Travel Package Purchase Prediction

Problem Statement:

As a Data Scientist for a tourism company named "Visit with us". The Policy Maker of the company wants to enable and establish a viable business model to expand the customer base.

A viable business model is a central concept that helps you to understand the existing ways of doing the business and how to change the ways for the benefit of the tourism sector.

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, King. Looking at the data of the last year, we observed that 18% of the customers purchased the packages.

However, the marketing cost was quite high because customers were contacted at random without looking at the available information.

The company is now planning to launch a new product i.e. Wellness Tourism Package. Wellness Tourism is defined as Travel that allows the traveler to maintain, enhance or kick-start a healthy lifestyle, and support or increase one's sense of well-being.

However, this time company wants to harness the available data of existing and potential customers to make the marketing expenditure more efficient.

As a Data Scientist at "Visit with us" travel company have to analyze the customers' data and information to provide recommendations to the Policy Maker and Marketing Team and also build a model to predict the potential customer who is going to purchase the newly introduced travel package.

Objective:

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

Data Description:

Customer details:

- CustomerID: Unique customer ID
- ProdTaken: Whether the customer has purchased a package or not (0: No, 1: Yes)
- Age: Age of customer
- TypeofContact: How customer was contacted (Company Invited or Self Inquiry)
- CityTier: City tier depends on the development of a city, population, facilities, and living standards. The categories are ordered i.e. Tier 1 > Tier 2 > Tier 3
- Occupation: Occupation of customer
- Gender: Gender of customer
- NumberOfPersonVisiting: Total number of persons planning to take the trip with the customer
- PreferredPropertyStar: Preferred hotel property rating by customer
- MaritalStatus: Marital status of customer
- NumberOfTrips: Average number of trips in a year by customer
- Passport: The customer has a passport or not (0: No, 1: Yes)
- OwnCar: Whether the customers own a car or not (0: No, 1: Yes)
- NumberOfChildrenVisiting: Total number of children with age less than 5 planning to take the trip with the customer
- Designation: Designation of the customer in the current organization
- MonthlyIncome: Gross monthly income of the customer

Customer interaction data:

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

Import necessary libraries

In [151…
```
# Library to suppress warnings or deprecation notes
```

```python
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 sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier

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

Read the dataset¶

In [152…

```python
tourism = pd.read_csv('Tourism-Table.csv')
```

In [153…

```python
# copying data to another varaible to avoid any changes to original data
data = tourism.copy()
```

View the first and last 5 rows of the dataset.

In [154…

```python
data.head()
```

Out[154…

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

In [155…

```python
data.tail()
```

Out[155…

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPersonVisiting	NumberOfFollowups
4883	204883	1	49.0	Self Enquiry	3	9.0	Small Business	Male	3	5.(
4884	204884	1	28.0	Company Invited	1	31.0	Salaried	Male	4	5.(
4885	204885	1	52.0	Self Enquiry	3	17.0	Salaried	Female	4	4.(
4886	204886	1	19.0	Self Enquiry	3	16.0	Small Business	Male	3	4.(
4887	204887	1	36.0	Self Enquiry	1	14.0	Salaried	Male	4	4.(

Understand the shape of the dataset.

In [156…

```python
data.shape
```

Out[156…

(4888, 20)

Out[156… (4888, 20)

- There are 4888 observations and 20 columns in the dataset

Check the data types of the columns for the dataset.

In [157…

```
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 some missing values in few columns
- We have numeric and string columns

Data Pre-Processing:

Fixing Datatypes

In [158…

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

In [159…

```
#selecting all object datatypes and converting to category
category_cols = ['CityTier','ProdTaken','NumberOfPersonVisiting','NumberOfChildrenVisiting','PreferredPropertySta
data[category_cols] = data[category_cols].astype('category')

cols = data.select_dtypes(['object'])
for i in cols.columns:
    data[i] = data[i].astype('category')

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype   
---  ------                   --------------  -----   
 0   ProdTaken                4888 non-null   category
 1   Age                      4662 non-null   float64 
 2   TypeofContact            4863 non-null   category
 3   CityTier                 4888 non-null   category
 4   DurationOfPitch          4637 non-null   float64 
 5   Occupation               4888 non-null   category
 6   Gender                   4888 non-null   category
 7   NumberOfPersonVisiting   4888 non-null   category
 8   NumberOfFollowups        4843 non-null   float64 
 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(5)
memory usage: 260.2 KB
```

- The datatypes have been fixed and the memory reduced.

Missing Value Treatment:

In [160…

```
data.isna().sum()
```

Out[160…

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

In [161…

```
missing_numerical = data.select_dtypes(include=np.number).columns.tolist()
missing_numerical.remove('Age')
missing_numerical.remove('MonthlyIncome')
missing_numerical
```

Out[161…

```
['DurationOfPitch', 'NumberOfFollowups', 'NumberOfTrips']
```

In [162…

```
#replacing with the Median value of the attributes

medianFiller = lambda x: x.fillna(x.median())
data[missing_numerical] = data[missing_numerical].apply(medianFiller,axis=0)
```

In [163…

```
#replacing the missing values with median

data["MonthlyIncome"] = data.groupby(['Designation'])['MonthlyIncome'].transform(lambda x: x.fillna(x.median()))
data["Age"] = data.groupby(['Designation'])['Age'].transform(lambda x: x.fillna(x.median()))
```

Summary of the dataset

In [164…

```
data.describe().T
```

Out[164…

	count	mean	std	min	25%	50%	75%	max
Age	4888.0	37.429828	9.149822	18.0	31.0	36.0	43.00	61.0
DurationOfPitch	4888.0	15.362930	8.316166	5.0	9.0	13.0	19.00	127.0
NumberOfFollowups	4888.0	3.711129	0.998271	1.0	3.0	4.0	4.00	6.0
NumberOfTrips	4888.0	3.229746	1.822769	1.0	2.0	3.0	4.00	22.0
MonthlyIncome	4888.0	23546.843903	5266.279293	1000.0	20485.0	22413.5	25424.75	98678.0

In [165…

```python
#summary of categorical variables
cat_cols =  data.select_dtypes(['category'])
for i in cat_cols.columns:
    print(cat_cols[i].value_counts())
    print('-'*50)
    print('\n')
```

```
0    3968
1     920
Name: ProdTaken, dtype: int64
--------------------------------------------------


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
--------------------------------------------------


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
--------------------------------------------------
```

Observations:

- In the Gender column, we have wrong category "Fe Male". We have to take care of this.
- We have to handle the missing values

In [166…

```
#Fixing Gender column issue
data.Gender = data.Gender.replace('Fe Male','Female')
```

In [167…

```
#fixing missing values in categorical variables
data['TypeofContact'] = data['TypeofContact'].fillna('Self Enquiry')
data['NumberOfChildrenVisiting'] = data['NumberOfChildrenVisiting'].fillna(1.0)
data['PreferredPropertyStar'] = data['PreferredPropertyStar'].fillna(3.0)
```

In [168…

```
#checking null values
data.isnull().sum()
```

Out[168…

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

- There are no missing values in the data

In [169…

```
#summary of categorical variables
data.describe(include="category").T
```

Out[169…

	count	unique	top	freq
ProdTaken	4888	2	0	3968
TypeofContact	4888	2	Self Enquiry	3469
CityTier	4888	3	1	3190
Occupation	4888	4	Salaried	2368
Gender	4888	2	Male	2916
NumberOfPersonVisiting	4888	5	3	2402
ProductPitched	4888	5	Basic	1842
PreferredPropertyStar	4888.0	3.0	3.0	3019.0
MaritalStatus	4888	4	Married	2340
Passport	4888	2	0	3466
PitchSatisfactionScore	4888	5	3	1478
OwnCar	4888	2	1	3032
NumberOfChildrenVisiting	4888.0	4.0	1.0	2146.0
Designation	4888	5	Executive	1842

Observations:

- ProdTaken: 3968 customers did not purchase any product
- TypeofContact: Self Inquiry is the most preffered Type of Contact
- CityTier: Most customers are from Tier 1
- Occupation: Most customers are salaried
- Gender: Male customers are higher than Female Customers
- NoOfPersonsVisting: Most number of person visiting is 3
- ProductPitched: Basic is the popular product
- PreferredPropertyStar: 3.0 is the highest property rating
- MaritalStatus: Most customers are married
- Passport: Most customers dont have a passport
- PitchSatisfactionScore: Most customers have rated 3.0
- OwnCar: Most customers own a car
- NumberofChildrenVisting: Most customers plan to take atleast 1 child under five with them for the trip.
- Designation: Most customers belong to Executive designation

Exploratory Data Analysis

Univariate Analysis

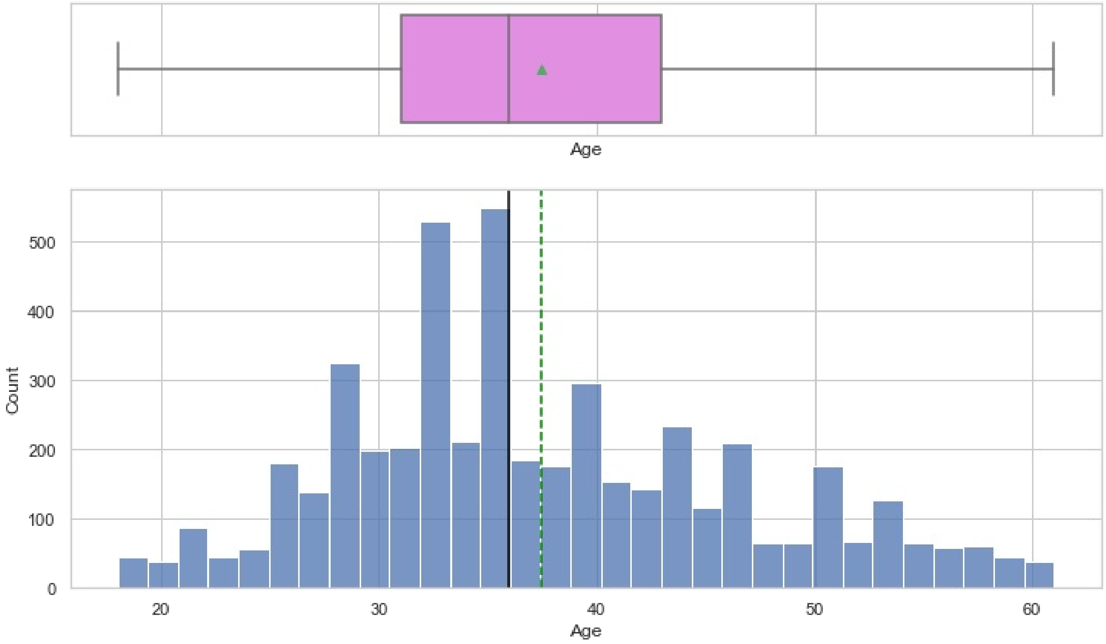
In [170…

```
# 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 column
    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 [171…

