Principal Component Analysis with R

Hervé Abdi

2018-09-16

Prelude

If you want to make sure that you have a clean start, you can execute the following commands:

```
rm(list = ls())
graphics.off()
```

Or, better (see below, preamble), you can use an `Rproject` for this project.

Set defaults for the chunks

Preamble

Make sure that you start this analysis as a new `Rproject` so that the default directory will be correctly set.

Before we start the analysis, we need to have our standard packages installed (some from `Github`) and the corresponding libraries loaded:

- `ExPosition`
- `InPosition`
- `ggplot2` for graphs
- `PTCA4CTA` (from *Github*)
- `data4PCCAR`(from *Github*)
- `dplyr` (to recode the data)
- `corrplot` (nice correlation heatmap)

and all their extensions:

```
# Decomment all/some these lines if the packages are not installed
# devtools::install_github('HerveAbdi/PTCA4CATA')
# devtools::install_github('HerveAbdi/sata4PCCAR') # of course!
#  install.packages(prettyGraphs)
#  install.packages('dplyr')
#  install.packages('corrplot')
#  load the libraries that we will need
suppressMessages(library(ExPosition))
suppressMessages(library(dplyr))
suppressMessages(library(ggplot2))
suppressMessages(library(PTCA4CATA))
suppressMessages(library(data4PCCAR))
suppressMessages(library(corrplot))
```

Introduction

The data sets can be found from the package `data4PCCAR.`

Twenty wines

The data (including supplementary variables and observations) are stored in a list called `twentyWines` available from the package `data4PCCAR`; To get the data use this code:

```
data('twentyWines')
df <- twentyWines$df.active
colnames(df) <- c("sweet","astringent")
```

The data frame for the active data set is now in stored in `df`.

More details about the twenty wines data

To know more about this data set, have a look at the help for `data4PCCAR::twentyWines`.

Compute the "unscaled" principal component analysis.

To compute the PCA, we use the function `ExPosition::epPCA` and store the results in the list `resPCA`. The ratings (`sweet` and `astringent`) use the same scale from 0 to 20, so we want to have a non-scaled v analysis (i.e., the variables will not be scaled and so they will keep their original variance). By default, `epPCA` scales the variables, to override the default, we need to specify the parameter `scale = FALSE`. By default, `epPCA` will provide a set of standard graphs that are very helpful for a first look at the results, but these graphs almost always need to be tailored for specific aspects of the data. In addition, `Rmarkdown` does not interface well with multiple graph produces by a single command. To override the defaults and prohibit `epPCA` from generating these graphs, we specify the parameter `graphs = FALSE`. To color the observations, we use the parameter `DESIGN` of `epPCA`. This parameter gives a vector (or a matrix) for a factor that describes the observations (here it gives the country of origin `F` for France versus `U` for USA). The analysis is performed with this code

```
resPCA <-  epPCA(df, scale = FALSE, #
                 DESIGN = twentyWines$supplementary.variables$Origin,
                 graph = FALSE)
```

The Loadings

There are several ways of computing loadings: Here we present the three essential versions of the ladings.

Loadings as "slices" of inertia

This is the standard loading computed by `ExPosition::epPCA`. These loadings are normalized such that the total sum of the squared loadings is equal to the total inertia of the data table. So:

```
loadings.1 <- resPCA$ExPosition.Data$fj
```

The squared row loadings sum to eigenvalues, and the squared column loadings sum to the to sums of squares of the variables. This can be checked with:

```
colSums(round(loadings.1^2))
#> [1] 392  52
rowSums(round(loadings.1^2))
#>      sweet astringent
#>        150        294
```

Loadings as correlations

`ExPosition` does not give these loadings, but they are easily computed as:

```
# Loadings as correlations
loadings.2 <- t(cor(df, resPCA$ExPosition.Data$fi))
```

Loadings as coefficients of the optimal linear combination

These loadings are the right singular vectors, we can also get them (almost) directly from `ExPosition` as:

```
loadings.3 <- resPCA$ExPosition.Data$pdq$q
```

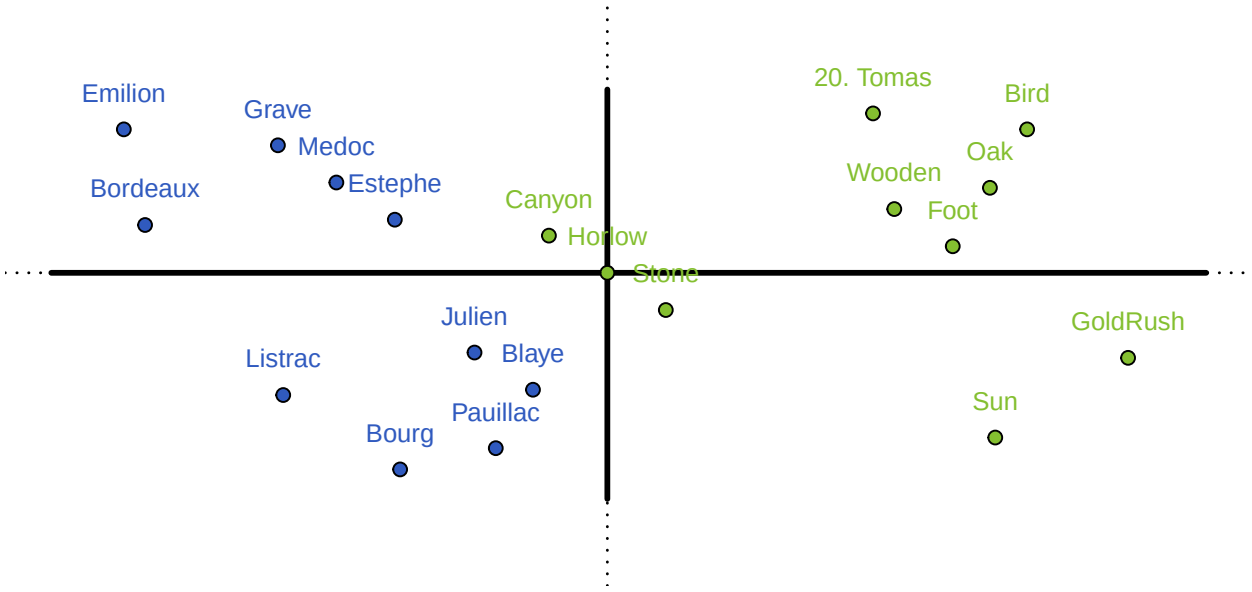
Some graphs

With prettyGraphs::prettyPlot

Graphs for the observations

To draw the graphs we first use the function `prettyPlot` (from the package `prettyGraphs`). We first create the plot and to keep the plot we need to store it in a variable. This is done with the function `recordPlot` used here to store the current graph in the variable `a1.JolieMap.1`. To color the observations, we use the color vector `resPCA$Plotting.Data$fi.col` generated by `epPCA`. Finally, because of the way `Rmarkdown` processes graphics we need to specify the option `dev.new = FALSE`. All is done with this code:

```
joli.plot1 <- prettyPlot(resPCA$ExPosition.Data$fi,
                         dev.new = FALSE,
                         col = resPCA$Plotting.Data$fi.col)
```




```
a1.JolieMap.1 <- recordPlot() #
```

To look at the graph, we just need to print it:

```
print(a1.JolieMap.1)
```

To save this graph as a `pdf` file, it we can use the function `pdf`. For example, to save the graph as a pdf file called `winesP12.pdf`, we use the following code

