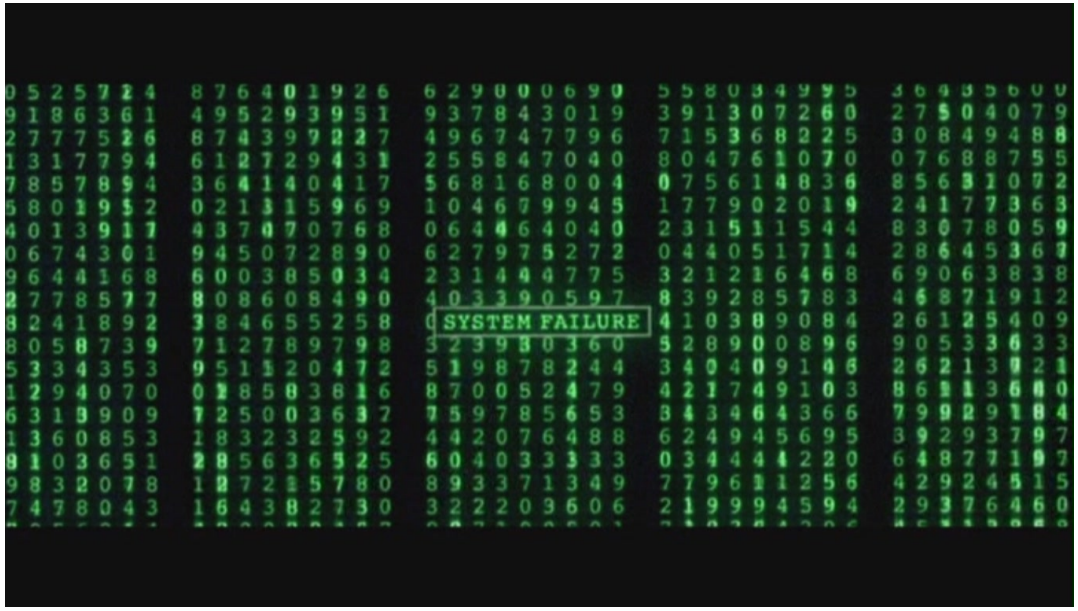
Multivariate101

Multivariate Analysis

Susan Holmes (c)



Matrices and their Motivation

It is current practice to measure many variables on the same patients, we may have all the biometrical characteristics, height, weight, BMI, age as well as clinical variables such as blood pressure, blood sugar, heart rate for 100 patients, these variables will not be independent.

What are the data?

- To start off, a useful toy example we'll use is from the sports world; performances of decathlon athletes.

  These are measurements of athletes' performances in the decathlon: the variables m100, m400, m1500 are performance times in seconds for the 100 metres, 400 metres and 1500 meters respectively, 'm110' is the time taken to finish the 110

meters hurdles whereas pole is the pole-jump height, weight is the length in metres the athletes threw the weight.

	m100	long	weight	highj	m400	m110	disc	pole	jave	m1500
1	11.25	7.43	15.48	2.27	48.90	15.13	49.28	4.70	61.32	268.95
2	10.87	7.45	14.97	1.97	47.71	14.46	44.36	5.10	61.76	273.02
3	11.18	7.44	14.20	1.97	48.29	14.81	43.66	5.20	64.16	263.20
4	10.62	7.38	15.02	2.03	49.06	14.72	44.80	4.90	64.04	285.11

Diabetes

- Clinical measurements (‘diabetes‘ data). This data measures glucose levels in the blood after fasting (glufast), after a test condition (glutest) as well as steady state plasma glucose (steady) and steady state (insulin) for diabetes, the sixth variable is not continuous and is considered a *supplementary* variable as we will see.

```
diabetes=read.table(url("http://bios221.stanford.edu/data/diabetes.txt"),header=TRUE,row.nam
es=1)
diabetes[1:4,]
```

```
##   relwt glufast glutest steady insulin Group
## 1  0.81      80     356    124      55     3
## 3  0.94     105     319    143     105     3
## 5  1.00      90     323    240     143     3
## 7  0.91     100     350    221     119     3
```

Microbial Ecology

- Operational Taxon Unit read counts in a microbial ecology study; the columns represent different ‘species’ of bacteria, the rows are labeled for the samples.

```
             469478 208196 378462 265971 570812
EKCM1.489478      0      0      2      0      0
EKCM7.489464      0      0      2      0      2
EKBM2.489466      0      0     12      0      0
PTCM3.489508      0      0     14      0      0
EKCF2.489571      0      0      4      0      0
```

RNA-Seq

- Here are some RNA-seq transcriptomic data showing numbers of mRNA reads present for different patient samples, the rows are patients and the columns are the genes.

```
           FBgn0000017 FBgn0000018 FBgn0000022 FBgn0000024 FBgn0000028 FBgn0000032
untreated1        4664         583           0          10           0        1446
untreated2        8714         761           1          11           1        1713
untreated4        3150         310           0           3           0         672
treated1          6205         722           0          10           0        1698
treated3          3334         308           0           5           1         757
```

Mass Spectroscopy

- Mass spectroscopy data where we have samples containing informative labels (knockout versus wildtype mice) and protein × features designated by their m/z number.

```
mz       129.9816   72.08144  151.6255  142.0349  169.0413    186.0355
KOGCHUM1  60515      181495          0    196526    25500    51504.40
WTGCHUM1 252579       54697        412    487800    48775    130491.15
WTGCHUM2 187859       56318      46425   454226    45626    100845.01
```

Expression Data (microarray)

- Here the rows are samples from different subjects and different T cell types and the columns are a subset of gene expression measurements on the 156 most differentially expressed genes (Holmes2005memory).

```
#######Melanoma/Tcell Data: Peter Lee, Susan Holmes, PNAS.
load(url("http://bios221.stanford.edu/data/Msig3transp.RData"))
round(Msig3transp,2)[1:5,1:6]
```

```
##              X3968 X14831 X13492 X5108 X16348  X585
## HEA26_EFFE_1 -2.61  -1.19  -0.06 -0.15   0.52 -0.02
## HEA26_MEM_1  -2.26  -0.47   0.28  0.54  -0.37  0.11
## HEA26_NAI_1  -0.27   0.82   0.81  0.72  -0.90  0.75
## MEL36_EFFE_1 -2.24  -1.08  -0.24 -0.18   0.64  0.01
## MEL36_MEM_1  -2.68  -0.15   0.25  0.95  -0.20  0.17
```

```
celltypes=factor(substr(rownames(Msig3transp),7,9))
status=factor(substr(rownames(Msig3transp),1,3))
```

The voting data

```
house=read.table("/Users/susan/Dropbox/CaseStudies/votes.txt")
head(house[,1:5])
```

```
##     V1   V2   V3   V4   V5
## 1 -0.5 -0.5  0.5 -0.5  0.0
## 2 -0.5 -0.5  0.5 -0.5  0.0
## 3  0.5  0.5 -0.5  0.5 -0.5
## 4  0.5  0.5 -0.5  0.5 -0.5
## 5  0.5  0.5 -0.5  0.5 -0.5
## 6 -0.5 -0.5  0.5 -0.5  0.0
```

```
party=scan("/Users/susan/Dropbox/CaseStudies/party.txt")
#table(party)
```

Biometrical Measurements

- Measurements: turtles

```
#require(MSBdata)
turtles=read.table(url("http://bios221.stanford.edu/data/PaintedTurtles.txt"),header=TRUE)
turtles[1:4,]
```

```
##   sex length width height
## 1   f     98    81     38
## 2   f    103    84     38
## 3   f    103    86     42
## 4   f    105    86     40
```

- Some biological traits of lizards are available in the ‘ade4‘ package

```
require(ade4)
data(lizards)
lizards$traits[1:4,c(1,5,6,7,8)]
```

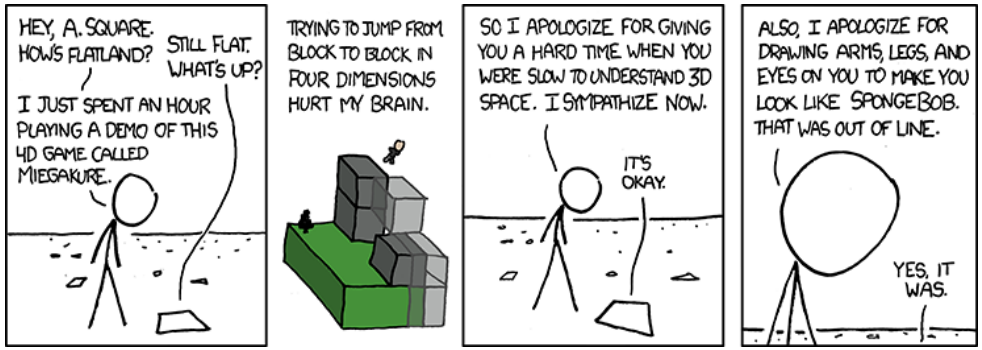
```
##    mean.L hatch.m clutch.S age.mat clutch.F
## Sa   69.2   0.572      6.0      13      1.5
## Sh   48.4   0.310      3.2       5      2.0
## Tl  168.4   2.235     16.9      19      1.0
## Mc   66.1   0.441      7.2      11      1.5
```

Data visualization and preparation

It is always beneficial to start a multidimensional analysis by checking the simple one dimensional and two dimensional summary statistics, we can visualize these using a graphics package that builds on ‘ggplot2‘ called ‘GGally‘.

Low dimensional data summaries and preparation

What do we mean by low dimensional ?



flatland

If we are studying only one variable, just one column of our matrix, we might call it $\mathbf{x}$ or $\mathbf{x}_{\bullet j}$; we call it one dimensional.

A *one dimensional summary* a histogram that shows that variable's distribution, or we could compute its mean $\bar{x}$ or median, these are zero-th dimensional summaries of one dimension data.

Two dimensional data

When considering two variables (x and y) measured together on a set of observations, the **correlation coefficient** measures how the variables co-vary.

This is a single number summarizes two dimensional data, its formula involves $\bar{x}$ and $\bar{y}$:

$$\hat{\rho} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

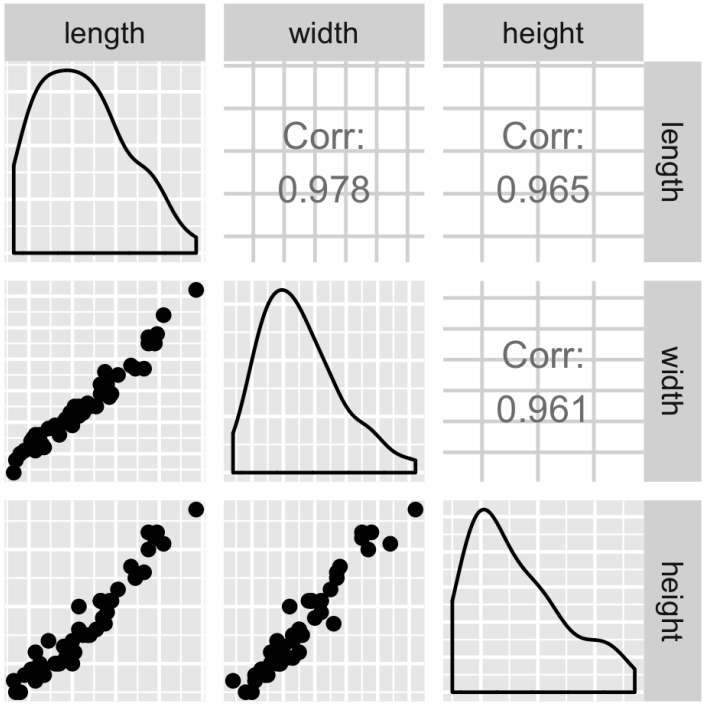
```
cor(turtles[,-1])
```

```
##        length width height
## length  1.000 0.978  0.965
## width   0.978 1.000  0.961
## height  0.965 0.961  1.000
```

```
library("GGally")
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

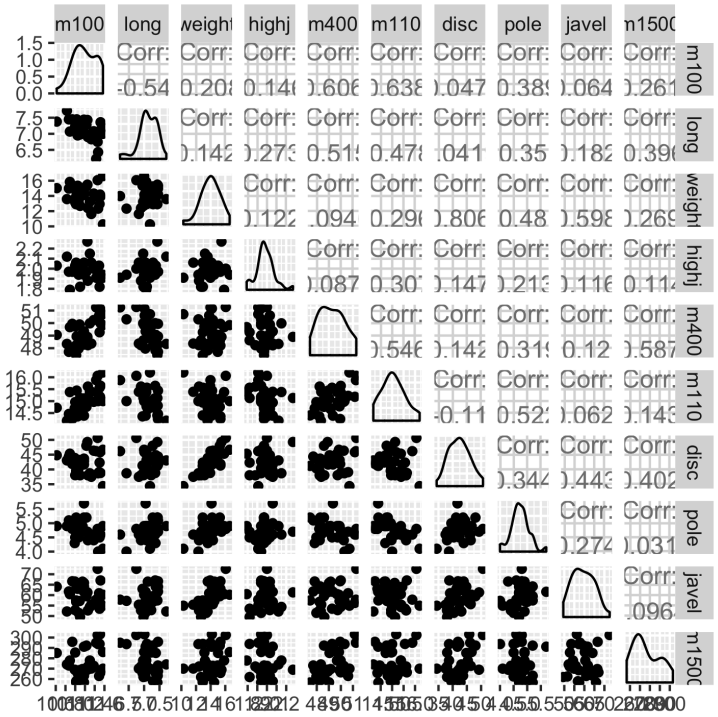
```
ggpairs(turtles[,-1],axisLabels = "none")
```



Pairs plot for turtles data

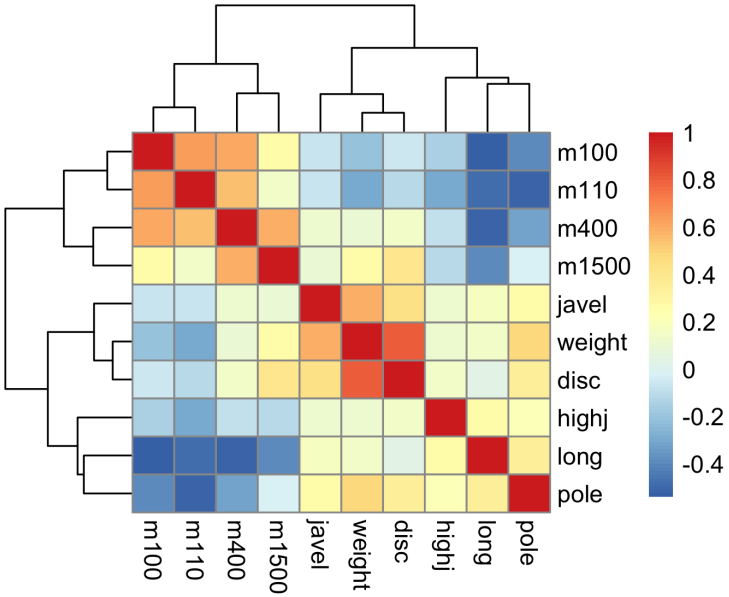
`pairs` plot for `athletes`

```
ggpairs(athletes)
```



Pairs athletes

```
library("pheatmap")
pheatmap(cor(athletes),cell.width=10,cell.height=10)
```



Heatmap athletes

Preprocessing the data

We usually **center** the cloud of points around the origin; the most common way of doing this is to make new variables whose means are all zero.

More robust scaling can be done also (median).

Different variables are measured in different units, and at different scales, and so would be hard to compare in their original form.

```
library("ggplot2")
library("factoextra")
```

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EF
CZ
```

```
apply(turtles[,-1],2,sd)
```

```
## length  width height
##  20.48  12.68   8.39
```

```
apply(turtles[,-1],2,mean)
```

```
## length  width height
##  124.7   95.4   46.3
```

Transform the data: standardizing

Making the data have a common standard deviation is the usual transformation. As in the correlation coefficient.

This rescaling is done using the scale function which makes every column have a variance of 1.

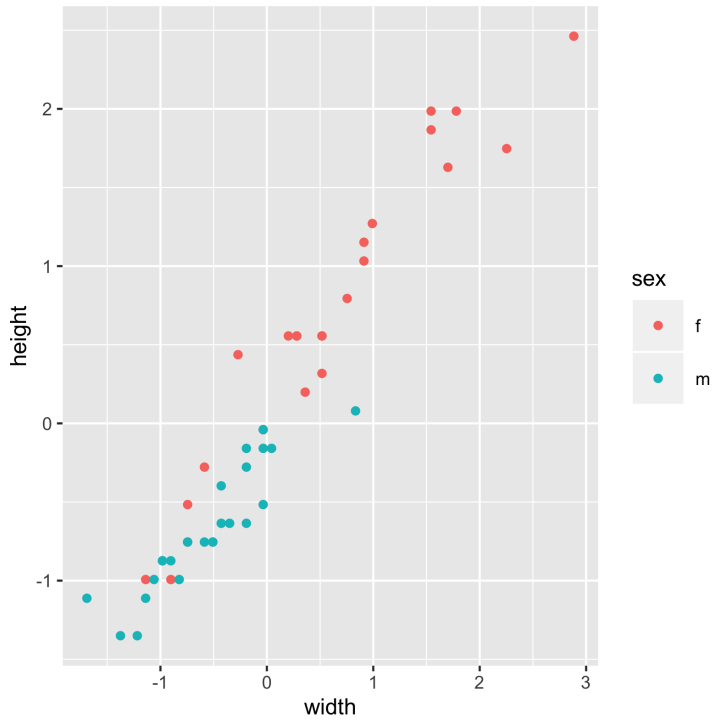
```
turtleMatScale=scale(turtles[,-1])
scaledturtles=data.frame(turtleMatScale,sex=turtles[,1])
apply(scaledturtles[,-4],2,mean)
```

```
##    length     width    height
## -1.43e-18  1.94e-17 -2.87e-16
```

```
apply(scaledturtles[,-4],2,sd)
```

```
## length  width height
##      1      1      1
```

```
ggplot(scaledturtles,aes(x=width,y=height, group =sex)) +
  geom_point(aes(color=sex))
```



A Little History

Invented in 1901 by Karl Pearson as a way to reduce a two variable scatterplot to a single coordinate.

Used by statisticians in the 1930s to summarize a battery of psychological tests run on the same subjects Hotelling:1933, extracting overall scores that could summarize many variables at once.

It is called Principal Component Analysis (abbreviated PCA).

Not principled

Dimension reduction

PCA is an 'unsupervised learning technique' because it treats all variables as having the same status.

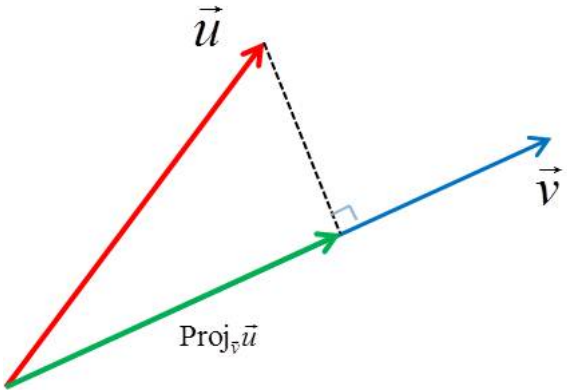
PCA is visualization technique which produces maps of both variables and observations.

We are going to give you a flavor of what is called multivariate analyses. As a useful first approximation we formulate many of the methods through manipulations called linear algebra.

The *raison d'être* for multivariate analyses is connections or associations between the different variables.

If the columns of the matrix are unrelated, we should just study each column separately and do standard univariate statistics on them one by one.

We use projections:

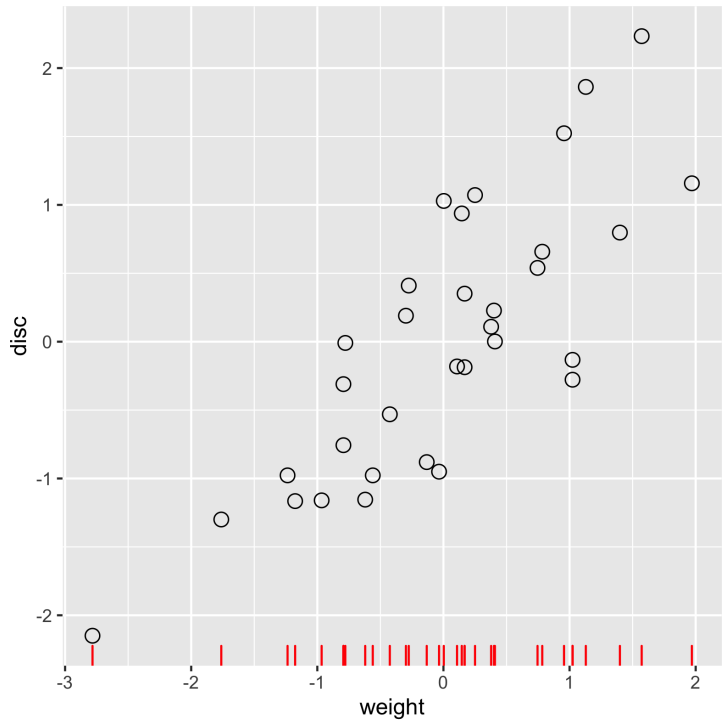


projectionvector

Low Dimensional Projections

Here we show one way of projecting two dimensional data onto a line.

The olympic data come from the `ade4` package, they are the performances of decathlon athletes in an olympic competition.



Scatterplot of two variables showing projection on the x coordinate in red.

How do we summarize two dimensional data by a line?

In general, we lose information about the points when we project down from two dimensions (a plane) to one (a line).

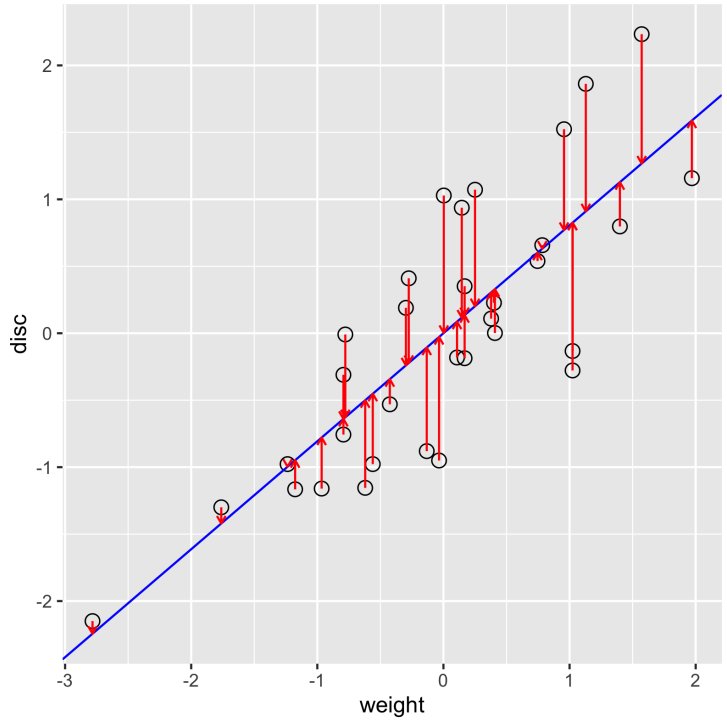
If we do it just by using the original coordinates, for instance the x coordinate as we did above, we lose all the information about the second one.