```
# visualizing Age column
histogram_boxplot(data,'Age')
```

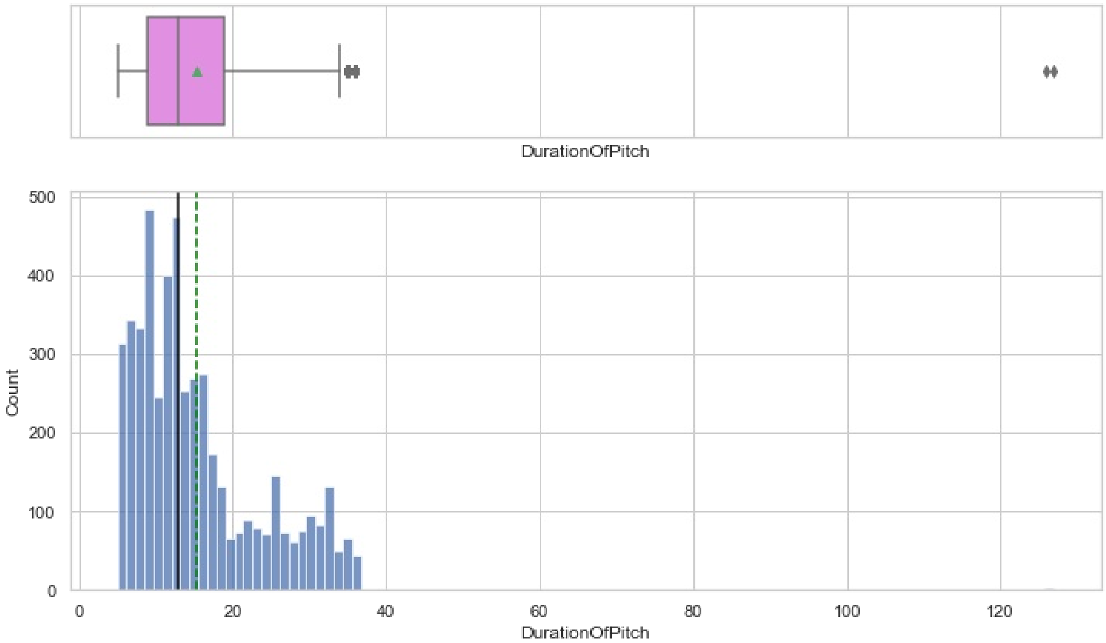


Observations:

- It's normally distributed with no outliers. we see that most customers age between 30-40 years.

In [172…

```
# visualizing DurationOfPitch column
histogram_boxplot(data,'DurationOfPitch')
```

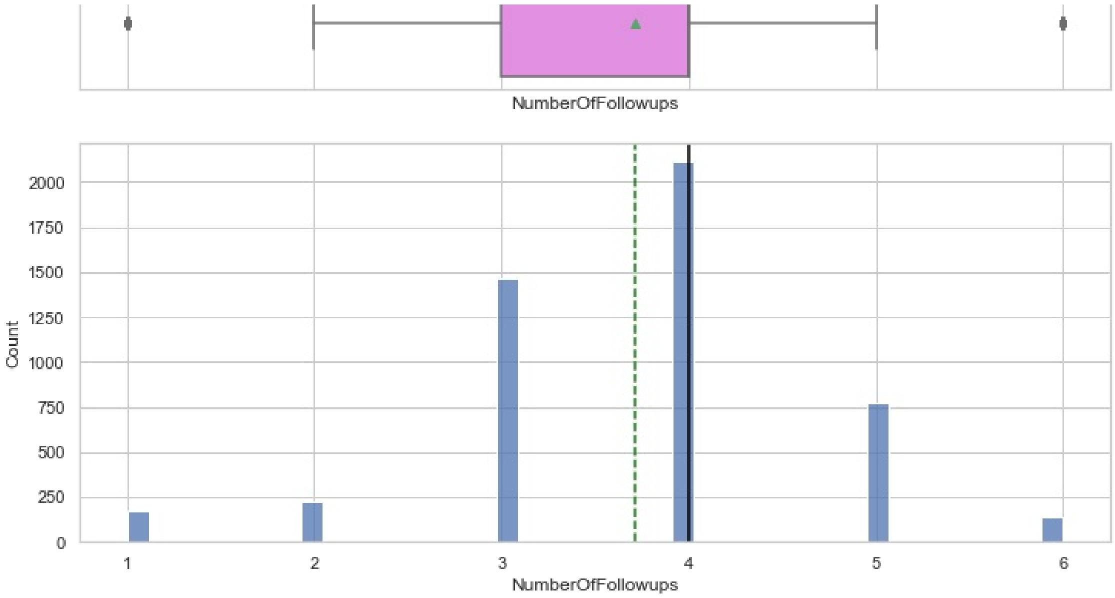


Observations:

- It's slightly right-skewed. We see that most customers pitch duration was under 20 mins.
- We also see few outliers at 40 mins and at ~120 mins.

In [173…

```
# visualizing NumberOfFollowups column
histogram_boxplot(data,'NumberOfFollowups')
```

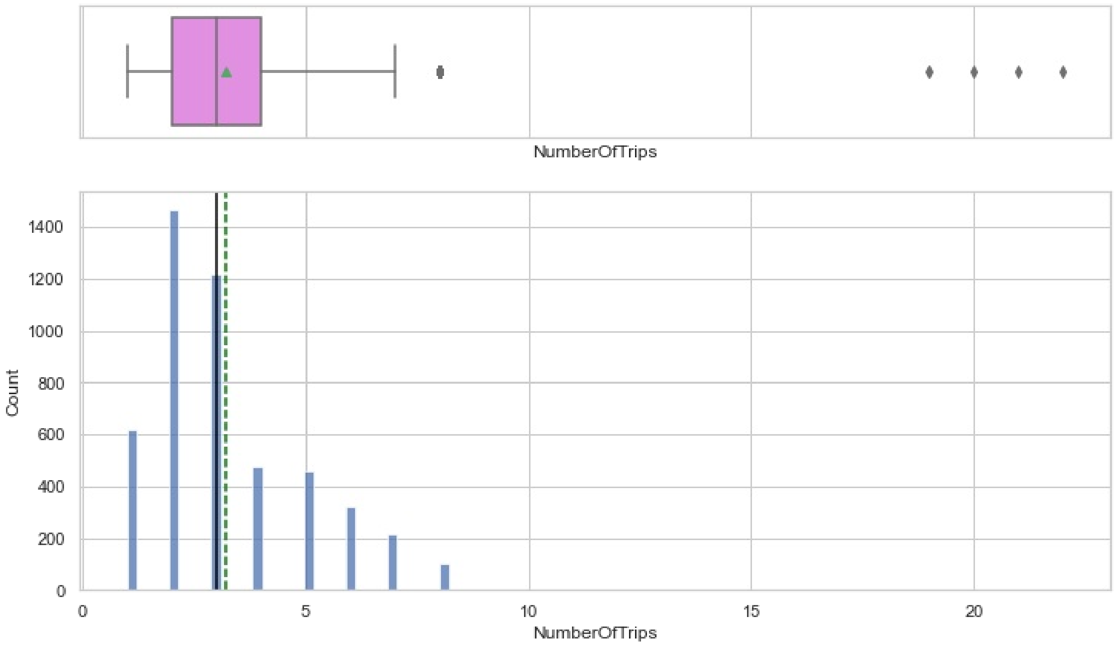


Observations:

- The highest number of followups is 4

In [174…

```
# visualizing NumberOfTrips column
histogram_boxplot(data,'NumberOfTrips')
```

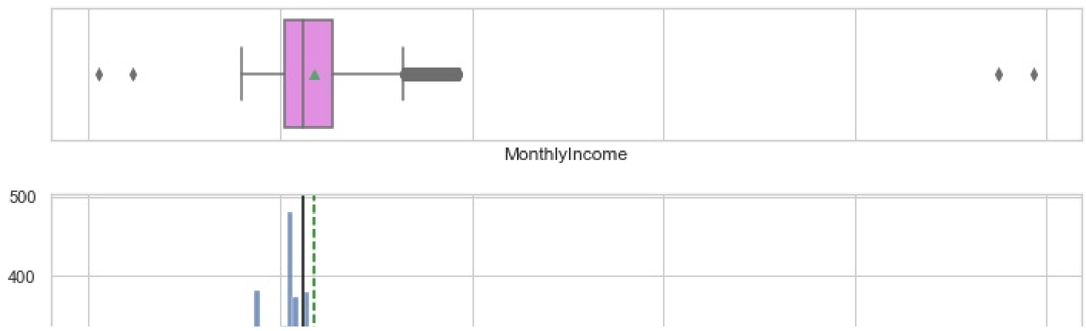


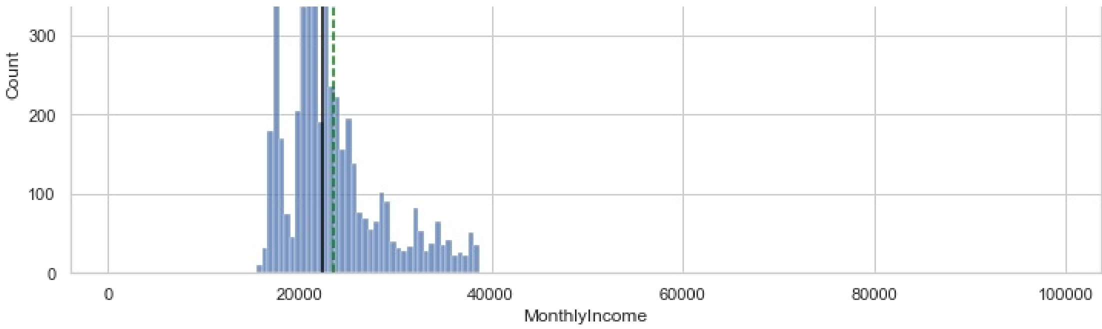
Observations:

- NumberofTrips is right-skewed and majority of the customers seem to take atleast 3 trips per year.
- We also see outliers between 10 and 20 trips.

In [175…

