Unsupervised Learning Techniques

Beaty

9/9/2021

Data Understanding

1. Define the question:

I am undertaking a project that will inform the marketing department on the most relevant marketing strategies that will result in the highest no. of sales (total price including tax).

2. Metric for success:

- Cleaned data.
- Graphical representation of the relationships in the data as well as the distributions of the different variables in the data.
- Perform the different Unsupervised learning techniques as required.
- Sound conclusions and recommendations to Carrefour as per the analysis done.

3. Understanding the context:

I am a data analyst at Carrefour Kenya and I am currently undertaking a project that will inform the marketing department on the most relevant marketing strategies that will result in the highest no. of sales (total price including tax).

My project has been divided into four parts where I will explore a recent marketing dataset by first performing an exploratory data analysis to understand the data, then I will carry out dimensionality reduction to obtain a set of principal variables. I will then an association analysis, to find relationships between the different items sold so as to be able to make recommendations for the customers based on whether they have similar items to boost Carrefour's sales. I will also check for anomalies on the available sales data.

4. Experimental design:

Steps to be undertaken during this study include:
- Loading the data & needed packages.
- Exploring the dataset.
- Cleaning the data.
- Exploratory data analysis.
- Implementing the solution for each of the unsupervised learning techniques.
- Conclusions & recommendations.

Loading packages

```
# Loading the libraries
library(corrplot)
```

```
## corrplot 0.90 loaded

library(PerformanceAnalytics)

## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

##
## Attaching package: 'PerformanceAnalytics'

## The following object is masked from 'package:graphics':
##
##     legend

library(ggplot2)
library(data.table)

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:xts':
##
##     first, last

library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:data.table':
##
##     between, first, last

## The following objects are masked from 'package:xts':
##
##     first, last

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v tibble  3.1.4     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::between()   masks data.table::between()
## x dplyr::filter()    masks stats::filter()
## x dplyr::first()     masks data.table::first(), xts::first()
## x dplyr::lag()       masks stats::lag()
## x dplyr::last()      masks data.table::last(), xts::last()
## x purrr::transpose() masks data.table::transpose()

library(data.table)
library(janitor)

##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test

library(caret)

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

library(scales)

##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(grid)
library(devtools)

## Loading required package: usethis

library(ggbiplot)

## Loading required package: plyr

## ------------------------------------------------------------------------------
----

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first,
then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------
----

##
## Attaching package: 'plyr'

## The following object is masked from 'package:purrr':
##
##     compact

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

library(mclust)

## Package 'mclust' version 5.4.7
## Type 'citation("mclust")' for citing this R package in publications.

##
## Attaching package: 'mclust'

## The following object is masked from 'package:purrr':
##
##     map

library(clustvarsel)

## Package 'clustvarsel' version 2.3.4

## Type 'citation("clustvarsel")' for citing this R package in publications.

library(lessR)

##
## lessR 4.0.3  feedback: gerbing@pdx.edu  web: lessRstats.com/new
```

```
## -------------------------------------------------------------
## > d <- Read("")   Read text, Excel, SPSS, SAS, or R data file
##   d is default data frame, data= in analysis routines optional
##
## Learn about reading, writing, and manipulating data, graphics,
## testing means and proportions, regression, factor analysis,
## customization, and descriptive statistics from pivot tables.
##   Enter:  browseVignettes("lessR")
##
## View changes in this new version of lessR.
##   Enter: help(package=lessR)  Click: Package NEWS

##
## Attaching package: 'lessR'

## The following object is masked from 'package:plyr':
##
##     .

## The following object is masked from 'package:scales':
##
##     rescale

## The following object is masked from 'package:dplyr':
##
##     recode

## The following object is masked from 'package:data.table':
##
##     set

## The following object is masked from 'package:PerformanceAnalytics':
##
##     kurtosis
```

Part 1

```
# Loading the data
data <- fread('http://bit.ly/CarreFourDataset')

# Previewing the data
head(data)

##     Invoice ID Branch Customer type Gender           Product line Unit
price
## 1: 750-67-8428      A        Member Female      Health and beauty
74.69
## 2: 226-31-3081      C        Normal Female Electronic accessories
15.28
## 3: 631-41-3108      A        Normal   Male     Home and lifestyle
46.33
```

```
## 4: 123-19-1176      A        Member   Male      Health and beauty
58.22
## 5: 373-73-7910      A        Normal   Male      Sports and travel
86.31
## 6: 699-14-3026      C        Normal   Male Electronic accessories
85.39
##    Quantity     Tax       Date  Time     Payment   cogs gross margin
percentage
## 1:        7 26.1415  1/5/2019 13:08     Ewallet 522.83
4.761905
## 2:        5  3.8200  3/8/2019 10:29        Cash  76.40
4.761905
## 3:        7 16.2155  3/3/2019 13:23 Credit card 324.31
4.761905
## 4:        8 23.2880 1/27/2019 20:33     Ewallet 465.76
4.761905
## 5:        7 30.2085  2/8/2019 10:37     Ewallet 604.17
4.761905
## 6:        7 29.8865 3/25/2019 18:30     Ewallet 597.73
4.761905
##    gross income Rating    Total
## 1:      26.1415    9.1 548.9715
## 2:       3.8200    9.6  80.2200
## 3:      16.2155    7.4 340.5255
## 4:      23.2880    8.4 489.0480
## 5:      30.2085    5.3 634.3785
## 6:      29.8865    4.1 627.6165
```

```
# Checking the shape of the data
dim(data)
```

```
## [1] 1000   16
```

The dataset has 1000 entries and 16 columns.

```
# Checking column names of our data
colnames(data)
```

```
##  [1] "Invoice ID"              "Branch"
##  [3] "Customer type"           "Gender"
##  [5] "Product line"            "Unit price"
##  [7] "Quantity"                "Tax"
##  [9] "Date"                    "Time"
## [11] "Payment"                 "cogs"
## [13] "gross margin percentage" "gross income"
## [15] "Rating"                  "Total"
```

Our column titles are as listed above. We will need to standardize our column names to remove the spaces

```r
#Checking column data types
str(data)

## Classes 'data.table' and 'data.frame':   1000 obs. of  16 variables:
##  $ Invoice ID             : chr  "750-67-8428" "226-31-3081" "631-41-3108"
"123-19-1176" ...
##  $ Branch                 : chr  "A" "C" "A" "A" ...
##  $ Customer type          : chr  "Member" "Normal" "Normal" "Member" ...
##  $ Gender                 : chr  "Female" "Female" "Male" "Male" ...
##  $ Product line           : chr  "Health and beauty" "Electronic
accessories" "Home and lifestyle" "Health and beauty" ...
##  $ Unit price             : num  74.7 15.3 46.3 58.2 86.3 ...
##  $ Quantity               : int  7 5 7 8 7 7 6 10 2 3 ...
##  $ Tax                    : num  26.14 3.82 16.22 23.29 30.21 ...
##  $ Date                   : chr  "1/5/2019" "3/8/2019" "3/3/2019"
"1/27/2019" ...
##  $ Time                   : chr  "13:08" "10:29" "13:23" "20:33" ...
##  $ Payment                : chr  "Ewallet" "Cash" "Credit card" "Ewallet"
...
##  $ cogs                   : num  522.8 76.4 324.3 465.8 604.2 ...
##  $ gross margin percentage: num  4.76 4.76 4.76 4.76 4.76 ...
##  $ gross income           : num  26.14 3.82 16.22 23.29 30.21 ...
##  $ Rating                 : num  9.1 9.6 7.4 8.4 5.3 4.1 5.8 8 7.2 5.9 ...
##  $ Total                  : num  549 80.2 340.5 489 634.4 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The data types are fine and we don't need to convert any.

```r
#Checking summary statistics of our dataset
summary(data)

##   Invoice ID           Branch          Customer type         Gender
##  Length:1000        Length:1000        Length:1000        Length:1000
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  Product line         Unit price       Quantity          Tax
##  Length:1000        Min.   :10.08   Min.   : 1.00   Min.   : 0.5085
##  Class :character   1st Qu.:32.88   1st Qu.: 3.00   1st Qu.: 5.9249
##  Mode  :character   Median :55.23   Median : 5.00   Median :12.0880
##                     Mean   :55.67   Mean   : 5.51   Mean   :15.3794
##                     3rd Qu.:77.94   3rd Qu.: 8.00   3rd Qu.:22.4453
##                     Max.   :99.96   Max.   :10.00   Max.   :49.6500
##      Date               Time             Payment               cogs
##  Length:1000        Length:1000        Length:1000        Min.   : 10.17
##  Class :character   Class :character   Class :character   1st Qu.:118.50
##  Mode  :character   Mode  :character   Mode  :character   Median :241.76
##                                                           Mean   :307.59
##                                                           3rd Qu.:448.90
```

```
##                                                                          Max.   :993.00
##  gross margin percentage  gross income        Rating          Total
##  Min.   :4.762           Min.   : 0.5085   Min.   : 4.000   Min.   :
10.68
##  1st Qu.:4.762           1st Qu.: 5.9249   1st Qu.: 5.500   1st Qu.:
124.42
##  Median :4.762           Median :12.0880   Median : 7.000   Median :
253.85
##  Mean   :4.762           Mean   :15.3794   Mean   : 6.973   Mean   :
322.97
##  3rd Qu.:4.762           3rd Qu.:22.4453   3rd Qu.: 8.500   3rd Qu.:
471.35
##  Max.   :4.762           Max.   :49.6500   Max.   :10.000   Max.
:1042.65
```

From the summaries we can see that the 'gross margin percentage' has only one value all through. Thus it won't give us much insight.

data Cleaning

```
#Checking for missing values
colSums(is.na(data))
```

```
##              Invoice ID                  Branch           Customer type
##                       0                       0                       0
##                  Gender            Product line              Unit price
##                       0                       0                       0
##                Quantity                     Tax                    Date
##                       0                       0                       0
##                    Time                 Payment                    cogs
##                       0                       0                       0
## gross margin percentage            gross income                  Rating
##                       0                       0                       0
##                   Total
##                       0
```

There are no null values in our data.

```
#Checking for duplicates
length(which(duplicated(data)))
```

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

There no duplicates in our data.

