Overview:

Currently, there are 5 types of packages the company is offering - Basic, Standard, Deluxe, Super Deluxe, King. Looking at the data of the last year, we observed that 18% of the customers purchased the packages. However, the marketing cost was quite high because customers were contacted at random without looking at the available information. The company is now planning to launch a new product i.e. Wellness Tourism Package. Wellness Tourism is defined as Travel that allows the traveler to maintain, enhance or kick-start a healthy lifestyle, and support or increase one's sense of well-being. However, this time company wants to harness the available data of existing and potential customers to make the marketing expenditure more efficient.Analyze the customers' data and information to provide recommendations to the Policy Maker and Marketing Team and also build a model to predict the potential customer who is going to purchase the newly introduced travel package.

Objective:

To predict which customer is more likely to purchase the newly introduced travel package.

Data Description:

Customer details:

1. CustomerID: Unique customer ID
2. ProdTaken: Whether the customer has purchased a package or not (0: No, 1: Yes)
3. Age: Age of customer
4. TypeofContact: How customer was contacted (Company Invited or Self Inquiry)
5. 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
6. Occupation: Occupation of customer
7. Gender: Gender of customer
8. NumberOfPersonVisiting: Total number of persons planning to take the trip with the customer
9. PreferredPropertyStar: Preferred hotel property rating by customer
10. MaritalStatus: Marital status of customer
11. NumberOfTrips: Average number of trips in a year by customer
12. Passport: The customer has a passport or not (0: No, 1: Yes)
13. OwnCar: Whether the customers own a car or not (0: No, 1: Yes)
14. NumberOfChildrenVisiting: Total number of children with age less than 5 planning to take the trip with the customer
15. Designation: Designation of the customer in the current organization
16. MonthlyIncome: Gross monthly income of the customer

Customer interaction data:

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

Import necesscary libraries

In [4]:

```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')


from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
# Libtune to tune model, get different metric scores
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_scor
from sklearn.model_selection import GridSearchCV
```

In [5]:
```
from xgboost import XGBClassifier
```

Read the dataset

In [6]:
```
data = pd.read_excel("Tourism.xlsx",sheet_name='Tourism', index_col = None)
```

In [7]:
```
data.head()
```

Out[7]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPersonVisiting	NumberOfFollowups	F
0	200000	1	41.0	Self Enquiry	3	6.0	Salaried	Female	3	3.0	
1	200001	0	49.0	Company Invited	1	14.0	Salaried	Male	3	4.0	
2	200002	1	37.0	Self Enquiry	1	8.0	Free Lancer	Male	3	4.0	
3	200003	0	33.0	Company Invited	1	9.0	Salaried	Female	2	3.0	
4	200004	0	NaN	Self Enquiry	1	8.0	Small Business	Male	2	3.0	

In [8]:
```
data.drop('CustomerID', inplace = True, axis = 1)
```

In [9]:
```
data.head()
```

Out[9]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPersonVisiting	NumberOfFollowups	ProductPitche
0	1	41.0	Self Enquiry	3	6.0	Salaried	Female	3	3.0	Delux
1	0	49.0	Company Invited	1	14.0	Salaried	Male	3	4.0	Delux
2	1	37.0	Self Enquiry	1	8.0	Free Lancer	Male	3	4.0	Basi
3	0	33.0	Company Invited	1	9.0	Salaried	Female	2	3.0	Basi
4	0	NaN	Self Enquiry	1	8.0	Small Business	Male	2	3.0	Basi

In [10]:
```
data.shape #columns and rows
```

Out[10]: (4888, 19)

In [11]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ProdTaken                4888 non-null   int64  
 1   Age                      4662 non-null   float64
 2   TypeofContact            4863 non-null   object 
 3   CityTier                 4888 non-null   int64  
 4   DurationOfPitch          4637 non-null   float64
 5   Occupation               4888 non-null   object 
 6   Gender                   4888 non-null   object 
 7   NumberOfPersonVisiting   4888 non-null   int64  
 8   NumberOfFollowups        4843 non-null   float64
 9   ProductPitched           4888 non-null   object 
 10  PreferredPropertyStar    4862 non-null   float64
 11  MaritalStatus            4888 non-null   object 
 12  NumberOfTrips            4748 non-null   float64
 13  Passport                 4888 non-null   int64  
 14  PitchSatisfactionScore   4888 non-null   int64  
 15  OwnCar                   4888 non-null   int64  
 16  NumberOfChildrenVisiting 4822 non-null   float64
 17  Designation              4888 non-null   object 
 18  MonthlyIncome            4655 non-null   float64
dtypes: float64(7), int64(6), object(6)
memory usage: 725.7+ KB
```

Obseravtions:

1. A lot of the data set are of non-numeric type.
2. For EDA purposes the data set is fine to use
3. For modelling purposes the data set will have to be in int/float form.

In [12]:
```
data.nunique()
```

Out[12]:
```
ProdTaken                      2
Age                           44
TypeofContact                  2
CityTier                       3
DurationOfPitch               34
Occupation                     4
Gender                         3
NumberOfPersonVisiting         5
NumberOfFollowups              6
ProductPitched                 5
PreferredPropertyStar          3
MaritalStatus                  4
NumberOfTrips                 12
Passport                       2
PitchSatisfactionScore         5
OwnCar                         2
NumberOfChildrenVisiting       4
Designation                    5
MonthlyIncome               2475
dtype: int64
```

In [14]:
```
replace = {'Gender': {'Fe Male': 'Female'}} #There are only two geneders mentioned in the variable set, trasfomri
data = data.replace(replace)
data.nunique()
```

Out[14]:
```
ProdTaken                      2
Age                           44
TypeofContact                  2
CityTier                       3
DurationOfPitch               34
Occupation                     4
Gender                         2
NumberOfPersonVisiting         5
NumberOfFollowups              6
ProductPitched                 5
PreferredPropertyStar          3
MaritalStatus                  4
NumberOfTrips                 12
Passport                       2
PitchSatisfactionScore         5
OwnCar                         2
NumberOfChildrenVisiting       4
Designation                    5
MonthlyIncome               2475
dtype: int64
```

In [15]:
```
data.isnull().sum().sort_values(ascending=False) #sum of all the null values per variable
```

Out[15]:
```
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
```

Observations:

1. There are significant number of null values missing in 8 of the variables.
2. Since most of the variables are of categorical type they can be categorized 'as_missing'
3. Rest of the missing values can be imputed with median

In [16]:
```
category = ['ProdTaken','TypeofContact','CityTier','Occupation', 'Gender','NumberOfPersonVisiting','NumberOfFollo
print(len(category))
for i in category:
   data[i] = data[i].astype('category')
```

```
16
```

In [17]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   ProdTaken                4888 non-null   category
 1   Age                      4662 non-null   float64
 2   TypeofContact            4863 non-null   category
 3   CityTier                 4888 non-null   category
 4   DurationOfPitch          4637 non-null   float64
 5   Occupation               4888 non-null   category
 6   Gender                   4888 non-null   category
 7   NumberOfPersonVisiting   4888 non-null   category
 8   NumberOfFollowups        4843 non-null   category
 9   ProductPitched           4888 non-null   category
 10  PreferredPropertyStar    4862 non-null   category
 11  MaritalStatus            4888 non-null   category
 12  NumberOfTrips            4748 non-null   category
 13  Passport                 4888 non-null   category
 14  PitchSatisfactionScore   4888 non-null   category
 15  OwnCar                   4888 non-null   category
 16  NumberOfChildrenVisiting 4822 non-null   category
 17  Designation              4888 non-null   category
 18  MonthlyIncome            4655 non-null   float64
dtypes: category(16), float64(3)
memory usage: 193.7 KB
```

In [18]:
```
median_imputation_values = ['DurationOfPitch', 'MonthlyIncome','Age' ]
for i in median_imputation_values: #converting missing values with median
    data[i].fillna(data[i].median(), inplace=True)
```

In [19]:
```
data['TypeofContact'] = data['TypeofContact'].astype(str).replace('nan', 'is_missing').astype('category') #conver
data['NumberOfFollowups'] = data['NumberOfFollowups'].astype(str).replace('nan', 'is_missing').astype('category')
data['PreferredPropertyStar'] = data['PreferredPropertyStar'].astype(str).replace('nan', 'is_missing').astype('ca
data['NumberOfChildrenVisiting'] = data['NumberOfChildrenVisiting'].astype(str).replace('nan', 'is_missing').asty
data['NumberOfTrips'] = data['NumberOfTrips'].astype(str).replace('nan', 'is_missing').astype('category')
```

In [20]:
```
data.isnull().sum().sort_values(ascending = False) #sum of all null values
```

Out[20]:
```
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
```

In [21]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype   
---  ------                   --------------  -----   
 0   ProdTaken                4888 non-null   category
 1   Age                      4888 non-null   float64 
 2   TypeofContact            4888 non-null   category
 3   CityTier                 4888 non-null   category
 4   DurationOfPitch          4888 non-null   float64 
 5   Occupation               4888 non-null   category
 6   Gender                   4888 non-null   category
 7   NumberOfPersonVisiting   4888 non-null   category
 8   NumberOfFollowups        4888 non-null   category
 9   ProductPitched           4888 non-null   category
 10  PreferredPropertyStar    4888 non-null   category
 11  MaritalStatus            4888 non-null   category
 12  NumberOfTrips            4888 non-null   category
 13  Passport                 4888 non-null   category
 14  PitchSatisfactionScore   4888 non-null   category
 15  OwnCar                   4888 non-null   category
 16  NumberOfChildrenVisiting 4888 non-null   category
 17  Designation              4888 non-null   category
 18  MonthlyIncome            4888 non-null   float64 
dtypes: category(16), float64(3)
memory usage: 194.3 KB
```

In [22]:
```
continous_columns = ['Age','DurationOfPitch', 'MonthlyIncome']
```

In [23]:
```
data.describe().T
```

Out[23]:

	count	mean	std	min	25%	50%	75%	max
Age	4888.0	37.547259	9.104795	18.0	31.0	36.0	43.00	61.0
DurationOfPitch	4888.0	15.362930	8.316166	5.0	9.0	13.0	19.00	127.0
MonthlyIncome	4888.0	23559.179419	5257.862921	1000.0	20485.0	22347.0	25424.75	98678.0

In [24]:
```
data.describe(include = 'category').T
```

Out[24]:

	count	unique	top	freq
ProdTaken	4888	2	0	3968
TypeofContact	4888	3	Self Enquiry	3444
CityTier	4888	3	1	3190
Occupation	4888	4	Salaried	2368
Gender	4888	2	Male	2916
NumberOfPersonVisiting	4888	5	3	2402
NumberOfFollowups	4888	7	4.0	2068
ProductPitched	4888	5	Basic	1842
PreferredPropertyStar	4888	4	3.0	2993
MaritalStatus	4888	4	Married	2340
NumberOfTrips	4888	13	2.0	1464
Passport	4888	2	0	3466
PitchSatisfactionScore	4888	5	3	1478
OwnCar	4888	2	1	3032
NumberOfChildrenVisiting	4888	5	1.0	2080
Designation	4888	5	Executive	1842

Observations on the data set:

1. There is a huge range in the Age variable
2. Duration of pitch also has a huge range in the variable.
3. Montlyhy income has a huge range in the variable and the mean is close to the 75% percentile.

4. Most of the categorical data have 2 to 5 different types of categories, except Number of trips.
5. Number of trips was considered under categorical data even though it has 13 different categories, as the categories are measured and are not on a huge scale.

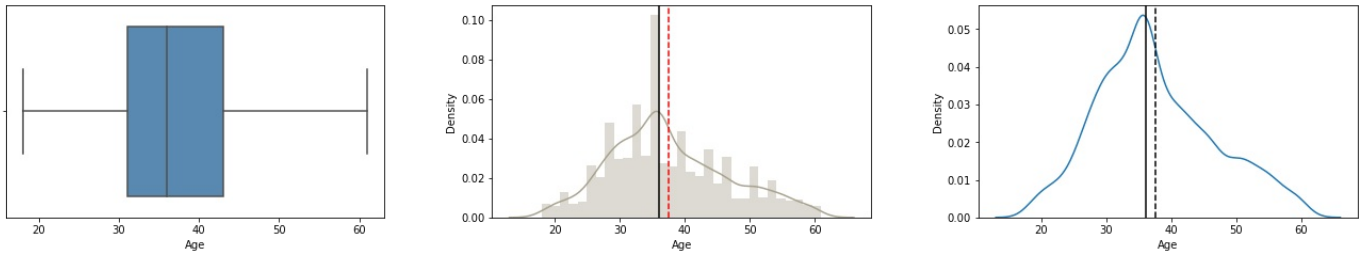
EDA

In [25]:
```
def histogram_boxplot(feature):
    """ Boxplot and histogram combined
    feature: 1-d feature array
    """
    figure, (ax_box2, ax_hist2, ax_hist3) = plt.subplots(
        nrows = 1, ncols=3,# Number of rows of the subplot grid= 2
        figsize  = (20,5)) # creating the 2 subplots
    figure.tight_layout(pad = 7)
    sns.boxplot(x = feature,ax=ax_box2, color = '#4B8BBE', orient = 'v') # boxplot will be created
    sns.distplot(feature, kde=True, ax=ax_hist2, color = '#a9a38f') # For histogram
    sns.distplot(feature, kde= True, ax=ax_hist3, hist = False) #Making an outline of the histogram
    ax_hist2.axvline(np.mean(feature), color='r', linestyle='--') # Add mean to the histogram
    ax_hist2.axvline(np.median(feature), color='black', linestyle='-') # Add median to the histogram
    ax_hist3.axvline(np.mean(feature), color = 'black', linestyle = '--') #Adding mean to second histogram
    ax_hist3.axvline(np.median(feature), color='black', linestyle='-') #Adding median to second histogram
```

Univariate Analysis

Observations of Age

In [26]:
```
histogram_boxplot(data[continous_columns[0]])
```

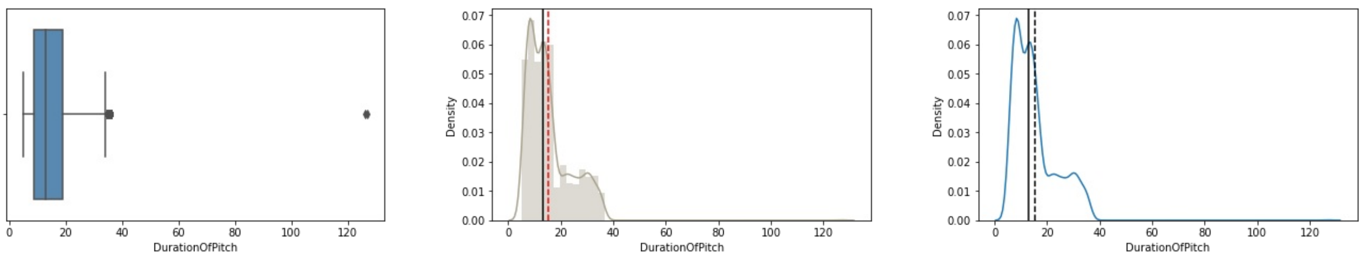


Observations:

1. The variable does not have any outliers
2. Pretty overal balanced data
3. The max age is around 61 years old.

Observations on Duration of Pitch

In [27]:
```
histogram_boxplot(data[continous_columns[1]])
```

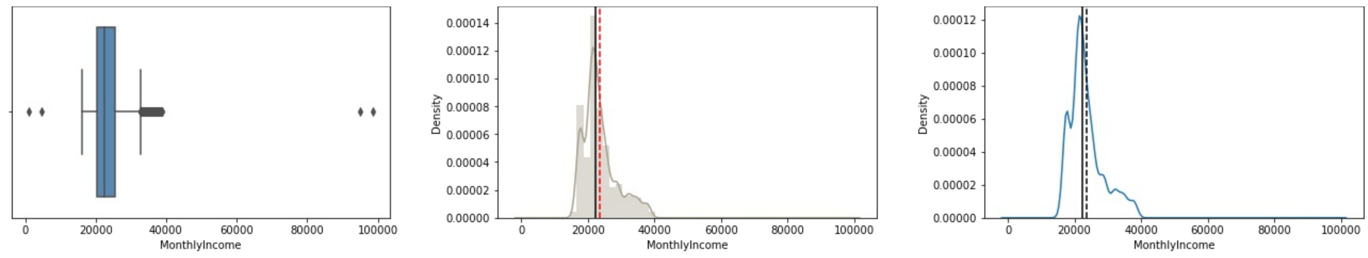


Observations:

1. There appear to be 2 outliers on the highersidde
2. The graph is pretty right-scaled
3. Log transformation may be applied to DUration of Pitch

Observations on Monlthy Income

In [28]:
```
histogram_boxplot(data[continous_columns[2]])
```



Observations:

1. A lot of outliers are visible.
2. The graph is right-skewed
3. Log transformation may be applied to monlthy income as well.

Outlier treatments

In [29]:

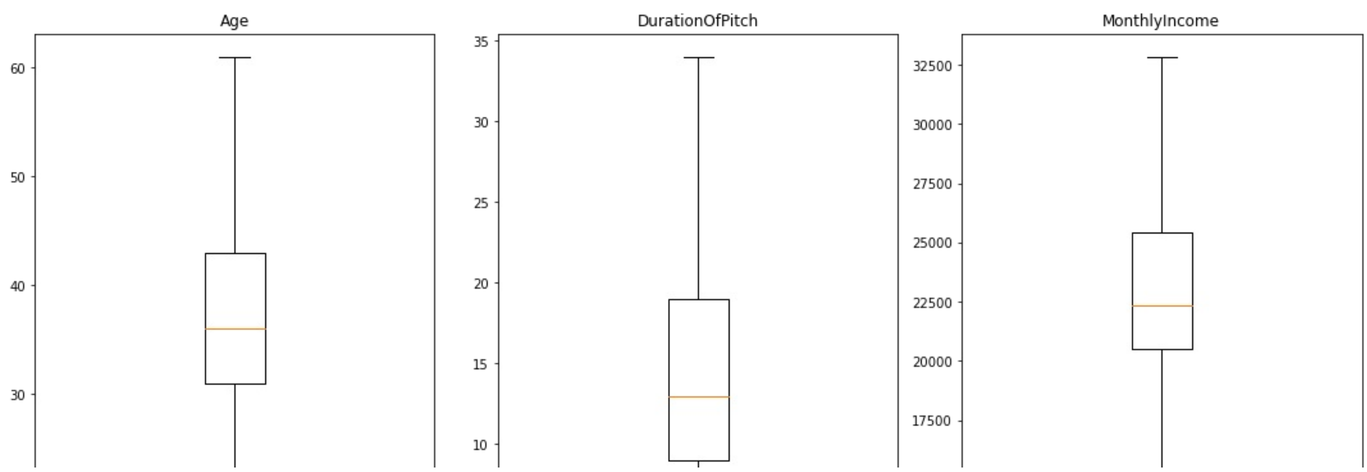
```
# Let's treat outliers by flooring and capping
def treat_outliers(df, col):
    """
    treats outliers in a variable
    col: str, name of the numerical variable
    df: dataframe
    col: name of the column
    """
    Q1 = df[col].quantile(0.25)  # 25th quantile
    Q3 = df[col].quantile(0.75)  # 75th quantile
    IQR = Q3 - Q1
    Lower_Whisker = Q1 - 1.5 * IQR
    Upper_Whisker = Q3 + 1.5 * IQR

