Capstone Project 2: *Online Retail*
Data Science Professional Certificate/HarvardX

Paw Hermansen

June 17, 2019

Executive Summary

The Data

I examine online retail sales data found in the `Online Retail` dataset that contains all transactions for about a year around 2011 for a UK based online retail. The products are unique gift items and many of the customers are wholesalers.

The dataset contains 25900 invoices for 4373 customers. It needed only little cleaning - I deleted invoices with no customer, I removed a few invoice rows with non-product items like handling and postage, and I removed two cancelled orders with exceptionally large quantities.

After cleaning, the dataset contains 21784 invoices for 4362 customers.

The Goals

Part 1: Customer Segmentation (Clustering)

RFM, **R**ecency, **F**requency, and **M**onetary Value, is an often used way to measure the value of a customer. I introduce a new Customer Segmentation based on the RFM data computed by the Kmeans clustering algorithm and compare it to a traditional fixed Customer Segmentation defined by PUTLER also based on RFM.

Part 2: Predicting Customer Segment for new Customers (Classification)

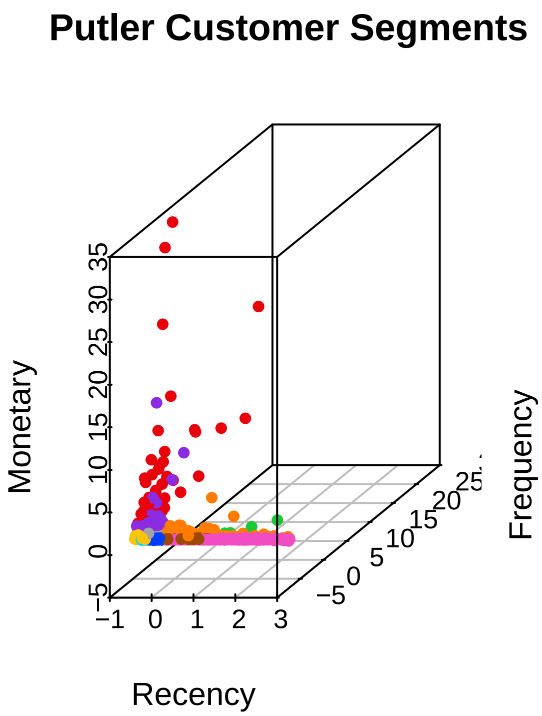
I examine if it is possible to predict which Customer Segment a new user will end up in at the last date of the dataset based on the size of the customers first invoice in the dataset and the customers demographics, i.e. the customers country.

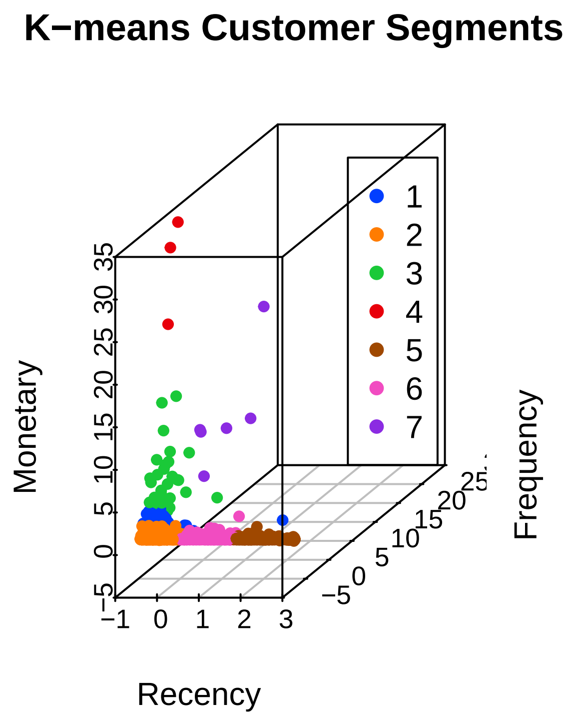
I classify to both of the two kinds of customer segment sets computed in the first part, and I use and compare the K-nearest neighbor (knn) algorithm, the Random Forest algorithm, and an approach with random segment selection. I chose the knn and Random Forest algorithms because they are base on very different principles.

The Results

The Result Part 1: Customer Segmentation

Each dot in the following two plots is a customer. The position represents the R, F, and M values and the colors represent the customer segment in each of the two systems:





The Putler segmentation has eleven segments and is shown for comparison in the left plot. The new Customer Segmentation computed by Kmeans clustering has seven segments and is shown in the right plot. The percentages of customers in each segment and the names I gave the segments are:

	Name	Cust. pct	Desciption
1	Small and Occasional	9.2%	Recently active but spends low and not very often
2	New	57.8%	Fairly recently active but spended low and have not been active earlier
3	Good Average	0.9%	Active and buys from seldom to often and spends from low to average
4	Big Spenders	0.1%	Spend most money of all and they recently did it again
5	Inactive	14.1%	Not active for a long time and spended low
6	Fading Away	17.7%	Not active for some time and spended low
7	Good and Loyal	0.1%	Spend average or more money and buy very often

The Result Part 2: Predicting Customer Segment

The accuracy of predicting the customer segment from the customers first invoice is:

Method	Putler Segments Accuracy	Kmeans Segments Accurary
Random	0.127	0.355
K-nearest neighbor	0.137	0.306
Random Forest	0.178	0.391

Conclusion

Customer Segmentation

My Machine Learning generated customer segments ends up with seven customer segments compared to eleven for the PUTLER approach. Other traditional RFM approaches, however, splits the the customers into between about five and twenty segments, and so the my Machine Learning generated customer segments are within what can be considered normal. The analysis, though, shows that the two sets of customer segments splits the customers in very different ways, and the segments of the two approaches do not look like each other. It is impossible to say which set of the segment sets that are best because it depends on the intended use.

Further work could be done to make an analysis of the development over time of customers. To find dynamic trends or customer lifecycles would however require a dataset that covers a longer period of at least several years for a business like the one, I analysed in this report.

Predicting Customer Segment

The K-nearest neigbor algorithm predicted the future customer segment worse than randomly for the new Kmeans segments! The Random Forest algorithm predicted the future customer segment of both customer segment sets a little bit better than random and also better the K-nearest neighbor algorithm. The accuracy is still low for both segment sets, though. Predicting using Random Forest might be a beginning of helping the business to a finer segmentation of new customers.

It is clearly easier to predict the Kmeans segments - this is quite naturally because of the fewer segments and the very high number of customers in a single segment.

In this report I did not use the information of the content of the first order and it would be interesting to investigate if using this data could help increase the accuracy of the predicting. Also the Random Forest analyses had a problem finding the best value of `mtry` and this should be fixed.

The Data and the Goal

I have chosen to examine online retail sales data found in the `Online Retail` dataset from https://archive.ics.uci.edu/ml/datasets/Online+Retail. The documentation describes the dataset as:

> This is a transnational data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail. The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers.

In the first part of this report I compare a traditional RFM Customer Segmentation of this data to a new Customer Segmentation of this data computed by the Kmeans clustering algorithm.

In the second part of this report I examine if it is possible to predict which Customer Segment a new user will end up in after a year, i.e classification of new customers. I classify to both of the two kinds of customer segment sets computed in the first part, and I use and compare the K-nearest neighbor (knn) algorithm and the Random Forest algorithm for the classifications. I chose knn and Random Forest because they work on different principles.

Data Exploration and Cleaning

Basic Information

The original data `Online Retail.xlsx` looks like:

```
## Observations: 541,909
## Variables: 8
## $ InvoiceNo   <chr> "536365", "536365", "536365", "536365", "536365", ...
## $ StockCode   <chr> "85123A", "71053", "84406B", "84029G", "84029E", "...
## $ Description <chr> "WHITE HANGING HEART T-LIGHT HOLDER", "WHITE METAL...
## $ Quantity    <dbl> 6, 6, 8, 6, 6, 2, 6, 6, 6, 32, 6, 6, 8, 6, 6, 3, 2...
## $ InvoiceDate <dttm> 2010-12-01 07:26:00, 2010-12-01 07:26:00, 2010-12...
## $ UnitPrice   <dbl> 2.55, 3.39, 2.75, 3.39, 3.39, 7.65, 4.25, 1.85, 1....
## $ CustomerID  <dbl> 17850, 17850, 17850, 17850, 17850, 17850, 17850, 1...
## $ Country     <chr> "United Kingdom", "United Kingdom", "United Kingdo...
```

Each of the 541909 rows is one line of an invoice. Rows with the same `InvoiceNo` make up an invoice for a customer.

The documentation of the dataset describes the fields:

- `InvoiceNo`: Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation.
- `StockCode`: Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.
- `Description`: Product (item) name. Nominal.
- `Quantity`: The quantities of each product (item) per transaction. Numeric.

- `InvoiceDate`: Invice Date and time. Numeric, the day and time when each transaction was generated.
- `UnitPrice`: Unit price. Numeric, Product price per unit in sterling.
- `CustomerID`: Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.
- `Country`: Country name. Nominal, the name of the country where each customer resides.

The data contains 25900 different invoices and 4373 different customers. The temporal extension of the data is from 2010-12-01 07:26:00 to 2011-12-09 11:50:00, i.e. a couple days more than one year of data.

The first few lines of the data and a summary of the data is:

```
##   InvoiceNo StockCode                        Description Quantity
## 1    536365    85123A  WHITE HANGING HEART T-LIGHT HOLDER        6
## 2    536365     71053                 WHITE METAL LANTERN        6
## 3    536365    84406B      CREAM CUPID HEARTS COAT HANGER        8
## 4    536365    84029G KNITTED UNION FLAG HOT WATER BOTTLE        6
## 5    536365    84029E      RED WOOLLY HOTTIE WHITE HEART.        6
## 6    536365     22752        SET 7 BABUSHKA NESTING BOXES        2
##           InvoiceDate UnitPrice CustomerID        Country
## 1 2010-12-01 07:26:00      2.55      17850 United Kingdom
## 2 2010-12-01 07:26:00      3.39      17850 United Kingdom
## 3 2010-12-01 07:26:00      2.75      17850 United Kingdom
## 4 2010-12-01 07:26:00      3.39      17850 United Kingdom
## 5 2010-12-01 07:26:00      3.39      17850 United Kingdom
## 6 2010-12-01 07:26:00      7.65      17850 United Kingdom
```

```
##   InvoiceNo          StockCode         Description
##  Length:541909      Length:541909      Length:541909
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##     Quantity          InvoiceDate                    UnitPrice
##  Min.   :-80995.00   Min.   :2010-12-01 07:26:00   Min.   :-11062.06
##  1st Qu.:     1.00   1st Qu.:2011-03-28 10:34:00   1st Qu.:     1.25
##  Median :     3.00   Median :2011-07-19 16:17:00   Median :     2.08
##  Mean   :     9.55   Mean   :2011-07-04 13:02:44   Mean   :     4.61
##  3rd Qu.:    10.00   3rd Qu.:2011-10-19 10:27:00   3rd Qu.:     4.13
##  Max.   : 80995.00   Max.   :2011-12-09 11:50:00   Max.   : 38970.00
##
##    CustomerID       Country
##  Min.   :12346    Length:541909
##  1st Qu.:13953    Class :character
##  Median :15152    Mode  :character
##  Mean   :15288
##  3rd Qu.:16791
##  Max.   :18287
##  NA's   :135080
```

From the summary we make two suspicious observations that need to be investigated further later in this report:

- The lowest quantity is negative and the highest quantity is the positive of the same number.
- The lowest unit-price is negative and the highest unit-price is nearly UK £ 40,000.

Explore and Clean the Data

Missing data

The number of missing data in each field is:

```
## Rows with missing InvoiceNo = 0
```

```
## Rows with missing StockCode = 0
```

```
## Rows with missing Description = 1454
```

```
## Rows with missing Quantity = 0
```

```
## Rows with missing InvoiceDate = 0
```

```
## Rows with missing UnitPrice = 0
```

```
## Rows with missing CustomerID = 135080
```

```
## Rows with missing Country = 0
```

I want to analyse the customers and I cannot use invoice rows with no customers. But before I remove the rows with no `CustomerID` I check if all invoices have exactly one customer. The following prints the number of invoices that have rows with different custumerID's or have a mix of NA's and different customerID's:

```r
onlineRetail %>%
    group_by(InvoiceNo) %>%
    summarize(numberOfCustomerIDs = length(unique(CustomerID))) %>%
    filter(numberOfCustomerIDs != 1) %>%
    nrow()
```

```
## [1] 0
```

This means that no invoices have rows with a mix of customers, or no customer, and I delete all rows without CustomerID:

```r
onlineRetail = onlineRetail[!is.na(onlineRetail$CustomerID), ]
nrow(onlineRetail)
```

```
## [1] 406829
```

And now all the empty descriptions are gone too:

```
## Description rows with missing data = 0
```

InvoiceNo

The data contains 22190 different invoice numbers, and all invoice numbers have the right format of 6 digits with or without a 'C' prefixed, as the resulting 0 in the following code shows. Nothing needs to be cleaned here:

```r
nrow(onlineRetail[!grepl("^C\\d{6}$", onlineRetail$InvoiceNo) & !grepl("^\\d{6}$", onlineRetail$Invoice
```

```
## [1] 0
```

StockCode

The data contains 3684 different stock codes. The documentation says that `StockCode` consist of 5 digits. I try to print up to ten rows where `StockCode` does not consist of exactly 5 digits:

```r
head(onlineRetail[!grepl("^\\d{5}$", onlineRetail$StockCode), ], 10)
```

```
##    InvoiceNo StockCode                         Description Quantity
## 1     536365    85123A  WHITE HANGING HEART T-LIGHT HOLDER        6
## 3     536365    84406B      CREAM CUPID HEARTS COAT HANGER        8
## 4     536365    84029G KNITTED UNION FLAG HOT WATER BOTTLE        6
## 5     536365    84029E      RED WOOLLY HOTTIE WHITE HEART.        6
## 46    536370      POST                             POSTAGE        3
```

```
## 50    536373    85123A  WHITE HANGING HEART T-LIGHT HOLDER        6
## 52    536373    84406B      CREAM CUPID HEARTS COAT HANGER        8
## 61    536373    82494L         WOODEN FRAME ANTIQUE WHITE         6
## 62    536373    84029G KNITTED UNION FLAG HOT WATER BOTTLE        6
## 63    536373    84029E      RED WOOLLY HOTTIE WHITE HEART.        6
##            InvoiceDate UnitPrice CustomerID        Country
## 1  2010-12-01 07:26:00      2.55      17850 United Kingdom
## 3  2010-12-01 07:26:00      2.75      17850 United Kingdom
## 4  2010-12-01 07:26:00      3.39      17850 United Kingdom
## 5  2010-12-01 07:26:00      3.39      17850 United Kingdom
## 46 2010-12-01 07:45:00     18.00      12583         France
## 50 2010-12-01 08:02:00      2.55      17850 United Kingdom
## 52 2010-12-01 08:02:00      2.75      17850 United Kingdom
## 61 2010-12-01 08:02:00      2.55      17850 United Kingdom
## 62 2010-12-01 08:02:00      3.39      17850 United Kingdom
## 63 2010-12-01 08:02:00      3.39      17850 United Kingdom
```

So it is seen that not all `StockCode` follow the documentation. Analysing a little deeper we see that no `StockCode` contains a lowercase letter:

```r
nrow(onlineRetail[onlineRetail$StockCode != toupper(onlineRetail$StockCode), ])
```

```
## [1] 0
```

We see, though, that the `StockCode` has a much broader format than told in the documentation. One other format is to append one or two letters to the StockCode which occurs in 33889 rows. Apparently this is often used to indicate the product color as in this example:

```r
nrow(onlineRetail[grepl("^\\d{5}[A-Z]{1,2}$", onlineRetail$StockCode), ])
```

```
## [1] 33889
```

All these products seem to be genuine and valid products and I leave them in the data.

