Main

Todos:

1) Briefly describe the data set (e.g., how many customers, what information are available, how do important variables look like).

```
# message = FALSE becasue we want no output when loading packages
library(ggplot2)
```

```
Warning: package 'ggplot2' was built under R version 4.2.3
```

```
library(ggcorrplot)
```

```
Warning: package 'ggcorrplot' was built under R version 4.2.3
```

```
library(ggpubr)
```

```
Warning: package 'ggpubr' was built under R version 4.2.3
```

```
library(rpart)
```

```
Warning: package 'rpart' was built under R version 4.2.3
```

```
library(partykit)
```

```
Warning: package 'partykit' was built under R version 4.2.3
```

```
Warning: package 'libcoin' was built under R version 4.2.3
```

```
Warning: package 'mvtnorm' was built under R version 4.2.3
```

```r
library(dplyr)
```

```
Warning: package 'dplyr' was built under R version 4.2.3
```

```r
library(gridExtra)
```

```
Warning: package 'gridExtra' was built under R version 4.2.3
```

```r
library(dbscan)
```

```
Warning: package 'dbscan' was built under R version 4.2.3
```

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

Running Code

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

2. Identify the relevant variables that would help you to segment/cluster/classify the users/customers and justify your choice.

3) Test several models and compare them. Consider the best approach for your problem - clustering, classification, anything else?

-> Predict if a customer is a Premium member or not Additionally predict number of stopped streams

4) Identifiying 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 prepartaion 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.

```
# Data frame with all customers
clustering_data  <- streaming_ds %>%
  select(stopped, age, income, premium, device, `living area`, months, `few hrs`) %>% # ca
  mutate(few_hrs = as.numeric(`few hrs`)) %>% # Save few hrs as numeric; also rename col
  mutate(device = as.numeric(device)) %>% # Save device as numeric
  mutate(living_area = as.numeric(`living area`)) %>% # Save living area as numeric
  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
```

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


## 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
print("----- Non-Premium Customers -----")
```

```
[1] "----- Non-Premium Customers -----"
```

```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.

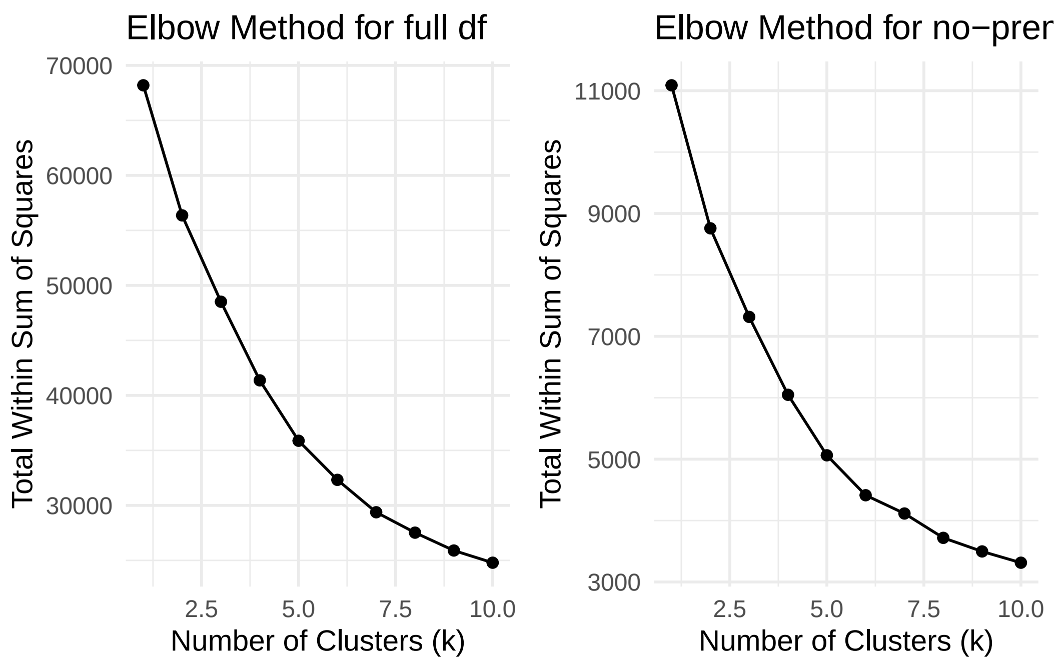
Getting number of clusters using the elbow method

