



[Home](#) / [Articles](#) / [Principal Component Methods in R: Practical Guide](#) / [PCA - Principal Component Analysis Essentials](#)



**MS Access
Automation**

Try MARS Free for 30-Days.
Download Now!



Articles - Principal Component Methods in R: Practical Guide

PCA - Principal Component Analysis Essentials

[kassambara](#) | 23/09/2017 | 28035 | [Comments \(12\)](#) | [Principal Component Methods in R: Practical Guide](#) | [Multi-variate Analysis](#)

Principal component analysis (PCA) allows us to summarize and to visualize the information in a data set containing individuals/observations described by multiple inter-correlated quantitative variables. Each variable could be considered as a different dimension. If you have more than 3 variables in your data sets, it could be very difficult to visualize a multi-dimensional hyperspace.

Principal component analysis is used to extract the important information from a multivariate data table and to express this information as a set of few new variables called **principal components**. These new variables correspond to a linear combination of the originals. The number of principal components is less than or equal to the number of original variables.

The information in a given data set corresponds to the **total variation** it contains. The goal of PCA is to identify directions (or principal components) along which the variation in the data is maximal.

In other words, PCA reduces the dimensionality of a multivariate data to two or three principal components, that can be visualized graphically, with minimal loss of information.

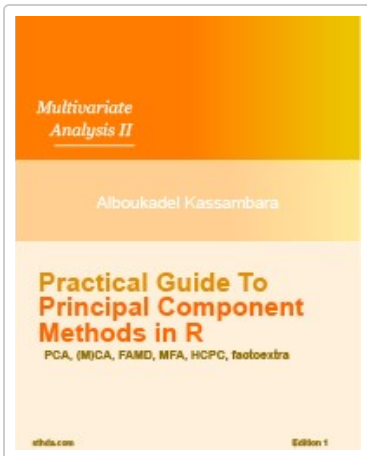
In this chapter, we describe the basic idea of PCA and, demonstrate how to compute and visualize PCA using R software. Additionally, we'll show how to reveal the most important variables that explain the variations in a data set.

Contents:

- [Basics](#)
- [Computation](#)
 - [R packages](#)
 - [Data format](#)
 - [Data standardization](#)
 - [R code](#)
- [Visualization and Interpretation](#)
 - [Eigenvalues / Variances](#)
 - [Graph of variables](#)
 - [Dimension description](#)
 - [Graph of individuals](#)
 - [Graph customization](#)
 - [Biplot](#)
- [Supplementary elements](#)
 - [Definition and types](#)
 - [Specification in PCA](#)
 - [Quantitative variables](#)
 - [Individuals](#)
 - [Qualitative variables](#)
- [Filtering results](#)
- [Exporting results](#)
 - [Export plots to PDF/PNG files](#)
 - [Export results to txt/csv files](#)
- [Summary](#)
- [Further reading](#)
- [References](#)

The Book:





Practical Guide to Principal Component Methods in R

Basics

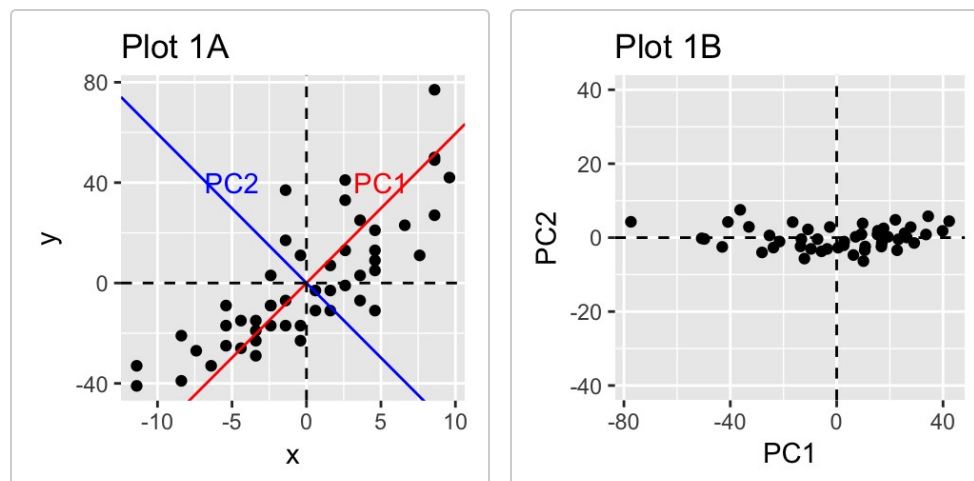
Understanding the details of PCA requires knowledge of linear algebra. Here, we'll explain only the basics with simple graphical representation of the data.

In the Plot 1A below, the data are represented in the X-Y coordinate system. The dimension reduction is achieved by identifying the principal directions, called principal components, in which the data varies.

PCA assumes that the directions with the largest variances are the most “important” (i.e, the most principal).

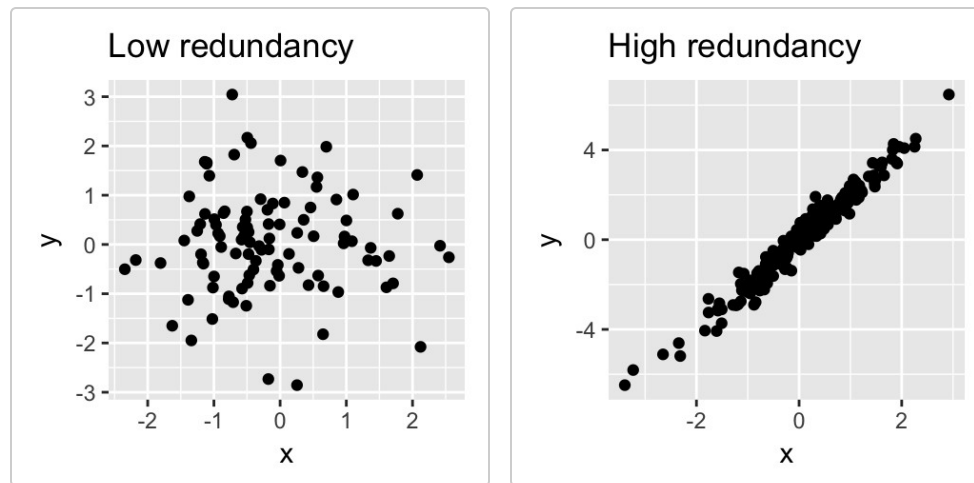
In the figure below, the **PC1 axis** is the **first principal direction** along which the samples show the largest variation. The **PC2 axis** is the **second most important direction** and it is **orthogonal** to the PC1 axis.

The dimensionality of our two-dimensional data can be reduced to a single dimension by projecting each sample onto the first principal component (Plot 1B)



Technically speaking, the amount of variance retained by each principal component is measured by the so-called **eigenvalue**.

Note that, the PCA method is particularly useful when the variables within the data set are highly correlated. Correlation indicates that there is redundancy in the data. Due to this redundancy, PCA can be used to reduce the original variables into a smaller number of new variables (= **principal components**) explaining most of the variance in the original variables.



Taken together, the main purpose of principal component analysis is to:

- identify hidden pattern in a data set,
- reduce the dimensionnality of the data by removing the noise and redundancy in the data,
- identify correlated variables

Computation

R packages

Several functions from different packages are available in the *R software* for computing PCA:

- `prcomp()` and `princomp()` [built-in R *stats* package],
- `PCA()` [*FactoMineR* package],
- `dudi.pca()` [*ade4* package],
- and `epPCA()` [*ExPosition* package]

No matter what function you decide to use, you can easily extract and visualize the results of PCA using R functions provided in the **factoextra** R package.

✓ Here, we'll use the two packages **FactoMineR** (for the analysis) and **factoextra** (for ggplot2-based visualization).

Install the two packages as follow:

```
install.packages(c("FactoMineR", "factoextra"))
```

Load them in R, by typing this:

```
library("FactoMineR")
library("factoextra")
```

Data format

We'll use the demo data sets `decathlon2` from the `factoextra` package:

```
data(decathlon2)
# head(decathlon2)
```

As illustrated in Figure 3.1, the data used here describes athletes' performance during two sporting events (Decstar and OlympicG). It contains 27 individuals (athletes) described by 13 variables.

name	100m	Long.jump	//	Javeline	1500m	Rank	Points	Competition
SEBRLE	11.04	7.58		63.19	291.7	1	8217	Decastar
CLAY	10.76	7.4		60.15	301.5	2	8122	Decastar
Macey	10.89	7.47		58.46	265.42	4	8414	OlympicG
Warners	10.62	7.74		55.39	278.05	5	8343	OlympicG
\\								
Zsivoczky	10.91	7.14		63.45	269.54	6	8287	OlympicG
Hernu	10.97	7.19		57.76	264.35	7	8237	OlympicG
Pogorelov	10.95	7.31		53.45	287.63	11	8084	OlympicG
Schoenbeck	10.9	7.3		60.89	278.82	12	8077	OlympicG
Barras	11.14	6.99		64.55	267.09	13	8067	OlympicG
KARPOV	11.02	7.3		50.31	300.2	3	8099	Decastar
WARNERS	11.11	7.6		51.77	278.1	6	8030	Decastar
Nool	10.8	7.53		61.33	276.33	8	8235	OlympicG
Drews	10.87	7.38		51.53	274.21	19	7926	OlympicG

Active individuals

Active variables

Supplementary quantitative variables

Supplementary qualitative variable

Supplementary individuals



Note that, only some of these individuals and variables will be used to perform the principal component analysis. The coordinates of the remaining individuals and variables on the factor map will be predicted after the PCA.

In PCA terminology, our data contains :

- *Active individuals* (in light blue, rows 1:23) : Individuals that are used during the principal component analysis.
- *Supplementary individuals* (in dark blue, rows 24:27) : The coordinates of these individuals will be predicted using the PCA information and parameters obtained with active individuals/variables
- *Active variables* (in pink, columns 1:10) : Variables that are used for the principal component analysis.
- *Supplementary variables*: As supplementary individuals, the coordinates of these variables will be predicted also. These can be:
 - *Supplementary continuous variables* (red): Columns 11 and 12 corresponding respectively to the rank and the points of athletes.
 - *Supplementary qualitative variables* (green): Column 13 corresponding to the two athletic meetings (2004 Olympic Game or 2004 Decastar). This is a categorical (or factor) variable factor. It can be used to color individuals by groups.

We start by subsetting active individuals and active variables for the principal component analysis:

```
decathlon2.active <- decathlon2[1:23, 1:10]
head(decathlon2.active[, 1:6], 4)
```

##		X100m	Long.jump	Shot.put	High.jump	X400m	X110m.hurdle
##	SEBRLE	11.0	7.58	14.8	2.07	49.8	14.7
##	CLAY	10.8	7.40	14.3	1.86	49.4	14.1
##	BERNARD	11.0	7.23	14.2	1.92	48.9	15.0
##	YURKOV	11.3	7.09	15.2	2.10	50.4	15.3

Data standardization

In principal component analysis, variables are often scaled (i.e. standardized). This is particularly recommended when variables are measured in different scales (e.g: kilograms, kilometers, centimeters, ...); otherwise, the PCA outputs obtained will be severely affected.

The goal is to make the variables comparable. Generally variables are scaled to have i) standard deviation one and ii) mean zero.

The standardization of data is an approach widely used in the context of gene expression data analysis before PCA and clustering analysis. We might also want to scale the data when the mean and/or the standard deviation of variables are largely different.

When scaling variables, the data can be transformed as follow:

$$\frac{x_i - \text{mean}(x)}{\text{sd}(x)}$$

Where $\text{mean}(x)$ is the mean of x values, and $\text{sd}(x)$ is the standard deviation (SD).

The R base function `scale()` can be used to standardize the data. It takes a numeric matrix as an input and performs the scaling on the columns.



Note that, by default, the function **PCA()** [in *FactoMineR*], standardizes the data automatically during the PCA; so you don't need do this transformation before the PCA.

R code

The function `PCA()` [*FactoMineR* package] can be used. A simplified format is :

```
PCA(X, scale.unit = TRUE, ncp = 5, graph = TRUE)
```

- **X**: a data frame. Rows are individuals and columns are numeric variables
- **scale.unit**: a logical value. If `TRUE`, the data are scaled to unit variance before the analysis. This standardization to the same scale avoids some variables to become dominant just because of their large measurement units. It makes variable comparable.
- **ncp**: number of dimensions kept in the final results.
- **graph**: a logical value. If `TRUE` a graph is displayed.

The R code below, computes principal component analysis on the active individuals/variables:

```
library("FactoMineR")
res.pca <- PCA(decathlon2.active, graph = FALSE)
```

The output of the function `PCA()` is a list, including the following components :

```
print(res.pca)
```

```
## **Results for the Principal Component Analysis (PCA)**
## The analysis was performed on 23 individuals, described by 10 variables
## *The results are available in the following objects:
##
##      name                description
## 1  "$eig"                "eigenvalues"
```

```
## 2 "$var" "results for the variables"
## 3 "$var$coord" "coord. for the variables"
## 4 "$var$cor" "correlations variables - dimensions"
## 5 "$var$cos2" "cos2 for the variables"
## 6 "$var$contrib" "contributions of the variables"
## 7 "$ind" "results for the individuals"
## 8 "$ind$coord" "coord. for the individuals"
## 9 "$ind$cos2" "cos2 for the individuals"
## 10 "$ind$contrib" "contributions of the individuals"
## 11 "$call" "summary statistics"
## 12 "$call$centre" "mean of the variables"
## 13 "$call$ecart.type" "standard error of the variables"
## 14 "$call$row.w" "weights for the individuals"
## 15 "$call$col.w" "weights for the variables"
```



The object that is created using the function `PCA()` contains many information found in many different lists and matrices. These values are described in the next section.

Visualization and Interpretation

We'll use the *factoextra* R package to help in the interpretation of PCA. No matter what function you decide to use [`stats::prcomp()`, `FactoMiner::PCA()`, `ade4::dudi.pca()`, `ExPosition::epPCA()`], you can easily extract and visualize the results of PCA using R functions provided in the *factoextra* R package.

These functions include:

- `get_eigenvalue(res.pca)`: Extract the eigenvalues/variances of principal components
- `fviz_eig(res.pca)`: Visualize the eigenvalues
- `get_pca_ind(res.pca)`, `get_pca_var(res.pca)`: Extract the results for individuals and variables, respectively.
- `fviz_pca_ind(res.pca)`, `fviz_pca_var(res.pca)`: Visualize the results individuals and variables, respectively.
- `fviz_pca_biplot(res.pca)`: Make a biplot of individuals and variables.

In the next sections, we'll illustrate each of these functions.

Eigenvalues / Variances

As described in previous sections, the *eigenvalues* measure the amount of variation retained by each principal component. *Eigenvalues* are large for the first PCs and small for the subsequent PCs. That is, the first PCs corresponds to the directions with the maximum amount of variation in the data set.