    # all the values smaller than Lower_Whisker will be assigned the value of Lower_Whisker
    # all the values greater than Upper_Whisker will be assigned the value of Upper_Whisker
    df[col] = np.clip(df[col], Lower_Whisker, Upper_Whisker)

    return df


def treat_outliers_all(df, col_list):
    """
    treat outlier in all numerical variables
    col_list: list of numerical variables
    df: data frame
    """
    for c in col_list:
        df = treat_outliers(df, c)

    return df
```

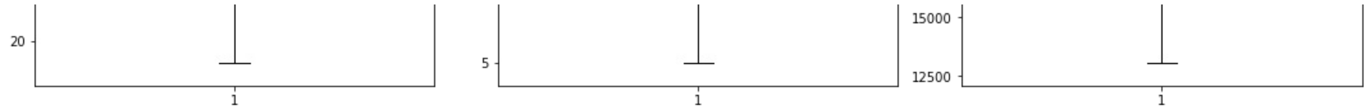
In [30]:

```
numerical_col = data.select_dtypes(include=np.number).columns.tolist()#converting numerical cols
data = treat_outliers_all(data, numerical_col) #treating outliers
```

In [31]:

```
plt.figure(figsize=(20, 30))

for i, variable in enumerate(numerical_col): #boxplots subplots
    plt.subplot(5, 4, i + 1)
    plt.boxplot(data[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```





Observations:

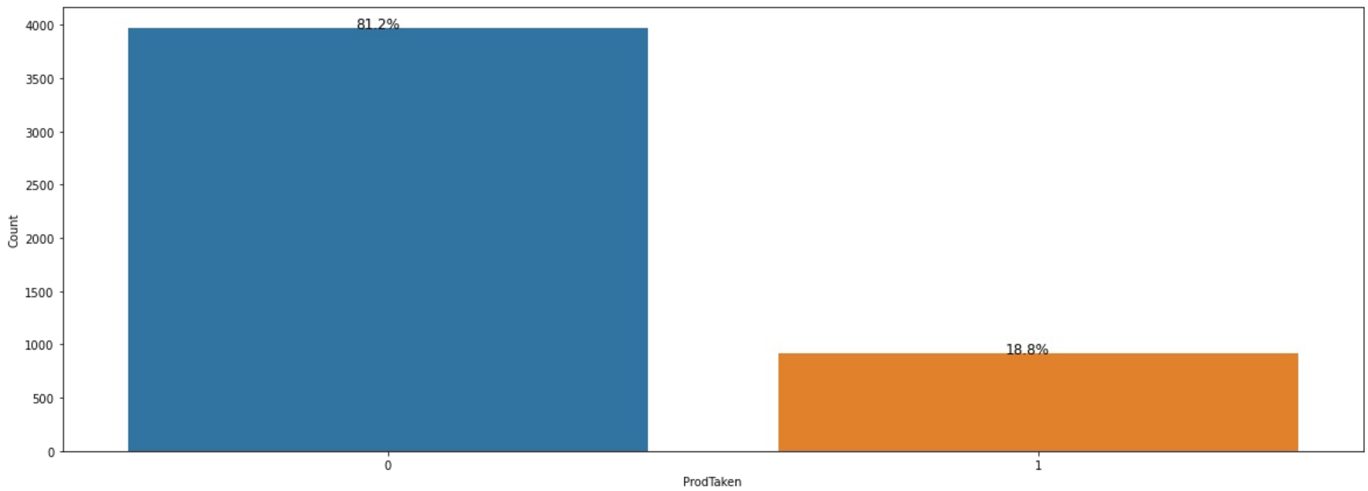
1. No visible outliers after the outlier treatment
2. Outlier treatment applied on the continous Variables Age, Duration of Pitch adn Monthly Income

Categorical Variables

In [32]:
```
def bar_perc(plot, feature):
    '''
    plot
    feature: 1-d categorical feature array
    '''
    total = len(feature) # length of the column
    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
```

Observation on ProdTaken

In [33]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[0]]) #count plot for Name
plt.xlabel(category[0])
plt.ylabel('Count')
bar_perc(ax,data[category[0]])
```

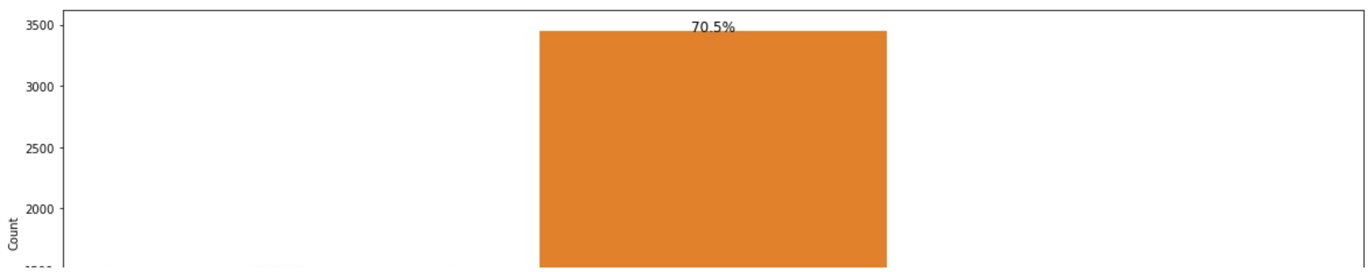


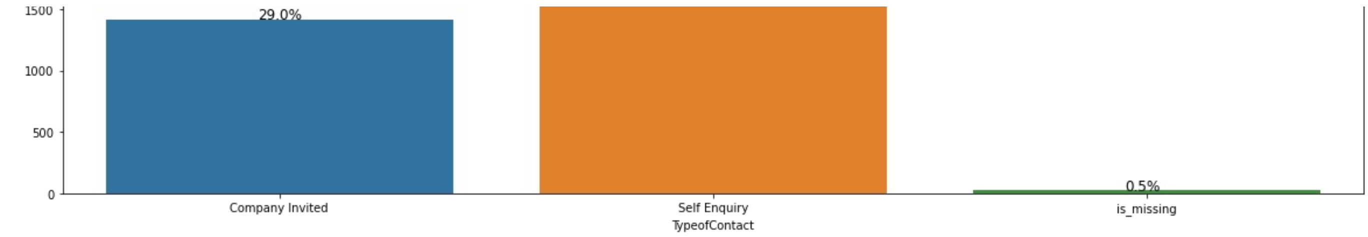
Observation:

1. 18.8% of clients did take the product 2.81.2 % of the clients did not take the product.

Observation Type Of Contact

In [34]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[1]]) #count plot for Name
plt.xlabel(category[1])
plt.ylabel('Count')
bar_perc(ax,data[category[1]])
```



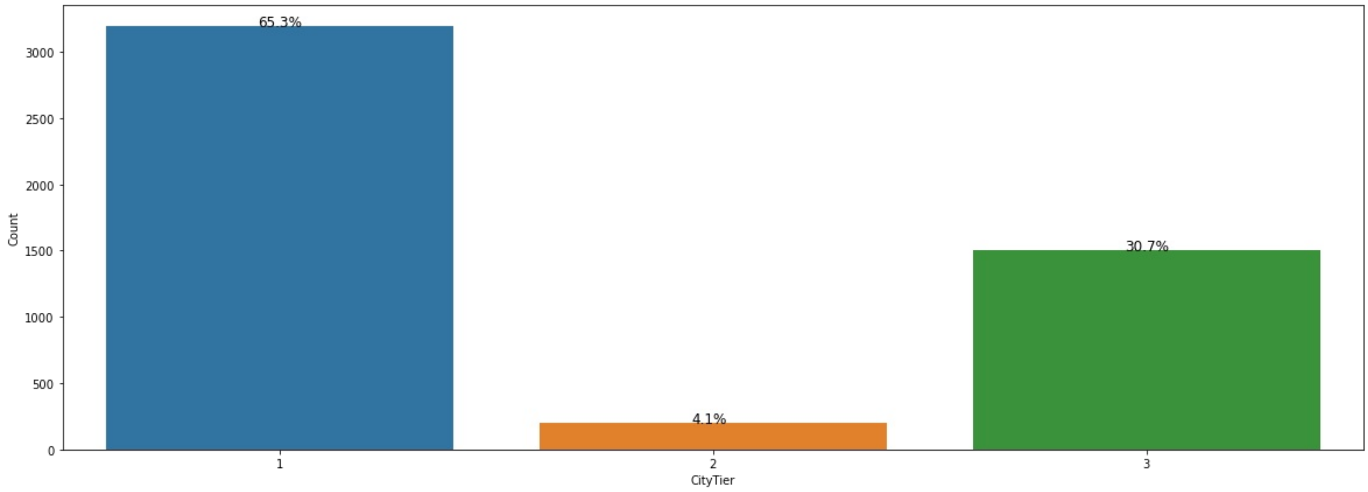


Observations:

1. Self-enquiry are 70.5% of the clients
2. 29.0% are company invited clients

Observation on CityTier

In [35]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[2]]) #count plot for Name
plt.xlabel(category[2])
plt.ylabel('Count')
bar_perc(ax,data[category[2]])
```

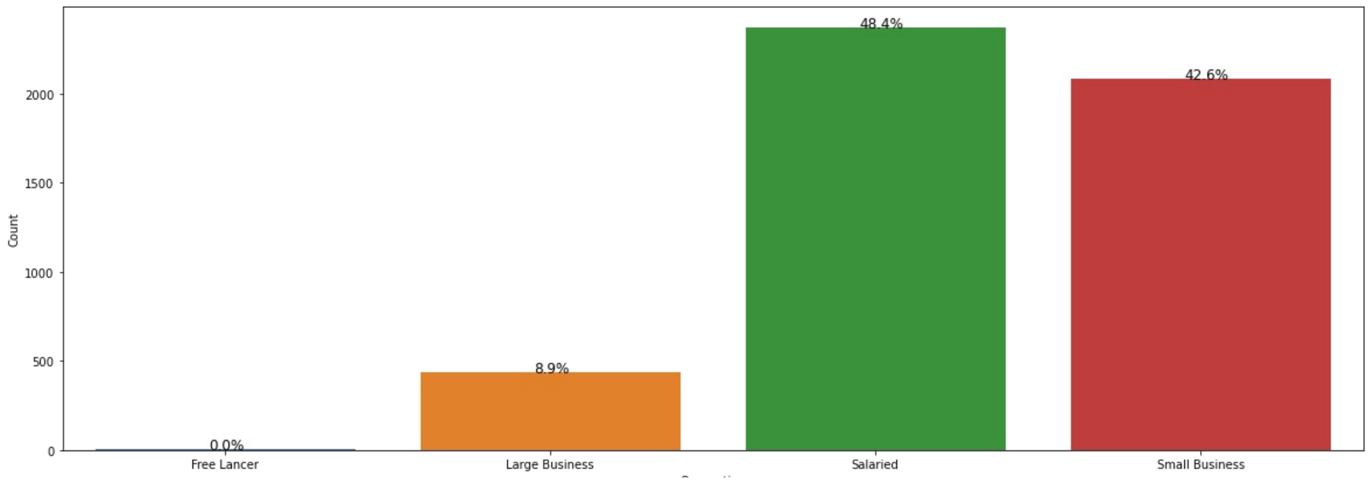


Observations:

1. City tier 1 are 65.3%
2. City Tier 2 is 4.15%
3. City tier 3 is 30.7%

Observation on Ocupation

In [36]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[3]]) #count plot for Name
plt.xlabel(category[3])
plt.ylabel('Count')
bar_perc(ax,data[category[3]])
```



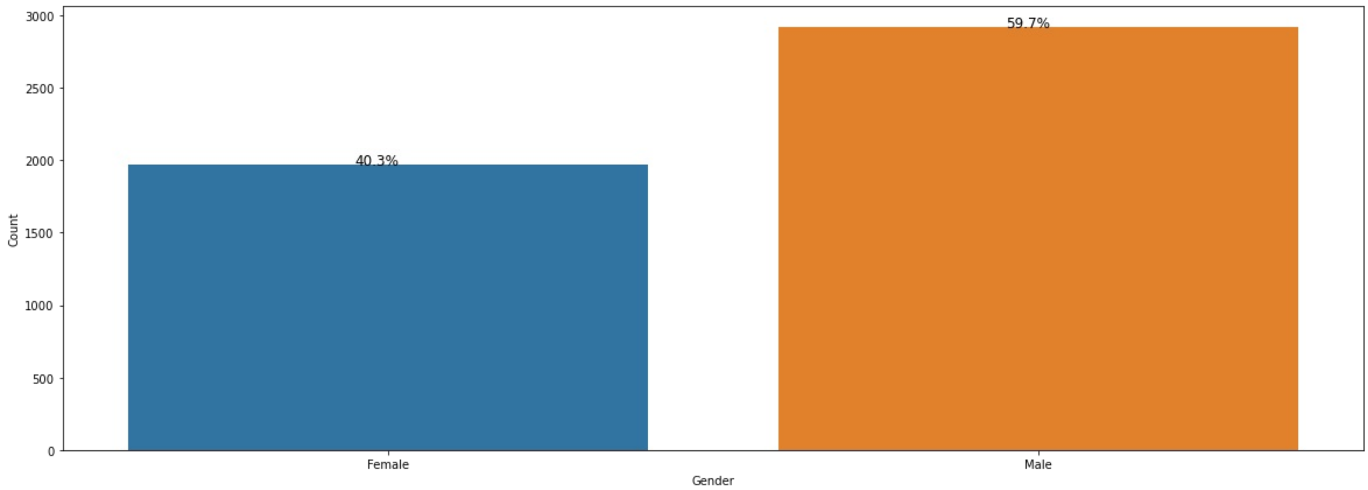
Occupation

Observations:

1. Ocupation of clients about 48.4% are Salaried
2. Large business are about 8.9% clients
3. Small business are about 42.6%

