Problem Statement

Context

The policymaker of a tourism company named "Visit with us" wants to enable and establish a viable business model to expand the customer base. One of the ways to expand the customer base is to introduce a new offering of packages. Currently, there are 5 types of packages the company is offering - Basic, Standard, Deluxe, Super Deluxe, and King. Looking at the data of the last year, it was observed that 18% of the customers purchased the packages. However, it was difficult to identify the potential customers 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 a package that allows the traveler to maintain, enhance, or kick-start a healthy lifestyle, and support or increase one's sense of well-being. For the launch of the new package, the company wants to harness the available data of existing and potential customers to target the right customers.

Objective

I as a Data Scientist at "Visit with us" travel company has to analyze the customer data and information to provide recommendations to the policymaker and build a model to predict the chances of a potential customer purchasing the newly introduced travel package before the customer is contacted.

Data Dictionary

- 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. It's the city the customer lives in.
- DurationOfPitch: Duration of the pitch by a salesperson to the customer
- Occupation: Occupation of customer
- Gender: Gender of customer
- NumberOfPersonVisiting: Total number of persons planning to take the trip with the customer
- NumberOfFollowups: Total number of follow-ups has been done by the salesperson after the sales pitch
- ProductPitched: Product pitched by the salesperson

- 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)
- PitchSatisfactionScore: Sales pitch satisfaction score
- 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

Importing necessary libraries

In [1]:

```
# Library to suppress warnings or deprecation notes
import warnings
warnings.filterwarnings('ignore')

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

# Libraries to help with data visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# Libraries to split data, impute missing values
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

# Libraries to import decision tree classifier and different ensemble classifiers
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
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
from sklearn.metrics import f1_score, roc_auc_score
from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import GridSearchCV

# To get diferent metric scores
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
```

```
    precision_score,
    confusion_matrix,
    plot_confusion_matrix,
    make_scorer,
    roc_auc_score,
    plot_confusion_matrix,
    precision_recall_curve,
    roc_curve,
)

# to check model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Command to tell Python to actually display the graphs
%matplotlib inline

# open-source Python graphing library for building beautiful, interactive visualiza
!pip install plotly
import plotly.express as px

from sklearn.preprocessing import StandardScaler, MinMaxScaler


import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

```
Requirement already satisfied: plotly in c:\users\munee\anaconda3\lib\site-packages
(5.11.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\munee\anaconda3\lib\site-
packages (from plotly) (8.1.0)
```

In [1]:

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

```
  Cell In[1], line 3
    pip install nbconvert[webpdf]
        ^
SyntaxError: invalid syntax
```

Loading the dataset

In [3]:

```
#Loading the dataset - sheet_name parameter is used if there are Basicple tabs in t
df0 = pd.read_csv("Tourism.csv")
data = df0.copy()
```

Data Overview

View the first and last 5 rows of the dataset.

In [4]:
```
data.head()
```

Out[4]:

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

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

Out[5]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation
4883	204883	1	49.0	Self Enquiry	3	9.0	Small Business
4884	204884	1	28.0	Company Invited	1	31.0	Salaried
4885	204885	1	52.0	Self Enquiry	3	17.0	Salaried
4886	204886	1	19.0	Self Enquiry	3	16.0	Small Business
4887	204887	1	36.0	Self Enquiry	1	14.0	Salaried

In [6]:
```
np.random.seed(1)
data.sample(n=15)
```

Out[6]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation
3015	203015	0	27.0	Company Invited	1	7.0	Salariec
1242	201242	0	40.0	Self Enquiry	3	13.0	Smal Business
3073	203073	0	29.0	Self Enquiry	2	15.0	Smal Business
804	200804	0	48.0	Company Invited	1	6.0	Smal Business
3339	203339	0	32.0	Self Enquiry	1	18.0	Smal Business
3080	203080	1	36.0	Company Invited	1	32.0	Salariec
2851	202851	0	46.0	Self Enquiry	1	17.0	Salariec

2883	202883	1	32.0	Company Invited	1	27.0	Salariec
1676	201676	0	22.0	Self Enquiry	1	11.0	Salariec
1140	201140	0	44.0	Self Enquiry	1	13.0	Smal Business
748	200748	1	26.0	Company Invited	3	35.0	Smal Business
2394	202394	1	NaN	Company Invited	1	8.0	Salariec
4881	204881	1	41.0	Self Enquiry	2	25.0	Salariec
3415	203415	0	52.0	Self Enquiry	1	18.0	Large Business
2253	202253	0	NaN	Self Enquiry	1	13.0	Large Business

Understand the shape of the dataset.

In [7]:
```
data.shape
```

Out[7]: (4888, 20)

Check the data types of the columns for the dataset

In [8]:
```
data.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
```

- There are total of 20 columns and 4,888 observations in the dataset
- We can see that 8 columns have less than 4,888 non-null values i.e. columns have missing values.

Checking the Statistical Summary

In [9]:
```
data.describe().T
```

Out[9]:

	count	mean	std	min	25%	50'
CustomerID	4888.0	202443.500000	1411.188388	200000.0	201221.75	202443
ProdTaken	4888.0	0.188216	0.390925	0.0	0.00	0
Age	4662.0	37.622265	9.316387	18.0	31.00	36
CityTier	4888.0	1.654255	0.916583	1.0	1.00	1
DurationOfPitch	4637.0	15.490835	8.519643	5.0	9.00	13
NumberOfPersonVisiting	4888.0	2.905074	0.724891	1.0	2.00	3
NumberOfFollowups	4843.0	3.708445	1.002509	1.0	3.00	4
PreferredPropertyStar	4862.0	3.581037	0.798009	3.0	3.00	3
NumberOfTrips	4748.0	3.236521	1.849019	1.0	2.00	3
Passport	4888.0	0.290917	0.454232	0.0	0.00	0
PitchSatisfactionScore	4888.0	3.078151	1.365792	1.0	2.00	3
OwnCar	4888.0	0.620295	0.485363	0.0	0.00	1
NumberOfChildrenVisiting	4822.0	1.187267	0.857861	0.0	1.00	1
MonthlyIncome	4655.0	23619.853491	5380.698361	1000.0	20346.00	22347

- Mean and median of age column are very close to each other i.e. approx 37 and 36 respectively.
- Duration of pitch has some outliers at the right end as the 75th percentile value is 20 and the max value is 127. We need to explore this further.
- It seems like monthly income has some outliers at both ends. We need to explore this further.
- The number of trips also has some outliers as the 75th percentile value is 4 and the max value is 22.
- We can see that the target variable - ProdTaken is imbalanced as most of the values are 0.

Checking for unique values for each of the column

In [10]:
```
data.nunique()
```

Out[10]:
```
CustomerID                 4888
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
```

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

In [11]:
```
#Dropping CustomerID column
data.drop(columns='CustomerID',inplace=True)
```

Checking for Missing Values

In [12]:
```
pd.DataFrame(data={'% of Missing Values':round(data.isna().sum()/data.isna().count(
```

Out[12]:

	% of Missing Values
ProdTaken	0.00
Age	4.62
TypeofContact	0.51
CityTier	0.00
DurationOfPitch	5.14
Occupation	0.00
Gender	0.00

NumberOfPersonVisiting	0.00
NumberOfFollowups	0.92
ProductPitched	0.00
PreferredPropertyStar	0.53
MaritalStatus	0.00
NumberOfTrips	2.86
Passport	0.00
PitchSatisfactionScore	0.00
OwnCar	0.00
NumberOfChildrenVisiting	1.35
Designation	0.00
MonthlyIncome	4.77

- The `Age` column has 4.62% missing values out of the total observations.
- `TypeofContact` column has 0.51% missing values out of the total observations.
- `DurationOfPitch` column has 5.14% missing values out of the total observations.
- The `NumberOfFollowups` column has 0.92% missing values out of the total observations.
- `PreferredPropertyStar` column has 0.53% missing values out of the total observations.
- `NumberOfTrips` column has 2.86% missing values out of the total observations.
- `NumberOfChildrenVisiting` column has 1.35% missing values out of the total observations.
- The `MonthlyIncome` column has 4.77% missing values out of the total observations.
- We will impute these values after we split the data into train and test sets.

Data Preprocessing

Checking for anomalous/repetitive values

In [13]:

```
#Making a list of all catrgorical variables
cat_col=['TypeofContact', 'CityTier','Occupation', 'Gender', 'NumberOfPersonVisitin
        'NumberOfFollowups', 'ProductPitched', 'PreferredPropertyStar',
        'MaritalStatus', 'Passport', 'PitchSatisfactionScore',
        'OwnCar', 'NumberOfChildrenVisiting', 'Designation']

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

```
Self Enquiry       3444
Company Invited    1419
Name: TypeofContact, dtype: int64
--------------------------------------------------
1    3190
3    1500
2     198
Name: CityTier, dtype: int64
--------------------------------------------------
Salaried          2368
Small Business    2084
Large Business     434
Free Lancer          2
Name: Occupation, dtype: int64
--------------------------------------------------
Male       2916
Female     1817
Fe Male     155
Name: Gender, dtype: int64
--------------------------------------------------
3    2402
2    1418
4    1026
1      39
5       3
Name: NumberOfPersonVisiting, dtype: int64
--------------------------------------------------
4.0    2068
3.0    1466
5.0     768
2.0     229
1.0     176
6.0     136
Name: NumberOfFollowups, dtype: int64
--------------------------------------------------
Basic           1842
Deluxe          1732
Standard         742
Super Deluxe     342
King             230
Name: ProductPitched, dtype: int64
--------------------------------------------------
3.0    2993
5.0     956
4.0     913
Name: PreferredPropertyStar, dtype: int64
--------------------------------------------------
Married      2340
Divorced      950
Single        916
Unmarried     682
Name: MaritalStatus, dtype: int64
--------------------------------------------------
0    3466
1    1422
Name: Passport, dtype: int64
```

```
-------------------------------------------------
3    1478
5     970
1     942
4     912
2     586
Name: PitchSatisfactionScore, dtype: int64
-------------------------------------------------
1    3032
0    1856
Name: OwnCar, dtype: int64
-------------------------------------------------
1.0    2080
2.0    1335
0.0    1082
3.0     325
Name: NumberOfChildrenVisiting, dtype: int64
-------------------------------------------------
Executive         1842
Manager           1732
Senior Manager     742
AVP                342
VP                 230
Name: Designation, dtype: int64
-------------------------------------------------
```

- The Free lancer category in the occupation column has just 2 entries out of 4,888 observations.
- We can see that Gender has 3 unique values which include - 'Fe Male' and 'Female'. This must be a data input error, we should replace 'Fe Male' with 'Female'.
- NumberOfPersonVisiting equal to 5 has a count equal to 3 only.
- The majority of the customers are married.
- The majority of the customers own a car.

In [14]:
```
#Replacing 'Fe Male' with 'Female'
data.Gender=data.Gender.replace('Fe Male', 'Female')
```

In [15]:
```
#Converting the data type of each categorical variable to 'category'
for column in cat_col:
    data[column]=data[column].astype('category')
```

In [16]:
```
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   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   float64
 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(14), float64(4), int64(1)
memory usage: 260.3 KB
```

In [17]:
```
## Creating a copy of data to perform detailed EDA in the appendix section
df = data.copy()
```

Exploratory Data Analysis (EDA)

Note: The EDA section has been covered multiple times in the previous case studies. In this case study, we will mainly focus on the model building aspects. We will only be looking at the key observations from EDA. The detailed EDA can be found in the appendix section.

Univariate Analysis

In [18]:
```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the col
```

```python
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [19]:

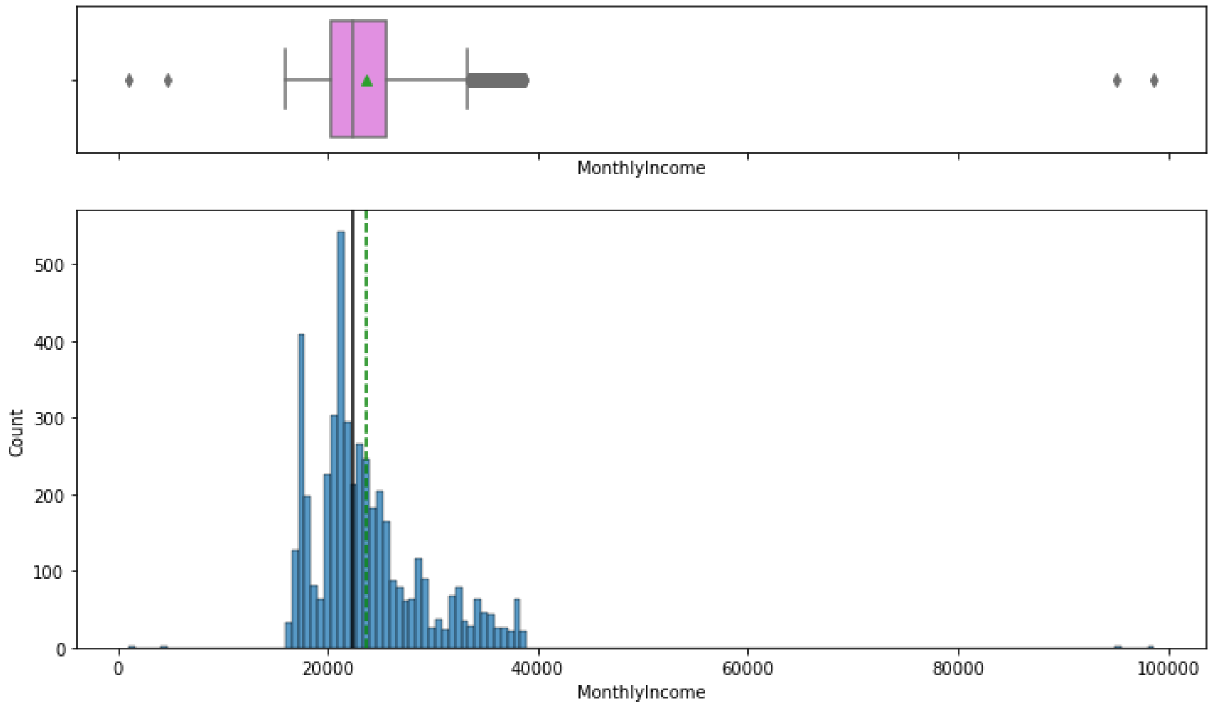
```python
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all level
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
```