We examine the eigenvalues to determine the number of principal components to be considered. The eigenvalues and the proportion of variances (i.e., information) retained by the principal components (PCs) can be extracted using the function `get_eigenvalue()` [*factoextra* package].


```
library("factoextra")
eig.val <- get_eigenvalue(res.pca)
eig.val
```

```
##          eigenvalue variance.percent cumulative.variance.percent
## Dim.1          4.124           41.24              41.2
## Dim.2          1.839           18.39              59.6
## Dim.3          1.239           12.39              72.0
## Dim.4          0.819            8.19              80.2
## Dim.5          0.702            7.02              87.2
## Dim.6          0.423            4.23              91.5
## Dim.7          0.303            3.03              94.5
## Dim.8          0.274            2.74              97.2
## Dim.9          0.155            1.55              98.8
## Dim.10         0.122            1.22             100.0
```

The sum of all the eigenvalues give a total variance of 10.

The proportion of variation explained by each eigenvalue is given in the second column. For example, 4.124 divided by 10 equals 0.4124, or, about 41.24% of the variation is explained by this first eigenvalue. The cumulative percentage explained is obtained by adding the successive proportions of variation explained to obtain the running total. For instance, 41.242% plus 18.385% equals 59.627%, and so forth. Therefore, about 59.627% of the variation is explained by the first two eigenvalues together.

Eigenvalues can be used to determine the number of principal components to retain after PCA (Kaiser 1961):

- An *eigenvalue* > 1 indicates that PCs account for more variance than accounted by one of the original variables in standardized data. This is commonly used as a cutoff point for which PCs are retained. This holds true only when the data are standardized.
- You can also limit the number of component to that number that accounts for a certain fraction of the total variance. For example, if you are satisfied with 70% of the total variance explained then use the number of components to achieve that.

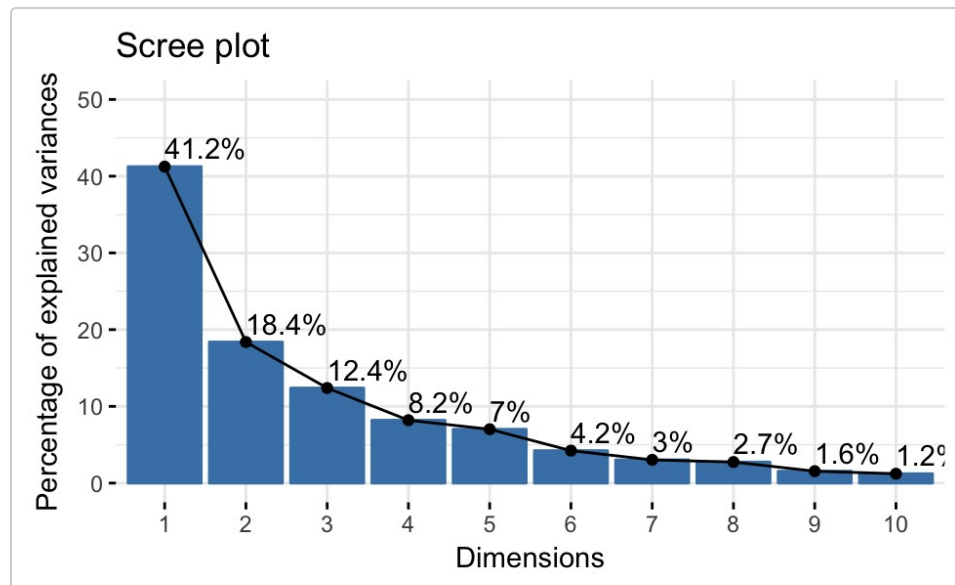
Unfortunately, there is no well-accepted objective way to decide how many principal components are enough. This will depend on the specific field of application and the specific data set. In practice, we tend to look at the first few principal components in order to find interesting patterns in the data.

In our analysis, the first three principal components explain 72% of the variation. This is an acceptably large percentage.

An alternative method to determine the number of principal components is to look at a Scree Plot, which is the plot of eigenvalues ordered from largest to the smallest. The number of component is determined at the point, beyond which the remaining eigenvalues are all relatively small and of comparable size (Jolliffe 2002, Peres-Neto, Jackson, and Somers (2005)).

The scree plot can be produced using the function `fviz_eig()` or `fviz_screplot()` [*factoextra* package].

```
fviz_eig(res.pca, addlabels = TRUE, ylim = c(0, 50))
```



From the plot above, we might want to stop at the fifth principal component. 87% of the information (variances) contained in the data are retained by the first five principal components.

Graph of variables

Results

A simple method to extract the results, for variables, from a PCA output is to use the function `get_pca_var()` [*factoextra* package]. This function provides a list of matrices containing all the results for the active variables (coordinates, correlation between variables and axes, squared cosine and contributions)

```
var <- get_pca_var(res.pca)
var
```

```
## Principal Component Analysis Results for variables
## =====
##   Name      Description
```

```
## 1 "$coord"    "Coordinates for the variables"
## 2 "$cor"      "Correlations between variables and dimensions"
## 3 "$cos2"     "Cos2 for the variables"
## 4 "$contrib"  "contributions of the variables"
```

The components of the `get_pca_var()` can be used in the plot of variables as follow:

- `var$coord`: coordinates of variables to create a scatter plot
- `var$cos2`: represents the quality of representation for variables on the factor map. It's calculated as the squared coordinates: $\text{var.cos2} = \text{var.coord} * \text{var.coord}$.
- `var$contrib`: contains the contributions (in percentage) of the variables to the principal components. The contribution of a variable (var) to a given principal component is (in percentage) : $(\text{var.cos2} * 100) / (\text{total cos2 of the component})$.

✓ Note that, it's possible to plot variables and to color them according to either i) their quality on the factor map (cos2) or ii) their contribution values to the principal components (contrib).

The different components can be accessed as follow:

```
# Coordinates
head(var$coord)
# Cos2: quality on the factor map
head(var$cos2)
# Contributions to the principal components
head(var$contrib)
```

In this section, we describe how to visualize variables and draw conclusions about their correlations. Next, we highlight variables according to either i) their quality of representation on the factor map or ii) their contributions to the principal components.

Correlation circle

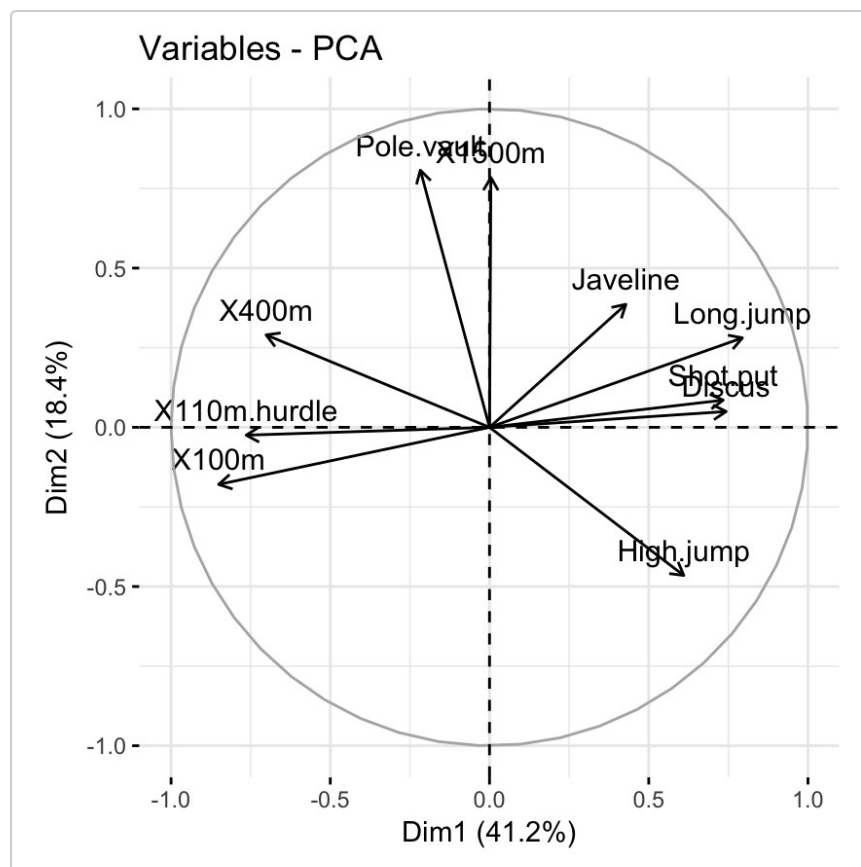
The correlation between a variable and a principal component (PC) is used as the coordinates of the variable on the PC. The representation of variables differs from the plot of the observations: The observations are represented by their projections, but the variables are represented by their correlations (Abdi and Williams 2010).

```
# Coordinates of variables
head(var$coord, 4)
```

```
##          Dim.1  Dim.2  Dim.3  Dim.4  Dim.5
## X100m      -0.851 -0.1794  0.302  0.0336 -0.194
## Long.jump  0.794  0.2809 -0.191 -0.1154  0.233
## Shot.put   0.734  0.0854  0.518  0.1285 -0.249
## High.jump  0.610 -0.4652  0.330  0.1446  0.403
```

To plot variables, type this:

```
fviz_pca_var(res.pca, col.var = "black")
```



The plot above is also known as variable correlation plots. It shows the relationships between all variables. It can be interpreted as follow:

- Positively correlated variables are grouped together.
- Negatively correlated variables are positioned on opposite sides of the plot origin (opposed quadrants).
- The distance between variables and the origin measures the quality of the variables on the factor map. Variables that are away from the origin are well represented on the factor map.

Quality of representation

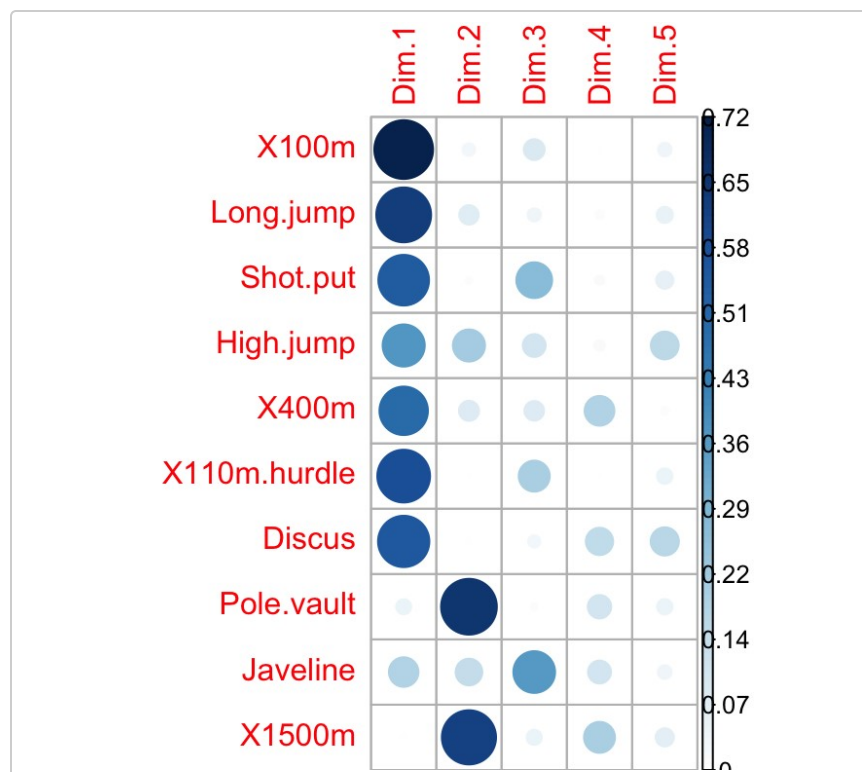
The quality of representation of the variables on factor map is called **cos2** (square cosine, squared coordinates) . You can access to the cos2 as follow:

```
head(var$cos2, 4)
```

```
##           Dim.1  Dim.2 Dim.3  Dim.4 Dim.5
## X100m      0.724 0.03218 0.0909 0.00113 0.0378
## Long.jump  0.631 0.07888 0.0363 0.01331 0.0544
## Shot.put   0.539 0.00729 0.2679 0.01650 0.0619
## High.jump  0.372 0.21642 0.1090 0.02089 0.1622
```

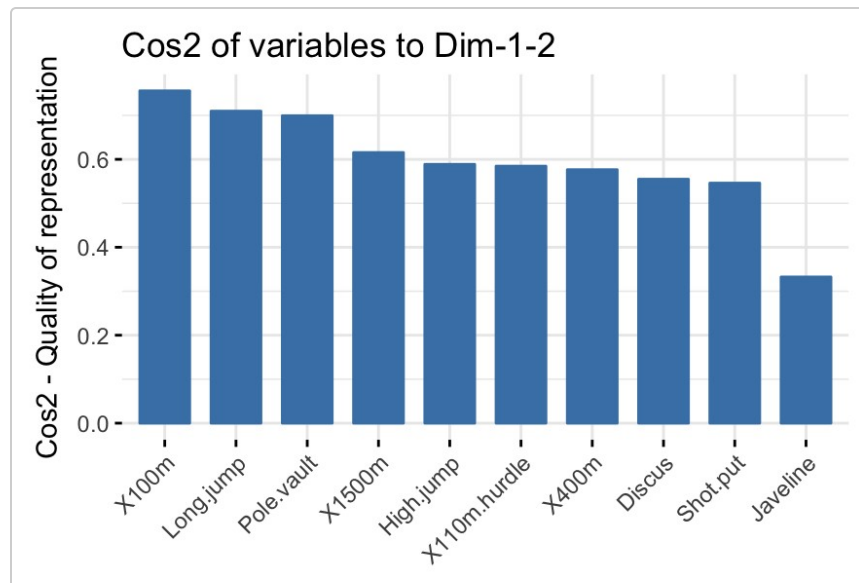
You can visualize the cos2 of variables on all the dimensions using the corrplot package:

```
library("corrplot")
corrplot(var$cos2, is.corr=FALSE)
```



It's also possible to create a bar plot of variables cos2 using the function `fviz_cos2()` [in factoextra]:

```
# Total cos2 of variables on Dim.1 and Dim.2  
fviz_cos2(res.pca, choice = "var", axes = 1:2)
```



Note that,

- A high cos2 indicates a good representation of the variable on the principal component. In this case the variable is positioned close to the circumference of the correlation circle.
- A low cos2 indicates that the variable is not perfectly represented by the PCs. In this case the variable is close to the center of the circle.

For a given variable, the sum of the cos2 on all the principal components is equal to one.

If a variable is perfectly represented by only two principal components (Dim.1 & Dim.2), the sum of the cos2 on these two PCs is equal to one. In this case the variables will be positioned on the circle of correlations.

For some of the variables, more than 2 components might be required to perfectly represent the data. In this case the variables are positioned inside the circle of correlations.

