Dimension Reduction

Erwan Le Pennec

Fall 2015

Dimension Reduction

As soon as the dimension is larger than 3, it becomes hard to visualize the raw data. Dimension reduction technique can be used to alleviate this issue by *projecting* the data in a low dimensional space, typically a 2D space. Note that those dimension reduction ideas can also be used a preprocessing step for any learning task.

Principal Component Analysis

We will start by the most classical dimension reduction algorithm, the Principal Component Analysis one. The idea is to find a subspace spanned by d' orthonormal columns $V^{(l)}$ such that the reconstruction error between the data and its projection on this subspace

$$\sum_{i=1}^{n} \|\mathbf{X}_i - m + V^t(\mathbf{X}_i - m)V\|^2$$

is as small as possible. An explicit solution to this problem is given by the space spanned by the d' eigenvectors of the $d \times d$ empirical covariance matrix of the observations.

The ACP is available in many package and will use the one provided by **FactoMineR**.

Decathlon dataset

We will first consider a *classical* dataset **decathlon**. It contains the performance of several athlets during two decathlons in 2004 the Decastar and the Olympic Game. Each observations consists of

- the 10 raw performance in the 10 events,
- the ranking in the event,
- the total number of points,
- the name of the event.

During this lab, we will mainly focus on the 10 first columns.

```
library("FactoMineR")
library("dplyr")
data(decathlon)
colnames(decathlon)
```

```
##  [1] "100m"         "Long.jump"    "Shot.put"     "High.jump"    "400m"
##  [6] "110m.hurdle"  "Discus"       "Pole.vault"   "Javeline"     "1500m"
## [11] "Rank"         "Points"       "Competition"
```

```
glimpse(decathlon)
```

```
## Observations: 41
## Variables: 13
## $ 100m        (dbl) 11.04, 10.76, 11.02, 11.02, 11.34, 11.11, 11.13, 1...
## $ Long.jump   (dbl) 7.58, 7.40, 7.30, 7.23, 7.09, 7.60, 7.30, 7.31, 6....
## $ Shot.put    (dbl) 14.83, 14.26, 14.77, 14.25, 15.19, 14.31, 13.48, 1...
## $ High.jump   (dbl) 2.07, 1.86, 2.04, 1.92, 2.10, 1.98, 2.01, 2.13, 1....
## $ 400m        (dbl) 49.81, 49.37, 48.37, 48.93, 50.42, 48.68, 48.62, 4...
## $ 110m.hurdle (dbl) 14.69, 14.05, 14.09, 14.99, 15.31, 14.23, 14.17, 1...
## $ Discus      (dbl) 43.75, 50.72, 48.95, 40.87, 46.26, 41.10, 45.67, 4...
## $ Pole.vault  (dbl) 5.02, 4.92, 4.92, 5.32, 4.72, 4.92, 4.42, 4.42, 4....
## $ Javeline    (dbl) 63.19, 60.15, 50.31, 62.77, 63.44, 51.77, 55.37, 5...
## $ 1500m       (dbl) 291.70, 301.50, 300.20, 280.10, 276.40, 278.10, 26...
## $ Rank        (int) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1, 2, 3...
## $ Points      (int) 8217, 8122, 8099, 8067, 8036, 8030, 8004, 7995, 78...
## $ Competition (fctr) Decastar, Decastar, Decastar, Decastar, Decastar,...
```

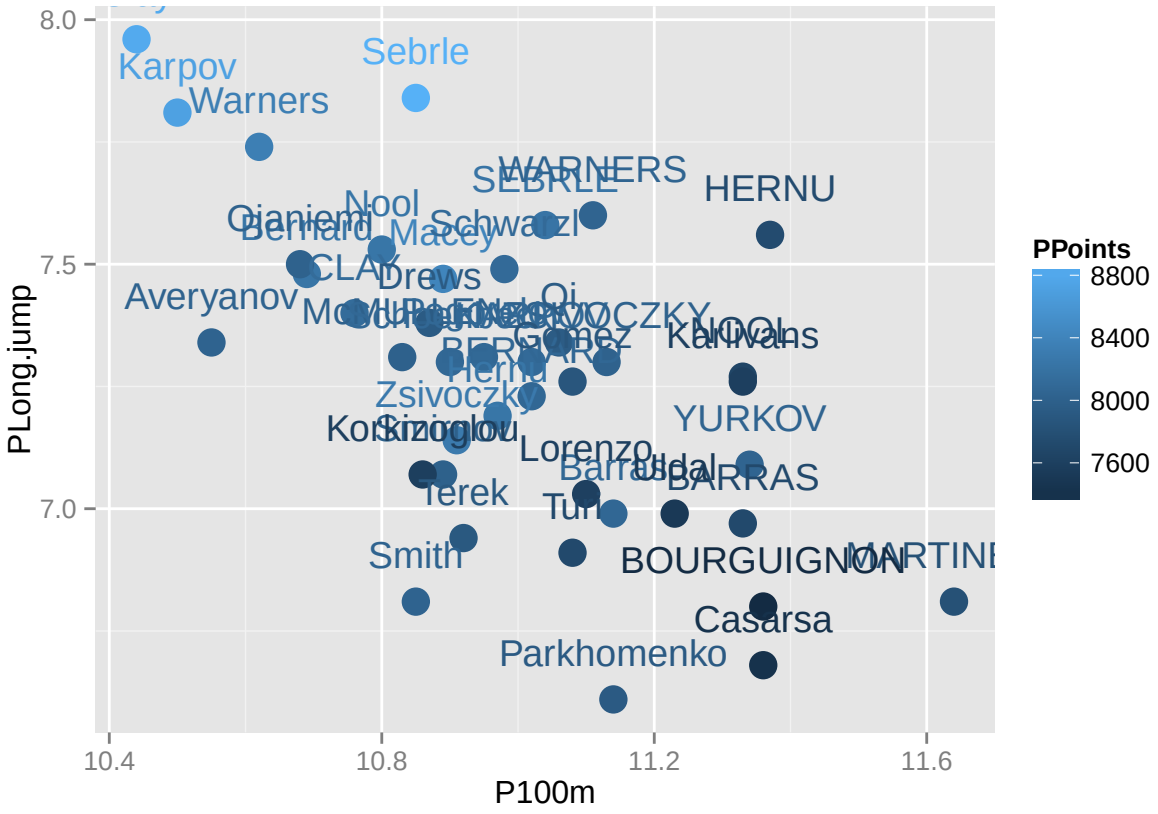
```
summary(decathlon)
```

```
##       100m         Long.jump       Shot.put       High.jump
##  Min.   :10.44   Min.   :6.61   Min.   :12.68   Min.   :1.850
##  1st Qu.:10.85   1st Qu.:7.03   1st Qu.:13.88   1st Qu.:1.920
##  Median :10.98   Median :7.30   Median :14.57   Median :1.950
##  Mean   :11.00   Mean   :7.26   Mean   :14.48   Mean   :1.977
##  3rd Qu.:11.14   3rd Qu.:7.48   3rd Qu.:14.97   3rd Qu.:2.040
##  Max.   :11.64   Max.   :7.96   Max.   :16.36   Max.   :2.150
##       400m        110m.hurdle        Discus        Pole.vault
##  Min.   :46.81   Min.   :13.97   Min.   :37.92   Min.   :4.200
##  1st Qu.:48.93   1st Qu.:14.21   1st Qu.:41.90   1st Qu.:4.500
##  Median :49.40   Median :14.48   Median :44.41   Median :4.800
##  Mean   :49.62   Mean   :14.61   Mean   :44.33   Mean   :4.762
##  3rd Qu.:50.30   3rd Qu.:14.98   3rd Qu.:46.07   3rd Qu.:4.920
##  Max.   :53.20   Max.   :15.67   Max.   :51.65   Max.   :5.400
##     Javeline         1500m            Rank           Points
##  Min.   :50.31   Min.   :262.1   Min.   : 1.00   Min.   :7313
##  1st Qu.:55.27   1st Qu.:271.0   1st Qu.: 6.00   1st Qu.:7802
##  Median :58.36   Median :278.1   Median :11.00   Median :8021
##  Mean   :58.32   Mean   :279.0   Mean   :12.12   Mean   :8005
##  3rd Qu.:60.89   3rd Qu.:285.1   3rd Qu.:18.00   3rd Qu.:8122
##  Max.   :70.52   Max.   :317.0   Max.   :28.00   Max.   :8893
##    Competition
##  Decastar:13
##  OlympicG:28
##
##
##
##
```

Pairwise analysis

1. We may plot the two first coordinates (**100m** and **Long.jump**) adding the **Points** information as a color.

```
library("ggplot2")
decathlon2 <- decathlon
names(decathlon2) <- paste("P", names(decathlon), sep = "") #Fix for the column names starting by a num
ggplot(data= decathlon2, aes(x = P100m, y = PLong.jump, color = PPoints)) + geom_point(size = 5) +
  geom_text(label = row.names(decathlon), vjust = -1.25)
```




```
  guides(color = FALSE)
```

```
## $colour
## [1] FALSE
##
## attr(,"class")
## [1] "guides"
```

Do you think those values are correlated?

2. Let's have a first look on the data by computing the correlation matrix between the 10 first variables.

```
cor(decathlon[,1:10])
```

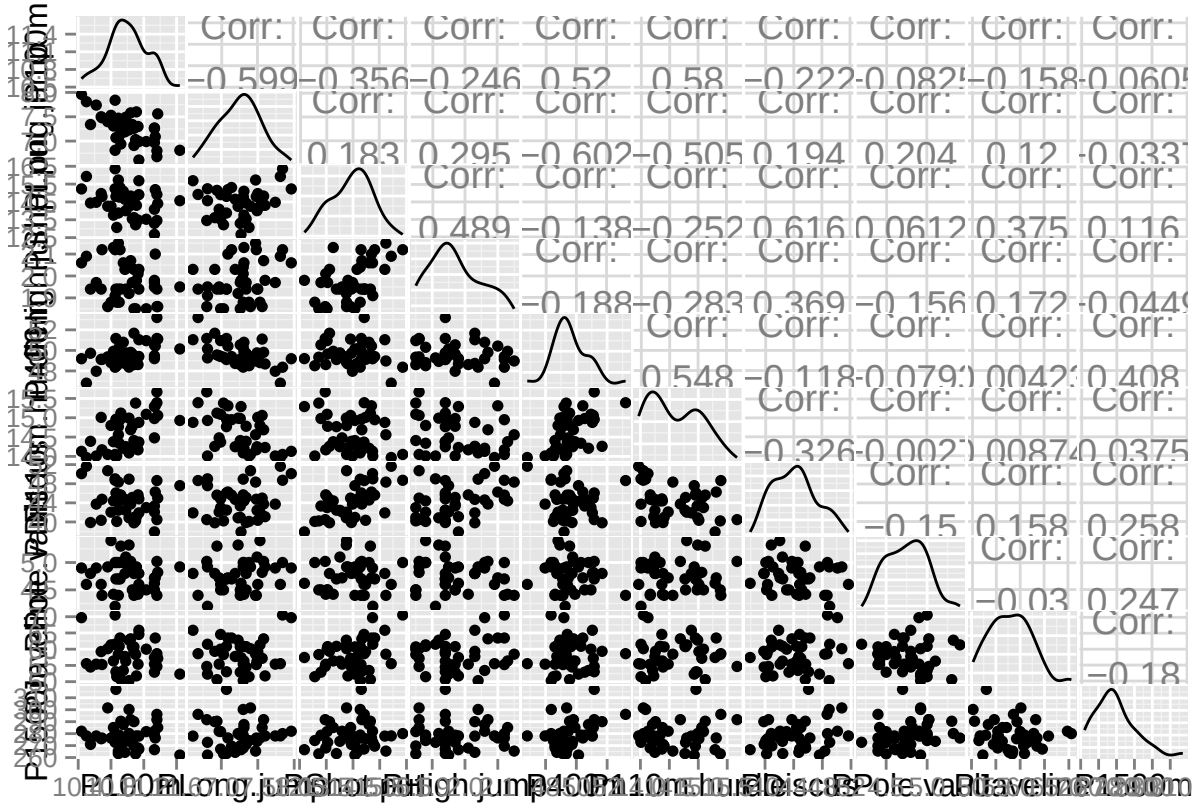
```
##                   100m   Long.jump    Shot.put   High.jump        400m
## 100m         1.00000000 -0.59867767 -0.35648227 -0.24625292  0.52029815
## Long.jump   -0.59867767  1.00000000  0.18330436  0.29464444 -0.60206261
## Shot.put    -0.35648227  0.18330436  1.00000000  0.48921153 -0.13843291
## High.jump   -0.24625292  0.29464444  0.48921153  1.00000000 -0.18795692
```

```
## 400m          0.52029815 -0.60206262 -0.13843292 -0.18795693  1.000000000
## 110m.hurdle   0.57988893 -0.50541009 -0.25161571 -0.28328909  0.547987756
## Discus       -0.22170757  0.19431009  0.61576810  0.36921834 -0.117879365
## Pole.vault   -0.08253683  0.20401411  0.06118185 -0.15618074 -0.079292469
## Javeline     -0.15774645  0.11975893  0.37495551  0.17188009  0.004232096
## 1500m        -0.06054645 -0.03368613  0.11580306 -0.04490252  0.408106432
##                110m.hurdle     Discus   Pole.vault     Javeline       1500m
## 100m          0.579888931 -0.2217076 -0.082536834 -0.157746452 -0.06054645
## Long.jump    -0.505410086  0.1943101  0.204014112  0.119758933 -0.03368613
## Shot.put     -0.251615714  0.6157681  0.061181853  0.374955509  0.11580306
## High.jump    -0.283289090  0.3692183 -0.156180742  0.171880092 -0.04490252
## 400m          0.547987756 -0.1178794 -0.079292469  0.004232096  0.40810643
## 110m.hurdle   1.000000000 -0.3262010 -0.002703885  0.008743251  0.03754024
## Discus       -0.326200961  1.0000000 -0.150072400  0.157889799  0.25817510
## Pole.vault   -0.002703885 -0.1500724  1.000000000 -0.030000603  0.24744778
## Javeline      0.008743251  0.1578898 -0.030000603  1.000000000 -0.18039313
## 1500m         0.037540240  0.2581751  0.247447780 -0.180393128  1.00000000
```