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.

```r
# 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
    km <- kmeans(data, centers = i, nstart = 20) # calc kmeans for each number of clusters
    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)
```

```r
# 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()


# 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.

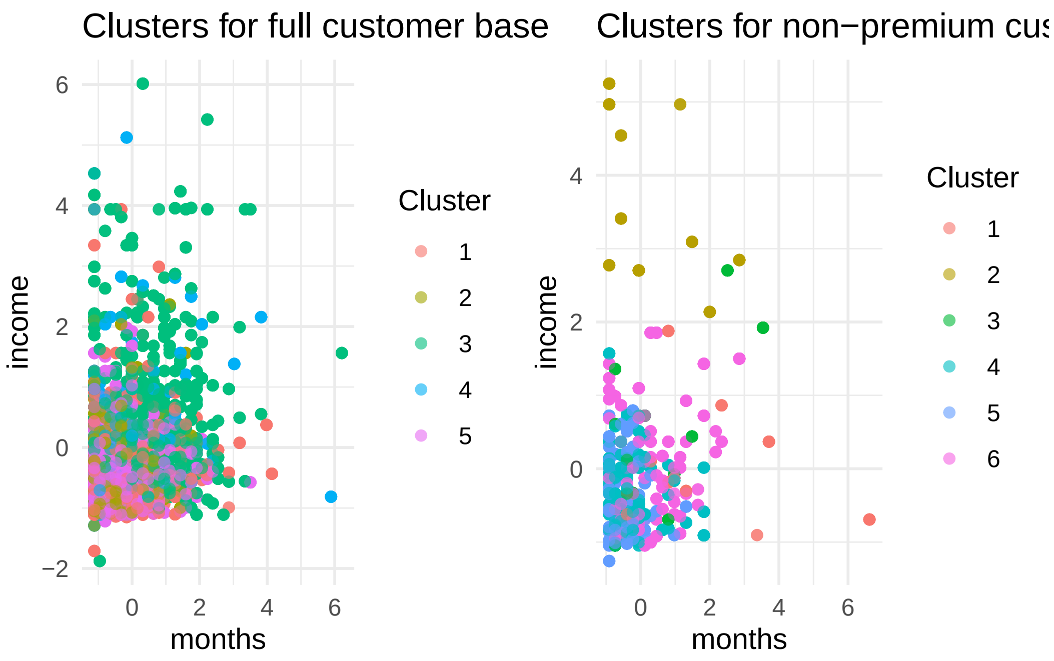
We made this decision by looking at the slope of the curve. The goal of the elbow method is to find the number of cluster where the curves slope is starting to flatten of, meaning increasing the number of clusters will only lead to diminishing returns.

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

```r
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
    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)
```



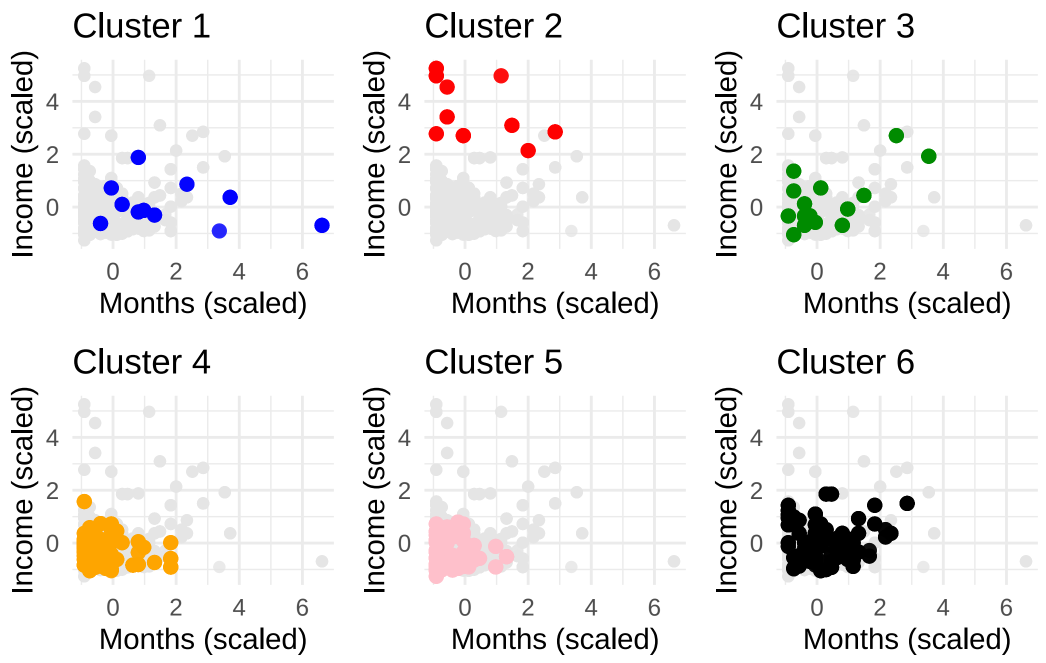
From the visualized cluster we see that it is very hard to identify any clusters for the full customer base. Because of this we decided to stick with only the non-premium customers for the further analysis.

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

# 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)") +
```

```r
  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))
```

