Project 1: Customer Base Analysis

Applications of Data Sciene (1340)

Baldauf Moritz, Heinze Valentina, Kiseleva Nataliia

Invalid Date

1. Project Overview

In this project, our group operates as analysts from the marketing department. The project focuses on analyzing the customer base to uncover behavioral patterns and insights that inform effective marketing strategies. By leveraging data analysis, the goal is to identify segments and patterns within the customer base and understand their preferences, engagement levels, and potential areas for improvement marketing strategy.

Using data-driven techniques to segment, cluster, classify customers based on demographics, usage patterns, and subscription status helps in tailoring marketing efforts to specific groups. We will investigate how different factors influence customer engagement with the service.

A deeper understanding of customer preferences and behaviors, leading to more effective marketing strategies. Based on the insights gained can be created personalized marketing campaigns aimed at different segments. This could include promotional offers for premium subscriptions, targeted content recommendations, and engagement strategies for low-activity users.

Overall, this project not only enhances understanding of the customer base but also lays the groundwork for future marketing initiatives that are data-driven and customer-focused.

Preparation

```
# message = FALSE becasue we want no output when loading packages
library(ggplot2)
library(ggcorrplot)
library(ggpubr)
library(rpart)
library(partykit)
library(dplyr)
library(gridExtra)
library(dbscan)
library(partykit)
library(caret)
library(e1071)
library(ranger)
library(plotly)
library(corrplot)
library(tidyr)
library(scatterplot3d)
```

```
set.seed(1234)
```

```
load("Group1_streaming_ds.rda")
```

2. Data Description

Structure

```
str(streaming_ds)  # display the structure of an R object
```

```
'data.frame':   8525 obs. of  8 variables:
 $ few hrs    : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ stopped    : num  0 0 0 0 0 0 0 0 0 0 ...
 $ age        : num  38 33 34 30 32 23 28 29 37 28 ...
 $ income     : num  4.52 2.42 4.5 2.54 9.79 ...
 $ device     : Factor w/ 2 levels "android","ios": 2 1 2 1 2 1 1 2 2 1 ...
 $ living area: Factor w/ 2 levels "urban","rest": 1 1 1 1 1 1 1 1 1 1 ...
 $ premium    : num  1 1 1 1 1 1 1 1 1 1 ...
 $ months     : num  12 13 5 7 5 1 5 3 6 18 ...
```

Our data set consists of 8 columns and 8525 entries (customers).

```
head(streaming_ds)  # display first 6 rows
```

```
  few hrs stopped age income  device living area premium months
1   FALSE       0  38 4.5200     ios       urban       1     12
2   FALSE       0  33 2.4200 android       urban       1     13
3   FALSE       0  34 4.5000     ios       urban       1      5
4   FALSE       0  30 2.5400 android       urban       1      7
5   FALSE       0  32 9.7867     ios       urban       1      5
6   FALSE       0  23 2.5000 android       urban       1      1
```

```
streaming_ds["few hrs"] <- as.factor(streaming_ds$`few hrs`) # change column from boolean
```

Summary

```
summary(streaming_ds)
```

```
 few hrs        stopped             age             income          device
FALSE:6413   Min.   : 0.0000   Min.   : 0.00   Min.   : 0.21   android:4744
TRUE :2112   1st Qu.: 0.0000   1st Qu.:25.00   1st Qu.: 2.25   ios    :3781
             Median : 0.0000   Median :31.00   Median : 2.95
             Mean   : 0.4712   Mean   :33.23   Mean   : 3.37
             3rd Qu.: 0.0000   3rd Qu.:40.00   3rd Qu.: 4.00
             Max.   :14.0000   Max.   :84.00   Max.   :13.50
living area     premium          months
urban:7921   Min.   :0.0000   Min.   : 0.000
rest : 604   1st Qu.:1.0000   1st Qu.: 2.000
             Median :1.0000   Median : 6.000
             Mean   :0.8141   Mean   : 7.013
             3rd Qu.:1.0000   3rd Qu.:11.000
             Max.   :1.0000   Max.   :46.000
```

Here we summarized maximum and minimum values in each column, mean and median values, and for the factor variables were counted the number of their value. We can see, for example, that the average age of our customers is 33 years. The average income pro year is 33700€

Missing values

```r
missing_values <- colSums(is.na(streaming_ds))
print(missing_values)
```

```
  few hrs     stopped         age      income      device living area
        0           0           0           0           0           0
  premium      months
        0           0
```

Our data set does not contain missing values.

Relevant variables that would help to segment/cluster/classify the customer

Here is a description of the variables and their analysis.

Few hrs (shows if customer has used the service for a few hours) highlights customers who have low engagement with the service, which is valuable for segmentation. It can help separate users with minimal usage from others and may identify groups at risk of churning or disengagement.

Age can provide insights into customer preferences and behaviors, as different age groups may have distinct content preferences or engagement levels. It's often relevant for segmentation to create age-based groups who may respond differently to marketing or content offerings.

Stopped (number of streams that the customer has stopped) indicates dissatisfaction or a low-quality experience, as stopping streams can be associated with poor content fit or technical issues. For segmentation, high "stopped" counts could indicate potential rejectionists, while low counts might signal satisfied users.

Income (in 10,000€) often correlates with spending power and can be a strong indicator of a user's likelihood to subscribe to premium services. Segmenting users based on income may reveal groups more inclined toward premium features, which is useful for targeting. Such variable can be best used in combination with another indicator such as age.

Device can help segment users based on the platform they use, which is important because device type often influences user experience, app compatibility, and even content type preferences.

Living Area can impact content preferences due to cultural or regional interests. Segmenting by living area can help in tailoring region-specific recommendations or marketing efforts (but in our case we don't have enough data: only 2 types: urban or other).

Premium subscribers often have different usage patterns and expectations than non-premium users. This variable can serve as both a segmentation basis and a target for classification models, helping distinguish high-value users from standard users.

Months of subscription duration reflects loyalty. Long-term subscribers may have different needs and engagement levels than newer users, making this an essential variable for segmentation or prediction.

Variables like Few hrs, Stopped, Premium, and Months directly relate to user engagement and service interaction, while Age, Income, Device, and Living Area capture demographic and environmental factors that can impact service usage patterns.

Marketing perspective of relevant variables

From marketing perspective clustering is based on income and months.

- Income can serve as a proxy for purchasing power, helping to identify high-value customers who might be more willing to spend on premium services.
- Months subscribed reflects customer loyalty and engagement. Long-term subscribers often have higher retention value, while newer customers may need additional incentives to continue their subscriptions.

Clusters based on income and months can reveal actionable insights, such as high-growth segments and potential risks, enabling the marketing team to target specific groups with tailored campaigns to increase customer satisfaction and retention.

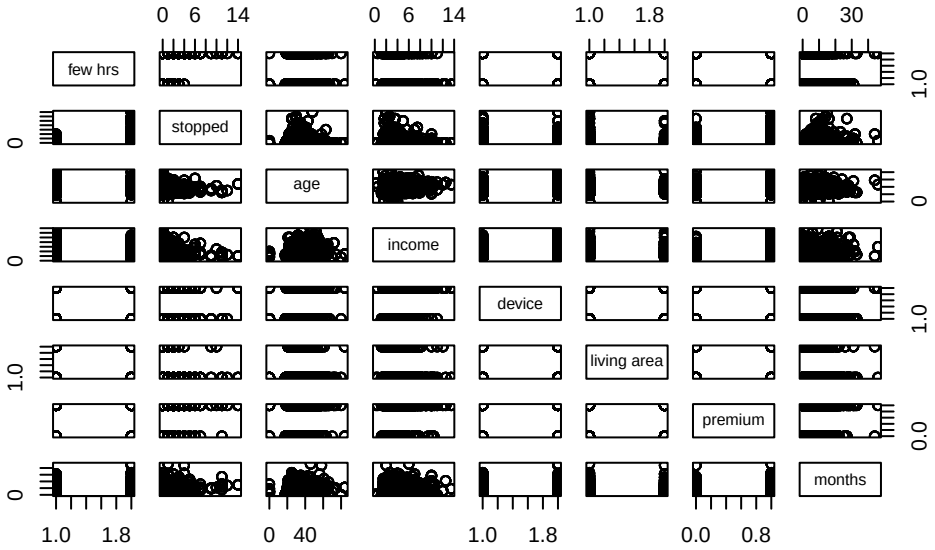
Also distribution of other variables in formed clusters enables the creation of personalized marketing strategies.

3. Exploratory Data Analysis

Firstly, we explore relations between variables and make a correlation matrix, that shows us no strong correlation between any of numeric variables, however we have a few combinations which might be intersting to investigate further: Income~Age; Months~Age; Months~Stopped.