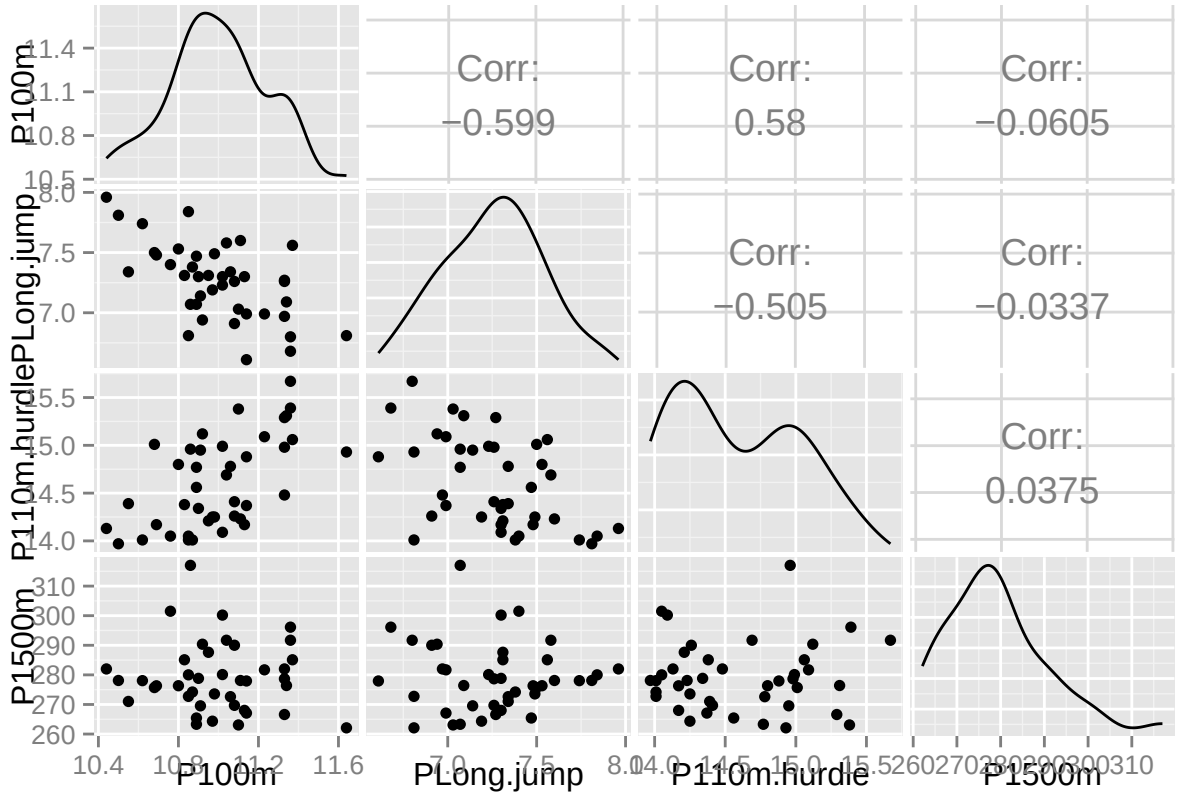
Can you pick the most correlated variables and the least ones?

3. We may look at the pairwise scatterplot to confirm those findings.

```
library(GGally)
ggpairs(decathlon2[,1:10])
```



```
ggpairs(decathlon2, columns = c(1,2,6,10))
```



3D plot

Now that we have visualize, our dataset in 2D we can try to explore it in 3D using the **rgl** package.

4. Use `plot3d` to display the 3D scatterplot of the first three variables. The plot is interactive and you can play with it.

```
library(rgl)
library(scales)
plot3d(as.matrix(decathlon[,1:3]), type = "s", size = 5,
       col = cscale(decathlon$Points,seq_gradient_pal("#132B43", high = "#56B1F7")))
```

Your browser does not support the HTML5 canvas element.

You must enable Javascript to view this page properly.

Can you find the projection that appears to lose the less information?

5. This task is similar to the PCA where we look a the subspace that minimizes the error between the data and its projection. We can draw the ellipse corresponding to the eigenvectors and the eigenvalues of the covariance matrix to verify this.

```
plot3d(as.matrix(decathlon[,1:3]), type = "s", size = 5,
       col = cscale(decathlon$Points,seq_gradient_pal("#132B43", high = "#56B1F7")))
plot3d(ellipse3d(cov(decathlon[,1:3]), centre = colMeans(decathlon[,1:3])), col="grey", alpha=0.25, add
par3d(FOV=1)
```

Your browser does not support the HTML5 canvas element.

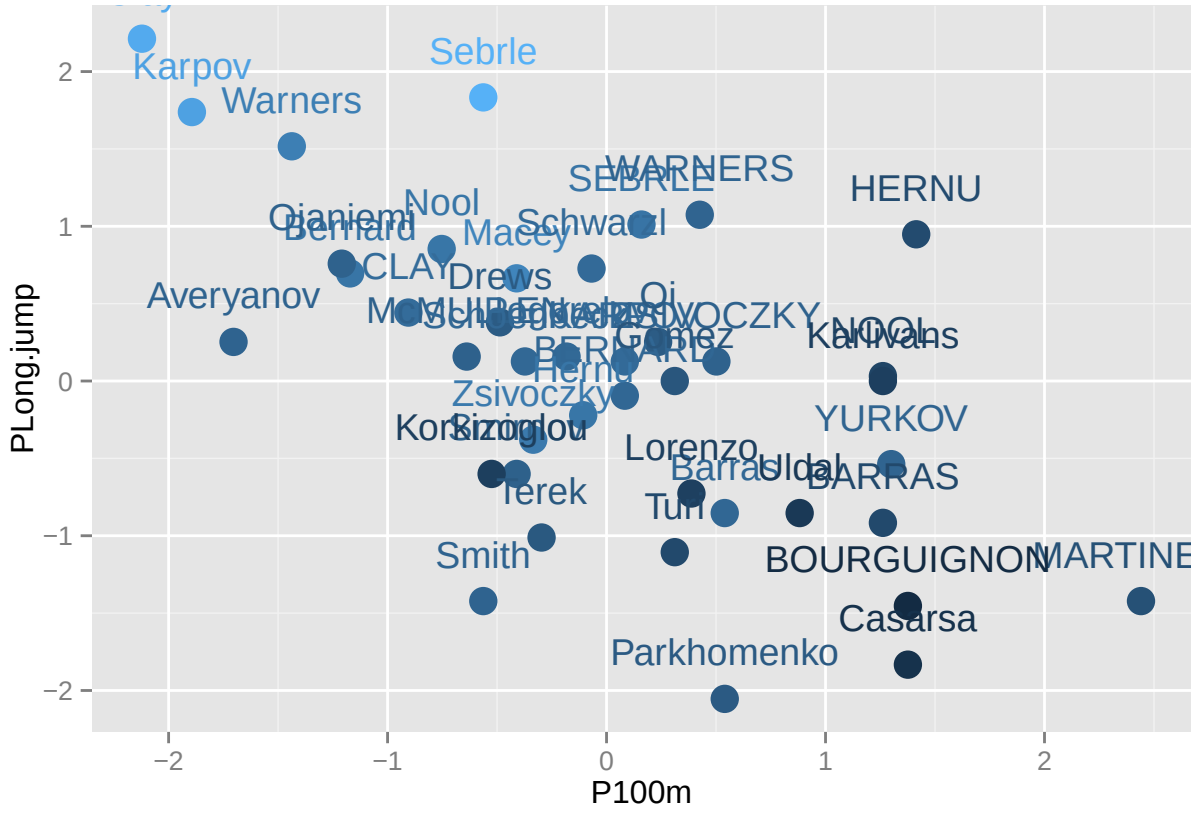
You must enable Javascript to view this page properly.

Can you verify that the best subspace is defined by the direction of the eigenvectors with the largest eigenvalues?

Scaling?

So far, we have use the raw performance. That is we compare duration with length as well as 100m time with 1500m time. Obviously, this is not a good idea. All those values should be measured in a common scale. The most classical technique is to *rescale* by substracting its mean and dividing by the standard deviation. We may repeat all the previous experiments with those rescaled data.

```
decathlonR <- decathlon
decathlonR[1:10] <- scale(decathlonR[1:10])
decathlonR2 <- decathlonR
names(decathlonR2) <- paste("P", names(decathlon), sep = "") #Fix for the column names starting by a nu
ggplot(data= decathlonR2, aes(x = P100m, y = PLong.jump, color = PPoints)) + geom_point(size = 5) +
  geom_text(label = row.names(decathlon), vjust = -1.25) +
  guides(color = FALSE)
```

