Carrefour

Faith

9/17/2020

RESEARCH QUESTION

You are a Data analyst at Carrefour Kenya and are 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). Your project has been divided into four parts where you'll explore a recent marketing dataset by performing various unsupervised learning techniques and later providing recommendations based on your insights.

PROBLEM STATEMENT

To identify on the most important market strategies that will result to high number of sales.

METRIC FOR SUCCESS

I. Perform PCA II.Perform feature selection and identify highly correlated values and drop them. III. Perform Association analysis to identify products that were bought together. Iv. Perform Anomaly detection to identify unusual sales

EXPERIMENTA DESIGN

I. Load the data II. Check for outliers and missing values III. Perform PCA/t-SNE, Feature selection, Association detection and Anomaly detection IV. Conclusions and Recommendations

```
getwd()
```

```
## [1] "C:/Users/FGakori/Documents/carrefourr"
```

1. DIMENSIONALITY REDUCTION AND FEATURE SELECTION

```
# load the libraries
library(tidyverse)
```

```
## -- Attaching packages ------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ---------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

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

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

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

## Loading required package: scales

##
## Attaching package: 'scales'
```

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

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

## Loading required package: grid
```

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

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

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

Load the Data

```
# Load the csv file
sales <- read.csv('Sales Data.csv')
```

```
# view the first five observations
head(sales)
```

```
##    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
```

```
#shape of our dataset
dim(sales)
```

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

The sales dataset contains 1000 observations and 16 columns

```
# variables datatypes
sapply(sales, class)
```

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

```
#listing all the columns
colnames(sales)
```

```
##  [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"
```

```
# statistical summary

summary(sales)
```

```
##    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
```

```
# check for missing vales
colSums(is.na(sales))
```

```
##              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 missing values in the dataset

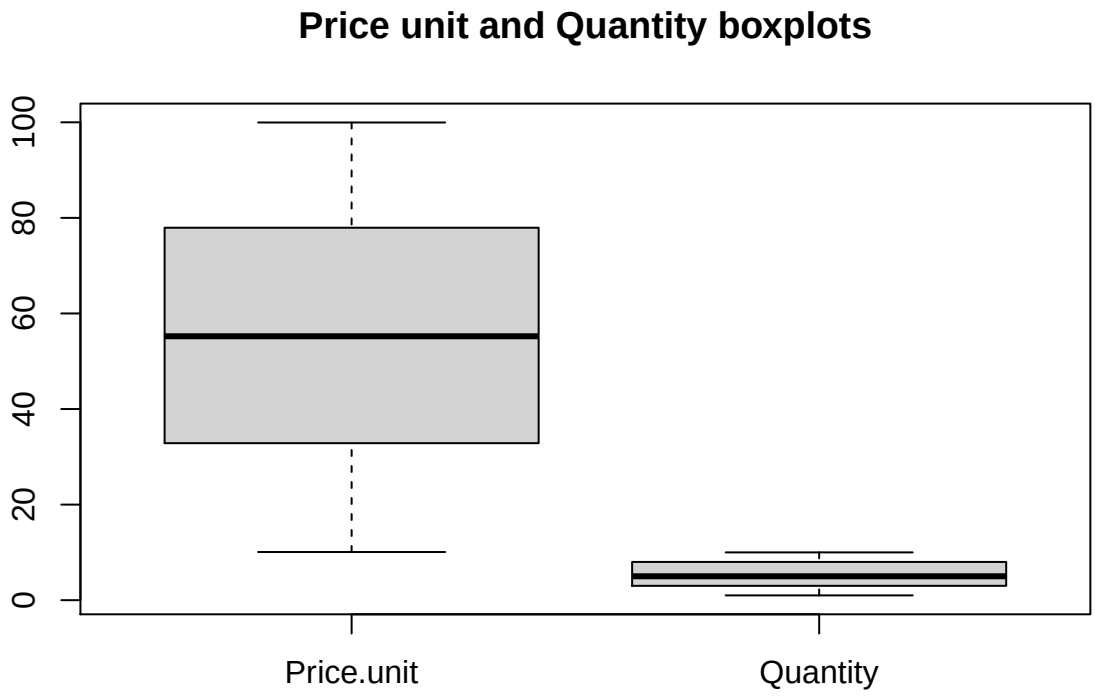
```
# check for duplicates
anyDuplicated(sales)
```

