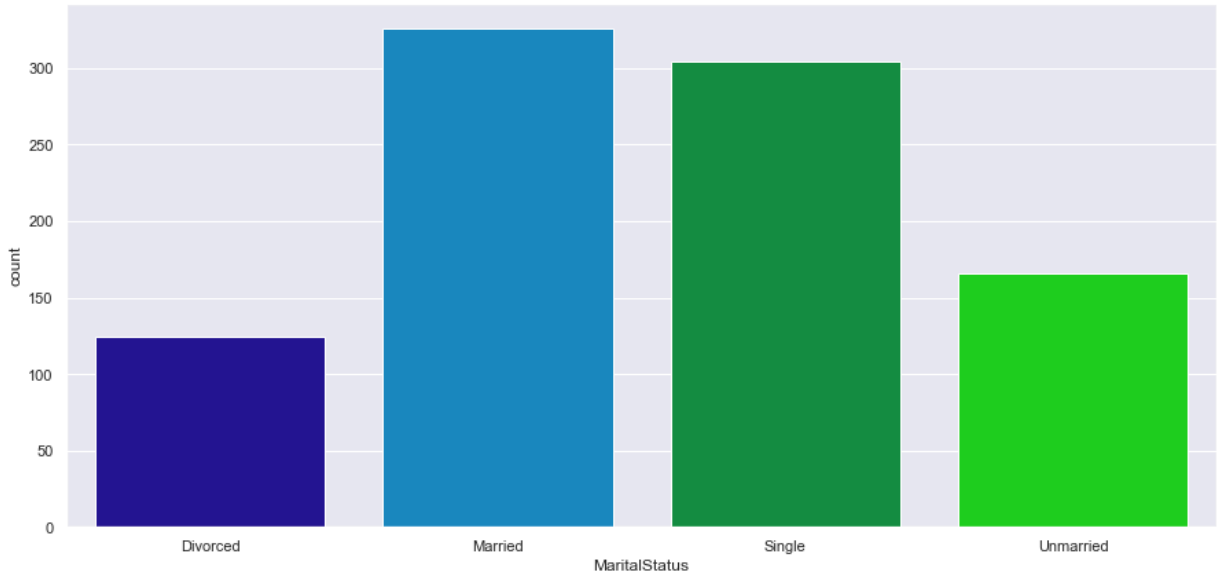
Out[95]:
```
<AxesSubplot:xlabel='Designation', ylabel='MonthlyIncome'>
```



- From buyers, we can see the correlation between Income, Designation and ProdPitched.

In [96]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x="MaritalStatus", data=df_temp)
```

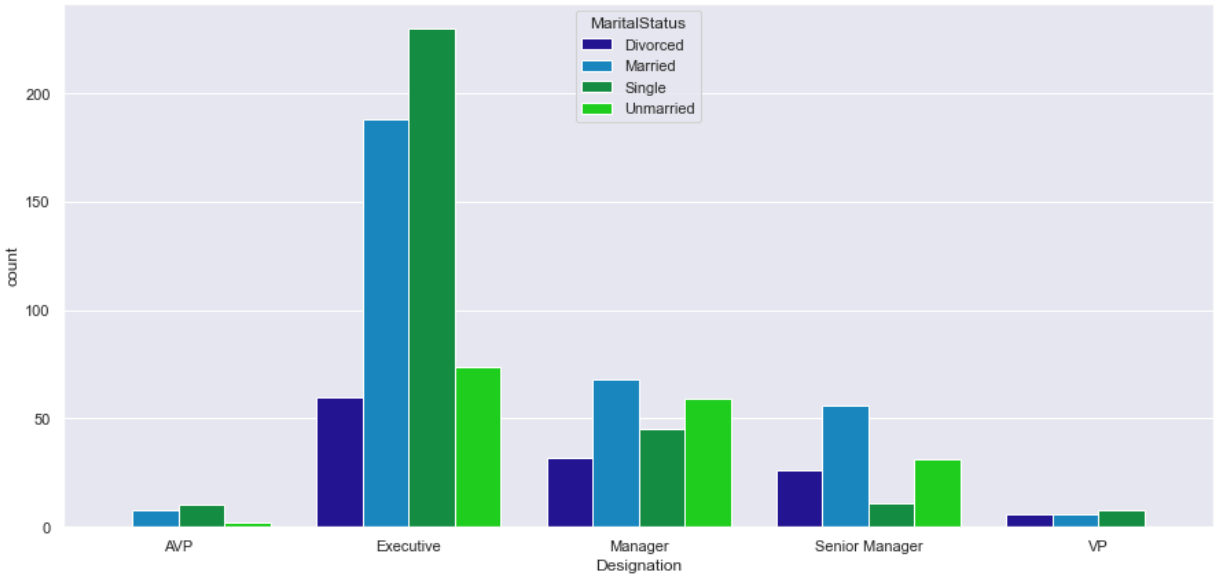
Out[96]:
```
<AxesSubplot:xlabel='MaritalStatus', ylabel='count'>
```



- Married and Single ar our target customers.

In [97]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x="Designation", hue="MaritalStatus", data=df_temp)
```

Out[97]:
```
<AxesSubplot:xlabel='Designation', ylabel='count'>
```



- Using the distribution between Marital Status VS Designation and knowing the packages they prefer, company can create packages specifics for the necessity of each profile of customer

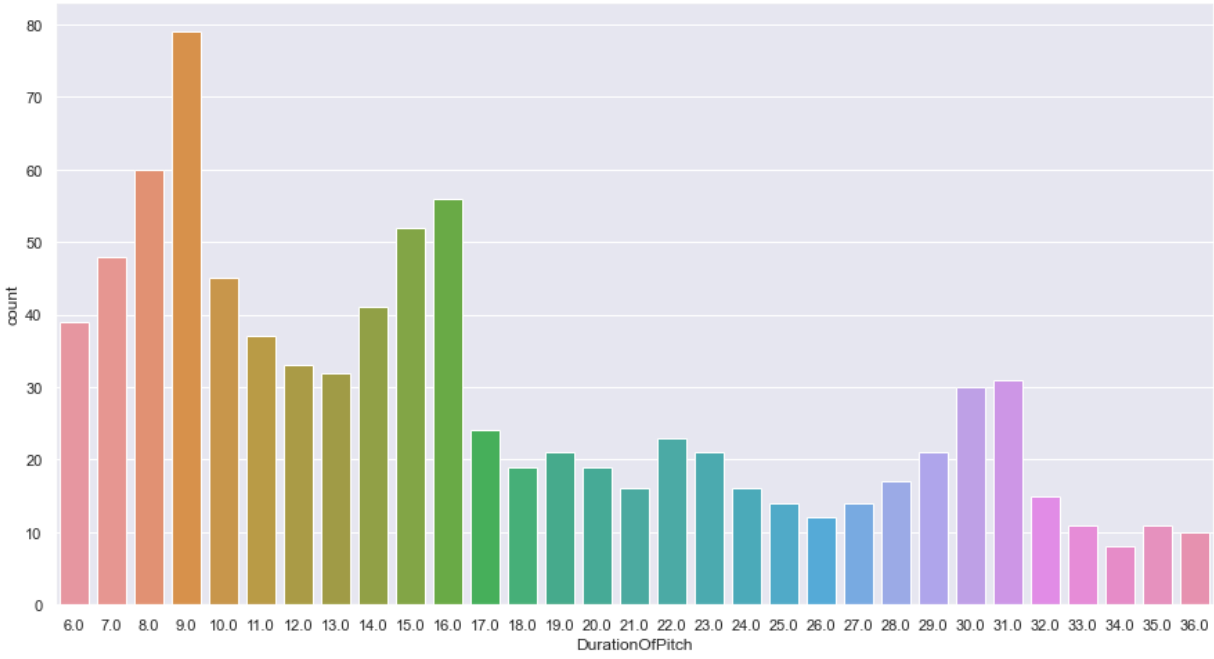
In [98]:
```
df_temp.groupby(["Designation"])["NumberOfChildrenVisiting"].value_counts()
```

Out[98]:
```
Designation     NumberOfChildrenVisiting
AVP             1.0                        9
                2.0                        6
                0.0                        4
                3.0                        1
Executive       1.0                      234
                2.0                      158
                0.0                      118
                3.0                       41
Manager         1.0                       90
                2.0                       56
                0.0                       45
                3.0                       12
Senior Manager  1.0                       52
                0.0                       33
                2.0                       28
                3.0                       10
VP              1.0                        7
                2.0                        5
                0.0                        2
                3.0                        2
Name: NumberOfChildrenVisiting, dtype: int64
```

Duration Of Pitch Customers who DID buy the package

In [99]:
```
# Ploting countplot for analysis between Monthly Income, Product Pitched and
plt.figure(figsize=(15, 8))

sns.countplot(
    x=df_temp["DurationOfPitch"],
)
```

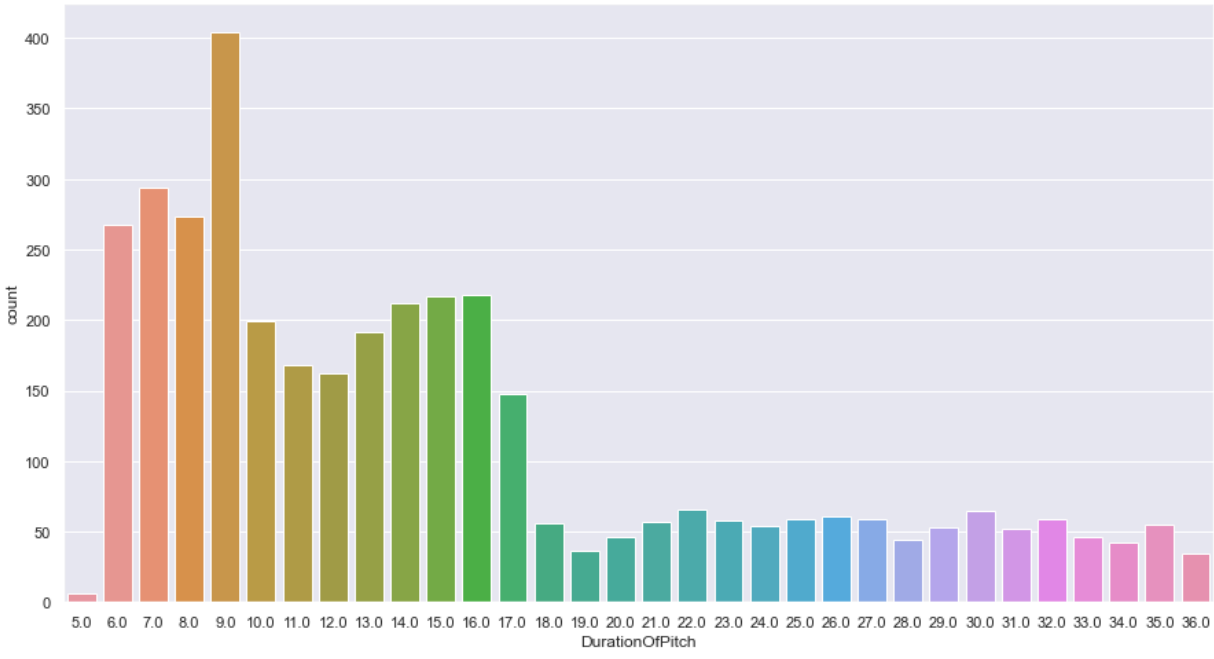
Out[99]:
```
<AxesSubplot:xlabel='DurationOfPitch', ylabel='count'>
```



Duration Of Pitch Customers who DID NOT buy the package

In [100…
```
# Ploting countplot for analysis between Monthly Income, Product Pitched and
plt.figure(figsize=(15, 8))

sns.countplot(
    x=df_tempNO["DurationOfPitch"],
)
```

Out[100… <AxesSubplot:xlabel='DurationOfPitch', ylabel='count'>



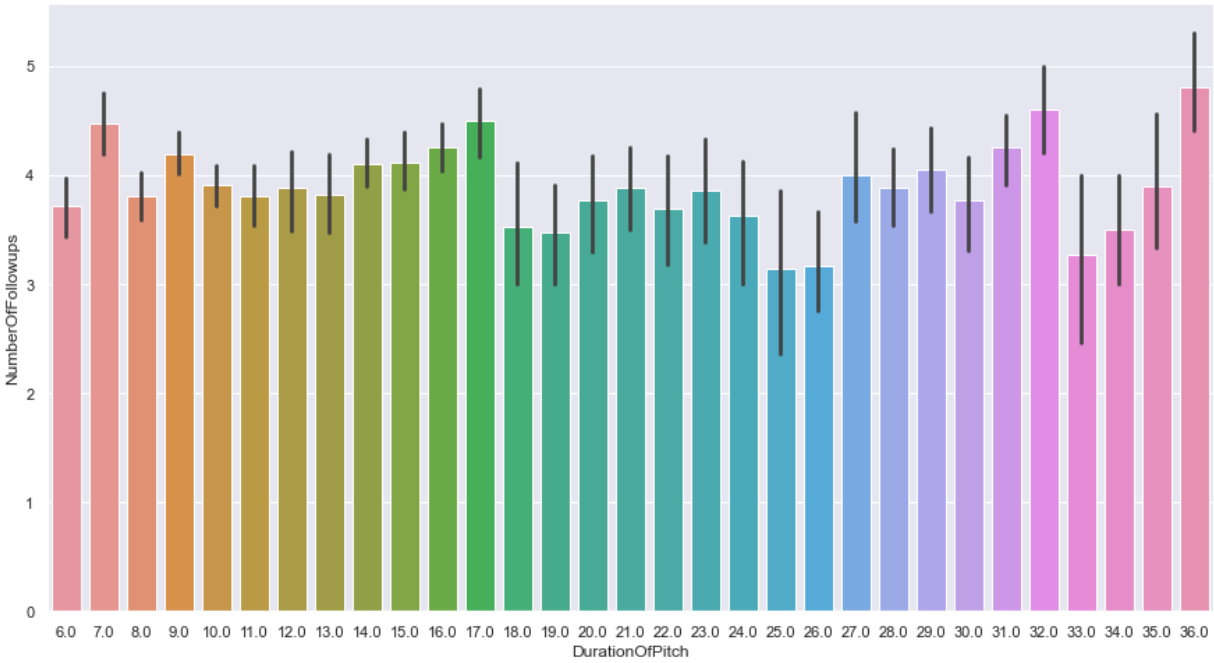
- Both case duration of pitch is ~9, so this is the time our team have to explaing the package and associate the best package to the right customer profile, raising the chances of them ti buy it

In [101…
```
# Ploting countplot for analysis between Monthly Income, Product Pitched and
plt.figure(figsize=(15, 8))
```

```
sns.barplot(
    x=df_temp["DurationOfPitch"],
    y=df_temp["NumberOfFollowups"],
)
```

Out[101…

```
<AxesSubplot:xlabel='DurationOfPitch', ylabel='NumberOfFollowups'>
```

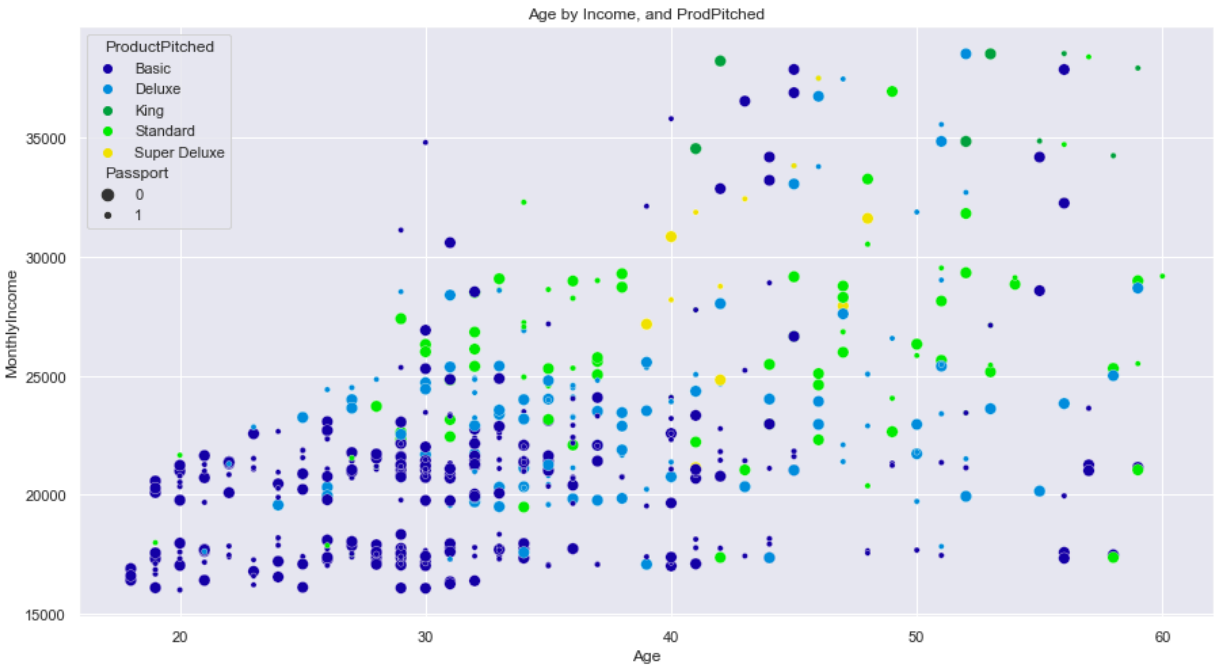


In [102…

```
# Ploting scatterplot for analysis about correlation
plt.figure(figsize=(15, 8))

sns.scatterplot(
    df_temp["Age"],
    df_temp["MonthlyIncome"],
    hue=df_temp["ProductPitched"],
    size=df_temp["Passport"],
)

plt.title("Age by Income, and ProdPitched")
```

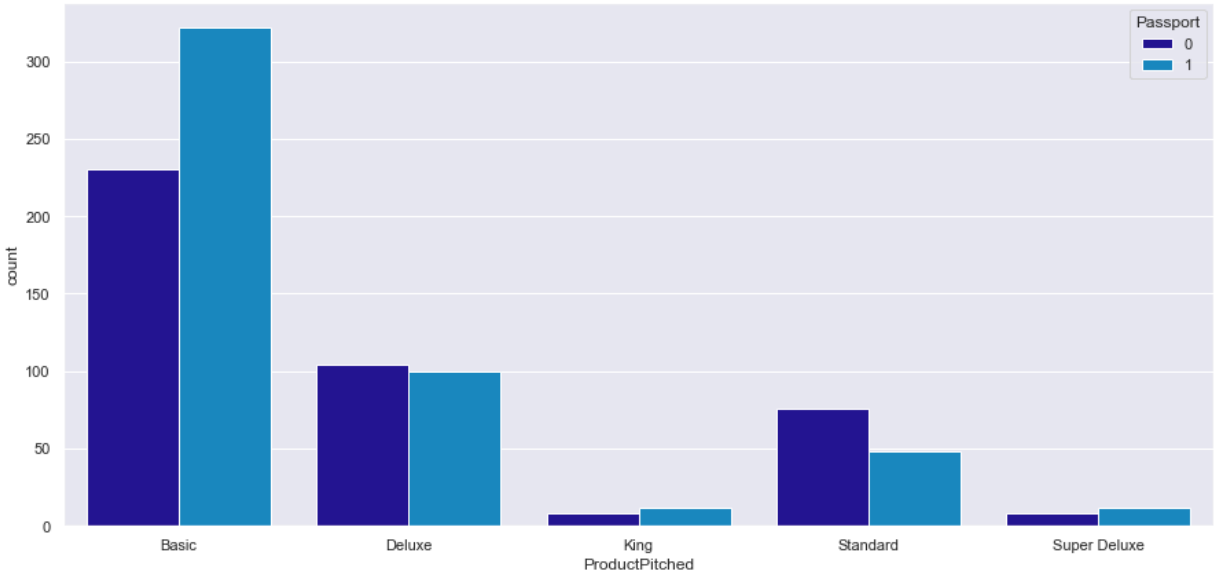
Out[102…

```
Text(0.5, 1.0, 'Age by Income, and ProdPitched')
```



In [103…
```
plt.figure(figsize=(15, 7))
sns.countplot(x="ProductPitched", hue="Passport", data=df_temp)
```

Out[103…
```
<AxesSubplot:xlabel='ProductPitched', ylabel='count'>
```



- Customer with passport on basic package is more likehood to buy it.
- Deluxe, Super Delux and King does not show a high variance between them.
- Standard customers usually does not gave passport

Missing Values

In [104…
```
# Before we start looking at the individual distributions and interactions, l
df.isnull().sum().sort_values(ascending=False)
```

Out[104…
```
DurationOfPitch             251
MonthlyIncome               233
Age                         226
NumberOfTrips               140
NumberOfChildrenVisiting     66
NumberOfFollowups            45
PreferredPropertyStar        26
TypeofContact                25
Gender                        0
CityTier                      0
Occupation                    0
ProductPitched                0
NumberOfPersonVisiting        0
Designation                   0
MaritalStatus                 0
Passport                      0
PitchSatisfactionScore        0
OwnCar                        0
ProdTaken                     0
dtype: int64
```

- DurationOfPitch displacement information of 251 observations is missing and MonthlyIncome of 233 entries is missing.
- Information about Age is not avaliable for 226 customer.

- NumberOfTrips has a missing count of 140.
- NumberOfChildrenVisiting is missing 66, NumberOfFollowups 45.
- PreferredPropertyStar 26 missing values and TypeofContact 25 not avaliable informations

---

We'll have to see if there is a pattern on the missing values, by groupby some informations.

- Before starting with Missing Values treatment, we gonna copy the dataset, so we can used it on our XGModel and see how it preform with and without treating missing values.