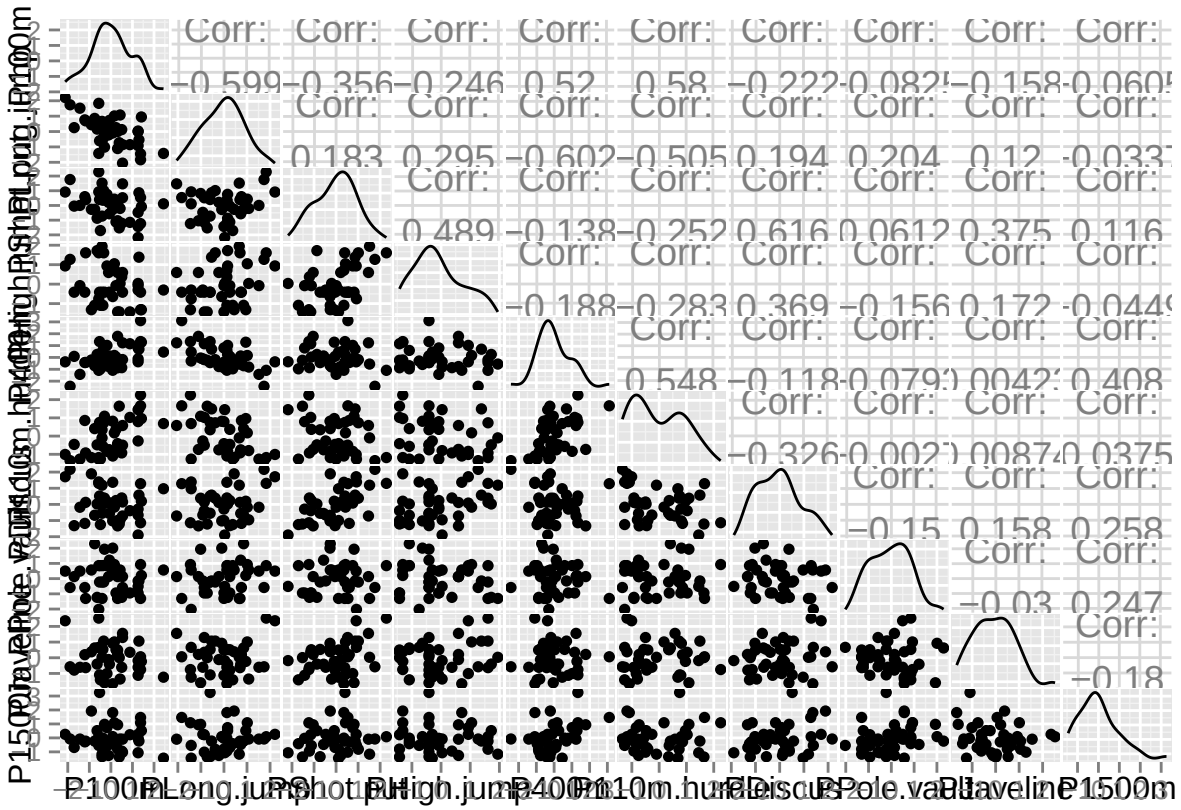



```
cor(decathlonR[,1:10])
```

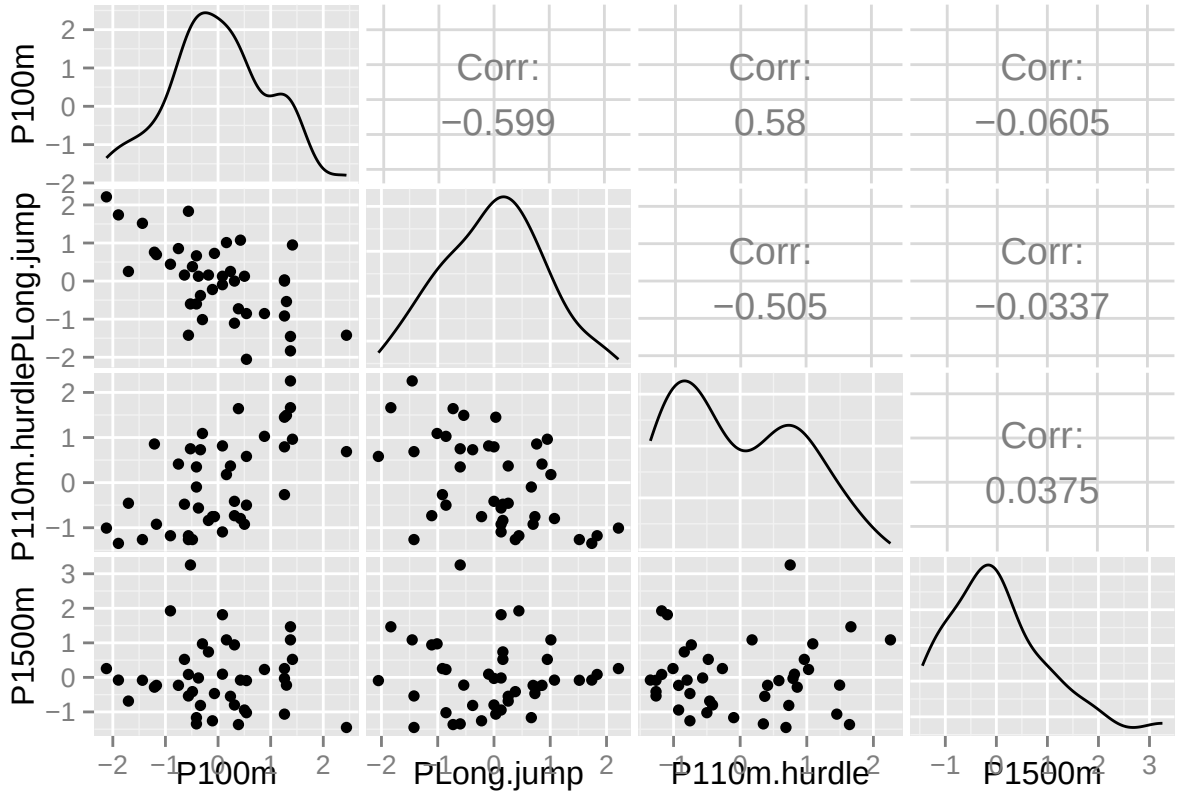
```
##                   100m   Long.jump    Shot.put   High.jump        400m
## 100m         1.00000000 -0.59867767 -0.35648227 -0.24625292  0.520298155
## Long.jump   -0.59867767  1.00000000  0.18330436  0.29464444 -0.602062618
```

```
## Shot.put    -0.35648227  0.18330436  1.00000000  0.48921153 -0.138432919
## High.jump   -0.24625292  0.29464444  0.48921153  1.00000000 -0.187956928
## 400m         0.52029815 -0.60206262 -0.13843292 -0.18795693  1.000000000
## 110m.hurdle  0.57988893 -0.50541009 -0.25161571 -0.28328909  0.547987756
## Discus      -0.22170757  0.19431009  0.61576810  0.36921834 -0.117879365
## Pole.vault  -0.08253683  0.20401411  0.06118185 -0.15618074 -0.079292469
## Javeline    -0.15774645  0.11975893  0.37495551  0.17188009  0.004232096
## 1500m       -0.06054645 -0.03368613  0.11580306 -0.04490252  0.408106432
##              110m.hurdle     Discus   Pole.vault     Javeline       1500m
## 100m         0.579888931 -0.2217076 -0.082536834 -0.157746452 -0.06054645
## Long.jump   -0.505410086  0.1943101  0.204014112  0.119758933 -0.03368613
## Shot.put    -0.251615714  0.6157681  0.061181853  0.374955509  0.11580306
## High.jump   -0.283289090  0.3692183 -0.156180742  0.171880092 -0.04490252
## 400m         0.547987756 -0.1178794 -0.079292469  0.004232096  0.40810643
## 110m.hurdle  1.000000000 -0.3262010 -0.002703885  0.008743251  0.03754024
## Discus      -0.326200961  1.0000000 -0.150072400  0.157889799  0.25817510
## Pole.vault  -0.002703885 -0.1500724  1.000000000 -0.030000603  0.24744778
## Javeline     0.008743251  0.1578898 -0.030000603  1.000000000 -0.18039313
## 1500m        0.037540240  0.2581751  0.247447780 -0.180393128  1.00000000
```

```
ggpairs(decathlonR2[,1:10])
```



```
ggpairs(decathlonR2, columns = c(1,2,6,10))
```




```
plot3d(as.matrix(decathlonR[,1:3]), type = "s", size = 5,
       col = cscale(decathlon$Points,seq_gradient_pal("#132B43", high = "#56B1F7")))
plot3d(ellipse3d(cov(decathlonR[,1:3]), centre = colMeans(decathlonR[,1:3])), col="grey", alpha=0.25, a
par3d(FOV=1)
```

Your browser does not support the HTML5 canvas element.

You must enable Javascript to view this page properly.

6. Which results are similar and which are different? Why?

Principal Component Analysis

7. The package **FactoMineR** contains a PCA function that we are going to use.

```
library("FactoMineR")
PCADecathlonR <- PCA(decathlonR[1:10], graph = FALSE)
str(PCADecathlonR)
```

```
## List of 5
##  $ eig :'data.frame':    10 obs. of  3 variables:
##   ..$ eigenvalue                    : num [1:10] 3.272 1.737 1.405 1.057 0.685 ...
##   ..$ percentage of variance        : num [1:10] 32.72 17.37 14.05 10.57 6.85 ...
##   ..$ cumulative percentage of variance: num [1:10] 32.7 50.1 64.1 74.7 81.6 ...
```

```
##  $ var :List of 4
##   ..$ coord  : num [1:10, 1:5] -0.775 0.742 0.623 0.572 -0.68 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:10] "100m" "Long.jump" "Shot.put" "High.jump" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ cor    : num [1:10, 1:5] -0.775 0.742 0.623 0.572 -0.68 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:10] "100m" "Long.jump" "Shot.put" "High.jump" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ cos2   : num [1:10, 1:5] 0.6 0.55 0.388 0.327 0.462 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:10] "100m" "Long.jump" "Shot.put" "High.jump" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ contrib: num [1:10, 1:5] 18.3 16.8 11.8 10 14.1 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:10] "100m" "Long.jump" "Shot.put" "High.jump" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##  $ ind :List of 4
##   ..$ coord  : num [1:41, 1:5] 0.792 1.235 1.358 -0.61 -0.586 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:41] "SEBRLE" "CLAY" "KARPOV" "BERNARD" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ cos2   : num [1:41, 1:5] 0.1117 0.124 0.1599 0.0487 0.0377 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:41] "SEBRLE" "CLAY" "KARPOV" "BERNARD" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ contrib: num [1:41, 1:5] 0.467 1.137 1.375 0.277 0.256 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:41] "SEBRLE" "CLAY" "KARPOV" "BERNARD" ...
##   .. .. ..$ : chr [1:5] "Dim.1" "Dim.2" "Dim.3" "Dim.4" ...
##   ..$ dist   : Named num [1:41] 2.37 3.51 3.4 2.76 3.02 ...
##   .. ..- attr(*, "names")= chr [1:41] "SEBRLE" "CLAY" "KARPOV" "BERNARD" ...
##  $ svd :List of 3
##   ..$ vs: num [1:10] 1.809 1.318 1.185 1.028 0.828 ...
##   ..$ U : num [1:41, 1:5] 0.438 0.683 0.751 -0.337 -0.324 ...
##   ..$ V : num [1:10, 1:5] -0.428 0.41 0.344 0.316 -0.376 ...
##  $ call:List of 9
##   ..$ row.w     : num [1:41] 0.0244 0.0244 0.0244 0.0244 0.0244 ...
##   ..$ col.w     : num [1:10] 1 1 1 1 1 1 1 1 1 1
##   ..$ scale.unit: logi TRUE
##   ..$ ncp       : num 5
##   ..$ centre    : num [1:10] -1.46e-16 5.13e-16 6.44e-16 -9.71e-17 -1.03e-15 ...
##   ..$ ecart.type: num [1:10] 0.988 0.988 0.988 0.988 0.988 ...
##   ..$ X         :'data.frame':   41 obs. of  10 variables:
##   .. ..$ 100m       : num [1:41] 0.1595 -0.905 0.0835 0.0835 1.3001 ...
##   .. ..$ Long.jump  : num [1:41] 1.0114 0.4425 0.1264 -0.0948 -0.5373 ...
##   .. ..$ Shot.put   : num [1:41] 0.428 -0.263 0.355 -0.275 0.865 ...
##   .. ..$ High.jump  : num [1:41] 1.047 -1.313 0.71 -0.639 1.385 ...
##   .. ..$ 400m       : num [1:41] 0.168 -0.214 -1.081 -0.595 0.697 ...
##   .. ..$ 110m.hurdle: num [1:41] 0.178 -1.178 -1.093 0.814 1.493 ...
##   .. ..$ Discus     : num [1:41] -0.17 1.893 1.369 -1.023 0.573 ...
##   .. ..$ Pole.vault : num [1:41] 0.926 0.567 0.567 2.006 -0.153 ...
##   .. ..$ Javeline   : num [1:41] 1.01 0.38 -1.659 0.923 1.061 ...
##   .. ..$ 1500m      : num [1:41] 1.0858 1.9254 1.814 0.0921 -0.2249 ...
```

```
##   ..$ row.w.init: num [1:41] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ call      : language PCA(X = decathlonR[1:10], graph = FALSE)
##  - attr(*, "class")= chr [1:2] "PCA" "list "
```