```
# visualizing MonthlyIncome column
histogram_boxplot(data,'MonthlyIncome')
```





Observations:

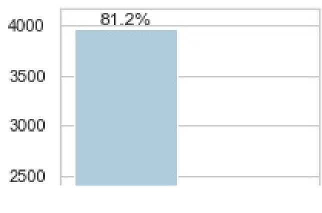
- MonthlyIncome is also right-skewd.
- We see that the majority of customers income are between 20K dollars and 30K dollars.
- We see two outliers in both ends.
- There are some outliers after the approx 30K dollars income level.

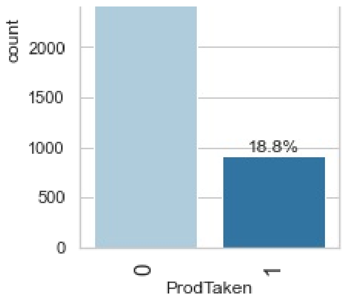
In [176…

```
# 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 levels)
    """

    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
```

In [177…

```
#visualizing ProdTaken column
labeled_barplot(data,"ProdTaken",perc=True)
```



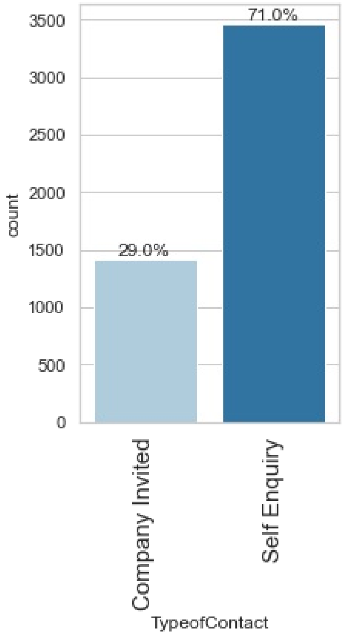


Observations:

- We see that only 18.8% of the total customers purchased any of the travel package.The plot shows heavy imbalance in the dataset

In [178…

```
#visualizing TypeofContact column
labeled_barplot(data,"TypeofContact",perc=True)
```

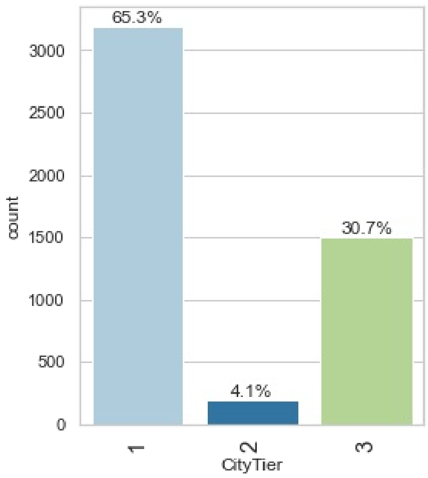


Observations:

- 71% of the customers prefered "Self Enquiry" contact method

In [179…

```
#visualizing CityTier column
labeled_barplot(data,"CityTier",perc=True)
```



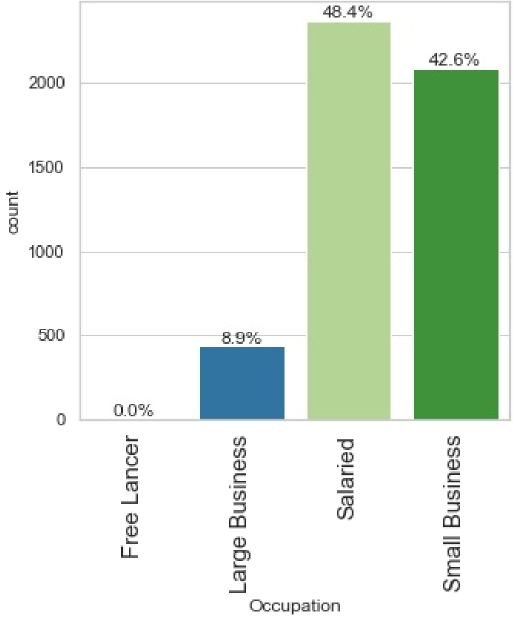
Observations:

- 65.3% of customers are from Tier 1 cities
- 30.7% of customers are from Tier 3 cities

- 4.1% of customers are from Tier 2 cities

In [180…

```
#visualizing Occupation column
labeled_barplot(data,"Occupation",perc=True)
```

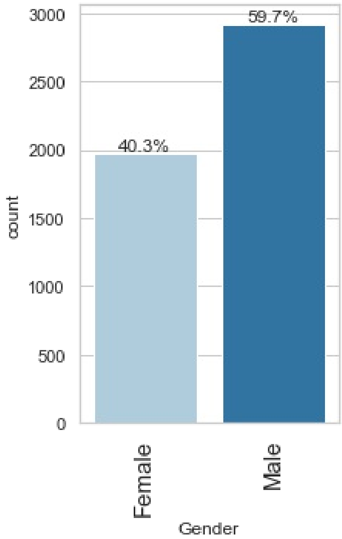


Observations:

- 48.4% of customers are Salaried
- 42.6% of customers are Small Business people
- 8.9 of customers are Large Business people
- Free Lancer customers are 0%

In [181…

```
#visualizing Gender column
labeled_barplot(data,"Gender",perc=True)
```

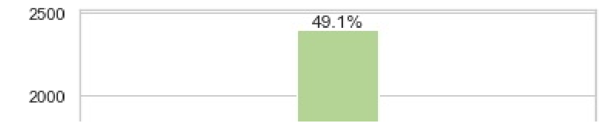


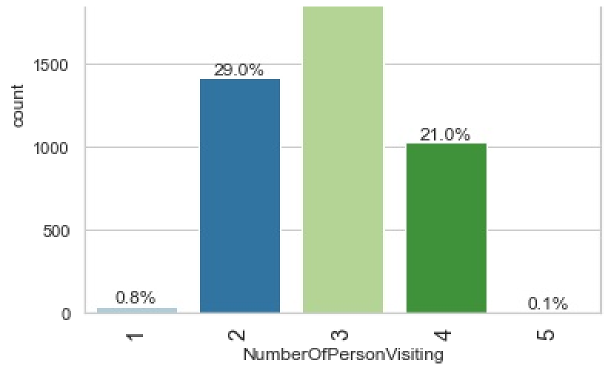
Observations:

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

In [182…

```
#visualizing NumberOfPersonVisiting column
labeled_barplot(data,"NumberOfPersonVisiting",perc=True)
```



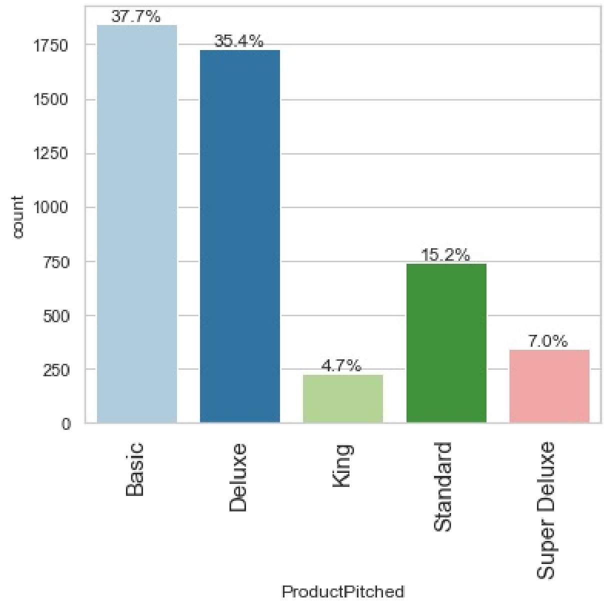


Observations:

- 49.1% of customers plan to take atleast 3 persons with them during trip
- 29% of customers plan to take atleast 2 persons with them during trip
- 21% of customers plan to take atleast 4 persons with them during trip
- Customers plan to take atleast 1 or 5 persons with them during trip are less than 1%

In [183…

```
#visualizing ProductPitched column
labeled_barplot(data,"ProductPitched",perc=True)
```

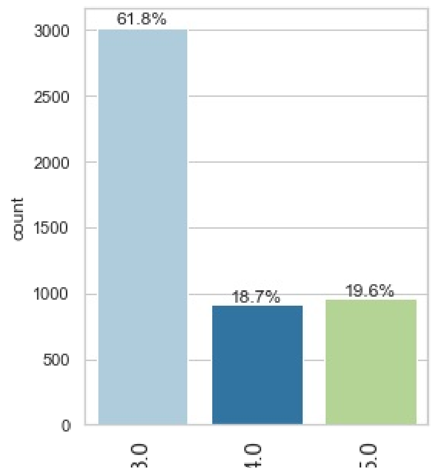


Observations:

- Basic is the most popular travel packages with 37.7%.
- The next slightly popular one is Deluxe travel package with 35.4%
- King travel package is comparatively lower than other packages with just 4.7%

In [184…

```
#visualizing PreferedPropertyStar column
labeled_barplot(data,"PreferredPropertyStar",perc=True)
```

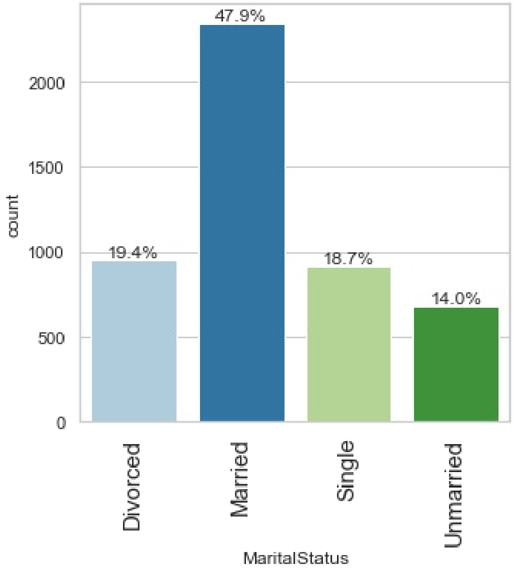


Observations:

- 61.8% of customers prefers three star hotel rating
- 18.7% of customers prefers four star hotel rating
- 19.6% of customers prefers five star hotel rating

In [185…