There are actually many ways of projecting the point cloud onto a line. One is to use what are known as regression lines. Let's look at these lines and how there are constructed in R:

Regressing one variable on the other

The disc variable on the weight

```
attach(athletes)
require(ggplot2)
reg1 <- lm(disc~weight,data=athletes)
#abline(reg1, col='red')
a <- reg1$coefficients[1] # Intercept
b <- reg1$coefficients[2] # slope
pline=p+geom_abline(intercept=a,slope=b, col="blue")
proj=pline+geom_segment(aes(x=weight, xend=weight, y=disc,
yend=reg1$fitted),linetype=1,colour="red",
arrow = arrow(length = unit(0.15,"cm")))
print(proj)
```



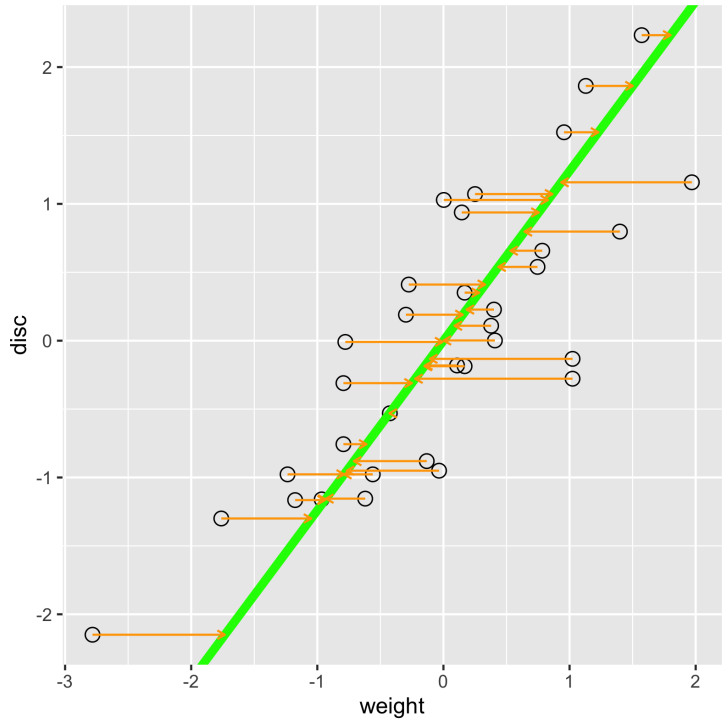
The blue line minimizes the sum of squares of the vertical residuals (in red),

What is the variance of the points along the blue line?

```
matproj=cbind(weight,reg1$fitted)
sum(apply(matproj,2,var))
```

```
## [1] 1.65
```

Regression of weight on discus



Variance of the data points

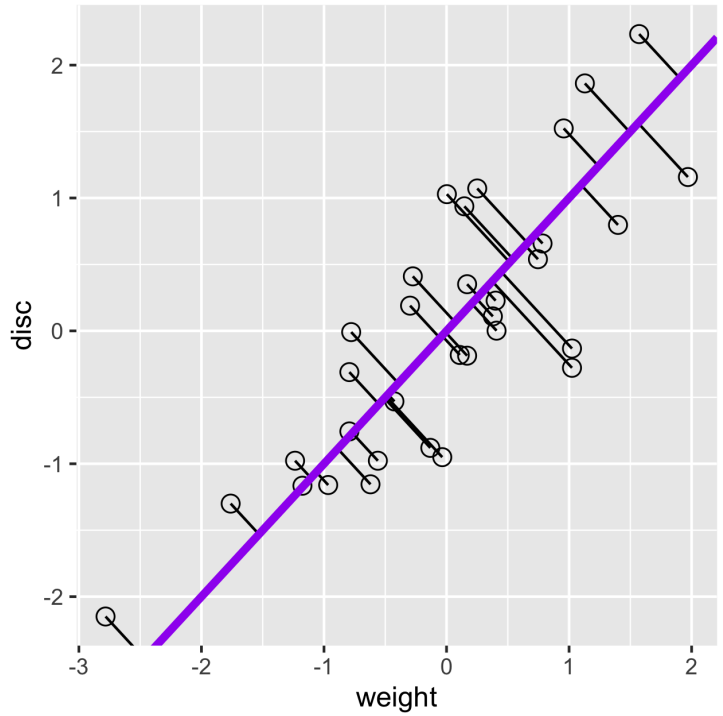
```
matproj2=cbind(weight,reg2$fitted)
sum(apply(matproj,2,var))
```

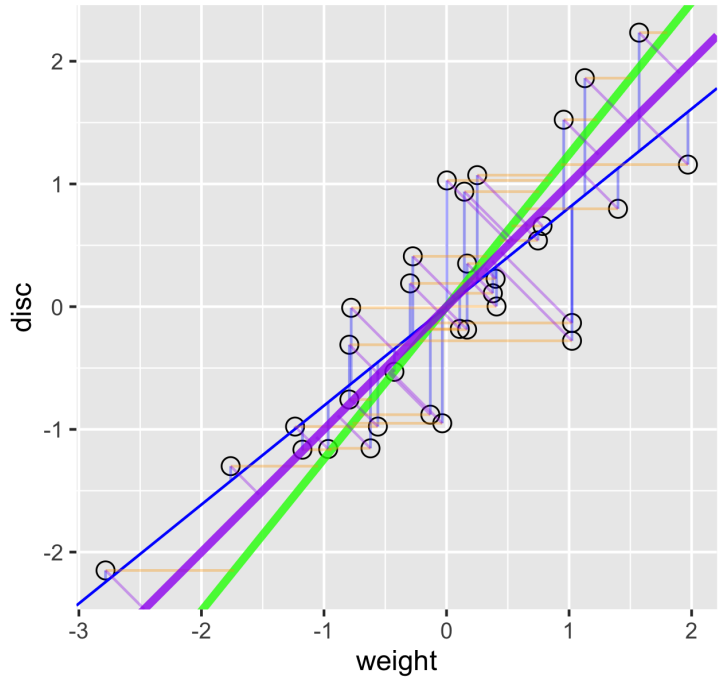
```
## [1] 1.65
```

The orange line minimizes the horizontal residuals for the weight variable in orange.

The PCA line: it minimizes in both directions

```
xy=cbind(athletes$disc,athletes$weight)
svda=svd(xy)
pc = xy %*% svda$v[,1] %*% t(svda$v[,1])
bp = svda$v[2,1] /svda$v[1,1]
ap = mean(pc[,2])-bp*mean(pc[,1])
p+geom_segment(xend=pc[,1],yend=pc[,2])+
geom_abline(intercept=ap,slope=bp, col="purple",lwd=1.5)
```


disc
weight
2
1
0
-1
-2
-3
-2
-1
0
1
2


disc
weight
2
1
0
-1
-2
-3
-2
-1
0
1
2

The purple line minimizes both residuals and thus (through Pythagoras) it minimizes the sum of squared distances from the points to the line.

Minimizing the distance to the line in both directions, the purple line is the principal component line, the green and blue line are the regression lines.

Variance along the line

The lines created here are sensitive to the choice of units; because we have made the standard deviations equal to one for both variables, the PCA line is the diagonal that cuts exactly in the middle of both regression lines.

The data were centered by subtracting their means thus ensuring that the line passes through the origin $(0, 0)$.

Compute the variance of the points on the purple line.

The coordinates of the points when we made the plot, these are in the `pc` vector:

```
apply(pc,2,var)
```

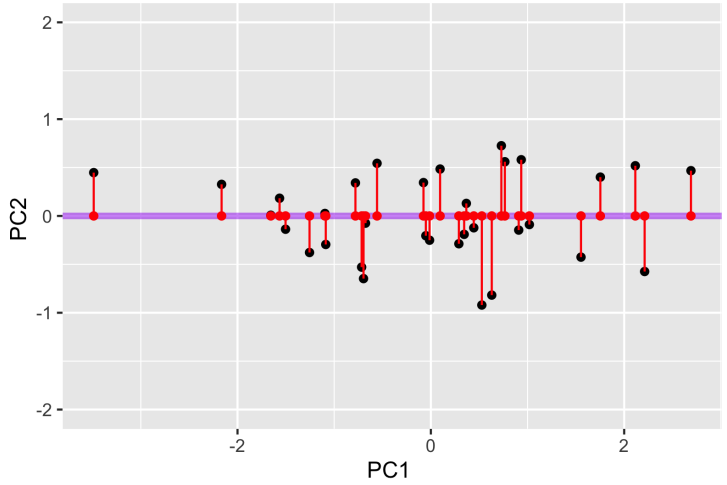
```
## [1] 0.903 0.903
```

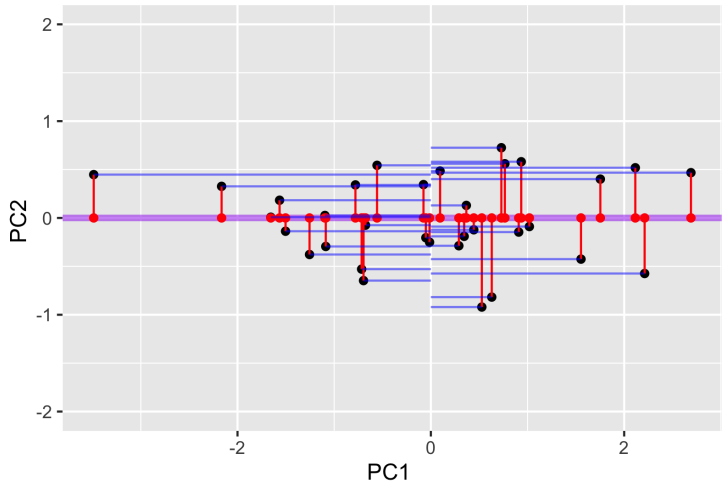
```
sum(apply(pc,2,var))
```

```
## [1] 1.81
```

PCA for 2 dimensional data

```
ppdf = data.frame(PC1n=-svda$u[,1]*svda$d[1],PC2n=svda$u[,2]*svda$d[2])
ggplot(ppdf,aes(x=PC1n,y=PC2n))+geom_point()+ ylab("PC2 ")+
     geom_hline(yintercept=0,color="purple",lwd=1.5,alpha=0.5) +
      geom_point(aes(x=PC1n,y=0),color="red")+ xlab("PC1 ")+
     xlim(-3.5, 2.7)+ylim(-2,2)+coord_fixed() +
     geom_segment(aes(xend=PC1n,yend=0), color="red")
```


2
1
0
-1
-2
PC2
-2
0
2
PC1


2
1
0
-1
-2
PC2
-2
0
2
PC1

Notes about Lines

The line created here is sensitive to the choice of units, and to the center of the cloud.

Note that Pythagoras' theorem tells us two interesting things here, if we are minimizing in both horizontal and vertical directions we are in fact minimizing the diagonal projections onto the line from each point.

Principal Components are Linear Combinations of the 'old' variables

To understand what that a linear combination really is, we can take an analogy, when making a healthy juice mix, you can follow a recipe.

image

ingredients

- 2 pounds beets (about 6 medium), trimmed, peeled, cut into 1" pieces
- 1 pound carrots (about 4 large), trimmed, peeled, cut into 1" pieces
- 1 Gala or Empire apple (about 8 ounces), cored, cut into 1" pieces
- 1 Granny smith apple (about 8 ounces), cored, cut into 1" pieces
- 1 3" piece fresh ginger, peeled, chopped into 1" pieces
- 3 tablespoons fresh lemon juice

image

$$V = 2 \times \text{ Beets } + 1 \times \text{Carrots } + \frac{1}{2} \text{ Gala} + \frac{1}{2} \text{ GrannySmith} + 0.02 \times \text{ Ginger} + 0.25 \text{ Lemon}$$

This recipe is a linear combination of individual juice types, in our analogy these are replaced by the original variables. The result is a new variable, the coefficients $(2, 1, \frac{1}{2}, \frac{1}{2}, 0.02, 0.25)$ are called the loadings.

Optimal lines

A linear combination of variables defines a line in our space in the same way we say lines in the scatterplot plane for two dimensions. As we saw in that case, there are many ways to choose lines onto which we project the data, there is however a 'best' line for our purposes.

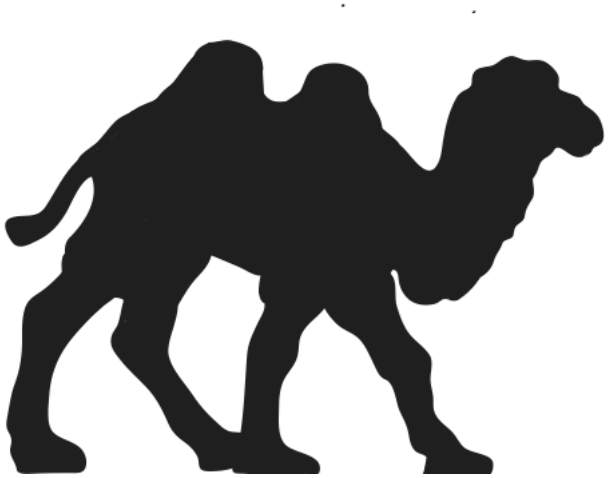
Total variance can de decomposed The total sums of squares of the distances between the points and any line can be decomposed into the distance to the line and the variance along the line.

We saw that the principal component minimizes the distance to the line, and it also maximizes the variance of the projections along the line.

Good Projections

What is this?

Good Projections

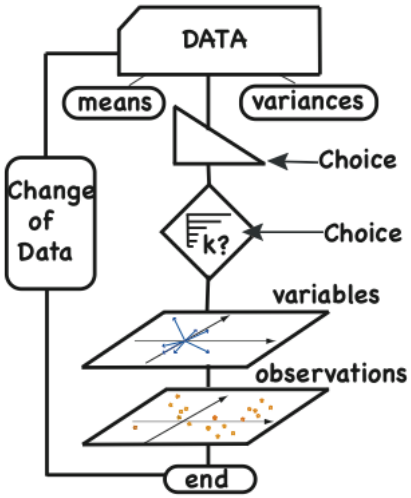


MysteryImage

Which projection do you think is better?

It's the projection that maximizes the area of the shadow and an equivalent measurement is the sums of squares of the distances between points in the projection, we want to see as much of the variation as possible, that's what PCA does.

The PCA workflow



Many Choices have to made during PCA processing.

PCA is based on the principle of finding the largest axis of inertia/variability and then iterating to find the next best axis which is orthogonal to the previous one and so on.

The Inner Workings of PCA: the Singular Value Decomposition

Eigenvalues of X'X or Singular values of X tell us the rank.

What does rank mean?

```
 X |  2  4  8
---| --------
 1 |
 2 |
 3 |
 4 |
```

```
X  |  2  4  8
-- | ---------
1  |  2
2  |  4
3  |  6
4  |  8
```

```
  X  |  2  4  8
 ---| --------
  1 |  2  4  8
  2 |  4  8 16
  3 |  6 12 24
  4 |  8 16 32
```

We say that the matrix

$$\begin{pmatrix} 2 & 4 & 8 \\ 4 & 8 & 16 \\ 6 & 12 & 24 \\ 8 & 16 & 32 \end{pmatrix}$$

is of rank one.

$$\begin{pmatrix} 2 & 4 & 8 \\ 4 & 8 & 16 \\ 6 & 12 & 24 \\ 8 & 16 & 32 \end{pmatrix} == u * t(v) = u * v', \qquad u = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \text{ and } v' = t(v) = (2\ 4\ 8).$$

Backwards from the matrix to decomposition

```
X=matrix(c(780, 75, 540, 936, 90, 648, 1300, 125, 900,
           728, 70, 504),nrow=3)
X
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  780  936 1300  728
## [2,]   75   90  125   70
## [3,]  540  648  900  504
```

```
u1=c(0.8,0.1,0.6)
v1=c(0.4,0.5,0.7,0.4)
sum(u1^2)
```

```
## [1] 1.01
```

```
sum(v1^2)
```

```
## [1] 1.06
```

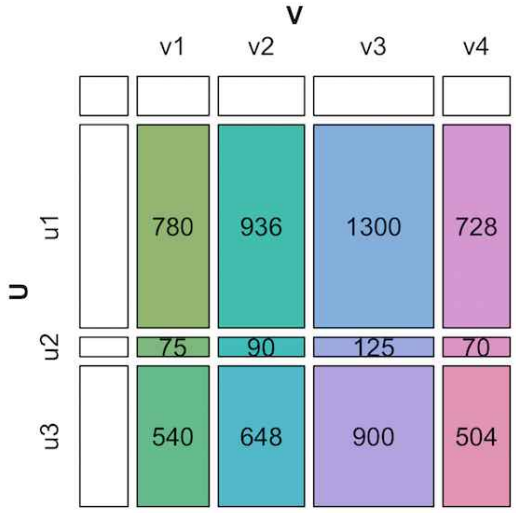
```
s1=2348.2
s1*u1 %*%t(v1)
```

```
##       [,1] [,2] [,3]  [,4]
## [1,] 751.4  939 1315 751.4
## [2,]  93.9  117  164  93.9
## [3,] 563.6  704  986 563.6
```

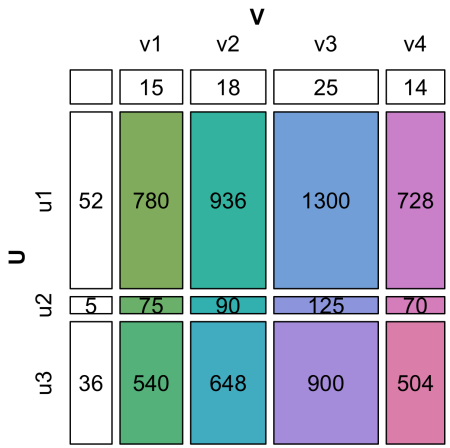
```
X-s1*u1 %*%t(v1)
```

```
##       [,1]   [,2]  [,3]  [,4]
## [1,]  28.6  -3.28 -15.0 -23.4
## [2,] -18.9 -27.41 -39.4 -23.9
## [3,] -23.6 -56.46 -86.2 -59.6
```

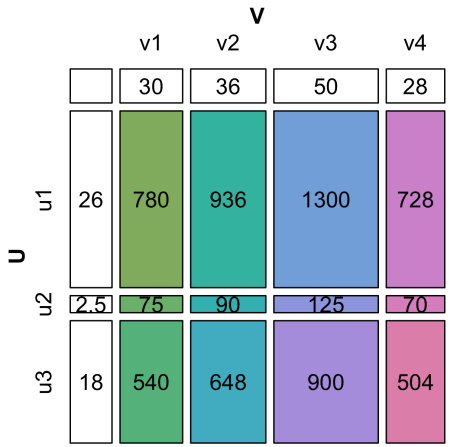
Graphical Decompositions



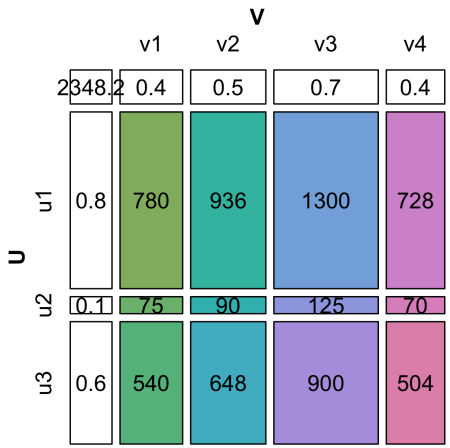
Matrix X we would like to decompose.