```python
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

In [20]:
```python
# function to plot stacked bar chart


def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_values
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 1, 5))
    plt.legend(
        loc="lower left",
        frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

Let us check the distribution of the monthly income of customers

In [21]:
```python
histogram_boxplot(df,'MonthlyIncome')
```



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

In [22]:
```
df[(df.MonthlyIncome>40000) | (df.MonthlyIncome<12000)]
```

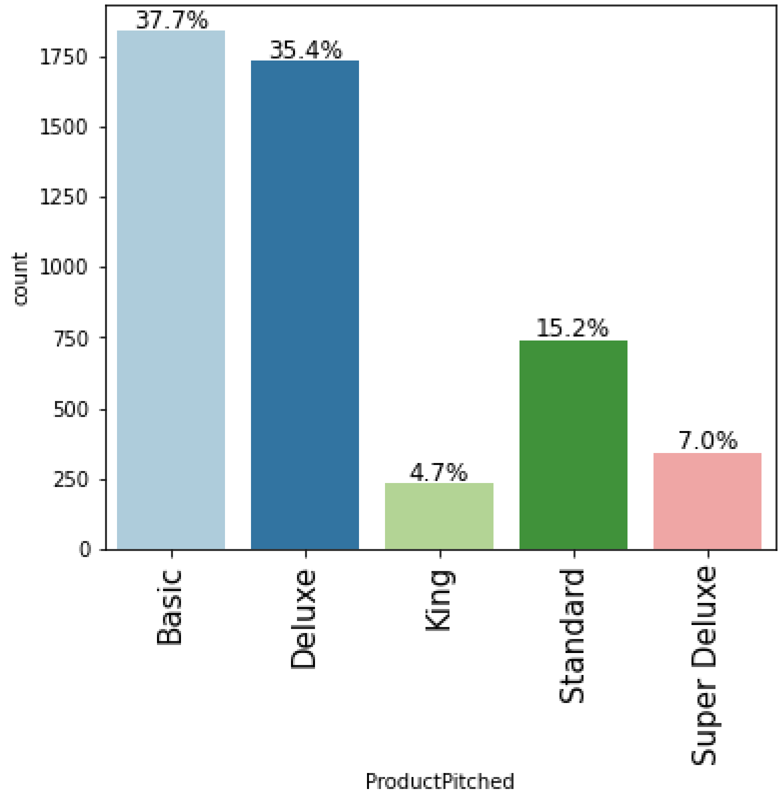
Out[22]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Nu
38	0	36.0	Self Enquiry	1	11.0	Salaried	Female	
142	0	38.0	Self Enquiry	1	9.0	Large Business	Female	
2482	0	37.0	Self Enquiry	1	12.0	Salaried	Female	
2586	0	39.0	Self Enquiry	1	10.0	Large Business	Female	

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

Let us check the distribution of travel packages pitched by the salespersons to the customers

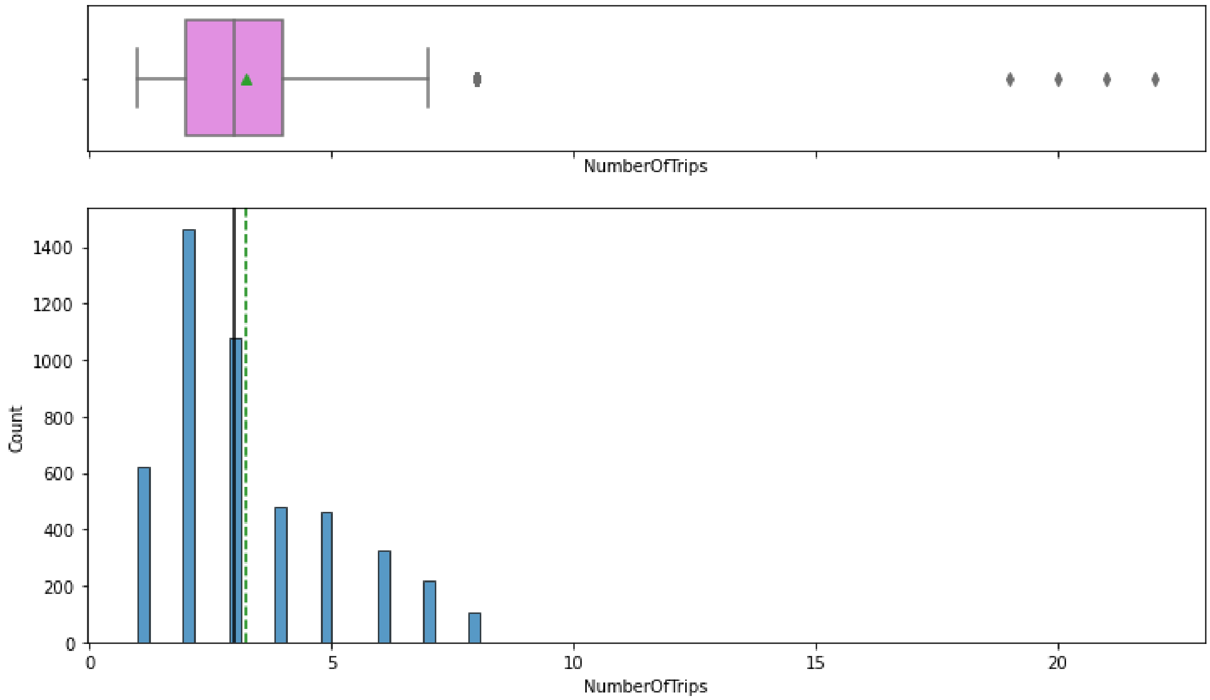
In [23]:
```
labeled_barplot(df, "ProductPitched", perc=True)
```



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

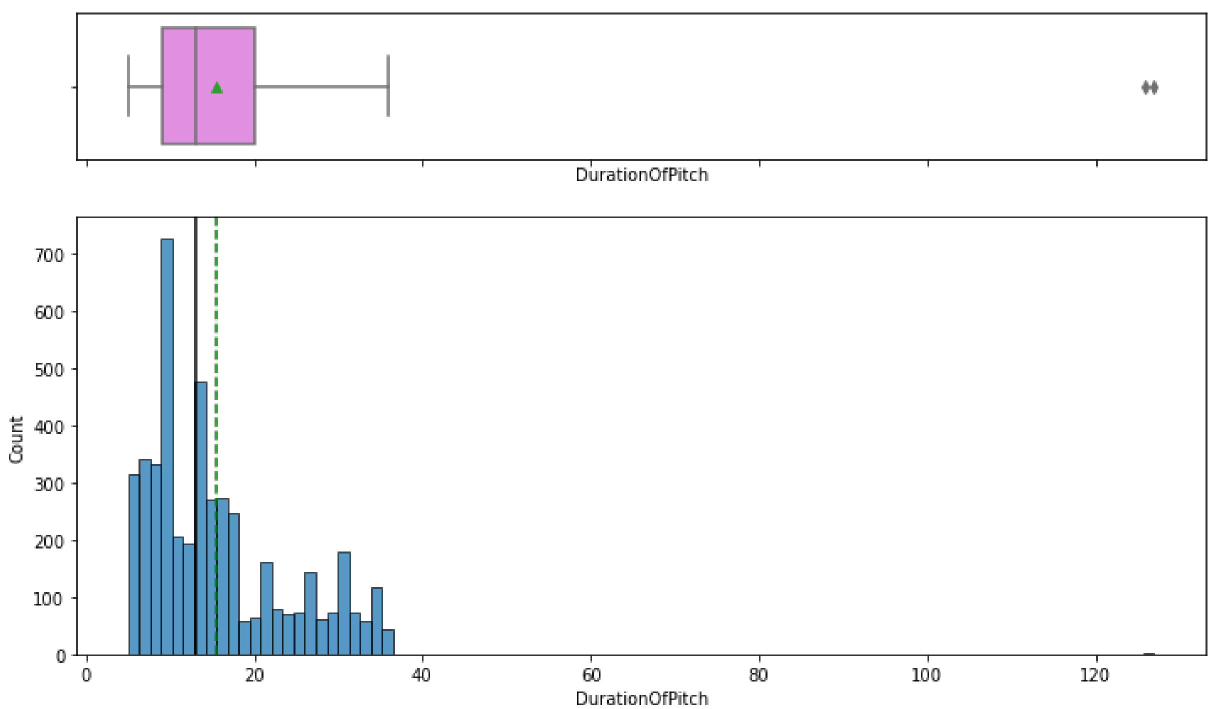
Observations on Number of Trips

In [24]:
```
histogram_boxplot(df,'NumberOfTrips')
```



- The distribution for the number of trips is right-skewed
- Boxplot shows that the number of trips has some outliers at the right end.

In [25]:
```
histogram_boxplot(df, 'DurationOfPitch')
```



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

In [26]:
```
df[df['DurationOfPitch']>40]
```

Out[26]:

	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	N
1434	0	NaN	Company Invited	3	126.0	Salaried	Male	
3878	0	53.0	Company Invited	3	127.0	Salaried	Male	

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

Bivariate Analysis

In [27]:
```
# function to plot stacked bar chart


def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_values
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 1, 5))
    plt.legend(
        loc="lower left",
        frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

Correlation Check

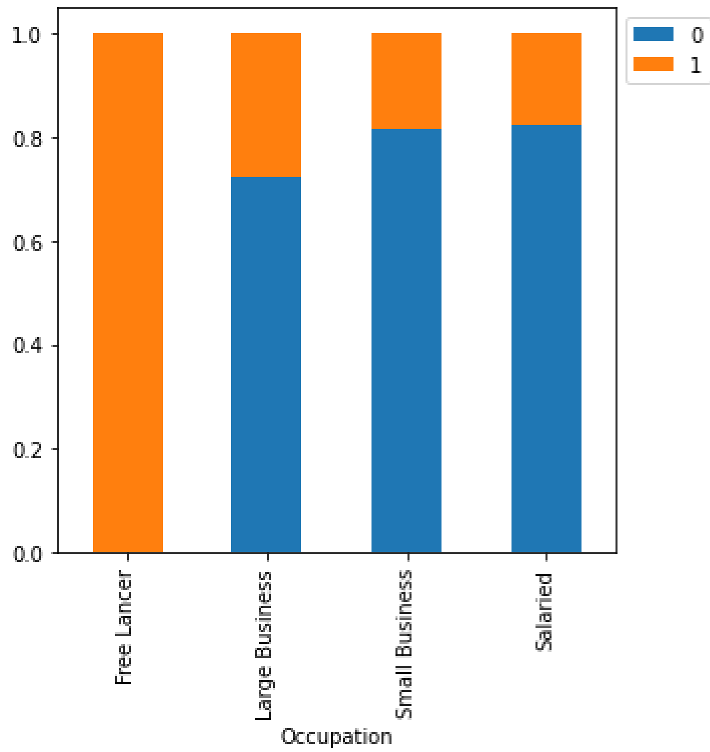
In [28]:
```
plt.figure(figsize=(15, 7))
sns.heatmap(data.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral")
plt.show()
```

- The Number of trips and age have a weak positive correlation, which makes sense as age increases number of trips is expected to increase.
- Age and monthly income are positively correlated.
- ProdTaken has a weak negative correlation with age which agrees with our earlier observation that as age increases the probability for purchasing a package decreases.
- No other variables have a high correlation among them.

Let us check how a customer's interest in purchasing the newly introduced travel package varies with their occupation

In [29]:
```
stacked_barplot(df, "Occupation", "ProdTaken" )
```

```
ProdTaken          0    1   All
Occupation
All             3968  920  4888
Salaried        1954  414  2368
Small Business  1700  384  2084
Large Business   314  120   434
Free Lancer        0    2     2
------------------------------------------------------------------------------------
------------------------------------
```

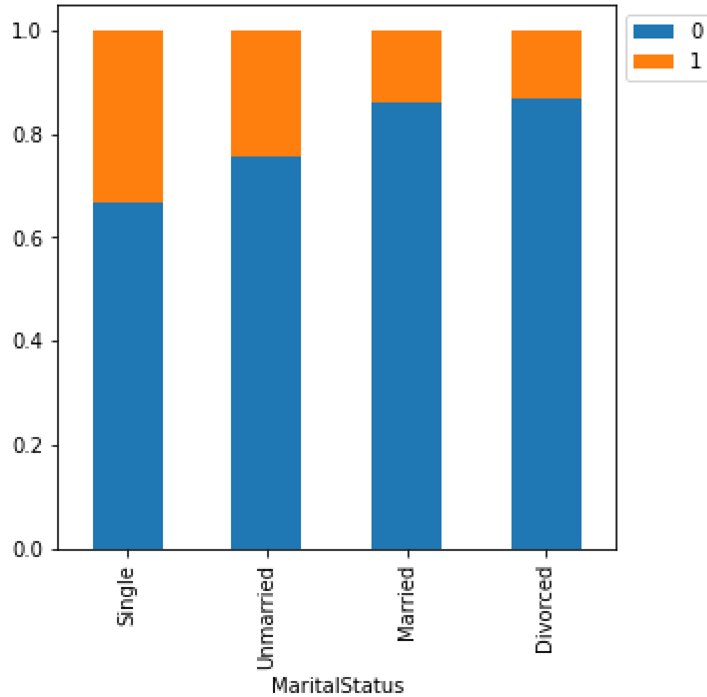


- The conversion rate for large business owners is higher than salaried or small business owners.
- This might be because large business owners have high income.
- Freelancer have 100% conversion rate but there is just 2 such observation, so cannot give any conclusive insights.

Let us check how a customer's interest in purchasing the newly introduced travel package vary with their marital status

In [30]:
```
stacked_barplot(data, "MaritalStatus", "ProdTaken" )
```

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

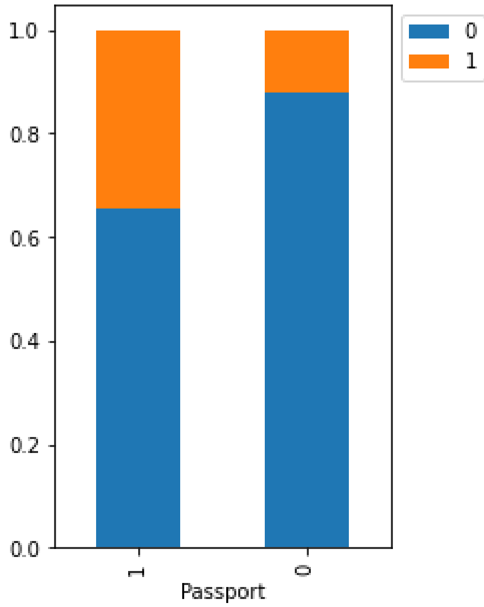


- The married people are the most common customer for the company but this graph shows that the conversion rate is higher for single and unmarried customers as compared to the married customers.
- The company can target single and unmarried customers more and can modify packages as per these customers.

Prod Taken vs Passport

In [31]:
```
stacked_barplot(data, "Passport", "ProdTaken" )
```

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

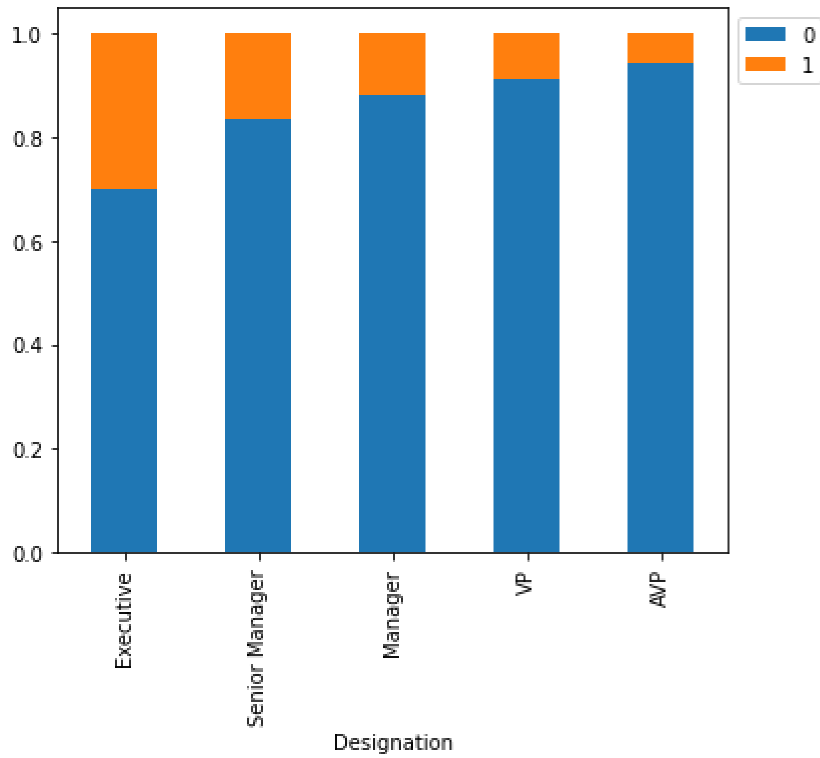


- The conversion rate for customers with a passport is higher as compared to the customers without a passport.
- The company should customize more international packages to attract more such customers.

ProdTaken vs Designation

In [32]:
```
stacked_barplot(data, "Designation", "ProdTaken" )
```

```
ProdTaken          0    1   All
Designation
All             3968  920  4888
Executive       1290  552  1842
Manager         1528  204  1732
Senior Manager   618  124   742
AVP              322   20   342
VP               210   20   230
------------------------------------------------------------------------------------
------------------------------------
```



- The conversion rate of executives is higher than other designations.
- Customers at VP and AVP positions have the least conversion rate.

Customer Profiles by Travel Package

Basic

In [33]:
```
data[(data['ProductPitched']=='Basic') & (data['ProdTaken']==1)].describe(include='
```

Out[33]:

	count	unique	top	freq	mean	std
ProdTaken	552.0	NaN	NaN	NaN	1.0	0.0
Age	515.0	NaN	NaN	NaN	31.28932	9.070829
TypeofContact	549	2	Self Enquiry	355	NaN	NaN
CityTier	552.0	3.0	1.0	392.0	NaN	NaN
DurationOfPitch	532.0	NaN	NaN	NaN	15.791353	7.906926
Occupation	552	4	Salaried	260	NaN	NaN
Gender	552	2	Male	344	NaN	NaN
NumberOfPersonVisiting	552.0	3.0	3.0	276.0	NaN	NaN
NumberOfFollowups	548.0	6.0	4.0	235.0	NaN	NaN
ProductPitched	552	1	Basic	552	NaN	NaN
PreferredPropertyStar	552.0	3.0	3.0	282.0	NaN	NaN

MaritalStatus	552	4	Single	230	NaN	NaN	N
NumberOfTrips	547.0	NaN	NaN	NaN	3.226691	2.081618	
Passport	552.0	2.0	1.0	322.0	NaN	NaN	N
PitchSatisfactionScore	552.0	5.0	3.0	178.0	NaN	NaN	N
OwnCar	552.0	2.0	1.0	316.0	NaN	NaN	N
NumberOfChildrenVisiting	551.0	4.0	1.0	234.0	NaN	NaN	N
Designation	552	1	Executive	552	NaN	NaN	N
MonthlyIncome	529.0	NaN	NaN	NaN	20161.529301	3313.289684	160

- Average monthly income for customers opting for the basic package is ~20,165.
- Average age of customers opting for the basic package is ~31
- Majority of the customers opting for the basic package are at the executive designation
- Majority of the customers opting for the basic package are single

Standard

In [34]:
```
data[(data['ProductPitched']=='Standard') & (data['ProdTaken']==1)].describe(includ
```

Out[34]:

	count	unique	top	freq	mean	std	n
ProdTaken	124.0	NaN	NaN	NaN	1.0	0.0	
Age	123.0	NaN	NaN	NaN	41.00813	9.876695	1
TypeofContact	124	2	Self Enquiry	92	NaN	NaN	N
CityTier	124.0	3.0	3.0	64.0	NaN	NaN	N
DurationOfPitch	123.0	NaN	NaN	NaN	19.065041	9.048811	
Occupation	124	3	Small Business	58	NaN	NaN	N
Gender	124	2	Male	76	NaN	NaN	N
NumberOfPersonVisiting	124.0	3.0	3.0	62.0	NaN	NaN	N
NumberOfFollowups	124.0	6.0	4.0	56.0	NaN	NaN	N
ProductPitched	124	1	Standard	124	NaN	NaN	N
PreferredPropertyStar	123.0	3.0	3.0	68.0	NaN	NaN	N
MaritalStatus	124	4	Married	56	NaN	NaN	N
NumberOfTrips	123.0	NaN	NaN	NaN	3.01626	1.815163	
Passport	124.0	2.0	0.0	76.0	NaN	NaN	N
PitchSatisfactionScore	124.0	5.0	3.0	42.0	NaN	NaN	N