Product codes that differ from five digits with possible one or two letters appended, looks like:

```r
head(onlineRetail[!grepl("^\\d{5}[A-Z]{0,2}$", onlineRetail$StockCode), ], 10)
```

```
##      InvoiceNo    StockCode  Description Quantity         InvoiceDate
## 46      536370         POST      POSTAGE        3 2010-12-01 07:45:00
## 142    C536379            D     Discount       -1 2010-12-01 08:41:00
## 387     536403         POST      POSTAGE        1 2010-12-01 10:27:00
## 1124    536527         POST      POSTAGE        1 2010-12-01 12:04:00
## 1424    536540           C2     CARRIAGE        1 2010-12-01 13:05:00
## 2240    536569            M       Manual        1 2010-12-01 14:35:00
## 2251    536569            M       Manual        1 2010-12-01 14:35:00
## 4407    536779 BANK CHARGES Bank Charges        1 2010-12-02 14:08:00
## 5074    536840         POST      POSTAGE        1 2010-12-02 17:27:00
## 5259    536852         POST      POSTAGE        1 2010-12-03 08:51:00
##      UnitPrice CustomerID        Country
## 46       18.00      12583         France
## 142      27.50      14527 United Kingdom
```

```
## 387       15.00      12791    Netherlands
## 1124      18.00      12662        Germany
## 1424      50.00      14911           EIRE
## 2240       1.25      16274 United Kingdom
## 2251      18.95      16274 United Kingdom
## 4407      15.00      15823 United Kingdom
## 5074      18.00      12738        Germany
## 5259      18.00      12686         France
```

When collected to a full list of all the different ones we see that they all are about non product items as handling and postage:

```r
onlineRetail[!grepl("^\\d{5}[A-Z]{0,2}$", onlineRetail$StockCode), ] %>%
    group_by(StockCode) %>%
    count(Description) %>%
    top_n(1, wt=n) %>%
    ungroup() %>%
    select(-n)
```

```
## # A tibble: 8 x 2
##   StockCode    Description
##   <chr>        <chr>
## 1 BANK CHARGES Bank Charges
## 2 C2           CARRIAGE
## 3 CRUK         CRUK Commission
## 4 D            Discount
## 5 DOT          DOTCOM POSTAGE
## 6 M            Manual
## 7 PADS         PADS TO MATCH ALL CUSHIONS
## 8 POST         POSTAGE
```

I remove them from the data because they only are non product items. It will probably not matter much because there only are 1920 invoice rows with one of these product codes:

```r
onlineRetail = onlineRetail[grepl("^\\d{5}[A-Z]{0,2}$", onlineRetail$StockCode),, ]
nrow(onlineRetail)
```

```
## [1] 404909
```

Cancelled Invoices

Invoices with `InvoiceNo` starting with a 'c' is a cancellation invoice that cancels one or more rows in other invoices. For example:

```r
head(onlineRetail[startsWith(tolower(onlineRetail$InvoiceNo), "c"), ], 10)
```

```
##      InvoiceNo StockCode                       Description Quantity
## 155    C536383    35004C   SET OF 3 COLOURED  FLYING DUCKS       -1
## 236    C536391     22556    PLASTERS IN TIN CIRCUS PARADE       -12
## 237    C536391     21984  PACK OF 12 PINK PAISLEY TISSUES       -24
## 238    C536391     21983  PACK OF 12 BLUE PAISLEY TISSUES       -24
```

```
## 239    C536391     21980 PACK OF 12 RED RETROSPOT TISSUES       -24
## 240    C536391     21484       CHICK GREY HOT WATER BOTTLE       -12
## 241    C536391     22557  PLASTERS IN TIN VINTAGE PAISLEY       -12
## 242    C536391     22553            PLASTERS IN TIN SKULLS      -24
## 940    C536506     22960          JAM MAKING SET WITH JARS       -6
## 1442   C536543     22632         HAND WARMER RED RETROSPOT       -1
##               InvoiceDate UnitPrice CustomerID        Country
## 155  2010-12-01 08:49:00      4.65      15311 United Kingdom
## 236  2010-12-01 09:24:00      1.65      17548 United Kingdom
## 237  2010-12-01 09:24:00      0.29      17548 United Kingdom
## 238  2010-12-01 09:24:00      0.29      17548 United Kingdom
## 239  2010-12-01 09:24:00      0.29      17548 United Kingdom
## 240  2010-12-01 09:24:00      3.45      17548 United Kingdom
## 241  2010-12-01 09:24:00      1.65      17548 United Kingdom
## 242  2010-12-01 09:24:00      1.65      17548 United Kingdom
## 940  2010-12-01 11:38:00      4.25      17897 United Kingdom
## 1442 2010-12-01 13:30:00      2.10      17841 United Kingdom
```

In these examples all the quantities are negative, and it can be shown that negative quantities correspond to exactly the cancellation invoices because there are no rows with a cancellation invoice with a positive quantity and no rows with a non-cancellation invoice with a negative quantity:

```r
nrow(onlineRetail[startsWith(tolower(onlineRetail$InvoiceNo), "c") ^ onlineRetail$Quantity < 0, ])
```

```
## [1] 0
```

The cancelled rows do not show which other invoice row they cancel. In some cases a probable original invoice row can be found manually, like the suspicious observation about the minimum and maximum quantity made earlier:

```r
onlineRetail[abs(onlineRetail$Quantity) >= 40000, ]
```

```
##        InvoiceNo StockCode                   Description Quantity
## 61620     541431     23166 MEDIUM CERAMIC TOP STORAGE JAR    74215
## 61625    C541433     23166 MEDIUM CERAMIC TOP STORAGE JAR   -74215
## 540422    581483     23843    PAPER CRAFT , LITTLE BIRDIE    80995
## 540423   C581484     23843    PAPER CRAFT , LITTLE BIRDIE   -80995
##                InvoiceDate UnitPrice CustomerID        Country
## 61620  2011-01-18 09:01:00      1.04      12346 United Kingdom
## 61625  2011-01-18 09:17:00      1.04      12346 United Kingdom
## 540422 2011-12-09 08:15:00      2.08      16446 United Kingdom
## 540423 2011-12-09 08:27:00      2.08      16446 United Kingdom
```

In other cases the cancelled original invoice row cannot be found. For example listing all the `onlineRetail` rows for customer 17548 show three invoices where one of them, C536391, is a cancellation but there is no rows showing that the customer ever ordered these items - perhaps because the original order was made before the first date of this data:

```r
onlineRetail[onlineRetail$CustomerID == 17548, ]
```

```
##        InvoiceNo StockCode                       Description Quantity
```

```
## 236      C536391     22556    PLASTERS IN TIN CIRCUS PARADE       -12
## 237      C536391     21984  PACK OF 12 PINK PAISLEY TISSUES       -24
## 238      C536391     21983  PACK OF 12 BLUE PAISLEY TISSUES       -24
## 239      C536391     21980 PACK OF 12 RED RETROSPOT TISSUES       -24
## 240      C536391     21484       CHICK GREY HOT WATER BOTTLE      -12
## 241      C536391     22557  PLASTERS IN TIN VINTAGE PAISLEY       -12
## 242      C536391     22553            PLASTERS IN TIN SKULLS      -24
## 165025    550755     22585          PACK OF 6 BIRDY GIFT TAGS      24
## 165026    550755     22082         RIBBON REEL STRIPES DESIGN      10
## 165027    550755     22081          RIBBON REEL FLORA + FAUNA      10
## 165028    550755     22079          RIBBON REEL HEARTS DESIGN      10
## 165029    550755     22926    IVORY GIANT GARDEN THERMOMETER        4
## 177224   C552049     22926    IVORY GIANT GARDEN THERMOMETER       -4
## 177225   C552049     22585          PACK OF 6 BIRDY GIFT TAGS     -24
## 177226   C552049     22082         RIBBON REEL STRIPES DESIGN     -10
## 177227   C552049     22081          RIBBON REEL FLORA + FAUNA     -10
## 177228   C552049     22079          RIBBON REEL HEARTS DESIGN     -10
##                InvoiceDate UnitPrice CustomerID        Country
## 236    2010-12-01 09:24:00      1.65      17548 United Kingdom
## 237    2010-12-01 09:24:00      0.29      17548 United Kingdom
## 238    2010-12-01 09:24:00      0.29      17548 United Kingdom
## 239    2010-12-01 09:24:00      0.29      17548 United Kingdom
## 240    2010-12-01 09:24:00      3.45      17548 United Kingdom
## 241    2010-12-01 09:24:00      1.65      17548 United Kingdom
## 242    2010-12-01 09:24:00      1.65      17548 United Kingdom
## 165025 2011-04-20 11:01:00      1.25      17548 United Kingdom
## 165026 2011-04-20 11:01:00      1.65      17548 United Kingdom
## 165027 2011-04-20 11:01:00      1.65      17548 United Kingdom
## 165028 2011-04-20 11:01:00      1.65      17548 United Kingdom
## 165029 2011-04-20 11:01:00      5.95      17548 United Kingdom
## 177224 2011-05-06 09:00:00      5.95      17548 United Kingdom
## 177225 2011-05-06 09:00:00      1.25      17548 United Kingdom
## 177226 2011-05-06 09:00:00      1.65      17548 United Kingdom
## 177227 2011-05-06 09:00:00      1.65      17548 United Kingdom
## 177228 2011-05-06 09:00:00      1.65      17548 United Kingdom
```

I decide not to delete cancellation invoice rows in general because the influence the quantity and the amount actually spend by each customer. However, I decide to remove two exceptional orders with Quantity larger than 70000 because they exceptionally large, they were clearly mistakes that were cancelled within 15 minutes, and I do not want them to skew the data:

```r
onlineRetail = onlineRetail[abs(onlineRetail$Quantity) < 40000, ]
```

Quantity

All quantities are integer numbers as expected as shown by the following zero count:

```r
nrow(onlineRetail[onlineRetail$Quantity != as.integer(onlineRetail$Quantity), ])
```

```
## [1] 0
```

and no quantity is zero:

```r
nrow(onlineRetail[onlineRetail$Quantity == 0, ])
```

```
## [1] 0
```

As seen earlier the lowest quantity is negative and the highest quantity is the positive of the same number. Furthermore, the two largest quantities, 80995 and 74215, are cancelled and I decide to leave the data as is.

```r
onlineRetail[2500 < abs(onlineRetail$Quantity), ]
```

```
##        InvoiceNo StockCode                         Description Quantity
## 4288     C536757     84347 ROTATING SILVER ANGELS T-LIGHT HLDR    -9360
## 4946      536830     84077   WORLD WAR 2 GLIDERS ASSTD DESIGNS     2880
## 52712     540815     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR     3114
## 80743     543057     84077   WORLD WAR 2 GLIDERS ASSTD DESIGNS     2592
## 97433     544612     22053               EMPIRE DESIGN ROSETTE     3906
## 160146   C550456     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR    -3114
## 160547    550461     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR     3114
## 201150    554272     21977  PACK OF 60 PINK PAISLEY CAKE CASES     2700
## 206122    554868     22197                SMALL POPCORN HOLDER     4300
## 270886    560599     18007 ESSENTIAL BALM 3.5g TIN IN ENVELOPE     3186
## 291250    562439     84879       ASSORTED COLOUR BIRD ORNAMENT     2880
## 421633    573008     84077   WORLD WAR 2 GLIDERS ASSTD DESIGNS     4800
## 433789    573995     16014         SMALL CHINESE STYLE SCISSOR     3000
## 502123    578841     84826      ASSTD DESIGN 3D PAPER STICKERS    12540
##                InvoiceDate UnitPrice CustomerID        Country
## 4288   2010-12-02 13:23:00      0.03      15838 United Kingdom
## 4946   2010-12-02 16:38:00      0.18      16754 United Kingdom
## 52712  2011-01-11 11:55:00      2.10      15749 United Kingdom
## 80743  2011-02-03 09:50:00      0.21      16333 United Kingdom
## 97433  2011-02-22 09:43:00      0.82      18087 United Kingdom
## 160146 2011-04-18 12:08:00      2.10      15749 United Kingdom
## 160547 2011-04-18 12:20:00      2.10      15749 United Kingdom
## 201150 2011-05-23 12:08:00      0.42      12901 United Kingdom
## 206122 2011-05-27 09:52:00      0.72      13135 United Kingdom
## 270886 2011-07-19 16:04:00      0.06      14609 United Kingdom
## 291250 2011-08-04 17:06:00      1.45      12931 United Kingdom
## 421633 2011-10-27 11:26:00      0.21      12901 United Kingdom
## 433789 2011-11-02 10:24:00      0.32      16308 United Kingdom
## 502123 2011-11-25 14:57:00      0.00      13256 United Kingdom
```

InvoiceDate

The following example of the lines making up a full invoice can have different timestamps. It looks like the rows have been keyed-in by hand and that it took a little while for each row:

```r
onlineRetail[onlineRetail$InvoiceNo == 544926, ]
```

```
##        InvoiceNo StockCode                        Description Quantity
## 101531    544926     37450  CERAMIC CAKE BOWL + HANGING CAKES        6
## 101532    544926     79321                      CHILLI LIGHTS        4
## 101533    544926    85123A WHITE HANGING HEART T-LIGHT HOLDER        6
```

```
## 101534    544926     22993       SET OF 4 PANTRY JELLY MOULDS       12
## 101535    544926     48184              DOORMAT ENGLISH ROSE         2
## 101536    544926     48185                 DOORMAT FAIRY CAKE        2
##                  InvoiceDate UnitPrice CustomerID        Country
## 101531 2011-02-24 16:50:00      2.95      13468 United Kingdom
## 101532 2011-02-24 16:50:00      5.75      13468 United Kingdom
## 101533 2011-02-24 16:50:00      2.95      13468 United Kingdom
## 101534 2011-02-24 16:50:00      1.25      13468 United Kingdom
## 101535 2011-02-24 16:50:00      7.95      13468 United Kingdom
## 101536 2011-02-24 16:51:00      7.95      13468 United Kingdom
```

The date, however, is constant on all of the invoices, as is seen from the following sorted list of the invoices having the most different dates, and they show that all invoices have at most 1 date:

```
onlineRetail %>%
    group_by(InvoiceNo) %>%
    summarize(numberOfDates = length(unique(as.Date(InvoiceDate, "%Y-%m-%d")))) %>%
    arrange(desc(numberOfDates)) %>%
    head(5)
```

```
## # A tibble: 5 x 2
##   InvoiceNo numberOfDates
##   <chr>             <int>
## 1 536365                1
## 2 536366                1
## 3 536367                1
## 4 536368                1
## 5 536369                1
```

