Lab 3: Principal Components Analysis (PCA)

Stat 154, Spring 2018

Introduction

The goal of this lab is to go over the various options and steps required to perform a Principal Components Analysis (PCA). You will also learn about the functions `prcomp()` and `princomp()`, and how to use their outputs to answer questions like:

- How many principal components to retain.
- How to visualize the observations.
- How to visualize the relationships among variables.
- How to visualize supplementary variables.

Dataset NBA Teams

In this lab we are going to use the data set about NBA teams, containing statistics per game during the regular season 2016-2017. The corresponding CSV file is available in the `data/` folder of the github repo:

https://github.com/ucb-stat154/stat154-spring-2018/tree/master/data

```r
# assemble the url so it fits on rendered pdf
repo <- 'https://github.com/ucb-stat154/stat154-spring-2018/'
csv_file <- 'raw/master/data/nba-teams-2017.csv'
url <- paste0(repo, csv_file)

# download file to your working directory
# (note: the field separator is a colon ";")
download.file(url, destfile = 'nba-teams-2017.csv')
dataset <- read.csv('nba-teams-2017.csv', stringsAsFactors = FALSE)
```

The data contains 30 rows and 27 columns. If you look at the structure you'll get the following information:

```r
str(dataset, vec.len = 1)
```

```
'data.frame':   30 obs. of  27 variables:
 $ team                 : chr  "Golden State Warriors" ...
 $ games_played         : int  82 82 ...
 $ wins                 : int  67 61 ...
 $ losses               : int  15 21 ...
 $ win_prop            : num  0.817 0.744 ...
 $ minutes             : num  48.2 48.3 ...
 $ points              : num  116 ...
 $ field_goals         : num  43.1 39.3 ...
 $ field_goals_attempted: num  87.1 83.7 ...
 $ field_goals_prop    : num  49.5 46.9 ...
 $ points3             : num  12 9.2 ...
 $ points3_attempted   : num  31.2 23.5 ...
 $ points3_prop        : num  38.3 39.1 ...
 $ free_throws         : num  17.8 17.6 ...
 $ free_throws_att     : num  22.6 22 ...
 $ free_throws_prop    : num  78.8 79.7 ...
 $ off_rebounds        : num  9.4 10 ...
 $ def_rebounds        : num  35 33.9 ...
 $ rebounds            : num  44.4 43.9 ...
 $ assists             : num  30.4 23.8 ...
 $ turnovers           : num  14.8 13.4 ...
 $ steals              : num  9.6 8 ...
 $ blocks              : num  6.8 5.9 ...
 $ block_fga           : num  3.8 4.1 ...
 $ personal_fouls      : num  19.3 18.3 ...
 $ personal_fouls_drawn : num  19.4 19.8 ...
 $ plus_minus          : num  11.6 7.2 ...
```

Your turn

Create a new data frame `dat` that contains the following columns:

- `wins`
- `losses`
- `points`
- `field_goals`
- `points3`
- `free_throws`
- `off_rebounds`
- `def_rebounds`
- `assists`
- `steals`
- `blocks`
- `personal_fouls`

Spend some time examining things like:

- descriptive statistics with `summary()`.
- univariate plots: boxplots, histograms, density curves.

- compute the correlation matrix.
- get a scatterplot matrix with `pairs()`
- What do you see in each scatterplot (e.g. pattern, trend, variability)?

R Functions for PCA

R provides two built-in functions to perform Principal Components Analysis:

1. `prcomp()`
2. `princomp()`

Both functions are part of the `"stats"` package which comes in the default distribution of R. In addition to `prcomp()` and `princomp()`, there are various packages (e.g. `"FactoMineR"`, `"ade4"`, etc.) that provide other functions for PCA. In this lab, we will only discuss the base functions.