Exploring relationships between variables

```
pairs(streaming_ds)
```



Correlation matrix

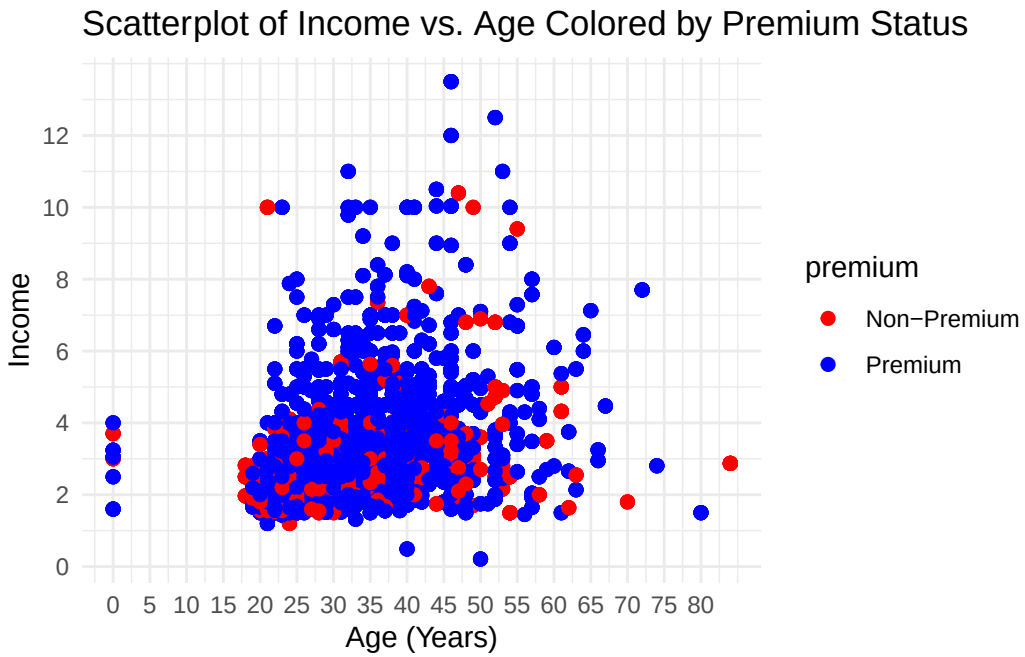
```
streaming_ds$premium <- as.numeric(as.character(streaming_ds$premium))
numeric_vars <- streaming_ds[sapply(streaming_ds, is.numeric)]
correlation_matrix <- cor(numeric_vars, use = "complete.obs")
corrplot(correlation_matrix,
         method = "shade",
         type = "full",
         tl.col = "black",
         tl.srt = 45,
```

```
          addCoef.col = "black",
          number.cex = 0.9,
          diag = FALSE)
```

Now let us visualize the main relevant variables and their relationship. Income and age have the strongest correlation.

Scatterplot of Income vs. Age Colored by Premium Status

```
streaming_ds$premium <- as.factor(streaming_ds$premium)

ggplot(streaming_ds, aes(x = age, y = income, color = premium)) +
  geom_point(size = 2) +
  labs(title = "Scatterplot of Income vs. Age Colored by Premium Status",
       x = "Age (Years)",        y = "Income") +
  scale_color_manual(values = c("0" = "red", "1" = "blue"),
                     labels = c("0" = "Non-Premium", "1" = "Premium")) +
  scale_x_continuous(breaks = seq(0, max(streaming_ds$age, na.rm = TRUE), by = 5)) +
  scale_y_continuous(breaks = seq(0, max(streaming_ds$income, na.rm = TRUE), by = 2)) +
  theme_minimal()
```



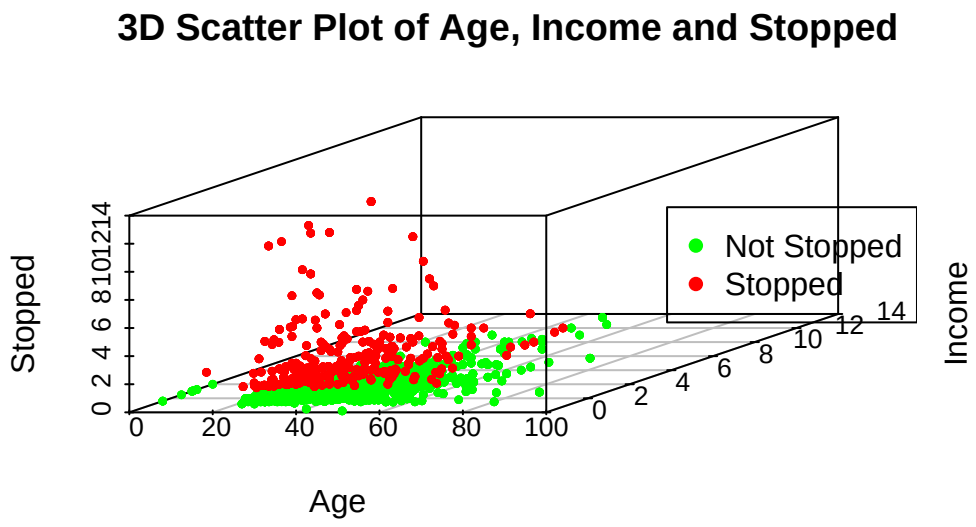
Create a 3D scatter plot

Because our final report is a PDF we can not display the HTML based interactive Visualization. However, we left the code because we can use it in our PowerPoint presentation.

```
# plot_ly(data = streaming_ds, x = ~age, y = ~income,
# z = ~stopped,color = ~stopped,
#         colors = c("green", "red"),
#         marker = list(size = 5),
#         type = 'scatter3d',
#         mode = 'markers') %>% layout(title = "3D Scatter Plot of Age,
#                                      Income and Stopped",
#                                      scene = list(xaxis = list(title = "Age"),
# yaxis = list(title = "Income"), zaxis = list(title = "Stopped")))
```

3D Plot for PDF Report

```
# Create color vector based on stopped values
colors <- ifelse(streaming_ds$stopped == 0, "green", "red")

# Create the 3D scatter plot
scatterplot3d(x = streaming_ds$age,
              y = streaming_ds$income,
              z = streaming_ds$stopped,
              color = colors,
              pch = 16,  # Solid circle points
              cex.symbols = 0.6,  # Point size
              main = "3D Scatter Plot of Age, Income and Stopped",
              xlab = "Age",
              ylab = "Income",
              zlab = "Stopped",
              angle = 45,  # Rotation angle
              box = TRUE)  # Add box around plot

# Add legend
legend("right",
       legend = c("Not Stopped", "Stopped"),
       col = c("green", "red"),
       pch = 16)
```



4. Identifying the core customer group for our marketing campaign

In order to grow the number of premium users we want to run a targeted marketing campaign focusing on promising customers which are not yet premium users. To identify this customer segment we perform clustering on our customer base data set.

Preparing the data

In order to perform the clustering we need to take a few preparation steps on the data. The main points are turning columns that are factors into numeric values. This is important because we need to scale the data before beginning the clustering process and the scaling only works on numeric values.

Scaling is important because when clustering we work with distances between points. Since we have multiple scales for our data we need to transform them into on scale to not bias data points in a higher scale. For this R has a base function implemented, which uses a Z-score normailzation:

$$\text{scaled value} = \frac{\text{original value - sample mean}}{\text{sample standard deviation}}$$

The scaling function returns a scaled matrix which we save as a data frame for further processing.

We perform this process two times because we want on data frame that contains all customers and one that only contains the ones which are not yet premium.

```
load("Group1_streaming_ds.rda")

# Data frame with all customers
clustering_data  <- streaming_ds %>%
  # can also be reomved sinc we select all columns
  select(stopped, age, income, premium, device,
         `living area`, months, `few hrs`) %>%
  # Save few hrs as numeric; also  rename col
  mutate(few_hrs = as.numeric(`few hrs`)) %>%
  mutate(device = as.numeric(device)) %>% # Save device as numeric
  # Save living area as numeric
  mutate(living_area = as.numeric(`living area`)) %>%
  select(-`few hrs`) %>% # remove old columns
  select(-`living area`) %>% # remove old columns
  scale() %>% # scale all columns -> returns a matrix
  as.data.frame() # turn matrix into a data frame


# Data frame for non-premium customers
clustering_data_no_premium <- streaming_ds %>%
  filter(premium == 0) %>% # Filter for premium customers
  # can also be reomved sinc we select all columns
  select(stopped, age, income, premium, device,
         `living area`, months, `few hrs`) %>%
  mutate(few_hrs = as.numeric(`few hrs`)) %>% #Save few hrs as numeric
  mutate(device = as.numeric(device)) %>% # Save device as numeric
  # Save living area as numeric
  mutate(living_area = as.numeric(`living area`)) %>%
  select(-`few hrs`) %>% # remove old columns
  select(-`living area`) %>% # remove old columns
  # drop because it is the same value for each entry
  # -> when scaling this would lead to NaN
  select(-premium) %>%
  scale() %>% # scale all columns -> returns a matrix
  as.data.frame() # turn matrix into a data frame
```

```r
## Check the resulting strucutre of both Data frames
str(clustering_data)
```

```
'data.frame':   8525 obs. of  8 variables:
 $ stopped    : num  -0.335 -0.335 -0.335 -0.335 -0.335 ...
 $ age        : num  0.4704 -0.0226 0.076 -0.3183 -0.1211 ...
 $ income     : num  0.683 -0.564 0.671 -0.493 3.81 ...
 $ premium    : num  0.478 0.478 0.478 0.478 0.478 ...
 $ device     : num  1.12 -0.893 1.12 -0.893 1.12 ...
 $ months     : num  0.79427 0.95355 -0.32064 -0.00209 -0.32064 ...
 $ few_hrs    : num  -0.574 -0.574 -0.574 -0.574 -0.574 ...
 $ living_area: num  -0.276 -0.276 -0.276 -0.276 -0.276 ...
```

```r
str(clustering_data_no_premium)
```

```
'data.frame':   1585 obs. of  7 variables:
 $ stopped    : num  -0.355 -0.355 0.445 -0.355 -0.355 ...
 $ age        : num  -0.262 0.171 -0.695 -0.955 -0.955 ...
 $ income     : num  -0.883 -0.758 -0.781 0.609 -0.673 ...
 $ device     : num  1.303 1.303 -0.767 -0.767 -0.767 ...
 $ months     : num  1.144 -0.054 -0.567 -0.738 -0.054 ...
 $ few_hrs    : num  1.399 -0.715 1.399 1.399 -0.715 ...
 $ living_area: num  -0.276 -0.276 -0.276 3.625 -0.276 ...
```

Starting with clustering

K-means

The first clustering algorithm we try is K-means. This is a rather simple algorithm which calculate the distance between the points and a centroid and groups them together depending on distance measurements. For the algorithm it is important to define the number of centroids before performing the clustering. This can be done by multiple approaches. We have decided to use the elbow method.

