MARKETING STRATEGIES RESULTING TO HIGHEST NUMBER OF SALES.

By Edwin Kutsushi

Defining the Question

Our main objective is to analyze the provided data and come up with most relevant marketing strategies that will result to highest number of sales.

Metrics of Success

1. Part 1, dimentionality reduction, reduce our data set to a low dimensional dataset using the t-SNE algorithm or PCA and provide insights gained from analysis.
2. Part 2, Feature Selection, perform the analysis and provide the insights that most contribute to the data set.
3. Part 3, Association Rule, create association rules that will allow us to identify relationships between variables in the data set.
4. Part 4, we are to check whether there are any anomalies in the given sales dataset and provide insights on fraud detection.

Understanding the Context

Carrefour was launched in the region in 1995 by UAE-based Majid Al Futtaim, which is the exclusive franchisee to operate Carrefour in over 30 countries across the Middle East, Africa, and Asia, and fully owns the operations in the region. Today, Majid Al Futtaim operates over 320 Carrefour stores in 16 countries, serving more than 750,000 customers daily and employing over 37,000 colleagues.

Carrefour operates different store formats, as well as multiple online offerings to meet the growing needs of its diversified customer base. In line with the brand's commitment to provide the widest range of quality products and value for money, Carrefour offers an unrivalled choice of more than 500,000 food and non-food products, and a locally inspired exemplary customer experience to create great moments for everyone every day. Across Carrefour's stores, Majid Al Futtaim sources over 80% of the products offered from the region, making it a key enabler in supporting local producers, suppliers, families and economies.

As a Data analyst team, 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). Our project has been divided into four parts where we'll explore a recent marketing dataset by performing various unsupervised learning techniques and later providing recommendations based on our insights.

Recording the experimental design.

The following steps will be followed in conducting this study:

1. Define the question, the metric for success, the context, experimental design taken.

2. Data Sourcing
3. Check the Data
4. Perform Data Cleaning
5. Perform Exploratory Data Analysis (Univariate, Bivariate & Multivariate)
6. Implement the Solution
7. Challenge the Solution
8. Follow up Questions

Data Relevance

The dataset for this Independent project can be found here The dataset files for part 1, 2, 3 and 4 can be found below:

Part 1 and 2: Dataset [Link]. Part 3: Dataset [Link]. Part 4: Dataset [Link].

Data sourcing

Loading the dataset and libraries.

```
carrefour_df <- read.csv("http://bit.ly/CarreFourDataset")

head(carrefour_df)
```

```
##    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
```

```
# finding the data summary
summary(carrefour_df)
```

Checking the summary and data type

```
##   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
```

```
# finding the data types of each column
str(carrefour_df)
```

```
## '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" "I
##  $ 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 ...
```

Data cleaning

Dropping the irrelevant column

```r
# dropping the invoice id column
carrefour_df <- subset(carrefour_df, select = -c(Invoice.ID))
```

Finding the null values

```r
# Lets Identify missing data in your dataset
# by using the function is.na()
# ---
#
colSums(is.na(carrefour_df))
```

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

Checking for the duplicates

```r
#
duplicated_rows <- carrefour_df[duplicated(carrefour_df),]
# Lets print out the variable duplicated_rows and see these duplicated rows
duplicated_rows
```

```
##  [1] Branch                  Customer.type           Gender
##  [4] Product.line            Unit.price              Quantity
##  [7] Tax                     Date                    Time
## [10] Payment                 cogs                    gross.margin.percentage
## [13] gross.income            Rating                  Total
## <0 rows> (or 0-length row.names)
```

Checking foroutliers

```r
#checking outliers in unit price
#boxplot(carrefour_df$Unit.price)
# checking for outliers in quantity
#boxplot(carrefour_df$Quantity)
# checking for outliers in Tax
#boxplot(carrefour_df$Tax)
# checking for outliers in cogs
#boxplot(carrefour_df$cogs)
# checking for outliers in gross margin percentage
#boxplot(carrefour_df$gross.margin.percentage)
# checking for outliers in gross income
#boxplot(carrefour_df$gross.income)
# checking for outliers in rating
```

```r
#boxplot(carrefour_df$Rating)
# checking for outliers in total
#boxplot(carrefour_df$Total)
```