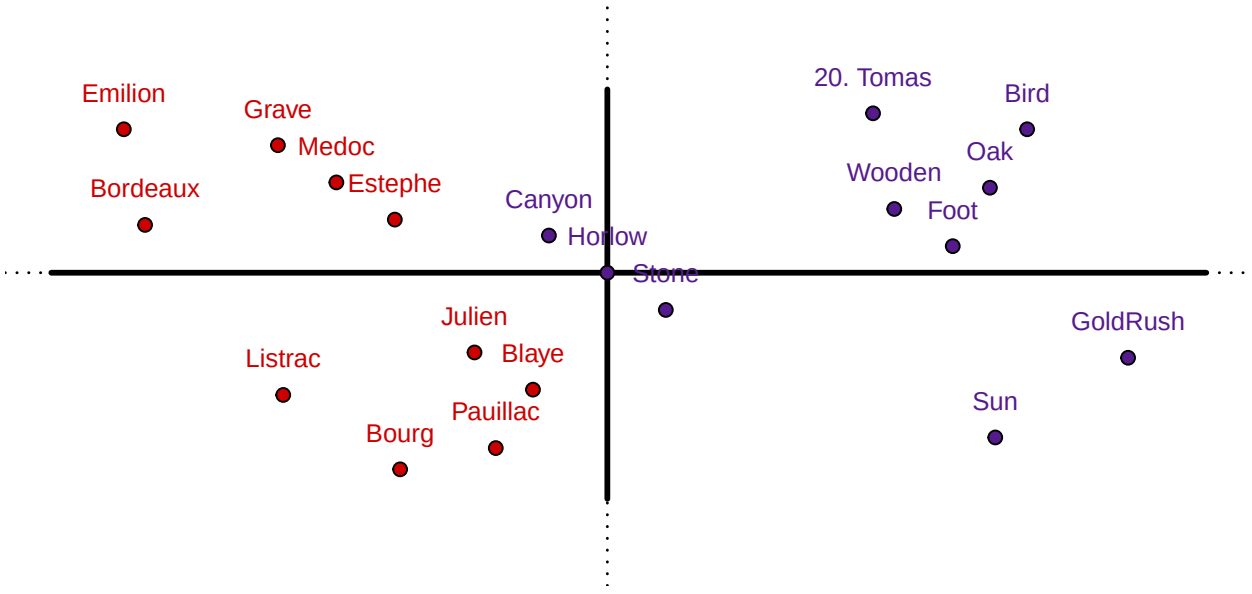
```
pdf('winesP12.pdf')
print(a1.JolieMap.1)
dev.off()# Do not forget dev.off(). The file is not saved without it
```

Other functions can be used to save graphs with other graphics formats (e.g., `jpg`, `png`, `wmf`, `postscript`, and even `xfig`),

Change the colors of the wines

The default colors generated by `epPCA` gives colors that are pretty and as different as possible. This color scheme works well most of the time; but not quite in this example, because readers expect to see some colors for red wines (a green wine is frightening, don't you think?). To change the colors of the wines, we recode (using the function `dplyr::recode`) the color vector and then use this new vector to draw the graph.

```
# Get a better color scheme than the default for the wines
col4F <- 'red3'
col4U <- 'purple4'
# recode with dplyr
col4Wines <- dplyr::recode(twentyWines$supplementary.variables$Origin,
                           'F' = col4F, 'U' = col4U  )
joli.plot2 <- prettyPlot(resPCA$ExPosition.Data$fi, dev.new = FALSE, col = col4Wines)
```

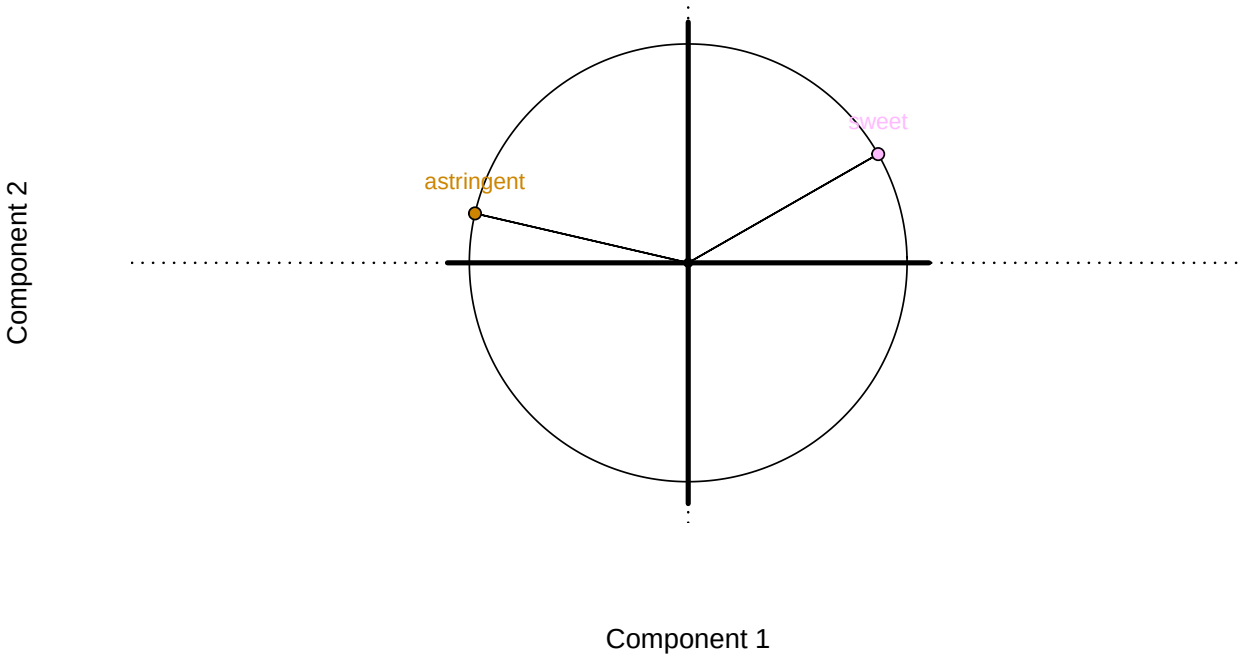



```
a2.JolieMap.2 <- recordPlot()
```

A circle of Correlation for the variables with `prettyGrapgs`

First get colors for the variables: Sweet is pink, astringent is yellowish (don't you think?).

```
col4J <-  c('plum1', 'orange3')
corC <- correlationPlotter(df, resPCA$ExPosition.Data$fi, col = col4J, dev.new = FALSE)
```




```
b0.circleOfCor <- recordPlot()
```

A set of ggplot2-based graphs from PTCA4CATA

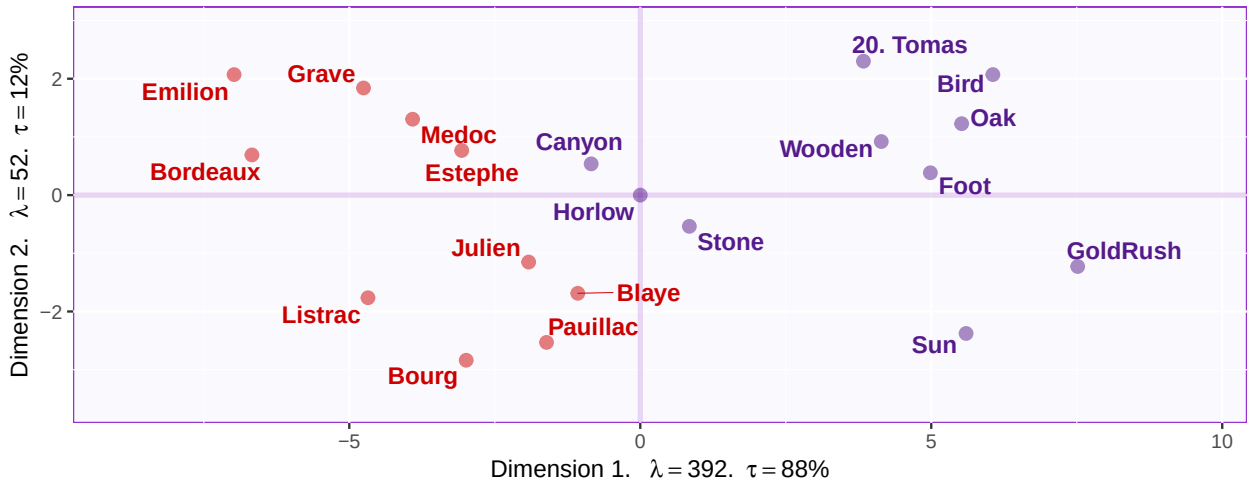
Graphs for the observations

Here we use the ggplot2-based graphic functions from PTCA4CATA to create the map:

```
# ggplot2 with PTCA4CATA ----
# look at the help for PTCA4CATA::createFactorMap
jolie.ggplot1 <- PTCA4CATA::createFactorMap(resPCA$ExPosition.Data$fi,
                              col.points = col4Wines,
                              col.labels = col4Wines)
# Create the labels for Inertia
label4Map <- createxyLabels.gen(1,2,lambda = resPCA$ExPosition.Data$eigs,
                                tau = resPCA$ExPosition.Data$t)
a3.JolieggMap <- jolie.ggplot1$zeMap + label4Map
```