Areas are proportional to the entries



Looking at different possible margins



Forcing the margins to have norm 1

Check with R

```
## ----checkX-----------------------------------------------------------------
u1=c(0.8196, 0.0788, 0.5674)
v1=c(0.4053, 0.4863, 0.6754, 0.3782)
s1=2348.2
s1*u1 %*%t(v1)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  780  936 1300  728
## [2,]   75   90  125   70
## [3,]  540  648  900  504
```

```
Xsub=matrix(c(12.5 , 35.0 , 25.0 , 25,9,14,26,18,16,21,49,
           32,18,28,52,36,18,10.5,64.5,36),ncol=4,byrow=T)
Xsub
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 12.5 35.0 25.0   25
## [2,]  9.0 14.0 26.0   18
## [3,] 16.0 21.0 49.0   32
## [4,] 18.0 28.0 52.0   36
## [5,] 18.0 10.5 64.5   36
```

```
USV=svd(Xsub)
USV
```

```
## $d
## [1] 1.35e+02 2.81e+01 3.10e-15 1.85e-15
##
## $u
##        [,1]    [,2]    [,3]   [,4]
## [1,] -0.344  0.7717  0.5193 -0.114
## [2,] -0.264  0.0713 -0.3086 -0.504
## [3,] -0.475 -0.0415 -0.0386  0.803
## [4,] -0.528  0.1426 -0.6423 -0.103
## [5,] -0.554 -0.6143  0.4702 -0.280
##
## $v
##        [,1]    [,2]   [,3]    [,4]
## [1,] -0.250  0.0404 -0.967  0.0244
## [2,] -0.343  0.8798  0.133  0.3010
## [3,] -0.755 -0.4668  0.186  0.4214
## [4,] -0.500  0.0808  0.111 -0.8551
```

```
## ----CheckUSV--------------------------------------------------------------
Xsub-(135*USV$u[,1]%*%t(USV$v[,1]))
```

```
##         [,1]   [,2]    [,3]    [,4]
## [1,]  0.8802  19.05 -10.088  1.7604
## [2,]  0.0849   1.76  -0.921  0.1698
## [3,] -0.0396  -1.01   0.565 -0.0792
## [4,]  0.1698   3.53  -1.842  0.3397
## [5,] -0.6877 -15.15   8.069 -1.3754
```

```
Xsub-(135*USV$u[,1]%*%t(USV$v[,1]))-(28.1*USV$u[,2]%*%t(USV$v[,2]))
```

```
##         [,1]     [,2]   [,3]    [,4]
## [1,] 0.00387 -0.02528 0.0335 0.00774
## [2,] 0.00398  0.00264 0.0140 0.00796
## [3,] 0.00749  0.01192 0.0214 0.01498
## [4,] 0.00796  0.00527 0.0281 0.01592
## [5,] 0.00983  0.03784 0.0123 0.01965
```

```
Xsub- USV$d[1]*USV$u[,1]%*%t(USV$v[,1])-USV$d[2]*USV$u[,2]%*%t(USV$v[,2])
```

```
##          [,1]      [,2]     [,3]     [,4]
## [1,] 7.22e-15 -1.07e-14 8.88e-15 4.88e-15
## [2,] 2.04e-15 -6.00e-15 1.05e-14 3.22e-15
## [3,] 2.87e-15 -9.55e-15 1.55e-15 6.23e-15
## [4,] 4.39e-15 -5.77e-15 1.78e-14 7.05e-15
## [5,] 5.11e-15 -1.78e-15 1.78e-14 1.78e-14
```

Another Example

```
Xsub=matrix(c(12.5 , 35.0 , 25.0 , 25,9,14,26,18,16,21,49,32,18,28,52,36,18,10.5,64.5,36),nc
ol=4,byrow=T)
Xsub
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 12.5 35.0 25.0   25
## [2,]  9.0 14.0 26.0   18
## [3,] 16.0 21.0 49.0   32
## [4,] 18.0 28.0 52.0   36
## [5,] 18.0 10.5 64.5   36
```

```
svd(Xsub)
```

```
## $d
## [1] 1.35e+02 2.81e+01 3.10e-15 1.85e-15
##
## $u
##        [,1]    [,2]    [,3]   [,4]
## [1,] -0.344  0.7717  0.5193 -0.114
## [2,] -0.264  0.0713 -0.3086 -0.504
## [3,] -0.475 -0.0415 -0.0386  0.803
## [4,] -0.528  0.1426 -0.6423 -0.103
## [5,] -0.554 -0.6143  0.4702 -0.280
##
## $v
##        [,1]    [,2]   [,3]    [,4]
## [1,] -0.250  0.0404 -0.967  0.0244
## [2,] -0.343  0.8798  0.133  0.3010
## [3,] -0.755 -0.4668  0.186  0.4214
## [4,] -0.500  0.0808  0.111 -0.8551
```

```
USV=svd(Xsub)
XS1=Xsub-USV$d[1]*(USV$u[,1]%*% t(USV$v[,1]))
XS1
```

```
##         [,1]   [,2]    [,3]   [,4]
## [1,]  0.8748  19.05 -10.104  1.750
## [2,]  0.0808   1.76  -0.933  0.162
## [3,] -0.0470  -1.02   0.543 -0.094
## [4,]  0.1616   3.52  -1.866  0.323
## [5,] -0.6963 -15.16   8.043 -1.393
```

```
XS2=XS1-USV$d[2]*(USV$u[,2]%*% t(USV$v[,2]))
XS2
```

```
##          [,1]      [,2]     [,3]     [,4]
## [1,] 7.22e-15 -1.07e-14 8.88e-15 4.88e-15
## [2,] 2.04e-15 -6.00e-15 1.05e-14 3.22e-15
## [3,] 2.87e-15 -9.55e-15 1.55e-15 6.23e-15
## [4,] 4.39e-15 -5.77e-15 1.78e-14 7.05e-15
## [5,] 5.11e-15 -1.78e-15 1.78e-14 1.78e-14
```

Special Example of Rank one matrix: independence

```
require(ade4)
HairColor=HairEyeColor[,,2]
HairColor
```

```
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    36    9     5     2
##   Brown    66   34    29    14
##   Red      16    7     7     7
##   Blond     4   64     5     8
```

```
chisq.test(HairColor)
```

```
## Warning in chisq.test(HairColor): Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  HairColor
## X-squared = 107, df = 9, p-value <2e-16
```

```
prows=sweep(HairColor,1,apply(HairColor,1,sum),"/")
pcols=sweep(HairColor,2,apply(HairColor,2,sum),"/")
Indep=313*as.matrix(prows)%*%t(as.matrix(pcols))
round(Indep)
```

```
##        Hair
## Hair    Black Brown Red Blond
##   Black    72   158  39    44
##   Brown    57   154  40    61
##   Red      55   155  44    59
##   Blond    28   108  27   149
```

```
sum((Indep-HairColor)^2/Indep)
```

```
## [1] 799
```

SVD for real data

```
## ------------------------------------------------------------------------
diabetes.svd=svd(scale(diabetes[,-5]))
names(diabetes.svd)
```

```
## [1] "d" "u" "v"
```

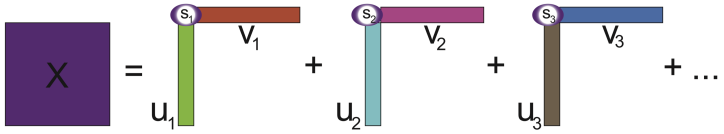
```
diabetes.svd$d
```

```
## [1] 20.09 13.38  9.89  5.63  1.70
```

```
turtles.svd=svd(scale(turtles[,-1]))
turtles.svd$d
```

```
## [1] 11.75  1.42  1.00
```

SVD



$$X \quad = u_{\bullet 1} * s_1 * v_{\bullet 1} \quad + \quad u_{\bullet 2} * s_2 * v_{\bullet 2} \quad + \quad u_{\bullet 3} * s_3 * v_{\bullet}$$

We write our horizontal/vertical decompostion of the matrix X in short hand as:

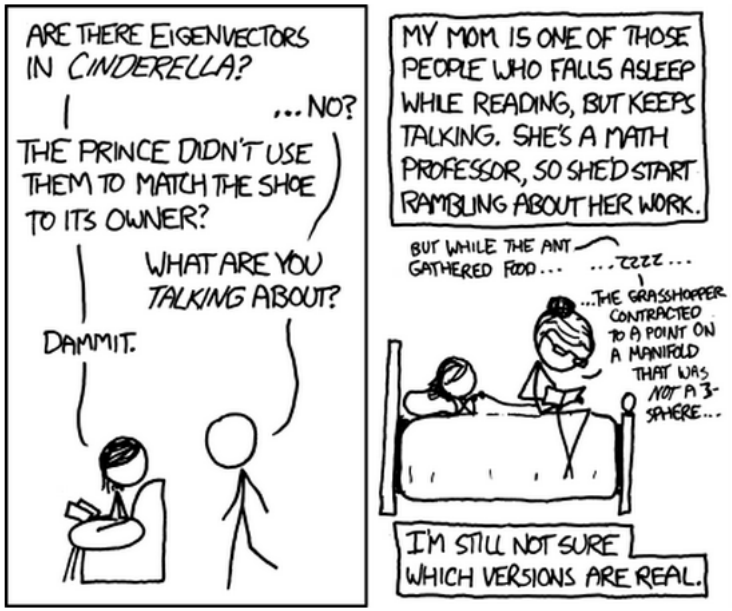
$$X = USV', V'V = I, U'U = I, S \text{ diagonal matrix of singular values, given by the } \{\text{tt}$$

The crossproduct of X with itself verifies

$$X'X = VSU'USV' = VS^2V' = V\Lambda V'$$

where V is called the eigenvector matrix of the symmetric matrix $X'X$ and Λ is the diagonal matrix of eigenvalues of $X'X$.

Why Eigenvectors are useful?



Why would eigenvectors come into use in Cinderella?

Khan's Academy (https://www.khanacademy.org/math/linear-algebra/alternate_bases/eigen_everything/v/linear-algebra--introduction-to-eigenvalues-and-eigenvectors)

Principal Components

The singular vectors from the singular value decomposition, `svd` function above tell us the coefficients to put in front of the old variables to make our new ones with better properties. We write this as :

$$PC_1 = c_1 X_{\bullet 1} + c_2 X_{\bullet 2} + c_3 X_{\bullet 3} + \cdots c_p X_{\bullet p}$$

Replace $X_{\bullet 1}, X_{\bullet 2}, \ldots X_{\bullet p}$ by

$$PC_1, PC_2, \ldots PC_k$$

What is the largest k can be ?

Suppose we have 5 samples with 23,000 genes measured on them, what is the dimensionality of these data?

The number of principal components is less than or equal to the number of original variables.

$$K \leq min(n, p)$$

The geometr(ies) of data: good trick look at size of vectors.

The Principal Component transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each successive component in turn has the highest variance possible under the constraint that it be orthogonal to the preceding components.

$$\max_{aX} \text{var}(Proj_{aX}(X))$$

Suppose the matrix of data X has been made to have column means 0 and standard deviations 1.

Matrix Decomposition

We call the principal components the columns of the matrix, $C = US$.

The columns of U (the matrix given as USV$u in the output from the svd function above) are rescaled to have norm s^2, the variance they are responsable for.

If the matrix X comes from the study of n different samples or specimens, then the principal components provides new coordinates for these n points these are sometimes also called the scores in some of the (many) PCA functions available in R (`princomp`, `prcomp`, `dudi.pca` in `ade4`).

Transition Formulae

If we only want the first one then it is just $c_1 = s_1 u_1$.

Variance explained by first principal component: s_1^2:

Notice that $||c_1||^2 = s_1' u_1 u_1' s_1 = s_1^2 u_1' u_1 = s_1^2 = \lambda_1$

$$X'C = VSU'US = VS^2$$

Remarks:

- 1. Each principal component is chosen to maximize the variance it explains, this variance is measured by the corresponding eigenvalue.
- 2. The new variables are made to be orthogonal, if the data are multivariate normal the new variables will be independent.
- 3. When the variables are rescaled or we choose the correlation matrix as the one we want to study instead of the covariance matrix then the sum of the variances of all the variables is the number of variables (=p), this is sometimes called the trace.
- 4. The principal components are always ordered by ``importance'', always look at what proportion of the variability you are interpreting (and check the screeplot before deciding how many components).

A few examples of using PCA

We start with the turtles data that has 3 continuous variables and a gender variable that we leave out for the original PCA analysis.

turtles Data

When computing the variance covariance matrix, many programs use 1/(n-1) as the denominator, here n=48 so the sum of the variances are off by a small fudge factor of 48/47.

```
turtles3var=turtles[,-1]
apply(turtles3var,2,mean)
```

```
## length  width height
##  124.7   95.4   46.3
```

```
turtles.pca=princomp(turtles3var)
print(turtles.pca)
```

```
## Call:
## princomp(x = turtles3var)
##
## Standard deviations:
## Comp.1 Comp.2 Comp.3
##  25.06   2.26   1.94
##
##  3  variables and  48 observations.
```

```
(25.06^2+2.26^2+1.94^2)*(48/47)
```

```
## [1] 650
```

```
apply(turtles3var,2,var)
```

```
## length  width height
##  419.5  160.7   70.4
```

```
apply(turtles[,-1],2,sd)
```

```
## length  width height
##  20.48  12.68   8.39
```

```
turtlesc=scale(turtles[,-1])
cor(turtlesc)
```

```
##        length width height
## length  1.000 0.978  0.965
## width   0.978 1.000  0.961
## height  0.965 0.961  1.000
```

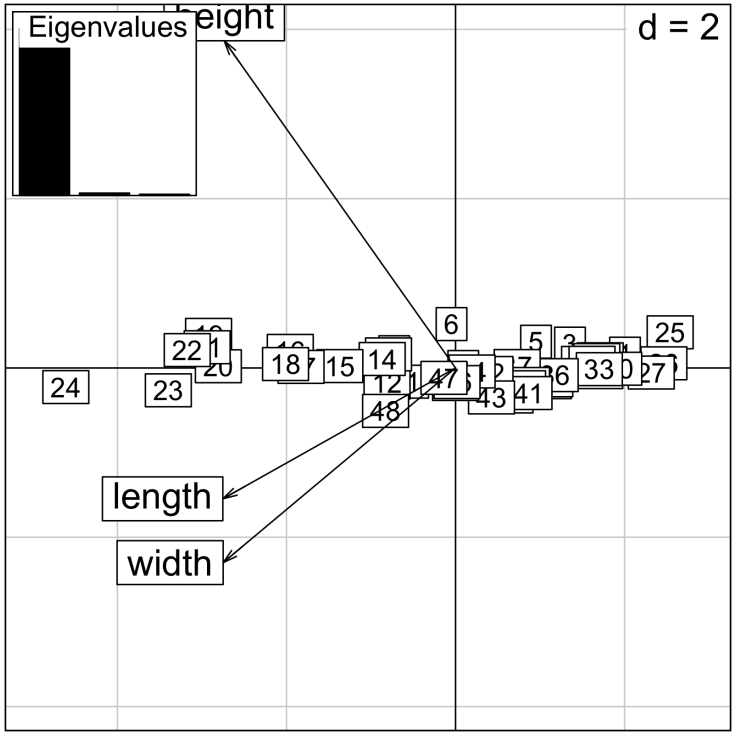
```
pca1=princomp(turtlesc)
pca1
```

```
## Call:
## princomp(x = turtlesc)
##
## Standard deviations:
## Comp.1 Comp.2 Comp.3
##  1.695  0.205  0.145
##
##  3  variables and  48 observations.
```

Step one: always the screeplot

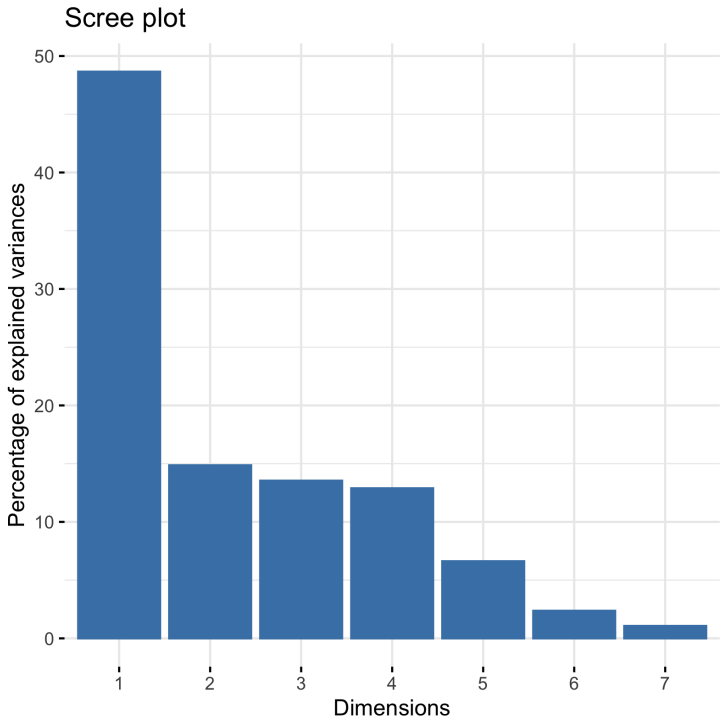
The screeplot showing the eigenvalues for the standardized data: one very large component in this case and two very small ones, the data are (almost) one dimensional.

```
pca.turtles=dudi.pca(turtles[,-1],scannf=F,nf=2)
scatter(pca.turtles)
```

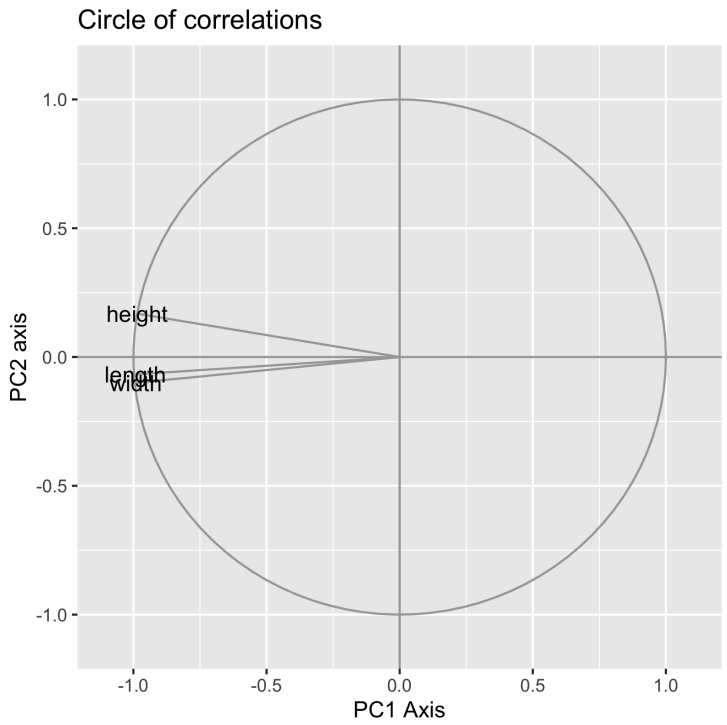


Why ?

Choose k carefully:

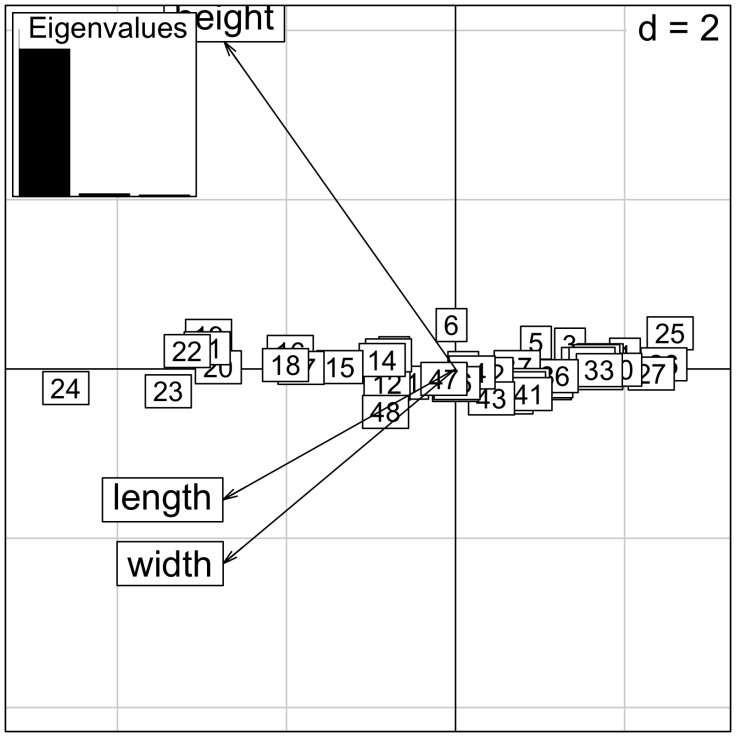


Step Two: Variables



All together “biplot”

```
scatter(pca.turtles)
```



Lizards Data Analyses

This data set describes 18 lizards as reported by Bauwens and D'iaz-Uriarte (1997). It also gives life-history traits corresponding to these 18 species.

- `mean.L` (mean length (mm)), `matur.L` (length at maturity (mm)),
- `max.L` (maximum length (mm)), `hatch.L` (hatchling length (mm)),
- `hatch.m` (hatchling mass (g)), `clutch.s` (Clutch size),
- `age.mat` (age at maturity (number of months of activity)),
- `clutch.F` (clutch frequency).

```
library(ade4)
data(lizards)
names(lizards)
```

```
## [1] "traits" "hprA"   "hprB"
```

```
lizards$traits[1:4,]
```

```
##    mean.L matur.L max.L hatch.L hatch.m clutch.S age.mat clutch.F
## Sa   69.2      58    82    27.8   0.572      6.0      13      1.5
## Sh   48.4      42    56    22.9   0.310      3.2       5      2.0
## Tl  168.4     132   190    42.8   2.235     16.9      19      1.0
## Mc   66.1      56    72    25.0   0.441      7.2      11      1.5
```

It is always a good idea to check the variables one at a time and two at a time to see what the basic statistics are for the data

```
tabtraits=lizards$traits
options(digits=2)
colMeans(tabtraits)
```

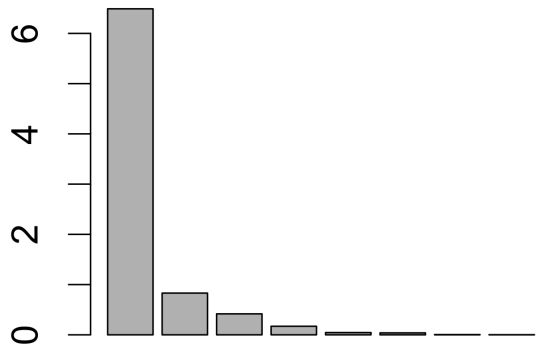
```
##   mean.L  matur.L    max.L  hatch.L  hatch.m clutch.S  age.mat clutch.F
##    71.34    59.39    82.83    26.88     0.56     5.87    10.89     1.56
```

```
cor(tabtraits)
```

```
##          mean.L matur.L max.L hatch.L hatch.m clutch.S age.mat clutch.F
## mean.L     1.00    0.99  0.99    0.89    0.94     0.92    0.77    -0.48
## matur.L    0.99    1.00  0.99    0.90    0.92     0.92    0.79    -0.49
## max.L      0.99    0.99  1.00    0.88    0.92     0.91    0.78    -0.51
## hatch.L    0.89    0.90  0.88    1.00    0.96     0.72    0.58    -0.42
## hatch.m    0.94    0.92  0.92    0.96    1.00     0.78    0.64    -0.45
## clutch.S   0.92    0.92  0.91    0.72    0.78     1.00    0.81    -0.55
## age.mat    0.77    0.79  0.78    0.58    0.64     0.81    1.00    -0.62
## clutch.F  -0.48   -0.49 -0.51   -0.42   -0.45    -0.55   -0.62     1.00
```