Let's look at those results:

- **eig** contains the eigenvalues (as well as the percentage of variance and the cumulative variance)
- **ind** contains the new coordinates $V^{(l)t}\mathbf{X}$ (as well as related information)
- **var** contains the projection of the original axis in the new ones (as well as related information)
- **svd** contains the SVD decomposition of $\mathbf{X}_{(n)}$ ($\mathbf{X}_{(n)} = UD(\lambda)V^t$)

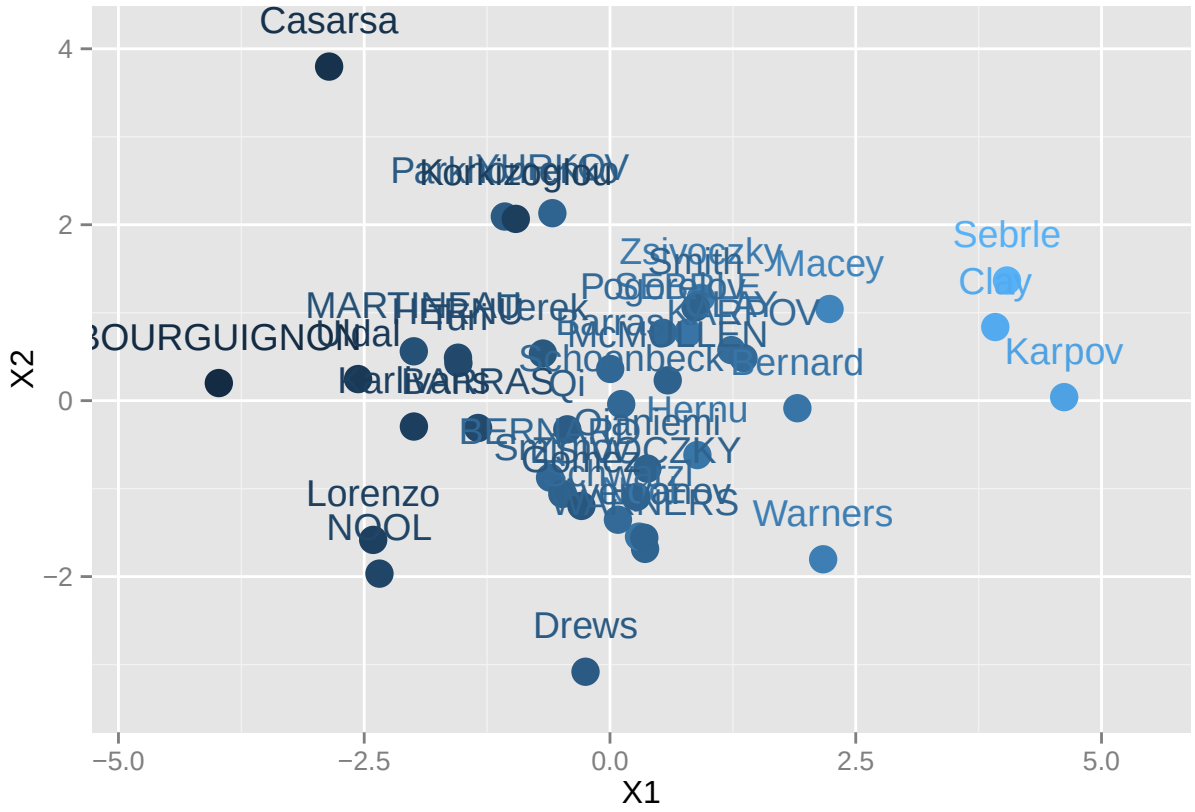
Can you check that the new coordinates are given by $UD(\lambda)$ and the new basis vectors by $VD(\lambda)$?

7. For those interested, can you recover the PCA using only linear algebra?

```
CDecathlonR <- cov(as.matrix(decathlonR[1:10]))
EDecathlonR <- eigen(CDecathlonR)
UDecathlonR <- EDecathlonR$vectors
LambdaDecathlonR <- EDecathlonR$values

PCADecathlonR2 <- as.matrix(decathlonR[1:10]) %*% UDecathlonR
```

8. Plot the points in the new coordinates

```
ggplot(data= data.frame(X1 = PCADecathlonR$ind$coord[,1], X2 = PCADecathlonR$ind$coord[,2], Col = decatl
       aes(x = X1, y = X2, color = Col)) + geom_point(size = 5) +
  geom_text(label = row.names(decathlon), vjust = -1.25) +
  scale_x_continuous(expand = c(.15,0)) + scale_y_continuous(expand = c(.1,0)) +
  guides(color = FALSE)
```



9. We are now interested in the interpretation of the axis themselves. We can compute the projection of those axis on the new ones, the ones made from the new coordinates $UD(\lambda)$. By construction, they are orthogonomal but no necessarily normalized. This however the case for U so that the projection of $\mathbf{X}_{(n)}$ is given by $P_{(n)} = U^t \mathbf{X}_{(n)} U$

Verify that indeed the **coords** of var are given by $U^t \mathbf{X}_{(n)}$ up to a scaling...

```
t(PCADecathlonR$svd$U) %*% as.matrix(decathlonR[,1:10]) / 40
```

```
##             100m   Long.jump    Shot.put  High.jump        400m
## [1,] -0.78434405  0.75111624  0.63023580  0.5790505 -0.68805263
## [2,]  0.18946683 -0.34971241  0.60573595  0.3546452  0.57651181
## [3,] -0.18669800  0.18447463 -0.02366886 -0.2627358  0.13310293
## [4,] -0.03828807  0.10305011  0.19295930 -0.1372787  0.02966599
## [5,]  0.30595053  0.03713369  0.11253163  0.5623397 -0.08878094
##      110m.hurdle      Discus Pole.vault   Javeline       1500m
## [1,]  -0.7555158  0.55932972  0.0509669  0.2805533 -0.05879854
## [2,]   0.2316356  0.61384552 -0.1825974  0.3209270  0.48011497
## [3,]  -0.0937882  0.04348584  0.7003502 -0.3944960  0.79185923
## [4,]   0.2944440 -0.26289729  0.5583856  0.7211258 -0.16309022
## [5,]   0.1663623 -0.10612937  0.3340583 -0.3089195 -0.15546956
```

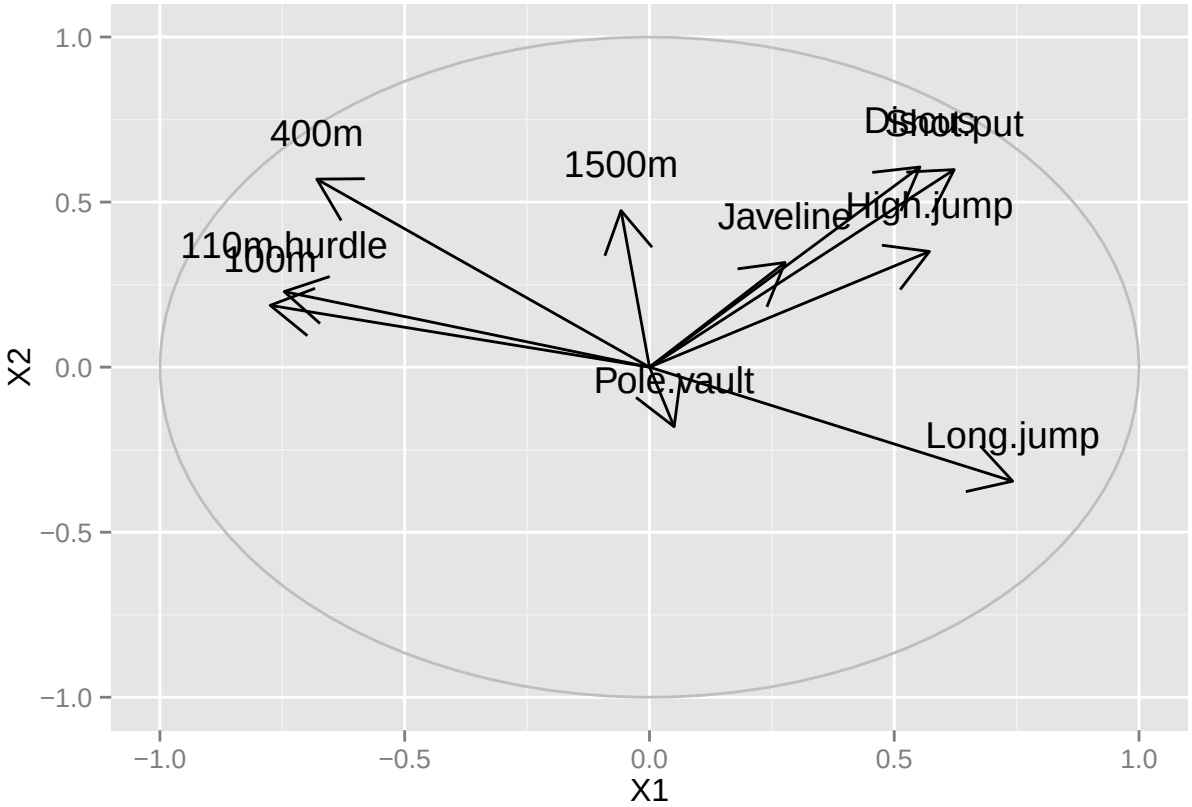
```
PCADecathlonR$var$coord
```

```
##                  Dim.1      Dim.2       Dim.3       Dim.4       Dim.5
## 100m        -0.77471983  0.1871420 -0.18440714 -0.03781826  0.30219639
```

```
## Long.jump    0.74189974 -0.3454213  0.18221105  0.10178564  0.03667805
## Shot.put     0.62250255  0.5983033 -0.02337844  0.19059161  0.11115082
## High.jump    0.57194530  0.3502936 -0.25951193 -0.13559420  0.55543957
## 400m        -0.67960994  0.5694378  0.13146970  0.02930198 -0.08769157
## 110m.hurdle -0.74624532  0.2287933 -0.09263738  0.29083103  0.16432095
## Discus       0.55246652  0.6063134  0.04295225 -0.25967143 -0.10482712
## Pole.vault   0.05034151 -0.1803569  0.69175665  0.55153397  0.32995932
## Javeline     0.27711085  0.3169891 -0.38965541  0.71227728 -0.30512892
## 1500m       -0.05807706  0.4742238  0.78214280 -0.16108904 -0.15356189
```

Note that because, we use a *rescaled* version of we know that all the coordinates have a variance 1, we can plot them in the plane spanned by the two first vectors and verify that they remains within the unit circle. The variables that are well represented within this plane would be close to reach the circle whereas the badly represented ones will remain close to 0.