```
#visualizing MaritalStatus column
labeled_barplot(data,"MaritalStatus",perc=True)
```

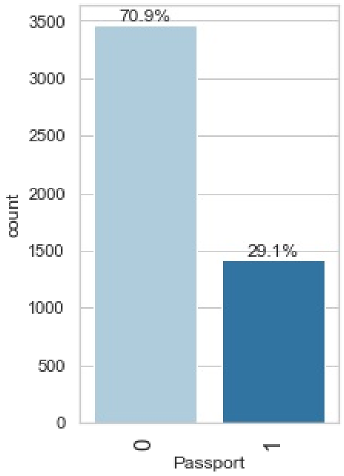


Observations:

- 47.9% of customers are Married customers
- 18.7% of customers are Single customers
- 19.4% of customers are Divorced customers
- 14% of customers are Unmarried customers

In [186…

```
#visualizing Passport column
labeled_barplot(data,"Passport",perc=True)
```

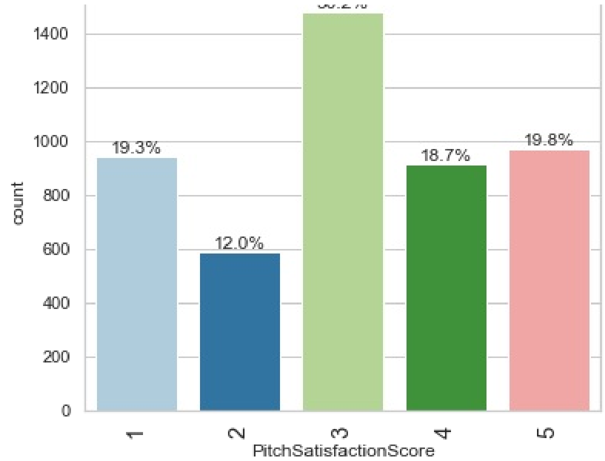


Observations:

- 70.9% of customers doesn't have a passport.
- Only 29.1% of customers have a passport

In [187…

```
#visualizing PitchSatisfactionScore column
labeled_barplot(data,"PitchSatisfactionScore",perc=True)
```

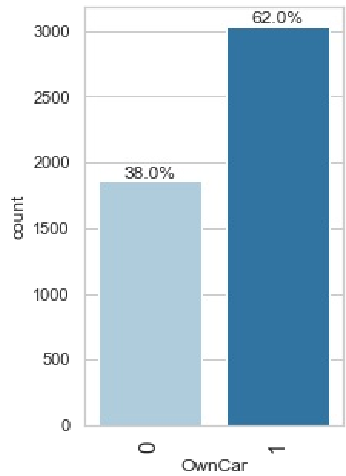


Observations:

- 30.2% of customers rated the Sales Pitch with a score of 3
- 18.7% of customers rated at 4
- 19.8% of customers rated a pitch score of 5
- 19.3% of customers rated the Sales pitch score at 1
- 12% of the customers rated the Sales pitch score at 2

In [188…

```
#visualizing OwnCar column
labeled_barplot(data,"OwnCar",perc=True)
```

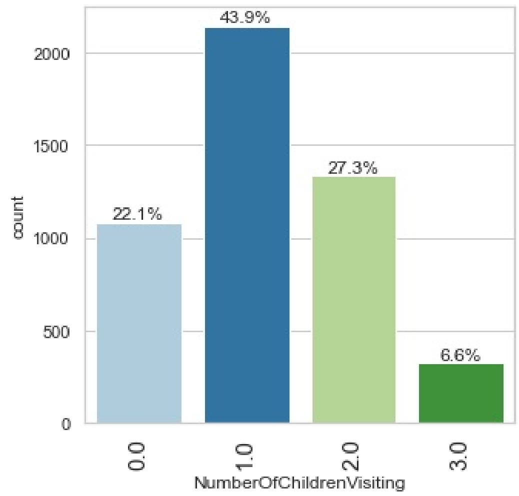


Observations:

- 62% of customers own a car
- 38% of customers own a car

In [189…

```
#visualizing NumberOfChildrenVisiting column
labeled_barplot(data,"NumberOfChildrenVisiting",perc=True)
```

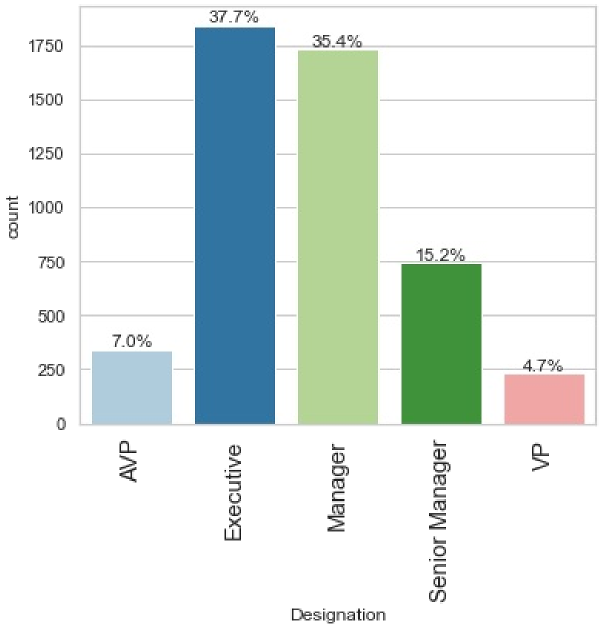


Observations:

- 43.9% of customers have 1 child with age less than 5 planning to take the trip with the customer
- 27.3% of customers have 2 childrens with age less than 5 planning to take the trip with the customer
- 22.1% of customers have no child with age less than 5 planning to take the trip with the customer
- 6.6% of customers have 3 childrens with age less than 5 planning to take the trip with the customer

In [190…

```
#visualizing Designation column
labeled_barplot(data,"Designation",perc=True)
```



Observations:

- Executive designation customers are higher in the dataset with 37.7%
- 35.4% of cutomers are Managers
- 15.2% of cutomers are Senior Managers
- 7% of cutomers are AVP
- 4.7% of cutomers are VP

Bivariate Analysis

In [191…

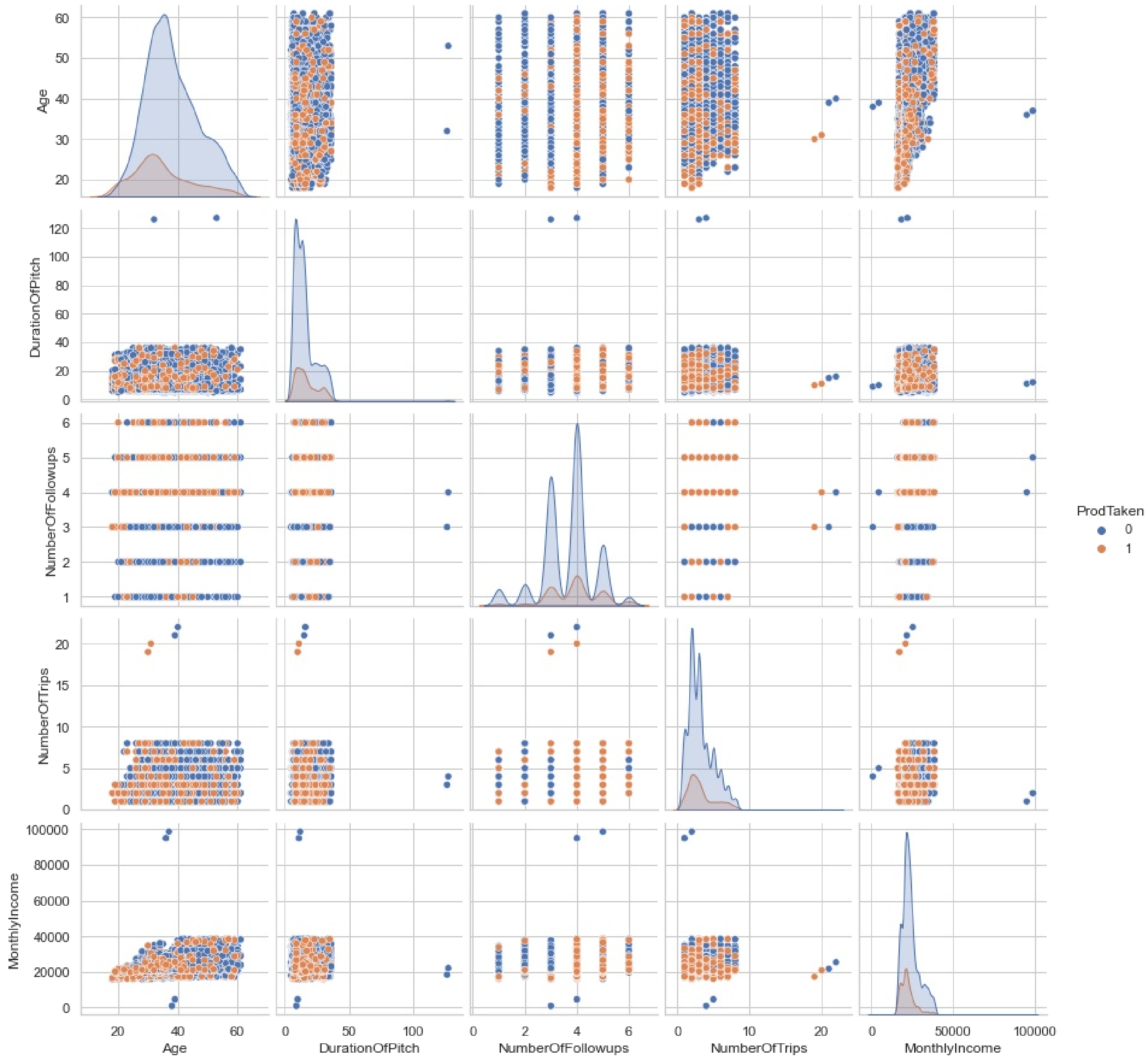
```
plt.figure(figsize=(10,7))
sns.heatmap(data.corr(),annot=True,vmin=-1,vmax=1,fmt='.1g',cmap="Spectral")
plt.show()
```

Observations:

- The correlation values are low between all the variables
- MonthlyIncome and Age have the highest positive correlation
- Age and DurationofPitch have the low negative correlation

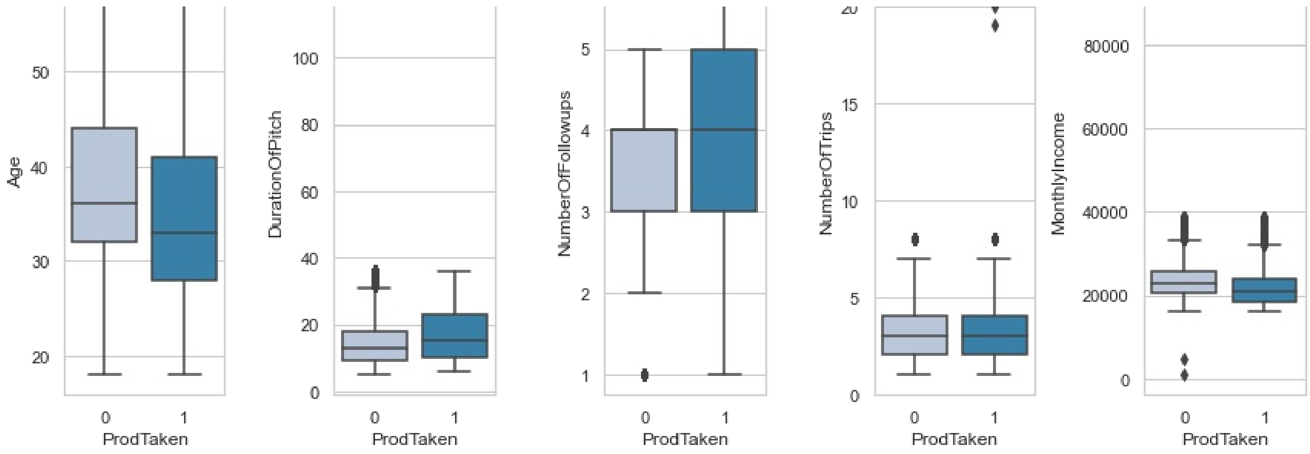
In [192…