In [105…
```
df_XG = df.copy()
```

DurationOfPitch

In [106…
```
# Looking at a few rows where # DurationOfPitch is missing
df[df["DurationOfPitch"].isnull()]
```

Out[106…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
46	0	34.0	Company Invited	3	NaN	Small Business	Male	
75	0	31.0	Self Enquiry	1	NaN	Salaried	Female	
76	0	35.0	Self Enquiry	3	NaN	Small Business	Male	
84	0	34.0	Self Enquiry	1	NaN	Small Business	Male	
103	0	34.0	Self Enquiry	1	NaN	Salaried	Female	
114	0	34.0	Self Enquiry	1	NaN	Salaried	Female	
130	0	43.0	Company Invited	1	NaN	Small Business	Female	
132	1	31.0	Self Enquiry	3	NaN	Salaried	Female	
144	0	32.0	Company Invited	3	NaN	Small Business	Male	
155	0	29.0	Company Invited	1	NaN	Large Business	Male	
175	0	56.0	Self Enquiry	1	NaN	Salaried	Female	
184	0	53.0	Self Enquiry	1	NaN	Small Business	Female	
196	0	35.0	Company Invited	1	NaN	Small Business	Female	
200	0	27.0	Company Invited	1	NaN	Large Business	Male	
208	0	40.0	Company Invited	1	NaN	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
224	0	31.0	NaN	1	NaN	Small Business	Male	
241	0	32.0	Company Invited	3	NaN	Small Business	Male	
255	0	25.0	Self Enquiry	1	NaN	Salaried	Female	
282	0	29.0	Company Invited	3	NaN	Salaried	Male	
284	0	26.0	Company Invited	1	NaN	Small Business	Male	
291	0	36.0	Self Enquiry	1	NaN	Large Business	Male	
309	0	31.0	Self Enquiry	1	NaN	Large Business	Male	
320	0	27.0	Self Enquiry	3	NaN	Salaried	Male	
328	0	33.0	Company Invited	3	NaN	Small Business	Male	
332	0	54.0	Company Invited	1	NaN	Salaried	Female	
349	0	29.0	Company Invited	3	NaN	Salaried	Male	
354	0	30.0	Company Invited	3	NaN	Large Business	Female	
380	0	24.0	Self Enquiry	3	NaN	Small Business	Female	
394	0	31.0	Self Enquiry	1	NaN	Small Business	Female	
396	0	43.0	Self Enquiry	1	NaN	Salaried	Female	
397	0	25.0	Self Enquiry	3	NaN	Salaried	Female	
398	0	37.0	Company Invited	1	NaN	Small Business	Female	
404	0	28.0	Self Enquiry	1	NaN	Small Business	Male	
409	0	42.0	Company Invited	1	NaN	Salaried	Female	
412	0	46.0	Self Enquiry	1	NaN	Small Business	Female	
413	0	42.0	Company Invited	1	NaN	Large Business	Female	
447	0	35.0	Self Enquiry	3	NaN	Small Business	Male	
452	0	45.0	Self Enquiry	3	NaN	Salaried	Male	
454	0	29.0	Self Enquiry	1	NaN	Large Business	Male	
460	0	26.0	Self Enquiry	3	NaN	Small Business	Male	
461	0	35.0	Self Enquiry	3	NaN	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
469	1	32.0	Company Invited	3	NaN	Salaried	Male	
504	1	45.0	Company Invited	3	NaN	Salaried	Female	
517	0	25.0	Self Enquiry	3	NaN	Salaried	Male	
521	0	27.0	Company Invited	3	NaN	Small Business	Female	
522	0	37.0	Self Enquiry	1	NaN	Salaried	Male	
525	1	24.0	Self Enquiry	3	NaN	Salaried	Female	
526	0	39.0	Self Enquiry	1	NaN	Large Business	Female	
570	0	52.0	Company Invited	1	NaN	Small Business	Male	
571	0	26.0	NaN	1	NaN	Salaried	Female	
572	0	29.0	NaN	1	NaN	Small Business	Female	
576	0	27.0	NaN	3	NaN	Small Business	Male	
579	0	34.0	NaN	1	NaN	Small Business	Female	
582	0	40.0	Company Invited	1	NaN	Small Business	Female	
598	1	28.0	NaN	1	NaN	Small Business	Male	
604	0	42.0	Self Enquiry	1	NaN	Salaried	Male	
612	0	28.0	Self Enquiry	3	NaN	Small Business	Female	
622	0	32.0	NaN	3	NaN	Salaried	Male	
630	0	22.0	Self Enquiry	1	NaN	Salaried	Male	
638	0	25.0	Self Enquiry	3	NaN	Small Business	Male	
651	0	47.0	Self Enquiry	3	NaN	Small Business	Female	
661	0	43.0	Self Enquiry	1	NaN	Salaried	Female	
680	0	36.0	Self Enquiry	1	NaN	Salaried	Male	
685	0	26.0	Company Invited	3	NaN	Small Business	Male	
686	0	41.0	Self Enquiry	1	NaN	Small Business	Male	
696	0	45.0	Company Invited	1	NaN	Salaried	Male	
698	0	35.0	Self Enquiry	3	NaN	Small Business	Female	
724	0	24.0	NaN	1	NaN	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
735	0	48.0	Self Enquiry	1	NaN	Salaried	Male	
744	1	37.0	Self Enquiry	1	NaN	Small Business	Female	
761	1	36.0	Self Enquiry	1	NaN	Salaried	Male	
778	0	46.0	Self Enquiry	1	NaN	Salaried	Female	
786	0	27.0	Company Invited	1	NaN	Salaried	Male	
792	1	33.0	Company Invited	1	NaN	Small Business	Female	
801	1	50.0	Company Invited	3	NaN	Salaried	Male	
802	0	33.0	Company Invited	3	NaN	Salaried	Female	
824	0	42.0	Self Enquiry	1	NaN	Small Business	Male	
830	0	41.0	Self Enquiry	1	NaN	Salaried	Male	
835	0	35.0	Self Enquiry	2	NaN	Large Business	Male	
843	0	26.0	NaN	1	NaN	Small Business	Male	
845	0	40.0	Company Invited	1	NaN	Small Business	Female	
854	0	45.0	Self Enquiry	1	NaN	Small Business	Female	
866	0	40.0	Company Invited	3	NaN	Small Business	Male	
872	0	33.0	Company Invited	3	NaN	Small Business	Female	
875	0	44.0	Self Enquiry	1	NaN	Salaried	Male	
917	0	34.0	Self Enquiry	3	NaN	Small Business	Female	
920	0	34.0	Company Invited	1	NaN	Small Business	Female	
923	0	34.0	Company Invited	2	NaN	Salaried	Male	
931	0	30.0	Company Invited	1	NaN	Small Business	Female	
939	1	32.0	Self Enquiry	1	NaN	Salaried	Male	
941	0	30.0	Self Enquiry	1	NaN	Large Business	Female	
949	0	39.0	Self Enquiry	1	NaN	Salaried	Male	
959	0	40.0	Self Enquiry	1	NaN	Salaried	Male	
961	0	35.0	Company Invited	1	NaN	Salaried	Male	
968	0	36.0	Company Invited	3	NaN	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
981	1	35.0	Company Invited	3	NaN	Small Business	Female	
984	0	28.0	Self Enquiry	3	NaN	Salaried	Male	
1006	1	49.0	Company Invited	1	NaN	Salaried	Male	
1013	0	30.0	Self Enquiry	3	NaN	Small Business	Female	
1021	1	25.0	NaN	3	NaN	Salaried	Male	
1047	0	33.0	NaN	3	NaN	Small Business	Male	
1048	0	34.0	Self Enquiry	3	NaN	Salaried	Male	
1051	0	44.0	Company Invited	3	NaN	Small Business	Female	
1058	1	34.0	Self Enquiry	3	NaN	Small Business	Female	
1067	0	47.0	Self Enquiry	3	NaN	Small Business	Female	
1070	0	28.0	Company Invited	3	NaN	Salaried	Male	
1071	0	49.0	Self Enquiry	1	NaN	Small Business	Female	
1088	0	42.0	Self Enquiry	1	NaN	Small Business	Male	
1089	0	37.0	Self Enquiry	1	NaN	Small Business	Male	
1091	0	33.0	Self Enquiry	1	NaN	Salaried	Male	
1112	1	38.0	Self Enquiry	1	NaN	Small Business	Male	
1122	0	29.0	Self Enquiry	1	NaN	Small Business	Male	
1132	0	40.0	Self Enquiry	3	NaN	Salaried	Female	
1133	0	43.0	Self Enquiry	1	NaN	Large Business	Male	
1143	0	45.0	NaN	3	NaN	Small Business	Male	
1145	0	36.0	Self Enquiry	1	NaN	Salaried	Female	
1146	0	34.0	Company Invited	1	NaN	Salaried	Male	
1182	0	36.0	NaN	1	NaN	Small Business	Female	
1186	1	35.0	Company Invited	3	NaN	Salaried	Male	
1187	0	38.0	Company Invited	1	NaN	Salaried	Male	
1217	0	24.0	NaN	1	NaN	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1220	0	36.0	Self Enquiry	3	NaN	Salaried	Female	
1254	0	49.0	Self Enquiry	3	NaN	Small Business	Female	
1304	0	40.0	Self Enquiry	1	NaN	Salaried	Female	
1309	0	26.0	Self Enquiry	3	NaN	Small Business	Male	
1319	0	32.0	Company Invited	3	NaN	Small Business	Male	
1334	0	27.0	Company Invited	1	NaN	Salaried	Female	
1344	0	37.0	Self Enquiry	1	NaN	Small Business	Male	
1345	0	35.0	Self Enquiry	1	NaN	Salaried	Female	
1356	0	41.0	NaN	3	NaN	Small Business	Female	
1376	0	38.0	Self Enquiry	1	NaN	Salaried	Male	
1404	0	42.0	Company Invited	1	NaN	Salaried	Male	
1406	0	54.0	Self Enquiry	1	NaN	Small Business	Female	
1407	0	24.0	Self Enquiry	1	NaN	Salaried	Male	
1416	0	38.0	Self Enquiry	3	NaN	Salaried	Male	
1425	0	33.0	Self Enquiry	1	NaN	Small Business	Female	
1442	1	29.0	Self Enquiry	1	NaN	Small Business	Male	
1454	0	45.0	Self Enquiry	3	NaN	Salaried	Female	
1469	0	34.0	NaN	1	NaN	Small Business	Male	
1516	0	34.0	Company Invited	3	NaN	Small Business	Male	
1545	0	31.0	Self Enquiry	1	NaN	Salaried	Female	
1546	0	35.0	Self Enquiry	3	NaN	Small Business	Male	
1554	0	34.0	Self Enquiry	1	NaN	Small Business	Male	
1573	0	34.0	Self Enquiry	1	NaN	Salaried	Female	
1584	0	34.0	Self Enquiry	1	NaN	Salaried	Female	
1600	0	43.0	Company Invited	1	NaN	Small Business	Female	
1602	1	31.0	Self Enquiry	3	NaN	Salaried	Female	
1612	0	38.0	Self Enquiry	1	NaN	Large Business	Female	
1614	0	32.0	Company Invited	3	NaN	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1625	0	29.0	Company Invited	1	NaN	Large Business	Male	
1645	0	56.0	Self Enquiry	1	NaN	Salaried	Female	
1654	0	53.0	Self Enquiry	1	NaN	Small Business	Female	
1666	0	35.0	Company Invited	1	NaN	Small Business	Female	
1670	0	27.0	Company Invited	1	NaN	Large Business	Male	
1678	0	40.0	Company Invited	1	NaN	Salaried	Male	
1694	0	31.0	NaN	1	NaN	Small Business	Male	
1711	0	32.0	Company Invited	3	NaN	Small Business	Male	
1725	0	25.0	Self Enquiry	1	NaN	Salaried	Female	
1752	0	29.0	Company Invited	3	NaN	Salaried	Male	
1754	0	26.0	Company Invited	1	NaN	Small Business	Male	
1761	0	36.0	Self Enquiry	1	NaN	Large Business	Male	
1779	0	31.0	Self Enquiry	1	NaN	Large Business	Male	
1790	0	27.0	Self Enquiry	3	NaN	Salaried	Male	
1798	0	33.0	Company Invited	3	NaN	Small Business	Male	
1802	0	54.0	Company Invited	1	NaN	Salaried	Female	
1819	0	29.0	Company Invited	3	NaN	Salaried	Male	
1824	0	30.0	Company Invited	3	NaN	Large Business	Female	
1850	0	24.0	Self Enquiry	3	NaN	Small Business	Female	
1864	0	31.0	Self Enquiry	1	NaN	Small Business	Female	
1866	0	43.0	Self Enquiry	1	NaN	Salaried	Female	
1867	0	25.0	Self Enquiry	3	NaN	Salaried	Female	
1868	0	37.0	Company Invited	1	NaN	Small Business	Female	
1874	0	28.0	Self Enquiry	1	NaN	Small Business	Male	
1879	0	42.0	Company Invited	1	NaN	Salaried	Female	
1882	0	46.0	Self Enquiry	1	NaN	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1883	0	42.0	Company Invited	1	NaN	Large Business	Female	
1917	0	35.0	Self Enquiry	3	NaN	Small Business	Male	
1922	0	45.0	Self Enquiry	3	NaN	Salaried	Male	
1924	0	29.0	Self Enquiry	1	NaN	Large Business	Male	
1930	0	26.0	Self Enquiry	3	NaN	Small Business	Male	
1931	0	35.0	Self Enquiry	3	NaN	Small Business	Female	
1939	1	32.0	Company Invited	3	NaN	Salaried	Male	
1952	1	31.0	Self Enquiry	3	NaN	Small Business	Male	
1974	1	45.0	Company Invited	3	NaN	Salaried	Female	
1987	0	25.0	Self Enquiry	3	NaN	Salaried	Male	
1991	0	27.0	Company Invited	3	NaN	Small Business	Female	
1992	0	37.0	Self Enquiry	1	NaN	Salaried	Male	
1995	1	24.0	Self Enquiry	3	NaN	Salaried	Female	
1996	0	39.0	Self Enquiry	1	NaN	Large Business	Female	
2040	0	52.0	Company Invited	1	NaN	Small Business	Male	
2041	0	26.0	NaN	1	NaN	Salaried	Female	
2042	0	29.0	NaN	1	NaN	Small Business	Female	
2046	0	27.0	NaN	3	NaN	Small Business	Male	
2049	0	34.0	NaN	1	NaN	Small Business	Female	
2052	0	40.0	Company Invited	1	NaN	Small Business	Female	
2068	1	28.0	NaN	1	NaN	Small Business	Male	
2074	0	42.0	Self Enquiry	1	NaN	Salaried	Male	
2082	0	28.0	Self Enquiry	3	NaN	Small Business	Female	
2092	0	32.0	NaN	3	NaN	Salaried	Male	
2100	0	22.0	Self Enquiry	1	NaN	Salaried	Male	
2108	0	25.0	Self Enquiry	3	NaN	Small Business	Male	
2121	0	47.0	Self Enquiry	3	NaN	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2131	0	43.0	Self Enquiry	1	NaN	Salaried	Female	
2150	0	36.0	Self Enquiry	1	NaN	Salaried	Male	
2155	0	26.0	Company Invited	3	NaN	Small Business	Male	
2156	0	41.0	Self Enquiry	1	NaN	Small Business	Male	
2166	0	45.0	Company Invited	1	NaN	Salaried	Male	
2168	0	35.0	Self Enquiry	3	NaN	Small Business	Female	
2194	0	24.0	NaN	1	NaN	Small Business	Female	
2205	0	48.0	Self Enquiry	1	NaN	Salaried	Male	
2214	1	37.0	Self Enquiry	1	NaN	Small Business	Female	
2231	1	36.0	Self Enquiry	1	NaN	Salaried	Male	
2248	0	46.0	Self Enquiry	1	NaN	Salaried	Female	
2256	0	27.0	Company Invited	1	NaN	Salaried	Male	
2262	1	33.0	Company Invited	1	NaN	Small Business	Female	
2271	1	50.0	Company Invited	3	NaN	Salaried	Male	
2272	0	33.0	Company Invited	3	NaN	Salaried	Female	
2294	0	42.0	Self Enquiry	1	NaN	Small Business	Male	
2300	0	41.0	Self Enquiry	1	NaN	Salaried	Male	
2305	0	35.0	Self Enquiry	2	NaN	Large Business	Male	
2313	0	26.0	NaN	1	NaN	Small Business	Male	
2315	0	40.0	Company Invited	1	NaN	Small Business	Female	
2324	0	45.0	Self Enquiry	1	NaN	Small Business	Female	
2336	0	40.0	Company Invited	3	NaN	Small Business	Male	
2342	0	33.0	Company Invited	3	NaN	Small Business	Female	
2345	0	44.0	Self Enquiry	1	NaN	Salaried	Male	
2387	1	34.0	Self Enquiry	1	NaN	Salaried	Female	
2390	1	34.0	Company Invited	3	NaN	Salaried	Female	
2393	1	34.0	Self Enquiry	3	NaN	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2401	1	30.0	Company Invited	1	NaN	Small Business	Male	
2409	1	32.0	Self Enquiry	1	NaN	Small Business	Male	
2411	1	30.0	Company Invited	3	NaN	Small Business	Female	
2419	1	39.0	Self Enquiry	3	NaN	Salaried	Male	
2429	1	40.0	Self Enquiry	3	NaN	Small Business	Male	
2431	1	35.0	Company Invited	1	NaN	Small Business	Male	
2438	1	36.0	Self Enquiry	2	NaN	Salaried	Male	
2621	1	20.0	Self Enquiry	1	NaN	Salaried	Male	
2745	0	19.0	Self Enquiry	3	NaN	Small Business	Female	
2957	1	21.0	Self Enquiry	1	NaN	Small Business	Male	
3171	0	19.0	Company Invited	1	NaN	Salaried	Male	
3208	0	29.0	Self Enquiry	3	NaN	Small Business	Male	
3355	1	26.0	Company Invited	3	NaN	Salaried	Male	
3782	1	31.0	Self Enquiry	3	NaN	Small Business	Male	
3809	1	30.0	Company Invited	3	NaN	Large Business	Male	
3846	0	32.0	Self Enquiry	1	NaN	Small Business	Female	
4091	1	20.0	Self Enquiry	1	NaN	Salaried	Male	
4215	0	19.0	Self Enquiry	3	NaN	Small Business	Female	
4427	1	21.0	Self Enquiry	1	NaN	Small Business	Male	
4641	0	19.0	Company Invited	1	NaN	Salaried	Male	
4678	0	29.0	Self Enquiry	3	NaN	Small Business	Male	
4825	1	26.0	Self Enquiry	1	NaN	Salaried	Male	

In [107…

```
# We'll impute these missing values one by one, by taking mean of Duration Of
df.groupby(["Occupation", "ProductPitched"], as_index=False)[
    "DurationOfPitch"
].mean().round(0)
```

Out[107…

	Occupation	ProductPitched	DurationOfPitch
0	Free Lancer	Basic	8.0
1	Free Lancer	Deluxe	NaN
2	Free Lancer	King	NaN
3	Free Lancer	Standard	NaN
4	Free Lancer	Super Deluxe	NaN
5	Large Business	Basic	14.0
6	Large Business	Deluxe	16.0
7	Large Business	King	13.0
8	Large Business	Standard	15.0
9	Large Business	Super Deluxe	13.0
10	Salaried	Basic	15.0
11	Salaried	Deluxe	15.0
12	Salaried	King	13.0
13	Salaried	Standard	16.0
14	Salaried	Super Deluxe	17.0
15	Small Business	Basic	16.0
16	Small Business	Deluxe	17.0
17	Small Business	King	11.0
18	Small Business	Standard	16.0
19	Small Business	Super Deluxe	16.0

In [108…

```
# Impute missing values of Duration of Pitch
df["DurationOfPitch"] = df.groupby(["Occupation", "ProductPitched"])[
    "DurationOfPitch"
].transform(lambda x: round(x.fillna(x.mean())))
```

In [109…

```
# Checking DurationOfPitch
df[df["DurationOfPitch"].isnull()]
```

Out[109…

ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPe

MonthlyIncome

In [110…

```
# Looking at a few rows where #MonthlyIncome is missing
df[df["MonthlyIncome"].isnull()]
```