Biplot

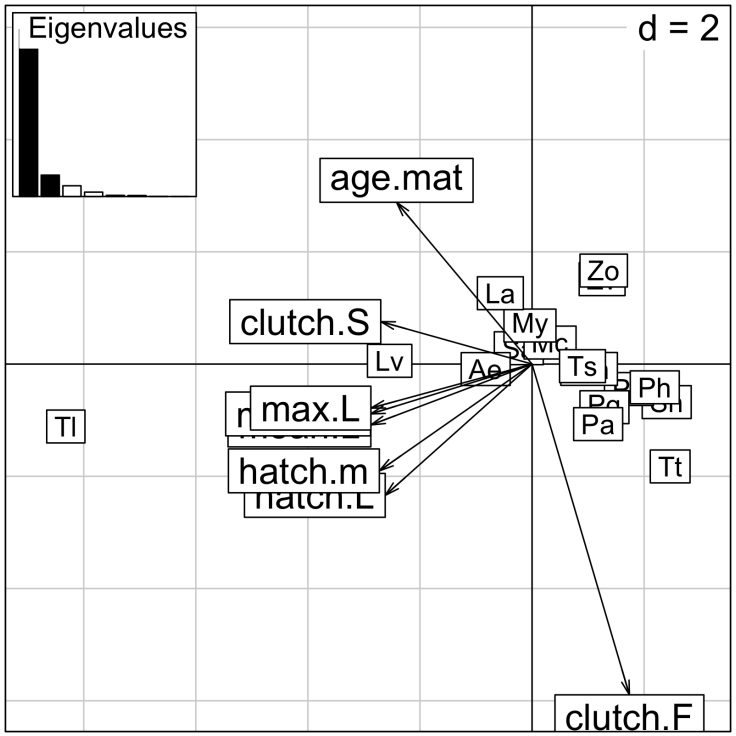
```
require(ade4)
res=dudi.pca(tabtraits,scannf=F,nf=2)
barplot(res$eig)
```




```
res
```

```
## Duality diagramm
## class: pca dudi
## $call: dudi.pca(df = tabtraits, scannf = F, nf = 2)
##
## $nf: 2 axis-components saved
## $rank: 8
## eigen values: 6.5 0.83 0.42 0.17 0.045 ...
##   vector length mode    content
## 1 $cw    8      numeric column weights
## 2 $lw    18     numeric row weights
## 3 $eig   8      numeric eigen values
##
##   data.frame nrow ncol content
## 1 $tab       18   8    modified array
## 2 $li        18   2    row coordinates
## 3 $l1        18   2    row normed scores
## 4 $co        8    2    column coordinates
## 5 $c1        8    2    column normed scores
## other elements: cent norm
```

```
biplot(res)
```




```
res$eig/(sum(res$eig))
```

```
## [1] 0.81118 0.10387 0.05219 0.02133 0.00563 0.00488 0.00061 0.00031
```

The Decathlon Athletes

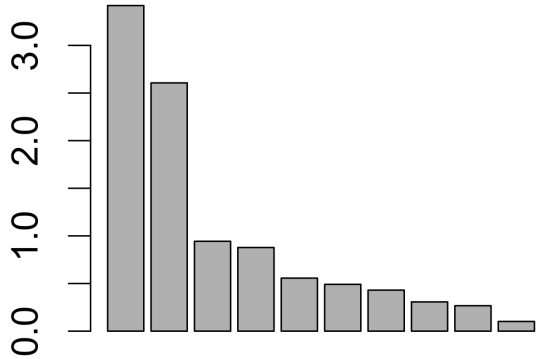
```
round(cor(athletes),1)
```

```
##        m100 long weight highj m400 m110 disc pole javel m1500
## m100    1.0 -0.5   -0.2  -0.1  0.6  0.6  0.0 -0.4  -0.1   0.3
## long   -0.5  1.0    0.1   0.3 -0.5 -0.5  0.0  0.3   0.2  -0.4
## weight -0.2  0.1    1.0   0.1  0.1 -0.3  0.8  0.5   0.6   0.3
## highj  -0.1  0.3    0.1   1.0 -0.1 -0.3  0.1  0.2   0.1  -0.1
## m400    0.6 -0.5    0.1  -0.1  1.0  0.5  0.1 -0.3   0.1   0.6
## m110    0.6 -0.5   -0.3  -0.3  0.5  1.0 -0.1 -0.5  -0.1   0.1
## disc    0.0  0.0    0.8   0.1  0.1 -0.1  1.0  0.3   0.4   0.4
## pole   -0.4  0.3    0.5   0.2 -0.3 -0.5  0.3  1.0   0.3   0.0
## javel  -0.1  0.2    0.6   0.1  0.1 -0.1  0.4  0.3   1.0   0.1
## m1500   0.3 -0.4    0.3  -0.1  0.6  0.1  0.4  0.0   0.1   1.0
```

```
pca.ath <- dudi.pca(athletes, scan = F)
pca.ath$eig
```

```
##  [1] 3.42 2.61 0.94 0.88 0.56 0.49 0.43 0.31 0.27 0.10
```

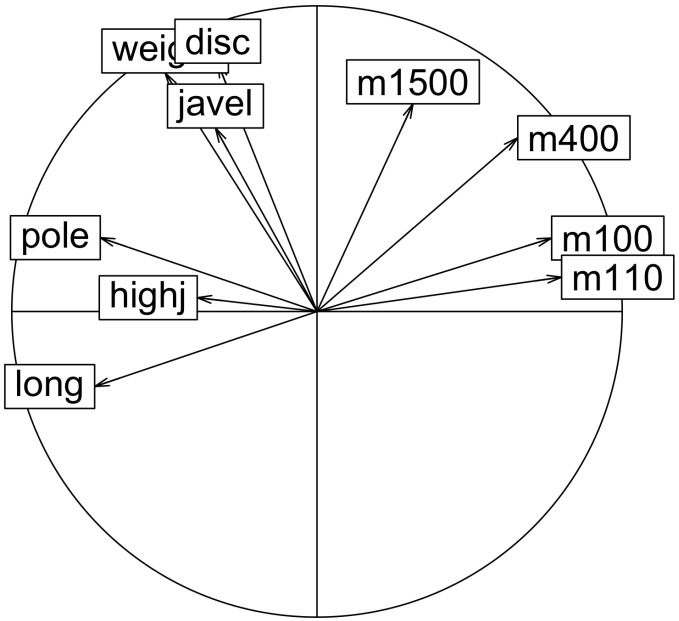
```
barplot(pca.ath$eig)
```



The screeplot is the first thing to look at, it tells us that it is satifactory to use a two dimensional plot.

Correlation Circle

```
s.corcircle(pca.ath$co,clab=1, grid=FALSE, fullcircle = TRUE,box=FALSE)
```



The correlation circle made by showing the projection of the old variables onto the two first new principal axes.

```
athletes[,c(1,5,6,10)]=-athletes[,c(1,5,6,10)]
round(cor(athletes),1)
```

```
##         m100 long weight highj m400 m110 disc pole javel m1500
## m100     1.0  0.5    0.2   0.1  0.6  0.6  0.0  0.4   0.1   0.3
## long     0.5  1.0    0.1   0.3  0.5  0.5  0.0  0.3   0.2   0.4
## weight   0.2  0.1    1.0   0.1 -0.1  0.3  0.8  0.5   0.6  -0.3
## highj    0.1  0.3    0.1   1.0  0.1  0.3  0.1  0.2   0.1   0.1
## m400     0.6  0.5   -0.1   0.1  1.0  0.5 -0.1  0.3  -0.1   0.6
## m110     0.6  0.5    0.3   0.3  0.5  1.0  0.1  0.5   0.1   0.1
## disc     0.0  0.0    0.8   0.1 -0.1  0.1  1.0  0.3   0.4  -0.4
## pole     0.4  0.3    0.5   0.2  0.3  0.5  0.3  1.0   0.3   0.0
## javel    0.1  0.2    0.6   0.1 -0.1  0.1  0.4  0.3   1.0  -0.1
## m1500    0.3  0.4   -0.3   0.1  0.6  0.1 -0.4  0.0  -0.1   1.0
```

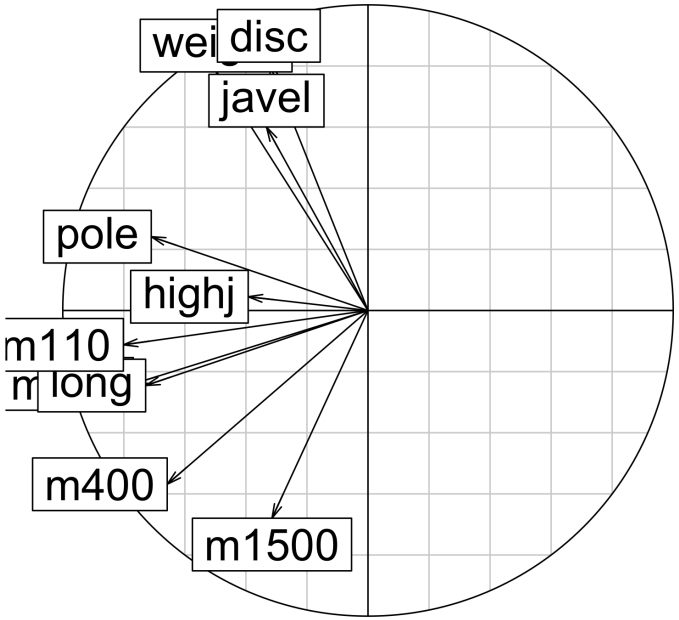
```
pcan.ath=dudi.pca(athletes,nf=2,scannf=F)
pcan.ath$eig
```

```
##  [1] 3.42 2.61 0.94 0.88 0.56 0.49 0.43 0.31 0.27 0.10
```

Now all the negative correlations are quite small ones. Doing the screeplot over again will show no change in the eigenvalues, the only thing that changes is the sign of loadings for the m variables.

New Data changing signs

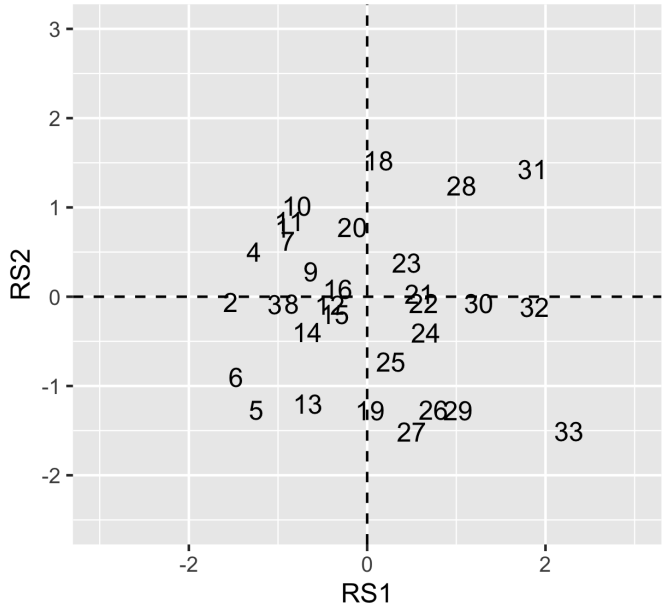
```
s.corcircle(pcan.ath$co,clab=1.2,box=FALSE)
```



Correlation circle after changing the signs of the running variables.

Observations

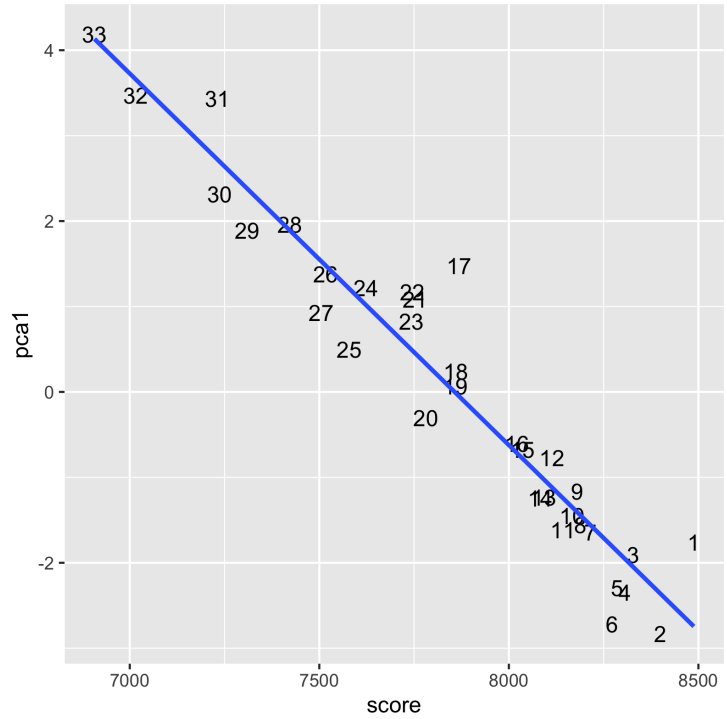
```
## Warning: Removed 1 rows containing missing values (geom_text).
```




```
data(olympic)
olympic$score
```

```
##  [1] 8488 8399 8328 8306 8286 8272 8216 8189 8180 8167 8143 8114 8093 8083 8036
## [16] 8021 7869 7860 7859 7781 7753 7745 7743 7623 7579 7517 7505 7422 7310 7237
## [31] 7231 7016 6907
```

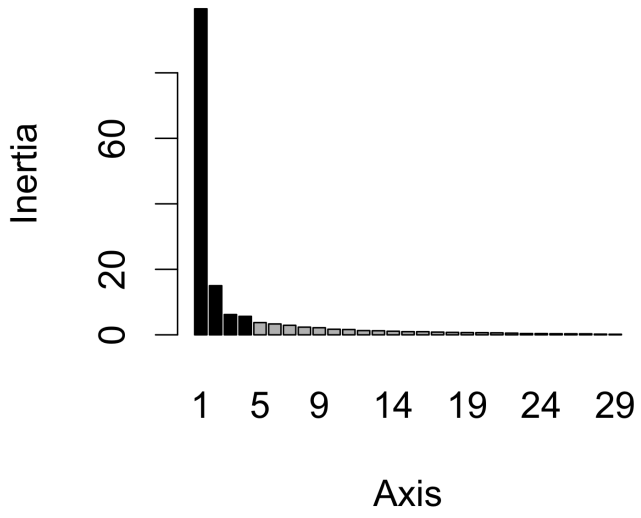
Link to overall scores



Scatterplot of the scores given as a supplementary variable and the first principal component, the points are labeled by their order in the data set.

PCA as an exploratory tool: using meta-information

```
######center and scale the data
###(they have already had variance normalization applied to them)
res.Msig3=dudi.pca(Msig3transp,center=TRUE,scale=TRUE,scannf=F,nf=4)
screeplot(res.Msig3,main="")
```

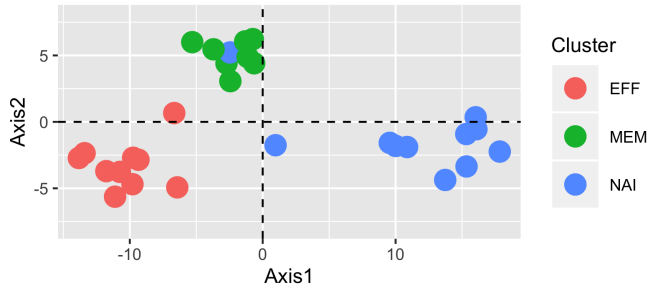


Plot by cell types

```
celltypes=factor(substr(rownames(Msig3transp),7,9))
table(celltypes)
```

```
## celltypes
## EFF MEM NAI
##  10   9  11
```

```
status=factor(substr(rownames(Msig3transp),1,3))
require(ggplot2)
gg <- cbind(res.Msig3$li,Cluster=celltypes)
gg <- cbind(sample=rownames(gg),gg)
ggplot(gg, aes(x=Axis1, y=Axis2)) +
  geom_point(aes(color=factor(Cluster)),size=5) +
  geom_hline(yintercept=0,linetype=2) +
  geom_vline(xintercept=0,linetype=2) +
  scale_color_discrete(name="Cluster") +
  coord_fixed()+ ylim(-8,+8)
```




```
  xlim(-14,18)
```

```
## <ScaleContinuousPosition>
##  Range:
##  Limits:  -14 --   18
```

PCA of gene expression for a subset of 156 genes involved in specificities of each of the three separate T cell types: effector, naive and memory

Mass Spectroscopy Data Analysis

Example from paper: Kashnap et al, PNAS, 2013 (http://www.pnas.org/content/110/42/17059.full)

```
###Just for record, this is how the matrix was made
require(xcms)
```

```
## Loading required package: xcms
```

```
## Loading required package: Biobase
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following object is masked from 'package:ade4':
##
##     score
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which, which.max, which.min
```

```
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
## Loading required package: BiocParallel
```

```
## Loading required package: MSnbase
```

```
## Loading required package: mzR
```

```
## Loading required package: Rcpp
```

```
## Loading required package: S4Vectors
```

```
## Loading required package: stats4
```

```
##
## Attaching package: 'S4Vectors'
```

```
## The following object is masked from 'package:base':
##
##     expand.grid
```

```
## Loading required package: ProtGenerics
```

```
##
## This is MSnbase version 2.10.1
##   Visit https://lgatto.github.io/MSnbase/ to get started.
```

```
##
## Attaching package: 'MSnbase'
```

```
## The following object is masked from 'package:stats':
##
##     smooth
```

```
## The following object is masked from 'package:base':
##
##     trimws
```

```
##
## This is xcms version 3.6.1
```

```
##
## Attaching package: 'xcms'
```

```
## The following object is masked from 'package:knitr':
##
##     stitch
```

```
## The following object is masked from 'package:stats':
##
##     sigma
```

```
load(url("http://bios221.stanford.edu/data/xset3.RData"))
mat1 =groupval(xset3, value="into")
##
head(mat1)
```

```
##               ko15    ko16    ko18    ko19    ko21    ko22    wt15    wt16
## 200.1/2927  147888  103548   65290   60144   85156  162012  175177   82619
## 205/2791   1778569 1567038 1482796 1039130 1223132 1072038 1950287 1466781
## 206/2791    237994  269714  201393  150107  176990  156797  276542  222366
## 207.1/2719  380873  460630  351750  219288  286849  235023  417170  324892
## 219.1/2524  235545  173623   82365   79480  185792  174459  244584  161184
## 231/2516         0   70796  222609  286232  435094  100076       0   73142
##                wt18    wt19    wt21    wt22
## 200.1/2927   51943   69198  153273   98144
## 205/2791   1572679 1275313 1356014 1231442
## 206/2791    211718  186851  188286  172349
## 207.1/2719  277991  220972  252874  236728
## 219.1/2524   72029   75097  238194  173830
## 231/2516    165383  240261  201316  179438
```

```
dim(mat1)
```

```
## [1] 399  12
```

```
## Matrix with with samples in rows and variables as columns
tmat= t(mat1)
head(tmat[,1:10])
```

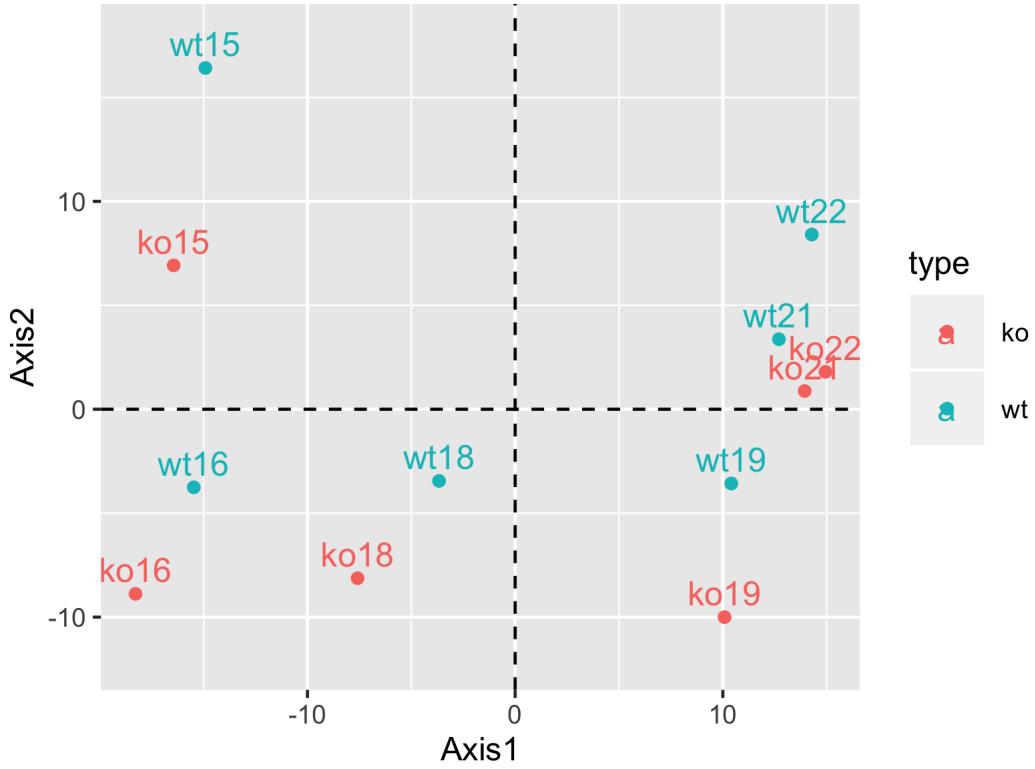
```
##      200.1/2927 205/2791 206/2791 207.1/2719 219.1/2524 231/2516 233/3023
## ko15     147888  1778569   237994     380873     235545        0   399145
## ko16     103548  1567038   269714     460630     173623    70796   356951
## ko18      65290  1482796   201393     351750      82365   222609   410551
## ko19      60144  1039130   150107     219288      79480   286232   198417
## ko21      85156  1223132   176990     286849     185792   435094   363382
## ko22     162012  1072038   156797     235023     174459   100076   317806
##      234/3024 236.1/2524 240.2/3681
## ko15    76881     252282     112441
## ko16    99480     206032     153376
## ko18    97428      71764     193769
## ko19    53440      67643     170641
## ko21    88228     186661      88800
## ko22    81072     198804     146563
```

```
logtmat=log(tmat+1)
```