To look at the map, we just need to print it:

```
print(a3.JolieggMap)
```



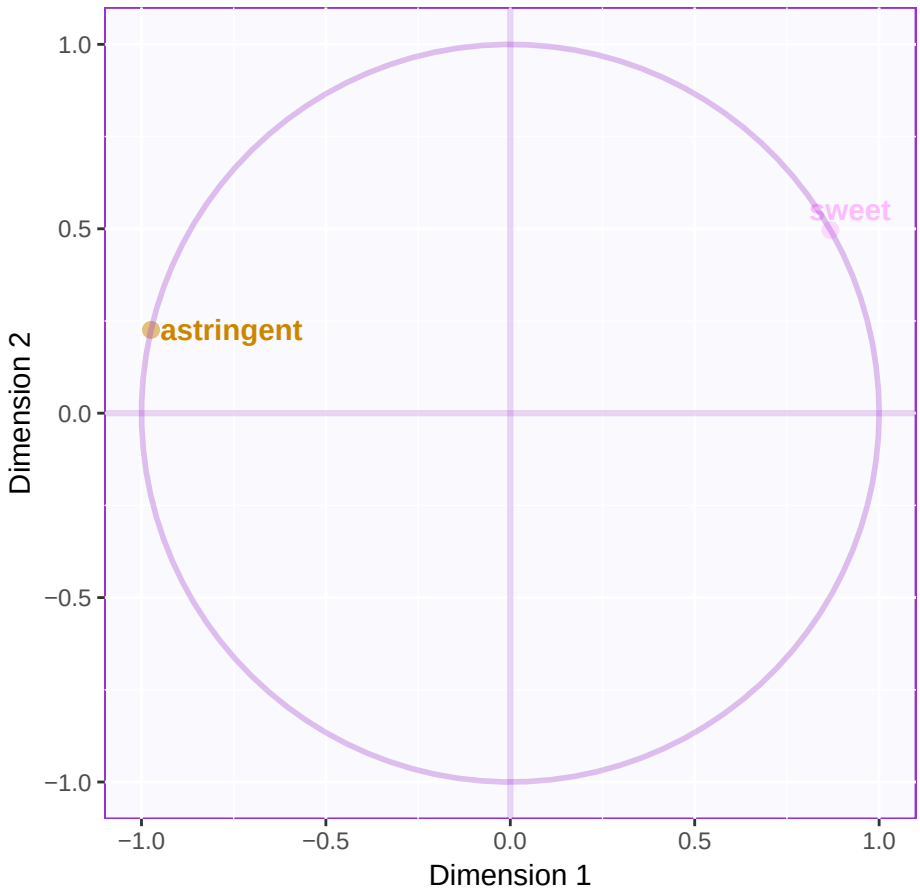
A circle of Correlation for the variables with circleOfCor

Here we first create the map with the correlations between the original variables and the factors scores; then we add the circle and the arrows. Finally we print the graph:

```
# Create the map
jolie.ggplot.J <- PTCA4CATA::createFactorMap(t(loadings.2),
                          col.points = col4J, col.labels = col4J,
                          constraints = list(minx = -1, miny = -1,
                          maxx = 1 , maxy = 1)  )
# draw the circle
b1.jolieggMap.J <- jolie.ggplot.J$zeMap + addCircleOfCor()
#  Add some arrows
arrows <- addArrows(t(loadings.2), color = col4J)
b2.jolieggMap.J <-  jolie.ggplot.J$zeMap_background +
                     jolie.ggplot.J$zeMap_text +
                     addCircleOfCor() +  arrows
```

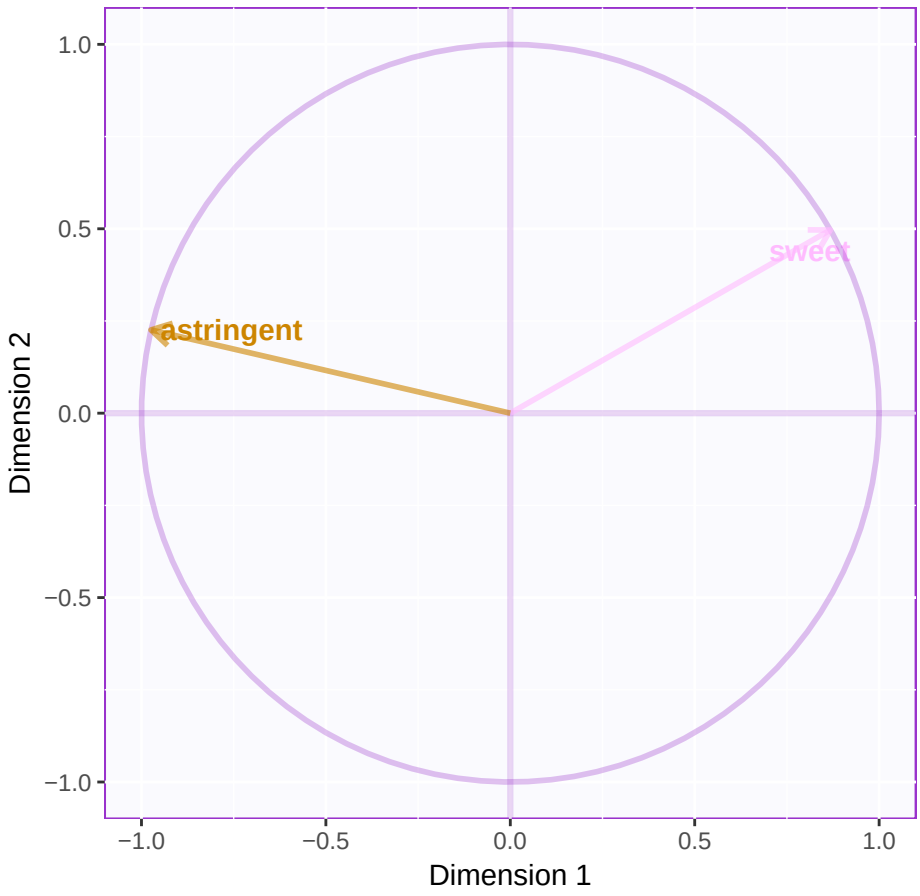
This is the circle without arrows:

```
print(b1.jolieggMap.J)
```



This is the circle with arrows (and without "dots')

```
print(b2.jolieggMap.J)
```



Another bigger PCA without scaling

The stroy

These data give the consumption in Francs of different "types of food" according to social class and number of children. The observations correspond to the average amount of money spent per month on a given type of food for a given social class and a given number of children. Because a franc spend on one item has the same value as one franc spent on another one, we want to keep the same unit of measurement for the complete space. T therefore we will analyze only the centered data (i.e., we will analyze the covariance matrix, not the correlation matrix).