PCA with `prcomp()`

The main input of `prcomp()` is a data frame or a data matrix. In order to perform PCA on standardized data (mean = 0, variance = 1), we use the argument `scale. = TRUE`.

```r
# PCA with prcomp()
pca_prcomp <- prcomp(dat, scale. = TRUE)

# what's in a prcomp object?
names(pca_prcomp)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

The object `pca_prcomp` is an object of class `"prcomp"`, basically a list that contains the following results:

- `sdev` corresponds to the standard deviations of the principal components (i.e. the square roots of the eigenvalues of $\frac{1}{n-1}\mathbf{X}^\mathsf{T}\mathbf{X}$).
- `rotation` is the rotation matrix or loadings (i.e. eigenvectors of $\frac{1}{n-1}\mathbf{X}^\mathsf{T}\mathbf{X}$)
- `center` is the vector of means of the raw data (i.e. the centroid).
- `scale` is the vector of standard deviations of the raw data.
- `x` is the matrix of principal components (aka scores).

Note: recall that the variance of each principal component is: $var(\mathbf{z_k}) = \lambda_k$. This is why the square roots of the eigenvalues are in `sdev`: they are the *standard deviations* of the PCs: $\sqrt{\lambda_k} = sd(\mathbf{z_k})$

Your turn

As we saw in lecture, the minimal output of a PCA procedure should consists of eigenvalues, loadings, and principal components:

Create the following objects:

- `eigenvalues`: vector of eigenvalues (i.e. $\lambda_1, \lambda_2, \dots$)
- `loadings`: matrix of eigenvectors (i.e. $\mathbf{V}$)
- `scores`: matrix of principal components (i.e. $\mathbf{Z} = \mathbf{XV}$)

Note: *The signs of the columns of the loadings and scores are arbitrary, and so may differ between different programs for PCA, and even between different builds of R. Look around at the output of your neighbors to see who has similar results to yours, and who has different outputs.*

Quickly inspect the objects created above:

- How many eigenvalues are almost zero (or zero)?
- What about the loading associated to the 12th PC?
- What about the 12th PC score?
- Can you guess what's going on with the values of the 12th dimension?

Before discussing what to do with the output of a PCA, let's quickly talk about the other function `princomp()`.

PCA with `princomp()`

Another function to perform PCA is `princomp()`. The main input is a data frame or a data matrix. In order to perform PCA on standardized data (mean = 0, variance = 1), we use the argument `cor = TRUE`, which means that the analysis is performed using the correlation matrix.

```r
# PCA with princomp()
pca_princomp <- princomp(dat, cor = TRUE)

# what's in a princomp object?
names(pca_princomp)
```

```
## [1] "sdev"     "loadings" "center"   "scale"    "n.obs"    "scores"
## [7] "call"
```

The object `pca_princomp` is an object of class `"princomp"`, basically a list that contains various results:

- `sdev` corresponds to the standard deviations of the principal components.
- `loadings`: object of class `"loadings"`

- `center`: vector of variable means of the raw data.
- `scale`: vector of standard deviations of the raw data.
- `n.obs`: number of observations in the data.
- `scores`: matrix of principal components.
- `call`: function call.

Your turn

- Compare the results of `prcomp()` against those of `princomp()` in terms of eigenvalues, loadings, and PCs
- If you carefully look at the `princomp()` loadings, you should notice that some values are left in blank. Why is this? Check the documentation `?princomp`

Challenge: Notice that `pca_princomp$loadings` is an object of class `"loadings"`. How would you retrieve just the matrix of loadings—like the one below?

