Carrefour Project in R

Velvin Samkul

23/04/2021

Problem Statement

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 three parts where you'll explore a recent marketing dataset by performing various unsupervised learning techniques and later providing recommendations based on your insights.

Markdown Sections.

1.Problem Definition

2.Data Sourcing

3.Check the Data

4.Perform Data Cleaning

5.Perform Exploratory Data Analysis (Univariate, Bivariate & Multivariate)

6.Dimensionality Reduction

7.Feature Selection

8.Association Analysis

9.Anomaly Detection

10.Implement the Solution

11.Challenge the Solution

12.Recommendation

Data

The ID's are all unique

There are 3 branches of the carrefour represented in the dataset

There are 2 customer types: Members and Normal customers
There are 6 product categories:

```
Electronic accessories

Fashion accessories

Food and beverages

Health and beauty

Home and lifestyle

Sports and travel
```

The Gross Margin Percentage is 4.762 for all the products The data is from 2019. It is recent thus very relevant for our analysis

Installing packages.

```r
install.packages("devtools")
library(devtools)
install_github("vqv/ggbiplot")
install.packages("rtools")
install.packages("DataExplorer")
install.packages("Hmisc")
install.packages("pastecs")
install.packages("psych")
install.packages("corrplot")
install.packages("factoextra")
install.packages("caret")
```

Loading the libraries

```r
#specify the path where the file is located
library("data.table")
library(tidyverse)
library(magrittr)
library(warn = -1)

library("ggbiplot")
library(ggplot2)
library(lattice)
library(corrplot)

library(DataExplorer)
library(Hmisc)
library(pastecs)
library(psych)
library(factoextra)
library(caret)
```

Loading the data

```r
#specify the path where the file is located
library("data.table")
```

obtaining the path to the working directrory

```r
getwd()
```

```
## [1] "C:/Users/hp/Documents"
```

Loading the datasets

```r
library("readr")
df_sales <- read.csv("Supermarket_Dataset_1 - Sales Data.csv")
df_association <- read.csv("Supermarket_Sales_Dataset_2.csv")
df_forecast <-
read.csv("Supermarket_Sales_Forecasting_Sales.csv")
print(head(df_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
```

```r
print(head(df_association))
```

```
## shrimp almonds avocado vegetables.mix green.grapes ## 1 burgers
```

```
meatballs eggs
## 2 chutney
## 3 turkey avocado
## 4 mineral water milk energy bar whole wheat rice green tea ## 5 low fat
yogurt
## 6 whole wheat pasta french fries
## whole.weat.flour yams cottage.cheese energy.drink tomato.juice
low.fat.yogurt
## 1
## 2
## 3
## 4
## 5
## 6
## green.tea honey salad mineral.water salmon antioxydant.juice
frozen.smoothie
## 1
## 2
## 3
## 4
## 5
## 6
## spinach olive.oil
## 1 NA
## 2 NA
## 3 NA
## 4 NA
## 5 NA
## 6 NA
```

```
print(head(df_forecast))
```

```
## 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
```

Data Cleaning

Missing Values

```
sum(is.na(df_forecast))
```

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

```
sum(is.na(df_association))
```

```
## [1] 7500
```

```r
sum(is.na(df_sales))
```

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

```r
#There are 7500 missing values in the df_association dataset.
```

Finding the categories per column

```r
print("Branch")
```

```
## [1] "Branch"
```

```r
unique(df_sales$Branch)
```

```
## [1] A C B
## Levels: A B C
```

```r
print("Customer Type")
```

```
## [1] "Customer Type"
```

```r
unique(df_sales$Customer.type)
```

```
## [1] Member Normal
## Levels: Member Normal
```

```r
print("Gender")
```

```
## [1] "Gender"
```

```r
unique(df_sales$Gender)
```

```
## [1] Female Male
## Levels: Female Male
```

```r
print("Product Line")
```

```
## [1] "Product Line"
```

```r
# Convert data types using as.integer
# Branch
df_sales$Branch_E<-as.integer(as.factor(df_sales$Branch)
) # Customer Type
df_sales$Customer_Type_E<-as.integer(as.factor(df_sales$Customer.type))
# Gender
df_sales$Gender_E<-as.integer(as.factor(df_sales$Gender)
) # Product.line
df_sales$Product_Line_E<-as.integer(as.factor(df_sales$Product.line)
) #Payment
```

```
df_sales$Payment_E<-as.integer(as.factor(df_sales$Payment)

) library(lubridate)

##
## Attaching package: 'lubridate'

## The following objects are masked from
'package:data.table': ##
## hour, isoweek, mday, minute, month, quarter, second, wday, week, ##
yday, year

## The following object is masked from
'package:base': ##
## date

# Split date year, month and day.
# Convert to date datatype first then split
thereafter df_sales$Date <- as.Date(df_sales$Date,
"%m/%d/%Y")
df_sales$year <- year(ymd(df_sales$Date))
df_sales$month <- month(ymd(df_sales$Date))
df_sales$day <- day(ymd(df_sales$Date))

df_sales$hour = format(strptime(df_sales$Time,"%H:%M"),'%H')
df_sales$minute =
format(strptime(df_sales$Time,"%H:%M"),'%M')

#install.packages(dplyr)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from
'package:lubridate': ##
## intersect, setdiff, union

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

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

## The following objects are masked from
'package:base': ##
## intersect, setdiff, setequal, union
df_sales_num <- select_if(df_sales,is.numeric)
```

```
str(df_sales_num)
```

```
## 'data.frame': 1000 obs. of 11 variables:
## $ 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 ... ## $ 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 ... ## $ year : num 2019 2019 2019 2019
2019 ... ## $ month : num 1 3 3 1 2 3 2 2 1 2 ... ## $ day : int 5 8 3 27 8
25 25 24 10 20 ...
```

```
# Identify the columns with zero column variance.
names(df_sales_num[, sapply(df_sales_num, function(v) var(v,
na.rm=TRUE)==0)])
```

```
## [1] "gross.margin.percentage" "year"
```

```
# Drop the columns as they result to error "stop("cannot rescale a
constant/zero column to unit variance")"
df_sales_num <- subset(df_sales_num, select = -c(gross.margin.percentage,
year))
```

```
dim(df_sales_num)
```