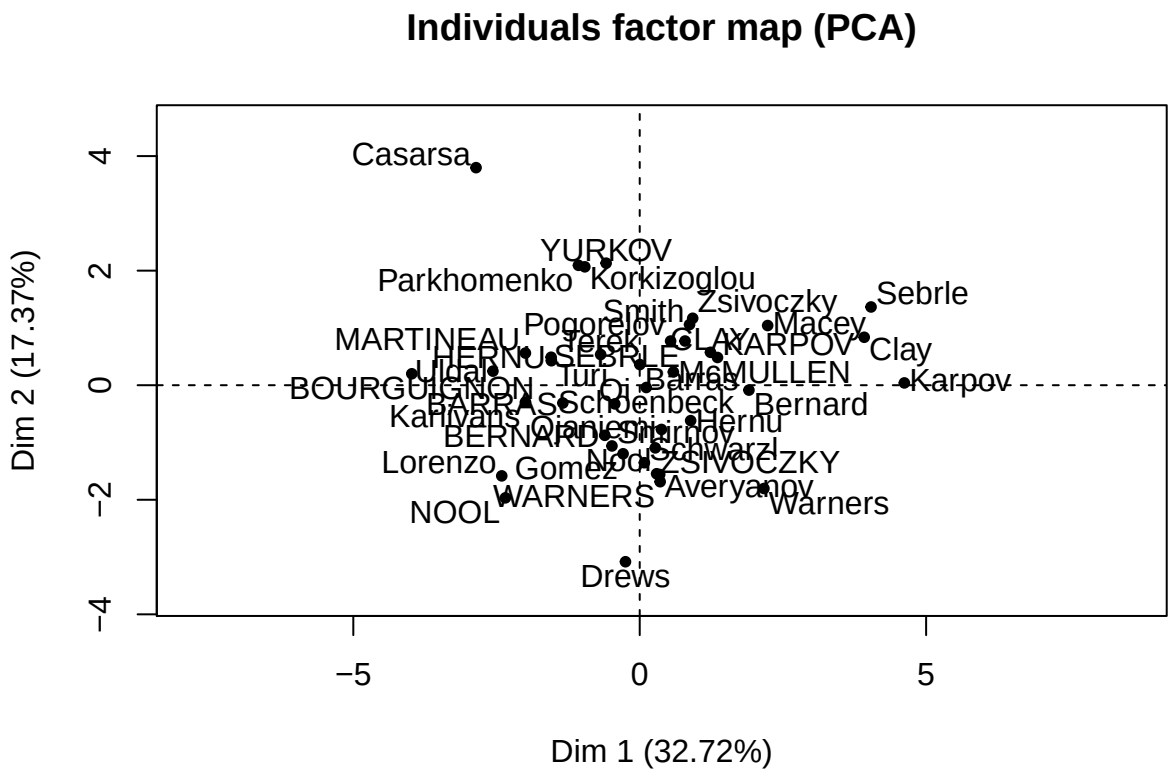
```
circle <- function(center = c(0, 0), npoints = 100) {
    r = 1
    tt = seq(0, 2 * pi, length = npoints)
    xx = center[1] + r * cos(tt)
    yy = center[1] + r * sin(tt)
    return(data.frame(X1 = xx, X2 = yy))
}
corcir = circle(c(0, 0), npoints = 100)

ggplot(data.frame(X1 = PCADecathlonR$var$coord[,1], X2 = PCADecathlonR$var$coord[,2]))+
  geom_path(data = corcir, aes(x = X1, y = X2), color = "gray") +
  geom_segment(aes(xend = X1, yend = X2), x = 0 , y = 0, arrow = grid::arrow()) +
  geom_text(aes(x = X1, y = X2), label = names(decathlonR[1:10]), vjust = -1.25)
```

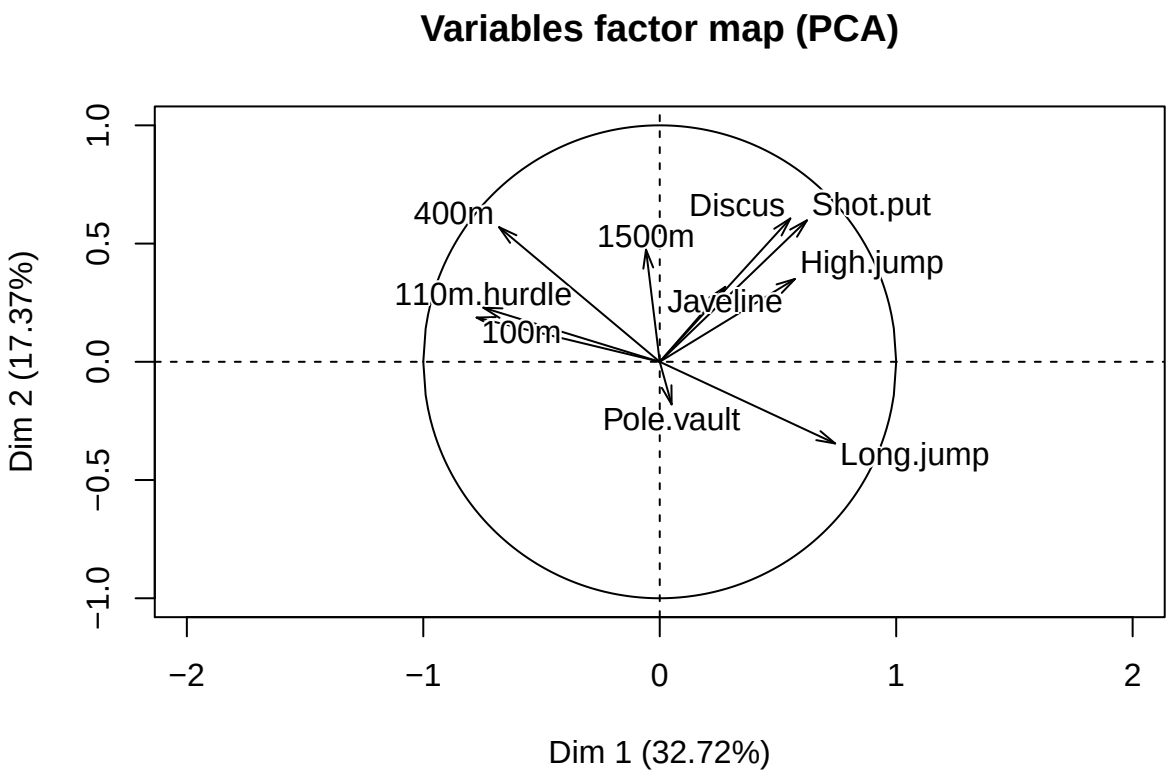


Which variables are well captured? Can you interpret the new axes (the horizontal and the vertical ones)? Can you explain why the long jump appears to be the opposite of the 100m?

10. Now, we may use PCA without the `graph = FALSE` option... and be able to understand the graphs.

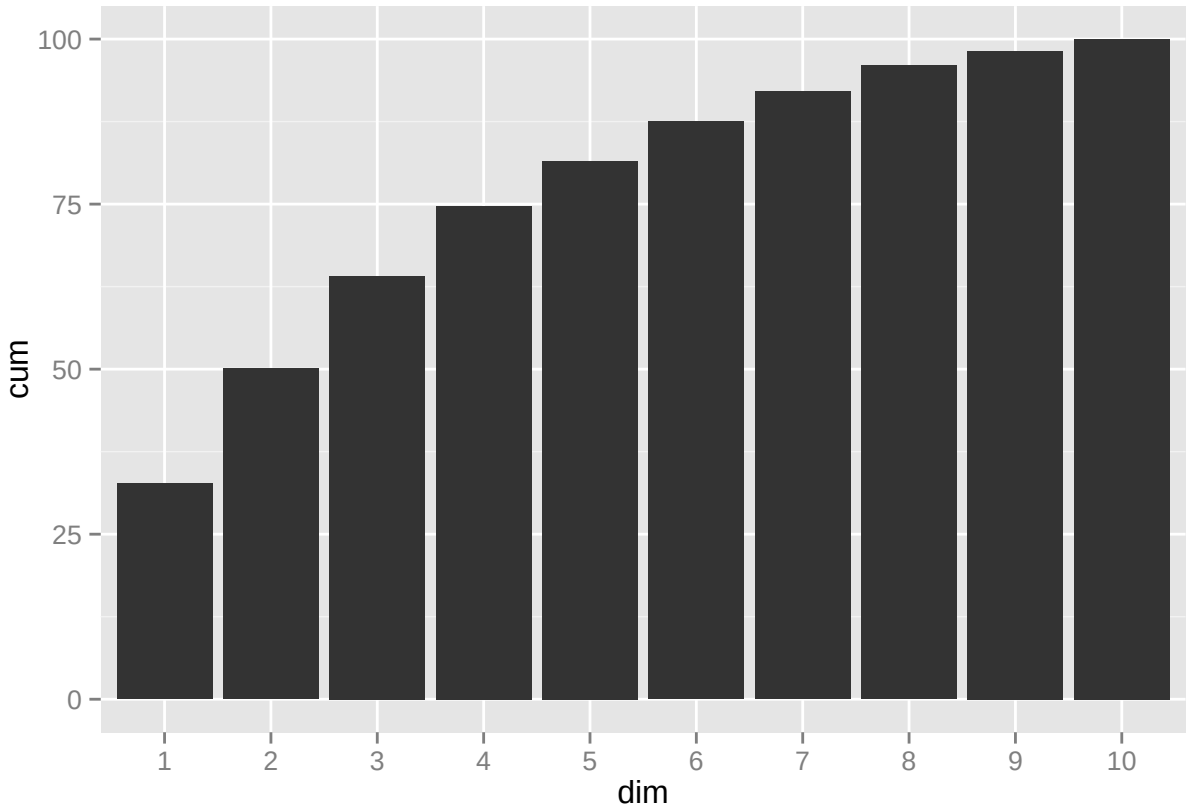
```
PCADecathlonR <- PCA(decathlonR[1:10])
```





11. We may check whether we need more dimensions by looking at the eigenvalues or rather the cumulative percentage of variance.

```
ggplot(data = data.frame(dim = factor(1:length(PCADecathlonR$eig$`cumulative percentage of variance`)),
                         cum = PCADecathlonR$eig$`cumulative percentage of variance`), aes(x = dim, y =
  geom_bar(stat = 'identity')
```



Do you think 2 dimensions is enough here?

Multiple Factor Analysis

We consider an extension of the PCA in which one can use qualitative variable and also group variables.

Data

We will use a dataset made of a sample of 21 red wine from the Loire region. Several attributes are recorded

- 2 qualitative ones:
- **Label**: *Saumur*, *Chinon*, *Bourgueil*
- **Soil**: *Reference*, *Env1*, *Env2*, *Env4*
- 29 quantitative ones classified by groups:
- Odor (5 variables)
- Visual (3 variables)
- Odor after shaking (10 variables)
- Taste (9 variables)
- Ocerall (2 variables)

```
data(wine)
colnames(wine)
```

```
##  [1] "Label"                          "Soil"
##  [3] "Odor.Intensity.before.shaking" "Aroma.quality.before.shaking"
```

```
##  [5] "Fruity.before.shaking"         "Flower.before.shaking"
##  [7] "Spice.before.shaking"          "Visual.intensity"
##  [9] "Nuance"                        "Surface.feeling"
## [11] "Odor.Intensity"                "Quality.of.odour"
## [13] "Fruity"                        "Flower"
## [15] "Spice"                         "Plante"
## [17] "Phenolic"                      "Aroma.intensity"
## [19] "Aroma.persistency"             "Aroma.quality"
## [21] "Attack.intensity"              "Acidity"
## [23] "Astringency"                   "Alcohol"
## [25] "Balance"                       "Smooth"
## [27] "Bitterness"                    "Intensity"
## [29] "Harmony"                       "Overall.quality"
## [31] "Typical"
```