OwnCar	124.0	2.0	1.0	82.0	NaN	NaN	N
NumberOfChildrenVisiting	123.0	4.0	1.0	52.0	NaN	NaN	N
Designation	124	1	Senior Manager	124	NaN	NaN	N
MonthlyIncome	124.0	NaN	NaN	NaN	26035.419355	3593.290353	1737

- Average monthly income of customers opting for the standard package is ~26,035.
- Average age for customers opting for the standard package is ~41
- Majority of the customers opting for the standard package are at senior manager designation
- Majority of the customers opting for the standard package are married

Deluxe

In [35]:
```
data[(data['ProductPitched']=='Deluxe') & (data['ProdTaken']==1)].describe(include=
```

Out[35]:

	count	unique	top	freq	mean	std	ı
ProdTaken	204.0	NaN	NaN	NaN	1.0	0.0	
Age	198.0	NaN	NaN	NaN	37.641414	8.469575	2
TypeofContact	204	2	Self Enquiry	136	NaN	NaN	N
CityTier	204.0	2.0	3.0	144.0	NaN	NaN	N
DurationOfPitch	180.0	NaN	NaN	NaN	19.1	9.227176	
Occupation	204	3	Small Business	108	NaN	NaN	N
Gender	204	2	Male	134	NaN	NaN	N
NumberOfPersonVisiting	204.0	3.0	3.0	102.0	NaN	NaN	N
NumberOfFollowups	200.0	6.0	4.0	78.0	NaN	NaN	N
ProductPitched	204	1	Deluxe	204	NaN	NaN	N
PreferredPropertyStar	203.0	3.0	3.0	114.0	NaN	NaN	N
MaritalStatus	204	4	Married	68	NaN	NaN	N
NumberOfTrips	202.0	NaN	NaN	NaN	3.70297	2.022483	
Passport	204.0	2.0	0.0	104.0	NaN	NaN	N
PitchSatisfactionScore	204.0	5.0	3.0	76.0	NaN	NaN	N
OwnCar	204.0	2.0	1.0	124.0	NaN	NaN	N
NumberOfChildrenVisiting	203.0	4.0	1.0	90.0	NaN	NaN	N
Designation	204	1	Manager	204	NaN	NaN	N

	count	unique	top	freq	mean	std	min
MonthlyIncome	195.0	NaN	NaN	NaN	23106.215385	3592.466947	170

- Average monthly income of customers opting for the deluxe package is ~23,106.
- Average age for customers opting for the deluxe package is ~37
- Majority of the customers opting for the deluxe package are at manager designation
- Majority of the customers opting for the deluxe package are married

Super Deluxe

In [36]:
```
data[(data['ProductPitched']=='Super Deluxe') & (data['ProdTaken']==1)].describe(in
```

Out[36]:

	count	unique	top	freq	mean	std	min
ProdTaken	20.0	NaN	NaN	NaN	1.0	0.0	1.0
Age	20.0	NaN	NaN	NaN	43.5	4.83953	39.0
TypeofContact	20	2	Company Invited	16	NaN	NaN	NaN
CityTier	20.0	2.0	3.0	16.0	NaN	NaN	NaN
DurationOfPitch	20.0	NaN	NaN	NaN	18.5	7.330542	8.0
Occupation	20	2	Salaried	16	NaN	NaN	NaN
Gender	20	2	Male	16	NaN	NaN	NaN
NumberOfPersonVisiting	20.0	3.0	3.0	10.0	NaN	NaN	NaN
NumberOfFollowups	20.0	6.0	1.0	4.0	NaN	NaN	NaN
ProductPitched	20	1	Super Deluxe	20	NaN	NaN	NaN
PreferredPropertyStar	20.0	3.0	3.0	12.0	NaN	NaN	NaN
MaritalStatus	20	3	Single	10	NaN	NaN	NaN
NumberOfTrips	19.0	NaN	NaN	NaN	3.263158	2.490919	1.0
Passport	20.0	2.0	1.0	12.0	NaN	NaN	NaN
PitchSatisfactionScore	20.0	2.0	3.0	12.0	NaN	NaN	NaN
OwnCar	20.0	1.0	1.0	20.0	NaN	NaN	NaN
NumberOfChildrenVisiting	20.0	4.0	1.0	9.0	NaN	NaN	NaN
Designation	20	1	AVP	20	NaN	NaN	NaN
MonthlyIncome	20.0	NaN	NaN	NaN	29823.8	3520.426404	21151.0

- Average monthly income of customers opting for the super deluxe package is ~29,823.
- Average age for customers opting for the super deluxe package is ~43
- Majority of the customers opting for the super deluxe package are at AVP designation

- Majority of the customers opting for the super deluxe package are single

King

In [37]:
```
data[(data['ProductPitched']=='King') & (data['ProdTaken']==1)].describe(include='a
```

Out[37]:

	count	unique	top	freq	mean	std	min
ProdTaken	20.0	NaN	NaN	NaN	1.0	0.0	1.0
Age	20.0	NaN	NaN	NaN	48.9	9.618513	27.0
TypeofContact	20	1	Self Enquiry	20	NaN	NaN	NaN
CityTier	20.0	2.0	1.0	12.0	NaN	NaN	NaN
DurationOfPitch	20.0	NaN	NaN	NaN	10.5	4.135851	8.0
Occupation	20	3	Small Business	12	NaN	NaN	NaN
Gender	20	2	Female	12	NaN	NaN	NaN
NumberOfPersonVisiting	20.0	3.0	3.0	10.0	NaN	NaN	NaN
NumberOfFollowups	20.0	4.0	3.0	6.0	NaN	NaN	NaN
ProductPitched	20	1	King	20	NaN	NaN	NaN
PreferredPropertyStar	16.0	3.0	4.0	8.0	NaN	NaN	NaN
MaritalStatus	20	3	Single	8	NaN	NaN	NaN
NumberOfTrips	17.0	NaN	NaN	NaN	3.411765	1.938389	1.0
Passport	20.0	2.0	1.0	12.0	NaN	NaN	NaN
PitchSatisfactionScore	20.0	5.0	3.0	8.0	NaN	NaN	NaN
OwnCar	20.0	2.0	1.0	18.0	NaN	NaN	NaN
NumberOfChildrenVisiting	16.0	4.0	1.0	7.0	NaN	NaN	NaN
Designation	20	1	VP	20	NaN	NaN	NaN
MonthlyIncome	20.0	NaN	NaN	NaN	34672.1	5577.603833	17517.0

- Average monthly income of customers opting for the king package is ~34,672.
- Average age for customers opting for the king package is ~49
- Majority of the customers opting for the king package are at VP designation
- Majority of the customers opting for the king package are single

In [38]:
```
# create a dictionary to store the customer profiles for each package
package_profiles = {}

# iterate over the packages and create a profile for each
for package in data["ProductPitched"].unique():
```

```python
    package_df = data[data["ProductPitched"] == package]  # filter the dataframe by
    profile = {}
    profile["Average Age"] = package_df["Age"].mean()
    profile["Gender Distribution"] = (
        package_df["Gender"].value_counts(normalize=True).to_dict()
    )
    profile["Product Taken Distribution"] = (
        package_df["ProdTaken"].value_counts(normalize=True).to_dict()
    )
    profile["Occupation Distribution"] = (
        package_df["Occupation"].value_counts(normalize=True).to_dict()
    )
    profile["Marital Status Distribution"] = (
        package_df["MaritalStatus"].value_counts(normalize=True).to_dict()
    )
    profile["Passport Distribution"] = (
        package_df["Passport"].value_counts(normalize=True).to_dict()
    )
    profile["Income Distribution"] = (
        package_df["MonthlyIncome"].describe()[["25%", "50%", "75%"]].to_dict()
    )
    package_profiles[package] = profile

# print the package profiles
for package, profile in package_profiles.items():
    print(f"Package: {package}")
    for feature, value in profile.items():
        print(f"{feature}: {value}")
    print("\n")
```

```
Package: Deluxe
Average Age: 37.382192610539065
Gender Distribution: {'Male': 0.581986143187067, 'Female': 0.418013856812933}
Product Taken Distribution: {0: 0.8822170900692841, 1: 0.11778290993071594}
Occupation Distribution: {'Salaried': 0.4722863741339492, 'Small Business': 0.456120
0923787529, 'Large Business': 0.07159353348729793, 'Free Lancer': 0.0}
Marital Status Distribution: {'Married': 0.49191685912240185, 'Divorced': 0.19399538
106235567, 'Unmarried': 0.18648960739030024, 'Single': 0.12759815242494227}
Passport Distribution: {0: 0.7228637413394919, 1: 0.27713625866050806}
Income Distribution: {'25%': 20737.75, '50%': 22922.0, '75%': 24199.25}


Package: Basic
Average Age: 33.054181389870436
Gender Distribution: {'Male': 0.6308360477741585, 'Female': 0.3691639522258415}
Product Taken Distribution: {0: 0.7003257328990228, 1: 0.2996742671009772}
Occupation Distribution: {'Salaried': 0.501628664495114, 'Small Business': 0.3908794
7882736157, 'Large Business': 0.10640608034744843, 'Free Lancer': 0.0010857763300760
044}
Marital Status Distribution: {'Married': 0.44299674267100975, 'Single': 0.2774158523
344191, 'Divorced': 0.18023887079261672, 'Unmarried': 0.0993485342019544}
Passport Distribution: {0: 0.6916395222584147, 1: 0.30836047774158526}
Income Distribution: {'25%': 17654.0, '50%': 20689.0, '75%': 21412.5}


Package: Standard
```

```
Average Age: 40.581646423751685
Gender Distribution: {'Male': 0.5606469002695418, 'Female': 0.4393530997304582}
Product Taken Distribution: {0: 0.8328840970350404, 1: 0.16711590296495957}
Occupation Distribution: {'Salaried': 0.4555256064690027, 'Small Business': 0.431266
846361186, 'Large Business': 0.11320754716981132, 'Free Lancer': 0.0}
Marital Status Distribution: {'Married': 0.5121293800539084, 'Unmarried': 0.22911051
212938005, 'Divorced': 0.19137466307277629, 'Single': 0.0673854447439353}
Passport Distribution: {0: 0.7169811320754716, 1: 0.2830188679245283}
Income Distribution: {'25%': 24860.0, '50%': 26425.0, '75%': 28716.0}


Package: Super Deluxe
Average Age: 48.026315789473685
Gender Distribution: {'Male': 0.5321637426900585, 'Female': 0.4678362573099415}
Product Taken Distribution: {0: 0.9415204678362573, 1: 0.05847953216374269}
Occupation Distribution: {'Salaried': 0.5087719298245614, 'Small Business': 0.438596
49122807015, 'Large Business': 0.05263157894736842, 'Free Lancer': 0.0}
Marital Status Distribution: {'Married': 0.4853801169590643, 'Divorced': 0.257309941
5204678, 'Single': 0.23976608187134502, 'Unmarried': 0.017543859649122806}
Passport Distribution: {0: 0.695906432748538, 1: 0.30409356725146197}
Income Distribution: {'25%': 30847.0, '50%': 32181.0, '75%': 34787.0}


Package: King
Average Age: 48.06521739130435
Gender Distribution: {'Male': 0.6434782608695652, 'Female': 0.3565217391304348}
Product Taken Distribution: {0: 0.9130434782608695, 1: 0.08695652173913043}
Occupation Distribution: {'Salaried': 0.4956521739130435, 'Small Business': 0.452173
91304347826, 'Large Business': 0.05217391304347826, 'Free Lancer': 0.0}
Marital Status Distribution: {'Married': 0.5478260869565217, 'Divorced': 0.226086956
52173913, 'Single': 0.22608695652173913, 'Unmarried': 0.0}
Passport Distribution: {0: 0.7391304347826086, 1: 0.2608695652173913}
Income Distribution: {'25%': 34202.0, '50%': 34999.0, '75%': 37880.0}
```

Data Preprocessing (contd.)

Outlier Detection and Treatment

In [39]:

```
100*data.NumberOfTrips.value_counts(normalize=True)
```

Out[39]:

```
2.0     30.834035
3.0     22.725358
1.0     13.058130
4.0     10.067397
5.0      9.646167
6.0      6.781803
7.0      4.591407
8.0      2.211457
20.0     0.021061
19.0     0.021061
22.0     0.021061
```

```
21.0    0.021061
Name: NumberOfTrips, dtype: float64
```

- We can see that most of the customers i.e. 52% have taken 2 or 3 trips.
- As expected, with the increase in the number of trips the percentage of customers is decreasing.
- The percentage of categories 19 or above is very less. We can consider these values as outliers.
- We can see that there are just four observations with a number of trips 8 or greater

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

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

#Dropping observation with monthly income less than 12000 or greater than 40000. Th
data.drop(index=data[(data.MonthlyIncome>40000) | (data.MonthlyIncome<12000)].index

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

Data Preparation for Modeling

In [41]:
```
#Separating target variable and other variables
X=data.drop(columns='ProdTaken')
Y=data['ProdTaken']
```

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

In [42]:
```
#Dropping columns
X.drop(columns=['DurationOfPitch','NumberOfFollowups','ProductPitched','PitchSatisf
```

In [43]:
```
#Splitting the data into train and test sets
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.30,random_state=1,st
```

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

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

In [44]:
```
si1=SimpleImputer(strategy='median')
```

```python
median_imputed_col=['Age','MonthlyIncome','NumberOfTrips']

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

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

In [45]:
```python
si2=SimpleImputer(strategy='most_frequent')

mode_imputed_col=['TypeofContact','PreferredPropertyStar','NumberOfChildrenVisiting

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

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

In [46]:
```python
#Checking that no column has missing values in train or test sets
print(X_train.isna().sum())
print('-'*30)
print(X_test.isna().sum())
```

```
Age                         0
TypeofContact               0
CityTier                    0
Occupation                  0
Gender                      0
NumberOfPersonVisiting      0
PreferredPropertyStar       0
MaritalStatus               0
NumberOfTrips               0
Passport                    0
OwnCar                      0
NumberOfChildrenVisiting    0
Designation                 0
MonthlyIncome               0
dtype: int64
------------------------------
Age                         0
TypeofContact               0
CityTier                    0
Occupation                  0
Gender                      0
NumberOfPersonVisiting      0
PreferredPropertyStar       0
MaritalStatus               0
NumberOfTrips               0
Passport                    0
OwnCar                      0
NumberOfChildrenVisiting    0
Designation                 0
MonthlyIncome               0
dtype: int64
```

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

In [47]:
```
#converting data types of columns to float
for column in ['NumberOfPersonVisiting', 'Passport', 'OwnCar']:
    X_train[column]=X_train[column].astype('float')
    X_test[column]=X_test[column].astype('float')
```

In [48]:
```
#List of columns to create a dummy variables
col_dummy=['TypeofContact', 'Occupation', 'Gender', 'MaritalStatus', 'Designation',
```

In [49]:
```
#Encoding categorical varaibles
X_train=pd.get_dummies(X_train, columns=col_dummy, drop_first=True)
X_test=pd.get_dummies(X_test, columns=col_dummy, drop_first=True)
```

Model Building

Model Evaluation Criterion

The model can make wrong predictions as:

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

Which case is more important?

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

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

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

Let's create two functions to calculate different metrics and confusion matrix, so that we don't have to use the same code repeatedly for each model.