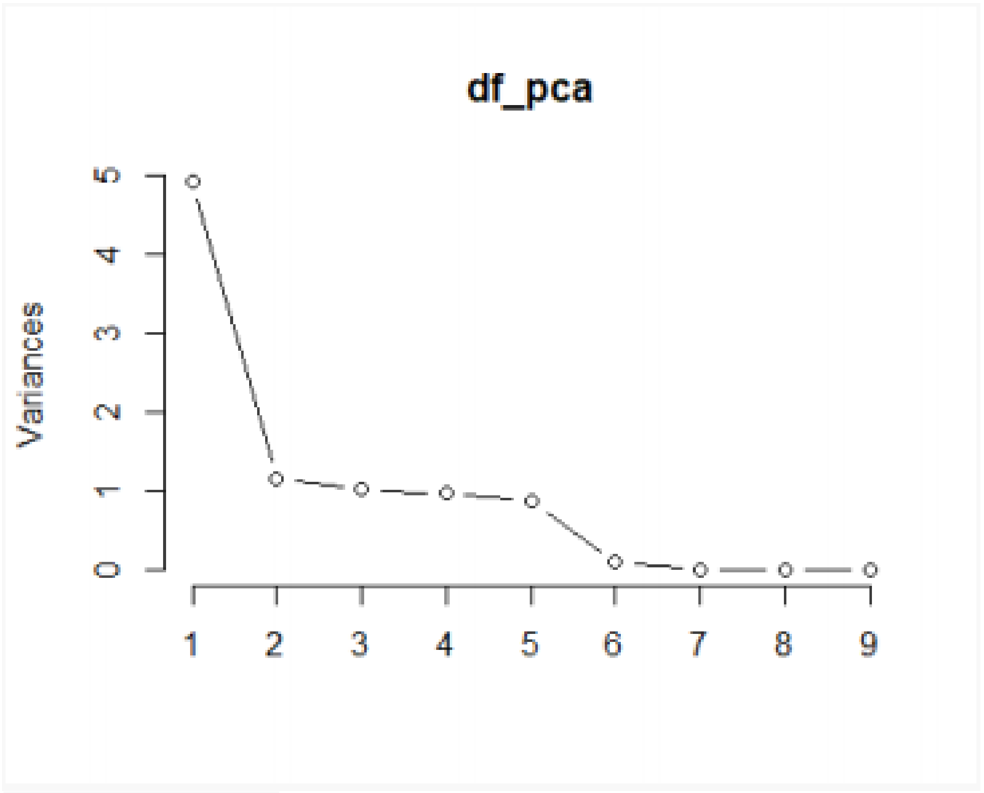
```
## [1] 1000 9
```

Principal Component Analysis

```
df_pca <- prcomp(df_sales_num, center = TRUE, scale. =
TRUE) summary(df_pca)
```

```
## Importance of components:
## PC1 PC2 PC3 PC4 PC5 PC6  PC7
## Standard deviation 2.2187 1.0704 1.0068 0.9858 0.92540 0.29986 3.216e 16
## Proportion of Variance 0.5469 0.1273 0.1126 0.1080 0.09515 0.00999
0.000e+00
## Cumulative Proportion 0.5469 0.6743 0.7869 0.8949 0.99001 1.00000
1.000e+00
## PC8 PC9
## Standard deviation 1.443e-16 1.017e-16
## Proportion of Variance 0.000e+00 0.000e+00
## Cumulative Proportion 1.000e+00 1.000e+00
```

```
plot(df_pca, type="l")
```