Getting number of clusters using the elbow method

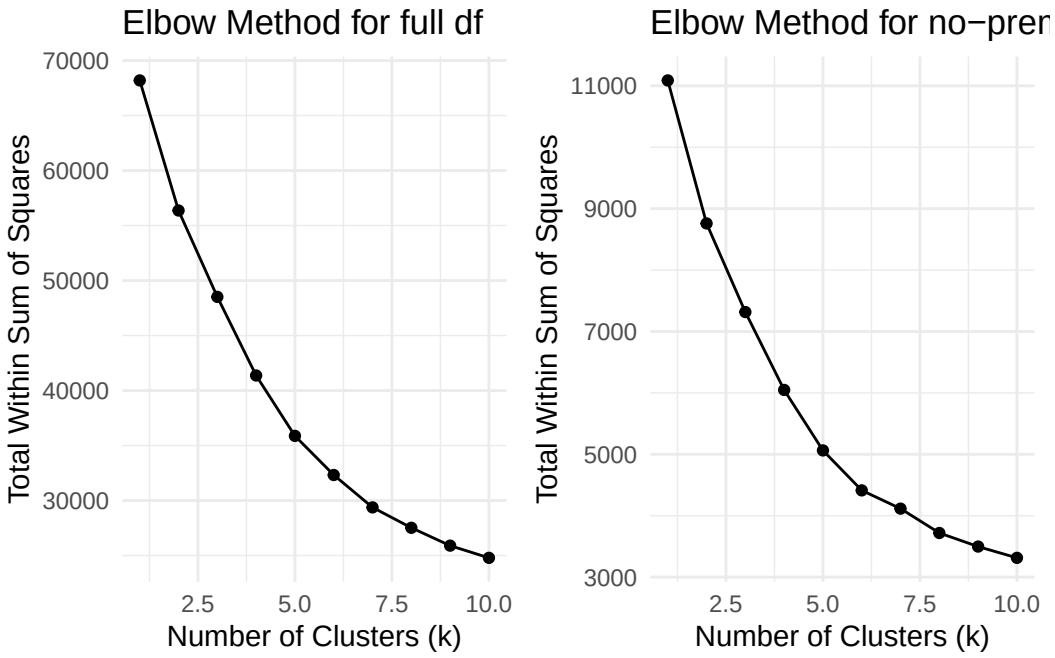
In order to determine the best number of clusters, we use the function from lecture two. This function takes an maximal cluster value as well as the data which we want to cluster as an

input. It then performs an kmeans for each number of clusters and saves the within sum of squares error for each number of clusters.

```
# Function for calculating wss
elbow_method <- function(n_clusters, data) {
  wss <- numeric(n_clusters) # making sure number of clusters is numeric

  for (i in 1:n_clusters) { # ittering through range of clusters
    # calc kmeans for each number of clusters
    km <- kmeans(data, centers = i, nstart = 20)
    wss[i] <- km$tot.withinss # saving wss from the kmeans object
  }
  return(wss)
}

# Caling the function for both data frames
wss_values_full <- elbow_method(10, clustering_data)
wss_values_no_premium <- elbow_method(10, clustering_data_no_premium)
```

```
# Turn wss values into df for plotting
# Repeat this for both data frames
df_data_full <- data.frame(k = 1:10, wss = wss_values_full)
df_data_no_premium <- data.frame(k = 1:10, wss = wss_values_no_premium)


# Plotting resulting wss and number of centroides
plot_data_full <- ggplot(df_data_full, aes(x = k, y = wss)) +
  geom_line() +
  geom_point() +
  labs(title = "Elbow Method for full df",
       x = "Number of Clusters (k)",
       y = "Total Within Sum of Squares") +
  theme_minimal()

plot_data_no_premium <- ggplot(df_data_no_premium, aes(x = k, y = wss)) +
  geom_line() +
  geom_point() +
  labs(title = "Elbow Method for no-premium df",
       x = "Number of Clusters (k)",
       y = "Total Within Sum of Squares") +
  theme_minimal()
```

```r
# Display both plots at the same time
grid.arrange(plot_data_full, plot_data_no_premium, ncol = 2)
```



From the Elbow Plots we can see that for the full data set 5 clusters are appropriate, for the no-premium data set we decide to use 6 clusters.

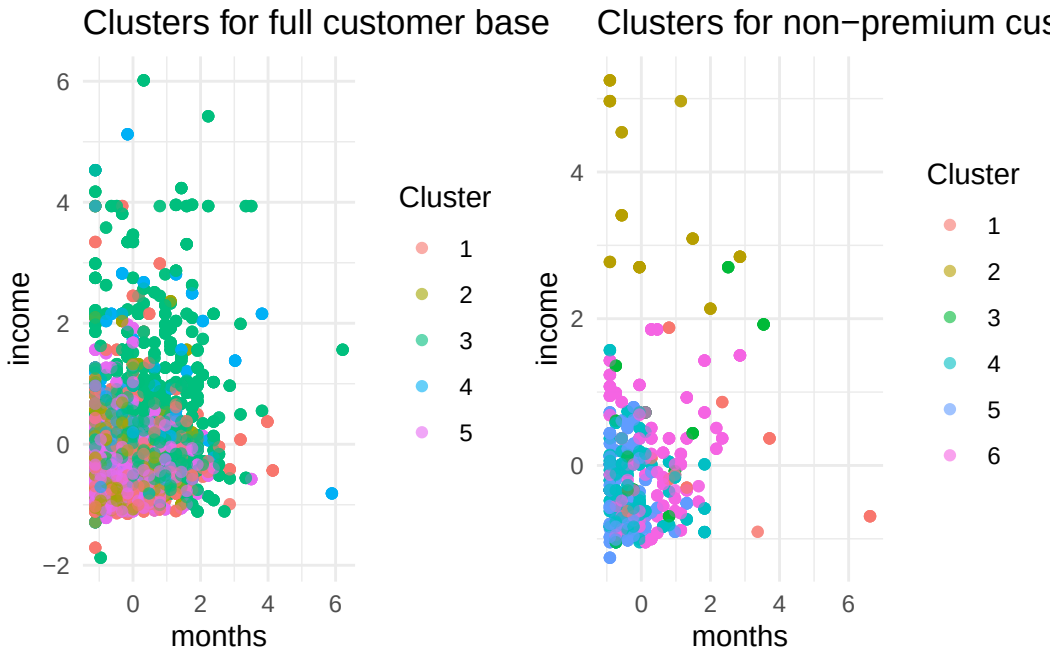
We made this decision by looking at the slope of the curve. The goal of the elbow method is to find the number of cluster where the curves slope is starting to flatten of, meaning increasing the number of clusters will only lead to diminishing returns.

Performing the k-means clustering

```r
# Running the kmeans with the predetermined number of clusters
kmeans_full <- kmeans(clustering_data, centers = 5, nstart = 10)
kmeans_no_premium <- kmeans(clustering_data_no_premium, centers = 6, nstart = 10)

# Add the resulting clusters to the data frame
clustering_data$Cluster <- as.factor(kmeans_full$cluster)
clustering_data_no_premium$Cluster <- as.factor(kmeans_no_premium$cluster)
```

Visualizing k-means clustering

```
plot_full_df <- ggplot(clustering_data, aes(x = months, y =income , color = Cluster)) +
    geom_point(alpha = 0.6) +
    labs(title = "Clusters for full customer base") +
    theme_minimal()


plot_no_premium_df <- ggplot(clustering_data_no_premium,
                             aes(x =  months, y =income, color = Cluster)) +
    geom_point(alpha = 0.6) +
    labs(title = "Clusters for non-premium customers") +
    theme_minimal()

grid.arrange(plot_full_df, plot_no_premium_df, ncol = 2)
```



From the visualized cluster we see that it is very hard to identify any clusters for the full customer base. Because of this we decided to stick with only the non-premium customers for the further analysis.

In order to better understand the non-premium customer base we will take a closer look at the resulting clusters by visualizing the separately. This helps us the see better if there are

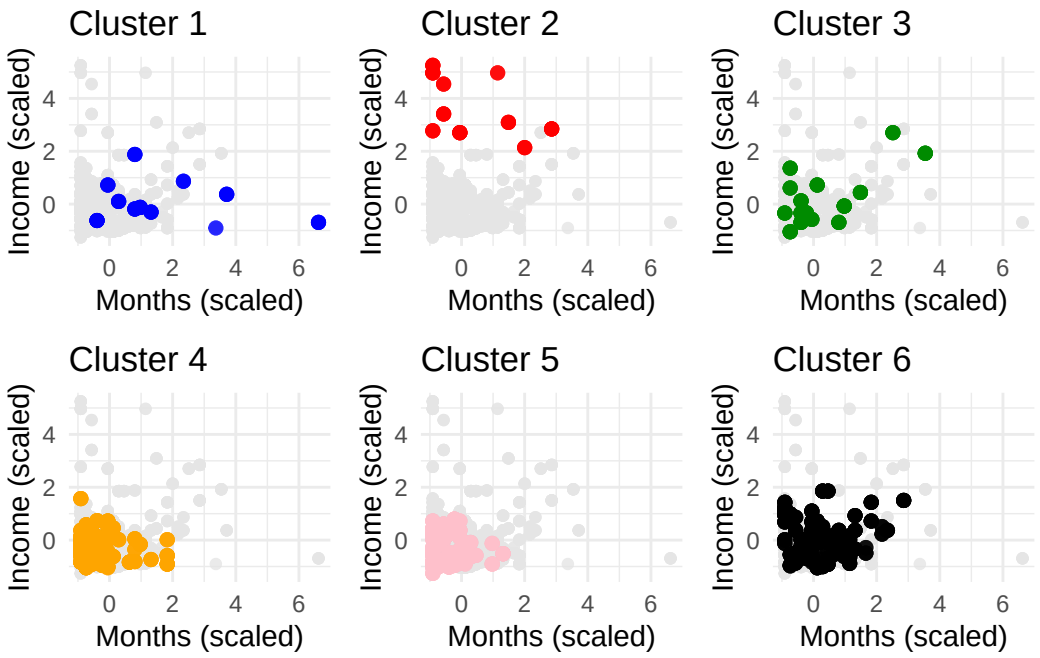
any meaningful cluster combinations we could use for a further marketing campaign.