```
                Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  Comp.7
wins            0.3981  0.1560  0.1075 -0.1533  0.4684  0.1898  0.0437
losses         -0.3981 -0.1560 -0.1075  0.1533 -0.4684 -0.1898 -0.0437
points          0.4198 -0.1994 -0.3091  0.0793 -0.0955 -0.1133  0.0758
field_goals     0.3574 -0.2382  0.0417  0.2926  0.0067 -0.4097  0.1694
points3         0.2826  0.2424 -0.4385 -0.2644 -0.1632 -0.0797  0.3705
free_throws     0.1671 -0.3451 -0.4262 -0.0495 -0.0977  0.5288 -0.4868
off_rebounds   -0.0097 -0.4423  0.0125  0.4557  0.4428 -0.1334 -0.0144
def_rebounds    0.2130  0.2000 -0.1030  0.6015 -0.2664  0.2762  0.0253
assists         0.3690  0.0674  0.1212 -0.1025 -0.3300 -0.4009 -0.2959
steals          0.2079 -0.3679  0.3919 -0.3367 -0.1495 -0.0525 -0.2991
blocks          0.1964 -0.0513  0.5591  0.1359 -0.3344  0.4370  0.3273
personal_fouls -0.0890 -0.5466 -0.1185 -0.2773 -0.0758  0.1129  0.5500
                Comp.8  Comp.9 Comp.10 Comp.11 Comp.12
wins            0.0447 -0.0309  0.1478  0.0005  0.7071
losses         -0.0447  0.0309 -0.1478 -0.0005  0.7071
points          0.1515  0.1717 -0.2000  0.7497  0.0000
field_goals     0.4516  0.2325 -0.0693 -0.5184  0.0000
points3        -0.3851 -0.2068 -0.3882 -0.2953  0.0000
free_throws    -0.0321  0.2443  0.0087 -0.2863  0.0000
off_rebounds   -0.5903 -0.1388 -0.1215  0.0011  0.0000
def_rebounds    0.1300 -0.5662  0.2385  0.0001  0.0000
assists        -0.4191  0.1085  0.5380  0.0027  0.0000
steals          0.1276 -0.5599 -0.3319  0.0001  0.0000
blocks         -0.2478  0.3463 -0.1908  0.0027  0.0000
personal_fouls  0.0338 -0.1646  0.5030  0.0017  0.0000
```

Your turn

What are the differences between `prcomp()` and `princomp()`? Spend some time reading the help documentation of both functions to find out the main differences between them. Are there any cases when it would be better to use one function or the other?

Stages of a Principal Components Analysis

The primary goal in PCA is to summarize the systematic patterns of variation in a data set, which is done via the principal components (PCs). Derived from this idea, the most common uses of the PCA results is to visualize multivariate data and/or to perform a dimension reduction (for other analytical purposes).

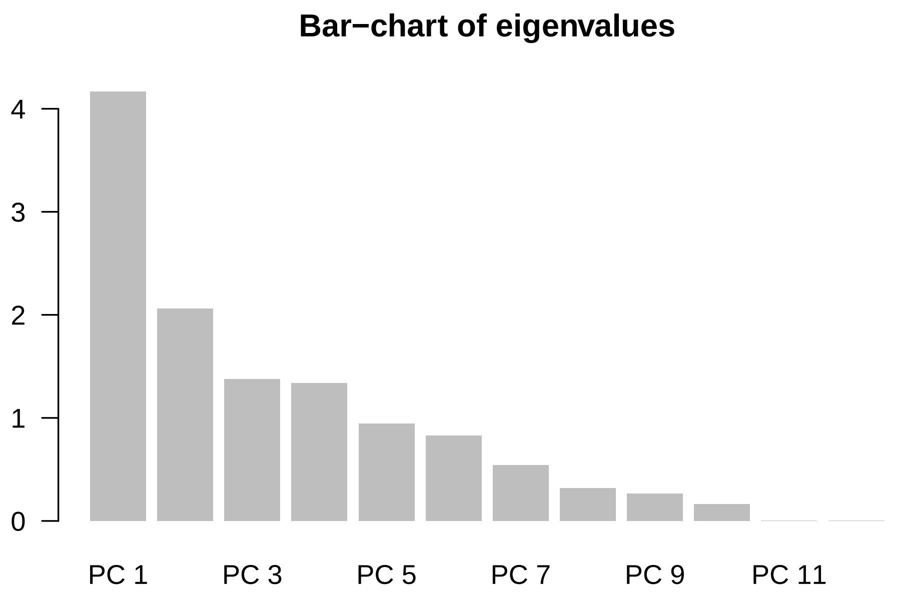
Eigenvalues and Proportion of Variance Explained

The first step when examining the results of a PCA is to look at how much variability is captured by each PC. This is done by examining the eigenvalues.

Your turn

Compute a table containing the eigenvalues, the variance in terms of percentages, and the cumulative percentages, like the table below. Analysts typically look at a bar-chart of the eigenvalues (see figure below). Plot your own bar-chart.