Sample situations in PC map

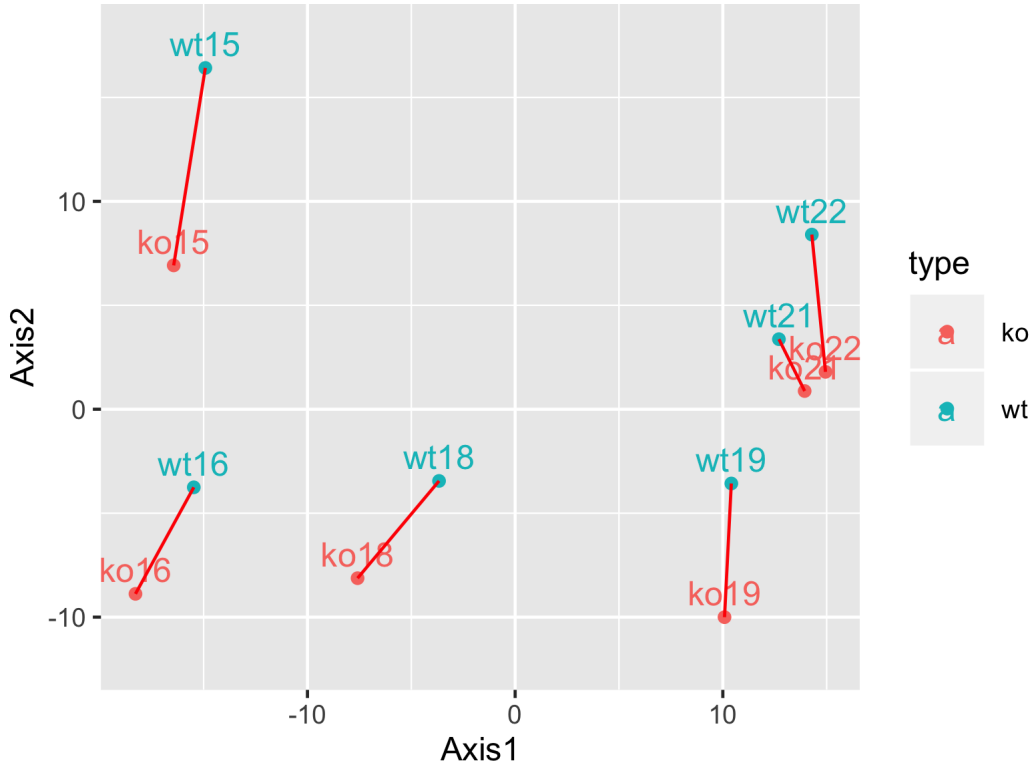
```
##  PCA Example
require(ade4)
require(ggplot2)
load(url("http://bios221.stanford.edu/data/logtmat.RData"))
pca.result=dudi.pca(logtmat, scannf=F,nf=3)
labs=rownames(pca.result$li)
nos=substr(labs,3,4)
type=as.factor(substr(labs,1,2))
kos=which(type=="ko")
wts=which(type=="wt")
pcs=data.frame(Axis1=pca.result$li[,1],Axis2=pca.result$li[,2],labs,type)

pcsplot=ggplot(pcs,aes(x=Axis1,y=Axis2,label=labs,group=nos,colour=type)) +
  geom_text(size=4,vjust=-0.5) + geom_point()
 pcsplot +  geom_hline(yintercept=0,linetype=2)  +coord_fixed() + ylim(-12,18) +
  geom_vline(xintercept=0,linetype=2)
```



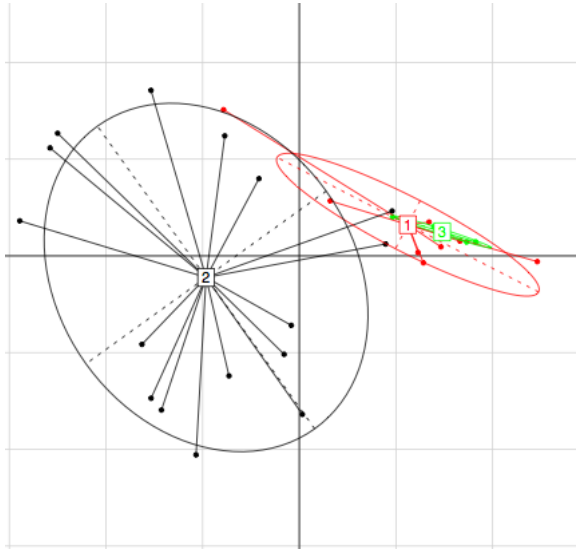
Extra Connections

```
pcsplot+geom_line(colour="red")  + coord_fixed()  + ylim(-12,18)
```



Checking data by frequent multivariate projections

Phylochip data allowed us to discover a batch effect (phylochip).



Phylochip data for three different batches and two different arrays, first principal plane explains 66% of the total variation.

Example from Single Cell experiment

Columns of the *DataFrame* represent different attributes of the features of interest, e.g., gene or transcript IDs, etc.

An example of hybrid data container from single cell experiments (see Bioconductor workflow in Perraudeau, 2017 for more details).

After the pre-processing and normalization steps prescribed in the workflow, we retain the 1,000 most variable genes measured on 747 cells.

```
require(SummarizedExperiment)
```

```
## Loading required package: SummarizedExperiment
```

```
## Loading required package: GenomicRanges
```

```
## Loading required package: IRanges
```

```
##
## Attaching package: 'IRanges'
```

```
## The following object is masked from 'package:xcms':
##
##     distance
```

```
## Loading required package: GenomeInfoDb
```

```
## Loading required package: DelayedArray
```

```
## Loading required package: matrixStats
```

```
##
## Attaching package: 'matrixStats'
```

```
## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians
```

```
##
## Attaching package: 'DelayedArray'
```

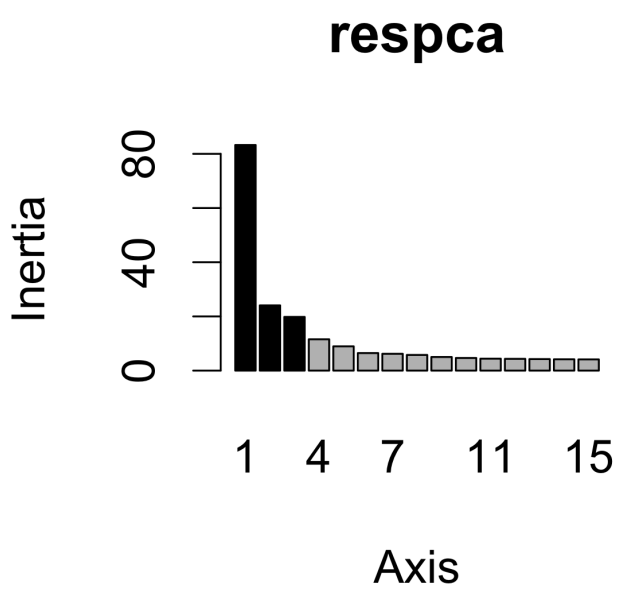
```
## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges
```

```
## The following objects are masked from 'package:base':
##
##     aperm, apply, rowsum
```

```
corese = readRDS(path.expand("~/Books/CUBook/data/normse.rds"))
norm = assays(corese)$normalizedValues
```

We can look at a PCA of the normalized values and check graphically that the batch effect has been removed:

```
respca = dudi.pca(t(norm), nf = 3, scannf = FALSE)
screeplot(respca, 15)
```



Screeplot of the PCA of the normalized data.

```
PCS = respca$li[, 1:3]
```

Since the screeplot shows us that we must not dissociate axes 2 and 3, we will make a three dimensional plot with the **rgl (https://cran.r-project.org/web/packages/rgl/)** package.

```
library("rgl")
batch = colData(corese)$Batch
plot3d(PCS,aspect=sqrt(c(84,24,20)),col=col_clus[batch])
plot3d(PCS,aspect=sqrt(c(84,24,20)),
col = col_clus[as.character(publishedClusters)])
```

With plotly:

```
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:IRanges':
##
##     slice
```

```
## The following object is masked from 'package:xcms':
##
##     groups
```

```
## The following object is masked from 'package:S4Vectors':
##
##     rename
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
p <- plot_ly(PCS,x=~Axis1,y=~Axis2,z=~Axis3,color=batch) %>% add_markers()
```

Summary for PCA, takeaway points so far:

- Multivariate data require `conscious` preprocessing, to make their variances comparable and their centers at the origin.
- When data are matrices (variables = columns numerical values), we can still make useful graphical representations by making projections on lower dimensions (planes and 3D are the most frequently used).
- PCA searches for new `more informative` variables which are linear combinations of the old ones.
- PCA is based on finding decompositions of the matrix X called SVD, this is equivalent to the eigenanalysis of $X'X$. The squares of the singular values are the

equal to the eigenvalues and to the variances of the new variables.
- Choosing k: You need to plot the variances/eigenvalues before you decide how many axes are necessary to reproduce the signal in the data.
- Interpretation of PCA is facilitated by redundant or contiguous meta-data about the observations.

See all the details here: Chapter on Multivariate (http://bios221.stanford.edu/book/Chap-Multivariate.html)

More examples of supplementary variables

One categorical variable: project the mean points

```
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data"
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────── tidyverse 1.2.1 ──
```

```
## ✔ tibble  2.1.3     ✔ purrr   0.3.2
## ✔ tidyr   0.8.3     ✔ dplyr   0.8.3
## ✔ readr   1.3.1     ✔ stringr 1.4.0
## ✔ tibble  2.1.3     ✔ forcats 0.4.0
```

```
## ── Conflicts ────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::collapse()        masks IRanges::collapse()
## ✖ dplyr::collect()         masks xcms::collect()
## ✖ dplyr::combine()         masks MSnbase::combine(), Biobase::combine(), BiocGenerics::co
mbine()
## ✖ dplyr::count()           masks matrixStats::count()
## ✖ dplyr::desc()            masks IRanges::desc()
## ✖ tidyr::expand()          masks S4Vectors::expand()
## ✖ dplyr::filter()          masks plotly::filter(), stats::filter()
## ✖ dplyr::first()           masks S4Vectors::first()
## ✖ dplyr::groups()          masks plotly::groups(), xcms::groups()
## ✖ dplyr::lag()             masks stats::lag()
## ✖ BiocGenerics::Position() masks ggplot2::Position(), base::Position()
## ✖ purrr::reduce()          masks GenomicRanges::reduce(), IRanges::reduce(), MSnbase::red
uce()
## ✖ dplyr::rename()          masks plotly::rename(), S4Vectors::rename()
## ✖ purrr::simplify()        masks DelayedArray::simplify()
## ✖ dplyr::slice()           masks plotly::slice(), IRanges::slice()
```

```
wine <- read_csv(url, col_names = FALSE)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double(),
##   X9 = col_double(),
##   X10 = col_double(),
##   X11 = col_double(),
##   X12 = col_double(),
##   X13 = col_double(),
##   X14 = col_double()
## )
```

```
colnames(wine) <- c("class", "Alcohol", "MalicAcid", "Ash", "AlcAsh", "Mg",
                    "Phenols", "Flav", "NonFlavPhenols", "Proa", "Color",
                    "Hue", "OD", "Proline")
head(wine)
```

```
## # A tibble: 6 x 14
##   class Alcohol MalicAcid   Ash AlcAsh    Mg Phenols  Flav NonFlavPhenols  Proa
##   <dbl>   <dbl>     <dbl> <dbl>  <dbl> <dbl>   <dbl> <dbl>          <dbl> <dbl>
## 1     1    14.2      1.71  2.43   15.6   127    2.8   3.06           0.28  2.29
## 2     1    13.2      1.78  2.14   11.2   100    2.65  2.76           0.26  1.28
## 3     1    13.2      2.36  2.67   18.6   101    2.8   3.24           0.3   2.81
## 4     1    14.4      1.95  2.5    16.8   113    3.85  3.49           0.24  2.18
## 5     1    13.2      2.59  2.87   21     118    2.8   2.69           0.39  1.82
## 6     1    14.2      1.76  2.45   15.2   112    3.27  3.39           0.34  1.97
## # … with 4 more variables: Color <dbl>, Hue <dbl>, OD <dbl>, Proline <dbl>
```

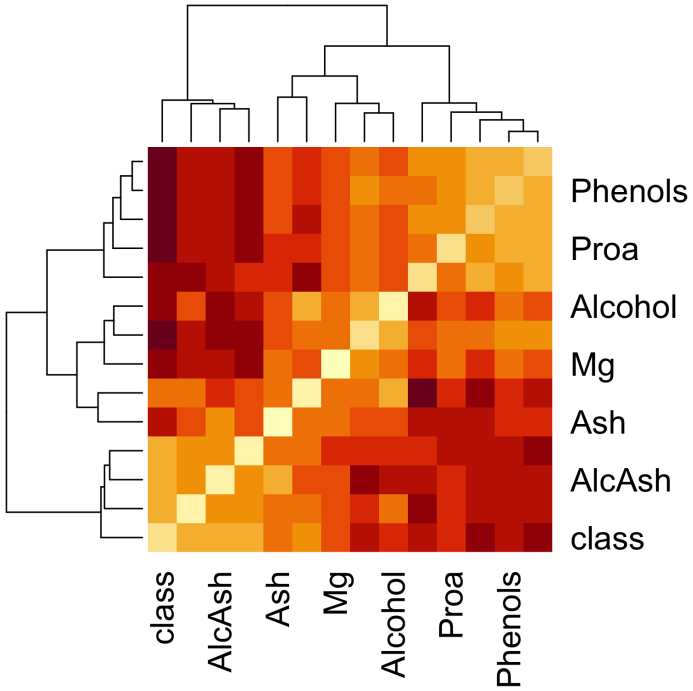
```
data(wine)
```

```
## Warning in data(wine): data set 'wine' not found
```

```
wine[1:3,1:7]
```

```
## # A tibble: 3 x 7
##   class Alcohol MalicAcid   Ash AlcAsh    Mg Phenols
##   <dbl>   <dbl>     <dbl> <dbl>  <dbl> <dbl>   <dbl>
## 1     1    14.2      1.71  2.43   15.6   127    2.8
## 2     1    13.2      1.78  2.14   11.2   100    2.65
## 3     1    13.2      2.36  2.67   18.6   101    2.8
```

```
heatmap(1-cor(wine))
```




```
wine.pca <- prcomp(wine, scale. = TRUE)
table(wine.class)
```

```
## Error in eval(quote(list(...)), env): object 'wine.class' not found
```

```
fviz_pca_biplot(wine.pca,
                habillage = wine.class, addEllipses = TRUE, circle = TRUE)
```

```
## Error in .is_grouping_var(habillage): object 'wine.class' not found
```

Projecting Ellipses

We'll see later when we look at Microbiome data that sometimes, this projection can be problematic.

Percentage of Inertia

```
require(ade4)
res.ath=dudi.pca(athletes,nf=2,scannf=F)
inertia.dudi(res.ath,col.inertia=TRUE)
```

```
## Inertia information:
## Call: inertia.dudi(x = res.ath, col.inertia = TRUE)
##
## Decomposition of total inertia:
##       inertia     cum  cum(%)
## Ax1    3.4182   3.418   34.18
## Ax2    2.6064   6.025   60.25
## Ax3    0.9433   6.968   69.68
## Ax4    0.8780   7.846   78.46
## Ax5    0.5566   8.403   84.03
## Ax6    0.4912   8.894   88.94
## Ax7    0.4306   9.324   93.24
## Ax8    0.3068   9.631   96.31
## Ax9    0.2669   9.898   98.98
## Ax10   0.1019  10.000  100.00
##
## Column contributions (%):
##    m100    long  weight   highj    m400    m110    disc    pole   javel   m1500
##      10      10      10      10      10      10      10      10      10      10
##
## Column absolute contributions (%):
##          Axis1    Axis2
## m100    17.296  2.21439
## long    15.528  2.31288
## weight   7.242 23.38084
## highj    4.506  0.07783
## m400    12.663 12.40165
## m110    18.791  0.48397
## disc     3.090 25.33458
## pole    14.752  2.23748
## javel    3.238 13.83520
## m1500    2.895 17.72118
##
## Signed column relative contributions:
##          Axis1    Axis2
## m100   -59.121  -5.7716
## long   -53.077  -6.0283
## weight -24.754  60.9397
## highj  -15.404   0.2029
## m400   -43.284 -32.3236
## m110   -64.231  -1.2614
## disc   -10.563  66.0319
## pole   -50.426   5.8317
## javel  -11.068  36.0600
## m1500   -9.895 -46.1884
##
## Cumulative sum of column relative contributions (%):
##          Axis1 Axis1:2 Axis3:10
## m100    59.121   64.89    35.11
## long    53.077   59.11    40.89
```

```
## weight  24.754   85.69    14.31
## highj   15.404   15.61    84.39
## m400    43.284   75.61    24.39
## m110    64.231   65.49    34.51
## disc    10.563   76.60    23.40
## pole    50.426   56.26    43.74
## javel   11.068   47.13    52.87
## m1500    9.895   56.08    43.92
```

Contributions are printed in 1/10000 and the sign is the sign of the coordinate.

Principal Coordinate Analysis (PCoA) and MDS

- Different starting point: distances instead of measurements/columns.
- Resulting `map' projections, same as PCA.
- Interpretation is different.

Important Distances and Dissimilarities

Most of these are available through the `dist` function, the `vegdist` function in the `vegan` package:

```
library(vegan)
```

```
## Loading required package: permute
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:xcms':
##
##     levelplot
```

```
## This is vegan 2.5-5
```

```
##
## Attaching package: 'vegan'
```

```
## The following object is masked from 'package:xcms':
##
##     calibrate
```

```
## The following object is masked from 'package:mzR':
##
##     tolerance
```

```
help(vegdist)
```

Reminder: Distances using continuous variables

- Euclidean and weighted Euclidean (sums of squares of differences in coordinates)
- L1 distances (`manhattan`)
- Minkowski
- Chisquare:

$$\text{Chisquare}(exp, obs) = \sum_j \frac{(exp_j - obs_j)^2}{exp_j}$$

Distances using discrete or binary variables

- Hamming distance
- DNA distances (`dist.dna` in `ape`)
- Bray Curtis (absolute difference of proportions)
- Built from Confusion matrices (not a true distance)
- Matching coefficient: Matching coefficient

$$\frac{\text{nb of matching attrs}}{\text{nb of attrs}} = \frac{f_{11} + f_{00}}{f_{11} + f_{00} + f_{10} + f_{01}}$$

- Jaccard distance

$$\frac{\text{nb of match.attrs}}{\text{nb of attrs with at least 1}} = \frac{f_{11}}{f_{11} + f_{10} + f_{01}}$$

Example of Distances}

```
SMC = function(p,q) {
   # Compute F01,F10,F11,F00
   F01 = sum((p == 0) & (q == 1))
   F10 = sum((p == 1) & (q == 0))
   F00 = sum((p == 0) & (q == 0))
   F11 = sum((p == 1) & (q == 1))
   return((F11+F00)/(F01+F10+F00+F11))
}

# function for computing Jaccard coefficient
JC = function(p,q) {
   # Compute F01,F10,F11,F00
   F01 = sum((p == 0) & (q == 1))
   F10 = sum((p == 1) & (q == 0))
   F11 = sum((p == 1) & (q == 1))
   return(F11/(F01+F10+F11))
}
```

```
cos.sim = function(p,q) {
   return(sum(p*q) / sqrt(sum(p^2)*sum(q^2)))
}
d1 = c(3,2,0,5,0,0,0,2,0,0)
d2 = c(1,0,0,0,0,0,0,1,0,2)
print(cos.sim(d1,d2))
```

```
## [1] 0.31
```

```
 p=c(rep(0,6),rep(1,4))
 p
```

```
##  [1] 0 0 0 0 0 0 1 1 1 1
```

```
 q=c(rep(0,6),1,0,0,1)
 q
```

```
##  [1] 0 0 0 0 0 0 1 0 0 1
```

```
print(JC(p,q))
```

```
## [1] 0.5
```

```
print(SMC(p,q))
```

```
## [1] 0.8
```

Distances between variables

- Pearsons correlation coefficient: $d(X, Y) = 1 - r(X, Y)$ where

$$r(X, Y) = \frac{1}{n} \sum_{i=1}^{n} (\frac{X(i) - \bar{X}}{\sigma_x})(\frac{Y(j) - \bar{Y}}{\sigma_y})$$

Special Distances

- Gower's distance for mixed type data.
- Unifrac and Weighted Unifrac (Wasserstein) that incorporates trees
- Distances on a graph

Distance function in R

```
require(ade4)
data(olympic)
disto=dist(scale(olympic$tab))
str(disto)
```

```
##  'dist' num [1:528] 4.36 4.11 4.18 5.19 4.28 ...
##  - attr(*, "Size")= int 33
##  - attr(*, "Labels")= chr [1:33] "1" "2" "3" "4" ...
##  - attr(*, "Diag")= logi FALSE
##  - attr(*, "Upper")= logi FALSE
##  - attr(*, "method")= chr "euclidean"
##  - attr(*, "call")= language dist(x = scale(olympic$tab))
```

```
length(disto)
```

```
## [1] 528
```

```
as.matrix(disto)[1:5,1:5]
```

```
##     1   2   3   4   5
## 1 0.0 4.4 4.1 4.2 5.2
## 2 4.4 0.0 1.9 2.2 2.4
## 3 4.1 1.9 0.0 3.2 2.2
## 4 4.2 2.2 3.2 0.0 4.0
## 5 5.2 2.4 2.2 4.0 0.0
```

```
33*32/2
```

```
## [1] 528
```

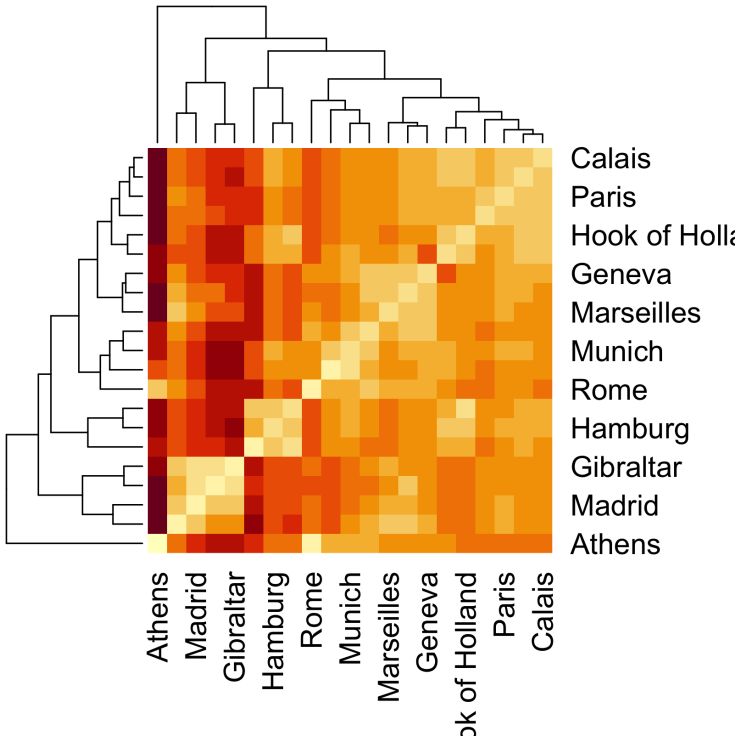
Multidimensional Scaling- Principal Coordinate Analyses

Suppose we are given a matrix of dissimilarities or distances and we want to build a **useful** map of the observations.

```
require(graphics)
data(eurodist)
# look at raw distances
as.matrix(eurodist)[1:7,1:7]
```

```
##            Athens Barcelona Brussels Calais Cherbourg Cologne Copenhagen
## Athens          0      3313     2963   3175      3339    2762       3276
## Barcelona    3313         0     1318   1326      1294    1498       2218
## Brussels     2963      1318        0    204       583     206        966
## Calais       3175      1326      204      0       460     409       1136
## Cherbourg    3339      1294      583    460         0     785       1545
## Cologne      2762      1498      206    409       785       0        760
## Copenhagen   3276      2218      966   1136      1545     760          0
```

```
# graphical representation
heatmap(as.matrix(eurodist))
```



These were computed as actualy distances across land so we actually know that we could find a 2-3 dimensional map that would represent the data well.