```
sns.pairplot(data,hue='ProdTaken')
plt.show()
```



In [193…

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

for i, variable in enumerate(cols):
                     plt.subplot(1,5,i+1)
                     sns.boxplot(data['ProdTaken'],data[variable],palette="PuBu")
                     plt.tight_layout()
                     plt.title(variable)
plt.show()
```



Observations:

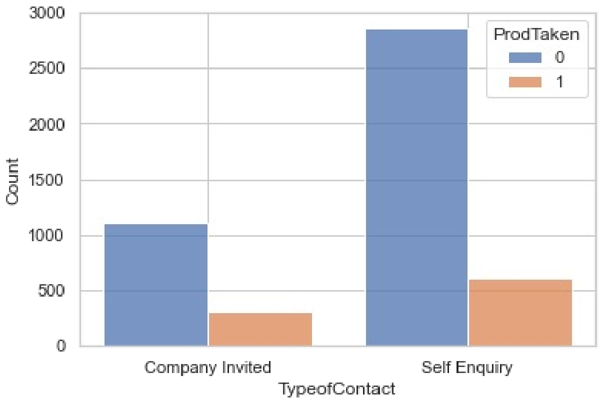
- The mean Age for customers who purchased any Product is slightly less than those who didnt. We also see that Age variable doesnt have any outliers.
- The mean DurationOfPitch for both classed of ProdTaken is almost equal. We see there are many outliers in Class '0' of ProdTaken.
- Customers who purchased the packages have four followups.
- In NumberofTrips both Classes of ProdTaken is almost equal and it has outliers on both classes.
- In MonthlyIncome both Classes of ProdTaken is almost equal and it has outliers in the higher end for both ProdTaken classes and few in low end of Class '0'.

In [194…

```
sns.histplot(data=data, x="TypeofContact", hue="ProdTaken", multiple="dodge", shrink=.8)
```

Out[194…

```
<AxesSubplot:xlabel='TypeofContact', ylabel='Count'>
```



Observation:

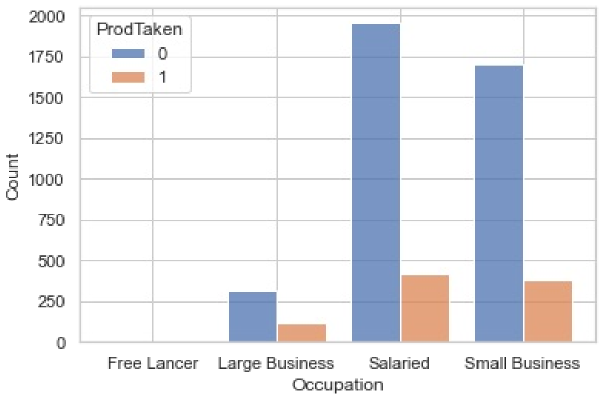
- More Customers with Company Invited contact have bought Travel Packages

In [195…

```
sns.histplot(data=data, x="Occupation", hue="ProdTaken", multiple="dodge", shrink=.8)
```

Out[195…

```
<AxesSubplot:xlabel='Occupation', ylabel='Count'>
```

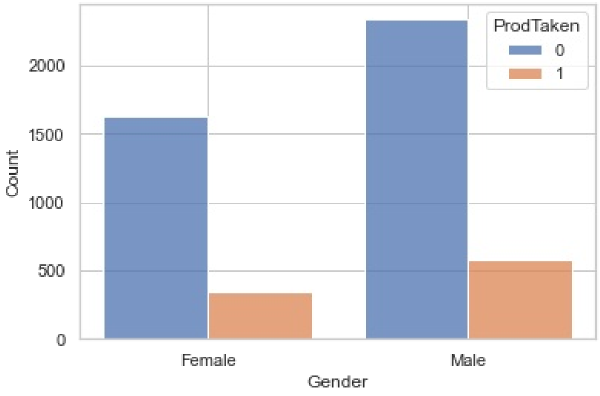


Observations:

- Large Business owning customers, bought travel packages more percentage than other occupations based on counts

In [196…
```
sns.histplot(data=data, x="Gender", hue="ProdTaken", multiple="dodge", shrink=.8)
```

Out[196…
```
<AxesSubplot:xlabel='Gender', ylabel='Count'>
```

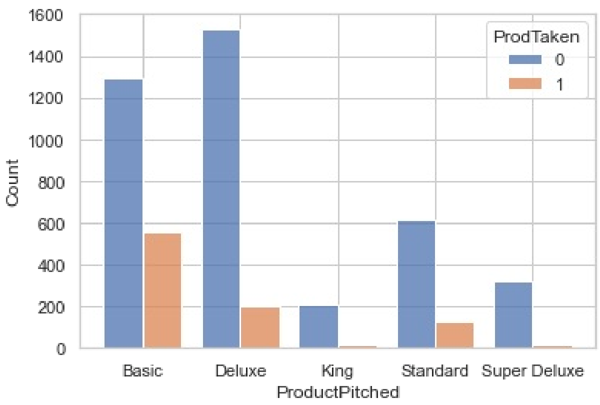


Observations:

- Eventhough male customers are more than female customers but buying percentage is almost equal.

In [197…
```
sns.histplot(data=data, x="ProductPitched", hue="ProdTaken", multiple="dodge", shrink=.8)
```

Out[197…
```
<AxesSubplot:xlabel='ProductPitched', ylabel='Count'>
```

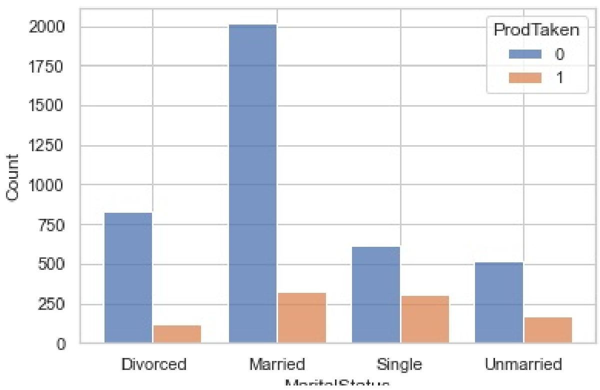


Observations:

- The Basic Package is the most preffered, with Standard and Deluxe following up.
- Comparitively very few customers purchased King and Super Deluxe products

In [198…
```
sns.histplot(data=data, x="MaritalStatus", hue="ProdTaken", multiple="dodge", shrink=.8)
```

Out[198…
```
<AxesSubplot:xlabel='MaritalStatus', ylabel='Count'>
```



MaritalStatus

Observations:

- Around 30% of all Single customers have bought a product and about 25% of Unmarried customers have also purchased a product
- Almost 50% of the total customers belong to the married category, but we see that only approx 15% of them have actually purchased any product.

In [199…
```
sns.histplot(data=data, x="Designation", hue="ProdTaken", multiple="dodge", shrink=.8)
```

Out[199…
```
<AxesSubplot:xlabel='Designation', ylabel='Count'>
```

Observations:

- ~30% Customers with Executive Designation have purchased a product
- ~15% Senior Manager Designation customers have purchased a product.
- ~11% Manager Designation customers have purchased a product.
- Only very few customers of VP and AVP Designation have purchased a product.