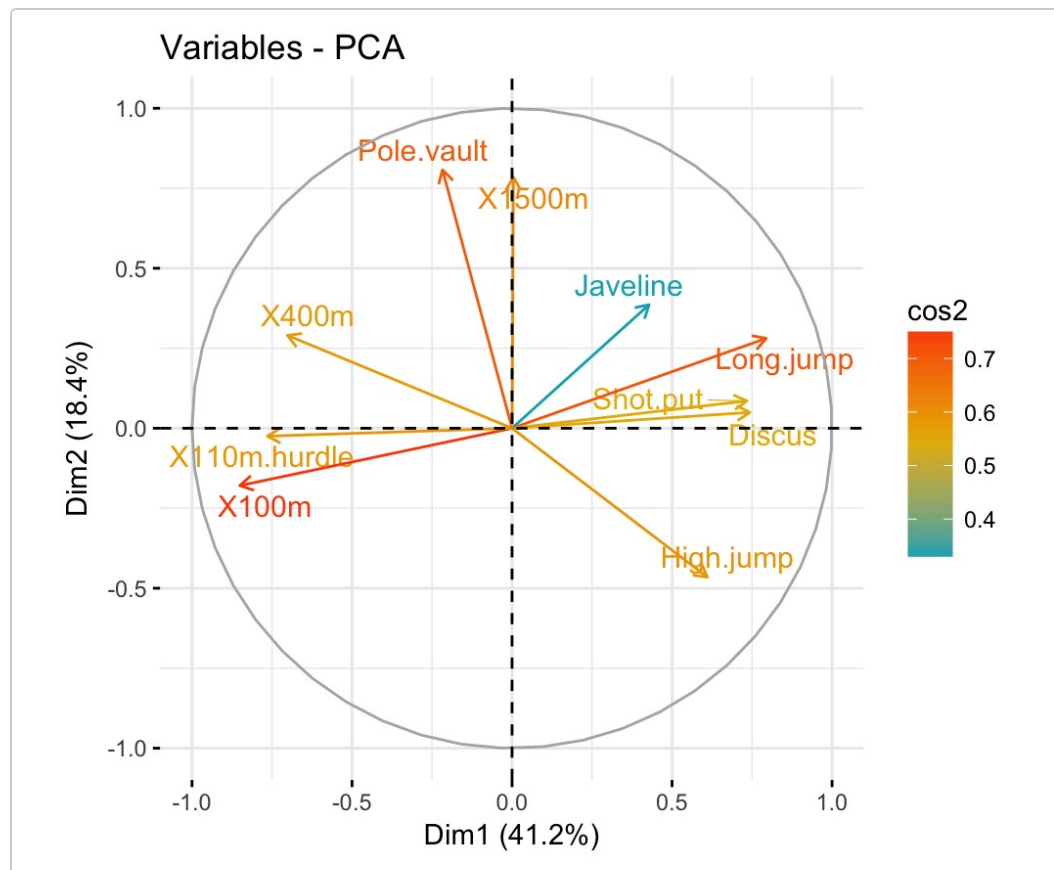
In summary:

- The cos2 values are used to estimate the quality of the representation
- The closer a variable is to the circle of correlations, the better its representation on the factor map (and the more important it is to interpret these components)
- Variables that are closed to the center of the plot are less important for the first components.

It's possible to color variables by their `cos2` values using the argument `col.var = "cos2"`. This produces a gradient colors. In this case, the argument `gradient.cols` can be used to provide a custom color. For instance, `gradient.cols = c("white", "blue", "red")` means that:

- variables with low `cos2` values will be colored in "white"
- variables with mid `cos2` values will be colored in "blue"
- variables with high `cos2` values will be colored in red

```
# Color by cos2 values: quality on the factor map
fviz_pca_var(res.pca, col.var = "cos2",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE # Avoid text overlapping
            )
```



Note that, it's also possible to change the transparency of the variables according to their `cos2` values using the option `alpha.var = "cos2"`. For example, type this:

```
# Change the transparency by cos2 values
fviz_pca_var(res.pca, alpha.var = "cos2")
```

Contributions of variables to PCs

The contributions of variables in accounting for the variability in a given principal component are expressed in percentage.

- Variables that are correlated with PC1 (i.e., Dim.1) and PC2 (i.e., Dim.2) are the most important in explaining the variability in the data set.
- Variables that do not correlated with any PC or correlated with the last dimensions are variables with low contribution and might be removed to simplify the overall analysis.

The contribution of variables can be extracted as follow :

```
head(var$contrib, 4)
```

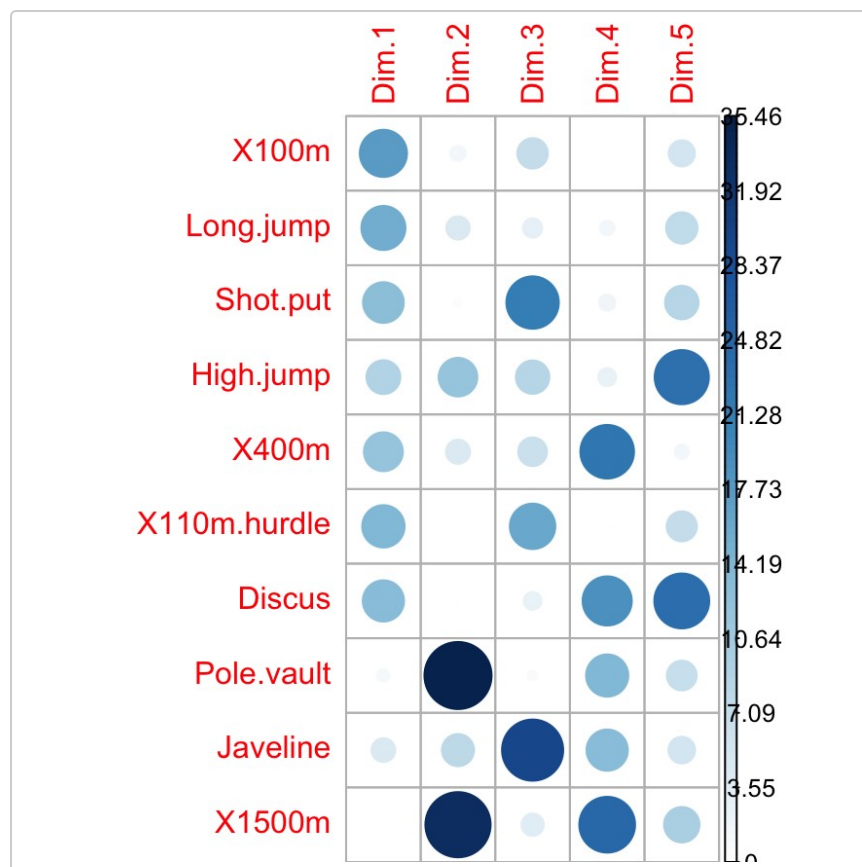
```
##           Dim.1 Dim.2 Dim.3 Dim.4 Dim.5
## X100m      17.54  1.751  7.34 0.138  5.39
## Long.jump  15.29  4.290  2.93 1.625  7.75
## Shot.put   13.06  0.397 21.62 2.014  8.82
## High.jump   9.02 11.772  8.79 2.550 23.12
```



The larger the value of the contribution, the more the variable contributes to the component.

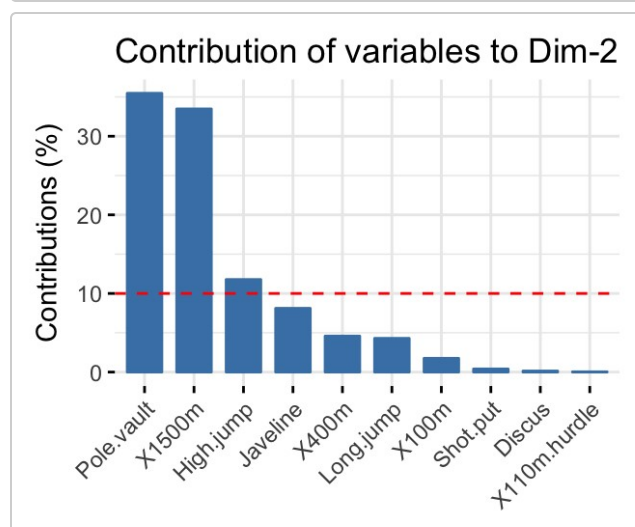
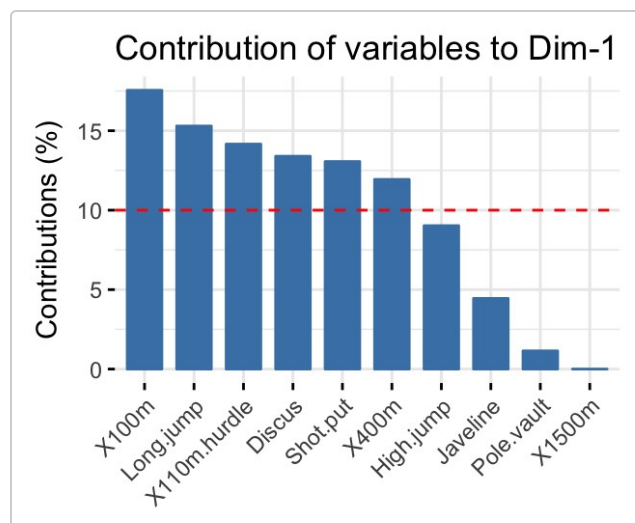
It's possible to use the function `corrplot()` [corrplot package] to highlight the most contributing variables for each dimension:

```
library("corrplot")
corrplot(var$contrib, is.corr=FALSE)
```

The function `fviz_contrib()` [factoextra package] can be used to draw a bar plot of variable contributions. If your data contains many variables, you can decide to show only the top contributing variables. The R code below shows the top 10 variables contributing to the principal components:

```
# Contributions of variables to PC1
fviz_contrib(res.pca, choice = "var", axes = 1, top = 10)
# Contributions of variables to PC2
fviz_contrib(res.pca, choice = "var", axes = 2, top = 10)
```



The total contribution to PC1 and PC2 is obtained with the following R code:

```
fviz_contrib(res.pca, choice = "var", axes = 1:2, top = 10)
```

The red dashed line on the graph above indicates the expected average contribution. If the contribution of the variables were uniform, the expected value would be $1/\text{length}(\text{variables}) = 1/10 = 10\%$. For a given component, a variable with a contribution larger than this cutoff could be considered as important in contributing to the component.

Note that, the total contribution of a given variable, on explaining the variations retained by two principal components, say PC1 and PC2, is calculated as $\text{contrib} = [(C1 * \text{Eig1}) + (C2 * \text{Eig2})]/(\text{Eig1} + \text{Eig2})$, where

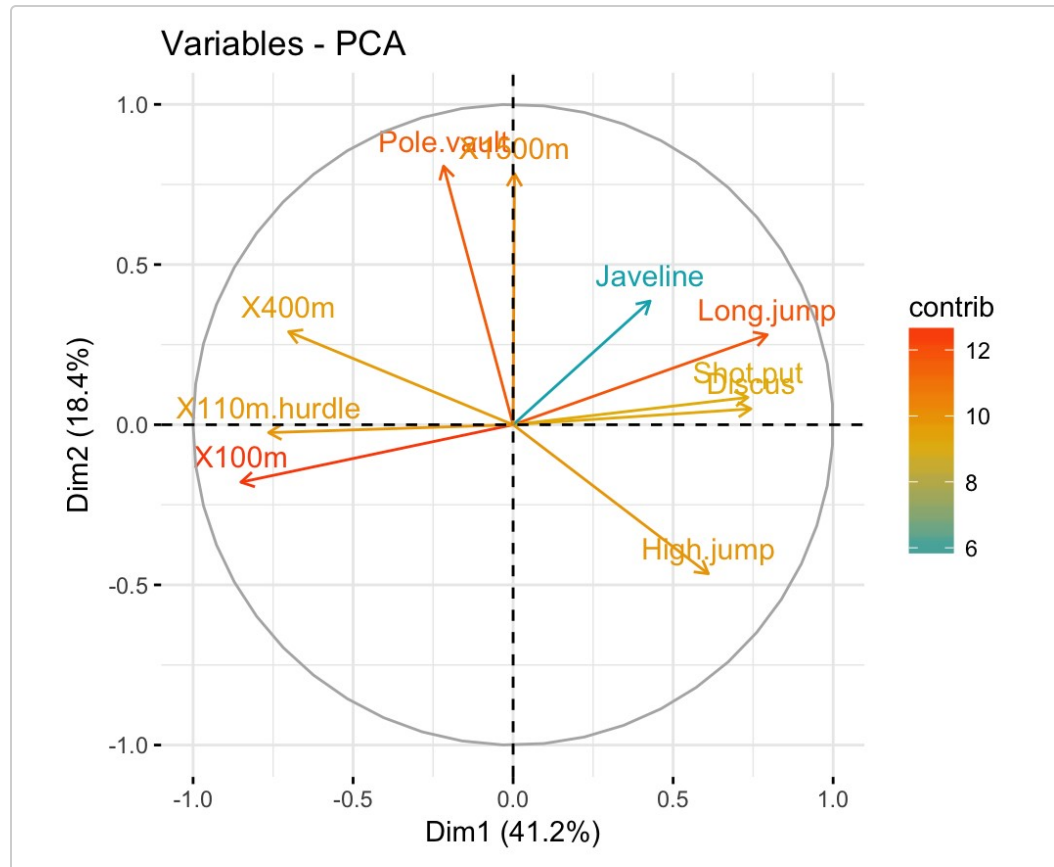
- C1 and C2 are the contributions of the variable on PC1 and PC2, respectively
- Eig1 and Eig2 are the eigenvalues of PC1 and PC2, respectively. Recall that eigenvalues measure the amount of variation retained by each PC.

In this case, the expected average contribution (cutoff) is calculated as follow: As mentioned above, if the contributions of the 10 variables were uniform, the expected average contribution on a given PC would be $1/10 = 10\%$. The expected average contribution of a variable for PC1 and PC2 is : $[(10 * Eig1) + (10 * Eig2)] / (Eig1 + Eig2)$

✓ It can be seen that the variables - X100m, Long.jump and Pole.vault - contribute the most to the dimensions 1 and 2.

