PCA

J. Di Iorio, F. Chiaromonte

2/21/2021

Principal Component Analysis (PCA) is a Dimensional Reduction technique for unsupervised data. It transforms data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. Working in high-dimensional spaces can be undesirable for many reasons so it could be useful to project the features to a space of fewer dimensions.

Libraries

We are going to use **cluster**,**factoextra** and **NbClust**

```
library(mvtnorm) #for the toy example
library(NbClust)
library(factoextra)
```

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

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

Data

Today we are going to use two dataset: **decathlon2** data set available in the **decathlon2** and the **food** data set in the **Food.txt** file.

The **decathlon2** data set consists of 27 observations (athletes) on the following 13 variables (performance).

```
library(factoextra)
help(decathlon2)
head(decathlon2)
```

```
##           X100m Long.jump Shot.put High.jump X400m X110m.hurdle Discus
## SEBRLE    11.04      7.58    14.83      2.07 49.81        14.69  43.75
## CLAY      10.76      7.40    14.26      1.86 49.37        14.05  50.72
## BERNARD   11.02      7.23    14.25      1.92 48.93        14.99  40.87
## YURKOV    11.34      7.09    15.19      2.10 50.42        15.31  46.26
## ZSIVOCZKY 11.13      7.30    13.48      2.01 48.62        14.17  45.67
## McMULLEN  10.83      7.31    13.76      2.13 49.91        14.38  44.41
##           Pole.vault Javeline X1500m Rank Points Competition
## SEBRLE          5.02    63.19  291.7    1   8217    Decastar
## CLAY            4.92    60.15  301.5    2   8122    Decastar
## BERNARD         5.32    62.77  280.1    4   8067    Decastar
## YURKOV          4.72    63.44  276.4    5   8036    Decastar
## ZSIVOCZKY       4.42    55.37  268.0    7   8004    Decastar
```

```
## McMULLEN        4.42    56.37   285.1    8   7995    Decastar
```

The **food** data set . . .

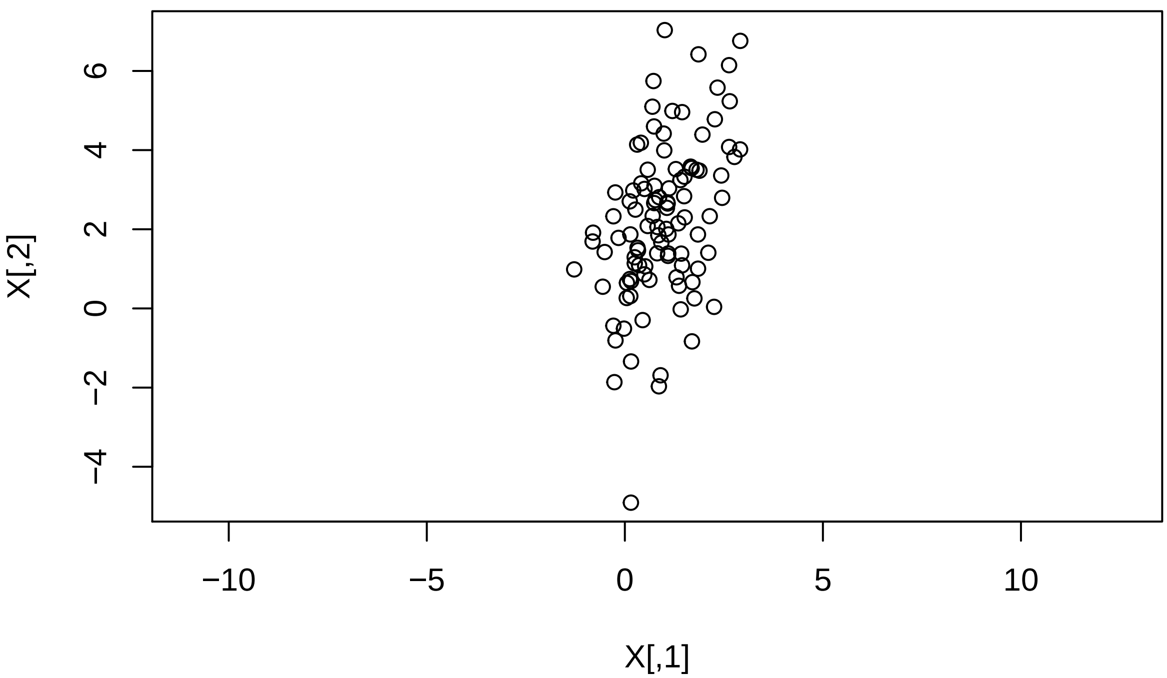
A Toy Example of Dimensional Reduction

Let us generate a bivariate data set linearly dependending from a Gaussian Distribution with higher variance on the y-axis.

```
library(mvtnorm)
mu  <-  c(1,2) # Mean vector: mean_column1 = 1 ; mean_column2 = 2
sig <-  cbind(c(1,1), c(1,4)) # Variance matrix
n   <-  100 # number of units

X <- rmvnorm(n, mu, sig) # data generation
head(X)
```

```
##            [,1]       [,2]
## [1,]  0.5767413  3.5047913
## [2,] -0.2639776 -1.8625683
## [3,]  1.3490729  2.1505055
## [4,]  0.1254003  0.7421766
## [5,]  0.0520088  0.6445841
## [6,]  0.5757847  2.0827678
```

```
plot(X, asp=1) # plot our data
```



How can we dimensionally reduce my data set? By reducing the number of features (columns) in different ways:

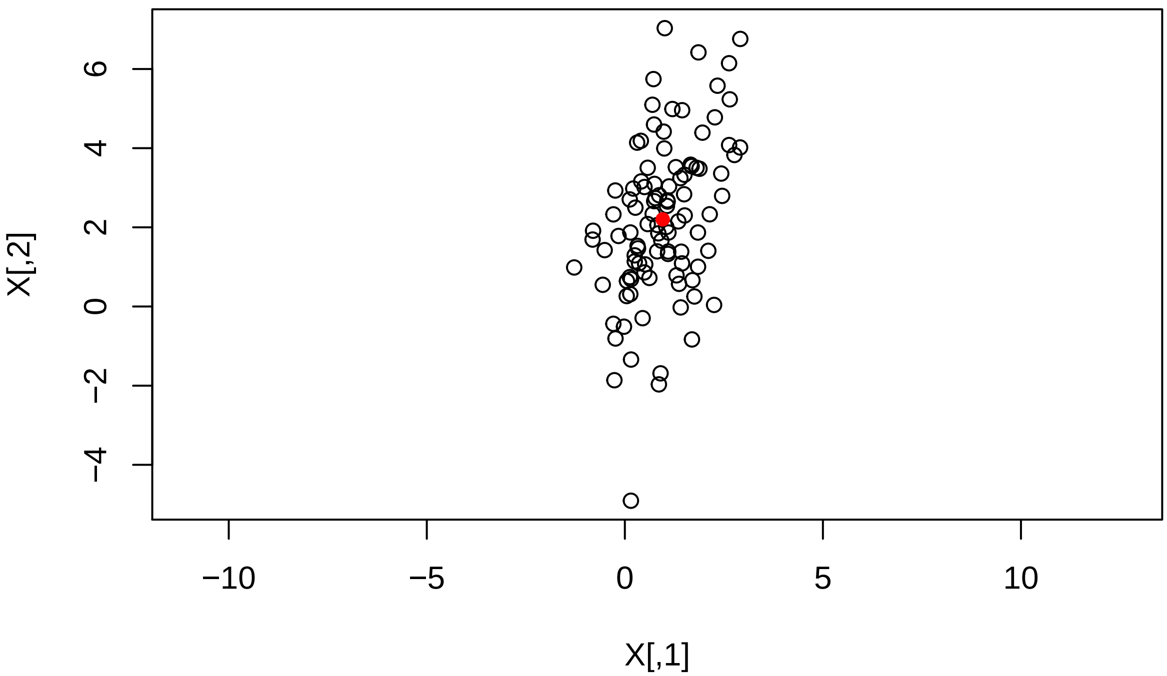
Using the sample mean

The sample mean is the easiest 0-dimensional reduction of data because it allows to reduct the data to one single point.

```r
med <- colMeans(X)
med
```

```
## [1] 0.9558386 2.2010916
```

```r
# plotting the mean
plot(X, asp=1)
points(med[1], med[2], col='red', pch=16)
```

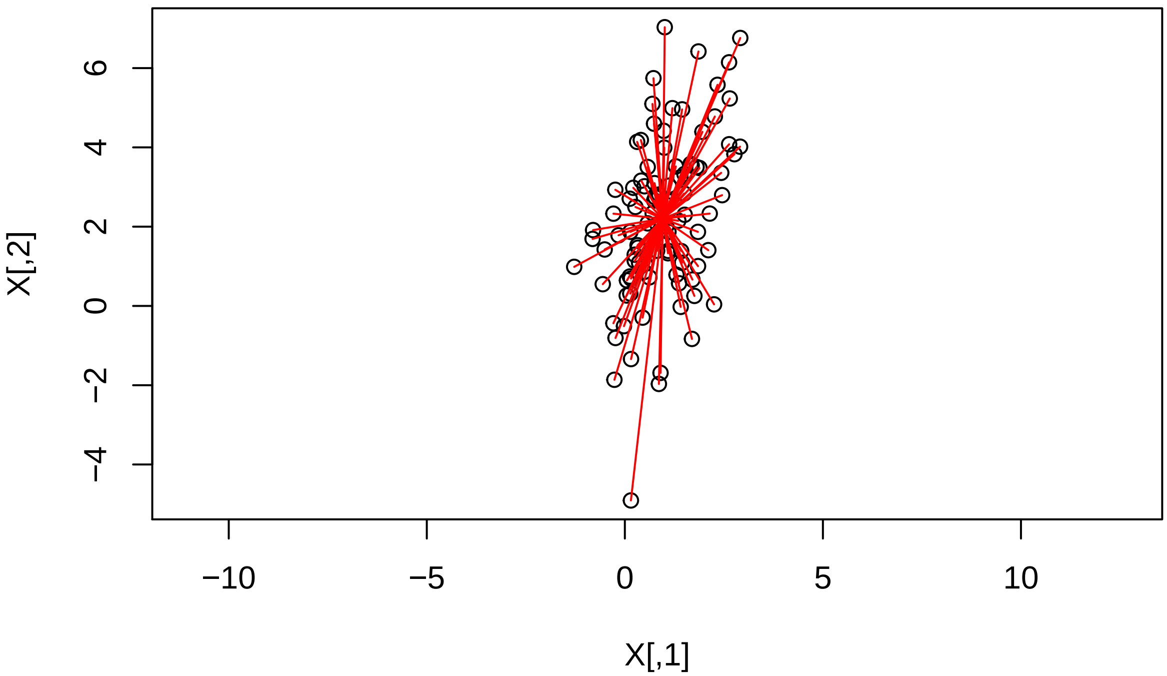


What is the Variance?

How much information I am losing in this way?

What is the error? I can represent the error in this way:

```r
# plotting the mean and the error
plot(X, asp=1)
points(med[1], med[2], col='red', pch=16)
for(i in 1:100)
  lines(rbind(X[i,], med), col='red')
```



We are collapsing our data to one point, the sample mean (also identified as PC0). The error is high.