In [50]:
```
# defining a function to compute different metrics to check performance of a classi
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
```

```python
    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1": f1,
        },
        index=[0],
    )

    return df_perf
```

In [51]:
```python
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().sum())
            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")
```

Model Building: Decision Tree

In [52]:
```python
#Fitting the model
d_tree = DecisionTreeClassifier(random_state=1)
d_tree.fit(X_train,y_train)
```

Out[52]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=1)
```

Checking model performance on the training data

```
In [53]: #Calculating different metrics on training data
d_tree_model_train_perf=model_performance_classification_sklearn(d_tree, X_train,y_
print("Training performance:\n", d_tree_model_train_perf)
```

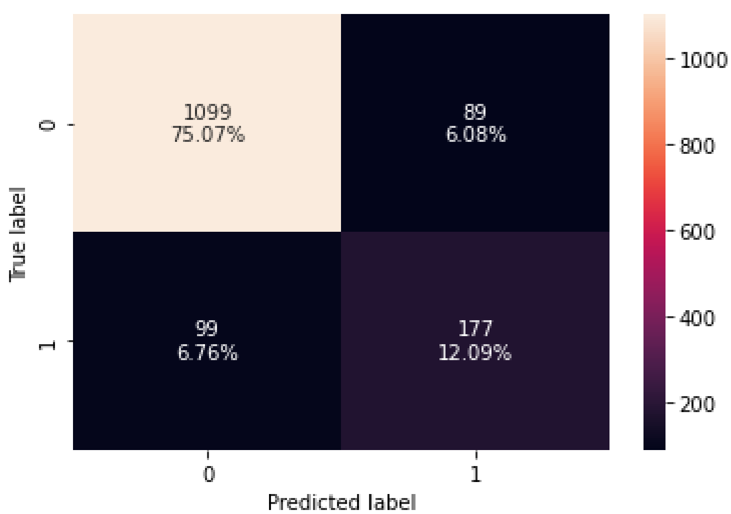
```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

Checking model performance on the test data

```
In [54]: #Calculating different metrics on test data
d_tree_model_test_perf=model_performance_classification_sklearn(d_tree, X_test,y_te
print("Testing performance:\n", d_tree_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.871585  0.641304   0.665414  0.653137
```

```
In [55]: # Creating confusion matrix on test data
confusion_matrix_sklearn(d_tree,X_test,y_test)
```



- The model is overfitting the training data as training recall/precision is much higher than the test recall/precision

Model Improvement: Decision Tree

```
In [56]: #Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight={0:0.18,1:0.72},random_state=

# 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[56]:

```
DecisionTreeClassifier

DecisionTreeClassifier(class_weight={0: 0.18, 1: 0.72}, max_depth=
5,
                       max_leaf_nodes=15, min_impurity_decrease=0.0
001,
                       min_samples_leaf=10, random_state=1)
```

Checking model performance on the training data

In [57]:

```
# Calculating different metrics on training data
dtree_estimator_model_train_perf=model_performance_classification_sklearn(dtree_est
print("Training performance:\n", dtree_estimator_model_train_perf)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.803456  0.663551   0.483541  0.559422
```

Checking model performance on the test data

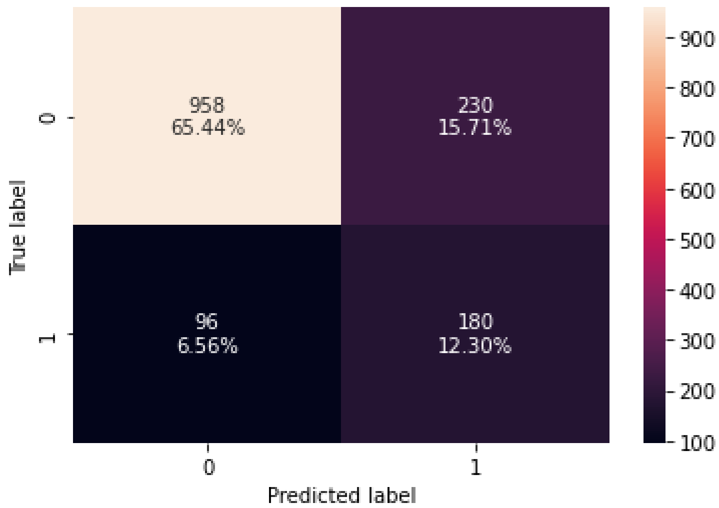
In [58]:

```
# Calculating different metrics on test data
dtree_estimator_model_test_perf=model_performance_classification_sklearn(dtree_esti
print("Testing performance:\n", dtree_estimator_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.777322  0.652174   0.439024  0.524781
```

In [59]:

```
# Creating confusion matrix on test data
confusion_matrix_sklearn(dtree_estimator,X_test,y_test)
```



- The model is generalizing well and not overfitting the data
- The recall is still similar on the test data but the precision has decreased significantly.

Model Building: Random Forest

In [60]:
```
# Initializing the model
rf_estimator = RandomForestClassifier(random_state=1)
rf_estimator.fit(X_train,y_train)
```

Out[60]:
```
▾           RandomForestClassifier
RandomForestClassifier(random_state=1)
```

Checking model performance on the training data

In [61]:
```
# Calculating different metrics on training data
rf_estimator_model_train_perf=model_performance_classification_sklearn(rf_estimator
print("Training performance:\n",rf_estimator_model_train_perf)
```

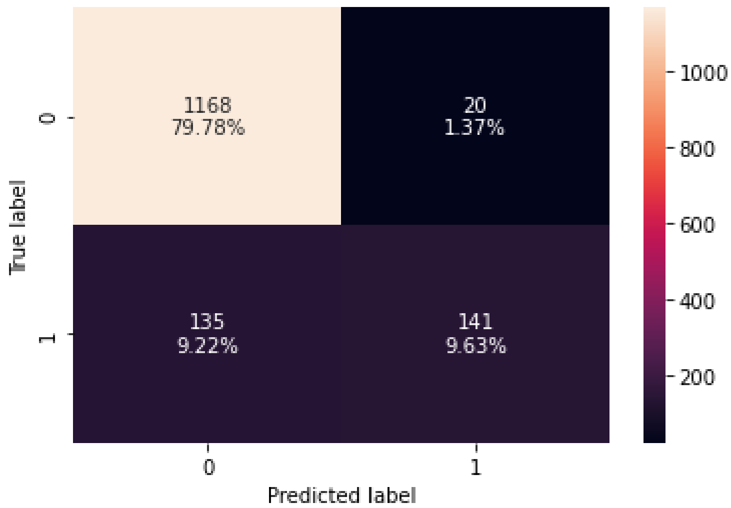
```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

Checking model performance on the test data

In [62]:
```
# Calculating different metrics on test data
rf_estimator_model_test_perf=model_performance_classification_sklearn(rf_estimator,
print("Testing performance:\n",rf_estimator_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.894126  0.51087   0.875776  0.645309
```

In [63]:
```
# Creating confusion matrix on test data
confusion_matrix_sklearn(rf_estimator,X_test,y_test)
```



- With default parameters, random forest is performing better than decision tree in terms of precision but has less recall.
- The model is overfitting the training data.
- We'll try to reduce overfitting and improve recall by hyperparameter tuning.

Model Improvement: Random Forest

In [64]:
```python
# Choose the type of classifier.
rf_tuned = RandomForestClassifier(class_weight={0:0.18,1:0.82},random_state=1,oob_s

parameters = {
                'max_depth': list(np.arange(5,30,5)) + [None],
                'max_features': ['sqrt','log2',None],
                'min_samples_leaf': np.arange(1,15,5),
                'min_samples_split': np.arange(2, 20, 5),
                'n_estimators': np.arange(10,110,10)}


# Run the grid search
grid_obj = GridSearchCV(rf_tuned, parameters, scoring='recall',cv=5,n_jobs=-1)
grid_obj = grid_obj.fit(X_train, y_train)

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

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

Out[64]:
```
▾                    RandomForestClassifier
RandomForestClassifier(class_weight={0: 0.18, 1: 0.82}, max_depth=1
5,
                       max_features=None, min_samples_leaf=11, n_es
timators=60,
                       oob_score=True, random_state=1)
```

Checking model performance on the training data

In [65]:
```
# Calculating different metrics on training data
rf_tuned_model_train_perf=model_performance_classification_sklearn(rf_tuned, X_trai
print("Training performance:\n",rf_tuned_model_train_perf)
```

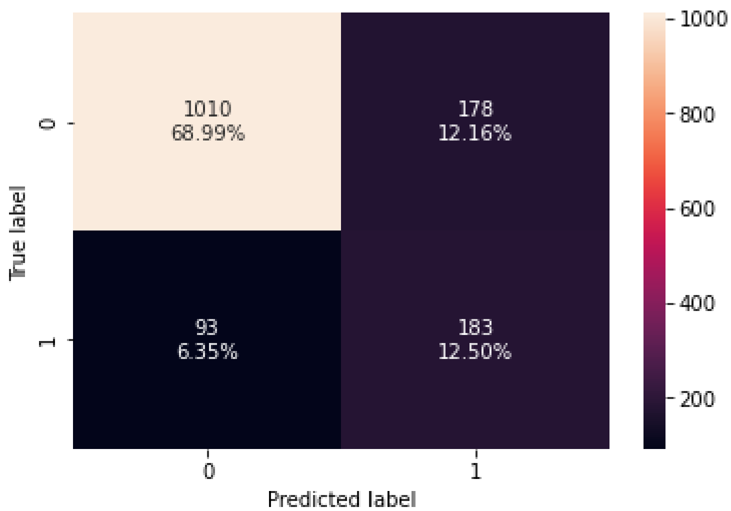
```
Training performance:
    Accuracy    Recall   Precision        F1
0   0.89133  0.88162   0.657375  0.75316
```

Checking model performance on the test data

In [66]:
```
# Calculating different metrics on test data
rf_tuned_model_test_perf=model_performance_classification_sklearn(rf_tuned, X_test,
print("Testing performance:\n",rf_tuned_model_test_perf)
```

```
Testing performance:
    Accuracy     Recall   Precision        F1
0  0.814891  0.663043   0.506925  0.574568
```

In [67]:
```
# Creating confusion matrix on test data
confusion_matrix_sklearn(rf_tuned,X_test,y_test)
```



- The overfitting has been reduced after tuning the model.
- The recall has improved on the test data but the precision has decreased significantly.

Model Building: Bagging

In [68]:
```
# Initializing the Bagging classifier
bagging_classifier = BaggingClassifier(random_state=1)
bagging_classifier.fit(X_train,y_train)
```

Out[68]:
```
▾          BaggingClassifier
BaggingClassifier(random_state=1)
```

Checking model performance on the training data

In [69]:
```
# Calculating different metrics on training data
bagging_classifier_model_train_perf=model_performance_classification_sklearn(baggin
print("Training performance:\n",bagging_classifier_model_train_perf)
```

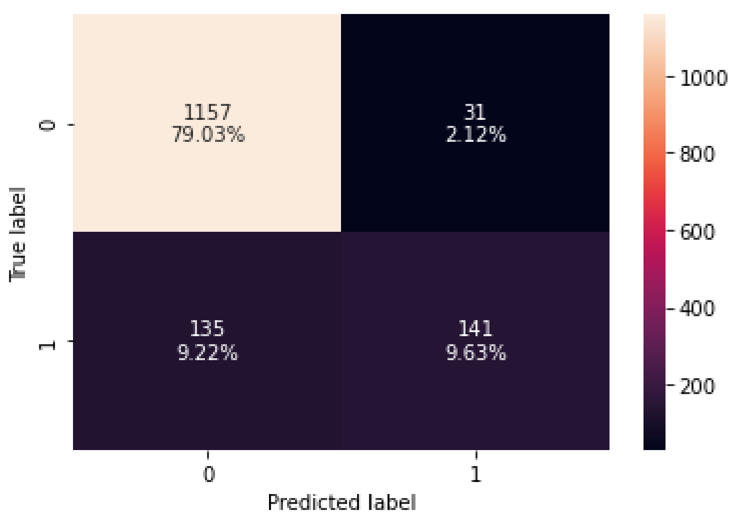
```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.990334  0.951713   0.996737  0.973705
```

Checking model performance on the test data

In [70]:
```
# Calculating different metrics on test data
bagging_classifier_model_test_perf=model_performance_classification_sklearn(bagging
print("Testing performance:\n",bagging_classifier_model_test_perf)
```

```
Testing performance:
    Accuracy   Recall  Precision        F1
0  0.886612  0.51087   0.819767  0.629464
```

In [71]:
```
# Creating confusion matrix on test
confusion_matrix_sklearn(bagging_classifier,X_test,y_test)
```



- With default parameters, the bagging classifier is performing well in terms of precision but has less recall.
- The model is overfitting the training data.
- We'll try to reduce overfitting and improve recall by hyperparameter tuning.

Model Improvement: Bagging

In [72]:
```
# Choose the type of classifier.
bagging_estimator_tuned = BaggingClassifier(random_state=1)

# 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
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(bagging_estimator_tuned, parameters, scoring=acc_scorer,cv=
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[72]:

```
BaggingClassifier

BaggingClassifier(max_features=0.9, max_samples=0.9, n_estimators=5
0,
                  random_state=1)
```

Checking model performance on the training data

In [73]:

```
# Calculating different metrics on training data
bagging_estimator_tuned_model_train_perf=model_performance_classification_sklearn(b
print("Training performance:\n",bagging_estimator_tuned_model_train_perf)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.999121  0.995327        1.0  0.997658
```

Checking model performance on the test data

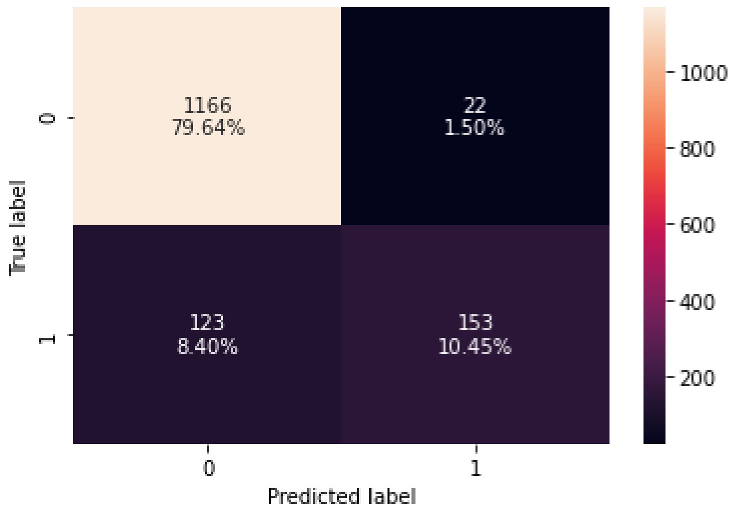
In [74]:

```
# Calculating different metrics on test data
bagging_estimator_tuned_model_test_perf=model_performance_classification_sklearn(ba
print("Testing performance:\n",bagging_estimator_tuned_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.900956  0.554348   0.874286  0.678492
```

In [75]:

```
# Creating confusion matrix on test data
confusion_matrix_sklearn(bagging_estimator_tuned,X_test,y_test)
```



- The test recall and test precision have improved but the model is still overfitting the training data.
- The recall is still very low.

Model Building: AdaBoost

In [76]:
```
# Initializing the AdaBoost classifier model
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train,y_train)
```

Out[76]:
```
▾          AdaBoostClassifier
AdaBoostClassifier(random_state=1)
```

Checking model performance on the training data

In [77]:
```
# Calculating different metrics on training data
ab_classifier_model_train_perf=model_performance_classification_sklearn(ab_classifi
print("Training performance:\n",ab_classifier_model_train_perf)
```

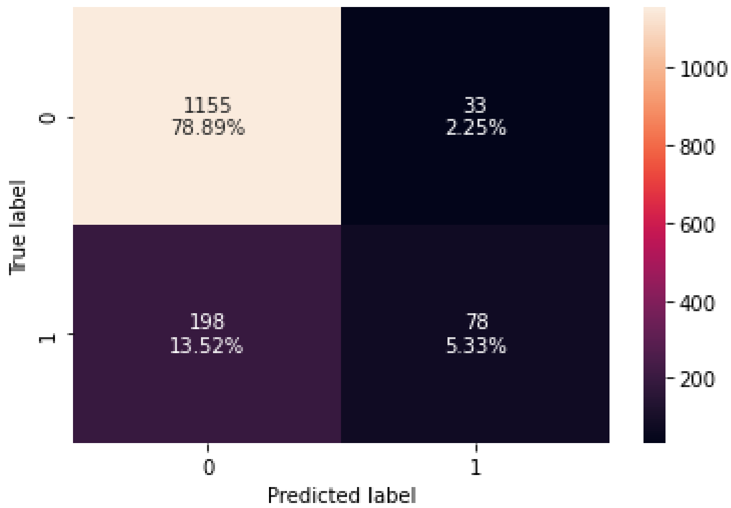
```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.845343  0.299065   0.711111  0.421053
```

Checking model performance on the test data

In [78]:
```
# Calculating different metrics on test data
ab_classifier_model_test_perf=model_performance_classification_sklearn(ab_classifie
print("Testing performance:\n",ab_classifier_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.842213  0.282609   0.702703  0.403101
```

In [79]:
```
# Creating confusion matrix on test data
confusion_matrix_sklearn(ab_classifier,X_test,y_test)
```



- The model is not overfitting the data but is giving very low recall on training and test data.

Model Improvement: AdaBoost

In [80]:
```
# 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(ma
                      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[80]:
```
▸ AdaBoostClassifier
▸ base_estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

Checking model performance on the training data

In [81]:
```
# Calculating different metrics on training data
abc_tuned_model_train_perf=model_performance_classification_sklearn(abc_tuned, X_tr
print("Training performance:\n",abc_tuned_model_train_perf)
```

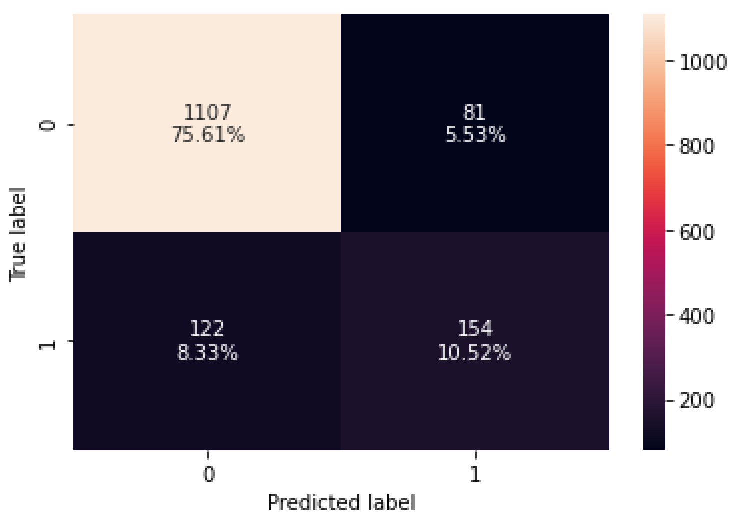
```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.983304  0.928349   0.981878  0.954363
```

Checking model performance on the test data

In [82]:
```
# Calculating different metrics on test data
abc_tuned_model_test_perf=model_performance_classification_sklearn(abc_tuned, X_tes
print("Testing performance:\n",abc_tuned_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision       F1
0  0.861339  0.557971   0.655319  0.60274
```

In [83]:
```
# Creating confusion matrix on test data
confusion_matrix_sklearn(abc_tuned,X_test,y_test)
```



- The train, as well as test recall, have improved significantly but the model is overfitting the training data now.