```
#Standardizing column names
#Changing case to lower case
names(data) <- tolower(names(data))

#Replacing spaces with underscore
names(data) <- gsub(" ","_", names(data))
```

```r
#Confirming the changes
colnames(data)
```

```
##  [1] "invoice_id"              "branch"
##  [3] "customer_type"           "gender"
##  [5] "product_line"            "unit_price"
##  [7] "quantity"                "tax"
##  [9] "date"                    "time"
## [11] "payment"                 "cogs"
## [13] "gross_margin_percentage" "gross_income"
## [15] "rating"                  "total"
```

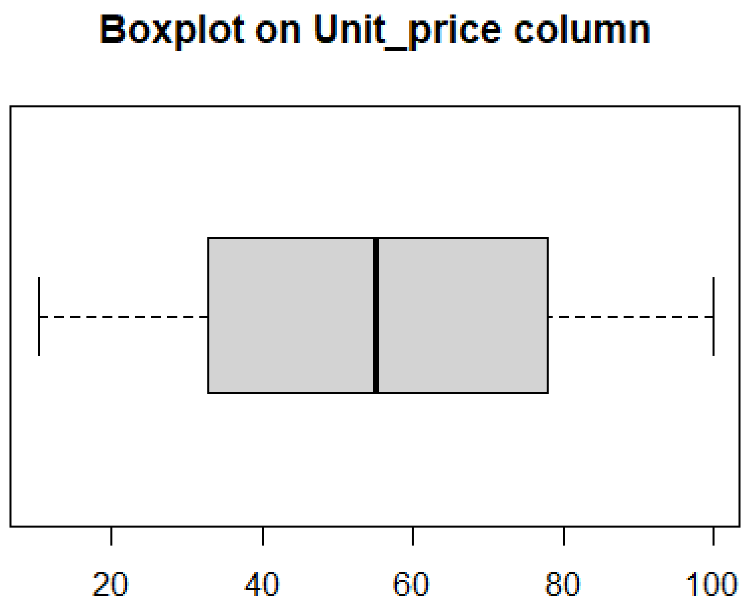
The column names are now standardized and uniform

```r
#Dropping irrelevant columns: we shall drop 'gross margin percentage' column
since it's values won't give us much insight
data<-data %>% select(-gross_margin_percentage)

#Previewing the dataset
colnames(data)
```

```
##  [1] "invoice_id"   "branch"       "customer_type" "gender"
##  [5] "product_line" "unit_price"   "quantity"      "tax"
##  [9] "date"         "time"         "payment"       "cogs"
## [13] "gross_income" "rating"       "total"
```

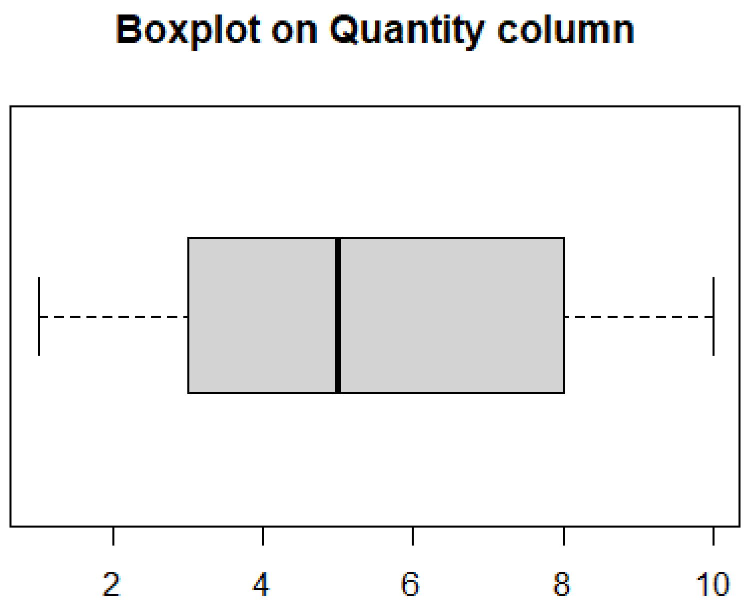
Checking for outliers in numerical columns

```r
#Checking for outliers in unit_price column
boxplot(data$unit_price, main = "Boxplot on Unit_price column", horizontal =
TRUE)
```



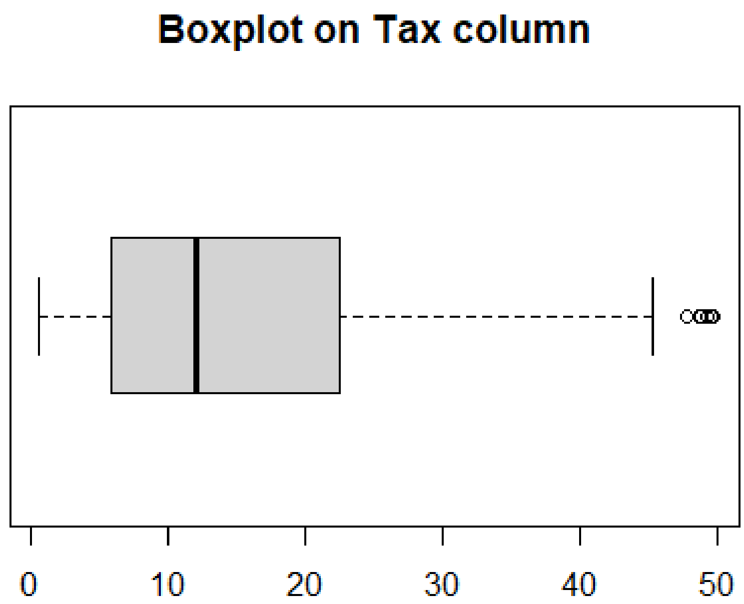
There are no outliers in the ‘unit_price’ column. The customers pick items worth between 5 to 100 shillings

```
#Checking for outliers in quantity column
boxplot(data$quantity, main = "Boxplot on Quantity column", horizontal =
TRUE)
```



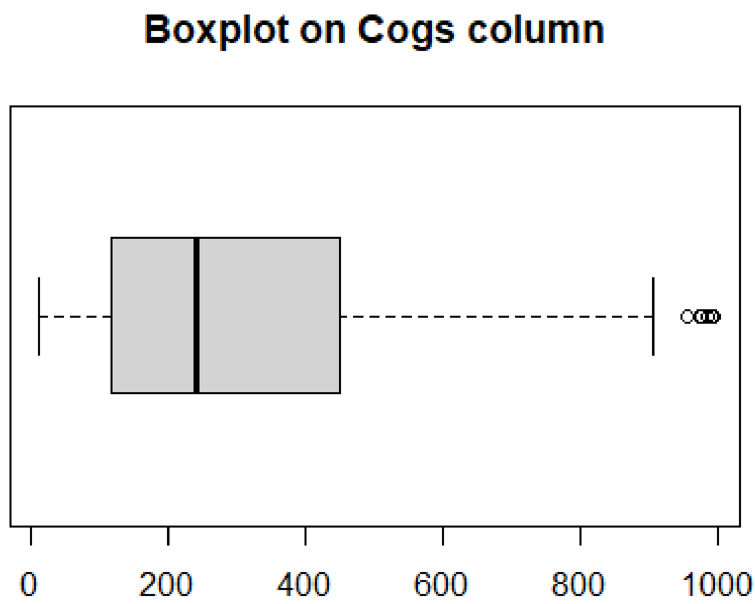
There are no outliers in the ‘quantity’ column. The customers pick between 1 to 10 items with most picking between 3-8.

```
#Checking for outliers in tax column
boxplot(data$tax, main = "Boxplot on Tax column", horizontal = TRUE)
```



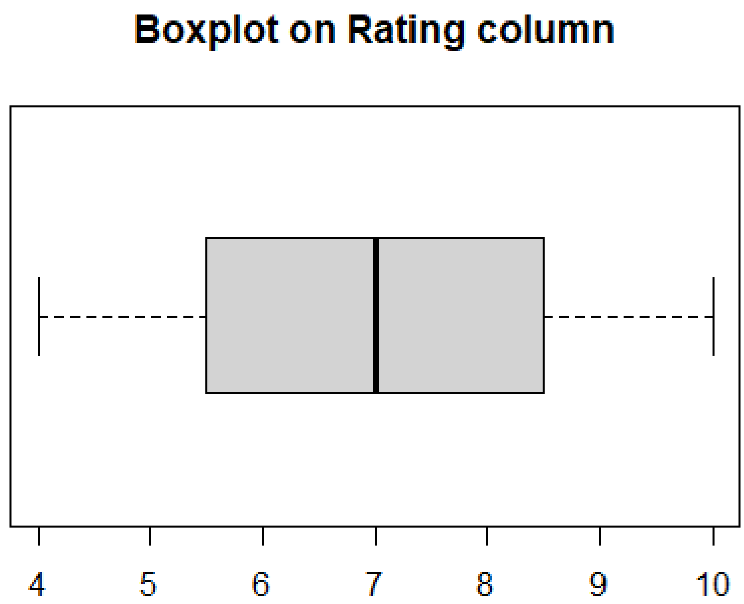
There are a few outliers on the tax column, with most having a tax value between 0-45. The outliers may be due to the fact that some products are usually heavily taxed. We wont remove these outliers.

```
#Checking for outliers in cogs column
boxplot(data$cogs, main = "Boxplot on Cogs column", horizontal = TRUE)
```



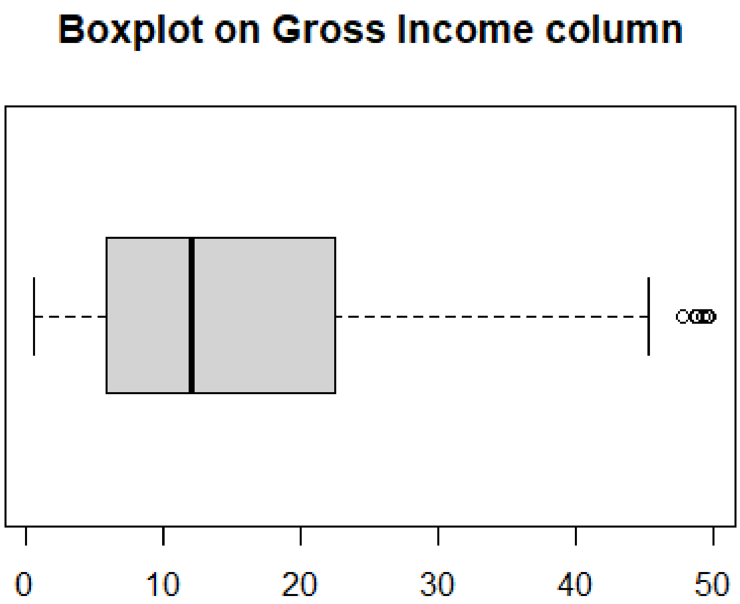
The cogs column also had some outliers. Most of its values lie between 0- 900. We wont remove the outliers too

```
#Checking for outliers in ratingcolumn
boxplot(data$rating, main = "Boxplot on Rating column", horizontal = TRUE)
```



The rating column had no outliers. The ratings were between 4-10, with most being 5.5-8.5

```r
#Checking for outliers in Gross Income column
boxplot(data$gross_income, main = "Boxplot on Gross Income column",
horizontal = TRUE)
```



The gross income has values between 0-45, with most having a gross income between 6-45. There were a few outliers, which we wont delete since it is possible that some values are much higher.

Checking for anomalies in categorical data

```
#Checking for anomalies in branch column
print(unique(data$branch))

## [1] "A" "C" "B"
```

There are 3 branches in our data and seems to be no anomalies.

```
#Checking for anomalies in customer type column
print(unique(data$customer_type))

## [1] "Member" "Normal"
```

There are 2 customer types in our data, still no anomalies

```
#Checking for anomalies in gender column
print(unique(data$gender))

## [1] "Female" "Male"
```

The values in the gender column have no anomalies as there are only 2 unique values

```
#Checking for anomalies in branch column
print(unique(data$product_line))
```

```
## [1] "Health and beauty"      "Electronic accessories" "Home and lifestyle"
## [4] "Sports and travel"      "Food and beverages"     "Fashion
accessories"
```

There are 6 unique values in the product line, and seem to be no anomalies.