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

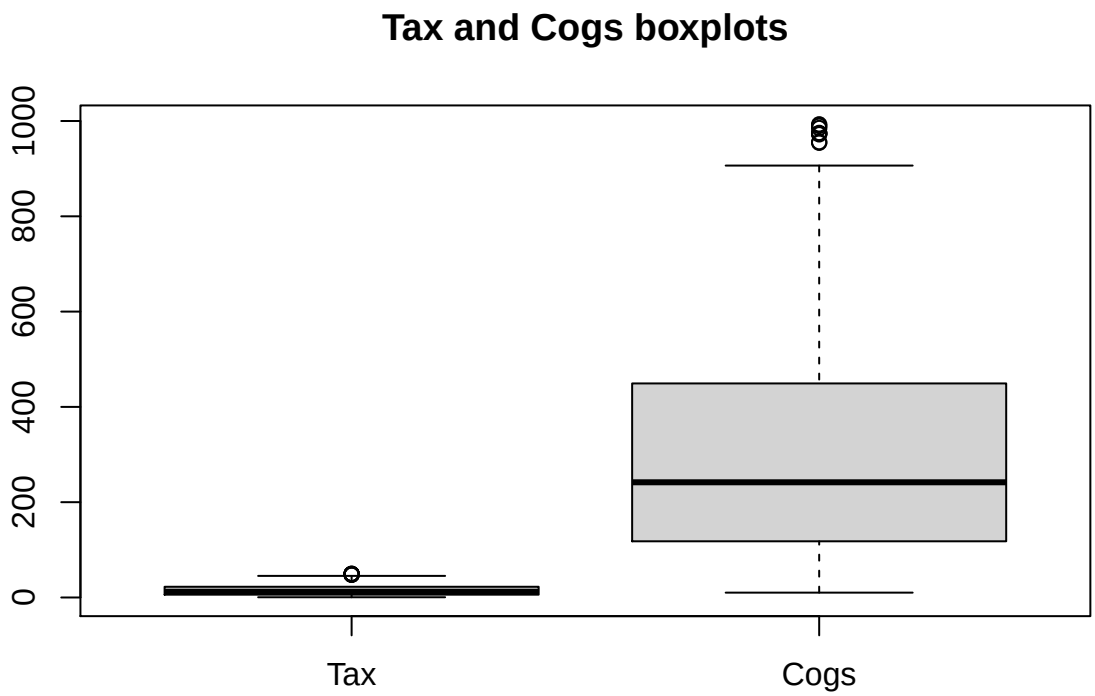
No duplicates

```
# check for outliers
# price and quantity boxplots
```

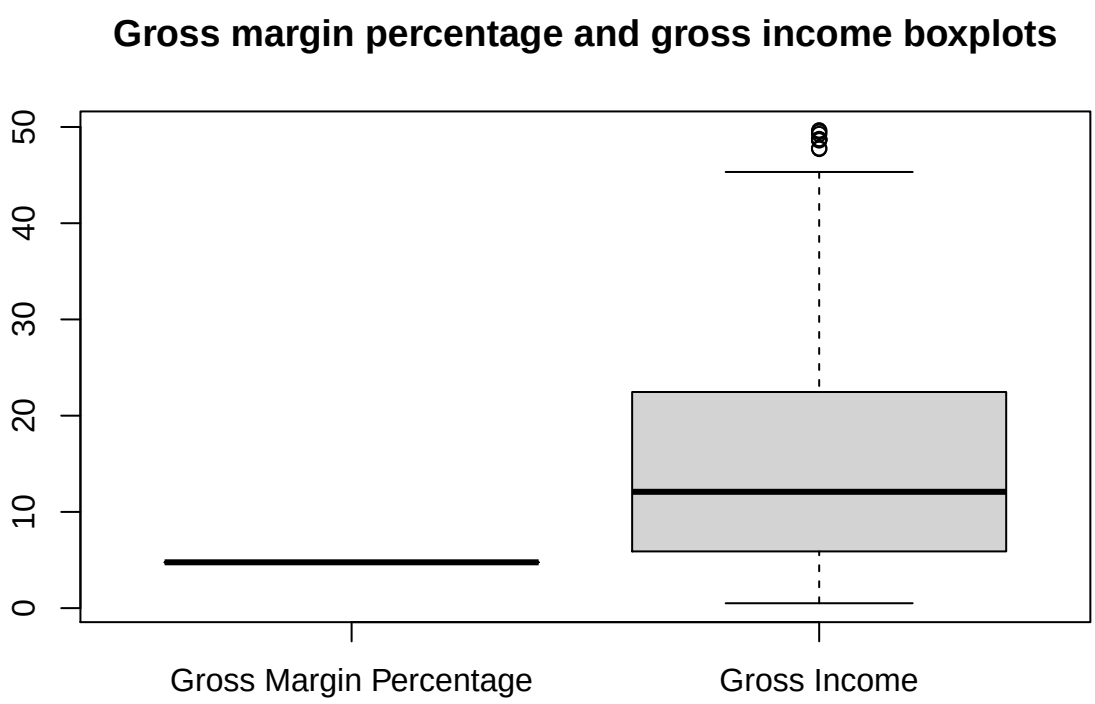
```r
price <- sales$Unit.price
quantity <- sales$Quantity

boxplot(price, quantity, main='Price unit and Quantity boxplots', names = c('Price.unit', 'Quantity'))
```