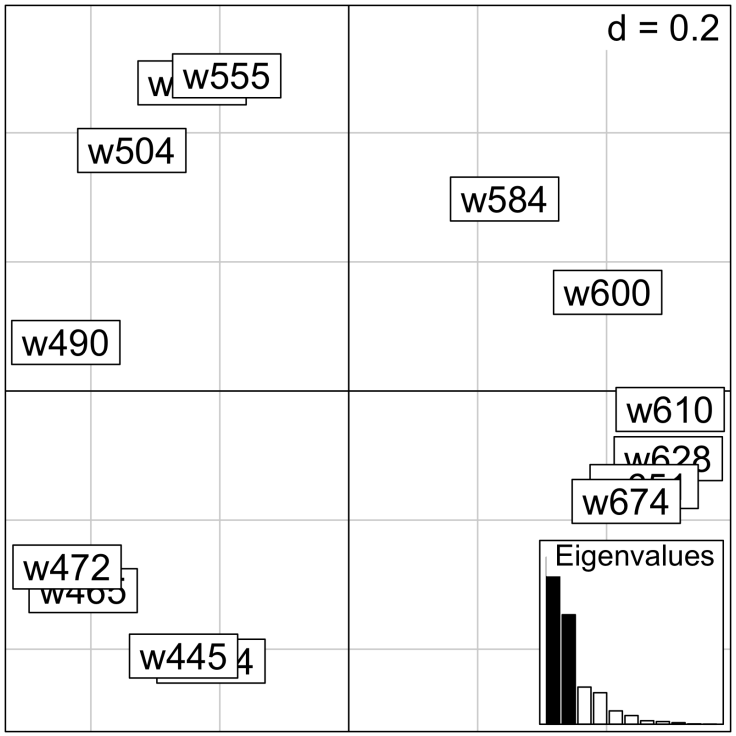
```
eck=read.table("http://bios221.stanford.edu/data/eckman.txt",header=TRUE)
nc=nrow(eck)
   eck[1:9,1:9]
```

```
##   w434 w445 w465 w472 w490 w504 w537 w555 w584
## 1 0.00 0.86 0.42 0.42 0.18 0.06 0.07 0.04 0.02
## 2 0.86 0.00 0.50 0.44 0.22 0.09 0.07 0.07 0.02
## 3 0.42 0.50 0.00 0.81 0.47 0.17 0.10 0.08 0.02
## 4 0.42 0.44 0.81 0.00 0.54 0.25 0.10 0.09 0.02
## 5 0.18 0.22 0.47 0.54 0.00 0.61 0.31 0.26 0.07
## 6 0.06 0.09 0.17 0.25 0.61 0.00 0.62 0.45 0.14
## 7 0.07 0.07 0.10 0.10 0.31 0.62 0.00 0.73 0.22
## 8 0.04 0.07 0.08 0.09 0.26 0.45 0.73 0.00 0.33
## 9 0.02 0.02 0.02 0.02 0.07 0.14 0.22 0.33 0.00
```

```
   require(ade4)
   queck=quasieuclid(as.dist(1-eck))
   eck.pco=dudi.pco(queck,scannf=F,nf=2)
   names(eck.pco)
```

```
##  [1] "eig"  "rank" "nf"   "cw"   "tab"  "li"   "l1"   "c1"   "co"   "lw"
## [11] "call"
```

```
   scatter(eck.pco,posi="bottomright")
```



Other available analysis and plotting options

```
require(vegan)
require(ggplot2)
ggscree=function(out){
  n=length(out)
  xs=1:n
  df=data.frame(eig=out,xs)
    ggplot(data=df, aes(x=xs, y=eig)) +
    geom_bar(stat="identity",width=0.3,color="red",fill="orange")
 }
res=cmdscale(as.dist(1-eck))
res
```

```
##       [,1]   [,2]
## w434 -0.21 -0.419
## w445 -0.26 -0.411
## w465 -0.41 -0.309
## w472 -0.44 -0.273
## w490 -0.44  0.075
## w504 -0.34  0.373
## w537 -0.24  0.477
## w555 -0.19  0.488
## w584  0.24  0.297
## w600  0.40  0.153
## w610  0.50 -0.029
## w628  0.50 -0.105
## w651  0.46 -0.148
## w674  0.43 -0.171
```

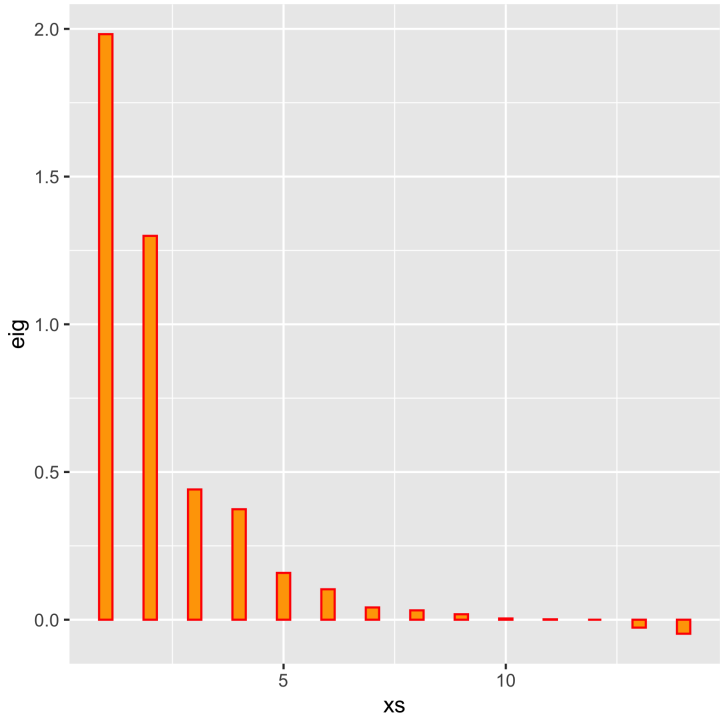
```
res=cmdscale(as.dist(1-eck),eig=TRUE)
names(res)
```

```
## [1] "points" "eig"    "x"      "ac"     "GOF"
```

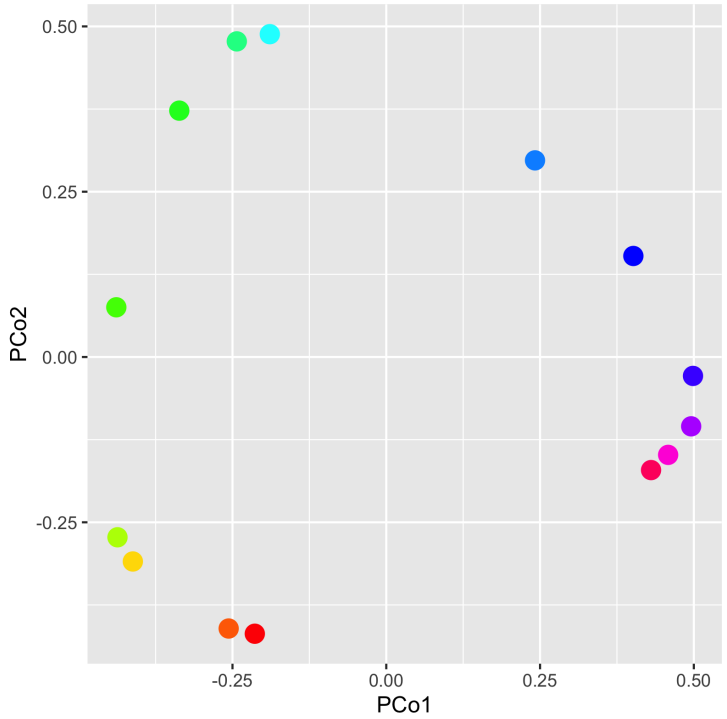
```
res$eig
```

```
##  [1]  2.0e+00  1.3e+00  4.4e-01  3.7e-01  1.6e-01  1.0e-01  4.2e-02  3.2e-02
##  [9]  1.8e-02  4.2e-03  1.3e-03  2.2e-16 -2.7e-02 -4.7e-02
```

```
out=res$eig
col14=rainbow(14)
ggscree(out)
```




```
MDS = data.frame(PCo1 = res$points[,1], PCo2 = res$points[,2])
ggplot(data = MDS, aes(PCo1, PCo2)) +
  geom_point(size=4,color = col14)
```



How does the method work?

From X to Distances D

Given a Euclidean distance D between the observation-rows.\ Call $B = XX'$, if $D^{(2)}$ is the matrix of squared distances between rows of X in the euclidean coordinates, we can show that

$$-\frac{1}{2}HD^{(2)}H = B$$

$$H = I - \frac{1}{n}11'$$

is the centering matrix.

$$d_{i,j} = \sqrt{(x_i^1 - x_j^1)^2 + \cdots + (x_i^p - x_j^p)^2}. \text{ and } -\frac{1}{2}HD^{(2)}H = B$$

Backward from D to X

We can go backwards from a matrix D to X by taking the eigendecomposition of B in much the same way that PCA provides the best rank r approximation for data by taking the singular value decomposition of X, or the eigendecomposition of XX'.

$$X^{(r)} = US^{(r)}V' \text{ with } S^{(r)} = \begin{pmatrix} s_1 & 0 & 0 & 0 & \dots \\ 0 & s_2 & 0 & 0 & \dots \\ 0 & 0 & \dots & \dots & \dots \\ 0 & 0 & \dots & s_r & \dots \\ \dots & \dots & \dots & 0 & 0 \end{pmatrix}$$

This provides the best approximate representation in an Euclidean space of dimension r. The algorithm provides points in a Euclidean space that have approximately the same distances as those provided by D^2.

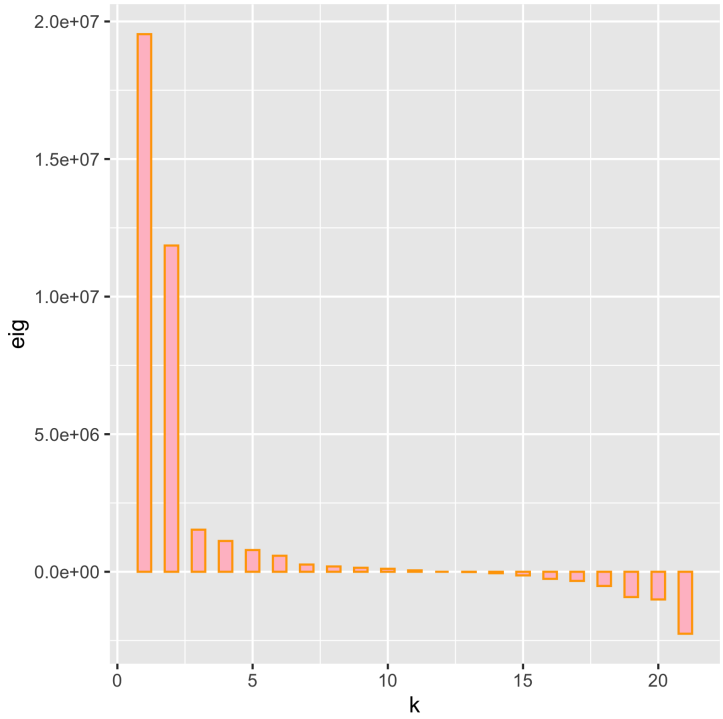
Classical MDS ALgorithm

In summary, given an $n \times n$ matrix of interpoint distances D, one can solve for points achieving these distances by: 1. Double centering the interpoint distance squared matrix: $B = -\frac{1}{2}HD^2H$. 2. Diagonalizing B: $B = U\Lambda U^T$. 3. Extracting $\tilde{X}$: $\tilde{X} = U\Lambda^{1/2}$.

Examples of output

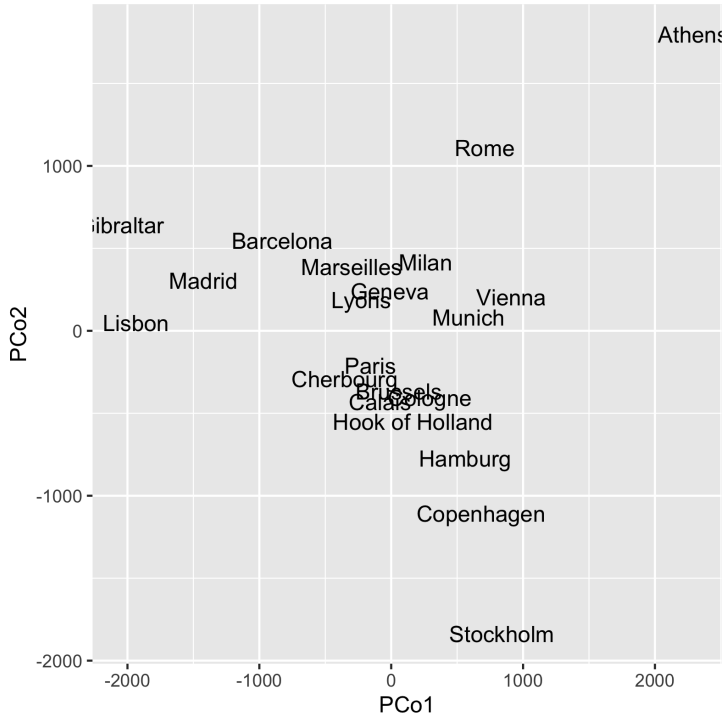
Screeplot for the Europe Data

```
res=cmdscale(eurodist,eig=TRUE)
n=length(res$eig)
out=data.frame(k=1:n,eig=res$eig)
ggplot(data=out, aes(x=k, y=eig)) +
  geom_bar(stat="identity",width=0.5,color="orange",fill="pink")
```



Position of Cities

```
MDS = data.frame(PCo1 = res$points[,1], PCo2 = res$points[,2],labs=rownames(res$points))
ggplot(data = MDS, aes(x=PCo1, y=PCo2,label=labs) ) + geom_text()
```



Notice the orientation. We also do not have `variables' to interpret.

Robust MDS: Non metric Multidimensional Scaling

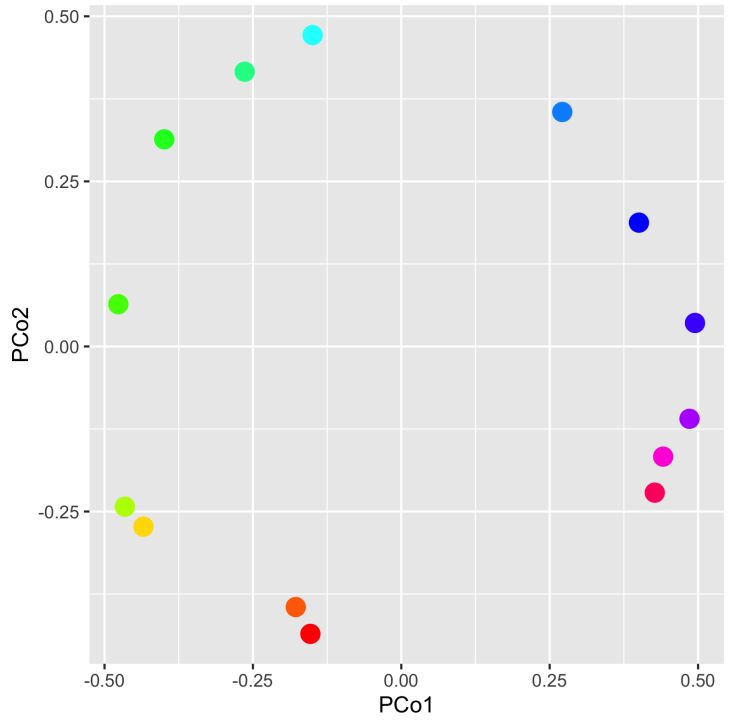
Respects the order of the distances, does not try to approximate their actual values.

```
require(vegan)
require(ggplot2)
res=metaMDS(as.dist(1-eck))
```

```
## Run 0 stress 0.023
## Run 1 stress 0.023
## ... Procrustes: rmse 2.2e-06  max resid 3.6e-06
## ... Similar to previous best
## Run 2 stress 0.023
## ... Procrustes: rmse 2.5e-06  max resid 4e-06
## ... Similar to previous best
## Run 3 stress 0.023
## ... Procrustes: rmse 5e-07  max resid 1.1e-06
## ... Similar to previous best
## Run 4 stress 0.023
## ... New best solution
## ... Procrustes: rmse 4.3e-07  max resid 8.2e-07
## ... Similar to previous best
## Run 5 stress 0.023
## ... Procrustes: rmse 1.8e-06  max resid 2.8e-06
## ... Similar to previous best
## Run 6 stress 0.023
## ... New best solution
## ... Procrustes: rmse 1.3e-06  max resid 2.1e-06
## ... Similar to previous best
## Run 7 stress 0.023
## ... Procrustes: rmse 1.2e-06  max resid 2.4e-06
## ... Similar to previous best
## Run 8 stress 0.023
## ... Procrustes: rmse 1.1e-06  max resid 1.8e-06
## ... Similar to previous best
## Run 9 stress 0.023
## ... Procrustes: rmse 1.7e-06  max resid 2.7e-06
## ... Similar to previous best
## Run 10 stress 0.023
## ... Procrustes: rmse 2.1e-06  max resid 4e-06
## ... Similar to previous best
## Run 11 stress 0.023
## ... Procrustes: rmse 1.1e-06  max resid 1.9e-06
## ... Similar to previous best
## Run 12 stress 0.023
## ... Procrustes: rmse 1.1e-06  max resid 2.1e-06
## ... Similar to previous best
## Run 13 stress 0.023
## ... Procrustes: rmse 4.7e-06  max resid 8e-06
## ... Similar to previous best
## Run 14 stress 0.023
## ... Procrustes: rmse 3.4e-06  max resid 5.6e-06
## ... Similar to previous best
## Run 15 stress 0.023
## ... New best solution
## ... Procrustes: rmse 1.4e-06  max resid 2.1e-06
## ... Similar to previous best
## Run 16 stress 0.023
```

```
## ... Procrustes: rmse 1.2e-06  max resid 2.3e-06
## ... Similar to previous best
## Run 17 stress 0.023
## ... Procrustes: rmse 3.9e-06  max resid 6.8e-06
## ... Similar to previous best
## Run 18 stress 0.023
## ... Procrustes: rmse 2.4e-06  max resid 3.6e-06
## ... Similar to previous best
## Run 19 stress 0.023
## ... New best solution
## ... Procrustes: rmse 8.9e-07  max resid 1.6e-06
## ... Similar to previous best
## Run 20 stress 0.023
## ... Procrustes: rmse 6.1e-07  max resid 1.1e-06
## ... Similar to previous best
## *** Solution reached
```

```
col14=rainbow(14)
NMDS = data.frame(PCo1 = res$points[,1], PCo2 = res$points[,2])
ggplot(data = NMDS, aes(PCo1, PCo2)) +
  geom_point(size=4,color = col14)
```



Correspondence Analysis:A weighted PCA for Contingency Tables

- Variance in replaced by the Chisquare (Inertia)
- Centering is both by rows and columns (symmetry of dimensions)
- Standard to represent both the rows and the columns on the same (bi)plot
- Good method for finding hidden gradients

What is a contingency table?

	black	*blue*	*green*	*grey*	*orange*	*purple*	*white*
quiet	27700	21500	21400	8750	12200	8210	25100
angry	29700	15300	17400	7520	10400	7100	17300
clever	16500	12700	13200	4950	6930	4160	14200
depressed	14800	9570	9830	1470	3300	1020	12700
happy	193000	83100	87300	19200	42200	26100	91500
lively	18400	12500	13500	6590	6210	4880	14800
perplexed	1100	713	801	189	233	152	1090
virtuous	1790	802	1020	200	247	173	1650

Categorical Data Representations

(the long version representation)

Indicator variables:

Patient	*Mutation*1	*Mutation*2	$ …
*AHY*789	0	0	
*AHX*717	1	0	

are transformed into a Mutation × Mutation matrix.

Hair Color, Eye Color

```
require(ade4)
HairColor=HairEyeColor[,,2]
HairColor
```

```
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    36    9     5     2
##   Brown    66   34    29    14
##   Red      16    7     7     7
##   Blond     4   64     5     8
```

```
chisq.test(HairColor)
```

```
## Warning in chisq.test(HairColor): Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  HairColor
## X-squared = 107, df = 9, p-value <2e-16
```

Conclusion The data are not independent, the categories show a pattern of dependency, what can we say about them?

Independence: computationally

```
rowsums=as.matrix(apply(HairColor,1,sum))
rowsums
```

```
##       [,1]
## Black   52
## Brown  143
## Red     37
## Blond   81
```

```
colsums=as.matrix(apply(HairColor,2,sum))
colsums
```

```
##       [,1]
## Brown  122
## Blue   114
## Hazel   46
## Green   31
```

```
HCexp=round(rowsums%*%t(colsums)/sum(colsums))
Exp = outer(apply(HairColor, 1, sum), apply(HairColor, 2, sum))
#Here is actually how the chisquare is computed
sum((HairColor - Exp)^2/Exp)
```

```
## [1] 97344
```

```
HairColor-HCexp
```

```
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    16  -10    -3    -3
##   Brown    10  -18     8     0
##   Red       2   -6     2     3
##   Blond   -28   34    -7     0
```

```
require(vcd)
```

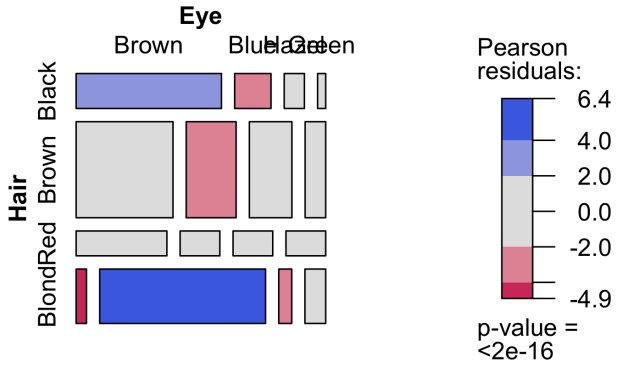
```
## Loading required package: vcd
```

```
##
## Attaching package: 'vcd'
```

```
## The following object is masked from 'package:GenomicRanges':
##
##     tile
```

```
## The following object is masked from 'package:IRanges':
##
##     tile
```

```
mosaic(HairColor,shade=TRUE)
```



What special property does the HCexp/Exp matrix have?