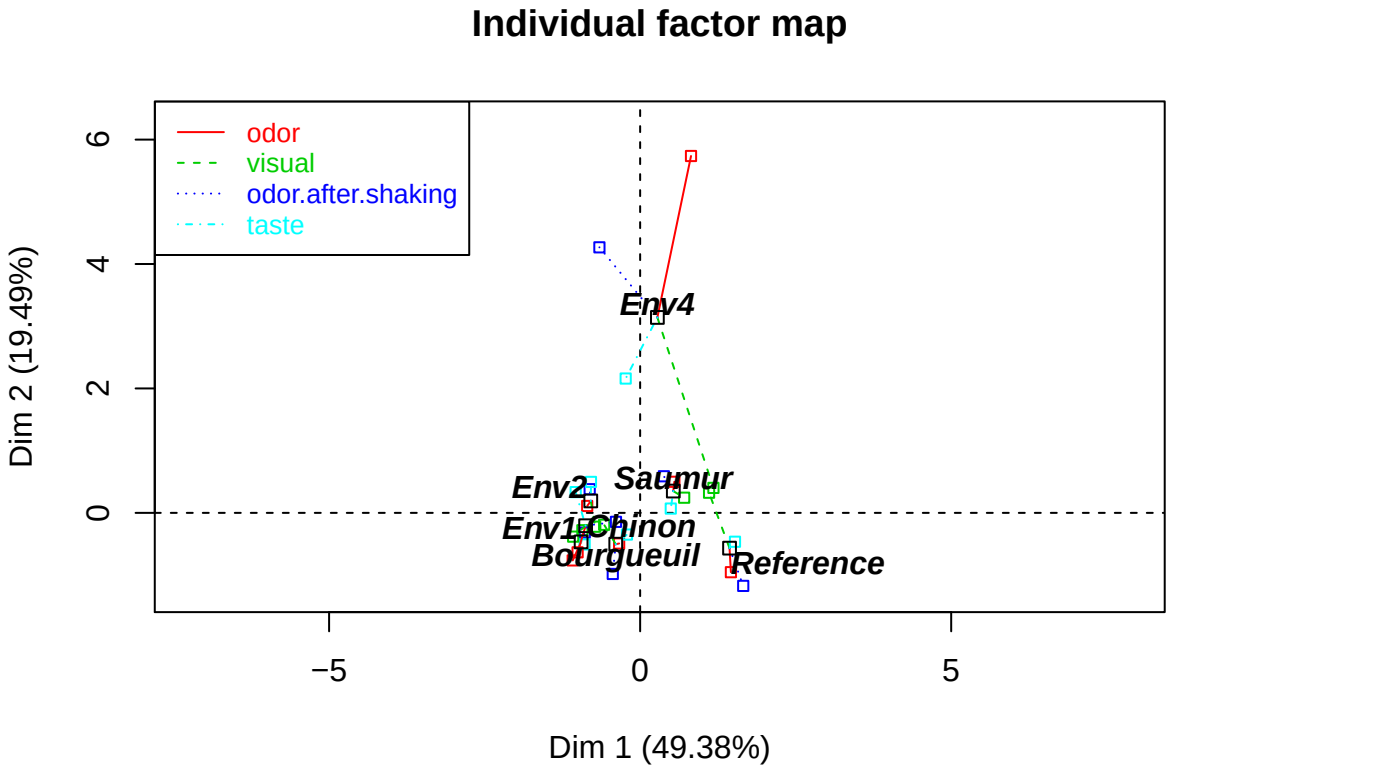
```
glimpse(wine)
```

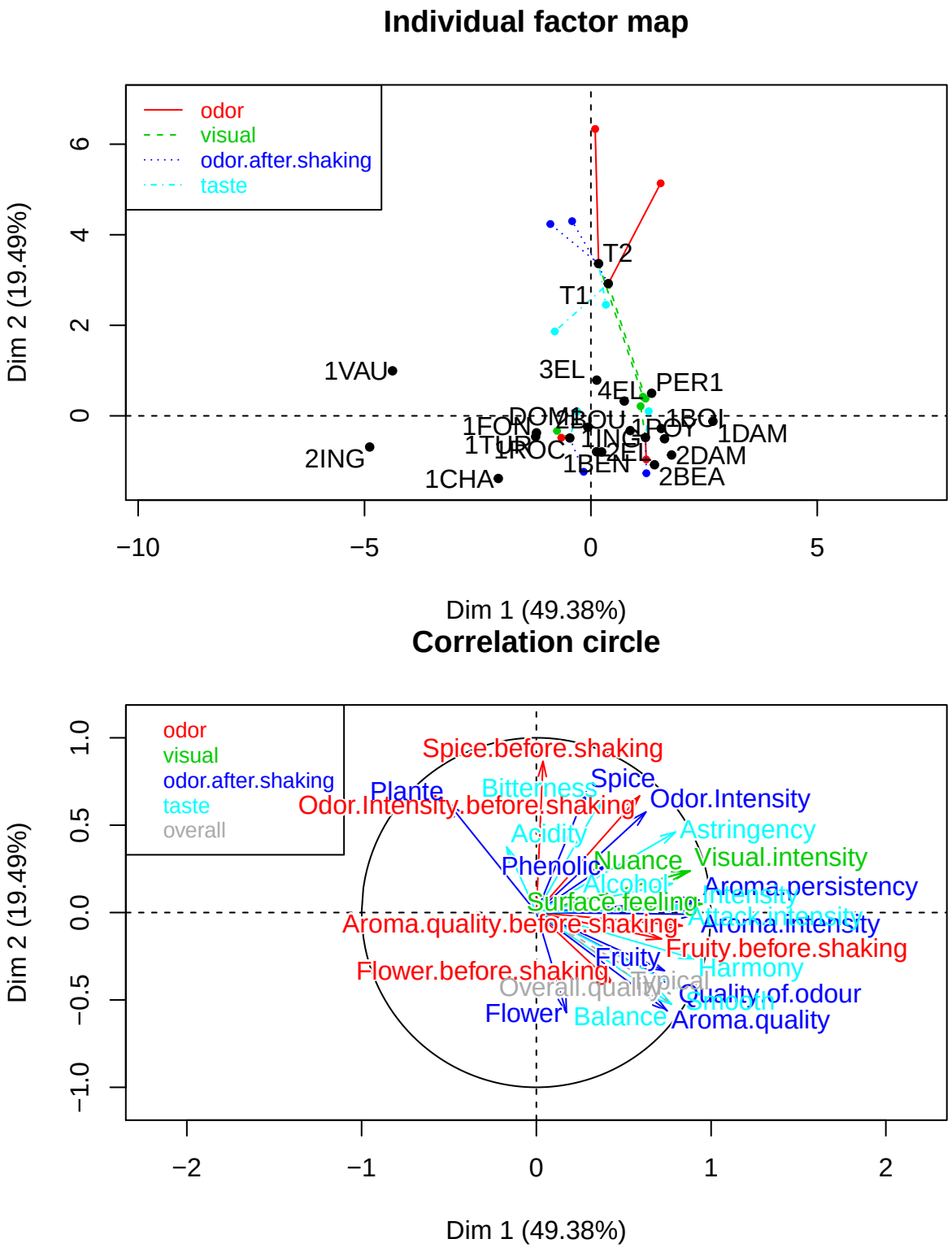
```
## Observations: 21
## Variables: 31
## $ Label                          (fctr) Saumur, Saumur, Bourgueuil, Chi...
## $ Soil                           (fctr) Env1, Env1, Env1, Env2, Referen...
## $ Odor.Intensity.before.shaking (dbl) 3.074, 2.964, 2.857, 2.808, 3.60...
## $ Aroma.quality.before.shaking  (dbl) 3.000, 2.821, 2.929, 2.593, 3.42...
## $ Fruity.before.shaking          (dbl) 2.714, 2.375, 2.560, 2.417, 3.15...
## $ Flower.before.shaking          (dbl) 2.280, 2.280, 1.960, 1.913, 2.15...
## $ Spice.before.shaking           (dbl) 1.960, 1.680, 2.077, 2.160, 2.04...
## $ Visual.intensity               (dbl) 4.321, 3.222, 3.536, 2.893, 4.39...
## $ Nuance                         (dbl) 4.000, 3.000, 3.393, 2.786, 4.03...
## $ Surface.feeling                (dbl) 3.269, 2.808, 3.000, 2.538, 3.38...
## $ Odor.Intensity                 (dbl) 3.407, 3.370, 3.250, 3.160, 3.53...
## $ Quality.of.odour               (dbl) 3.308, 3.000, 2.929, 2.880, 3.36...
## $ Fruity                         (dbl) 2.885, 2.560, 2.769, 2.391, 3.16...
## $ Flower                         (dbl) 2.320, 2.440, 2.192, 2.083, 2.23...
## $ Spice                          (dbl) 1.840, 1.739, 2.250, 2.167, 2.14...
## $ Plante                         (dbl) 2.000, 2.000, 1.750, 2.304, 1.76...
## $ Phenolic                       (dbl) 1.650, 1.381, 1.250, 1.476, 1.60...
## $ Aroma.intensity                (dbl) 3.259, 2.962, 3.077, 2.542, 3.61...
## $ Aroma.persistency              (dbl) 2.963, 2.808, 2.800, 2.583, 3.29...
## $ Aroma.quality                  (dbl) 3.200, 2.926, 3.077, 2.478, 3.46...
## $ Attack.intensity               (dbl) 2.963, 3.036, 3.222, 2.704, 3.46...
## $ Acidity                        (dbl) 2.107, 2.107, 2.179, 3.179, 2.57...
## $ Astringency                    (dbl) 2.429, 2.179, 2.250, 2.185, 2.53...
## $ Alcohol                        (dbl) 2.500, 2.654, 2.643, 2.500, 2.78...
## $ Balance                        (dbl) 3.250, 2.926, 3.321, 2.333, 3.46...
## $ Smooth                         (dbl) 2.731, 2.500, 2.679, 1.680, 3.03...
## $ Bitterness                     (dbl) 1.926, 1.926, 2.000, 1.963, 2.07...
## $ Intensity                      (dbl) 2.857, 2.893, 3.074, 2.462, 3.64...
## $ Harmony                        (dbl) 3.143, 2.964, 3.143, 2.038, 3.64...
## $ Overall.quality                (dbl) 3.393, 3.214, 3.536, 2.464, 3.74...
## $ Typical                        (dbl) 3.250, 3.036, 3.179, 2.250, 3.44...
```