Observation on Gender

In [37]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[4]]) #count plot for Name
plt.xlabel(category[4])
plt.ylabel('Count')
bar_perc(ax,data[category[4]])
```

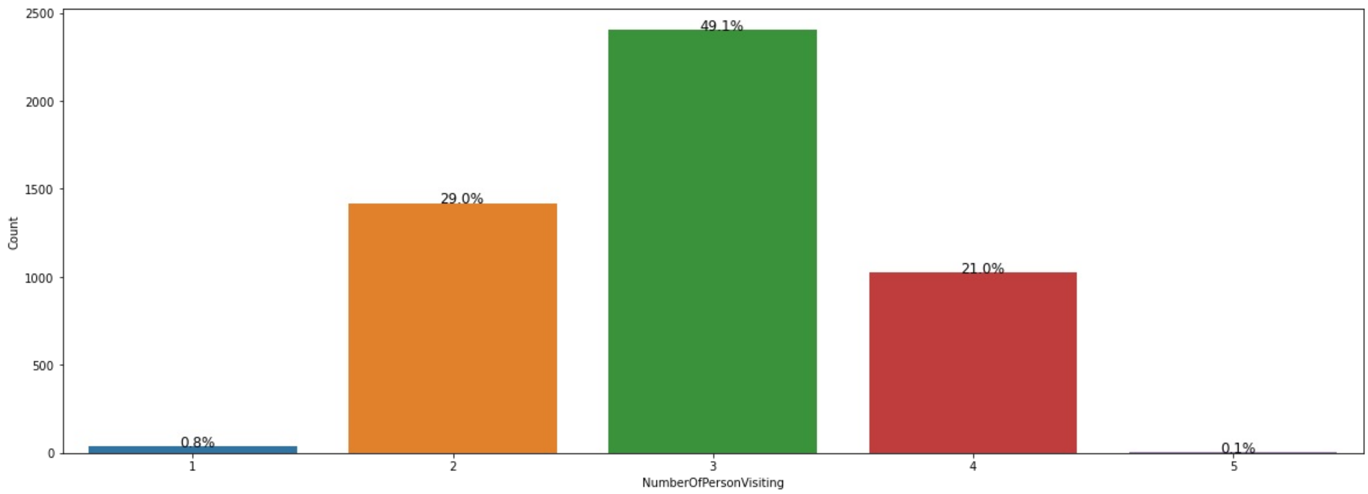


Observations:

1. 40.3% of clients are female
2. 59.7% of the clients are male.

Observation on Number of Person Visiting

In [38]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[5]]) #count plot for Name
plt.xlabel(category[5])
plt.ylabel('Count')
bar_perc(ax,data[category[5]])
```

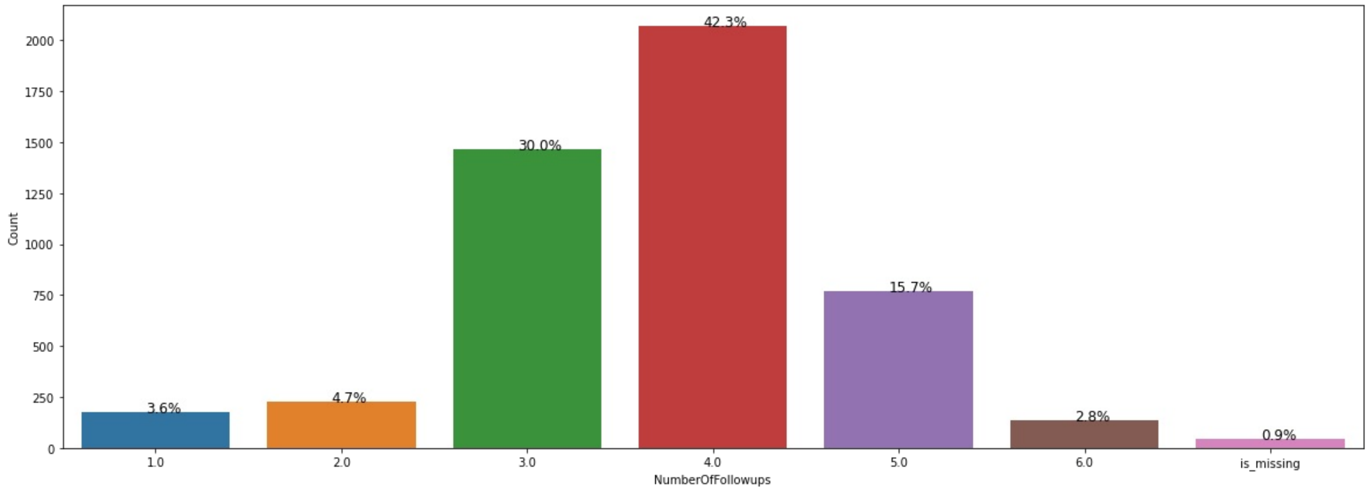


Observations:

1. Highest Number of persons visitng are 3 with 49.1%
2. Lowest number of persons visitng are 5 followed by 1 with 0.1% and 0.8% respectively

Observations on Number of Followups

In [39]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[6]]) #count plot for Name
plt.xlabel(category[6])
plt.ylabel('Count')
bar_perc(ax,data[category[6]])
```

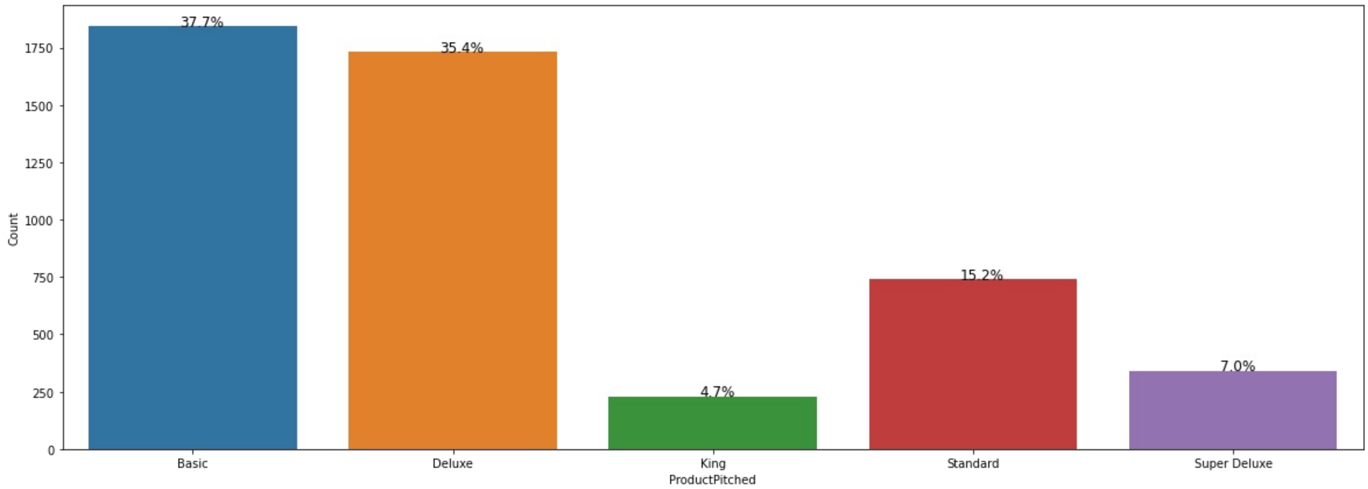


Observations:

1. Highest number of followups are 4, 3 and 2 with 42.3%, 30% and 15.7% respectively
2. Lowest number of follows are 6 and 1 with 2.8% and 3.6% respectively.

Observations on Product Pitched

In [40]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[7]]) #count plot for Name
plt.xlabel(category[7])
plt.ylabel('Count')
bar_perc(ax,data[category[7]])
```

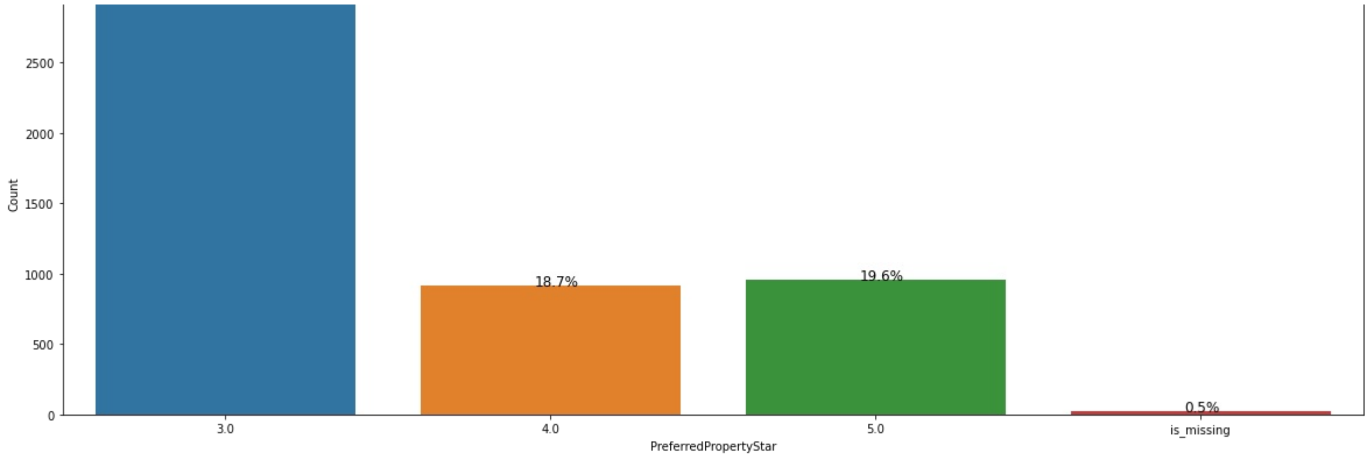


Observations:

1. Most type of product pitched are basic and deluxe, with 37.7% and 35.4% respectively.
2. Least type of product pitched is the king and super deluxe with 4.7% and 7%.

Observations on Preffered Property Star

In [41]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[8]]) #count plot for Name
plt.xlabel(category[8])
plt.ylabel('Count')
bar_perc(ax,data[category[8]])
```

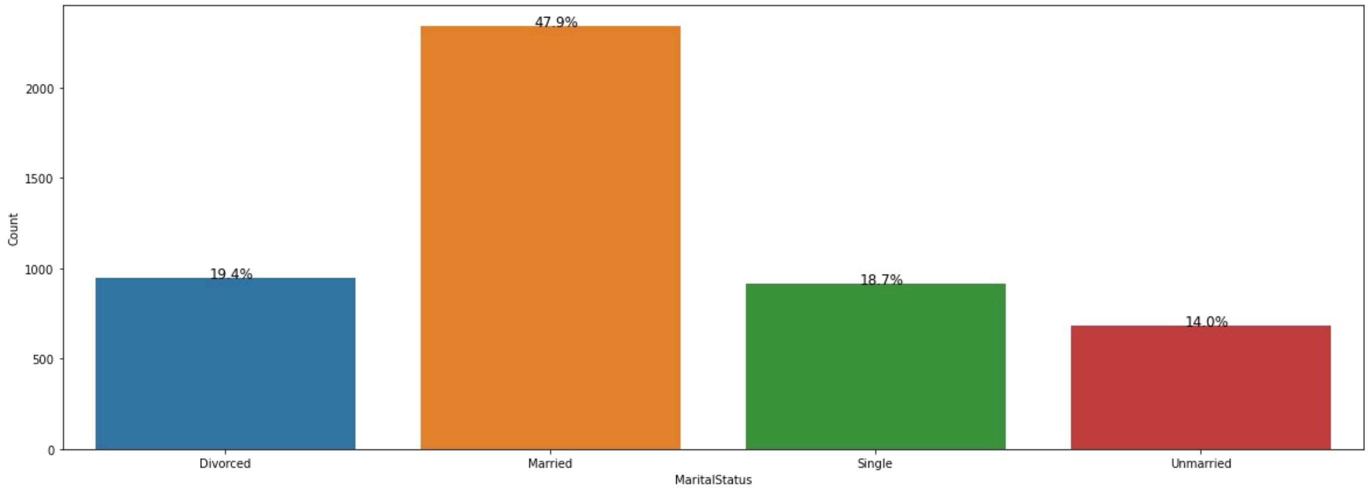


Observations:

1. Highest preffered property star is 3 with 6.12%
2. Least prefffered property star is 4 with 18.7%

Observations on Martial Status

In [42]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[9]]) #count plot for Name
plt.xlabel(category[9])
plt.ylabel('Count')
bar_perc(ax,data[category[9]])
```

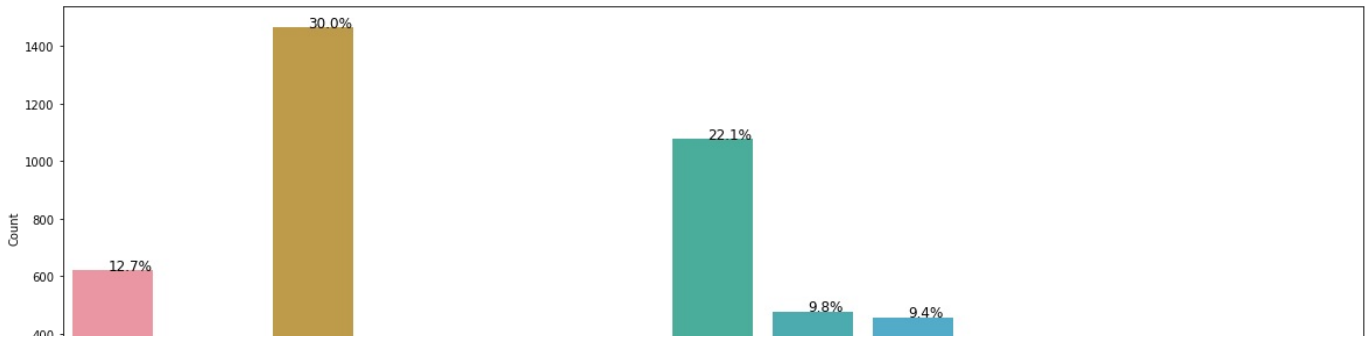


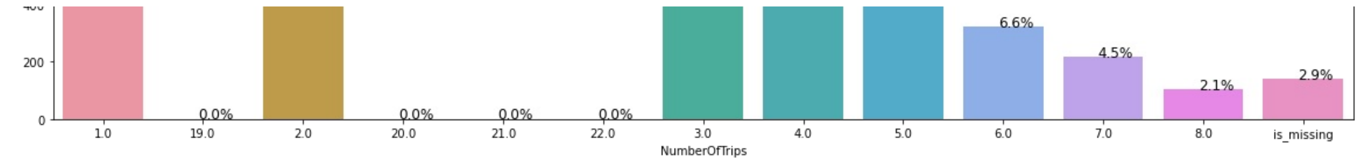
Observations:

1. Married clients are the most frequest with 47.9%
2. Least common clients are Unmarrier and Diworced with 14.0% and 19.4% respectively.