Model Building: Gradient Boosting

In [84]:
```
# Initializing the Gradient bossting classifier
gb_classifier = GradientBoostingClassifier(random_state=1)
gb_classifier.fit(X_train,y_train)
```

Out[84]:
```
▾          GradientBoostingClassifier
GradientBoostingClassifier(random_state=1)
```

Checking model performance on the training data

In [85]:
```
# Calculating different metrics on training data
gb_classifier_model_train_perf=model_performance_classification_sklearn(gb_classifi
print("Training performance:\n",gb_classifier_model_train_perf)
```

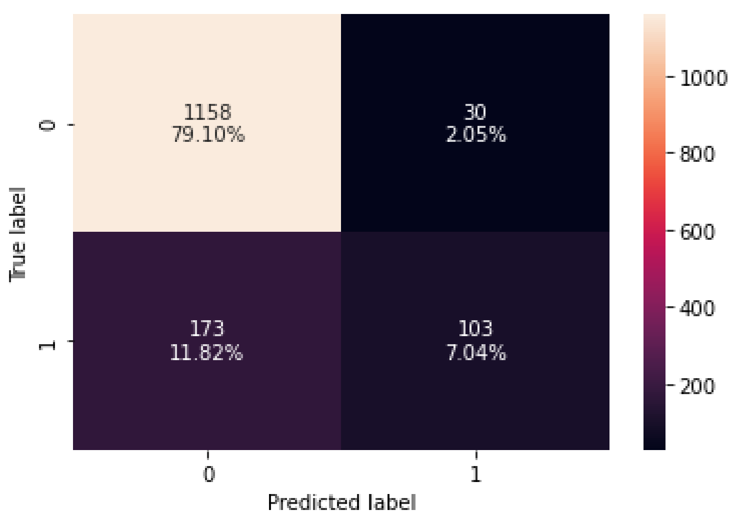
```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.878735  0.433022   0.847561  0.573196
```

Checking model performance on the test data

In [86]:
```
# Calculating different metrics on test data
gb_classifier_model_test_perf=model_performance_classification_sklearn(gb_classifie
print("Testing performance:\n",gb_classifier_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.861339  0.373188   0.774436  0.503667
```

In [87]:
```
#Creating confusion matrix
confusion_matrix_sklearn(gb_classifier,X_test,y_test)
```



- The model is slightly overfitting the training data in terms of recall and precision but is giving very low recall on training and test data.
- The recall is better as compared to AdaBoost with default parameters but still not great.

Model Improvement: Gradient Boosting

In [88]:
```
# Choose the type of classifier.
gbc_tuned = GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),rand

# 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
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[88]:

```
▸ GradientBoostingClassifier
  ▸ init: AdaBoostClassifier
    ▸ AdaBoostClassifier
```

Checking model performance on the training data

In [89]:

```
# Calculating different metrics on training data
gbc_tuned_model_train_perf=model_performance_classification_sklearn(gbc_tuned, X_tr
print("Training performance:\n",gbc_tuned_model_train_perf)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.911541  0.590343   0.906699  0.715094
```

Checking model performance on the test data

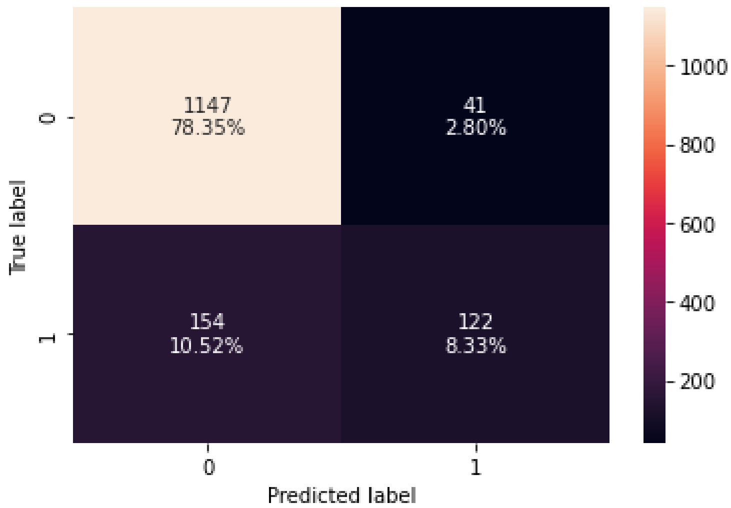
In [90]:

```
# Calculating different metrics on test data
gbc_tuned_model_test_perf=model_performance_classification_sklearn(gbc_tuned, X_tes
print("Testing performance:\n",gbc_tuned_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.866803  0.442029   0.748466  0.555809
```

In [91]:

```
# Creating confusion matrix on test data
confusion_matrix_sklearn(gbc_tuned,X_test,y_test)
```



- The model performance has improved slightly after hyperparameter tuning but the model is still overfitting the training data.
- The test precision has decreased slightly and the test recall has increased slightly but still very low.

Model Building: XGBoost

In [92]:
```
# Initializing the XGBoost model
xgb_classifier = XGBClassifier(random_state=1, eval_metric= "error")
xgb_classifier.fit(X_train.astype('int'),y_train)
```

Out[92]:
```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='error', featur
e_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance
_type=None,
              interaction_constraints=None, learning_rate=None, max
_bin=None,
```

Checking model performance on the training data

In [93]:
```
# Calculating different metrics on training data
xgb_classifier_model_train_perf=model_performance_classification_sklearn(xgb_classi
print("Training performance:\n",xgb_classifier_model_train_perf)
```

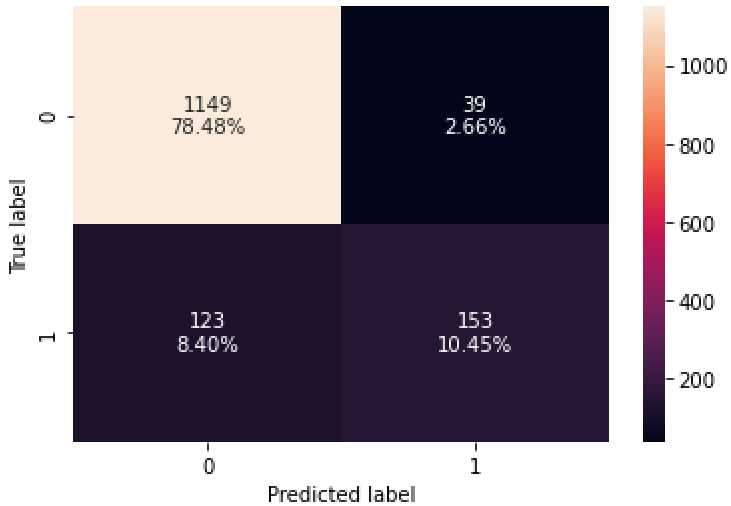
```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.994728  0.971963        1.0  0.985782
```

Checking model performance on the test data

```
In [94]: # Calculating different metrics on test data
xgb_classifier_model_test_perf=model_performance_classification_sklearn(xgb_classif
print("Testing performance:\n",xgb_classifier_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.889344  0.554348   0.796875  0.653846
```

```
In [95]: # Creating confusion matrix on test data
confusion_matrix_sklearn(xgb_classifier,X_test.astype('int'),y_test)
```



- With default parameters, the model is overfitting the training data.
- The model is not able to correctly identify potential customers i.e. the test recall is very low.

Model Improvement: XGBoost (optional)

```
In [96]: # 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": [0, 1],
    "subsample": [0.5, 0.9],
    "learning_rate": [0.1, 0.2],
    "gamma": [0, 1],
    "colsample_bytree": [0.5, 0.9],
    "colsample_bylevel": [0.5, 0.9]
}

# Type of scoring used to compare parameter combinations
scoring = ['accuracy', 'recall']

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned, parameters, scoring=scoring, cv=3, refit='recall
```

```
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[96]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=0.9, colsample_bynode=None,
              colsample_bytree=0.9, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=0, gpu_id=None, grow_policy
=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.2, max_bin=None, max_cat_threshold=No
ne,
```

In [97]:

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

# Grid of parameters to choose from
parameters = {
    "n_estimators": [10, 20, 30],
    "scale_pos_weight": [0, 1],
    "subsample": [0.7, 0.9],
    "learning_rate": [0.1, 0.2],
    "gamma": [0, 1],
    "colsample_bytree": [0.7, 0.9],
    "colsample_bylevel": [0.7, 0.9]
}

# Type of scoring used to compare parameter combinations
scoring = ['accuracy', 'recall']

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned, parameters, scoring=scoring, cv=3, refit='recall
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[97]:

```
▾                    XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=0.7, colsample_bynode=None,
              colsample_bytree=0.9, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=1, gpu_id=None, grow_policy
=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.2, max_bin=None, max_cat_threshold=No
ne,
```

Checking model performance on the training data

In [98]:
```
# Calculating different metrics on training data
xgb_tuned_model_train_perf=model_performance_classification_sklearn(xgb_tuned, X_tr
print("Training performance:\n",xgb_tuned_model_train_perf)
```

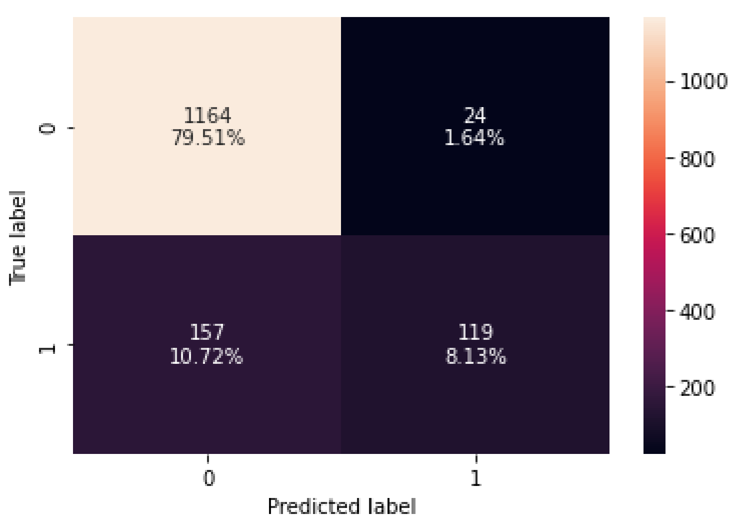
```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.923843  0.615265   0.968137  0.752381
```

Checking model performance on the test data

In [99]:
```
# Calculating different metrics on test data
xgb_tuned_model_test_perf=model_performance_classification_sklearn(xgb_tuned, X_tes
print("Testing performance:\n",xgb_tuned_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.876366  0.431159   0.832168  0.568019
```

In [100…
```
# Creating confusion matrix on test data
confusion_matrix_sklearn(xgb_tuned,X_test.astype('int'),y_test)
```



- The overfitting has reduced after hyperparameter tuning but is still an overfit model.

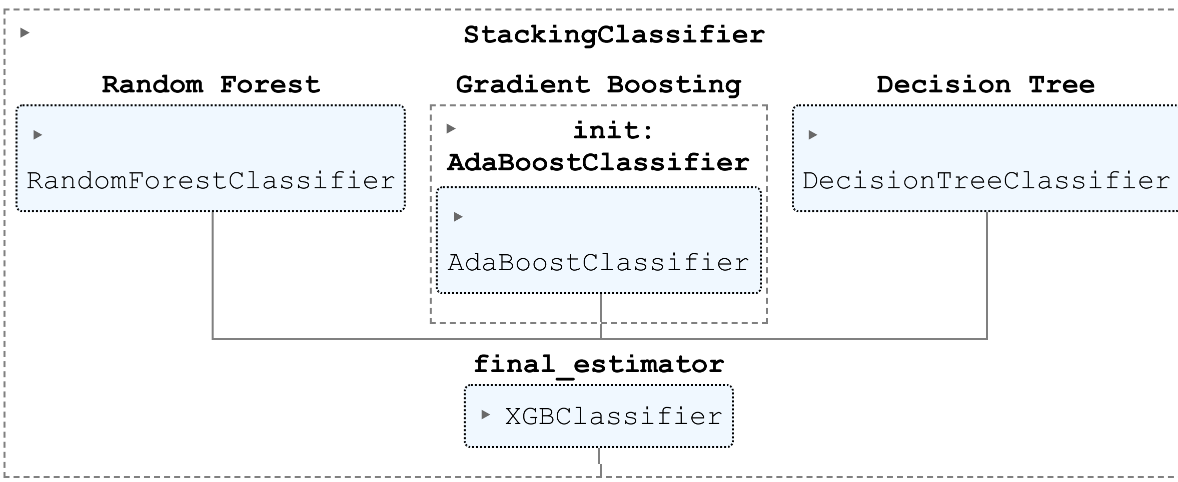
- The tuned xgboost model is giving the highest recall yet among all the models we built.
- Let's try one more model - Stacking classifier.

Model Building: Stacking

- Stacking classifier stacks the output of individual estimators and use a classifier to compute the final prediction
- Stacking allows using the strength of each estimator by using their output as input of a final estimator

In [101…
```
estimators = [('Random Forest',rf_tuned), ('Gradient Boosting',gbc_tuned), ('Decisi

final_estimator = xgb_tuned

stacking_classifier= StackingClassifier(estimators=estimators,final_estimator=final

stacking_classifier.fit(X_train,y_train)
```

Out[101]:



Checking model performance on the training data

In [102…
```
# Calculating different metrics on training data
stacking_classifier_model_train_perf=model_performance_classification_sklearn(stack
print("Training performance:\n",stacking_classifier_model_train_perf)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.926479  0.690031   0.894949  0.779244
```

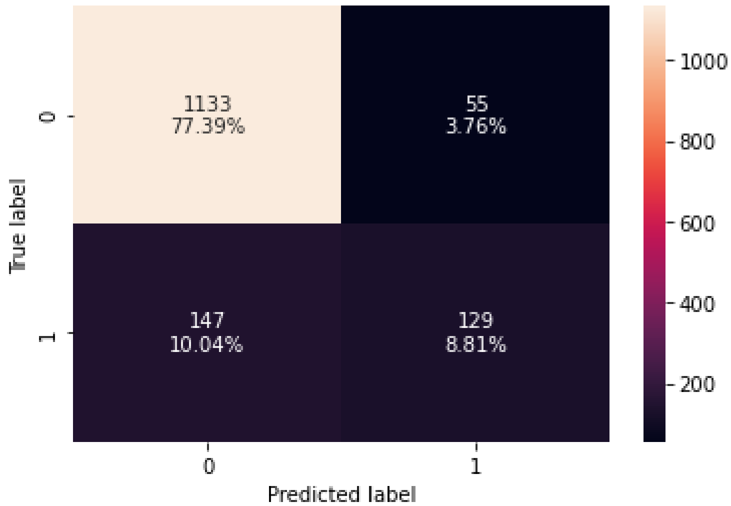
Checking model performance on the test data

In [103…
```
# Calculating different metrics on test data
stacking_classifier_model_test_perf=model_performance_classification_sklearn(stacki
print("Testing performance:\n",stacking_classifier_model_test_perf)
```

```
Testing performance:
    Accuracy    Recall  Precision       F1
0  0.862022  0.467391   0.701087  0.56087
```

In [104…

```
# Creating confusion matrix on test data
confusion_matrix_sklearn(stacking_classifier,X_test,y_test)
```



- Stacking classifier has further increased the recall that we got from the xgboost model but reduced the precision as well.
- Model is overfitting the training data.

Model Comparison and Final Model Selection

In [105…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [d_tree_model_train_perf.T, dtree_estimator_model_train_perf.T, rf_estimator_mo
    rf_tuned_model_train_perf.T,bagging_classifier_model_train_perf.T,bagging_estim
     abc_tuned_model_train_perf.T,gb_classifier_model_train_perf.T,gbc_tuned_model_
     xgb_tuned_model_train_perf.T, stacking_classifier_model_train_perf.T],
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree",
    "Decision Tree Estimator",
    "Random Forest Estimator",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Estimator Tuned",
    "Adaboost Classifier",
    "Adaboost Classifier Tuned",
     "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
     "XGBoost Classifier",
    "XGBoost Classifier Tuned","Stacking Classifier"]
print("Training performance comparison:")

models_train_comp_df
```

```
Training performance comparison:
```

Out[105]:

	Decision Tree	Decision Tree Estimator	Random Forest Estimator	Random Forest Tuned	Bagging Classifier	Bagging Estimator Tuned	Adaboost Classifier	Adabo Classi Tu
Accuracy	1.0	0.803456	1.0	0.891330	0.990334	0.999121	0.845343	0.983
Recall	1.0	0.663551	1.0	0.881620	0.951713	0.995327	0.299065	0.928
Precision	1.0	0.483541	1.0	0.657375	0.996737	1.000000	0.711111	0.981
F1	1.0	0.559422	1.0	0.753160	0.973705	0.997658	0.421053	0.954