Separate visualizing of resulting clusters

```
# The code is quite repreitive so please
# refere to the comments for the plot of
# cluster 1 if any questions should arise

# Cluster 1
c1 <- ggplot(clustering_data_no_premium, aes(x = months, y = income)) +
  # Add background points in grey
  geom_point(color = "grey90", alpha = 0.5) +
  # Add cluster points in color by creating a subset for only the first cluster
  geom_point(data = subset(clustering_data_no_premium, Cluster == "1"),
             aes(color = Cluster), size = 2, alpha = 0.6) +
  # Specifiy that we want the color blue
  scale_color_manual(values = "blue") +
  theme_minimal() +
  labs(title = "Cluster 1",
       x = "Months (scaled)",
       y = "Income (scaled)") +
  # Deactive legnend to save space
  theme(legend.position = "none")

# Cluster 2
c2 <- ggplot(clustering_data_no_premium, aes(x= months, y = income)) +
  geom_point(color = "grey90", alpha = 0.5) +
  geom_point(data = subset(clustering_data_no_premium, Cluster == "2"),
             aes(color = Cluster), size = 2, alpha = 0.6) +
  scale_color_manual(values = "red") +
  theme_minimal() +
  labs(title = "Cluster 2",
       x = "Months (scaled)",
       y = "Income (scaled)") +
  theme(legend.position = "none")

# Cluster 3
c3 <- ggplot(clustering_data_no_premium, aes(x = months, y = income)) +
  geom_point(color = "grey90", alpha = 0.5) +
  geom_point(data = subset(clustering_data_no_premium, Cluster == "3"),
             aes(color = Cluster), size = 2, alpha = 0.6) +
```

```
  scale_color_manual(values = "green4") +
  theme_minimal() +
  labs(title = "Cluster 3",
       x = "Months (scaled)",
       y = "Income (scaled)") +
  theme(legend.position = "none")

# Cluster 4
c4 <- ggplot(clustering_data_no_premium, aes(x = months, y = income)) +
  geom_point(color = "grey90", alpha = 0.5) +
  geom_point(data = subset(clustering_data_no_premium, Cluster == "4"),
             aes(color = Cluster), size = 2, alpha = 0.6) +
  scale_color_manual(values = "orange") +
  theme_minimal() +
  labs(title = "Cluster 4",
       x = "Months (scaled)",
       y = "Income (scaled)") +
  theme(legend.position = "none")

# Cluster 5
c5 <- ggplot(clustering_data_no_premium, aes(x = months, y = income)) +
  geom_point(color = "grey90", alpha = 0.5) +
  geom_point(data = subset(clustering_data_no_premium, Cluster == "5"),
             aes(color = Cluster), size = 2, alpha = 0.6) +
  scale_color_manual(values = "pink") +
  theme_minimal() +
  labs(title = "Cluster 5",
       x = "Months (scaled)",
       y = "Income (scaled)") +
  theme(legend.position = "none")

# Cluster 6
c6 <- ggplot(clustering_data_no_premium, aes(x = months, y = income)) +
  geom_point(color = "grey90", alpha = 0.5) +
  geom_point(data = subset(clustering_data_no_premium, Cluster == "6"),
             aes(color = Cluster), size = 2, alpha = 0.6) +
  scale_color_manual(values = "black") +
  theme_minimal() +
  labs(title = "Cluster 6",
       x = "Months (scaled)",
       y = "Income (scaled)") +
```

```
  theme(legend.position = "none")

# Arrange plots
grid.arrange(c1, c2, c3, c4, c5, c6, nrow = 2, ncol=3)
```



When looking at the resulting clusters we see that no clear cutoff for the clusters exists. For example cluster 4 & 5 & 6 seem very similar. This is not optimal as our goal is to identify unique customer groups that can be targeted by tailored marketing campaigns.

Further visualizations for our target customer segment

In order to investigate the resulting clusters deeper we visualize the Device and Living Area distribution in each clusters. This is especial interesting because depending on these factors our marketing strategy needs to be adapted. For example, should we run more adds for IOS in urban areas or Android in order to address the target customer group?

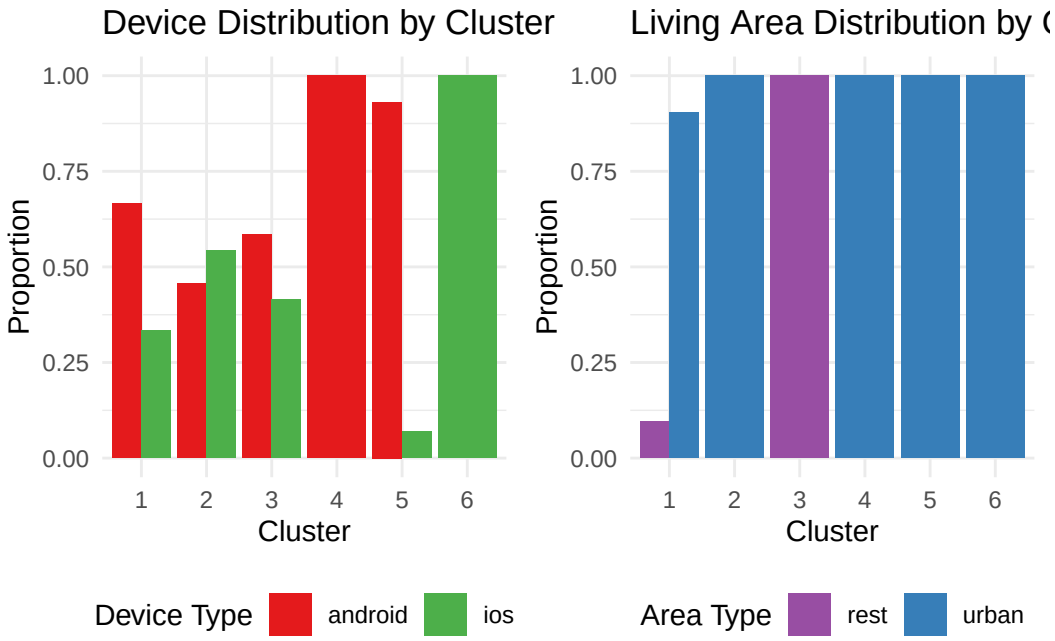
The trick part for this is to turn our scaled data frame back into factor values which we can identify as ios or android as well as urban or rest. That is because we need to perform some data manipulation before plotting. The main thing we do is to check if the device values is bigger than 0 (IOS) or smaller (android) and assign the correct label to that value. We perform the same for the living area column.

```r
# Calculate proportions for device
device_props <- clustering_data_no_premium %>%
  mutate(
    # ios was coded as 2 and android as 1
    device_type = ifelse(device > 0, "ios", "android")
  ) %>%
  # Group for each cluster and device
  group_by(Cluster, device_type) %>%
  # Count number of data points for each device
  summarise(count = n()) %>%
  # Calc proportion in each cluster
  mutate(proportion = count / sum(count))


# Calculate proportions for living area
living_area_props <- clustering_data_no_premium %>%
  mutate(
    # rest was coded as 2 and urban as 1
    area_type = ifelse(living_area > 0, "rest", "urban")
  ) %>%
  # group for each cluster and area type
  group_by(Cluster, area_type) %>%
  # Count number of data points for each region
  summarise(count = n()) %>%
   # Calc proportion in each cluster
  mutate(proportion = count / sum(count))
```

Plotting

```r
# Create bar chart for device distribution
p_device <- ggplot(device_props,
             aes(x = Cluster, y = proportion, fill = device_type)) +
  # identity to directly use the values of proportion in the df
  # dodge to create two bars (ios, android)
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  scale_fill_manual(values = c("android" = "#E41A1C", "ios" = "#4DAF4A")) +
  labs(title = "Device Distribution by Cluster",
       y = "Proportion",
       fill = "Device Type") +
```

```
  theme(legend.position = "bottom")

# Create bar chart for living area distribution
p_areas <- ggplot(living_area_props,
              aes(x = Cluster, y = proportion, fill = area_type)) +
    # identity to directly use the values of proportion in the df
  # dodge to create two bars (ios, android)
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  scale_fill_manual(values = c("urban" = "#377EB8", "rest" = "#984EA3")) +
  labs(title = "Living Area Distribution by Cluster",
       y = "Proportion",
       fill = "Area Type") +
  theme(legend.position = "bottom")

# Arrange plots side by side
grid.arrange(p_device, p_areas, ncol = 2)
```



When looking at the distribution of device and living areas in the different clusters we see that there is not much variation. For example in cluster 4 & 6 the device type and living area is the same for all data points. Also cluster 4 & 5 mainly consists of urban android users.

Conclusion K-means

For our data and goal the k-means algorithm seems a bit of an overfitt, as it creates quite a lot of clusters which do not seem to have enough distinguishing features to justified customized marketing measures.

DBSCAN

Because we are not quite satisfied with the results from the kmeans clustering we also try out an DBSCAN algorithm. The algorithm works by clustering data points based on the density of there distance. Which typically leads to very crowed cluster as well as some outlines.