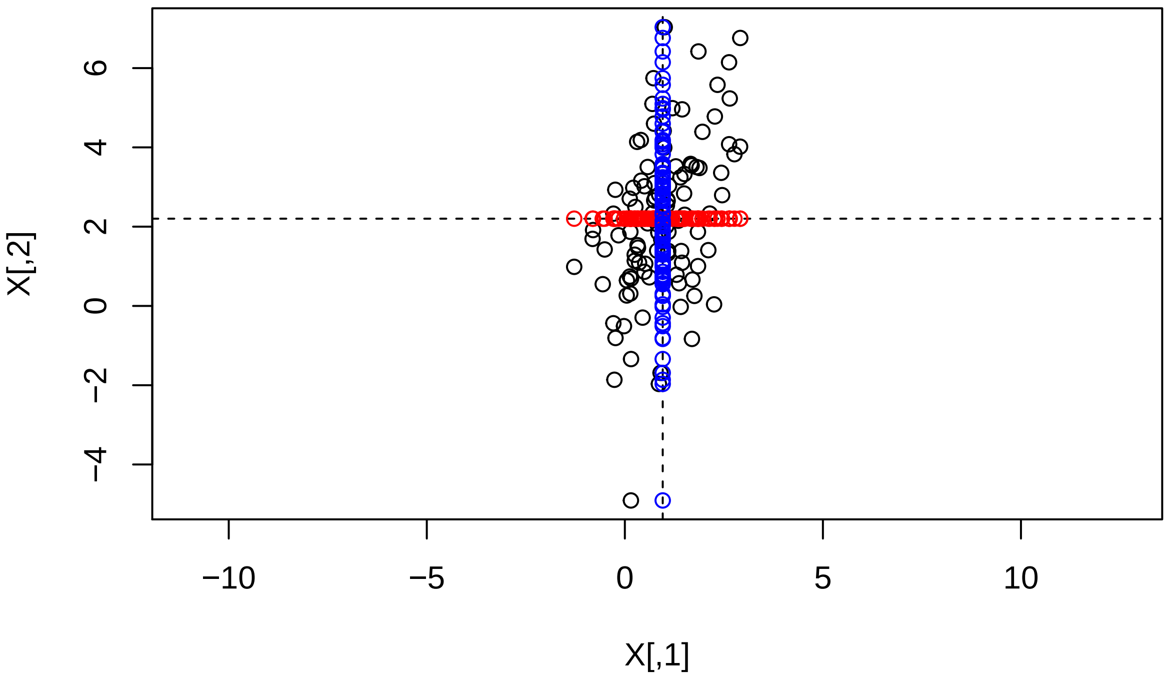
Projecting the data on a new axis

I can identify two axis from the sample mean: an *horizontal*-axis, and a *vertical*-axis.

```
# for the horizontal-axis
plot(X, asp=1)
points(med[1], med[2], col='red', pch=16)
abline(h=med[2], lty=2)
# projecting data point on the axis
points(X[,1], rep(med[2], n), col='red')
# computing the variance of the red dots
var(X[,1])
```

```
## [1] 0.8072694
```

```
# for the vertical-axis
abline(v=med[1], lty=2)
# projecting data point on the axis
points(rep(med[1], n), X[,2], col='blue')
```




```
# computing the variance of the blue dots
var(X[,2])
```

```
## [1] 4.090592
```

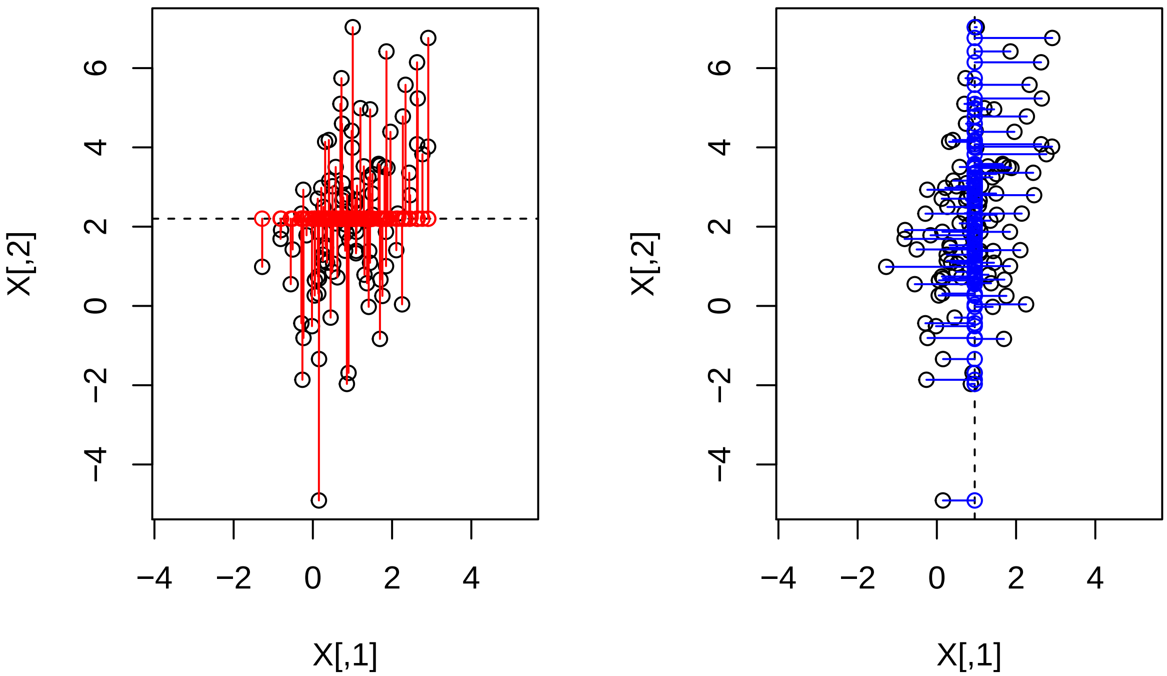
Which of the two axis maximizes the variance?

```
var(X[,2]) > var(X[,1])
```

```
## [1] TRUE
```