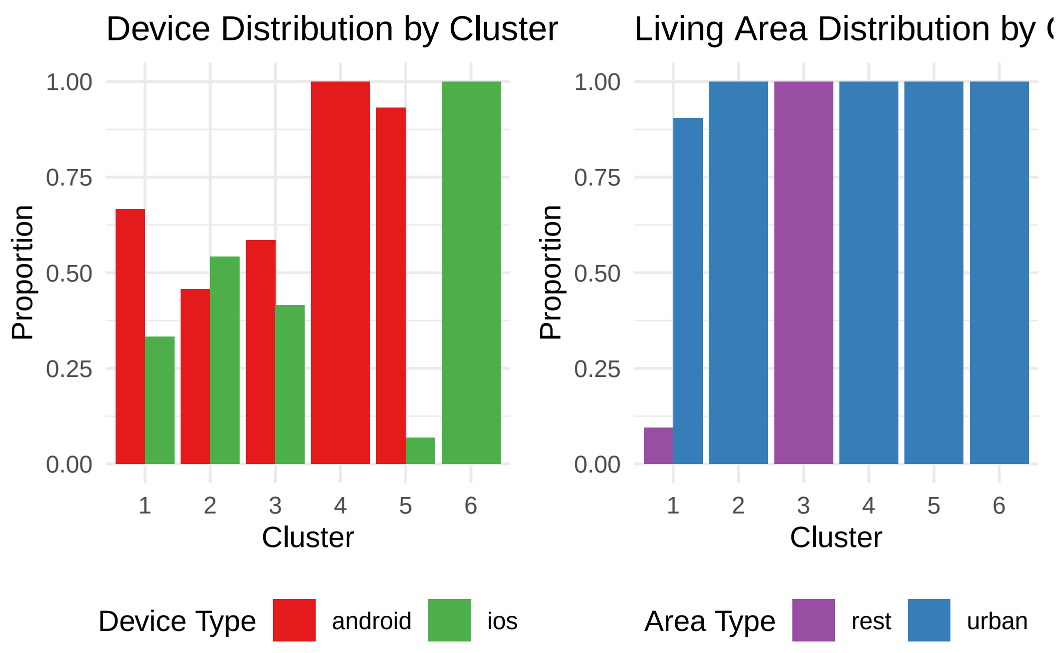
```
`summarise()` has grouped output by 'Cluster'. You can override using the
`.groups` argument.
```

```r
# 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))
```

```
`summarise()` has grouped output by 'Cluster'. You can override using the
`.groups` argument.
```

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.