```
#Checking for anomalies in branch column
print(unique(data$payment))

## [1] "Ewallet"     "Cash"        "Credit card"
```

On the payment types, there are 3 unique values and no anomalies.

We shall split the date & time columns, so as to get more insights from them

```
#Splitting the date column
data <- separate(data, "date", c("month", "day", "year"), sep = "/")

#Splitting time column
data <- separate(data, "time", c("hour", "minutes"), sep = ":")
#Changing into factors
data$year<- factor (data$year)
data$month<- factor(data$month)
data$day <- factor(data$day)
data$hour <- factor(data$hour)
```

Removing irrelevant columns

```
#Checking our columns
colnames(data)

##  [1] "invoice_id"    "branch"        "customer_type" "gender"
##  [5] "product_line"  "unit_price"    "quantity"      "tax"
##  [9] "month"         "day"           "year"          "hour"
## [13] "minutes"       "payment"       "cogs"          "gross_income"
## [17] "rating"        "total"

#Removing irrelevant columns
data = data %>% select(-invoice_id)
colnames(data)

##  [1] "branch"        "customer_type" "gender"        "product_line"
##  [5] "unit_price"    "quantity"      "tax"           "month"
##  [9] "day"           "year"          "hour"          "minutes"
## [13] "payment"       "cogs"          "gross_income"  "rating"
## [17] "total"
```

Univariate Analysis

```
#Create a subset of numerical columns
num_col <- unlist(lapply(data, is.numeric))
```

```
data_num <- subset(data, select=num_col)

#Previewing the dataset
head(data_num)

##    unit_price quantity     tax   cogs gross_income rating    total
## 1:      74.69        7 26.1415 522.83      26.1415    9.1 548.9715
## 2:      15.28        5  3.8200  76.40       3.8200    9.6  80.2200
## 3:      46.33        7 16.2155 324.31      16.2155    7.4 340.5255
## 4:      58.22        8 23.2880 465.76      23.2880    8.4 489.0480
## 5:      86.31        7 30.2085 604.17      30.2085    5.3 634.3785
## 6:      85.39        7 29.8865 597.73      29.8865    4.1 627.6165
```

Measures of Central tendency

```
#Create a dataframe with measures of central tendency of our numerical
columns

stats <- data.frame(
  Mean = apply(data_num, 2, mean),
  Median = apply(data_num, 2, median),
  Min = apply(data_num, 2, min),
  Max = apply(data_num, 2, max))
stats

##                   Mean  Median     Min     Max
## unit_price    55.67213  55.230 10.0800   99.96
## quantity       5.51000   5.000  1.0000   10.00
## tax           15.37937  12.088  0.5085   49.65
## cogs         307.58738 241.760 10.1700  993.00
## gross_income  15.37937  12.088  0.5085   49.65
## rating         6.97270   7.000  4.0000   10.00
## total        322.96675 253.848 10.6785 1042.65
```

The numerical columns have their means and medians as lised above. We can also see the ranges for each column in the dataframe above.

Measured of spread

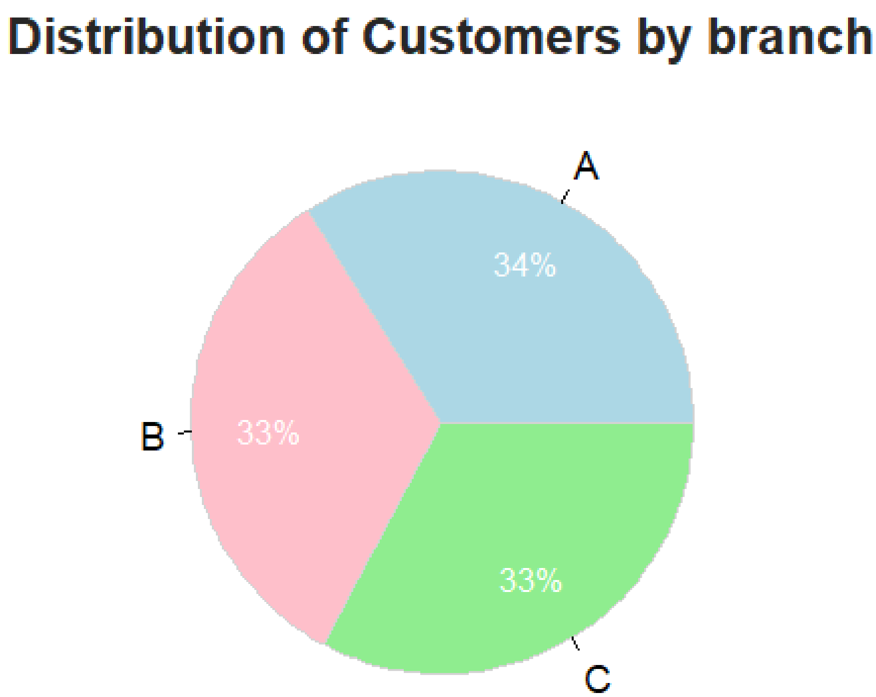
```
#Create a dataframe with measures of spread of our numerical columns
stats2 <- data.frame(
  Variance= apply(data_num, 2, var),
  Std = apply(data_num, 2, sd),
  Skewness = apply(data_num, 2, skewness),
  Kurtosis = apply(data_num, 2, kurtosis))
stats2

##                  Variance        Std    Skewness    Kurtosis
## unit_price     701.965331  26.494628 0.007066827 -1.21859143
## quantity         8.546446   2.923431 0.012921628 -1.21554723
## tax            137.096594  11.708825 0.891230392 -0.08188476
```

```
## cogs         54838.637658 234.176510 0.891230392 -0.08188476
## gross_income   137.096594  11.708825 0.891230392 -0.08188476
## rating           2.953518   1.718580 0.008996129 -1.15158684
## total        60459.598018 245.885335 0.891230392 -0.08188476
```

Our measures of spread are as shown above. We can see the variance, std deviation, kurtosis and skewness

Plots

```
# A pie chart showing the distribution of Customers by branch
branch <- data.frame(bran = data$branch)
PieChart(bran, hole = 0, values = "%", data = branch,
         fill = c("lightblue", "pink", "lightgreen" ), main = "Distribution
of Customers by branch")
```

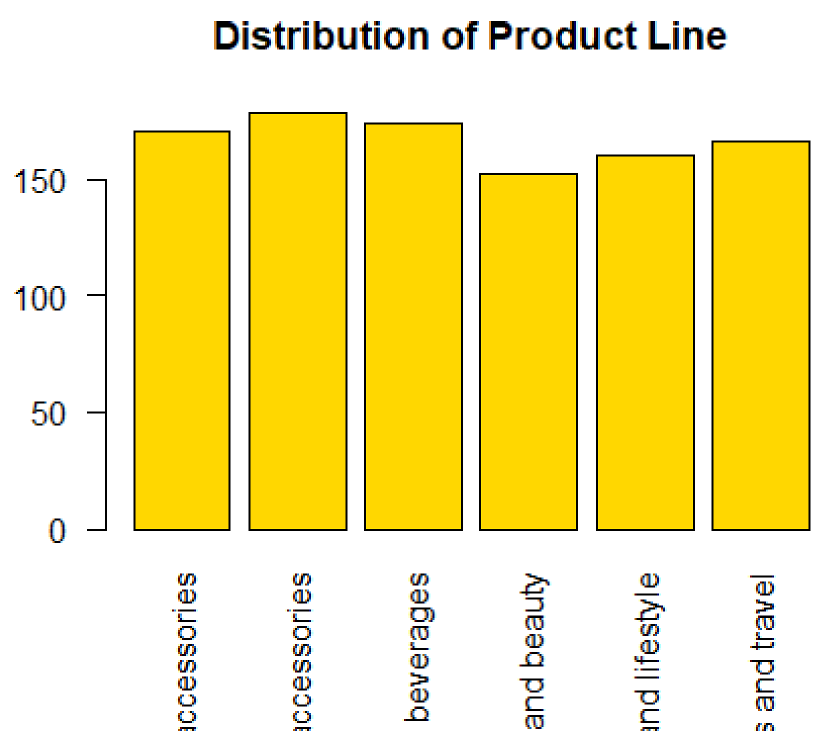



```
## >>> Suggestions
## PieChart(bran, hole=0)  # traditional pie chart
## PieChart(bran, values="%")  # display %'s on the chart
## BarChart(bran)  # bar chart
## Plot(bran)  # bubble plot
## Plot(bran, values="count")  # lollipop plot
##
##
## --- bran ---
##
##
```