In [106…

```python
# Testing performance comparison

models_test_comp_df = pd.concat(
    [d_tree_model_test_perf.T, dtree_estimator_model_test_perf.T, rf_estimator_mode
    rf_tuned_model_test_perf.T,bagging_classifier_model_test_perf.T,bagging_estimat
     abc_tuned_model_test_perf.T,gb_classifier_model_test_perf.T,gbc_tuned_model_te
     xgb_tuned_model_test_perf.T, stacking_classifier_model_test_perf.T],
    axis=1,
)
models_test_comp_df.columns = [
    "Decision Tree",
    "Decision Tree Estimator",
    "Random Forest Estimator",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Estimator Tuned",
    "Adaboost Classifier",
    "Adaboost Classifier Tuned",
     "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
     "XGBoost Classifier",
    "XGBoost Classifier Tuned","Stacking Classifier"]
print("Testing performance comparison:")

models_test_comp_df
```

Testing performance comparison:

Out[106]:

	Decision Tree	Decision Tree Estimator	Random Forest Estimator	Random Forest Tuned	Bagging Classifier	Bagging Estimator Tuned	Adaboost Classifier	Adabo Classi Tu
Accuracy	0.871585	0.777322	0.894126	0.814891	0.886612	0.900956	0.842213	0.861
Recall	0.641304	0.652174	0.510870	0.663043	0.510870	0.554348	0.282609	0.557
Precision	0.665414	0.439024	0.875776	0.506925	0.819767	0.874286	0.702703	0.655
F1	0.653137	0.524781	0.645309	0.574568	0.629464	0.678492	0.403101	0.602

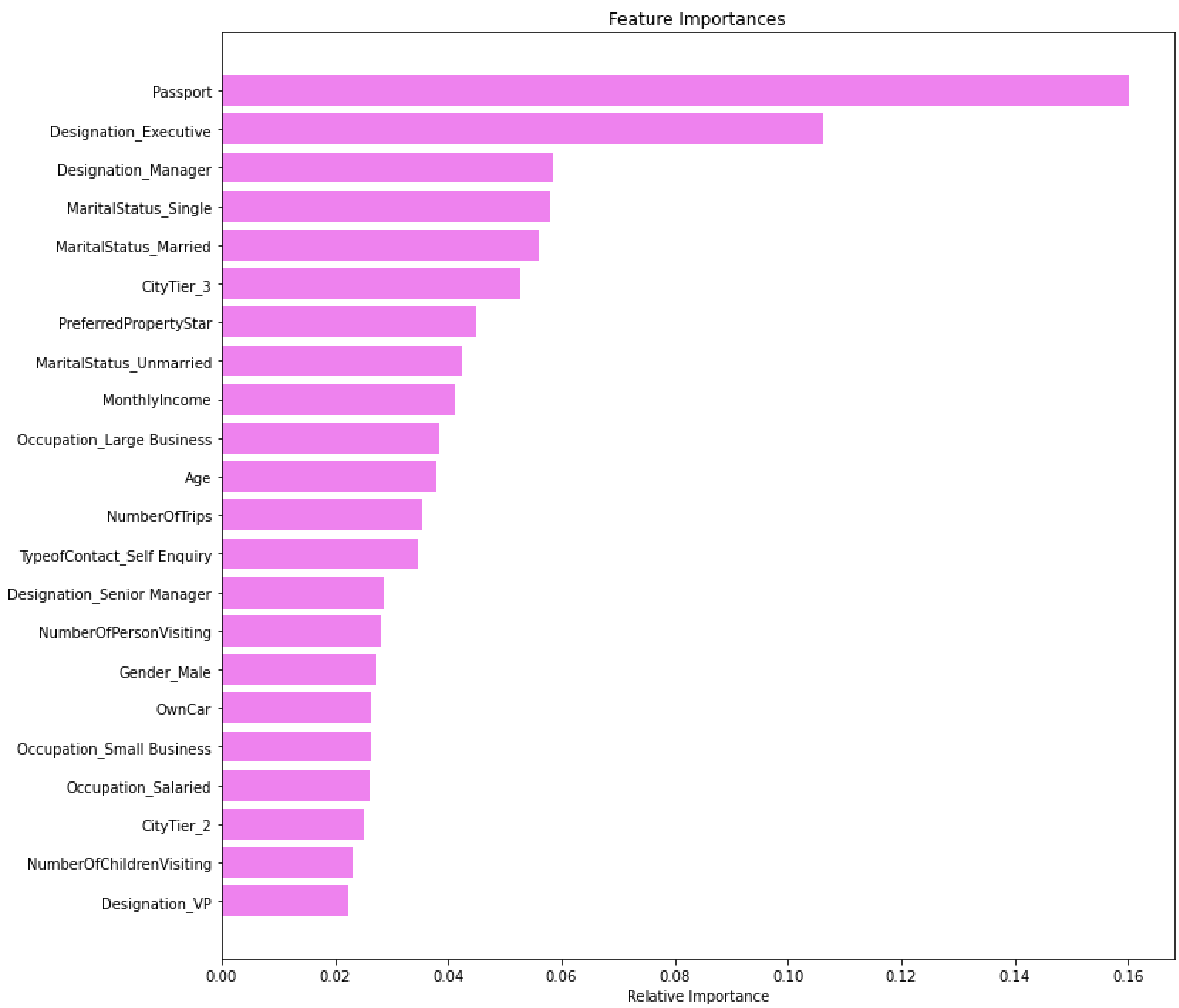
- Stacking classifier is giving the best test recall but it has no attribute to calculate feature importance.

- Tuned xgboost is giving the second-highest test recall but the tuned decision tree is giving a generalized performance on the train and the test set. We have used xgboost model to demonstrate the calculation of the feature importance.

Feature Importance for best model

In [107…

```
feature_names = X_train.columns
importances = xgb_tuned.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='center')
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 designation, marital status, and city tier, as per the tuned xgboost model.

Actionable Insights and Business Recommendations

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

Appendix: Detailed Exploratory Data Analysis (EDA)

Univariate Analysis

In [108…

```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
```

```
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the col
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [109…

```
df.columns
```

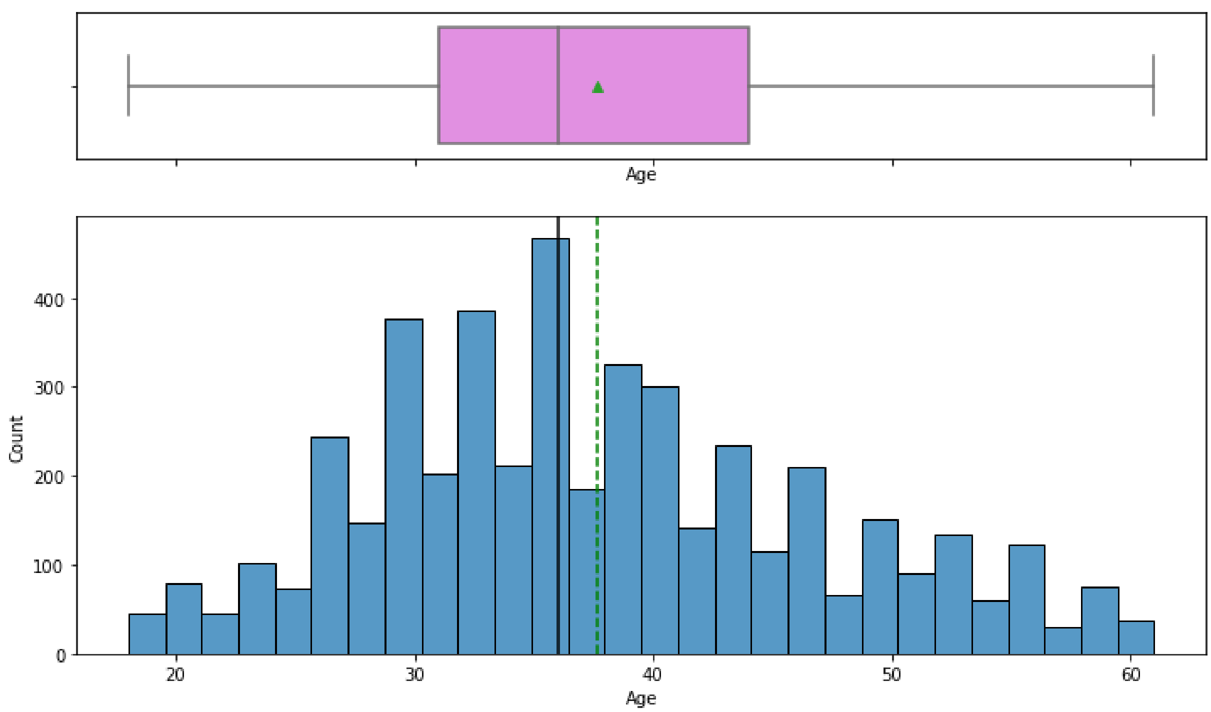
Out[109]:

```
Index(['ProdTaken', 'Age', 'TypeofContact', 'CityTier', 'DurationOfPitch',
       'Occupation', 'Gender', 'NumberOfPersonVisiting', 'NumberOfFollowups',
       'ProductPitched', 'PreferredPropertyStar', 'MaritalStatus',
       'NumberOfTrips', 'Passport', 'PitchSatisfactionScore', 'OwnCar',
       'NumberOfChildrenVisiting', 'Designation', 'MonthlyIncome'],
      dtype='object')
```

Observations on Age

In [110…

```
histogram_boxplot(df, "Age")
```



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

In [111…

```
df.NumberOfTrips.value_counts(normalize=True)
```

Out[111]:

```
2.0     0.308340
3.0     0.227254
1.0     0.130581
4.0     0.100674
5.0     0.096462
6.0     0.067818
7.0     0.045914
8.0     0.022115
20.0    0.000211
19.0    0.000211
22.0    0.000211
21.0    0.000211
Name: NumberOfTrips, dtype: float64
```

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

In [112…

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

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

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

Let's define a function to create barplots for the categorical variables indicating percentage of each category for that variables.

In [113…

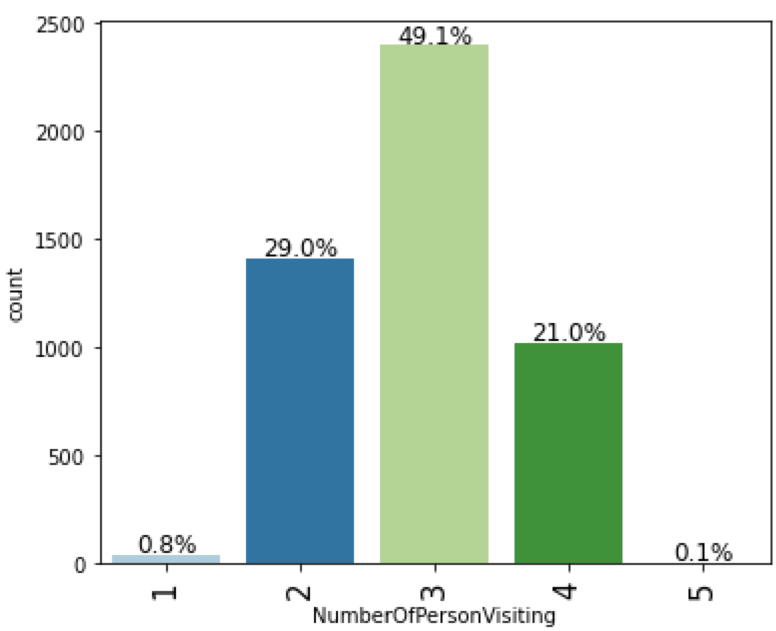
```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all level
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
```

```
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

Observations on Number of Person Visiting

In [114…

```
labeled_barplot(df, "NumberOfPersonVisiting", perc=True)
```

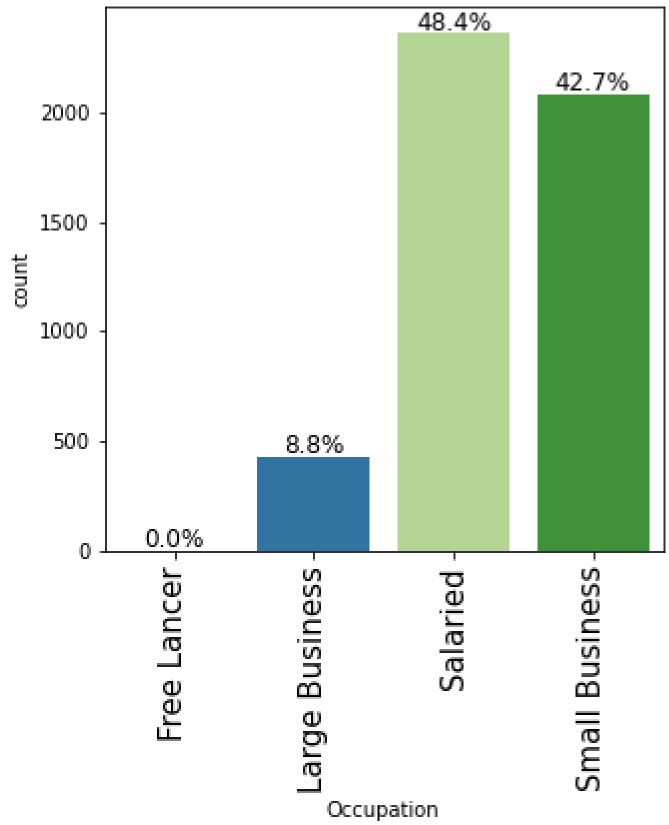


- Most customers have 3 persons who are visiting with them. This can be because most people like to travel with family.

- As mentioned earlier, there are just 3 observations where the number of persons visiting with the customers are 5 i.e. 0.1%.

Observations on Occupation

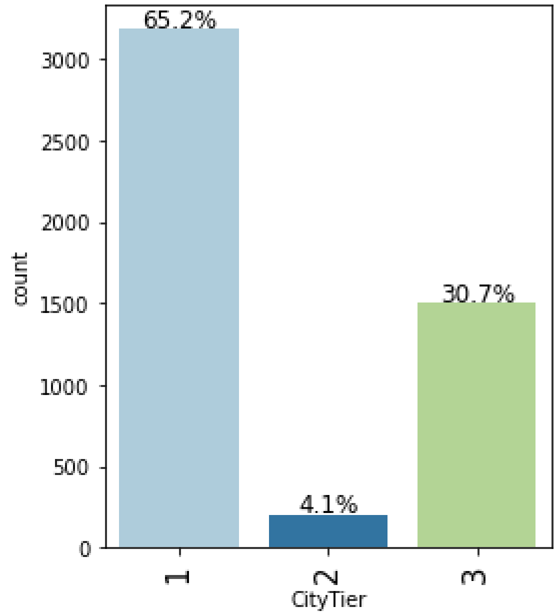
In [115...
```
labeled_barplot(df, "Occupation", perc=True)
```



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

Observations on City Tier

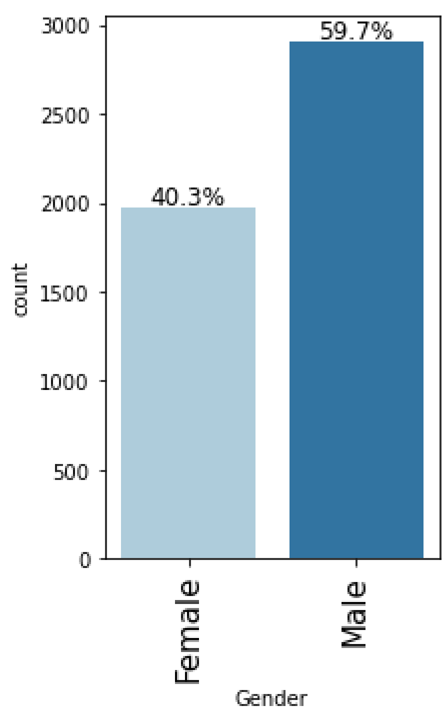
In [116...
```
labeled_barplot(df, "CityTier", perc=True)
```



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

Observations on Gender

In [117…
```
labeled_barplot(df, "Gender", perc=True)
```



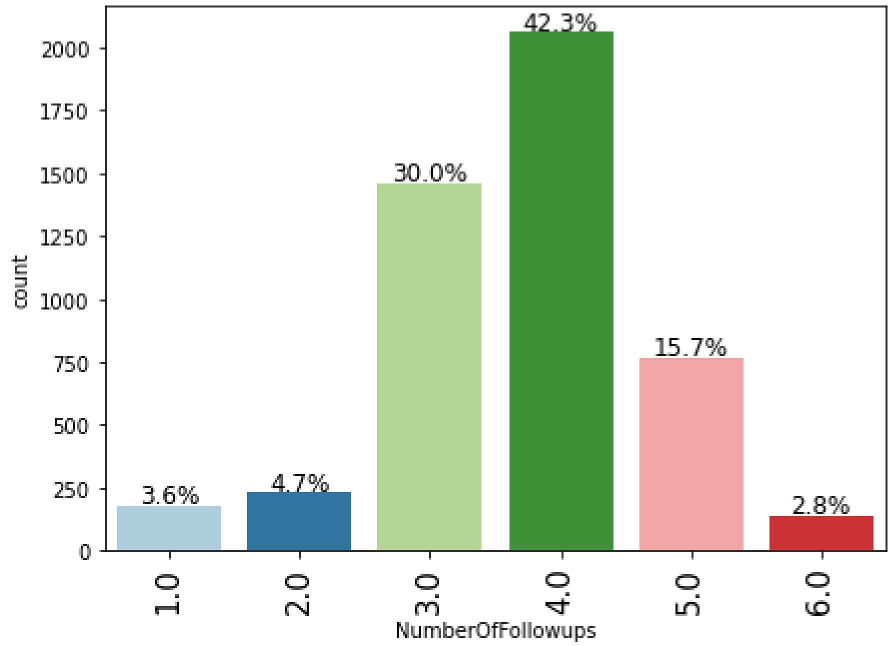
- Male customers are more than the number of female customers

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

Observations on Number of Follow ups

In [118…