```
    eigenvalue  percentage  cumulative.percentage
1       4.1646     34.7051                34.7051
2       2.0616     17.1802                51.8853
3       1.3777     11.4805                63.3658
4       1.3390     11.1586                74.5245
5       0.9445      7.8710                82.3954
6       0.8266      6.8886                89.2841
7       0.5419      4.5157                93.7997
8       0.3172      2.6435                96.4432
9       0.2635      2.1957                98.6389
10      0.1632      1.3600                99.9989
11      0.0001      0.0011               100.0000
12      0.0000      0.0000               100.0000
```



- How much of the variation in the data is captured by the first PC?
- How much of the variation in the data is captured by the second PC?
- How much of the variation in the data is captured by the first two PCs?

Choosing the number of components

A related concern is to be able to determine how many PCs should be retained. There are various strategies:

- Retain just enough components to explain some specified, large percentage of the total variation of the original variables. For example, how many PCs would you retain to capture 70% of the total variation?
- Exclude those PCs whose eigenvalues are less than the average: $\sum_{h=1}^{p} \lambda_h / p$. Using this criterion, how many PCs would you retain?
- When the PCs are extracted from the correlation matrix, like in this case, the average eigenvalue is one; components with eigenvalues less than one are therefore excluded. This rule is known as *Kaiser's rule.* Using this criterion, how many PCs would you retain?
- Jollife (1972) proposed a variation of the Kaiser's rule: exclude PCs whose eigenvalues are less than 0.7. Under this criterion: how many PCs would you retain?
- Cattel (1965) suggests plotting the eigenvalues with the so-called *scree-plot* or *scree diagram*—like the bar-chart of eigenvalues. Cattel suggests looking for an "elbow" in the diagram, this would correspond to a point where "large" eigenvalues cease and "small" eigenvalues begin. With this rule, how many PCs would you retain?

Variable Loadings and Correlations with PCs

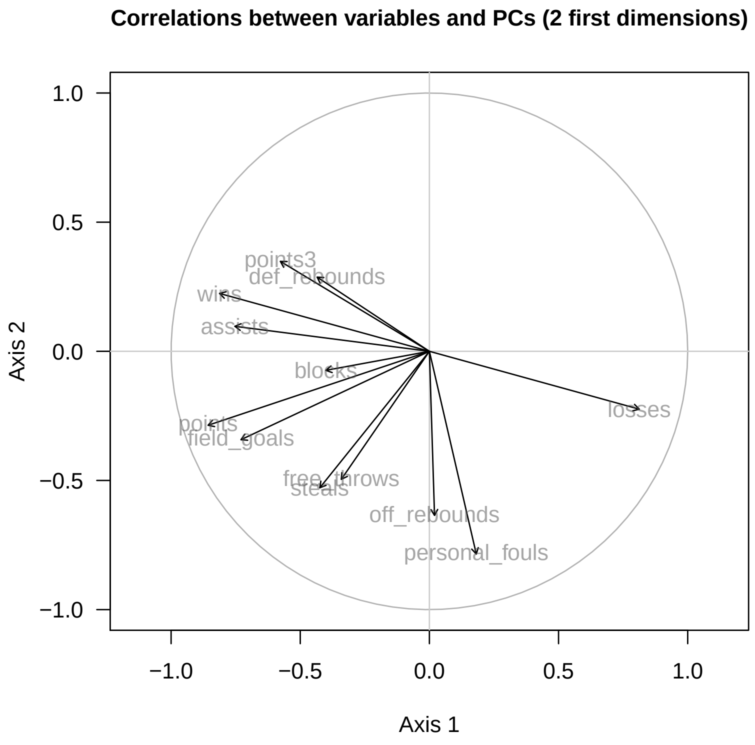