The data

```
data("foodInFrance")
df.food <- foodInFrance$df.active
```

Some colors

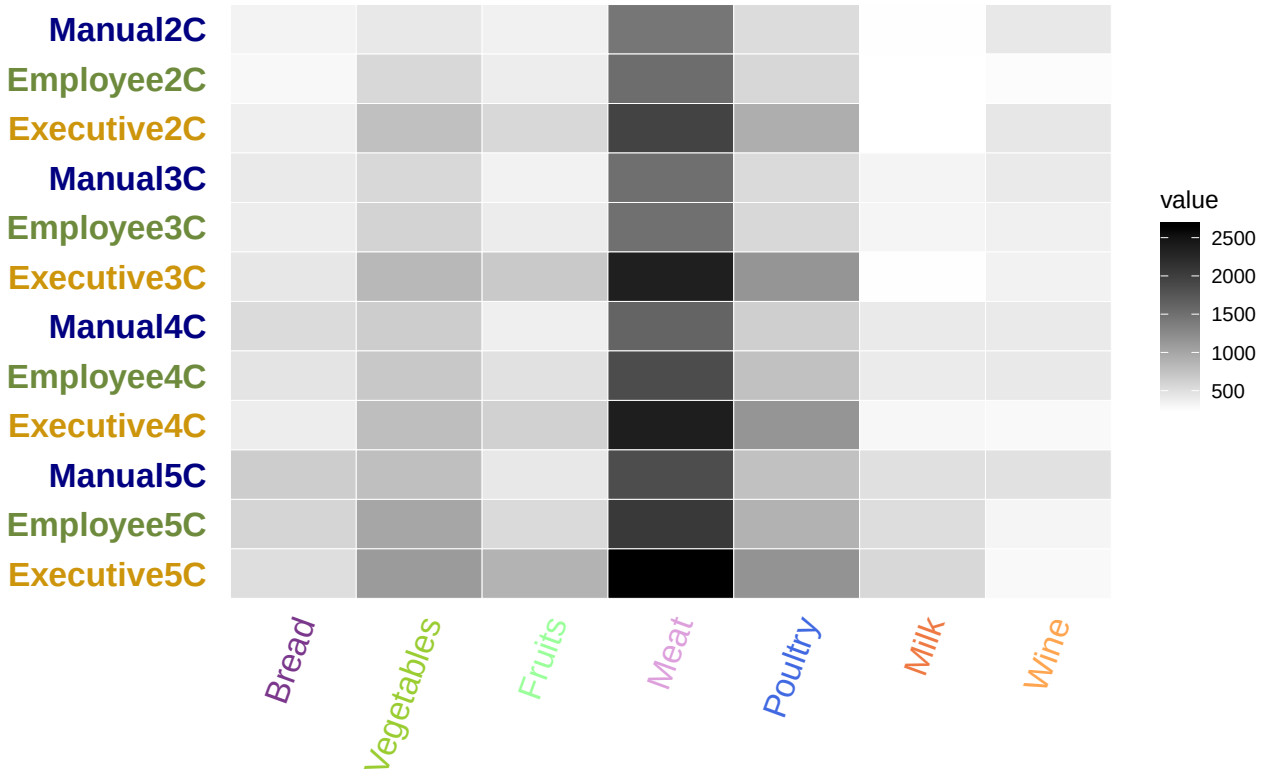
```
manual   <- "navyblue"
employee <- "darkolivegreen4"
```

```
executive <- "darkgoldenrod3"
# recode
col4I <- dplyr::recode(foodInFrance$supplementary.variables$socialClass,
                       'M' = manual , 'E' = employee, 'X' = executive  )
```

Two "heat maps"

The raw data

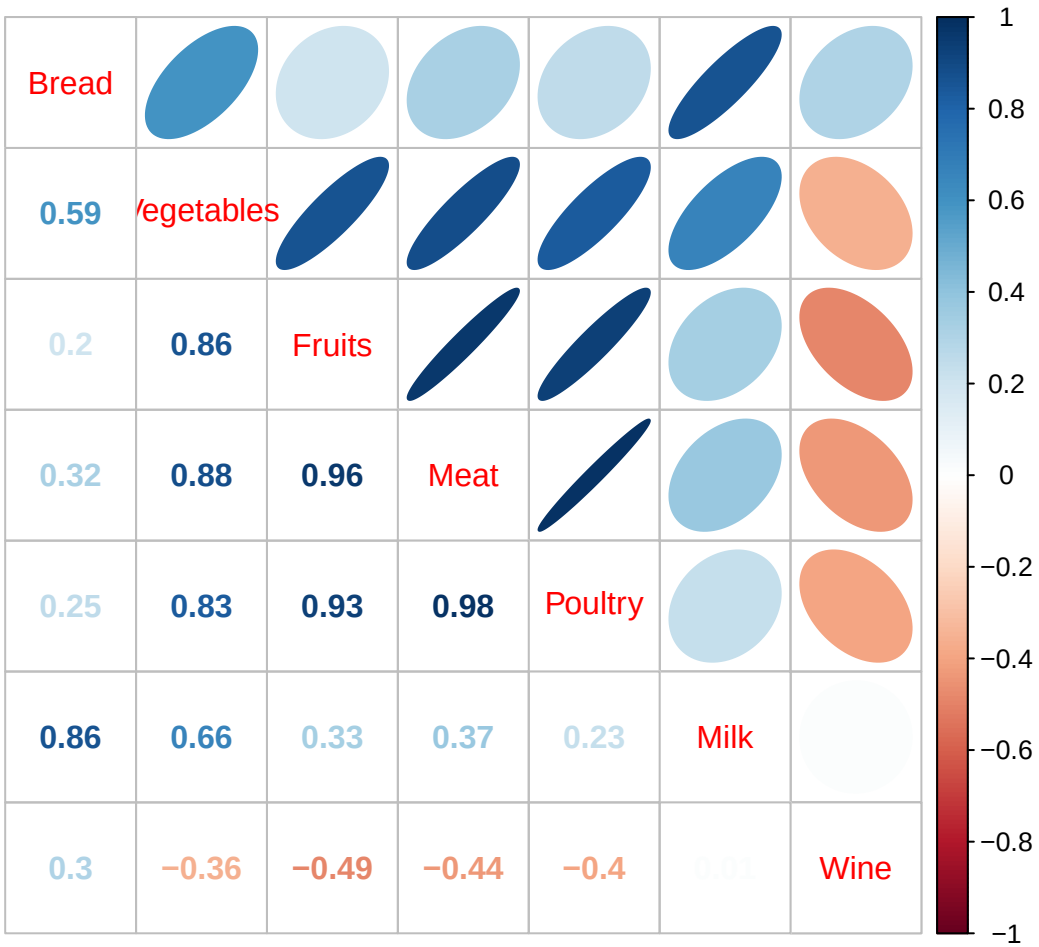
```
# define the J-colors here
col4J.food <- prettyGraphsColorSelection(NCOL(df.food))
c000.heatMapIJ.food <- makeggHeatMap4CT(df.food,
                 colorAttributes = col4J.food,
                 colorProducts = col4I,
                 fontSize.x = 15)
print(c000.heatMapIJ.food)
```



The correlation and covariance matrices

Correlation

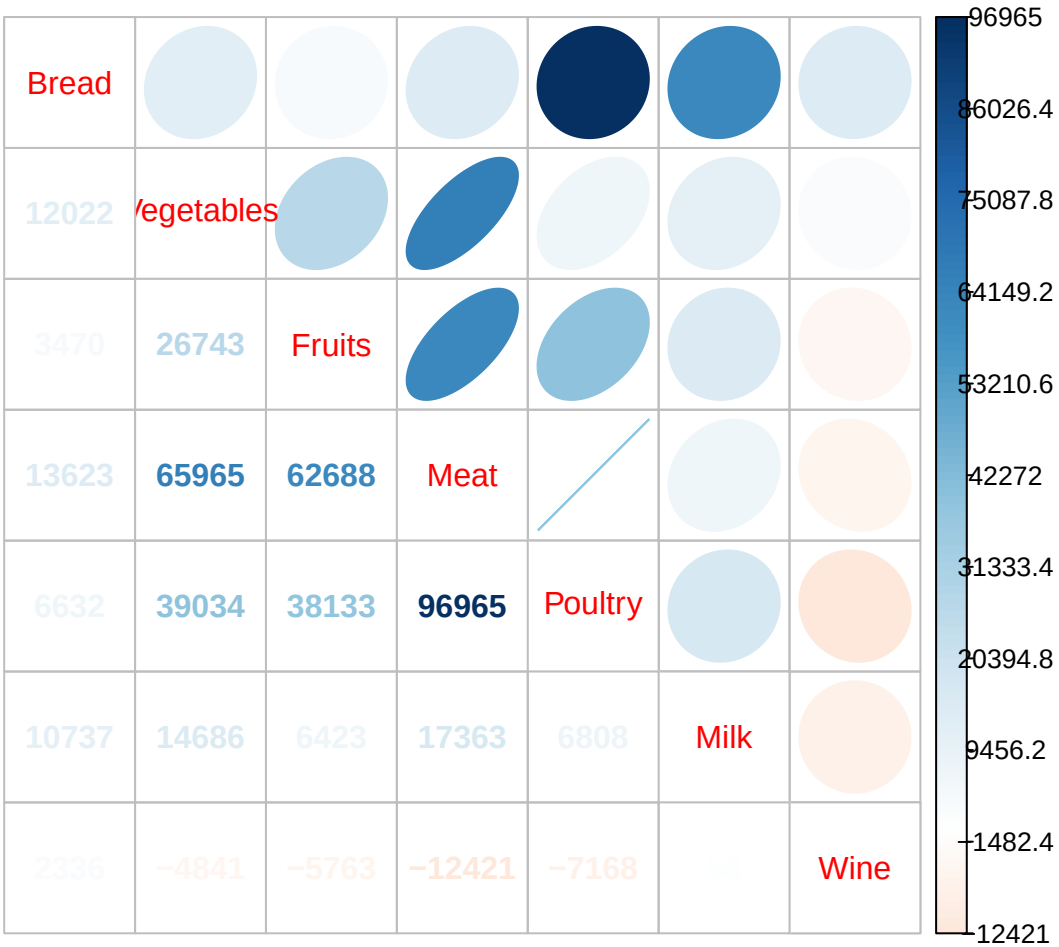
```
cor.food <- cor(df.food)
corrplot.mixed(round(cor.food, 2), lower = 'number',
               upper = 'ellipse', )
```




```
c001.map.corFood <- recordPlot()
```