Observations on Number of trips

In [43]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[10]]) #count plot for Name
plt.xlabel(category[10])
plt.ylabel('Count')
bar_perc(ax,data[category[10]])
```



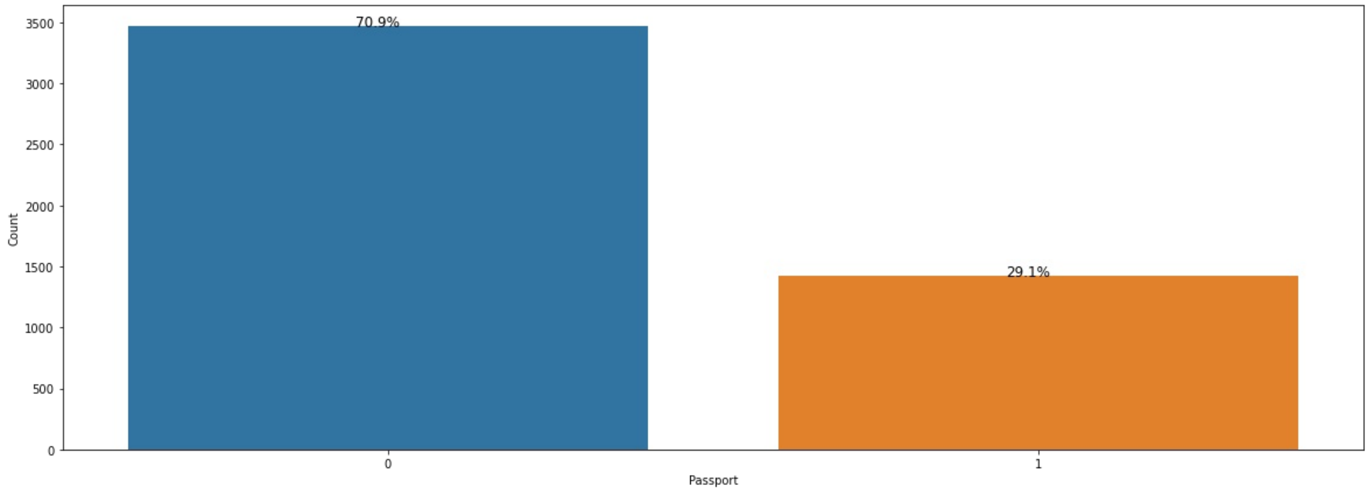


Observations:

1. Highest number of trips taken is2 with 30%
2. Least number of trips taken is 8 with 2.1%

Observations on Passport avaiablity

In [44]:
```python
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[11]]) #count plot for Name
plt.xlabel(category[11])
plt.ylabel('Count')
bar_perc(ax,data[category[11]])
```

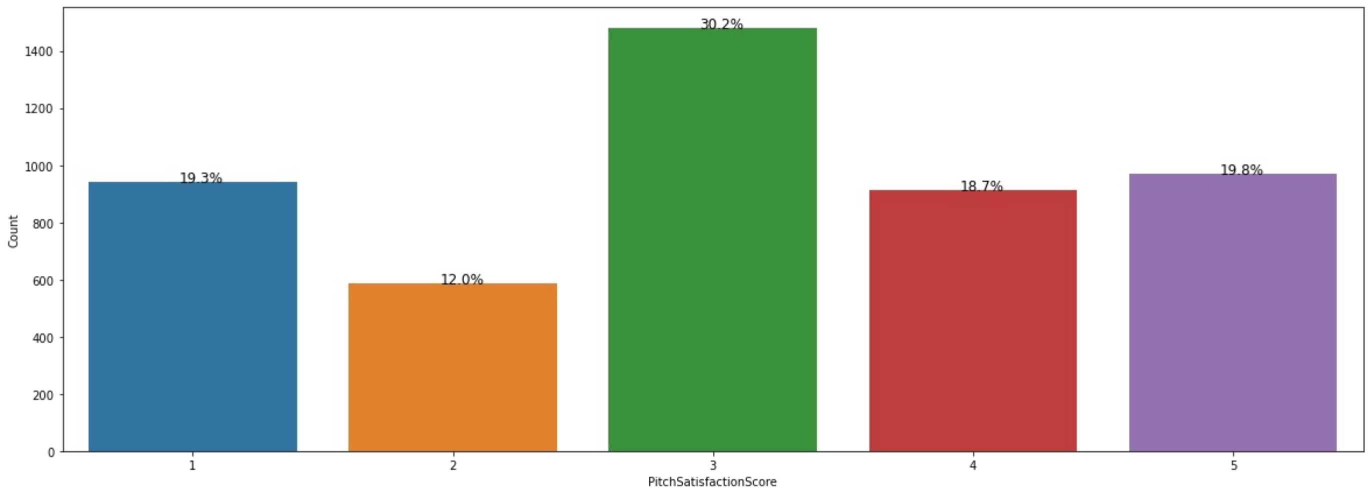


Observations:

1. 70.9% of the clients do not own passports
2. Only 29.1% of the clients own passports

Observation on Pitch satisfactory score

In [45]:
```python
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[12]]) #count plot for Name
plt.xlabel(category[12])
plt.ylabel('Count')
bar_perc(ax,data[category[12]])
```



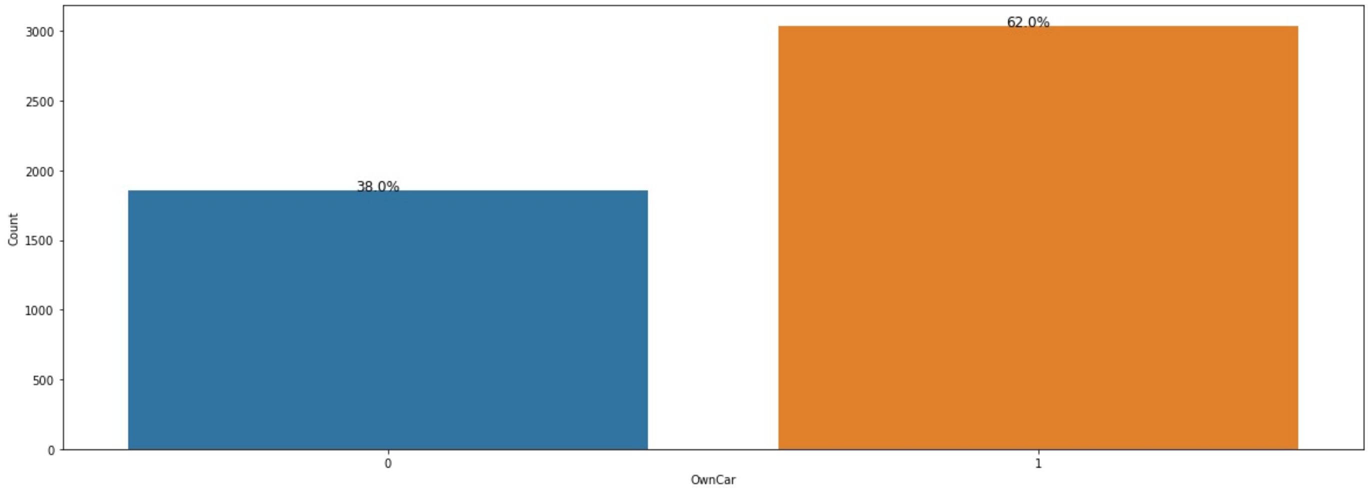
Observations:

1. Highest pitch satisfactory score is 3 with 30.2%
2. Lowest ptich satisfactory score is 2 with 12%

Observation on Owned car

In [46]:

```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[13]]) #count plot for Name
plt.xlabel(category[13])
plt.ylabel('Count')
bar_perc(ax,data[category[13]])
```



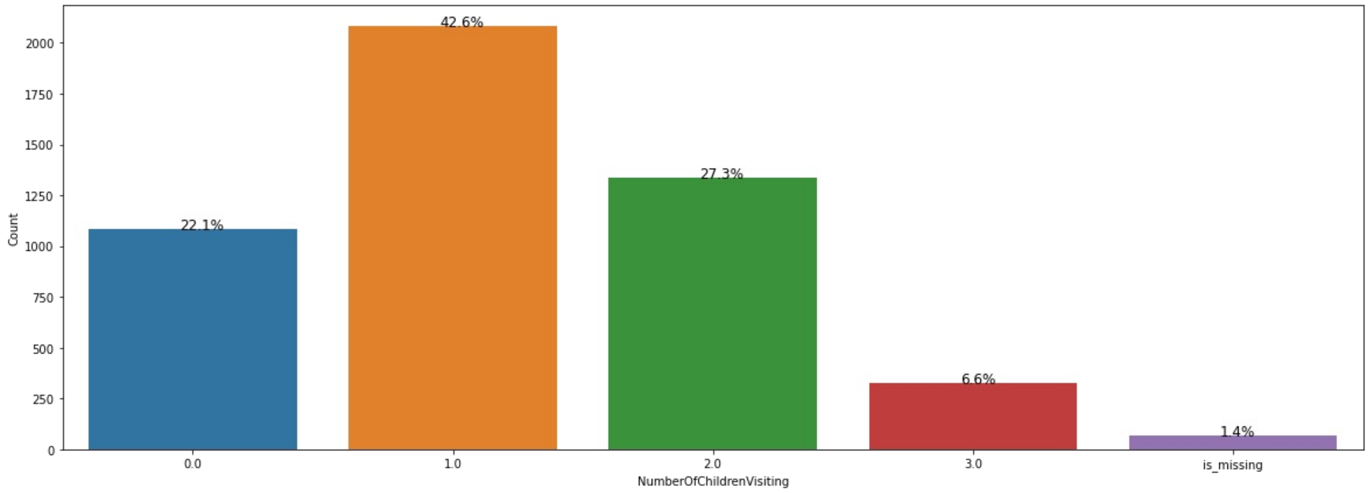
Observations:

1. 38% of the clients do not own a car
2. 62% of the cleints do own a car

Observations on Number of children visting

In [47]:

```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[14]]) #count plot for Name
plt.xlabel(category[14])
plt.ylabel('Count')
bar_perc(ax,data[category[14]])
```



Observations:

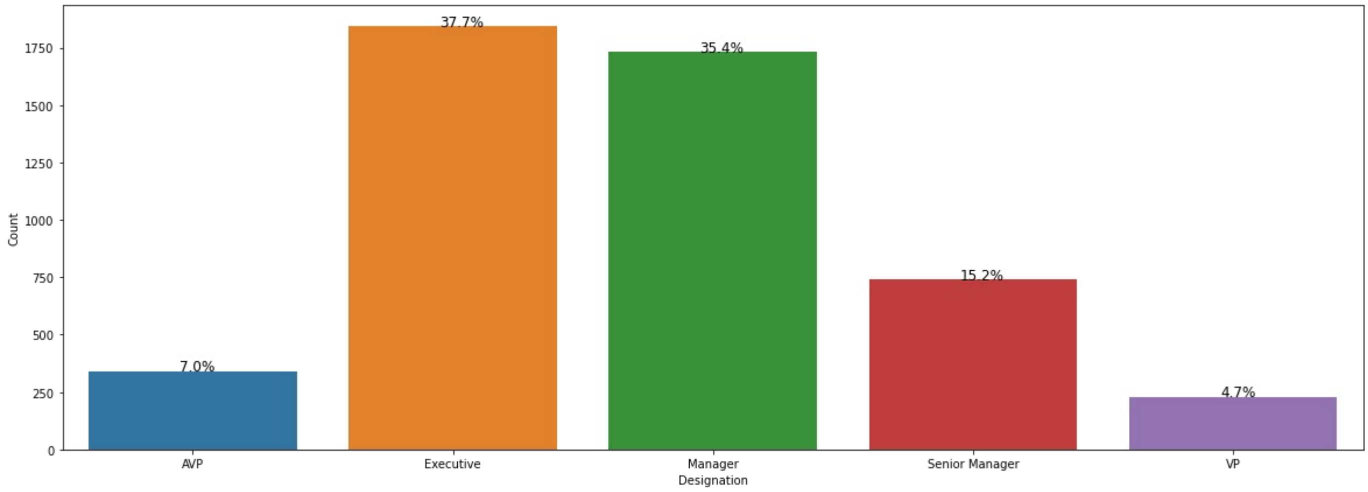
1. Lowest number of children visitng are 3 with 6.6%
2. Highest nunmber of children visiting are 1 with 42.6%

Observations on Designation

In [48]:

```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[category[15]]) #count plot for Name
plt.xlabel(category[15])
```

```
plt.ylabel('Count')
bar_perc(ax,data[category[15]])
```



Observations:

1. The least type of clients are VP with 4.7%
2. While the most common type of clients are executives and managers with 37.7% AND 35.4% respectively.

Bivariate Analysis

In [49]:

```
data.corr()
```

Out[49]:

	Age	DurationOfPitch	MonthlyIncome
Age	1.000000	-0.011045	0.480201
DurationOfPitch	-0.011045	1.000000	0.018051
MonthlyIncome	0.480201	0.018051	1.000000

In [50]:

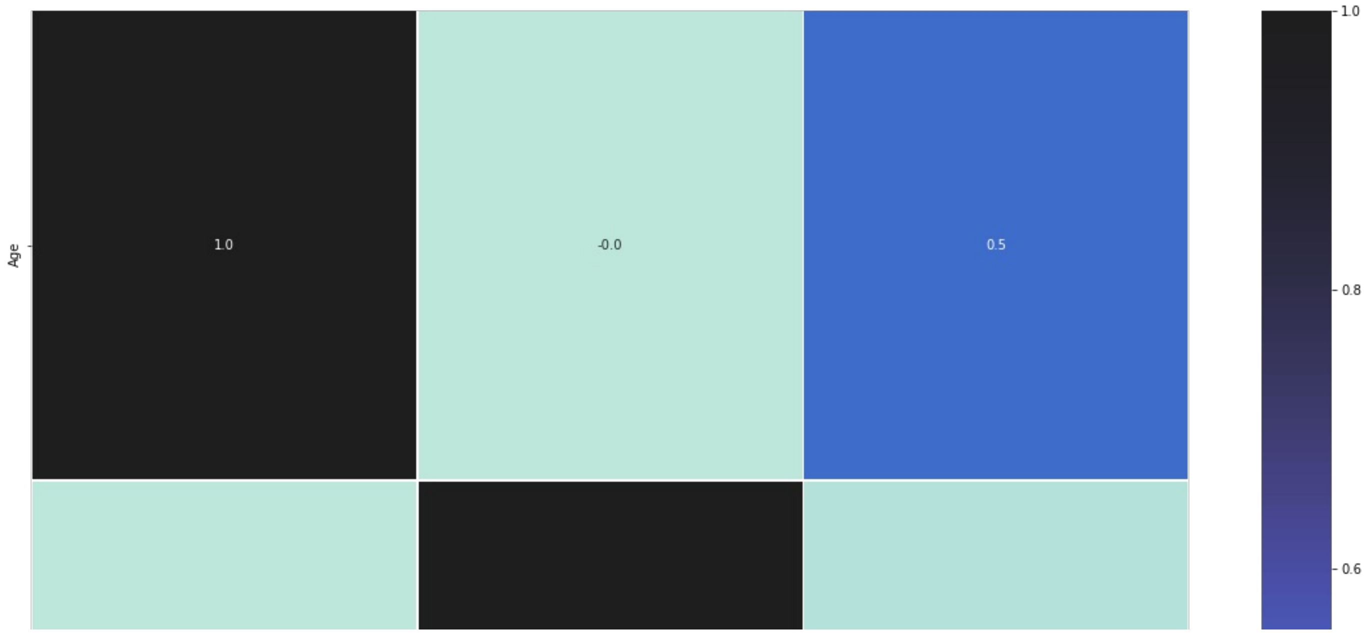
```
data.cov()
```

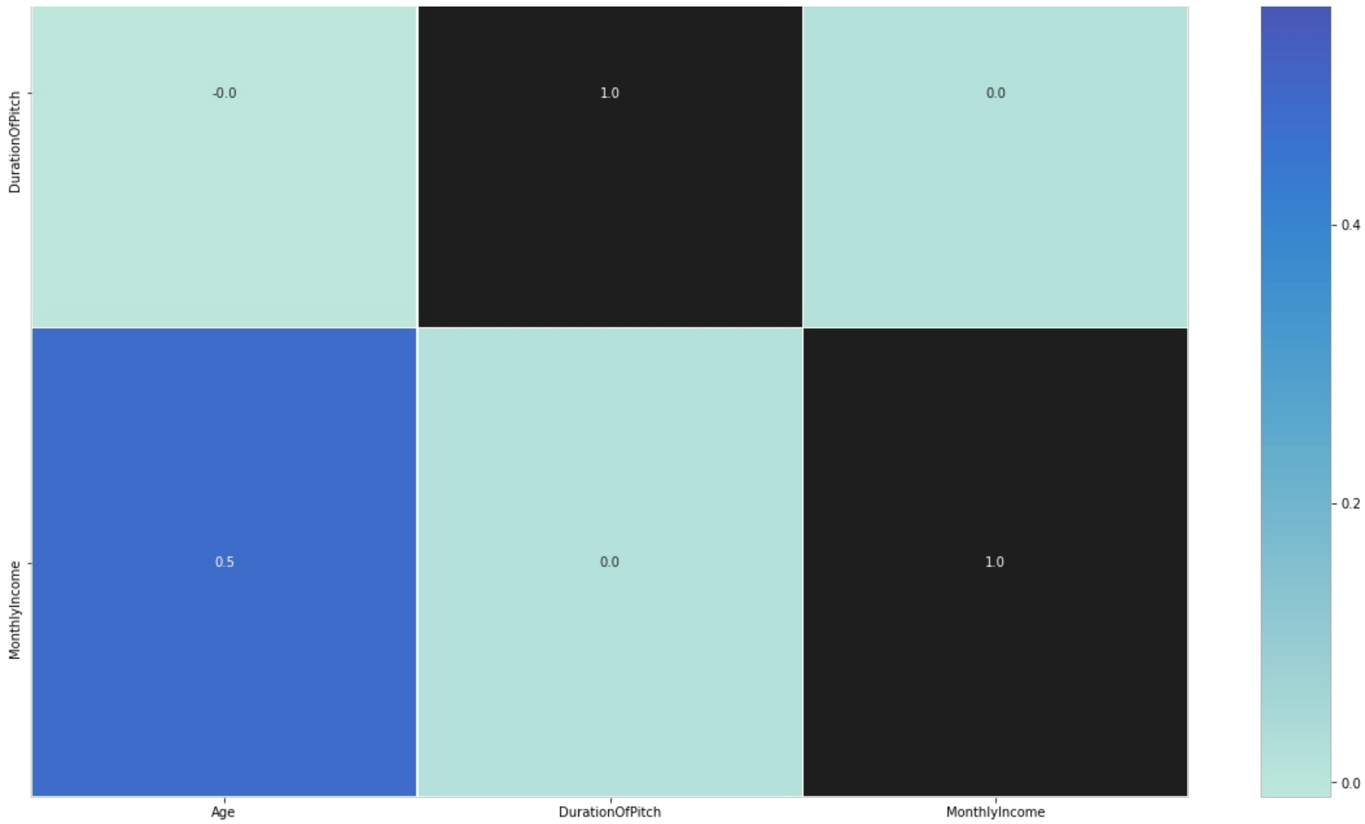
Out[50]:

	Age	DurationOfPitch	MonthlyIncome
Age	82.897296	-0.798305	1.973554e+04
DurationOfPitch	-0.798305	63.016926	6.468061e+02
MonthlyIncome	19735.539424	646.806059	2.037568e+07

In [51]:

```
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(), annot=True, linewidths=.5, fmt= '.1f', center = 1 )  # heatmap
plt.show()
```



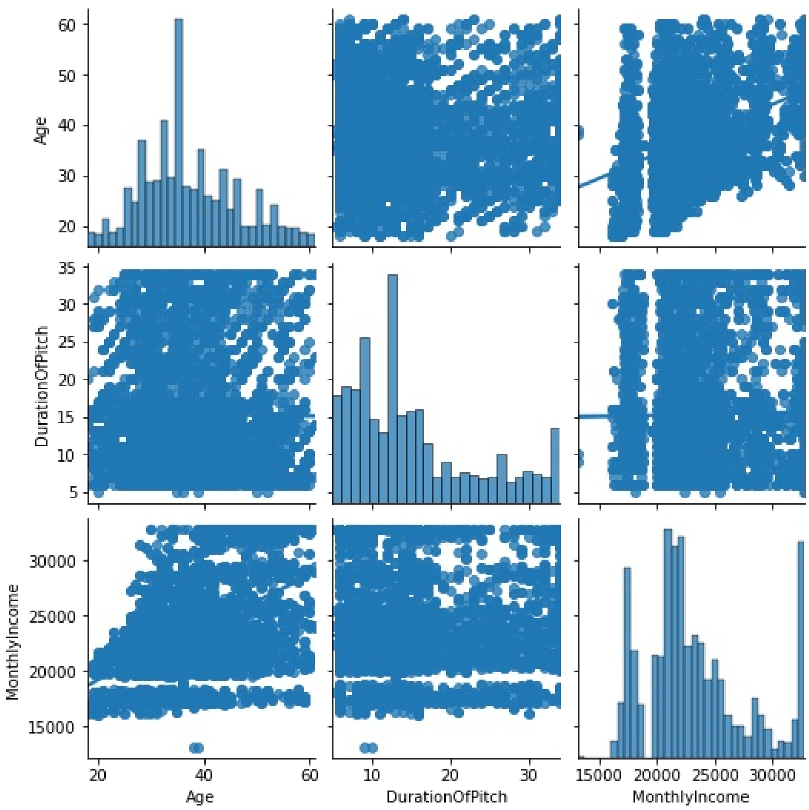


Observations:

1. Age and income show some correlation between each other but nothing too strong.
2. None of the variables really correlate well with each other

In [52]:
```
plt.figure(figsize = (20,20))
sns.pairplot(data = data[numerical_col], kind = 'reg') #pairplot
plt.show()
```

```
<Figure size 1440x1440 with 0 Axes>
```



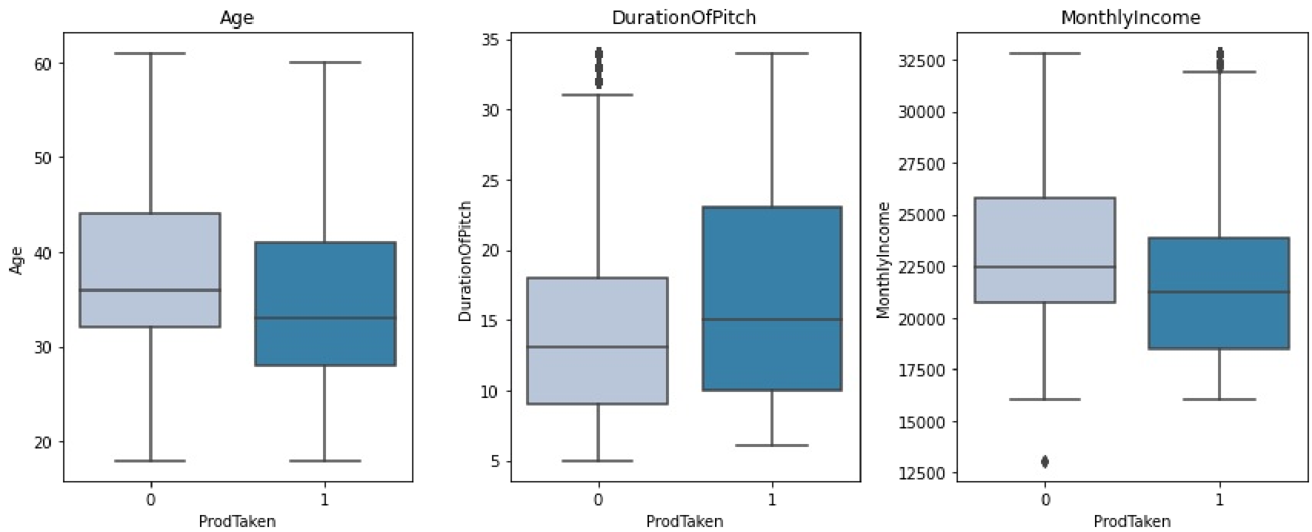
Observations:

1. As indicated in the correlation table and the heatmap, there seems to be no strong correlation between any variables

2. No trend can be identified from the scatter plots as there is no correlation between the variables

Product taken vs Continous Variables (Age, Duration of Pitch and Monthly Income)

In [99]:
```
cols = data[['Age', 'DurationOfPitch', 'MonthlyIncome']].columns.tolist()
plt.figure(figsize=(12,5))

for i, variable in enumerate(cols):
                     plt.subplot(1,3,i+1)
                     sns.boxplot(data['ProdTaken'],data[variable],palette="PuBu")
                     plt.tight_layout()
                     plt.title(variable)
plt.show()
```



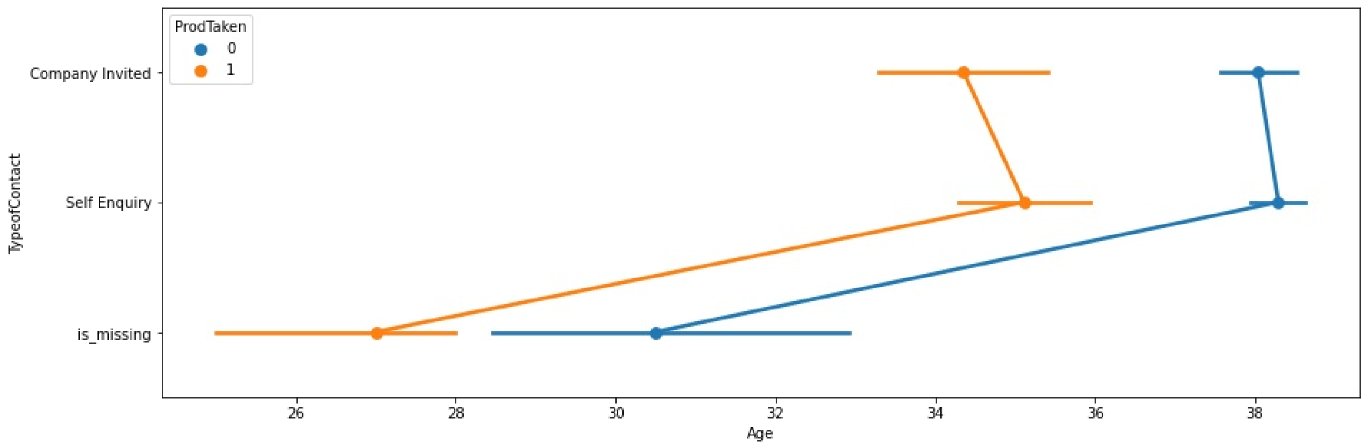
Observations:

1. The average age for the clients that did not take the product are higher than the clients that did take the product.
2. The average of the duration of pitch is higher for the clients that did take the product.
3. The average for the montly income is slightly hgiher for clients that did not take the product.
4. There appear to be some outliers withthin the duration of pitch and monthyly income clients in product not taken and product taken respectively.

Important Categorical variables vs Product Taken and Age

Type of Contact vs Age vs Product Taken

In [104…
```
plt.figure(figsize=(15,5))

sns.pointplot(x="Age", y="TypeofContact", hue = 'ProdTaken',  data=data)
plt.show()
```



Observation:

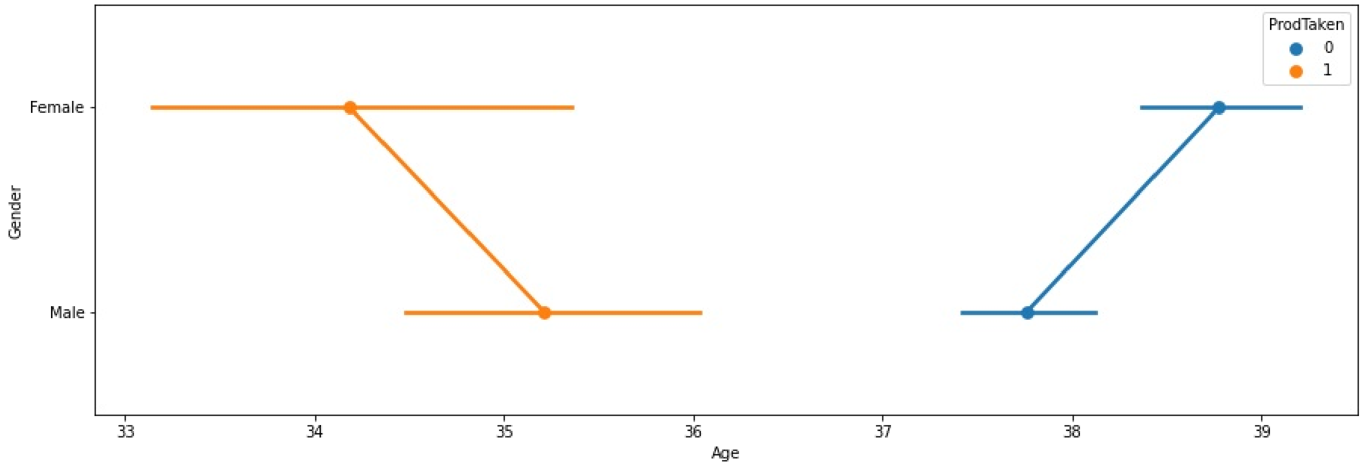
1. On average higher age clients did not take the product, for any type of contact.

2. In each category of type of cotanct higher age clients chose not to take the product.

Gender vs Age vs Product Taken

In [105…

```
plt.figure(figsize=(15,5))

sns.pointplot(x="Age", y="Gender", hue = 'ProdTaken',  data=data)
plt.show()
```



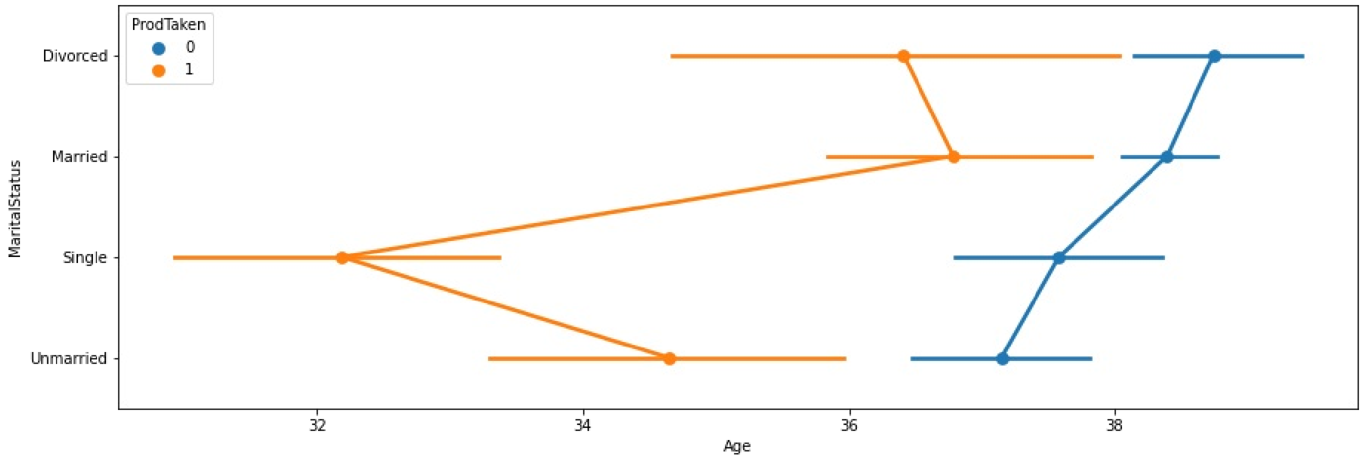
Observation:

1. On average higher aged cleints both male and female opt to not take the product.

Marital Status vs Age vs Product Taken

In [108…

```
plt.figure(figsize=(15,5))

sns.pointplot(x="Age", y="MaritalStatus", hue = 'ProdTaken',  data=data)
plt.show()
```



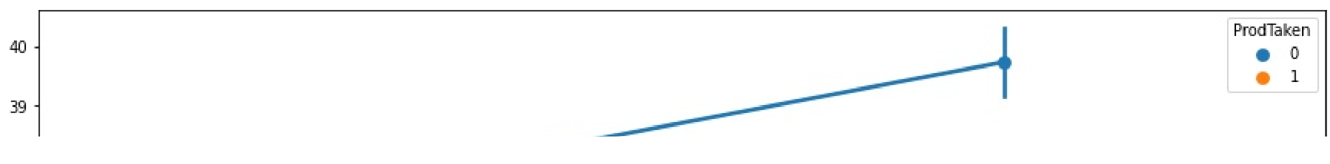
Observation:

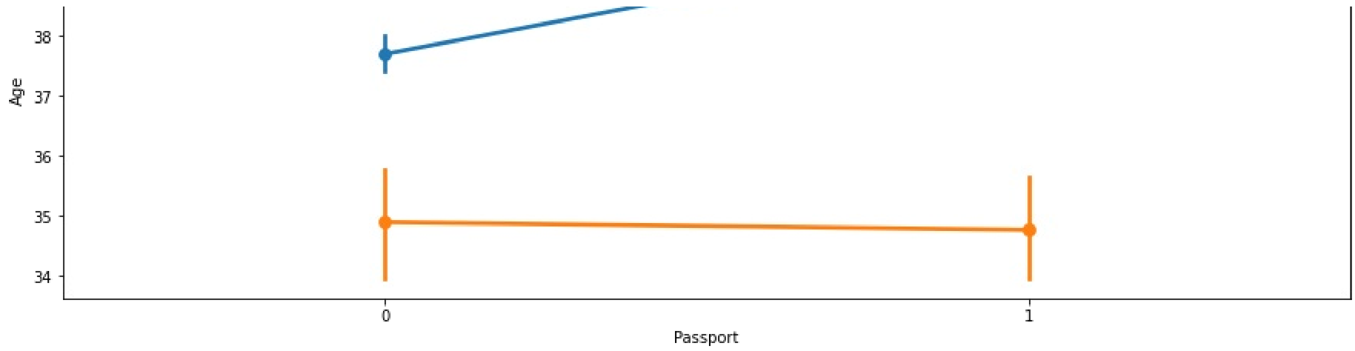
1. On Average higher aged individuals of all marital status did not take the product.

Passport vs Age vs Product Taken

In [112…