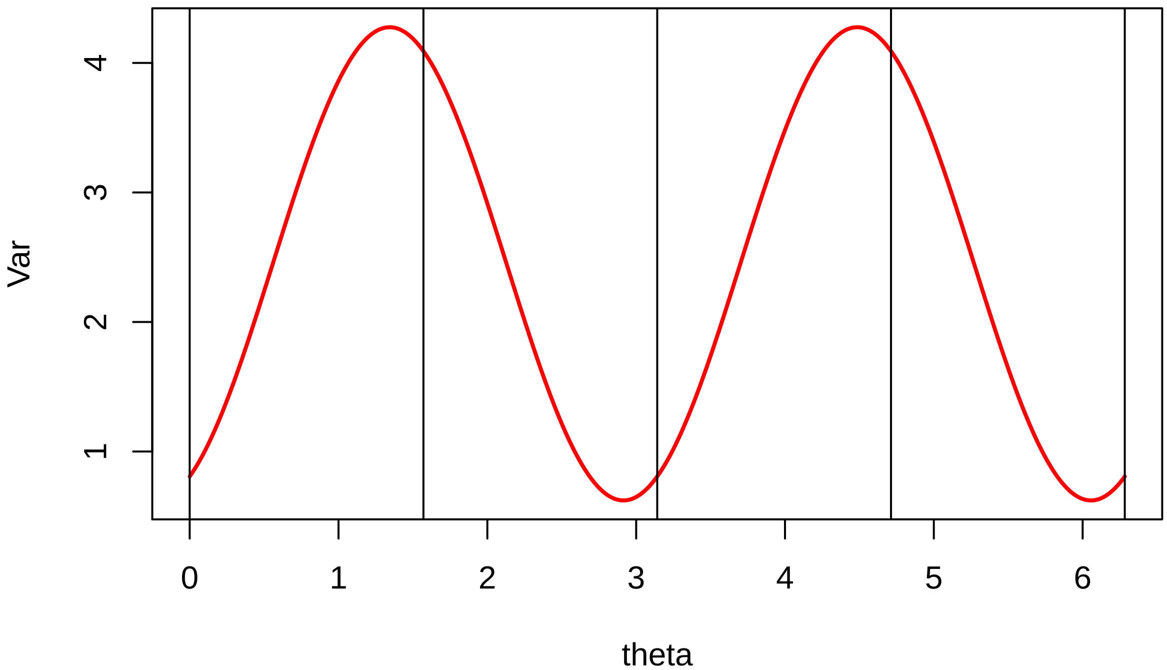
The vertical axis. Blue points are more scattered and the error (sum of the lenghts of blue segments) is lower.

```
par(mfrow=c(1,2))
# ASSE ORIZZONTALE
plot(X, asp=1)
abline(h=med[2], lty=2)
points(X[,1], rep(med[2], n), col='red')
for(i in 1:100)
  lines(rbind(X[i,], c(X[i,1], med[2])), col='red')

# ASSE VERTICALE
plot(X, asp=1)
abline(v=med[1], lty=2)
points(rep(med[1], n), X[,2], col='blue')
for(i in 1:100)
  lines(rbind(c(med[1],X[i,2]),X[i,]), col='blue')
```



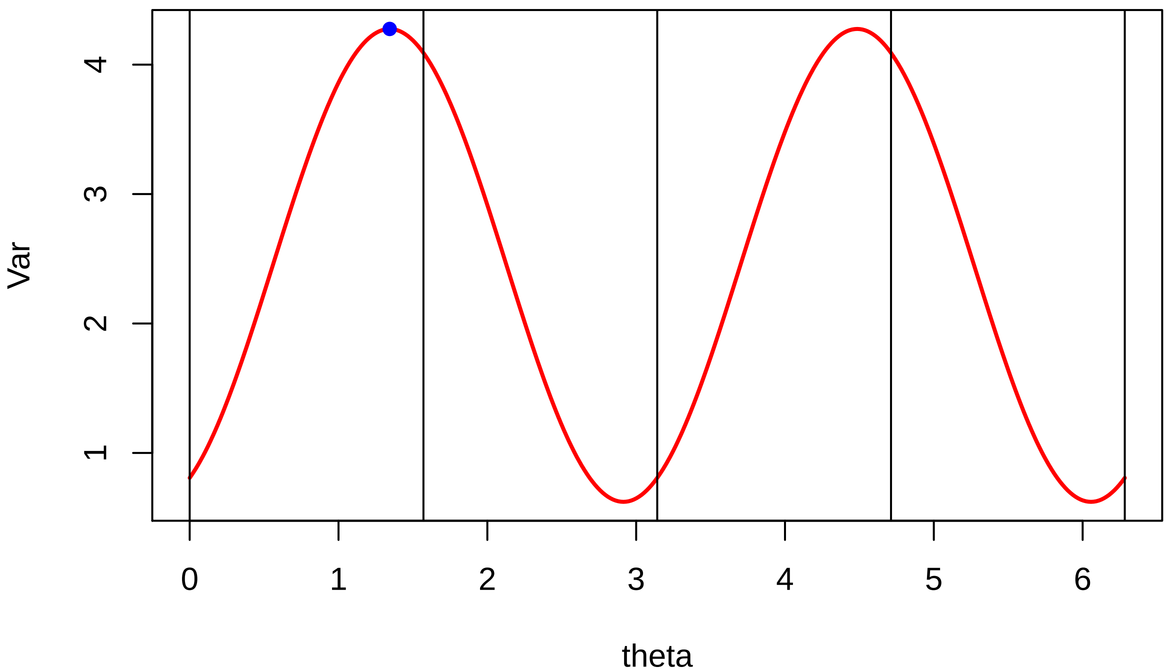
Using this strategy we can find the best axis, the axis that maximizes the variance.

```
# Compute the variance of all the possible directions
theta   <- seq(0, 2*pi, by = 2*pi/360) # angles
Var       <- NULL
for(i in 1:length(theta))
{
  a   <- c(cos(theta[i]), sin(theta[i])) # directional vector
  v   <- cov(X %*% a) # projecting points
  Var <- c(Var, v) # computing variance
}

# plotting the Variance for each direction/angle
plot(theta, Var, type = 'l', col='red', lwd = 2)
abline(v=c(0, pi/2, pi, 3/2*pi, 2*pi)) # fundamental angles
```



The maximum of the variance is identified by the maximun of the function.

```
max.var   <- max(Var) # maximum variance
max.theta <- theta[which.max(Var)] # theta angle with maximum variance

# plotting the Variance for each direction/angle
plot(theta, Var, type = 'l', col='red', lwd = 2)
abline(v=c(0, pi/2, pi, 3/2*pi, 2*pi)) # fundamental angles
points(max.theta, max.var, pch=16, col='blue')
```



We just found the first principal component (PC1) as the axis maximizing the variance.