Exploratory Data Analysis

Univariate Analysis

Label Encoding

```r
# label encoding branch column data
carrefour_df$Branch <-as.integer(as.factor(carrefour_df$Branch))
# label encoding customer column data
carrefour_df$Customer.type <-as.integer(as.factor(carrefour_df$Customer.type))
# label encoding gender column data
carrefour_df$Gender <-as.integer(as.factor(carrefour_df$Gender))
# label encoding product line column data
carrefour_df$Product.line <-as.integer(as.factor(carrefour_df$Product.line))
# label encoding payment column data
carrefour_df$Payment <-as.integer(as.factor(carrefour_df$Payment))
# label encoding date column data
carrefour_df$Date <-as.integer(as.factor(carrefour_df$Date))
# label encoding customer column data
carrefour_df$Time <-as.integer(as.factor(carrefour_df$Time))

summary(carrefour_df)
```

```
##      Branch      Customer.type       Gender       Product.line
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:2.000
##  Median :2.000   Median :1.000   Median :1.000   Median :3.000
##  Mean   :1.988   Mean   :1.499   Mean   :1.499   Mean   :3.452
##  3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:5.000
##  Max.   :3.000   Max.   :2.000   Max.   :2.000   Max.   :6.000
##    Unit.price       Quantity          Tax               Date
##  Min.   :10.08   Min.   : 1.00   Min.   : 0.5085   Min.   : 1.00
##  1st Qu.:32.88   1st Qu.: 3.00   1st Qu.: 5.9249   1st Qu.:22.00
##  Median :55.23   Median : 5.00   Median :12.0880   Median :47.00
##  Mean   :55.67   Mean   : 5.51   Mean   :15.3794   Mean   :45.58
##  3rd Qu.:77.94   3rd Qu.: 8.00   3rd Qu.:22.4453   3rd Qu.:68.00
##  Max.   :99.96   Max.   :10.00   Max.   :49.6500   Max.   :89.00
##       Time          Payment           cogs        gross.margin.percentage
##  Min.   :  1.0   Min.   :1.000   Min.   : 10.17   Min.   :4.762
##  1st Qu.:128.0   1st Qu.:1.000   1st Qu.:118.50   1st Qu.:4.762
##  Median :249.0   Median :2.000   Median :241.76   Median :4.762
##  Mean   :252.2   Mean   :2.001   Mean   :307.59   Mean   :4.762
##  3rd Qu.:384.0   3rd Qu.:3.000   3rd Qu.:448.90   3rd Qu.:4.762
##  Max.   :506.0   Max.   :3.000   Max.   :993.00   Max.   :4.762
##   gross.income        Rating          Total
##  Min.   : 0.5085   Min.   : 4.000   Min.   :  10.68
##  1st Qu.: 5.9249   1st Qu.: 5.500   1st Qu.: 124.42
##  Median :12.0880   Median : 7.000   Median : 253.85
##  Mean   :15.3794   Mean   : 6.973   Mean   : 322.97
```

```
##   3rd Qu.:22.4453   3rd Qu.: 8.500   3rd Qu.: 471.35
##   Max.   :49.6500   Max.   :10.000   Max.   :1042.65
```

Implementing the solution

Principal Component Analysis *Selecting relevant columns*

```r
# Selecting the numerical data.
# ---
#
carrefour <- carrefour_df[,c(5:7, 11, 13:15)]
head(carrefour)
```

```
##   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
```

```r
# We then pass df to the prcomp(). We also set two arguments, center and scale,
# to be TRUE then preview our object with summary
# ---
#
carrefour_df.pca <- prcomp(carrefour_df[,c(5:7, 11, 13:15)], center = TRUE, scale. =
TRUE)
summary(carrefour_df.pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4       PC5       PC6
## Standard deviation     2.2185 1.0002 0.9939 0.30001 2.981e-16 1.493e-16
## Proportion of Variance 0.7031 0.1429 0.1411 0.01286 0.000e+00 0.000e+00
## Cumulative Proportion  0.7031 0.8460 0.9871 1.00000 1.000e+00 1.000e+00
##                              PC7
## Standard deviation     9.831e-17
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

```r
# As a result we obtain 7 principal components,
# each which explain a percentate of the total variation of the dataset
# PC1 explains 70% of the total variance, which means that nearly two thirds
# of the information in the dataset (7 variables) can be encapsulated
# by just that one Principal Component. PC2 explains 14.3% and PC3 explains 14.1% of the variance.  etc
```

```r
# Calling str() to have a look at your PCA object
# ---
#
str(carrefour_df.pca)
```

```
## List of 5
```

```
## $ sdev    : num [1:7] 2.22 1.00 9.94e-01 3.00e-01 2.98e-16 ...
## $ rotation: num [1:7, 1:7] -0.292 -0.325 -0.45 -0.45 -0.45 ...
##  ..- attr(*, "dimnames")=List of 2
##  .. ..$ : chr [1:7] "Unit.price" "Quantity" "Tax" "cogs" ...
##  .. ..$ : chr [1:7] "PC1" "PC2" "PC3" "PC4" ...
## $ center  : Named num [1:7] 55.67 5.51 15.38 307.59 15.38 ...
##  ..- attr(*, "names")= chr [1:7] "Unit.price" "Quantity" "Tax" "cogs" ...
## $ scale   : Named num [1:7] 26.49 2.92 11.71 234.18 11.71 ...
##  ..- attr(*, "names")= chr [1:7] "Unit.price" "Quantity" "Tax" "cogs" ...
## $ x       : num [1:1000, 1:7] -2.005 2.306 -0.186 -1.504 -2.8 ...
##  ..- attr(*, "dimnames")=List of 2
##  .. ..$ : chr [1:1000] "1" "2" "3" "4" ...
##  .. ..$ : chr [1:7] "PC1" "PC2" "PC3" "PC4" ...
## - attr(*, "class")= chr "prcomp"
```