Covariance

```
cov.food <- cov(df.food)
corrplot.mixed(round(cov.food), lower = 'number',
               upper = 'ellipse', is.corr =FALSE)
```




```
c002.map.covFood <- recordPlot()
```

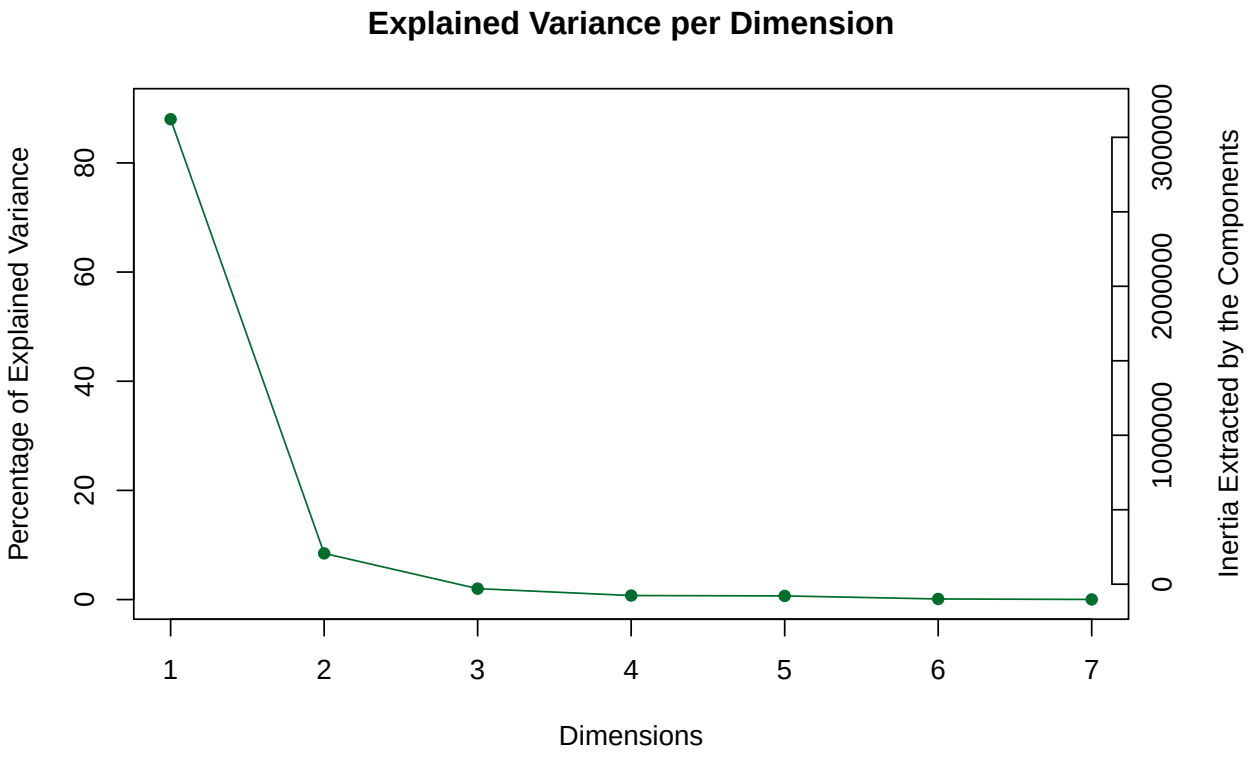
Compute the PCA

```
resPCA.food <- epPCA(df.food, scale = FALSE, graphs = FALSE)
```

Graphs

The Scree

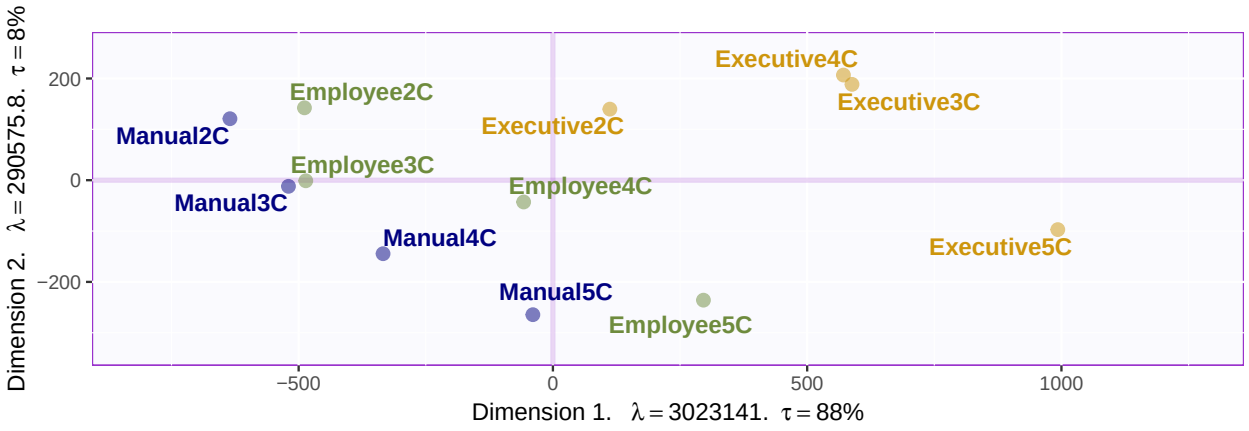
```
PlotScree(ev = resPCA.food$ExPosition.Data$eigs)
```



The scree suggests to keep two components for further investigation.