The next stage consists of examining how PCs are formed. Recall that PCs are linear combinations of the input variables, in which the coefficients of such linear combinations are given by the loadings. A starting point is to check the matrix of loadings. The larger the loading of a variable in a given PC, the more associated the variable is with that PC.

Another way to examine how variables are associated with the PCs is to look at their correlations.

Your turn

- Calculate a matrix (ot table) of correlations between the variables and the PCs. In other words, what are the correlations of the variables with the 1st PC, with the 2nd PC, and so on.
- What variables seem to be more correlated with PC1?
- What variables seem to be more correlated with PC2?

Less common, but extremely helpful, is to use the columns of the correlation matrix to visualize how variables are associated with the PCs.



This graph is known as the *circle of correlations*. The variables are plotted as arrows, with the length of the arrow equal to the correlation coefficient. The closer the arrowhead to the circumference (of radious 1), the better its representation with the associated PCs. Try to plot a graph like the one above (if you are running out of time ignore plotting the circle).

With either the loadings, or the correlations between variables and PCs, analysts typically try to give labels to the PCs. Looking at how variables are associated with the PCs:

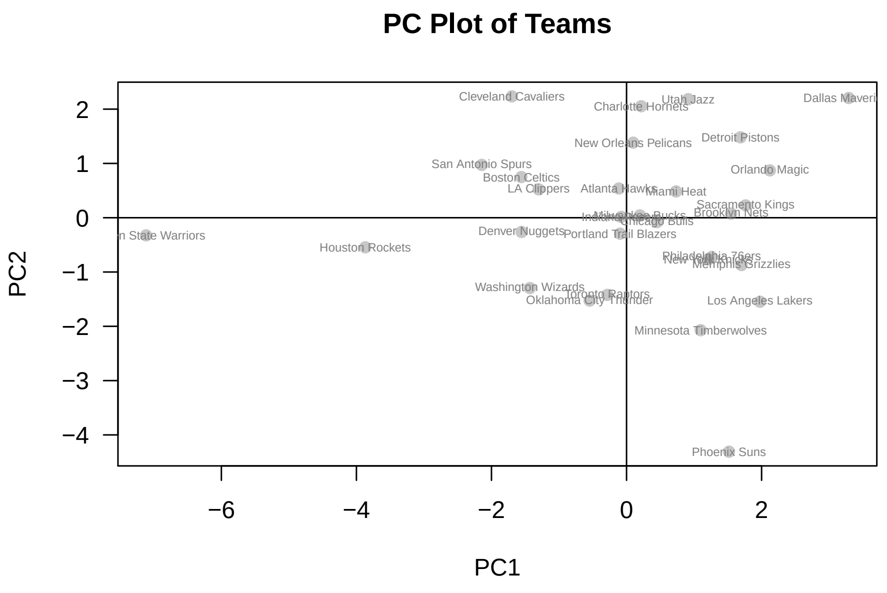
- What label—and interpretation—would you give to PC1?
- What label—and interpretation—would you give to PC2?
- What label—and interpretation—would you give to PC3?

Visualizing observations

The next stage involves visualizing the observations in a low dimensional space. This is typically achieved with scatterplots of the principal components.

Your turn

Begin with a scatterplot of the first two PCs (see figure below).



- Also plot PC1 - PC3, and then plot PC2 - PC3. If you want, continue visualizing other scatterplots.

- What patterns do you see?

- Try adding numeric labels to the points to see which observations seem to be potential outliers.

- Try getting scatterplots with the R package `"plotly"` and its function `plot_ly()`

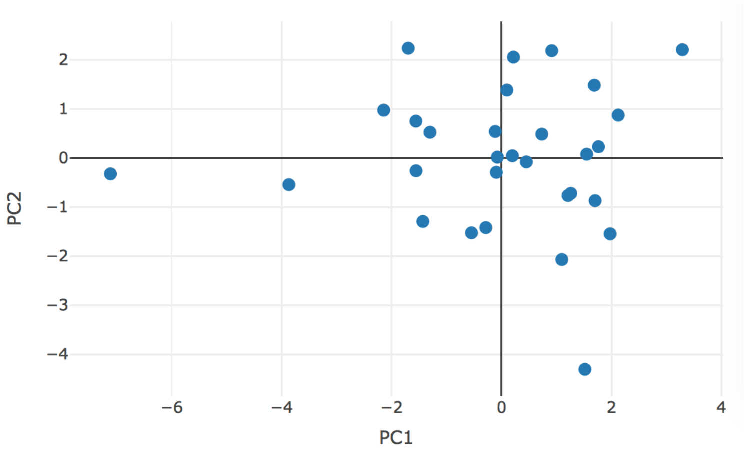
```
# install.packages("plotly")
library(plotly)
```

