Tourism Package Prediction

HarvardX Data Science Professional Certificate: Capstone Project 2

Ricardo Mendes

2023-02-22

Contents

1 Introduction **5**

2 Analysis **6**

2.1 Loading and visualizing the data 6

2.2 Exploratory Data Analysis 12

2.2.1 Univariate Analysis 13

2.2.1.1 Age 13

2.2.1.2 Duration of Pitch 14

2.2.1.3 Monthly Income 14

2.2.1.4 Number of Trips 15

2.2.1.5 Number of Persons Visiting 16

2.2.1.6 Occupation 17

2.2.1.7 City Tier 18

2.2.1.8 Gender 18

2.2.1.9 Number of Follow-ups 19

2.2.1.10 Product Pitched 19

2.2.1.11 Type of Contact 20

2.2.1.12 Designation 21

2.2.1.13 Product Taken 21

2.2.2 Bivariate Analysis 22

2.3 Data Preparation 26

2.3.1 Remove unnecessary variables 26

2.3.2 Create dummy variables 26

2.3.3 Separate training and validation sets 27

2.3.4 Separate the independent variable 27

2.4 Building the Models 27

2.4.1 Evaluation Criterion 27

2.4.2 Logistic Regression 28

2.4.3 Support Vector Machine 31

2.4.3.1 Linear Kernel 31

2.4.3.2 RBF Kernel 35

2.4.4 Decision Tree 39

2.4.5 Random Forest 44

3 Results **47**

4 Conclusion **48**

```
if (!require("tools")) {
    install.packages("tools", dependencies = TRUE)
}
library(tools)

if (!require("base")) {
    install.packages("base", dependencies = TRUE)
}
library(base)

if (!require("dplyr")) {
    install.packages("dplyr", dependencies = TRUE)
}
library(dplyr)

if (!require("ggplot2")) {
    install.packages("ggplot2", dependencies = TRUE)
}
library(ggplot2)

if (!require("ggthemes")) {
    install.packages("ggthemes", dependencies = TRUE)
}
library(ggthemes)

if (!require("caret")) {
    install.packages("caret", dependencies = TRUE)
}
library(caret)

if (!require("readxl")) {
    install.packages("readxl", dependencies = TRUE)
}
library(readxl)

if (!require("tidyr")) {
    install.packages("tidyr", dependencies = TRUE)
}
library(tidyr)

if (!require("kableExtra")) {
    install.packages("kableExtra", dependencies = TRUE)
}
library(kableExtra)

if (!require("gridExtra")) {
    install.packages("gridExtra", dependencies = TRUE)
}
library(gridExtra)

if (!require("reshape2")) {
    install.packages("reshape2", dependencies = TRUE)
}
```

```
library(reshape2)

if (!require("useful")) {
    install.packages("useful", dependencies = TRUE)
}
library(useful)

if (!require("tidymodels")) {
    install.packages("tidymodels", dependencies = TRUE)
}
library(tidymodels)
tidymodels_prefer()

if (!require("fastDummies")) {
    install.packages("fastDummies", dependencies = TRUE)
}
library(fastDummies)

if (!require("glmnet")) {
    install.packages("glmnet", dependencies = TRUE)
}
library(glmnet)

if (!require("e1071")) {
    install.packages("e1071", dependencies = TRUE)
}
library(e1071)

if (!require("pROC")) {
    install.packages("pROC", dependencies = TRUE)
}
library(pROC)

if (!require("rpart")) {
    install.packages("rpart", dependencies = TRUE)
}
library(rpart)

if (!require("randomForest")) {
    install.packages("randomForest", dependencies = TRUE)
}
library(randomForest)
```

1 Introduction

A tourism company has five types of packages on its portfolio: Basic, Standard, Deluxe, Super Deluxe, and King. Not many customers purchased a package last year. It was hard to identify potential customers because they were contacted at random. That means none of the available information about each customer has been analyzed before deciding to offer a package.

The company is now planning to launch a new package called "Wellness Tourism". Its purpuse is to help those people looking for a way to start or maintain a healthy lifestyle. To avoid the previous mistakes, the company now wants to use the existing data to target the right customers, thus expanding the customer base.

The purpose of this project is to predict those customers who are potentially going to purchase the new package. Those predictions will be made before a customer is contacted.

The dataset comprises the following variables:

- **CustomerID:** Unique customer ID
- **ProdTaken:** Whether the customer has purchased a package or not (0: No, 1: Yes)
- **Age:** Age of the customer
- **TypeofContact:** How the 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 refers to the city the customer lives in.
- **DurationOfPitch:** Duration of the pitch by a salesperson to the customer
- **Occupation:** Occupation of the customer
- **Gender:** Gender of the customer ("Female" or "Male")
- **NumberOfPersonVisiting:** Total number of persons planning to take the trip with the customer
- **NumberOfFollowups:** Total number of follow-ups that have been done by the salesperson after the sales pitch
- **ProductPitched:** Product pitched by the salesperson
- **PreferredPropertyStar:** Hotel property rating preferred by the customer
- **MaritalStatus:** Marital status of the customer
- **NumberOfTrips:** Customer's average number of trips in a year
- **Passport:** Indicates if the customer has a passport or not (0: No, 1: Yes)
- **PitchSatisfactionScore:** Sales pitch satisfaction score
- **OwnCar:** Whether the customer owns 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

2 Analysis

2.1 Loading and visualizing the data

```
# Reads the data
data <- read_excel("Tourism.xlsx", sheet = "Tourism")

# Shows the first five rows
data %>%
    select(CustomerID:Occupation) %>%
    head() %>%
    kable()
```

CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation
200000	1	41	Self Enquiry	3	6	Salaried
200001	0	49	Company Invited	1	14	Salaried
200002	1	37	Self Enquiry	1	8	Free Lancer
200003	0	33	Company Invited	1	9	Salaried
200004	0	NA	Self Enquiry	1	8	Small Business
200005	0	32	Company Invited	1	8	Salaried

```
data %>%
    select(Gender:PreferredPropertyStar) %>%
    head() %>%
    kable()
```

Gender	NumberOfPersonVisiting	NumberOfFollowups	ProductPitched	PreferredPropertyStar
Female	3	3	Deluxe	3
Male	3	4	Deluxe	4
Male	3	4	Basic	3
Female	2	3	Basic	3
Male	2	3	Basic	4
Male	3	3	Basic	3

```
data %>%
    select(MaritalStatus:NumberOfChildrenVisiting) %>%
    head() %>%
    kable()
```

MaritalStatus	NumberOfTrips	Passport	PitchSatisfactionScore	OwnCar	NumberOfChildrenVisiting
Single	1	1	2	1	0
Divorced	2	0	3	1	2
Single	7	1	3	0	0
Divorced	2	1	5	1	1
Divorced	1	0	5	1	0
Single	1	0	5	1	1

```
data %>%
    select(Designation:MonthlyIncome) %>%
    head() %>%
    kable()
```

Designation	MonthlyIncome
Manager	20993
Manager	20130
Executive	17090
Executive	17909
Executive	18468
Executive	18068

This is the dataset size:

```
dim(data)
```

```
## [1] 4888   20
```

Checking if there are missing values:

```
as.data.frame(colSums(is.na(data))) %>%
    kable(col.names = c("# of NA"))
```

	# of NA
CustomerID	0
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

Some columns do have missing values that will have to be filled.

Checking the number of unique values in each column:

```
data %>%
    summarise_all((n_distinct)) %>%
    pivot_longer(everything(), names_to = c("Column Name"),
        values_to = c("# Unique Values"))
```

Column Name	# Unique Values
CustomerID	4888
ProdTaken	2
Age	45
TypeofContact	3
CityTier	3
DurationOfPitch	35
Occupation	4
Gender	3
NumberOfPersonVisiting	5
NumberOfFollowups	7
ProductPitched	5
PreferredPropertyStar	4
MaritalStatus	4
NumberOfTrips	13
Passport	2
PitchSatisfactionScore	5
OwnCar	2
NumberOfChildrenVisiting	5
Designation	5
MonthlyIncome	2476

The **CustomerID** column can be dropped since it has a unique value for each customer.

```
data <- select(data, -"CustomerID")
```

Looking at the statistics summary:

```
summary(data)
```

```
##    ProdTaken           Age        TypeofContact         CityTier
##  Min.   :0.0000   Min.   :18.00   Length:4888        Min.   :1.000
##  1st Qu.:0.0000   1st Qu.:31.00   Class :character   1st Qu.:1.000
##  Median :0.0000   Median :36.00   Mode  :character   Median :1.000
##  Mean   :0.1882   Mean   :37.62                      Mean   :1.654
##  3rd Qu.:0.0000   3rd Qu.:44.00                      3rd Qu.:3.000
##  Max.   :1.0000   Max.   :61.00                      Max.   :3.000
##                   NA's   :226
##  DurationOfPitch   Occupation          Gender          NumberOfPersonVisiting
##  Min.   :  5.00   Length:4888        Length:4888        Min.   :1.000
##  1st Qu.:  9.00   Class :character   Class :character   1st Qu.:2.000
##  Median : 13.00   Mode  :character   Mode  :character   Median :3.000
##  Mean   : 15.49                                        Mean   :2.905
##  3rd Qu.: 20.00                                        3rd Qu.:3.000
##  Max.   :127.00                                        Max.   :5.000
##  NA's   :251
##  NumberOfFollowups ProductPitched     PreferredPropertyStar MaritalStatus
##  Min.   :1.000     Length:4888        Min.   :3.000         Length:4888
##  1st Qu.:3.000     Class :character   1st Qu.:3.000         Class :character
##  Median :4.000     Mode  :character   Median :3.000         Mode  :character
##  Mean   :3.708                        Mean   :3.581
##  3rd Qu.:4.000                        3rd Qu.:4.000
##  Max.   :6.000                        Max.   :5.000
```

```
##  NA's   :45                            NA's   :26
##  NumberOfTrips       Passport      PitchSatisfactionScore     OwnCar
##  Min.   : 1.000   Min.   :0.0000   Min.   :1.000          Min.   :0.0000
##  1st Qu.: 2.000   1st Qu.:0.0000   1st Qu.:2.000          1st Qu.:0.0000
##  Median : 3.000   Median :0.0000   Median :3.000          Median :1.0000
##  Mean   : 3.237   Mean   :0.2909   Mean   :3.078          Mean   :0.6203
##  3rd Qu.: 4.000   3rd Qu.:1.0000   3rd Qu.:4.000          3rd Qu.:1.0000
##  Max.   :22.000   Max.   :1.0000   Max.   :5.000          Max.   :1.0000
##  NA's   :140
##  NumberOfChildrenVisiting Designation        MonthlyIncome
##  Min.   :0.000            Length:4888        Min.   : 1000
##  1st Qu.:1.000            Class :character   1st Qu.:20346
##  Median :1.000            Mode  :character   Median :22347
##  Mean   :1.187                               Mean   :23620
##  3rd Qu.:2.000                               3rd Qu.:25571
##  Max.   :3.000                               Max.   :98678
##  NA's   :66                                  NA's   :233
```

Some findings about the data:

- **Age** ranges from 18 to 61 years old but most people are between 31 and 44 years old.
- **DurationOfPitch** has some outliers since most values are less than 20 but the max value is 127.
- **MonthlyIncome** has outliers on both ends. While most people are between ~20K and ~26K, the minimum value is 1,000 and the maximum is 98,678.
- **ProdTaken** has most values equal to 0.

Since there are many categorical columns, here are the counts for each unique value:

```
table(data["TypeofContact"]) %>%
    kable()
```

TypeofContact	Freq
Company Invited	1419
Self Enquiry	3444

```
table(data["CityTier"]) %>%
    kable()
```

CityTier	Freq
1	3190
2	198
3	1500

```
table(data["Occupation"]) %>%
    kable()
```

Occupation	Freq
Free Lancer	2
Large Business	434
Salaried	2368
Small Business	2084

```
table(data["Gender"]) %>%
    kable()
```

Gender	Freq
Fe Male	155
Female	1817
Male	2916

```
table(data["NumberOfPersonVisiting"]) %>%
    kable()
```

NumberOfPersonVisiting	Freq
1	39
2	1418
3	2402
4	1026
5	3

```
table(data["NumberOfFollowups"]) %>%
    kable()
```

NumberOfFollowups	Freq
1	176
2	229
3	1466
4	2068
5	768
6	136

```
table(data["ProductPitched"]) %>%
    kable()
```

ProductPitched	Freq
Basic	1842
Deluxe	1732
King	230
Standard	742
Super Deluxe	342

```
table(data["PreferredPropertyStar"]) %>%
    kable()
```

PreferredPropertyStar	Freq
3	2993
4	913
5	956

```
table(data["MaritalStatus"]) %>%
    kable()
```

MaritalStatus	Freq
Divorced	950
Married	2340
Single	916
Unmarried	682

```
table(data["Passport"]) %>%
    kable()
```

Passport	Freq
0	3466
1	1422

```
table(data["PitchSatisfactionScore"]) %>%
    kable()
```

PitchSatisfactionScore	Freq
1	942
2	586
3	1478
4	912
5	970

```
table(data["OwnCar"]) %>%
    kable()
```

OwnCar	Freq
0	1856
1	3032

```
table(data["NumberOfChildrenVisiting"]) %>%
    kable()
```

NumberOfChildrenVisiting	Freq
0	1082
1	2080
2	1335
3	325

```
table(data["Designation"]) %>%
    kable()
```

Designation	Freq
AVP	342
Executive	1842
Manager	1732
Senior Manager	742
VP	230

By looking at these numbers, we can notice that:

- Under **Occupation**, there are only 2 customers in the "Free Lancer" category.
- There are three genders because of an input error. "Fe Male" must be replaced with "Female".
- For **NumberOfPersonVisiting**, there are only three customers whose value is 5.
- Most customers are married and own a car.