UnitPrice

No unit prices are negative but 33 are zero:

```
onlineRetail[onlineRetail$UnitPrice <= 0, ]
```

```
##        InvoiceNo StockCode                         Description Quantity
## 9303      537197     22841        ROUND CAKE TIN VINTAGE GREEN        1
## 33577     539263     22580        ADVENT CALENDAR GINGHAM SACK        4
## 40090     539722     22423            REGENCY CAKESTAND 3 TIER       10
## 47069     540372     22090             PAPER BUNTING RETROSPOT       24
## 47071     540372     22553              PLASTERS IN TIN SKULLS       24
## 56675     541109     22168       ORGANISER WOOD ANTIQUE WHITE         1
## 86790     543599    84535B        FAIRY CAKES NOTEBOOK A6 SIZE       16
## 130189    547417     22062 CERAMIC BOWL WITH LOVE HEART DESIGN       36
## 139454    548318     22055  MINI CAKE STAND  HANGING STRAWBERY        5
## 145209    548871     22162          HEART GARLAND RUSTIC PADDED        2
## 157043    550188     22636  CHILDS BREAKFAST SET CIRCUS PARADE        1
## 187614    553000     47566                       PARTY BUNTING        4
## 198384    554037     22619            SET OF 6 SOLDIER SKITTLES       80
## 279325    561284     22167          OVAL WALL MIRROR DIAMANTE         1
## 282913    561669     22960             JAM MAKING SET WITH JARS       11
## 298055    562973     23157          SET OF 6 NATIVITY MAGNETS        240
```

```
## 314746    564651     23270    SET OF 2 CERAMIC PAINTED HEARTS        96
## 314747    564651     23268 SET OF 2 CERAMIC CHRISTMAS REINDEER      192
## 314748    564651     22955           36 FOIL STAR CAKE CASES       144
## 314749    564651     21786                 POLKADOT RAIN HAT       144
## 379914    569716     22778                 GLASS CLOCHE SMALL        2
## 420405    572893     21208        PASTEL COLOUR HONEYCOMB FAN        5
## 436429    574138     23234      BISCUIT TIN VINTAGE CHRISTMAS      216
## 436598    574175     22065     CHRISTMAS PUDDING TRINKET POT        12
## 439362    574469     22385          JUMBO BAG SPACEBOY DESIGN       12
## 446126    574879     22625                 RED KITCHEN SCALES        2
## 446794    574920     22899       CHILDREN'S APRON DOLLY GIRL         1
## 446795    574920     23480     MINI LIGHTS WOODLAND MUSHROOMS        1
## 454464    575579     22437      SET OF 9 BLACK SKULL BALLOONS       20
## 454465    575579     22089      PAPER BUNTING VINTAGE PAISLEY       24
## 479080    577129     22464        HANGING METAL HEART LANTERN        4
## 480650    577314     23407     SET OF 2 TRAYS HOME SWEET HOME        2
## 502123    578841     84826     ASSTD DESIGN 3D PAPER STICKERS    12540
##                InoiceDate UnitPrice CustomerID        Country
## 9303   2010-12-05 13:02:00         0      12647        Germany
## 33577  2010-12-16 13:36:00         0      16560 United Kingdom
## 40090  2010-12-21 12:45:00         0      14911           EIRE
## 47069  2011-01-06 15:41:00         0      13081 United Kingdom
## 47071  2011-01-06 15:41:00         0      13081 United Kingdom
## 56675  2011-01-13 14:10:00         0      15107 United Kingdom
## 86790  2011-02-10 12:08:00         0      17560 United Kingdom
## 130189 2011-03-23 09:25:00         0      13239 United Kingdom
## 139454 2011-03-30 11:45:00         0      13113 United Kingdom
## 145209 2011-04-04 13:42:00         0      14410 United Kingdom
## 157043 2011-04-14 17:57:00         0      12457    Switzerland
## 187614 2011-05-12 14:21:00         0      17667 United Kingdom
## 198384 2011-05-20 13:13:00         0      12415      Australia
## 279325 2011-07-26 11:24:00         0      16818 United Kingdom
## 282913 2011-07-28 16:09:00         0      12507          Spain
## 298055 2011-08-11 10:42:00         0      14911           EIRE
## 314746 2011-08-26 13:19:00         0      14646    Netherlands
## 314747 2011-08-26 13:19:00         0      14646    Netherlands
## 314748 2011-08-26 13:19:00         0      14646    Netherlands
## 314749 2011-08-26 13:19:00         0      14646    Netherlands
## 379914 2011-10-06 07:17:00         0      15804 United Kingdom
## 420405 2011-10-26 13:36:00         0      18059 United Kingdom
## 436429 2011-11-03 10:26:00         0      12415      Australia
## 436598 2011-11-03 10:47:00         0      14110 United Kingdom
## 439362 2011-11-04 10:55:00         0      12431      Australia
## 446126 2011-11-07 12:22:00         0      13014 United Kingdom
## 446794 2011-11-07 15:34:00         0      13985 United Kingdom
## 446795 2011-11-07 15:34:00         0      13985 United Kingdom
## 454464 2011-11-10 10:49:00         0      13081 United Kingdom
## 454465 2011-11-10 10:49:00         0      13081 United Kingdom
## 479080 2011-11-17 18:52:00         0      15602 United Kingdom
## 480650 2011-11-18 12:23:00         0      12444         Norway
## 502123 2011-11-25 14:57:00         0      13256 United Kingdom
```

Apparently the company gives away free stuff from time to time. If the items were actually ordered by the customer and then given free to the customer I would want them to the stay in the data to reflect the customers intention. If the items were not ordered by the customer but just given as a free promotional gift they say nothing about the customer and I would delete them from the data. The problem is that I do not know which one is true. Because there are only 33 invoice rows where the unit price is zero I decide to leave them in the data.

CustomerID

The data contains 4362 different customer ID's. The customer ID's are all five digit integer as said in the documentation of the data and there is no need to clean-up anything here:

```r
nrow(onlineRetail[onlineRetail$CustomerID != as.integer(onlineRetail$CustomerID), ])
```

```
## [1] 0
```

```r
nrow(onlineRetail[onlineRetail$CustomerID < 10000 | 99999 < onlineRetail$CustomerID, ])
```

```
## [1] 0
```

Each invoice row has a customer and it could be that invoice rows making up a full invoice would have several different customers. This is not so, however, and it is easy to see, that all invoices have no more than one customer. The following is a sorted list of the invoices having the most different customers, and they show that all invoices have at most 1 customer:

```r
onlineRetail %>%
    group_by(InvoiceNo) %>%
    summarize(numberOfCustomers = length(unique(CustomerID))) %>%
    arrange(desc(numberOfCustomers)) %>%
    head(5)
```

```
## # A tibble: 5 x 2
##   InvoiceNo numberOfCustomers
##   <chr>                 <int>
## 1 536365                    1
## 2 536366                    1
## 3 536367                    1
## 4 536368                    1
## 5 536369                    1
```

Country

The countries include an 'Unspecified' and all seems Ok except that it also includes 'European Community' that includes several of the other countries:

```r
unique(onlineRetail$Country)
```

```
##  [1] "United Kingdom"       "France"               "Australia"
##  [4] "Netherlands"          "Germany"              "Norway"
##  [7] "EIRE"                 "Switzerland"          "Spain"
## [10] "Poland"               "Portugal"             "Italy"
## [13] "Belgium"              "Lithuania"            "Japan"
```

```
## [16] "Iceland"             "Channel Islands"      "Denmark"
## [19] "Cyprus"              "Sweden"               "Austria"
## [22] "Israel"              "Finland"              "Greece"
## [25] "Singapore"           "Lebanon"              "United Arab Emirates"
## [28] "Saudi Arabia"        "Czech Republic"       "Canada"
## [31] "Unspecified"         "Brazil"               "USA"
## [34] "European Community"  "Bahrain"              "Malta"
## [37] "RSA"
```

It it seen, though, that only one customer (with four invoices) has 'European Community' as country and I decide to leave it in the data:

```
onlineRetail[onlineRetail$Country == 'European Community', ] %>%
    distinct(InvoiceNo, CustomerID)
```

```
##   InvoiceNo CustomerID
## 1    551013      15108
## 2    555542      15108
## 3   C556294      15108
## 4    560783      15108
```

The Data Types

Finally, I change some of the fields from being strings or numerics to be categorial data. For example the `CustomerID` that is read as a number from the original data, but it of course make no sense to do arithmetic on customer ID's, you cannot add two customer ID's for example. And all in all I end up with the following data ready to be analysed:

```
## Observations: 404,905
## Variables: 8
## $ InvoiceNo   <fct> 536365, 536365, 536365, 536365, 536365, 536365, 53...
## $ StockCode   <fct> 85123A, 71053, 84406B, 84029G, 84029E, 22752, 2173...
## $ Description <chr> "WHITE HANGING HEART T-LIGHT HOLDER", "WHITE METAL...
## $ Quantity    <int> 6, 6, 8, 6, 6, 2, 6, 6, 6, 32, 6, 6, 8, 6, 6, 3, 2...
## $ InvoiceDate <dttm> 2010-12-01 07:26:00, 2010-12-01 07:26:00, 2010-12...
## $ UnitPrice   <dbl> 2.55, 3.39, 2.75, 3.39, 3.39, 7.65, 4.25, 1.85, 1....
## $ CustomerID  <fct> 17850, 17850, 17850, 17850, 17850, 17850, 17850, 1...
## $ Country     <fct> UnitedKingdom, UnitedKingdom, UnitedKingdom, Unite...
```

Sizes after Cleaning the Data

After cleaning the dataset it contains 21784 different invoices and 4362 different customers.

Part 1: Analysis of Customer RFM and Customer Segments

Introduction

RFM is an often used way to measure the value of a customer. The three letters stand for **R**ecency, **F**requency, and **M**onetary Value. For the dataset `onlineRetail` I use the following:

- Recency: the number of days before the last date of the period
- Frequency: the number of different invoices and cancellations over the period
- Monetary Value: the total amount paid over the period (does take cancellations inside the period into account)

That is, I take the full year-long dataset as the period to compute for. In a real business setting you would get new data every day, and you could periodically recalculate the results I get in this report on data for the last year or two.

My goal is to cluster the customers from their RFM, first using a traditional, fixed algorithm, and second using a machine learning clustering algorithm and finally compare the two customer segmentations.

Calculate RFM for each Customer

I build a new dataset `CustomerRFM` that contains the RFM values for each customer from the `onlineRetail` dataset:

```
RecencyDate = as.Date(max(onlineRetail$InvoiceDate))
CustomerRFM <- onlineRetail %>%
    group_by(CustomerID) %>%
    summarize(Recency = as.numeric(RecencyDate - as.Date(max(InvoiceDate))),
              Frequency = n_distinct(InvoiceNo),
              Monetary = sum(Quantity * UnitPrice))
head(CustomerRFM)
```

```
## # A tibble: 6 x 4
##   CustomerID Recency Frequency Monetary
##   <fct>        <dbl>     <int>    <dbl>
## 1 12347            2         7    4310
## 2 12348           75         4    1437.
## 3 12349           18         1    1458.
## 4 12350          310         1     294.
## 5 12352           36         8    1265.
## 6 12353          204         1      89
```

```
summary(CustomerRFM)
```

```
##    CustomerID      Recency         Frequency          Monetary
##   12347   :    1   Min.   :  0.00   Min.   :  1.000   Min.   : -1192.2
##   12348   :    1   1st Qu.: 16.00   1st Qu.:  1.000   1st Qu.:   294.3
##   12349   :    1   Median : 50.00   Median :  3.000   Median :   645.7
##   12350   :    1   Mean   : 91.68   Mean   :  4.994   Mean   :  1899.7
##   12352   :    1   3rd Qu.:143.00   3rd Qu.:  5.000   3rd Qu.:  1596.4
##   12353   :    1   Max.   :373.00   Max.   :242.000   Max.   :278778.0
##   (Other):4356
```

I need to scale the values before using them. If for example the first feature in a dataset ranges from 0 to 1 and the second feature ranges from 0 to 1000, the computed Euclidean distance between two points, or rows, from the dataset will be nearly totally dominated by the second feature. To avoid this problem the features must be scaled to the same ranges before using them in a clustering algorithm.

I use the following to scale the three feature R, F, and M. The method I use assumes that the values in each feature is close to be normal distributed and scale the values to have mean 0 and standard deviation 1. The values in `CustomerRFM` are not normally distributed but scaling this way is much better than not scaling:

```r
CustomerRFM$Recency <- scale(CustomerRFM$Recency)
CustomerRFM$Frequency <- scale(CustomerRFM$Frequency)
CustomerRFM$Monetary <- scale(CustomerRFM$Monetary)
head(CustomerRFM)
```

```
## # A tibble: 6 x 4
##   CustomerID Recency[,1] Frequency[,1] Monetary[,1]
##   <fct>            <dbl>         <dbl>        <dbl>
## 1 12347           -0.889         0.221       0.292
## 2 12348           -0.165        -0.109      -0.0560
## 3 12349           -0.730        -0.440      -0.0535
## 4 12350            2.16         -0.440      -0.194
## 5 12352           -0.552         0.331      -0.0767
## 6 12353            1.11         -0.440      -0.219
```

And we end as expected up with scaled values for each feature:

```
## [1] "Recency:   mean = -0.000000, sd = 1.000000"
```

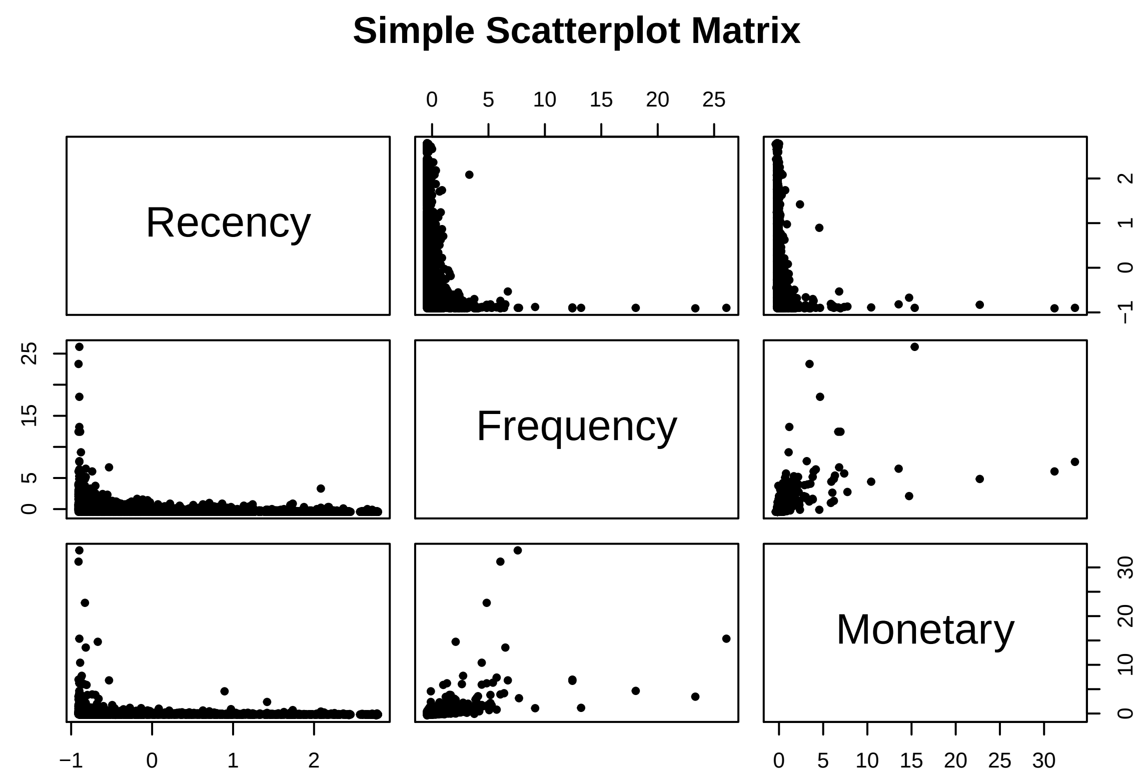
```
## [1] "Frequency: mean = 0.000000, sd = 1.000000"
```

```
## [1] "Monetary:  mean = 0.000000, sd = 1.000000"
```