Out[110…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
11	0	NaN	Self Enquiry	1	21.0	Salaried	Female	
19	0	NaN	Self Enquiry	1	8.0	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
20	0	NaN	Company Invited	1	17.0	Salaried	Female	
26	1	NaN	Company Invited	1	22.0	Salaried	Female	
44	0	NaN	Company Invited	1	6.0	Small Business	Female	
54	0	NaN	Self Enquiry	3	29.0	Small Business	Female	
57	0	NaN	Self Enquiry	1	29.0	Small Business	Female	
75	0	31.0	Self Enquiry	1	15.0	Salaried	Female	
76	0	35.0	Self Enquiry	3	17.0	Small Business	Male	
84	0	34.0	Self Enquiry	1	17.0	Small Business	Male	
88	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
97	0	NaN	Company Invited	3	10.0	Small Business	Male	
140	1	NaN	Self Enquiry	1	15.0	Small Business	Female	
155	0	29.0	Company Invited	1	16.0	Large Business	Male	
175	0	56.0	Self Enquiry	1	15.0	Salaried	Female	
180	0	NaN	Self Enquiry	1	18.0	Small Business	Female	
184	0	53.0	Self Enquiry	1	17.0	Small Business	Female	
196	0	35.0	Company Invited	1	17.0	Small Business	Female	
200	0	27.0	Company Invited	1	16.0	Large Business	Male	
202	0	NaN	Company Invited	1	16.0	Small Business	Male	
224	0	31.0	NaN	1	17.0	Small Business	Male	
238	0	NaN	Self Enquiry	3	10.0	Salaried	Female	
239	1	NaN	Self Enquiry	1	6.0	Salaried	Male	
241	0	32.0	Company Invited	3	17.0	Small Business	Male	
248	0	NaN	Self Enquiry	1	6.0	Small Business	Female	
267	0	NaN	Company Invited	1	11.0	Salaried	Male	
320	0	27.0	Self Enquiry	3	15.0	Salaried	Male	
337	0	NaN	Self Enquiry	1	15.0	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
373	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
380	0	24.0	Self Enquiry	3	17.0	Small Business	Female	
389	0	NaN	Self Enquiry	1	16.0	Salaried	Male	
394	0	31.0	Self Enquiry	1	17.0	Small Business	Female	
398	0	37.0	Company Invited	1	17.0	Small Business	Female	
405	1	NaN	Self Enquiry	1	9.0	Small Business	Male	
430	0	35.0	Self Enquiry	1	28.0	Salaried	Male	
431	0	NaN	Self Enquiry	1	14.0	Salaried	Female	
443	1	NaN	Company Invited	1	10.0	Large Business	Male	
444	0	NaN	Self Enquiry	3	8.0	Small Business	Female	
449	0	NaN	Company Invited	1	14.0	Salaried	Female	
454	0	29.0	Self Enquiry	1	16.0	Large Business	Male	
460	0	26.0	Self Enquiry	3	17.0	Small Business	Male	
482	1	31.0	Self Enquiry	3	19.0	Small Business	Male	
488	0	NaN	Self Enquiry	1	8.0	Salaried	Female	
504	1	45.0	Company Invited	3	15.0	Salaried	Female	
518	0	NaN	Self Enquiry	3	13.0	Small Business	Female	
539	0	NaN	Self Enquiry	3	14.0	Small Business	Male	
570	0	52.0	Company Invited	1	16.0	Small Business	Male	
571	0	26.0	NaN	1	15.0	Salaried	Female	
572	0	29.0	NaN	1	17.0	Small Business	Female	
576	0	27.0	NaN	3	17.0	Small Business	Male	
579	0	34.0	NaN	1	16.0	Small Business	Female	
581	0	NaN	Self Enquiry	1	6.0	Salaried	Male	
582	0	40.0	Company Invited	1	17.0	Small Business	Female	
598	1	28.0	NaN	1	16.0	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
604	0	42.0	Self Enquiry	1	15.0	Salaried	Male	
613	0	NaN	Self Enquiry	2	9.0	Salaried	Male	
619	0	NaN	Self Enquiry	3	6.0	Small Business	Male	
622	0	32.0	NaN	3	15.0	Salaried	Male	
623	0	NaN	Company Invited	1	11.0	Salaried	Male	
634	0	NaN	Self Enquiry	3	9.0	Salaried	Male	
638	0	25.0	Self Enquiry	3	17.0	Small Business	Male	
666	1	NaN	Self Enquiry	1	9.0	Salaried	Female	
676	0	NaN	Self Enquiry	1	27.0	Salaried	Female	
685	0	26.0	Company Invited	3	17.0	Small Business	Male	
696	0	45.0	Company Invited	1	15.0	Salaried	Male	
719	0	NaN	Self Enquiry	3	10.0	Salaried	Female	
724	0	24.0	NaN	1	17.0	Small Business	Female	
725	1	NaN	Self Enquiry	1	20.0	Salaried	Male	
726	0	NaN	Company Invited	1	6.0	Salaried	Female	
735	0	48.0	Self Enquiry	1	15.0	Salaried	Male	
739	0	27.0	Self Enquiry	1	8.0	Salaried	Female	
740	0	NaN	Self Enquiry	1	16.0	Salaried	Male	
756	0	NaN	Company Invited	1	35.0	Small Business	Female	
767	0	NaN	Self Enquiry	1	9.0	Salaried	Female	
781	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
802	0	33.0	Company Invited	3	15.0	Salaried	Female	
822	0	NaN	Company Invited	1	8.0	Salaried	Male	
824	0	42.0	Self Enquiry	1	17.0	Small Business	Male	
835	0	35.0	Self Enquiry	2	14.0	Large Business	Male	
843	0	26.0	NaN	1	16.0	Small Business	Male	
845	0	40.0	Company Invited	1	17.0	Small Business	Female	
854	0	45.0	Self Enquiry	1	16.0	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
865	0	NaN	Self Enquiry	3	35.0	Salaried	Male	
866	0	40.0	Company Invited	3	17.0	Small Business	Male	
893	0	NaN	Self Enquiry	1	6.0	Salaried	Female	
920	0	34.0	Company Invited	1	17.0	Small Business	Female	
929	0	NaN	Company Invited	1	8.0	Salaried	Male	
940	1	NaN	Self Enquiry	1	29.0	Small Business	Male	
960	0	NaN	Company Invited	3	6.0	Small Business	Female	
961	0	35.0	Company Invited	1	15.0	Salaried	Male	
991	0	NaN	Self Enquiry	3	8.0	Small Business	Male	
995	0	NaN	Self Enquiry	1	12.0	Small Business	Female	
998	0	NaN	Self Enquiry	1	8.0	Small Business	Male	
1006	1	49.0	Company Invited	1	15.0	Salaried	Male	
1021	1	25.0	NaN	3	15.0	Salaried	Male	
1025	0	NaN	Self Enquiry	3	10.0	Small Business	Female	
1029	0	NaN	Company Invited	1	15.0	Salaried	Male	
1036	1	NaN	Company Invited	1	8.0	Salaried	Male	
1047	0	33.0	NaN	3	17.0	Small Business	Male	
1065	0	NaN	Self Enquiry	1	10.0	Salaried	Male	
1066	0	NaN	Self Enquiry	1	10.0	Small Business	Female	
1085	1	NaN	Company Invited	1	9.0	Salaried	Female	
1089	0	37.0	Self Enquiry	1	17.0	Small Business	Male	
1091	0	33.0	Self Enquiry	1	15.0	Salaried	Male	
1098	0	NaN	Company Invited	1	14.0	Salaried	Male	
1113	0	NaN	Company Invited	1	6.0	Large Business	Male	
1120	0	NaN	Self Enquiry	3	22.0	Salaried	Female	
1143	0	45.0	NaN	3	17.0	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1149	0	NaN	Self Enquiry	1	25.0	Salaried	Male	
1157	0	NaN	Company Invited	1	14.0	Salaried	Female	
1163	0	NaN	Self Enquiry	1	16.0	Small Business	Female	
1168	0	NaN	Company Invited	1	8.0	Large Business	Female	
1182	0	36.0	NaN	1	17.0	Small Business	Female	
1188	0	NaN	Self Enquiry	3	11.0	Small Business	Male	
1201	1	NaN	Self Enquiry	1	14.0	Small Business	Male	
1208	0	NaN	Self Enquiry	1	11.0	Small Business	Male	
1217	0	24.0	NaN	1	16.0	Small Business	Male	
1230	0	NaN	Self Enquiry	1	35.0	Small Business	Male	
1267	0	NaN	Company Invited	3	16.0	Salaried	Male	
1276	0	NaN	Self Enquiry	3	15.0	Small Business	Male	
1280	0	NaN	Self Enquiry	2	14.0	Salaried	Male	
1291	1	NaN	Self Enquiry	1	16.0	Small Business	Male	
1292	0	NaN	Company Invited	3	26.0	Salaried	Male	
1304	0	40.0	Self Enquiry	1	15.0	Salaried	Female	
1318	0	NaN	Company Invited	1	26.0	Small Business	Male	
1325	0	NaN	Self Enquiry	1	14.0	Salaried	Male	
1335	0	NaN	Self Enquiry	1	25.0	Salaried	Male	
1341	0	NaN	Self Enquiry	1	26.0	Salaried	Male	
1344	0	37.0	Self Enquiry	1	17.0	Small Business	Male	
1347	0	NaN	Company Invited	2	8.0	Salaried	Male	
1356	0	41.0	NaN	3	17.0	Small Business	Female	
1360	0	NaN	Self Enquiry	1	10.0	Small Business	Female	
1393	0	NaN	Self Enquiry	3	15.0	Small Business	Male	
1404	0	42.0	Company Invited	1	15.0	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1406	0	54.0	Self Enquiry	1	17.0	Small Business	Female	
1412	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
1413	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
1416	0	38.0	Self Enquiry	3	15.0	Salaried	Male	
1429	0	NaN	Self Enquiry	1	30.0	Salaried	Male	
1459	0	NaN	Self Enquiry	1	19.0	Salaried	Male	
1460	0	NaN	Self Enquiry	1	34.0	Small Business	Female	
1469	0	34.0	NaN	1	17.0	Small Business	Male	
1481	0	NaN	Self Enquiry	1	21.0	Salaried	Female	
1489	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
1490	0	NaN	Company Invited	1	17.0	Salaried	Female	
1496	1	NaN	Company Invited	1	22.0	Salaried	Female	
1514	0	NaN	Company Invited	1	6.0	Small Business	Female	
1524	0	NaN	Self Enquiry	3	29.0	Small Business	Female	
1527	0	NaN	Self Enquiry	1	29.0	Small Business	Female	
1545	0	31.0	Self Enquiry	1	15.0	Salaried	Female	
1546	0	35.0	Self Enquiry	3	17.0	Small Business	Male	
1554	0	34.0	Self Enquiry	1	17.0	Small Business	Male	
1558	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
1567	0	28.0	Company Invited	3	10.0	Small Business	Male	
1610	1	NaN	Self Enquiry	1	15.0	Small Business	Female	
1612	0	38.0	Self Enquiry	1	16.0	Large Business	Female	
1625	0	29.0	Company Invited	1	16.0	Large Business	Male	
1645	0	56.0	Self Enquiry	1	15.0	Salaried	Female	
1650	0	NaN	Self Enquiry	1	18.0	Small Business	Female	
1654	0	53.0	Self Enquiry	1	17.0	Small Business	Female	
1666	0	35.0	Company Invited	1	17.0	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1670	0	27.0	Company Invited	1	16.0	Large Business	Male	
1672	0	NaN	Company Invited	1	16.0	Small Business	Male	
1694	0	31.0	NaN	1	17.0	Small Business	Male	
1708	0	NaN	Self Enquiry	3	10.0	Salaried	Female	
1709	1	NaN	Self Enquiry	1	6.0	Salaried	Male	
1711	0	32.0	Company Invited	3	17.0	Small Business	Male	
1718	0	NaN	Self Enquiry	1	6.0	Small Business	Female	
1737	0	NaN	Company Invited	1	11.0	Salaried	Male	
1790	0	27.0	Self Enquiry	3	15.0	Salaried	Male	
1807	0	NaN	Self Enquiry	1	15.0	Salaried	Male	
1843	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
1850	0	24.0	Self Enquiry	3	17.0	Small Business	Female	
1859	0	NaN	Self Enquiry	1	16.0	Salaried	Male	
1864	0	31.0	Self Enquiry	1	17.0	Small Business	Female	
1868	0	37.0	Company Invited	1	17.0	Small Business	Female	
1875	1	NaN	Self Enquiry	1	9.0	Small Business	Male	
1900	0	35.0	Self Enquiry	1	28.0	Salaried	Male	
1901	0	NaN	Self Enquiry	1	5.0	Salaried	Female	
1913	1	NaN	Company Invited	1	10.0	Large Business	Male	
1914	0	NaN	Self Enquiry	3	8.0	Small Business	Female	
1919	0	NaN	Company Invited	1	14.0	Salaried	Female	
1924	0	29.0	Self Enquiry	1	16.0	Large Business	Male	
1930	0	26.0	Self Enquiry	3	17.0	Small Business	Male	
1952	1	31.0	Self Enquiry	3	17.0	Small Business	Male	
1958	0	NaN	Self Enquiry	1	8.0	Salaried	Female	
1974	1	45.0	Company Invited	3	15.0	Salaried	Female	
1988	0	NaN	Self Enquiry	3	13.0	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2009	0	NaN	Self Enquiry	3	14.0	Small Business	Male	
2040	0	52.0	Company Invited	1	16.0	Small Business	Male	
2041	0	26.0	NaN	1	15.0	Salaried	Female	
2042	0	29.0	NaN	1	17.0	Small Business	Female	
2046	0	27.0	NaN	3	17.0	Small Business	Male	
2049	0	34.0	NaN	1	16.0	Small Business	Female	
2051	0	NaN	Self Enquiry	1	6.0	Salaried	Male	
2052	0	40.0	Company Invited	1	17.0	Small Business	Female	
2068	1	28.0	NaN	1	16.0	Small Business	Male	
2074	0	42.0	Self Enquiry	1	15.0	Salaried	Male	
2083	0	NaN	Self Enquiry	2	9.0	Salaried	Male	
2089	0	NaN	Self Enquiry	3	6.0	Small Business	Male	
2092	0	32.0	NaN	3	15.0	Salaried	Male	
2093	0	NaN	Company Invited	1	11.0	Salaried	Male	
2104	0	NaN	Self Enquiry	3	9.0	Salaried	Male	
2108	0	25.0	Self Enquiry	3	17.0	Small Business	Male	
2136	1	NaN	Self Enquiry	1	9.0	Salaried	Female	
2146	0	NaN	Self Enquiry	1	27.0	Salaried	Female	
2155	0	26.0	Company Invited	3	17.0	Small Business	Male	
2166	0	45.0	Company Invited	1	15.0	Salaried	Male	
2189	0	NaN	Self Enquiry	3	10.0	Salaried	Female	
2194	0	24.0	NaN	1	17.0	Small Business	Female	
2195	1	NaN	Self Enquiry	1	20.0	Salaried	Male	
2196	0	NaN	Company Invited	1	6.0	Salaried	Female	
2205	0	48.0	Self Enquiry	1	15.0	Salaried	Male	
2209	0	27.0	Self Enquiry	1	8.0	Salaried	Female	
2210	0	NaN	Self Enquiry	1	16.0	Salaried	Male	
2226	0	NaN	Company Invited	1	35.0	Small Business	Female	
2237	0	NaN	Self Enquiry	1	9.0	Salaried	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2251	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
2272	0	33.0	Company Invited	3	15.0	Salaried	Female	
2292	0	NaN	Company Invited	1	8.0	Salaried	Male	
2294	0	42.0	Self Enquiry	1	17.0	Small Business	Male	
2305	0	35.0	Self Enquiry	2	14.0	Large Business	Male	
2313	0	26.0	NaN	1	16.0	Small Business	Male	
2315	0	40.0	Company Invited	1	17.0	Small Business	Female	
2324	0	45.0	Self Enquiry	1	16.0	Small Business	Female	
2335	0	NaN	Self Enquiry	3	35.0	Salaried	Male	
2336	0	40.0	Company Invited	3	17.0	Small Business	Male	
2363	0	NaN	Self Enquiry	1	7.0	Salaried	Female	
2390	1	34.0	Company Invited	3	15.0	Salaried	Female	
2399	1	NaN	Company Invited	3	19.0	Large Business	Female	
2410	1	NaN	Self Enquiry	1	30.0	Small Business	Male	
2430	1	NaN	Self Enquiry	1	14.0	Small Business	Female	
2431	1	35.0	Company Invited	1	16.0	Small Business	Male	

In [111…
```
df.MonthlyIncome.mean()
```

Out[111…
```
23592.533619763693
```

In [112…
```
# We'll impute these missing values one by one, by taking mean of MonthlyInco
df.groupby(["Gender", "Occupation", "Designation"], as_index=False)[
    "MonthlyIncome"
].mean().round(0)
```

Out[112…

	Gender	Occupation	Designation	MonthlyIncome
0	Female	Free Lancer	AVP	NaN
1	Female	Free Lancer	Executive	NaN
2	Female	Free Lancer	Manager	NaN
3	Female	Free Lancer	Senior Manager	NaN

	Gender	Occupation	Designation	MonthlyIncome
4	Female	Free Lancer	VP	NaN
5	Female	Large Business	AVP	31802.0
6	Female	Large Business	Executive	20146.0
7	Female	Large Business	Manager	22101.0
8	Female	Large Business	Senior Manager	27140.0
9	Female	Large Business	VP	36583.0
10	Female	Salaried	AVP	32358.0
11	Female	Salaried	Executive	19932.0
12	Female	Salaried	Manager	22476.0
13	Female	Salaried	Senior Manager	26895.0
14	Female	Salaried	VP	35911.0
15	Female	Small Business	AVP	32106.0
16	Female	Small Business	Executive	19859.0
17	Female	Small Business	Manager	22639.0
18	Female	Small Business	Senior Manager	26720.0
19	Female	Small Business	VP	35084.0
20	Male	Free Lancer	AVP	NaN
21	Male	Free Lancer	Executive	18929.0
22	Male	Free Lancer	Manager	NaN
23	Male	Free Lancer	Senior Manager	NaN
24	Male	Free Lancer	VP	NaN
25	Male	Large Business	AVP	29959.0
26	Male	Large Business	Executive	19894.0
27	Male	Large Business	Manager	22238.0
28	Male	Large Business	Senior Manager	26780.0
29	Male	Large Business	VP	36071.0
30	Male	Salaried	AVP	32554.0
31	Male	Salaried	Executive	19778.0
32	Male	Salaried	Manager	22889.0
33	Male	Salaried	Senior Manager	26292.0
34	Male	Salaried	VP	36089.0
35	Male	Small Business	AVP	32079.0
36	Male	Small Business	Executive	19829.0
37	Male	Small Business	Manager	22697.0
38	Male	Small Business	Senior Manager	26599.0
39	Male	Small Business	VP	35997.0

In [113…
```
# Impute missing values of Monthly Income
df["MonthlyIncome"] = df.groupby(["Gender", "Occupation", "Designation"])[
    "MonthlyIncome"
].transform(lambda x: round(x.fillna(x.mean())))
```

In [114…
```
df[df["MonthlyIncome"].isnull()]
```

Out[114…

ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPe

Age

In [115…
```
# Looking rows where Age is null
df[df["Age"].isnull()]
```

Out[115…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
4	0	NaN	Self Enquiry	1	8.0	Small Business	Male	
11	0	NaN	Self Enquiry	1	21.0	Salaried	Female	
19	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
20	0	NaN	Company Invited	1	17.0	Salaried	Female	
21	1	NaN	Self Enquiry	3	15.0	Salaried	Male	
26	1	NaN	Company Invited	1	22.0	Salaried	Female	
44	0	NaN	Company Invited	1	6.0	Small Business	Female	
51	1	NaN	Self Enquiry	1	11.0	Large Business	Male	
54	0	NaN	Self Enquiry	3	29.0	Small Business	Female	
57	0	NaN	Self Enquiry	1	29.0	Small Business	Female	
69	1	NaN	Self Enquiry	1	15.0	Small Business	Male	
88	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
97	0	NaN	Company Invited	3	10.0	Small Business	Male	
140	1	NaN	Self Enquiry	1	15.0	Small Business	Female	
141	0	NaN	Self Enquiry	1	35.0	Small Business	Male	
180	0	NaN	Self Enquiry	1	18.0	Small Business	Female	
183	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
195	0	NaN	Self Enquiry	1	27.0	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
202	0	NaN	Company Invited	1	16.0	Small Business	Male	
238	0	NaN	Self Enquiry	3	10.0	Salaried	Female	
239	1	NaN	Self Enquiry	1	6.0	Salaried	Male	
248	0	NaN	Self Enquiry	1	6.0	Small Business	Female	
259	1	NaN	Company Invited	1	35.0	Small Business	Male	
264	1	NaN	Self Enquiry	1	8.0	Salaried	Male	
267	0	NaN	Company Invited	1	11.0	Salaried	Male	
298	0	NaN	Company Invited	1	24.0	Salaried	Male	
323	1	NaN	Self Enquiry	1	8.0	Small Business	Male	
334	0	NaN	Self Enquiry	1	14.0	Salaried	Male	
337	0	NaN	Self Enquiry	1	15.0	Salaried	Male	
364	0	NaN	Self Enquiry	1	16.0	Small Business	Female	
373	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
389	0	NaN	Self Enquiry	1	16.0	Salaried	Male	
391	0	NaN	Self Enquiry	1	8.0	Small Business	Female	
405	1	NaN	Self Enquiry	1	9.0	Small Business	Male	
431	0	NaN	Self Enquiry	1	14.0	Salaried	Female	
436	1	NaN	Self Enquiry	1	16.0	Small Business	Male	
443	1	NaN	Company Invited	1	10.0	Large Business	Male	
444	0	NaN	Self Enquiry	3	8.0	Small Business	Female	
449	0	NaN	Company Invited	1	14.0	Salaried	Female	
481	0	NaN	Self Enquiry	1	6.0	Salaried	Male	
483	0	NaN	Self Enquiry	1	31.0	Salaried	Male	
488	0	NaN	Self Enquiry	1	8.0	Salaried	Female	
496	0	NaN	Self Enquiry	3	28.0	Large Business	Male	
518	0	NaN	Self Enquiry	3	13.0	Small Business	Female	
539	0	NaN	Self Enquiry	3	14.0	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
543	0	NaN	Company Invited	1	30.0	Small Business	Male	
565	0	NaN	Self Enquiry	1	16.0	Small Business	Male	
581	0	NaN	Self Enquiry	1	6.0	Salaried	Male	
613	0	NaN	Self Enquiry	2	9.0	Salaried	Male	
618	0	NaN	Self Enquiry	1	8.0	Small Business	Male	
619	0	NaN	Self Enquiry	3	6.0	Small Business	Male	
623	0	NaN	Company Invited	1	11.0	Salaried	Male	
634	0	NaN	Self Enquiry	3	9.0	Salaried	Male	
639	0	NaN	Self Enquiry	1	6.0	Large Business	Female	
666	1	NaN	Self Enquiry	1	9.0	Salaried	Female	
676	0	NaN	Self Enquiry	1	27.0	Salaried	Female	
691	0	NaN	Company Invited	1	15.0	Small Business	Male	
712	0	NaN	Self Enquiry	1	19.0	Salaried	Female	
718	0	NaN	Company Invited	1	29.0	Salaried	Male	
719	0	NaN	Self Enquiry	3	10.0	Salaried	Female	
725	1	NaN	Self Enquiry	1	20.0	Salaried	Male	
726	0	NaN	Company Invited	1	6.0	Salaried	Female	
740	0	NaN	Self Enquiry	1	16.0	Salaried	Male	
756	0	NaN	Company Invited	1	35.0	Small Business	Female	
767	0	NaN	Self Enquiry	1	9.0	Salaried	Female	
781	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
783	0	NaN	Self Enquiry	1	13.0	Large Business	Female	
788	0	NaN	Self Enquiry	1	16.0	Salaried	Female	
796	1	NaN	Self Enquiry	1	10.0	Large Business	Male	
822	0	NaN	Company Invited	1	8.0	Salaried	Male	
841	0	NaN	Self Enquiry	1	30.0	Small Business	Male	
863	0	NaN	Self Enquiry	2	8.0	Salaried	Male	
865	0	NaN	Self Enquiry	3	35.0	Salaried	Male	
874	0	NaN	Self Enquiry	1	13.0	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
886	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
893	0	NaN	Self Enquiry	1	6.0	Salaried	Female	
900	0	NaN	Company Invited	1	9.0	Large Business	Male	
924	0	NaN	Self Enquiry	1	12.0	Salaried	Male	
929	0	NaN	Company Invited	1	8.0	Salaried	Male	
940	1	NaN	Self Enquiry	1	29.0	Small Business	Male	
943	0	NaN	Self Enquiry	2	6.0	Salaried	Female	
957	0	NaN	Company Invited	1	22.0	Salaried	Male	
960	0	NaN	Company Invited	3	6.0	Small Business	Female	
965	0	NaN	Self Enquiry	1	25.0	Small Business	Male	
991	0	NaN	Self Enquiry	3	8.0	Small Business	Male	
995	0	NaN	Self Enquiry	1	12.0	Small Business	Female	
998	0	NaN	Self Enquiry	1	8.0	Small Business	Male	
1001	0	NaN	Self Enquiry	1	17.0	Small Business	Female	
1004	0	NaN	Self Enquiry	1	13.0	Salaried	Male	
1020	0	NaN	Self Enquiry	1	6.0	Large Business	Male	
1022	0	NaN	Company Invited	1	11.0	Large Business	Male	
1025	0	NaN	Self Enquiry	3	10.0	Small Business	Female	
1029	0	NaN	Company Invited	1	15.0	Salaried	Male	
1032	1	NaN	Company Invited	1	8.0	Salaried	Female	
1036	1	NaN	Company Invited	1	8.0	Salaried	Male	
1046	0	NaN	Self Enquiry	1	26.0	Salaried	Male	
1050	0	NaN	Company Invited	1	15.0	Small Business	Female	
1065	0	NaN	Self Enquiry	1	10.0	Salaried	Male	
1066	0	NaN	Self Enquiry	1	10.0	Small Business	Female	
1085	1	NaN	Company Invited	1	9.0	Salaried	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1098	0	NaN	Company Invited	1	14.0	Salaried	Male	
1113	0	NaN	Company Invited	1	6.0	Large Business	Male	
1120	0	NaN	Self Enquiry	3	22.0	Salaried	Female	
1130	0	NaN	Self Enquiry	1	34.0	Salaried	Male	
1149	0	NaN	Self Enquiry	1	25.0	Salaried	Male	
1157	0	NaN	Company Invited	1	14.0	Salaried	Female	
1163	0	NaN	Self Enquiry	1	16.0	Small Business	Female	
1168	0	NaN	Company Invited	1	8.0	Large Business	Female	
1169	0	NaN	Self Enquiry	1	14.0	Small Business	Female	
1188	0	NaN	Self Enquiry	3	11.0	Small Business	Male	
1201	1	NaN	Self Enquiry	1	14.0	Small Business	Male	
1207	0	NaN	Self Enquiry	1	28.0	Large Business	Male	
1208	0	NaN	Self Enquiry	1	11.0	Small Business	Male	
1226	0	NaN	Company Invited	1	16.0	Salaried	Male	
1227	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
1230	0	NaN	Self Enquiry	1	35.0	Small Business	Male	
1248	0	NaN	Self Enquiry	1	14.0	Small Business	Female	
1267	0	NaN	Company Invited	3	16.0	Salaried	Male	
1269	0	NaN	Self Enquiry	2	8.0	Salaried	Male	
1272	0	NaN	Self Enquiry	1	12.0	Salaried	Female	
1276	0	NaN	Self Enquiry	3	15.0	Small Business	Male	
1280	0	NaN	Self Enquiry	2	14.0	Salaried	Male	
1286	0	NaN	Self Enquiry	1	8.0	Salaried	Female	
1291	1	NaN	Self Enquiry	1	16.0	Small Business	Male	
1292	0	NaN	Company Invited	3	26.0	Salaried	Male	
1307	0	NaN	Self Enquiry	1	6.0	Small Business	Female	
1318	0	NaN	Company Invited	1	26.0	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1325	0	NaN	Self Enquiry	1	14.0	Salaried	Male	
1328	0	NaN	Self Enquiry	3	29.0	Small Business	Female	
1335	0	NaN	Self Enquiry	1	25.0	Salaried	Male	
1341	0	NaN	Self Enquiry	1	26.0	Salaried	Male	
1347	0	NaN	Company Invited	2	8.0	Salaried	Male	
1360	0	NaN	Self Enquiry	1	10.0	Small Business	Female	
1393	0	NaN	Self Enquiry	3	15.0	Small Business	Male	
1412	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
1413	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
1423	0	NaN	Self Enquiry	1	6.0	Salaried	Male	
1429	0	NaN	Self Enquiry	1	30.0	Salaried	Male	
1434	0	NaN	Company Invited	3	26.0	Salaried	Male	
1459	0	NaN	Self Enquiry	1	19.0	Salaried	Male	
1460	0	NaN	Self Enquiry	1	34.0	Small Business	Female	
1474	0	NaN	Self Enquiry	1	8.0	Small Business	Male	
1481	0	NaN	Self Enquiry	1	21.0	Salaried	Female	
1489	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
1490	0	NaN	Company Invited	1	17.0	Salaried	Female	
1491	1	NaN	Self Enquiry	3	15.0	Salaried	Male	
1496	1	NaN	Company Invited	1	22.0	Salaried	Female	
1508	0	NaN	Self Enquiry	1	11.0	Salaried	Female	
1514	0	NaN	Company Invited	1	6.0	Small Business	Female	
1521	1	NaN	Self Enquiry	1	11.0	Large Business	Male	
1524	0	NaN	Self Enquiry	3	29.0	Small Business	Female	
1527	0	NaN	Self Enquiry	1	29.0	Small Business	Female	
1539	1	NaN	Self Enquiry	1	15.0	Small Business	Male	
1558	0	NaN	Self Enquiry	1	8.0	Salaried	Male	
1610	1	NaN	Self Enquiry	1	15.0	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1611	0	NaN	Self Enquiry	1	35.0	Small Business	Male	
1650	0	NaN	Self Enquiry	1	18.0	Small Business	Female	
1653	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
1665	0	NaN	Self Enquiry	1	27.0	Salaried	Male	
1672	0	NaN	Company Invited	1	16.0	Small Business	Male	
1708	0	NaN	Self Enquiry	3	10.0	Salaried	Female	
1709	1	NaN	Self Enquiry	1	6.0	Salaried	Male	
1718	0	NaN	Self Enquiry	1	6.0	Small Business	Female	
1729	1	NaN	Company Invited	1	35.0	Small Business	Male	
1734	1	NaN	Self Enquiry	1	8.0	Salaried	Male	
1737	0	NaN	Company Invited	1	11.0	Salaried	Male	
1768	0	NaN	Company Invited	1	24.0	Salaried	Male	
1793	1	NaN	Self Enquiry	1	8.0	Small Business	Male	
1804	0	NaN	Self Enquiry	1	14.0	Salaried	Male	
1807	0	NaN	Self Enquiry	1	15.0	Salaried	Male	
1834	0	NaN	Self Enquiry	1	16.0	Small Business	Female	
1843	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
1859	0	NaN	Self Enquiry	1	16.0	Salaried	Male	
1861	0	NaN	Self Enquiry	1	8.0	Small Business	Female	
1875	1	NaN	Self Enquiry	1	9.0	Small Business	Male	
1901	0	NaN	Self Enquiry	1	5.0	Salaried	Female	
1906	1	NaN	Self Enquiry	1	16.0	Small Business	Male	
1913	1	NaN	Company Invited	1	10.0	Large Business	Male	
1914	0	NaN	Self Enquiry	3	8.0	Small Business	Female	
1919	0	NaN	Company Invited	1	14.0	Salaried	Female	
1951	0	NaN	Self Enquiry	1	6.0	Salaried	Male	
1953	0	NaN	Self Enquiry	1	31.0	Salaried	Male	
1958	0	NaN	Self Enquiry	1	8.0	Salaried	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
1966	0	NaN	Self Enquiry	3	28.0	Large Business	Male	
1984	1	NaN	Company Invited	1	9.0	Salaried	Male	
1988	0	NaN	Self Enquiry	3	13.0	Small Business	Female	
2009	0	NaN	Self Enquiry	3	14.0	Small Business	Male	
2013	0	NaN	Company Invited	1	30.0	Small Business	Male	
2035	0	NaN	Self Enquiry	1	16.0	Small Business	Male	
2051	0	NaN	Self Enquiry	1	6.0	Salaried	Male	
2083	0	NaN	Self Enquiry	2	9.0	Salaried	Male	
2088	0	NaN	Self Enquiry	1	8.0	Small Business	Male	
2089	0	NaN	Self Enquiry	3	6.0	Small Business	Male	
2093	0	NaN	Company Invited	1	11.0	Salaried	Male	
2104	0	NaN	Self Enquiry	3	9.0	Salaried	Male	
2109	0	NaN	Self Enquiry	1	6.0	Large Business	Female	
2136	1	NaN	Self Enquiry	1	9.0	Salaried	Female	
2146	0	NaN	Self Enquiry	1	27.0	Salaried	Female	
2161	0	NaN	Company Invited	1	15.0	Small Business	Male	
2182	0	NaN	Self Enquiry	1	19.0	Salaried	Female	
2188	0	NaN	Company Invited	1	29.0	Salaried	Male	
2189	0	NaN	Self Enquiry	3	10.0	Salaried	Female	
2195	1	NaN	Self Enquiry	1	20.0	Salaried	Male	
2196	0	NaN	Company Invited	1	6.0	Salaried	Female	
2210	0	NaN	Self Enquiry	1	16.0	Salaried	Male	
2226	0	NaN	Company Invited	1	35.0	Small Business	Female	
2237	0	NaN	Self Enquiry	1	9.0	Salaried	Female	
2251	0	NaN	Self Enquiry	1	6.0	Small Business	Male	
2253	0	NaN	Self Enquiry	1	13.0	Large Business	Female	
2258	0	NaN	Self Enquiry	1	16.0	Salaried	Female	
2266	1	NaN	Self Enquiry	1	10.0	Large Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2292	0	NaN	Company Invited	1	8.0	Salaried	Male	
2311	0	NaN	Self Enquiry	1	30.0	Small Business	Male	
2333	0	NaN	Self Enquiry	2	8.0	Salaried	Male	
2335	0	NaN	Self Enquiry	3	35.0	Salaried	Male	
2344	0	NaN	Self Enquiry	1	13.0	Salaried	Male	
2356	0	NaN	Self Enquiry	1	7.0	Small Business	Male	
2363	0	NaN	Self Enquiry	1	7.0	Salaried	Female	
2370	0	NaN	Company Invited	1	9.0	Large Business	Male	
2394	1	NaN	Company Invited	1	8.0	Salaried	Female	
2399	1	NaN	Company Invited	3	19.0	Large Business	Female	
2410	1	NaN	Self Enquiry	1	30.0	Small Business	Male	
2413	1	NaN	Self Enquiry	3	21.0	Small Business	Male	
2427	1	NaN	Self Enquiry	3	22.0	Small Business	Male	
2430	1	NaN	Self Enquiry	1	14.0	Small Business	Female	
2435	1	NaN	Self Enquiry	2	26.0	Small Business	Female	