Selecting the target variables

To run the algorithm we create a sub data frame which only contains the columns we want to included in our clustering workflow. For us it is again income and months.

```
clustering_vars <- clustering_data_no_premium %>%
  select(income, months)
```

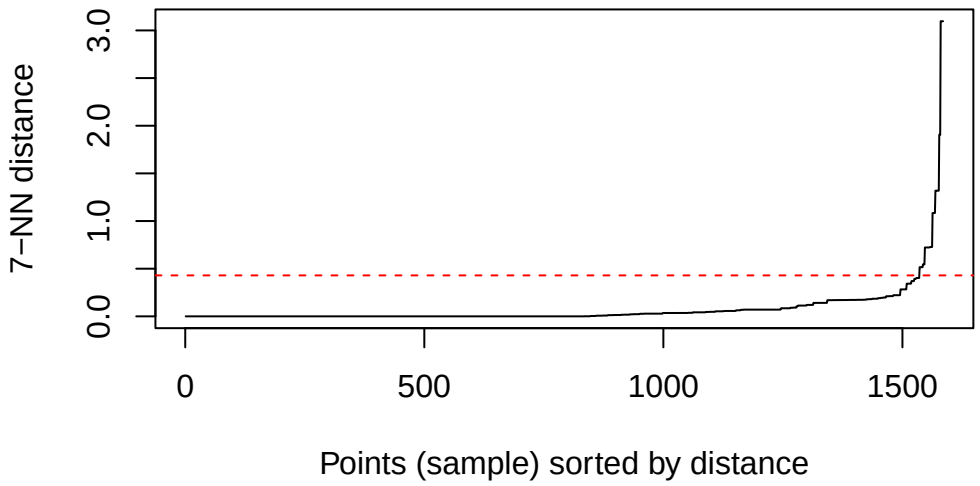
Finding optimal eps for DBSCAN

In order to create the best possible implementation of DBSCAN we use the elbow method to identify the correct eps for the algorithm. The eps is the size or radius of the area that we want to cluster together.

We implement the elbow method by plotting the distance of the k-nearest neighbors in ascending order. To get a relatively smooth curve we determine to use the distance between seven points. Also we add a red horizontal line to visualize where the “elbow” is located, in our case right around 0,43.

Thus we use an esp value of 0,43 for our DBSCAN algorithm.

```
kNNdistplot(clustering_vars, k = 7)
abline(h = 0.43, col = "red", lty = 2)
```



Perform DBSCAN clustering

We now run or DBSCAN algorithm with the predefined eps values as well as a number of points that need to be in a cluster to be created. After experimenting a bit, we selected 30 points as this gives a good balance between generalization and avoiding over fitting the algorithm.

The resulting clusters are then added as factors to our main data frame. Here it is important to know that DBSCAN assigns outliers or noise a separate cluster which is -1. We handle this by assigning those a label. However, in our tests with the data this was not the case.

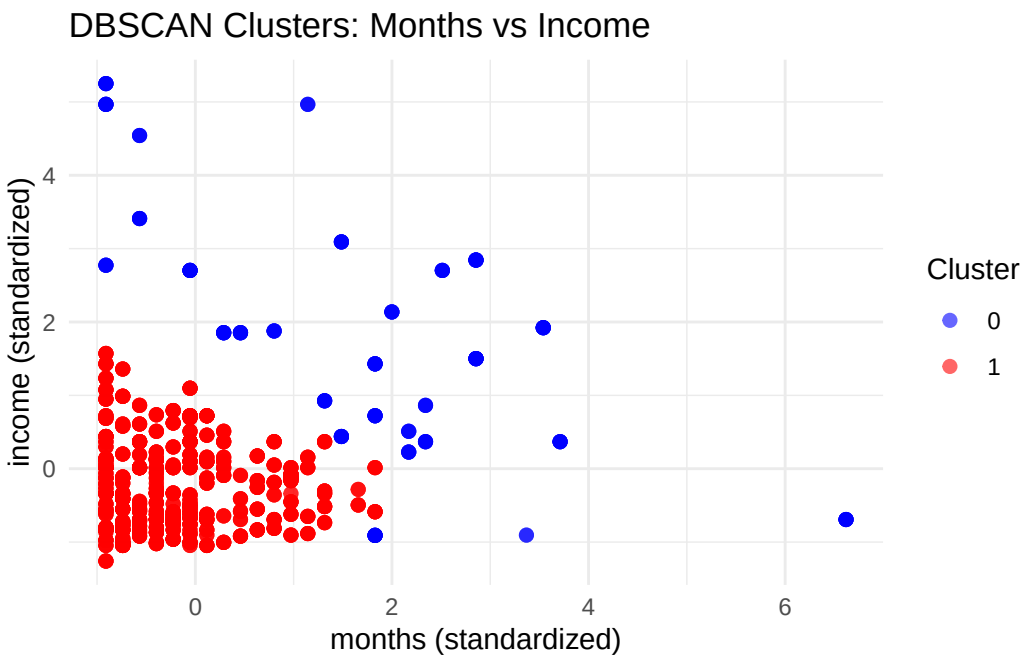
```
# Running the DBSCAN
dbscan_result <- dbscan(clustering_vars, eps = 0.43, minPts = 30)

# Add cluster assignments back to original dataframe
clustered_data <- clustering_data_no_premium %>%
  # Adding them as clusters
  mutate(Cluster = factor(dbscan_result$cluster),
         # renaming points with label -1 (noise)
         Cluster = recode(Cluster, `-1` = "Outlier"))
```

Plotting the clusters

We create a simple visualization of our clusters to check if the created cluster make sense. The results seem promising as one cluster identifies points with a low income (in the range of 0 to 2) an a months range of 0 to 2. The other cluster consisted of the points which are spread farther away and do not seem to be in a partial arrangement. When it comes to the scale of the x and y Axis it is important to know that those values are still scaled for clustering purpose.

For our marketing campaign the red points of cluster 1 seem to be interesting as they are relative new customers with a low income. We could this information in the rationalization of our messaging to them.

```
# Scatter plot
ggplot(clustered_data, aes(x = months, y =income, color = Cluster)) +
  geom_point(size = 2, alpha = 0.6) +
  theme_minimal() +
  # Also include a colur for outliers which are not the case in our clusters
  scale_color_manual(values = c("Outlier" = "grey50",
                                "0" = "blue",
                                "1" = "red")) +
  labs(title = "DBSCAN Clusters: Months vs Income",
       x = "months (standardized)",
       y = "income (standardized)")
```



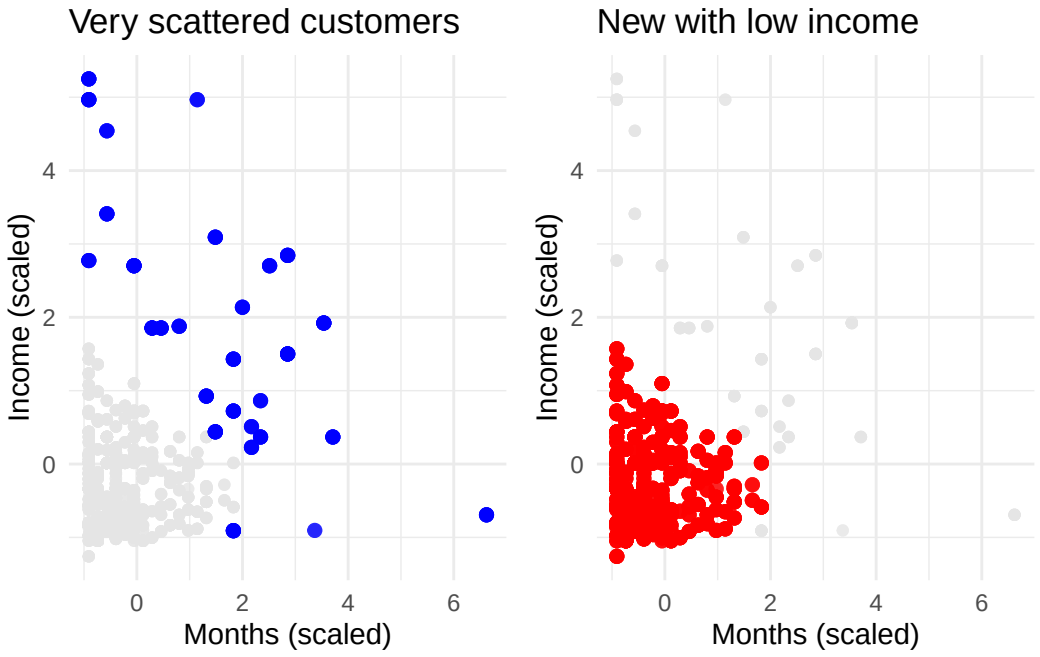
Separate visualization of the clusters

To explore the resulting Clusters deeper we create separate visualizations for each of them.

Also we assign provisional names to the clusters. The blue cluster is the one with very scatter customers and the red cluster is the one with new and low income customers.

```
cluster1 <- ggplot(clustered_data, aes(x = months, y = income)) +
  geom_point(color = "grey90", alpha = 0.5) +
  # Creating a subset for only cluster 0
  geom_point(data = subset(clustered_data, Cluster == "0"),
             aes(color = Cluster), size = 2, alpha = 0.6) +
  scale_color_manual(values = "blue") +
  theme_minimal() +
  labs(title = "Very scattered customers",
       x = "Months (scaled)",
       y = "Income (scaled)") +
  # Reomve legenend
  theme(legend.position = "none")


cluster2 <- ggplot(clustered_data, aes(x = months, y = income)) +
```

```
  geom_point(color = "grey90", alpha = 0.5) +
  # Creating a subset for only cluster 1
  geom_point(data = subset(clustered_data, Cluster == "1"),
             aes(color = Cluster), size = 2, alpha = 0.6) +
  scale_color_manual(values = "red") +
  theme_minimal() +
  labs(title = "New with low income",
       x = "Months (scaled)",
       y = "Income (scaled)") +
  theme(legend.position = "none")

# Arrange plots
grid.arrange(cluster1, cluster2, ncol = 2)
```



From the visualization we can tell that our target cluster is a majority of android users (around 75%) as well as living in Urban areas (around 90%).

Un-scaling the data frame

In order to present our campaign to management we want to predict the likleyhood of our target customers becoming premium users. To do this we create an "average user" on whose

information we will performer the predictions.