Outliers Detection and Treatment

In [200…
```
#finding the percentage of outliers using IQR
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)

IQR = Q3 - Q1

lower=Q1-1.5*IQR
upper=Q3+1.5*IQR
```

In [201…
```
outlier_num = data.select_dtypes(include=np.number)
```

In [202…
```
((outlier_num<lower)|(outlier_num>upper)).sum()/len(data)*100
```

Out[202…
```
Age                  0.000000
DurationOfPitch      2.291326
NumberOfFollowups    6.382979
NumberOfTrips        2.229951
MonthlyIncome        7.671849
dtype: float64
```

Observations:

- MonthlyIncome and NumberofFollowups have high outliers compared to the other features.
- However, we will not be treating outliers, as we will be building Decision Tree based models and Decision Tree models are not influenced by Outliers.
- Furthermore, in real case scenario, we will encounter similar outliers and that would require the model to investigate if there is any pattern among the customers

Model Building - Approach

- Data preparation
- Split the data into the train and test set.
- Train models on the training data.
- Try to improve the model performance using hyperparameter tuning.
- Test the performance on the test data.

Model Evaluation Criterion

The model can make wrong predictions as:

- Predicting that the customer will purchase a Travel Package when they dont. - False Positive
- Predicting that the customer will not purchase a Travel Package when they do. - False Negative

Which case is more important?

- Target potential customers who have higher chances of buying a product.
- Predict and Identify all potential customers who will purchase the newly introduced travel package.

Which metric to optimize?

- We would want F1-Score to be maximized, the greater the F1-Score higher the chances of predicting both the classes correctly.

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

In [203…

```
# defining a function to compute different metrics to check performance of a classification model built using skl
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
    """

    # 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 [204…

```
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")
```

Split Data

In [205…
```
X= data.drop(['ProdTaken','PitchSatisfactionScore','ProductPitched','NumberOfFollowups','DurationOfPitch'],axis=1
y= data['ProdTaken']
```

In [206…
```
X = pd.get_dummies(X, drop_first=True)
# Splitting data into training and test set:
X_train,X_test, y_train, y_test =train_test_split(X,y, test_size=0.3, random_state=25,stratify=y)
print(X_train.shape,X_test.shape)
```

```
(3421, 28) (1467, 28)
```

The Stratify arguments maintain the original distribution of classes in the target variable while splitting the data into train and test sets.

In [207…
```
y.value_counts(1)
```

Out[207…
```
0    0.811784
1    0.188216
Name: ProdTaken, dtype: float64
```

In [208…
```
y_test.value_counts(1)
```

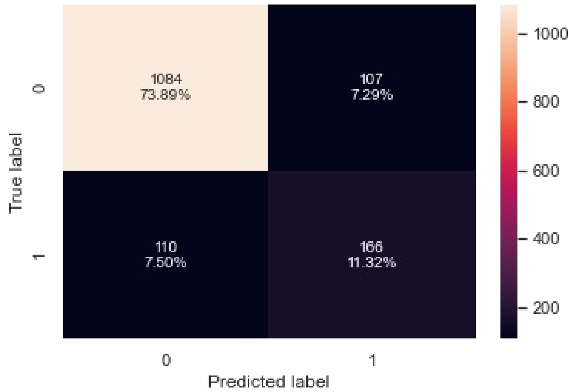
Out[208…
```
0    0.811861
1    0.188139
Name: ProdTaken, dtype: float64
```

Decision Tree Classifier

- Due to class imbalance in the dependent variable, we will add class_weight hyperparameter to give more importance to class 1
- We will keep the same randomstate = 25 for all the models so that the same random values are chosen

In [218…
```
#Fitting the model
d_tree = DecisionTreeClassifier(criterion='gini',class_weight={0:0.15,1:0.85},random_state=1)
d_tree.fit(X_train,y_train)

#Calculating different metrics
d_tree_model_train_perf=model_performance_classification_sklearn(d_tree,X_train,y_train)
print("Training performance:\n",d_tree_model_train_perf)
d_tree_model_test_perf=model_performance_classification_sklearn(d_tree,X_test,y_test)
print("Testing performance:\n",d_tree_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(d_tree,X_test,y_test)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.852079  0.601449   0.608059  0.604736
```



Observations:

- The Decision Tree model seems to be overfitting in the train set.
- The F1Score for test set is 0.60

Hyperparameter Tuning

In [223…

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

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

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.f1_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)
```

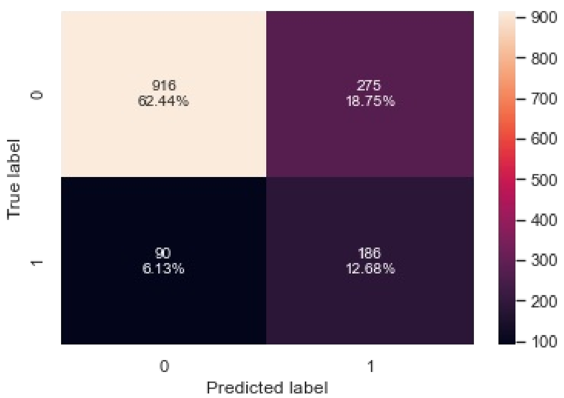
Out[223…

```
DecisionTreeClassifier(class_weight={0: 0.18, 1: 0.72}, max_depth=5,
                       max_leaf_nodes=15, min_impurity_decrease=0.0001,
                       random_state=1)
```

In [224…

```
#Calculating different metrics
dtree_estimator_model_train_perf=model_performance_classification_sklearn(d_tree,X_train,y_train)
print("Training performance:\n",dtree_estimator_model_train_perf)
dtree_estimator_model_test_perf=model_performance_classification_sklearn(d_tree,X_test,y_test)
print("Testing performance:\n",dtree_estimator_model_test_perf)

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

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.852079  0.601449   0.608059  0.604736
```



Observations:

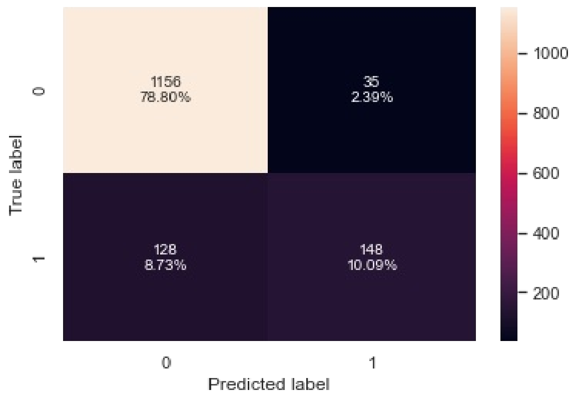
- There are no big diffferenes in the scores
- Let's try Bagging classifier

Bagging Classifier

In [219…

```
#Fitting the model
bagging_classifier = BaggingClassifier(random_state=1)
bagging_classifier.fit(X_train,y_train)

#Calculating different metrics
bagging_classifier_model_train_perf=model_performance_classification_sklearn(bagging_classifier,X_train,y_train)
print(bagging_classifier_model_train_perf)
bagging_classifier_model_test_perf=model_performance_classification_sklearn(bagging_classifier,X_test,y_test)
print(bagging_classifier_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(bagging_classifier,X_test,y_test)
```

```
   Accuracy    Recall  Precision        F1
0  0.989769  0.947205   0.998363  0.972112
   Accuracy    Recall  Precision       F1
0  0.888889  0.536232   0.808743  0.64488
```



Observations:

- The Bagging classifier has a better accuracy metric and the F1 score is also higher.
- Bagging classifier is overfitting the training data.
- Let's try hyperparameter tuning and see if the model performance improves.

Hyperparameter Tuning

In [145…

```
# 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
scorer = metrics.make_scorer(metrics.f1_score)

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

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

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

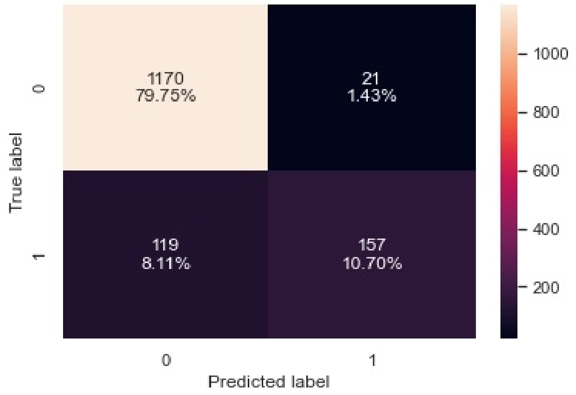
Out[145…

```
BaggingClassifier(max_features=0.9, max_samples=0.9, n_estimators=50,
                  random_state=1)
```

In [146…

```
#Calculating different metrics
bagging_estimator_tuned_model_train_perf=model_performance_classification_sklearn(bagging_estimator_tuned,X_trair
print(bagging_estimator_tuned_model_train_perf)
bagging_estimator_tuned_model_test_perf=model_performance_classification_sklearn(bagging_estimator_tuned,X_test,y
print(bagging_estimator_tuned_model_test_perf)

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

```
   Accuracy    Recall  Precision        F1
0  0.999415  0.996894        1.0  0.998445
   Accuracy    Recall  Precision       F1
0  0.904567  0.568841   0.882022  0.69163
```



Observations:

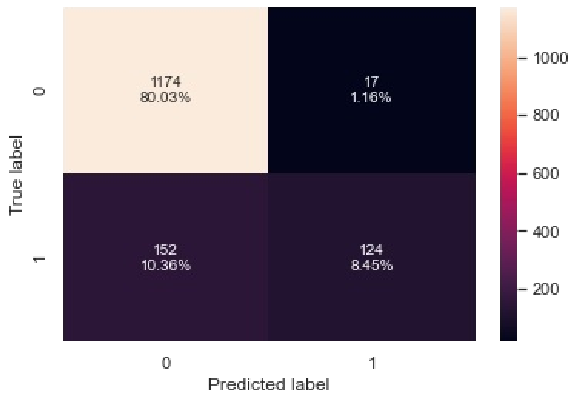
- Hyper tuning has a better accuracy metric and the F1 score is also higher.
- Bagging classifier is overfitting the training data.
- Let's try Random Forest Classifiers and see if the model performance improves.

Random Forest Classifier

In [220…

```
#Fitting the model
rf_estimator = RandomForestClassifier(random_state=1)
rf_estimator.fit(X_train,y_train)

#Calculating different metrics
rf_estimator_model_train_perf=model_performance_classification_sklearn(rf_estimator,X_train,y_train)
print("Training performance:\n",rf_estimator_model_train_perf)
rf_estimator_model_test_perf=model_performance_classification_sklearn(rf_estimator,X_test,y_test)
print("Testing performance:\n",rf_estimator_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(rf_estimator,X_test,y_test)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.884799  0.449275   0.879433  0.594724
```



Observations:

- Random Forest classifier is also overfitting for the training set.
- F1 score metric also reduced.
- Let's try hyperparameter tuning and see if the model performance improves.

