R Notebook sandbox: Playing with PCA

Contents

Top of the world

Some of you may have difficulties accessing Wikipedia (e.g., out of the country, or whatever). The notebook should benefit you in your efforts (especially for project 2); however, I have uploaded the concluding data frames:

http://md5.mshaffer.com/WSU_STATS419/_data_/state-capitals/

or

http://md5.mshaffer.com/WSU_STATS419/_data_/state-capitals/final/

You can access them there, if need be.

Principle Components Analysis

Mathematically, a n-dimensional space can be transformed using a variety of techniques that are mathematically equivalent. This is the concept of a vector having a basis.

So if I have 2-dimensional data, I can transform it into a new basis, still having 2-dimensions.

And if I have 3-dimensional data, I can transform it into a new basis, still having 3-dimensions.

...

And if I have 7-dimensional data, I can transform it into a new basis, still having 7-dimensions.

...

And if I have n-dimensional data, I can transform it into a new basis, still having n-dimensions.

2-Dimensions

Elliptical

If the variances would be equal, these ellipses would be circles.

```
# cache.rebuild ... This will prevent Xs from being used from previous cache ...

library(mvnfast);
source_url( paste0(path.github, "humanVerseWSU/R/functions-maths.R") );   # deg2rad
```

```
## SHA-1 hash of file is 8fa5b087bebc710c368e8e8131ac3d83d4d2cf4c
```

```
                                          # zeroIsh

set.seed(1222015);

mu = c(1,3); # centers for x,y
Sigma = diag(c(2,23)); # variance for x,y

nsim = 9000;
X = rmvn(nsim, mu, Sigma, ncores=2);  # this is parallelizability with cores
# ncores    ... Number of cores used. The parallelization will take place only if OpenMP is supported.

xy.lim = c(min(X), max(X)); # square

print("################   X   ################");
```

```
## [1] "################   X   ################"
```

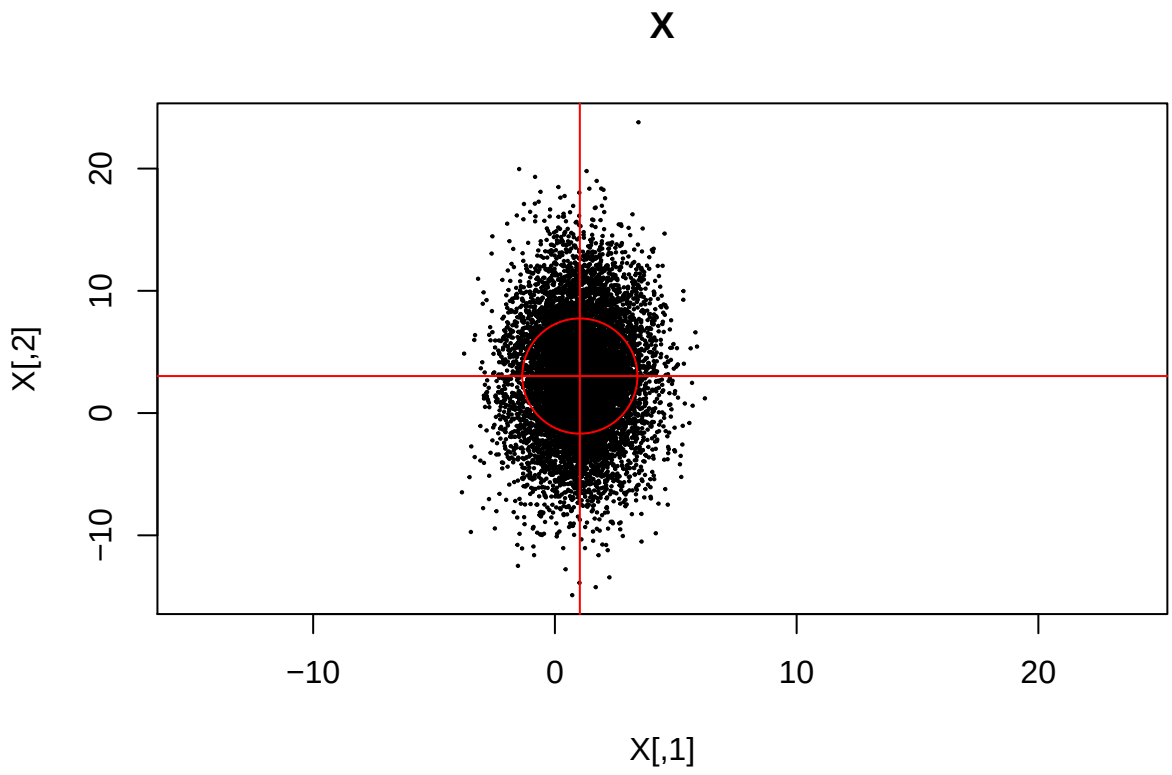
```
print(paste0("MEANS:    x = ",round(mean(X[,1]),3),
             "       y = ",round(mean(X[,2]),3) ));
```

```
## [1] "MEANS:    x = 1.032       y = 3.029"
```

```
print(paste0("VARIANCE: x = ",round(var(X[,1]),3),
             "       y = ",round(var(X[,2]),3) ));
```

```
## [1] "VARIANCE: x = 1.965       y = 22.735"
```

```
plot(X, pch=20, cex=0.25, main="X",
        xlim=xy.lim, ylim=xy.lim );
  abline(v=mean(X[,1]), col="red");
  abline(h=mean(X[,2]), col="red");
points(x=mean(X[,1]),y=mean(X[,2]),
          pch=21, col="red", cex=8);
```




```
Xs = scale(X);

print("#################   Xs   ################");
```

```
## [1] "#################   Xs   ################"
```

```
print(paste0("MEANS:    x = ",round(mean(Xs[,1]),3),
               "       y = ",round(mean(Xs[,2]),3) ));
```

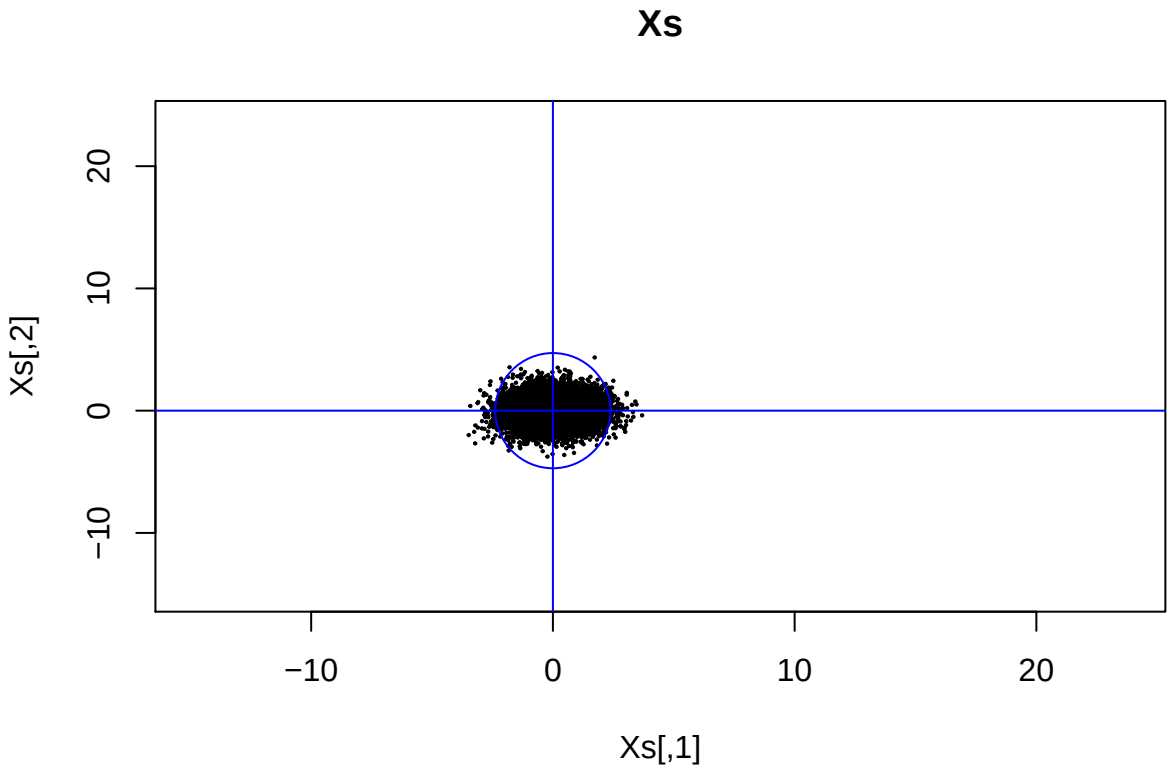
```
## [1] "MEANS:    x = 0       y = 0"
```

```
print(paste0("VARIANCE: x = ",round(var(Xs[,1]),3),
               "       y = ",round(var(Xs[,2]),3) ));
```