The most important (or, contributing) variables can be highlighted on the correlation plot as follow:

```
fviz_pca_var(res.pca, col.var = "contrib",
              gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07")
            )
```



Note that, it's also possible to change the transparency of variables according to their contrib values using the option `alpha.var = "contrib"`. For example, type this:

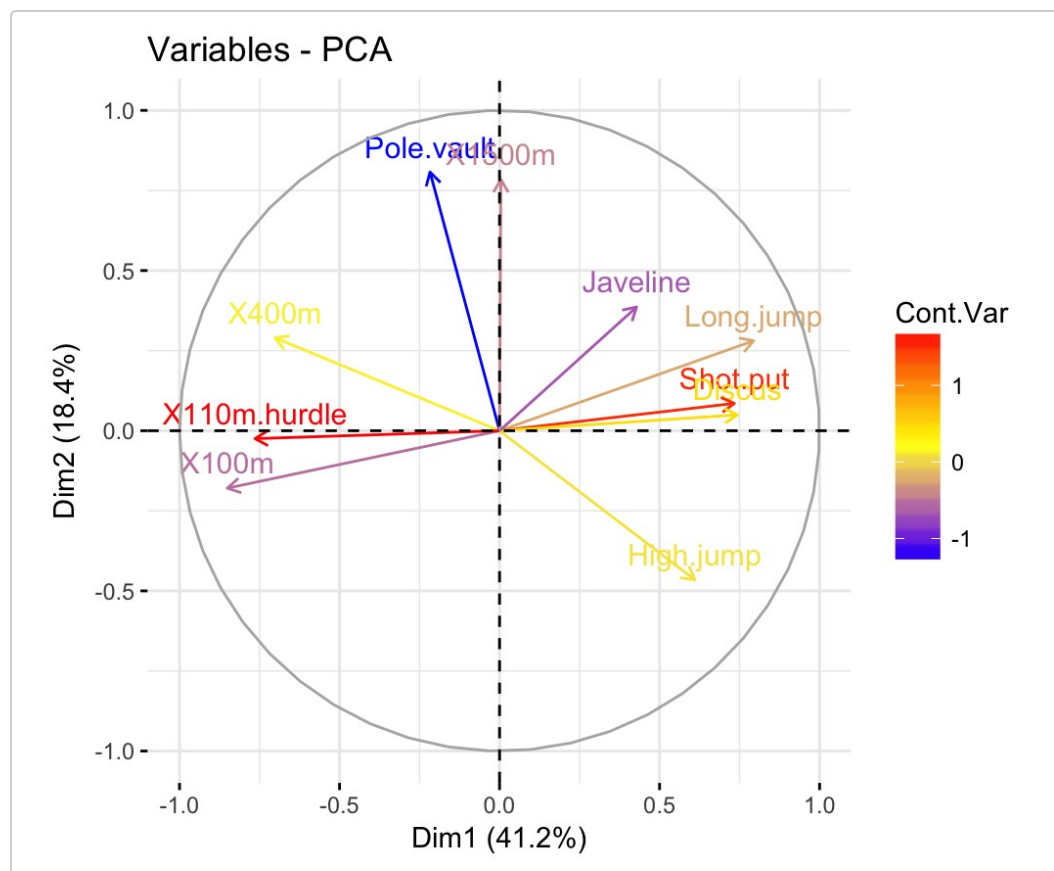
```
# Change the transparency by contrib values
fviz_pca_var(res.pca, alpha.var = "contrib")
```

Color by a custom continuous variable

In the previous sections, we showed how to color variables by their contributions and their \cos^2 . Note that, it's possible to color variables by any custom continuous variable. The coloring variable should have the same length as the number of active variables in the PCA (here $n = 10$).

For example, type this:

```
# Create a random continuous variable of length 10
set.seed(123)
my.cont.var <- rnorm(10)
# Color variables by the continuous variable
fviz_pca_var(res.pca, col.var = my.cont.var,
             gradient.cols = c("blue", "yellow", "red"),
             legend.title = "Cont.Var")
```



Color by groups

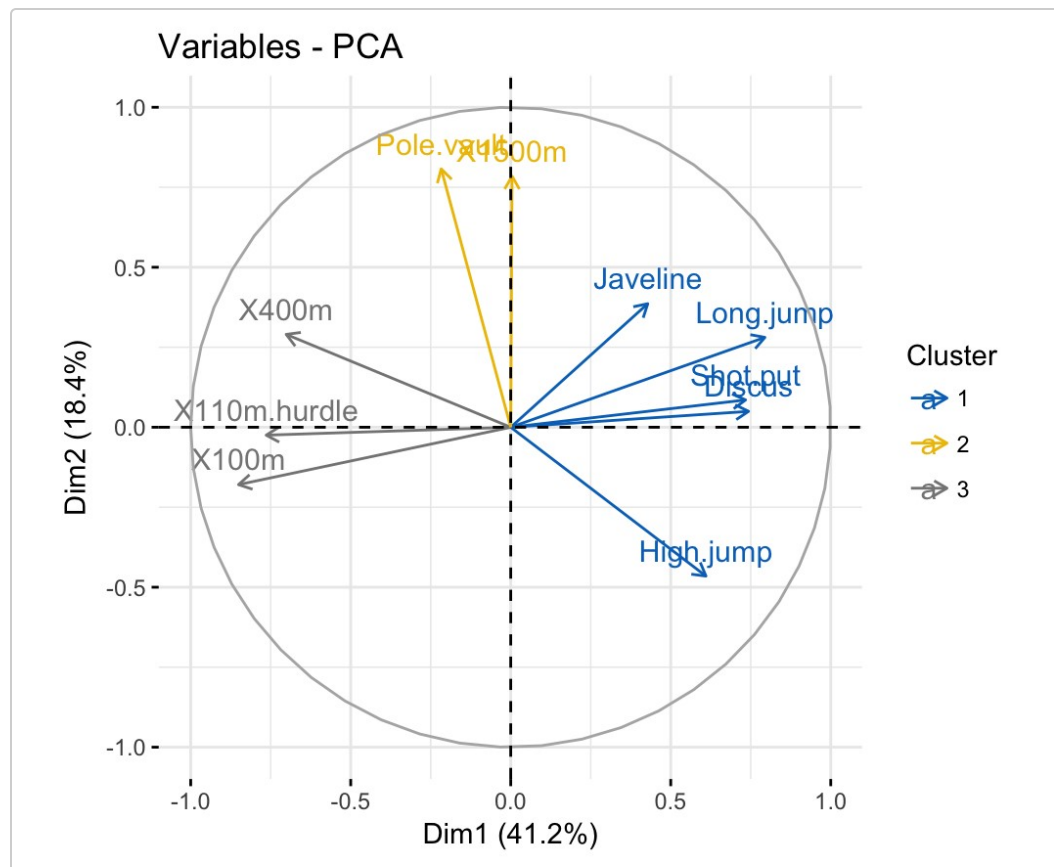
It's also possible to change the color of variables by groups defined by a qualitative/categorical variable, also called **factor** in R terminology.

As we don't have any grouping variable in our data sets for classifying variables, we'll create it.

In the following demo example, we start by classifying the variables into 3 groups using the **kmeans** clustering algorithm. Next, we use the clusters returned by the kmeans algorithm to color variables.

! Note that, if you are interested in learning clustering, we previously published a book named "Practical Guide To Cluster Analysis in R" (<https://goo.gl/DmJ5y5>).

```
# Create a grouping variable using kmeans
# Create 3 groups of variables (centers = 3)
set.seed(123)
res.km <- kmeans(var$coord, centers = 3, nstart = 25)
grp <- as.factor(res.km$cluster)
# Color variables by groups
fviz_pca_var(res.pca, col.var = grp,
             palette = c("#0073C2FF", "#EFC000FF", "#868686FF"),
             legend.title = "Cluster")
```



! Note that, to change the color of groups the argument palette should be used. To change gradient colors, the argument gradient.cols should be used.

Dimension description

In the section [@ref\(pca-variable-contributions\)](#), we described how to highlight variables according to their contributions to the principal components.

Note also that, the function `dimdesc()` [in FactoMineR], for dimension description, can be used to identify the most significantly associated variables with a given principal component . It can be used as follow:

```
res.desc <- dimdesc(res.pca, axes = c(1,2), proba = 0.05)
# Description of dimension 1
res.desc$Dim.1
```

```
## $quanti
##           correlation  p.value
## Long.jump      0.794 6.06e-06
## Discus         0.743 4.84e-05
## Shot.put       0.734 6.72e-05
## High.jump      0.610 1.99e-03
## Javeline       0.428 4.15e-02
## X400m          -0.702 1.91e-04
## X110m.hurdle   -0.764 2.20e-05
## X100m          -0.851 2.73e-07
```

```
# Description of dimension 2
res.desc$Dim.2
```

```
## $quanti
##           correlation  p.value
## Pole.vault      0.807 3.21e-06
## X1500m          0.784 9.38e-06
## High.jump       -0.465 2.53e-02
```

In the output above, *\$quanti* means results for quantitative variables. Note that, variables are sorted by the p-value of the correlation.

Graph of individuals

Results

The results, for individuals can be extracted using the function `get_pca_ind()` [*factoextra* package]. Similarly to the `get_pca_var()`, the function `get_pca_ind()` provides a list of matrices containing all the results for the individuals (coordinates, correlation between individuals and axes, squared cosine and contributions)

```
ind <- get_pca_ind(res.pca)
ind
```

```
## Principal Component Analysis Results for individuals
## =====
##   Name      Description
## 1 "$coord"   "Coordinates for the individuals"
## 2 "$cos2"    "Cos2 for the individuals"
## 3 "$contrib" "contributions of the individuals"
```

To get access to the different components, use this:

```
# Coordinates of individuals
head(ind$coord)
# Quality of individuals
head(ind$cos2)
# Contributions of individuals
head(ind$contrib)
```

Plots: quality and contribution

The `fviz_pca_ind()` is used to produce the graph of individuals. To create a simple plot, type this:

```
fviz_pca_ind(res.pca)
```

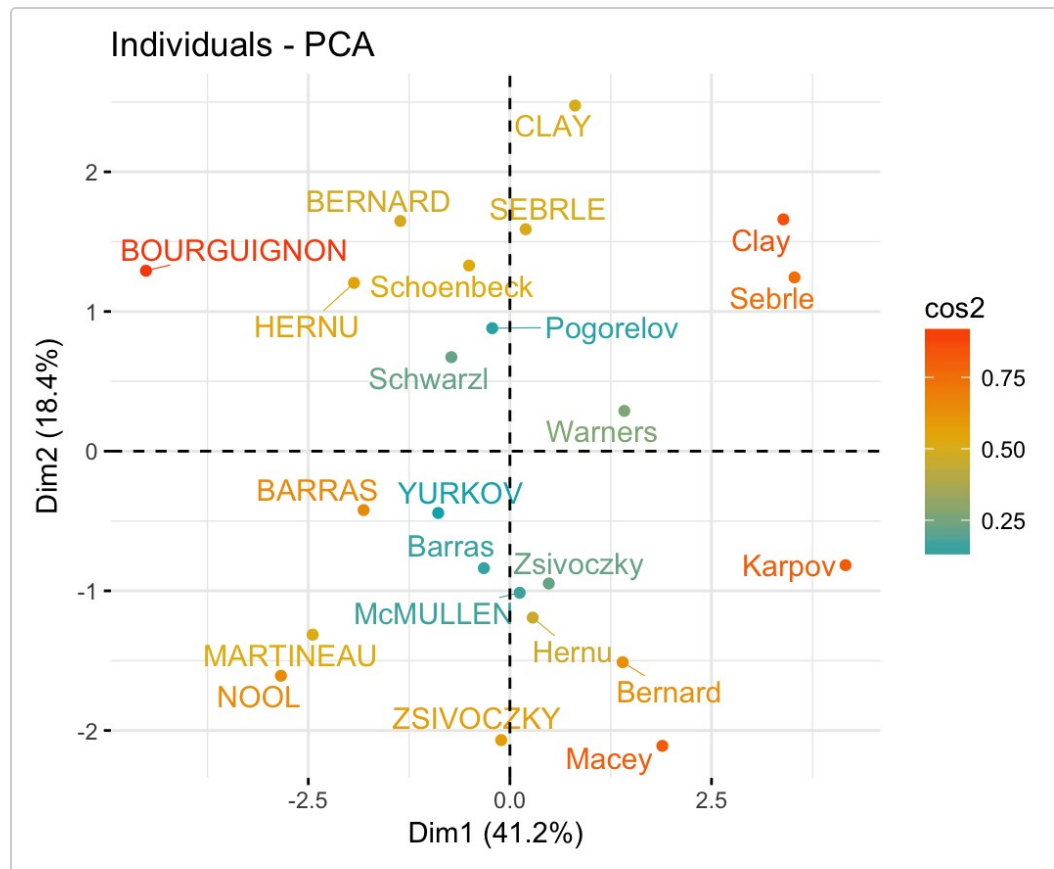
Like variables, it's also possible to color individuals by their cos2 values:

```
fviz_pca_ind(res.pca, col.ind = "cos2",
              gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
```

```

repel = TRUE # Avoid text overlapping (slow if many points)
)

```



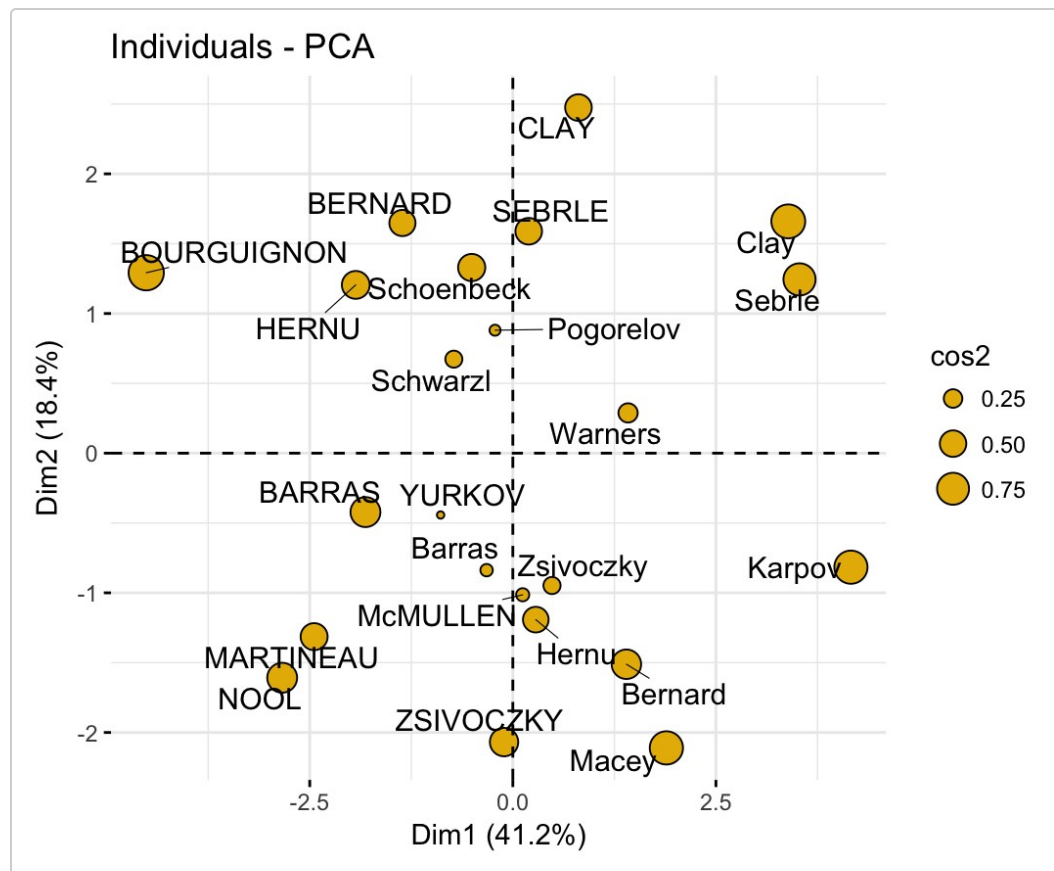
Note that, individuals that are similar are grouped together on the plot.