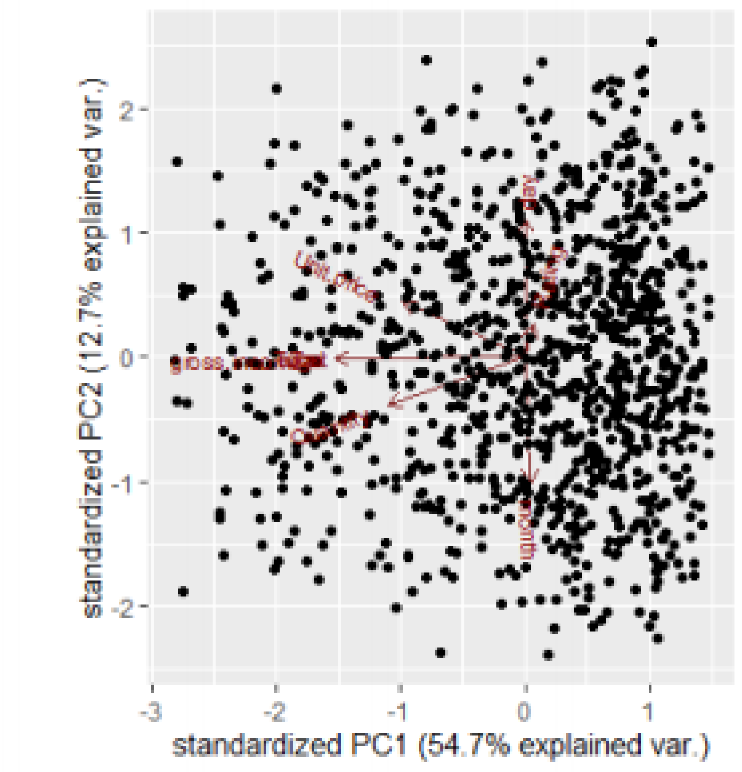


```
library(ggbiplot)
```

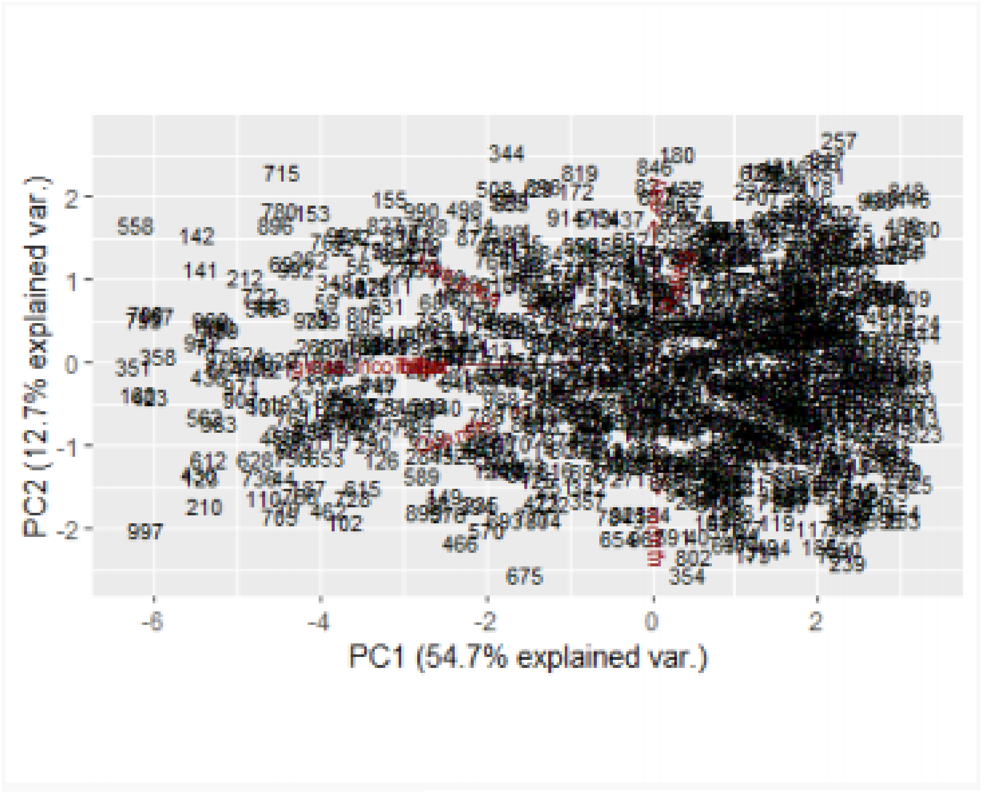
```
## Loading required package: ggplot2

## 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:lubridate': ##
## here
## Loading required package: scales

##
## Attaching package: 'scales'

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

## Loading required package: grid
```

```
ggbiplot(df_pca)
```



```
ggbiplot(df_pca, labels=rownames(df_sales_num), obs.scale = 1, var.scale = 1)
```