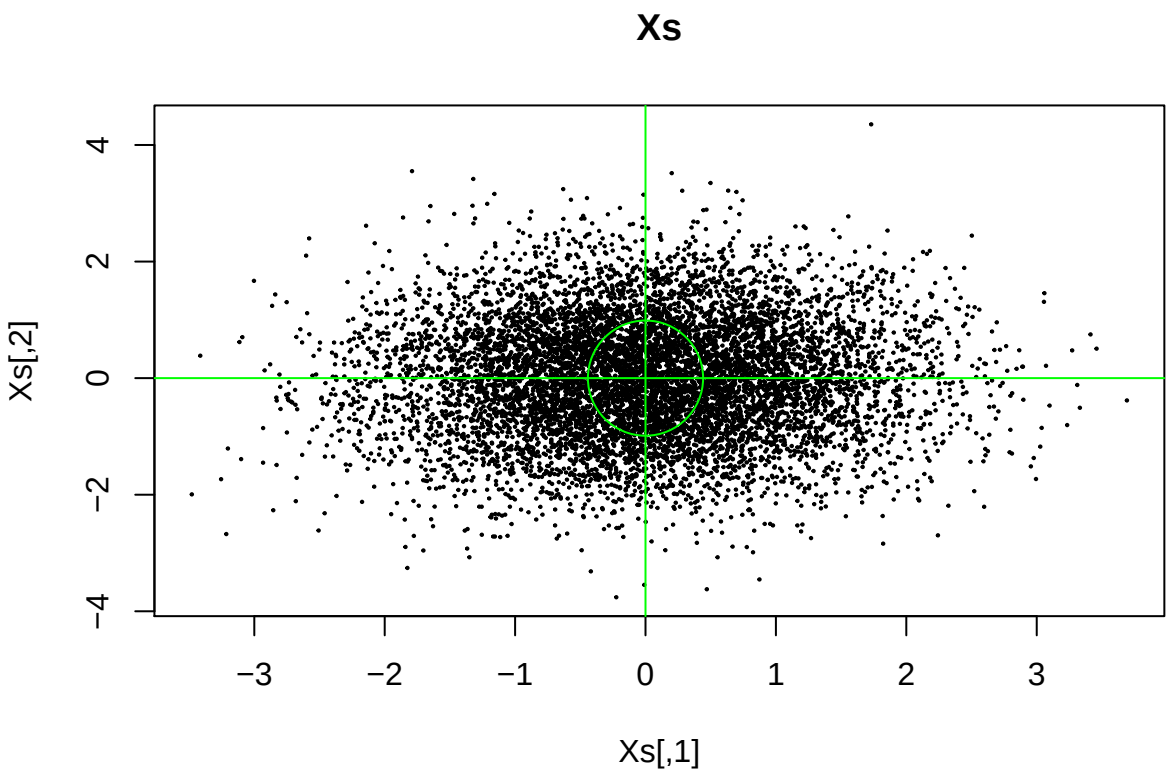
```
## [1] "VARIANCE: x = 1       y = 1"
```

```
plot(Xs, pch=20, cex=0.25, main="Xs",
        xlim=xy.lim, ylim=xy.lim );
  abline(v=mean(Xs[,1]), col="blue");
  abline(h=mean(Xs[,2]), col="blue");
points(x=mean(Xs[,1]),y=mean(Xs[,2]),
```

```
          pch=21, col="blue", cex=8);
```




```
plot(Xs, pch=20, cex=0.25, main="Xs");
  abline(v=mean(Xs[,1]), col="green");
  abline(h=mean(Xs[,2]), col="green");
points(x=mean(Xs[,1]),y=mean(Xs[,2]),
          pch=21, col="green", cex=8);
```



It is called "scaling" for a reason. A translation in the (x,y) to "mean-center" at zero. And a scaling factor of (x,y) ... e.g., our z-scores.

I keep the `xlim` and `ylim` the same to see this ("red" vs "blue"). Then I let "green" auto-scale, although it is equal to "blue".

Rotated

We can take that same data and rotate it ... eventually we will call this `phi` ... currently, hardcoded as $\phi = 60$ in degrees, so I have the helper functions which I have placed in `functions-maths.R`.

```
x = X[,1];
y = X[,2];

XR = X; # let's manually rotate 60 ...

XR[,1] = x * cos(deg2rad(60)) - y * sin(deg2rad(60));
XR[,2] = x * sin(deg2rad(60)) + y * cos(deg2rad(60));

xyr.lim = c(min(XR),max(XR));

print("#################   XR   #################");
```

```
## [1] "#################   XR   #################"
```

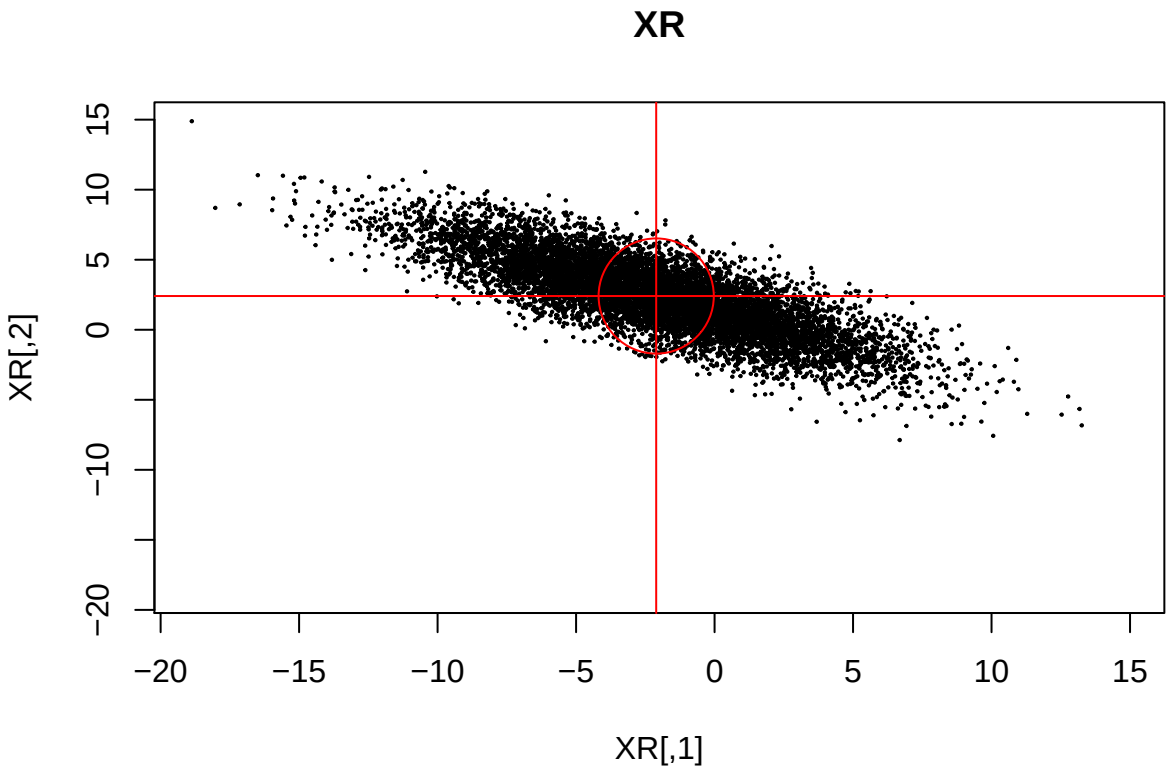
```
print(paste0("MEANS:    x = ",round(mean(XR[,1]),3),
             "       y = ",round(mean(XR[,2]),3) ));
```

```
## [1] "MEANS:    x = -2.107       y = 2.408"
```

```
print(paste0("VARIANCE: x = ",round(var(XR[,1]),3),
             "       y = ",round(var(XR[,2]),3) ));
```

```
## [1] "VARIANCE: x = 17.39       y = 7.31"
```

```
plot(XR, pch=20, cex=0.25, main="XR",
       xlim=xyr.lim, ylim=xyr.lim );
  abline(v=mean(XR[,1]), col="red");
  abline(h=mean(XR[,2]), col="red");
points(x=mean(XR[,1]),y=mean(XR[,2]),
         pch=21, col="red", cex=8);
```




```
XRs = scale(XR);

print("################   XRs   ################");
```

```
## [1] "################   XRs   ################"
```