```
# Here we note that our pca object: The center point ($center), scaling ($scale),
# standard deviation(sdev) of each principal component.
# The relationship (correlation or anticorrelation, etc)
# between the initial variables and the principal components ($rotation).
# The values of each sample in terms of the principal components ($x)
```

```
# Then Loading our ggbiplot library
#
library(ggbiplot)
```

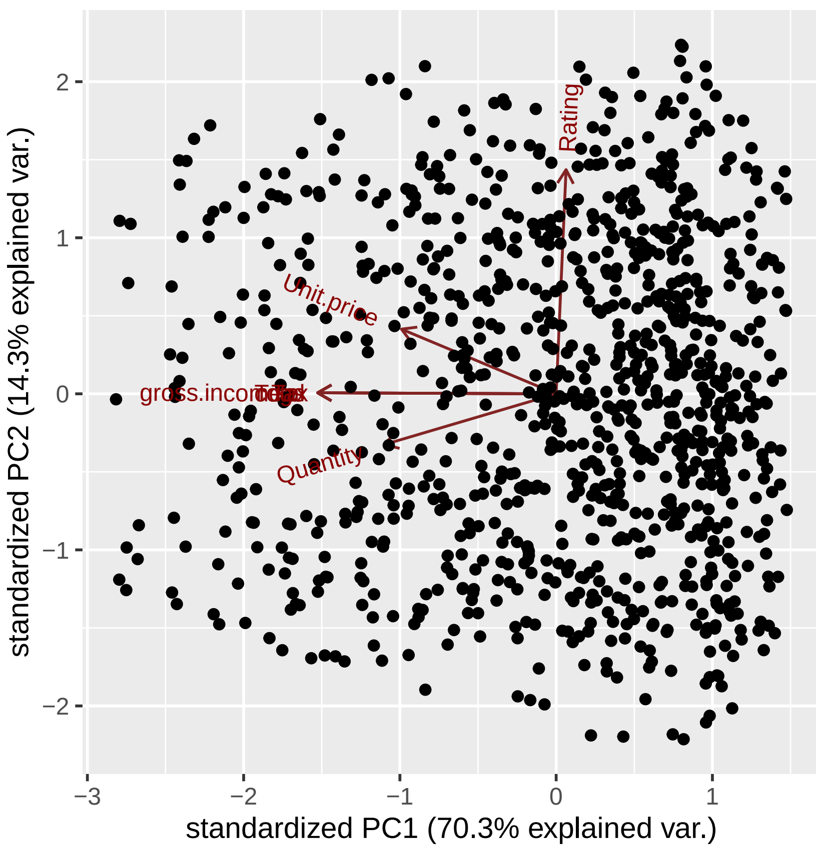
```
## Loading required package: ggplot2
```

```
## Loading required package: plyr
```

```
## Loading required package: scales
```

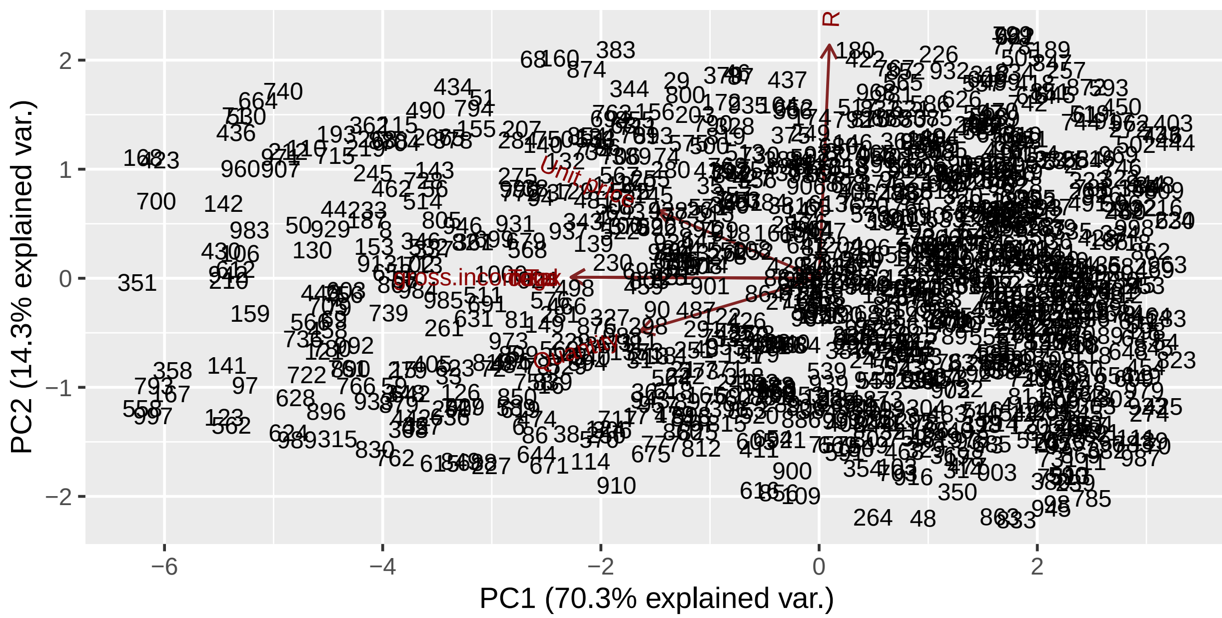
```
## Loading required package: grid
```