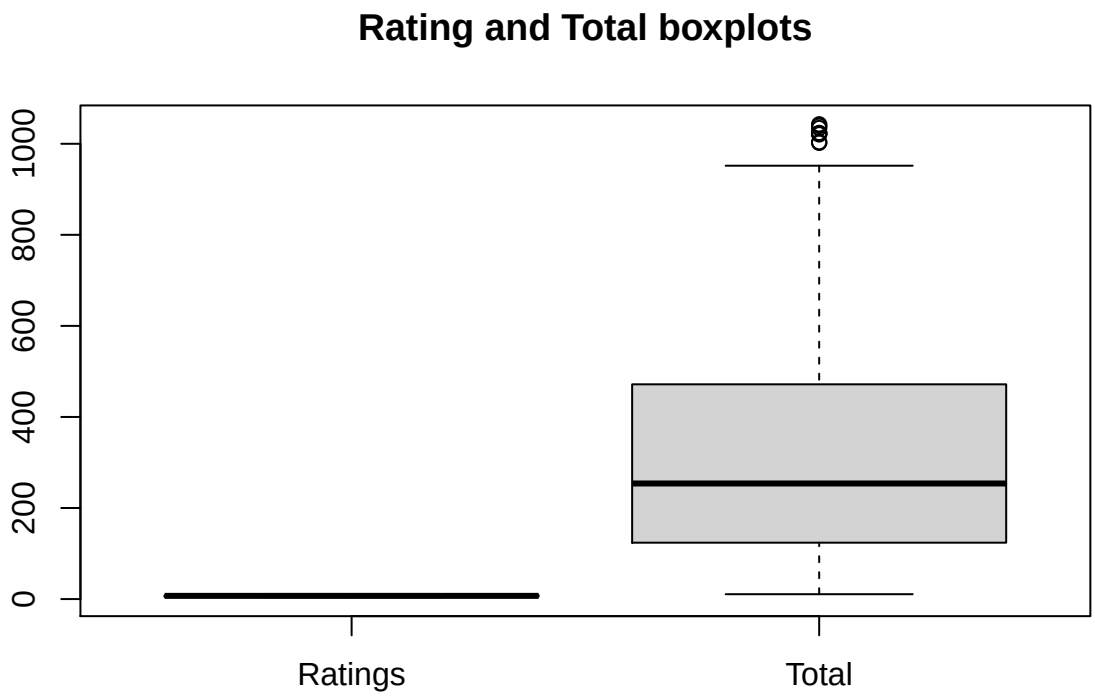

```r
# tax and cogs boxplots

tax <- sales$Tax
cogs <- sales$cogs

boxplot(tax, cogs, main='Tax and Cogs boxplots', names = c('Tax', 'Cogs'))
```




```
# gross margin and gross income

grossmargin <- sales$gross.margin.percentage
grossincome <- sales$gross.income

boxplot(grossmargin, grossincome, main='Gross margin percentage and gross income boxplots', names = c('
```




```
# price and quantity boxplots

rating <- sales$Rating
total <- sales$Total

boxplot(rating, total, main='Rating and Total boxplots', names = c('Ratings', 'Total'))
```



Encoding categorical columns

```
# convert into factor
sales$Branch <- as.integer(as.factor(sales$Branch))
sales$Customer.type <- as.integer(as.factor(sales$Customer.type))
sales$Gender <- as.integer(as.factor(sales$Gender))
sales$Product.line <- as.integer(as.factor(sales$Product.line))
sales$Payment <- as.integer(as.factor(sales$Payment))
```

```
# review the converted columns
sapply(sales, class)
```

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

Converting date column into year, month and day

```
sales <- sales %>% mutate(Date = lubridate::mdy(Date),
              year = lubridate::year(Date),
              month = lubridate::month(Date),
              day = lubridate::day(Date))
```