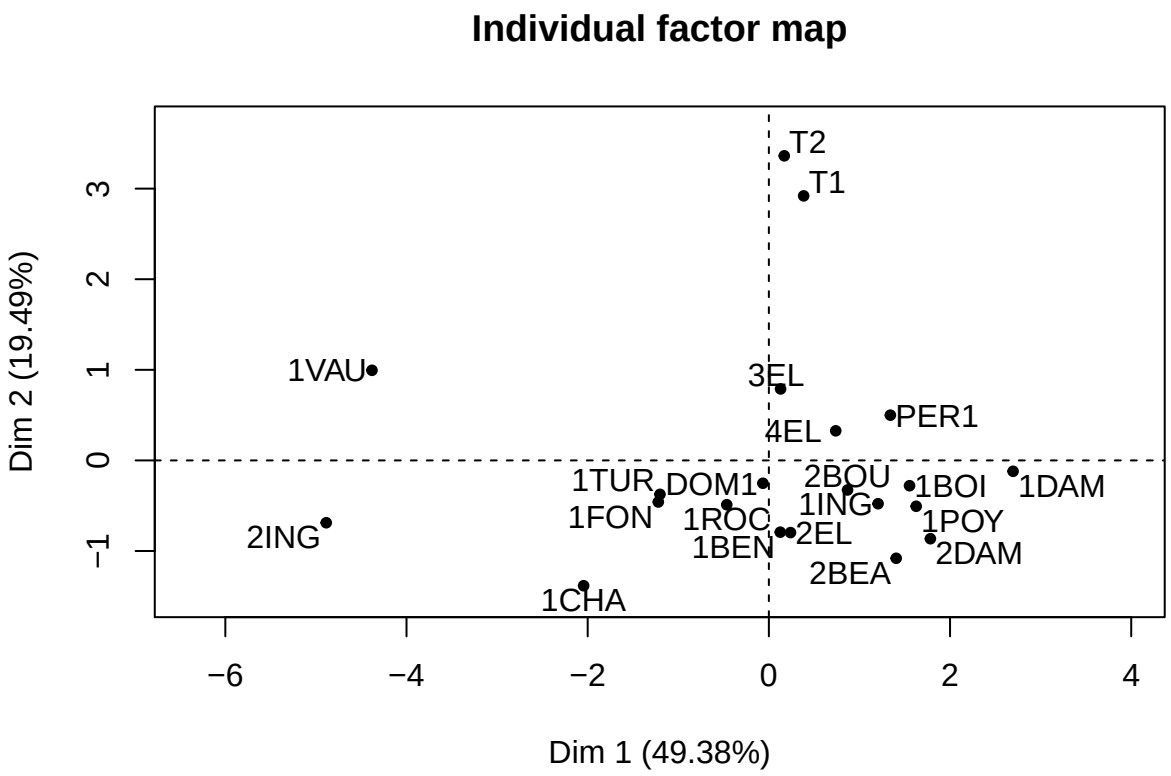
MFA

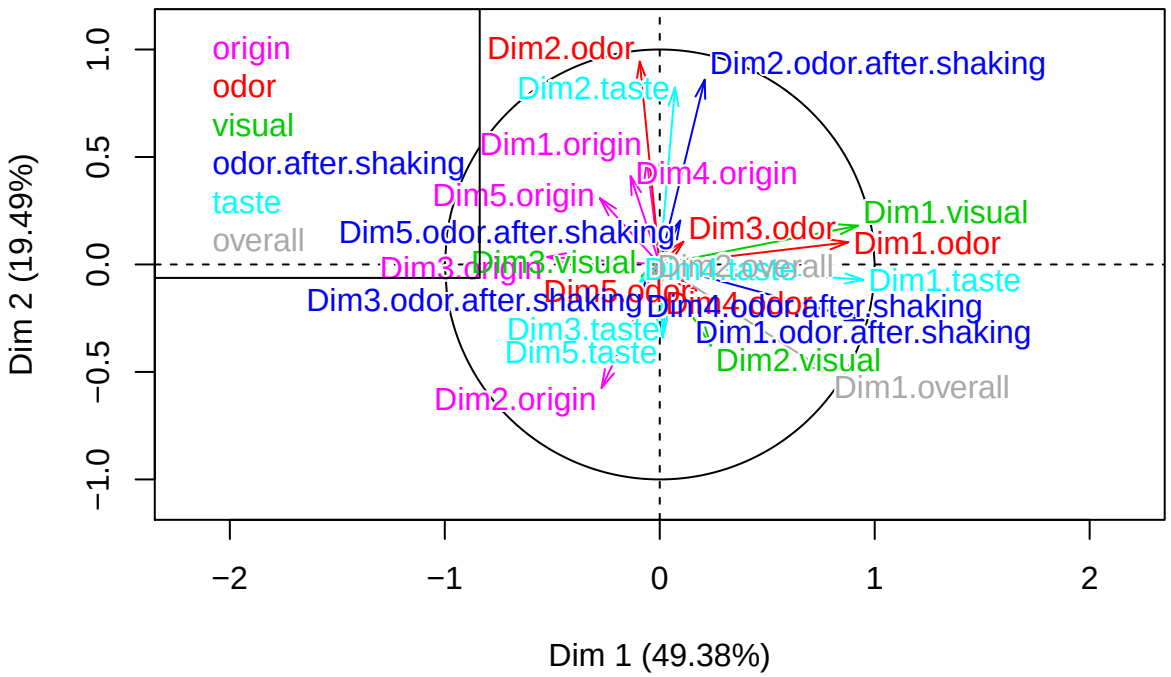
10. Can you guess what MFA is doing?

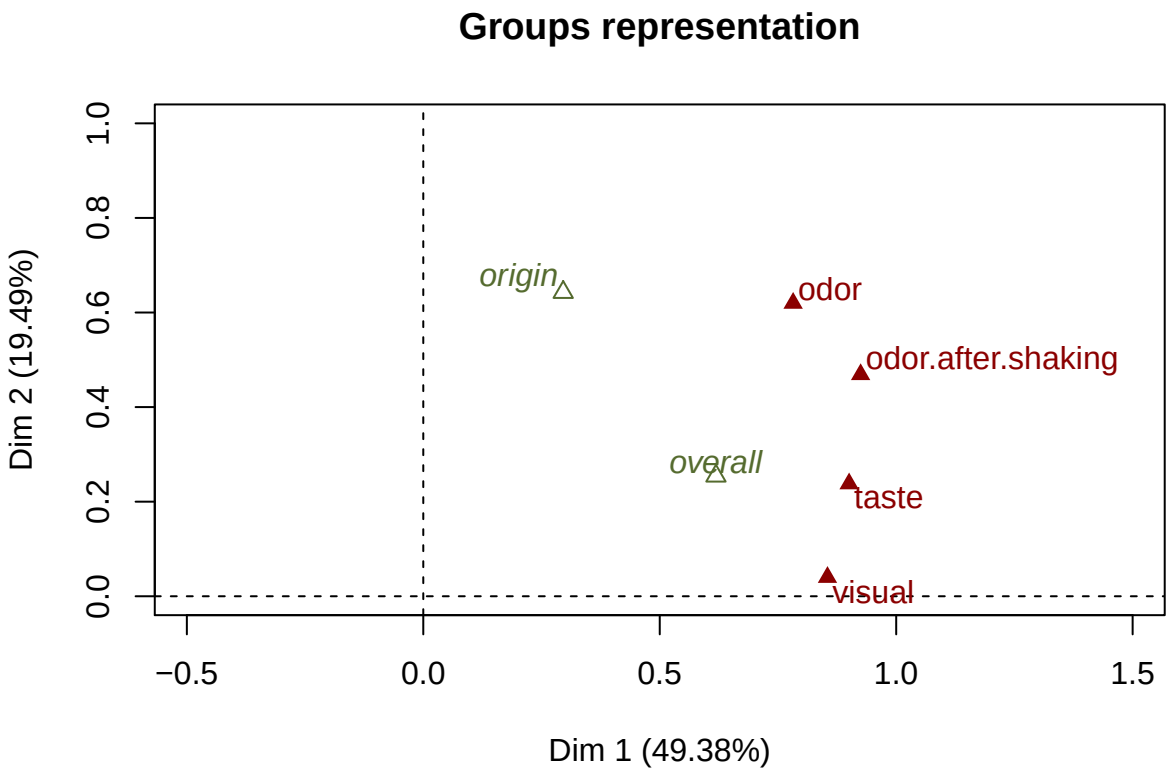
```
res.mfa = MFA(wine, group = c(2, 5, 3, 10, 9, 2), type = c("n", rep("s",5)), ncp = 5,
              name.group = c("origin", "odor", "visual", "odor.after.shaking", "taste", "overall"),
              num.group.sup=c(1,6))
```




Individual factor map
odor
visual
odor.after.shaking
taste
T2
T1
1VAU
3EL
4EL
PER1
1DAM
2DAM
2BEA
2ING
1CHA
Dim 2 (19.49%)
Dim 1 (49.38%)
Correlation circle
odor
visual
odor.after.shaking
taste
overall
Spice.before.shaking
Spice
Plante
Bitterness
Odor.Intensity.before.shaking
Odor.Intensity
Acidity
Astringency
Phenolic
Nuance
Visual.intensity
Alcohol
Aroma.persistency
Surface.feeling
Attack.intensity
Aroma.quality.before.shaking
Aroma.Intensity
Fruity.before.shaking
Fruity
Harmony
Flower.before.shaking
Overall.quality
Typical
Quality.of.odour
Flower
Balance
Aroma.quality
Dim 2 (19.49%)
Dim 1 (49.38%)

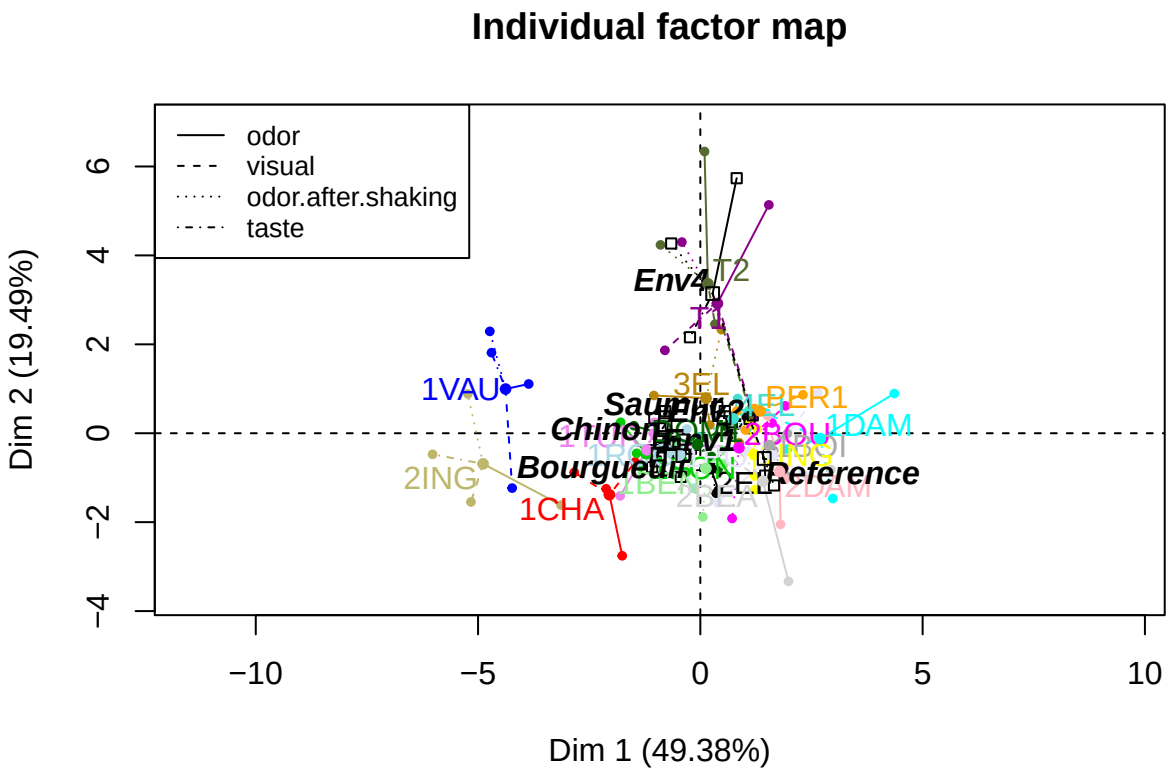

Individual factor map
T2
T1
1VAU
3EL
PER1
4EL
1TUR
DOM1
2BOU
1BOI
1DAM
1FON
1ROC
1ING
1POY
2ING
1BEN
2EL
2DAM
2BEA
1CHA
Dim 2 (19.49%)
Dim 1 (49.38%)


Partial axes
origin
odor
visual
odor.after.shaking
taste
overall
Dim2.odor
Dim2.odor.after.shaking
Dim2.taste
Dim1.origin
Dim4.origin
Dim5.origin
Dim1.visual
Dim5.odor.after.shaking
Dim3.odor
Dim1.odor
Dim3.origin
Dim3.visual
Dim2.overall
Dim1.taste
Dim5.odor
Dim3.odor.after.shaking
Dim4.odor.after.shaking
Dim3.taste
Dim1.odor.after.shaking
Dim5.taste
Dim2.visual
Dim2.origin
Dim1.overall
Dim 2 (19.49%)
Dim 1 (49.38%)



11. The *strange* lines in the individual factor map is an attempt to view the contribution of each group. It corresponds to the barycenter of the contributions of the given group. By default, only the two most dispersed and concentrated groups are plotted. We can ask MFA to plot them all.

```
plot(res.mfa, choix = "ind", partial = "all")
```

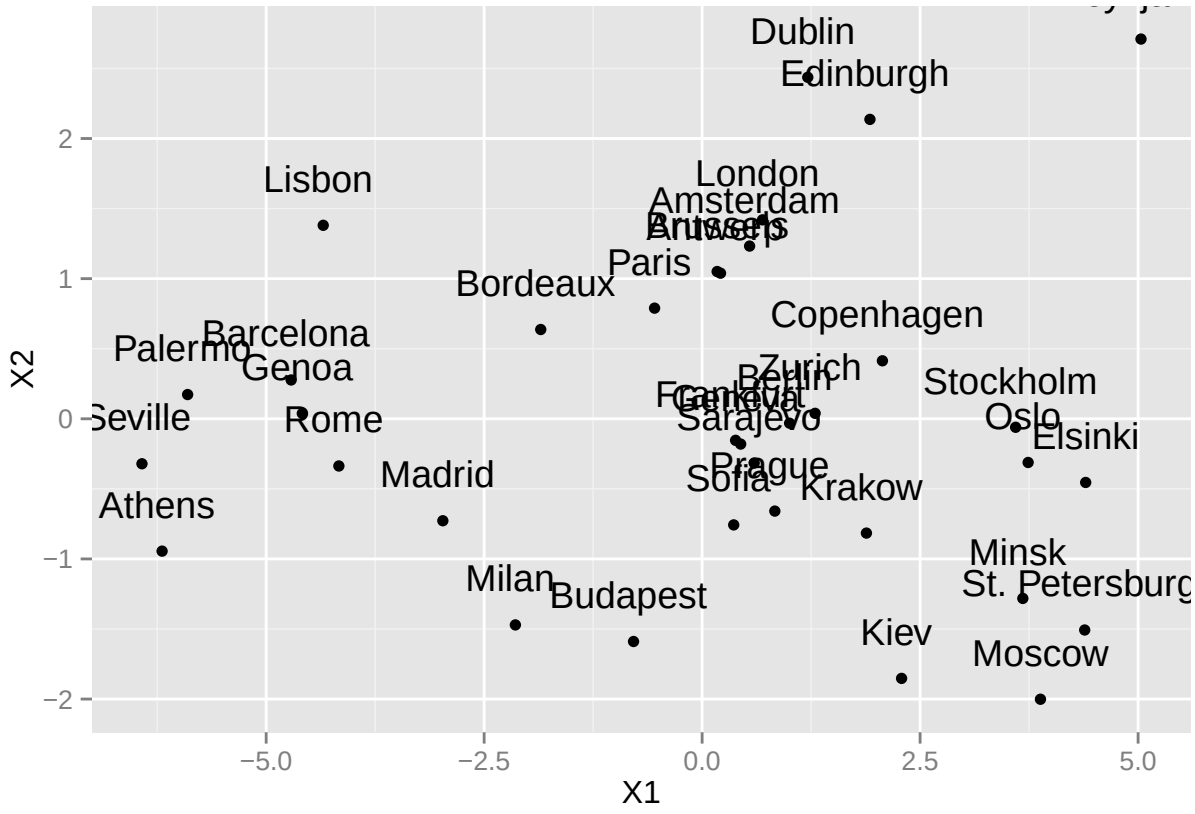


MDS

We will apply the MultiDimensional Scaling available in **R** `cmdscale` to an interesting dataset of the monthly average temperature for 35 cities in Europe. We will try to apply the MDS methodology to visualize those cities in a plane according to their temperature profiles.