```
ggbiplot(carrefour_df.pca)
```




```
# From the graph we will see that the variables rating, unit price and Quantity contribute to PC1,
# with higher values in those variables moving the samples to the right on the plot.
```

```
# Adding more detail to the plot, we provide arguments rownames aslabels
#
ggbiplot(carrefour_df.pca, labels=rownames(carrefour_df), obs.scale = 1, var.scale
= 1)
```




```
# We now see which cars are similar to one another.
# The sports cars Maserati Bora, Ferrari Dino and Ford Pantera L all cluster together at the top
```

Challenging our solution

t_SNE

```
# Loading our tnse library
#
library(Rtsne)
```

```
# Curating the database for analysis
#
Quantitys<-carrefour_df$Quantity
carrefour_df$Quantity <-as.factor(carrefour_df$Quantity)
# For plotting
#
colors = rainbow(length(unique(carrefour_df$Quantity)))
names(colors) = unique(carrefour_df$Quantity)
```

Executing our algorithm

```
# Executing the algorithm on curated data
#
```

```
#tsne <- Rtsne(train[,-1], dims = 2, perplexity=30, verbose=TRUE,
#max_iter = 500)
# Getting the duration of execution
#
#exeTimeTsne <- system.time(Rtsne(train[,-1], dims = 2, perplexity=30,
#verbose=TRUE, max_iter = 500))
```

Ploting the graph

```
# Plotting our graph and closely examining the graph
#
#plot(tsne$Y, t='n', main="tsne")
#text(tsne$Y, labels=carrefour_df$Quantity, col=colors[carrefour_df$Quantity])
```

Part 2: Feature Selection

Importing Libraries

```
# Importing caret library
library(caret)
```

```
## Loading required package: lattice
```

```
# importing corrplot library
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.0.5
```

```
## corrplot 0.84 loaded
```

```
# Importing clustvarsel library
library(clustvarsel)
```

```
## Warning: package 'clustvarsel' was built under R version 4.0.5
```

```
## Loading required package: mclust
```

```
## Warning: package 'mclust' was built under R version 4.0.5
```

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

```
## Package 'clustvarsel' version 2.3.4
```

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

```r
# importing the mclust library
library(mclust)
```

```r
# Selecting the numerical data.
# ---
#
carrefour_df$Quantity <- as.integer(as.integer(carrefour_df$Quantity))
carrefour <- carrefour_df[,c(5:7, 11, 13:15)]
head(carrefour)
```

```
##   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
```

```r
# Calculating the correlation matrix
# ---
#
correlationMatrix <- cor(carrefour)
# Find attributes that are highly correlated
# ---
#
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.75)
# Highly correlated attributes
# ---
#
highlyCorrelated
```

```
## [1] 4 7 3
```

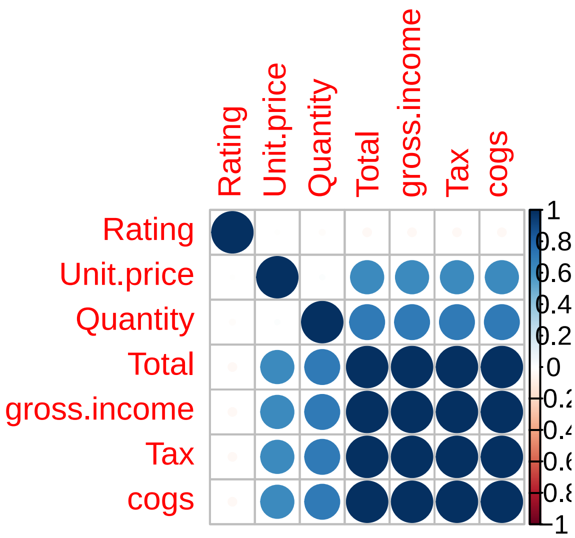
```r
names(carrefour[,highlyCorrelated])
```

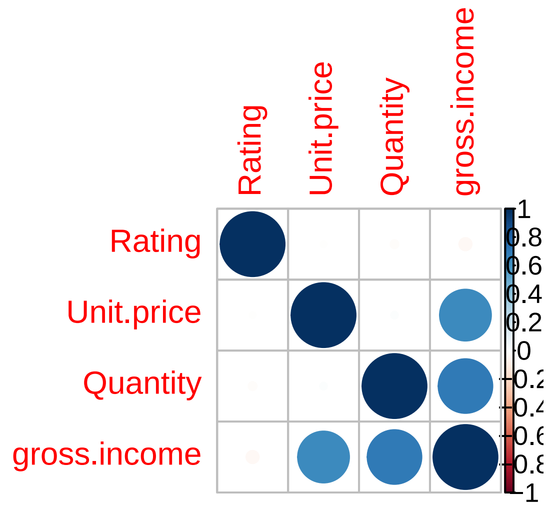
```
## [1] "cogs"  "Total" "Tax"
```