Since our data is still scaled for the clustering algorithm we first need to unscale it. Unscaling works by simply using the scaling formula but the other way around:

$$\text{original value} = (\text{scaled value} * \text{sample standard deviation}) + \text{sample mean}$$

To calculate this we create an unscaled data frame from the original input data, as well as a function which takes the scaled data frame and the original data frame as inputs. The function returns an unscaled data frame of our previously clustered data frame.

```
# Creating the reference data frame
# These are the same steps as in the begining of clusterign, just without scaling
original_df <- streaming_ds %>%
  filter(premium == 0) %>%
  select(stopped, age, income, premium, device, `living area`, months, `few hrs`) %>%
  mutate(few_hrs = as.numeric(`few hrs`)) %>%
  mutate(device = as.numeric(device)) %>%
  mutate(living_area = as.numeric(`living area`)) %>%
  select(-`few hrs`) %>%
  select(-`living area`) %>%
  select(-premium)


# Function to unscale all numeric columns while preserving Cluster
unscale_matched <- function(scaled_data, original_data) {
  # Create a copy of the scaled data
  result <- scaled_data

  # List all columns except Cluster
  cols_to_unscale <- setdiff(names(scaled_data), "Cluster")

  # Process each column
  for(col in cols_to_unscale) {
    # Calculate original mean and sd
    orig_mean <- mean(original_data[[col]], na.rm = TRUE)
    orig_sd <- sd(original_data[[col]], na.rm = TRUE)

    # Reverse the scaling
    result[[col]] <- scaled_data[[col]] * orig_sd + orig_mean
  }
```

```
  return(result)
}

# Run the function
unscaled_result <- unscale_matched(clustered_data, original_df)

# Quick comparison if scaling worked
# If the summary is the same for each column it worked
summary(original_df)
```

```
   stopped             age             income          device
Min.   : 0.0000   Min.   : 0.00   Min.   : 1.200   Min.   :1.00
1st Qu.: 0.0000   1st Qu.:24.00   1st Qu.: 2.000   1st Qu.:1.00
Median : 0.0000   Median :31.00   Median : 2.690   Median :1.00
Mean   : 0.4435   Mean   :33.03   Mean   : 2.979   Mean   :1.37
3rd Qu.: 0.0000   3rd Qu.:39.00   3rd Qu.: 3.500   3rd Qu.:2.00
Max.   :11.0000   Max.   :84.00   Max.   :10.400   Max.   :2.00
    months          few_hrs        living_area
Min.   : 0.000   Min.   :0.0000   Min.   :1.000
1st Qu.: 1.000   1st Qu.:0.0000   1st Qu.:1.000
Median : 4.000   Median :0.0000   Median :1.000
Mean   : 5.315   Mean   :0.3382   Mean   :1.071
3rd Qu.: 7.000   3rd Qu.:1.0000   3rd Qu.:1.000
Max.   :44.000   Max.   :1.0000   Max.   :2.000
```

```
summary(unscaled_result)
```

```
   stopped             age             income          device
Min.   : 0.0000   Min.   : 0.00   Min.   : 1.200   Min.   :1.00
1st Qu.: 0.0000   1st Qu.:24.00   1st Qu.: 2.000   1st Qu.:1.00
Median : 0.0000   Median :31.00   Median : 2.690   Median :1.00
Mean   : 0.4435   Mean   :33.03   Mean   : 2.979   Mean   :1.37
3rd Qu.: 0.0000   3rd Qu.:39.00   3rd Qu.: 3.500   3rd Qu.:2.00
Max.   :11.0000   Max.   :84.00   Max.   :10.400   Max.   :2.00
    months          few_hrs        living_area    Cluster
Min.   : 0.000   Min.   :0.0000   Min.   :1.000   0: 177
1st Qu.: 1.000   1st Qu.:0.0000   1st Qu.:1.000   1:1408
Median : 4.000   Median :0.0000   Median :1.000
Mean   : 5.315   Mean   :0.3382   Mean   :1.071
3rd Qu.: 7.000   3rd Qu.:1.0000   3rd Qu.:1.000
Max.   :44.000   Max.   :1.0000   Max.   :2.000
```

```r
str(unscaled_result)
```

```
'data.frame':   1585 obs. of  8 variables:
 $ stopped    : num  0 0 1 0 0 0 0 0 0 1 ...
 $ age        : num  30 35 25 22 22 30 38 26 50 33 ...
 $ income     : num  1.73 1.91 1.88 3.84 2.03 ...
 $ device     : num  2 2 1 1 1 1 1 1 1 1 ...
 $ months     : num  12 5 2 1 5 1 1 2 0 5 ...
 $ few_hrs    : num  1 0 1 1 0 0 1 0 1 0 ...
 $ living_area: num  1 1 1 2 1 1 1 1 1 1 ...
 $ Cluster    : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
```

Assigning text back to factor columns

Since in our orignal data frame we also have text saved as factors we need to recreate this as well.

```r
# Recreating device
# Factor of 1 = android; 2 = ios
unscaled_result$device <- factor(
  unscaled_result$device,
  levels = c(1, 2),
  labels = c("android", "ios")
)

# Recreating living area
# Factor of 1 = urban; 2 = rest
unscaled_result$living_area <- factor(
  unscaled_result$living_area,
  levels = c(1, 2),
  labels = c("urban", "rest")
)

str(unscaled_result)
```

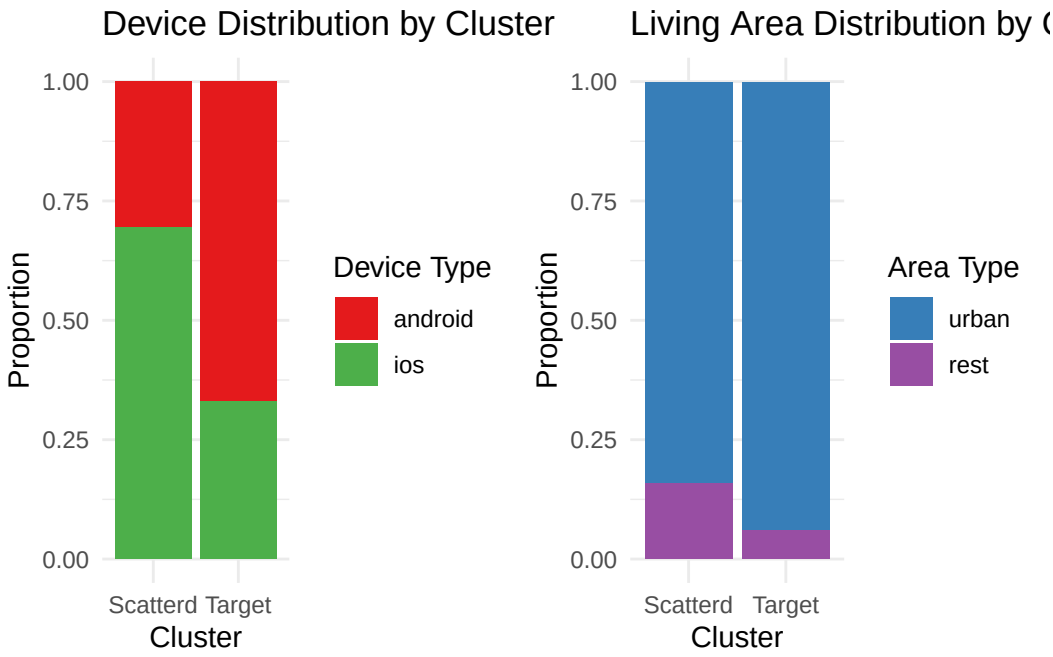
```
'data.frame':   1585 obs. of  8 variables:
 $ stopped    : num  0 0 1 0 0 0 0 0 0 1 ...
 $ age        : num  30 35 25 22 22 30 38 26 50 33 ...
 $ income     : num  1.73 1.91 1.88 3.84 2.03 ...
 $ device     : Factor w/ 2 levels "android","ios": 2 2 1 1 1 1 1 1 1 1 ...
 $ months     : num  12 5 2 1 5 1 1 2 0 5 ...
```

```
$ few_hrs    : num  1 0 1 1 0 0 1 0 1 0 ...
$ living_area: Factor w/ 2 levels "urban","rest": 1 1 1 2 1 1 1 1 1 1 ...
$ Cluster    : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
```

Further visualizations for our target customer segment

To better plan our campaign we explore which type of customer we are focusing on. For that we take a look at the distribution of devices and living areas for our target cluster as well as the distributed one.

```r
# Device distribution by cluster
device_plot <- ggplot(unscaled_result,
       aes(x = Cluster, fill = device)) +
  geom_bar(position = "fill") +
  # Assing labels
  scale_x_discrete(labels = c("0" = "Scatterd", "1" = "Target")) +
  scale_fill_manual(values = c("android" = "#E41A1C", "ios" = "#4DAF4A")) +
  labs(title = "Device Distribution by Cluster",
       y = "Proportion",
       fill = "Device Type") +
  theme_minimal()

# Living area distribution by cluster
area_plot <- ggplot(unscaled_result,
       aes(x = Cluster, fill = living_area)) +
  geom_bar(position = "fill") +
  scale_x_discrete(labels = c("0" = "Scatterd", "1" = "Target")) +
  scale_fill_manual(values = c("urban" = "#377EB8", "rest" = "#984EA3")) +
  labs(title = "Living Area Distribution by Cluster",
       y = "Proportion",
       fill = "Area Type") +
  theme_minimal()

# Display plots side by side
grid.arrange(device_plot, area_plot, ncol = 2)
```



Conclusion DBSCAN

For our given data and task the DBSCAN offers a fitting algorithm, as it allows us to determine a target cluster which consist of a good set of customers which features are sufficiently related to be targeted by specific marketing measures.

Calculate summary statistics for target customer base

Finally we create the mean values of our customers in each cluster. This is pretty straight forward as we simply take the mean of the numeric columns. For the columns saved as factor we check with the corresponding values to calculate a mean and assign a value based on the results.