The I-set graphs

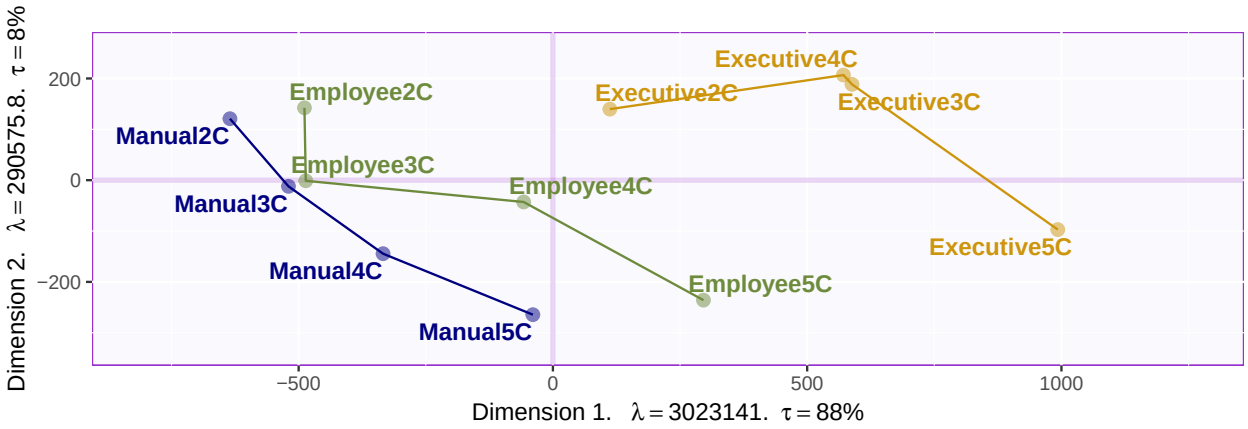
```
food.jolie.ggplot1 <- PTCA4CATA::createFactorMap(
                                resPCA.food$ExPosition.Data$fi,
                                col.points = col4I,
                                col.labels = col4I)
# Create the labels for Inertia
label4Map.food <- createxyLabels.gen(1,2,
                      lambda =resPCA.food$ExPosition.Data$eigs,
                     tau = resPCA.food$ExPosition.Data$t)
c1.JolieggMap <- food.jolie.ggplot1$zeMap + label4Map.food
print(c1.JolieggMap)
```



Add segments

To be improved

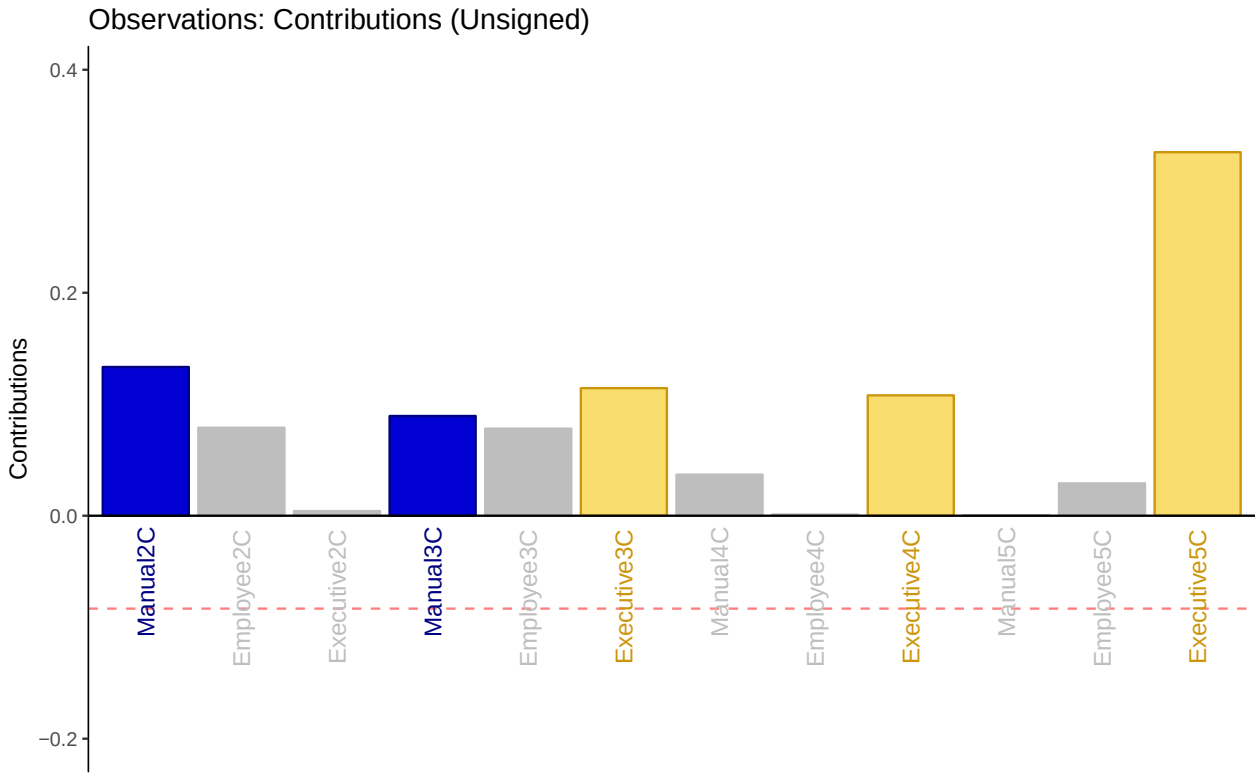
```
Fi <- resPCA.food$ExPosition.Data$fi
colnames(Fi) <- paste0('Dimension ',1:ncol(Fi))
df4seg.1 <- as.data.frame(Fi[c(1,4,7,10),])
df4seg.2 <- as.data.frame(Fi[c(2,5,8,11),])
df4seg.3 <- as.data.frame(Fi[c(3,6,9,12),])
axis1 = 1
axis2 = 2
 colnames(df4seg.1) <- paste0('Dimension ',1:ncol(Fi))
 c1a.JolieMap <-   c1.JolieggMap +
   geom_line(data = df4seg.1, col =  col4I[c(1,4,7,10)] ) +
   geom_line(data = df4seg.2, col = col4I[c(2,5,8,11)] ) +
   geom_line(data = df4seg.3, col = col4I[c(3,6,9,12)] )
 print(c1a.JolieMap)
```



Bar plots for the contributions: New ggplot `PrettyBarPlot` (under test)

Unsigned contributions

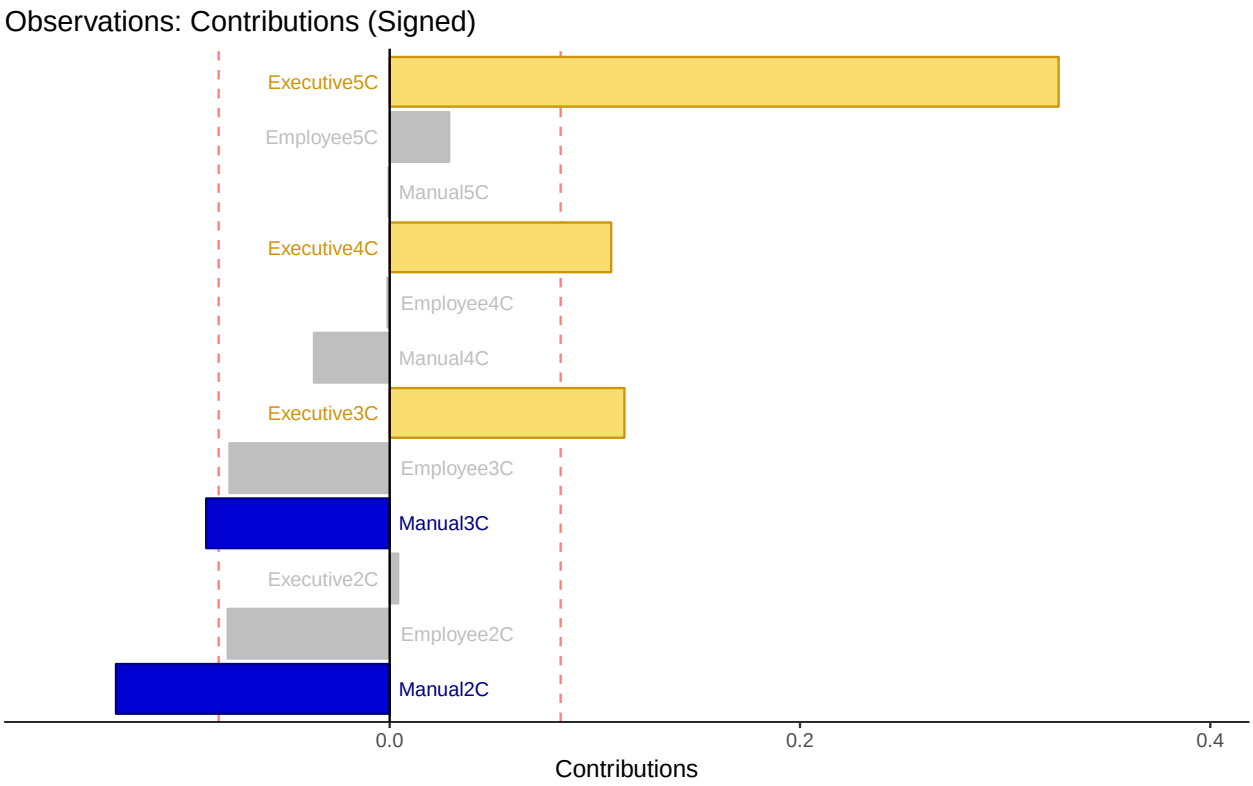
```
ctr.I <- resPCA.food$ExPosition.Data$ci[,1]
ctrI <- PrettyBarPlot2(ctr.I,
              threshold = 1 / NROW(ctr.I),
              font.size = 4,
              color4bar = gplots::col2hex(col4I),
              color4ns = 'grey',
              main = 'Observations: Contributions (Unsigned)',
              ylab = 'Contributions',
              ylim = c(-.2, 1.2*max(ctr.I)),
              horizontal = TRUE
              )
print(ctrI)
```



Signed contributions

Note that the new `PrettyBarPlot2` can be printed both horizontally or vertically (use the parameter `horizontal = FALSE`).