In [116…
```
# We'll impute these missing values one by one, by taking mean of Age conside
df.groupby(["Occupation", "Designation", "ProductPitched"])["Age"].mean().rou
```

Out[116…
```
Occupation      Designation     ProductPitched
Free Lancer     AVP             Basic             NaN
                                Deluxe            NaN
                                King              NaN
                                Standard          NaN
                                Super Deluxe      NaN
                Executive       Basic            38.0
                                Deluxe            NaN
                                King              NaN
                                Standard          NaN
                                Super Deluxe      NaN
                Manager         Basic             NaN
                                Deluxe            NaN
                                King              NaN
                                Standard          NaN
                                Super Deluxe      NaN
                Senior Manager  Basic             NaN
                                Deluxe            NaN
                                King              NaN
                                Standard          NaN
                                Super Deluxe      NaN
                VP              Basic             NaN
```

		Deluxe	NaN
		King	NaN
		Standard	NaN
		Super Deluxe	NaN
Large Business	AVP	Basic	NaN
		Deluxe	NaN
		King	NaN
		Standard	NaN
		Super Deluxe	48.0
	Executive	Basic	32.0
		Deluxe	NaN
		King	NaN
		Standard	NaN
		Super Deluxe	NaN
	Manager	Basic	NaN
		Deluxe	38.0
		King	NaN
		Standard	NaN
		Super Deluxe	NaN
	Senior Manager	Basic	NaN
		Deluxe	NaN
		King	NaN
		Standard	42.0
		Super Deluxe	NaN
	VP	Basic	NaN
		Deluxe	NaN
		King	47.0
		Standard	NaN
		Super Deluxe	NaN
Salaried	AVP	Basic	NaN
		Deluxe	NaN
		King	NaN
		Standard	NaN
		Super Deluxe	47.0
	Executive	Basic	34.0
		Deluxe	NaN
		King	NaN
		Standard	NaN
		Super Deluxe	NaN
	Manager	Basic	NaN
		Deluxe	37.0
		King	NaN
		Standard	NaN
		Super Deluxe	NaN
	Senior Manager	Basic	NaN
		Deluxe	NaN
		King	NaN
		Standard	40.0
		Super Deluxe	NaN
	VP	Basic	NaN
		Deluxe	NaN
		King	48.0
		Standard	NaN
		Super Deluxe	NaN
Small Business	AVP	Basic	NaN
		Deluxe	NaN
		King	NaN
		Standard	NaN
		Super Deluxe	49.0
	Executive	Basic	33.0
		Deluxe	NaN
		King	NaN
		Standard	NaN
		Super Deluxe	NaN
	Manager	Basic	NaN
		Deluxe	37.0
		King	NaN
		Standard	NaN
		Super Deluxe	NaN

```
                    Senior Manager  Basic              NaN
                                    Deluxe             NaN
                                    King               NaN
                                    Standard          41.0
                                    Super Deluxe       NaN
                    VP              Basic              NaN
                                    Deluxe             NaN
                                    King              48.0
                                    Standard           NaN
                                    Super Deluxe       NaN
Name: Age, dtype: float64
```

In [117…
```
# Impute missing values of Age
df["Age"] = df.groupby(["Occupation", "Designation", "ProductPitched"])[
    "Age"
].transform(lambda x: round(x.fillna(x.mean())))
```

In [118…
```
df[df["Age"].isnull()]
```

Out[118…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPe

Number Of Trips

In [119…
```
# Cheeking the rows with NumberOfTrips missing
df[df["NumberOfTrips"].isnull()]
```

Out[119…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2469	0	54.0	Self Enquiry	1	12.0	Salaried	Female	
2473	0	47.0	Self Enquiry	3	9.0	Small Business	Female	
2506	0	51.0	Self Enquiry	1	14.0	Small Business	Female	
2549	0	60.0	Company Invited	2	9.0	Salaried	Female	
2550	0	51.0	Company Invited	1	7.0	Salaried	Female	
2556	0	55.0	Company Invited	2	33.0	Salaried	Female	
2563	0	44.0	Company Invited	3	33.0	Salaried	Male	
2567	0	52.0	Self Enquiry	1	13.0	Salaried	Male	
2591	0	42.0	Company Invited	1	17.0	Salaried	Male	
2630	0	41.0	Self Enquiry	1	11.0	Small Business	Female	
2631	0	56.0	Self Enquiry	1	21.0	Small Business	Male	
2675	0	43.0	Self Enquiry	1	11.0	Large Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2677	0	51.0	Self Enquiry	3	7.0	Small Business	Female	
2681	0	53.0	Company Invited	3	9.0	Salaried	Male	
2688	0	46.0	Self Enquiry	1	7.0	Salaried	Male	
2701	0	41.0	Self Enquiry	1	9.0	Small Business	Male	
2714	0	56.0	Self Enquiry	1	7.0	Small Business	Male	
2723	0	51.0	Self Enquiry	1	11.0	Salaried	Male	
2724	0	54.0	Self Enquiry	1	10.0	Small Business	Male	
2734	0	50.0	Company Invited	1	17.0	Salaried	Female	
2758	0	40.0	Self Enquiry	1	17.0	Small Business	Male	
2770	0	40.0	Company Invited	1	6.0	Small Business	Male	
2773	0	48.0	Self Enquiry	1	12.0	Salaried	Male	
2836	0	55.0	Self Enquiry	1	12.0	Small Business	Male	
2844	0	40.0	Company Invited	1	7.0	Salaried	Male	
2861	0	41.0	Self Enquiry	3	9.0	Salaried	Female	
2869	0	51.0	Self Enquiry	1	36.0	Salaried	Male	
2873	0	47.0	Self Enquiry	1	9.0	Salaried	Male	
2917	0	50.0	Self Enquiry	1	25.0	Salaried	Male	
2921	0	51.0	Company Invited	2	10.0	Small Business	Male	
2941	0	45.0	Self Enquiry	1	10.0	Salaried	Male	
2979	0	42.0	Self Enquiry	2	17.0	Salaried	Male	
2982	0	42.0	Self Enquiry	2	7.0	Salaried	Male	
3028	0	43.0	Company Invited	1	15.0	Salaried	Male	
3032	0	51.0	Self Enquiry	1	9.0	Small Business	Male	
3039	1	59.0	Self Enquiry	1	9.0	Salaried	Male	
3053	0	44.0	Self Enquiry	1	21.0	Salaried	Male	
3097	0	51.0	Company Invited	1	9.0	Salaried	Male	
3143	0	53.0	Self Enquiry	1	7.0	Salaried	Male	
3154	0	34.0	Company Invited	3	24.0	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
3158	0	51.0	Self Enquiry	1	7.0	Small Business	Male	
3160	0	42.0	Company Invited	1	16.0	Small Business	Male	
3185	0	43.0	Self Enquiry	3	12.0	Small Business	Male	
3199	0	46.0	Self Enquiry	3	18.0	Salaried	Female	
3210	0	51.0	Self Enquiry	1	9.0	Small Business	Male	
3243	0	43.0	Self Enquiry	1	9.0	Small Business	Male	
3254	0	47.0	Self Enquiry	3	10.0	Small Business	Male	
3302	0	54.0	Self Enquiry	1	14.0	Small Business	Female	
3305	0	47.0	Self Enquiry	3	9.0	Small Business	Female	
3311	0	51.0	Company Invited	1	9.0	Small Business	Female	
3313	0	47.0	Self Enquiry	1	22.0	Salaried	Male	
3338	0	55.0	Self Enquiry	1	10.0	Salaried	Male	
3343	0	50.0	Self Enquiry	1	11.0	Small Business	Male	
3348	0	49.0	Self Enquiry	1	7.0	Salaried	Male	
3351	0	45.0	Self Enquiry	3	12.0	Small Business	Male	
3357	1	46.0	Self Enquiry	3	9.0	Small Business	Female	
3360	0	47.0	Self Enquiry	3	11.0	Small Business	Female	
3366	0	45.0	Self Enquiry	1	11.0	Salaried	Male	
3380	0	46.0	Company Invited	1	32.0	Small Business	Female	
3381	0	40.0	Self Enquiry	1	20.0	Small Business	Female	
3398	0	43.0	Company Invited	1	9.0	Salaried	Male	
3399	0	56.0	Self Enquiry	1	9.0	Small Business	Female	
3452	0	55.0	Self Enquiry	1	7.0	Small Business	Female	
3468	0	48.0	Self Enquiry	1	9.0	Small Business	Female	
3499	0	35.0	Company Invited	1	22.0	Small Business	Male	
3570	0	51.0	Self Enquiry	3	6.0	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
3579	0	47.0	Self Enquiry	3	7.0	Salaried	Male	
3584	0	45.0	Self Enquiry	1	14.0	Small Business	Female	
3628	0	55.0	Self Enquiry	1	29.0	Small Business	Female	
3629	0	44.0	Self Enquiry	1	22.0	Salaried	Male	
3708	0	56.0	Self Enquiry	1	9.0	Small Business	Male	
3721	0	47.0	Self Enquiry	1	9.0	Small Business	Male	
3774	0	44.0	Self Enquiry	1	13.0	Small Business	Female	
3795	0	49.0	Company Invited	1	29.0	Small Business	Female	
3818	0	59.0	Self Enquiry	3	28.0	Salaried	Female	
3821	0	50.0	Company Invited	1	9.0	Salaried	Male	
3881	0	40.0	Company Invited	1	16.0	Salaried	Male	
3887	0	43.0	Self Enquiry	1	9.0	Salaried	Male	
3939	0	54.0	Self Enquiry	1	12.0	Salaried	Female	
3943	0	47.0	Self Enquiry	3	9.0	Small Business	Female	
3976	0	51.0	Self Enquiry	1	14.0	Small Business	Female	
4019	0	60.0	Company Invited	2	9.0	Salaried	Female	
4020	0	51.0	Company Invited	1	7.0	Salaried	Female	
4026	0	55.0	Company Invited	2	33.0	Salaried	Female	
4033	0	44.0	Company Invited	3	33.0	Salaried	Male	
4061	0	42.0	Company Invited	1	17.0	Salaried	Male	
4100	0	41.0	Self Enquiry	1	11.0	Small Business	Female	
4101	0	56.0	Self Enquiry	1	21.0	Small Business	Male	
4145	0	43.0	Self Enquiry	1	11.0	Large Business	Male	
4147	0	51.0	Self Enquiry	3	7.0	Small Business	Female	
4151	0	53.0	Company Invited	3	9.0	Salaried	Male	
4158	0	46.0	Self Enquiry	1	7.0	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
4171	0	41.0	Self Enquiry	1	9.0	Small Business	Male	
4184	0	56.0	Self Enquiry	1	7.0	Small Business	Male	
4193	0	51.0	Self Enquiry	1	11.0	Salaried	Male	
4194	0	54.0	Self Enquiry	1	10.0	Small Business	Male	
4204	0	50.0	Company Invited	1	17.0	Salaried	Female	
4228	0	40.0	Self Enquiry	1	17.0	Small Business	Male	
4240	0	40.0	Company Invited	1	14.0	Small Business	Male	
4243	0	48.0	Self Enquiry	1	12.0	Salaried	Male	
4306	0	55.0	Self Enquiry	1	12.0	Small Business	Male	
4314	0	40.0	Company Invited	1	7.0	Salaried	Male	
4331	0	41.0	Self Enquiry	3	9.0	Salaried	Female	
4339	0	51.0	Self Enquiry	1	36.0	Salaried	Male	
4343	0	47.0	Self Enquiry	1	9.0	Salaried	Male	
4387	0	50.0	Self Enquiry	1	25.0	Salaried	Male	
4391	0	51.0	Company Invited	2	10.0	Small Business	Male	
4411	0	45.0	Self Enquiry	1	10.0	Salaried	Male	
4449	0	42.0	Self Enquiry	2	17.0	Salaried	Male	
4452	0	42.0	Self Enquiry	2	7.0	Salaried	Male	
4498	0	43.0	Company Invited	1	15.0	Salaried	Male	
4502	0	51.0	Self Enquiry	1	9.0	Small Business	Male	
4509	1	59.0	Self Enquiry	1	9.0	Salaried	Male	
4523	0	44.0	Self Enquiry	1	21.0	Salaried	Male	
4567	0	51.0	Company Invited	1	9.0	Salaried	Male	
4613	0	53.0	Self Enquiry	1	7.0	Salaried	Male	
4624	0	34.0	Company Invited	3	24.0	Salaried	Male	
4628	0	51.0	Self Enquiry	1	7.0	Small Business	Male	
4630	0	42.0	Company Invited	1	16.0	Small Business	Male	
4655	0	43.0	Self Enquiry	3	12.0	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
4669	0	46.0	Self Enquiry	3	18.0	Salaried	Female	
4680	0	51.0	Self Enquiry	1	9.0	Small Business	Male	
4713	0	43.0	Self Enquiry	1	9.0	Small Business	Male	
4718	0	49.0	Company Invited	1	7.0	Small Business	Male	
4724	0	47.0	Self Enquiry	3	10.0	Small Business	Male	
4772	0	54.0	Self Enquiry	1	14.0	Small Business	Female	
4775	0	47.0	Self Enquiry	3	9.0	Small Business	Female	
4781	0	51.0	Company Invited	1	9.0	Small Business	Female	
4783	0	47.0	Self Enquiry	1	22.0	Salaried	Male	
4808	0	55.0	Self Enquiry	1	10.0	Salaried	Male	
4813	0	50.0	Self Enquiry	1	11.0	Small Business	Male	
4818	1	49.0	Self Enquiry	1	22.0	Small Business	Female	
4821	1	45.0	Self Enquiry	1	30.0	Small Business	Male	
4827	1	46.0	Self Enquiry	3	20.0	Small Business	Male	
4830	1	47.0	Self Enquiry	1	29.0	Small Business	Male	
4836	1	45.0	Self Enquiry	3	16.0	Salaried	Male	
4850	1	46.0	Self Enquiry	3	8.0	Salaried	Male	
4851	1	40.0	Self Enquiry	1	9.0	Salaried	Female	
4868	1	43.0	Company Invited	2	15.0	Salaried	Female	
4869	1	56.0	Self Enquiry	3	16.0	Small Business	Female	

In [120…
```
# Checking for the mean, considering that we could not find a good pattern on
df.NumberOfTrips.mean()
```

Out[120…
```
3.236520640269587
```

In [121…
```
# We'll impute these missing values one by one, by taking mean of NumberOfTri
df.groupby(["Designation"])["NumberOfTrips"].mean().round(0)
```

Out[121…
```
Designation
AVP                4.0
Executive          3.0
```

```
Manager           3.0
Senior Manager    3.0
VP                3.0
Name: NumberOfTrips, dtype: float64
```

In [122…
```
# Impute missing values of NumberOfTrips
df["NumberOfTrips"] = df.groupby(["Designation"])["NumberOfTrips"].transform(
    lambda x: round(x.fillna(x.mean()))
)
```

In [123…
```
df[df["NumberOfTrips"].isnull()]
```

Out[123…

ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPe

Number Of Children Visiting

In [124…
```
# Cheeking the rows with NumberOfChildrenVisiting missing
df[df["NumberOfChildrenVisiting"].isnull()]
```

Out[124…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
165	0	50.0	Self Enquiry	1	17.0	Salaried	Female	
190	0	52.0	Self Enquiry	1	6.0	Salaried	Male	
568	1	55.0	Self Enquiry	1	8.0	Small Business	Male	
746	0	41.0	Company Invited	1	13.0	Salaried	Female	
749	1	52.0	Self Enquiry	3	8.0	Small Business	Female	
851	0	56.0	Self Enquiry	1	10.0	Large Business	Female	
898	0	43.0	Self Enquiry	1	9.0	Salaried	Male	
918	0	51.0	Company Invited	3	15.0	Salaried	Male	
956	0	56.0	Self Enquiry	2	14.0	Salaried	Male	
1009	0	58.0	Self Enquiry	1	6.0	Small Business	Female	
1154	0	47.0	Self Enquiry	2	32.0	Salaried	Female	
1242	0	40.0	Self Enquiry	3	13.0	Small Business	Male	
1331	0	48.0	Self Enquiry	1	16.0	Salaried	Male	
1401	0	55.0	Self Enquiry	2	32.0	Salaried	Male	
1635	0	50.0	Self Enquiry	1	17.0	Salaried	Female	
1660	0	52.0	Self Enquiry	1	6.0	Salaried	Male	
2038	1	55.0	Self Enquiry	1	8.0	Small Business	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2216	0	41.0	Company Invited	1	13.0	Salaried	Female	
2219	1	52.0	Self Enquiry	3	8.0	Small Business	Female	
2321	0	56.0	Self Enquiry	1	10.0	Large Business	Female	
2368	0	43.0	Self Enquiry	1	9.0	Salaried	Male	
2388	1	51.0	Company Invited	1	34.0	Salaried	Male	
2426	1	56.0	Self Enquiry	3	22.0	Salaried	Female	
2638	0	46.0	Company Invited	1	9.0	Small Business	Male	
2679	0	44.0	Self Enquiry	3	23.0	Small Business	Female	
2707	0	47.0	Self Enquiry	3	9.0	Large Business	Female	
2744	0	42.0	Self Enquiry	3	9.0	Salaried	Male	
2792	0	43.0	Self Enquiry	1	30.0	Salaried	Female	
2823	0	56.0	Self Enquiry	1	9.0	Salaried	Female	
2852	0	53.0	Self Enquiry	1	11.0	Salaried	Female	
2889	0	56.0	Self Enquiry	3	25.0	Salaried	Female	
2899	0	34.0	Self Enquiry	1	7.0	Small Business	Female	
2910	0	42.0	Self Enquiry	3	9.0	Salaried	Female	
2933	0	44.0	Self Enquiry	1	13.0	Salaried	Male	
3005	0	53.0	Self Enquiry	3	10.0	Small Business	Male	
3036	0	48.0	Self Enquiry	1	9.0	Salaried	Female	
3060	0	52.0	Self Enquiry	3	33.0	Small Business	Female	
3218	0	56.0	Company Invited	1	9.0	Small Business	Male	
3349	0	30.0	Self Enquiry	1	7.0	Salaried	Female	
3389	0	51.0	Self Enquiry	1	35.0	Salaried	Female	
3443	0	43.0	Self Enquiry	2	17.0	Salaried	Female	
3458	0	32.0	Self Enquiry	1	15.0	Salaried	Female	
3487	0	54.0	Self Enquiry	1	9.0	Small Business	Male	
3520	0	55.0	Company Invited	1	18.0	Small Business	Female	
3522	0	45.0	Self Enquiry	1	35.0	Salaried	Male	
3524	0	47.0	Self Enquiry	3	10.0	Salaried	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
3540	0	41.0	Self Enquiry	2	13.0	Small Business	Male	
3620	0	50.0	Self Enquiry	1	29.0	Salaried	Female	
3638	0	48.0	Self Enquiry	3	9.0	Salaried	Female	
3669	0	46.0	Self Enquiry	1	35.0	Large Business	Female	
3792	0	41.0	Self Enquiry	1	7.0	Salaried	Male	
4108	0	46.0	Company Invited	1	9.0	Small Business	Male	
4149	0	44.0	Self Enquiry	3	23.0	Small Business	Female	
4214	0	42.0	Self Enquiry	3	9.0	Salaried	Male	
4262	0	43.0	Self Enquiry	1	30.0	Salaried	Female	
4293	0	56.0	Self Enquiry	1	9.0	Salaried	Female	
4322	0	53.0	Self Enquiry	1	11.0	Salaried	Female	
4359	0	56.0	Self Enquiry	3	25.0	Salaried	Female	
4369	0	34.0	Self Enquiry	1	7.0	Small Business	Female	
4380	0	42.0	Self Enquiry	3	9.0	Salaried	Female	
4403	0	44.0	Self Enquiry	1	13.0	Salaried	Male	
4475	0	53.0	Self Enquiry	3	10.0	Small Business	Male	
4506	0	48.0	Self Enquiry	1	9.0	Salaried	Female	
4530	0	52.0	Self Enquiry	3	33.0	Small Business	Female	
4688	0	56.0	Company Invited	1	9.0	Small Business	Male	
4819	1	30.0	Self Enquiry	1	14.0	Large Business	Female	

In [125…
```
df.NumberOfChildrenVisiting.mean()
```

Out[125…
```
1.1872666943177106
```

In [126…
```
# We'll impute these missing values one by one, by taking mean of NumberOfChi
df.groupby(["NumberOfPersonVisiting", "MaritalStatus"])[
    "NumberOfChildrenVisiting"
].mean().round(0)
```

Out[126…
```
NumberOfPersonVisiting  MaritalStatus
1                       Divorced         0.0
                        Married          0.0
                        Single           0.0
                        Unmarried        0.0
2                       Divorced         0.0
                        Married          1.0
```

```
                        Single           0.0
                        Unmarried        1.0
3                       Divorced         1.0
                        Married          1.0
                        Single           1.0
                        Unmarried        1.0
4                       Divorced         2.0
                        Married          2.0
                        Single           2.0
                        Unmarried        2.0
5                       Divorced         NaN
                        Married          2.0
                        Single           2.0
                        Unmarried        2.0
Name: NumberOfChildrenVisiting, dtype: float64
```

In [127…
```
# Impute missing values of NumberOfChildrenVisiting
df["NumberOfChildrenVisiting"] = df.groupby(
    ["NumberOfPersonVisiting", "MaritalStatus"]
)["NumberOfChildrenVisiting"].transform(lambda x: round(x.fillna(x.mean())))
```