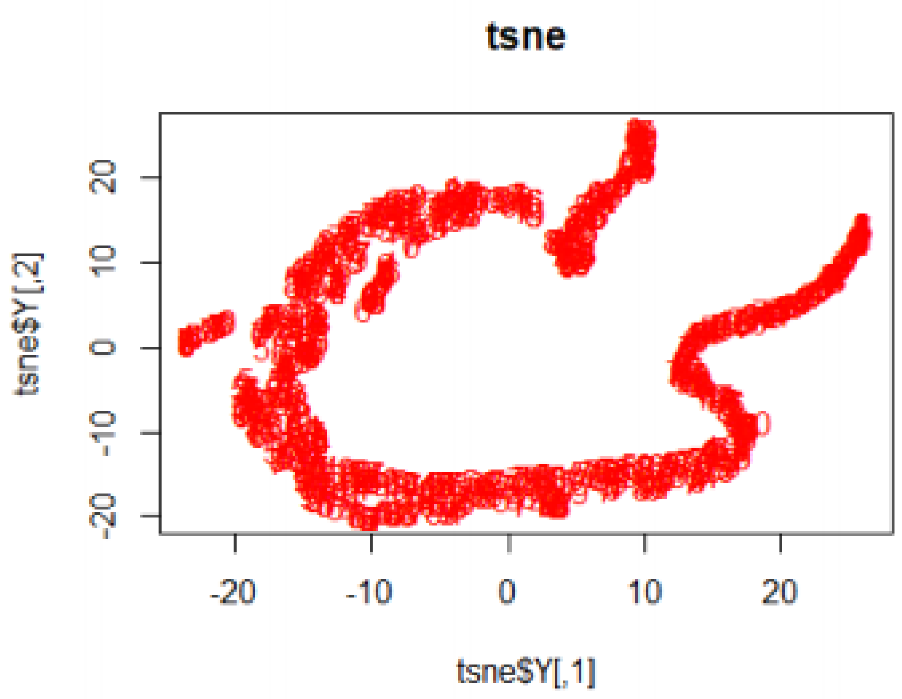


```
#install.packages("Rtsne")
library(Rtsne)
tsne <- Rtsne(df_sales_num, dims = 2, perplexity=30, verbose=TRUE, max_iter
= 500)
```

```
## Performing PCA
## Read the 1000 x 9 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta =
0.500000 ## Computing input similarities...
## Building tree...
## Done in 0.18 seconds (sparsity = 0.101252)!
## Learning embedding...
## Iteration 50: error is 61.016440 (50 iterations in 0.18 seconds)
## Iteration 100: error is 52.462216 (50 iterations in 0.12
seconds) ## Iteration 150: error is 50.931346 (50 iterations in
0.12 seconds) ## Iteration 200: error is 50.334593 (50 iterations
in 0.13 seconds) ## Iteration 250: error is 50.043407 (50
iterations in 0.13 seconds) ## Iteration 300: error is 0.634322 (50
iterations in 0.11 seconds) ## Iteration 350: error is 0.464235 (50
iterations in 0.11 seconds) ## Iteration 400: error is 0.420407 (50
iterations in 0.11 seconds) ## Iteration 450: error is 0.399157 (50
iterations in 0.10 seconds) ## Iteration 500: error is 0.381726 (50
iterations in 0.12 seconds) ## Fitting performed in 1.23 seconds.
```

```
df_sales_num$Rating_num = as.integer(df_sales_num$Rating)
```

```r
#Preparing the database for analysis
Labels<-df_sales_num$Rating_num
df_sales_num$Rating_num<-as.factor(df_sales_num$Rating_num)

# For plotting
colors = rainbow(length(df_sales_num$Rating_num))
names(colors) = unique(df_sales_num$Rating_num)

plot(tsne$Y, t='n', main="tsne")
text(tsne$Y, labels=df_sales_num$Rating_num,
col=colors[df_sales_num$Rating_num])
```



```r
path<-"http://bit.ly/FeatureSelectionDataset"

Dataset<-read.csv(path, sep = ",", dec = ".",row.names =
1) Dataset<-Dataset[-4]
head(Dataset,3)
```

```
## crim zn indus nox rm age dis rad tax ptratio b lstat  medv
## 1 0.00632 18 2.31 0.538 6.575 65.2 4.0900 1 296 15.3 396.90 4.98  24.0
## 2 0.02731 0 7.07 0.469 6.421 78.9 4.9671 2 242 17.8 396.90 9.14  21.6
## 3 0.02729 0 7.07 0.469 7.185 61.1 4.9671 2 242 17.8 392.83 4.03  34.7
library(corrplot)

## corrplot 0.84 loaded

library(caret)
```

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

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

```
## [1] 3 9 4
```

```
names(Dataset[,highlyCorrelated])
```

```
## [1] "indus" "tax" "nox"
```

```
#removing highly correlated variables

# We can remove the variables with a higher correlation
# and comparing the results graphically as shown below
# ---
#
# Removing Redundant Features
# ---
#
Dataset2<-Dataset[-highlyCorrelated]
head(Dataset2)
```

```
## crim zn rm age dis rad ptratio b lstat medv ## 1 0.00632 18
6.575 65.2 4.0900 1 15.3 396.90 4.98 24.0 ## 2 0.02731 0 6.421
78.9 4.9671 2 17.8 396.90 9.14 21.6 ## 3 0.02729 0 7.185 61.1
4.9671  2  17.8  392.83  4.03  34.7  ##  4  0.03237  0  6.998  45.8
6.0622  3  18.7  394.63  2.94  33.4  ##  5  0.06905  0  7.147  54.2
6.0622  3  18.7  396.90  5.33  36.2  ##  6  0.02985  0  6.430  58.7
6.0622 3 18.7 394.12 5.21 28.7
```