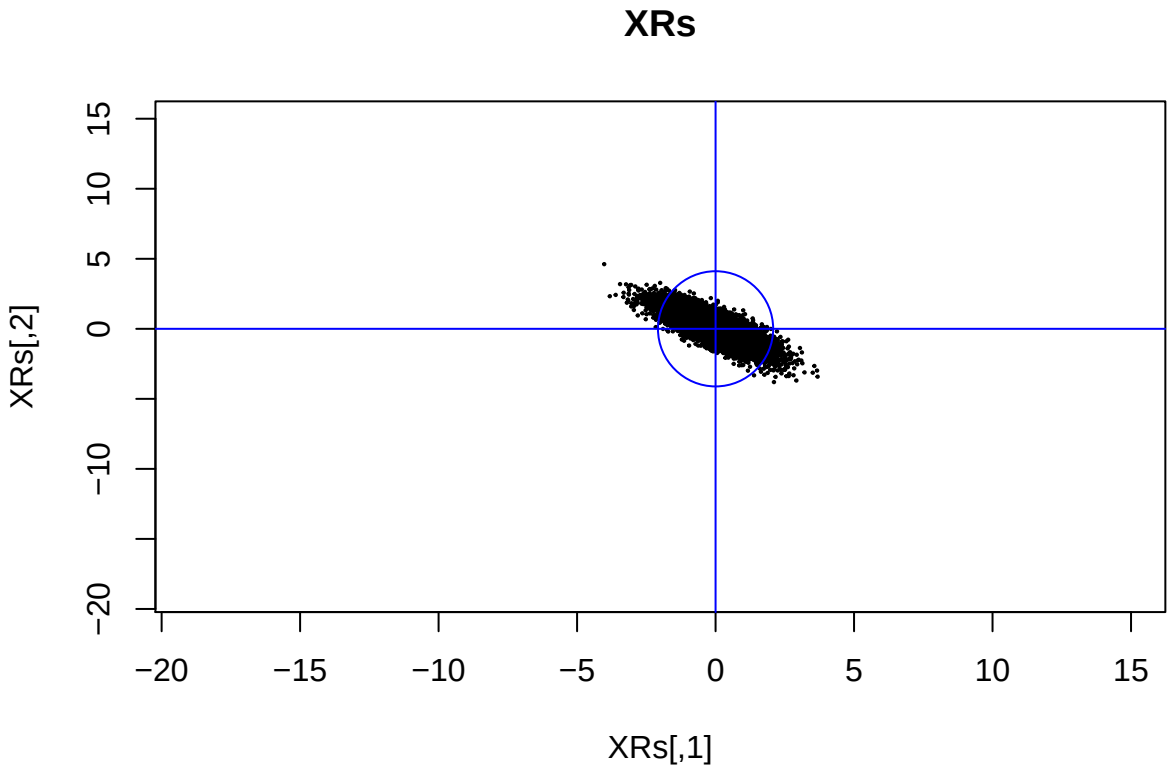
```
print(paste0("MEANS:    x = ",round(mean(XRs[,1]),3),
             "       y = ",round(mean(XRs[,2]),3) ));
```

```
## [1] "MEANS:    x = 0       y = 0"
```

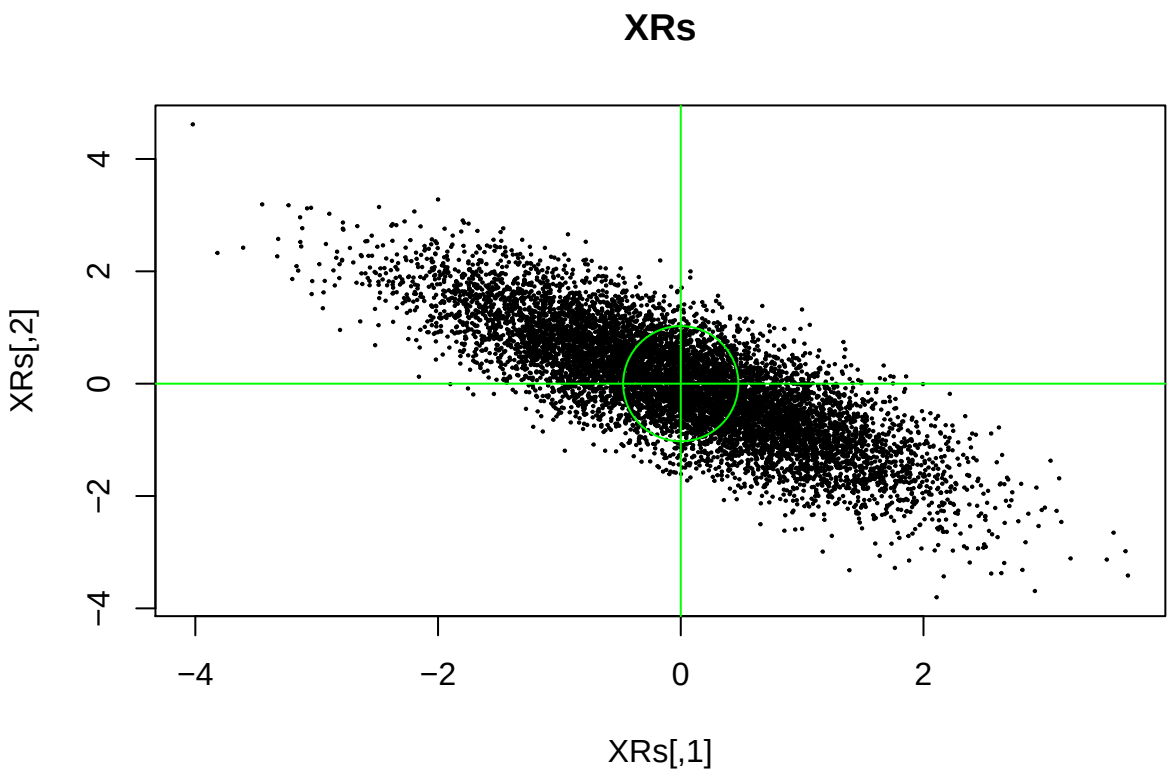
```
print(paste0("VARIANCE: x = ",round(var(XRs[,1]),3),
             "       y = ",round(var(XRs[,2]),3) ));
```

```
## [1] "VARIANCE: x = 1       y = 1"
```

```
plot(XRs, pch=20, cex=0.25, main="XRs",
        xlim=xyr.lim, ylim=xyr.lim );
  abline(v=mean(XRs[,1]), col="blue");
  abline(h=mean(XRs[,2]), col="blue");
points(x=mean(XRs[,1]),y=mean(XRs[,2]),
          pch=21, col="blue", cex=8);
```




```
plot(XRs, pch=20, cex=0.25, main="XRs");
  abline(v=mean(XRs[,1]), col="green");
  abline(h=mean(XRs[,2]), col="green");
points(x=mean(XRs[,1]),y=mean(XRs[,2]),
          pch=21, col="green", cex=8);
```