```
labeled_barplot(df, "NumberOfFollowups", perc=True)
```

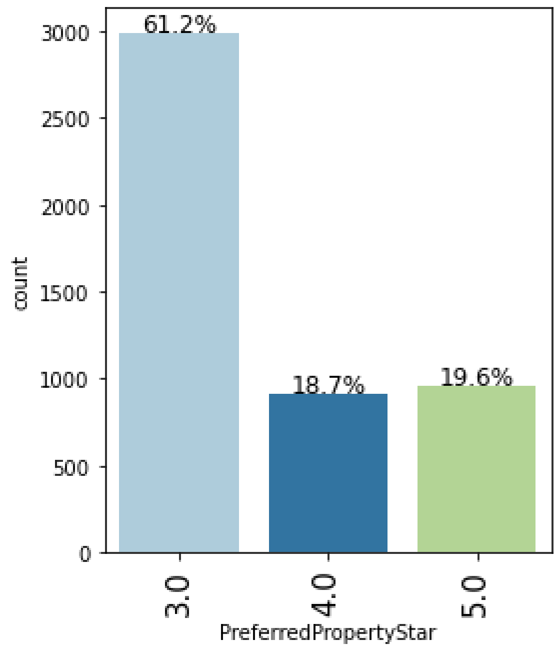


- We can see that company usually follow-ups with 3 or 4 times with their customers
- We can explore this further and observe which number of follow-ups have more customers who buy the product.

Observations on Preferred Property Star

In [119…

```
labeled_barplot(df, "PreferredPropertyStar", perc=True)
```

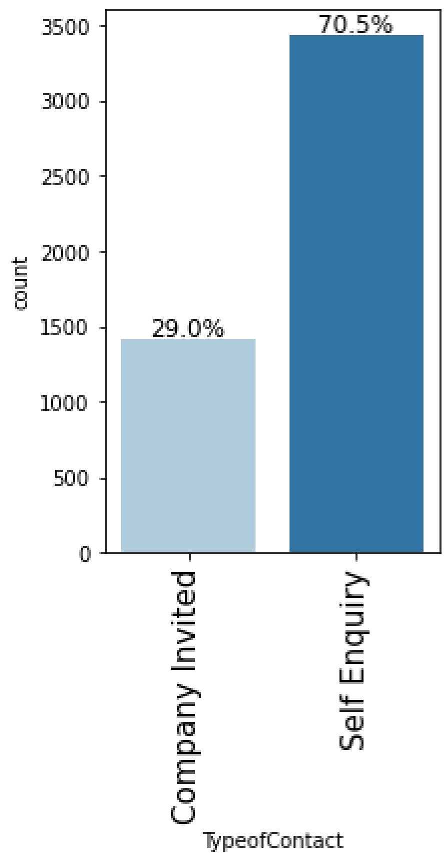


- Approx 61% of customers prefer the three-star property.
- Approx 39% of customers prefer 4 or 5 star properties. These can be the high-income customers with high income.

Observations on Type of Contact

In [120…

```
labeled_barplot(df, "TypeofContact", perc=True)
```

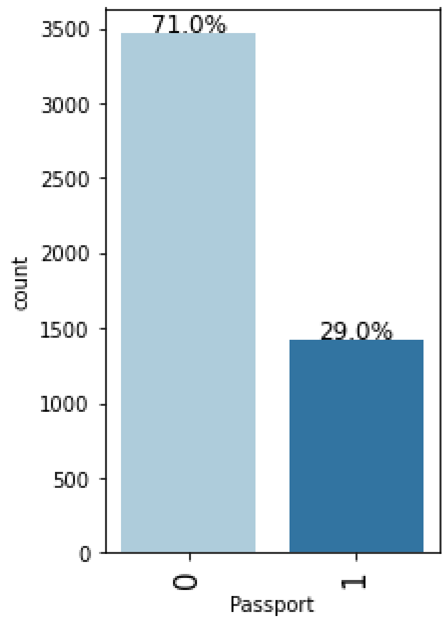


- There are approx 70% of customers who reached out to the company first i.e. self-inquiry.
- This shows the positive outreach of the company as most of the inquires are initiated from the customer's end.

Observations on Passport

In [121…

```
labeled_barplot(df, "Passport", perc=True)
```

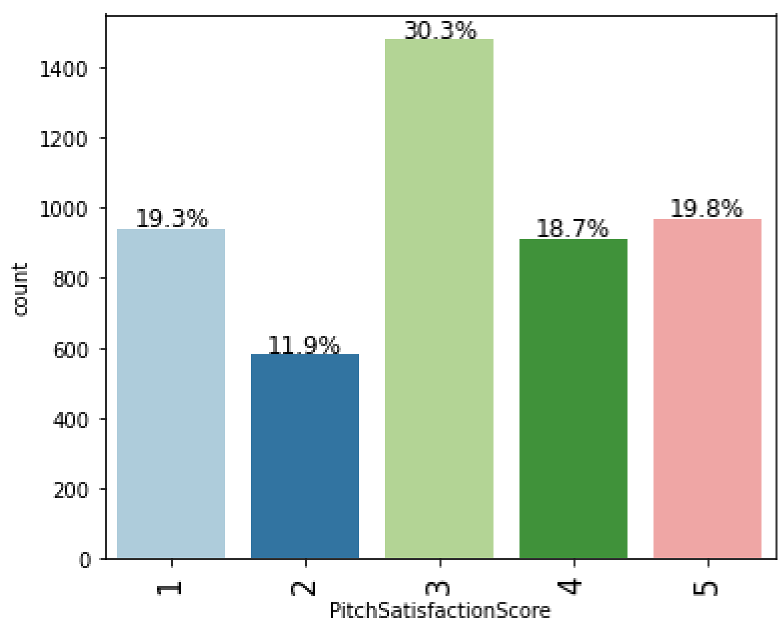


- Most of the customers i.e. approx 71% do not have a passport
- The company can provide services to help customers with getting new or renewing their passport as most of the customers do not have a passport

Observations on Pitch Satisfaction Score

In [122…

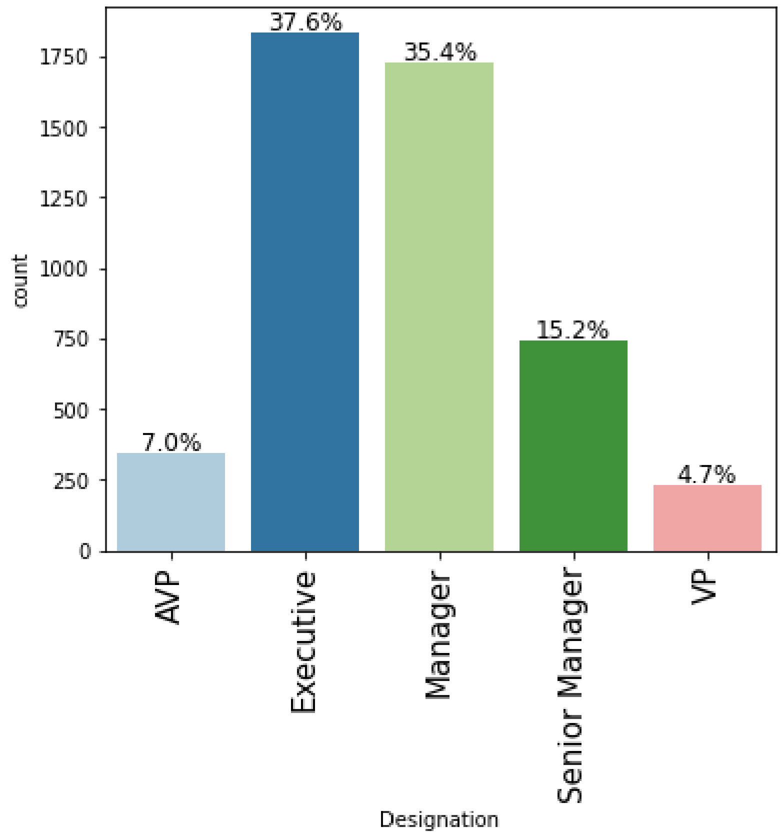
```
labeled_barplot(df, "PitchSatisfactionScore", perc=True)
```



- Average i.e. 3 is the most common pitch satisfaction score given by customers.
- We can explore this further and observe which satisfaction score has more customers who actually buy the product.

Observations on Designation

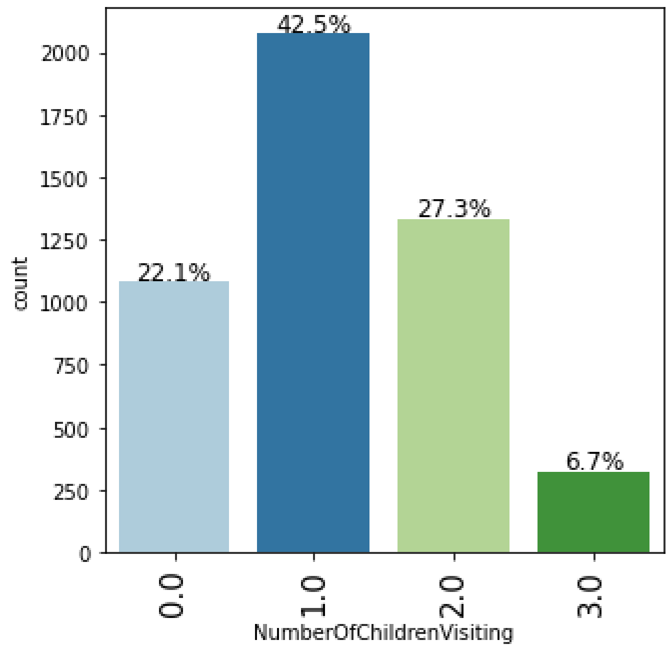
In [123…
```
labeled_barplot(df, "Designation", perc=True)
```



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

Observations on Number of Children Visiting

In [124…
```
labeled_barplot(df, "NumberOfChildrenVisiting", perc=True)
```

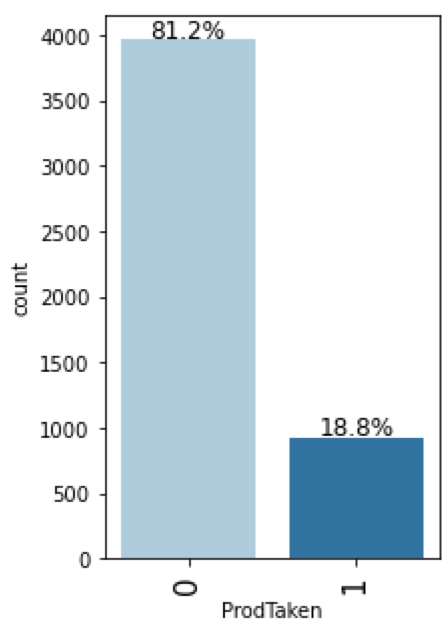


- Approx 78% of customers visit with their children and approx 34% of them have more than 1 child with them.
- 22% of customers visit without children. These may be the single/unmarried customers or recently married.

Observations on Product Taken

In [125…

```
labeled_barplot(df, "ProdTaken", perc=True)
```

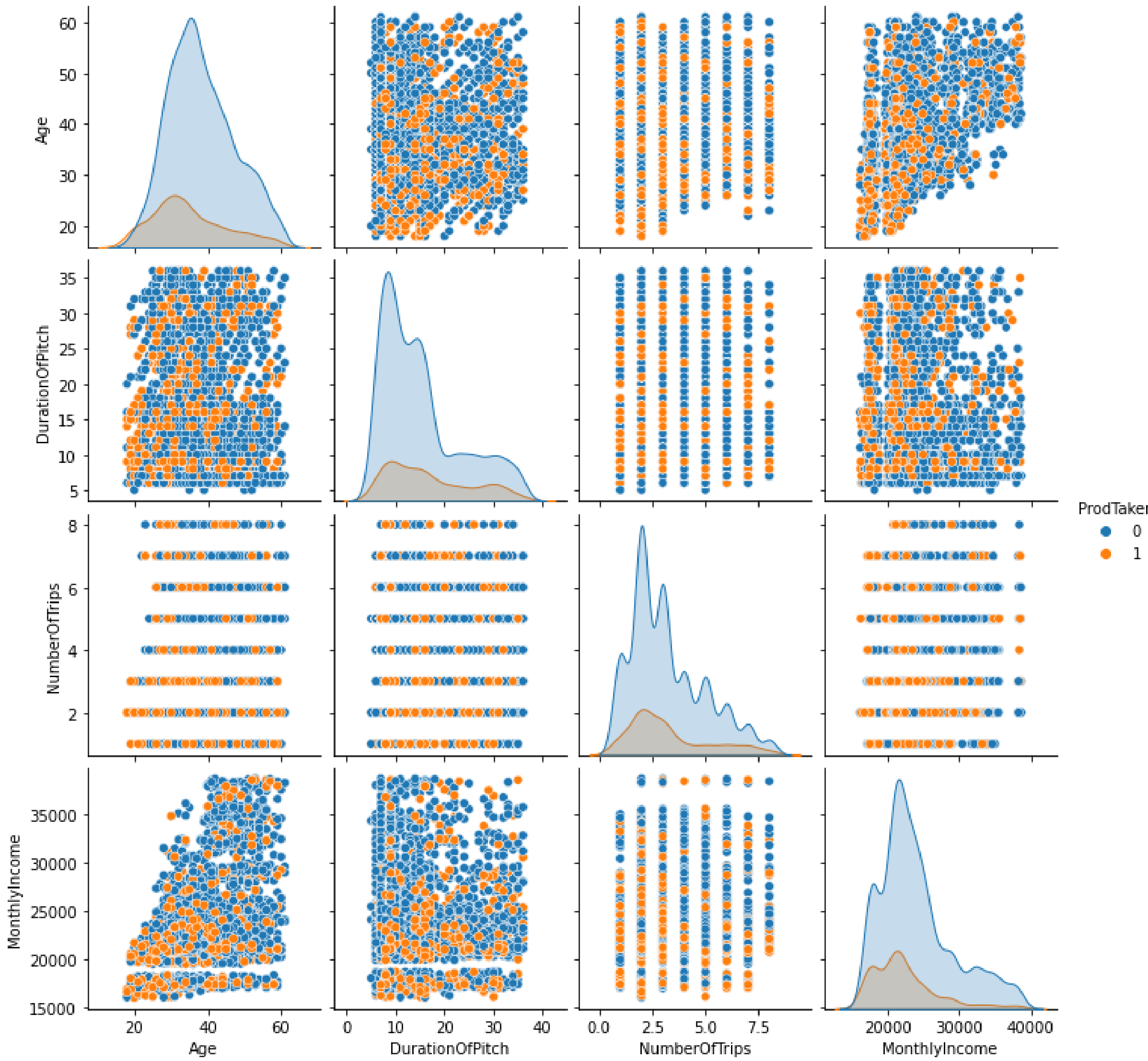


- This plot shows the distribution of both classes in the target variable is `imbalanced`.
- We only have approx 19% of customers who have purchased the product.

Bivariate Analysis

In [126…
```
sns.pairplot(data=df,hue='ProdTaken')
```

Out[126]: <seaborn.axisgrid.PairGrid at 0x1c3a4bd2040>



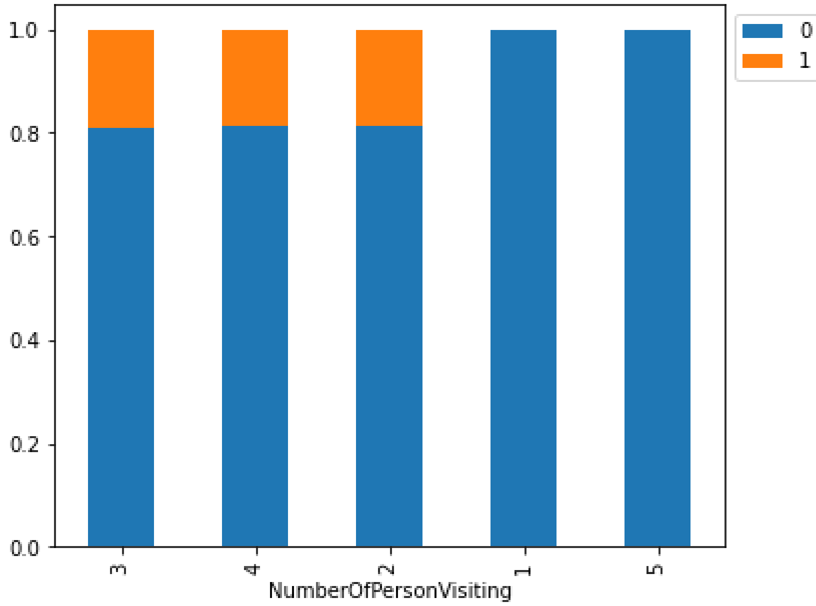
- There are overlaps i.e. no clear distinction in the distribution of variables for people who have taken the product and did not take the product.
- Let's explore this further with the help of other plots.

Prod Taken vs Number of Person Visiting

In [127…
```
stacked_barplot(df, "NumberOfPersonVisiting", "ProdTaken" )
```

```
ProdTaken                  0    1   All
NumberOfPersonVisiting
All                     3960  918  4878
3                       1938  459  2397
2                       1148  266  1414
4                        832  193  1025
```