Before we proceed, let's fix the **Gender** variable:

```
# Replacing 'Fe Male' with 'Female'
data$Gender[data$Gender == "Fe Male"] <- "Female"
table(data["Gender"]) %>%
    kable()
```

Gender	Freq
Female	1972
Male	2916

Let's also remove the customers with "Free Lancer" as their occupation:

```
# Removing 'Free Lancer'
data <- data %>%
    filter(Occupation != "Free Lancer")
table(data["Occupation"]) %>%
    kable()
```

Occupation	Freq
Large Business	434
Salaried	2368
Small Business	2084

We also need to change some columns to factors:

```
factor_variables = c("TypeofContact", "CityTier", "Occupation",
    "Gender", "NumberOfPersonVisiting", "NumberOfFollowups",
    "ProductPitched", "PreferredPropertyStar", "MaritalStatus",
    "Passport", "PitchSatisfactionScore", "OwnCar",
    "NumberOfChildrenVisiting", "Designation")

data[factor_variables] <- lapply(data[factor_variables],
    factor)
```

2.2 Exploratory Data Analysis

In order to gain some knowledge about the data, we need to analyze its distribution. The following code contains a function that plots the distribution of continuous variables and another one that plots the number of occurrences for categorical variables:

```
# Plot the distribution of continuous variables
plot_distribution <- function(data, column, columnName) {
    # Create a histogram to analyze distribution
    hist_plot <- data %>%
        ggplot(aes(x = column)) + geom_histogram(aes(y = ..density..),
        fill = "#0000DD", binwidth = 0.5) + geom_density(alpha = 0.2,
        fill = "#DD0000") + xlab(columnName) + ylab("Density") +
        theme_economist()

    # Create a boxplot to analyze outliers and
    # common values
    box_plot <- data %>%
        ggplot(aes(x = column)) + geom_boxplot(fill = "#3377EE",
        color = "#000077", width = 0.05) + xlab("") +
```

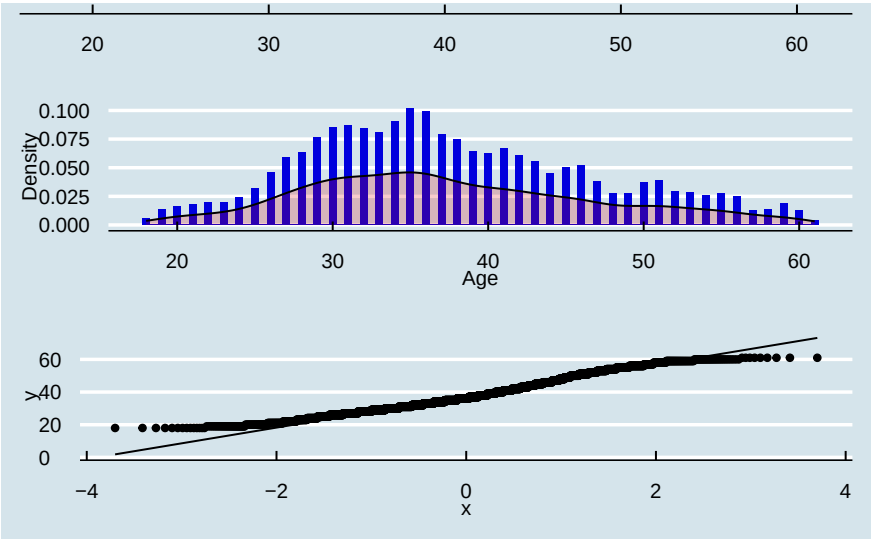
```
        theme_economist() + theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank())

    # Create the QQ-plot to determine if the
    # variable follows the normal distribution
    qq_plot <- data %>%
        ggplot(aes(sample = column)) + stat_qq() +
        stat_qq_line() + theme_economist()

    # Combine all plots
    grid.arrange(box_plot, hist_plot, qq_plot, ncol = 1,
        nrow = 3, heights = c(1, 3, 3))
}

# Plot the number of occurrences for categorical
# variables
plot_count <- function(data, column, columnName) {
    # Create a histogram with the number of
    # occurrences of each category
    data %>%
        ggplot(aes(x = column)) + geom_bar(fill = "#3377EE") +
        geom_text(aes(label = ..count..), stat = "count",
            vjust = -0.5, color = "#000077") + xlab(columnName) +
        theme_economist()
}
```

2.2.1 Univariate Analysis

Let's now check some variables.

2.2.1.1 Age

```
plot_distribution(data, data$Age, "Age")
```

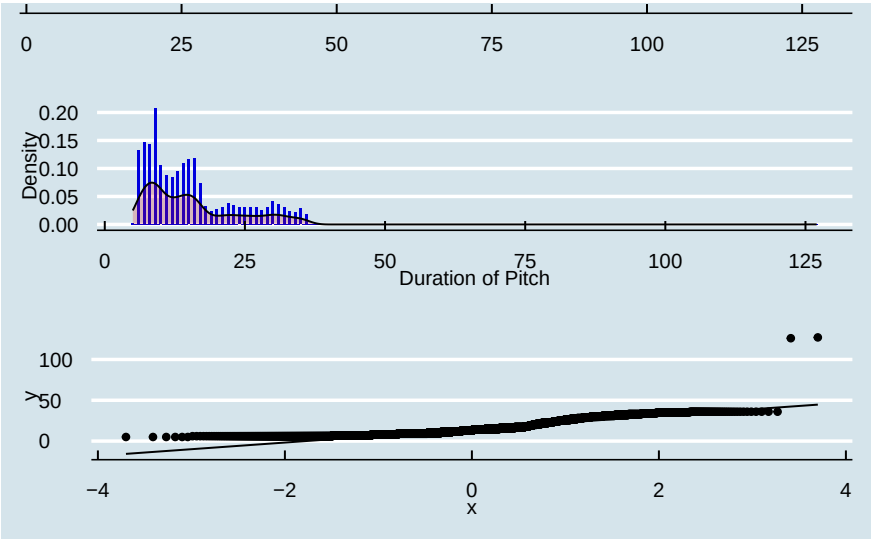


Observations:

- Ages look approximately normally distributed.
- There are no outliers.

2.2.1.2 Duration of Pitch

```
plot_distribution(data, data$DurationOfPitch, "Duration of Pitch")
```



Observations:

- The distribution is right-skewed.
- In most cases, the pitch lasts less than 20 minutes.
- There are some outliers.

```
data %>%
    filter(DurationOfPitch > 40)
```

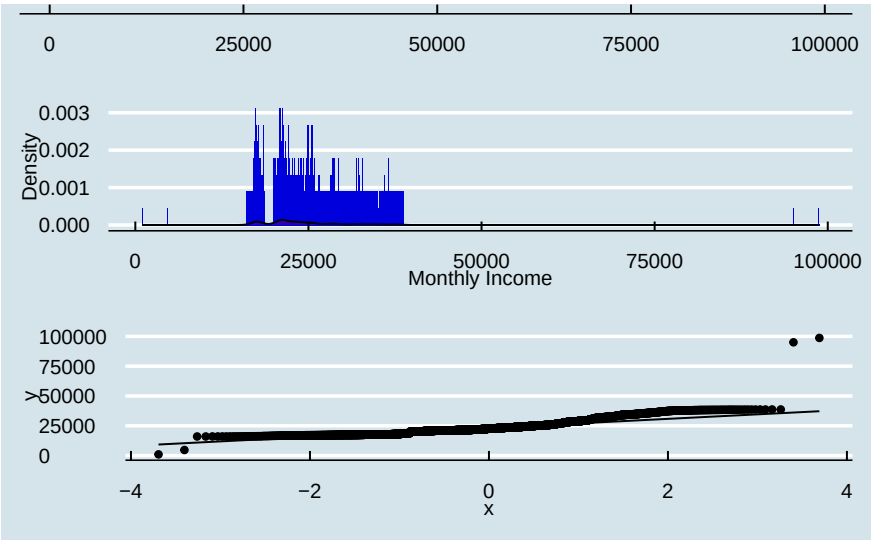
ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPersonVisiting
0	NA	Company Invited	3	126	Salaried	Male	2
0	53	Company Invited	3	127	Salaried	Male	3

From the above, we can see that there are only two outliers. Let's remove them.

```
data <- data %>%
    filter(DurationOfPitch <= 40)
```

2.2.1.3 Monthly Income

```
plot_distribution(data, data$MonthlyIncome, "Monthly Income")
```



Observations:

- Most values are between 20,000 and 40,000.
- There are some outliers on both extremes.

```
data %>%
    filter(MonthlyIncome < 12000 | MonthlyIncome >
        40000)
```

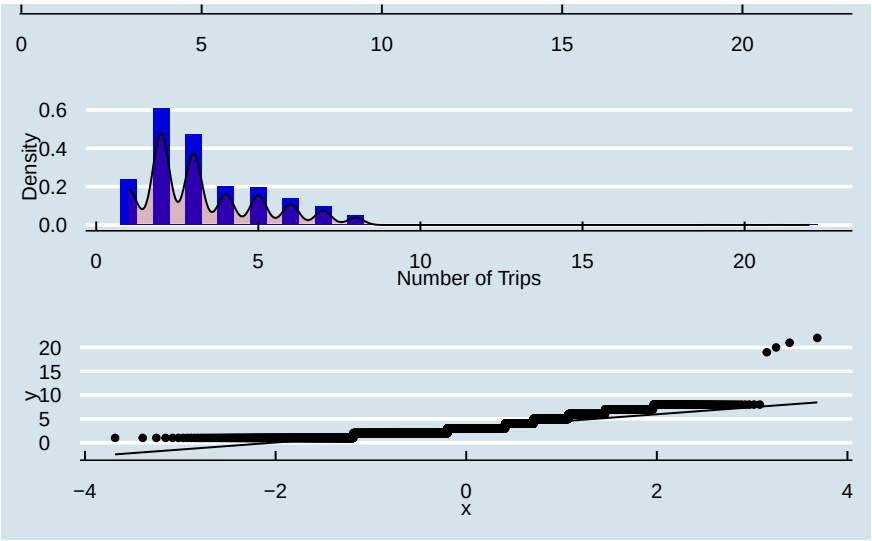
ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPersonVisiting
0	36	Self Enquiry	1	11	Salaried	Female	2
0	38	Self Enquiry	1	9	Large Business	Female	2
0	37	Self Enquiry	1	12	Salaried	Female	3
0	39	Self Enquiry	1	10	Large Business	Female	3

There are just four outliers. Let's remove them:

```
data <- data %>%
    filter(MonthlyIncome >= 12000 & MonthlyIncome <=
        40000)
```

2.2.1.4 Number of Trips

```
plot_distribution(data, data$NumberOfTrips, "Number of Trips")
```



Observations:

- The distribution is right-skewed.
- There are some outliers at the right end.

```
data %>%
    filter(NumberOfTrips > 10)
```

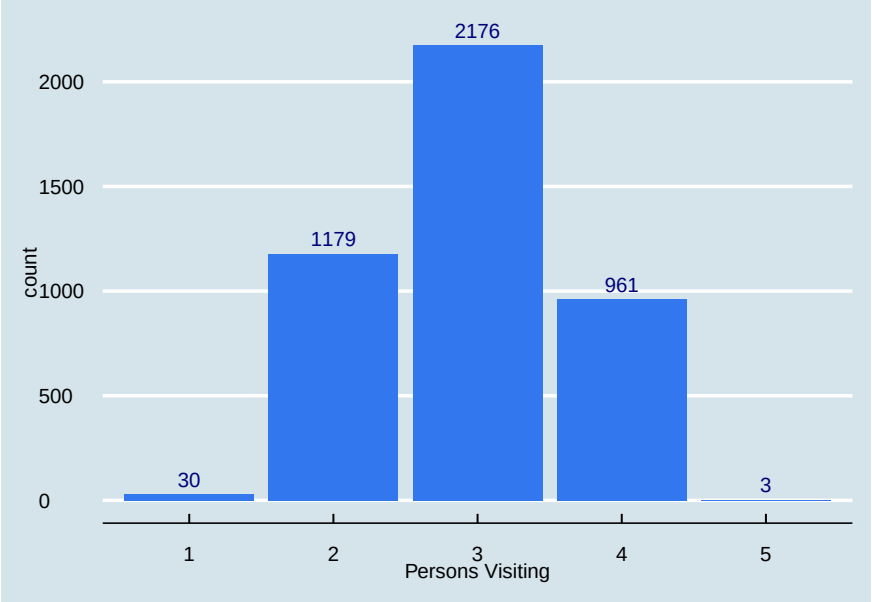
ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfPersonVisiti
1	30	Company Invited	1	10	Large Business	Male	2
0	39	Company Invited	1	15	Salaried	Male	3
1	31	Company Invited	1	11	Large Business	Male	3
0	40	Company Invited	1	16	Salaried	Male	4

There are only four outliers (more than 10 trips a year). Let's remove them:

```
data <- data %>%
    filter(NumberOfTrips <= 10)
```

2.2.1.5 Number of Persons Visiting

```
plot_count(data, data$NumberOfPersonVisiting, "Persons Visiting")
```

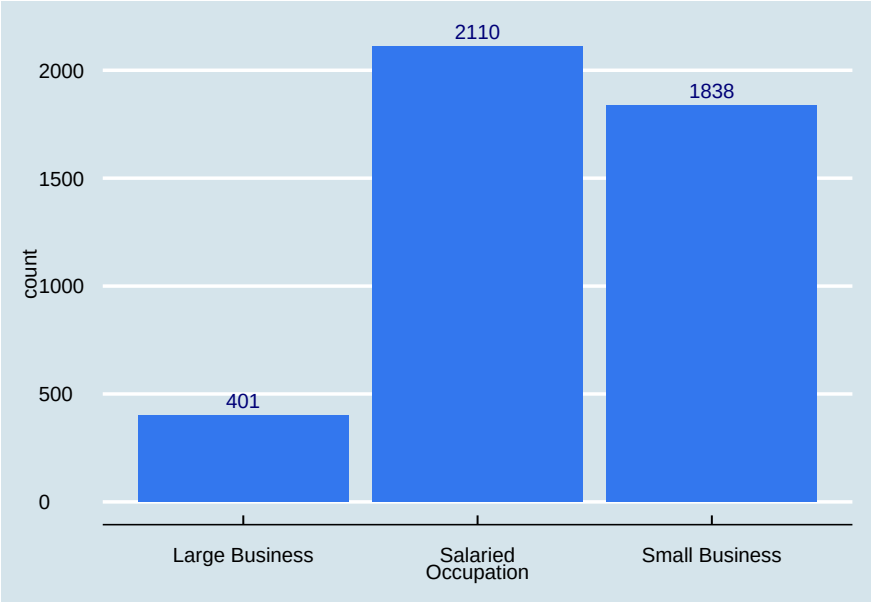


Observations:

- Most customers travel with 3 more persons.
- Only 3 customers traveled with 5 more persons.