Visually Illustrating the Values and Principal Component Analysis

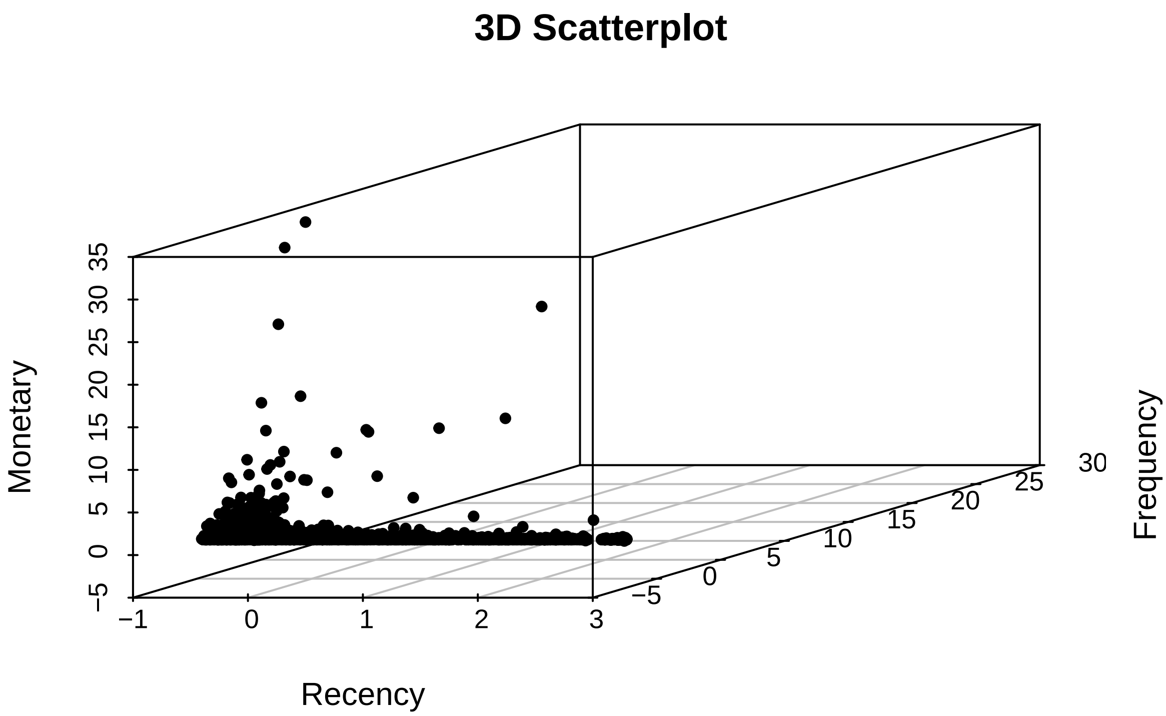
In the following plot it is seen that all three features are pairwise correlated but not in a simple, for example, linear way. Recency, though, seems to be correlated in somewhat the same way to both Frequency and Monetary which suggests that much of the variability in the data can be expressed with only two features, if we can find the suitable features.

```r
pairs(~Recency+Frequency+Monetary,data=CustomerRFM,
    main="Simple Scatterplot Matrix", pch = 20)
```



To illustrate the RFM values for the customers I first draw a three-dimensional plot where each point is a customer and each of the three coordinates is one of the three RFM features such that the points position is the RFM values of that customer.

```
scatterplot3d(CustomerRFM$Recency, CustomerRFM$Frequency, CustomerRFM$Monetary,
              pch = 20,
              main="3D Scatterplot",
              xlab = "Recency", ylab = "Frequency", zlab = "Monetary")
```



It is worth noting that a big part of the space is empty. For example there are no customers that buy frquently but just not recently. A possible explanation is that freqency and recency is connected in that if a customer has not bought in a while then the frequency falls.

It is hard, at least for me, to see the exact three-dimensional positions of the points in this three-dimensional plot. It would be much easier to look at in a turnable, and possibly interactive, three-dimensional plot but this is not possible in the pdf format of this report.

So in addition to the three dimensional plot I also illustrate the positions of the points in a two-dimensional plot where I map all the points from three dimensions into two dimensions using Principal Component Analysis (PCA). PCA computes a new three-dimensional coordinate system where each coordinate, in order, covers as much variability as possible. So dropping the least important coordinate after PCA will usually give a much better result, and never be worse, than just dropping one of the coordinates without PCA.

```
CustomerRFM_PCA <- prcomp(CustomerRFM[c(2,3,4)])
summary(CustomerRFM_PCA)
```

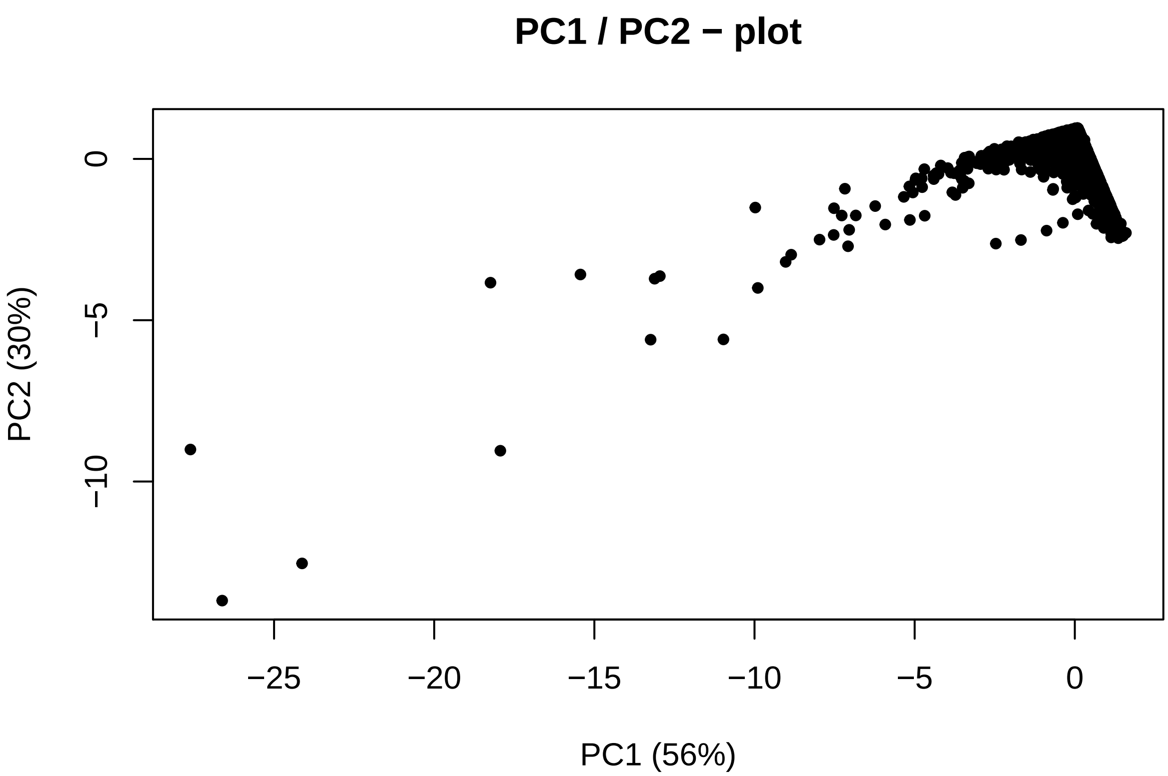
```
## Importance of components:
##                           PC1    PC2    PC3
## Standard deviation     1.2928 0.9498 0.6532
## Proportion of Variance 0.5571 0.3007 0.1422
## Cumulative Proportion  0.5571 0.8578 1.0000
```

The three new dimensions are sorted by the amount of variability of the data in the coordinates direction (*standard deviation*). It is seen that the two first coordinates together explain close to 86% of the variance of the data, which means that using the first two coordinates will give a reasonable illustration of the customers

RFM values. However, it is an approximation only and there is no need to reduce the number of features from three to two in the further calculations. So I use the PCA for illustration only.

In the following plot it is easy to see, that most of the customers clusters together in one big angular formed cluster with fewer customers radiating out of the big cluster. I had hoped that a small number, five to ten, clusters formed by the positions of the points, i.e. customers, were clearly visible. This is however not the case and any customer segmentation will only be a rough and simplifying description of the much more continuously distributed customers.

```
plot(CustomerRFM_PCA$x[,1], CustomerRFM_PCA$x[,2],
     xlab="PC1 (56%)", ylab = "PC2 (30%)",
     main = "PC1 / PC2 - plot",
     pch = 20, col = "black")
```



Customer Segmentation using a Traditional Fixed Method

The Internet contains many sources that explain their way to split customers into Customer Segment using their RFM values. While there seems to be many different ways to segment customers based on RFM, all I have found is based on the same principles and differ only on the details and on the definition of the final customer segments. I use the process from https://www.putler.com/rfm-analysis/ because their method is close to all other methods that I found and they describe it detailed.

First a score 1 to 5 is assigned to each of R, F, and M using quintiles, i.e. it scores 5 if the value is ranked in the best 20%, it scores 4 if ranked in the best 20-40%, etc. For R lower values are best, for F and M higher values are best. I do not need to scale the original RFM values but because the rank is not changed by scaling it does not matter and I use the scaled `CustomerRFM` that I already have build.

```r
n = nrow(CustomerRFM)
CustomerRFM$RecencyScore   <- as.integer(1 + 5 * (1 - rank(CustomerRFM$Recency) / n))
CustomerRFM$FrequencyScore <- as.integer(1 + 5 * rank(CustomerRFM$Frequency) / n)
CustomerRFM$MonetaryScore  <- as.integer(1 + 5 * rank(CustomerRFM$Monetary) / n)
head(CustomerRFM)
```

```
## # A tibble: 6 x 7
##   CustomerID Recency[,1] Frequency[,1] Monetary[,1] RecencyScore
##   <fct>            <dbl>         <dbl>        <dbl>        <int>
## 1 12347           -0.889         0.221       0.292             5
## 2 12348           -0.165        -0.109      -0.0560            2
## 3 12349           -0.730        -0.440      -0.0535            4
## 4 12350            2.16         -0.440      -0.194             1
## 5 12352           -0.552         0.331      -0.0767            3
## 6 12353            1.11         -0.440      -0.219             1
## # ... with 2 more variables: FrequencyScore <int>, MonetaryScore <int>
```

The customer are now assigned to one of eleven customer segments dependent on their R, F, and M scores. For example customers with R, F, and M scores of 5, 5, and 5 are in the Champions customer segment - they bought recently, buy often and spend a lot. The eleven customer segments are called: “Champions”, “Loyal Customers”, “Potential Loyalist”, “Recent Customers”, “Promising”, “Customers Needing Attention”, “About To Sleep”, “At Risk”, “Can’t Lose Them”, “Hibernating”, “Lost”.

Most of the eleven customer segments combine different R, F, and M scores. The following function computes them all:

```r
putlerSegment <- function(row) {
    fm <- as.integer((as.integer(row["FrequencyScore"]) + as.integer(row["MonetaryScore"])) / 2) - 1
    idx = 1 + 5 * (as.integer(row["RecencyScore"]) - 1) + fm
    switch(idx,
        "Lost",
        "Lost",
        "AtRisk",
        "AtRisk",
        "CantLoseThem",
        "Lost",
        "Hibernating",
        "AtRisk",
        "AtRisk",
        "AtRisk",
        "AboutToSleep",
        "AboutToSleep",
        "NeedingAttention",
        "LoyalCustomers",
        "LoyalCustomers",
        "Promising",
        "PotentialLoyalist",
        "PotentialLoyalist",
        "LoyalCustomers",
        "LoyalCustomers",
        "RecentCustomers",
        "PotentialLoyalist",
        "PotentialLoyalist",
```

```r
        "LoyalCustomers",
        "Champions")
}
```

and after computing the Putler Customer Segments the data looks like:

```r
CustomerRFM$putlerSegment <- apply(CustomerRFM, 1, putlerSegment)
head(CustomerRFM)
```

```
## # A tibble: 6 x 8
##   CustomerID Recency[,1] Frequency[,1] Monetary[,1] RecencyScore
##   <fct>            <dbl>         <dbl>        <dbl>        <int>
## 1 12347           -0.889         0.221       0.292             5
## 2 12348           -0.165        -0.109      -0.0560            2
## 3 12349           -0.730        -0.440      -0.0535            4
## 4 12350            2.16         -0.440      -0.194             1
## 5 12352           -0.552         0.331      -0.0767            3
## 6 12353            1.11         -0.440      -0.219             1
## # ... with 3 more variables: FrequencyScore <int>, MonetaryScore <int>,
## #   putlerSegment <chr>
```

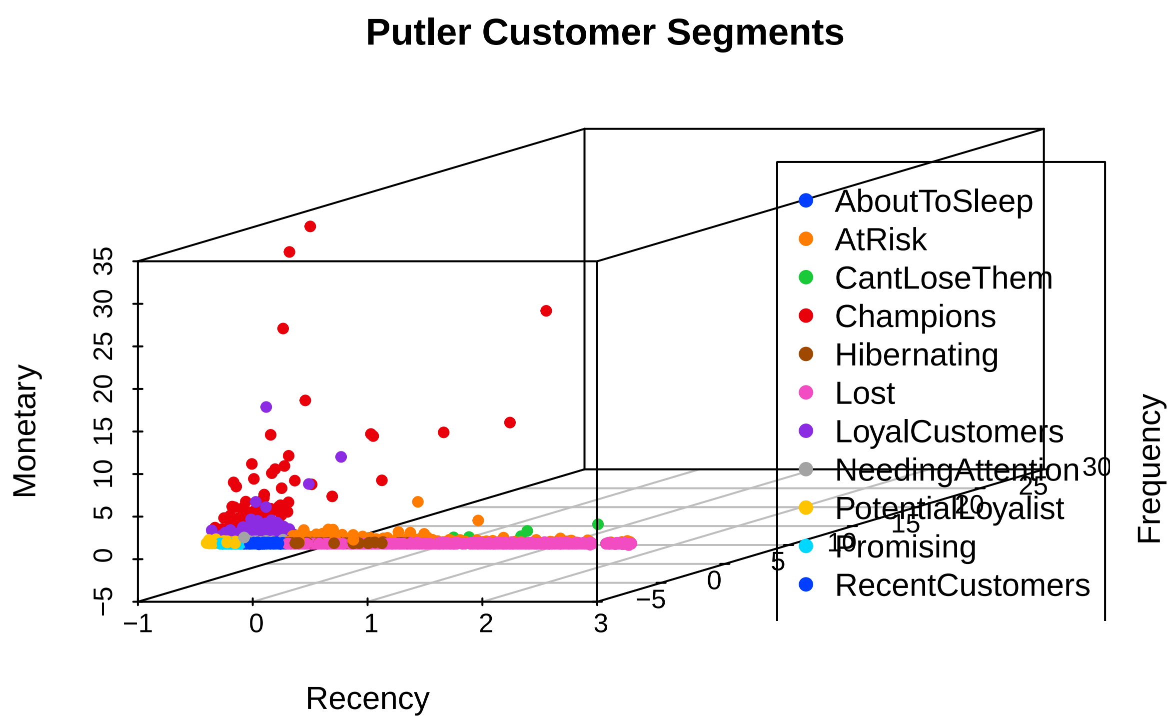
Cleaning up the sums and balances:

```r
CustomerRFM$RecencyScore <- NULL
CustomerRFM$FrequencyScore <- NULL
CustomerRFM$MonetaryScore <- NULL
CustomerRFM$putlerSegment <- as.factor(CustomerRFM$putlerSegment)

head(CustomerRFM)
```