```
##                        A       B       C       Total
## Frequencies:      340     332     328        1000
## Proportions:   0.340   0.332   0.328       1.000
##
##
## Chi-squared test of null hypothesis of equal probabilities
##   Chisq = 0.224, df = 2, p-value = 0.894
```

34% of the customers were from branch A, while branch B & C both had 33% of the total customers.

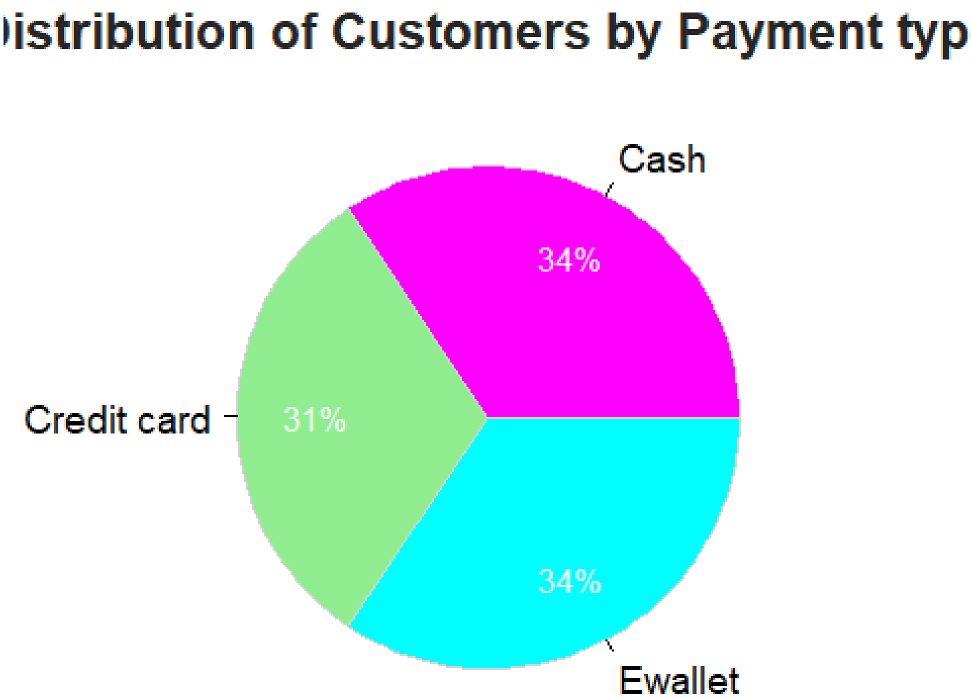
```
# A bar plot showing the distribution of products by product line
barplot(table(data$product_line), col = "gold", main = "Distribution of
Product Line", las=2)
```




```
tabyl(data$product_line, sort = TRUE)

##       data$product_line   n percent
##  Electronic accessories 170   0.170
##     Fashion accessories 178   0.178
##      Food and beverages 174   0.174
##       Health and beauty 152   0.152
##      Home and lifestyle 160   0.160
##       Sports and travel 166   0.166
```

Fashion accessories had the highest frequency at 17.8%, followed by Food and bevarages product line at 17.4 %. Health and beauty had the lowest frequency at 15.2%, followed by Home and lifestyle at 16%

```
# A pie chart showing the distribution of Customers by Payment type
payment <- data.frame(pay = data$payment)
PieChart(pay, hole = 0, values = "%", data = payment,
         fill = c("magenta", "lightgreen", "cyan" ), main = "Distribution of
Customers by Payment type")
```

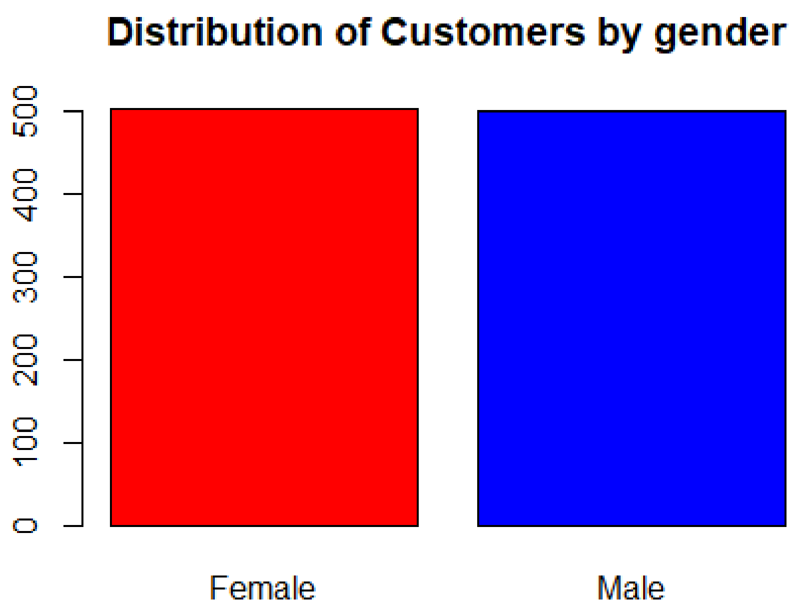



```
## >>> Suggestions
## PieChart(pay, hole=0)  # traditional pie chart
## PieChart(pay, values="%")  # display %'s on the chart
## BarChart(pay)  # bar chart
## Plot(pay)  # bubble plot
## Plot(pay, values="count")  # lollipop plot
##
##
## --- pay ---
##
##
##                    Cash  Credit card  Ewallet      Total
## Frequencies:       344          311      345       1000
## Proportions:     0.344        0.311    0.345      1.000
##
##
```

```
## Chi-squared test of null hypothesis of equal probabilities
##   Chisq = 2.246, df = 2, p-value = 0.325
```

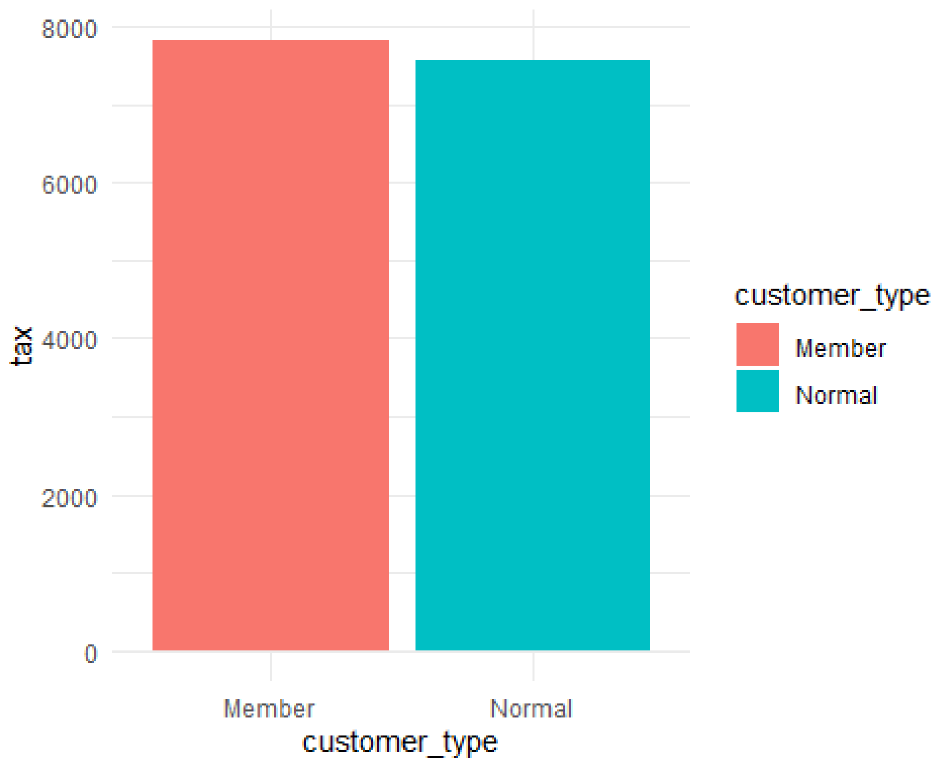
Cash and e-wallet had an equal proportion of customers at 34% while credit cards were 31% of the total dataset.

```
# A bar plot showing the distribution of Customers by Gender
barplot(table(data$gender), col = c("red","blue"), main = "Distribution of Customers by gender")
```



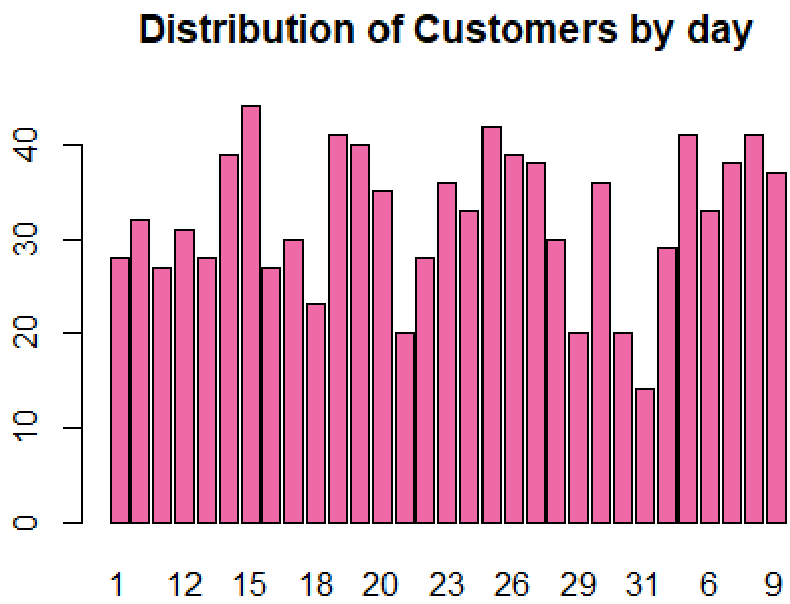
There was an equal distribution of male and female customers.

```
# A bar plot showing the distribution of Customers by Customer type
customer_type <-ggplot(data, aes(x=customer_type, y = tax,
fill=customer_type)) +
  geom_bar(stat="identity")+theme_minimal()
customer_type
```



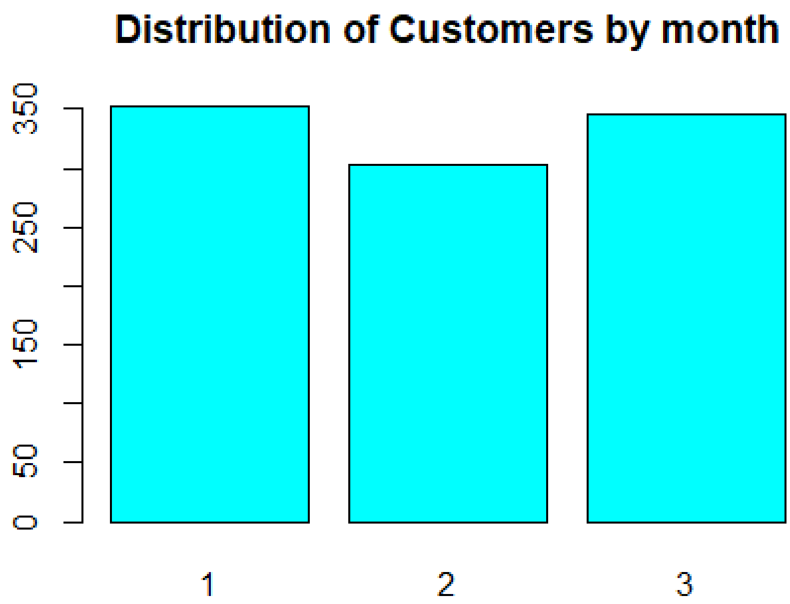
Carrefour members are slightly higher than the normal customer type.