```
# data frame for plot_ly()
scores_df <- cbind.data.frame(
```

```r
  scores,
  team = dataset$team,
  stringsAsFactors = FALSE
)
```

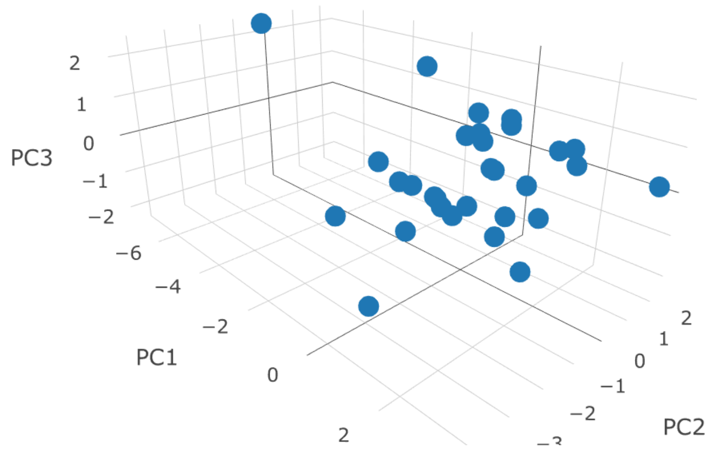
Here's a scatterplot of PC1 and PC2

```r
# scatter plot
plot_ly(data = scores_df, x = ~PC1, y = ~PC2,
        type = 'scatter',
        mode = 'markers',
        text = ~team,
        marker = list(size = 10))
```



Here's a 3D-scatterplot of PC1, PC2, and PC3