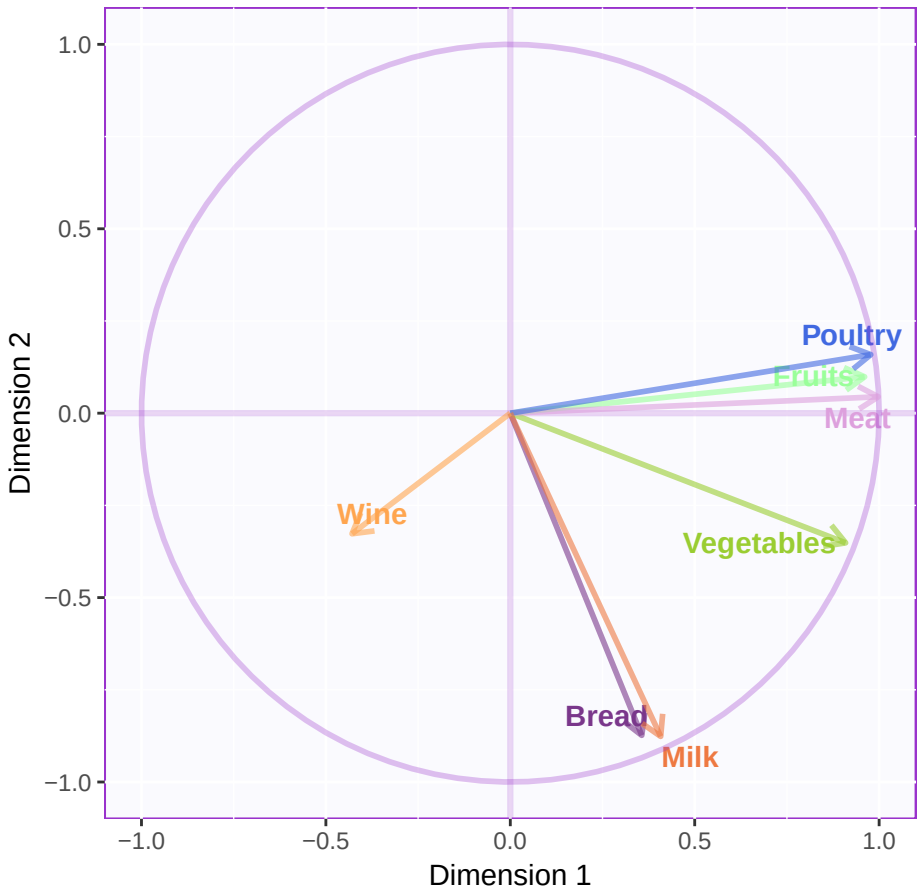
```
ctr.I.signed <-  ctr.I * sign(resPCA.food$ExPosition.Data$fi[,1])
ctrI.s <- PrettyBarPlot2(ctr.I.signed,
              threshold = 1 / NROW(ctr.I),
              font.size = 3,
              color4bar = gplots::col2hex(col4I), # we need hex code
              main = 'Observations: Contributions (Signed)',
              ylab = 'Contributions',
              ylim = c(1.2*min(ctr.I.signed), 1.2*max(ctr.I.signed) ),
              horizontal = FALSE
              )
print(ctrI.s)
```



The J-set graphs

A circle of correlation

```
# Create the map
loadings.food <- (cor(df.food, resPCA.food$ExPosition.Data$fi))
mapJ.food <- PTCA4CATA::createFactorMap(loadings.food ,
                          col.points = col4J.food,
                          col.labels = col4J.food,
                          constraints = list(minx = -1, miny = -1,
                          maxx = 1 , maxy = 1)  )
# Circle with arrows
arrows.food <- addArrows(loadings.food, color = col4J.food)
d1.jolieggMap.J <-  mapJ.food$zeMap_background +
                     mapJ.food$zeMap_text +
                     addCircleOfCor() +  arrows.food
# and print
print(d1.jolieggMap.J)
```



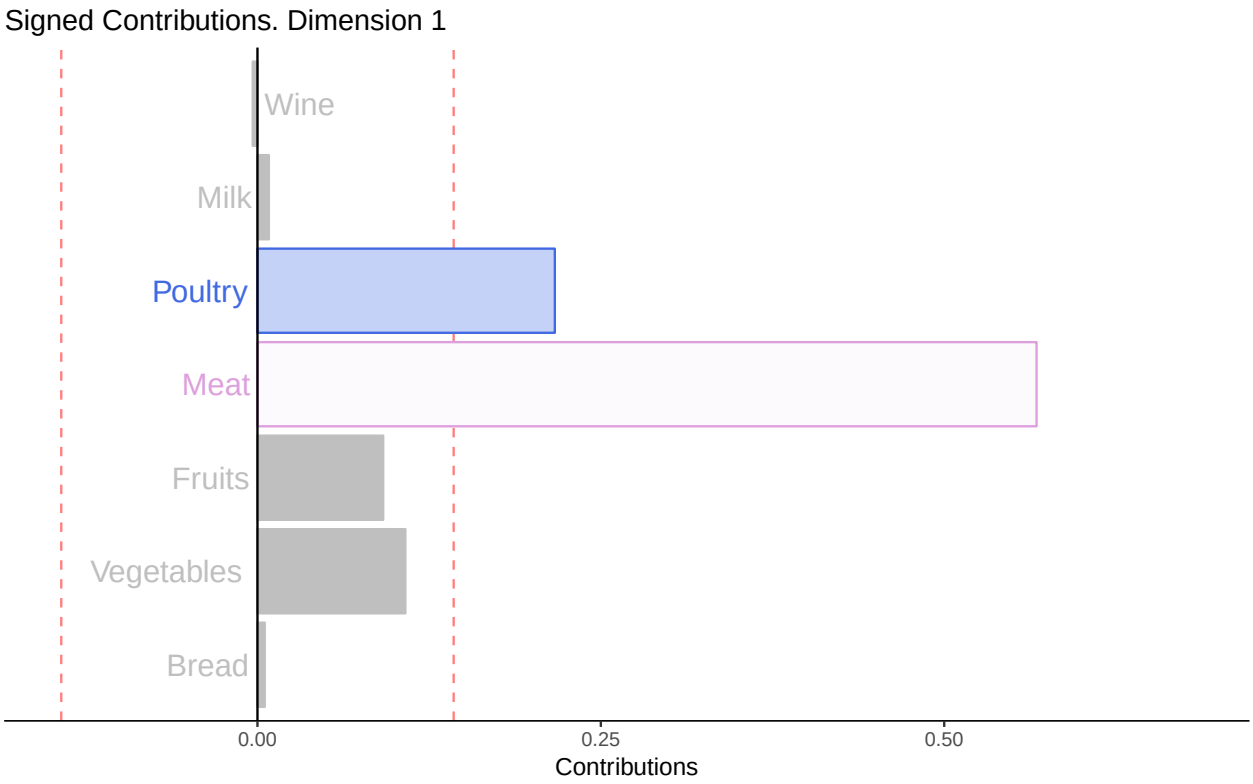
Some contributions

We will plot only the signed contributions here. Even thought the scree test would select only two components, a closer look at the data suggests that three components maybe worth looking at. With three components to investigate, we want to draw three contribution plots; So it is worth writing a short function to draw a contribution plot. Here it is:

```
create.ctrPlot <- function(res, axis = 1,
                           col = NULL,
                           set = 'J', ...){
  if (set == 'I'){
      ctr.s <- res$ExPosition.Data$ci[,axis] *
           sign(res$ExPosition.Data$fi[,axis])
  } else {
  ctr.s <- res$ExPosition.Data$cj[,axis] *
           sign(res$ExPosition.Data$fj[,axis])
  }
  zebars  <- PrettyBarPlot2(ctr.s,
              threshold = 1 / length(ctr.s),
              color4bar = gplots::col2hex(col), # we need hex code
              main = paste0('Signed Contributions. Dimension ',axis),
              ylab = 'Contributions',
              ylim = c(1.2*min(ctr.s), 1.2*max(ctr.s) ), ...)
  return(zebars)
} # end of function create.ctrPlot
```