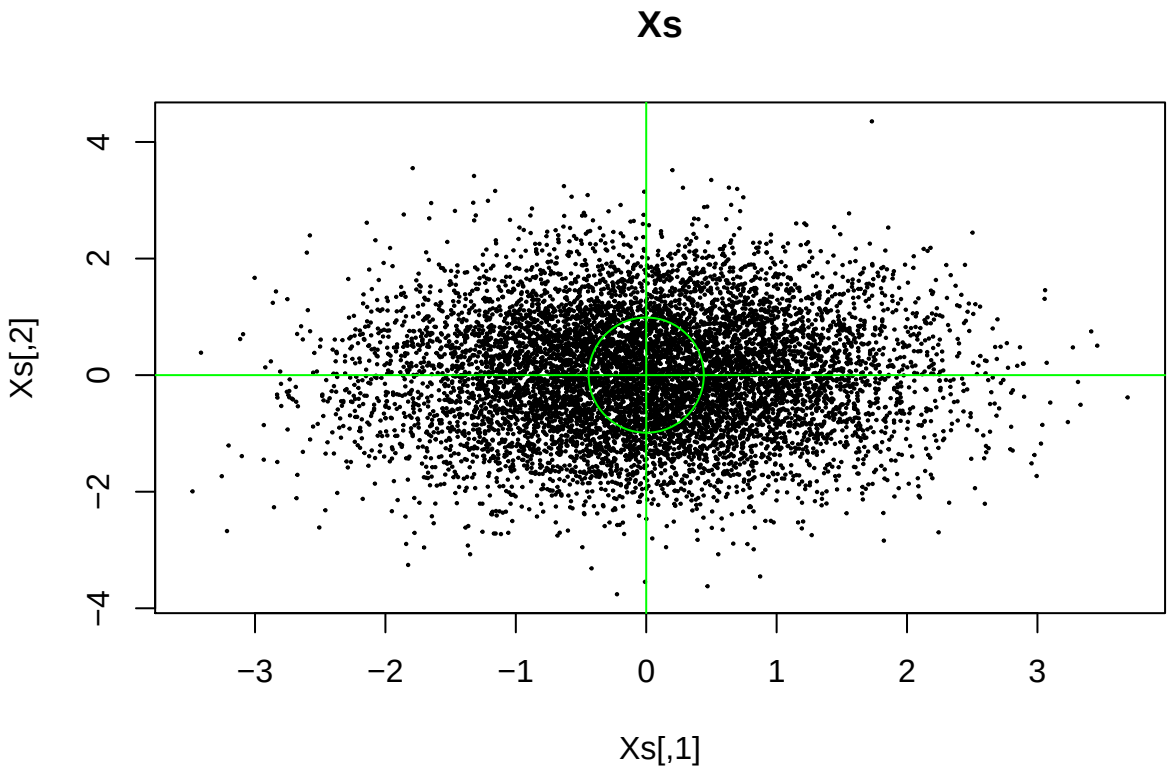

```
# non-rotated form of doing it :: https://statisticsglobe.com/plot-in-r-example
```

Translations, scalings, and rotations are not changing the overall basis.

Principle Components Analysis of Xs

If we don't scale, the one dimension will outweigh another dimension. This will create uninterpretable results.

```
plot(Xs, pch=20, cex=0.25, main="Xs");
  abline(v=mean(Xs[,1]), col="green");
  abline(h=mean(Xs[,2]), col="green");
points(x=mean(Xs[,1]),y=mean(Xs[,2]),
          pch=21, col="green", cex=8);
```



Xs

```
Xs.PCA = princomp(Xs);

summary(Xs.PCA);
```

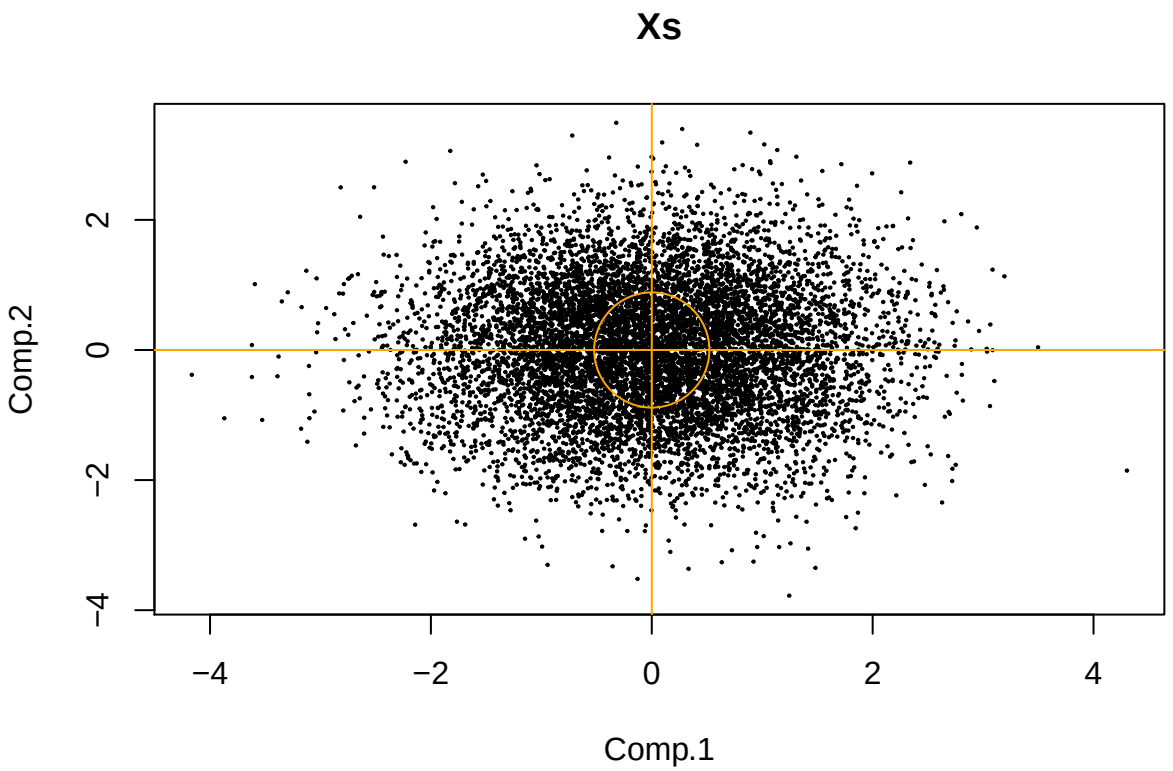
```
## Importance of components:
##                           Comp.1    Comp.2
## Standard deviation     1.0130539 0.9866608
## Proportion of Variance 0.5131961 0.4868039
## Cumulative Proportion  0.5131961 1.0000000
```

```
str(Xs.PCA);
```

```
## List of 7
##  $ sdev    : Named num [1:2] 1.013 0.987
##   ..- attr(*, "names")= chr [1:2] "Comp.1" "Comp.2"
##  $ loadings: 'loadings' num [1:2, 1:2] 0.707 0.707 0.707 -0.707
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "Comp.1" "Comp.2"
##  $ center  : num [1:2] 0.000000000000000552 -0.000000000000000431
##  $ scale   : num [1:2] 1 1
##  $ n.obs   : int 9000
##  $ scores  : num [1:9000, 1:2] 0.844 -1.016 -0.19 0.959 -0.657 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "Comp.1" "Comp.2"
##  $ call    : language princomp(x = Xs)
```