```r
# The highly correlated columns are cogs, total and tax
# we shall drop these highly correlated columns
```

```r
# We can remove the variables with a higher correlation
# and comparing the results graphically as shown below
# ---
#
# Removing Redundant Features
# ---
#
carrefour1 <- carrefour[-highlyCorrelated]
# Performing our graphical comparison
# ---
#
```

```
par(mfrow = c(1, 2))
corrplot(correlationMatrix, order = "hclust")
corrplot(cor(carrefour1), order = "hclust")
```






```
# after droping the highly correlated columns we remain with rating, unit price, quantity and gross inc
```

Challenging our solution

Using the Embedded Method *Importing the libraries*

```
# Selecting the numerical data.
# ---
#
carrefour <- carrefour_df[,c(5:7, 11, 13:15)]
head(carrefour)
```

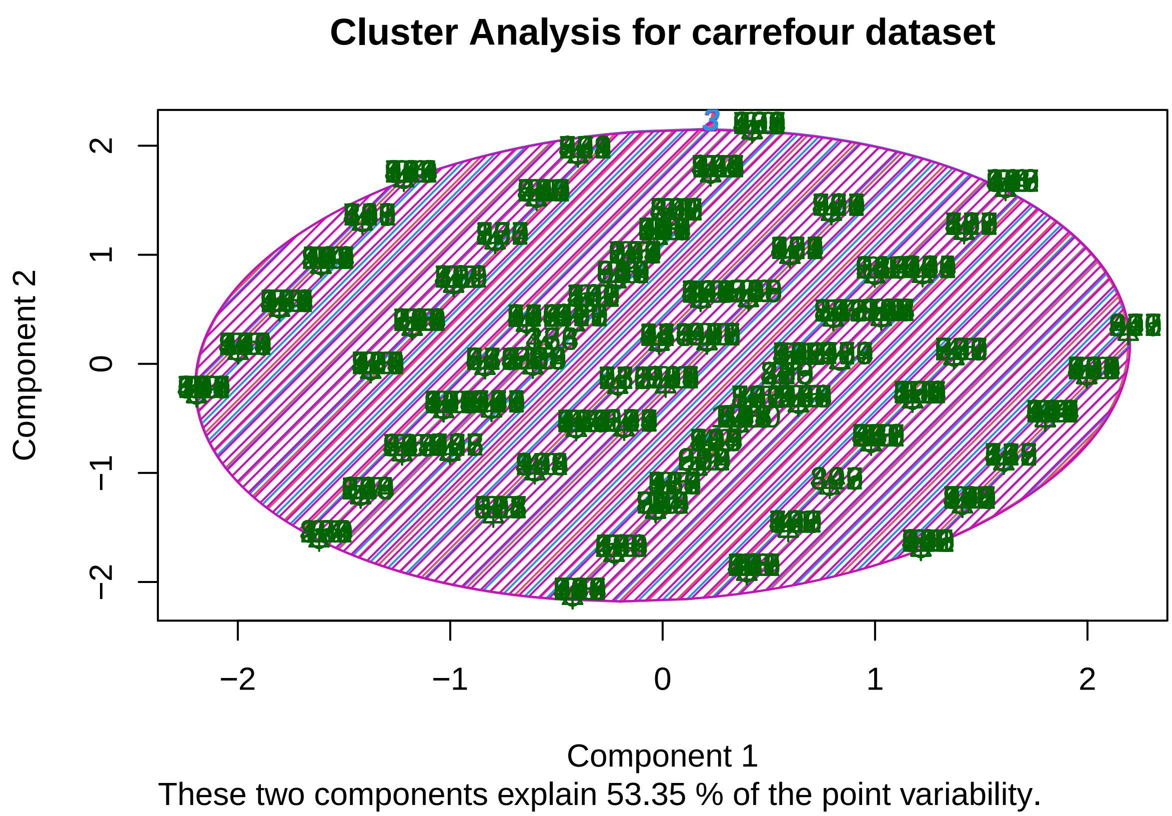
```
##   Unit.price Quantity     Tax   cogs gross.income Rating    Total
## 1      74.69        7 26.1415 522.83      26.1415    9.1 548.9715
## 2      15.28        5  3.8200  76.40       3.8200    9.6  80.2200
## 3      46.33        7 16.2155 324.31      16.2155    7.4 340.5255
## 4      58.22        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
```

```r
# We will use the ewkm function from the wskm package.
# This is a weighted subspace clustering algorithm that is well suited to very high dimensional data.
#
# We install and load our wskm package
# ---
library(wskm)
```