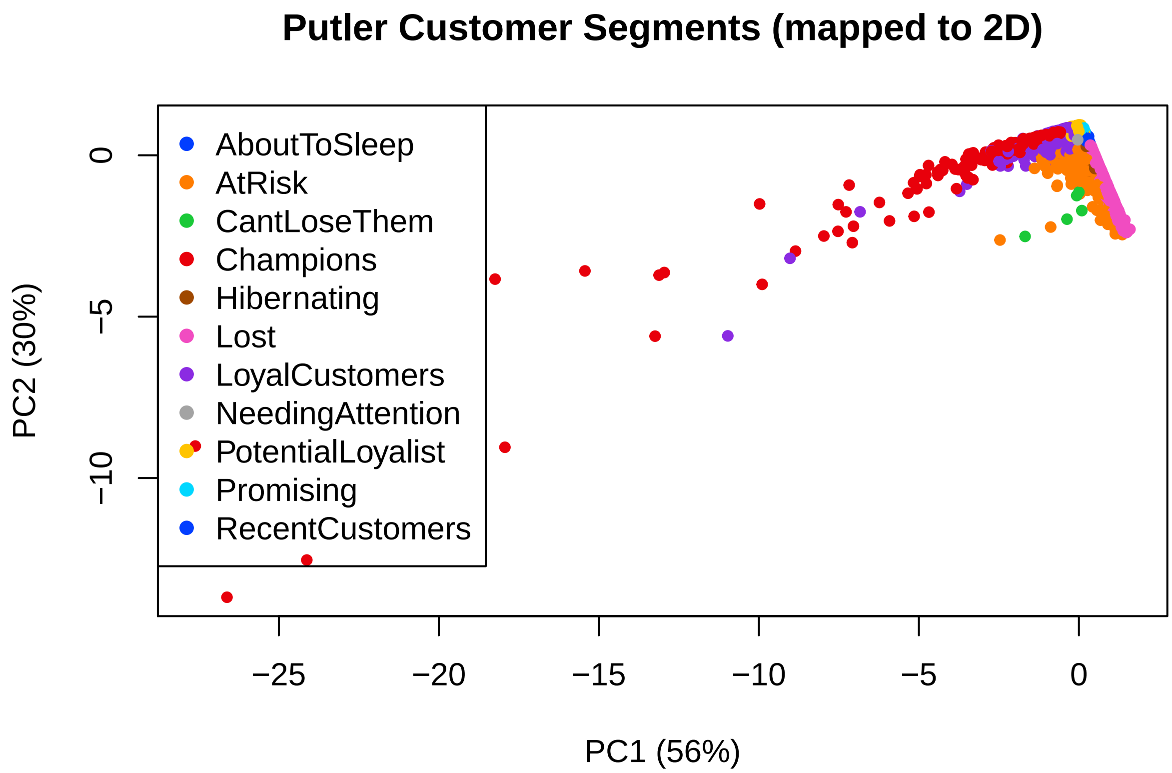
```
## # A tibble: 6 x 5
##   CustomerID Recency[,1] Frequency[,1] Monetary[,1] putlerSegment
##   <fct>            <dbl>         <dbl>        <dbl> <fct>
## 1 12347           -0.889         0.221       0.292  Champions
## 2 12348           -0.165        -0.109      -0.0560 AtRisk
## 3 12349           -0.730        -0.440      -0.0535 PotentialLoyalist
## 4 12350            2.16         -0.440      -0.194  Lost
## 5 12352           -0.552         0.331      -0.0767 LoyalCustomers
## 6 12353            1.11         -0.440      -0.219  Lost
```

The following two plots illustrate the customers Putler segments with each segment being a different color:

```r
colors <- colorpalette[CustomerRFM$putlerSegment]

plot3d <- scatterplot3d(CustomerRFM$Recency, CustomerRFM$Frequency, CustomerRFM$Monetary,
                        xlab = "Recency", ylab = "Frequency", zlab = "Monetary",
                        main="Putler Customer Segments", color = colors, pch = 20)
legend(plot3d$xyz.convert(2.9, 10, 40),
       legend = levels(CustomerRFM$putlerSegment),
       col = colorpalette, pch = 16)
```




```r
colors <- colorpalette[CustomerRFM$putlerSegment]

plot(CustomerRFM_PCA$x[,1], CustomerRFM_PCA$x[,2],
     xlab="PC1 (56%)", ylab = "PC2 (30%)",
     main = "Putler Customer Segments (mapped to 2D)",
     pch = 20, col = colors)
legend("topleft", legend = levels(CustomerRFM$putlerSegment),
       col = colorpalette, pch = 16)
```



The plots show that the Putler segments seems to differ a lot in how many customers are in each segment and especially the Champions segment seems to cover a much larger area than the other segments.

The number of customers in each Putler segment is:

```
CustomerRFM %>%
    group_by(putlerSegment) %>%
    summarize(NumberOfCustomers = n()) %>%
    arrange(desc(NumberOfCustomers))
```

```
## # A tibble: 11 x 2
##    putlerSegment     NumberOfCustomers
##    <fct>                         <int>
##  1 Lost                           1020
##  2 LoyalCustomers                  898
##  3 PotentialLoyalist               548
##  4 AtRisk                          465
##  5 AboutToSleep                    435
##  6 Champions                       354
##  7 Hibernating                     254
##  8 NeedingAttention                162
##  9 Promising                       155
## 10 RecentCustomers                  66
## 11 CantLoseThem                      5
```

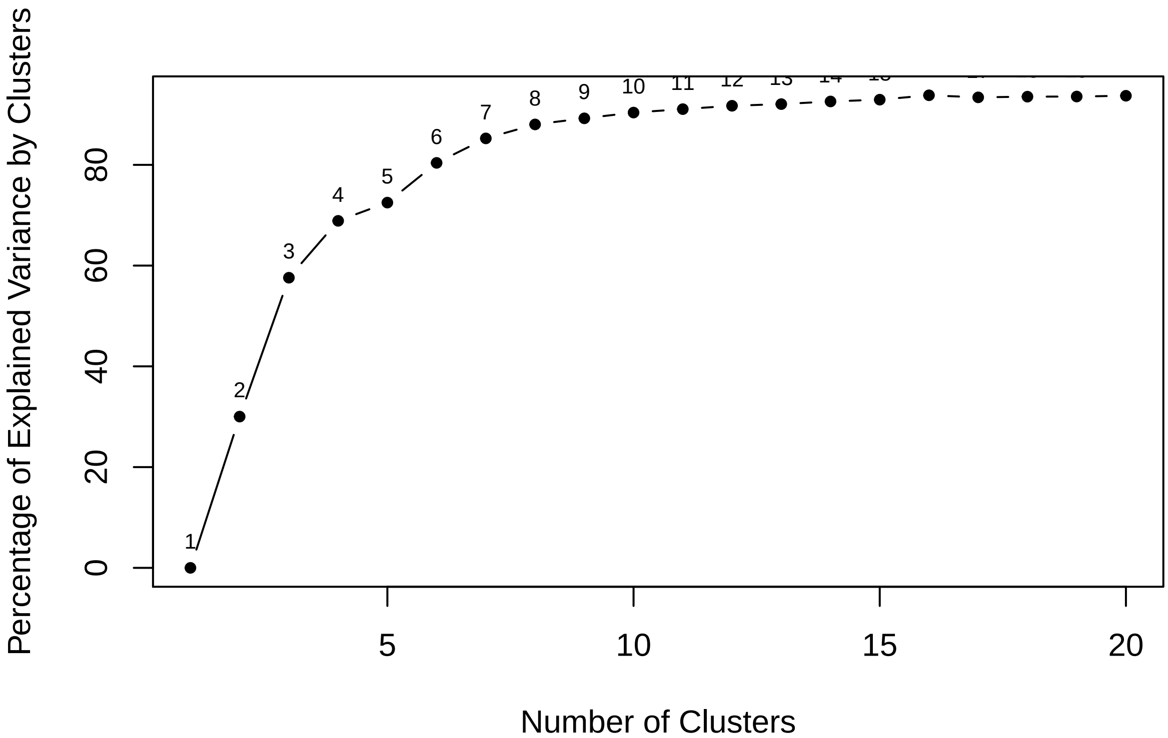
Customer Segmentation using K-means Machine Learning Clustering Method

Clustering is an unsupervised machine learning technique which means that no examples of a correct solution are given beforehand. Instead the goal is to find clusters in the given data using some measure of distance. Several different algorithms exist to find clusters. I choose to use the K-means algorithm with Euclidean distance because it is easy to understand and fast enough for the use here.

K-means starts by selecting k cluster centers randomly or in some predefined pattern. In step 1 all the data points are assigned to its closest cluster center. In step 2 the cluster centers are moved to the mean of all the data points in the cluster. Step1 and 2 are repeated until the cluster centers have converged or a maximal predefined number of steps have been executed. Actually there are implementations of several variations of the basic algorithm but I just use the default.

First I want to find the optimal number of clusters to use. The goal is to have high similarity within each cluster and low similarity across clusters and I use the socalled elbow method where I plot the the percentage of explained variance against the number of clusters. At a certain point adding one more cluster does not increase the percentage of explained variance as much as before. This is the socalled elbow point and the number of clusters at the elbow point is optimal in this sense.

```
set.seed(9)
pExplainedVar = rep(0,20)
df <- select(CustomerRFM, Recency, Frequency, Monetary)
for (i in 2:20) {
    fit = kmeans(df, centers = i, iter.max = 15, nstart = 20)
    pExplainedVar[i] = 100 * fit$betweenss / fit$totss
}

plot(1:20, pExplainedVar, type="b", pch = 20, xlab="Number of Clusters", ylab="Percentage of Explained V
text(1:20, pExplainedVar, labels = 1:20, cex = 0.7, pos = 3)
```



From the above plot I think that the flattening-out of the increase begins somewhere around seven or eight and I choose to continue to find seven clusters using the K-means algorithm.

```
set.seed(9)
df <- select(CustomerRFM, Recency, Frequency, Monetary)
k_result <- kmeans(df, centers = 7, iter.max = 15, nstart = 25)

k_result$centers
```

```
##      Recency  Frequency    Monetary
## 1 -0.7583631  1.3418178  0.41970033
## 2 -0.5613750 -0.1462504 -0.09410573
## 3 -0.8054969  4.0383745  4.18725744
## 4 -0.8787235  6.1646198 29.13352530
## 5  2.0342762 -0.3861998 -0.18589284
## 6  0.6569728 -0.2858412 -0.14727425
## 7 -0.9001920 17.5936359  6.38494369
```

Adding the found clusters to the `CustomerRFM` and draw the same plots as for the Putler segments:

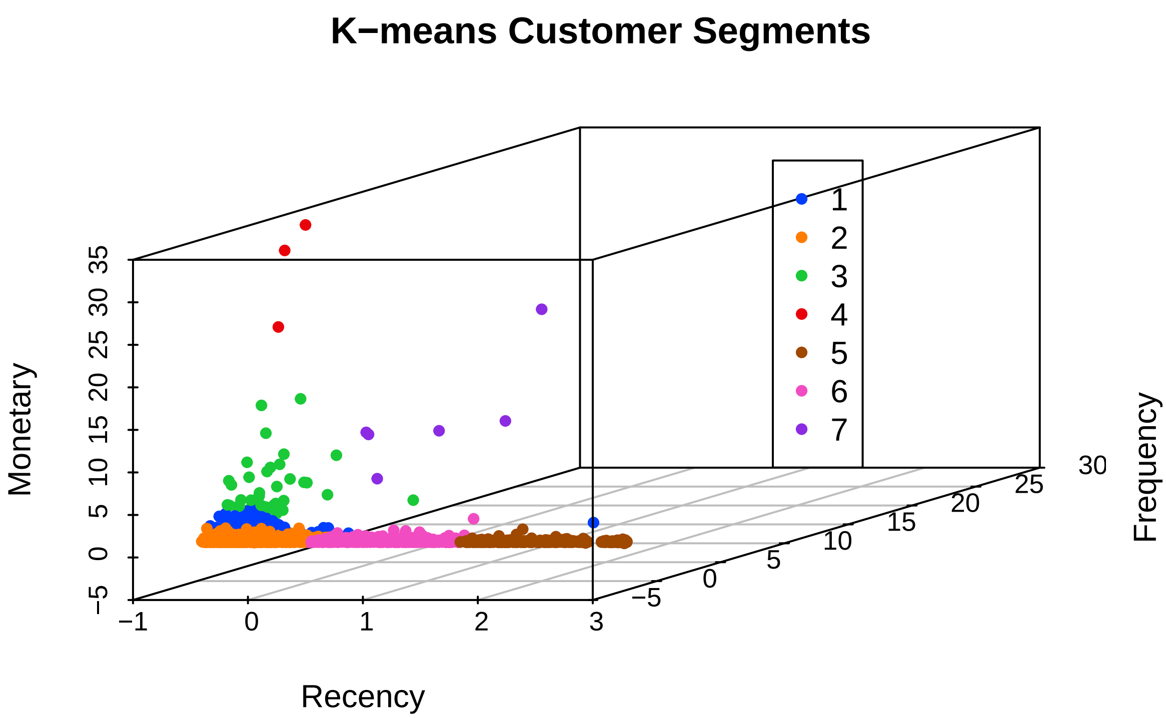
```
CustomerRFM$kmeansSegment <- as.factor(k_result$cluster)
head(CustomerRFM)
```

```
## # A tibble: 6 x 6
##   CustomerID Recency[,1] Frequency[,1] Monetary[,1] putlerSegment
```

```
##   <fct>             <dbl>         <dbl>        <dbl> <fct>
## 1 12347            -0.889         0.221        0.292  Champions
## 2 12348            -0.165        -0.109       -0.0560 AtRisk
## 3 12349            -0.730        -0.440       -0.0535 PotentialLoy~
## 4 12350             2.16         -0.440       -0.194  Lost
## 5 12352            -0.552         0.331       -0.0767 LoyalCustome~
## 6 12353             1.11         -0.440       -0.219  Lost
## # ... with 1 more variable: kmeansSegment <fct>
```