split time into hours and minutes

```
sales <- sales %>% separate(Time, c("hours", "minutes"), sep = "(?=\\d{2}$)")
head(sales)
```

```
##    Invoice.ID Branch Customer.type Gender Product.line Unit.price Quantity
## 1 750-67-8428      1             1      1            4      74.69        7
## 2 226-31-3081      3             2      1            1      15.28        5
## 3 631-41-3108      1             2      2            5      46.33        7
## 4 123-19-1176      1             1      2            4      58.22        8
## 5 373-73-7910      1             2      2            6      86.31        7
## 6 699-14-3026      3             2      2            1      85.39        7
##       Tax       Date hours minutes Payment   cogs gross.margin.percentage
## 1 26.1415 2019-01-05   13:      08       3 522.83                4.761905
## 2  3.8200 2019-03-08   10:      29       1  76.40                4.761905
## 3 16.2155 2019-03-03   13:      23       2 324.31                4.761905
## 4 23.2880 2019-01-27   20:      33       3 465.76                4.761905
## 5 30.2085 2019-02-08   10:      37       3 604.17                4.761905
## 6 29.8865 2019-03-25   18:      30       3 597.73                4.761905
##   gross.income Rating    Total year month day
## 1      26.1415    9.1 548.9715 2019     1   5
## 2       3.8200    9.6  80.2200 2019     3   8
## 3      16.2155    7.4 340.5255 2019     3   3
## 4      23.2880    8.4 489.0480 2019     1  27
## 5      30.2085    5.3 634.3785 2019     2   8
## 6      29.8865    4.1 627.6165 2019     3  25
```

```
#drop the date column
sales$Date<-NULL
```

```
colSums(is.na(sales))
```

```
##              Invoice.ID                  Branch           Customer.type
##                       0                       0                       0
##                  Gender           Product.line              Unit.price
##                       0                       0                       0
##                Quantity                     Tax                   hours
##                       0                       0                       0
##                 minutes                 Payment                    cogs
##                       0                       0                       0
## gross.margin.percentage            gross.income                  Rating
##                       0                       0                       0
##                   Total                    year                   month
##                       0                       0                       0
##                     day
##                       0
```

```
sapply(sales, class)
```

```
##               Invoice.ID                   Branch            Customer.type
##              "character"                "integer"                "integer"
##                   Gender             Product.line               Unit.price
##                "integer"                "integer"                "numeric"
##                 Quantity                      Tax                    hours
##                "integer"                "numeric"              "character"
##                  minutes                  Payment                     cogs
##              "character"                "integer"                "numeric"
## gross.margin.percentage            gross.income                   Rating
##                "numeric"                "numeric"                "numeric"
##                    Total                     year                    month
##                "numeric"                "numeric"                "numeric"
##                      day
##                "integer"
```

Select numeric columns

```
sales_df <- subset(sales, select = c(Branch, Customer.type, Gender, Product.line, Unit.price, Payment, 
names(sales_df)
```

```
##  [1] "Branch"        "Customer.type" "Gender"        "Product.line"
##  [5] "Unit.price"    "Payment"       "Quantity"      "Tax"
##  [9] "cogs"          "gross.income"  "Rating"        "Total"
## [13] "month"         "day"
```

1. PCA

```
supply_pca <- prcomp(sales_df, center = TRUE, scale. = TRUE)
summary(supply_pca)
```

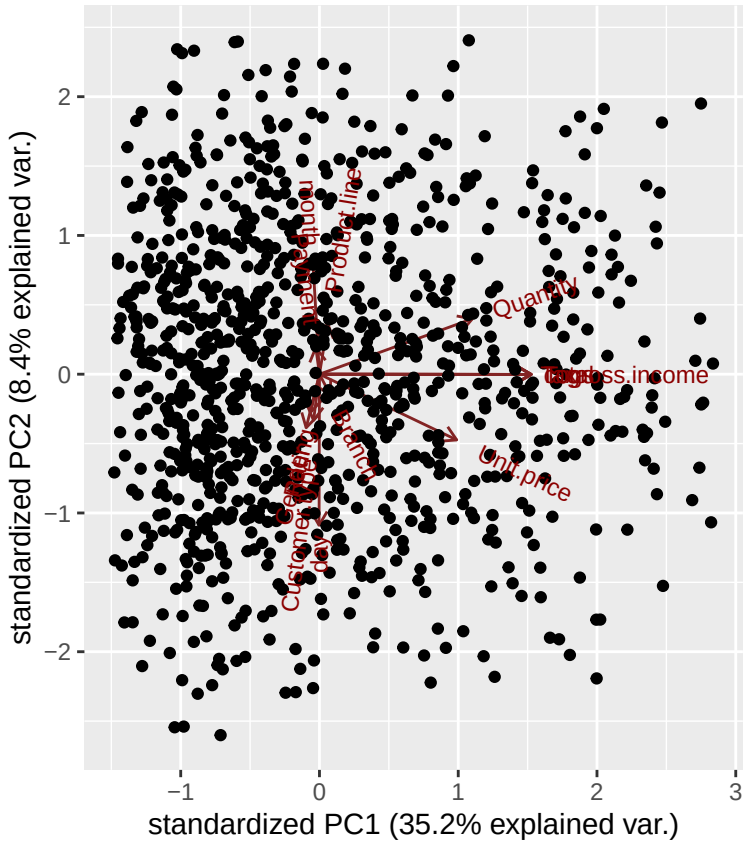
```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.2203 1.08227 1.06969 1.02580 1.00895 0.99359 0.97647
## Proportion of Variance 0.3521 0.08367 0.08173 0.07516 0.07271 0.07052 0.06811
## Cumulative Proportion  0.3521 0.43578 0.51751 0.59267 0.66538 0.73590 0.80401
##                           PC8     PC9    PC10    PC11      PC12      PC13
## Standard deviation     0.96596 0.94903 0.9057 0.29961 4.207e-16 1.418e-16
## Proportion of Variance 0.06665 0.06433 0.0586 0.00641 0.000e+00 0.000e+00
## Cumulative Proportion  0.87066 0.93499 0.9936 1.00000 1.000e+00 1.000e+00
##                           PC14
## Standard deviation     1.06e-16
## Proportion of Variance 0.00e+00
## Cumulative Proportion  1.00e+00
```