```
## Warning: package 'wskm' was built under R version 4.0.5

## Loading required package: latticeExtra

## Warning: package 'latticeExtra' was built under R version 4.0.5

##
## Attaching package: 'latticeExtra'

## The following object is masked from 'package:ggplot2':
##
##     layer

## Loading required package: fpc

## Warning: package 'fpc' was built under R version 4.0.5
```

```r
set.seed(2)
model <- ewkm(carrefour_df[,c(5:7, 11, 13:15)], 3, lambda=2, maxiter=1000)
```

```r
# Loading and installing our cluster package
# ---
#
library("cluster")
# Cluster Plot against 1st 2 principal components
# ---
#
clusplot(carrefour_df[1:4], model$cluster, color=TRUE, shade=TRUE,
 labels=2, lines=1,main='Cluster Analysis for carrefour dataset')
```




```
# Weights are calculated for each variable and cluster.
# They are a measure of the relative importance of each variable
# with regards to the membership of the observations to that cluster.
# The weights are incorporated into the distance function,
# typically reducing the distance for more important variables.
# Weights remain stored in the model and we can check them as follows:
#
round(model$weights*100,2)
```

```
##   Unit.price Quantity Tax cogs gross.income Rating Total
## 1          0        0  50    0           50   0.00     0
## 2          0        0   0    0            0  99.99     0
## 3          0        0  50    0           50   0.00     0
```

Part 3: Association Rule

Importing the library

```
# Loading the arules library
#
library(arules)
```

```
## Warning: package 'arules' was built under R version 4.0.5
```

```
## Loading required package: Matrix

##
## Attaching package: 'arules'

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

Importing the dataset

```
# Loading our dataset
path <- "http://bit.ly/SupermarketDatasetII"
supermarket_df <- read.transactions(path, sep = ",")
```

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

```
supermarket_df
```

```
## transactions in sparse format with
##  7501 transactions (rows) and
##  119 items (columns)
```

Data cleaning

```
#
duplicated_rows <- supermarket_df[!duplicated(supermarket_df),]
# Lets print out the variable duplicated_rows and see these duplicated rows
duplicated_rows
```

```
## transactions in sparse format with
##  5154 transactions (rows) and
##  119 items (columns)
```

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

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

```
# Previewing our first 5 transactions
#
inspect(supermarket_df[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}
```

```
# If we wanted to preview the items that make up our dataset,
# alternatively we can do the following
# ---
#
items<-as.data.frame(itemLabels(supermarket_df))
colnames(items) <- "Item"
head(items, 10)
```

```
##                 Item
## 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
```

```r
# Generating a summary of the transaction dataset
# ---
# This would give us some information such as the most purchased items,
# distribution of the item sets (no. of items purchased in each transaction), etc.
# ---
#
summary(supermarket_df)
```

```
## 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
```

```r
# Exploring the frequency of some articles
# i.e. transacations ranging from 8 to 10 and performing
# some operation in percentage terms of the total transactions
#
itemFrequency(supermarket_df[, 8:10],type = "absolute")
```

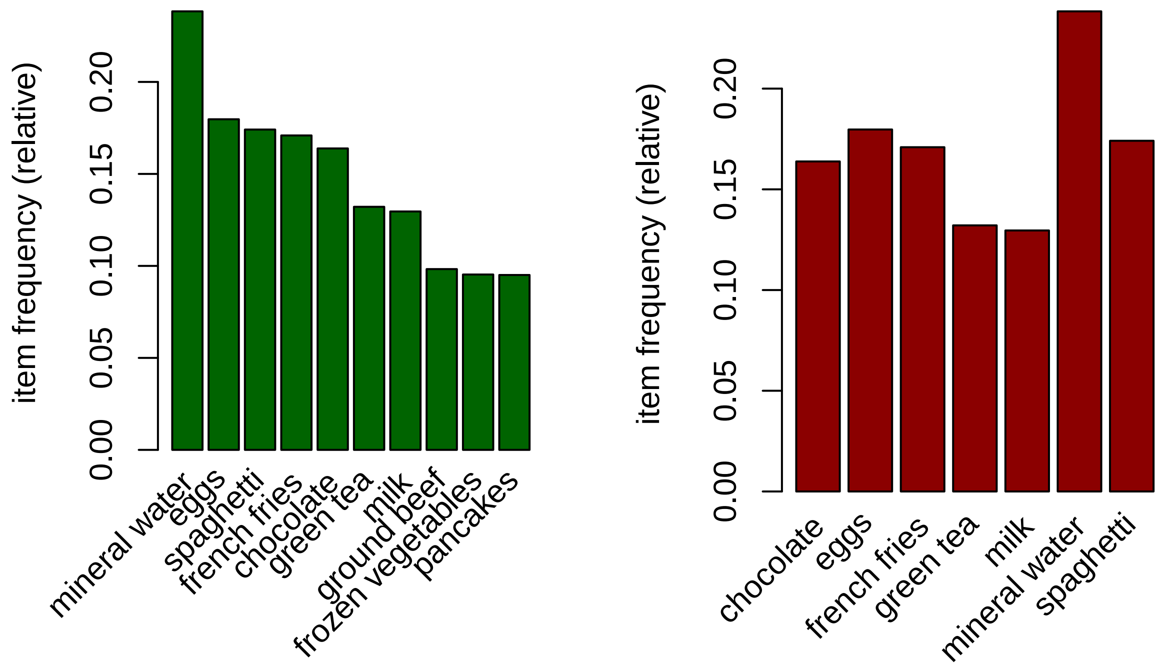
```
##   black tea blueberries  body spray
##         107          69          86
```