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

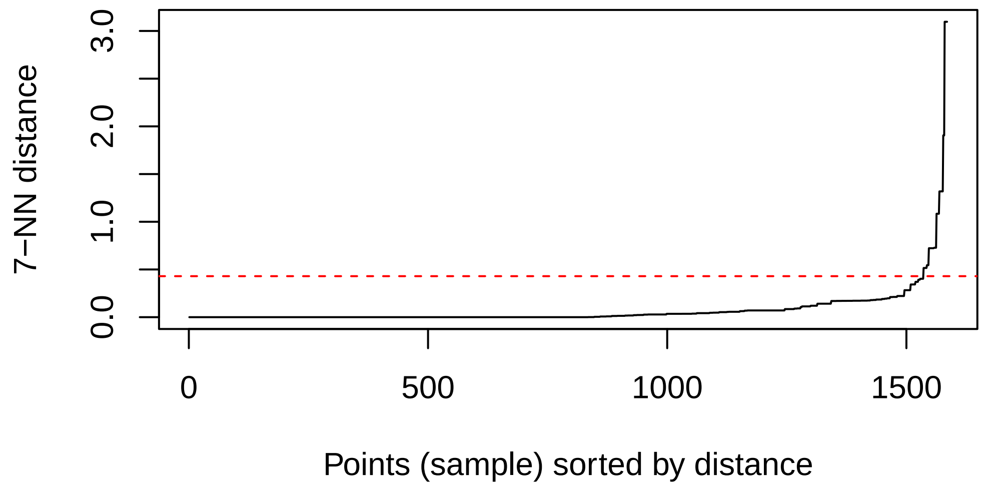
Finding optimal eps for DBSCAN

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.

```r
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.

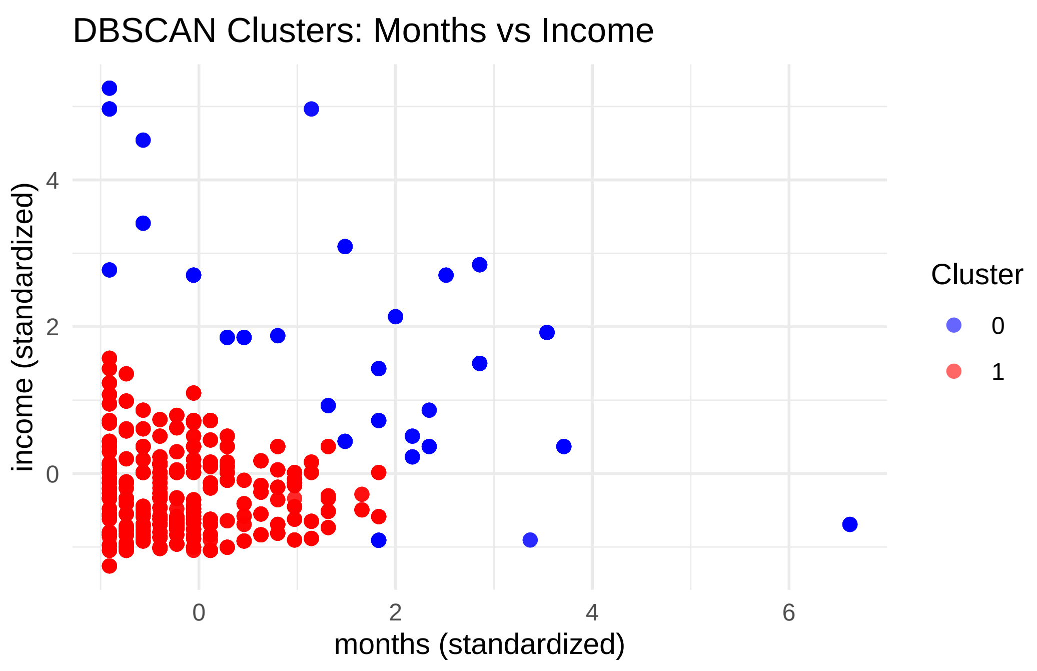
```
# Running the DBSCAN
dbscan_result <- dbscan(clustering_vars, eps = 0.43, minPts = 30)

# 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.

```r
# 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)")
```