```
#Distribution of Customers by day
plot(data$day, col = "hotpink2", main = "Distribution of Customers by day")
```



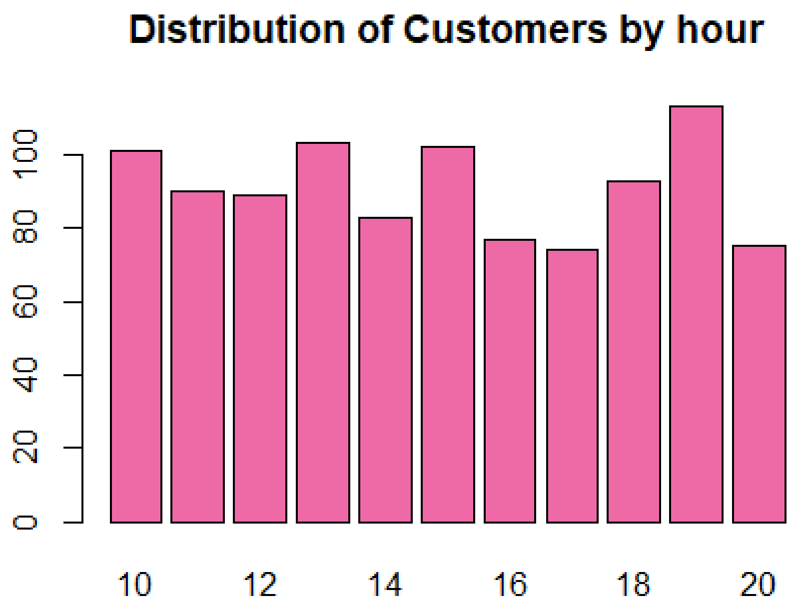
Most sales happened on the 15th(which was quite surprising), followed by the 25th. This could be attributed to restocking of monthly supplies due to payday.

```
#Distribution of Customers by month
plot(data$month, col = "cyan", main = "Distribution of Customers by month")
```



January had the highest number of sales as compared to the rest of the months, which may be attributed to back-to-school shenigans, as well as people restocking supplies at the beginning of the year. then March. February had the least amount of sales of all months.

```
#Distribution of Customers by hour of day
plot(data$hour, col = "hotpink2", main = "Distribution of Customers by hour")
```



Most customers come in at 7pm, probably as they are headed home after work. The second highest traffic rate is at 1pm, probably to get items for lunch. The least amount of traffic is at 5pm.

Implementing the solution

```
#Creating a copy of the dataset
data_copy <- data

#Feature Engineering
data_copy$branch1<-as.integer(factor(data_copy$branch))
data_copy$customer_type1<-as.integer(factor(data_copy$customer_type))
data_copy$gender1<-as.integer(factor(data_copy$gender))
data_copy$productline<-as.integer(factor(data_copy$product_line))
data_copy$payment1<- as.integer(factor(data_copy$payment))
drop_cols = c('branch', 'customer_type',    'gender',   'product_line',
'payment','year','day', 'month', 'hour', 'minutes')
data_copy<-data_copy%>%select(-drop_cols)

## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(drop_cols)` instead of `drop_cols` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.

head(data_copy)
```

```
##    unit_price quantity     tax   cogs gross_income rating    total branch1
## 1:      74.69        7 26.1415 522.83      26.1415    9.1 548.9715       1
## 2:      15.28        5  3.8200  76.40       3.8200    9.6  80.2200       3
## 3:      46.33        7 16.2155 324.31      16.2155    7.4 340.5255       1
## 4:      58.22        8 23.2880 465.76      23.2880    8.4 489.0480       1
## 5:      86.31        7 30.2085 604.17      30.2085    5.3 634.3785       1
## 6:      85.39        7 29.8865 597.73      29.8865    4.1 627.6165       3
##    customer_type1 gender1 productline payment1
## 1:              1       1           4        3
## 2:              2       1           1        1
## 3:              2       2           5        2
## 4:              1       2           4        3
## 5:              2       2           6        3
## 6:              2       2           1        3
```

Dimensionality Reduction

```
colnames(data_copy)
```

```
##  [1] "unit_price"     "quantity"       "tax"            "cogs"
##  [5] "gross_income"   "rating"         "total"          "branch1"
##  [9] "customer_type1" "gender1"        "productline"    "payment1"
```

PCA

We shall exclude categorical variables since PCA works best with numerical data

```
#passing dataset to prcomp
data_copy.pca <- prcomp(data_copy[,c(1,3,4,5,7)], center = TRUE, scale= TRUE)
#previewing the object
summary(data_copy.pca)
```

```
## Importance of components:
##                           PC1    PC2                  PC3
## Standard deviation     2.1128 0.7321 0.0000000000000002582
## Proportion of Variance 0.8928 0.1072 0.0000000000000000000
## Cumulative Proportion  0.8928 1.0000 1.0000000000000000000
##                                          PC4                  PC5
## Standard deviation     0.0000000000000001735 0.0000000000000001036
## Proportion of Variance 0.0000000000000000000 0.0000000000000000000
## Cumulative Proportion  1.0000000000000000000 1.0000000000000000000
```

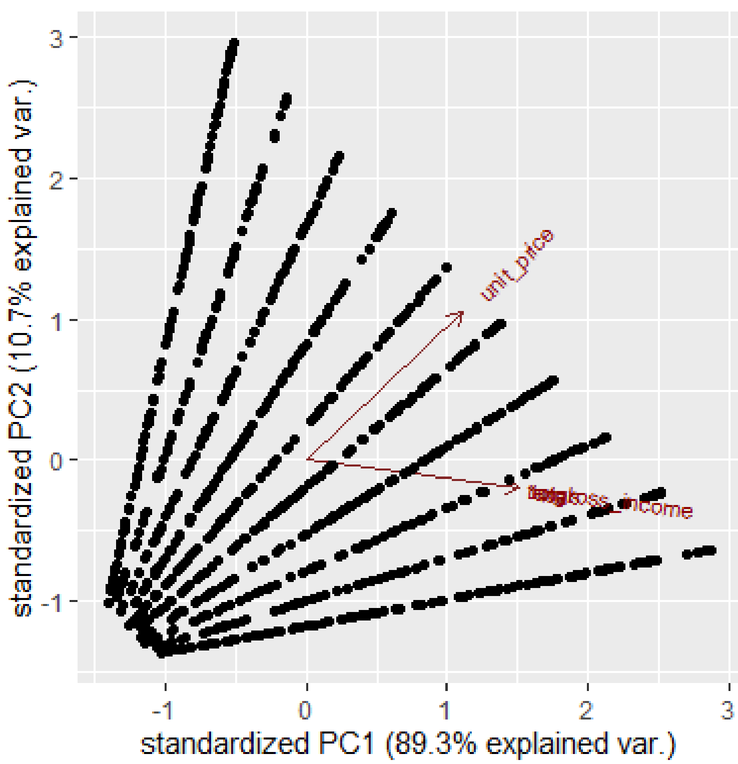
PC1 explains 89% of the variation in the dataset. PC2 explains 10% of the variation.

```
#call str to have a look at the PCA object
str(data_copy.pca)
```

```
## List of 5
##  $ sdev    : num [1:5] 2.112838248976040223 0.732061837322408149
0.000000000000000258 0.000000000000000174 0.000000000000000104
##  $ rotation: num [1:5, 1:5] 0.344 0.47 0.47 0.47 0.47 ...
##   ..- attr(*, "dimnames")=List of 2
```

```
##   .. ..$ : chr [1:5] "unit_price" "tax" "cogs" "gross_income" ...
##   .. ..$ : chr [1:5] "PC1" "PC2" "PC3" "PC4" ...
##  $ center  : Named num [1:5] 55.7 15.4 307.6 15.4 323
##   ..- attr(*, "names")= chr [1:5] "unit_price" "tax" "cogs" "gross_income"
...
##  $ scale   : Named num [1:5] 26.5 11.7 234.2 11.7 245.9
##   ..- attr(*, "names")= chr [1:5] "unit_price" "tax" "cogs" "gross_income"
...
##  $ x       : num [1:1000, 1:5] 1.973 -2.3782 0.0129 1.3016 2.7761 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:5] "PC1" "PC2" "PC3" "PC4" ...
##  - attr(*, "class")= chr "prcomp"
```

```
options(repr.plot.height = 20, repr.plot.width = 20)

ggbiplot(data_copy.pca)
```

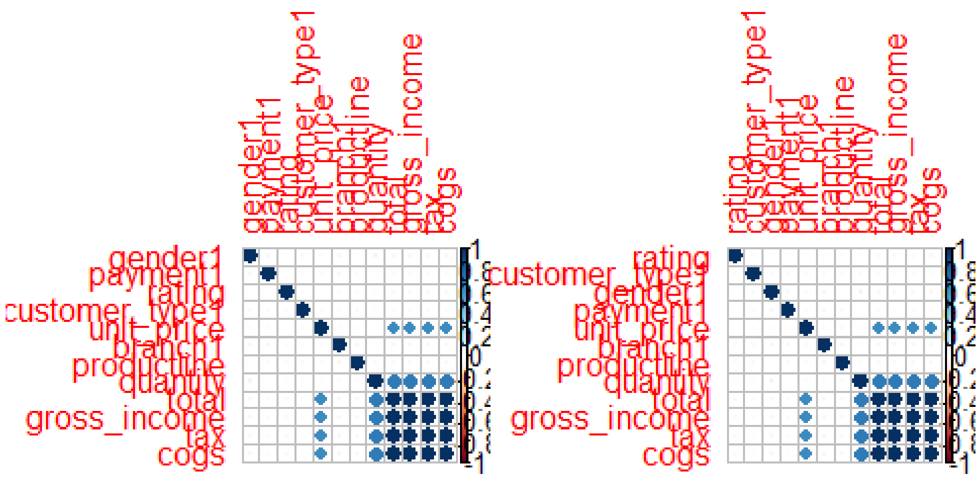


Feature Selection

a) Filter method

```
# Determining the correlated features
# create correlation matrix
correlationMatrix <- cor(data_copy)

# find variables that are highly correlated
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.75) # selects
```

```r
features with more than 0.