In [128…
```
df[df["NumberOfChildrenVisiting"].isnull()]
```

Out[128…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPe

Number Of Followups

In [129…
```
# Cheeking the rows with NumberOfFollowups missing
df[df["NumberOfFollowups"].isnull()]
```

Out[129…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
79	0	46.0	Self Enquiry	2	11.0	Small Business	Male	
94	0	32.0	Self Enquiry	3	12.0	Small Business	Male	
96	0	24.0	Self Enquiry	3	9.0	Salaried	Female	
122	1	56.0	Self Enquiry	1	20.0	Salaried	Female	
135	0	36.0	Self Enquiry	1	12.0	Small Business	Male	
174	0	45.0	Self Enquiry	3	10.0	Salaried	Female	
317	1	52.0	Self Enquiry	1	14.0	Small Business	Male	
322	0	32.0	Self Enquiry	1	8.0	Small Business	Female	
376	0	51.0	Self Enquiry	3	20.0	Salaried	Female	
532	0	47.0	Self Enquiry	3	20.0	Small Business	Male	
629	0	28.0	Self Enquiry	2	14.0	Small Business	Male	
737	0	41.0	Self Enquiry	1	13.0	Small Business	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
748	1	26.0	Company Invited	3	35.0	Small Business	Male	
820	0	35.0	Company Invited	3	17.0	Small Business	Male	
881	0	32.0	Company Invited	1	8.0	Salaried	Female	
885	0	25.0	Self Enquiry	3	16.0	Salaried	Male	
1159	0	39.0	Company Invited	1	10.0	Small Business	Female	
1215	0	35.0	Company Invited	1	8.0	Small Business	Male	
1234	0	47.0	Company Invited	3	8.0	Small Business	Male	
1244	0	30.0	Company Invited	1	8.0	Large Business	Female	
1352	0	44.0	Self Enquiry	1	6.0	Salaried	Male	
1389	0	31.0	Company Invited	1	6.0	Salaried	Male	
1549	0	46.0	Self Enquiry	2	11.0	Small Business	Male	
1564	0	32.0	Self Enquiry	3	12.0	Small Business	Male	
1566	0	24.0	Self Enquiry	3	9.0	Salaried	Female	
1592	1	56.0	Self Enquiry	1	20.0	Salaried	Female	
1605	0	36.0	Self Enquiry	1	12.0	Small Business	Male	
1644	0	45.0	Self Enquiry	3	10.0	Salaried	Female	
1787	1	52.0	Self Enquiry	1	14.0	Small Business	Male	
1846	0	51.0	Self Enquiry	3	20.0	Salaried	Female	
2002	0	47.0	Self Enquiry	3	20.0	Small Business	Male	
2099	0	28.0	Self Enquiry	2	14.0	Small Business	Male	
2129	0	28.0	Self Enquiry	3	11.0	Small Business	Male	
2207	0	41.0	Self Enquiry	1	13.0	Small Business	Female	
2218	1	26.0	Company Invited	3	35.0	Small Business	Male	
2290	0	35.0	Company Invited	3	17.0	Small Business	Male	
2351	0	32.0	Company Invited	1	8.0	Salaried	Female	
2355	0	25.0	Self Enquiry	3	16.0	Salaried	Male	
2467	0	22.0	Self Enquiry	1	22.0	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
2959	0	36.0	Company Invited	1	10.0	Salaried	Male	
3456	1	32.0	Company Invited	3	7.0	Salaried	Female	
3460	1	32.0	Self Enquiry	1	15.0	Salaried	Female	
3496	0	31.0	Company Invited	1	14.0	Large Business	Male	
3937	0	22.0	Self Enquiry	1	22.0	Salaried	Male	
4429	0	36.0	Company Invited	1	10.0	Salaried	Male	

In [130…
```
# We'll impute these missing values one by one, by taking mean of NumberOfFol
df.groupby(["Designation", "DurationOfPitch"])["NumberOfFollowups"].mean().ro
```

Out[130…
```
Designation     DurationOfPitch
AVP             5.0                NaN
                6.0                3.0
                7.0                4.0
                8.0                3.0
                9.0                4.0
                10.0               4.0
                11.0               4.0
                12.0               3.0
                13.0               4.0
                14.0               3.0
                15.0               4.0
                16.0               4.0
                17.0               4.0
                18.0               4.0
                19.0               3.0
                20.0               3.0
                21.0               4.0
                22.0               4.0
                23.0               4.0
                24.0               4.0
                25.0               4.0
                26.0               3.0
                27.0               4.0
                28.0               4.0
                29.0               4.0
                30.0               3.0
                31.0               4.0
                32.0               3.0
                33.0               4.0
                34.0               2.0
                35.0               4.0
                36.0               5.0
Executive       5.0                4.0
                6.0                3.0
                7.0                4.0
                8.0                3.0
                9.0                4.0
                10.0               4.0
                11.0               3.0
                12.0               4.0
                13.0               4.0
                14.0               3.0
                15.0               4.0
                16.0               4.0
```

	17.0	4.0
	18.0	3.0
	19.0	4.0
	20.0	4.0
	21.0	4.0
	22.0	3.0
	23.0	4.0
	24.0	4.0
	25.0	3.0
	26.0	3.0
	27.0	3.0
	28.0	4.0
	29.0	4.0
	30.0	4.0
	31.0	4.0
	32.0	4.0
	33.0	4.0
	34.0	3.0
	35.0	3.0
	36.0	4.0
Manager	5.0	3.0
	6.0	3.0
	7.0	4.0
	8.0	3.0
	9.0	4.0
	10.0	4.0
	11.0	4.0
	12.0	4.0
	13.0	4.0
	14.0	4.0
	15.0	4.0
	16.0	4.0
	17.0	4.0
	18.0	4.0
	19.0	4.0
	20.0	4.0
	21.0	4.0
	22.0	4.0
	23.0	4.0
	24.0	3.0
	25.0	4.0
	26.0	3.0
	27.0	4.0
	28.0	4.0
	29.0	4.0
	30.0	4.0
	31.0	4.0
	32.0	4.0
	33.0	4.0
	34.0	4.0
	35.0	4.0
	36.0	5.0
Senior Manager	5.0	3.0
	6.0	3.0
	7.0	4.0
	8.0	3.0
	9.0	4.0
	10.0	4.0
	11.0	4.0
	12.0	4.0
	13.0	4.0
	14.0	4.0
	15.0	4.0
	16.0	4.0
	17.0	4.0
	18.0	4.0
	19.0	4.0
	20.0	3.0
	21.0	4.0

```
                    22.0                4.0
                    23.0                4.0
                    24.0                4.0
                    25.0                4.0
                    26.0                4.0
                    27.0                4.0
                    28.0                2.0
                    29.0                3.0
                    30.0                4.0
                    31.0                4.0
                    32.0                4.0
                    33.0                3.0
                    34.0                4.0
                    35.0                4.0
                    36.0                4.0
VP                  5.0                 4.0
                    6.0                 3.0
                    7.0                 4.0
                    8.0                 4.0
                    9.0                 4.0
                    10.0                4.0
                    11.0                4.0
                    12.0                4.0
                    13.0                4.0
                    14.0                4.0
                    15.0                4.0
                    16.0                4.0
                    17.0                4.0
                    18.0                4.0
                    19.0                4.0
                    20.0                3.0
                    21.0                4.0
                    22.0                4.0
                    23.0                NaN
                    24.0                4.0
                    25.0                5.0
                    26.0                NaN
                    27.0                NaN
                    28.0                NaN
                    29.0                NaN
                    30.0                NaN
                    31.0                3.0
                    32.0                3.0
                    33.0                4.0
                    34.0                NaN
                    35.0                NaN
                    36.0                NaN
Name: NumberOfFollowups, dtype: float64
```

In [131…

```
# Impute missing values of NumberOfFollowups
df["NumberOfFollowups"] = df.groupby(["Designation", "DurationOfPitch"])[
    "NumberOfFollowups"
].transform(lambda x: round(x.fillna(x.mean())))
```

In [132…

```
df[df["NumberOfFollowups"].isnull()]
```

Out[132…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPe

Prefer Property Star

In [133…
```
# Cheeking the rows with PreferredPropertyStar missing
df[df["PreferredPropertyStar"].isnull()]
```

Out[133…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
38	0	36.0	Self Enquiry	1	11.0	Salaried	Female	
2609	0	51.0	Self Enquiry	1	18.0	Salaried	Female	
2634	0	53.0	Self Enquiry	1	7.0	Salaried	Male	
3012	1	56.0	Self Enquiry	1	9.0	Small Business	Male	
3190	0	42.0	Company Invited	1	14.0	Salaried	Female	
3193	1	53.0	Self Enquiry	3	9.0	Small Business	Female	
3214	0	47.0	Self Enquiry	1	7.0	Small Business	Male	
3295	0	57.0	Self Enquiry	1	11.0	Large Business	Female	
3342	0	44.0	Self Enquiry	1	10.0	Salaried	Male	
3362	0	52.0	Company Invited	3	16.0	Salaried	Male	
3400	0	57.0	Self Enquiry	2	15.0	Salaried	Male	
3453	0	59.0	Self Enquiry	1	7.0	Small Business	Female	
3598	0	48.0	Self Enquiry	2	33.0	Salaried	Female	
3686	0	41.0	Self Enquiry	3	14.0	Small Business	Male	
3775	0	49.0	Self Enquiry	1	17.0	Salaried	Male	
3845	0	56.0	Self Enquiry	2	33.0	Salaried	Male	
4079	0	51.0	Self Enquiry	1	18.0	Salaried	Female	
4104	0	53.0	Self Enquiry	1	7.0	Salaried	Male	
4482	1	56.0	Self Enquiry	1	9.0	Small Business	Male	
4660	0	42.0	Company Invited	1	14.0	Salaried	Female	
4663	1	53.0	Self Enquiry	3	9.0	Small Business	Female	
4684	0	47.0	Self Enquiry	1	7.0	Small Business	Male	
4765	0	57.0	Self Enquiry	1	11.0	Large Business	Female	
4812	0	44.0	Self Enquiry	1	10.0	Salaried	Male	
4832	1	52.0	Company Invited	1	35.0	Salaried	Male	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
4870	1	57.0	Self Enquiry	3	23.0	Salaried	Female	

In [134…
```
# We'll impute these missing values one by one, by taking mean of PreferredPr
df.groupby(["Designation", "ProductPitched"])["PreferredPropertyStar"].mean()
```

Out[134…
```
Designation     ProductPitched
AVP             Basic             NaN
                Deluxe            NaN
                King              NaN
                Standard          NaN
                Super Deluxe      4.0
Executive       Basic             4.0
                Deluxe            NaN
                King              NaN
                Standard          NaN
                Super Deluxe      NaN
Manager         Basic             NaN
                Deluxe            4.0
                King              NaN
                Standard          NaN
                Super Deluxe      NaN
Senior Manager  Basic             NaN
                Deluxe            NaN
                King              NaN
                Standard          4.0
                Super Deluxe      NaN
VP              Basic             NaN
                Deluxe            NaN
                King              3.0
                Standard          NaN
                Super Deluxe      NaN
Name: PreferredPropertyStar, dtype: float64
```

In [135…
```
# Impute missing values of PreferredPropertyStar
df["PreferredPropertyStar"] = df.groupby(["Designation", "ProductPitched"])[
    "PreferredPropertyStar"
].transform(lambda x: round(x.fillna(x.mean())))
```

In [136…
```
df[df["PreferredPropertyStar"].isnull()]
```

Out[136…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPe

Type Of Contact

In [137…
```
# Cheeking the rows with TypeofContact missing
df[df["TypeofContact"].isnull()]
```

Out[137…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
224	0	31.0	NaN	1	17.0	Small Business	Male	
571	0	26.0	NaN	1	15.0	Salaried	Female	

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Numbe
572	0	29.0	NaN	1	17.0	Small Business	Female	
576	0	27.0	NaN	3	17.0	Small Business	Male	
579	0	34.0	NaN	1	16.0	Small Business	Female	
598	1	28.0	NaN	1	16.0	Small Business	Male	
622	0	32.0	NaN	3	15.0	Salaried	Male	
724	0	24.0	NaN	1	17.0	Small Business	Female	
843	0	26.0	NaN	1	16.0	Small Business	Male	
1021	1	25.0	NaN	3	15.0	Salaried	Male	
1047	0	33.0	NaN	3	17.0	Small Business	Male	
1143	0	45.0	NaN	3	17.0	Small Business	Male	
1182	0	36.0	NaN	1	17.0	Small Business	Female	
1217	0	24.0	NaN	1	16.0	Small Business	Male	
1356	0	41.0	NaN	3	17.0	Small Business	Female	
1469	0	34.0	NaN	1	17.0	Small Business	Male	
1694	0	31.0	NaN	1	17.0	Small Business	Male	
2041	0	26.0	NaN	1	15.0	Salaried	Female	
2042	0	29.0	NaN	1	17.0	Small Business	Female	
2046	0	27.0	NaN	3	17.0	Small Business	Male	
2049	0	34.0	NaN	1	16.0	Small Business	Female	
2068	1	28.0	NaN	1	16.0	Small Business	Male	
2092	0	32.0	NaN	3	15.0	Salaried	Male	
2194	0	24.0	NaN	1	17.0	Small Business	Female	
2313	0	26.0	NaN	1	16.0	Small Business	Male	

In [138…

```
df.TypeofContact.value_counts() / len(df.TypeofContact)
```

Out[138…
```
Self Enquiry       0.704583
Company Invited    0.290303
Name: TypeofContact, dtype: float64
```

As checked on EDA, TypeOfContact has no pattern with others variables. Considering that 70.5% of customer Self Enquiry and 29.5% Company Invited, we gonna consider that this 25 customer (0.5% of dataset) belongs to Self Enquiry.

In [139…
```
# replacing na values in TypeofContact with Self Enquiry
df["TypeofContact"].fillna("Self Enquiry", inplace=True)
```

In [140…
```
df[df["TypeofContact"].isnull()]
```

Out[140…

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPe

In [141…
```
# Checking if all missing data were fill it up
df.isnull().sum().sort_values(ascending=False)
```

Out[141…
```
MonthlyIncome               0
NumberOfFollowups           0
Age                         0
TypeofContact               0
CityTier                    0
DurationOfPitch             0
Occupation                  0
Gender                      0
NumberOfPersonVisiting      0
ProductPitched              0
Designation                 0
PreferredPropertyStar       0
MaritalStatus               0
NumberOfTrips               0
Passport                    0
PitchSatisfactionScore      0
OwnCar                      0
NumberOfChildrenVisiting    0
ProdTaken                   0
dtype: int64
```

Let's find the percentage of outliers, in each column of the data, using IQR

In [142…
```
# To find the 25th percentile and 75th percentile.
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)

# Inter Quantile Range (75th perentile - 25th percentile)
IQR = Q3 - Q1


# Finding lower and upper bounds for all values. All values outside these bou
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR
```

In [143…
```
(
    (df.select_dtypes(include=["float64", "int64"]) < lower)
```

```
    | (df.select_dtypes(include=["float64", "int64"]) > upper)
).sum() / len(df) * 100
```

Out[143…

```
ProdTaken                 18.821604
Age                        0.000000
CityTier                   0.000000
DurationOfPitch            2.250409
NumberOfPersonVisiting     0.061375
NumberOfFollowups          6.382979
PreferredPropertyStar      0.000000
NumberOfTrips              2.229951
Passport                   0.000000
PitchSatisfactionScore     0.000000
OwnCar                     0.000000
NumberOfChildrenVisiting   0.000000
MonthlyIncome              7.221768
dtype: float64
```

- We analyzed the extrem values and replaced the ones that was error like Low and Higher Incomes,Duration of Pitch.
- NumberOfTrips does seems to be a real data, so we not gonna treat as outliers.

> We are not going to treat them as there will be outliers and we would want our model to learn the underlying pattern for such customers.

Building the Model

1. Data preparation
2. Partition the data into train and test set.
3. Build model on the train data.
4. Tune the model if required.
5. Test the data on test set.

In [144…

```
# Data Preparation:
X = df.drop(
    ["ProdTaken"], axis=1
)  # Creating x with all independent variable, removing target variable
X = pd.get_dummies(
    X, drop_first=True
)  # All category variables we creat dummies and drop first.
y = df["ProdTaken"]
```

In [145…

```
# Partition the data into train and test set:
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y
)  # using stratify because of imbalance distribution of the target classes
print(f"X_Train: {X_train.shape},\nX_test: {X_test.shape}")
```

```
X_Train: (3421, 28),
X_test: (1467, 28)
```

In [146…

```
# Checking the % distribution of the target classes
y.value_counts(1)
```

Out[146…
```
0    0.811784
1    0.188216
Name: ProdTaken, dtype: float64
```

In [147…
```
y_train.value_counts(1)
```

Out[147…
```
0    0.811751
1    0.188249
Name: ProdTaken, dtype: float64
```

In [148…
```
y_test.value_counts(1)
```

Out[148…
```
0    0.811861
1    0.188139
Name: ProdTaken, dtype: float64
```

> We can see that train and test kept the proportion of the target classes (0 ~ 81% ; 1 ~ 19%)

Model evaluation criterion

Model can make wrong predictions as:

1. Predicting an customer will buy and the customer doesn't buy the travel package.
2. Predicting an customer will NOT buy and the customer buy it.

Which case is more important?

- Predicting that customer will not buy but it would i.e. losing a potential buyer.

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

- Company wants Recall to be maximized, greater the Recall higher the chances of minimizing false negatives. Hence, the focus should be on increasing Recall or minimizing the false negatives or in other words identifying the true positives(i.e. Class 1) so that the company can ofer the new package for the right customers thereby optimizing the overall project cost in convert the customers as buyers .

DECISION TREE BAGGING AND RANDOM FOREST

Let's define function to provide metric scores(accuracy,recall and precision) on train and test set and a function to show confusion matrix so that we do not have use the same code repetitively while evaluating models.

In [149…
```
##  Function to calculate different metric scores of the model - Accuracy, Re
def get_metrics_score(model, flag=True):
    """
    model : classifier to predict values of X

    """
    # defining an empty list to store train and test results
    score_list = []
```

```
    # Predicting on train and tests
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    # Accuracy of the model
    train_acc = model.score(X_train, y_train)
    test_acc = model.score(X_test, y_test)

    # Recall of the model
    train_recall = metrics.recall_score(y_train, pred_train)
    test_recall = metrics.recall_score(y_test, pred_test)

    # Precision of the model
    train_precision = metrics.precision_score(y_train, pred_train)
    test_precision = metrics.precision_score(y_test, pred_test)

    # F1-score of the model
    train_F1 = metrics.f1_score(y_train, pred_train)
    test_F1 = metrics.f1_score(y_test, pred_test)

    score_list.extend(
        (
            train_acc,
            test_acc,
            train_recall,
            test_recall,
            train_precision,
            test_precision,
            train_F1,
            test_F1,
        )
    )

    # If the flag is set to True then only the following print statements wil
    if flag == True:
        print("Accuracy on training set : {:.3f}".format(model.score(X_train,
        print("Accuracy on test set : {:.3f}".format(model.score(X_test, y_te
        print(
            "Recall on training set : {:.3f}".format(
                metrics.recall_score(y_train, pred_train)
            )
        )
        print(
            "Recall on test set : {:.3f}".format(
                metrics.recall_score(y_test, pred_test)
            )
        )
        print(
            "Precision on training set : {:.3f}".format(
                metrics.precision_score(y_train, pred_train)
            )
        )
        print(
            "Precision on test set : {:.3f}".format(
                metrics.precision_score(y_test, pred_test)
            )
        )
        print(
            "F1_score on training set : {:.3f}".format(
                metrics.f1_score(y_train, pred_train)
            )
        )
        print(
            "F1_score on test set : {:.3f}".format(metrics.f1_score(y_test, p
```

```
        )

    return score_list  # returning the list with train and test scores
```

In [150…
```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten()
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Decision Tree Model

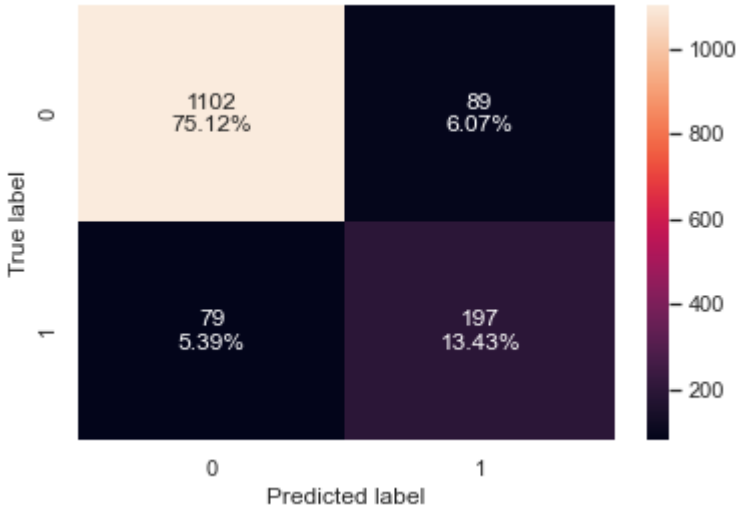
- We will build our model using the DecisionTreeClassifier function. Using default 'gini' criteria to split.
- Considering that our data is imbalanced, class 0 will become the dominant class and the decision tree will become biased toward the dominant classes. So we gonna pass a dictionary {0:0.19,1:0.81} to the model to specify the weight of each class and the decision tree will give more weightage to class 1.
- class_weight is a hyperparameter for the decision tree classifier.

In [151…
```
dtree = DecisionTreeClassifier(
    criterion="gini", class_weight={0: 0.19, 1: 0.81}, random_state=1
)
```

In [152…
```
dtree.fit(X_train, y_train)
```

Out[152…
```
DecisionTreeClassifier(class_weight={0: 0.19, 1: 0.81}, random_state=1)
```

In [153…
```
confusion_matrix_sklearn(dtree, X_test, y_test)
```



In [154…
```
dtree_model_perf = get_metrics_score(dtree, True)
```

```
Accuracy on training set : 1.000
Accuracy on test set : 0.885
Recall on training set : 1.000
Recall on test set : 0.714
Precision on training set : 1.000
Precision on test set : 0.689
F1_score on training set : 1.000
F1_score on test set : 0.701
```

- Decision tree is working well on the `training data` , but not on `test data` . We can see that our Decision Tree is ***overfitting***.

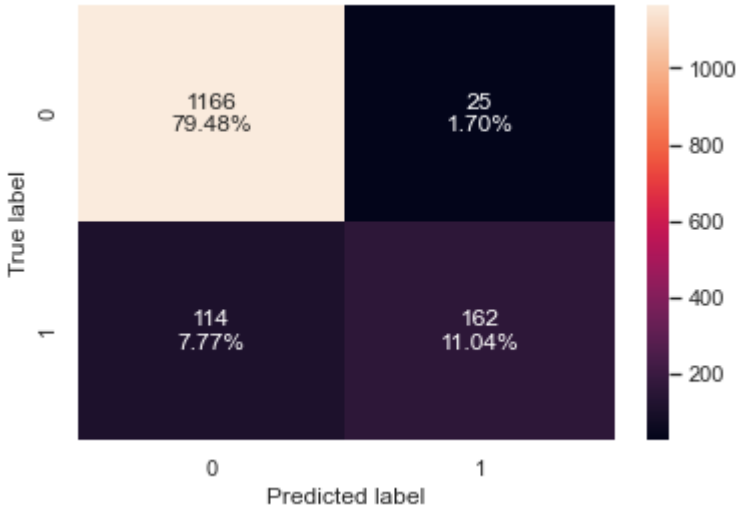
Bagging Classifier Model

Bagging Classifier with *default* decision tree

In [155…
```
# Building our model with default bagging classifier
bagging = BaggingClassifier(random_state=1)

# Fitting our model on our training dataset
bagging.fit(X_train, y_train)
```

Out[155… `BaggingClassifier(random_state=1)`

In [156…
```
confusion_matrix_sklearn(bagging, X_test, y_test)
```



In [157…
```
bagging_model_perf = get_metrics_score(bagging)
```

```
Accuracy on training set : 0.994
Accuracy on test set : 0.905
Recall on training set : 0.969
Recall on test set : 0.587
Precision on training set : 0.998
Precision on test set : 0.866
F1_score on training set : 0.983
F1_score on test set : 0.700
```

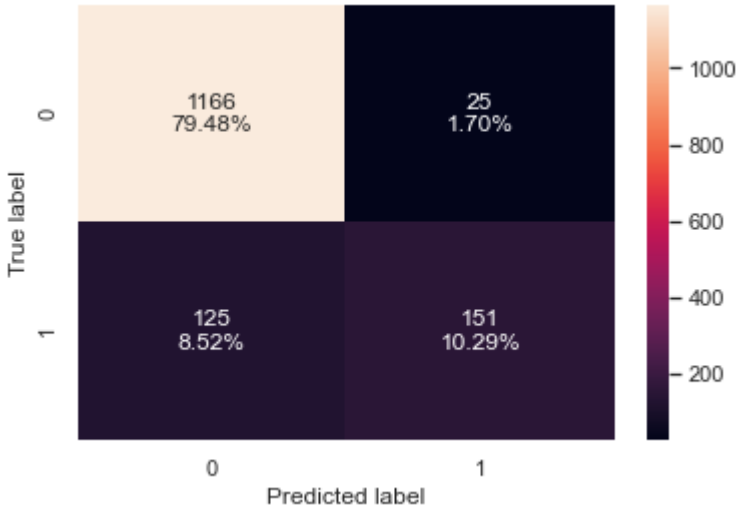
- Bagging Classifier is ***overfitting*** on the `training data` and, is ***performing poorly*** on the `test data` mainly in the recall metric.

Bagging Classifier with *weighted* and decision tree base estimator

In [158…
```
bagging_wt = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(
        criterion="gini", class_weight={0: 0.19, 1: 0.81}, random_state=1
    ),
    random_state=1,
)
bagging_wt.fit(X_train, y_train)
```

Out[158…
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight={0: 0.1
9,
                                                                       1: 0.8
1},
                                                        random_state=1),
                  random_state=1)
```

In [159…
```
confusion_matrix_sklearn(bagging_wt, X_test, y_test)
```



In [160…
```
bagging_wt_model_perf = get_metrics_score(bagging_wt)
```

```
Accuracy on training set : 0.994
Accuracy on test set : 0.898
Recall on training set : 0.967
Recall on test set : 0.547
Precision on training set : 0.998
Precision on test set : 0.858
F1_score on training set : 0.983
F1_score on test set : 0.668
```

- Bagging Classifier with weighted decision tree performance is not much different from bagging classifier default performance.
- It still ***overfitting*** on the `training data` and, is ***performing poorly*** on the `test data` mainly in the recall metric.