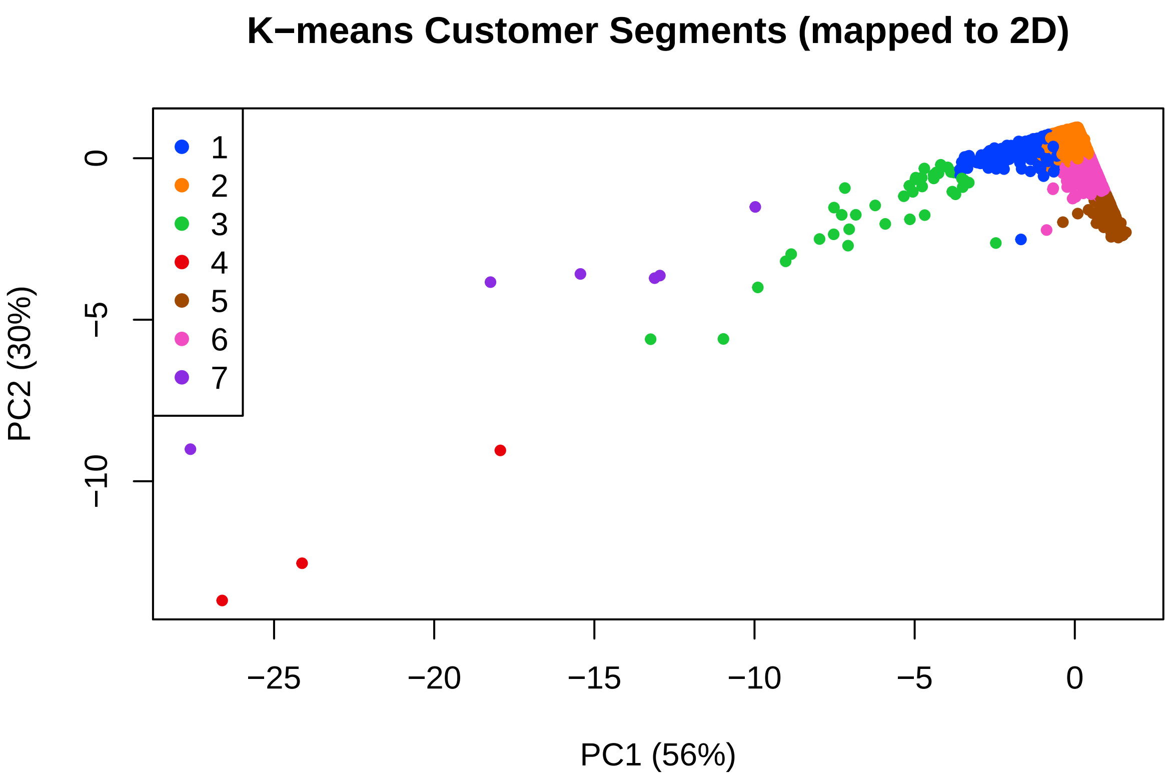
The following two plots illustrate the customers K-means segments with each segment being a different color:

```
colors <- colorpalette[CustomerRFM$kmeansSegment]

plot3d <- scatterplot3d(CustomerRFM$Recency, CustomerRFM$Frequency, CustomerRFM$Monetary,
                        xlab = "Recency", ylab = "Frequency", zlab = "Monetary",
                        main="K-means Customer Segments", color = colors, pch = 20)
legend(plot3d$xyz.convert(2.9, 10, 40),
       legend = levels(CustomerRFM$kmeansSegment),
       col = colorpalette, pch = 20)
```




```
colors <- colorpalette[CustomerRFM$kmeansSegment]

plot(CustomerRFM_PCA$x[,1], CustomerRFM_PCA$x[,2],
     xlab="PC1 (56%)", ylab = "PC2 (30%)",
     main = "K-means Customer Segments (mapped to 2D)",
```

```
      pch = 20, col = colors)
legend("topleft", legend = levels(CustomerRFM$kmeansSegment),
        col = colorpalette, pch = 16)
```



The plots show that the K-means segments seems like the Putler segments to differ a lot in how many customers are in each segment and in how large area they cover.

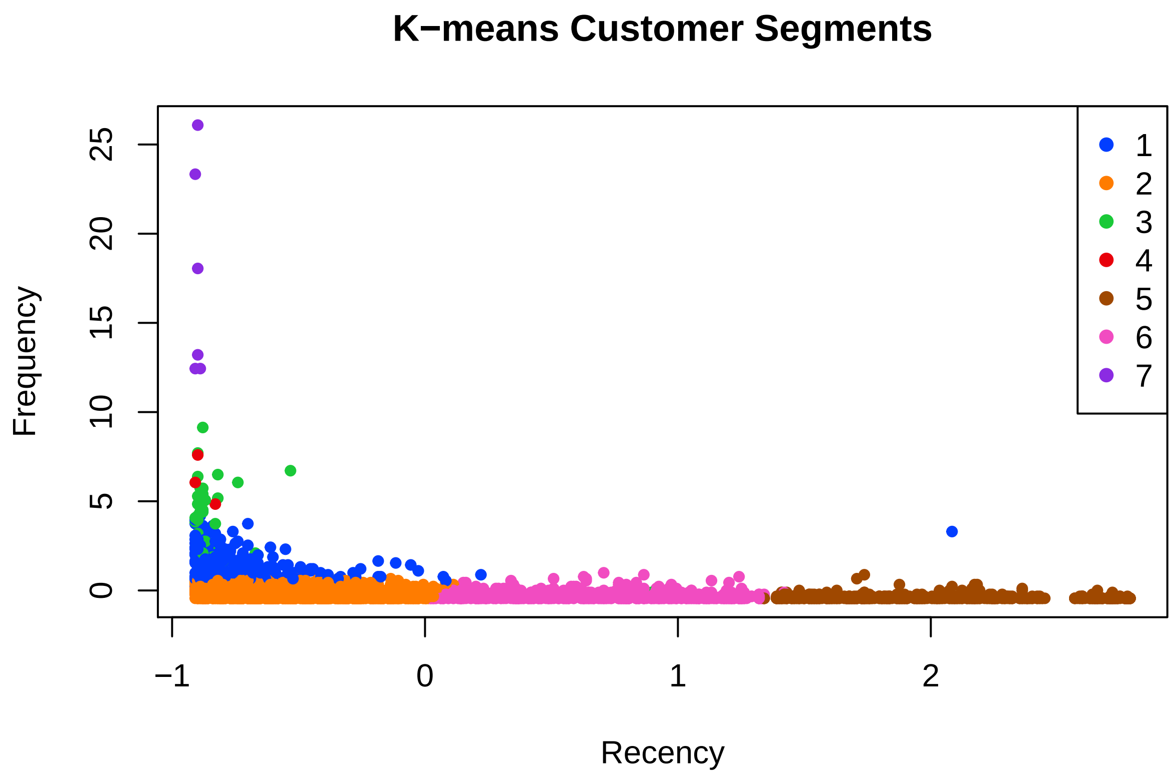
The number of customers in each K-means segment is:

```
CustomerRFM %>%
    group_by(kmeansSegment) %>%
    summarize(NumberOfCustomers = n()) %>%
    arrange(desc(NumberOfCustomers))
```

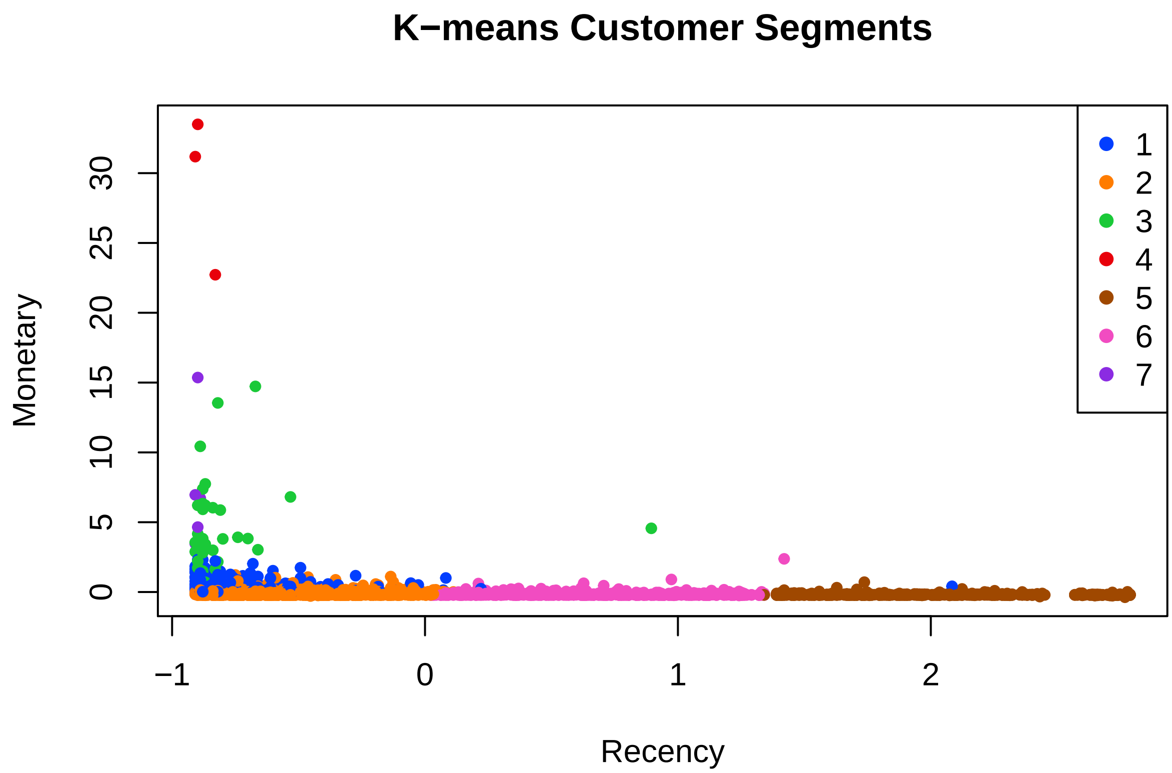
```
## # A tibble: 7 x 2
##   kmeansSegment NumberOfCustomers
##   <fct>                     <int>
## 1 2                          2522
## 2 6                           773
## 3 5                           616
## 4 1                           401
## 5 3                            41
## 6 7                             6
## 7 4                             3
```

To be useful we need a business intepretation of the seven K-means clusters. I make that by looking at three projections of the three dimesional plot of the K-means clusters by leaving out one of the coordinates.

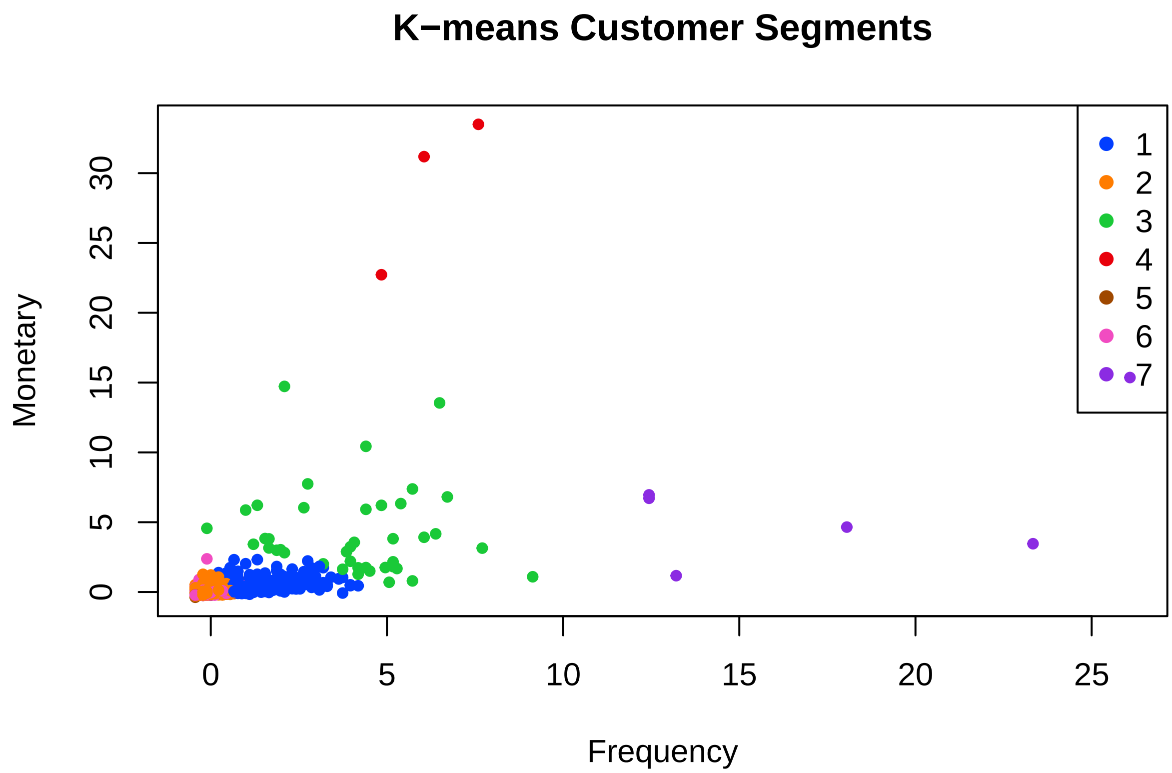
```
colors <- colorpalette[CustomerRFM$kmeansSegment]
plot(CustomerRFM$Recency, CustomerRFM$Frequency,
                        xlab = "Recency", ylab = "Frequency",
                        main="K-means Customer Segments", col = colors, pch = 20)
legend("topright", legend = levels(CustomerRFM$kmeansSegment),
       col = colorpalette, pch = 16)
```




```
colors <- colorpalette[CustomerRFM$kmeansSegment]
plot(CustomerRFM$Recency, CustomerRFM$Monetary,
                        xlab = "Recency", ylab = "Monetary",
                        main="K-means Customer Segments", col = colors, pch = 20)
legend("topright", legend = levels(CustomerRFM$kmeansSegment),
       col = colorpalette, pch = 16)
```




```
colors <- colorpalette[CustomerRFM$kmeansSegment]
plot(CustomerRFM$Frequency, CustomerRFM$Monetary,
                            xlab = "Frequency", ylab = "Monetary",
                            main="K-means Customer Segments", col = colors, pch = 20)
legend("topright", legend = levels(CustomerRFM$kmeansSegment),
       col = colorpalette, pch = 16)
```



From looking at the three two-dimsional projections of the K-means clusters I suggest the following description of the K-means segments:

Number	Name	Desciption
1	Small and Occasional	Recently active but spends low and not very often
2	New	Fairly recently active but spended low and have not been active earlier
3	Good Average	Active and buys from seldom to often and spends from low to average
4	Big Spenders	Spend most money of all and they recently did it again
5	Inactive	Not active for a long time and spended low
6	Fading Away	Not active for some time and spended low
7	Good and Loyal	Spend average or more money and buy very often

To compare with the Putler segments I computer for each K-means cluster the percentual composition of customers that are in each of the Putler clusters:

```
CustomerRFM %>%
    group_by(kmeansSegment) %>%
    count(putlerSegment) %>%
    mutate(percent = 100*n/sum(n))
```

```
## # A tibble: 26 x 4
## # Groups:   kmeansSegment [7]
```

```
##    kmeansSegment putlerSegment      n percent
##    <fct>         <fct>          <int>   <dbl>
##  1 1             AtRisk             9   2.24
##  2 1             CantLoseThem       1   0.249
##  3 1             Champions        229  57.1
##  4 1             LoyalCustomers   162  40.4
##  5 2             AboutToSleep     435  17.2
##  6 2             AtRisk           171   6.78
##  7 2             Champions         81   3.21
##  8 2             Hibernating       92   3.65
##  9 2             Lost              81   3.21
## 10 2             LoyalCustomers   731  29.0
## # ... with 16 more rows
```

It is seen from the table that especially K-means segment 7 overlap with many Putler segments, and it is seen that for example the Putler Champions segment is split into several of the K-means segments. All in all the two sets of segments parts the customers in very different ways.

Part 2: Analysis of Predicting Customer Segments from their First Order

Introduction

I want to predict in which segment a new customer ends up at the end of the period of the given data. I will use all the given features of the customer available at the time of the the customers first purchase. This a classification problem and it is grouped as a supervised machine learning problem because we train a model from a training set where the correct answer is given.

I train two different models for each of the K-means segment prediction and the Putler segment prediction. First I use k-nearest neighbors and then I use random forest because I want to try more than one algorithm to see how they compare. Also I compare to a simple random approach of guessing the segment randomly according to the segment sizes.

For this investigation I assume that each customers first non-cancellation order is the customers first order, even though I cannot know if the customer has been a customer for years before the first date in my dataset.

The Class Imbalance Problem

We saw earlier that the number of customers in the Putler segments varies from a few to about 1000, and that the numbers of customers in the Kmeans segments varies from a few to about 2500. The large difference of the class sizes, the imbalanced classes, makes it very possible that the classification algorithms optimises the accuracy at the expense of the precision, and we risk an algorithm that always predicts the most populated category.