Independence: mathematically

If we are comparing two categorical variables, (hair color, eye color), (color, emotion), Counts in the table approximately the margin products: for a $I \times J$ contingency table with a total sample size of $n = \sum_{i=1}^{I} \sum_{j=1}^{J} n_{ij} = n_{..}$.

$$n_{ij} \doteq \frac{n_{i.}}{n} \frac{n_{.j}}{n} n$$

can also be written: $\mathbf{N} \doteq \mathbf{c}\mathbf{r}' n$, where

$$c = \frac{1}{n}\mathbf{N}\mathbf{1}_m \quad \text{and} \quad \mathbf{r}' = \frac{1}{n}\mathbf{N}'\mathbf{1}_p$$

The departure from independence is measured by the χ^2 statistic

$$\chi^2 = \sum_{i,j} [\frac{(n_{ij} - \frac{n_{i.}}{n}\frac{n_{.j}}{n}n)^2}{\frac{n_{i.}n_{.j}}{n^2}n}]$$

Correspondence Analysis is like PCA using χ^2 distances

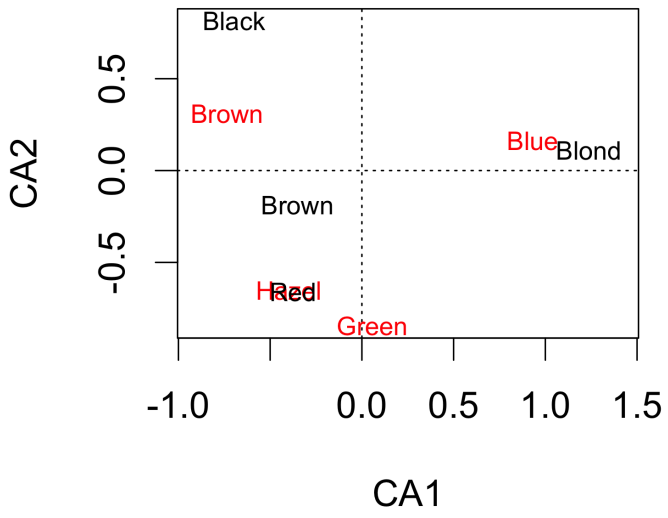
Many implemetations: - `dudi.coa` in `ade4` - `CCA` in `vegan` - ordination in `phyloseq`

```
library("ade4")
HairColor = HairEyeColor[,,2]
chisq.test(HairColor)
```

```
##
##  Pearson's Chi-squared test
##
## data:  HairColor
## X-squared = 107, df = 9, p-value <2e-16
```

`CCA` in `vegan`

```
library(vegan)
HairColor = HairEyeColor[,,2]
res.ca=vegan::cca(HairColor)
plot(res.ca,scaling=3)
```



Example: Plato's Sentence Endings

TABLE 1

Percentage Distribution of Sentence Endings

Type of Ending	Π_0 *Rep.*	Π_1 *Laws*	$\hat{s}_t$—*Natural Log of Ratio*	*Crit.*	*Phil.*	*Pol.*	*Soph.*	*Tim*
⏑ ⏑ ⏑ ⏑ ⏑ .	1·1	2·4	0·779	3·3	2·5	1·7	2·8	2·4
– ⏑ ⏑ ⏑ ⏑ .	1·6	3·8	0·867	2·0	2·8	2·5	3·6	3·9
⏑ – ⏑ ⏑ ⏑ .	1·7	1·9	0·113	2·0	2·1	3·1	3·4	6·0
⏑ ⏑ – ⏑ ⏑ .	1·9	2·6	0·315	1·3	2·6	2·6	2·6	1·8
⏑ ⏑ ⏑ – ⏑ .	2·1	3·0	0·358	6·7	4·0	3·3	2·4	3·4
⏑ ⏑ ⏑ ⏑ – .	2·0	3·8	0·642	4·0	4·8	2·9	2·5	3·5
– – ⏑ ⏑ ⏑ .	2·1	2·7	0·255	3·3	4·3	3·3	3·3	3·4
– ⏑ – ⏑ ⏑ .	2·2	1·8	−0·199	2·0	1·5	2·3	4·0	3·4
– ⏑ ⏑ – ⏑ .	2·8	0·6	−1·541	1·3	0·7	0·4	2·1	1·7
– ⏑ ⏑ ⏑ – .	4·6	8·8	0·647	6·0	6·5	4·0	2·3	3·3
⏑ – – ⏑ ⏑ .	3·3	3·4	0·030	2·7	6·7	5·3	3·3	3·4
⏑ – ⏑ – ⏑ .	2·6	1·0	−0·956	2·7	0·6	0·9	1·6	2·2
⏑ – ⏑ ⏑ – .	4·6	1·1	−1·430	2·0	0·7	1·0	3·0	2·7
⏑ ⏑ – – ⏑ .	2·6	1·5	−0·548	2·7	3·1	3·1	3·0	3·0
⏑ ⏑ – ⏑ – .	4·4	3·0	−0·385	3·3	1·9	3·0	3·0	2·2
⏑ ⏑ ⏑ – – .	2·5	5·7	0·824	6·7	5·4	4·4	5·1	3·9
– – – ⏑ ⏑ .	2·9	4·2	0·372	2·7	5·5	6·9	5·2	3·0
– – ⏑ – ⏑ .	3·0	1·4	−0·761	2·0	0·7	2·7	2·6	3·3
– – ⏑ ⏑ – .	3·4	1·0	−1·224	0·7	0·4	0·7	2·3	3·3
– ⏑ – – ⏑ .	2·0	2·3	0·140	2·0	1·2	3·4	3·7	3·3
– ⏑ – ⏑ – .	6·4	2·4	−0·982	1·3	2·8	1·8	2·1	3·0
– ⏑ ⏑ – – .	4·2	0·6	−1·946	4·7	0·7	0·8	3·0	2·8
⏑ ⏑ – – – .	2·8	2·9	0·039	1·3	2·6	4·6	3·4	3·0
⏑ – ⏑ – – .	4·2	1·2	−1·253	2·7	1·3	1·0	1·3	3·3
⏑ – – ⏑ – .	4·8	8·2	0·536	5·3	5·3	4·5	4·6	3·0
⏑ – – – ⏑ .	2·4	1·9	−0·231	3·3	3·3	2·5	2·5	2·2
⏑ – – – – .	3·5	4·1	0·157	2·0	3·3	3·8	2·9	2·4
– ⏑ – – – .	4·0	3·7	−0·077	4·7	3·3	4·9	3·5	3·0
– – ⏑ – – .	4·1	2·1	−0·668	6·0	2·3	2·1	4·1	6·4
– – – ⏑ – .	4·1	8·8	0·765	2·0	9·0	6·8	4·7	3·8
– – – – ⏑ .	2·0	3·0	0·405	3·3	2·9	2·9	2·6	2·2
– – – – – .	4·2	5·2	0·215	4·0	4·9	7·3	3·4	1·8
Number of sentences .	3,778	3,783	—	150	958	770	919	762

There are minor errors in the third decimal place of the scores $\hat{s}_t$.

Table from Brandwood and Cox

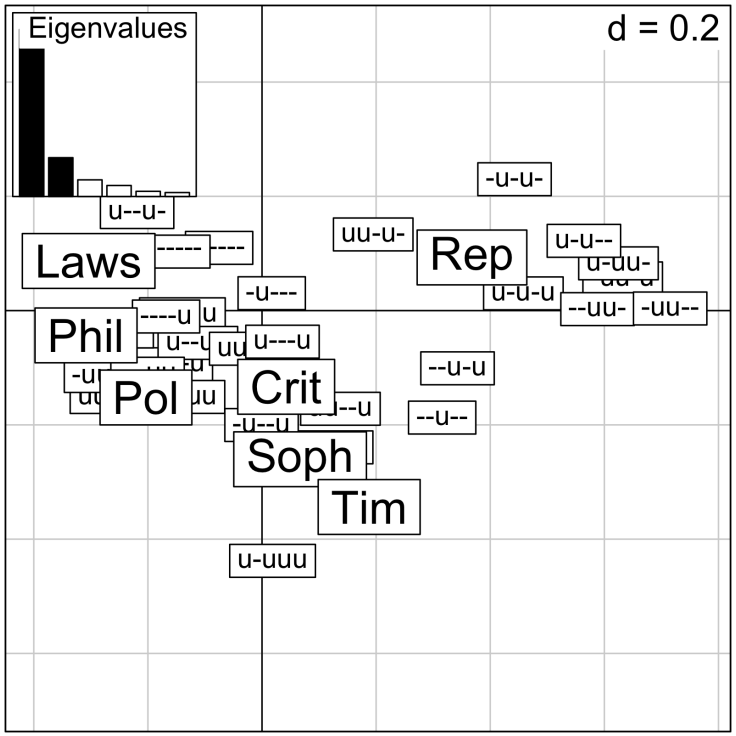
The dates Plato wrote these various `books' is not known. Sentence endings and use those pattern frequencies as the data.

The first (earliest) is known to be Republica. The last (latest) is known to be Laws.

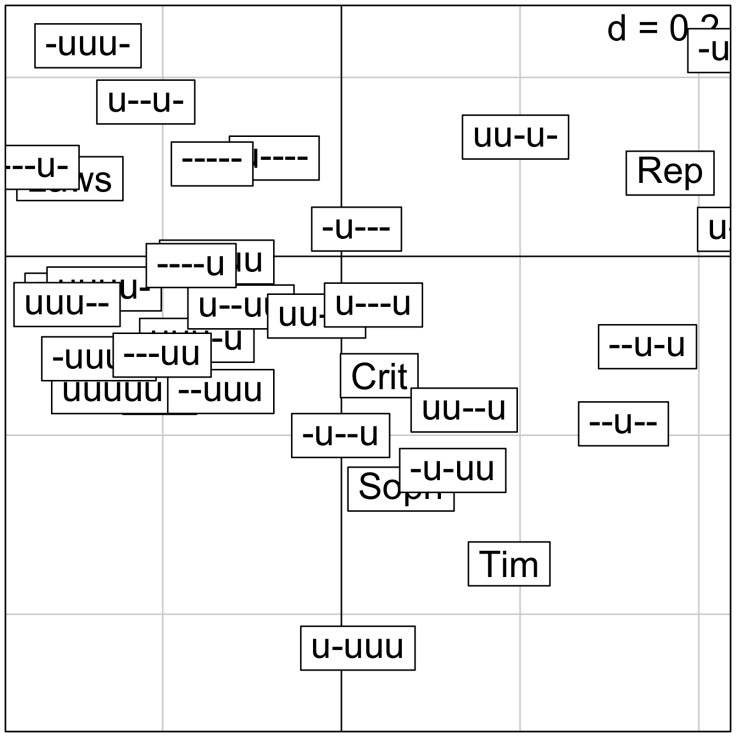
```
platof=read.table("~/Books/CUBook/data/platof.txt",header=T)
platof[1:8,]
```

```
##       Rep Laws Crit Phil Pol Soph Tim
## uuuuu  42   91    5   24  13   26  18
## -uuuu  60  144    3   27  19   33  30
## u-uuu  64   72    3   20  24   31  46
## uu-uu  72   98    2   25  20   24  14
## uuu-u  79  113   10   38  25   22  26
## uuuu-  76  144    6   46  22   23  27
## --uuu  79  102    5   41  25   30  26
## -u-uu  83   68    3   14  18   37  26
```

```
res.plato=dudi.coa(platof,scannf=FALSE,nf=2)
scatter(res.plato)
```




```
s.label(res.plato$co)
s.label(res.plato$li,add.plot=T)
```




```
names(res.plato)
```

```
##  [1] "tab"  "cw"   "lw"   "eig"  "rank" "nf"   "c1"   "li"   "co"   "l1"
## [11] "call" "N"
```

```
sum(res.plato$eig)
```

```
## [1] 0.13
```

```
round(res.plato$eig,2)
```

```
## [1] 0.09 0.02 0.01 0.01 0.00 0.00
```

More About Gradients

The Boomer Lake example: Arch Effect for Ecologists (http://ordination.okstate.edu/PCA.htm)

Two species count matrices

```
lakelike
```

```
##       plant1 plant2 plant3 plant4 plant5 plant6 plant7 plant8 plant9 plant10
## loc1       5      5      2      1      1      1      2      0      0        0
## loc2       5      6      7      4      2      1      0      1      0        0
## loc3       4      5      5      5      3      2      2      0      1        1
## loc4       2      4      5      6      5      4      3      1      0        0
## loc5       2      2      3      5      5      4      4      5      2        0
## loc6       0      1      2      5      4      5      4      3      2        1
## loc7       0      1      1      4      4      5      6      4      3        3
## loc8       0      0      2      2      3      3      4      6      4        3
## loc9       1      0      1      0      1      3      3      5      6        4
## loc10      0      0      0      1      0      1      3      4      4        5
##       plant11 plant12 plant13 plant14 plant15
## loc1        0       1       0       2       2
## loc2        1       0       2       0       0
## loc3        1       0       1       1       1
## loc4        1       1       1       0       0
## loc5        0       0       1       0       0
## loc6        1       0       1       1       0
## loc7        1       0       0       0       0
## loc8        3       2       3       1       1
## loc9        5       3       1       0       1
## loc10       4       3       2       2       1
```

```
lakelikeh
```

```
##       plant1 plant2 plant3 plant4 plant5 plant6 plant7 plant8 plant9 plant10
## loc1       5      5      2      1      1      1      2      0      0        0
## loc2       5      6      7      4      2      1      0      1      0        0
## loc3       4      5      5      5      3      2      2      0      1        1
## loc4      20     40     50     60     50     40     30     10      0        0
## loc5       2      2      3      5      5      4      4      5      2        0
## loc6       0      1      2      5      4      5      4      3      2        1
## loc7       0      1      1      4      4      5      6      4      3        3
## loc8       0      0      2      2      3      3      4      6      4        3
## loc9       1      0      1      0      1      3      3      5      6        4
## loc10      0      0      0      1      0      1      3      4      4        5
##       plant11 plant12 plant13 plant14 plant15
## loc1        0       1       0       2       2
## loc2        1       0       2       0       0
## loc3        1       0       1       1       1
## loc4       10      10      10       0       0
## loc5        0       0       1       0       0
## loc6        1       0       1       1       0
## loc7        1       0       0       0       0
## loc8        3       2       3       1       1
## loc9        5       3       1       0       1
## loc10       4       3       2       2       1
```

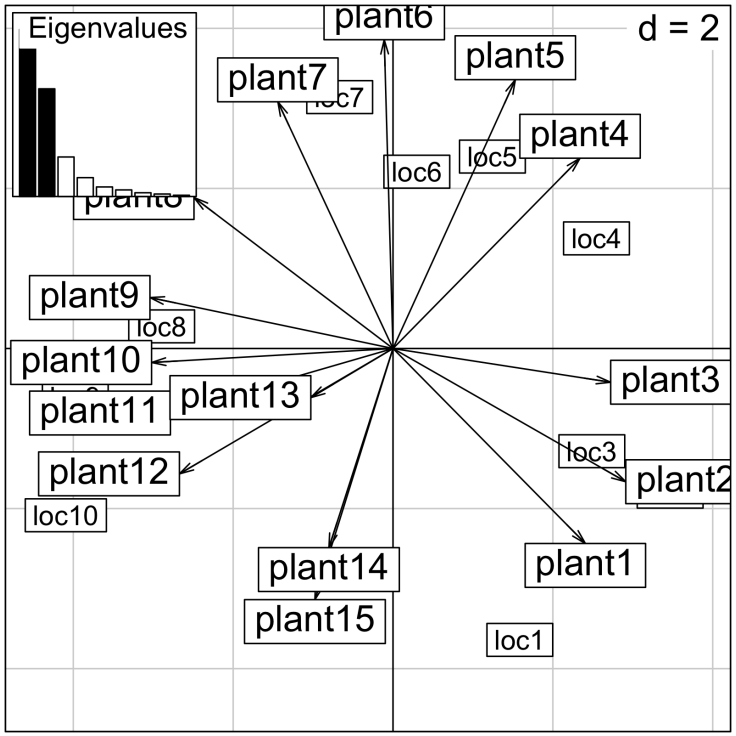
```
reslake=dudi.coa(lakelike,scannf=FALSE,nf=2)
reslake2=dudi.pca(lakelike,scannf=FALSE,nf=2)
reslakeh=dudi.coa(lakelikeh,scannf=FALSE,nf=2)
reslakeh2=dudi.pca(lakelikeh,scannf=FALSE,nf=2)
```