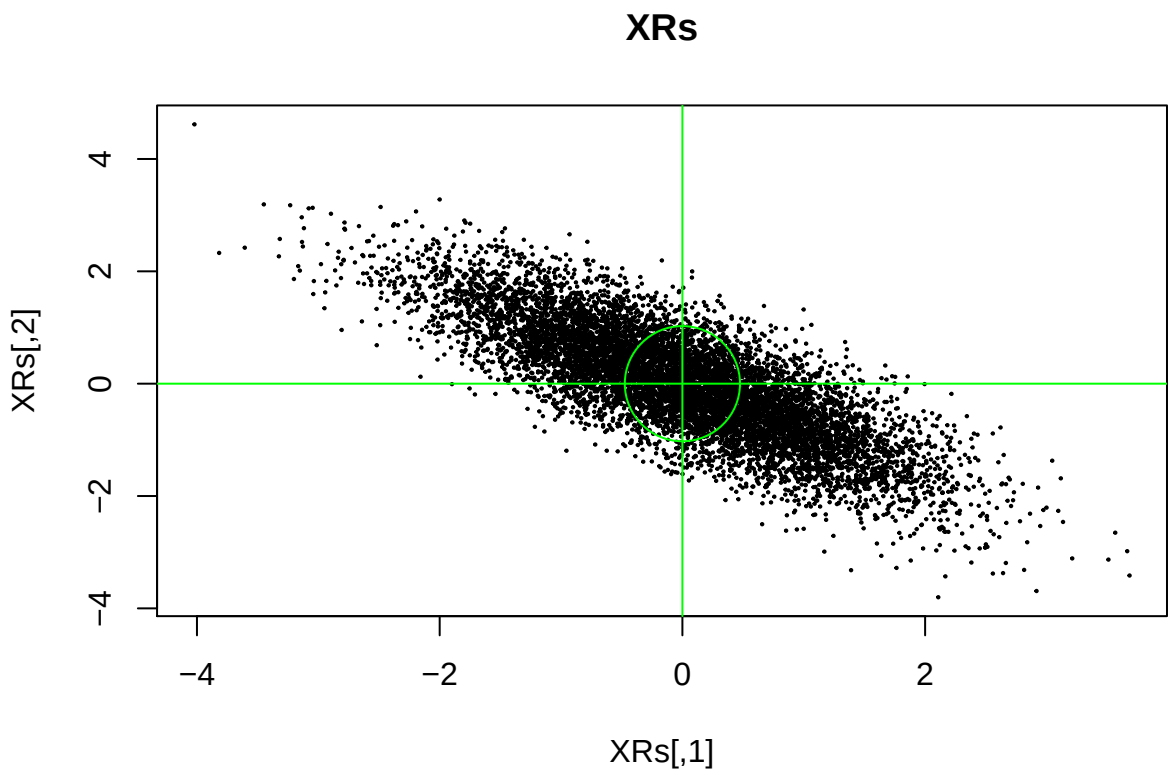
```
##  - attr(*, "class")= chr "princomp"
```

```
Xs.PCA.scores = Xs.PCA$scores;

plot(Xs.PCA.scores, pch=20, cex=0.25, main="Xs");
  abline(v=mean(Xs.PCA.scores[,1]), col="orange");
  abline(h=mean(Xs.PCA.scores[,2]), col="orange");
points(x=mean(Xs.PCA.scores[,1]),y=mean(Xs.PCA.scores[,2]),
          pch=21, col="orange", cex=8);
```




```
# Xs.PCA

# XR[,1] = x * cos(deg2rad(60)) - y * sin(deg2rad(60));
```

```
plot(XRs, pch=20, cex=0.25, main="XRs");
  abline(v=mean(XRs[,1]), col="green");
  abline(h=mean(XRs[,2]), col="green");
points(x=mean(XRs[,1]),y=mean(XRs[,2]),
          pch=21, col="green", cex=8);
```



XRs

```
XRs.PCA = princomp(XRs);

summary(XRs.PCA);
```

```
## Importance of components:
##                            Comp.1     Comp.2
## Standard deviation     1.3436177 0.44098663
## Proportion of Variance 0.9027546 0.09724541
## Cumulative Proportion  0.9027546 1.00000000
```

```
str(XRs.PCA);
```

```
## List of 7
##  $ sdev    : Named num [1:2] 1.344 0.441
##   ..- attr(*, "names")= chr [1:2] "Comp.1" "Comp.2"
##  $ loadings: 'loadings' num [1:2, 1:2] 0.707 -0.707 0.707 0.707
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "Comp.1" "Comp.2"
##  $ center  : num [1:2] -0.000000000000000428 -0.000000000000000359
##  $ scale   : num [1:2] 1 1
##  $ n.obs   : int 9000
##  $ scores  : num [1:9000, 1:2] -1.351 0.757 0.23 0.188 1.426 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "Comp.1" "Comp.2"
##  $ call    : language princomp(x = XRs)
```

```
##  - attr(*, "class")= chr "princomp"
```

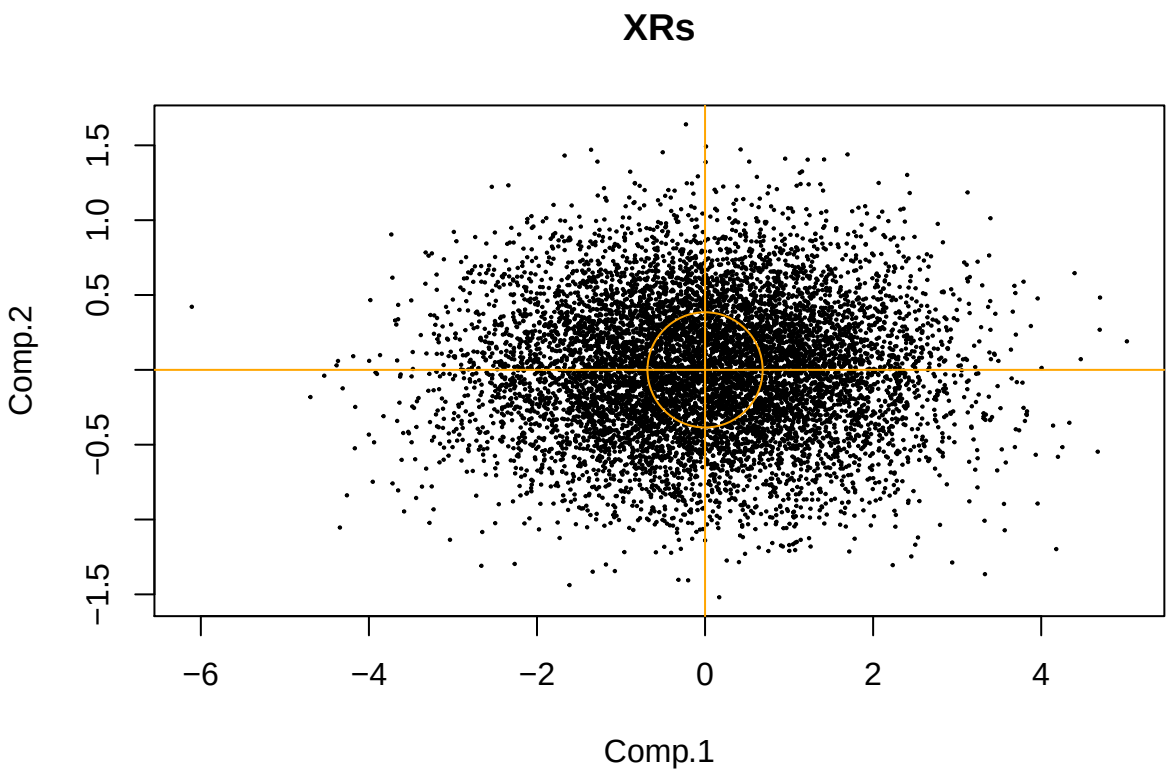
```
summary(XRs.PCA);
```

```
## Importance of components:
##                           Comp.1     Comp.2
## Standard deviation     1.3436177 0.44098663
## Proportion of Variance 0.9027546 0.09724541
## Cumulative Proportion  0.9027546 1.00000000
```

```
str(Xs.PCA);
```

```
## List of 7
##  $ sdev    : Named num [1:2] 1.013 0.987
##   ..- attr(*, "names")= chr [1:2] "Comp.1" "Comp.2"
##  $ loadings: 'loadings' num [1:2, 1:2] 0.707 0.707 0.707 -0.707
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "Comp.1" "Comp.2"
##  $ center  : num [1:2] 0.0000000000000000552 -0.0000000000000000431
##  $ scale   : num [1:2] 1 1
##  $ n.obs   : int 9000
##  $ scores  : num [1:9000, 1:2] 0.844 -1.016 -0.19 0.959 -0.657 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:2] "Comp.1" "Comp.2"
##  $ call    : language princomp(x = Xs)
##  - attr(*, "class")= chr "princomp"
```

```
XRs.PCA.scores = XRs.PCA$scores;

plot(XRs.PCA.scores, pch=20, cex=0.25, main="XRs");
  abline(v=mean(XRs.PCA.scores[,1]), col="orange");
  abline(h=mean(XRs.PCA.scores[,2]), col="orange");
points(x=mean(XRs.PCA.scores[,1]),y=mean(XRs.PCA.scores[,2]),
          pch=21, col="orange", cex=8);
```