From performing PCA, we have obtained 14 principle components. PC1 has the highest variance of 35%.

```
#pca objects
str(supply_pca)
```

```
## List of 5
##  $ sdev    : num [1:14] 2.22 1.08 1.07 1.03 1.01 ...
##  $ rotation: num [1:14, 1:14] 0.0226 -0.0126 -0.0283 0.0175 0.2912 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:14] "Branch" "Customer.type" "Gender" "Product.line" ...
##   .. ..$ : chr [1:14] "PC1" "PC2" "PC3" "PC4" ...
##  $ center  : Named num [1:14] 1.99 1.5 1.5 3.45 55.67 ...
##   ..- attr(*, "names")= chr [1:14] "Branch" "Customer.type" "Gender" "Product.line" ...
##  $ scale   : Named num [1:14] 0.818 0.5 0.5 1.715 26.495 ...
##   ..- attr(*, "names")= chr [1:14] "Branch" "Customer.type" "Gender" "Product.line" ...
##  $ x       : num [1:1000, 1:14] 2.03 -2.291 0.118 1.471 2.745 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1000] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:14] "PC1" "PC2" "PC3" "PC4" ...
##  - attr(*, "class")= chr "prcomp"
```

```
# visualize the principal components
ggbiplot(supply_pca)
```



The quantity, gross income, unit price and ranch contribute to PC1.

2. FEATURE SELECTION

a.Filter Method

```
# calculate correlation matrix
correlationmatrix <- cor(sales_df)
```

```
# find high correlated values
hcorrelated <- findCorrelation(correlationmatrix, cutoff = 0.75)
hcorrelated
```

```
## [1]  8  9 10
```

variables 8,9 and 10 are highly correlated

```
# naming the highly correlated values
names(sales_df[, hcorrelated])
```

```
## [1] "Tax"          "cogs"         "gross.income"
```

```
# remove the highly correlated vales
sales_df <- sales_df[-hcorrelated]
```

Tax, cogs and gross income have been dropped because they are highly correlated.

3. Association Analysis

It is used when you want to find an association between different objects in a set and find frequent patterns in a transaction database. It helps in knowing which items customers frequently buy together by generating a set of rules called Association Rules.The Support is the freq

Load the dataset

```
# import the necessary libraries
library(arules) # provides the infrastructure for representing, manipulation and analyzing transactions
```

```
## 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:dplyr':
##
##     recode

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

```
library(arulesViz) # for various viz techniques
```

```
## Registered S3 method overwritten by 'seriation':
##   method         from
##   reorder.hclust gclus
```

```
library(ggplot2) # create graphics and charts
```

```
# import the csv file using transactions
supermarket <- read.transactions('Supermarket_Sales_Dataset II.csv', sep = ',')
```

```
## Warning in asMethod(object): removing duplicated items in transactions
```

```
# preview the first 5 columns
inspect(supermarket[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}
```

```
class(supermarket)
```

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

```
# preview items that make p our dataset
items <- as.data.frame(itemLabels(supermarket))
colnames(items) <- 'items'
head(items,10)
```

```
##                items
## 1            almonds
## 2  antioxydant juice
## 3          asparagus
## 4            avocado
## 5        babies food
## 6              bacon
## 7     barbecue sauce
## 8          black tea
## 9        blueberries
## 10        body spray
```

This displays some of the items in the dataset we will be working with.