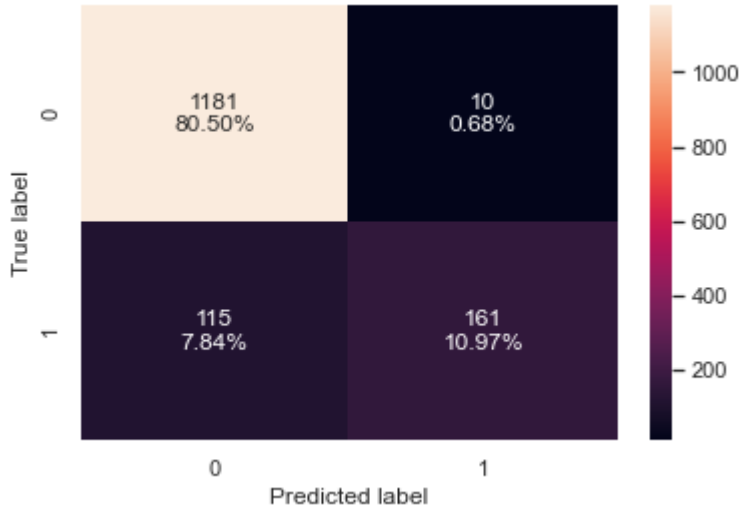
Random Forest Model

Random Forest with *default* decision tree

In [161…
```
rf = RandomForestClassifier(random_state=1)
rf.fit(X_train, y_train)
```

Out[161…
```
RandomForestClassifier(random_state=1)
```

In [162…
```
confusion_matrix_sklearn(rf, X_test, y_test)
```



In [163…
```
rf_model_perf = get_metrics_score(rf)
```

```
Accuracy on training set : 1.000
Accuracy on test set : 0.915
Recall on training set : 1.000
Recall on test set : 0.583
Precision on training set : 1.000
Precision on test set : 0.942
F1_score on training set : 1.000
F1_score on test set : 0.720
```

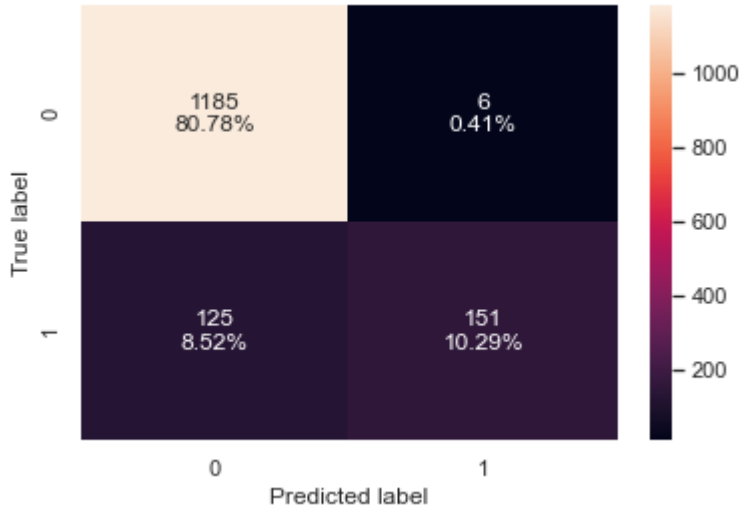
- Random Forest has performed well in terms of `accuracy and precision` , but it is ***not able*** to generalize well on the `test data` in terms of recall.
- This model still ***overfitting***.

Random Forest with *class weights* decision tree

In [164…
```
rf_wt = RandomForestClassifier(class_weight={0: 0.19, 1: 0.81}, random_state=
rf_wt.fit(X_train, y_train)
```

Out[164…
```
RandomForestClassifier(class_weight={0: 0.19, 1: 0.81}, random_state=1)
```

In [165…
```
confusion_matrix_sklearn(rf_wt, X_test, y_test)
```



In [166…
```
rf_wt_model_perf = get_metrics_score(rf_wt)
```

```
Accuracy on training set : 1.000
Accuracy on test set : 0.911
Recall on training set : 1.000
Recall on test set : 0.547
Precision on training set : 1.000
Precision on test set : 0.962
F1_score on training set : 1.000
F1_score on test set : 0.697
```

- Only improvement was on `Precision` compared with Random Forest default, hence there is not much improvement in metrics of weighted random forest as compared to the unweighted random forest.
- Model still ***overfitting*** and performing badly.

Tuning Models

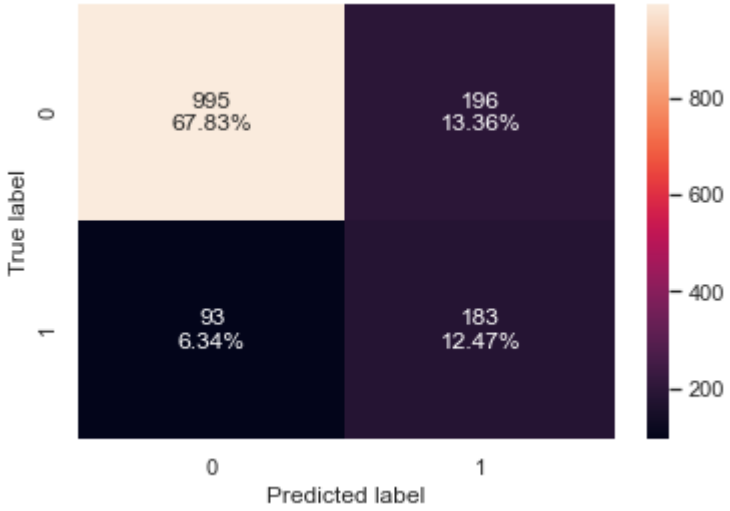
Using GridSearch for Hyperparameter tuning model

Tuning Decision Tree

In [167…
```
# Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(
    class_weight={0: 0.19, 1: 0.81}, random_state=1
)

# Grid of parameters to choose from
parameters = {
    "max_depth": np.arange(2, 30),
    "min_samples_leaf": [1, 2, 5, 7, 10],
    "max_leaf_nodes": [2, 3, 5, 10, 15],
    "min_impurity_decrease": [0.0001, 0.001, 0.01, 0.1],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(dtree_estimator, parameters, scoring=scorer)
grid_obj = grid_obj.fit(X_train, y_train)

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

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

Out[167…
```
DecisionTreeClassifier(class_weight={0: 0.19, 1: 0.81}, max_depth=7,
                       max_leaf_nodes=15, min_impurity_decrease=0.0001,
                       min_samples_leaf=10, random_state=1)
```

In [168…
```
confusion_matrix_sklearn(dtree_estimator, X_test, y_test)
```



In [169…
```
dtree_estimator_model_perf = get_metrics_score(dtree_estimator)
```

```
Accuracy on training set : 0.796
Accuracy on test set : 0.803
Recall on training set : 0.648
Recall on test set : 0.663
Precision on training set : 0.469
Precision on test set : 0.483
F1_score on training set : 0.544
F1_score on test set : 0.559
```

- Overfitting in decision tree has reduced and all metrics is performing poorly.
- Recall has also reduced a litte bit compared to the Decision Tree default 0.71 to 0.66 on test.
- So far this model is with no overfitting but ***poorly performing***, we gonna try tunning Bagging and Random Forest to check if there are improvements.

Tuning Bagging Classifier

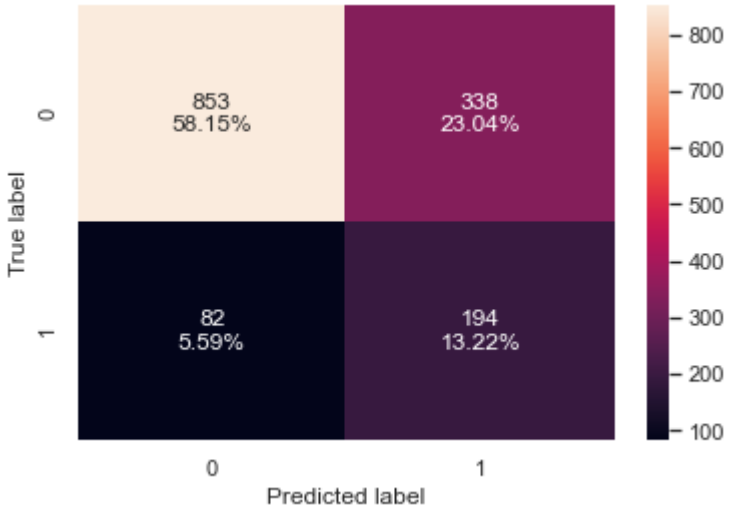
In [170…
```
# grid search for bagging classifier
cl1 = DecisionTreeClassifier(class_weight={0: 0.19, 1: 0.81}, random_state=1)
param_grid = {
    "base_estimator": [cl1],
    "n_estimators": [5, 7, 15, 51, 101],
    "max_features": [0.7, 0.8, 0.9, 1],
}

grid = GridSearchCV(
    BaggingClassifier(random_state=1, bootstrap=True),
    param_grid=param_grid,
    scoring="recall",
    cv=5,
)
grid.fit(X_train, y_train)
```

Out[170…
```
GridSearchCV(cv=5, estimator=BaggingClassifier(random_state=1),
             param_grid={'base_estimator': [DecisionTreeClassifier(class_weig
ht={0: 0.19,

1: 0.81},
                                                                   random_sta

te=1)],
                         'max_features': [0.7, 0.8, 0.9, 1],
                         'n_estimators': [5, 7, 15, 51, 101]},
             scoring='recall')
```

In [171…
```
## getting the best estimator
bagging_estimator = grid.best_estimator_
bagging_estimator.fit(X_train, y_train)
```

Out[171…
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight={0: 0.1
9,
                                                        1: 0.8
1},
                                                        random_state=1),
                  max_features=1, n_estimators=15, random_state=1)
```

In [172…
```
confusion_matrix_sklearn(bagging_estimator, X_test, y_test)
```



In [173…
```
bagging_estimator_model_perf = get_metrics_score(bagging_estimator)
```

```
Accuracy on training set : 0.716
Accuracy on test set : 0.714
Recall on training set : 0.666
Recall on test set : 0.703
Precision on training set : 0.362
Precision on test set : 0.365
F1_score on training set : 0.469
F1_score on test set : 0.480
```

- Recall is a litte bit better but all others metrics dropped drastically. ***This model is making many mistakes***.

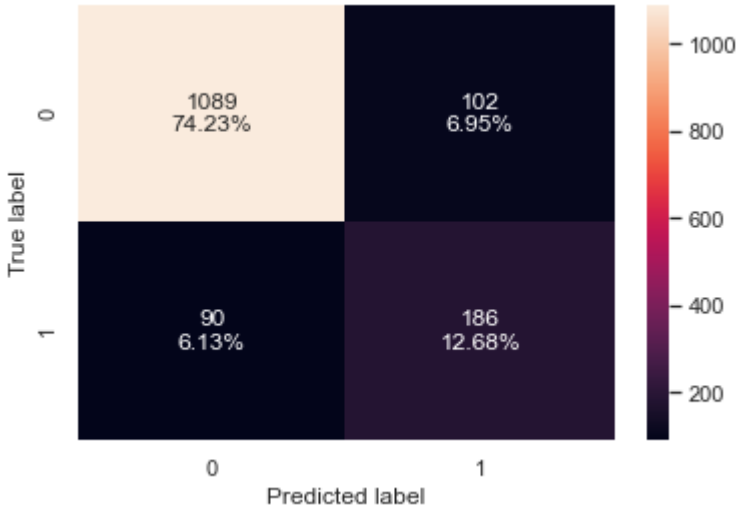
Tuning Random Forest

In [174…
```
# Choose the type of classifier.
rf_estimator = RandomForestClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    "class_weight": [{0: 0.19, 1: 0.81}],
    "n_estimators": [100, 150, 200, 250],
    "min_samples_leaf": np.arange(5, 10),
    "max_features": np.arange(0.2, 0.7, 0.1),
    "max_samples": np.arange(0.3, 0.7, 0.1),
}

# Run the grid search
grid_obj = GridSearchCV(rf_estimator, parameters, scoring="recall", cv=5)
```

```
grid_obj = grid_obj.fit(X_train, y_train)

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

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

Out[174…
```
RandomForestClassifier(class_weight={0: 0.19, 1: 0.81}, max_features=0.2,
                       max_samples=0.6000000000000001, min_samples_leaf=8,
                       random_state=1)
```

In [175…
```
confusion_matrix_sklearn(rf_estimator, X_test, y_test)
```



In [176…
```
rf_estimator_model_perf = get_metrics_score(rf_estimator)
```

```
Accuracy on training set : 0.903
Accuracy on test set : 0.869
Recall on training set : 0.848
Recall on test set : 0.674
Precision on training set : 0.699
Precision on test set : 0.646
F1_score on training set : 0.766
F1_score on test set : 0.660
```

- Random forest after tuning has given a better performance on recall and reduced the overfitting compared to un-tuned random forest.
- Although data is not overfitting, the model performance still getting only 0.674 on recall.
- We can keep changing the parameters and try it for better performance but, Hyperparameter tuning is tricky and exhaustive in the sense that there is no direct way to calculate how a change in it will reduce the loss of your model until we try those hyperparameters.

Comparing all ***Baggin*** models

In [177…
```
# defining list of models
models = [
    dtree,
    dtree_estimator,
    bagging,
    bagging_wt,
    bagging_estimator,
```

```
    rf,
    rf_wt,
    rf_estimator,
]

# defining empty lists to add train and test results
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []
F1_score_train = []
F1_score_test = []

# looping through all the models to get the accuracy, recall and precision sc
for model in models:
    j = get_metrics_score(model, False)
    acc_train.append(np.round(j[0], 3))
    acc_test.append(np.round(j[1], 3))
    recall_train.append(np.round(j[2], 3))
    recall_test.append(np.round(j[3], 3))
    precision_train.append(np.round(j[4], 3))
    precision_test.append(np.round(j[5], 3))
    F1_score_train.append(np.round(j[6], 3))
    F1_score_test.append(np.round(j[7], 3))
```

In [178…

```
comparison_frame = pd.DataFrame(
    {
        "Model": [
            "Decision Tree - default parameters",
            "Decision Tree - Tunned",
            "Bagging classifier - default parameters",
            "Bagging Classifier - weighted",
            "Bagging Classifier - Tunned",
            "Random Forest - deafult parameters",
            "Random Forest - weighted",
            "Random Forest - Tunned",
        ],
        "Train_Accuracy": acc_train,
        "Test_Accuracy": acc_test,
        "Train_Recall": recall_train,
        "Test_Recall": recall_test,
        "Train_Precision": precision_train,
        "Test_Precision": precision_test,
        "Train_F1_Score": F1_score_train,
        "Test_F1_Score": F1_score_test,
    }
)
comparison_frame
```

Out[178…

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_
0	Decision Tree - default parameters	1.000	0.885	1.000	0.714	1.000	
1	Decision Tree - Tunned	0.796	0.803	0.648	0.663	0.469	
2	Bagging classifier -	0.994	0.905	0.969	0.587	0.998	

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_
	default parameters						
3	Bagging Classifier - weighted	0.994	0.898	0.967	0.547	0.998	
4	Bagging Classifier - Tunned	0.716	0.714	0.666	0.703	0.362	
5	Random Forest - deafult parameters	1.000	0.915	1.000	0.583	1.000	
6	Random Forest - weighted	1.000	0.911	1.000	0.547	1.000	
7	Random Forest - Tunned	0.903	0.869	0.848	0.674	0.699	

Conclusion:

All model are performing poorly and/or overfitting.

- Default Decision Tree is overfitting on the other hand Tunned Decision Tree is performing poorly;
- Default Bagging is performing poorly on *recall* although is performing well on the others metrics. Considering that *recall* is important for our company, this is not a good model;
- Weighted Bagging is performing a litte bit worse on test and still not good on *recall*;
- Bagging tunned reduced drasticaly the perform in training and test data, is not overfitting but is not performing well;
- Random Forest default and weighted is performing almost the same, overfitting on *recall* and F1_score;

> Random Forest Tunned reduced the overfitting and is performing better compare to the others models. Performance is balanced between metrics (recall, precision and F1_score). It still need improvement considering that is performing *0.674 on test recall*.

Feature importance of tunned Random Forest

In [179…

```
# importance of features in the tree building on Random Forest Tunned

print(
    pd.DataFrame(
        rf_estimator.feature_importances_, columns=["Imp"], index=X_train.col
    ).sort_values(by="Imp", ascending=False)
)
```

```
                          Imp
Passport             0.125679
MonthlyIncome        0.117081
```

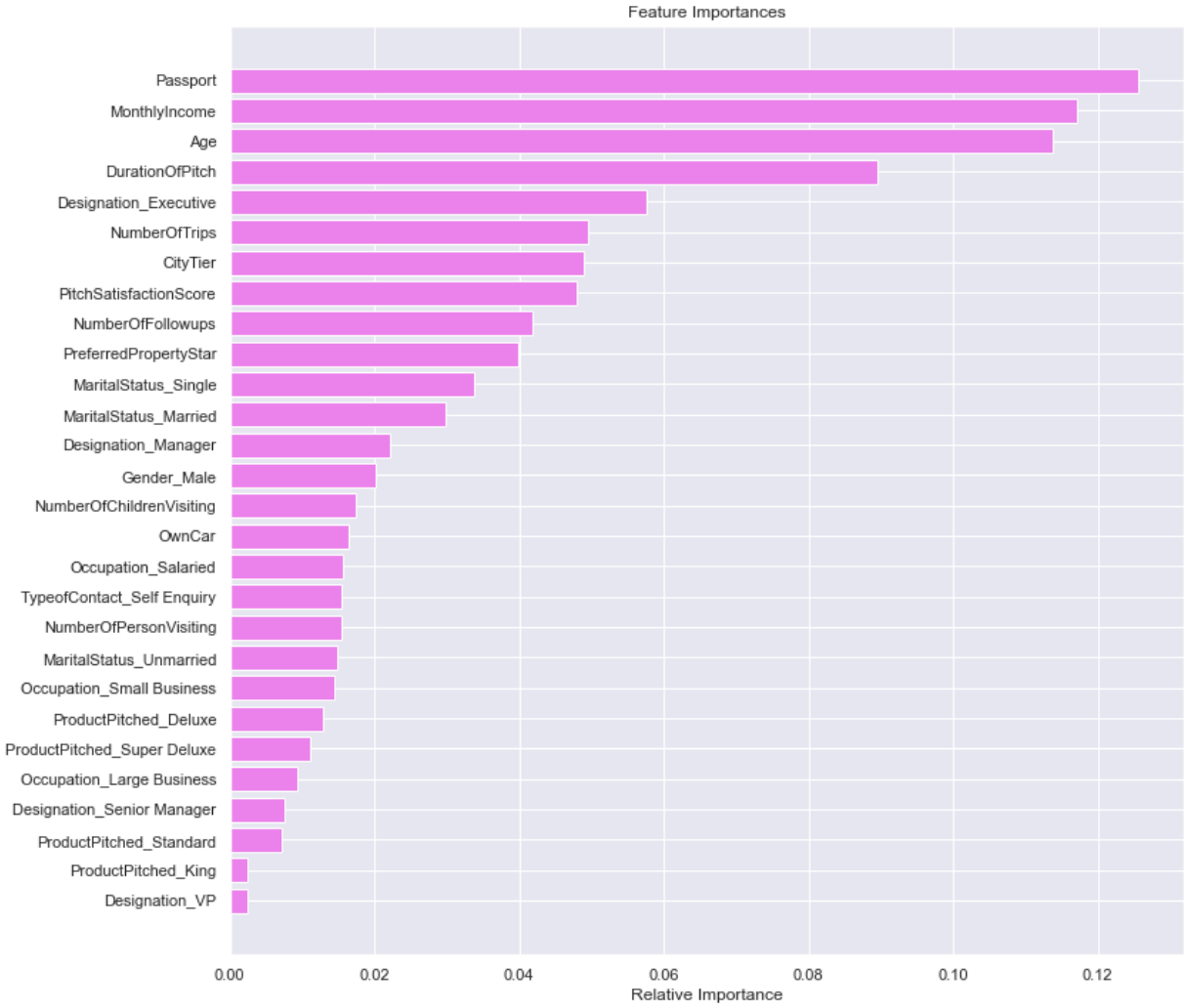
```
Age                            0.113792
DurationOfPitch                0.089557
Designation_Executive          0.057541
NumberOfTrips                  0.049538
CityTier                       0.048926
PitchSatisfactionScore         0.047908
NumberOfFollowups              0.041924
PreferredPropertyStar          0.039876
MaritalStatus_Single           0.033724
MaritalStatus_Married          0.029896
Designation_Manager            0.022138
Gender_Male                    0.020064
NumberOfChildrenVisiting       0.017322
OwnCar                         0.016488
Occupation_Salaried            0.015689
TypeofContact_Self Enquiry     0.015494
NumberOfPersonVisiting         0.015404
MaritalStatus_Unmarried        0.014788
Occupation_Small Business      0.014505
ProductPitched_Deluxe          0.012871
ProductPitched_Super Deluxe    0.011050
Occupation_Large Business      0.009240
Designation_Senior Manager     0.007464
ProductPitched_Standard        0.007243
ProductPitched_King            0.002440
Designation_VP                 0.002359
```

In [180…

```
feature_names = X_train.columns
```

In [181…

```
importances = rf_estimator.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="ce
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Passport is the most important feature followed by Monthly Income, Age, Duration Of Pitch and Designation_Executive;
- Number Of Trips, City Tier, Pitch Satisfaction Score have almost equal importance;
- Designation_VP and ProductPitched_King have very little importance.