3-Dimensions

Elliptical

If the variances would be equal, these ellipses would be circles.

```
# cache.rebuild ... This will prevent Xs from being used from previous cache ...

library(scatterplot3d);
```

```
## Warning: package 'scatterplot3d' was built under R version 4.0.3
```

```
library(rgl);
```

```
## Warning: package 'rgl' was built under R version 4.0.3
```

```
set.seed(1222015);

mu = c(1,3,8); # centers for x,y
Sigma = diag(c(2,23,13)); # variance for x,y

X = rmvn(nsim, mu, Sigma, ncores=2);  # this is parallelizability with cores

xyz.lim = c(min(X), max(X)); # square

print("#################   X   #################");
```

```
## [1] "#################   X   #################"
```

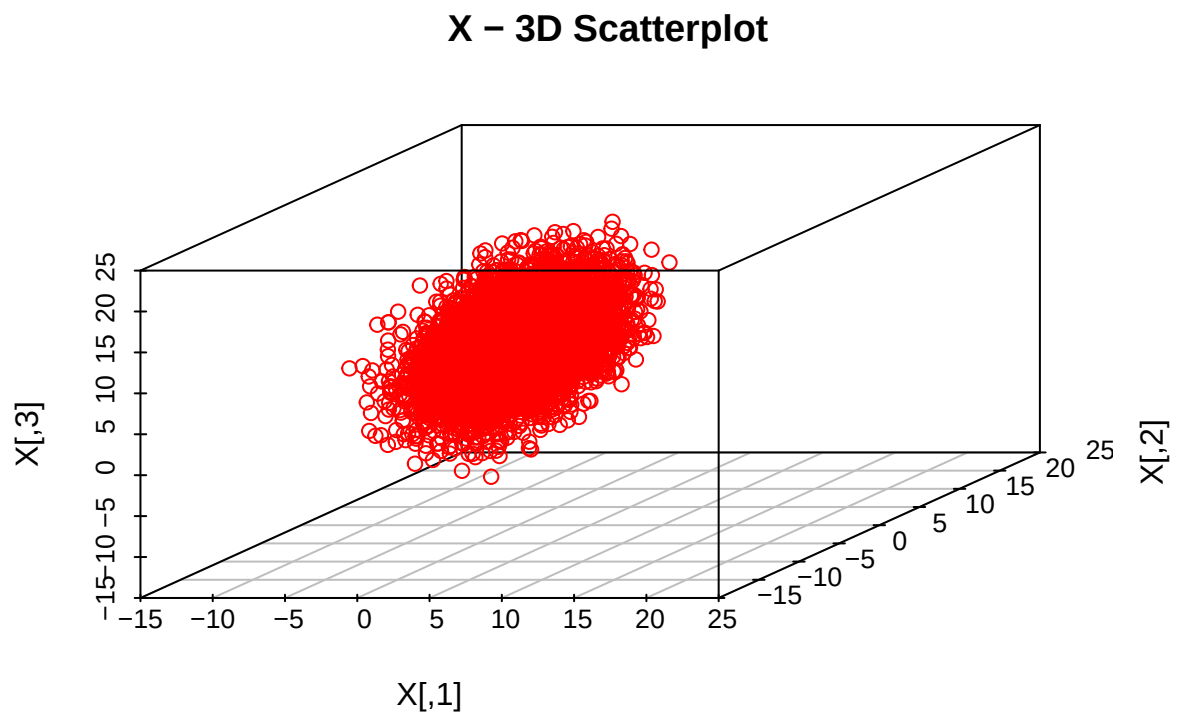
```
print(paste0("MEANS:    x = ",round(mean(X[,1]),3),
             "       y = ",round(mean(X[,2]),3),
             "       z = ",round(mean(X[,3]),3)));
```

```
## [1] "MEANS:    x = 1.015       y = 3.043       z = 7.94"
```

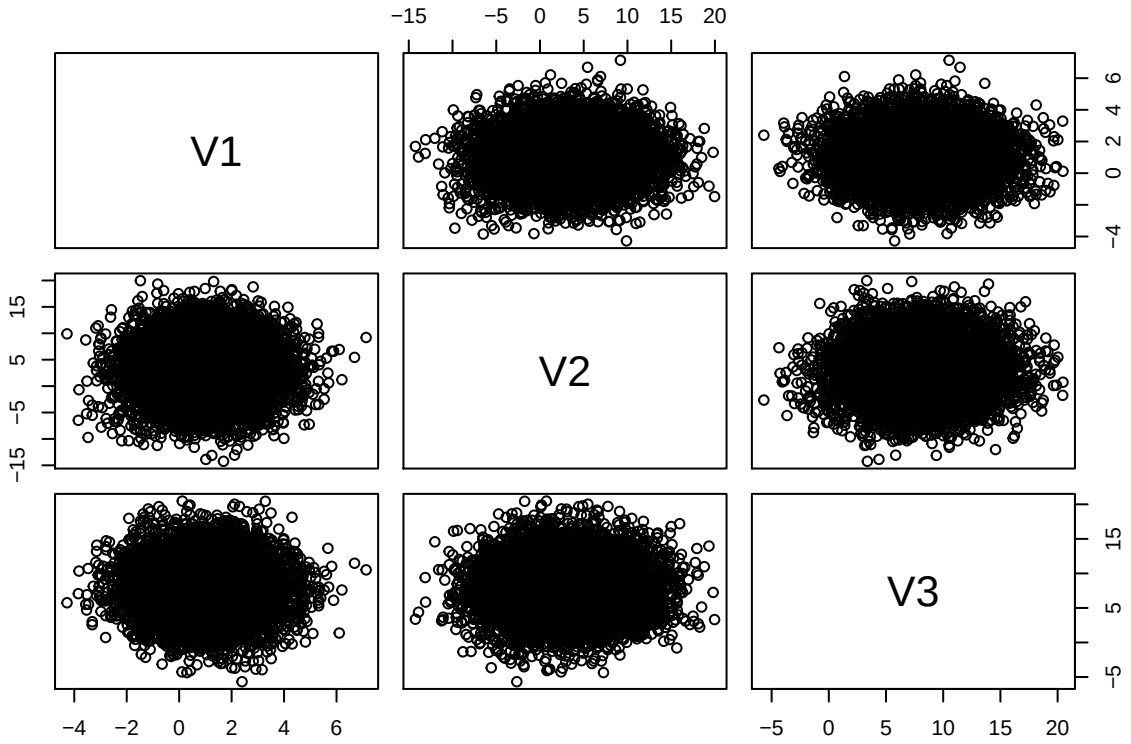
```
print(paste0("VARIANCE: x = ",round(var(X[,1]),3),
             "       y = ",round(var(X[,2]),3),
             "       z = ",round(var(X[,3]),3)));
```

```
## [1] "VARIANCE: x = 2.003       y = 22.924       z = 12.666"
```

```
scatterplot3d(X, xlim=xyz.lim, ylim=xyz.lim, zlim=xyz.lim, highlight.3d=FALSE, main="X - 3D Scatterplot
```




```
# this is interactive, and will open in its own window
plot3d(X, xlim=xyz.lim, ylim=xyz.lim, zlim=xyz.lim, main="X - 3D plot",col="red" );

graphics::plot( as.data.frame(X) );
```




```
Xs = scale(X);

print("#################   Xs   #################");
```

```
## [1] "#################   Xs   #################"
```