```
# data summary
summary(supermarket)
```

```
## 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
```

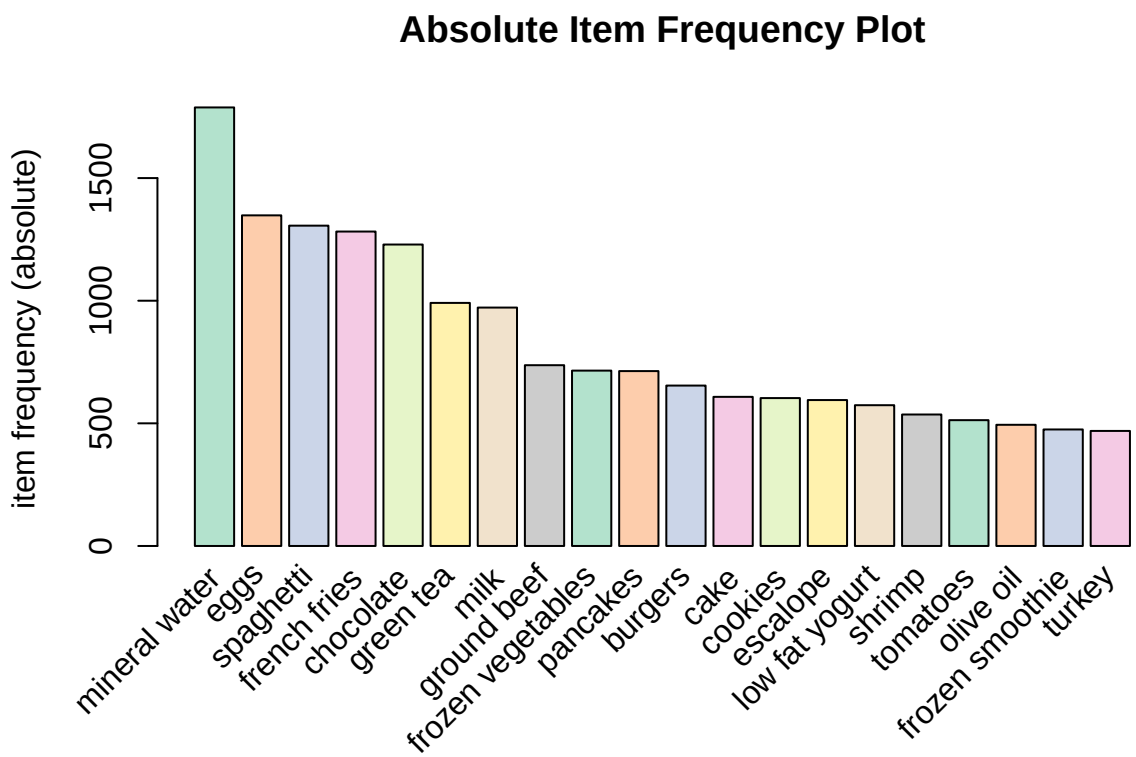
There are 7501 rows and 5729 columns. The most frequent items are mineral water, eggs, spaghetti, french fries and chocolate The element(itemset) is telling us how many transactions are there for 1-itemset and so on. The first row represents the number of items and the second row represents the number of transactions.

An item frequency plot to create an item frequency bar plot to view the distribution of objects. Plot for top 20 items

```
# Create an item frequency plot for the top 20 items
if (!require("RColorBrewer")) {
  # install color package of R
install.packages("RColorBrewer")
#include library RColorBrewer
library(RColorBrewer)
}
```

```
## Loading required package: RColorBrewer
```

```
itemFrequencyPlot(supermarket,topN=20,type="absolute",col=brewer.pal(8,'Pastel2'), main="Absolute Item F
```



The frequency plot shows the top 20 items that appeared frequently. mineral water and eggs have the most frequent sales.

Generating rules use Apriori algorithm

```
# use the default values
mkt_rules <- apriori(supermarket)
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5     0.1      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: 750
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s].
## sorting and recoding items ... [7 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

Using the default value outputs 0 rules. Therefore we will choose the appropriate parameters to work with.

```
# min support as 0.001, confidence as 0.8
rules <- apriori(supermarket, 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 [0.01s].
```

```
## 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].
```

A support of 0.001 and a confidence of 0.8 . The absolute minimum support count is 7

```
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
##  supermarket          7501   0.001        0.8
```

The set of rules is 74. The maximum support is 0.002 and its minimum is 0.001. The average support is 0.001. The maximum support is at 1 and the minimum is at 0.8 The max lift is 12.72 and the lowest is 3.35

```
# inspect the first 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
```

From the above inspection, we observe that a customer who purchased frozen smoothie,spinach also purchased mineral water. 81% of the customers who bought bacon and pancakes also bought spaghetti Most lift are greater than 1 meaning the variables are positively correlated.

Sorting the rules by confidence

```
rules <- sort(rules, by = 'confidence', decreasing = T)
inspect(rules[1:10])
```

```
##      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
```

The above rules are sorted by confidence. 100% of the customers who bought french fries,mushrrom cream sauce and pasta also bought escalope

checking for duplicates

```
redundant_rules <- is.redundant(rules)
summary(redundant_rules)
```

```
##    Mode   FALSE    TRUE
## logical      73       1
```

only one rule which is duplicate

```
#remove the duplicates
rules <-rules[!redundant_rules]
rules
```

```
## set of 73 rules
```

```
#review the items

inspect(rules[1:10])
```

```
##      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
```

Choosing other parameters to increase the number of rules

```
# min support as 0.001, confidence as 0.6
rules2 <- apriori(supermarket, 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 [0.00s].
## 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 [0.00s].
## writing ... [545 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

The number of rules increases to 545

```
summary(rules2)
```

```
## set of 545 rules
##
## rule length distribution (lhs + rhs):sizes
##   3   4   5   6
## 146 329  67   3
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   3.000   4.000   3.866   4.000   6.000
##
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.001067   Min.   :0.6000   Min.   :0.001067   Min.   : 2.517
##  1st Qu.:0.001067   1st Qu.:0.6250   1st Qu.:0.001600   1st Qu.: 2.797
##  Median :0.001200   Median :0.6667   Median :0.001866   Median : 3.446
##  Mean   :0.001409   Mean   :0.6893   Mean   :0.002081   Mean   : 3.889
##  3rd Qu.:0.001466   3rd Qu.:0.7273   3rd Qu.:0.002266   3rd Qu.: 4.177
##  Max.   :0.005066   Max.   :1.0000   Max.   :0.007999   Max.   :34.970
##      count
##  Min.   : 8.00
##  1st Qu.: 8.00
##  Median : 9.00
##  Mean   :10.57
```