Hyperparameter Tuning

In [236

In [236...

```python
# Choose the type of classifier.
rf_tuned = RandomForestClassifier(class_weight={0:0.15,1:0.85},random_state=29)

parameters = {"n_estimators": np.arange(10,60,5),
              'criterion':['gini','entropy'],
            "min_samples_leaf": np.arange(5,11,1),
            "max_features":['sqrt','log2'],
            "max_samples": np.arange(0.5, 1, 0.1),
             }

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.f1_score)

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

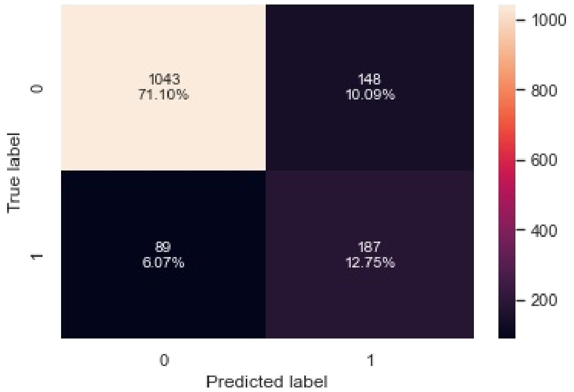
Out[236...

```
RandomForestClassifier(class_weight={0: 0.15, 1: 0.85}, max_features='sqrt',
                       max_samples=0.8999999999999999, min_samples_leaf=5,
                       n_estimators=35, random_state=29)
```

In [235...

```python
#Calculating different metrics
rf_tuned_model_train_perf=model_performance_classification_sklearn(rf_tuned,X_train,y_train)
print("Training performance:\n",rf_tuned_model_train_perf)
rf_tuned_model_test_perf=model_performance_classification_sklearn(rf_tuned,X_test,y_test)
print("Testing performance:\n",rf_tuned_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(rf_tuned,X_test,y_test)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.905583  0.939441    0.68054  0.789302
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.838446  0.677536   0.558209  0.612111
```



Observations:

- The overall model performance metric has increased after Hypertuning, but it looks like its still overfitting the training data set.
- Let's try boosting models

Boosting Models

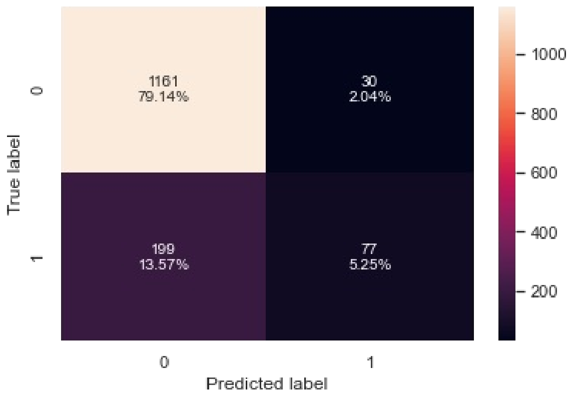
AdaBoost Classifier

In [211...

```python
#Fitting the model
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train,y_train)

#Calculating different metrics
ab_classifier_model_train_perf=model_performance_classification_sklearn(ab_classifier,X_train,y_train)
print(ab_classifier_model_train_perf)
```

```
ab_classifier_model_test_perf=model_performance_classification_sklearn(ab_classifier,X_test,y_test)
print(ab_classifier_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(ab_classifier,X_test,y_test)
```

```
   Accuracy  Recall  Precision        F1
0  0.835136    0.25   0.665289  0.363431
   Accuracy    Recall  Precision        F1
0  0.843899  0.278986   0.719626  0.402089
```



Observation:

- Adaboost is giving more generalized performance than previous models but the test f1-score is too low.
- Let's try hyperparameter tuning and see if the model performance improves.

Hyperparameter Tuning

In [212…

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

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

# Type of scoring used to compare parameter  combinations
scorer = metrics.make_scorer(metrics.f1_score)

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

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

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

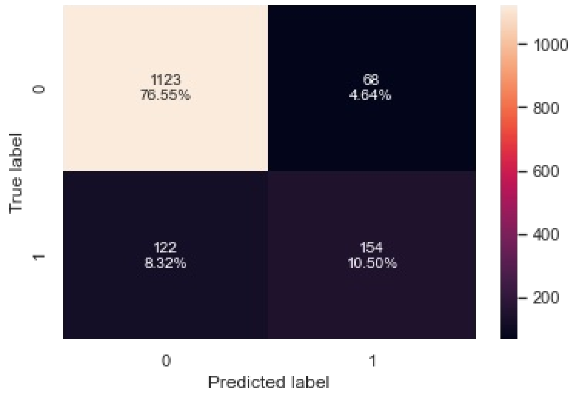
Out[212…

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                   learning_rate=1.1, n_estimators=100, random_state=1)
```

In [213…

```
#Calculating different metrics
abc_tuned_model_train_perf=model_performance_classification_sklearn(abc_tuned,X_train,y_train)
print(abc_tuned_model_train_perf)
abc_tuned_model_test_perf=model_performance_classification_sklearn(abc_tuned,X_test,y_test)
print(abc_tuned_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(abc_tuned,X_test,y_test)
```

```
   Accuracy    Recall  Precision        F1
0  0.978077  0.913043   0.968699  0.940048
   Accuracy    Recall  Precision        F1
0  0.870484  0.557971   0.693694  0.618474
```



Observations:

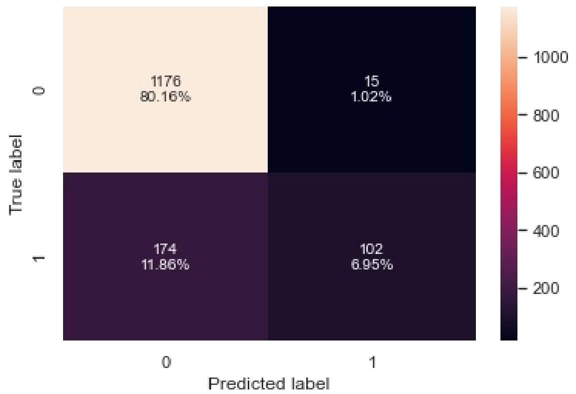
- F1-Score has increased but the model has started to overfit the training data
- Not better performance than Random forest classifier
- Let's try Gradient Boosting Classifier

Gradient Boosting Classifier

In [214…

```
#Fitting the model
gb_classifier = GradientBoostingClassifier(random_state=1)
gb_classifier.fit(X_train,y_train)

#Calculating different metrics
gb_classifier_model_train_perf=model_performance_classification_sklearn(gb_classifier,X_train,y_train)
print("Training performance:\n",gb_classifier_model_train_perf)
gb_classifier_model_test_perf=model_performance_classification_sklearn(gb_classifier,X_test,y_test)
print("Testing performance:\n",gb_classifier_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(gb_classifier,X_test,y_test)
```

```
Training performance:
    Accuracy   Recall  Precision        F1
0   0.87869  0.43323   0.848024  0.573484
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.871166  0.369565   0.871795  0.519084
```



Observations:

- The metrics are comparable and close for both train and test set and the F1Score metric has increased by compare with AdaBoost Classifier.

Hyperparameter Tuning

In [215…

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

# Grid of parameters to choose from
parameters = {
    "n_estimators": [100,150,200,250],
    "subsample":[0.8,0.9,1],
```

```
    "max_features":[0.7,0.8,0.9,1]
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.f1_score)

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

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

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

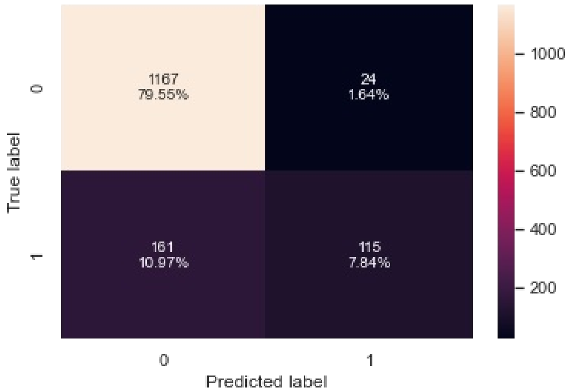
Out[215…

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

In [216…

```
#Calculating different metrics
gbc_tuned_model_train_perf=model_performance_classification_sklearn(gbc_tuned,X_train,y_train)
print("Training performance:\n",gbc_tuned_model_train_perf)
gbc_tuned_model_test_perf=model_performance_classification_sklearn(gbc_tuned,X_test,y_test)
print("Testing performance:\n",gbc_tuned_model_test_perf)

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

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.912014  0.586957   0.915254  0.715232
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.873892  0.416667   0.827338  0.554217
```



Observations:

- F1-Score has increased but the model has started to overfit the training data
- Let's try Stacking Classifier

In [227…

```
estimators = [('Random Forest',rf_tuned), ('Gradient Boosting',gbc_tuned), ('Decision Tree',dtree_estimator)]

final_estimator = abc_tuned

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

stacking_classifier.fit(X_train,y_train)
```

Out[227…

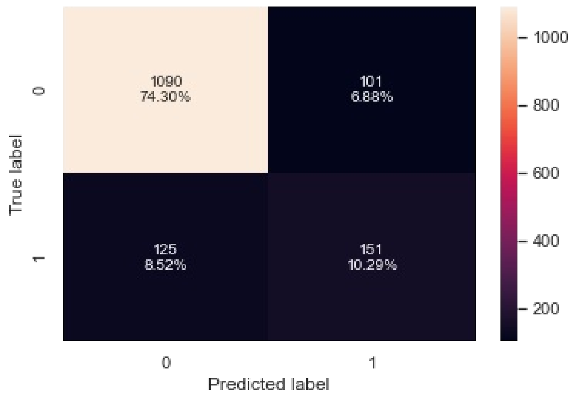
```
StackingClassifier(estimators=[('Random Forest',
                                RandomForestClassifier(class_weight={0: 0.18,
                                                                     1: 0.82},
                                                       max_depth=20,
                                                       max_features=None,
                                                       min_samples_split=7,
                                                       n_estimators=90,
                                                       oob_score=True,
                                                       random_state=1)),
                               ('Gradient Boosting',
                                GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                                                           max_features=0.8,
```

```
                                                           n_estimators=250,
                                                           random_state=1,
                                                           subsample=0.9)),
                                ('Decision Tree',
                                 DecisionTreeClassifier(class_weight={0: 0.18,
                                                                      1: 0.72},
                                                        max_depth=5,
                                                        max_leaf_nodes=15,
                                                        min_impurity_decrease=0.0001,
                                                        random_state=1))],
                   final_estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                                                      learning_rate=1.1,
                                                      n_estimators=100,
                                                      random_state=1))
```