Association Analysis

```
# View sample supermarket data on which we will run association
rules head(df_association)
```

```
## shrimp almonds avocado vegetables.mix green.grapes ## 1 burgers
meatballs eggs
## 2 chutney
## 3 turkey avocado
## 4 mineral water milk energy bar whole wheat rice green tea ## 5 low fat
yogurt
## 6 whole wheat pasta french fries
## whole.weat.flour yams cottage.cheese energy.drink tomato.juice
low.fat.yogurt
```

```
## 1
## 2
## 3
## 4
## 5
## 6
## green.tea honey salad mineral.water salmon antioxydant.juice
frozen.smoothie
## 1
## 2
## 3
## 4
## 5
## 6
## spinach olive.oil
## 1 NA
## 2 NA
## 3 NA
## 4 NA
## 5 NA
## 6 NA
```

```
# Data dimensions
dim(df_association)
```

```
## [1] 7500 20
```

```
#Structure
str(df_association)
```

```
## 'data.frame': 7500 obs. of 20 variables:
## $ shrimp : Factor w/ 115 levels "almonds","antioxydant
juice",..: 15 27 108 72 65 112 98 49 43 37 ...
## $ almonds : Factor w/ 118 levels "","almonds","antioxydant
juice",..: 69 1 5 71 1 43 63 99 1 85 ...
## $ avocado : Factor w/ 116 levels "","almonds","antioxydant
juice",..: 36 1 1 37 1 1 93 53 1 1 ...
## $ vegetables.mix : Factor w/ 115 levels "","almonds","antioxydant
juice",..: 1 1 1 112 1 1 1 1 1 1 ...
## $ green.grapes : Factor w/ 111 levels "","almonds","antioxydant
juice",..: 1 1 1 51 1 1 1 1 1 1 ...
## $ whole.weat.flour : Factor w/ 107 levels "","almonds","antioxydant
juice",..: 1 1 1 1 1 1 1 1 1 1 ...
## $ yams : Factor w/ 103 levels "","almonds","antioxydant  juice",..:
1 1 1 1 1 1 1 1 1 1 ...
## $ cottage.cheese : Factor w/ 99 levels ""," asparagus",..: 1 1 1 1 1 1  1
1 1 1 ...
## $ energy.drink : Factor w/ 89 levels "","almonds","antioxydant
```

```
juice",..: 1 1 1 1 1 1 1 1 1 1 ...
## $ tomato.juice : Factor w/ 81 levels "","asparagus",..: 1 1 1 1 1 1 1  1 1
1 ...
## $ low.fat.yogurt : Factor w/ 67 levels "","asparagus",..: 1 1 1 1 1 1 1  1
1 1 ...
## $ green.tea : Factor w/ 51 levels "","blueberries",..: 1 1 1 1 1 1  1 1 1
1 ...
## $ honey : Factor w/ 43 levels "","asparagus",..: 1 1 1 1 1 1 1  1 1 1 ...
## $ salad : Factor w/ 29 levels "","babies food",..: 1 1 1 1 1 1  1 1 1 1
...
## $ mineral.water : Factor w/ 19 levels "","candy bars",..: 1 1 1 1 1 1  1
1 1 1 ...
## $ salmon : Factor w/ 8 levels "","antioxydant juice",..: 1 1 1  1 1 1 1 1
1 1 ...
## $ antioxydant.juice: Factor w/ 3 levels "","french fries",..: 1 1 1 1 1 1
1 1 1 1 ...
## $ frozen.smoothie : Factor w/ 3 levels "","protein bar",..: 1 1 1 1 1 1
1 1 1 1 ...
## $ spinach : Factor w/ 3 levels "","cereals","mayonnaise": 1 1 1  1 1 1 1
1 1 1 ...
## $ olive.oil : logi NA NA NA NA NA NA ...
```

```
# Summary to show information such as the most purchased items,no. of items
purchased in each transaction etc
summary(df_association)
```

```
## shrimp almonds avocado  ## mineral water : 577 :1754 :3112  ##
burgers : 576 mineral water: 484 mineral water: 375  ## turkey : 458
spaghetti : 411 spaghetti : 279  ## chocolate : 391 eggs : 302 eggs :
225  ## frozen vegetables: 373 ground beef : 291 milk : 213  ##
spaghetti : 354 french fries : 243 french fries : 180  ## (Other)
:4771 (Other) :4015 (Other) :3116  ## vegetables.mix green.grapes
whole.weat.flour ## :4156 :4972 :5637  ## mineral water: 201 green
tea : 153 french fries: 107  ## eggs : 181 eggs : 134 eggs : 102  ##
french fries : 174 french fries: 130 green tea : 100  ## spaghetti :
167 chocolate : 115 chocolate : 71  ## milk : 149 milk : 114 pancakes
: 69  ## (Other) :2472 (Other) :1882 (Other) :1414  ## yams
cottage.cheese energy.drink  ## :6132 :6520 :6847  ## green tea : 96
green tea : 67 green tea : 57  ## french fries : 81 pancakes : 44 low
fat yogurt : 38  ## pancakes : 69 low fat yogurt: 43 frozen smoothie:
35  ## eggs : 59 french fries : 40 french fries : 34  ## low fat
yogurt: 55 chocolate : 38 fresh bread : 28  ## (Other) :1008 (Other)
: 748 (Other) : 461
## tomato.juice low.fat.yogurt green.tea  ## :7106 :7245 :7347  ##
green tea : 31 low fat yogurt: 21 green tea : 14  ## french fries :
19 green tea : 20 french fries : 10  ## low fat yogurt: 17 fresh
bread : 14 frozen smoothie: 10  ## tomato juice : 16 french fries :
12 low fat yogurt : 9  ## pancakes : 14 light mayo : 9 fresh bread :
```

```
7  ## (Other) : 297 (Other) : 179 (Other) : 103
## honey salad mineral.water  ## :7414 :7454 :7476  ## green tea : 8
green tea : 4 magazines : 3  ## fresh bread : 6 french fries : 3
fresh bread : 2  ## low fat yogurt: 6 frozen smoothie: 3 green tea :
2  ## escalope : 4 cottage cheese : 2 low fat yogurt: 2  ## french
fries : 4 eggplant : 2 pancakes : 2  ## (Other) : 58 (Other) : 32
(Other) : 13
## salmon antioxydant.juice frozen.smoothie ## :7493 :7497 :7497  ##
antioxydant juice: 1 french fries : 1 protein bar: 2  ## cake : 1
frozen smoothie: 2 spinach : 1  ## chocolate : 1
## frozen smoothie : 1
## magazines : 1
## (Other) : 2
## spinach olive.oil
## :7498 Mode:logical
## cereals : 1 NA's:7500
## mayonnaise: 1
##
##
##
##
```

```
# Count the missing values
colSums(is.na(df_association))
```

```
##  shrimp  almonds  avocado  vegetables.mix   ##  0  0  0  0   ##  green.grapes
whole.weat.flour  yams  cottage.cheese   ##  0  0  0  0   ##  energy.drink
tomato.juice  low.fat.yogurt  green.tea   ##  0  0  0  0   ##  honey  salad
mineral.water  salmon   ##  0  0  0  0   ##  antioxydant.juice  frozen.smoothie
spinach olive.oil  ## 0 0 0 7500
```

```
# Drop olive oil column from dataframe
df_association$olive.oil <- NULL
# Verify that column is successfully dropped
str(df_association)
```