```
##  3rd Qu.:11.00
##  Max.   :38.00
##
## mining info:
##         data ntransactions support confidence
##  supermarket          7501   0.001        0.6
```

```
# inspect the first rules
inspect(rules2[1:10])
```

```
##      lhs                          rhs              support     confidence
## [1]  {cookies,shallot}         => {low fat yogurt} 0.001199840 0.6000000
## [2]  {low fat yogurt,shallot}  => {cookies}        0.001199840 0.6923077
## [3]  {cookies,shallot}         => {green tea}      0.001199840 0.6000000
## [4]  {cookies,shallot}         => {french fries}   0.001199840 0.6000000
## [5]  {low fat yogurt,shallot}  => {french fries}   0.001066524 0.6153846
## [6]  {burger sauce,chicken}    => {mineral water}  0.001066524 0.6666667
## [7]  {frozen smoothie,spinach} => {mineral water}  0.001066524 0.8888889
## [8]  {milk,spinach}            => {mineral water}  0.001066524 0.6666667
## [9]  {spaghetti,spinach}       => {mineral water}  0.001333156 0.7142857
## [10] {olive oil,strong cheese} => {spaghetti}      0.001066524 0.7272727
##      coverage    lift     count
## [1]  0.001999733 7.840767  9
## [2]  0.001733102 8.611940  9
## [3]  0.001999733 4.541473  9
## [4]  0.001999733 3.510608  9
## [5]  0.001733102 3.600624  8
## [6]  0.001599787 2.796793  8
## [7]  0.001199840 3.729058  8
## [8]  0.001599787 2.796793  8
## [9]  0.001866418 2.996564 10
## [10] 0.001466471 4.177085  8
```

60% of customers who bought cookies and shallot also bough low fat yogurt.

sort by confidence

```
rules2 <- sort(rules2, by = 'confidence', decreasing = T)
inspect(rules2[1:10])
```

```
##      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
```

The maximum confidence is at 100% and the minimum is at 60%

```
redundant_rules2 <- is.redundant(rules2)
summary(redundant_rules2)
```

```
##    Mode   FALSE    TRUE
## logical     533      12
```

```
#drop the duplicates
#remove the duplicates
rules2 <-rules2[!redundant_rules2]
rules2
```

```
## set of 533 rules
```

```
# inspect first 5 rules
inspect(rules[1:10])
```

```
##      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
```

ANOMALY DETECTION

It is the process of identifying unexpected items or events in datasets, which differ from the norm.

```
# load the necessary libraries
library(tidyverse)
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(tibbletime)
```

```
##
## Attaching package: 'tibbletime'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
library('dplyr')
library(devtools)
```

```
## Loading required package: usethis
```

Load the data

```
df <- read.csv('Supermarket_Sales_Forecasting - Sales.csv')
head(df)
```

```
##        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
```

change date into datetime format

```
df$Date <- as.POSIXct(df$Date, format ='%m/%d/%Y')
```

create class tbl_time

```
df <- as_tbl_time(df, Date)
```

```
str(df)
```

```
## tibble [1,000 x 2] (S3: tbl_time/tbl_df/tbl/data.frame)
##  $ Date : POSIXct[1:1000], format: "2019-01-05" "2019-03-08" ...
##  $ Sales: num [1:1000] 549 80.2 340.5 489 634.4 ...
##  - attr(*, "index_quo")= language ~Date
##   ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##  - attr(*, "index_time_zone")= chr ""
```

apply anomaly detection which contains 3 functions i.e time_decompose(), anomalize() and time_recompose()

```
df_anomalized <- df %>%
    time_decompose(Sales, merge = TRUE) %>%
    anomalize(remainder) %>%
    time_recompose()
```

```
## Note: Index not ordered. tibbletime assumes index is in ascending order. Results may not be as desir
```

```
## frequency = 12 seconds
```

```
## Note: Index not ordered. tibbletime assumes index is in ascending order. Results may not be as desir
```