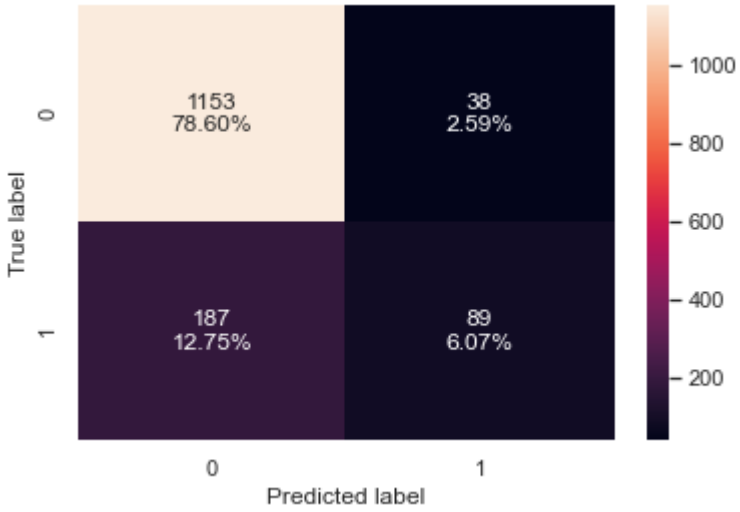
BOOSTING

AdaBoost Classifier - default

In [182…
```
abc = AdaBoostClassifier(random_state=1)
abc.fit(X_train, y_train)
```

Out[182… `AdaBoostClassifier(random_state=1)`

In [183…
```
confusion_matrix_sklearn(abc, X_test, y_test)
```



In [184…
```
# Using above defined function to get accuracy, recall and precision on train
abc_score = get_metrics_score(abc)
```

```
Accuracy on training set : 0.849
Accuracy on test set : 0.847
Recall on training set : 0.331
Recall on test set : 0.322
Precision on training set : 0.712
Precision on test set : 0.701
F1_score on training set : 0.452
F1_score on test set : 0.442
```

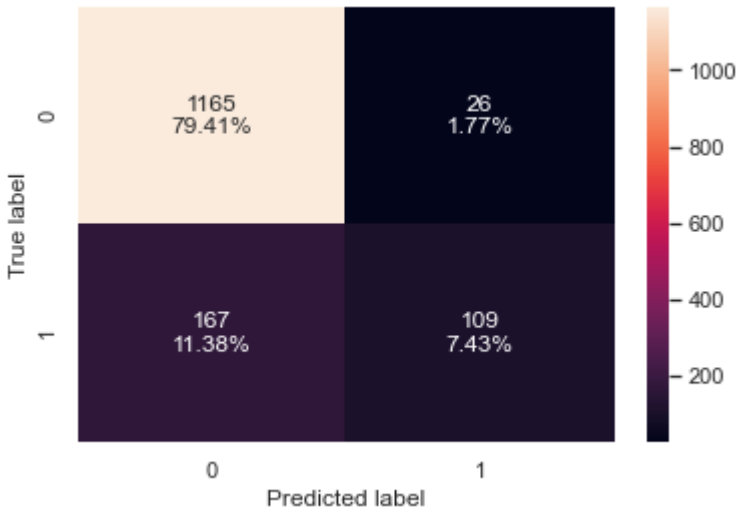
- AdaBoost Classifier Default does not show overfitting however is performing unsuccessfully, recall is doing 0.331 on training and 0.322 on test.

Gradient Boosting Classifier

In [185…
```
gbc = GradientBoostingClassifier(random_state=1)
gbc.fit(X_train, y_train)
```

Out[185…
```
GradientBoostingClassifier(random_state=1)
```

In [186…
```
confusion_matrix_sklearn(gbc, X_test, y_test)
```



In [187…
```
# Using above defined function to get accuracy, recall and precision on train
gbc_score = get_metrics_score(gbc)
```

```
Accuracy on training set : 0.888
Accuracy on test set : 0.868
Recall on training set : 0.449
Recall on test set : 0.395
Precision on training set : 0.909
Precision on test set : 0.807
F1_score on training set : 0.601
F1_score on test set : 0.530
```

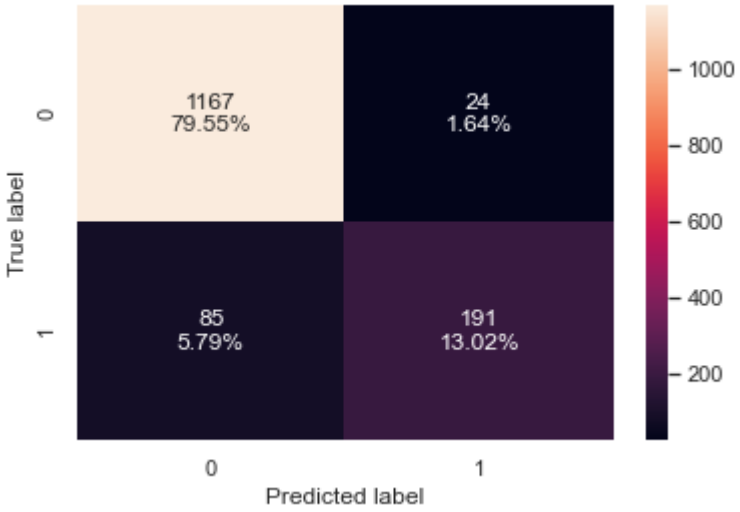
- Gradient Boosting Default does not show overfitting and is performing better than Adaboost however, still performing poorly, only 0.395 on recall test data set.

XGBoost Classifier

In [188…
```
xgb = XGBClassifier(random_state=1, eval_metric="error")
xgb.fit(X_train, y_train)
```

Out[188…
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='error',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=12,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [189…
```
confusion_matrix_sklearn(xgb, X_test, y_test)
```



In [190…
```
# Using above defined function to get accuracy, recall and precision on train
xgb_score = get_metrics_score(xgb)
```

```
Accuracy on training set : 1.000
Accuracy on test set : 0.926
Recall on training set : 0.998
Recall on test set : 0.692
Precision on training set : 1.000
Precision on test set : 0.888
F1_score on training set : 0.999
F1_score on test set : 0.778
```

- XGBoost Default does show an overfitting but is performing better than Adaboost and Gradient Boosting.

> ***Considering that XBBoost treats missing values, we gonna check the performance of the model with the data `before` missing values treatment***

In [191…
```
# Data (before missing values treatment) Preparation:
X_xg = df_XG.drop(
    ["ProdTaken"], axis=1
)  # Creating x with all independent variable, removing target variable
X_xg = pd.get_dummies(
    X_xg, drop_first=True
)  # All category variables we creat dummies and drop first.
y_xg = df_XG["ProdTaken"]
```

In [192…
```
# Partition the data into train and test set:
X_xg_train, X_xg_test, y_xg_train, y_xg_test = train_test_split(
    X_xg, y_xg, test_size=0.3, random_state=1, stratify=y_xg
)  # using stratify because of imbalance distribution of the target classes
print(f"X_Train: {X_xg_train.shape},\nX_test: {X_test.shape}")
```

```
X_Train: (3421, 28),
X_test: (1467, 28)
```

In [193…
```
# Checking the % distribution of the target classes
y_xg.value_counts(1)
```

Out[193…
```
0    0.811784
1    0.188216
Name: ProdTaken, dtype: float64
```

In [194…
```
y_xg_train.value_counts(1)
```

Out[194…
```
0    0.811751
1    0.188249
Name: ProdTaken, dtype: float64
```

In [195…
```
y_xg_test.value_counts(1)
```

Out[195…
```
0    0.811861
1    0.188139
Name: ProdTaken, dtype: float64
```

In [196…
```
##  Function to calculate different metric scores of the model - Accuracy, Re
def get_metrics_score_2(model, flag=True):
    """
    model : classifier to predict values of X

    """
    # defining an empty list to store train and test results
    score_list = []

    # Predicting on train and tests
    pred_train = model.predict(X_xg_train)
    pred_test = model.predict(X_xg_test)
```

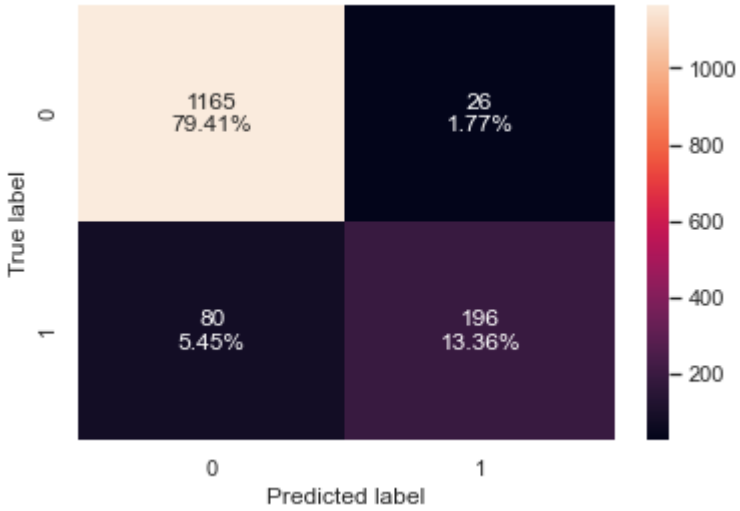
```
    # Accuracy of the model
    train_acc = model.score(X_xg_train, y_xg_train)
    test_acc = model.score(X_xg_test, y_xg_test)

    # Recall of the model
    train_recall = metrics.recall_score(y_xg_train, pred_train)
    test_recall = metrics.recall_score(y_xg_test, pred_test)

    # Precision of the model
    train_precision = metrics.precision_score(y_xg_train, pred_train)
    test_precision = metrics.precision_score(y_xg_test, pred_test)

    # F1-score of the model
    train_F1 = metrics.f1_score(y_xg_train, pred_train)
    test_F1 = metrics.f1_score(y_xg_test, pred_test)

    score_list.extend(
        (
            train_acc,
            test_acc,
            train_recall,
            test_recall,
            train_precision,
            test_precision,
            train_F1,
            test_F1,
        )
    )

    # If the flag is set to True then only the following print statements wil
    if flag == True:
        print(
            "Accuracy on training set : {:.3f}".format(
                model.score(X_xg_train, y_xg_train)
            )
        )
        print("Accuracy on test set : {:.3f}".format(model.score(X_xg_test, y
        print(
            "Recall on training set : {:.3f}".format(
                metrics.recall_score(y_xg_train, pred_train)
            )
        )
        print(
            "Recall on test set : {:.3f}".format(
                metrics.recall_score(y_xg_test, pred_test)
            )
        )
        print(
            "Precision on training set : {:.3f}".format(
                metrics.precision_score(y_xg_train, pred_train)
            )
        )
        print(
            "Precision on test set : {:.3f}".format(
                metrics.precision_score(y_xg_test, pred_test)
            )
        )
        print(
            "F1_score on training set : {:.3f}".format(
                metrics.f1_score(y_xg_train, pred_train)
            )
        )
        print(
            "F1_score on test set : {:.3f}".format(
```

```
                metrics.f1_score(y_xg_test, pred_test)
            )
        )

    return score_list  # returning the list with train and test scores
```

In [197…
```
xgb_2 = XGBClassifier(random_state=1, eval_metric="error")
xgb_2.fit(X_xg_train, y_xg_train)
```

Out[197…
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='error',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=12,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [198…
```
confusion_matrix_sklearn(xgb_2, X_xg_test, y_xg_test)
```



In [199…
```
# Using above defined function to get accuracy, recall and precision on train
xgb_2_score = get_metrics_score_2(xgb_2)
```

```
Accuracy on training set : 1.000
Accuracy on test set : 0.928
Recall on training set : 0.998
Recall on test set : 0.710
Precision on training set : 1.000
Precision on test set : 0.883
F1_score on training set : 0.999
F1_score on test set : 0.787
```

- XGBoost Default `without missing values treatment` does show an overfitting on *recall* and *F1_score* but is performing better than the others models.
- XGBoost Default `without missing values treatment` is performing a litter bit better than XGBoost Default. Recall on test 0.710 vs 0.692 respectivally.

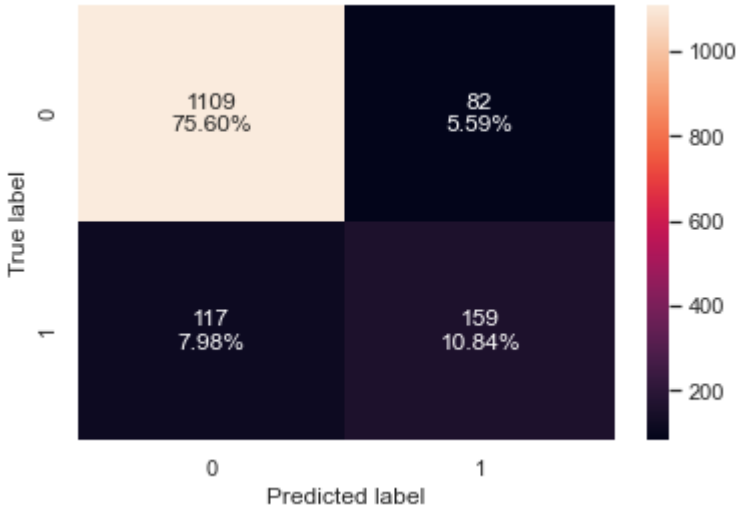
Hyperparameter Tuning

AdaBoost Classifier

In [200…
```
# Choose the type of classifier.
abc_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
## add from article
parameters = {
    # Let's try different max_depth for base_estimator
    "base_estimator": [
        DecisionTreeClassifier(max_depth=1),
        DecisionTreeClassifier(max_depth=2),
        DecisionTreeClassifier(max_depth=3),
    ],
    "n_estimators": np.arange(10, 110, 10),
    "learning_rate": np.arange(0.1, 2, 0.1),
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(abc_tuned, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

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

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

Out[200…
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                   learning_rate=1.6, n_estimators=100, random_state=1)
```

In [201…
```
confusion_matrix_sklearn(abc_tuned, X_test, y_test)
```



In [202…
```
# Using above defined function to get accuracy, recall and precision on train
abc_tuned_score = get_metrics_score(abc_tuned)
```

```
Accuracy on training set : 0.990
Accuracy on test set : 0.864
Recall on training set : 0.964
Recall on test set : 0.576
Precision on training set : 0.981
Precision on test set : 0.660
F1_score on training set : 0.973
F1_score on test set : 0.615
```

- The model is overfitting on the train data. The train recall is much higher than the test recall.
- The model has low performance on test recall. This implies that the model is not good at identifying potencial buyers.
- In contrast, AdaBoost Tunned is performing better that AdaBoost default but still not better compared to XGBoost `without missing values treatment` .

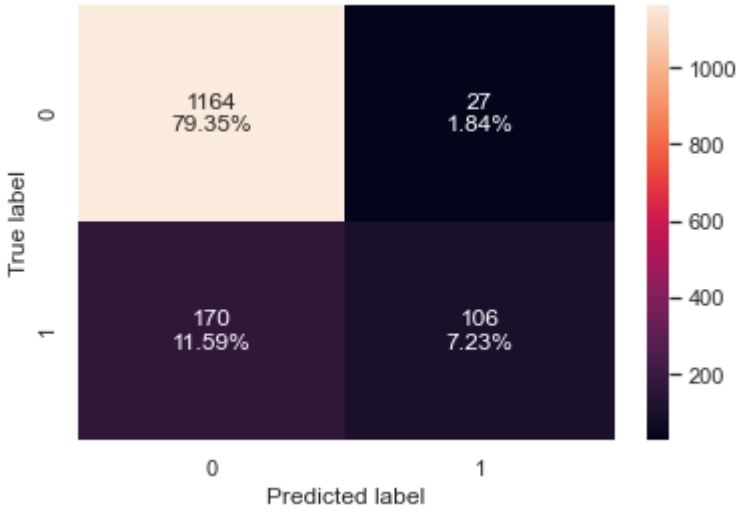
Gradient Boosting Classifier

using AdaBoost classifier as the estimator for initial predictions

In [203…
```
gbc_init = GradientBoostingClassifier(
    init=AdaBoostClassifier(random_state=1), random_state=1
)
gbc_init.fit(X_train, y_train)
```

Out[203…
```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           random_state=1)
```

In [204…
```
confusion_matrix_sklearn(gbc_init, X_test, y_test)
```



In [205…
```
# Using above defined function to get accuracy, recall and precision on train
gbc_init_score = get_metrics_score(gbc_init)
```

```
Accuracy on training set : 0.886
Accuracy on test set : 0.866
Recall on training set : 0.452
Recall on test set : 0.384
Precision on training set : 0.890
Precision on test set : 0.797
F1_score on training set : 0.599
F1_score on test set : 0.518
```

As compared to the model with default parameters:

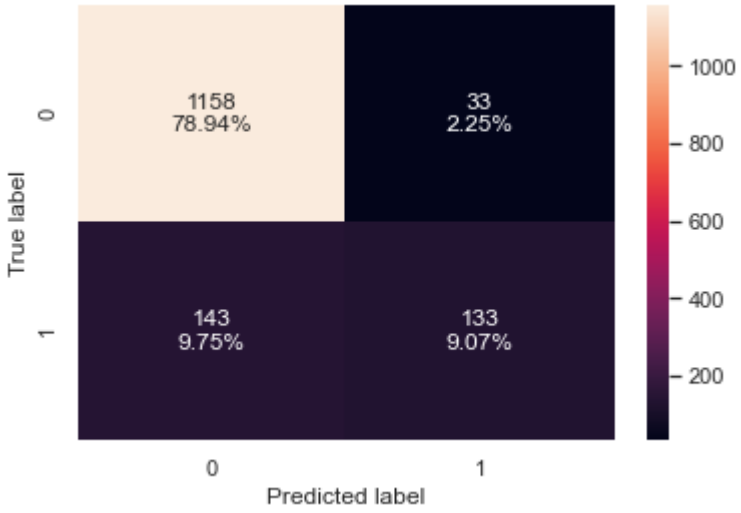
- Test accuracy and test recall have increased slightly.
- As we are getting better results, we will use init = AdaBoostClassifier() to tune the gradient boosting model.

Gradient Boosting Tuned

In [206…
```
# Choose the type of classifier.
gbc_tuned = GradientBoostingClassifier(
    init=AdaBoostClassifier(random_state=1), random_state=1
)

# Grid of parameters to choose from
## add from article
parameters = {
    "n_estimators": [100, 150, 200, 250],
    "subsample": [0.8, 0.9, 1],
    "max_features": [0.7, 0.8, 0.9, 1],
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(gbc_tuned, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

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

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

Out[206…
```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.9, n_estimators=250, random_state=
1,
                           subsample=0.8)
```

In [207…
```
confusion_matrix_sklearn(gbc_tuned, X_test, y_test)
```



In [208…
```
# Using above defined function to get accuracy, recall and precision on train
gbc_tuned_score = get_metrics_score(gbc_tuned)
```

```
Accuracy on training set : 0.923
Accuracy on test set : 0.880
Recall on training set : 0.623
Recall on test set : 0.482
Precision on training set : 0.948
Precision on test set : 0.801
F1_score on training set : 0.752
F1_score on test set : 0.602
```

- The model performace has not increased by much.
- The model has started to overfit the train data in terms of recall.
- It is better at identifying non-defaulters than identifying defaulters which is the opposite of the result we need.

XGBoost Classifier - tuned

In [209…

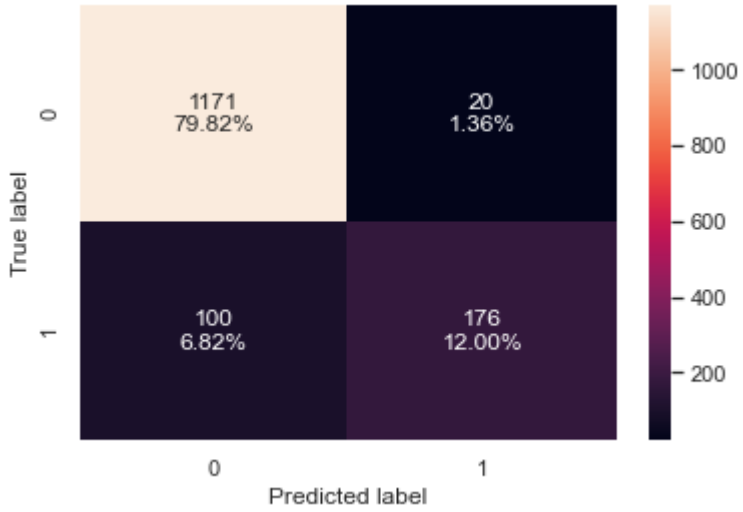
```
# Choose the type of classifier.
xgb_tuned = XGBClassifier(eval_metric="error", random_state=1)

# Grid of parameters to choose from
## add from
parameters = {
    "n_estimators": np.arange(10, 100, 20),
    "subsample": [0.5, 0.7, 0.9, 1],
    "learning_rate": [0.01, 0.1, 0.2, 0.05],
    "gamma": [0, 1, 3],
    "colsample_bytree": [0.5, 0.7, 0.9, 1],
    "colsample_bylevel": [0.5, 0.7, 0.9, 1],
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

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

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

Out[209…

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.9, eval_metric='error',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.2, max_delta_step=
0,
              max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=90, n_jobs=12,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [210…

```
confusion_matrix_sklearn(xgb_tuned, X_test, y_test)
```



In [211…
```
# Using above defined function to get accuracy, recall and precision on train
xgb_tuned_score = get_metrics_score(xgb_tuned)
```

```
Accuracy on training set : 0.994
Accuracy on test set : 0.918
Recall on training set : 0.966
Recall on test set : 0.638
Precision on training set : 1.000
Precision on test set : 0.898
F1_score on training set : 0.983
F1_score on test set : 0.746
```

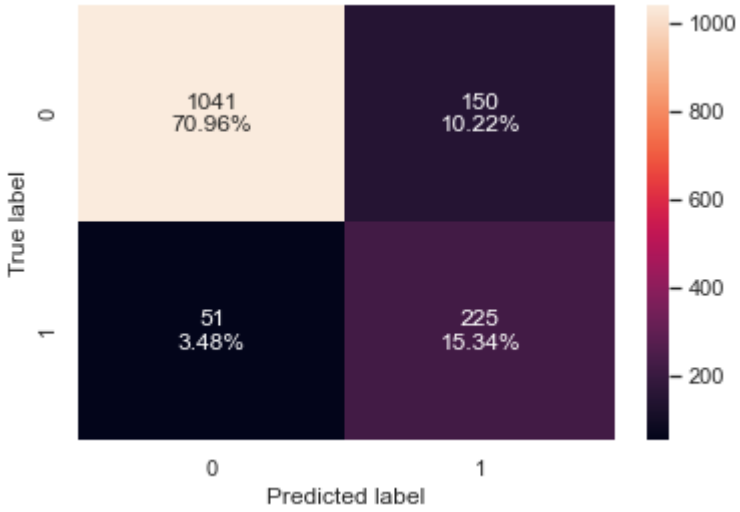
- XGB Tuned is better than other models but still performing badly on recall and overfitting on Precission ans F1_score.

XGB Tuned 2 - different parameters

In [212…
```
# Choose the type of classifier.
xgb_tuned2 = XGBClassifier(eval_metric="error", random_state=1)

# Grid of parameters to choose from
## add from
parameters = {
    "n_estimators": [10, 30, 50],
    "scale_pos_weight": [1, 2, 5],
    "subsample": [0.7, 0.9, 1],
    "learning_rate": [0.05, 0.1, 0.2],
    "colsample_bytree": [0.7, 0.9, 1],
    "colsample_bylevel": [0.5, 0.7, 1],
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned2, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
xgb_tuned2 = grid_obj.best_estimator_

# Fit the best algorithm to the data.
xgb_tuned2.fit(X_train, y_train)
```

Out[212…
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7, eval_metric='error',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.1, max_delta_step=
0,
              max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=30, n_jobs=12,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=5, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [213…
```
confusion_matrix_sklearn(xgb_tuned2, X_test, y_test)
```

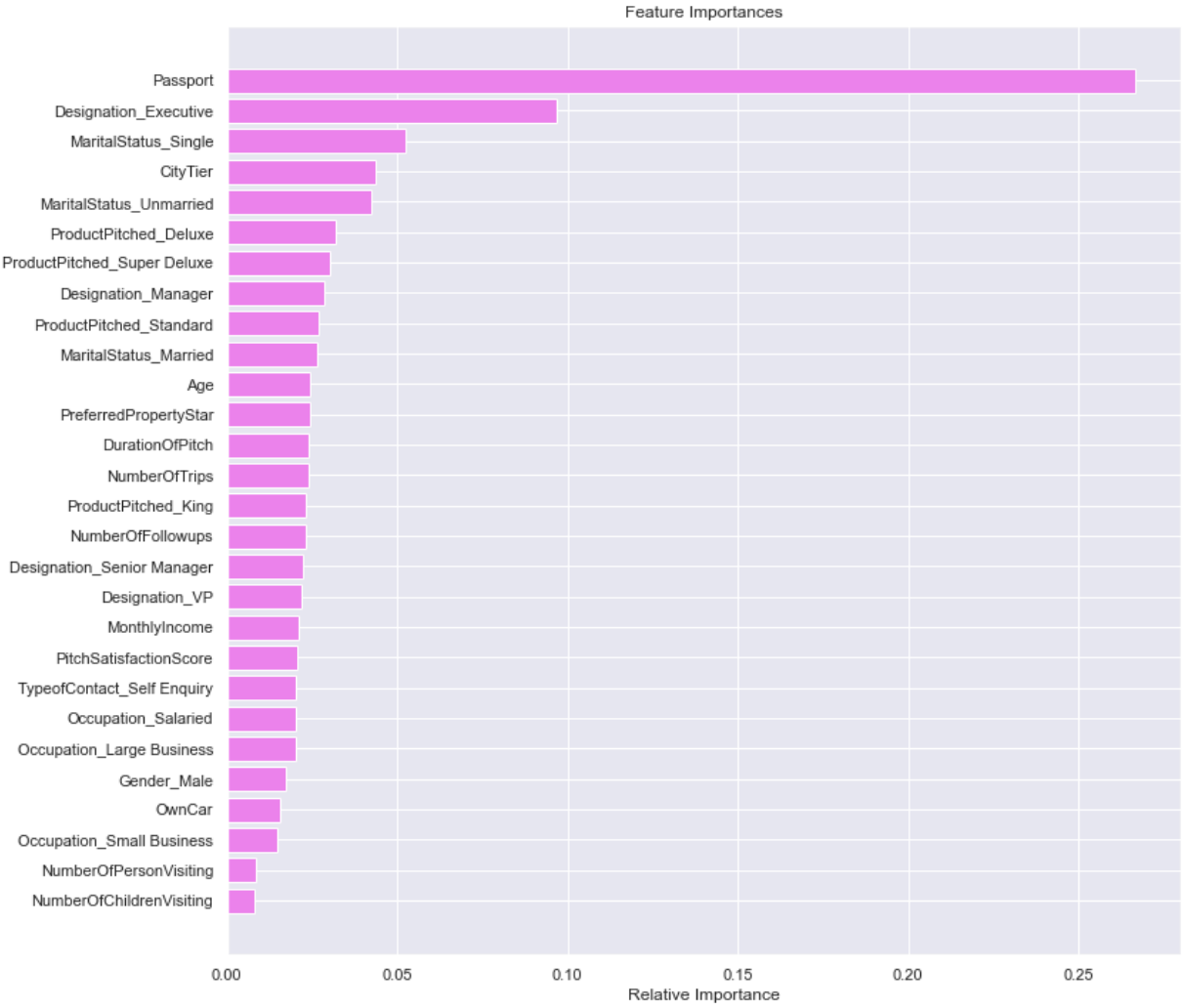


In [214…
```
# Using above defined function to get accuracy, recall and precision on train
xgb_tuned2_score = get_metrics_score(xgb_tuned2)
```

```
Accuracy on training set : 0.906
Accuracy on test set : 0.863
Recall on training set : 0.935
Recall on test set : 0.815
Precision on training set : 0.684
Precision on test set : 0.600
F1_score on training set : 0.790
F1_score on test set : 0.691
```

- The test accuracy of the model has reduced as compared to the model with default parameters but the recall has increased significantly and the model is able to identify most of the defaulters.
- Decreasing number of false negatives has increased the number of false positives here, so precission is not permorfing well.
- The tuned model is not overfitting.