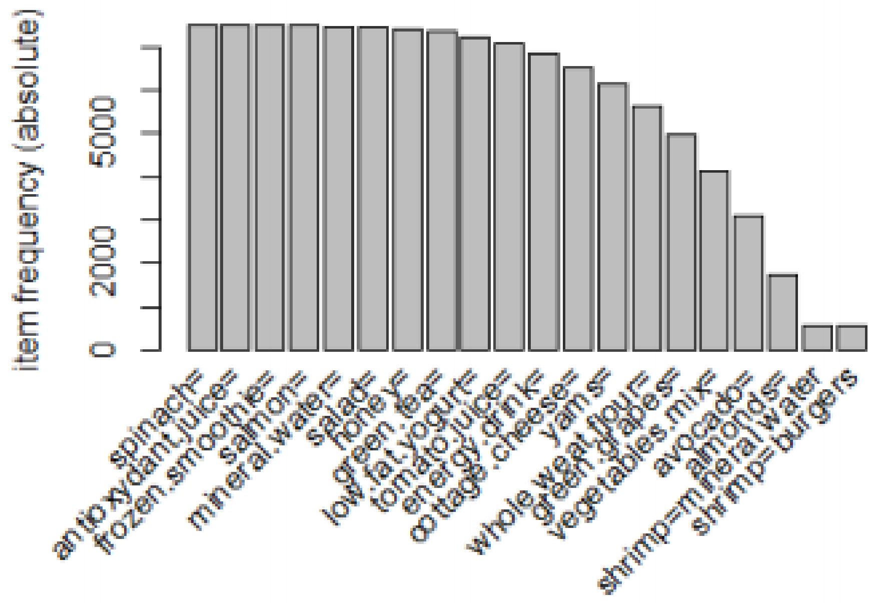
```
## 'data.frame': 7500 obs. of 19 variables:
## $ shrimp : Factor w/ 115 levels "almonds","antioxydant
juice",..: 15 27 108 72 65 112 98 49 43 37 ...
## $ almonds : Factor w/ 118 levels "","almonds","antioxydant
juice",..: 69 1 5 71 1 43 63 99 1 85 ...
## $ avocado : Factor w/ 116 levels "","almonds","antioxydant
juice",..: 36 1 1 37 1 1 93 53 1 1 ...
## $ vegetables.mix : Factor w/ 115 levels "","almonds","antioxydant
juice",..: 1 1 1 112 1 1 1 1 1 1 ...
## $ green.grapes : Factor w/ 111 levels "","almonds","antioxydant
juice",..: 1 1 1 51 1 1 1 1 1 1 ...
## $ whole.weat.flour : Factor w/ 107 levels "","almonds","antioxydant
```

```
juice",..: 1 1 1 1 1 1 1 1 1 1 ...
## $ yams : Factor w/ 103 levels "","almonds","antioxydant  juice",..:
1 1 1 1 1 1 1 1 1 1 ...
## $ cottage.cheese : Factor w/ 99 levels ""," asparagus",..: 1 1 1 1 1 1  1
1 1 1 ...
## $ energy.drink : Factor w/ 89 levels "","almonds","antioxydant
juice",..: 1 1 1 1 1 1 1 1 1 1 ...
## $ tomato.juice : Factor w/ 81 levels "","asparagus",..: 1 1 1 1 1 1 1  1 1
1 ...
## $ low.fat.yogurt : Factor w/ 67 levels "","asparagus",..: 1 1 1 1 1 1 1  1
1 1 ...
## $ green.tea : Factor w/ 51 levels "","blueberries",..: 1 1 1 1 1 1  1 1 1
1 ...
## $ honey : Factor w/ 43 levels "","asparagus",..: 1 1 1 1 1 1 1  1 1 1 ...
## $ salad : Factor w/ 29 levels "","babies food",..: 1 1 1 1 1 1  1 1 1 1
...
## $ mineral.water : Factor w/ 19 levels "","candy bars",..: 1 1 1 1 1 1  1
1 1 1 ...
## $ salmon : Factor w/ 8 levels "","antioxydant juice",..: 1 1 1  1 1 1 1 1
1 1 ...
## $ antioxydant.juice: Factor w/ 3 levels "","french fries",..: 1 1 1 1 1 1
1 1 1 1 ...
## $ frozen.smoothie : Factor w/ 3 levels "","protein bar",..: 1 1 1 1 1 1
1 1 1 1 ...
## $ spinach : Factor w/ 3 levels "","cereals","mayonnaise": 1 1 1  1 1 1 1
1 1 1 ...
```

```r
library(arules)
```

```
## Loading required package: Matrix

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

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

```r
## Create an item frequency plot for the top 20 items
# coerce data frame into transaction. Plotting the dataframe directly
fails transact <- as(df_association, "transactions")
# plot item frequency
itemFrequencyPlot(transact,topN=20,type="absolute")
```



Rules for Association