PCA on decathlon2 data

We are going to perform PCA to the first 10 columns of the **decathlon2** data set (the ones about athletes performances).

```
library("factoextra")
data(decathlon2)

# Considering the first the columns
decathlon2<- decathlon2[, 1:10]
head(decathlon2)
```

```
##           X100m Long.jump Shot.put High.jump X400m X110m.hurdle Discus
## SEBRLE    11.04      7.58    14.83      2.07 49.81        14.69  43.75
## CLAY      10.76      7.40    14.26      1.86 49.37        14.05  50.72
## BERNARD   11.02      7.23    14.25      1.92 48.93        14.99  40.87
## YURKOV    11.34      7.09    15.19      2.10 50.42        15.31  46.26
## ZSIVOCZKY 11.13      7.30    13.48      2.01 48.62        14.17  45.67
## McMULLEN  10.83      7.31    13.76      2.13 49.91        14.38  44.41
##           Pole.vault Javeline X1500m
## SEBRLE          5.02    63.19  291.7
## CLAY            4.92    60.15  301.5
## BERNARD         5.32    62.77  280.1
## YURKOV          4.72    63.44  276.4
## ZSIVOCZKY       4.42    55.37  268.0
## McMULLEN        4.42    56.37  285.1
```

To perform PCA we can use the function **prcomp**.

```
help(prcomp)
```

We can see that the function requires different parameters. Let us focus on:

- **data:** a data frame
- **scale:** a logical value (TRUE/FALSE) indicating whether the variables should be scaled to have unit variance before the analysis takes place

Therefore, to perform PCA after scaling the data, we do:

```
res <- prcomp(decathlon2, scale = TRUE)
str(res)
```

```
## List of 5
##  $ sdev    : num [1:10] 1.936 1.321 1.232 1.016 0.786 ...
##  $ rotation: num [1:10, 1:10] -0.423 0.392 0.369 0.314 -0.332 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:10] "X100m" "Long.jump" "Shot.put" "High.jump" ...
##   .. ..$ : chr [1:10] "PC1" "PC2" "PC3" "PC4" ...
##  $ center  : Named num [1:10] 10.99 7.36 14.54 2 49.31 ...
##   ..- attr(*, "names")= chr [1:10] "X100m" "Long.jump" "Shot.put" "High.jump" ...
##  $ scale   : Named num [1:10] 0.2817 0.2944 0.8364 0.0956 0.9773 ...
##   ..- attr(*, "names")= chr [1:10] "X100m" "Long.jump" "Shot.put" "High.jump" ...
##  $ x       : num [1:27, 1:10] 0.273 0.888 -1.347 -0.911 -0.102 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:27] "SEBRLE" "CLAY" "BERNARD" "YURKOV" ...
##   .. ..$ : chr [1:10] "PC1" "PC2" "PC3" "PC4" ...
##  - attr(*, "class")= chr "prcomp"
```

The result is a list containing 5 elements:

- **sdev:** the standard deviations of the principal components;
- **rotation:** the matrix of variable loadings;
- **center:** the centering used in **scale=TRUE**;
- **scale:** the scaling used in **scale=TRUE**;
- **x:** the scores, i.e. the rotated data;

Selecting the number of components

The selection of the number of components is necessarily ad-hoc because an unsupervised analysis does not have a prediction otucome that allows to select tuning parameters through cross-validation.

However, some less subjective approaches are available. For instance, it is possible to consider the **percentage of variance explained** (**PVE**), or the **cumulative PVE**.

```
# eigenvalue, PVE and cumulative PVE for each PC
get_eig(res)
```

```
##         eigenvalue variance.percent cumulative.variance.percent
## Dim.1    3.7499727        37.499727                    37.49973
## Dim.2    1.7451681        17.451681                    54.95141
## Dim.3    1.5178280        15.178280                    70.12969
## Dim.4    1.0322001        10.322001                    80.45169
## Dim.5    0.6178387         6.178387                    86.63008
## Dim.6    0.4282908         4.282908                    90.91298
## Dim.7    0.3259103         3.259103                    94.17209
## Dim.8    0.2793827         2.793827                    96.96591
## Dim.9    0.1911128         1.911128                    98.87704
## Dim.10   0.1122959         1.122959                   100.00000
```

Nello scree plot andro a cercare dei gomiti, come visto con elbow method.