```
1                          39    0    39
5                           3    0     3
------------------------------------------------------------------------------------------------------------------------
```



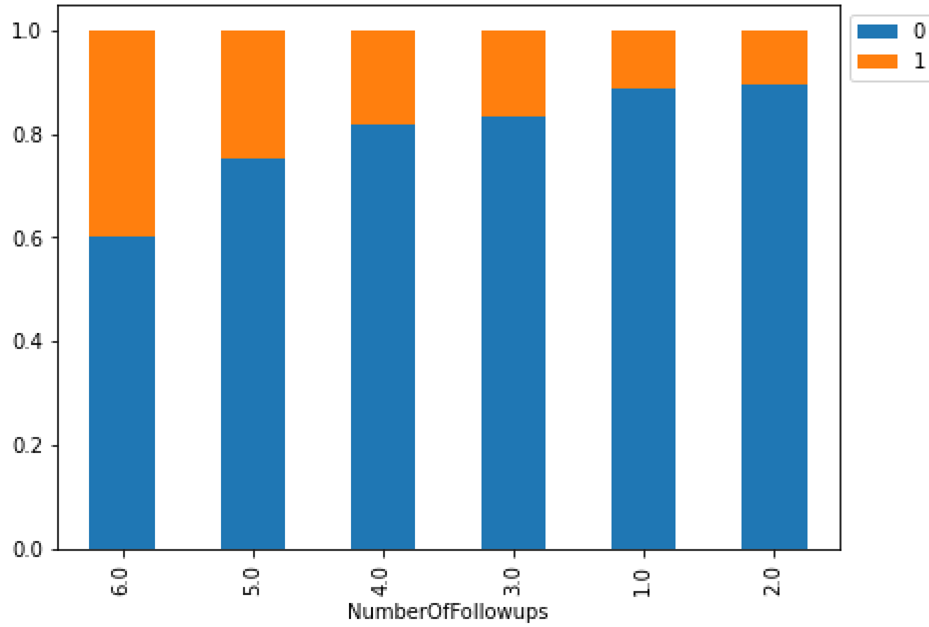
- The plot shows that the conversion rate is high when the number of persons is more than 1.
- This might be because the company is not providing good solo packages.
- The conversion rate is zero when the number of persons visiting is 5. However, there are just 3 such observations so cannot give any conclusive insights.

Prod Taken vs Number of Follow ups

In [128…

```
stacked_barplot(df, "NumberOfFollowups", "ProdTaken" )
```

```
ProdTaken              0    1   All
NumberOfFollowups
All                 3923  910  4833
4.0                 1685  378  2063
3.0                 1219  243  1462
5.0                  576  191   767
6.0                   82   54   136
2.0                  205   24   229
1.0                  156   20   176
------------------------------------------------------------------------------------------------------------------------
```



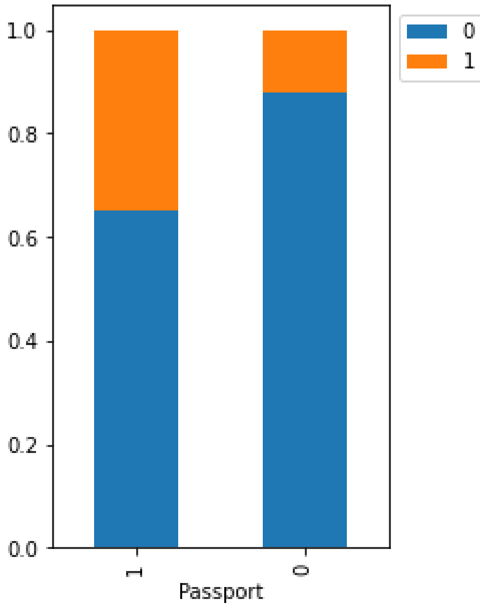
- We saw earlier that the company usually follows up 3 or 4 times but this plot shows that as number of follow ups increases, the conversion rate for customers increases.
- The Salesperson should ensure to follow up with the customers who are interested in buying the product.

Prod Taken vs Passport

In [129…

```
stacked_barplot(df, "Passport", "ProdTaken" )
```

```
ProdTaken     0    1   All
Passport
All        3960  918  4878
1           924  492  1416
0          3036  426  3462
------------------------------------------------------------------------------------
------------------------------------
```

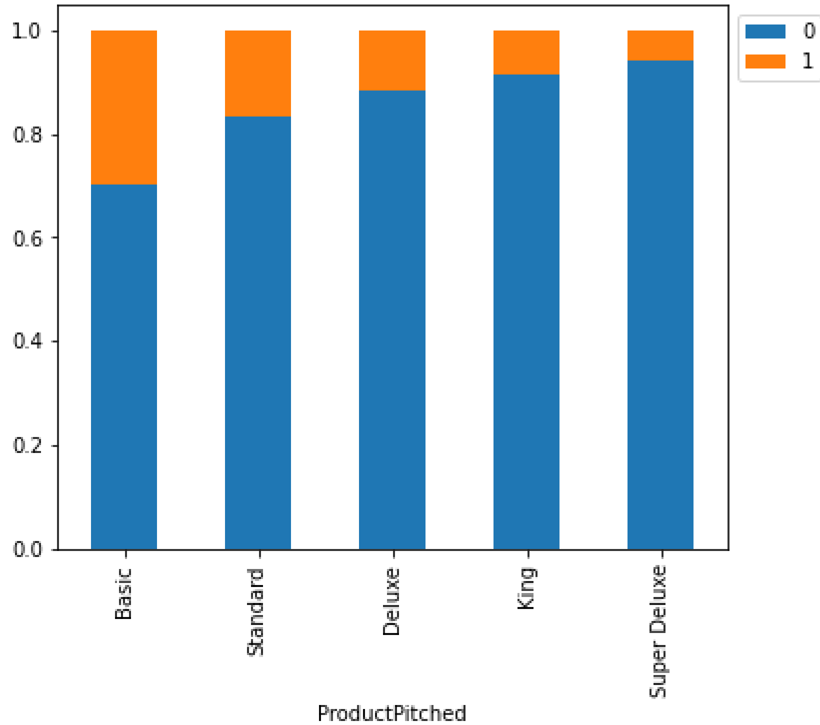


- The conversion rate for customers with a passport is higher as compared to the customers without a passport.
- The company should customize more international packages to attract more such customers.

Prod Taken vs Product Pitched

In [130…
```
stacked_barplot(df, "ProductPitched", "ProdTaken" )
```

```
ProdTaken          0    1   All
ProductPitched
All             3960  918  4878
Basic           1286  550  1836
Deluxe          1524  204  1728
Standard         618  124   742
King             210   20   230
Super Deluxe     322   20   342
------------------------------------------------------------------------------------
------------------------------------
```



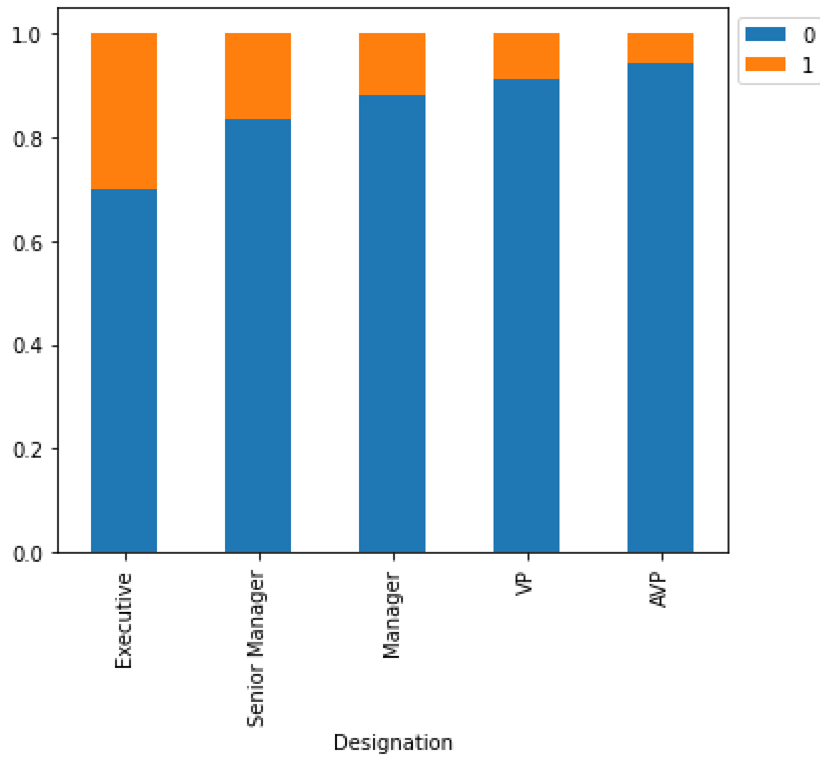
- The conversion rate of customers is higher if the product pitched is Basic. This might be because the basic package is less expensive.
- We saw earlier that company pitches the deluxe package more than the standard package, but the standard package shows a higher conversion rate than the deluxe package. The company can pitch standard packages more often.

Prod Taken vs Designation

In [131…

```
stacked_barplot(df, "Designation", "ProdTaken" )
```

```
ProdTaken          0    1   All
Designation
All             3960  918  4878
Executive       1286  550  1836
Manager         1524  204  1728
Senior Manager   618  124   742
AVP              322   20   342
VP               210   20   230
------------------------------------------------------------------------------------
------------------------------------
```

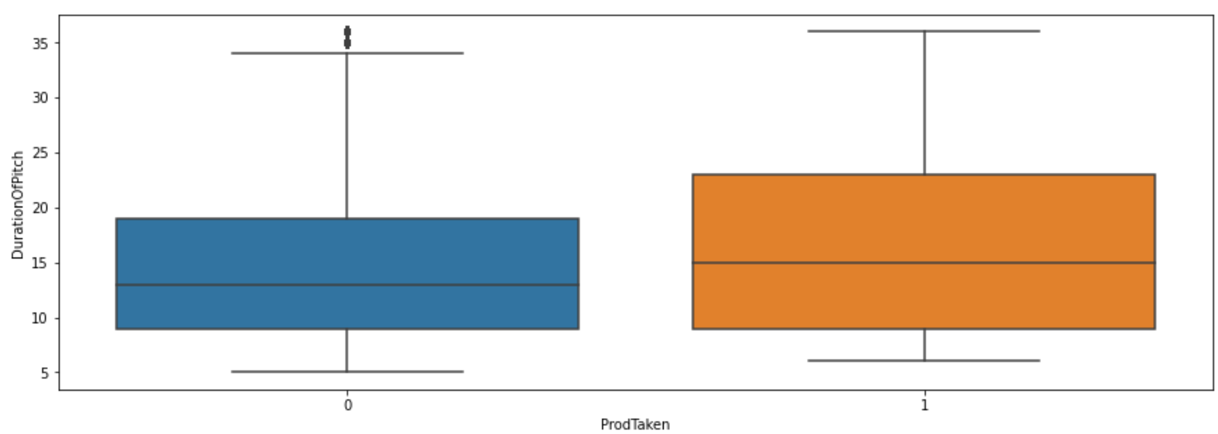


- The conversion rate of executives is higher than other designations.
- Customers at VP and AVP positions have the least conversion rate.

Prod Taken vs Duration of Pitch

In [132…

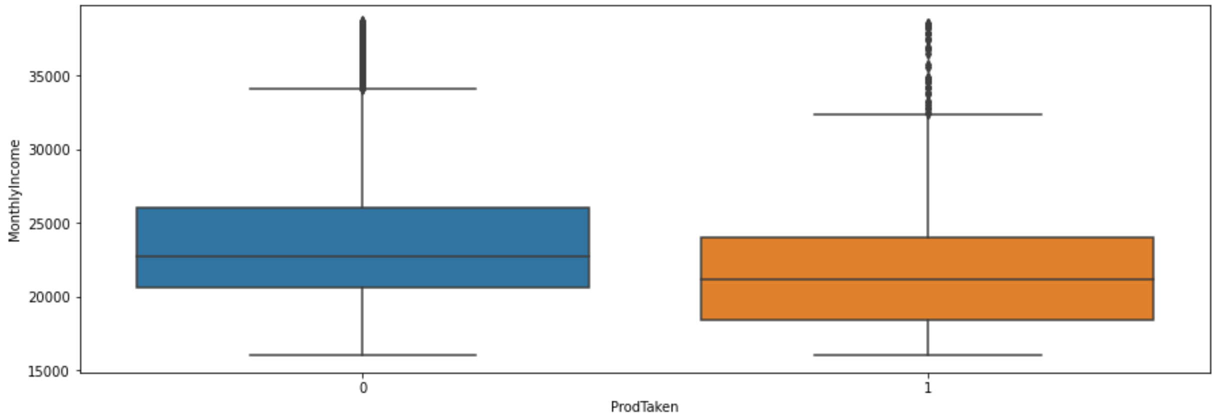
```
plt.figure(figsize=(15,5))
sns.boxplot(y='DurationOfPitch',x='ProdTaken',data=df)
plt.show()
```



- We can clearly see that customers who purchased a package have a longer duration of pitch.
- The company salesperson should give more time while pitching a certain package and convey relevant information to the customer

Prod Taken vs Monthly Income

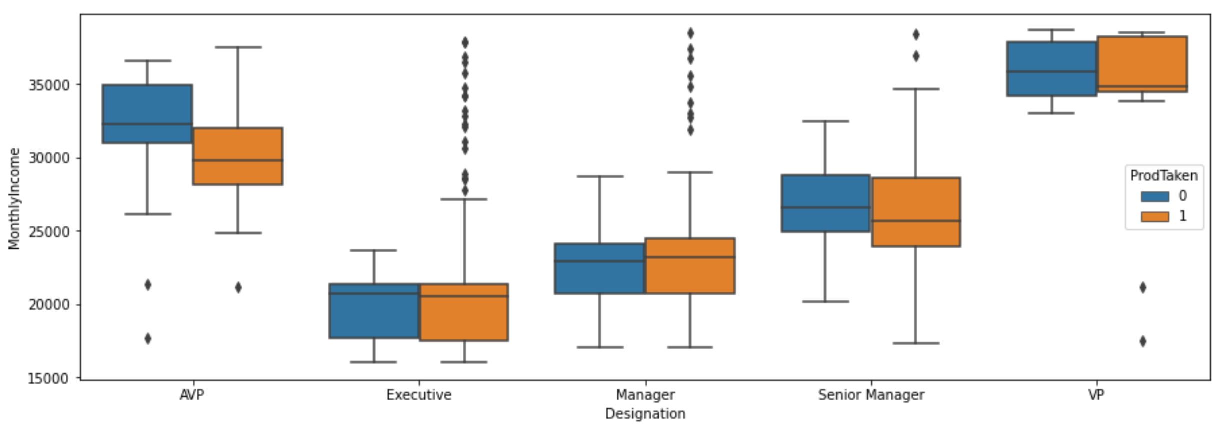
In [133…
```
plt.figure(figsize=(15,5))
sns.boxplot(y='MonthlyIncome',x='ProdTaken',data=df)
plt.show()
```



- The distribution looks right-skewed for class 0 as well as class 1 which can be expected.
- Customers who purchased a package have a lower median income than customers who did not purchase a package. This might be because of our earlier observation that executives are more likely to purchase a package.
- Let's check this by adding the variable 'Designation' to this plot.

Prod Taken vs Monthly Income vs Designation

In [134…
```
plt.figure(figsize=(15,5))
sns.boxplot(y='MonthlyIncome',x='Designation',hue='ProdTaken',data =df)
plt.show()
```



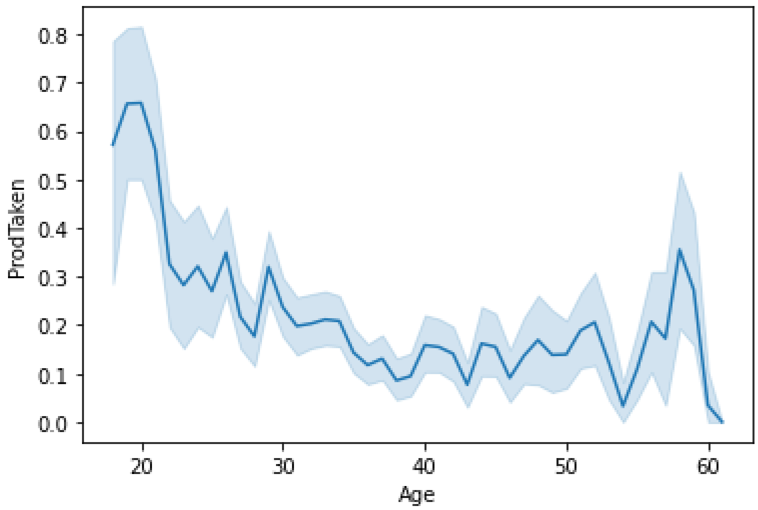
- As expected, the higher the position higher the monthly income of the customer.
- Not much difference in the income of customers at the executive or manager level who did/did not purchase a package. There are many outliers for customers who purchased a package.
- Customers at VP or AVP positions who purchase a package have a slightly lower median income.

Prod Taken vs Age

In [135...

```
sns.lineplot(x='Age',y='ProdTaken',data=df)
```

Out[135]: <AxesSubplot:xlabel='Age', ylabel='ProdTaken'>



- This plot shows that younger people are more likely to take the product as compared to middle-aged or old people.
- There is a small peak at the age near 60. These might be people who are retired or about to be retired.

To jump back to the EDA summary section, click here.