```
tail(df_association)
```

```
## shrimp almonds avocado vegetables.mix green.grapes ## 7495 pancakes
light mayo
## 7496 butter light mayo fresh bread
## 7497 burgers frozen vegetables eggs french fries magazines ## 7498
chicken
## 7499 escalope green tea
## 7500 eggs frozen smoothie yogurt cake low fat yogurt  ##
whole.weat.flour yams cottage.cheese energy.drink tomato.juice ##
7495
## 7496
## 7497 green tea
## 7498
## 7499
## 7500
## low.fat.yogurt green.tea honey salad mineral.water salmon ##
7495
## 7496
## 7497
## 7498
## 7499
## 7500
## antioxydant.juice frozen.smoothie spinach
## 7495
## 7496
## 7497
## 7498
## 7499
## 7500
```

```r
# Get the rules
rules <- apriori(df_association, parameter = list(supp = 0.5, conf
= 0.8,target = "rules",minlen=2))
```

```
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
## 0.8 0.1 1 none FALSE TRUE 5 0.5 2 ## maxlen target ext
## 10 rules FALSE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE
##
## Absolute minimum support count: 3750
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[1280 item(s), 7500 transaction(s)] done
[0.05s]. ## sorting and recoding items ... [16 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10
```

```
## Warning in apriori(df_association, parameter = list(supp = 0.5, conf =
0.8, :
## Mining stopped (maxlen reached). Only patterns up to a length of 10
returned!
```

```
## done [0.02s].
## writing ... [425218 rule(s)] done [0.10s].
## creating S4 object ... done [0.20s].
#rules <- sort(rules, by="lift", decreasing=TRUE)
```

```r
summary(rules)
```

```
## set of 425218 rules
##
## rule length distribution (lhs + rhs):sizes
## 2 3 4 5 6 7 8 9 10
## 204 1478 6576 20134 45002 75943 98616 99417 77848
##
```

```
## Min. 1st Qu. Median Mean 3rd Qu. Max.
## 2.000 7.000 8.000 7.986 9.000 10.000
##
## summary of quality measures:
## support confidence lift count  ## Min. :0.5541 Min. :0.8108
Min. :1.000 Min. :4156  ## 1st Qu.:0.5541 1st Qu.:1.0000 1st
Qu.:1.001 1st Qu.:4156  ## Median :0.6629 Median :1.0000 Median
:1.021 Median :4972  ## Mean :0.6455 Mean :0.9882 Mean :1.095 Mean
:4841  ## 3rd Qu.:0.7516 3rd Qu.:1.0000 3rd Qu.:1.150 3rd Qu.:5637
## Max. :0.9996 Max. :1.0000 Max. :1.508 Max. :7497  ##
## mining info:
## data ntransactions support confidence
## df_association 7500 0.5 0.8
```

```
# Show the top 3 rules, but only 2 digits.
#options(digits=2)
inspect(rules[1:20])
```

```
## lhs rhs support confidence lift  ## [1] {vegetables.mix=} =>
{green.grapes=} 0.5541333 1.0000000  1.508447
## [2] {green.grapes=} => {vegetables.mix=} 0.5541333 0.8358809
1.508447
## [3] {vegetables.mix=} => {whole.weat.flour=} 0.5541333 1.0000000
1.330495
## [4] {vegetables.mix=} => {yams=} 0.5541333 1.0000000  1.223092
## [5] {vegetables.mix=} => {cottage.cheese=} 0.5541333 1.0000000
1.150307
## [6] {vegetables.mix=} => {energy.drink=} 0.5541333 1.0000000
1.095370
## [7] {vegetables.mix=} => {tomato.juice=} 0.5541333 1.0000000
1.055446
## [8] {vegetables.mix=} => {low.fat.yogurt=} 0.5541333 1.0000000
1.035197
## [9] {vegetables.mix=} => {green.tea=} 0.5541333 1.0000000  1.020825
## [10] {vegetables.mix=} => {honey=} 0.5541333 1.0000000  1.011600
## [11] {vegetables.mix=} => {salad=} 0.5541333 1.0000000  1.006171
## [12] {vegetables.mix=} => {mineral.water=} 0.5541333 1.0000000
1.003210
## [13] {vegetables.mix=} => {salmon=} 0.5541333 1.0000000  1.000934
## [14] {vegetables.mix=} => {antioxydant.juice=} 0.5541333 1.0000000
1.000400
## [15] {vegetables.mix=} => {frozen.smoothie=} 0.5541333 1.0000000
1.000400
## [16] {vegetables.mix=} => {spinach=} 0.5541333 1.0000000  1.000267
## [17] {green.grapes=} => {whole.weat.flour=} 0.6629333 1.0000000
1.330495
## [18] {whole.weat.flour=} => {green.grapes=} 0.6629333 0.8820294
1.330495
```

```
## [19] {green.grapes=} => {yams=} 0.6629333 1.0000000  1.223092
## [20] {yams=} => {green.grapes=} 0.6629333 0.8108284  1.223092
## count
## [1] 4156
## [2] 4156
## [3] 4156
## [4] 4156
## [5] 4156
## [6] 4156
## [7] 4156
## [8] 4156
## [9] 4156
## [10] 4156
## [11] 4156
## [12] 4156
## [13] 4156
## [14] 4156
## [15] 4156
## [16] 4156
## [17] 4972
## [18] 4972
## [19] 4972
## [20] 4972
```

Anomaly Detection

```
#install.packages("anomalize") #Anormally detection
library(anomalize)
library(lubridate)
library(tibbletime)

# View the data to check anormalies on
head(df_forecast)
## 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

str(df_forecast)

## 'data.frame': 1000 obs. of 2 variables:
## $ Date : Factor w/ 89 levels "1/1/2019","1/10/2019",..: 27 88 82 20 58 77
49 48 2 44 ...
## $ Sales: num 549 80.2 340.5 489 634.4 ...