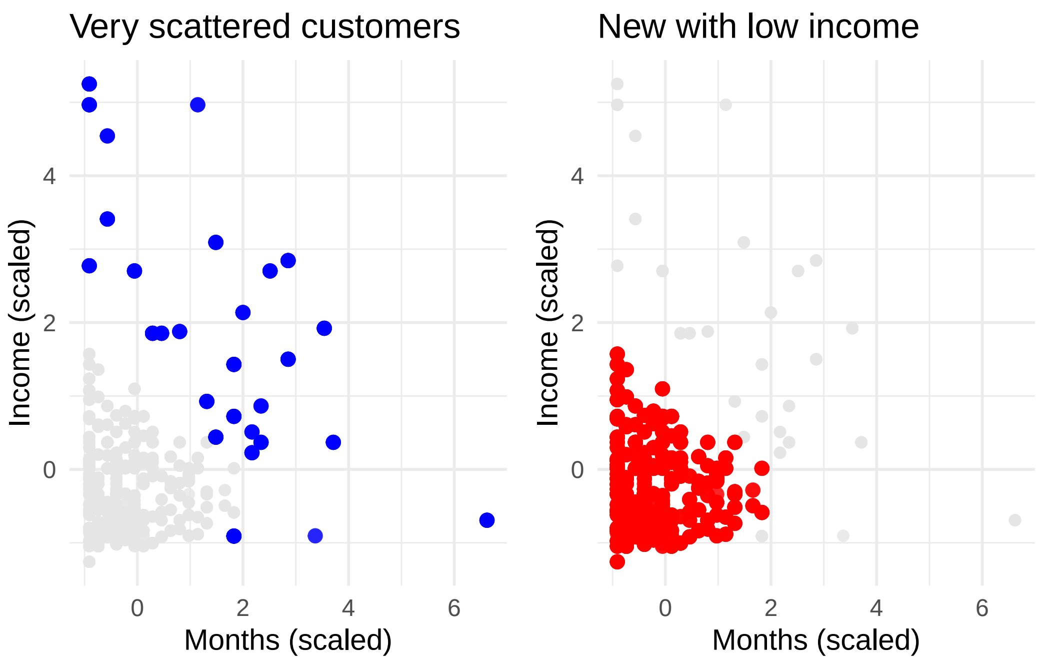


Separate visualization of the clusters

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:

original value = (scaled value * sample standard deviation) + 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.

```r
# 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 ...
```

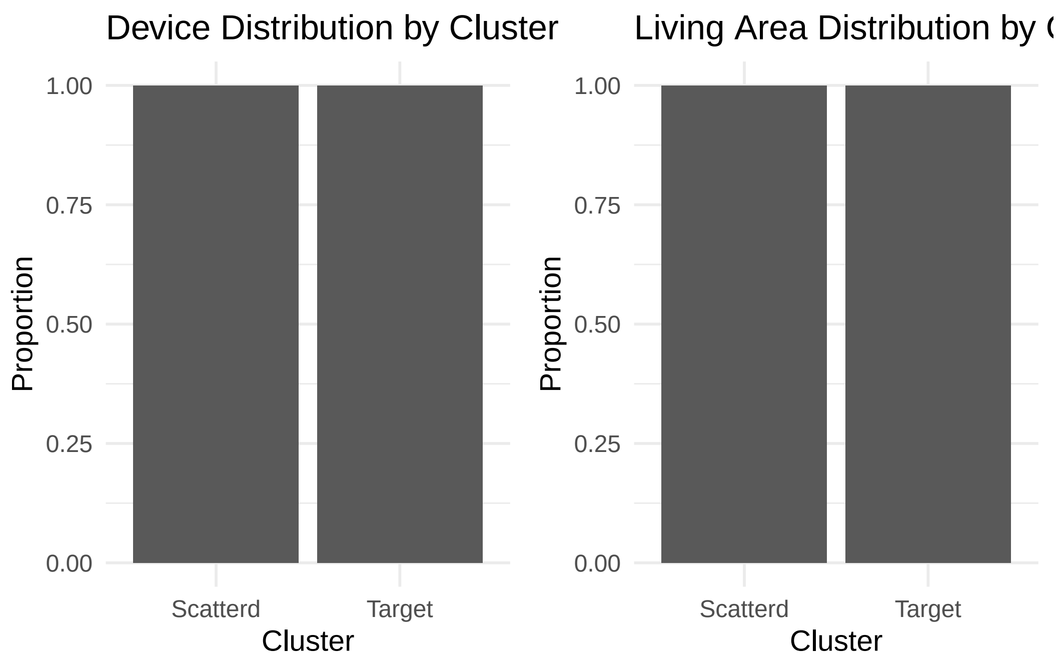
Further visualizations for our target customer segment

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()
```

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

```
Warning: The following aesthetics were dropped during statistical transformation: fill.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?

Warning: No shared levels found between `names(values)` of the manual scale and the
data's fill values.

Warning: The following aesthetics were dropped during statistical transformation: fill.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?

Warning: No shared levels found between `names(values)` of the manual scale and the
data's fill values.
```



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.

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 ...
```

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 assinged 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_satistic_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",
  )


str(summary_satistic_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"
```

```
write.csv(summary_satistic_target, "Summary_statistic_clusters.csv")
```

5) Do the data reveal how do the chosen customers differ from the others?