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

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

```r
# Producing a chart of frequencies and fitering
# to consider only items with a minimum percentage
# of support/ considering a top x of items
```

```r
# ---
# 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(supermarket_df, topN = 10,col="darkgreen")
itemFrequencyPlot(supermarket_df, support = 0.1,col="darkred")
```




```r
# 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 (supermarket_df, 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 410 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.
rules1 <- apriori (supermarket_df,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 [0.00s].
## 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 [0.00s].
## writing ... [2 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
# Building apriori model with Min Support as 0.002 and confidence as 0.6.
rules2 <- apriori (supermarket_df, 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 [0.01s].
## writing ... [545 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
rules1
```

```
## set of 2 rules
```

```r
rules2
```

```
## set of 545 rules
```

> In our first example, we increased the minimum support of 0.001 to 0.002 and model rules went from 72 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 72 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.

```r
# 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.
# ---
#
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_df          7501   0.001        0.8
```

```r
# Observing rules built in our model i.e. first 5 model rules
# ---
#
inspect(rules[1:5])
```

```
##     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
##     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
```

```r
# Interpretations:
# ---
# If someone buys frozen smoothie and spinach, they are 89% likely to buy mineral water too
# ---
```

If someone buys frozen smoothie and spinach, they are 89% likely to buy mineral water too If someone buys becon and pancakes, they are 81% likely to buy spaghetti If someone buys nonfat milk and turkey, they are 82% likely to buy mineral water If someone buys ground beef and nonfat milk, they are 86% likely to buy mineral water If someone buys mushroom cream sauce and pasta, they are 95% likely to buy escalope

```
# Ordering these rules by a criteria such as the level of confidence
# then looking at the first five rules.
# We can also use different criteria such as: (by = "lift" or by = "support")
#
rules<-sort(rules, by="confidence", decreasing=TRUE)
inspect(rules[1:5])
```

```
##     lhs                                          rhs             support
## [1] {french fries,mushroom cream sauce,pasta} => {escalope}      0.001066524
## [2] {ground beef,light cream,olive oil}       => {mineral water} 0.001199840
## [3] {cake,meatballs,mineral water}            => {milk}          0.001066524
## [4] {cake,olive oil,shrimp}                   => {mineral water} 0.001199840
## [5] {mushroom cream sauce,pasta}              => {escalope}      0.002532996
##     confidence coverage    lift      count
## [1] 1.00       0.001066524 12.606723  8
## [2] 1.00       0.001199840  4.195190  9
## [3] 1.00       0.001066524  7.717078  8
## [4] 1.00       0.001199840  4.195190  9
## [5] 0.95       0.002666311 11.976387 19
```

```
# Interpretation
# ---
# The given four rules have a confidence of 100 with only rule five having a confidence of 95%.
# ---
```

```
# If we're interested in making a promotion relating to the sale of escalope,
# we could create a subset of rules concerning these products
# ---
# This would tell us the items that the customers bought before purchasing escalope
# ---
#
escalope <- subset(rules, subset = rhs %pin% "escalope")
# Then order by confidence
escalope<-sort(escalope, by="confidence", decreasing=TRUE)
inspect(escalope[1:2])
```

```
##     lhs                                          rhs        support
## [1] {french fries,mushroom cream sauce,pasta} => {escalope} 0.001066524
## [2] {mushroom cream sauce,pasta}              => {escalope} 0.002532996
##     confidence coverage    lift     count
## [1] 1.00       0.001066524 12.60672  8
## [2] 0.95       0.002666311 11.97639 19
```

We are 100% confident that customers who bought french fries, mushroom creem sauce and pasta are likely to buy escalope. we are 95% confident that customers who bought mushroom cream sauce and pasta are likely to buy escalope in future.

```r
# What if we wanted to determine items that customers might buy
# who have previously bought escalope?
# ---
#
# Subset the rules
escalope <- subset(rules, subset = lhs %pin% "escalope")
# Order by confidence
escalope<-sort(escalope, by="confidence", decreasing=TRUE)
# inspect top 5
inspect(escalope[1:2])
```

```
##     lhs                                   rhs         support     confidence
## [1] {escalope,hot dogs,mineral water} => {milk}      0.001066524 0.8888889
## [2] {escalope,french fries,shrimp}    => {chocolate} 0.001066524 0.8888889
##     coverage   lift     count
## [1] 0.00119984 6.859625 8
## [2] 0.00119984 5.425188 8
```

We are 89% confident that customers who bought escalope previously are likely to buy escalope, hot dogs, mineral water and milk in future. we are 89% confident that customers who bought escalopes are likely to buy escalope, french fries, shrimp and chocolate in future.