```
## trend = 12 seconds
```

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

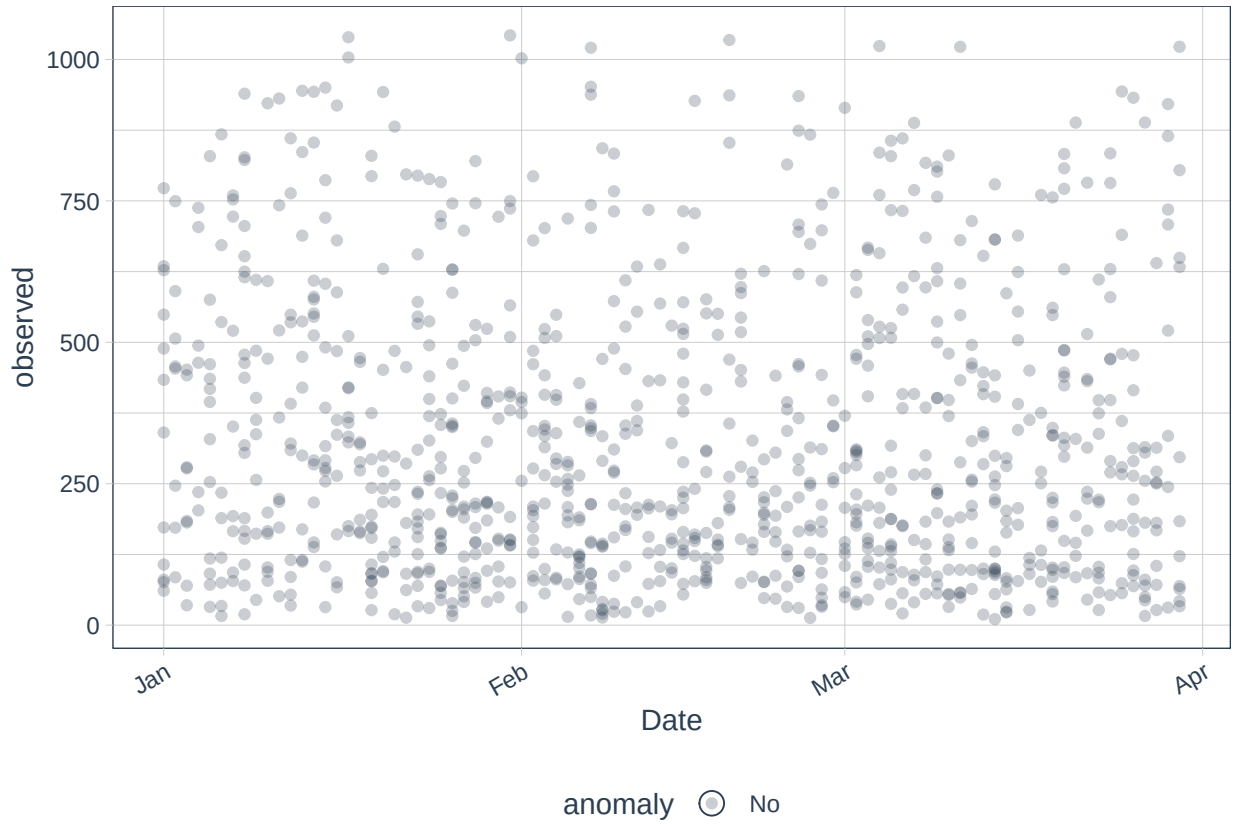
```
df_anomalized %>% glimpse()
```

```
## Rows: 1,000
## Columns: 11
## $ Date          <dttm> 2019-01-01, 2019-01-01, 2019-01-01, 2019-01-01, 2019...
## $ Sales         <dbl> 457.4430, 399.7560, 470.6730, 388.2900, 132.7620, 132...
## $ observed      <dbl> 548.9715, 80.2200, 340.5255, 489.0480, 634.3785, 627....
## $ season        <dbl> -14.359190, -4.462252, 28.744495, 23.243172, -13.8441...
## $ trend         <dbl> 445.2248, 445.5012, 445.7776, 435.9271, 426.0767, 416...
```

```
## $ remainder     <dbl> 118.105886, -360.818930, -133.996555, 29.877684, 222....
## $ remainder_l1  <dbl> -917.358, -917.358, -917.358, -917.358, -917.358, -91...
## $ remainder_l2  <dbl> 946.1539, 946.1539, 946.1539, 946.1539, 946.1539, 946...
## $ anomaly       <chr> "No", "No", "No", "No", "No", "No", "No", "No", "No",...
## $ recomposed_l1 <dbl> -486.4924, -476.3191, -442.8360, -458.1877, -505.1254...
## $ recomposed_l2 <dbl> 1377.020, 1387.193, 1420.676, 1405.324, 1358.387, 137...
```

Visualize the anomalies

```
df_anomalized %>%
    plot_anomalies(ncol = 3, alpha_dots = 0.25)
```



There are no anomalies in the data.

CONCLUSIONS AND RECOMMENDATIONS

In PCA, There are 14 principal components. PC1 has the highest variance of 35%. The quantity, gross income, unit price and ranch contribute to PC1. The other principal components that do not contribute should be dropped.

In feature selection using the filter method, we found out that there were highly correlated values which were dropped.

In association analysis, decreasing the value of confidence increases the number of rules. The most sold product was mineral water and eggs. The least were tukey and frozen smoothie.To increase the sales of turkey and smoothie, I would advice the sales department at carrefour to put the products near the mineral water and eggs.