# print indexes of highly correlated attributes
highlyCorrelated

## [1] 4 7 3

#Removing redundant features
data_copy2<-data_copy[-highlyCorrelated]

#Performing our graphical comparison of the correlation matrices
par(mfrow = c(1, 2))
corrplot(correlationMatrix, order = "hclust")
corrplot(cor(data_copy2), order = "hclust")
```



b)Wrapper Method

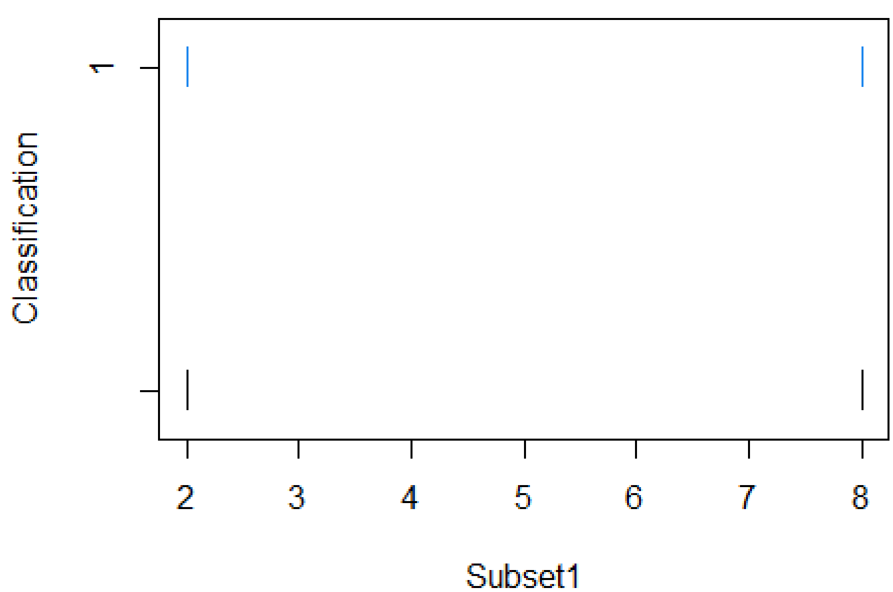
```r
#Sequential forward greedy search (default)
out = clustvarsel(data_copy, G = 1:16)
out

## ----------------------------------------------------
## Variable selection for Gaussian model-based clustering
## Stepwise (forward/backward) greedy search
## ----------------------------------------------------
##
##  Variable proposed Type of step  BICclust Model  G     BICdiff Decision
```

```
##            quantity          Add -4192.156      E  9    804.0515 Accepted
##             branch1          Add 55588.282    EEI 11  62228.4480 Accepted
##            payment1          Add -9066.142    VEV  4 -62175.2469 Rejected
##             branch1       Remove -4192.156      E  9  62228.4480 Rejected
##
## Selected subset: quantity, branch1
```

```
Subset1 = data_copy[,out$subset]
mod = Mclust(Subset1, G = 1:5)
summary(mod)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust X (univariate normal) model with 1 component:
##
##  log-likelihood n df       BIC       ICL
##       -5.035102 2  2 -11.4565 -11.4565
##
## Clustering table:
## 1
## 2
```

```
plot(mod,c("classification"))
```



Part 3

```
#Loading package
library(arules)

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

##
## Attaching package: 'arules'

## The following object is masked from 'package:lessR':
##
##     recode

## The following object is masked from 'package:dplyr':
##
##     recode

## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

Loading the library & data

```
#Loading the dataset

data2 <-read.transactions("C:/Users/user/Downloads/R
Markdowns/Supermarket_Sales_Dataset II.csv", sep = ",")

#Previewing the first 5 transactions
inspect(data2[1:5])

##     items
## [1] {almonds,
##      antioxydant juice,
##      avocado,
##      cottage cheese,
##      energy drink,
##      frozen smoothie,
##      green grapes,
##      green tea,
##      honey,
##      low fat yogurt,
##      mineral water,
##      olive oil,
```

```
##      salad,
##      salmon,
##      shrimp,
##      spinach,
##      tomato juice,
##      vegetables mix,
##      whole weat flour,
##      yams}
## [2] {burgers,
##      eggs,
##      meatballs}
## [3] {chutney}
## [4] {avocado,
##      turkey}
## [5] {energy bar,
##      green tea,
##      milk,
##      mineral water,
##      whole wheat rice}
```

```
# Verifying the object's class to show us transactions as the type of data
that we will need
# ---
#
class(data2)
```

```
## [1] "transactions"
## attr(,"package")
## [1] "arules"
```

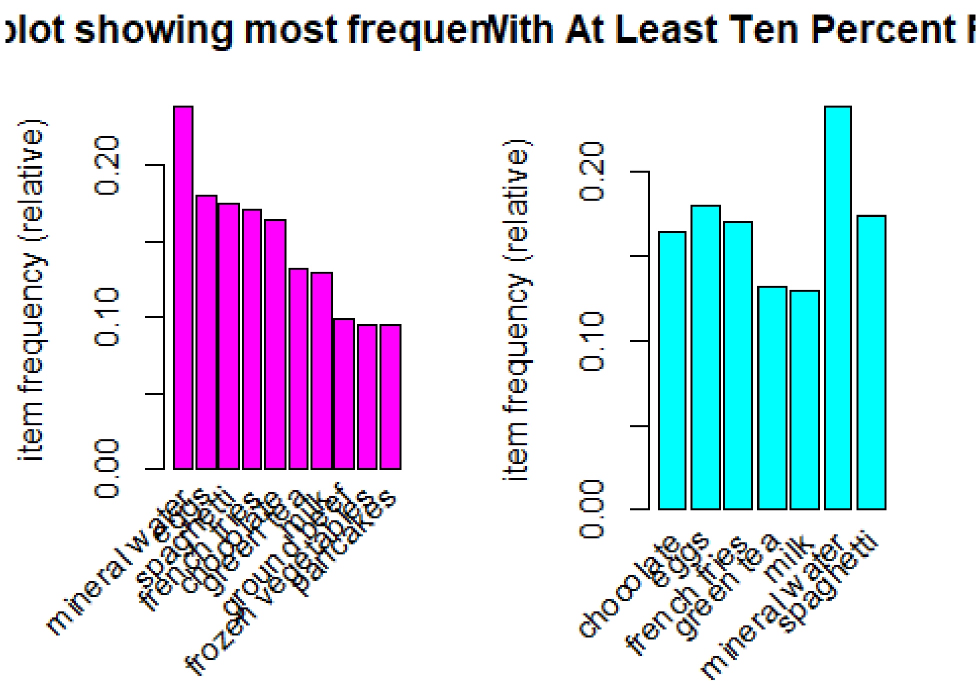
```
#Generating a summary of the transactions to give us some information on the
most purchased items, distribution of the item sets (no. of items purchased
in each transaction), etc.

summary(data2)
```

```
## transactions as itemMatrix in sparse format with
##  7501 rows (elements/itemsets/transactions) and
##  119 columns (items) and a density of 0.03288973
##
## most frequent items:
## mineral water          eggs     spaghetti  french fries     chocolate
##          1788          1348          1306          1282          1229
##       (Other)
##         22405
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
16
## 1754 1358 1044  816  667  493  391  324  259  139  102   67   40   22   17
```

```
4
##   18   19   20
##    1    2    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   3.914   5.000  20.000
##
## includes extended item information - examples:
##              labels
## 1           almonds
## 2 antioxydant juice
## 3         asparagus
```

The top 5 most frequently bought items are mineral water, eggs, spaghetti, french fries & chocolate

```
#Exploring the frequency of transactions
itemFrequency(data2[, 8:10],type = "absolute")

##   black tea blueberries  body spray
##         107          69          86

round(itemFrequency(data2[, 8:10],type = "relative")*100,2)

##   black tea blueberries  body spray
##        1.43        0.92        1.15

#Displaying top 10 most common items in the transactions dataset and the
items whose relative importance is at least 10%

par(mfrow = c(1, 2))

# plot the frequency of items
itemFrequencyPlot(data2, topN = 10,col="magenta", main = "Frequency plot
showing most frequently bought items")
itemFrequencyPlot(data2, support = 0.1,col="cyan", main = "Items With At
Least Ten Percent Frequency ")
```




```
#Checking for the 10 least popular items
least_items = itemFrequency(data2, type = "relative")
head(sort(least_items), 10)

##      water spray          napkins            cream          bramble
tea
##     0.0003999467     0.0006665778     0.0009332089     0.0018664178
0.0038661512
##          chutney   mashed potato chocolate bread     dessert wine
ketchup
##     0.0041327823     0.0041327823     0.0042660979     0.0043994134
0.0043994134
```

Building a model based on association rules.

```
#using the apriori function

# We use Min Support as 0.001 and confidence as 0.8

rules <- apriori (data2, parameter = list(supp = 0.001, conf = 0.8))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
```

```
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE   FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 7
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s].
## sorting and recoding items ... [116 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.01s].
## writing ... [74 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
rules
```

```
## set of 74 rules
```

We use measures of significance and interest on the rules, determining which ones are interesting and which to discard. However since we built the model using 0.001 Min support and confidence as 0.8 we obtained 74 rules.

However, in order to illustrate the sensitivity of the model to these two parameters, we will see what happens if we increase the support or lower the confidence level.

```
# Building a apriori model with Min Support as 0.002 and confidence as 0.8.
rules2 <- apriori (data2,parameter = list(supp = 0.002, conf = 0.8))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   0.002      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE   FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 15
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[119 item(s), 7501 transaction(s)] done [0.01s].
## sorting and recoding items ... [115 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.01s].
## writing ... [2 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
# Building apriori model with Min Support as 0.002 and confidence as 0.6.
rules3 <- apriori (data2, parameter = list(supp = 0.001, conf = 0.6))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 7
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[119 item(s), 7501 transaction(s)] done [0.01s].
## sorting and recoding items ... [116 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.01s].
## writing ... [545 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

rules2

## set of 2 rules

rules3

## set of 545 rules
```

In our first example, we increased the minimum support of 0.001 to 0.002 and model rules went from 271 to only 2. This would lead us to understand that using a high level of support can make the model lose interesting rules.

In the second example, we decreased the minimum confidence level to 0.6 and the number of model rules went from 271 to 545. This would mean that using a low confidence level increases the number of rules to quite an extent and many will not be useful.

We can perform an exploration of our model through the use of the summary function as shown

Upon running the code, the function would give us information about the model i.e. the size of rules, depending on the items that contain these rules.

In our above case, most rules have 3 and 4 items though some rules do have upto 6.