In [228…

```
#Calculating different metrics
stacking_classifier_model_train_perf=model_performance_classification_sklearn(stacking_classifier,X_train,y_trair
print("Training performance:\n",stacking_classifier_model_train_perf)
stacking_classifier_model_test_perf=model_performance_classification_sklearn(stacking_classifier,X_test,y_test)
print("Testing performance:\n",stacking_classifier_model_test_perf)

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

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.932768  0.826087   0.818462  0.822257
Testing performance:
    Accuracy    Recall  Precision       F1
0  0.845944  0.547101   0.599206  0.57197
```



Observations:

- F1-Score has increased but the model has started to overfit the training data

Comparing all models

In [237…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [d_tree_model_train_perf.T,dtree_estimator_model_train_perf.T,rf_estimator_model_train_perf.T,rf_tuned_model_
     bagging_classifier_model_train_perf.T,bagging_estimator_tuned_model_train_perf.T,ab_classifier_model_train_p
     abc_tuned_model_train_perf.T,gb_classifier_model_train_perf.T,gbc_tuned_model_train_perf.T,stacking_classifi
    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",
    "Adabosst Classifier Tuned",
    "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
    "Stacking Classifier"]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[237…

	Decision Tree	Decision Tree Estimator	Random Forest Estimator	Random Forest Tuned	Bagging Classifier	Bagging Estimator Tuned	Adaboost Classifier	Adabosst Classifier Tuned	Gradient Boost Classifier	Gradient Boost Classifier Tuned	Stacking Classifier
Accuracy	1.0	1.0	1.0	0.905583	0.989769	0.999415	0.835136	0.978077	0.878690	0.912014	0.932768
Recall	1.0	1.0	1.0	0.939441	0.947205	0.996894	0.250000	0.913043	0.433230	0.586957	0.826087
Precision	1.0	1.0	1.0	0.680540	0.998363	1.000000	0.665289	0.968699	0.848024	0.915254	0.818462
F1	1.0	1.0	1.0	0.789302	0.972112	0.998445	0.363431	0.940048	0.573484	0.715232	0.822257

In [238…

```
# testing performance comparison

models_test_comp_df = pd.concat(
    [d_tree_model_test_perf.T,dtree_estimator_model_test_perf.T,rf_estimator_model_test_perf.T,rf_tuned_model_tes
     bagging_classifier_model_test_perf.T,bagging_estimator_tuned_model_test_perf.T,ab_classifier_model_test_perf
     abc_tuned_model_test_perf.T,gb_classifier_model_test_perf.T,gbc_tuned_model_test_perf.T,stacking_classifier_
    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",
    "Adabosst Classifier Tuned",
    "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
    "Stacking Classifier"]
print("Testing performance comparison:")
models_test_comp_df
```

```
Testing performance comparison:
```

Out[238…

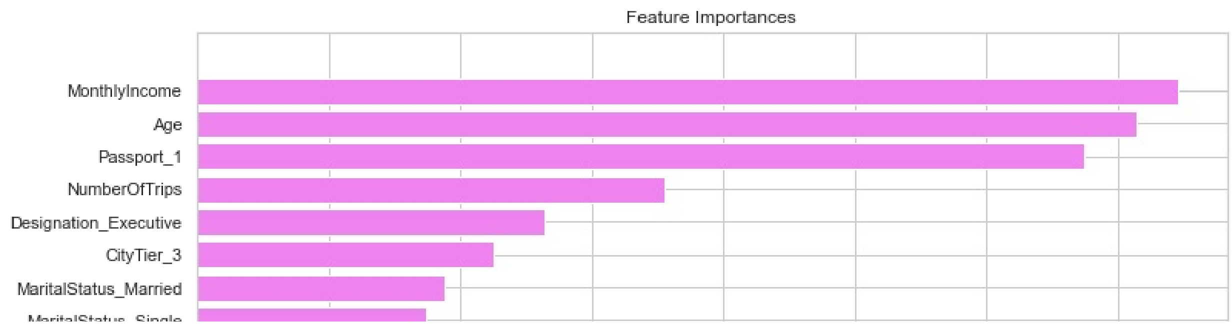
	Decision Tree	Decision Tree Estimator	Random Forest Estimator	Random Forest Tuned	Bagging Classifier	Bagging Estimator Tuned	Adaboost Classifier	Adabosst Classifier Tuned	Gradient Boost Classifier	Gradient Boost Classifier Tuned	Stacking Classifier
Accuracy	0.852079	0.852079	0.884799	0.838446	0.888889	0.904567	0.843899	0.870484	0.871166	0.873892	0.845944
Recall	0.601449	0.601449	0.449275	0.677536	0.536232	0.568841	0.278986	0.557971	0.369565	0.416667	0.547101
Precision	0.608059	0.608059	0.879433	0.558209	0.808743	0.882022	0.719626	0.693694	0.871795	0.827338	0.599206
F1	0.604736	0.604736	0.594724	0.612111	0.644880	0.691630	0.402089	0.618474	0.519084	0.554217	0.571970

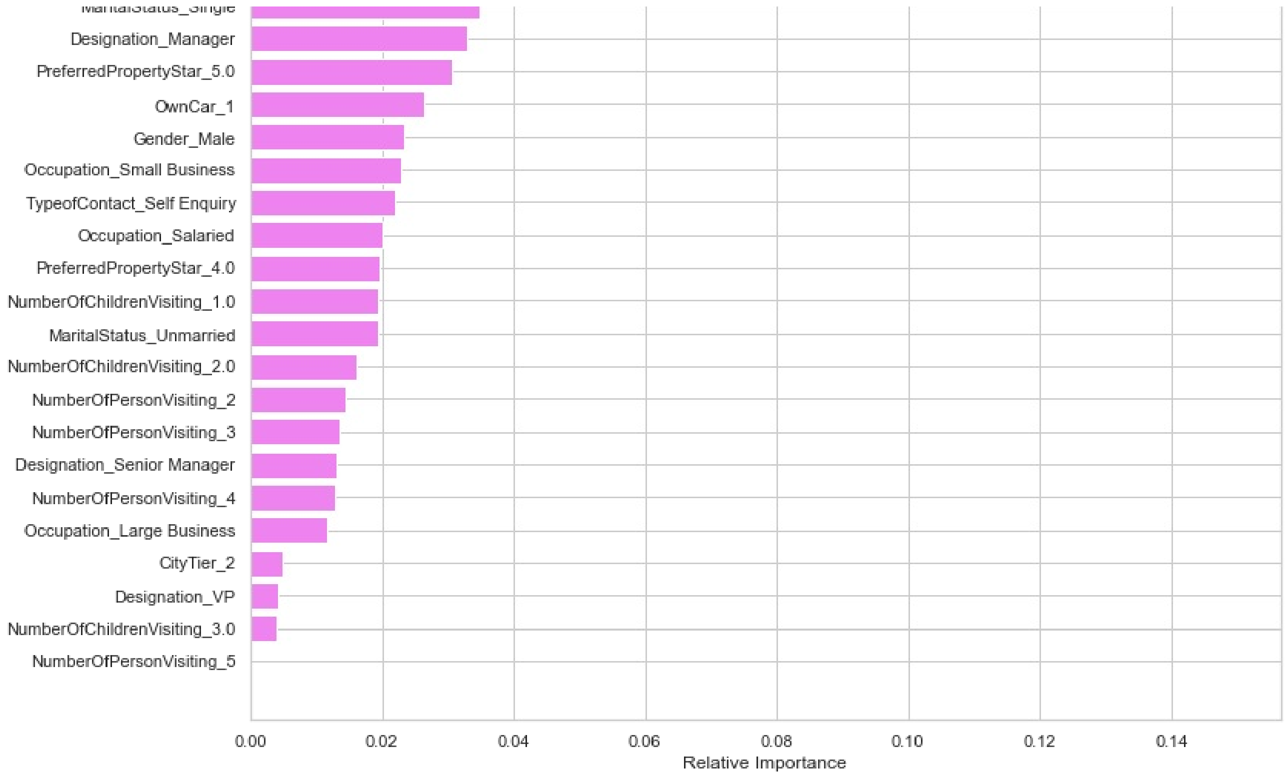
Observations:

- The majority of the models are overfitting the training data in terms of f1-score.
- The bagging estimator tuned is giving the highest f1-score on the test data but is overfitting the training data.
- Tuned Random Forest has more generalized metric scores and doesnt seem to be over-fitting the data

In [239…

```
feature_names = X_train.columns
importances = rf_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()
```





Observation:

- MonthlyIncome, Age and Passport_1 is the most important feature in identifying premium quality wine followed by sulfates and volatile acidity.
- This model has an 83.8% accuracy rate.

Recommendations:

- Age, MonthlyIncome and Passport are most important features for the prediction so the business can target customers with passport, higher age and higher monthly income customers.
- Average DurationofPitch is 3. Longer pitch duration doesnt effective on the product purchase. We should keep this in mind and plan the future presentations.
- Basic and Deluxe are the most popular packages. We can increase other package sales by marketing for example first class, second class and third class so we can attract all category customers.
- There was imbalance in data, only 18% of customers bought any product. This must be fixed for future analysis.
- NumberofChilden and NumberofPeoplevisiting doesnt have great impact on the prediction
- Since Single customers are buying product higher, the business can provide offers for married people to attract customers
- For customers whose NumberOfTrips is higher we can provide them some credit points/cash back to redeem for the future buy so that customers stick with us

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js