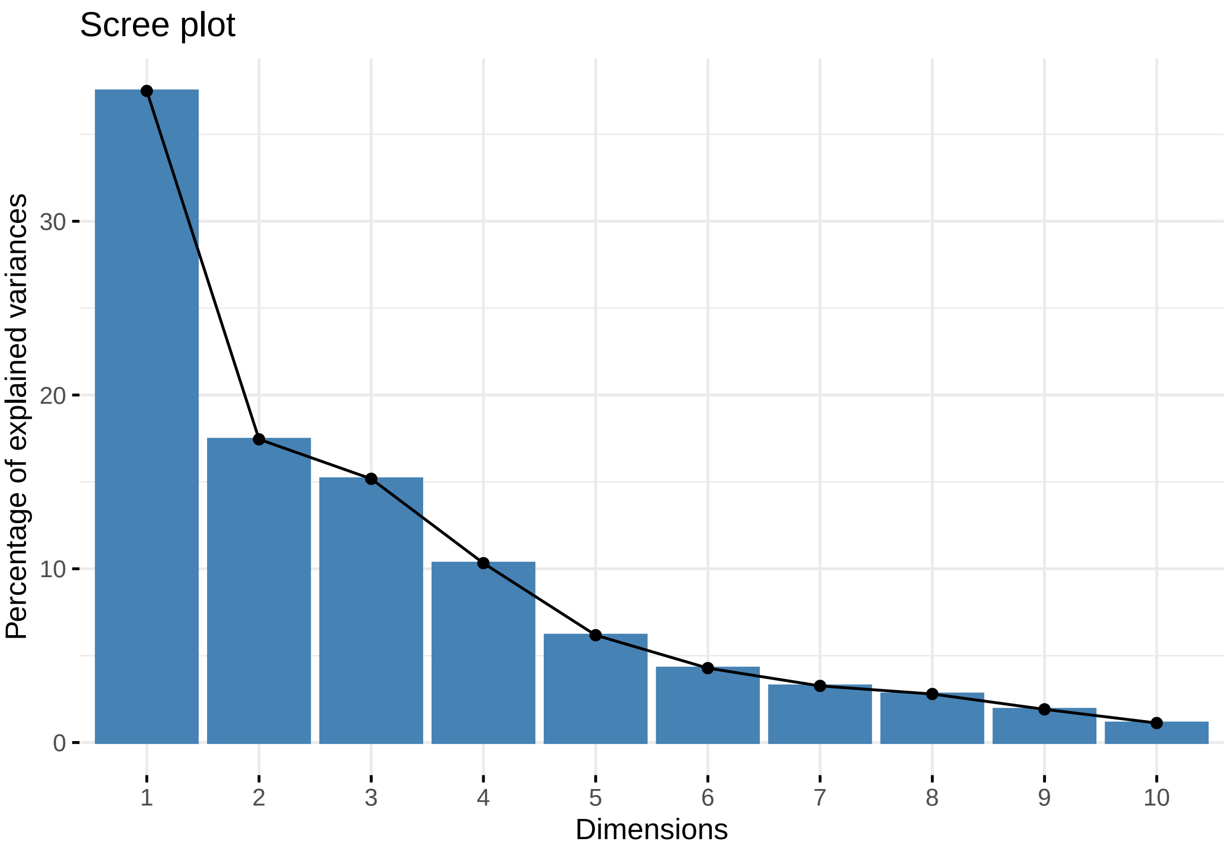
Plotto manualmente la cumulata settando una soglia orizzontale di "accettazione"

dell'80%

Using this information I can plot the **Scree plot**.

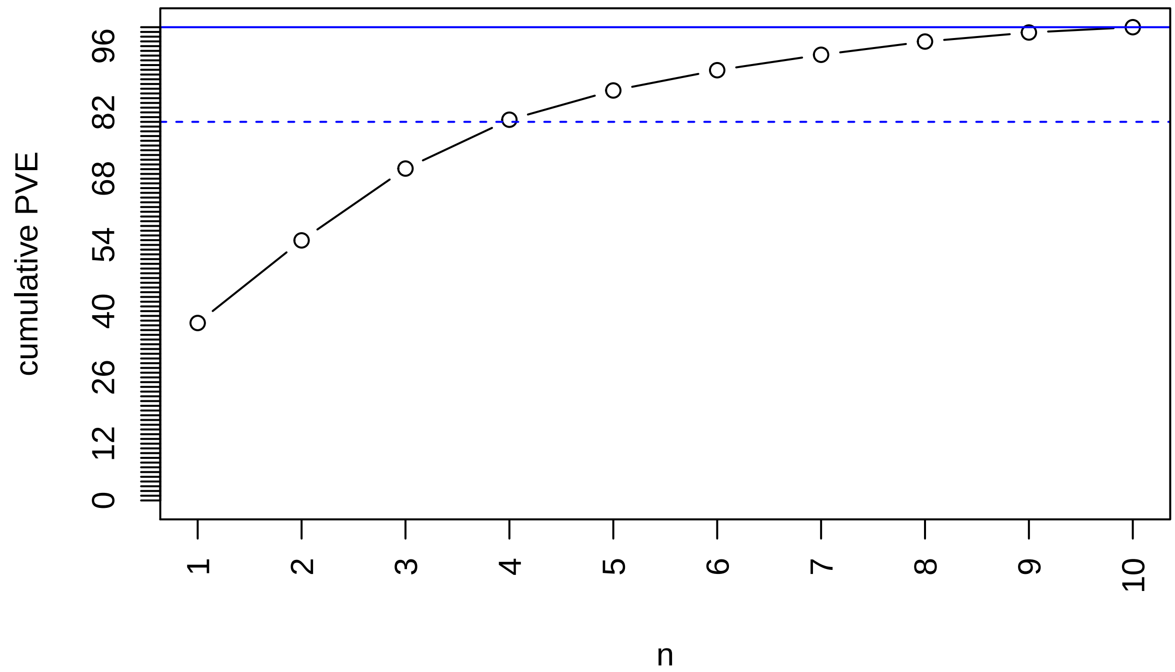
In the **Scree plot** I am looking for an *elbow*, i.e. an inflection point.

```
fviz_eig(res)
```



However, sometimes, it is not easy to identify an elbow or you want the cumulative PVE is too low. In this case, we recommend to plot the **cumulative PVE** after setting an **acceptance threshold**.

```
plot(get_eig(res)$cumulative.variance.percent, type='b', axes=F, xlab='n', ylab='cumulative PVE', ylim=
abline(h=100, col='blue')
abline(h=80, lty=2, col='blue') # thesholding
box()
axis(2, at=0:100,labels=0:100)
axis(1,at=1:ncol(decathlon2),labels=1:ncol(decathlon2),las=2)
```