You can also change the point size according the cos2 of the corresponding individuals:

```

fviz_pca_ind(res.pca, pointsize = "cos2",
  pointshape = 21, fill = "#E7B800",
  repel = TRUE # Avoid text overlapping (slow if many points)
)

```

To change both point size and color by $\cos 2$, try this:

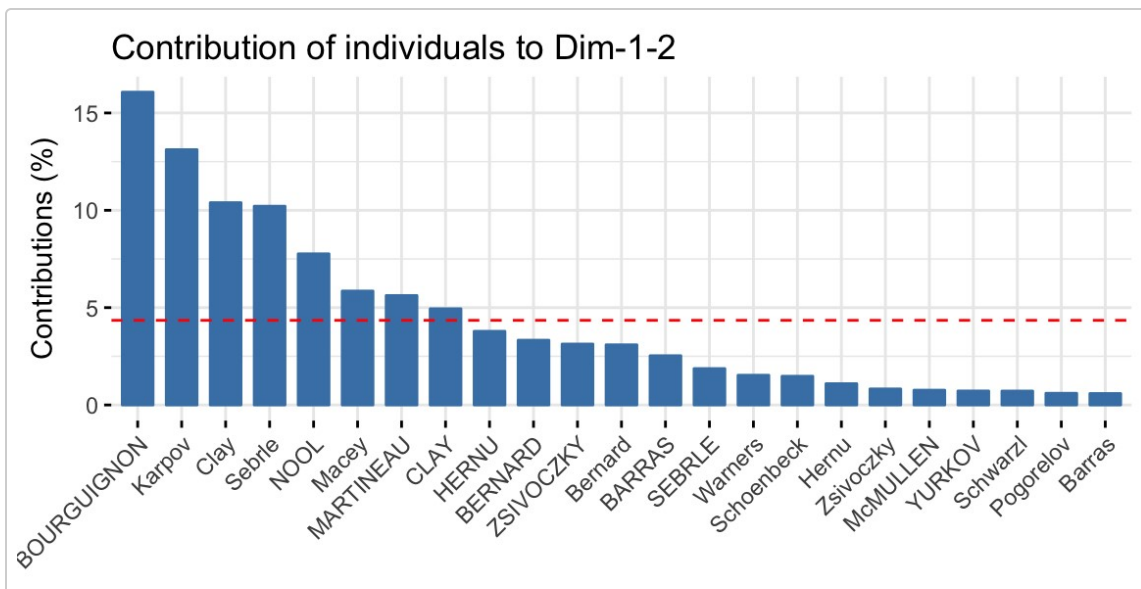
```
fviz_pca_ind(res.pca, col.ind = "cos2", pointsize = "cos2",
  gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
  repel = TRUE # Avoid text overlapping (slow if many points)
)
```

To create a bar plot of the quality of representation (\cos^2) of individuals on the factor map, you can use the function `fviz_cos2()` as previously described for variables:

```
fviz_cos2(res.pca, choice = "ind")
```

To visualize the contribution of individuals to the first two principal components, type this:

```
# Total contribution on PC1 and PC2
fviz_contrib(res.pca, choice = "ind", axes = 1:2)
```



Color by a custom continuous variable

As for variables, individuals can be colored by any custom continuous variable by specifying the argument `col.ind`.

For example, type this:

```
# Create a random continuous variable of length 23,
# Same length as the number of active individuals in the PCA
set.seed(123)
my.cont.var <- rnorm(23)
# Color individuals by the continuous variable
fviz_pca_ind(res.pca, col.ind = my.cont.var,
             gradient.cols = c("blue", "yellow", "red"),
             legend.title = "Cont.Var")
```

Color by groups

Here, we describe how to color individuals by group. Additionally, we show how to add *concentration ellipses* and *confidence ellipses* by groups. For this, we'll use the iris data as demo data sets.

Iris data sets look like this:

```
head(iris, 3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1         5.1         3.5         1.4         0.2   setosa
```

```
## 2          4.9          3.0          1.4          0.2  setosa
## 3          4.7          3.2          1.3          0.2  setosa
```

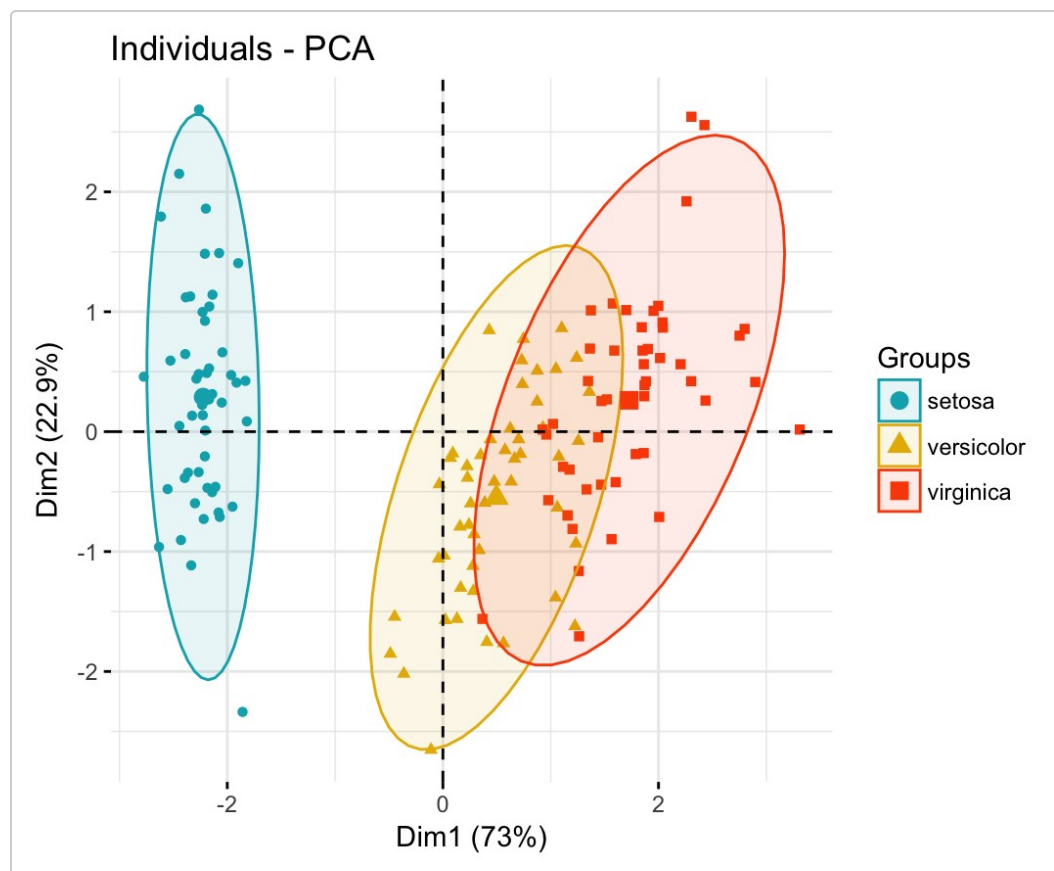
The column "Species" will be used as grouping variable. We start by computing principal component analysis as follow:

```
# The variable Species (index = 5) is removed
# before PCA analysis
iris.pca <- PCA(iris[,-5], graph = FALSE)
```

In the R code below: the argument `habillage` or `col.ind` can be used to specify the factor variable for coloring the individuals by groups.

To add a concentration ellipse around each group, specify the argument `addEllipses = TRUE`. The argument `palette` can be used to change group colors.

```
fviz_pca_ind(iris.pca,
  geom.ind = "point", # show points only (nbut not "text")
  col.ind = iris$Species, # color by groups
  palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  addEllipses = TRUE, # Concentration ellipses
  legend.title = "Groups"
)
```



To remove the group mean point, specify the argument *mean.point* = *FALSE*.

If you want confidence ellipses instead of concentration ellipses, use *ellipse.type* = "confidence".

```
# Add confidence ellipses
fviz_pca_ind(iris.pca, geom.ind = "point", col.ind = iris$Species,
  palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  addEllipses = TRUE, ellipse.type = "confidence",
  legend.title = "Groups"
)
```

Note that, allowed values for palette include:

- "grey" for grey color palettes;
- brewer palettes e.g. "RdBu", "Blues", ...; To view all, type this in R: `RColorBrewer::display.brewer.all()`.
- custom color palette e.g. `c("blue", "red")`;
- and scientific journal palettes from `ggsci` R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".

For example, to use the jco (journal of clinical oncology) color palette, type this:

```
fviz_pca_ind(iris.pca,
  label = "none", # hide individual labels
  habillage = iris$Species, # color by groups
  addEllipses = TRUE, # Concentration ellipses
  palette = "jco"
)
```

Graph customization

Note that, `fviz_pca_ind()` and `fviz_pca_var()` and related functions are wrapper around the core function `fviz()` [in *factoextra*]. `fviz()` is a wrapper around the function `ggscatter()` [in *ggpubr*]. Therefore, further arguments, to be passed to the function `fviz()` and `ggscatter()`, can be specified in `fviz_pca_ind()` and `fviz_pca_var()`.

Here, we present some of these additional arguments to customize the PCA graph of variables and individuals.

Dimensions

By default, variables/individuals are represented on dimensions 1 and 2. If you want to visualize them on dimensions 2 and 3, for example, you should specify the argument `axes = c(2, 3)`.

```
# Variables on dimensions 2 and 3
fviz_pca_var(res.pca, axes = c(2, 3))
# Individuals on dimensions 2 and 3
fviz_pca_ind(res.pca, axes = c(2, 3))
```

Plot elements: point, text, arrow

The argument `geom` (for geometry) and derivatives are used to specify the geometry elements or graphical elements to be used for plotting.

1. `geom.var`: a text specifying the geometry to be used for plotting variables. Allowed values are the combination of `c("point", "arrow", "text")`.
 - Use `geom.var = "point"`, to show only points;
 - Use `geom.var = "text"` to show only text labels;
 - Use `geom.var = c("point", "text")` to show both points and text labels
 - Use `geom.var = c("arrow", "text")` to show arrows and labels (default).

For example, type this:

```
# Show variable points and text labels
fviz_pca_var(res.pca, geom.var = c("point", "text"))
```

2. **geom.ind**: a text specifying the geometry to be used for plotting individuals. Allowed values are the combination of `c("point", "text")`.

- Use **geom.ind** = "point", to show only points;
- Use **geom.ind** = "text" to show only text labels;
- Use **geom.ind** = `c("point", "text")` to show both point and text labels (default)

For example, type this:

```
# Show individuals text labels only
fviz_pca_ind(res.pca, geom.ind = "text")
```

Size and shape of plot elements

1. **labelsize**: font size for the text labels, e.g.: **labelsize** = 4.
2. **pointsize**: the size of points, e.g.: **pointsize** = 1.5.
3. **arrowsize**: the size of arrows. Controls the thickness of arrows, e.g.: **arrowsize** = 0.5.
4. **pointshape**: the shape of points, **pointshape** = 21. Type `ggpubr::show_point_shapes()` to see available point shapes.

```
# Change the size of arrows and labels
fviz_pca_var(res.pca, arrowsize = 1, labelsize = 5,
             repel = TRUE)
# Change points size, shape and fill color
# Change labelsize
fviz_pca_ind(res.pca,
             pointsize = 3, pointshape = 21, fill = "lightblue",
             labelsize = 5, repel = TRUE)
```

Ellipses

As we described in the previous section `@ref(color-ind-by-groups)`, when coloring individuals by groups, you can add point concentration ellipses using the argument **addEllipses** = TRUE.

Note that, the argument **ellipse.type** can be used to change the type of ellipses. Possible values are:

- **"convex"**: plot convex hull of a set of points.

- **"confidence"**: plot confidence ellipses around group mean points as the function `coord.ellipse()` [in FactoMineR].
- **"t"**: assumes a multivariate t-distribution.
- **"norm"**: assumes a multivariate normal distribution.
- **"euclid"**: draws a circle with the radius equal to level, representing the euclidean distance from the center. This ellipse probably won't appear circular unless `coord_fixed()` is applied.

The argument `ellipse.level` is also available to change the size of the concentration ellipse in normal probability. For example, specify `ellipse.level = 0.95` or `ellipse.level = 0.66`.

```
# Add confidence ellipses
fviz_pca_ind(iris.pca, geom.ind = "point",
             col.ind = iris$Species, # color by groups
             palette = c("#00AFBB", "#E7B800", "#FC4E07"),
             addEllipses = TRUE, ellipse.type = "confidence",
             legend.title = "Groups"
            )

# Convex hull
fviz_pca_ind(iris.pca, geom.ind = "point",
             col.ind = iris$Species, # color by groups
             palette = c("#00AFBB", "#E7B800", "#FC4E07"),
             addEllipses = TRUE, ellipse.type = "convex",
             legend.title = "Groups"
            )
```

Group mean points

When coloring individuals by groups (section `@ref(color-ind-by-groups)`), the mean points of groups (barycenters) are also displayed by default.

To remove the mean points, use the argument `mean.point = FALSE`.

```
fviz_pca_ind(iris.pca,
             geom.ind = "point", # show points only (but not "text")
             group.ind = iris$Species, # color by groups
             legend.title = "Groups",
             mean.point = FALSE)
```

Axis lines

The argument `axes.linetype` can be used to specify the line type of axes. Default is "dashed". Allowed values include "blank", "solid", "dotted", etc. To see all possible values type `ggpubr::show_line_types()` in R.