# totalling the sales based on their common shared dates
```

```
sales_aggregate <- aggregate(df_forecast$Sales, by = list(Date
= df_forecast$Date), FUN = sum)

head(sales_aggregate)
```

```
## Date x
## 1 1/1/2019 4745.181
## 2 1/10/2019 3560.949
## 3 1/11/2019 2114.963
## 4 1/12/2019 5184.764
## 5 1/13/2019 2451.204
## 6 1/14/2019 3966.617
```

```
# getting a data frame of the frequency table of Date
date_table <- data.frame(table(df_forecast$Date))
head(date_table)
```

```
## Var1 Freq
## 1 1/1/2019 12
## 2 1/10/2019 9
## 3 1/11/2019 8
## 4 1/12/2019 11
## 5 1/13/2019 10
## 6 1/14/2019 13
```

```
library(tidyverse)
```

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

## v tibble 2.1.3 v stringr 1.4.0
## v tidyr 1.0.2 v forcats 0.5.0
## v purrr 0.3.3

## -- Conflicts
 ------------------------------------------------------------- -
 tidyverse_conflicts() --
## x plyr::arrange() masks dplyr::arrange()
## x lubridate::as.difftime() masks base::as.difftime()
## x dplyr::between() masks data.table::between()
## x scales::col_factor() masks readr::col_factor()
## x purrr::compact() masks plyr::compact()
## x plyr::count() masks dplyr::count()
## x lubridate::date() masks base::date()
## x purrr::discard() masks scales::discard()
## x tidyr::expand() masks Matrix::expand()
## x plyr::failwith() masks dplyr::failwith()
## x dplyr::filter() masks stats::filter()
## x dplyr::first() masks data.table::first()
## x plyr::here() masks lubridate::here()
## x lubridate::hour() masks data.table::hour()
## x plyr::id() masks dplyr::id()
## x arules::intersect() masks lubridate::intersect(), base::intersect() ##
x lubridate::isoweek() masks data.table::isoweek()
## x dplyr::lag() masks stats::lag()
## x dplyr::last() masks data.table::last()
## x purrr::lift() masks caret::lift()
## x lubridate::mday() masks data.table::mday()
## x lubridate::minute() masks data.table::minute()
## x lubridate::month() masks data.table::month()
## x plyr::mutate() masks dplyr::mutate()
## x tidyr::pack() masks Matrix::pack()
## x lubridate::quarter() masks data.table::quarter()
## x arules::recode() masks dplyr::recode()
## x plyr::rename() masks dplyr::rename()
## x lubridate::second() masks data.table::second()
## x arules::setdiff() masks lubridate::setdiff(), base::setdiff() ## x
plyr::summarise() masks dplyr::summarise()
## x plyr::summarize() masks dplyr::summarize()
## x purrr::transpose() masks data.table::transpose() ## x
arules::union() masks lubridate::union(), base::union() ## x
tidyr::unpack() masks Matrix::unpack()
## x lubridate::wday() masks data.table::wday()
## x lubridate::week() masks data.table::week()
## x lubridate::yday() masks data.table::yday()
## x lubridate::year() masks data.table::year()
```

```
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(lubridate)
library(tibbletime)
```

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

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

```
# combining both data frames
final_df <- merge(sales_aggregate, date_table, by.x = "Date", by.y = "Var1")
```

```r
# renaming the columns
names(final_df) <- c("Date", "Total.Sales", "count")
head(final_df)
```

```
## Date Total.Sales count
## 1 1/1/2019 4745.181 12
## 2 1/10/2019 3560.949 9
## 3 1/11/2019 2114.963 8
## 4 1/12/2019 5184.764 11
## 5 1/13/2019 2451.204 10
## 6 1/14/2019 3966.617 13
```

```r
# changing the Date column to Date format
final_df$Date <- mdy(final_df$Date)
str(final_df)
```

```
## 'data.frame': 89 obs. of 3 variables:
## $ Date : Date, format: "2019-01-01" "2019-01-10" ... ## $
Total.Sales: num 4745 3561 2115 5185 2451 ...
## $ count : int 12 9 8 11 10 13 13 10 11 9 ...
```

```r
final_df$Date <- as_tbl_time(final_df, index = 'Date')
str(final_df$Date)
```

```
## Classes 'tbl_time', 'tbl_df', 'tbl' and 'data.frame': 89 obs. of 3
variables:
## $ Date : Date, format: "2019-01-01" "2019-01-10" ... ## $
Total.Sales: num 4745 3561 2115 5185 2451 ...
## $ count : int 12 9 8 11 10 13 13 10 11 9 ...
## - attr(*, "index_quo")= language ~"Date"
## ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
## - attr(*, "index_time_zone")= chr "UTC"
```

```r
class(final_df)
```

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

```r
final_df %>%
 time_decompose(count) %>%
 anomalize(remainder) %>%
 time_recompose() %>%
 plot_anomalies(time_recomposed = TRUE, ncol = 3, alpha_dots = 0.5)
```