For few_hrs we also create a mean which is in the range of 1 - 0, so we we assigned TRUE if it is >0.5 or FALSE otherwise.

The final data frame is saved as a csv to be used for the prediction tasks.

```r
summary_statistic_target <- unscaled_result %>%
  group_by(Cluster) %>%
  summarise(
    n_customers = n(),

    avg_stopped = round(mean(stopped), 0),

    avg_age = round(mean(age), 0),


    avg_income = round(mean(income), 4),

    avg_device = ifelse(round(mean(as.numeric(device)),
                              digits = 0) > 1, "ios", "android"),

    avg_months = round(mean(months), 0),

    few_hrs = as.logical(ifelse(round(mean(few_hrs),
                                      digits = 1) > 0.5, "TRUE", "FALSE")),

    avg_living_area = ifelse(round(mean(
      as.numeric(living_area)), digits = 0) > 1, "rest", "urban")
  )


str(summary_statistic_target)
```

```
tibble [2 x 9] (S3: tbl_df/tbl/data.frame)
 $ Cluster        : Factor w/ 2 levels "0","1": 1 2
 $ n_customers    : int [1:2] 177 1408
 $ avg_stopped    : num [1:2] 1 0
 $ avg_age        : num [1:2] 39 32
 $ avg_income     : num [1:2] 5.54 2.66
 $ avg_device     : chr [1:2] "ios" "android"
 $ avg_months     : num [1:2] 15 4
 $ few_hrs        : logi [1:2] FALSE FALSE
 $ avg_living_area: chr [1:2] "urban" "urban"
```

```r
write.csv(summary_statistic_target, "Summary_statistic_clusters.csv")
```

5. Predicting their likelihood of becoming premium members

Having summary statistics for each of the clusters allows us to roughly estimate how likely customers within the respective clusters are to become premium members. In order to estimate the likelihood of being premium, we would like to develop a model that is able to accurately predict whether or not a customer is a premium member given certain other characteristics. To do so, we compare four different classifiers: k-nearest neighbour, naive bayes, classification tree and random forest.

```
load("Group1_streaming_ds.rda")
data <- streaming_ds
# adjust the column names to avoid complications later on
colnames(data)[1] <- "fewhrs"
colnames(data)[6] <- "livingarea"

# convert target variable to a factor

data$premium <- factor(data$premium)
levels(data$premium) <- c("no","yes")
```

In order to be able to later evaluate our models, we divide our data into training and test data, using an 80-20 split.

```
n <- nrow(data)
n1 <- floor(0.8*n)
train_ind <- sample(1:n, n1)
data_train <- data[train_ind,]
data_test <- data[-train_ind,]
```

Naive Bayes Classifier

The first classifier we would like to fit to our data is a Naive Bayes classifier. Naive Bayes assumes independence among attributes and chooses the class such that the probability of observing the given features within that class is maximized. Key to the algorithm is the Bayes Rule:

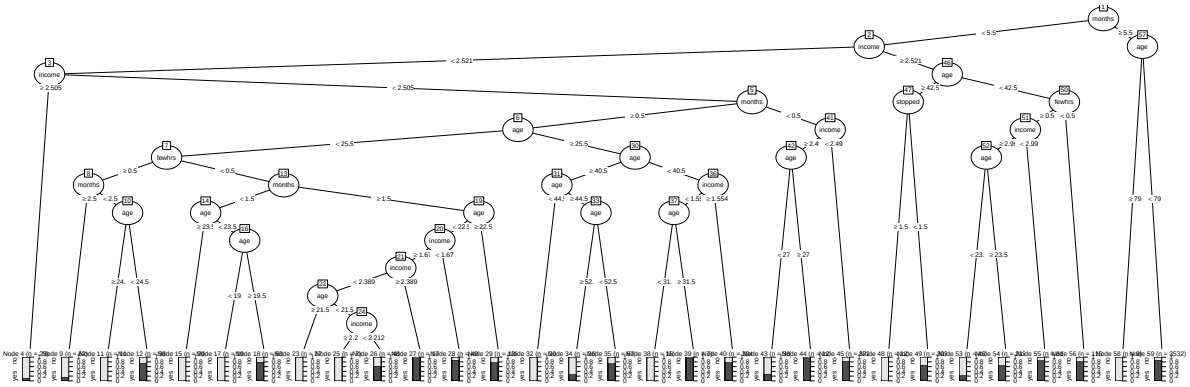
$$P(Y|X_1, X_2, ..., X_p) = \frac{P(X_1, X_2, ..., X_p|Y)P(Y)}{P(X1, X_2, ..., X_p)}$$

```
model_nb <- naiveBayes(premium ~ . , data=data_train)
```

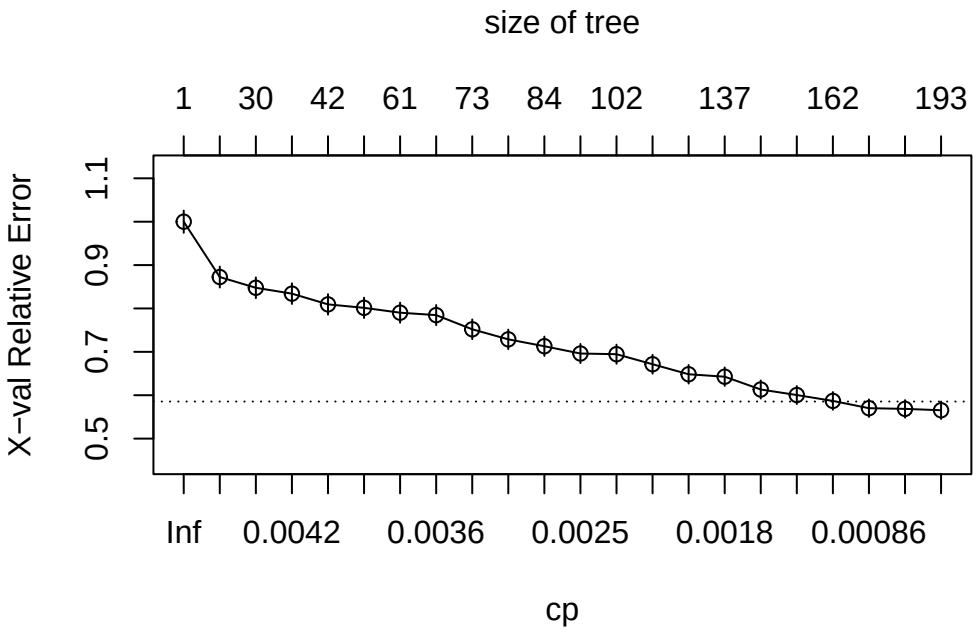
Classification tree

Additionally, we fit a classification tree to our data. These offer the advantage of being easily interpretable. Since smaller trees usually generalize better to the data, we decide to manually prune to tree to fewer splits.

```
set.seed(1234)
library("rpart")
library(partykit)
full_tree <- rpart(premium ~., data =data_train, control=list(cp=0))

pruned_tree <- prune(full_tree, cp=0.0046)

plot(as.party(pruned_tree))
```



```
full_tree <- rpart(premium ~., data =data_train, control=list(cp=0))

plotcp(full_tree)
```




```
full_tree$cptable[1:10,]
```

```
            CP nsplit rel error    xerror       xstd
1  0.005183413      0 1.0000000 1.0000000 0.02551119
2  0.004784689     27 0.8062201 0.8724083 0.02416822
3  0.004385965     29 0.7966507 0.8476874 0.02388774
4  0.004066986     31 0.7878788 0.8341308 0.02373092
5  0.003987241     41 0.7472089 0.8094099 0.02343929
6  0.003854333     54 0.6770335 0.8014354 0.02334362
7  0.003721425     60 0.6539075 0.7902711 0.02320834
8  0.003455609     64 0.6363636 0.7846890 0.02314011
9  0.003189793     72 0.6052632 0.7519936 0.02273233
10 0.002870813     78 0.5845295 0.7288676 0.02243521
```

As can already be observed from the plot and the table above, when choosing CP such that we have a smaller number of splits, we still have a very high relative error. This suggests that a much more complex tree would be necessary to classify our customers accurately. More complex trees do, however, come at the cost of much higher variance, meaning that small changes in the training data can lead to large changes in the tree structure. Furthermore, we would loose the advantage of interpretability that decision trees have as compared to other classifiers.

The tree suggests that the most important variables for classifcation are `months`, `income`, `age` and `stopped`.

Random Forest

Finally, we fit a random forest to our data. Random forests are bagged classification trees trained using a random subset of predictors. Random forests often usually yield much better prediction performance than single trees when being used out-of-the-box.

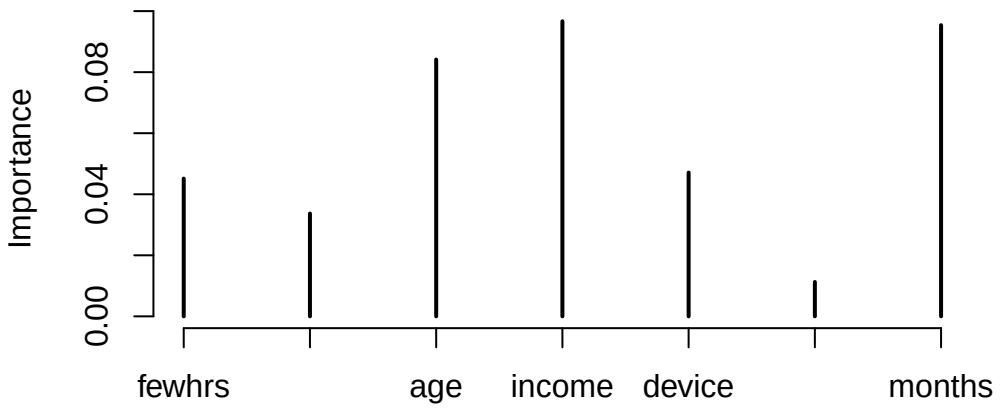
```
randomforest <- ranger(premium~., data=data_train,
                       probability=TRUE, importance="permutation")
print(randomforest)
```

```
Ranger result

Call:
 ranger(premium ~ ., data = data_train, probability = TRUE, importance = "permutation")

Type:                             Probability estimation
Number of trees:                  500
Sample size:                      6820
Number of independent variables:  7
Mtry:                             2
Target node size:                 10
Variable importance mode:         permutation
Splitrule:                        gini
OOB prediction error (Brier s.):  0.05434913
```

Variable importance

We would like to better understand how our random forest classifies by visualizing the importance of variables within our random forest.

```
plot(as.table(importance(randomforest)), ylab="Importance")
```



Again, variables that are considered particularly important are `months`, `income` and `age`.