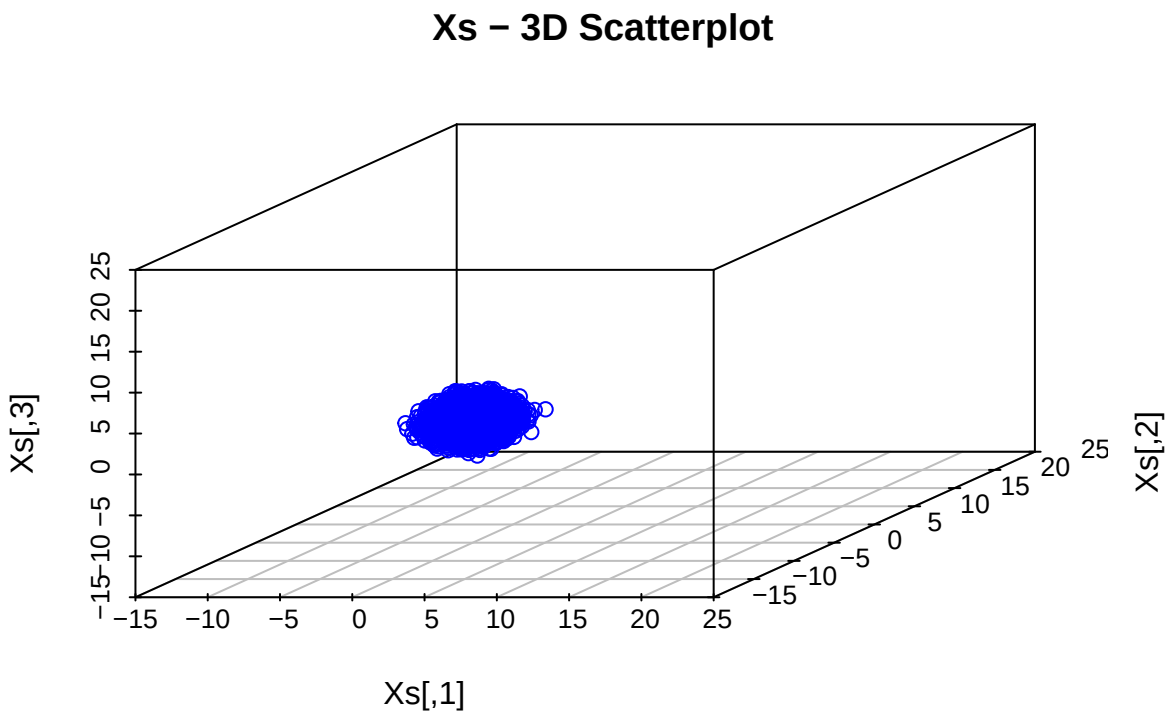
```
print(paste0("MEANS:    x = ",round(mean(Xs[,1]),3),
             "       y = ",round(mean(Xs[,2]),3),
             "       z = ",round(mean(Xs[,3]),3)));
```

```
## [1] "MEANS:    x = 0       y = 0       z = 0"
```

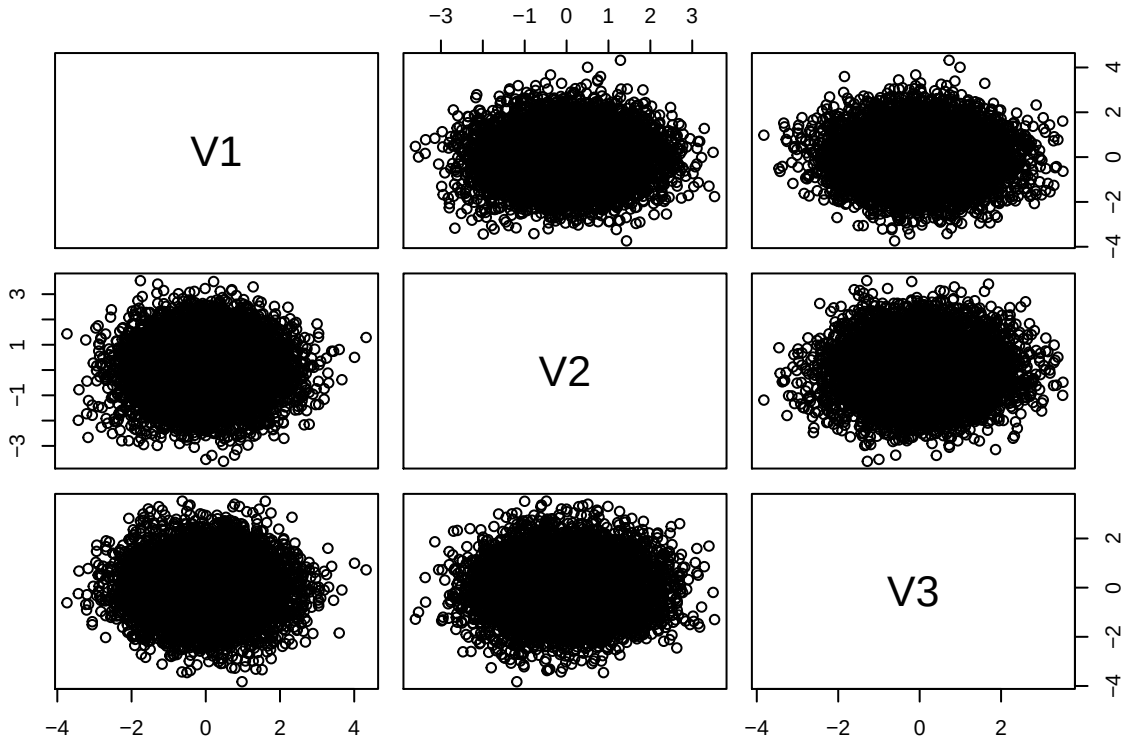
```
print(paste0("VARIANCE: x = ",round(var(Xs[,1]),3),
             "       y = ",round(var(Xs[,2]),3),
             "       z = ",round(var(Xs[,3]),3)));
```

```
## [1] "VARIANCE: x = 1       y = 1       z = 1"
```

```
scatterplot3d(Xs, xlim=xyz.lim, ylim=xyz.lim, zlim=xyz.lim, highlight.3d=FALSE, main="Xs - 3D Scatterpl
```




```
# this is interactive, and will open in its own window
plot3d(Xs, xlim=xyz.lim, ylim=xyz.lim, zlim=xyz.lim, main="Xs - 3D plot",col="blue" );

graphics::plot( as.data.frame(Xs) );
```



It is called "scaling" for a reason. A translation in the (x,y) to "mean-center" at zero. And a scaling factor of (x,y) ... e.g., our z-scores.

I keep the `xlim` and `ylim` the same to see this ("red" vs "blue"). Then I let "green" auto-scale, although it is equal to "blue".

Rotated

We can take that same data and rotate it ... eventually we will call this `phi` ... currently, hardcoded as $\phi = 60$ in degrees, so I have the helper functions which I have placed in `functions-maths.R`.

```
XR = X; # let's manually rotate 60 ...

x = X[,1];
y = X[,2];
z = X[,3];

# rotate around z-axis by angle phi
# https://stackoverflow.com/questions/20759214/
r = sqrt(x*x + y*y);
theta = atan(y/x);
phi = deg2rad(60);
# (r * cos(theta + phi), r * sin(theta + phi))

XR[,1] = r * cos(theta + phi);  # x
XR[,1] = r * sin(theta + phi);  # y
```

```
xyzr.lim = c(min(XR),max(XR));


print("#################   XR   #################");
```

```
## [1] "#################   XR   #################"
```