In [215…
```
importances = xgb_tuned2.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="ce
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Passaport is the most importance feature followed by Designation_Executive and Marita Status_Single.
- City Tier and Marital Status_Unmarried have almost same importance.
- Number of Children and Person Visiting have the lowest importance.

XGB tuned without missing values treatment

In [216…

```
# Choose the type of classifier.
xgb2_tuned = XGBClassifier(eval_metric="error", random_state=1)

# Grid of parameters to choose from
## add from
parameters = {
    "n_estimators": np.arange(10, 100, 20),
    "scale_pos_weight": [0, 1, 2, 5],
    "subsample": [0.5, 0.7, 0.9, 1],
    "learning_rate": [0.01, 0.1, 0.2, 0.05],
    "gamma": [0, 1, 3],
    "colsample_bytree": [0.5, 0.7, 0.9, 1],
    "colsample_bylevel": [0.5, 0.7, 0.9, 1],
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(xgb2_tuned, parameters, scoring=acc_scorer, cv=5)
```

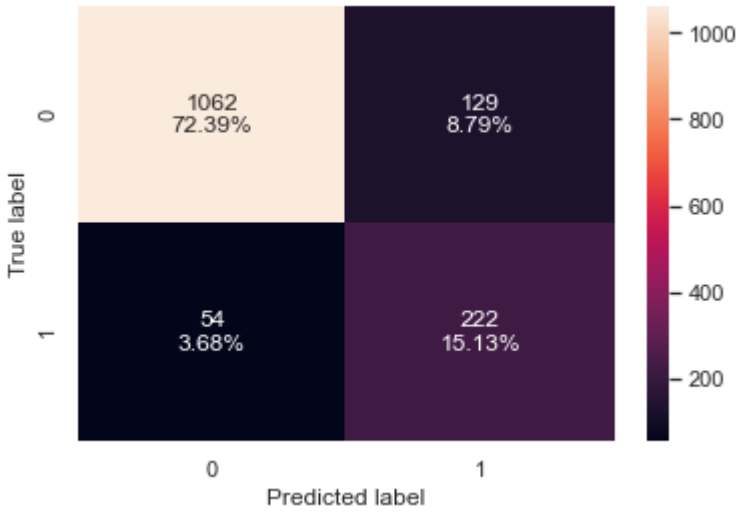
```
grid_obj = grid_obj.fit(X_xg_train, y_xg_train)

# Set the clf to the best combination of parameters
xgb2_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
xgb2_tuned.fit(X_xg_train, y_xg_train)
```

Out[216…

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
              colsample_bynode=1, colsample_bytree=0.9, eval_metric='error',
              gamma=3, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.2, max_delta_step=
0,
              max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=30, n_jobs=12,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=5, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [217…

```
confusion_matrix_sklearn(xgb2_tuned, X_xg_test, y_xg_test)
```



In [218…

```
# Using above defined function to get accuracy, recall and precision on train
xgb2_tuned_score = get_metrics_score_2(xgb2_tuned)
```

```
Accuracy on training set : 0.946
Accuracy on test set : 0.875
Recall on training set : 0.980
Recall on test set : 0.804
Precision on training set : 0.785
Precision on test set : 0.632
F1_score on training set : 0.872
F1_score on test set : 0.708
```

- XGB tuned with data before missing values treatment is performing better than other models.
- Model is not overfitting and Recall is performing 0.804, lets try other parameter and see if we can improve it.

XGB tuned 2 - without missing values treatment

In [219…

```
# Choose the type of classifier.
xgb2_tuned2 = XGBClassifier(eval_metric="error", random_state=1)

# Grid of parameters to choose from
```

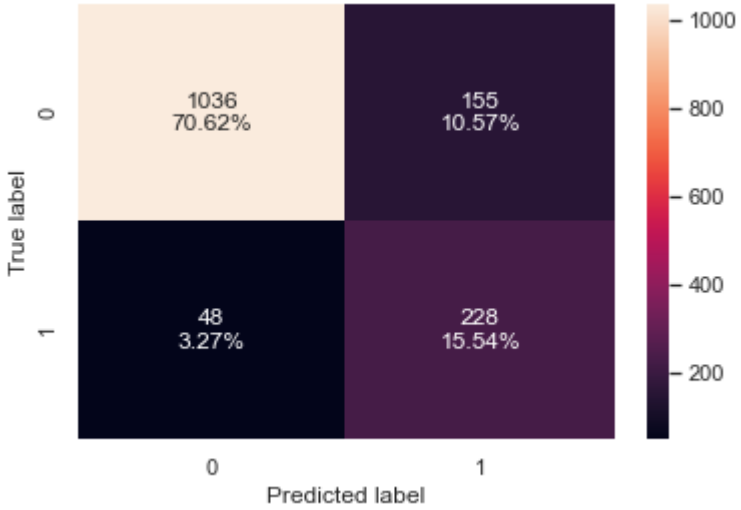
```
## add from
parameters = {
    "n_estimators": [10, 30, 50],
    "scale_pos_weight": [1, 2, 5],
    "subsample": [0.7, 0.9, 1],
    "learning_rate": [0.05, 0.1, 0.2],
    "colsample_bytree": [0.7, 0.9, 1],
    "colsample_bylevel": [0.5, 0.7, 1],
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(xgb2_tuned2, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_xg_train, y_xg_train)

# Set the clf to the best combination of parameters
xgb2_tuned2 = grid_obj.best_estimator_

# Fit the best algorithm to the data.
xgb2_tuned2.fit(X_xg_train, y_xg_train)
```

Out[219…

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7, eval_metric='error',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.1, max_delta_step=
0,
              max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=30, n_jobs=12,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=5, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [220…

```
confusion_matrix_sklearn(xgb2_tuned2, X_xg_test, y_xg_test)
```



In [221…

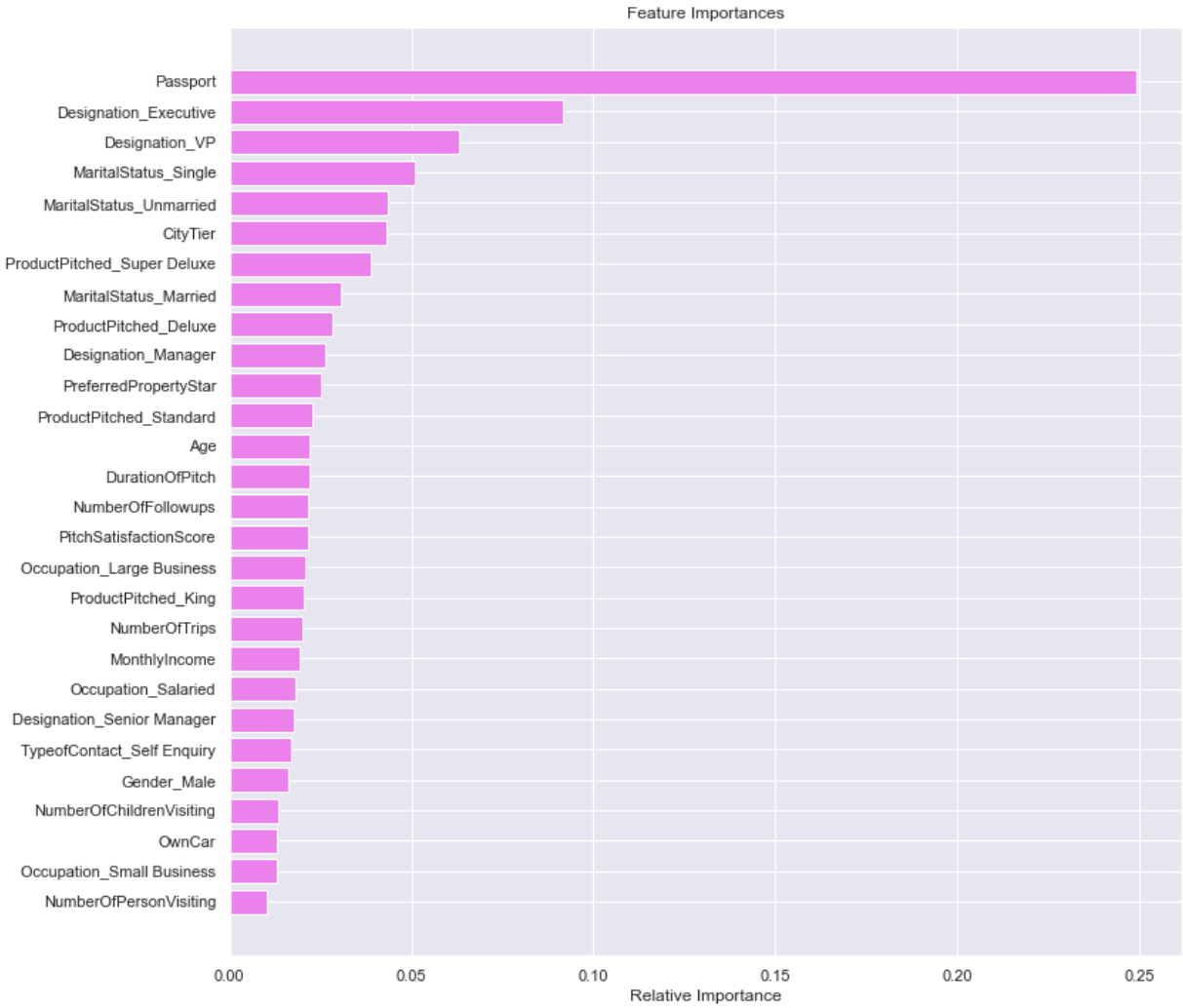
```
xgb2_tuned2_score = get_metrics_score_2(xgb2_tuned2)
```

```
Accuracy on training set : 0.909
Accuracy on test set : 0.862
Recall on training set : 0.929
Recall on test set : 0.826
Precision on training set : 0.694
Precision on test set : 0.595
F1_score on training set : 0.794
F1_score on test set : 0.692
```

- This model reduced even more the overfitting, performance on training and test is closer.
- Recall is performing better 0.826 on test, precission and F1_socre is not that well 0.595 and 0.692 respectivally, but still the best model.

In [222…

```
importances = xgb2_tuned2.feature_importances_
indices = np.argsort(importances)
feature_names = list(X_xg.columns)

plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="ce
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Passaport is the most importance feature followed by Designation_Executive, Designation_VP and Marital Status_Single.
- Marital Status_Unmarried and City Tier have almost same importance.
- Number of Person Visiting and Occupation_Small Business have the lowest importance.

Stacking Classifier

In [224…

```
estimators = [
    ("Random Forest", rf_estimator),
```

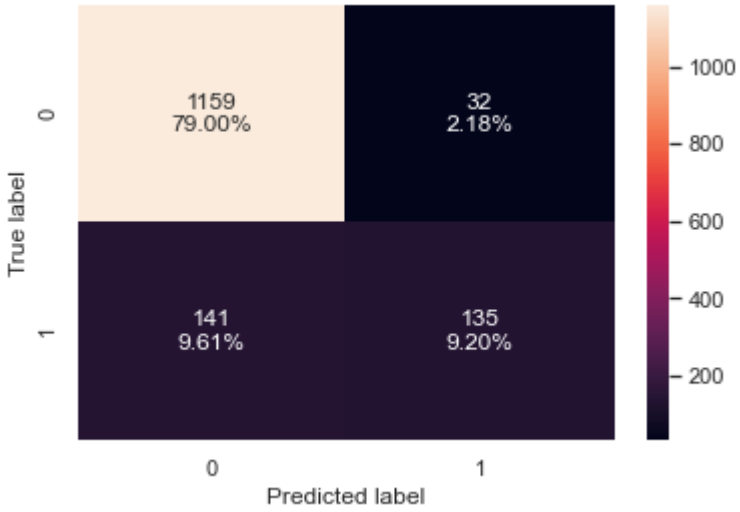
```
    ("Gradient Boosting", gbc_tuned),
    ("Decision Tree", dtree_estimator),
]

final_estimator = xgb_tuned

stacking_classifier = StackingClassifier(
    estimators=estimators, final_estimator=final_estimator
)

stacking_classifier.fit(X_train, y_train)
```

Out[224…

```
StackingClassifier(estimators=[('Random Forest',
                                RandomForestClassifier(class_weight={0: 0.19,
                                                                     1: 0.8
1},
                                                       max_features=0.2,
                                                       max_samples=0.60000000
00000001,
                                                       min_samples_leaf=8,
                                                       random_state=1)),
                               ('Gradient Boosting',
                                GradientBoostingClassifier(init=AdaBoostClass
ifier(random_state=1),
                                                           max_features=0.9,
                                                           n_estimators=250,
                                                           random_state=1,
                                                           subsample=0.8)),
                               ('Decision Tree'...
                                                       eval_metric='error', gamma=
0,
                                                       gpu_id=-1,
                                                       importance_type='gain',
                                                       interaction_constraints='',
                                                       learning_rate=0.2,
                                                       max_delta_step=0, max_depth=
6,
                                                       min_child_weight=1,
                                                       missing=nan,
                                                       monotone_constraints='()',
                                                       n_estimators=90, n_jobs=12,
                                                       num_parallel_tree=1,
                                                       random_state=1, reg_alpha=0,
                                                       reg_lambda=1,
                                                       scale_pos_weight=1,
                                                       subsample=1,
                                                       tree_method='exact',
                                                       validate_parameters=1,
                                                       verbosity=None))
```

In [225…

```
confusion_matrix_sklearn(stacking_classifier, X_test, y_test)
```



In [226…
```
# Using above defined function to get accuracy, recall and precision on train
stacking_classifier_score = get_metrics_score(stacking_classifier)
```

```
Accuracy on training set : 0.918
Accuracy on test set : 0.882
Recall on training set : 0.630
Recall on test set : 0.489
Precision on training set : 0.908
Precision on test set : 0.808
F1_score on training set : 0.744
F1_score on test set : 0.609
```

- Stacking is overfitting and Recall (0.0.489) is performing badly compare to XGB2 tuned before missing values treatment (0.826)

In [227…
```
# defining list of models X_train
models = [
    abc,
    abc_tuned,
    gbc,
    gbc_init,
    gbc_tuned,
    xgb,
    xgb_tuned,
    xgb_tuned2,
]

# defining list of models X_xg_train
models_2 = [
    xgb_2,
    xgb2_tuned,
    xgb2_tuned2,
]


# defining list of models X_train
models_3 = [
    stacking_classifier,
]

# defining empty lists to add train and test results
acc_train = []
acc_test = []
recall_train = []
recall_test = []
```

```
precision_train = []
precision_test = []
F1_score_train = []
F1_score_test = []

# looping through all the models to get the accuracy, recall and precision sc
for model in models:
    j = get_metrics_score(model, False)
    acc_train.append(np.round(j[0], 3))
    acc_test.append(np.round(j[1], 3))
    recall_train.append(np.round(j[2], 3))
    recall_test.append(np.round(j[3], 3))
    precision_train.append(np.round(j[4], 3))
    precision_test.append(np.round(j[5], 3))
    F1_score_train.append(np.round(j[6], 3))
    F1_score_test.append(np.round(j[7], 3))


# looping through all the models to get the accuracy, recall and precision sc
for model in models_2:
    j = get_metrics_score_2(model, False)
    acc_train.append(np.round(j[0], 3))
    acc_test.append(np.round(j[1], 3))
    recall_train.append(np.round(j[2], 3))
    recall_test.append(np.round(j[3], 3))
    precision_train.append(np.round(j[4], 3))
    precision_test.append(np.round(j[5], 3))
    F1_score_train.append(np.round(j[6], 3))
    F1_score_test.append(np.round(j[7], 3))


# looping through all the models to get the accuracy, recall and precision sc
for model in models_3:
    j = get_metrics_score(model, False)
    acc_train.append(np.round(j[0], 3))
    acc_test.append(np.round(j[1], 3))
    recall_train.append(np.round(j[2], 3))
    recall_test.append(np.round(j[3], 3))
    precision_train.append(np.round(j[4], 3))
    precision_test.append(np.round(j[5], 3))
    F1_score_train.append(np.round(j[6], 3))
    F1_score_test.append(np.round(j[7], 3))
```

In [228…

```
comparison_frame2 = pd.DataFrame(
    {
        "Model": [
            "AdaBoost - default parameters",
            "AdaBoost - Tunned",
            "Gradient Boosting - default parameters",
            "Gradient Boosting - init",
            "Gradient Boosting - Tunned",
            "XGB - deafult parameters",
            "XGB - Tunned",
            "XGB2 - Tunned other parameters",
            "XGB_2 - deafult (No treatment Missing values)",
            "XGB_2 - Tunned (No treatment Missing values)",
            "XGB2_2 - Tunned other parameters (No treatment Missing values)",
            "stacking_classifier",
        ],
        "Train_Accuracy": acc_train,
        "Test_Accuracy": acc_test,
        "Train_Recall": recall_train,
        "Test_Recall": recall_test,
```

```
        "Train_Precision": precision_train,
        "Test_Precision": precision_test,
        "Train_F1_Score": F1_score_train,
        "Test_F1_Score": F1_score_test,
    }
)
comparison_frame2
```

Out [228...

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precisio
0	AdaBoost - default parameters	0.849	0.847	0.331	0.322	0.71
1	AdaBoost - Tunned	0.990	0.864	0.964	0.576	0.98
2	Gradient Boosting - default parameters	0.888	0.868	0.449	0.395	0.90
3	Gradient Boosting - init	0.886	0.866	0.452	0.384	0.89
4	Gradient Boosting - Tunned	0.923	0.880	0.623	0.482	0.94
5	XGB - deafult parameters	1.000	0.926	0.998	0.692	1.00
6	XGB - Tunned	0.994	0.918	0.966	0.638	1.00
7	XGB2 - Tunned other parameters	0.906	0.863	0.935	0.815	0.68
8	XGB_2 - deafult (No treatment Missing values)	1.000	0.928	0.998	0.710	1.00
9	XGB_2 - Tunned (No treatment Missing values)	0.946	0.875	0.980	0.804	0.78
10	XGB2_2 - Tunned other parameters (No treatment...	0.909	0.862	0.929	0.826	0.69
11	stacking_classifier	0.918	0.882	0.630	0.489	0.90

Conclusion:

- Default AdaBoost is performing poorly on the other hand Tunned AdaBoost is overfitting;
- Default Gradient is performing poorly on *recall* although is performing well on the others metrics. Considering that *recall* is important for our company, this is not a good model;
- Gradient Boosting - init is not overfittin, on the other hand is performing bad on *recall* and Gradient Boosting - Tunned is overfitting.
- XGB - deafult parameters, XGB - Tunned and XGB_2 - deafult (No treatment Missing values) and stacking_classifier are overfitting.

> XGB2 - Tunned other parameters and XGB_2 - Tunned (No treatment Missing values) are performing well and not overfitting but ***XGB2_2 -***

> ***Tunned other parameters (No treatment Missing values)*** is performing better on *recall*.

In [229…
```
comparison_frame.append(comparison_frame2, ignore_index=True)
```

Out[229…

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precisio
0	Decision Tree - default parameters	1.000	0.885	1.000	0.714	1.00
1	Decision Tree - Tunned	0.796	0.803	0.648	0.663	0.46
2	Bagging classifier - default parameters	0.994	0.905	0.969	0.587	0.99
3	Bagging Classifier - weighted	0.994	0.898	0.967	0.547	0.99
4	Bagging Classifier - Tunned	0.716	0.714	0.666	0.703	0.36
5	Random Forest - deafult parameters	1.000	0.915	1.000	0.583	1.00
6	Random Forest - weighted	1.000	0.911	1.000	0.547	1.00
7	Random Forest - Tunned	0.903	0.869	0.848	0.674	0.69
8	AdaBoost - default parameters	0.849	0.847	0.331	0.322	0.71
9	AdaBoost - Tunned	0.990	0.864	0.964	0.576	0.98
10	Gradient Boosting - default parameters	0.888	0.868	0.449	0.395	0.90
11	Gradient Boosting - init	0.886	0.866	0.452	0.384	0.89
12	Gradient Boosting - Tunned	0.923	0.880	0.623	0.482	0.94
13	XGB - deafult parameters	1.000	0.926	0.998	0.692	1.00
14	XGB - Tunned	0.994	0.918	0.966	0.638	1.00
15	XGB2 - Tunned other parameters	0.906	0.863	0.935	0.815	0.68
16	XGB_2 - deafult (No treatment Missing values)	1.000	0.928	0.998	0.710	1.00
17	XGB_2 - Tunned (No treatment Missing values)	0.946	0.875	0.980	0.804	0.78
18	XGB2_2 - Tunned other parameters (No treatment...	0.909	0.862	0.929	0.826	0.69

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precisio
19	stacking_classifier	0.918	0.882	0.630	0.489	0.90

Conclusion

- Overall we can see that the XGBoost performs better on this dataset then others models.
- Considering that XGboost treats missing values, we can see that the model applyed on data set `without missing values treatment` is the one performing better.
- We testes 2 differents parameter on them and the second one come out performing a litte bit better on Recal and reducing even more the overfitting.

> XGB2_2 - Tunned other parameters (No treatment missing values) is our best model, it reduced the overfitting and is performing better compare to the others models. Performance on Precision and F1_score is not performing that well. It still need improvement considering that is performing *0.826 on test recall and 0.595 on precission*.

Business Recommendations

- We have been able to build a predictive model:

  a) that the company can deploy to identify customers who will be interested in buy a travel package.

  b) that the company can use to find the key factors that will have an impact on converting a custumer as a buyer.

- Factors that have an impact: Passport, Designation_Executive, Designation_VP and Marital Status_Single.
- Customer with Passport is more likehood to buy it.
- If customer Designation is Executive higher are the chances for them to buy a travel package.
- Customer with Marital Status Single or Unmaried are also more likehood to buy the travel package.
- Model improvement can be done with more Data points, more informations about the customer, more data points to compare patterns and make better predictions.

Recommendations

- We should understand the needs of each profile and provide the best travel package thinking in all points that will made the difference and Increase the customer satisfaction.

- Create strategies to best use our 5 to 10 time of pitch with our customer, considering this is our mean time spend with them before they say yes or no.
- Third and Fourth follow-ups is the KEY to convert this customer as a buyer, to do so we need to train our team to do the best presentation of the travel package and identify customer necessities.
- A new travel package BUSINESS should be considering, to target companies that have their employees traveling a lot.
- King and Super Delux need to be reviewed and improved to target more buyers.