Model comparison

Performance Measures

We would now like to choose the classifier with the best prediction performance on unseen data.

```
absolute_counts <- table(data$premium)
relative_counts <- table(data$premium) / nrow(data)


summary_df <- data.frame(
  Category = names(absolute_counts),
  Absolute = as.numeric(absolute_counts),
  Relative = round(as.numeric(relative_counts), 3)
)

knitr::kable(summary_df, caption = "Premium")
```

Table 1: Premium

Category	Absolute	Relative
no	1585	0.186
yes	6940	0.814

Since we are faced with a rather unbalanced distribution of our target variable, we decide to not only consider accuracy, i.e. the percentage of correctly classified data, but also look at other performance measures. Precision provides the percentage of predicted positives which are correctly classified, thereby accounting for false positives, whereas recall computes the percentage of correctly classified positives, thereby accounting for false negatives.

```
yhat_tree <- predict(pruned_tree, newdata=data_test, type="class")
yhat_rf <- as.numeric((predict(randomforest,
                               data=data_test)$predictions >0.5)[,2])
yhat_nb <- predict(model_nb, newdata=data_test)

tab1 <- table(predicted_tree=yhat_tree, observations=data_test$premium)
tab2 <- table(predicted_rf=yhat_rf, observations=data_test$premium)
tab3 <- table(predicted_nB=yhat_nb, observations=data_test$premium)
```

```
acc1 <- (tab1[1,1]+tab1[2,2])/sum(tab1)
acc2 <- (tab2[1,1]+tab2[2,2])/sum(tab2)
acc3 <- (tab3[1,1]+tab3[2,2])/sum(tab3)

rec1 <- tab1[2,2]/sum(tab1[,2])
rec2 <- tab2[2,2]/sum(tab2[,2])
rec3 <- tab3[2,2]/sum(tab3[,2])

prec1 <- tab1[2,2]/sum(tab1[2,])
prec2 <- tab2[2,2]/sum(tab2[2,])
prec3 <- tab3[2,2]/sum(tab3[2,])


evaluation <- data.frame(  Model = c("Tree", "Random Forest", "Naive Bayes"),
                           accuracy = c(acc1, acc2, acc3),
                           recall = c(rec1, rec2, rec3),
                           precision = c(prec1, prec2, prec3)  )

evaluation
```

```
          Model  accuracy    recall precision
1          Tree 0.8287390 0.9905386 0.8298780
2 Random Forest 0.9143695 1.0000000 0.9039474
3   Naive Bayes 0.8046921 0.9934498 0.8081705
```

As can be concluded from the table above, the random forests yields the best prediction performance, regardless of the performance measure used for evalutation. We have a closer look at how the data was classified by the random forest.

```
rownames(tab2)<- c("no", "yes")
knitr::kable(tab2)
```

	no	yes
no	185	0
yes	146	1374

10-Fold Cross Validation

Additionally, we perform a 10-fold cross validation. We generate ten folds, loop over them, fit the Naive Bayes, classification tree and random forest on the train data and evaluate the models on the test data by computing the Brier score (mean squared error for binary responses).

```
n <- nrow(data)
fold <- 10
folds <- sample(rep(1:fold, ceiling(n/fold)),n)
```

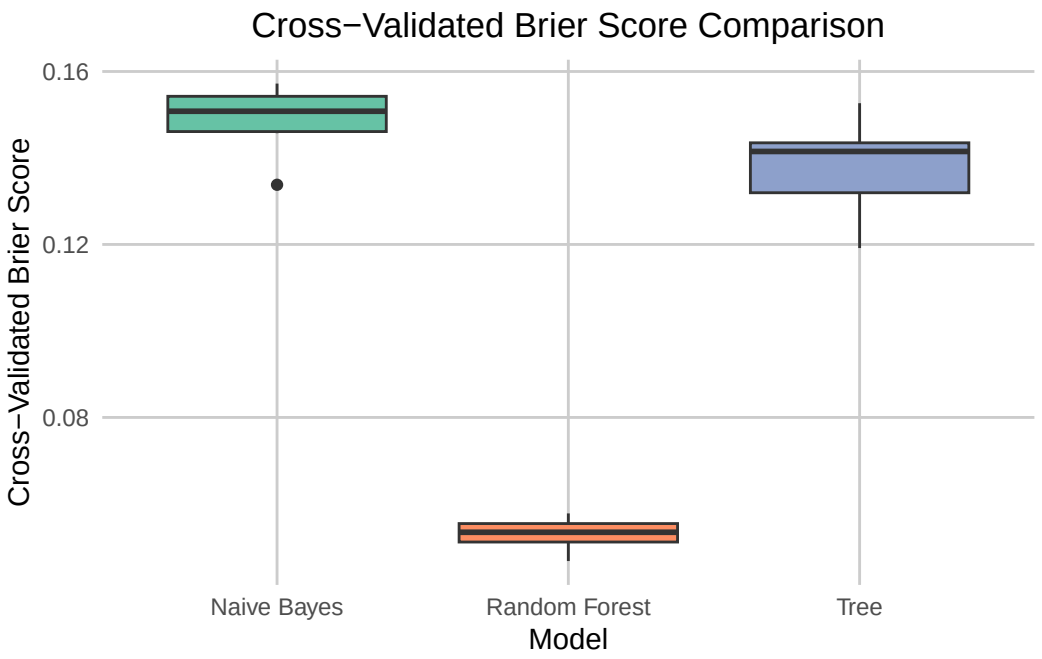
```
brier <- list()

for (tfold in seq_len(fold)){

  train_idx <- which(folds != tfold)
  test_idx <- which(folds == tfold)

  # fit the models to the respective training data

  rf <- ranger(premium ~., data=data[train_idx,], probability=TRUE)
  tree <- prune(rpart(premium ~.,
                      data =data[train_idx,],
                      control=list(cp=0)),
                      cp=rpart(premium ~.,
                      data =data,
```

```
                      control=list(cp=0))$cptable[3, "CP"])

  nb <- naiveBayes(premium ~ . , data=data[train_idx,])

  p_rf <- predict(rf, data=data[test_idx,])$predictions[,2]
  p_tree <- predict(tree, newdata=data[test_idx,])[,2]
  p_nb <- predict(nb, newdata=data[test_idx,], type="raw")[,2]

  brier[[tfold]] <- unlist(lapply(
    list("Random Forest"=p_rf, "Tree"=p_tree,
         "Naive Bayes"=p_nb), \(predicted){
      mean((as.numeric(data[test_idx,"premium"]=="yes")-predicted)^2)
    }
  ))
}
```

```
brier_df <- do.call(rbind, brier)
colnames(brier_df) <- c("Random Forest", "Tree", "Naive Bayes")
brier_df <- as.data.frame(brier_df)
brier_df$Fold <- seq_len(nrow(brier_df))
brier_long <- pivot_longer(brier_df,
                           cols = c("Random Forest", "Tree", "Naive Bayes"),
                            names_to = "Model", values_to = "Brier Score")
```

This allows us to plot the distribution of brier scores within the ten folds.

```
ggplot(brier_long, aes(x = Model, y = `Brier Score`, fill = Model)) +
  geom_boxplot() +
  theme_minimal() +
  labs(title = "Cross-Validated Brier Score Comparison",
       x = "Model",
       y = "Cross-Validated Brier Score") +
  scale_fill_brewer(palette = "Set2") +
  theme(legend.position = "none",
        plot.title = element_text(hjust = 0.5),
        axis.text.x = element_text(angle = 0),
        panel.grid.major = element_line(color = "grey80"),
        panel.grid.minor = element_blank())
```



Again, we come to the conclusion that we are able to provide the most appropriate prediction when using our random forest as a classifier.

Predicting the likelihood of our average target customer to go premium

Using our developed classifier, we are now able to predict the likelihood with which the average customer in our targeted clusters will go premium.

```
cluster_data <- summary_statistic_target[2,] # we target the second cluster
```

We adjust the names of the columns such that they fit the names of the variables that the classifier was trained on.

```
colnames(cluster_data) <- c("Cluster", "n_customers","stopped", "age",
                            "income", "device", "months",
                            "fewhrs", "livingarea")
```

Using the random forest we are able to compute how likely the average customer of our two targeted clusters is to buy a premium membership.

```
as.data.frame(predict(randomforest, data=cluster_data[,-c(1,2)])$prediction)
```

```
    no      yes
```

```
1 0.1811427 0.8188573
```

```
cluster_data$Prediction <- predict(randomforest, data = cluster_data[, -c(1, 2)])$predicti

cluster_data_rounded <- cluster_data %>%   mutate(across(where(is.numeric), ~ round(., 3))

knitr::kable(cluster_data_rounded)
```

Cluster	n_customers	stopped	age	income	device	months	fewhrs	livingarea	Prediction
1	1408	0	32	2.657	android	4	FALSE	urban	0.819

We apply this predicted probability to the number of customers within the targeted cluster in order to get a rough estimate regarding the number of premium memberships we could potentially sell.

```
as.numeric(cluster_data$n_customers*cluster_data$Prediction)
```

```
[1] 1152.951
```