```
plt.figure(figsize=(15,5))

sns.pointplot(x="Passport", y="Age", hue = 'ProdTaken',  data=data)
plt.show()
```





Observations:

1. Higher aged clients on average who did not take the product did not have a passport.
2. Lower aged customers did take the product did have the passport.

In [53]:
```
data.to_csv('new_data.csv', index = False)
```

Data Preparation For Modelling

In [54]:
```
new_data = pd.read_csv('new_data.csv')
```

In [55]:
```
new_data.head()
```

Out[55]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPersonVisiting	NumberOfFollowups	ProductPitched
0	1	41.0	Self Enquiry	3	6.0	Salaried	Female	3	3.0	Deluxe
1	0	49.0	Company Invited	1	14.0	Salaried	Male	3	4.0	Deluxe
2	1	37.0	Self Enquiry	1	8.0	Free Lancer	Male	3	4.0	Basic
3	0	33.0	Company Invited	1	9.0	Salaried	Female	2	3.0	Basic
4	0	36.0	Self Enquiry	1	8.0	Small Business	Male	2	3.0	Basic

In [56]:
```
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ProdTaken                4888 non-null   int64  
 1   Age                      4888 non-null   float64
 2   TypeofContact            4888 non-null   object 
 3   CityTier                 4888 non-null   int64  
 4   DurationOfPitch          4888 non-null   float64
 5   Occupation               4888 non-null   object 
 6   Gender                   4888 non-null   object 
 7   NumberOfPersonVisiting   4888 non-null   int64  
 8   NumberOfFollowups        4888 non-null   object 
 9   ProductPitched           4888 non-null   object 
 10  PreferredPropertyStar    4888 non-null   object 
 11  MaritalStatus            4888 non-null   object 
 12  NumberOfTrips            4888 non-null   object 
 13  Passport                 4888 non-null   int64  
 14  PitchSatisfactionScore   4888 non-null   int64  
 15  OwnCar                   4888 non-null   int64  
 16  NumberOfChildrenVisiting 4888 non-null   object 
 17  Designation              4888 non-null   object 
 18  MonthlyIncome            4888 non-null   float64
dtypes: float64(3), int64(6), object(10)
memory usage: 725.7+ KB
```

In [57]:
```
print(new_data['TypeofContact'].value_counts())
print('*****************************************************')
print(new_data['Occupation'].value_counts())
print('*****************************************************')
```

```
print(new_data['Gender'].value_counts())
print('*******************************************************')
print(new_data['ProductPitched'].value_counts())
print('*******************************************************')
print(new_data['MaritalStatus'].value_counts())
print('*******************************************************')
print(new_data['Designation'].value_counts())
print('*******************************************************')
```

```
Self Enquiry       3444
Company Invited    1419
is_missing           25
Name: TypeofContact, dtype: int64
*******************************************************
Salaried          2368
Small Business    2084
Large Business     434
Free Lancer          2
Name: Occupation, dtype: int64
*******************************************************
Male      2916
Female    1972
Name: Gender, dtype: int64
*******************************************************
Basic           1842
Deluxe          1732
Standard         742
Super Deluxe     342
King             230
Name: ProductPitched, dtype: int64
*******************************************************
Married      2340
Divorced      950
Single        916
Unmarried     682
Name: MaritalStatus, dtype: int64
*******************************************************
Executive         1842
Manager           1732
Senior Manager     742
AVP                342
VP                 230
Name: Designation, dtype: int64
*******************************************************
```

In [58]:
```
replaceStruct = {
                "TypeofContact": {"Self Enquiry": 1, "Company Invited": 2 ,"is_missing": -1},
                "Occupation": {"Salaried": 1, "Small Business":2 , "Large Business": 3, "Free Lancer": 4},
                 "Gender": {"Male": 1, "Female":2},
                 "ProductPitched": {"Basic": 1, "Standard": 2 ,"Deluxe": 3 ,"Super Deluxe": 4 ,"King": 5},
                "MaritalStatus": {"Unmarried": 1, "Single": 2, "Married":3, "Divorced":4 },
                "Designation": {"Executive": 1, "Manager": 2 ,"Senior Manager": 3 ,"AVP": 4 ,"VP": 5}
                    }
oneHotCols=["NumberOfPersonVisiting","NumberOfFollowups",'PreferredPropertyStar', 'NumberOfTrips','PitchSatisfact
```

In [59]:
```
new_data = new_data.replace(replaceStruct)#replacing all the string columns with the integer associates
new_data = pd.get_dummies(new_data, columns=oneHotCols) #one code encoding
```

In [60]:
```
new_data.head()
```

Out[60]:

	ProdTaken	Age	DurationOfPitch	MonthlyIncome	NumberOfPersonVisiting_1	NumberOfPersonVisiting_2	NumberOfPersonVisiting_3	Number(
0	1	41.0	6.0	20993.0	0	0	1	
1	0	49.0	14.0	20130.0	0	0	1	
2	1	37.0	8.0	17090.0	0	0	1	
3	0	33.0	9.0	17909.0	0	1	0	
4	0	36.0	8.0	18468.0	0	1	0	

5 rows × 73 columns

In [61]:
```
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 73 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   ProdTaken                       4888 non-null   int64
```

```
 1   Age                                 4888 non-null   float64
 2   DurationOfPitch                     4888 non-null   float64
 3   MonthlyIncome                       4888 non-null   float64
 4   NumberOfPersonVisiting_1            4888 non-null   uint8
 5   NumberOfPersonVisiting_2            4888 non-null   uint8
 6   NumberOfPersonVisiting_3            4888 non-null   uint8
 7   NumberOfPersonVisiting_4            4888 non-null   uint8
 8   NumberOfPersonVisiting_5            4888 non-null   uint8
 9   NumberOfFollowups_1.0               4888 non-null   uint8
 10  NumberOfFollowups_2.0               4888 non-null   uint8
 11  NumberOfFollowups_3.0               4888 non-null   uint8
 12  NumberOfFollowups_4.0               4888 non-null   uint8
 13  NumberOfFollowups_5.0               4888 non-null   uint8
 14  NumberOfFollowups_6.0               4888 non-null   uint8
 15  NumberOfFollowups_is_missing        4888 non-null   uint8
 16  PreferredPropertyStar_3.0           4888 non-null   uint8
 17  PreferredPropertyStar_4.0           4888 non-null   uint8
 18  PreferredPropertyStar_5.0           4888 non-null   uint8
 19  PreferredPropertyStar_is_missing    4888 non-null   uint8
 20  NumberOfTrips_1.0                   4888 non-null   uint8
 21  NumberOfTrips_19.0                  4888 non-null   uint8
 22  NumberOfTrips_2.0                   4888 non-null   uint8
 23  NumberOfTrips_20.0                  4888 non-null   uint8
 24  NumberOfTrips_21.0                  4888 non-null   uint8
 25  NumberOfTrips_22.0                  4888 non-null   uint8
 26  NumberOfTrips_3.0                   4888 non-null   uint8
 27  NumberOfTrips_4.0                   4888 non-null   uint8
 28  NumberOfTrips_5.0                   4888 non-null   uint8
 29  NumberOfTrips_6.0                   4888 non-null   uint8
 30  NumberOfTrips_7.0                   4888 non-null   uint8
 31  NumberOfTrips_8.0                   4888 non-null   uint8
 32  NumberOfTrips_is_missing            4888 non-null   uint8
 33  PitchSatisfactionScore_1            4888 non-null   uint8
 34  PitchSatisfactionScore_2            4888 non-null   uint8
 35  PitchSatisfactionScore_3            4888 non-null   uint8
 36  PitchSatisfactionScore_4            4888 non-null   uint8
 37  PitchSatisfactionScore_5            4888 non-null   uint8
 38  OwnCar_0                            4888 non-null   uint8
 39  OwnCar_1                            4888 non-null   uint8
 40  NumberOfChildrenVisiting_0.0        4888 non-null   uint8
 41  NumberOfChildrenVisiting_1.0        4888 non-null   uint8
 42  NumberOfChildrenVisiting_2.0        4888 non-null   uint8
 43  NumberOfChildrenVisiting_3.0        4888 non-null   uint8
 44  NumberOfChildrenVisiting_is_missing 4888 non-null   uint8
 45  CityTier_1                          4888 non-null   uint8
 46  CityTier_2                          4888 non-null   uint8
 47  CityTier_3                          4888 non-null   uint8
 48  Passport_0                          4888 non-null   uint8
 49  Passport_1                          4888 non-null   uint8
 50  TypeofContact_-1                    4888 non-null   uint8
 51  TypeofContact_1                     4888 non-null   uint8
 52  TypeofContact_2                     4888 non-null   uint8
 53  Occupation_1                        4888 non-null   uint8
 54  Occupation_2                        4888 non-null   uint8
 55  Occupation_3                        4888 non-null   uint8
 56  Occupation_4                        4888 non-null   uint8
 57  Gender_1                            4888 non-null   uint8
 58  Gender_2                            4888 non-null   uint8
 59  ProductPitched_1                    4888 non-null   uint8
 60  ProductPitched_2                    4888 non-null   uint8
 61  ProductPitched_3                    4888 non-null   uint8
 62  ProductPitched_4                    4888 non-null   uint8
 63  ProductPitched_5                    4888 non-null   uint8
 64  MaritalStatus_1                     4888 non-null   uint8
 65  MaritalStatus_2                     4888 non-null   uint8
 66  MaritalStatus_3                     4888 non-null   uint8
 67  MaritalStatus_4                     4888 non-null   uint8
 68  Designation_1                       4888 non-null   uint8
 69  Designation_2                       4888 non-null   uint8
 70  Designation_3                       4888 non-null   uint8
 71  Designation_4                       4888 non-null   uint8
 72  Designation_5                       4888 non-null   uint8
dtypes: float64(3), int64(1), uint8(69)
memory usage: 482.2 KB
```

Split the Data

In [62]:
```
X = new_data.drop("ProdTaken" , axis=1)
y = new_data.pop("ProdTaken")
```

In [63]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30, random_state=1,stratify=y)
```

Before building the model, let's create functions to calculate different metrics- Accuracy, Recall and Precision and plot the confusion matrix.

In [64]:
```
## Function to create confusion matrix
def make_confusion_matrix(model,y_actual,labels=[1, 0]):
    '''
    model : classifier to predict values of X
    y_actual : ground truth

    '''
    y_predict = model.predict(X_test)
    cm=metrics.confusion_matrix( y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Actual - No","Actual - Yes"]],
                  columns = [i for i in ['Predicted - No','Predicted - Yes']])
    group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [65]:
```
##  Function to calculate different metric scores of the model - Accuracy, Recall and Precision
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)

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

    # If the flag is set to True then only the following print statements will be dispayed. The default value is
    if flag == True:
        print("Accuracy on training set : ",model.score(X_train,y_train))
        print("Accuracy on test set : ",model.score(X_test,y_test))
        print("Recall on training set : ",metrics.recall_score(y_train,pred_train))
        print("Recall on test set : ",metrics.recall_score(y_test,pred_test))
        print("Precision on training set : ",metrics.precision_score(y_train,pred_train))
        print("Precision on test set : ",metrics.precision_score(y_test,pred_test))

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

Building Models

The real problem is to enhance the customer base. Now to enhance the customer base it is needed to not lose those customers who are actually willing to buy the package but they are predicted that they will not buy the package. These are false-negative cases. Now to hold these customers we need to reduce the FN as much as possible. As a consequence, the recall has to increase. Due to this, the recall is selected as a performance metric.

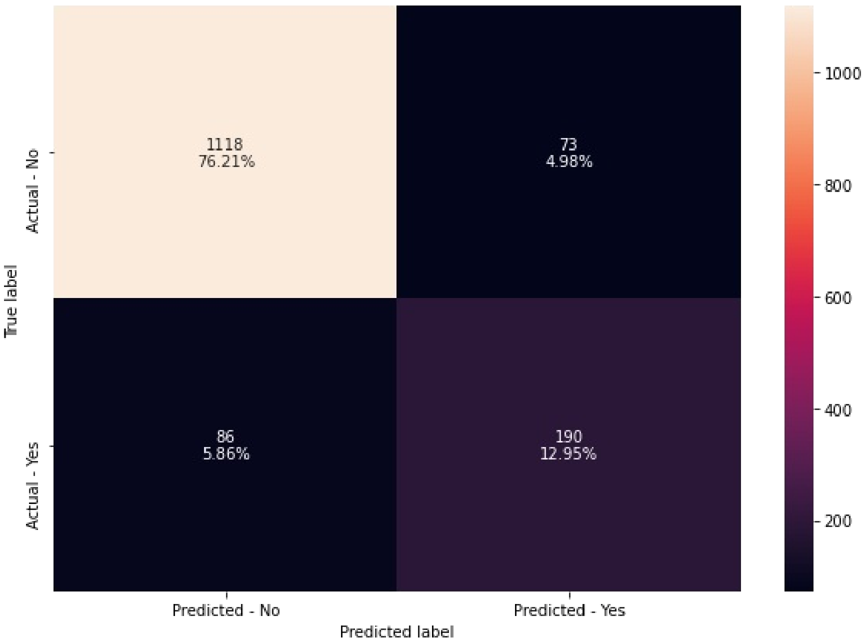
For the models we will be completing the Decision Tree, Bagging Classifier, Randomforest, ADAboosting, Gradient Boosting, XGBoost and Stacking. After running each model, each model will be further tuned with their respective hyperparameters.

Decision Tree

In [66]:
```
d_tree = DecisionTreeClassifier(random_state=1)
d_tree.fit(X_train,y_train)
```

```python
#Calculating different metrics
get_metrics_score(d_tree)

#Creating confusion matrix
make_confusion_matrix(d_tree,y_test)
```

```
Accuracy on training set :  1.0
Accuracy on test set :  0.8916155419222904
Recall on training set :  1.0
Recall on test set :  0.6884057971014492
Precision on training set :  1.0
Precision on test set :  0.7224334600760456
```



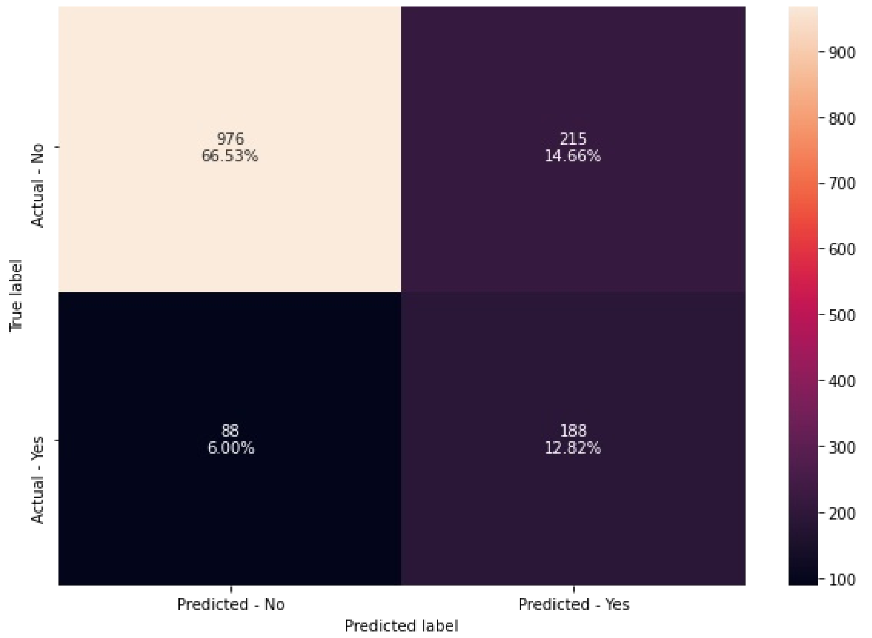
Tuning Decision Tree

In [67]:
```python
#Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight={0:0.18,1:0.72},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,n_jobs=-1)
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[67]:
```
DecisionTreeClassifier(class_weight={0: 0.18, 1: 0.72}, max_depth=5,
                       max_leaf_nodes=15, min_impurity_decrease=0.0001,
                       random_state=1)
```

In [68]:
```python
get_metrics_score(dtree_estimator)

make_confusion_matrix(dtree_estimator,y_test)
```

```
Accuracy on training set :  0.7863197895352236
Accuracy on test set :  0.7934560327198364
Recall on training set :  0.6630434782608695
Recall on test set :  0.6811594202898551
Precision on training set :  0.4537725823591923
Precision on test set :  0.4665012406947891
```



Observations:

1. The decision tree model did overfit the data as seen from the huge discrepency in the test and train data.
2. There appears to be no overfitting on the Tuning Decision tree model as the train and test data are similar.
3. There appears to be no change in the recall score from either of the two models at 68%.

Bagging Classifier

In [69]:

```
bagging_estimator=BaggingClassifier(random_state=1)
bagging_estimator.fit(X_train,y_train)
#Using above defined function to get accuracy, recall and precision on train and test set
bagging_estimator_score=get_metrics_score(bagging_estimator)
make_confusion_matrix(bagging_estimator,y_test)
```