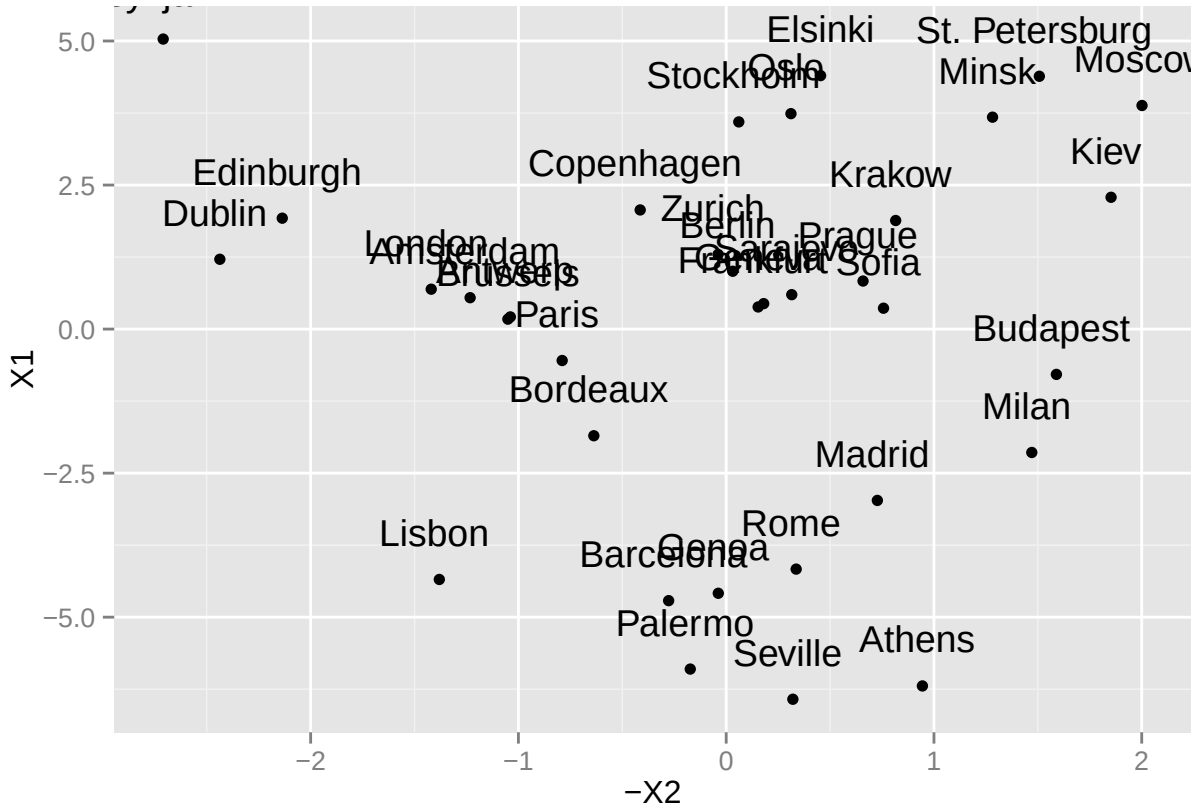
12. Read the data and apply the MDS algorithm.

```
temperature <- read.csv("temperature.csv", row.names = 1)
temperature[1:12] <- scale(temperature[1:12])

DistTemperature <- dist(temperature[1:12])
TemperatureMDS <- cmdscale(DistTemperature)

ggplot(data = data.frame(X1 = TemperatureMDS[,1], X2 = TemperatureMDS[,2]), aes(x = X1, y = X2)) +
  geom_point() + geom_text(label = row.names(temperature), vjust = -1.25)
```



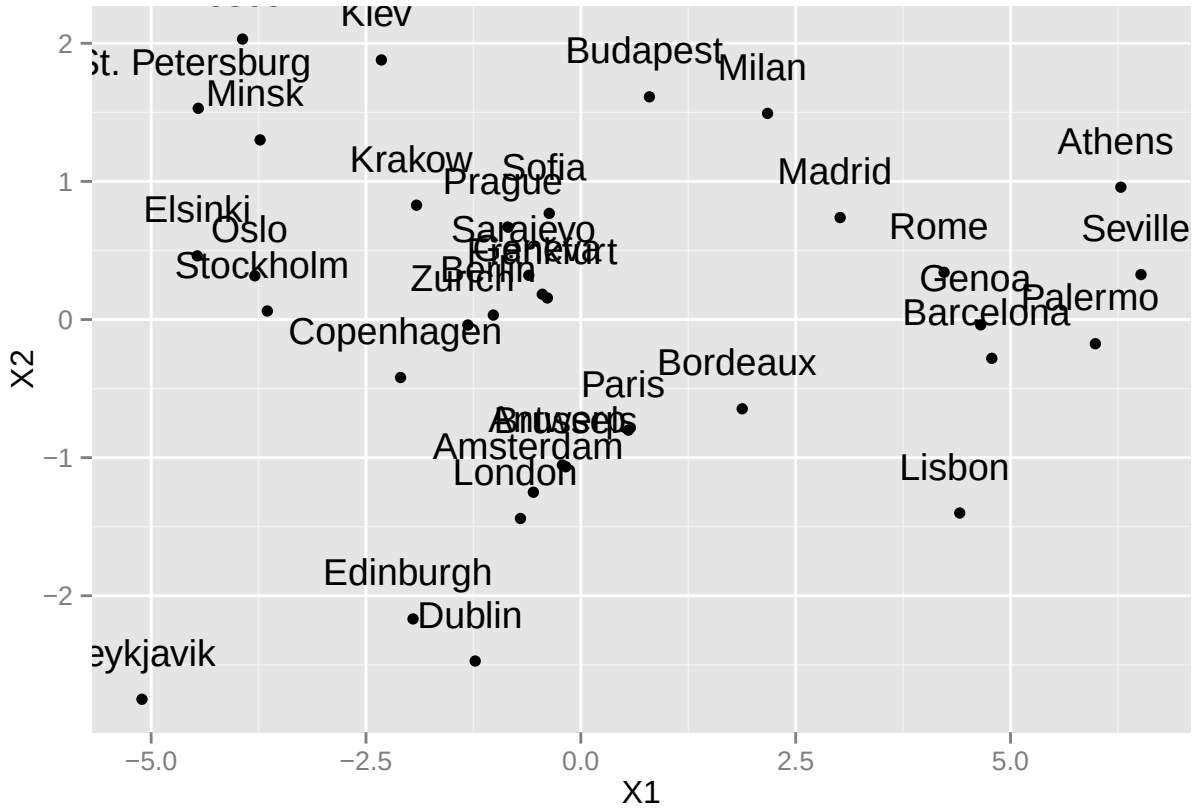
13. Can you obtain a better fit with the geography by swapping the axis?

```
ggplot(data = data.frame(X1 = TemperatureMDS[,1], X2 = TemperatureMDS[,2]), aes(x = -X2, y = X1)) +
  geom_point() + geom_text(label = row.names(temperature), vjust = -1.25)
```

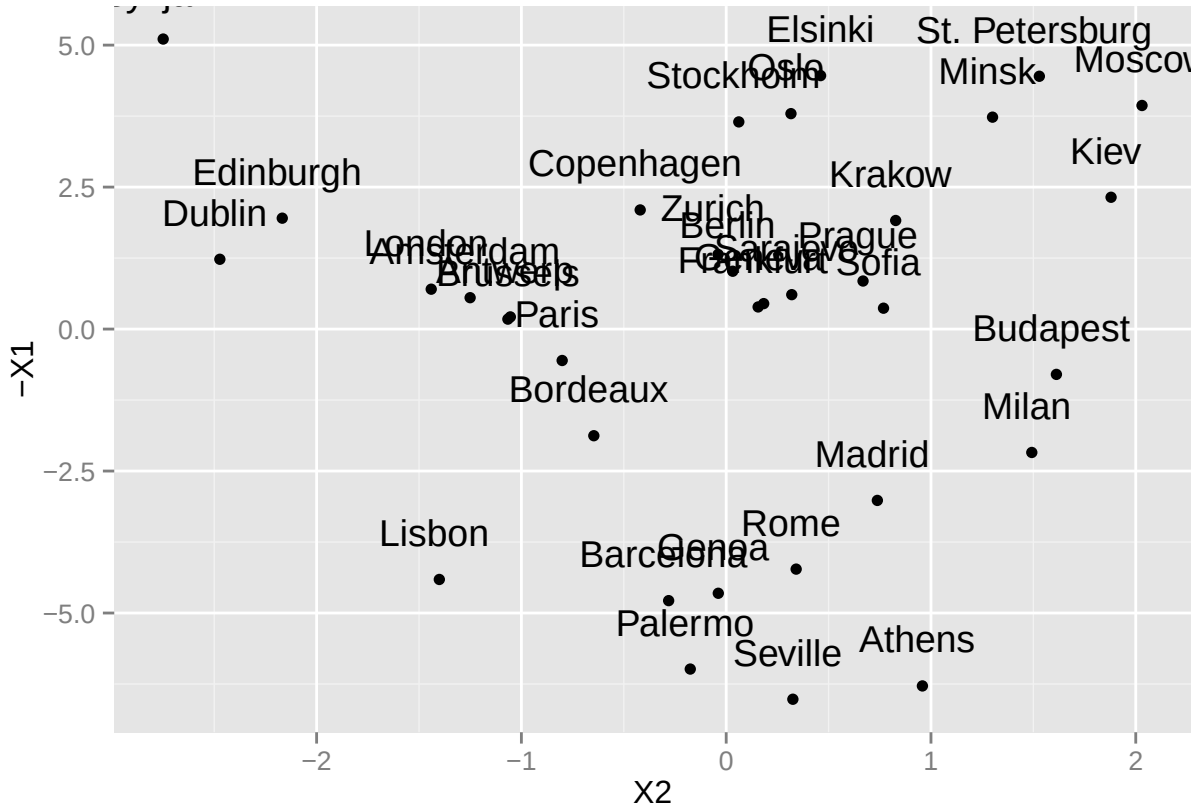


14. Compare with the results obtain with the PCA. Is this a suprise?

```
TemperaturePCA <- PCA(temperature[1:12], graph = FALSE)
ggplot(data = data.frame(X1 = TemperaturePCA$ind$coord[,1], X2 = TemperaturePCA$ind$coord[,2]), aes(x =
  geom_point() + geom_text(label = row.names(temperature), vjust = -1.25)
```




```
ggplot(data = data.frame(X1 = TemperaturePCA$ind$coord[,1], X2 = TemperaturePCA$ind$coord[,2]), aes(x =
  geom_point() + geom_text(label = row.names(temperature), vjust = -1.25)
```



15. Use the **igraph** package to perform a graph Laplacian based dimension reduction.