To remove axis lines, use `axes.linetype = "blank"`:

```
fviz_pca_var(res.pca, axes.linetype = "blank")
```

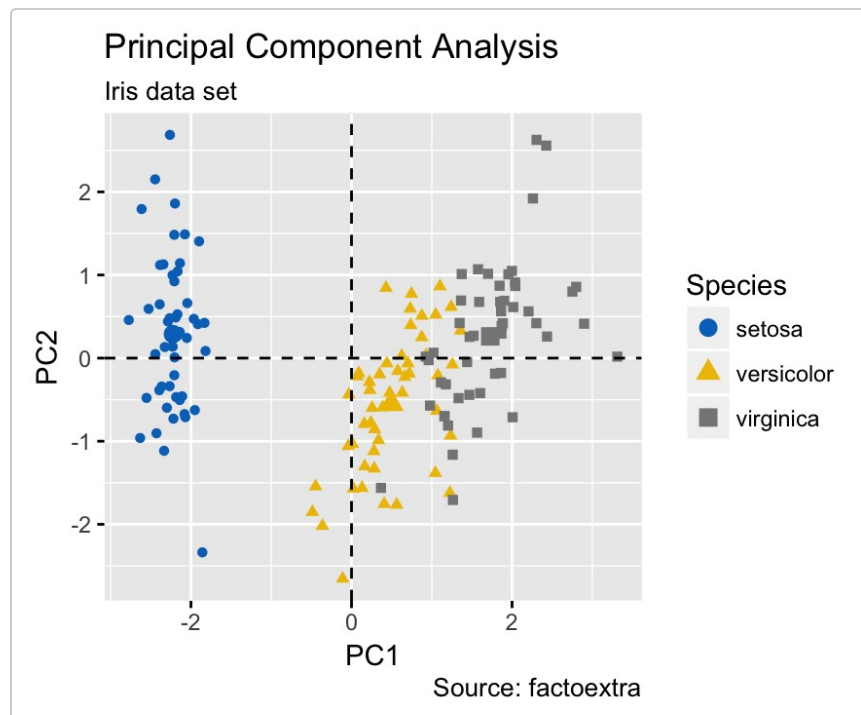
Graphical parameters

To change easily the graphical of any ggplots, you can use the function `ggpar()` [ggpubr package]

The graphical parameters that can be changed using `ggpar()` include:

- Main titles, axis labels and legend titles
- Legend position. Possible values: "top", "bottom", "left", "right", "none".
- Color palette.
- Themes. Allowed values include: `theme_gray()`, `theme_bw()`, `theme_minimal()`, `theme_classic()`, `theme_void()`.

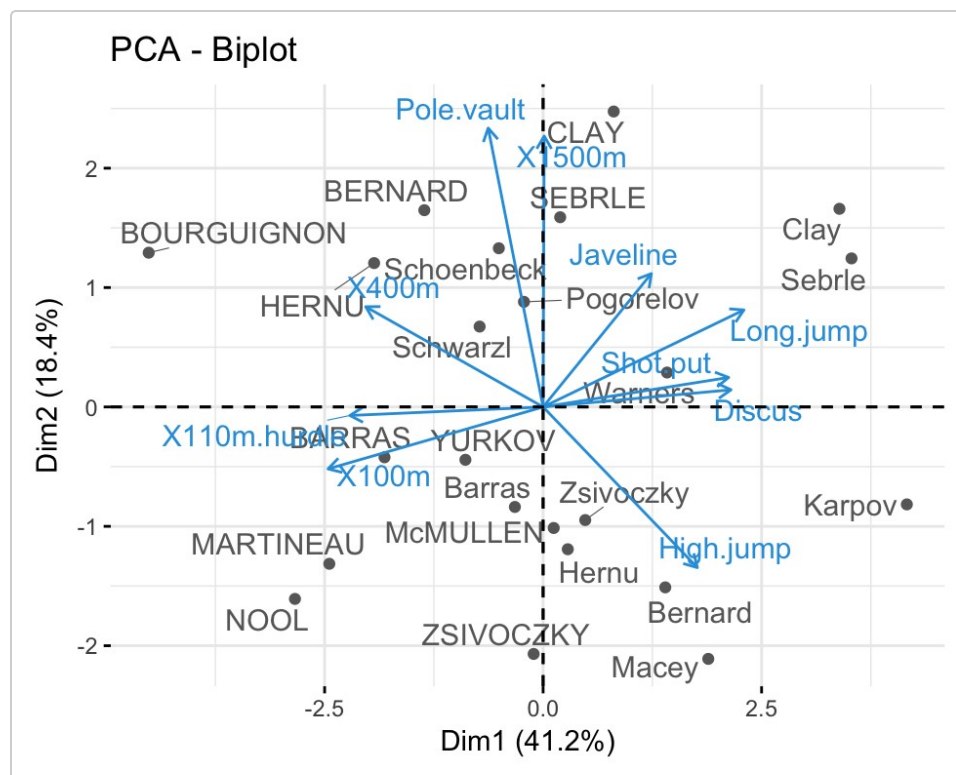
```
ind.p <- fviz_pca_ind(iris.pca, geom = "point", col.ind = iris$Species)
ggpubr::ggpar(ind.p,
  title = "Principal Component Analysis",
  subtitle = "Iris data set",
  caption = "Source: factoextra",
  xlab = "PC1", ylab = "PC2",
  legend.title = "Species", legend.position = "top",
  ggtheme = theme_gray(), palette = "jco"
)
```

Biplot

To make a simple biplot of individuals and variables, type this:

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



Note that, the biplot might be only useful when there is a low number of variables and individuals in the data set; otherwise the final plot would be unreadable.

Note also that, the coordinate of individuals and variables are not constructed on the same space. Therefore, in the biplot, you should mainly focus on the direction of variables but not on their absolute positions on the plot.

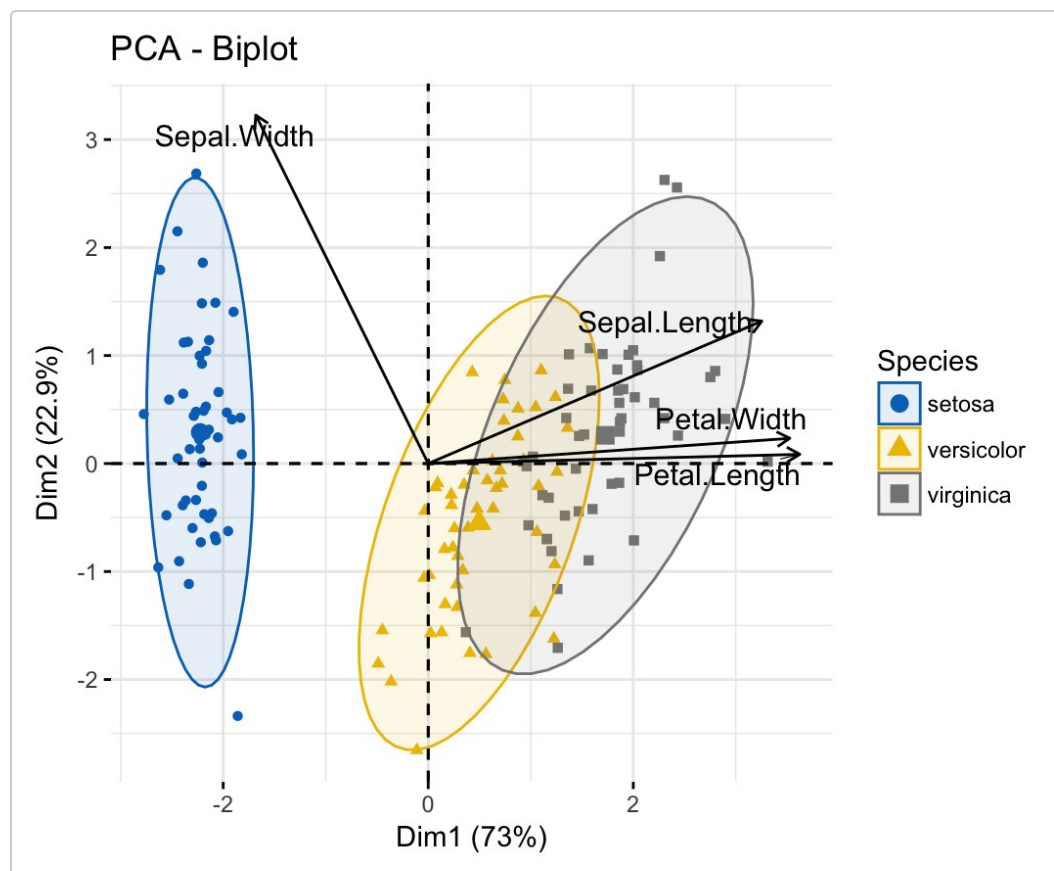
Roughly speaking a biplot can be interpreted as follow:

- an individual that is on the same side of a given variable has a high value for this variable;
- an individual that is on the opposite side of a given variable has a low value for this variable.

Now, using the `iris.pca` output, let's :

- make a biplot of individuals and variables
- change the color of individuals by groups: `col.ind = iris$Species`
- show only the labels for variables: `label = "var"` or use `geom.ind = "point"`

```
fviz_pca_biplot(iris.pca,
  col.ind = iris$Species, palette = "jco",
  addEllipses = TRUE, label = "var",
  col.var = "black", repel = TRUE,
  legend.title = "Species")
```

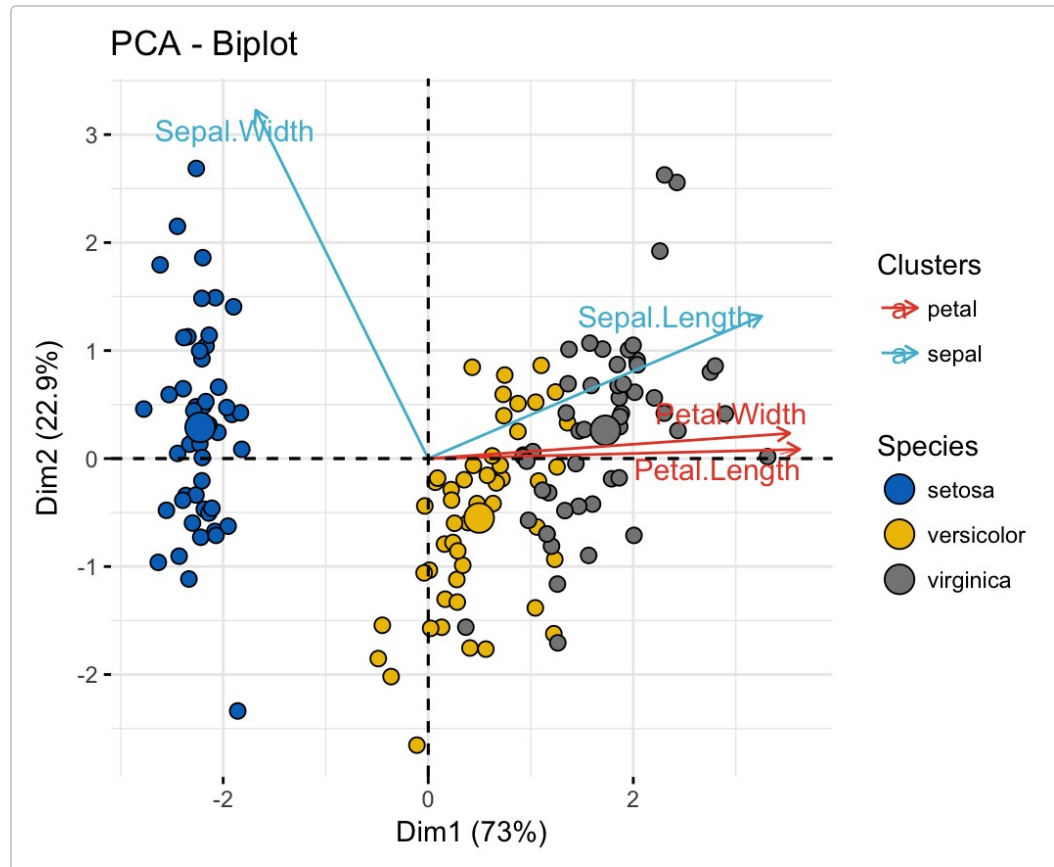


In the following example, we want to color both individuals and variables by groups. The trick is to use `pointshape = 21` for individual points. This particular point shape can be filled by a color using the argument `fill.ind`. The border line color of individual points is set to "black" using `col.ind`. To color variable by groups, the argument `col.var` will be used.

To customize individuals and variable colors, we use the helper functions `fill_palette()` and `color_palette()` [in ggpubr package].

```
fviz_pca_biplot(iris.pca,
  # Fill individuals by groups
  geom.ind = "point",
  pointshape = 21,
  pointsize = 2.5,
  fill.ind = iris$Species,
  col.ind = "black",
  # Color variable by groups
  col.var = factor(c("sepal", "sepal", "petal", "petal")),

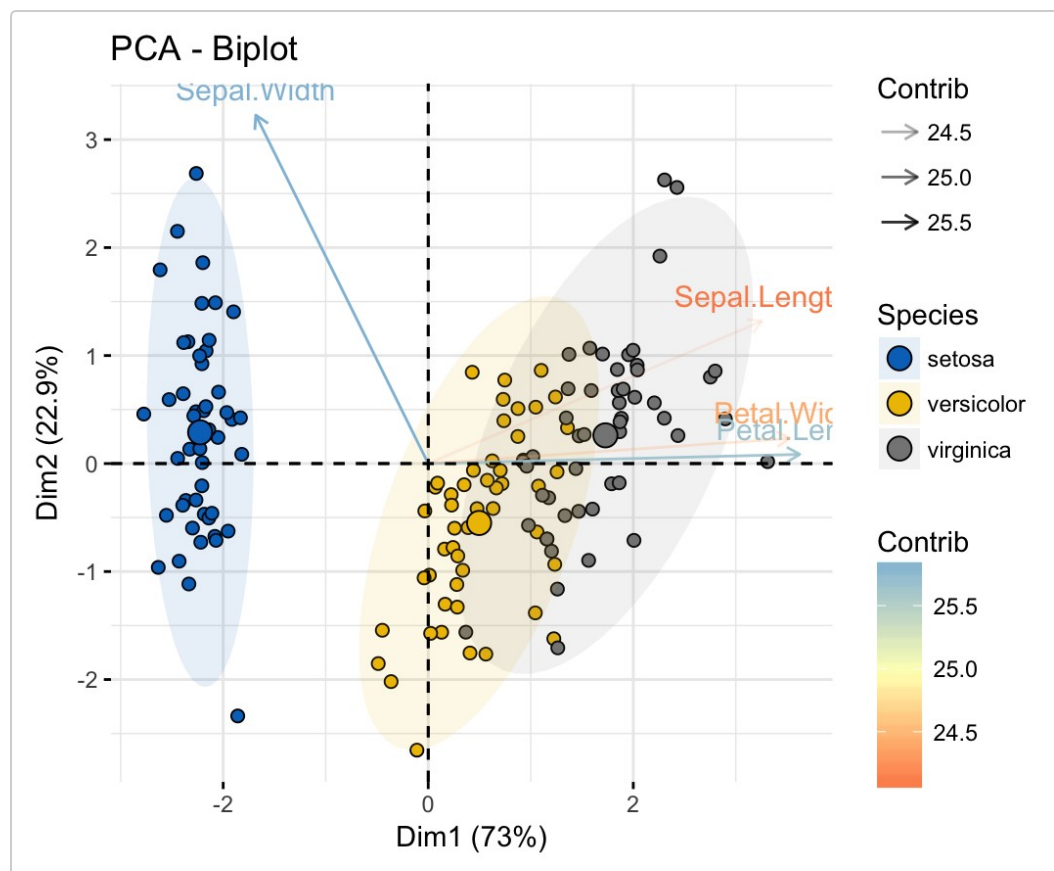
  legend.title = list(fill = "Species", color = "Clusters"),
  repel = TRUE      # Avoid label overplotting
)+
ggpubr::fill_palette("jco")+      # Individual fill color
ggpubr::color_palette("npg")      # Variable colors
```



Another complex example is to color individuals by groups (discrete color) and variables by their contributions to the principal components (gradient colors). Additionally, we'll change the transparency of variables by their contributions using the argument `alpha.var`.

```
fviz_pca_biplot(iris.pca,
  # Individuals
  geom.ind = "point",
  fill.ind = iris$Species, col.ind = "black",
  pointshape = 21, pointsize = 2,
  palette = "jco",
  addEllipses = TRUE,
  # Variables
  alpha.var = "contrib", col.var = "contrib",
  gradient.cols = "RdYlBu",

  legend.title = list(fill = "Species", color = "Contrib",
    alpha = "Contrib")
)
```



Supplementary elements

Definition and types

As described above (section @ref(pca-data-format)), the **deathlon2** data sets contain *supplementary continuous variables* (quanti.sup, columns 11:12), *supplementary qualitative variables* (quali.sup, column 13) and *supplementary individuals* (ind.sup, rows 24:27).