Different approaches to prevent this exists, but it is not the purpose of this report to compare different possible solution to the class imbalance problem and I just choose a method that often works and after the training of the models I check if the predictions are spread on several categories. The approach I choose is oversampling of the small categories. It is chosen in the caret train method by adding `sampling = "up"` to the trainControl.

Feature extraction

First I construct a data frame that holds information about the first purchase for each customer. I ignore all the cancellations because I only want the first non-cancellation order for each customer. In the process we loose about twenty customers who only had cancellations in the dataset.

```
firstOrder <- onlineRetail %>%
    filter(!startsWith(tolower(as.character(InvoiceNo)), "c")) %>%
    group_by(CustomerID, InvoiceNo) %>%
    summarize(N = n(), TotalQuantity = sum(Quantity),
              TimeOfDay = (60 * hour(min(InvoiceDate)) + minute(min(InvoiceDate))),
              TotalPrice = sum(Quantity*UnitPrice), Country = first(Country),
              InvoiceDate = min(InvoiceDate)) %>%
    slice(which.min(InvoiceDate)) %>%
    ungroup() %>%
    inner_join(CustomerRFM, by = "CustomerID") %>%
    select(-CustomerID, -Recency, -Frequency, -Monetary, -InvoiceDate, -InvoiceNo)

print(nrow(firstOrder))
```

```
## [1] 4334
```

```
head(firstOrder)
```

```
## # A tibble: 6 x 7
##       N TotalQuantity TimeOfDay TotalPrice Country putlerSegment
##   <int>         <int>     <dbl>      <dbl> <fct>   <fct>
## 1    31           319       837       712. Iceland Champions
## 2    16          1248      1089       653. Finland AtRisk
## 3    72           630       531      1458. Italy   PotentialLoy~
## 4    16           196       901       294. Norway  Lost
## 5    15            98       693       296. Norway  LoyalCustome~
## 6     4            20      1007        89  Bahrain Lost
## # ... with 1 more variable: kmeansSegment <fct>
```

Because I want to use the k-nearest neighbor algorithm, see more about this in a moment, and because the k-nearest neighbor algorithm use the distance between points I need to scale the numeric data before I train the models. Also I change the `Country` feature to dummy variable, aka one hot encoding, to make it numeric:

```
dummy <- dummyVars("~ Country", data = firstOrder, sep = NULL)
firstOrderEncoded <- cbind(firstOrder, predict(dummy, newdata = firstOrder))

firstOrderEncoded <- firstOrderEncoded %>%
    select(-Country) %>%
    mutate_if(is.numeric, function(clm) { as.vector(scale(clm)) })

glimpse(firstOrderEncoded)
```

```
## Observations: 4,334
## Variables: 43
## $ N                           <dbl> 0.29595976, -0.33561666, 2.02226866,...
```

```
## $ TotalQuantity              <dbl> 0.17380512, 2.42427831, 0.92719281, ...
## $ TimeOfDay                  <dbl> 0.80532341, 2.59495502, -1.36780069,...
## $ TotalPrice                 <dbl> 0.560181157, 0.453026216, 1.91484921...
## $ putlerSegment              <fct> Champions, AtRisk, PotentialLoyalist...
## $ kmeansSegment              <fct> 2, 2, 2, 5, 2, 6, 5, 6, 2, 2, 2, 2, ...
## $ CountryAustralia           <dbl> -0.04561189, -0.04561189, -0.0456118...
## $ CountryAustria             <dbl> -0.04561189, -0.04561189, -0.0456118...
## $ CountryBahrain             <dbl> -0.02148427, -0.02148427, -0.0214842...
## $ CountryBelgium             <dbl> -0.07461341, -0.07461341, -0.0746134...
## $ CountryBrazil              <dbl> -0.01518992, -0.01518992, -0.0151899...
## $ CountryCanada              <dbl> -0.03039037, -0.03039037, -0.0303903...
## $ CountryChannelIslands      <dbl> -0.04561189, -0.04561189, -0.0456118...
## $ CountryCyprus              <dbl> -0.04021661, -0.04021661, -0.0402166...
## $ CountryCzechRepublic       <dbl> -0.01518992, -0.01518992, -0.0151899...
## $ CountryDenmark             <dbl> -0.04299833, -0.04299833, -0.0429983...
## $ CountryEIRE                <dbl> -0.02631579, -0.02631579, -0.0263157...
## $ CountryEuropeanCommunity   <dbl> -0.01518992, -0.01518992, -0.0151899...
## $ CountryFinland             <dbl> -0.05268635, 18.97586794, -0.0526863...
## $ CountryFrance              <dbl> -0.1431095, -0.1431095, -0.1431095, ...
## $ CountryGermany             <dbl> -0.1488781, -0.1488781, -0.1488781, ...
## $ CountryGreece              <dbl> -0.03039037, -0.03039037, -0.0303903...
## $ CountryIceland             <dbl> 65.81793244, -0.01518992, -0.0151899...
## $ CountryIsrael              <dbl> -0.02631579, -0.02631579, -0.0263157...
## $ CountryItaly               <dbl> -0.05692094, -0.05692094, 17.5641746...
## $ CountryJapan               <dbl> -0.04299833, -0.04299833, -0.0429983...
## $ CountryLebanon             <dbl> -0.01518992, -0.01518992, -0.0151899...
## $ CountryLithuania           <dbl> -0.01518992, -0.01518992, -0.0151899...
## $ CountryMalta               <dbl> -0.02148427, -0.02148427, -0.0214842...
## $ CountryNetherlands         <dbl> -0.04561189, -0.04561189, -0.0456118...
## $ CountryNorway              <dbl> -0.04808472, -0.04808472, -0.0480847...
## $ CountryPoland              <dbl> -0.03722904, -0.03722904, -0.0372290...
## $ CountryPortugal            <dbl> -0.06634929, -0.06634929, -0.0663492...
## $ CountryRSA                 <dbl> -0.01518992, -0.01518992, -0.0151899...
## $ CountrySaudiArabia         <dbl> -0.01518992, -0.01518992, -0.0151899...
## $ CountrySingapore           <dbl> -0.01518992, -0.01518992, -0.0151899...
## $ CountrySpain               <dbl> -0.08062912, -0.08062912, -0.0806291...
## $ CountrySweden              <dbl> -0.04299833, -0.04299833, -0.0429983...
## $ CountrySwitzerland         <dbl> -0.06808083, -0.06808083, -0.0680808...
## $ CountryUnitedArabEmirates <dbl> -0.02148427, -0.02148427, -0.0214842...
## $ CountryUnitedKingdom       <dbl> -3.060435, -3.060435, -3.060435, -3....
## $ CountryUnspecified         <dbl> -0.03039037, -0.03039037, -0.0303903...
## $ CountryUSA                 <dbl> -0.03039037, -0.03039037, -0.0303903...
```

Using the R caret package I first split my data into test and training sets with 70% of the original data being the training set and 30% being the test set:

```r
set.seed(9)
test_index <- createDataPartition(firstOrderEncoded$kmeansSegment, times = 1, p = 0.3, list = FALSE)

test_set <- firstOrderEncoded[test_index, ]
train_set <- firstOrderEncoded[-test_index, ]

cat("Size of test_set =", nrow(test_set), "\n")
```

```
## Size of test_set = 1303
```

```
cat("Size of train_set =", nrow(train_set), "\n")
```

```
## Size of train_set = 3031
```

Random Guessing for Putler Segments

As a baseline I use a simple predictor that for any customer always predicts randomly according to the sizes of each segment. First for the Putler segments:

```
putlerSegments <- CustomerRFM %>%
    group_by(putlerSegment) %>%
    summarize(NumberOfCustomers = n())
```

and when predicting the testset we get the following accuracy:

```
set.seed(9)
idx <- sample(nrow(putlerSegments), size = nrow(test_set), replace=TRUE, prob = putlerSegments$NumberOf(
test_pred <- putlerSegments[idx, ]$putlerSegment
accuracy <- mean(test_pred == test_set$putlerSegment)
accuracy_random_onPutler = sprintf("%.2f", accuracy)
accuracy_random_onPutler
```

```
## [1] "0.13"
```

Random Guessing for Kmeans Segments

The baseline for the Kmeans segments is:

```
kmeansSegments <- CustomerRFM %>%
    group_by(kmeansSegment) %>%
    summarize(NumberOfCustomers = n())
```

and when predicting the testset we get the following accuracy:

```
set.seed(9)
idx <- sample(nrow(kmeansSegments), size = nrow(test_set), replace=TRUE, prob = kmeansSegments$NumberOf(
test_pred <- kmeansSegments[idx, ]$kmeansSegment
accuracy <- mean(test_pred == test_set$kmeansSegment)
accuracy_random_onKmeans = sprintf("%.2f", accuracy)
accuracy_random_onKmeans
```

```
## [1] "0.35"
```

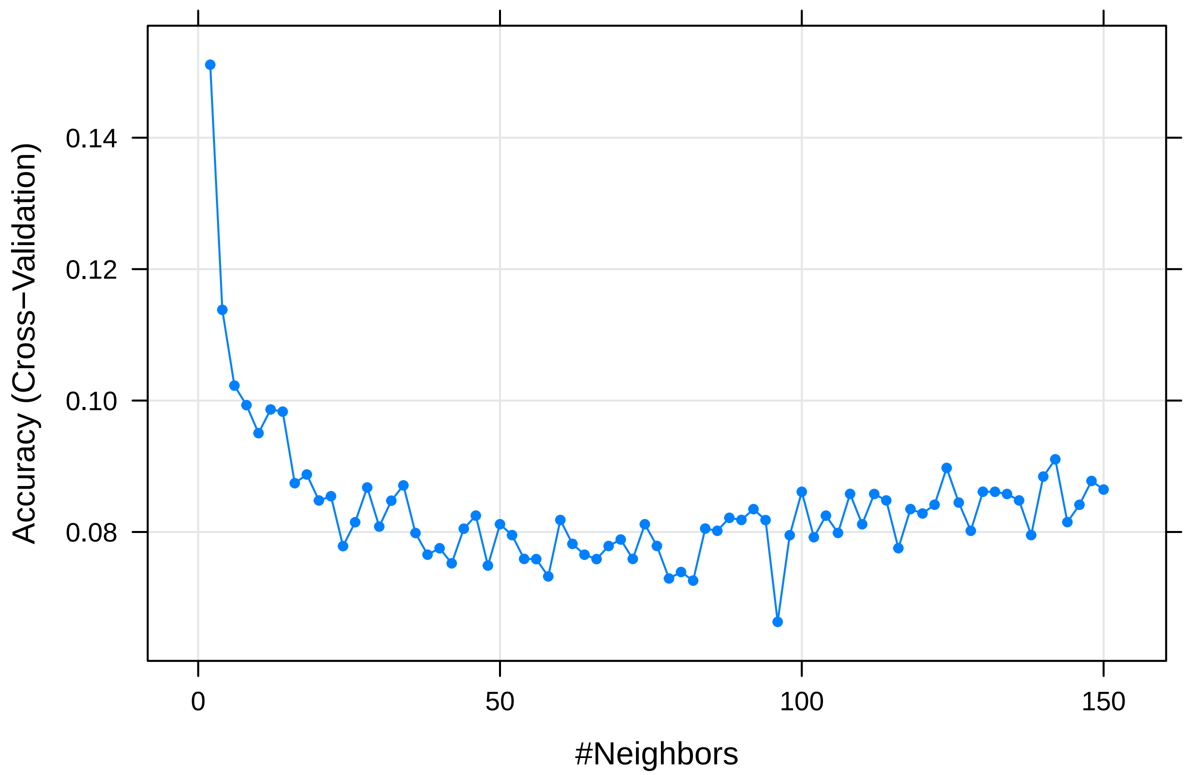
Train and test K-nearest Neigbor Algorithm for Putler Segments

Training

I use the caret package that takes care of cross validation and running on different values of `k`, the number of neighbors to look at, to find the best value. First I try to find the best `k` for Putler segments using 5-fold cross validation:

```
set.seed(9)
trained_knn_onPutler <- train(putlerSegment ~ .,
                              method = "knn",
                              data = select(train_set, -kmeansSegment),
                              trControl = trainControl(method = "cv", number = 5),
                              tuneGrid = data.frame(k = seq(2, 150, 2)))
```

```
plot(trained_knn_onPutler, pch = 20)
```



The algorithm finds the best `k = 2` which I find is surprisingly low.

Results

Using `k = 2` I now find the accuracy when trying to predict the testset:

```r
test_pred <- predict(trained_knn_onPutler, newdata = test_set)
accuracy <- mean(test_pred == test_set$putlerSegment)
accuracy_knn_onPutler = sprintf("%.2f", accuracy)
accuracy_knn_onPutler
```

```
## [1] "0.14"
```

Looking at the Confusion Matrix it is seen that the method predicts all segments and in different frequency as expected and there seems to be no problems with class imbalance:

```r
confusionMatrix(test_pred, test_set$putlerSegment)$table
```

```
##                     Reference
## Prediction           AboutToSleep AtRisk CantLoseThem Champions Hibernating
##   AboutToSleep                 18     15            0        12          14
##   AtRisk                       16     24            0        15          14
##   CantLoseThem                  0      1            0         0           0
##   Champions                    14     15            0        14           7
##   Hibernating                   8     17            0        13           4
##   Lost                         16      8            0         9           8
##   LoyalCustomers               10     19            0        20          11
##   NeedingAttention              5     10            0         9           4
##   PotentialLoyalist            21     24            0        14          14
##   Promising                     8      2            0         2           5
##   RecentCustomers               2      1            0         4           2
##                     Reference
## Prediction           Lost LoyalCustomers NeedingAttention PotentialLoyalist
##   AboutToSleep         36             33                3                26
##   AtRisk               38             36                6                23
##   CantLoseThem          2              0                0                 0
##   Champions            27             25                8                21
##   Hibernating          13             28                2                14
##   Lost                 44             25                6                23
##   LoyalCustomers       43             46               12                29
##   NeedingAttention     19             17                2                10
##   PotentialLoyalist    49             29                7                25
##   Promising            21              9                2                11
##   RecentCustomers       5              3                2                 3
##                     Reference
## Prediction           Promising RecentCustomers
##   AboutToSleep              11               6
##   AtRisk                     4               3
##   CantLoseThem               0               0
##   Champions                  4               1
##   Hibernating                2               1
##   Lost                       5               5
##   LoyalCustomers             6               2
##   NeedingAttention           1               1
##   PotentialLoyalist          7               5
##   Promising                  5               0
##   RecentCustomers            2               0
```