2.2.1.6 Occupation

```
plot_count(data, data$Occupation, "Occupation")
```

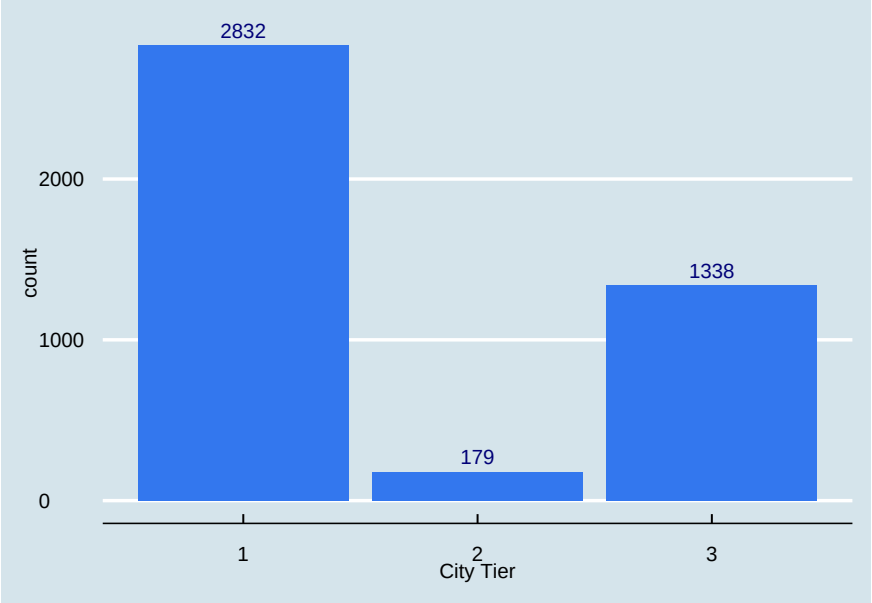


Observations:

- Most customers are either salaried or own a small business.

2.2.1.7 City Tier

```
plot_count(data, data$CityTier, "City Tier")
```

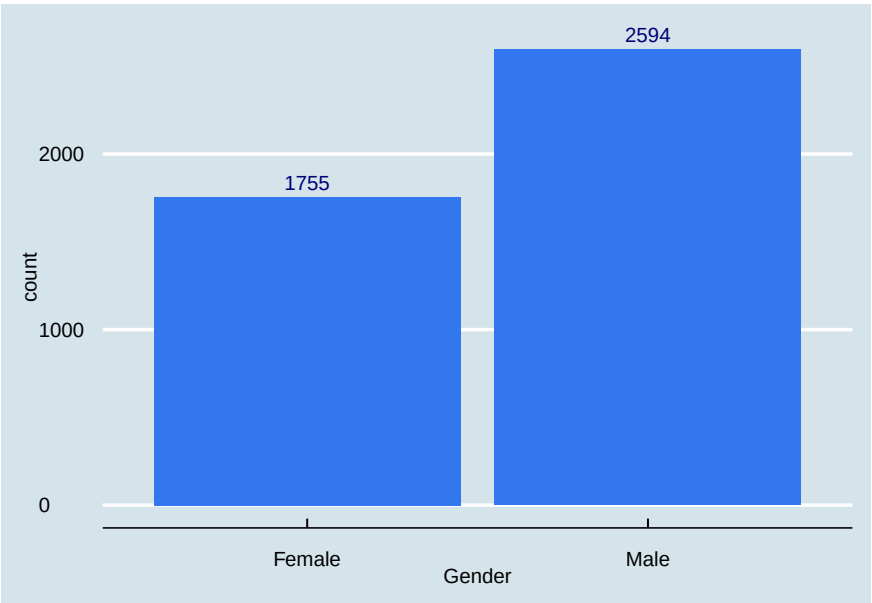


Observations:

- Most customers are from tier 1 cities. That could happen due to the better living standards compared to tiers 2 and 3.
- There are much more customers from tier 3 cities than from tier 2 cities. That is unexpected, but could be due to less marketing in tier 2 cities.

2.2.1.8 Gender

```
plot_count(data, data$Gender, "Gender")
```

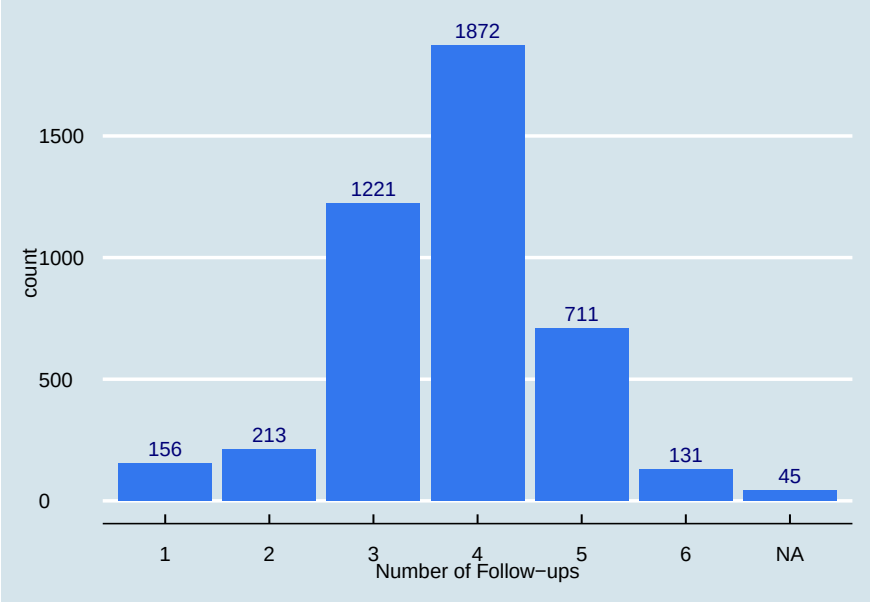


Observations:

- There are around 60% of males and 40% of females. That could happen because when families travel together, men usually do the inquiry and booking.

2.2.1.9 Number of Follow-ups

```
plot_count(data, data$NumberOfFollowups, "Number of Follow-ups")
```

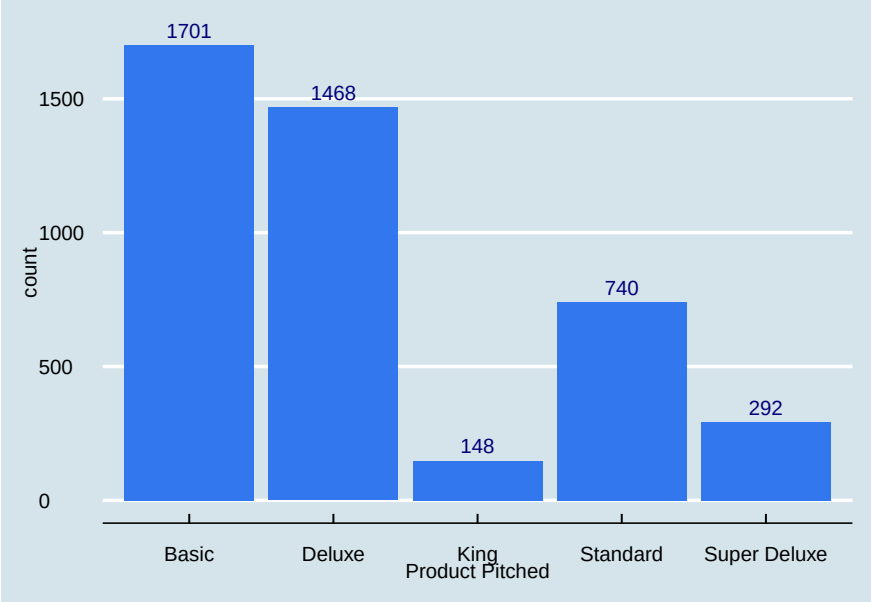


Observations:

- It usually takes 3-5 follow-ups with customers.

2.2.1.10 Product Pitched

```
plot_count(data, data$ProductPitched, "Product Pitched")
```

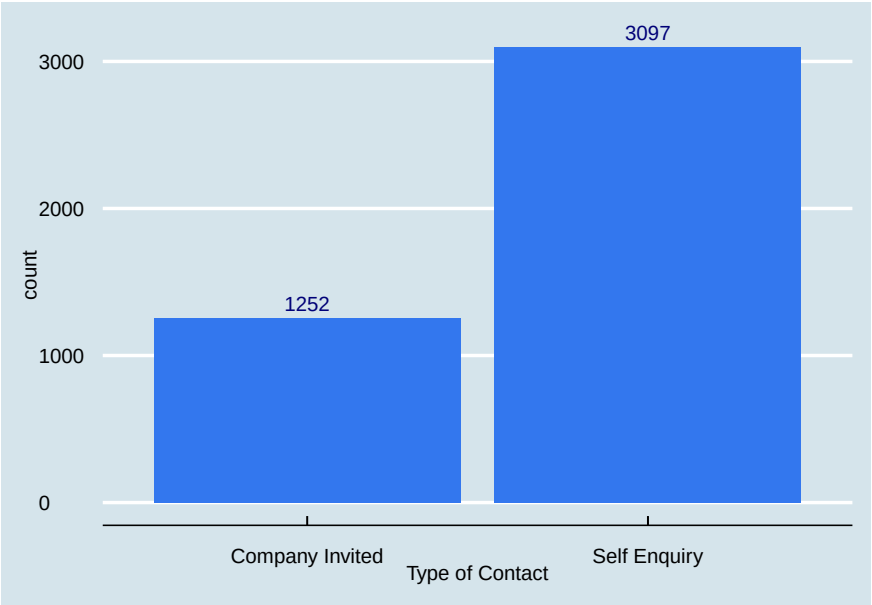


Observations:

- The company pitches the Basic and Deluxe packages more than the other packages. That might be because the profit is greater for these packages or because they are cheaper and, therefore, preferred by most customers.

2.2.1.11 Type of Contact

```
plot_count(data, data$TypeofContact, "Type of Contact")
```

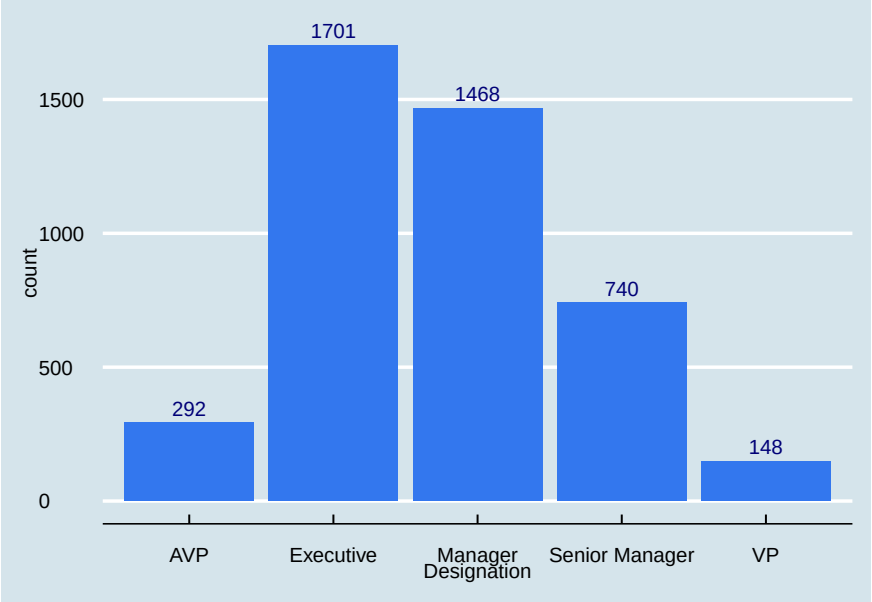


Observations:

- Around 70% of the customers reached out to the company first, showing a positive outreach.