```
Accuracy on training set :  0.994153756211634
Accuracy on test set :  0.9161554192229039
Recall on training set :  0.9720496894409938
Recall on test set :  0.6268115942028986
Precision on training set :  0.9968152866242038
Precision on test set :  0.8963730569948186
```

Hypertuning Bagging Classifier

In [70]:

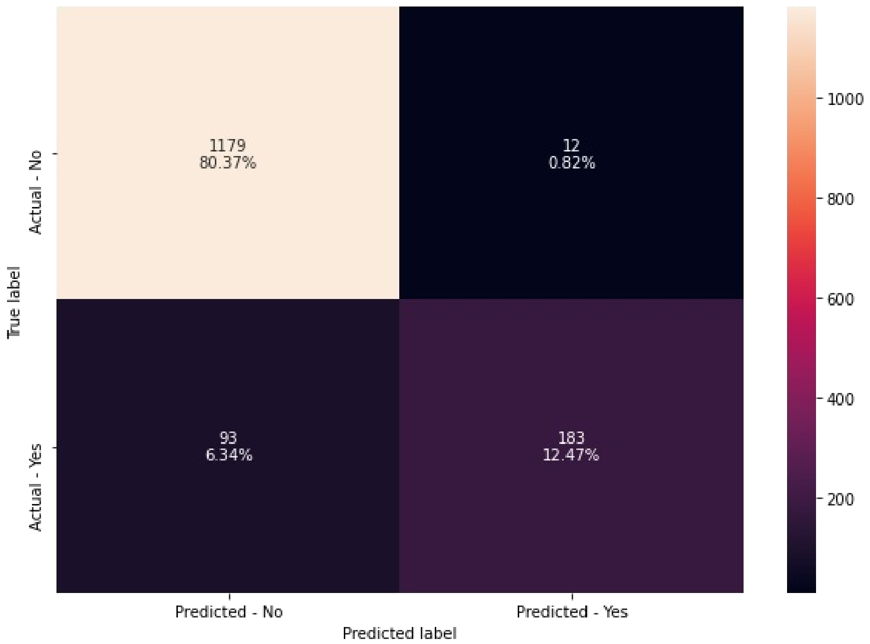
```
# Choose the type of classifier.
bagging_estimator_tuned = BaggingClassifier(random_state=1)
```

```python
# Grid of parameters to choose from
parameters = {'max_samples': [0.7,0.8,0.9,1],
              'max_features': [0.7,0.8,0.9,1],
              'n_estimators' : [10,20,30,40,50],
             }

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

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

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

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

Out[70]:
```
BaggingClassifier(max_features=0.9, max_samples=0.9, n_estimators=50,
                  random_state=1)
```

In [71]:
```python
#Calculating different metrics
get_metrics_score(bagging_estimator_tuned)

#Creating confusion matrix
make_confusion_matrix(bagging_estimator_tuned,y_test)
```

```
Accuracy on training set :  0.9994153756211634
Accuracy on test set :  0.9284253578732107
Recall on training set :  0.9968944099378882
Recall on test set :  0.6630434782608695
Precision on training set :  1.0
Precision on test set :  0.9384615384615385
```



Observations:

1. Overfitting is visible in the first model with the discrepency in the test and train data.
2. Similar overfitting can be visible in the hypertuned model as well.
3. There does not appear to be any good change in the recall score on the test data from either models at 66%.

Random Forests

In [72]:
```python
#Fitting the model
rf_estimator = RandomForestClassifier(random_state=1)
rf_estimator.fit(X_train,y_train)

#Calculating different metrics
get_metrics_score(rf_estimator)

#Creating confusion matrix
```

```
make_confusion_matrix(rf_estimator,y_test)
```

```
Accuracy on training set :  1.0
Accuracy on test set :  0.9004771642808452
Recall on training set :  1.0
Recall on test set :  0.5217391304347826
Precision on training set :  1.0
Precision on test set :  0.9113924050632911
```

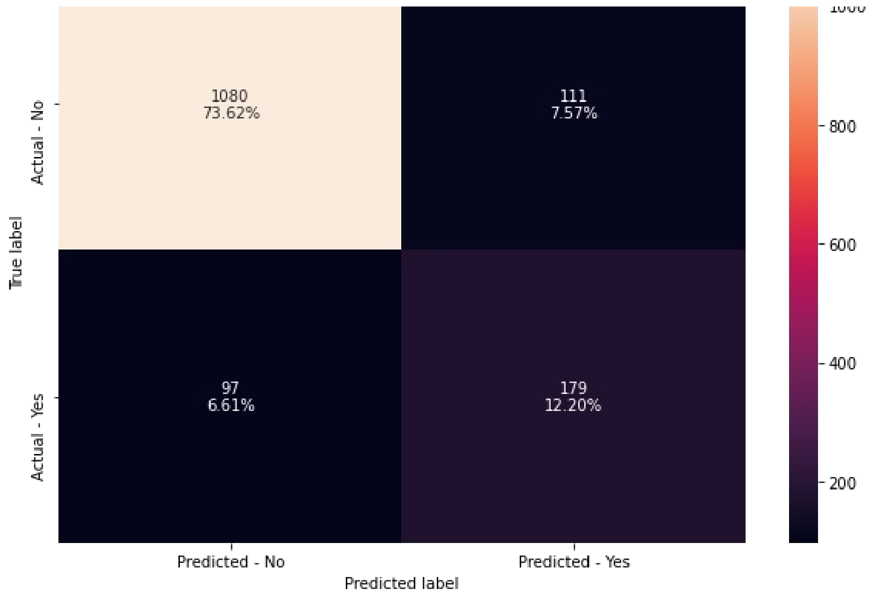
Tuning Random Forests

In [73]:
```
# Choose the type of classifier.
rf_estimator_weighted = RandomForestClassifier(random_state=1)

# Grid of parameters to choose from
## add from article
parameters = {
    "class_weight": [{0: 0.2, 1: 0.8}],
    "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),
}

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

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

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

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

Out[73]:
```
RandomForestClassifier(class_weight={0: 0.2, 1: 0.8}, max_features=0.2,
                       max_samples=0.6000000000000001, min_samples_leaf=9,
                       n_estimators=150, random_state=1)
```

In [74]:
```
#Using above defined function to get accuracy, recall and precision on train and test set
rf_estimator_weighted_score=get_metrics_score(rf_estimator_weighted)

make_confusion_matrix(rf_estimator_weighted,y_test)
```

```
Accuracy on training set :  0.8757673194972231
Accuracy on test set :  0.858214042263122
Recall on training set :  0.7577639751552795
Recall on test set :  0.6485507246376812
Precision on training set :  0.6446499339498019
Precision on test set :  0.6172413793103448
```



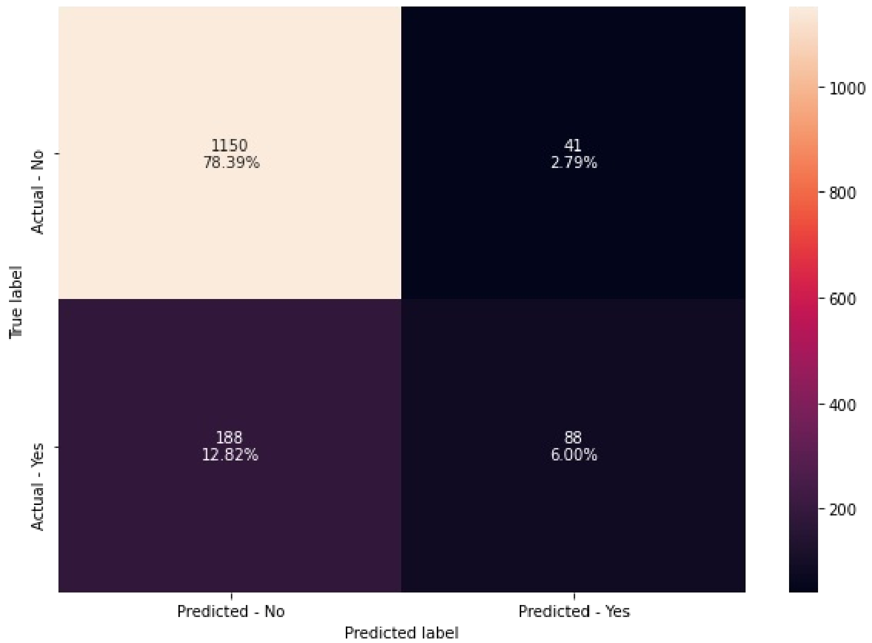
Observations:

1. To match the uneven balance in the data, the class weights has been assigned in similar weightage to the data imbalance 80-20%.
2. There is definite overfitting of the data in the first model as compared to the second model.
3. The recall score for the untuned data is around 55% while the tuned data shows significant change in the recall score by 64%.

AdaBoost Classifier

In [75]:
```
#Fitting the model
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train,y_train)

#Calculating different metrics
get_metrics_score(ab_classifier)

#Creating confusion matrix
make_confusion_matrix(ab_classifier,y_test)
```

```
Accuracy on training set :  0.8488745980707395
Accuracy on test set :  0.8438991138377642
Recall on training set :  0.3338509316770186
Recall on test set :  0.3188405797101449
Precision on training set :  0.7095709570957096
Precision on test set :  0.6821705426356589
```



Hyperparameter Tuning

In [76]:

```
# Choose the type of classifier.
abc_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
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
scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(abc_tuned, parameters, scoring=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)
```

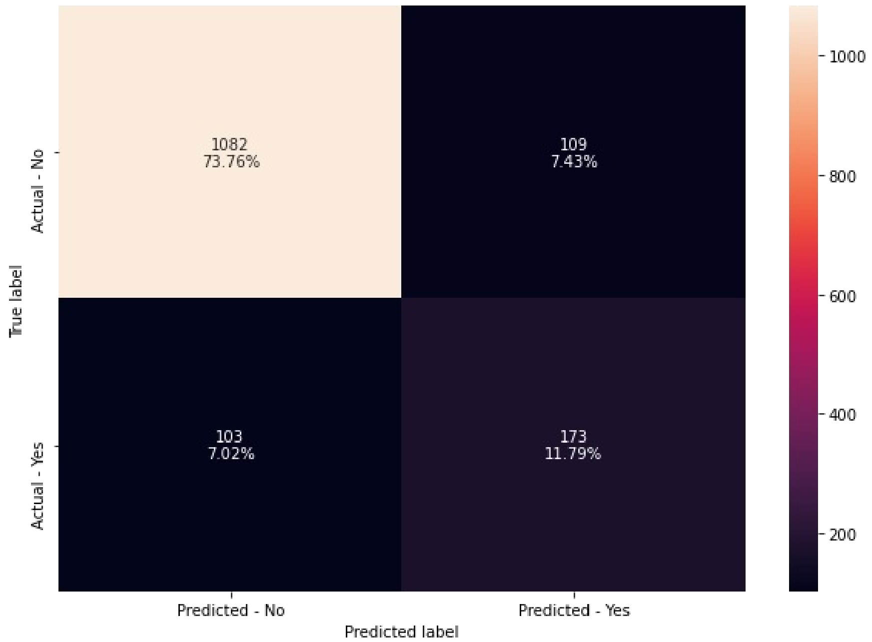
Out[76]:

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                   learning_rate=1.7000000000000002, n_estimators=100,
                   random_state=1)
```

In [77]:

```
#Calculating different metrics
get_metrics_score(abc_tuned)

#Creating confusion matrix
make_confusion_matrix(abc_tuned,y_test)
```

```
Accuracy on training set :  0.9777842736042093
Accuracy on test set :  0.8554873892297206
Recall on training set :  0.9270186335403726
Recall on test set :  0.6268115942028986
Precision on training set :  0.9536741214057508
Precision on test set :  0.6134751773049646
```



Observations:

1. The recall score on the first AdaBoost model is very low with 33%.
2. The recall score on teh hypertuned model is much better but is still low with 62% as it is with other models.
3. There is overfitting visible in the hypertuned model with teh huge discrepency in the recall score at 92% and 62%

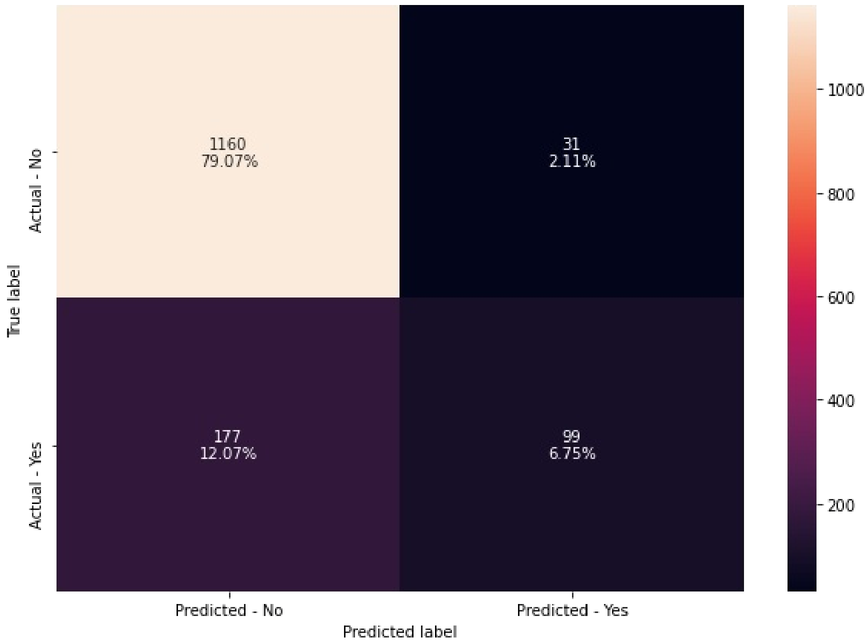
Gradient Boosting Classifier

In [78]:

```
#Fitting the model
```

```
gb_classifier = GradientBoostingClassifier(random_state=1)
gb_classifier.fit(X_train,y_train)

#Calculating different metrics
get_metrics_score(gb_classifier)

#Creating confusion matrix
make_confusion_matrix(gb_classifier,y_test)
```

```
Accuracy on training set :  0.8851213095586086
Accuracy on test set :  0.858214042263122
Recall on training set :  0.4409937888198758
Recall on test set :  0.358695652173913
Precision on training set :  0.8958990536277602
Precision on test set :  0.7615384615384615
```



Hyperparameter Tuning

In [79]:

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

# Grid of parameters to choose from
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
scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(gbc_tuned, parameters, scoring=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[79]:

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

In [80]:

```
#Calculating different metrics
get_metrics_score(gbc_tuned)

#Creating confusion matrix
make_confusion_matrix(gbc_tuned,y_test)
```

```
Accuracy on training set :  0.9237065185618241
Accuracy on test set :  0.8793456032719836
Recall on training set :  0.6335403726708074
```

```
Recall on test set :  0.4782608695652174
Precision on training set :  0.9422632794457275
Precision on test set :  0.8
```

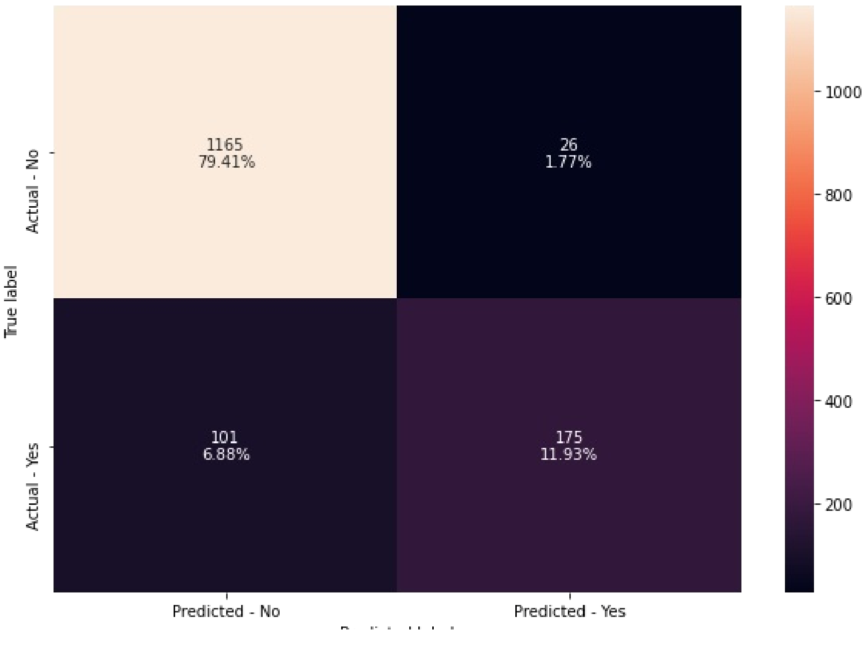
Observations:

1. In the gradient boosting model there does not appear to be overfitting but the model score is bad, as it is around the 35%.
2. There is slight change to teh hypertuned gradient boosting model with an increase in overal 10% in the recal score to 47%.
3. There does not appear to be any overfitting in either of the two models.