Part 4: Anomaly Detection

Loading our dataset

```r
# Load tidyverse and anomalize
# ---
#
library(tidyverse)
```

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

## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::arrange()      masks plyr::arrange()
## x readr::col_factor()   masks scales::col_factor()
## x purrr::compact()      masks plyr::compact()
## x dplyr::count()        masks plyr::count()
## x purrr::discard()      masks scales::discard()
## x tidyr::expand()       masks Matrix::expand()
## x dplyr::failwith()     masks plyr::failwith()
## x dplyr::filter()       masks stats::filter()
## x dplyr::id()           masks plyr::id()
## x dplyr::lag()          masks stats::lag()
## x latticeExtra::layer() masks ggplot2::layer()
## x purrr::lift()         masks caret::lift()
```

```
## x purrr::map()          masks mclust::map()
## x dplyr::mutate()       masks plyr::mutate()
## x tidyr::pack()         masks Matrix::pack()
## x dplyr::recode()       masks arules::recode()
## x dplyr::rename()       masks plyr::rename()
## x dplyr::summarise()    masks plyr::summarise()
## x dplyr::summarize()    masks plyr::summarize()
## x tidyr::unpack()       masks Matrix::unpack()
```

```r
library(tibbletime)
```

```
## Warning: package 'tibbletime' was built under R version 4.0.5
```

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

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

```r
library(anomalize)
```

```
## Warning: package 'anomalize' was built under R version 4.0.5
```

```
## == 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 </>
```

```r
library(timetk)
```

```
## Warning: package 'timetk' was built under R version 4.0.5
```

```r
sales <- "http://bit.ly/CarreFourSalesDataset"

sales_dataset <- read.csv(sales)
head(sales_dataset)
```

```
##        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
```

```r
# viewing the tail of the data
tail(sales_dataset)
```

```
##           Date     Sales
## 995  2/18/2019   63.9975
## 996  1/29/2019   42.3675
## 997   3/2/2019 1022.4900
## 998   2/9/2019   33.4320
## 999  2/22/2019   69.1110
## 1000 2/18/2019  649.2990
```

Converting date column to date data type

```
# convert date info in format 'mm/dd/yyyy'
strDates <- c("01/05/2019", "3/31/2019")
sales_dataset$Date <- as.Date(strDates, "%m/%d/%Y")
head(sales_dataset)
```

```
##         Date    Sales
## 1 2019-01-05 548.9715
## 2 2019-03-31  80.2200
## 3 2019-01-05 340.5255
## 4 2019-03-31 489.0480
## 5 2019-01-05 634.3785
## 6 2019-03-31 627.6165
```

Chenging the dataset to tibble

```
# Convert df to a tibble
sales_dataset <- as_tibble(sales_dataset)
class(sales_dataset)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
#sales_dataset_anomalized <- sales_dataset %>%
 #   time_decompose(overall, merge = TRUE) %>%
  #  anomalize(remainder) %>%
   # time_recompose()
#sales_dataset_anomalized %>% glimpse()
```

```
# Detecting our anomalies
# ----
# We now use the following functions to detect and visualize anomalies;
# We decomposed the "count" column into "observed", "season", "trend", and "remainder" columns.
# The default values for time series decompose are method = "stl",
# which is just seasonal decomposition using a Loess smoother (refer to stats::stl()).
# The frequency and trend parameters are automatically set based on the time scale (or periodicity)
# of the time series using tibbletime based function under the hood.
# time_decompose() - this function would help with time series decomposition.
#
# anomalize() -
# We perform anomaly detection on the decomposed data using
# the remainder column through the use of the anomalize() function
# which procides 3 new columns; "remainder_l1" (lower limit),
# "remainder_l2" (upper limit), and "anomaly" (Yes/No Flag).
```

```
# The default method is method = "iqr", which is fast and relatively
# accurate at detecting anomalies.
# The alpha parameter is by default set to alpha = 0.05,
# but can be adjusted to increase or decrease the height of the anomaly bands,
# making it more difficult or less difficult for data to be anomalous.
# The max_anoms parameter is by default set to a maximum of max_anoms = 0.2
# for 20% of data that can be anomalous.
#
# time_recompose()-
# We create the lower and upper bounds around the "observed" values
# through the use of the time_recompose() function, which recomposes
# the lower and upper bounds of the anomalies around the observed values.
# We create new columns created: "recomposed_l1" (lower limit)
# and "recomposed_l2" (upper limit).
#
# plot_anomalies() -
# we now plot using plot_anomaly_decomposition() to visualize out data.
#
# ---
#
#sales_dataset %>%
 # time_decompose(sales) %>%
  #anomalize(remainder) %>%
  #time_recompose() %>%
  #plot_anomalies(time_recomposed = TRUE, ncol = 3, alpha_dots = 0.5)
```