2.2.1.12 Designation

```
plot_count(data, data$Designation, "Designation")
```

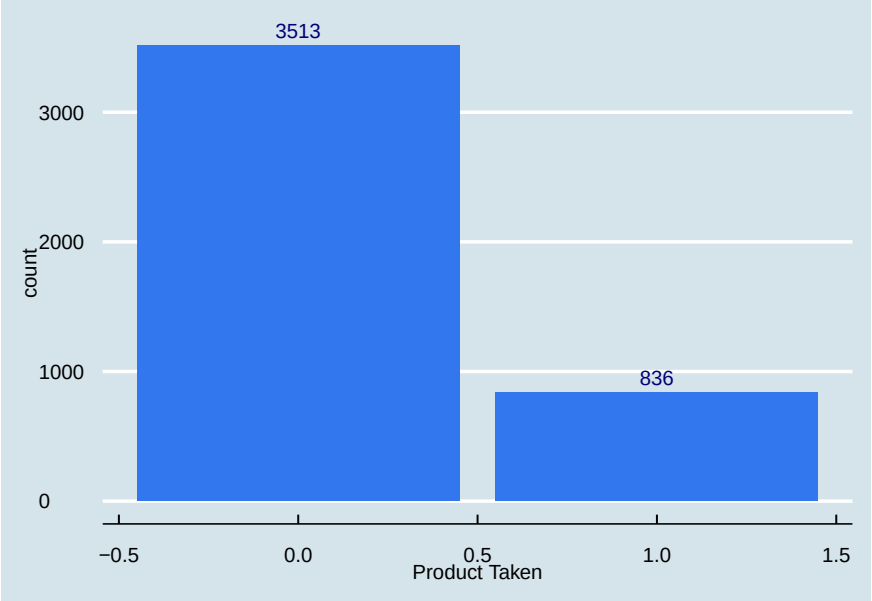


Observations:

- Most customers are at the manager or executive level.
- The higher the position, the lesser number of observations. That is probably caused because there are fewer people in higher roles.

2.2.1.13 Product Taken

```
plot_count(data, data$ProdTaken, "Product Taken")
```



Observations:

- Only ~20% of the customers purchased the product.

2.2.2 Bivariate Analysis

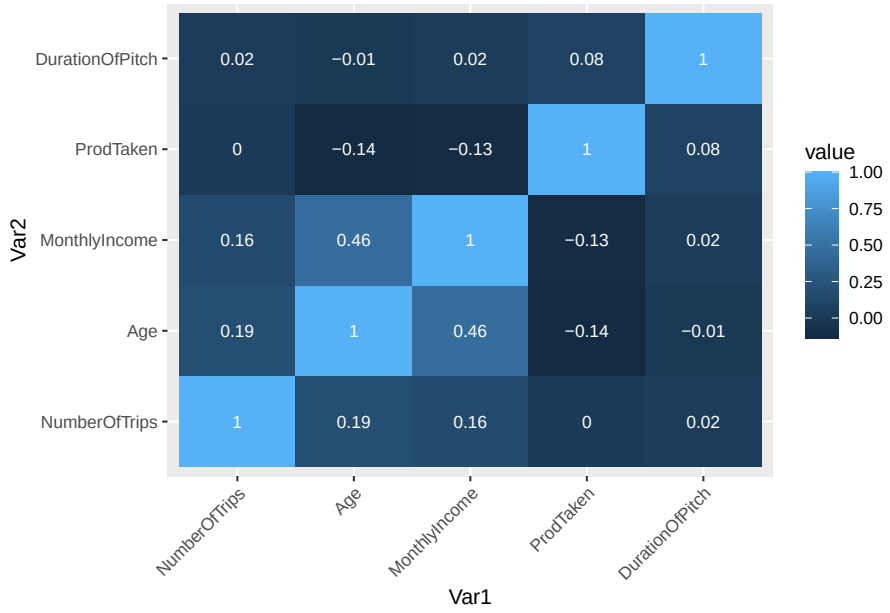
Before we can make some analysis, we need to fix the missing values on the dataset.

```
# Set the missing values to the median on the Age
# column
data$Age <- simple.impute(data$Age, fun = median)

# Set the missing values to the most frequent
# value on categorical columns

Mode <- function(x) {
    ux <- sort(unique(x))
    ux[which.max(tabulate(match(x, ux)))]
}


data$NumberOfFollowups <- simple.impute(data$NumberOfFollowups,
    fun = Mode)
data$PreferredPropertyStar <- simple.impute(data$PreferredPropertyStar,
    fun = Mode)
data$NumberOfChildrenVisiting <- simple.impute(data$NumberOfChildrenVisiting,
    fun = Mode)

# Check that there are no missing values anymore
as.data.frame(colSums(is.na(data))) %>%
    kable(col.names = c("# of NA"))
```

	# of NA
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

Now, let's take a look at the correlation matrix:

```
numeric_data <- select_if(data, is.numeric)
correlation_matrix <- round(cor(numeric_data), 2)
distance <- as.dist((1 - correlation_matrix)/2)
hc <- hclust(distance)

correlation_matrix <- correlation_matrix[hc$order,
    hc$order]

melt(correlation_matrix) %>%
    ggplot(aes(x = Var1, y = Var2, fill = value)) +
    geom_tile() + geom_text(aes(Var2, Var1, label = value),
    color = "#FFFFFF", size = 3) + theme(axis.text.x = element_text(angle = 45,
    hjust = 1))
```



Observations:

- There is a weak correlation between **Age** and **NumberOfTrips**, which is expected since the older someone is, the more trips they are expected to have made.
- There is also a correlation between **Age** and **MonthlyIncome**, which is also expected since the person tends to get higher roles and earn more with time.
- There is a negative correlation between **Age** and **ProdTaken**, indicating that older people are less interested in purchasing packages.
- There is no significative correlation between the other variables.

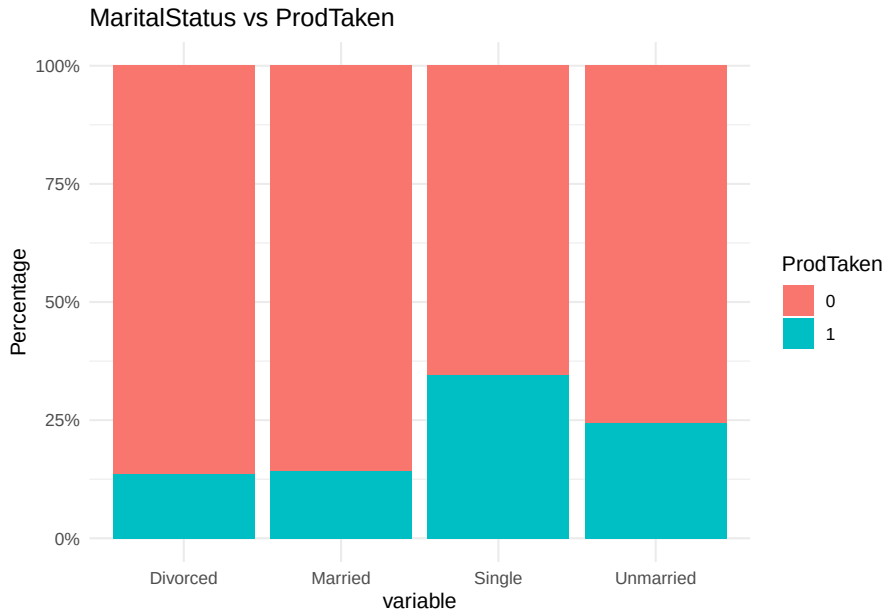
Since the correlation matrix considers only the numeric variables, we can check how the categorical variables influence the purchase of a package.

Let's first convert the **ProdTaken** variable to factor and create a function to plot a variable against **ProdTaken**:

```
# Convert ProdTaken to factor
data["ProdTaken"] <- lapply(data["ProdTaken"], factor)
```

```
plot_var_vs_prod_taken <- function(data, variable,
    variableName) {
    data %>%
        ggplot(aes(x = variable, fill = ProdTaken)) +
        geom_bar(position = "fill", stat = "count") +
        labs(y = "Percentage") + labs(title = paste(variableName,
        "vs ProdTaken")) + theme_minimal() + scale_y_continuous(labels = scales::percent)
}
```

```
plot_var_vs_prod_taken(data, data$MaritalStatus, "MaritalStatus")
```

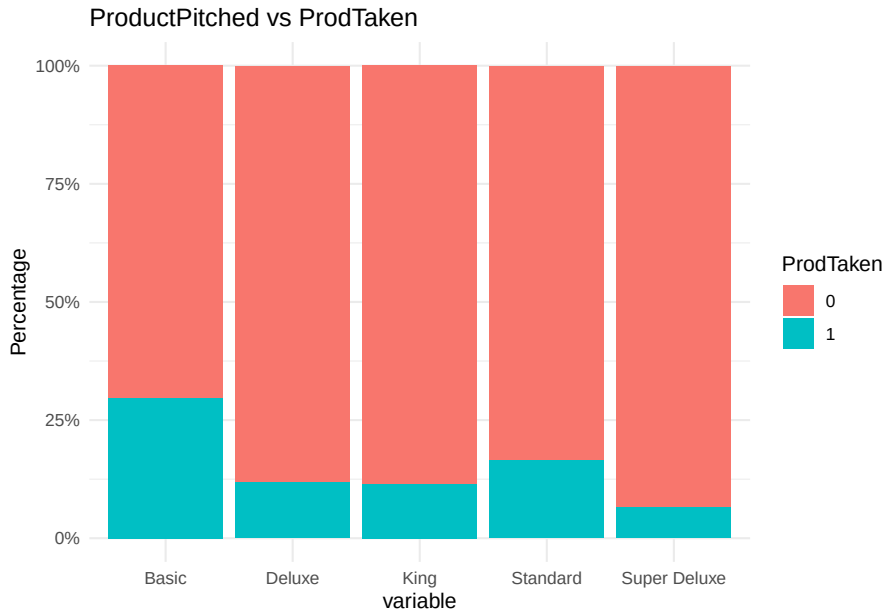


Observations:

- Although there are more married customers, those who purchase a package are mostly single and unmarried.
- The company may want to target single and unmarried customers more and adapt the packages for these types of customer.

Let's now take a look at the products pitched:

```
plot_var_vs_prod_taken(data, data$ProductPitched, "ProductPitched")
```

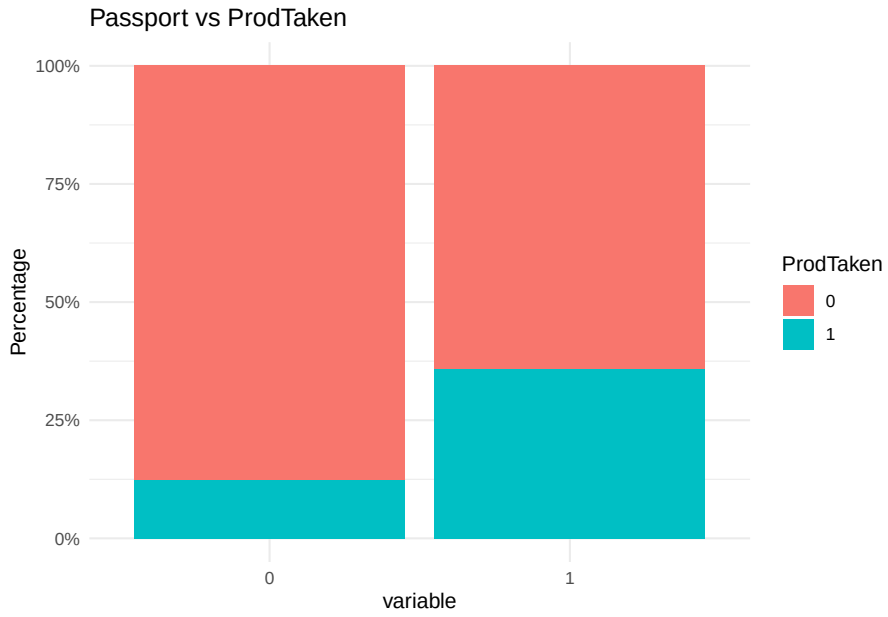


Observations:

- The Basic package has a higher conversion rate, possibly because it is the less expensive.
- Although the Deluxe package is pitched more often, the Standard package has a higher conversion rate. Therefore, the company could pitch the Standard package more often.

Let's now check if owning a passport is an important variable for taking a product:

```
plot_var_vs_prod_taken(data, data$Passport, "Passport")
```



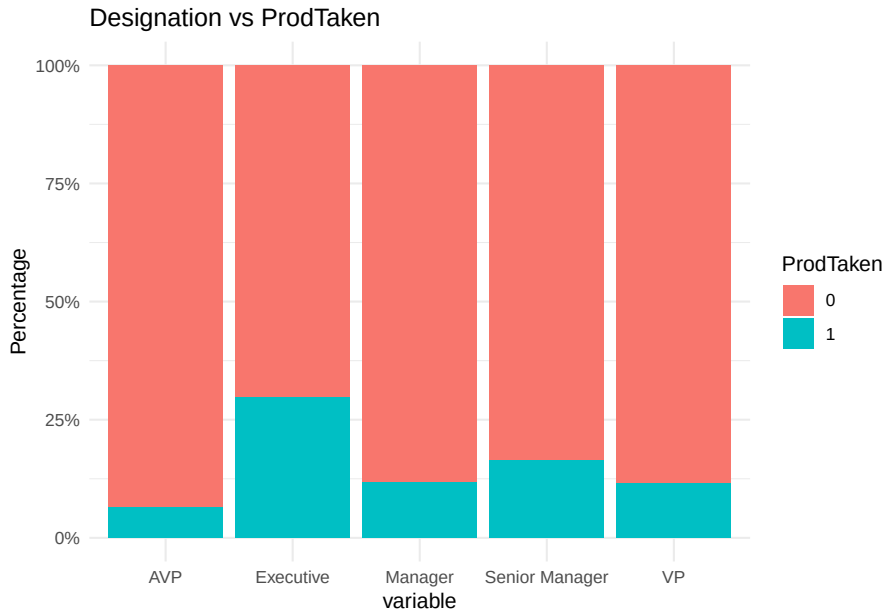
Observations:

- Customers with a passport purchase packages more than those who don't own one.