Dimension 1

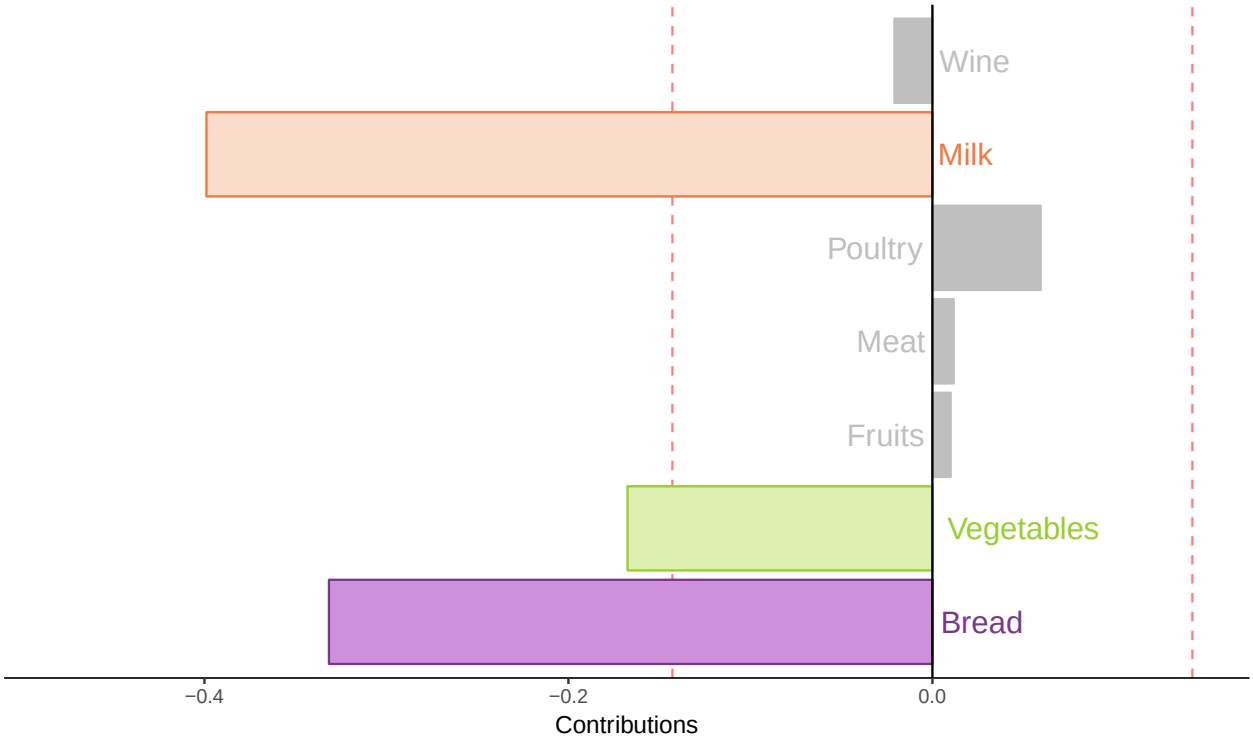
```
e1.ctrJ <- create.ctrPlot(resPCA.food, axis = 1, font.size = 5,
                          col =  col4J.food, horizontal = FALSE)
print(e1.ctrJ)
```



Dimension 2

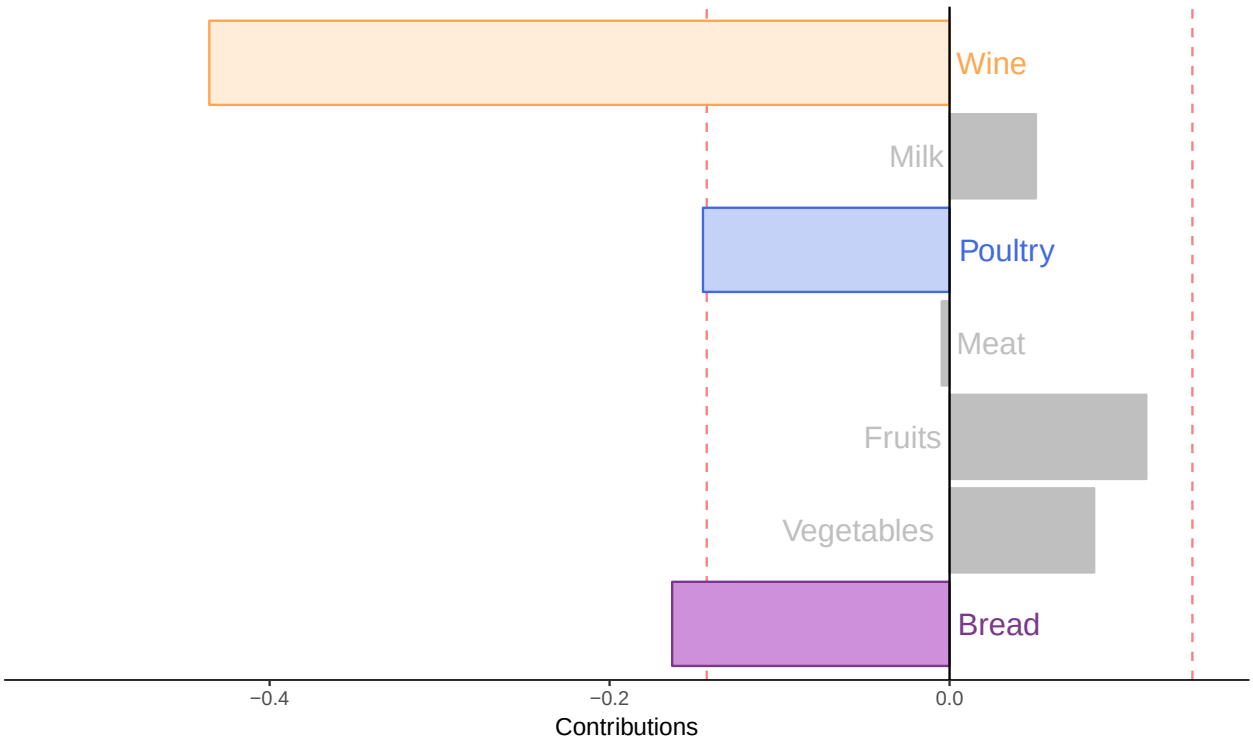
```
e2.ctrJ <- create.ctrPlot(resPCA.food, axis = 2, font.size = 5,
                          col =  col4J.food, horizontal = FALSE)
print(e2.ctrJ)
```

Signed Contributions. Dimension 2



Dimension 3

```
e3.ctrJ <- create.ctrPlot(resPCA.food, axis = 3, font.size = 5,
                          col =  col4J.food, horizontal = FALSE)
print(e3.ctrJ)
```



Save the results in a powerpoint

To save the graphs

```
listSaved <- saveGraph2pptx(
    file2Save.pptx = 'twoUnscaledPCA.pptx',
    title = "Two Unscaled PCA. Wines and 'Food in France' ",
    addGraphNames = TRUE)
```