Supplementary variables and individuals are not used for the determination of the principal components. Their coordinates are predicted using only the information provided by the performed principal component analysis on active variables/individuals.

Specification in PCA

To specify supplementary individuals and variables, the function `PCA()` can be used as follow:

```
PCA(X, ind.sup = NULL,
    quanti.sup = NULL, quali.sup = NULL, graph = TRUE)
```

- **X** : a data frame. Rows are individuals and columns are numeric variables.
- **ind.sup** : a numeric vector specifying the indexes of the supplementary individuals
- **quanti.sup**, **quali.sup** : a numeric vector specifying, respectively, the indexes of the quantitative and qualitative variables
- **graph** : a logical value. If TRUE a graph is displayed.

For example, type this:

```
res.pca <- PCA(decathlon2, ind.sup = 24:27,
               quanti.sup = 11:12, quali.sup = 13, graph=FALSE)
```

Quantitative variables

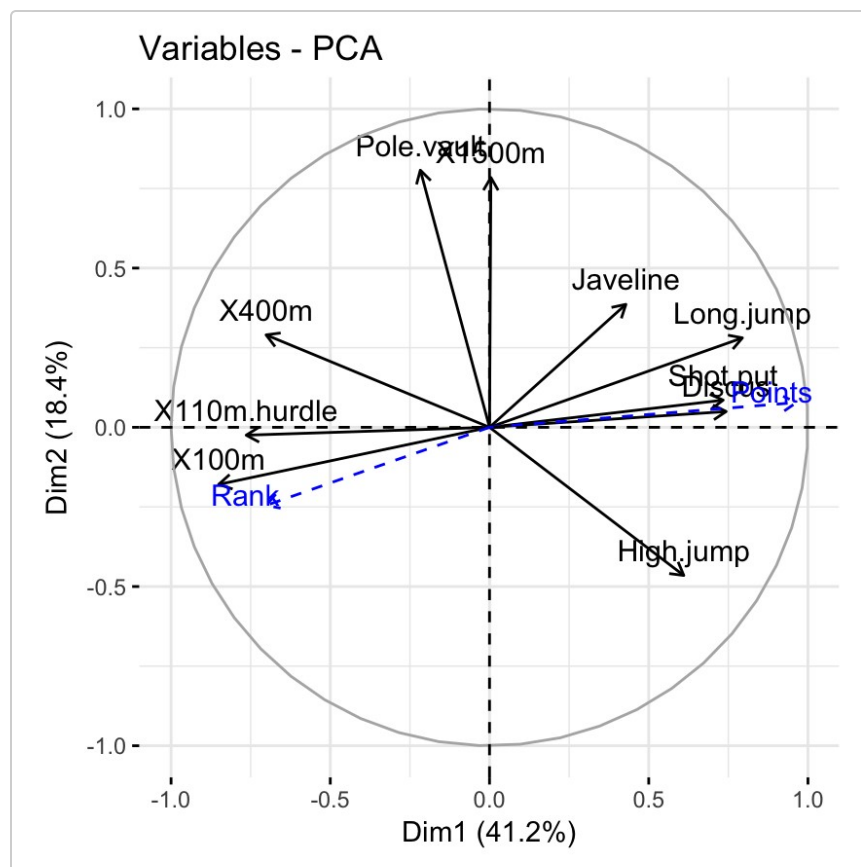
- Predicted results (coordinates, correlation and cos2) for the supplementary quantitative variables:

```
res.pca$quanti.sup
```

```
## $coord
##      Dim.1  Dim.2  Dim.3  Dim.4  Dim.5
## Rank    -0.701 -0.2452 -0.183  0.0558 -0.0738
## Points   0.964  0.0777  0.158 -0.1662 -0.0311
##
## $cor
##      Dim.1  Dim.2  Dim.3  Dim.4  Dim.5
## Rank    -0.701 -0.2452 -0.183  0.0558 -0.0738
## Points   0.964  0.0777  0.158 -0.1662 -0.0311
##
## $cos2
##      Dim.1  Dim.2  Dim.3  Dim.4  Dim.5
## Rank    0.492 0.06012 0.0336 0.00311 0.00545
## Points  0.929 0.00603 0.0250 0.02763 0.00097
```

- Visualize all variables (active and supplementary ones):

```
fviz_pca_var(res.pca)
```



✓ Note that, by default, supplementary quantitative variables are shown in blue color and dashed lines.

Further arguments to customize the plot:

```
# Change color of variables
fviz_pca_var(res.pca,
             col.var = "black",      # Active variables
             col.quant.sup = "red" # Suppl. quantitative variables
            )

# Hide active variables on the plot,
# show only supplementary variables
fviz_pca_var(res.pca, invisible = "var")

# Hide supplementary variables
fviz_pca_var(res.pca, invisible = "quanti.sup")
```

⚠ Using the `fviz_pca_var()`, the quantitative supplementary variables are displayed automatically on the correlation circle plot. Note that, you can add the `quanti.sup` variables manually, using the `fviz_add()` function, for further customization. An example is shown below.

```
# Plot of active variables
p <- fviz_pca_var(res.pca, invisible = "quanti.sup")
# Add supplementary active variables
fviz_add(p, res.pca$quanti.sup$coord,
         geom = c("arrow", "text"),
         color = "red")
```

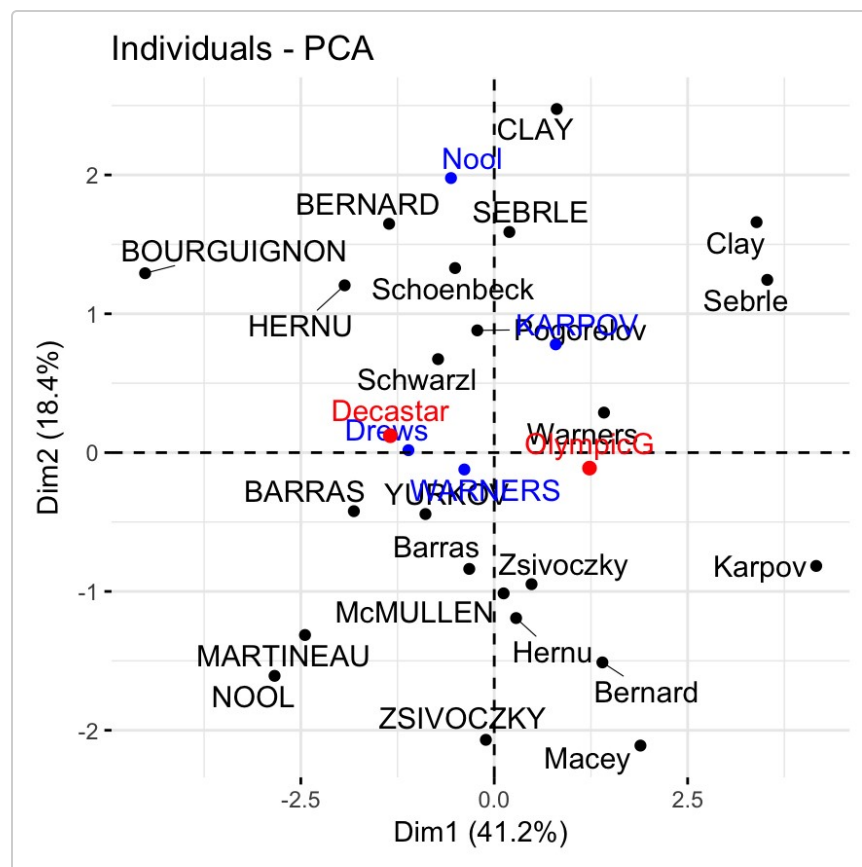
Individuals

- Predicted results for the supplementary individuals (**ind.sup**):

```
res.pca$ind.sup
```

- Visualize all individuals (active and supplementary ones). On the graph, you can add also the supplementary qualitative variables (**quali.sup**), which coordinates is accessible using *res.pca\$quali.sup\$coord*.

```
p <- fviz_pca_ind(res.pca, col.ind.sup = "blue", repel = TRUE)
p <- fviz_add(p, res.pca$quali.sup$coord, color = "red")
p
```

Supplementary individuals are shown in blue. The levels of the supplementary qualitative variable are shown in red color.

Qualitative variables

In the previous section, we showed that you can add the supplementary qualitative variables on individuals plot using `fviz_add()`.

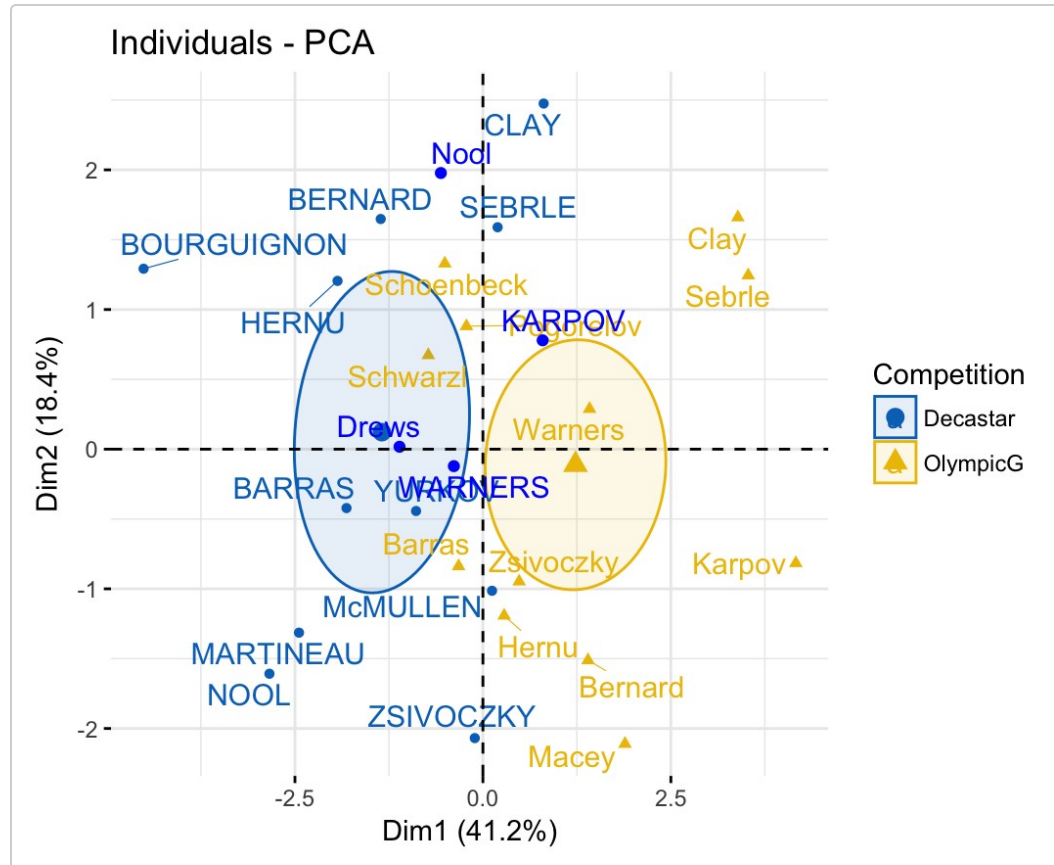
Note that, the supplementary qualitative variables can be also used for coloring individuals by groups. This can help to interpret the data. The data sets `decathlon2` contain a *supplementary qualitative variable* at columns 13 corresponding to the type of competitions.

The results concerning the supplementary qualitative variable are:

```
res.pca$quali
```

To color individuals by a supplementary qualitative variable, the argument `habillage` is used to specify the index of the supplementary qualitative variable. Historically, this argument name comes from the FactoMineR package. It's a french word meaning "dressing" in english. To keep consistency between FactoMineR and factoextra, we decided to keep the same argument name

```
fviz_pca_ind(res.pca, habillage = 13,
             addEllipses = TRUE, ellipse.type = "confidence",
             palette = "jco", repel = TRUE)
```



❗ Recall that, to remove the mean points of groups, specify the argument *mean.point* = *FALSE*.

Filtering results

If you have many individuals/variable, it's possible to visualize only some of them using the arguments *select.ind* and *select.var*.

select.ind, *select.var*: a selection of individuals/variable to be plotted. Allowed values are *NULL* or a *list* containing the arguments name, *cos2* or *contrib*:

- *name*: is a character vector containing individuals/variable names to be plotted
- *cos2*: if *cos2* is in $[0, 1]$, ex: 0.6, then individuals/variables with a *cos2* > 0.6 are plotted
- if *cos2* > 1, ex: 5, then the top 5 active individuals/variables and top 5 supplementary columns/rows with the highest *cos2* are plotted
- *contrib*: if *contrib* > 1, ex: 5, then the top 5 individuals/variables with the highest contributions are plotted

```
# Visualize variable with cos2 >= 0.6
fviz_pca_var(res.pca, select.var = list(cos2 = 0.6))
# Top 5 active variables with the highest cos2
fviz_pca_var(res.pca, select.var= list(cos2 = 5))
# Select by names
name <- list(name = c("Long.jump", "High.jump", "X100m"))
fviz_pca_var(res.pca, select.var = name)
# top 5 contributing individuals and variable
fviz_pca_biplot(res.pca, select.ind = list(contrib = 5),
                select.var = list(contrib = 5),
                ggtheme = theme_minimal())
```



When the selection is done according to the contribution values, supplementary individuals/variables are not shown because they don't contribute to the construction of the axes.

Exporting results

Export plots to PDF/PNG files

The **factoextra** package produces a ggplot2-based graphs. To save any ggplots, the standard R code is as follow:

```
# Print the plot to a pdf file
pdf("myplot.pdf")
print(myplot)
dev.off()
```

In the following examples, we'll show you how to save the different graphs into pdf or png files.