- There could be more international packages to attract such customers.

Let's now see how the **Designation** influences the **ProdTaken** variable:

```
plot_var_vs_prod_taken(data, data$Designation, "Designation")
```



Observations:

- The conversion rate is higher among executives and lower among VPs and AVPs.

2.3 Data Preparation

2.3.1 Remove unnecessary variables

Since the company aims to predict if a customer will purchase a package before reaching out to it, variables that hold data that is only available after the first contact should not be used. Let's remove them:

```
data <- select(data, -c("DurationOfPitch", "NumberOfFollowups",
    "ProductPitched", "PitchSatisfactionScore"))
```

2.3.2 Create dummy variables

Categorical (factor) variables that do not contain only TRUE/FALSE values, must be split in dummy variables to indicate each value.

```
data <- dummy_cols(data, select_columns = c("TypeofContact",
    "Occupation", "Gender", "MaritalStatus", "Designation",
    "CityTier"), remove_selected_columns = TRUE)
```

2.3.3 Separate training and validation sets

The dataset contains a large imbalance between positive and negative cases of the target variable. There are a lot more cases of no packages having been purchased than otherwise. Because of that, we need to separate the training and the validation sets preserving the frequency of both cases.

```
# Setting a seed to keep the results reproducible
set.seed(100)

# Splits the data in 70% for training and 30% for
# testing
data_split <- initial_split(data, prop = 0.7, strata = ProdTaken)
data_train <- training(data_split)
data_test <- testing(data_split)
```

2.3.4 Separate the independent variable

Since the **ProdTaken** variable is our target (or independent) variable, we need to separate it from the others.

```
# Creates a X dataset containing the dependent
# variables and Y containing the independent
# variable for both training and testing
X_train <- select(data_train, -"ProdTaken")
Y_train <- data_train["ProdTaken"]

X_test <- select(data_test, -"ProdTaken")
Y_test <- data_test["ProdTaken"]
```

2.4 Building the Models

2.4.1 Evaluation Criterion

No model is perfect and it will make wrong predictions. These errors can be of two types:

- Predicting a customer will buy a product and they don't buy it: this is a false positive. In business terms, it causes waste of resources since the company believes it is worth to follow up with them and try to convince them to purchase the product.
- Predicting a customer won't buy a product and they buy it: this is a false negative. In the business perspective, it causes an opportunity to be lost, decreasing the revenue. Since the model predicts that that customer will not purchase anything, they will not be targeted by the marketing team.

It is clear that the second type of errors is worst for the business, that loses a potential source of income. To reduce it, we need to maximize the recall since the greater the recall the lesser the chances of false negatives.

In order to select the best model, we are going to try four approaches:

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

For each model, we are going to perform the following steps:

1. Train the model.
2. Predict the target variable using the training data and check the model performance.
3. Predict the target variable using the test data and check the model performance.
4. Optimize the model, if possible.
5. Use the optimized model to predict the training data and check its performance.
6. Use the optimized model to predict the test data and check its performance.

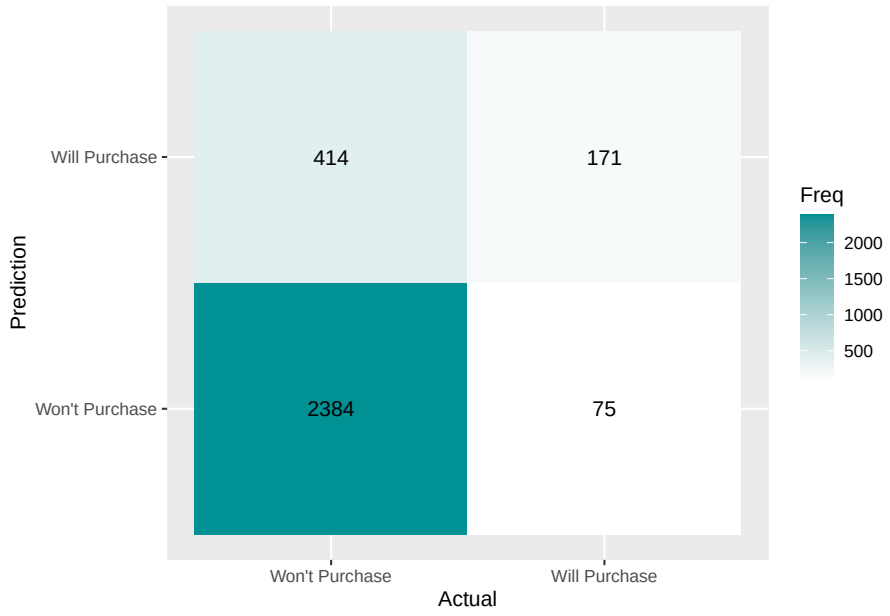
We will also need a function to show the results of each one:

```
show_results <- function(actual, predicted) {
    # Generate the confusion matrix
    cm <- confusionMatrix(predicted, actual)
    plt <- as.data.frame(cm$table)
    plt$Prediction <- factor(plt$Prediction, levels = levels(plt$Prediction))

    # Plot the confusion matrix in a graphical
    # way
    print(ggplot(plt, aes(Prediction, Reference, fill = Freq)) +
        geom_tile() + geom_text(aes(label = Freq)) +
        scale_fill_gradient(low = "white", high = "#009194") +
        labs(x = "Actual", y = "Prediction") + scale_x_discrete(labels = c("Won't Purchase",
        "Will Purchase")) + scale_y_discrete(labels = c("Won't Purchase",
        "Will Purchase")))

    # Return the accuracy and the recall so that
    # we can compare the models
    return(list(accuracy = cm$overall["Accuracy"],
        recall = caret::recall(actual, predicted)))
}
```

2.4.2 Logistic Regression

The first model to be tested is logistic regression.

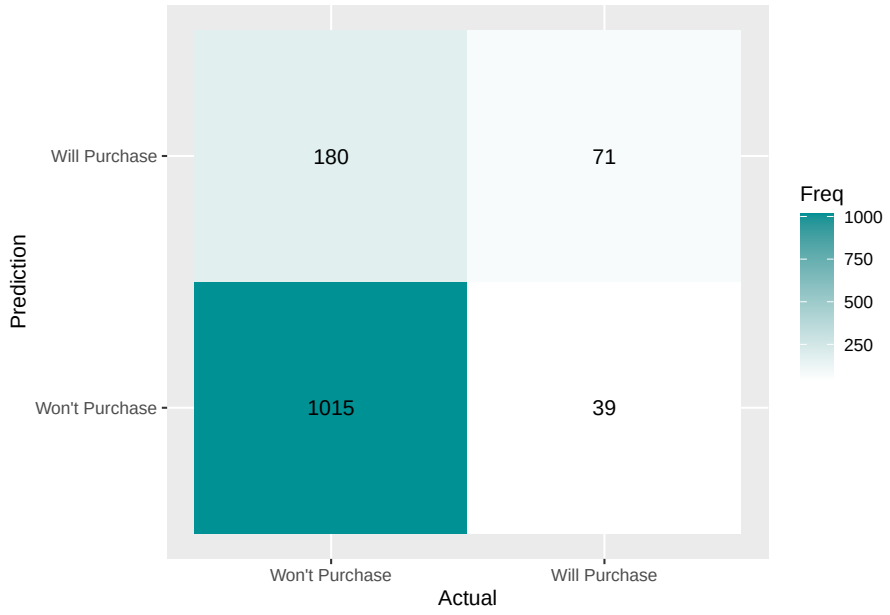
```
# Train the logistic regression model using the
# training data
lg_fit <- glm(ProdTaken ~ ., data = data_train, family = "binomial")

# Predict the target variable using the training
# data
lg_scores <- predict(lg_fit, data_train, type = "response")
y_hat_train <- ifelse(lg_scores > 0.5, 1, 0) |>
    factor()

# Evaluate the model performance on the training
# data
lg_train_result <- show_results(Y_train$ProdTaken,
    y_hat_train)
```




```
# Store the accuracy and the recall
lg_train_accuracy <- lg_train_result$accuracy
lg_train_recall <- lg_train_result$recall
```

Logistic regression achieved an accuracy of 0.8393561 and a recall of 0.8520372 on the training set.

```
# Predict the target variable on the test set
y_hat_test <- ifelse(predict(lg_fit, data_test, type = "response") >
    0.5, 1, 0) |>
    factor()

# Evaluate the model performance on the test set
lg_test_result <- show_results(Y_test$ProdTaken, y_hat_test)
```



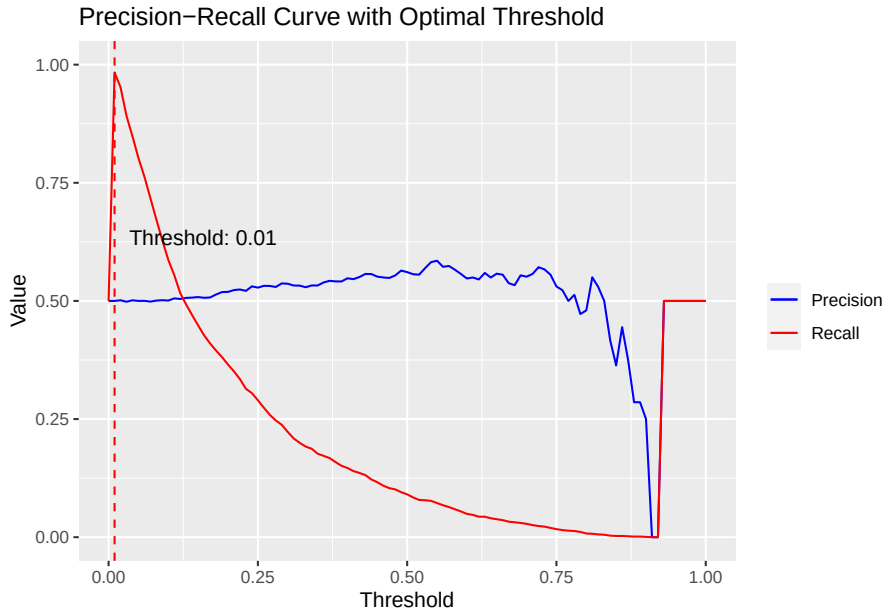
```
# Store the accuracy and the recall
lg_test_accuracy <- lg_test_result$accuracy
lg_test_recall <- lg_test_result$recall
```

Logistic regression achieved an accuracy of 0.8321839 and a recall of 0.8493724 on the test set.

Let's check if there is a threshold value that could help us maximize the recall without losing too much precision.

```
# Create a data frame with the predicted scores
# and actual labels
pr_df <- data.frame(scores = as.numeric(lg_scores),
    labels = as.numeric(levels(data_train$ProdTaken)))

# Calculate precision and recall for different
# threshold values
precision_recall <- lapply(seq(0, 1, 0.01), function(threshold) {
    # Compute confusion matrix
    cm <- matrix(c(table(pr_df$labels, pr_df$scores >
        threshold)), ncol = 2, nrow = 2)

    # Calculate precision and recall
    precision <- cm[2, 2]/sum(cm[, 2])
    recall <- cm[2, 2]/sum(cm[2, ])

    # Return a data frame with threshold,
    # precision, and recall
    data.frame(threshold = threshold, precision = precision,
        recall = recall)
})

# Combine the results into a single data frame
precision_recall_df <- do.call(rbind, precision_recall)

# Calculate F1 score
precision_recall_df$F1 <- 2 * precision_recall_df$precision *
    precision_recall_df$recall/(precision_recall_df$precision +
    precision_recall_df$recall)

# Find the threshold value that maximizes the F1
# score to have precision and recall balanced
optimal_threshold <- precision_recall_df$threshold[which.max(precision_recall_df$F1)]

# Plot precision-recall curve
ggplot(data = precision_recall_df, aes(x = threshold)) +
    geom_line(aes(y = precision, color = "Precision")) +
    geom_line(aes(y = recall, color = "Recall")) +
    scale_x_continuous(name = "Threshold", limits = c(0,
        1)) + scale_y_continuous(name = "Value", limits = c(0,
    1)) + labs(title = "Precision-Recall Curve with Optimal Threshold") +
    geom_vline(xintercept = optimal_threshold, linetype = "dashed",
        color = "red") + scale_color_manual(name = "",
    values = c(Precision = "blue", Recall = "red")) +
    annotate("text", x = optimal_threshold, y = max(precision_recall_df$precision),
```

```
        label = paste("Threshold:", round(optimal_threshold,
            2)), vjust = -1, hjust = -0.1)
```



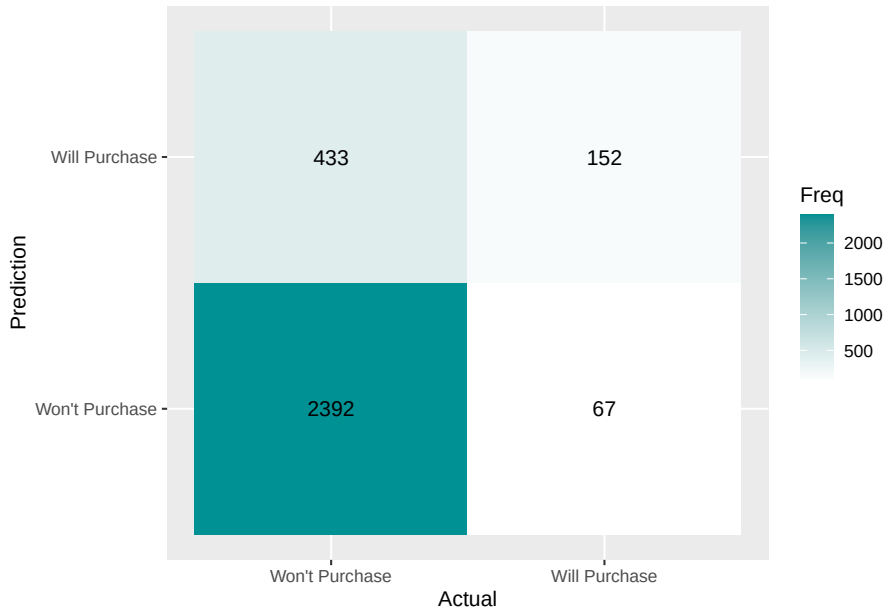
Since the optimal threshold is 0.01, there is no need to test with it. It wouldn't make sense to consider that only those scores lower than 0.01 represent customers that will not purchase a package as they are the most.