Compare output from PCA and Correspondence Analysis

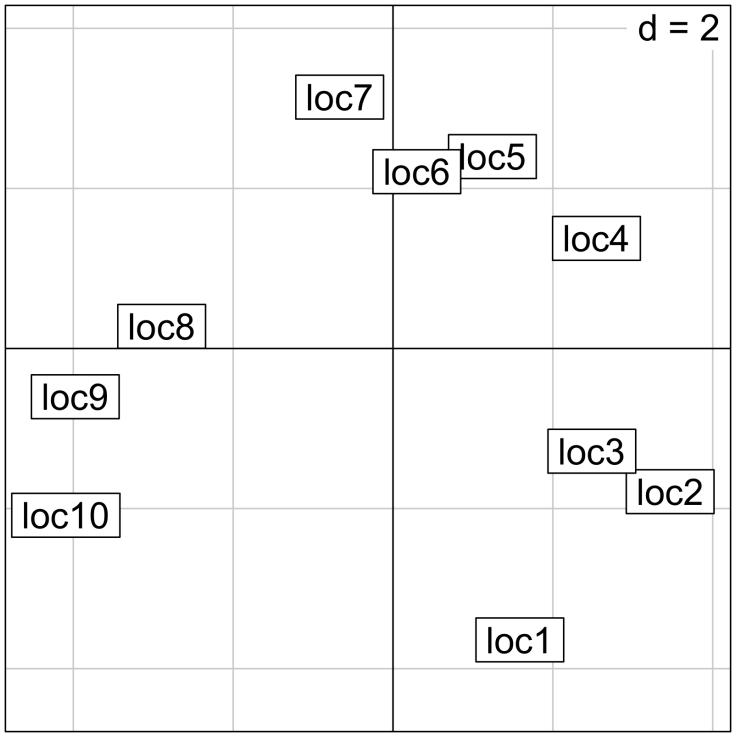
First Data Set

Principal Components

```
scatter(reslake2)
```

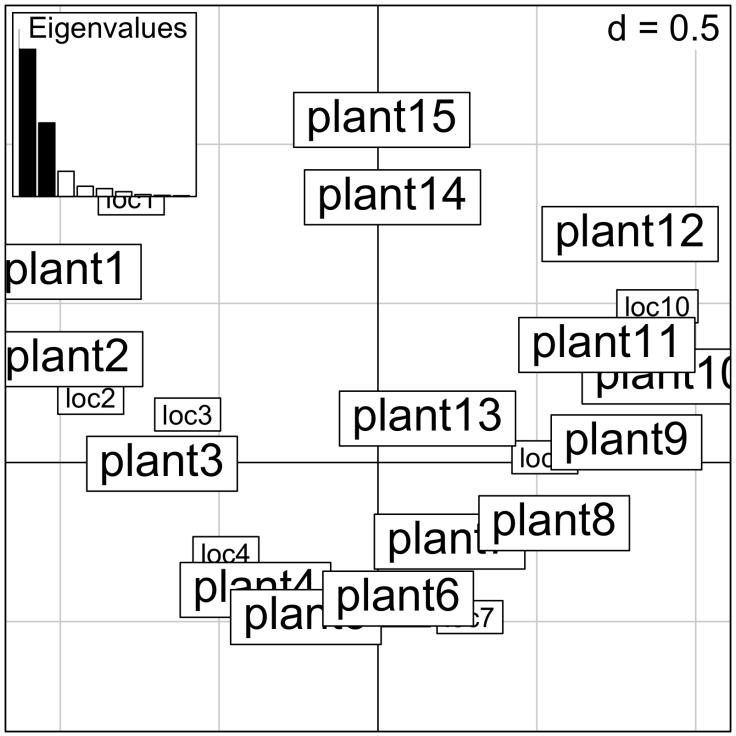



```
s.label(reslake2$li)
```

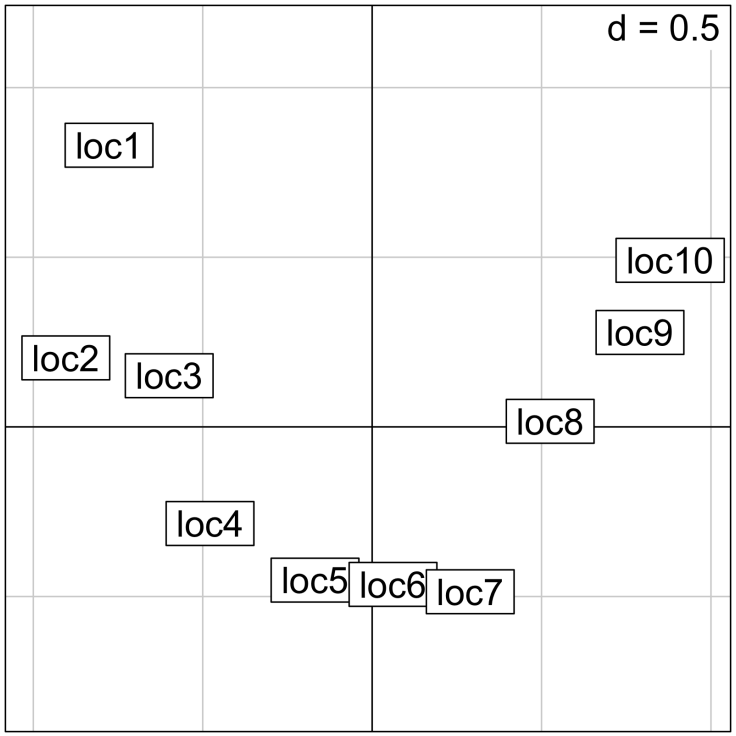


Correspondence Analysis

```
scatter(reslake)
```



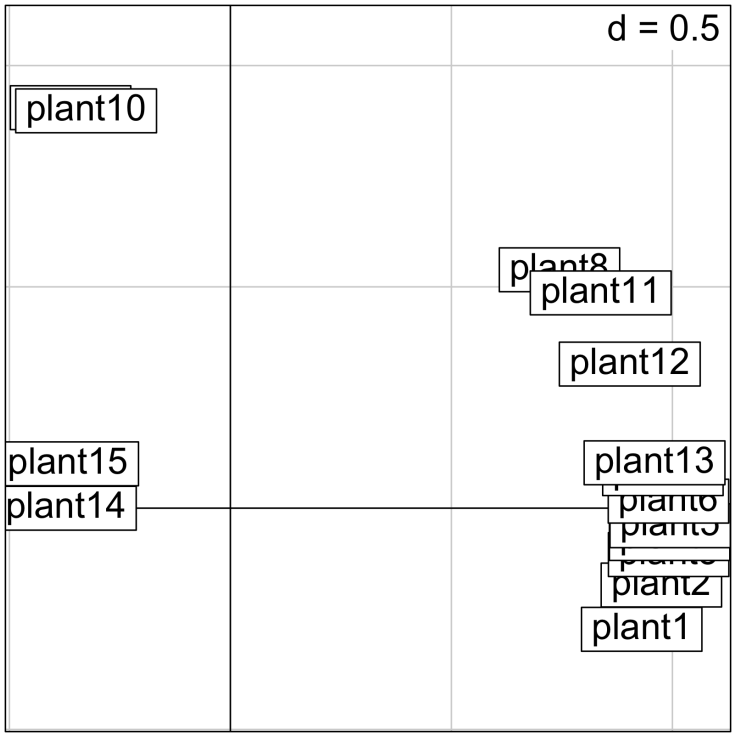

```
s.label(reslake$li)
```



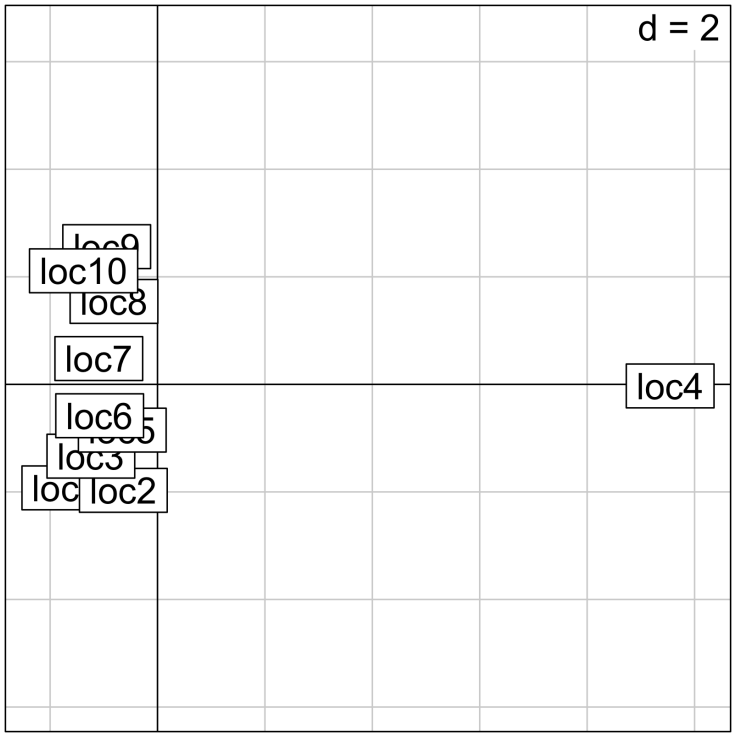
Second Data set

Principal Components

```
s.label(reslakeh2$co)
```

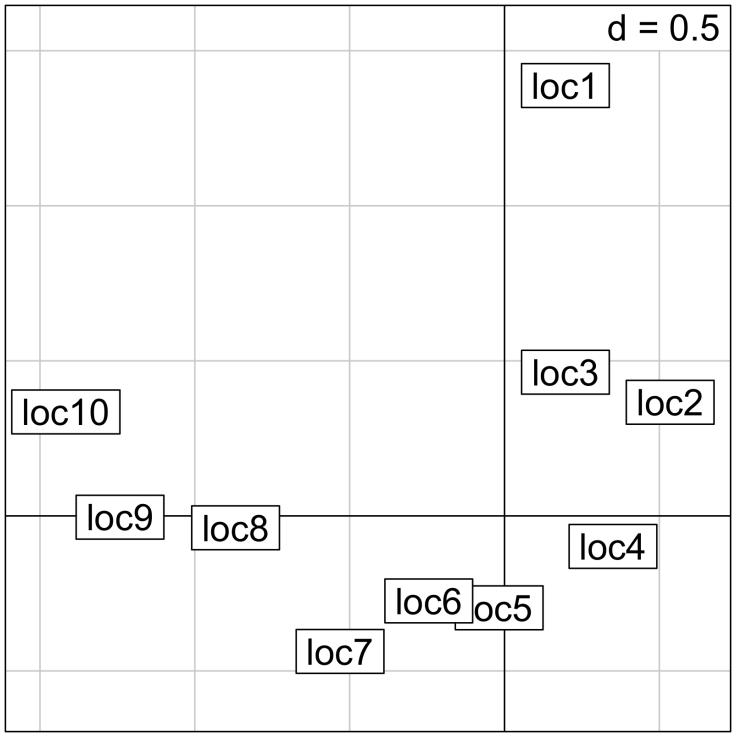



```
s.label(reslakeh2$li)
```

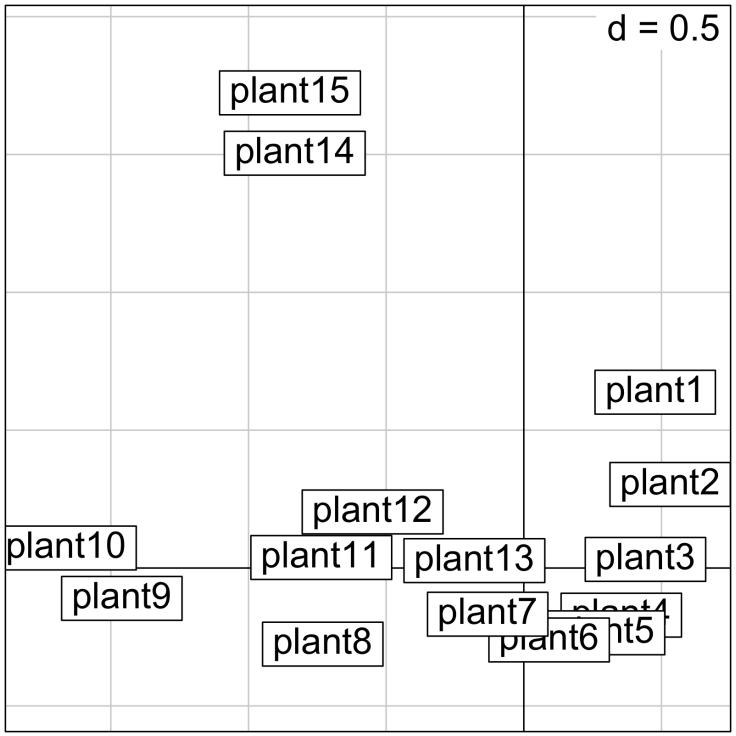


Correspondence Analysis

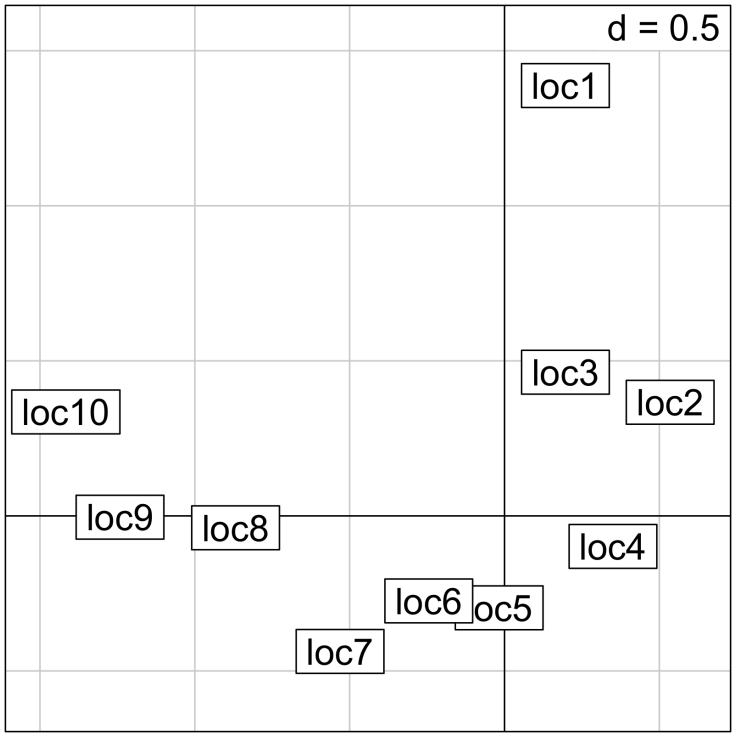
```
s.label(reslakeh$li)
```



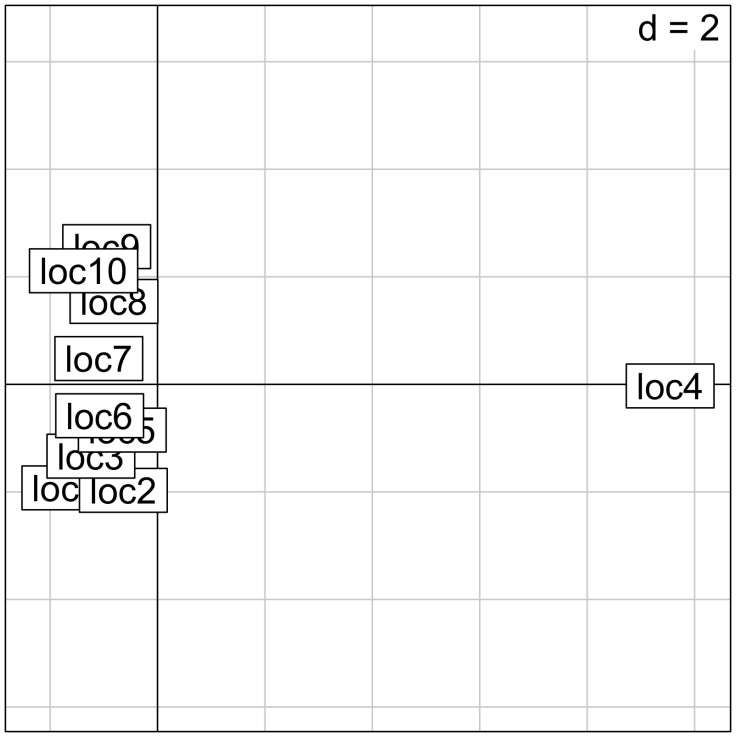

```
s.label(reslakeh$co)
```




```
s.label(reslakeh$li)
```



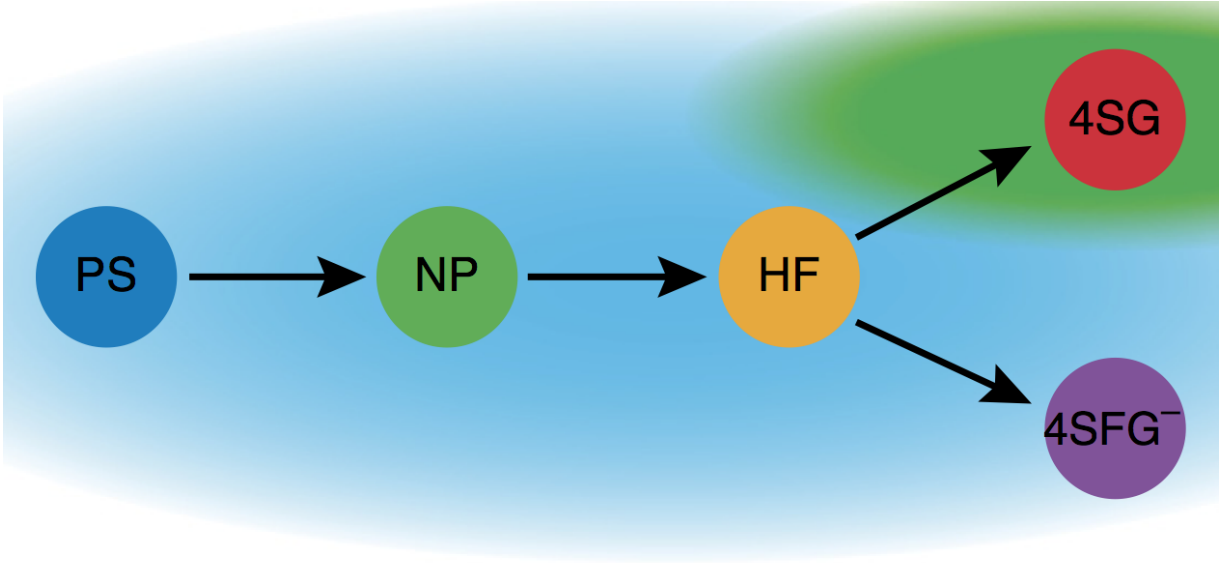

```
s.label(reslakeh2$li)
```



Other methods for gradients

We will follow an analysis of Moignard et al., 2015.

This paper describes the dynamics of blood cell development. The data are single cell gene expression measurements of 3,934 cells with blood and endothelial potential from five populations from between embryonic day (E)7.0 and E8.25.



Cell dynamics

The four cell populations studied here are representative of three sequential states (PS,NP,HF) and two possible final branches (4SG and $4SFG^{-}$).

```
## [1] 3934   46
```

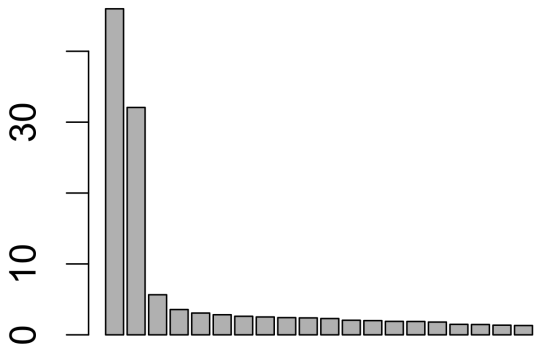
```
## typesort sortA sortB
##
##           3175   759
```

The classical multidimensional scaling on two distances matrices can be carried out using:

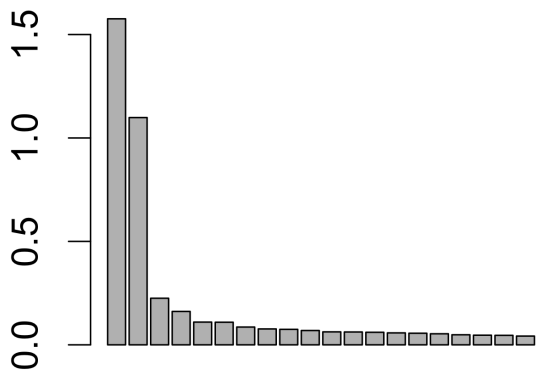
```
dist2n.euclid=dist(Norm)
dist1n.l1=dist(Norm,"manhattan")
cellsn.cmds=cmdscale(dist1n.l1,k=20,eig=TRUE)
cellsn2.cmds=cmdscale(dist2n.euclid,k=20,eig=TRUE)
perc1=round(100*sum(cellsn.cmds$eig[1:2])/sum(cellsn.cmds$eig))
perc2=round(100*sum(cellsn2.cmds$eig[1:2])/sum(cellsn2.cmds$eig))
```

We look at the underlying dimension and see below that two dimensions can provide a substantial percentage of the variance.

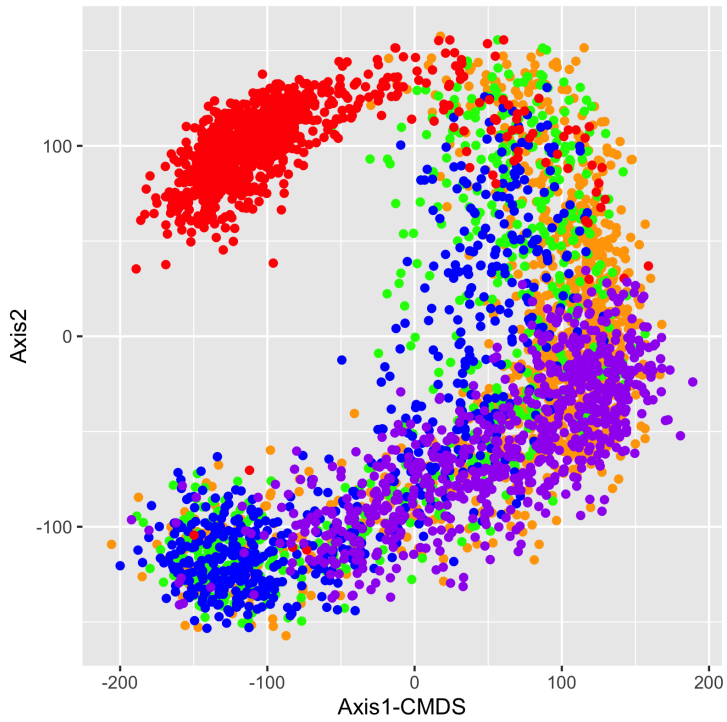
```
barplot(100*cellsn.cmds$eig[1:20]/sum(cellsn.cmds$eig))
```



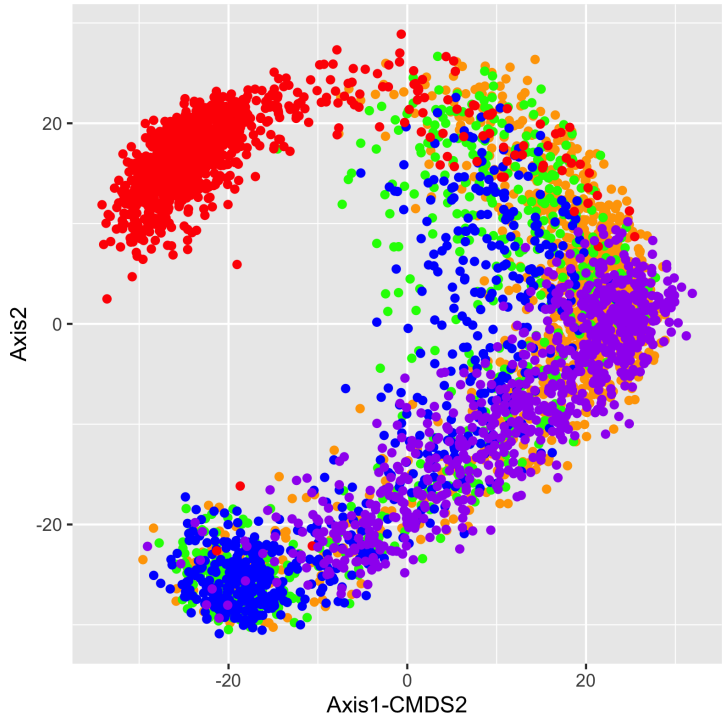

```
barplot(100*cellsn2.cmds$eig[1:20]/sum(cellsn.cmds$eig))
```




```
cellsmds=data.frame(Axis1=cellsn.cmds$points[,1],Axis2=cellsn.cmds$points[,2])
cells2mds=data.frame(Axis1=cellsn2.cmds$points[,1],Axis2=cellsn2.cmds$points[,2])
ggplot(cellsmds,aes(x=Axis1,y=Axis2))+
  geom_point(aes(color=celltypes),color=cellcol) +xlab("Axis1-CMDS") +ylab("Axis2")
```




```
ggplot(cells2mds,aes(x=Axis1,y=Axis2))+
  geom_point(aes(color=celltypes),color=cellcol) +xlab("Axis1-CMDS2") +ylab("Axis2")
```



Nonlinear methods

The cells are not distributed uniformly in the lower dimensions we have been looking at, they form a horseshoe.

Finding Time

Horseshoes: A la recherche (http://www.huber.embl.de/users/whuber/pub/horseshoe.html)

Multidimensional scaling and non metric multidimensional scaling aims to represent all distances as precisely as possible and the large distances between far away points skew the representations.

It can be beneficial when looking for gradients or low dimensional manifolds to restrict ourselves to approximations of points that are close together.

These methods try to represent local (small) distances well and do not try to approximate distances between faraway points with too much accuracy.

The use of Kernels computed using the calculated interpoint distances allows us to decrease the importance of points that are far apart. A radial basis kernel can be of the form

$$1 - \exp\{-\frac{d(x,y)^2}{\sigma^2}\}, \text{ where } \sigma^2 \text{ is fixed.}$$

or the l1 version:

$$1 - \exp\{-\frac{d(x,y)}{\sigma}\}, \text{ where } \sigma \text{ is fixed.}$$

t-SNE

Change the distance: allow the σ^2 parameter to vary locally.

Thus we obtain a probability distribution that serves as the probability that pairs of points in the high dimensional space are neighbors.

The t-SNE method then constructs Y_i points in low dimensions so their distances are proportional to $(1 + ||y_i - y_j||^2)^{-1}$ minimizing the (non-symmetric) Kullback–Leibler divergence of the distribution Q from P.

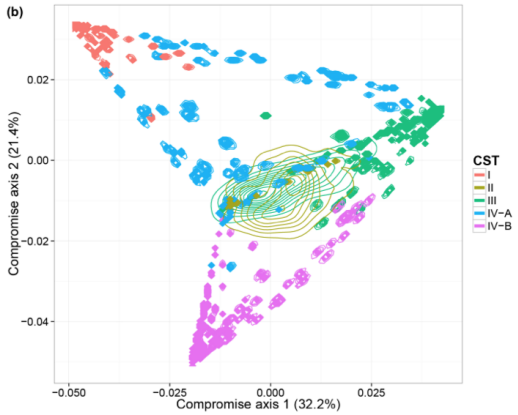
This method is not robust and has the property of separating clusters of points artificially; this can clarify a complex situation. One can think it as a method akin to the network layout algorithms.

They stretch the data to clarify relations, but the distances between point cannot be interpreted as on the same scales in different points in the plane.

See examples in chapter 9 (http://web.stanford.edu/class/bios221/book/Chap-MultivaHetero.html)

Ten quick tips (TenQuickTips.html)

Confidence Regions



Paper: Boyu Ren, Sergio Bacallado, Stefano Favro, Susan Holmes, Lorenzo Trippa JASA: open access preprint (https://arxiv.org/abs/1601.05156)

Whole talk with model for uncertainty (https://www.dropbox.com/s/9axbd6fljwckxe4/MSR_Microbiome_Oct.pdf?dl=0)

Uses a **Bayesian** statistical model.