More statistical information such as support, lift and confidence is also provided.

```r
# Generating a summary
summary(rules)

## set of 74 rules
##
## rule length distribution (lhs + rhs):sizes
##  3  4  5  6
## 15 42 16  1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   4.000   4.000   4.041   4.000   6.000
##
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.001067   Min.   :0.8000   Min.   :0.001067   Min.   : 3.356
##  1st Qu.:0.001067   1st Qu.:0.8000   1st Qu.:0.001333   1st Qu.: 3.432
##  Median :0.001133   Median :0.8333   Median :0.001333   Median : 3.795
##  Mean   :0.001256   Mean   :0.8504   Mean   :0.001479   Mean   : 4.823
##  3rd Qu.:0.001333   3rd Qu.:0.8889   3rd Qu.:0.001600   3rd Qu.: 4.877
##  Max.   :0.002533   Max.   :1.0000   Max.   :0.002666   Max.   :12.722
##      count
##  Min.   : 8.000
##  1st Qu.: 8.000
##  Median : 8.500
##  Mean   : 9.419
##  3rd Qu.:10.000
##  Max.   :19.000
##
## mining info:
##   data ntransactions support confidence
##  data2          7501   0.001        0.8

#Observing rules built in our model i.e. first 10 model rules

inspect(rules[1:10])

##      lhs                              rhs             support
confidence
## [1]  {frozen smoothie,spinach}    => {mineral water} 0.001066524 0.8888889
## [2]  {bacon,pancakes}             => {spaghetti}     0.001733102 0.8125000
## [3]  {nonfat milk,turkey}         => {mineral water} 0.001199840 0.8181818
## [4]  {ground beef,nonfat milk}    => {mineral water} 0.001599787 0.8571429
## [5]  {mushroom cream sauce,pasta} => {escalope}      0.002532996 0.9500000
## [6]  {milk,pasta}                 => {shrimp}        0.001599787 0.8571429
## [7]  {cooking oil,fromage blanc}  => {mineral water} 0.001199840 0.8181818
## [8]  {black tea,salmon}           => {mineral water} 0.001066524 0.8000000
## [9]  {black tea,frozen smoothie}  => {milk}          0.001199840 0.8181818
## [10] {red wine,tomato sauce}      => {chocolate}     0.001066524 0.8000000
##      coverage    lift      count
## [1]  0.001199840  3.729058  8
```

```
## [2]  0.002133049  4.666587 13
## [3]  0.001466471  3.432428  9
## [4]  0.001866418  3.595877 12
## [5]  0.002666311 11.976387 19
## [6]  0.001866418 11.995203 12
## [7]  0.001466471  3.432428  9
## [8]  0.001333156  3.356152  8
## [9]  0.001466471  6.313973  9
## [10] 0.001333156  4.882669  8
```

```
rules<-sort(rules, by="confidence", decreasing=TRUE)
inspect(rules[1:20])
```

```
##      lhs                          rhs                 support confidence
coverage      lift count
## [1]  {french fries,
##       mushroom cream sauce,
##       pasta}                   => {escalope}      0.001066524  1.0000000
0.001066524 12.606723     8
## [2]  {ground beef,
##       light cream,
##       olive oil}               => {mineral water} 0.001199840  1.0000000
0.001199840  4.195190     9
## [3]  {cake,
##       meatballs,
##       mineral water}           => {milk}          0.001066524  1.0000000
0.001066524  7.717078     8
## [4]  {cake,
##       olive oil,
##       shrimp}                  => {mineral water} 0.001199840  1.0000000
0.001199840  4.195190     9
## [5]  {mushroom cream sauce,
##       pasta}                   => {escalope}      0.002532996  0.9500000
0.002666311 11.976387    19
## [6]  {red wine,
##       soup}                    => {mineral water} 0.001866418  0.9333333
0.001999733  3.915511    14
## [7]  {eggs,
##       mineral water,
##       pasta}                   => {shrimp}        0.001333156  0.9090909
0.001466471 12.722185    10
## [8]  {herb & pepper,
##       mineral water,
##       rice}                    => {ground beef}   0.001333156  0.9090909
0.001466471  9.252498    10
## [9]  {ground beef,
##       pancakes,
##       whole wheat rice}        => {mineral water} 0.001333156  0.9090909
0.001466471  3.813809    10
## [10] {frozen vegetables,
```

```
##       milk,
##       spaghetti,
##       turkey}               => {mineral water} 0.001199840  0.9000000
0.001333156  3.775671     9
## [11] {chocolate,
##       frozen vegetables,
##       olive oil,
##       shrimp}               => {mineral water} 0.001199840  0.9000000
0.001333156  3.775671     9
## [12] {frozen smoothie,
##       spinach}              => {mineral water} 0.001066524  0.8888889
0.001199840  3.729058     8
## [13] {black tea,
##       spaghetti,
##       turkey}               => {eggs}          0.001066524  0.8888889
0.001199840  4.946258     8
## [14] {light cream,
##       mineral water,
##       shrimp}               => {spaghetti}     0.001066524  0.8888889
0.001199840  5.105326     8
## [15] {cake,
##       meatballs,
##       milk}                 => {mineral water} 0.001066524  0.8888889
0.001199840  3.729058     8
## [16] {grated cheese,
##       mineral water,
##       rice}                 => {ground beef}   0.001066524  0.8888889
0.001199840  9.046887     8
## [17] {cake,
##       olive oil,
##       whole wheat pasta}    => {mineral water} 0.001066524  0.8888889
0.001199840  3.729058     8
## [18] {escalope,
##       hot dogs,
##       mineral water}        => {milk}          0.001066524  0.8888889
0.001199840  6.859625     8
## [19] {brownies,
##       eggs,
##       ground beef}          => {mineral water} 0.001066524  0.8888889
0.001199840  3.729058     8
## [20] {chicken,
##       fresh bread,
##       pancakes}             => {mineral water} 0.001066524  0.8888889
0.001199840  3.729058     8
```

The results reveal that the model is 100% confident that a person buying french fries, mushroom cream sauce and pasta will buy escalope, 91% confident that a person buying eggs, mineral water and pasta will buy shrimp or 89% confident that a person buying brownies, eggs & ground beef will buy mineral water, etc,.

```
# If we're interested in making a promotion  and we wanted to determine the
items that customers buying shrimps might buy

# Subset the rules
shrimp <- subset(rules, subset = lhs %pin% "shrimp")

# Order by confidence
shrimp<-sort(shrimp, by="confidence", decreasing=TRUE)

# inspect top 5
inspect(shrimp[1:5])

##     lhs                        rhs                 support confidence
coverage     lift count
## [1] {cake,
##      olive oil,
##      shrimp}                => {mineral water} 0.001199840  1.0000000
0.001199840 4.195190     9
## [2] {chocolate,
##      frozen vegetables,
##      olive oil,
##      shrimp}                => {mineral water} 0.001199840  0.9000000
0.001333156 3.775671     9
## [3] {light cream,
##      mineral water,
##      shrimp}                => {spaghetti}     0.001066524  0.8888889
0.001199840 5.105326     8
## [4] {ground beef,
##      salmon,
##      shrimp}                => {spaghetti}     0.001066524  0.8888889
0.001199840 5.105326     8
## [5] {escalope,
##      french fries,
##      shrimp}                => {chocolate}     0.001066524  0.8888889
0.001199840 5.425188     8
```

Since there is a 100% chance that one buying shrimp will pick it up with cake, olive oil and mineral water, the supermarket can bundle these up during promotion season. Or showcase offers for these products on their apps for users who've previously bought shrimp.

Part 4

Loading the packages

```
library(anomalize)

## == Use anomalize to improve your Forecasts by 50%!
=============================
```

```
## Business Science offers a 1-hour course - Lab #18: Time Series Anomaly Detection!
## </> Learn more at: https://university.business-science.io/p/learning-labs-pro </>

library(tidyverse)
library(tibble)
```

Loading the data

```
data3 = fread('http://bit.ly/CarreFourSalesDataset')
head(data3)

##         Date    Sales
## 1:  1/5/2019 548.9715
## 2:  3/8/2019  80.2200
## 3:  3/3/2019 340.5255
## 4: 1/27/2019 489.0480
## 5:  2/8/2019 634.3785
## 6: 3/25/2019 627.6165

#Checking the shape of the dataset
dim(data3)

## [1] 1000    2
```

The dataset has 1000 entries and 2 rows

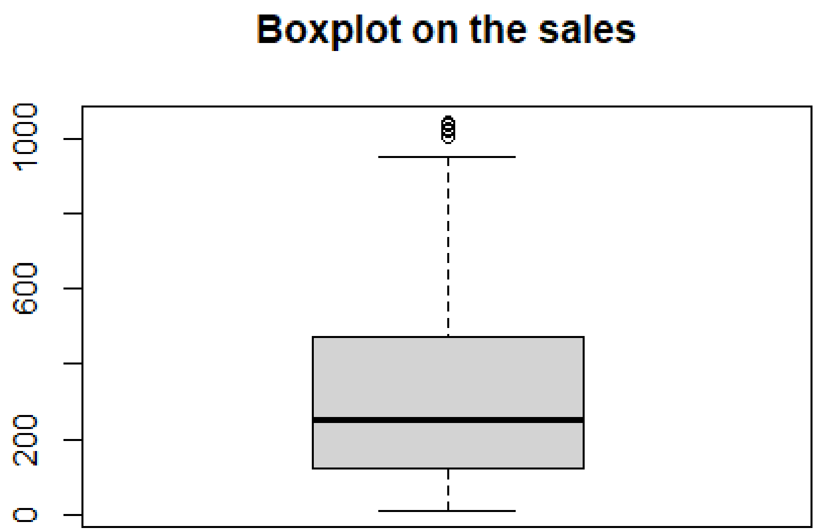
```
#Checking for missing values
colSums(is.na(data3))

##  Date Sales
##     0     0
```

No missing values in our dataset

```
#Checking for duplicates
length(which(duplicated(data3)))

## [1] 0
```

No duplicates in our dataset

```
#Checking for outliers
boxplot(data3$Sales, main= 'Boxplot on the sales')
```



There a few outliers on the sales column. We won't remove them since it is possible some of the sales figures are quite high.

```
#Checking for anomalies
print(unique(data3$Date))
```

```
##  [1] "1/5/2019"  "3/8/2019"  "3/3/2019"  "1/27/2019" "2/8/2019"
"3/25/2019"
##  [7] "2/25/2019" "2/24/2019" "1/10/2019" "2/20/2019" "2/6/2019"
"3/9/2019"
## [13] "2/12/2019" "2/7/2019"  "3/29/2019" "1/15/2019" "3/11/2019"
"1/1/2019"
## [19] "1/21/2019" "3/5/2019"  "3/15/2019" "2/17/2019" "3/2/2019"
"3/22/2019"
## [25] "3/10/2019" "1/25/2019" "1/28/2019" "1/7/2019"  "3/23/2019"
"1/17/2019"
## [31] "2/2/2019"  "3/4/2019"  "3/16/2019" "2/27/2019" "2/10/2019"
"3/19/2019"
## [37] "2/3/2019"  "3/7/2019"  "2/28/2019" "3/27/2019" "1/20/2019"
"3/12/2019"
## [43] "2/15/2019" "3/6/2019"  "2/14/2019" "3/13/2019" "1/24/2019"
"1/6/2019"
## [49] "2/11/2019" "1/22/2019" "1/13/2019" "1/9/2019"  "1/12/2019"
"1/26/2019"
## [55] "1/23/2019" "2/23/2019" "1/2/2019"  "2/9/2019"  "3/26/2019"
"3/1/2019"
## [61] "2/1/2019"  "3/28/2019" "3/24/2019" "2/5/2019"  "1/19/2019"
```

```
"1/16/2019"
## [67] "1/8/2019"  "2/18/2019" "1/18/2019" "2/16/2019" "2/22/2019"
"1/29/2019"
## [73] "1/4/2019"  "3/30/2019" "1/30/2019" "1/3/2019"  "3/21/2019"
"2/13/2019"
## [79] "1/14/2019" "3/18/2019" "3/20/2019" "2/21/2019" "1/31/2019"
"1/11/2019"
## [85] "2/26/2019" "3/17/2019" "3/14/2019" "2/4/2019"  "2/19/2019"
```