2.4.3 Support Vector Machine

The second model we are going to test is the Support Vector Machine (SVM). It will be tested with both linear and RBF kernels.

2.4.3.1 Linear Kernel

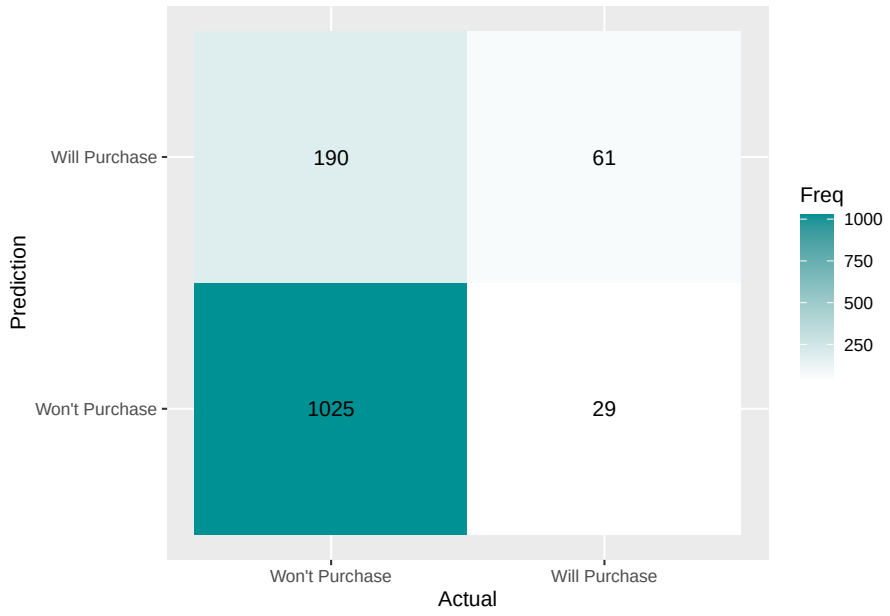
```
# Create an SVM model using the linear kernel and
# train it using the training data
svm_fit_linear <- svm(ProdTaken ~ ., data = data_train,
    kernel = "linear", probability = TRUE)

# Use the model to predict the target variable on
# the training set
y_hat_svm_linear_train <- predict(svm_fit_linear, data_train,
    probability = TRUE)

# Evaluate the model performance on the training
# data
svm_linear_train_result <- show_results(data_train$ProdTaken,
    y_hat_svm_linear_train)
```




```
# Store the accuracy and the recall
svm_linear_train_accuracy <- svm_linear_train_result$accuracy
svm_linear_train_recall <- svm_linear_train_result$recall
```

The SVM model using the linear kernel achieved an accuracy of 0.8357424 and a recall of 0.8467257 on the training data.

```
# Use the model to predict the target variable on
# the test set
y_hat_svm_linear_test <- predict(svm_fit_linear, data_test,
    probability = TRUE)

# Evaluate the model performance on the test set
svm_linear_test_result <- show_results(data_test$ProdTaken,
    y_hat_svm_linear_test)
```



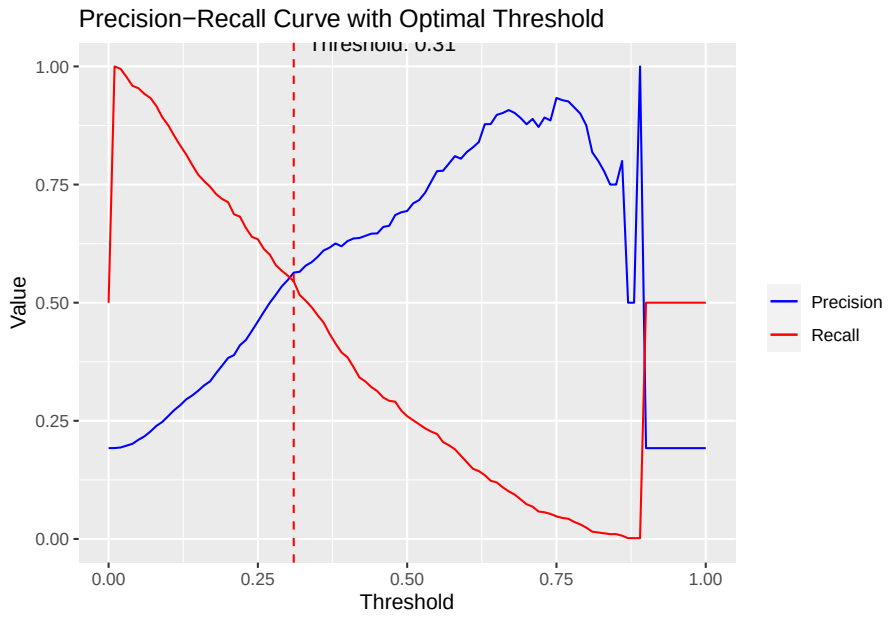

```
# Store the accuracy and the recall
svm_linear_test_accuracy <- svm_linear_test_result$accuracy
svm_linear_test_recall <- svm_linear_test_result$recall
```

The SVM model using the linear kernel achieved an accuracy of 0.8321839 and a recall of 0.8436214 on the test data.

Let's check if there is a threshold value that could help maximize the recall.

```
svm_scores <- attr(y_hat_svm_linear_train, "probabilities")

y_scores_pos <- svm_scores[, 2]

# Create a data frame with the predicted scores
# and actual labels
pr_df <- data.frame(scores = y_scores_pos, labels = data_train$ProdTaken)

# Calculate precision and recall for different
# threshold values
precision_recall <- lapply(seq(0, 1, 0.01), function(threshold) {
    # Compute confusion matrix
    cm <- matrix(c(table(pr_df$labels, pr_df$scores >
        threshold)), ncol = 2, nrow = 2)

    # Calculate precision and recall
    precision <- cm[2, 2]/sum(cm[, 2])
    recall <- cm[2, 2]/sum(cm[2, ])

    # Return a data frame with threshold,
    # precision, and recall
    data.frame(threshold = threshold, precision = precision,
        recall = recall)
})
```

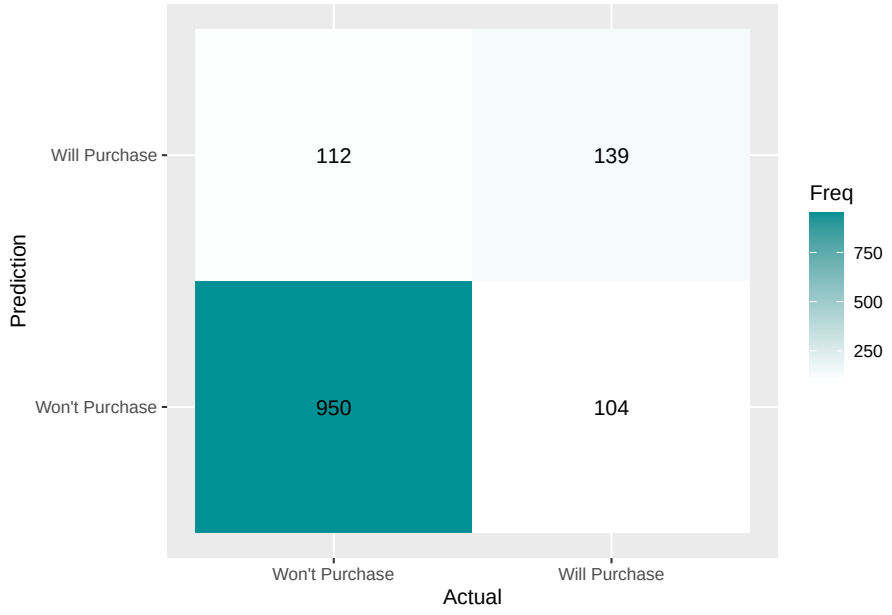
```r
# Combine the results into a single data frame
precision_recall_df <- do.call(rbind, precision_recall)

# Calculate F1 score
precision_recall_df$F1 <- 2 * precision_recall_df$precision *
    precision_recall_df$recall/(precision_recall_df$precision +
    precision_recall_df$recall)

# Find the threshold value that maximizes the F1
# score to have precision and recall balanced
optimal_threshold <- precision_recall_df$threshold[which.max(precision_recall_df$F1)]

# Plot precision-recall curve
ggplot(data = precision_recall_df, aes(x = threshold)) +
    geom_line(aes(y = precision, color = "Precision")) +
    geom_line(aes(y = recall, color = "Recall")) +
    scale_x_continuous(name = "Threshold", limits = c(0,
        1)) + scale_y_continuous(name = "Value", limits = c(0,
    1)) + labs(title = "Precision-Recall Curve with Optimal Threshold") +
    geom_vline(xintercept = optimal_threshold, linetype = "dashed",
        color = "red") + scale_color_manual(name = "",
    values = c(Precision = "blue", Recall = "red")) +
    annotate("text", x = optimal_threshold, y = max(precision_recall_df$precision),
        label = paste("Threshold:", round(optimal_threshold,
            2)), vjust = -1, hjust = -0.1)
```



As shown in the chart above, the optimal threshold is 0.31. Let's use it and check if recall is improved.

```r
# Predict the target variable
y_hat_svm_linear_opt <- predict(svm_fit_linear, data_test,
    probability = TRUE)

# Check the model performance using the optimal
```

```
# threshold.
svm_linear_opt_result <- show_results(data_test$ProdTaken,
    factor(as.numeric(attr(y_hat_svm_linear_opt, "probabilities")[,
        2] > optimal_threshold)))
```

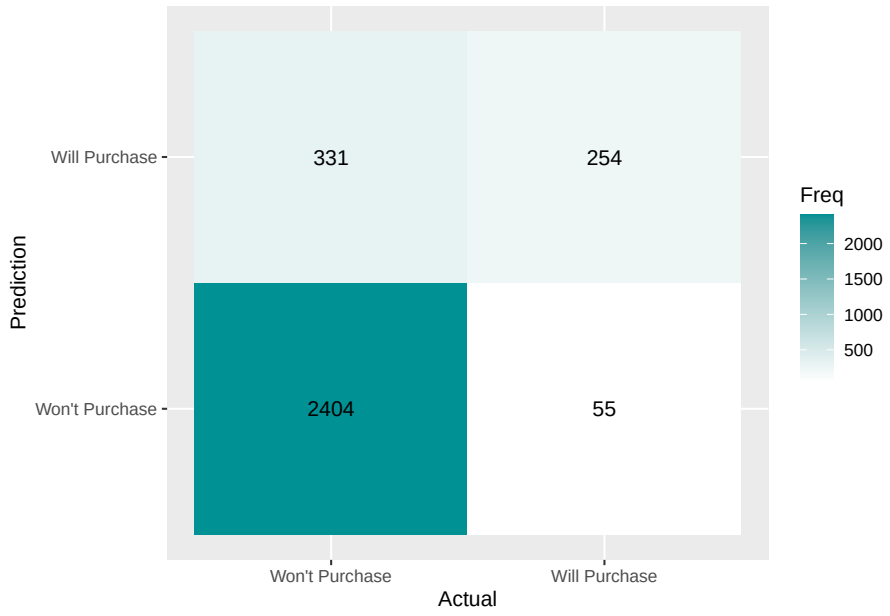



```
# Store the accuracy and the recall
svm_linear_opt_accuracy <- svm_linear_opt_result$accuracy
svm_linear_opt_recall <- svm_linear_opt_result$recall
```

Using the optimal threshold, the SVM model with the linear kernel achieved an accuracy of 0.8344828 and a recall of 0.8945386.

2.4.3.2 RBF Kernel

```
# Create an SVM model using the RBF kernel and
# train it using the training data
svm_fit_rbf <- svm(ProdTaken ~ ., data = data_train,
    kernel = "radial", probability = TRUE)

# Predict the target variable on the training
# data
y_hat_svm_rbf_train <- predict(svm_fit_rbf, data_train,
    probability = TRUE)

# Evaluate the performance of the model
svm_rbf_train_result <- show_results(data_train$ProdTaken,
    y_hat_svm_rbf_train)
```

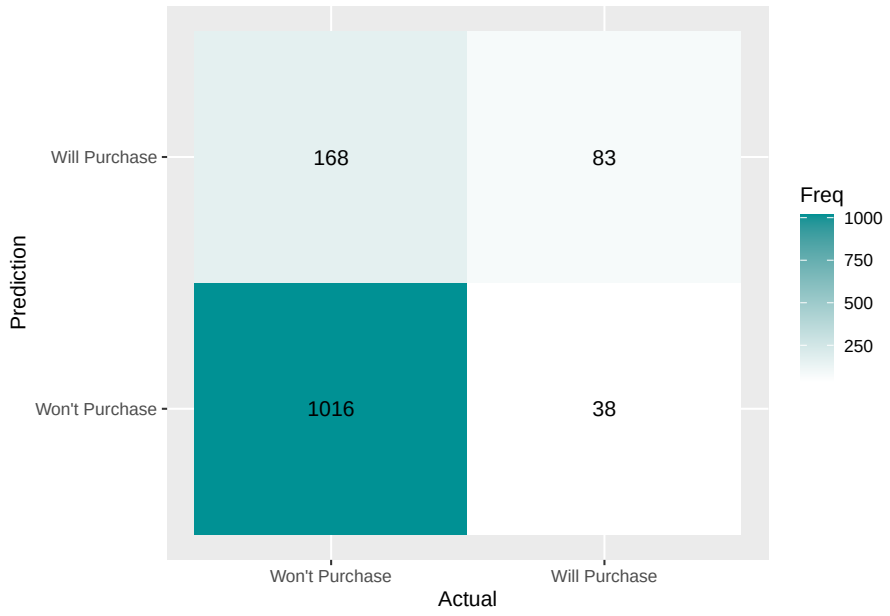



```
# Store the accuracy and the recall
svm_rbf_train_accuracy <- svm_rbf_train_result$accuracy
svm_rbf_train_recall <- svm_rbf_train_result$recall
```

The SVM model using the RBF kernel achieved an accuracy of 0.8731932 and a recall of 0.8789762 on the training data.