```r
# 3d scatter plot
plot_ly(data = scores_df, x = ~PC1, y = ~PC2, z = ~PC3,
        type = 'scatter3d',
        mode = 'markers',
        text = ~team)
```

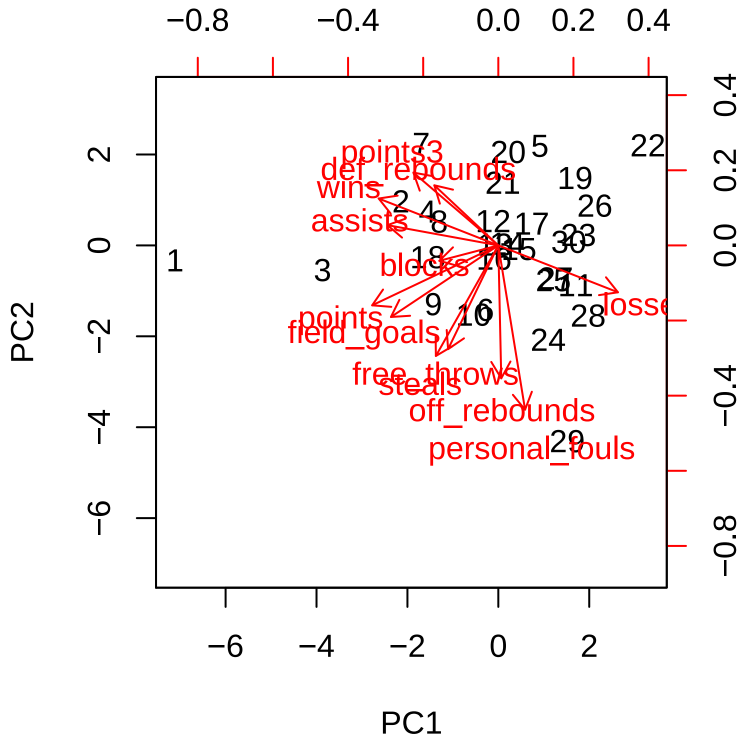


Biplot: Another visual display

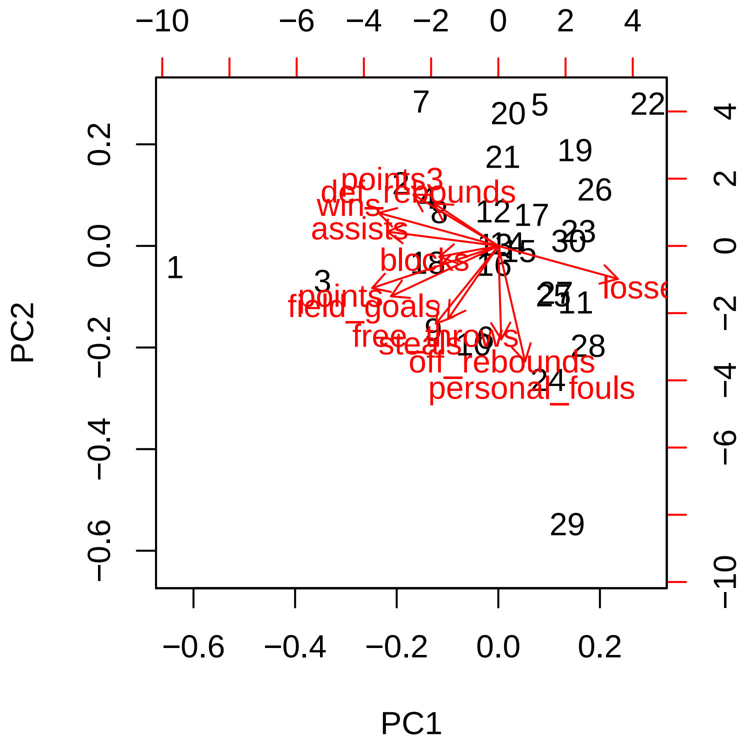
A fourth (optional) stage consists of obtaining a simultaneous visual display of both the observations and the variables in a low dimensional space. This is done with the so-called **biplot**, originally proposed by Ruben Gabriel in 1971, and available in R with the homonym function `biplot()`.

Some authors are opposed to this type of visualization arguing that it is a fictitious display. Personally, I'm not opposed to this type of display, as long as you know how to read it. The important thing to keep in mind is that, in a biplot, you are superimposing two different low-dimensional spaces: one corresponds to the observations, and the other corresponds to the variables. Also, because you are superimposing two clouds of objects, typically the scale of one—or both—of them will be distorted.

```r
# biplot
biplot(pca_prcomp, scale = 0)
```



```r
biplot(pca_prcomp, scale = 1)
```



The `scale = 0` argument to `biplot()` ensures that the arrows are scaled to represent the loadings, while the PCs are scaled to unit variance; also, when specifying `scale = 0` the distances between the observations correspond to Mahalanobis distances. Other values for `scale` give slightly different biplots with different scaling distortions.

Your turn: Graph various `biplot()`'s with different values of `scale` (e.g. 0, 0.3, 0.5, 1). How do the relative positions of the arrows change with respect to the points? Under which scale value you find it easier to read the biplot?