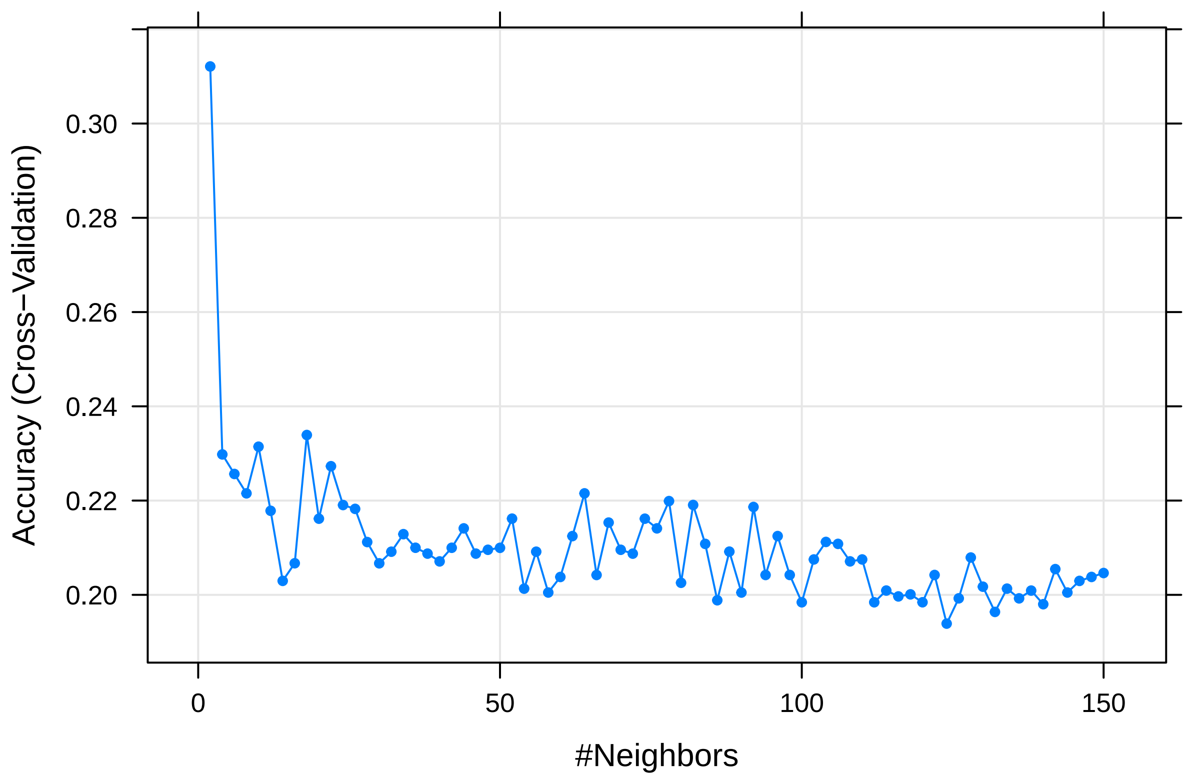
Train and Test K-nearest Neigbor Algorithm for Kmeans Segments

Training

Again I first try to find the best `k` for Kmeans segments using 5-fold cross validation:

```
set.seed(9)
trained_knn_onKmeans <- train(kmeansSegment ~ .,
                              method = "knn",
                              data = select(train_set, -putlerSegment),
                              trControl = trainControl(method = "cv", number = 5),
                              tuneGrid = data.frame(k = seq(2, 150, 2)))
```

```
plot(trained_knn_onKmeans, pch = 20)
```



As for the Putler segments the best `k` is surprisingly low at `k = 2`.

Results

Using `k = 2` I now find the accuracy when trying to predict the testset:

```
test_pred <- predict(trained_knn_onKmeans, newdata = test_set)
accuracy <- mean(test_pred == test_set$kmeansSegment)
accuracy_knn_onKmeans = sprintf("%.2f", accuracy)
accuracy_knn_onKmeans
```

```
## [1] "0.31"
```

Looking at the Confusion Matrix it is seen that the method predicts all segments and in different frequency as expected and there seems to be no problems with class imbalance:

```r
confusionMatrix(test_pred, test_set$kmeansSegment)$table
```

```
##           Reference
## Prediction   1   2   3   4   5   6   7
##          1  20  82   3   0  15  32   0
##          2  49 276   4   0  78  75   1
##          3   0   9   0   0   1   3   0
##          4   0   1   0   0   0   0   0
##          5  22 169   2   1  37  52   0
##          6  30 218   4   0  47  70   1
##          7   0   1   0   0   0   0   0
```

Train and Test Random Forest Algorithm for Putler Segments

Training

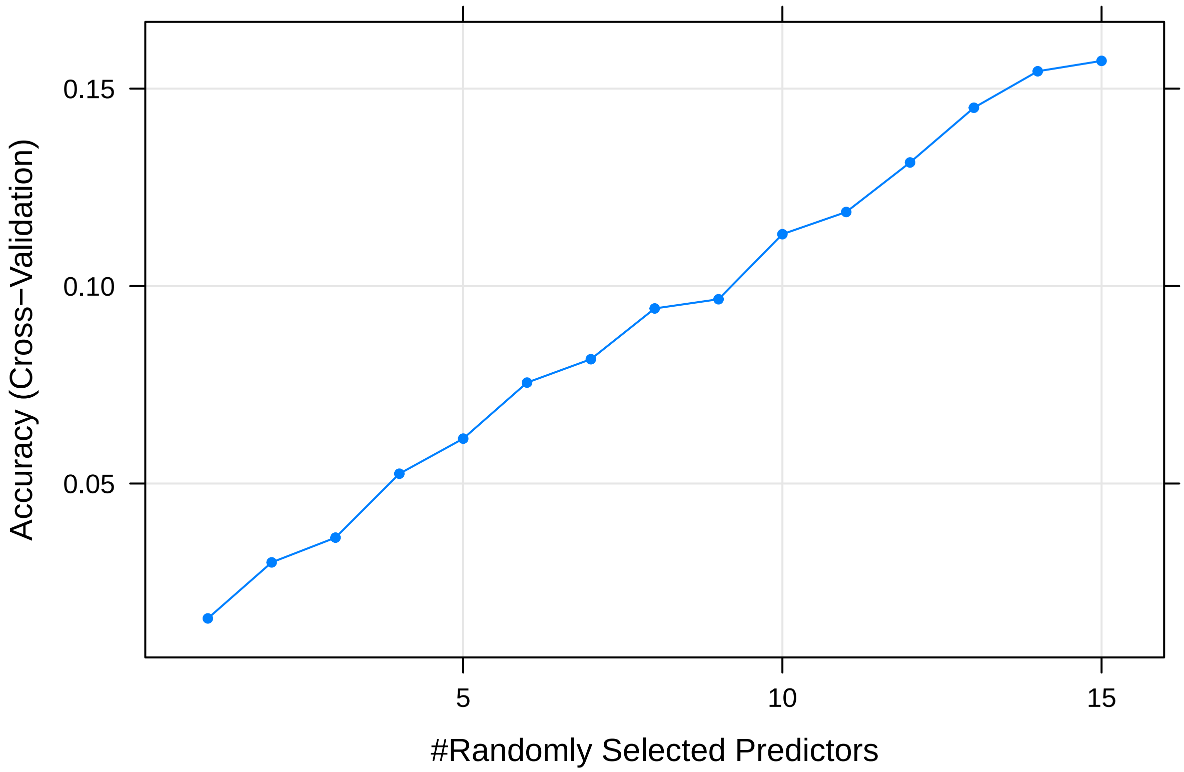
A Random Forest works by building many different Decision Trees during training. The predictions are made by majority vote by the decision trees.

The Random Forest algorithm has several parameters - the one that is said to have most influence on the result is `mtry` that is the number of variables randomly sampled as candidates at each split of decision tress.

Using 5-fold cross validation and the default of 500 decision trees the training is:

```r
set.seed(9)
trained_Forest_onPutler <- train(putlerSegment ~ .,
                              method = "rf",
                              data = select(train_set, -kmeansSegment),
                              trControl = trainControl(method = "cv", number = 5),
                              tuneGrid = data.frame(mtry = seq(1,15,1)))
```

```r
plot(trained_Forest_onPutler, pch = 20)
```



Results

The algorithm found the best `mtry = 15` but it is at the end of my search interval. From different other peoples experience I expected `mtry` to be lower than 15 but I was wrong. Because the training is very slow and because the curve seems to flatten at 15 I choose to keep `mtry = 15`. Using this value I now find the accuracy when trying to predict the testset:

```
test_pred <- predict(trained_Forest_onPutler, newdata = test_set)
accuracy <- mean(test_pred == test_set$putlerSegment)
accuracy_Forest_onPutler = sprintf("%.2f", accuracy)
accuracy_Forest_onPutler
```

```
## [1] "0.18"
```

Looking at the Confusion Matrix it is seen that the method predicts all segments and in different frequency as expected and there seems to be no problems with class imbalance:

```
confusionMatrix(test_pred, test_set$putlerSegment)$table
```

```
##                      Reference
## Prediction             AboutToSleep AtRisk CantLoseThem Champions Hibernating
##   AboutToSleep                    9     16            0        12           9
##   AtRisk                          9     17            0        13          10
##   CantLoseThem                    0      0            0         0           1
```

```
##    Champions                 15      20              0         23            8
##    Hibernating                3       9              0          6            2
##    Lost                      25      21              0         19           18
##    LoyalCustomers            13      25              0         22           15
##    NeedingAttention          10       8              0          7            3
##    PotentialLoyalist         12      11              0          7           11
##    Promising                 15       6              0          3            3
##    RecentCustomers            7       3              0          0            3
##                       Reference
## Prediction             Lost  LoyalCustomers  NeedingAttention  PotentialLoyalist
##    AboutToSleep         24              27                 6                 28
##    AtRisk               17              31                 4                 12
##    CantLoseThem          1               1                 0                  0
##    Champions            34              25                 7                 19
##    Hibernating           7              19                 0                  8
##    Lost                 91              38                11                 35
##    LoyalCustomers       31              58                 9                 31
##    NeedingAttention     21              10                 2                 12
##    PotentialLoyalist    35              20                 7                 17
##    Promising            25              15                 2                 14
##    RecentCustomers      11               7                 2                  9
##                       Reference
## Prediction             Promising  RecentCustomers
##    AboutToSleep               10                4
##    AtRisk                      2                1
##    CantLoseThem                0                0
##    Champions                   1                0
##    Hibernating                 0                1
##    Lost                        7                6
##    LoyalCustomers              4                3
##    NeedingAttention            5                0
##    PotentialLoyalist           5                4
##    Promising                   9                2
##    RecentCustomers             4                3
```

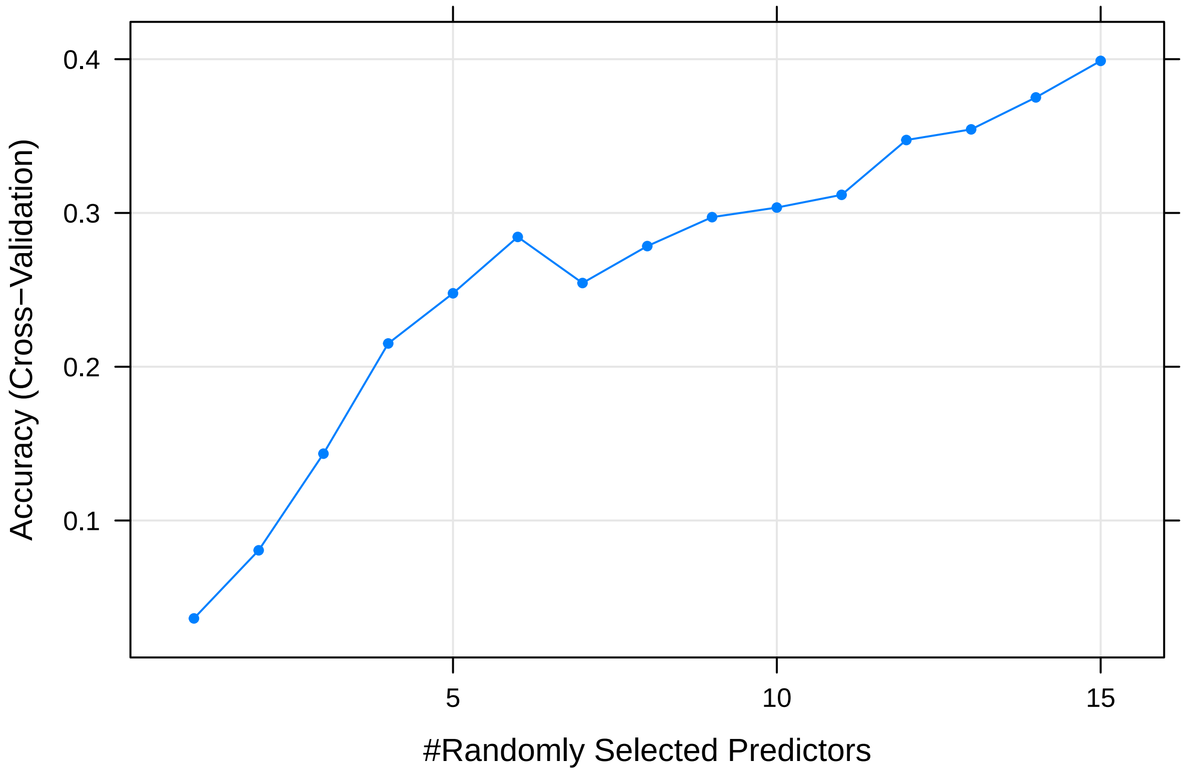
Train and test Random Forest Algorithm for Kmeans Segments

Training

For the Kmeans segments I train:

```
set.seed(9)
trained_Forest_onKmeans <- train(kmeansSegment ~ .,
                                 method = "rf",
                                 data = select(train_set, -putlerSegment),
                                 trControl = trainControl(method = "cv", number = 5),
                                 tuneGrid = data.frame(mtry = seq(1,15,1)))
```

```
plot(trained_Forest_onKmeans, pch = 20)
```



Results

Again the algorithm found the best `mtry = 15` and as before I choose to keep `mtry = 15`. Using this value I now find the accuracy when trying to predict the testset:

```
test_pred <- predict(trained_Forest_onKmeans, newdata = test_set)
accuracy <- mean(test_pred == test_set$kmeansSegment)
accuracy_Forest_onKmeans = sprintf("%.2f", accuracy)
accuracy_Forest_onKmeans
```

```
## [1] "0.39"
```

Looking at the Confusion Matrix it is seen that the method predicts all segments and in different frequency as expected and there seems to be no problems with class imbalance:

```
confusionMatrix(test_pred, test_set$kmeansSegment)$table
```

```
##           Reference
## Prediction   1   2   3   4   5   6   7
##          1  21  83   2   0  15  19   0
##          2  66 417   5   0  89 124   1
##          3   0  12   0   0   2   4   0
##          4   0   1   0   0   0   0   0
##          5  20 136   4   1  40  52   0
```

```
##            6   14  106    2    0   32   33    1
##            7    0    1    0    0    0    0    0
```

Collected Results of Predicting

Method	Putler Segments Accuracy	Kmeans Segments Accurary
Random	0.13	0.35
K-nearest neighbor	0.14	0.31
Random Forest	0.18	0.39

Conclusion

Customer Segments

The traditional, fixed RFM analysis that I used defines eleven different customer segments. The Machine Learning generated customer segments turn out to have only seven customer segments. The analysis shows that the two sets of customer segments splits the customers in very different ways.

The machine learning method created a reasonable sized set of segments that when interpreted seems to make good business sense. It is, however, impossible to say which set of the segment sets that are best because it of course depends on the intended use. If we want a set of segments that is good at describing the states a customer goes through during time, it might be possible to make an analysis of the development over time of customers. In this report I assumed that the customers were close to static over the year of the dataset. I belive that finding dynamic trends or customer lifecycles requires a dataset that covers a longer period of at least several years for a business like the one, I analysed in this report.

Predicting Customer Segment

Using the K-nearest neigbor algorithm to predict the future customer segment from customers first order and their country did worse than random for the Kmeans segment! However, the Random Forest algorithm predicted the future customer segment from customers first order and their country a little better than random for both segment sets. Even though better than random, the accuracy is still low for both segment sets. Predicting using Random Forest might, though, be a beginning of helping the business to a finer segmentation of new customers.

It is clearly easier to predict the Kmeans segments - this is quite naturally because of the fewer segments and the very high number of customers in a single segment.

In this report I did not use the information of the content of the first order. It would be interesting to investigate if using this data, or perhaps even making a Market Basket Analysis, could help increase the accuracy of the predicting. Also I should refine the Random Forest analyses to be sure to find the really best value of `mtry`.