Let's check how it performs on the testing data.

```
# Predict the target variable using the test data
y_hat_svm_rbf_test <- predict(svm_fit_rbf, data_test,
    probability = TRUE)

# Evaluate the model performance on the test data
svm_rbf_test_result <- show_results(data_test$ProdTaken,
    y_hat_svm_rbf_test)
```



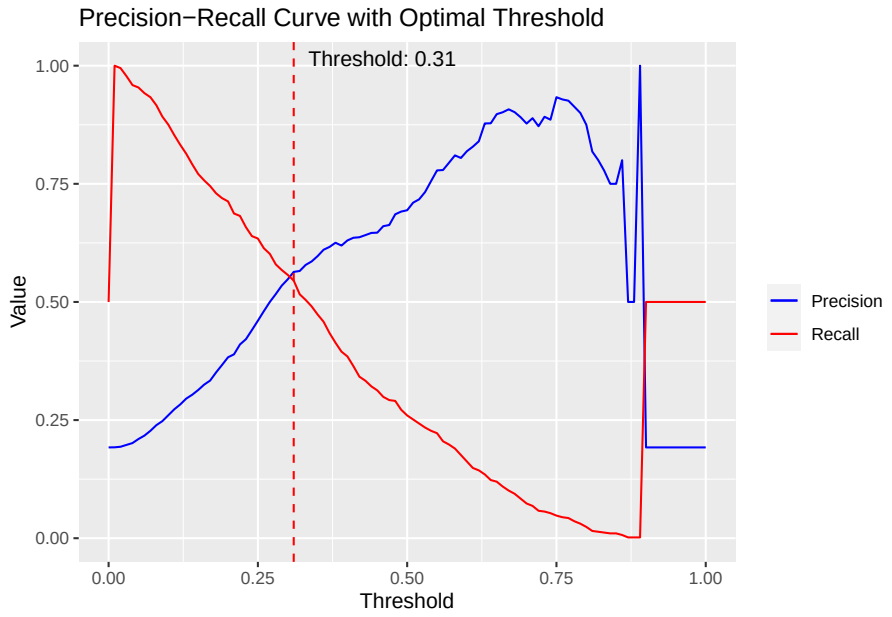

```
# Store the accuracy and the recall
svm_rbf_test_accuracy <- svm_rbf_test_result$accuracy
svm_rbf_test_recall <- svm_rbf_test_result$recall
```

The SVM model with the RBF kernel achieved an accuracy of 0.8421456 and a recall of 0.8581081 on the testing data.

Let's check if there is an optimal threshold that help maximize the recall.

```
svm_scores <- attr(y_hat_svm_linear_train, "probabilities")

y_scores_pos <- svm_scores[, 2]

# Create a data frame with the predicted scores
# and actual labels
pr_df <- data.frame(scores = y_scores_pos, labels = data_train$ProdTaken)

# Calculate precision and recall for different
# threshold values
precision_recall <- lapply(seq(0, 1, 0.01), function(threshold) {
    # Compute confusion matrix
    cm <- matrix(c(table(pr_df$labels, pr_df$scores >
        threshold)), ncol = 2, nrow = 2)

    # Calculate precision and recall
    precision <- cm[2, 2]/sum(cm[, 2])
    recall <- cm[2, 2]/sum(cm[2, ])

    # Return a data frame with threshold,
    # precision, and recall
    data.frame(threshold = threshold, precision = precision,
        recall = recall)
})
```

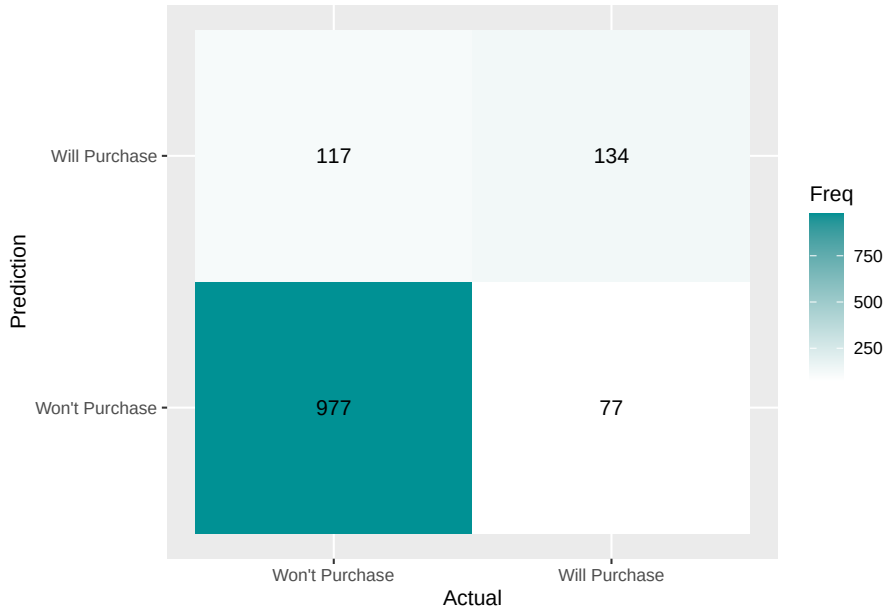
```
# Combine the results into a single data frame
precision_recall_df <- do.call(rbind, precision_recall)

# Calculate F1 score
precision_recall_df$F1 <- 2 * precision_recall_df$precision *
    precision_recall_df$recall/(precision_recall_df$precision +
    precision_recall_df$recall)

# Find the threshold value that maximizes the F1
# score to have precision and recall balanced
optimal_threshold <- precision_recall_df$threshold[which.max(precision_recall_df$F1)]

# Plot precision-recall curve
ggplot(data = precision_recall_df, aes(x = threshold)) +
    geom_line(aes(y = precision, color = "Precision")) +
    geom_line(aes(y = recall, color = "Recall")) +
    scale_x_continuous(name = "Threshold", limits = c(0,
        1)) + scale_y_continuous(name = "Value", limits = c(0,
    1)) + labs(title = "Precision-Recall Curve with Optimal Threshold") +
    geom_vline(xintercept = optimal_threshold, linetype = "dashed",
        color = "red") + scale_color_manual(name = "",
    values = c(Precision = "blue", Recall = "red")) +
    annotate("text", x = optimal_threshold, y = max(precision_recall_df$precision),
        label = paste("Threshold:", round(optimal_threshold,
            2)), vjust = 0, hjust = -0.1)
```



The optimal threshold is 0.31. Let's check if the recall improves when we use it.

```
y_hat_svm_rbf_opt <- predict(svm_fit_rbf, data_test,
    probability = TRUE)
# y_hat_svm_linear_opt <-
# y_hat_svm_linear_opt[,2] > optimal_threshold
```

```
svm_rbf_opt_result <- show_results(data_test$ProdTaken,
    factor(as.numeric(attr(y_hat_svm_rbf_opt, "probabilities")[,
        2] > optimal_threshold)))
```



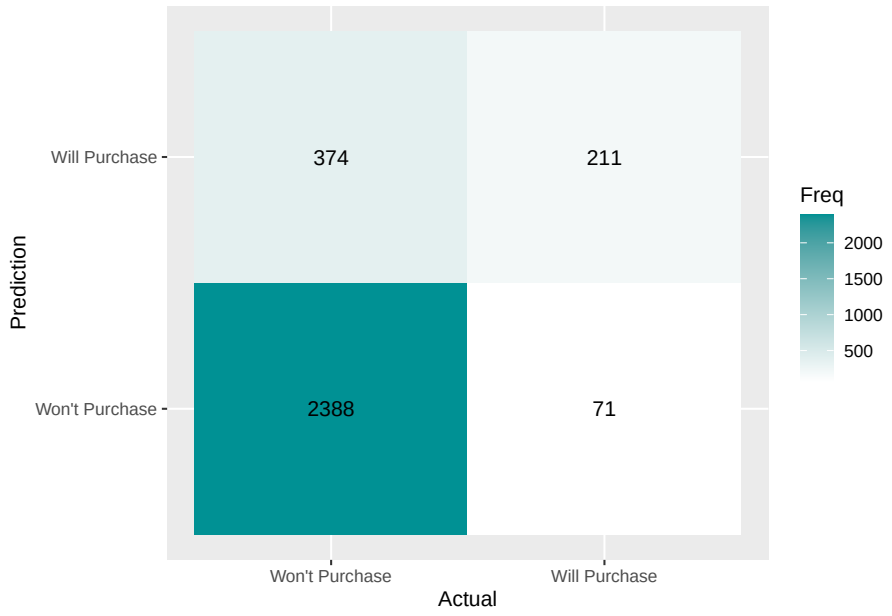

```
# Store the accuracy and the recall
svm_rbf_opt_accuracy <- svm_rbf_opt_result$accuracy
svm_rbf_opt_recall <- svm_rbf_opt_result$recall
```

Using the optimal threshold, the model achieved an accuracy of 0.851341 and a recall of 0.893053.

2.4.4 Decision Tree

The third model we are going to test is the decision tree. First, we will use the training data to build it.

```
set.seed(1)

# Train the decision tree model using the
# training data
model_dt <- rpart(ProdTaken ~ ., data = data_train,
    method = "class")

# Use the model to predict the training data
y_hat_dt_train <- predict(model_dt, newdata = data_train,
    type = "class")

# Evaluate the model performance on the training
# data
dt_train_result <- show_results(data_train$ProdTaken,
    y_hat_dt_train)
```




```
# Store the accuracy and the recall
dt_train_accuracy <- dt_train_result$accuracy
dt_train_recall <- dt_train_result$recall
```

The decision tree model achieved an accuracy of 0.8538108 and a recall of 0.8645909 on the training set. Let's check how it behaves with the testing data.

```
set.seed(1)

# Use the model to perform predictions on the
# test data
y_hat_dt_test <- predict(model_dt, newdata = data_test,
    type = "class")

# Evaluate the model performance on the test data
dt_test_result <- show_results(data_test$ProdTaken,
    y_hat_dt_test)
```

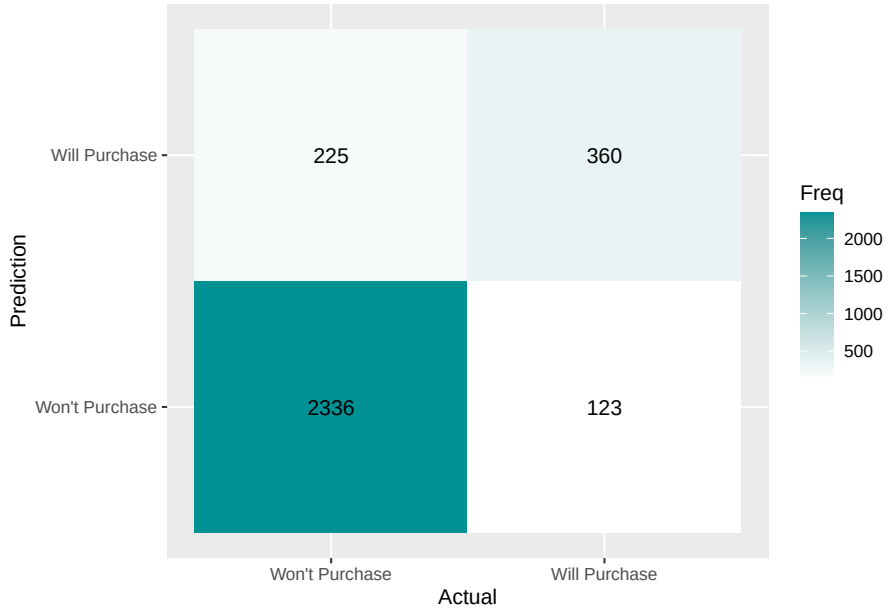

```
# Store the accuracy and the recall
dt_test_accuracy <- dt_test_result$accuracy
dt_test_recall <- dt_test_result$recall
```

The decision tree model achieved an accuracy of 0.8429119 and a recall of 0.8606627 on the testing set.