The first step is to create the plots you want as an R object:

```
# Scree plot
scree.plot <- fviz_eig(res.pca)
# Plot of individuals
ind.plot <- fviz_pca_ind(res.pca)
# Plot of variables
var.plot <- fviz_pca_var(res.pca)
```

Next, the plots can be exported into a single pdf file as follow:

```
pdf("PCA.pdf") # Create a new pdf device
print(scree.plot)
print(ind.plot)
print(var.plot)
dev.off() # Close the pdf device
```



Note that, using the above R code will create the PDF file into your current working directory. To see the path of your current working directory, type `getwd()` in the R console.

To print each plot to specific png file, the R code looks like this:

```
# Print scree plot to a png file
png("pca-scree-plot.png")
print(scree.plot)
dev.off()
# Print individuals plot to a png file
png("pca-variables.png")
print(var.plot)
dev.off()
# Print variables plot to a png file
png("pca-individuals.png")
print(ind.plot)
dev.off()
```

Another alternative, to export ggplots, is to use the function `ggexport()` [in ggpubr package]. We like `ggexport()`, because it's very simple. With one line R code, it allows us to export individual plots to a file (pdf, eps or png) (one plot per page). It can also arrange the plots (2 plot per page, for example) before exporting them. The examples below demonstrates how to export ggplots using `ggexport()`.

Export individual plots to a pdf file (one plot per page):

```
library(ggpubr)
ggexport(plotlist = list(scree.plot, ind.plot, var.plot),
         filename = "PCA.pdf")
```

Arrange and export. Specify `nrow` and `ncol` to display multiple plots on the same page:

```
ggexport(plotlist = list(scree.plot, ind.plot, var.plot),  
         nrow = 2, ncol = 2,  
         filename = "PCA.pdf")
```

Export plots to png files. If you specify a list of plots, then multiple png files will be automatically created to hold each plot.

```
ggexport(plotlist = list(scree.plot, ind.plot, var.plot),  
         filename = "PCA.png")
```

Export results to txt/csv files

All the outputs of the PCA (individuals/variables coordinates, contributions, etc) can be exported at once, into a TXT/CSV file, using the function `write.infile()` [in *FactoMineR*] package:

```
# Export into a TXT file  
write.infile(res.pca, "pca.txt", sep = "\t")  
# Export into a CSV file  
write.infile(res.pca, "pca.csv", sep = ";")
```

Summary

In conclusion, we described how to perform and interpret principal component analysis (PCA). We computed PCA using the **PCA()** function [FactoMineR]. Next, we used the **factoextra** R package to produce ggplot2-based visualization of the PCA results.

There are other functions [packages] to compute PCA in R:

1. Using **prcomp()** [stats]

```
res.pca <- prcomp(iris[, -5], scale. = TRUE)
```

Read more: <http://www.sthda.com/english/wiki/pca-using-prcomp-and-princomp>

2. Using **princomp()** [stats]

```
res.pca <- princomp(iris[, -5], cor = TRUE)
```

Read more: <http://www.sthda.com/english/wiki/pca-using-prcomp-and-princomp>

3. Using **dudi.pca()** [ade4]

```
library("ade4")
res.pca <- dudi.pca(iris[, -5], scannf = FALSE, nf = 5)
```

Read more: <http://www.sthda.com/english/wiki/pca-using-ade4-and-factoextra>

4. Using **epPCA()** [ExPosition]

```
library("ExPosition")
res.pca <- epPCA(iris[, -5], graph = FALSE)
```

No matter what functions you decide to use, in the list above, the factoextra package can handle the output for creating beautiful plots similar to what we described in the previous sections for FactoMineR:

```
fviz_eig(res.pca)      # Scree plot
fviz_pca_ind(res.pca) # Graph of individuals
fviz_pca_var(res.pca) # Graph of variables
```

Further reading

For the mathematical background behind CA, refer to the following video courses, articles and books:

- Principal component analysis (article) (Abdi and Williams 2010). <https://goo.gl/1Vtwq1>.
- Principal Component Analysis Course Using FactoMineR (Video courses). <https://goo.gl/VZjsnM>
- Exploratory Multivariate Analysis by Example Using R (book) (Husson, Le, and Pagès 2017).
- Principal Component Analysis (book) (Jolliffe 2002).

See also:

- PCA using **prcomp()** and **princomp()** (tutorial). <http://www.sthda.com/english/wiki/pca-using-prcomp-and-princomp>
- PCA using **ade4** and **factoextra** (tutorial). <http://www.sthda.com/english/wiki/pca-using-ade4-and-factoextra>

References

Abdi, Hervé, and Lynne J. Williams. 2010. "Principal Component Analysis." *John Wiley and Sons, Inc. WIREs Comp Stat* 2: 433–59. <http://staff.ustc.edu.cn/~zwp/teach/MVA/abdi-awPCA2010.pdf>.

Husson, Francois, Sebastien Le, and Jérôme Pagès. 2017. *Exploratory Multivariate Analysis by Example Using R*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. <http://factominer.free.fr/bookV2/index.html>.

Jolliffe, I.T. 2002. *Principal Component Analysis*. 2nd ed. New York: Springer-Verlag. <https://goo.gl/SB86SR>.

Kaiser, Henry F. 1961. "A Note on Guttman's Lower Bound for the Number of Common Factors." *British Journal of Statistical Psychology* 14: 1–2.

Peres-Neto, Pedro R., Donald A. Jackson, and Keith M. Somers. 2005. "How Many Principal Components? Stopping Rules for Determining the Number of Non-Trivial Axes Revisited." *British Journal of Statistical Psychology* 49: 974–97.

Last update : 24/09/2017

☆☆☆☆☆ 0 Note



Enjoyed this article? Give us 5 stars ★★★★★ (just above this text block)! I'd be very grateful if you'd help it spread by emailing it to a friend, or sharing it on Twitter, Facebook or Linked In.

Show me some love with the like buttons below... Thank you and please don't forget to share and comment below!!

AdChoices

PDF to PNG

Create a Graph

PDF Excel

Share 7

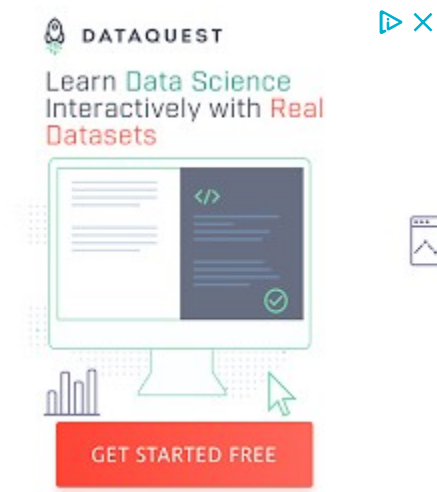
Like 7

Tweet

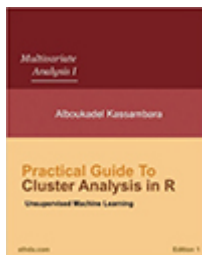
Share

0

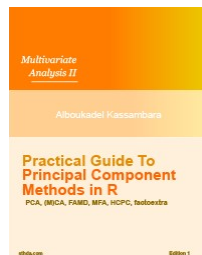
G+



Recommended for You!



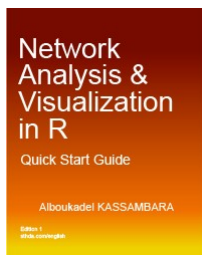
Practical Guide to Cluster Analysis in R



Practical Guide to Principal Component Methods in R



R Graphics Essentials for Great Data Visualization



Network Analysis and Visualization in R



More books on R and data science

The fields marked with a * are required !

Add a comment

Visitor

[illegible]

Preview

☐ I'm not a robot

reCAPTCHA
[Privacy](#) - [Terms](#)

Submit Reset

Visitor

Hi,

None of the above biplots for the Iris dataset work:

Examples:

```
fviz_pca_ind(iris.pca,  
geom.ind = "point", # show points only (nbut not "text")  
col.ind = iris$Species, # color by groups  
palette = c("#00AFBB", "#E7B800", "#FC4E07"),  
addEllipses = TRUE, # Concentration ellipses  
legend.title = "Groups"  
)
```

```
> Error: Aesthetics must be either length 1 or the same as the data (52): colour,
```

fill, alpha

```
fviz_pca_biplot(iris.pca,  
col.ind = iris$Species, palette = "jco",  
addEllipses = TRUE, label = "var",  
col.var = "black", repel = TRUE,  
legend.title = "Species")
```

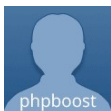
> **Error: Aesthetics must be either length 1 or the same as the data (52): colour, fill, alpha**

```
fviz_pca_biplot(iris.pca,  
# Fill individuals by groups  
geom.ind = "point",  
pointshape = 21,  
pointsize = 2.5,  
fill.ind = iris$Species,  
col.ind = "black",  
# Color variable by groups  
col.var = factor(c("sepal", "sepal", "petal", "petal")),
```

```
legend.title = list(fill = "Species", color = "Clusters"),  
repel = TRUE # Avoid label overplotting  
) +  
ggpubr::fill_palette("jco") + # Individual fill color  
ggpubr::color_palette("npg") # Variable colors
```

> **Error in grDevices::col2rgb(colour, TRUE) : invalid color name 'sepal'**

Is there something I'm doing wrong?



If not, please fix because these would be amazing plots for my analyses.

Thanks,

Gary

#255

Visitor 10/09/2017 at 14h23

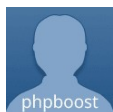
Visitor

I updated the Factoextra package to **version 1.0.5** and it now works..

Thanks!

Gary

#256

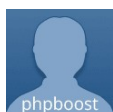


visitor 11/03/2017 at 19h42

Visitor

I am not sure this PCA work with time series data . You do not take the full account of eigenvalue when try to compute the scree plot.

#282



Visitor 11/07/2017 at 04h04

Visitor

Hi, I need to know hoe to get the eigenvectors for each eigenvalue! I want to obtained wich is the weight of each original variable in each new component. I mean, for example:

$FPC = a_{11}X_1 + a_{12}X_2 + a_{13}X_3 + \dots$

I need to obtained the a_i

thanks,

#287

kassambara 11/09/2017 at 05h49

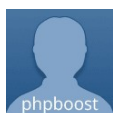
Administrator

Hi,

You need the loadings, they are the linear combination weights (coefficients).

Loadings (i.e. standard coordinates) are not given by FactoMineR's functions. They return only principal coordinates.

You can calculate them by dividing variables' coordinates on a dimension by this dimension's eigenvalue's square root.



For example, use the following R code:

Code R :

Copy to Clipboard

```
res.pca <- FactoMineR::PCA(iris[, -5], graph = FALSE)
sweep(
  res.pca$var$coord, 2,
  sqrt(res.pca$eig[1:ncol(res.pca$var$coord),1]),
  FUN="/"
)
```

You can read more about loadings and eigenvectors at:

<https://stats.stackexchange.com/questions/143905/loadings-vs-eigenvectors-in-pca-when-to-use-one-or-another>

#292



Visitor 11/24/2017 at 10h24

Visitor

Is there any similar package in python? Thanks!

#306

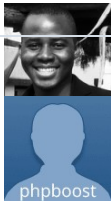


Visitor 12/07/2017 at 13h01

Visitor

Very nice

#315



Visitor 12/10/2017 at 03h38

Visitor

Hi, I have bought your book because I really appreciate you for creating such a simple yet powerful package in R. However, can you explain how mathematically you would count contribution of each variable to the given dimension?

Best

#317

kassambara 12/10/2017 at 16h39

Administrator

Hi,

For the mathematical background behind PCA and variable contributions, please refer to the video series available at:

[PCA in R Using FactoMineR: Quick Scripts and Videos](#)

Read also, the following PDF file:

[Principal component analysis](#)

#318



Visitor 01/08/2018 at 14h32

Visitor

Hi,

Thank you very much for your tutorial. It is very helpful.

Here I get one question, how could we change the font size on the xlab and y label and title?

Thanks

#342



kassambara 01/12/2018 at 22h54

Administrator

To change the graphical parameter of a ggplot, you can use the function ggpar() as described at this link <http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/82-ggplot2-easy-way-to-change-graphical-parameters/>

#346



Senthilnathan 03/09/2018 at 07h14

Visitor

Neat and clear explanation. Very nice....

#382

[Sign in](#)



[Login](#)




Password

Auto connect



 Register 

 [Forgotten password](#)

Welcome!

Want to Learn More on R Programming and Data Science?

Follow us [by Email](#)

[Subscribe](#)

by [FeedBurner](#)

on Social Networks

 AdChoices

PCA

Data Analysis R

PDF to PNG

Create a Graph

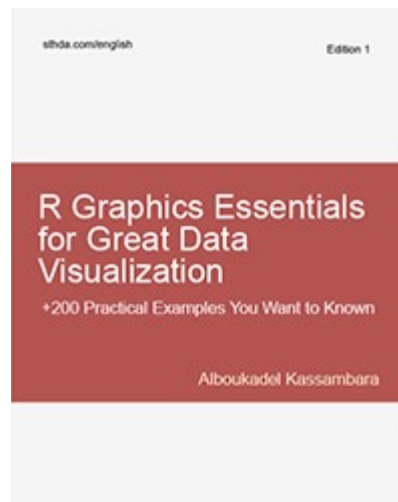
 **factoextra**

 **survminer**

-  `ggpubr`
-  `ggcorrplot`
-  `fastqcr`

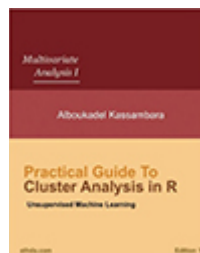


Our Books

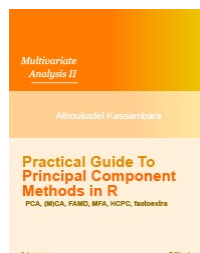


R Graphics Essentials for Great Data Visualization: 200 Practical Examples You Want to Know for Data Science

★ NEW!!

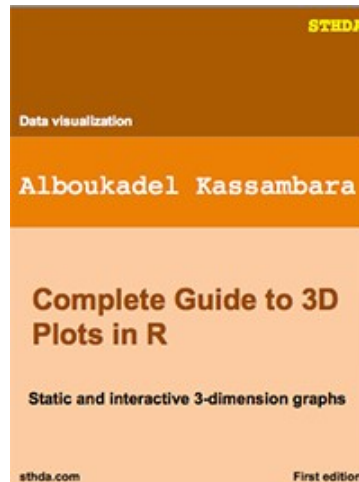


Practical Guide to Cluster Analysis in R



Practical Guide to Principal Component Methods in R

3D Plots in R



Guest Book

Very helpful guide for using the xlsx package.

R.H.

By *Visitor*

[Guest Book](#)

 **R-Bloggers**

Newsletter

Email



Boosted by [PHPBoost](#)

Recommended for you



Fuzzy Clustering Essentials - Articles - STHDA

www.sthda.com



ggplot2 boxplot : Easy box and whisker plots...

www.sthda.com



Matrice de corrélation: Guide simple pour an...

www.sthda.com

AddThis