Loadings interpretation

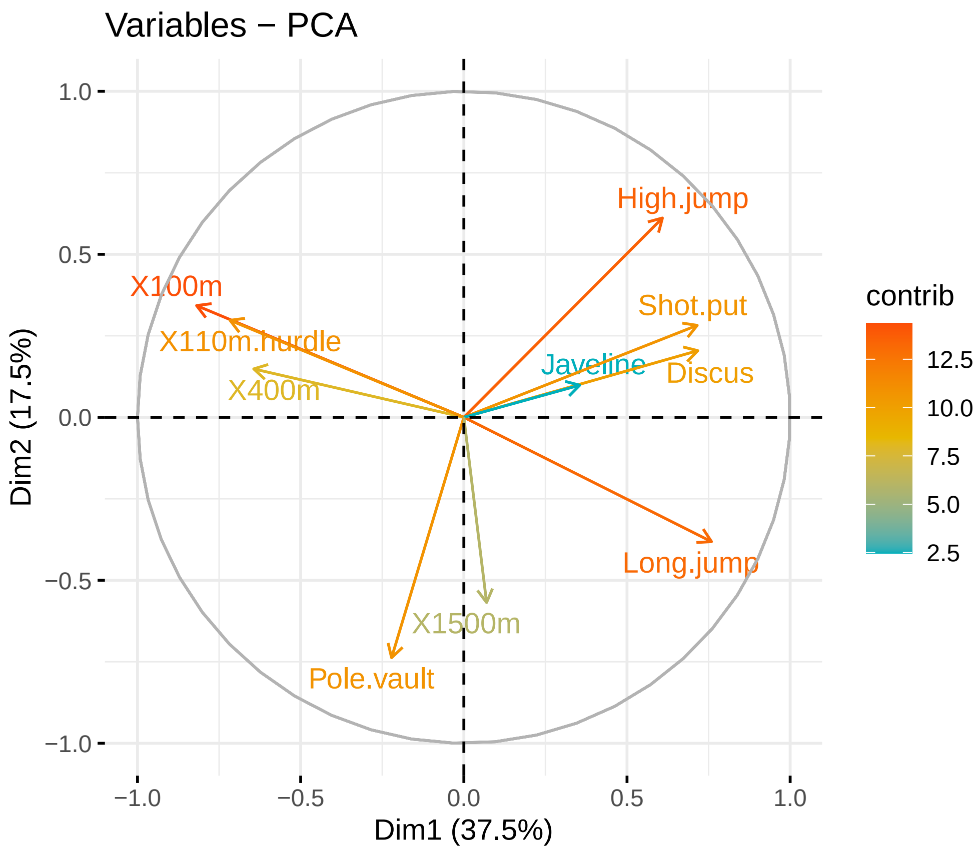
Let us focus on the loadings, i.e the eigenvectors representing the directions of the PCs.

```
loadings <- res$rotation
loadings
```

```
##                        PC1        PC2          PC3         PC4        PC5
## X100m        -0.42290657  0.2594748 -0.081870461  0.09974877 -0.2796419
## Long.jump     0.39189495 -0.2887806  0.005082180 -0.18250903  0.3355025
## Shot.put      0.36926619  0.2135552 -0.384621732  0.03553644 -0.3544877
## High.jump     0.31422571  0.4627797 -0.003738604  0.07012348  0.3824125
## X400m        -0.33248297  0.1123521 -0.418635317  0.26554389  0.2534755
## X110m.hurdle -0.36995919  0.2252392 -0.338027983 -0.15726889  0.2048540
## Discus        0.37020078  0.1547241 -0.219417086  0.39137188 -0.4319091
## Pole.vault   -0.11433982 -0.5583051 -0.327177839 -0.24759476 -0.3340758
## Javeline      0.18341259  0.0745854 -0.564474643 -0.47792535  0.1697426
## X1500m        0.03599937 -0.4300522 -0.286328973  0.64220377  0.3227349
##                      PC6         PC7         PC8         PC9        PC10
## X100m         0.16023494 -0.03227949  0.35266427 -0.71190625  0.03272397
## Long.jump     0.07384658  0.24902853  0.72986071 -0.12801382  0.02395904
## Shot.put      0.32207320  0.23059438 -0.01767069  0.07184807 -0.61708920
## High.jump     0.52738027  0.03992994 -0.25003572 -0.14583529  0.41523052
## X400m        -0.23884715  0.69014364 -0.01543618  0.13706918  0.12016951
## X110m.hurdle  0.26249611 -0.42797378  0.36415520  0.49550598 -0.03514180
## Discus       -0.28217086 -0.18416631  0.26865454  0.18621144  0.48037792
## Pole.vault    0.43606610  0.12654370 -0.16086549  0.02983660  0.40290423
## Javeline     -0.42368592 -0.23324548 -0.19922452 -0.33300936  0.02100398
## X1500m        0.10850981 -0.34406521 -0.09752169 -0.19899138 -0.18954698
```

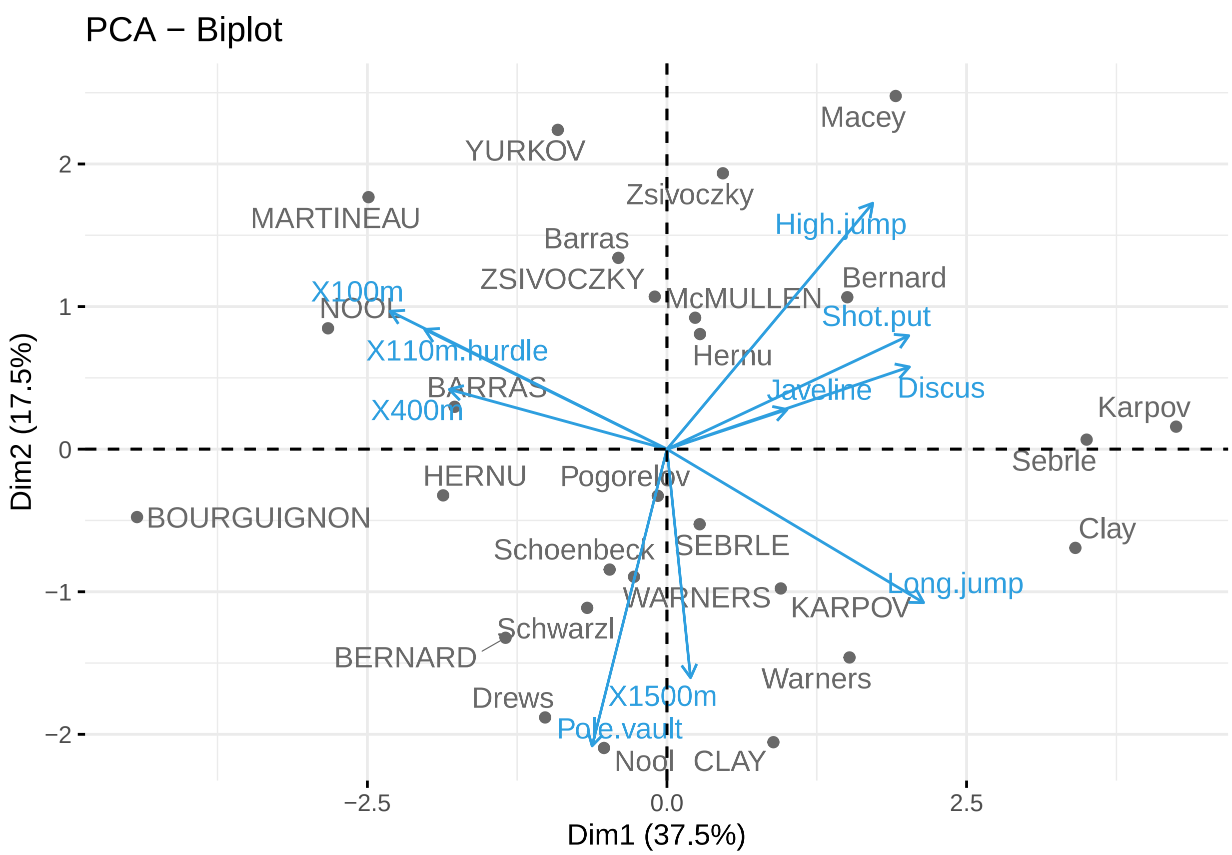
We can plot the first two PCs (PC1 and PC2) in the **graph of variables**. In this plot the importance of the original feature is represented by the **color code** (red: *high* - medium: *blue* - white: *low*), and by the lenght of the vector (*close or not* to the circumference).

```
fviz_pca_var(res,
             col.var = "contrib", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE )     # Avoid text overlapping
```