XGBoost Classifier

In [81]:
```
#Fitting the model
xgb_classifier = XGBClassifier(random_state=1, eval_metric='logloss')
xgb_classifier.fit(X_train,y_train)

#Calculating different metrics
get_metrics_score(xgb_classifier)

#Creating confusion matrix
make_confusion_matrix(xgb_classifier,y_test)
```

```
Accuracy on training set :  0.9988307512423268
Accuracy on test set :  0.9134287661895024
Recall on training set :  0.9937888198757764
Recall on test set :  0.6340579710144928
Precision on training set :  1.0
Precision on test set :  0.8706467661691543
```



Predicted label

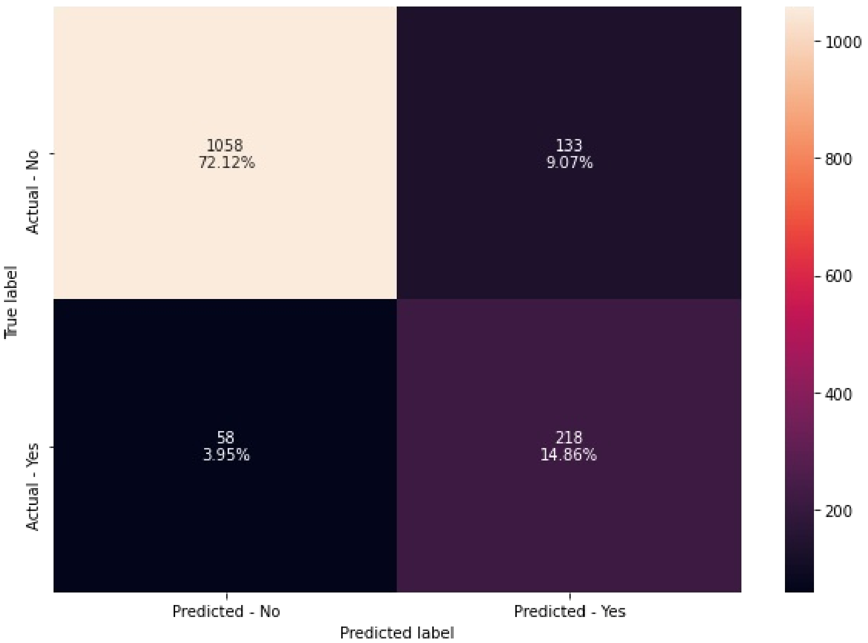
Hyperparameter Tuning

In [82]:
```
# Choose the type of classifier.
xgb_tuned = XGBClassifier(random_state=1, eval_metric='logloss')

# Grid of parameters to choose 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
scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned, parameters,scoring=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[82]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.9, eval_metric='logloss',
              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=50, n_jobs=8,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=5, subsample=0.9, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [83]:
```
#Calculating different metrics
get_metrics_score(xgb_tuned)

#Creating confusion matrix
make_confusion_matrix(xgb_tuned,y_test)
```

```
Accuracy on training set :  0.9301373867290266
Accuracy on test set :  0.8698023176550784
Recall on training set :  0.9611801242236024
Recall on test set :  0.7898550724637681
Precision on training set :  0.7430972388955582
Precision on test set :  0.6210826210826211
```



Observation:

1. There appears to be overfitting in the XGboost model as there is discrepency between the test and train data set.
2. The best recall score is visible in the hypertuned XGboost model with 78%, which overall is a good score.

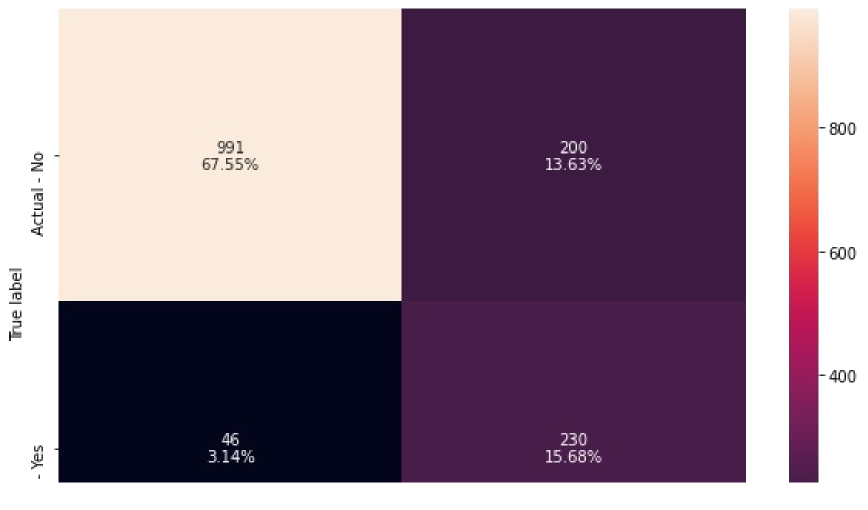
Stacking Classifier

In [84]:
```
estimators = [('Random Forest',rf_estimator_weighted), ('Gradient Boosting',gbc_tuned), ('Decision Tree',dtree_es

final_estimator = xgb_tuned

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

stacking_classifier.fit(X_train,y_train)
```

Out[84]:
```
StackingClassifier(estimators=[('Random Forest',
                                RandomForestClassifier(class_weight={0: 0.2,
                                                                     1: 0.8},
                                                       max_features=0.2,
                                                       max_samples=0.6000000000000001,
                                                       min_samples_leaf=9,
                                                       n_estimators=150,
                                                       random_state=1)),
                               ('Gradient Boosting',
                                GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                                                           max_features=0.8,
                                                           n_estimators=250,
                                                           random_state=1,
                                                           subsample=0.9)),...
                                                 eval_metric='logloss', 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=50, n_jobs=8,
                                                 num_parallel_tree=1,
                                                 random_state=1, reg_alpha=0,
                                                 reg_lambda=1,
                                                 scale_pos_weight=5,
                                                 subsample=0.9,
                                                 tree_method='exact',
                                                 validate_parameters=1,
                                                 verbosity=None))
```

In [85]:
```
#Calculating different metrics
get_metrics_score(stacking_classifier)

#Creating confusion matrix
make_confusion_matrix(stacking_classifier,y_test)
```

```
Accuracy on training set :  0.8842443729903537
Accuracy on test set :  0.8323108384458078
Recall on training set :  0.9285714285714286
Recall on test set :  0.8333333333333334
Precision on training set :  0.630801687763713
Precision on test set :  0.5348837209302325
```



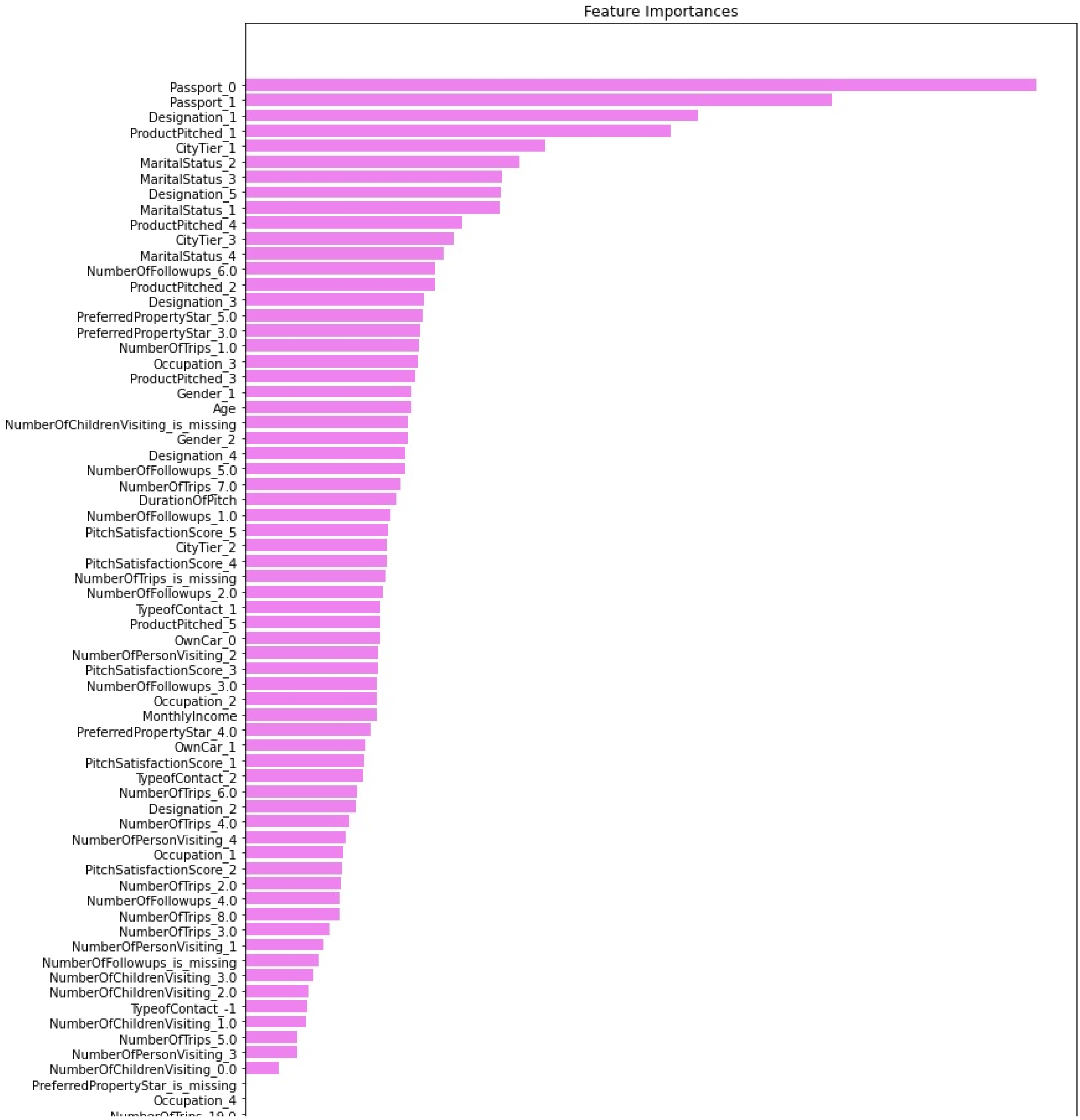
Observation:

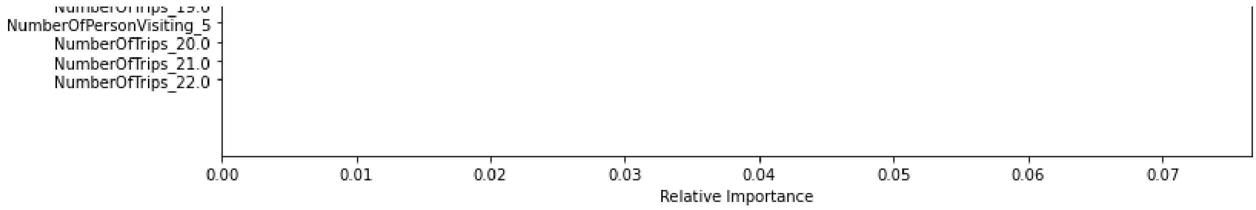
1. The Stacking classifier has showed great results compared to any other model.
2. There does not appear to be any over fitting between the train and test data.
3. The recall score on the stacking classifier is about 83%, which is overall a good score for the model.

Feature Importances

In [86]:

```
feature_names = X_train.columns
importances = xgb_tuned.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12,18))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```





Observation:

1. The most important variable is the avaiablity of Passport. Not having passport has the highest importance from all the variables.
2. Least important variable seems to be the Number of trips.

In [91]:

```
# defining list of models
models = [d_tree, dtree_estimator,rf_estimator, rf_estimator_weighted, bagging_estimator,bagging_estimator_tuned,
          ab_classifier, abc_tuned, gb_classifier, gbc_tuned, xgb_classifier,xgb_tuned, stacking_classifier]

# defining empty lists to add train and test results
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []
f1_train = []
f1_test = []

# looping through all the models to get the metrics score - Accuracy, Recall and Precision
for model in models:

    j = get_metrics_score(model,False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
```

In [97]:

```
comparison_frame = pd.DataFrame({'Model':['Decision Tree','Tuned Decision Tree','Random Forest','Tuned Random For
                                          'Bagging Classifier','Bagging Classifier Tuned','AdaBoost Classifier','
                                          'Gradient Boosting Classifier', 'Tuned Gradient Boosting Classifier',
                                          'XGBoost Classifier',  'Tuned XGBoost Classifier', '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})

#Sorting models in decreasing order of test recall
comparison_frame
```

Out[97]:

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision
0	Decision Tree	1.000000	0.891616	1.000000	0.688406	1.000000	0.722433
1	Tuned Decision Tree	0.786320	0.793456	0.663043	0.681159	0.453773	0.466501
2	Random Forest	1.000000	0.900477	1.000000	0.521739	1.000000	0.911392
3	Tuned Random Forest	0.875767	0.858214	0.757764	0.648551	0.644650	0.617241
4	Bagging Classifier	0.994154	0.916155	0.972050	0.626812	0.996815	0.896373
5	Bagging Classifier Tuned	0.999415	0.928425	0.996894	0.663043	1.000000	0.938462
6	AdaBoost Classifier	0.848875	0.843899	0.333851	0.318841	0.709571	0.682171
7	Tuned AdaBoost Classifier	0.977784	0.855487	0.927019	0.626812	0.953674	0.613475
8	Gradient Boosting Classifier	0.885121	0.858214	0.440994	0.358696	0.895899	0.761538
9	Tuned Gradient Boosting Classifier	0.923707	0.879346	0.633540	0.478261	0.942263	0.800000
10	XGBoost Classifier	0.998831	0.913429	0.993789	0.634058	1.000000	0.870647
11	Tuned XGBoost Classifier	0.930137	0.869802	0.961180	0.789855	0.743097	0.621083
12	Stacking Classifier	0.884244	0.832311	0.928571	0.833333	0.630802	0.534884

Observations:

1. From the overall models all models have good accuracy scores.
2. But since the focus is on the recall scores, the best recall score is shown by the Tumned XGboost classifier and the Stacking classifier.
3. For feature importance as the Stacking classifier does not have that feature Tuned XGboost classifier has been used.

Conclusion:

Conclusion:

1. From the EDA it was found out that on average lower aged, higher product time pitched and lower income individuals were more likely to opt for the product.
2. Most clients were from tier 1 cities, arrived with self-enquiry, was on an executive and salaried individuals that were more likely to opt for the product.
3. Males that are married, with passports, with 2 number of trips were more likely to opt for the basic type of travel package.
4. Now to enhance the customer base it is needed to not lose those customers who are actually willing to buy the package but they are predicted that they will not buy the package. These are false-negative cases. Now to hold these customers we need to reduce the FN as much as possible. As a consequence, the recall has to increase. Due to this, the recall is selected as a performance metric.
5. 13 different types of models were used, from Decision tree, random forest, bagging classifier, Adaboost, gradient boosting, XGboost and Stacking classifier with thier respective hypertuned models were used.
6. The best model from all the models were found out to be the stacking classifier and the tuned XGboost classifier that had the following recall scores 83% and 78% respectively.
7. The two scores listed above are good indication of the good models predicted by the two classifiers.
8. For feature importance, tuned XGboost classifier was chosen, and the most important features were the clients not having a passport and the designation they held. While the least important feature Number of Trips and Occupation.

Recommendation:

1. I would recommend to market individuals that are lower aged, lower income, from tier 1 cities, arrived with self-enquirt, works on an exectutive postion that are salaried. Markting towards males that are married that have passports are more likely to enquire about the basic travel package offered.
2. Most sought out travel package is the basic travel package.
3. Not having a passport is one of the main features that needs to be considered.
4. The best model from all the models were found out to be the stacking classifier and the tuned XGboost classifier that had the following recall scores 83% and 78% respectively. The two scores listed above are good indication of the good models predicted by the two classifiers.For feature importance, tuned XGboost classifier was chosen, and the most important features were the clients not having a passport and the designation they held. While the least important feature Number of Trips and Occupation.

```
In [ ]:
```