From the 89 unique values, we can see there are no anomalies in our data

```
#Checking the datatype
str(data3)

## Classes 'data.table' and 'data.frame':   1000 obs. of  2 variables:
##  $ Date : chr  "1/5/2019" "3/8/2019" "3/3/2019" "1/27/2019" ...
##  $ Sales: num  549 80.2 340.5 489 634.4 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The date is a character, thus we will have to convert it

```
#Converting the date to date datatype
data3$Date = as.Date(data3$Date, format = "%m/%d/%y")

#Confirm the change
str(data3)

## Classes 'data.table' and 'data.frame':   1000 obs. of  2 variables:
##  $ Date : Date, format: "2020-01-05" "2020-03-08" ...
##  $ Sales: num  549 80.2 340.5 489 634.4 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

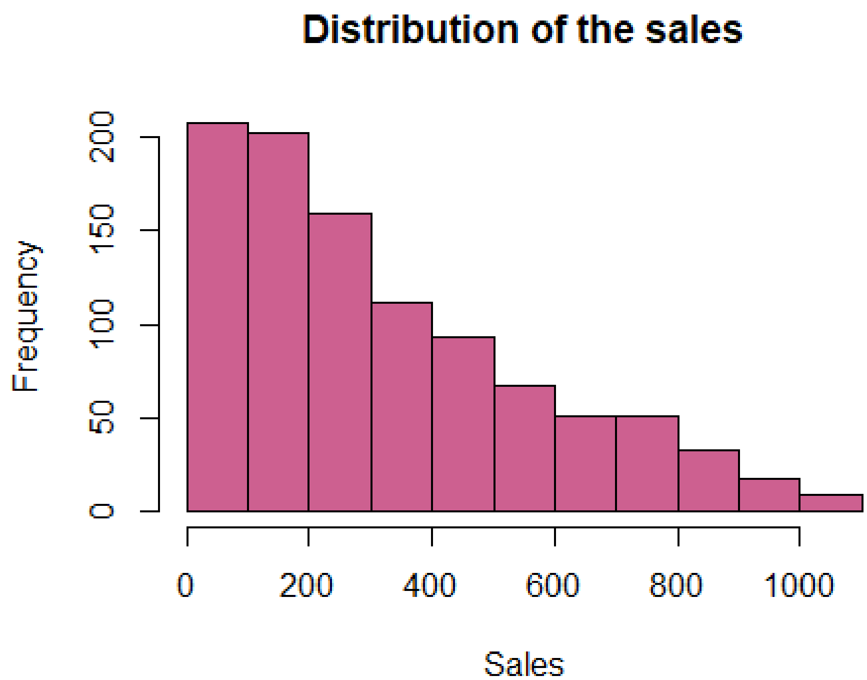
Explorative Data Analysis

a)Univariate analysis

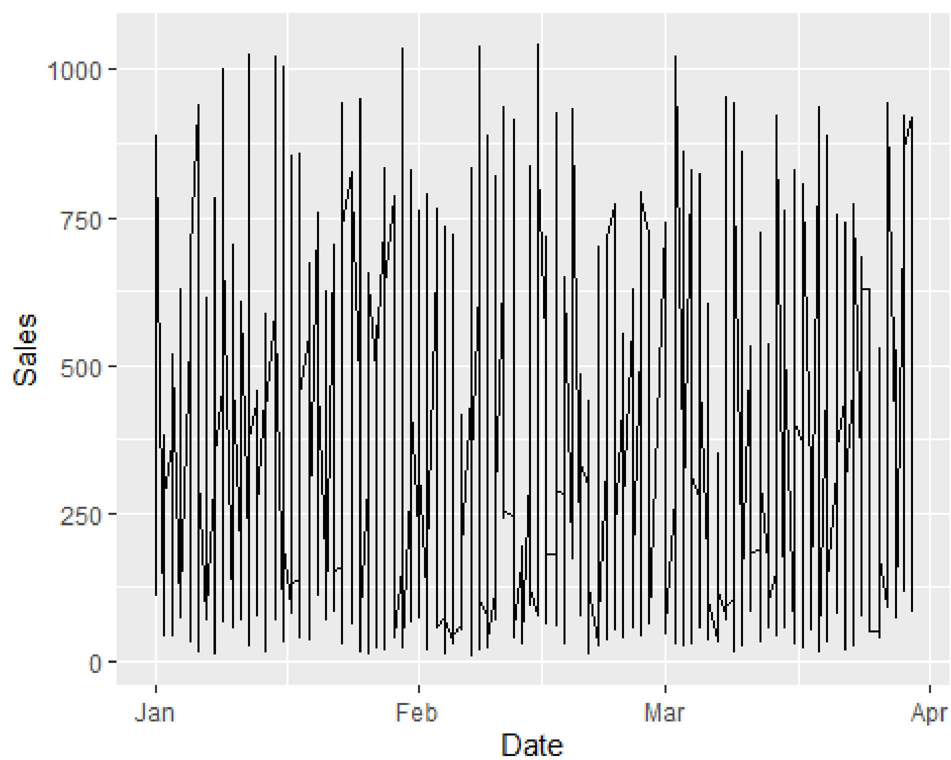
```
summary(data3)

##       Date                Sales
##  Min.   :2020-01-01   Min.   :  10.68
##  1st Qu.:2020-01-24   1st Qu.: 124.42
##  Median :2020-02-13   Median : 253.85
##  Mean   :2020-02-14   Mean   : 322.97
##  3rd Qu.:2020-03-08   3rd Qu.: 471.35
##  Max.   :2020-03-30   Max.   :1042.65
```

We can see the minimum sale was at 10.68, with the highest amount being at 1042.65. Our mean sale amount was 322.97 while the median was 253.85.

```
#Visualizing the distribution of Sales column
hist(data3$Sales, col = 'hotpink3', main = "Distribution of the sales", xlab
= "Sales")
```




```
#Visualizing the distribution of the date column
ggplot(data3, aes(x=Date, y=Sales)) + geom_line()
```



Anomalies in Transactions

```
#Sorting dates in ascending order
data3  = data3[order(data3$Date),]

#Converting the dataset to tibble
data3_tb <- as_tibble(data3)
head(data3_tb)
```

```
## # A tibble: 6 x 2
##   Date       Sales
##   <date>     <dbl>
## 1 2020-01-01  457.
## 2 2020-01-01  400.
## 3 2020-01-01  471.
## 4 2020-01-01  388.
## 5 2020-01-01  133.
## 6 2020-01-01  132.
```

```
#Grouping the daily transactions
data3_count <- data3 %>% group_by(Date) %>% tally()
colnames(data3_count) <- c('Date', 'Count')
head(data3_count)
```

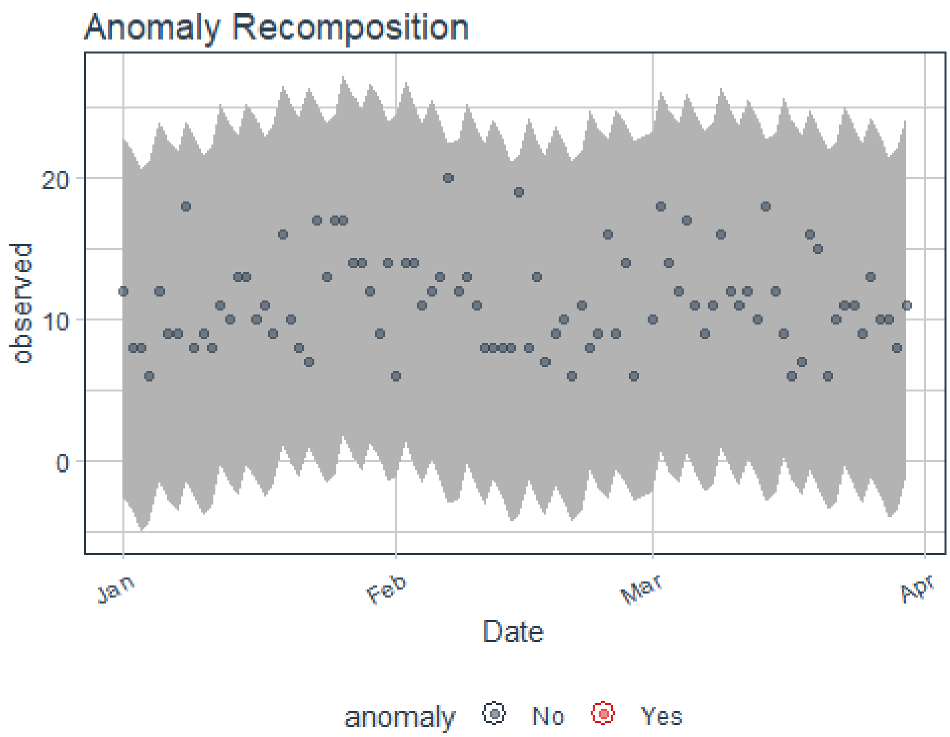
```
## # A tibble: 6 x 2
##   Date       Count
##   <date>     <int>
## 1 2020-01-01    12
## 2 2020-01-02     8
## 3 2020-01-03     8
## 4 2020-01-04     6
## 5 2020-01-05    12
## 6 2020-01-06     9
```

We can see the total number of sales daily. There were 12 sales on 1/1/2020, 8 sales on 2/1/2020 etc

```
#Visualizing our data
data3_count %>%
    time_decompose(Count) %>%
    anomalize(remainder) %>%
    time_recompose() %>%
    plot_anomalies(time_recomposed = TRUE, ncol = 3, alpha_dots = 0.5) +
    ggtitle( "Anomaly Recomposition")
```

```
## Converting from tbl_df to tbl_time.
## Auto-index message: index = Date

## frequency = 7 days

## trend = 30 days
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```



There were no anomalies detected in the number of daily transactions done.