If we want to show also the individuals we can use the **biplot of infividuals and variables**.

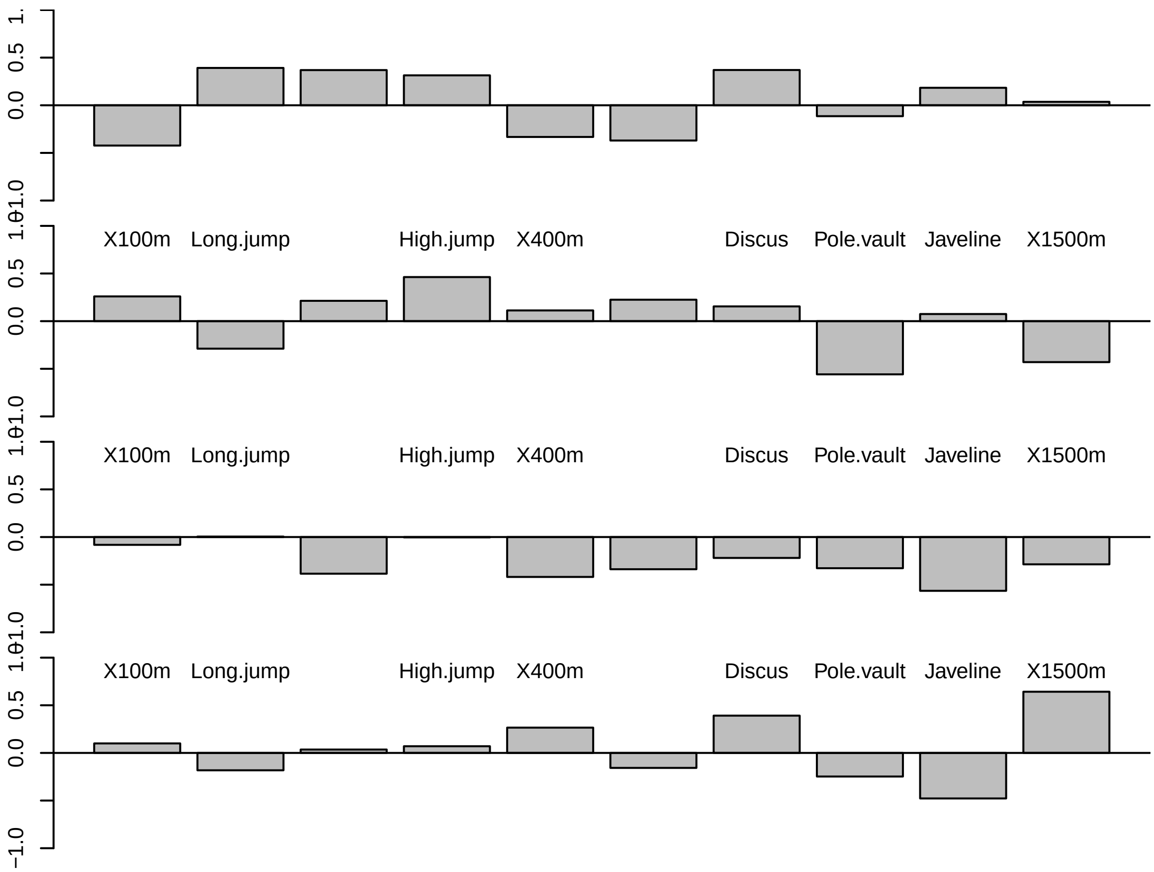
```
fviz_pca_biplot(res, repel = TRUE,
                col.var = "#2E9FDF", # Variables color
                col.ind = "#696969"  # Individuals color
)
```



In both the plots, positively correlated variables have same direction. Negatively correlated variables have opposite directions.

Information regarding all the PCs (for instance the first 4) can be obtained in the following way:

```
plot.new()
par(mar = c(1,4,0,2), mfrow = c(4,1))
for(i in 1:4)
{
  barplot(loadings[,i], ylim = c(-1, 1))
  abline(h=0)
}
```


1.
0.5
0.0
−1.0
X100m Long.jump High.jump X400m Discus Pole.vault Javeline X1500m
X100m Long.jump High.jump X400m Discus Pole.vault Javeline X1500m
X100m Long.jump High.jump X400m Discus Pole.vault Javeline X1500m