Let's perform some tuning and check if the recall improves.

```
# Grid of parameters to choose from
cp_values <- seq(0, 0.1, 0.01)
parameters <- expand.grid(cp = cp_values, maxdepth = seq(10,
    30, 10))

# Run the grid search, testing the model with
# different parameters
recall_values <- vector(mode = "numeric", length = nrow(parameters))

for (i in 1:nrow(parameters)) {
    model <- rpart(ProdTaken ~ ., data = data_train,
        method = "class", cp = parameters$cp[i], maxdepth = parameters$maxdepth[i])
    pred <- predict(model, newdata = data_train, type = "class")
    recall_values[i] <- caret::recall(Y_train$ProdTaken,
        pred)
}

# Get the best model
best_dt_model <- rpart(ProdTaken ~ ., data = data_train,
    method = "class", cp = parameters$cp[which.max(recall_values)],
    maxdepth = parameters$maxdepth[which.max(recall_values)])

# Perform predictions using the training data
y_hat_dt_train_opt <- predict(best_dt_model, newdata = data_train,
    type = "class")
```

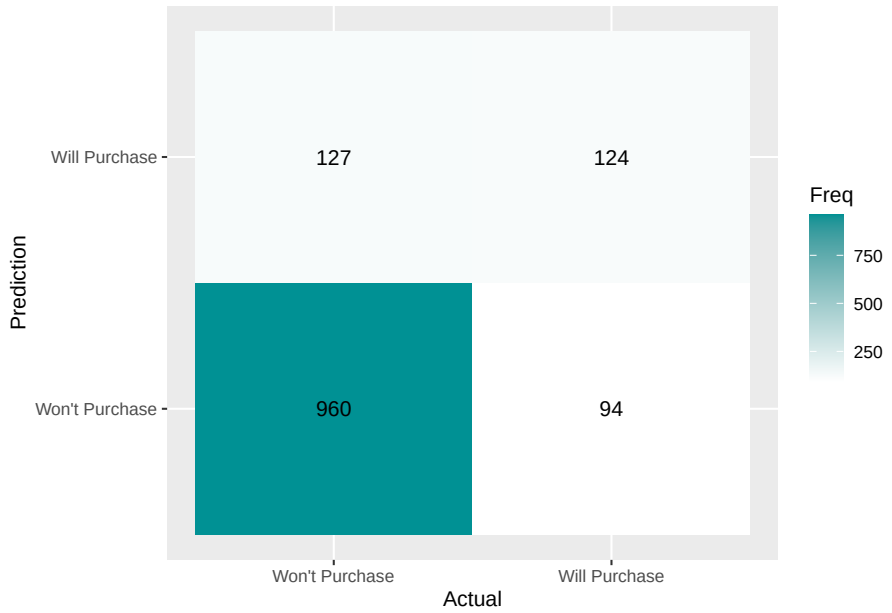
```
# Evaluate the model performance on the training
# data
dt_train_opt_result <- show_results(data_train$ProdTaken,
    y_hat_dt_train_opt)
```




```
# Store the accuracy and the recall
dt_train_opt_accuracy <- dt_train_opt_result$accuracy
dt_train_opt_recall <- dt_train_opt_result$recall
```

After tuning, the decision tree model achieved an accuracy of 0.8856767 and a recall of 0.9121437 on the training set. Let's check its performance with the testing set.

```
# Predict the target variable on the test data
y_hat_dt_test_opt <- predict(best_dt_model, newdata = data_test,
    type = "class")

# Check the performance of the model on the test
# data
dt_test_opt_result <- show_results(data_test$ProdTaken,
    y_hat_dt_test_opt)
```




```
# Store the accuracy and the recall
dt_test_opt_accuracy <- dt_test_opt_result$accuracy
dt_test_opt_recall <- dt_test_opt_result$recall
```

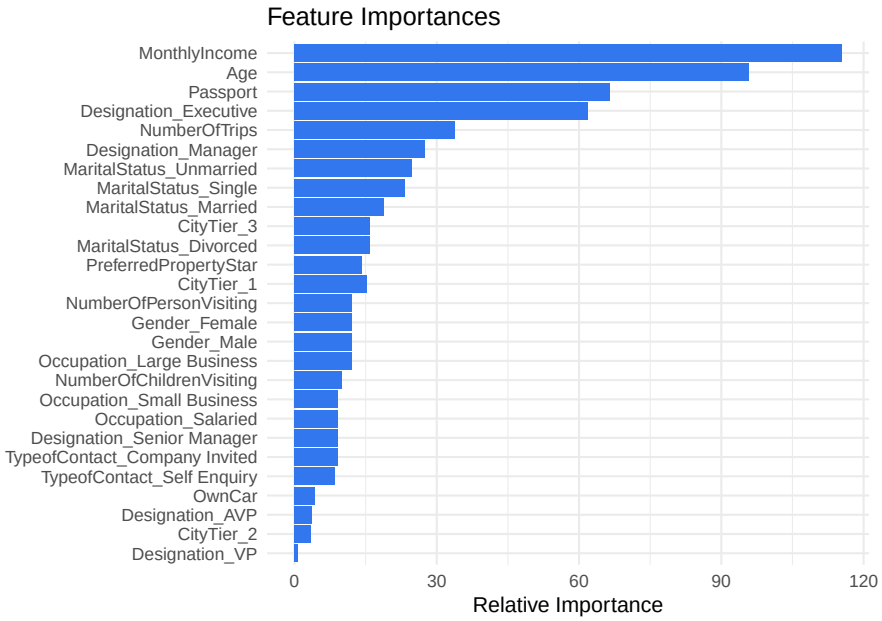
After tuning, the decision tree model achieved an accuracy of 0.8856767 and a recall of 0.9121437 on the training set.

Using the decision tree model, it is possible to determine how much each feature contributes to predict the target variable. The function below plots a chart with that information.

```
# Show the features ordered by their importance
plot_feature_importance <- function(importances, features) {
    # Create dataframe for plotting
    importances_df <- data.frame(feature = features,
        importance = importances, stringsAsFactors = FALSE)

    # Order by importance in descending order
    importances_df <- importances_df %>%
        arrange(desc(importance))

    # Create plot
    importances_df %>%
        ggplot(aes(x = importance, y = reorder(feature,
            sort(importance), decreasing = TRUE))) +
        geom_bar(stat = "identity", fill = "#3377EE") +
        xlab("Relative Importance") + ylab("") + ggtitle("Feature Importances") +
        theme_minimal()
}
```

Using the function above, we can obtain the chart that shows the features ordered by their importance to predict if a customer will purchase a package or not.

```
# Show a list of features ordered by their
# importance
plot_feature_importance(best_dt_model$variable.importance,
    names(best_dt_model$variable.importance))
```



From the chart above, we can see that the most important variables are Age, MonthlyIncome, and Designation_Executive.

2.4.5 Random Forest

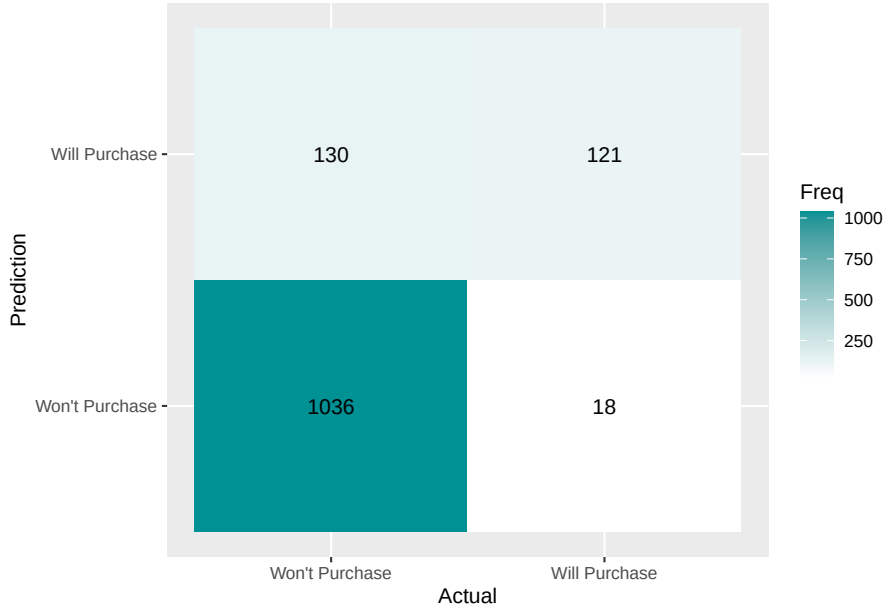
The last model we are going to test is the random forest. First, we will train it using the training data.

```
# Train the random forest model
model_rf <- randomForest(X_train, Y_train$ProdTaken,
    importance = TRUE, ntree = 500, mtry = floor(sqrt(ncol(X_train))))

# Predict the training data
y_hat_rf_train <- predict(model_rf, newdata = data_train)

# Evaluate the model performance on the training
# data
rf_train_result <- show_results(data_train$ProdTaken,
    y_hat_rf_train)
```


```
# Store the accuracy and the recall
rf_train_accuracy <- rf_train_result$accuracy
rf_train_recall <- rf_train_result$recall
```

The random forest model achieved an accuracy of 0.9983574 and a recall of 0.9979708 on the training data.

```
# Predict the test data
y_hat_rf_test <- predict(model_rf, newdata = data_test)

# Evaluate the model performance on the test data
rf_test_result <- show_results(data_test$ProdTaken,
    y_hat_rf_test)
```

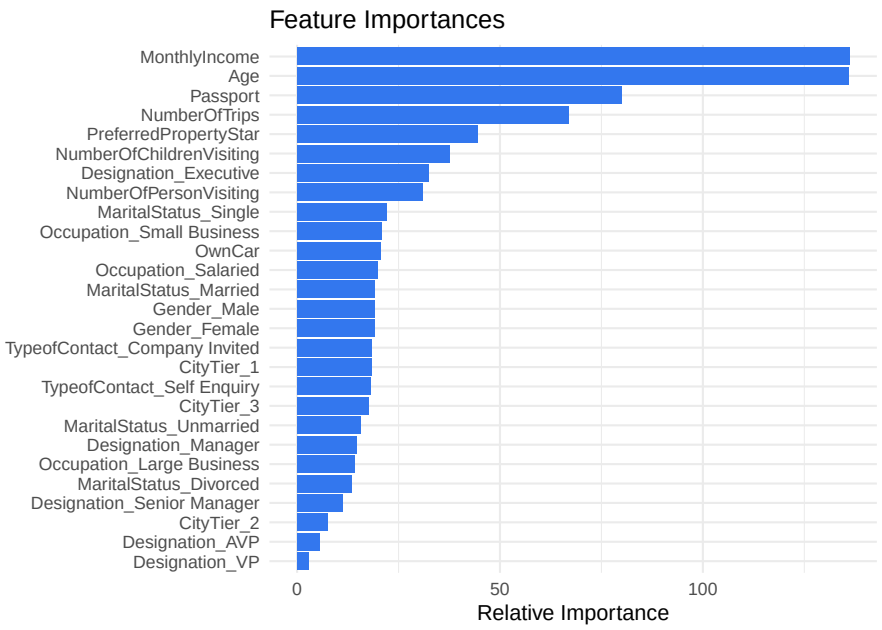


```
# Store the accuracy and the recall
rf_test_accuracy <- rf_test_result$accuracy
rf_test_recall <- rf_test_result$recall
```

The random forest model achieved an accuracy of 0.88659 and a recall of 0.8885077 on the test data.

We can also obtain a list of the most important features to predict if a customer will purchase a package or not:

```
# Show a list of features ordered by their
# importance
plot_feature_importance(model_rf$importance[, "MeanDecreaseGini"],
    rownames(model_rf$importance))
```



According to the chart, the most important features are:

- **Age:** Customers with age 25-50 are most likely to accept the tour package.
- **MonthlyIncome:** If the customer income is high, the chances of purchasing a package is higher.
- **Passport:** Customers that go abroad tend to purchase tour packages.
- **NumberOfTrips:** Customers that travel more are most likely to buy a package.

3 Results

The table below compares the performance of all models that have been created:

```
# Create a dataframe to compare the performance
# of the models
comparison_df <- data.frame(Model = c("Logistic Regression",
    "SVM - Linear Kernel", "SVM - RBF Kernel", "Decision Tree",
    "Random Forest"), Recall = c(lg_test_recall, svm_linear_opt_recall,
    svm_rbf_opt_recall, dt_test_opt_recall, rf_test_recall),
    Accuracy = c(lg_test_accuracy, svm_linear_opt_accuracy,
        svm_rbf_opt_accuracy, dt_test_opt_accuracy,
        rf_test_accuracy))

# Format recall and accuracy to show as
# percentages
comparison_df <- comparison_df %>%
    mutate(Recall = lapply(Recall, function(x) sprintf("%.2f%%",
        x * 100)), Accuracy = lapply(Accuracy, function(x) sprintf("%.2f%%",
        x * 100)))

# Show the comparison table
comparison_df %>%
    arrange(desc(unlist(Recall))) %>%
    select(Model, Recall, Accuracy) %>%
    kable(booktabs = TRUE, escape = FALSE, format = "pandoc") %>%
    kable_styling(full_width = FALSE) %>%
    column_spec(2:3) %>%
    row_spec(0, bold = TRUE)
```

Model	Recall	Accuracy
SVM - Linear Kernel	89.45%	83.45%
SVM - RBF Kernel	89.31%	85.13%
Random Forest	88.85%	88.66%
Decision Tree	88.32%	83.07%
Logistic Regression	84.94%	83.22%

The SVM models outperformed the other models. The model that uses the linear kernel has a little higher recall but with less accuracy. On the other hand, the model with the RBF kernel has a better accuracy with almost the same recall.

4 Conclusion

After this analysis, we can make some recommendations to this tourism company:

- Create and customize more international packages to attract the customers that own a passport.
- Expand the marketing initiatives on tier 2 cities since there are very few customers from these cities.
- Create short-term packages and include luxuries to target those customers with high income and high positions, such as VPs and AVPs.
- Pay attention to follow-ups, as they increase the chance of a customer purchase a package.
- Choose the package to be pitched to a customer based on their monthly income. Pitching an expensive package to a customer with low income is a sure way to lose a sale.
- Customize packages or offer discounts to couples, families, and customers above 30 years of age. Young and single people are more likely to buy the packages and the customer base could be expanded by attracting those other groups.