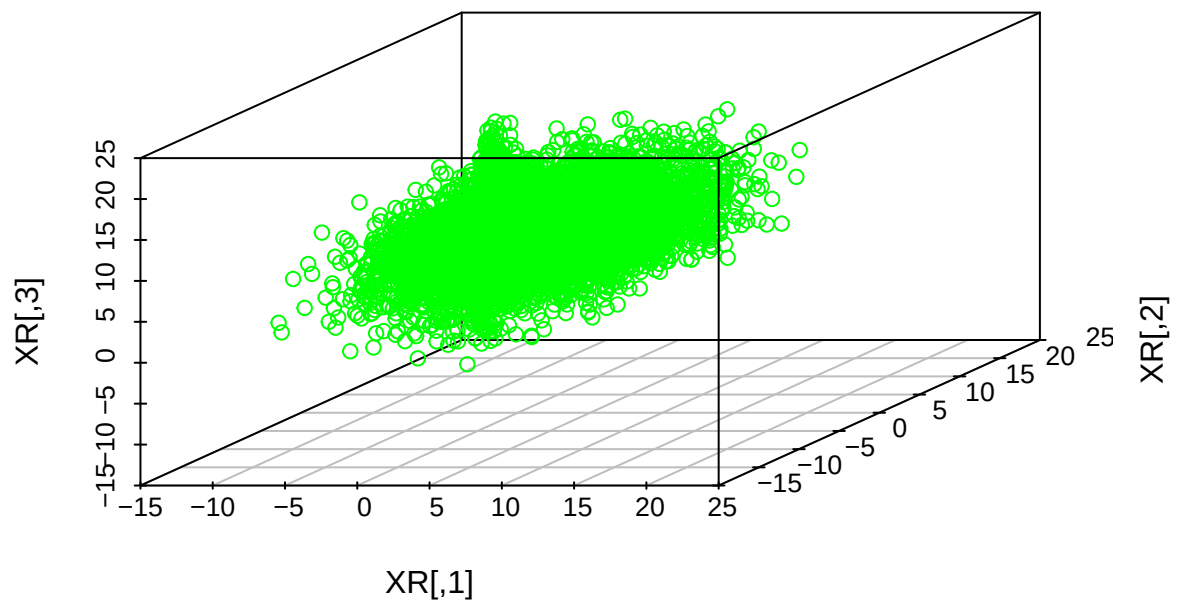
```
print(paste0("MEANS:    x = ",round(mean(XR[,1]),3),
             "       y = ",round(mean(XR[,2]),3),
             "       z = ",round(mean(XR[,3]),3)));
```

```
## [1] "MEANS:    x = 2.007       y = 3.043       z = 7.94"
```

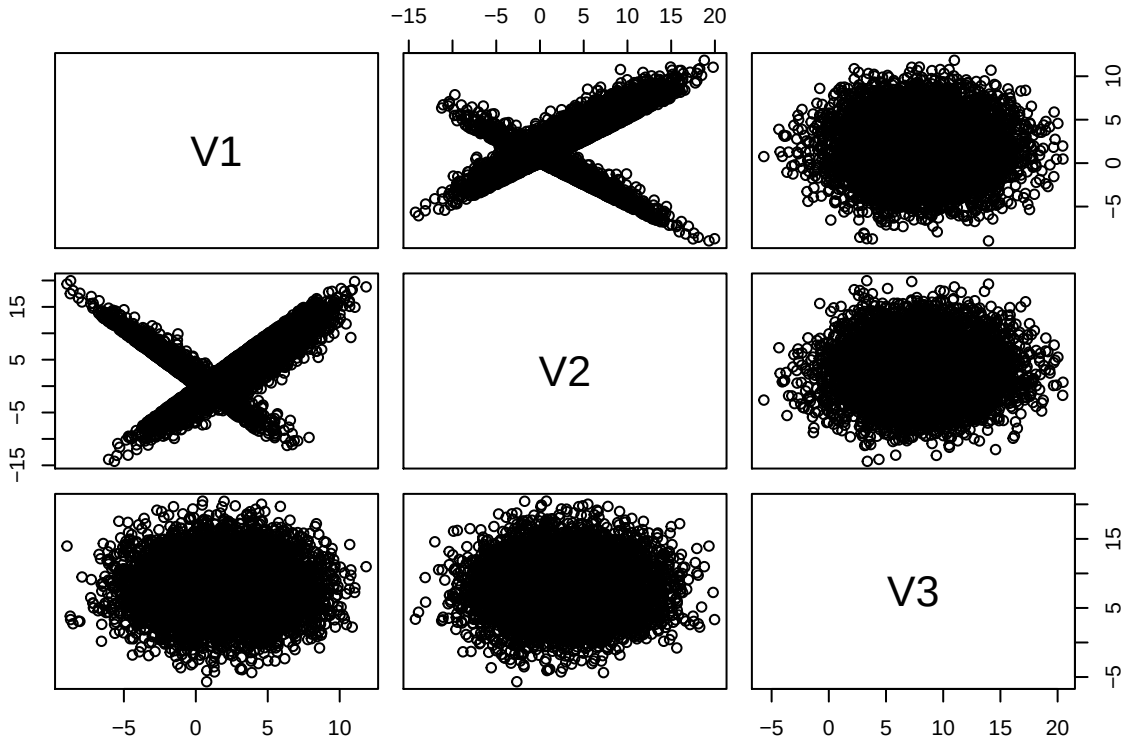
```
print(paste0("VARIANCE: x = ",round(var(XR[,1]),3),
             "       y = ",round(var(XR[,2]),3),
             "       z = ",round(var(XR[,3]),3)));
```

```
## [1] "VARIANCE: x = 9.005       y = 22.924       z = 12.666"
```

```
scatterplot3d(XR, xlim=xyzr.lim, ylim=xyzr.lim, zlim=xyzr.lim, highlight.3d=FALSE, main="XR - 3D Scatte
```




```
# this is interactive, and will open in its own window
plot3d(XR, xlim=xyzr.lim, ylim=xyzr.lim, zlim=xyzr.lim, main="XR - 3D plot", col="green");


graphics::plot( as.data.frame(XR) );
```




```
XRs = scale(XR);


print("#################   XRs   #################");
```

```
## [1] "#################   XRs   #################"
```

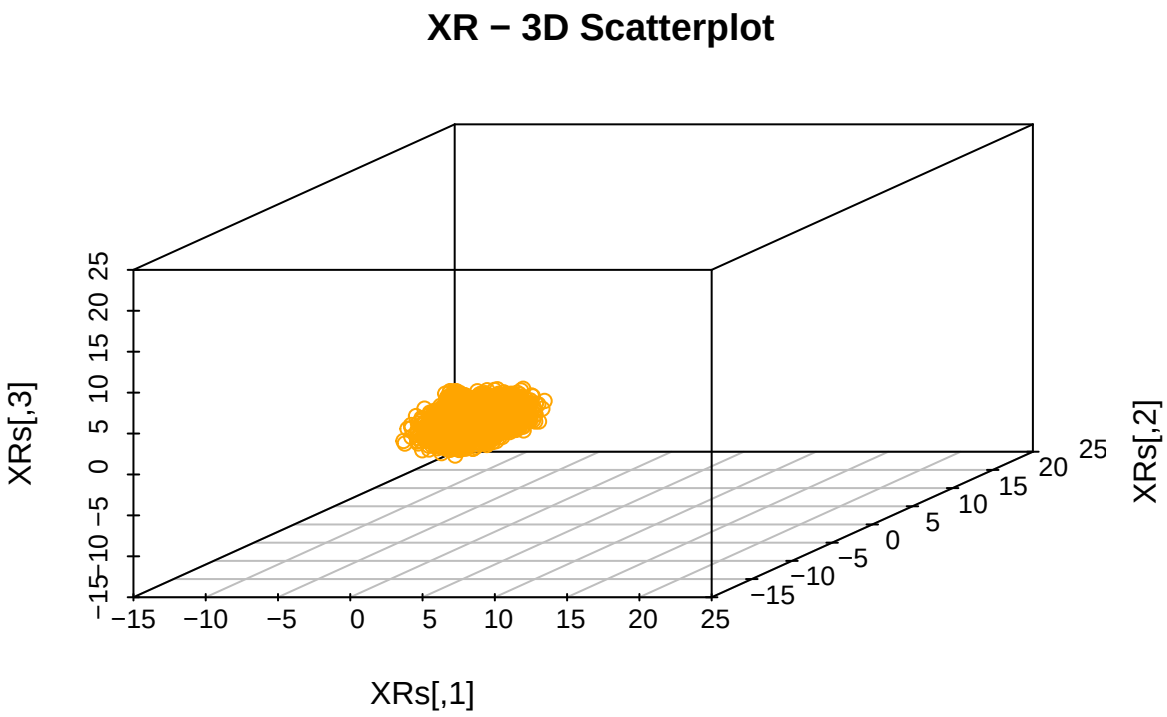
```
print(paste0("MEANS:    x = ",round(mean(XRs[,1]),3),
             "       y = ",round(mean(XRs[,2]),3),
             "       z = ",round(mean(XRs[,3]),3)));
```

```
## [1] "MEANS:    x = 0       y = 0       z = 0"
```

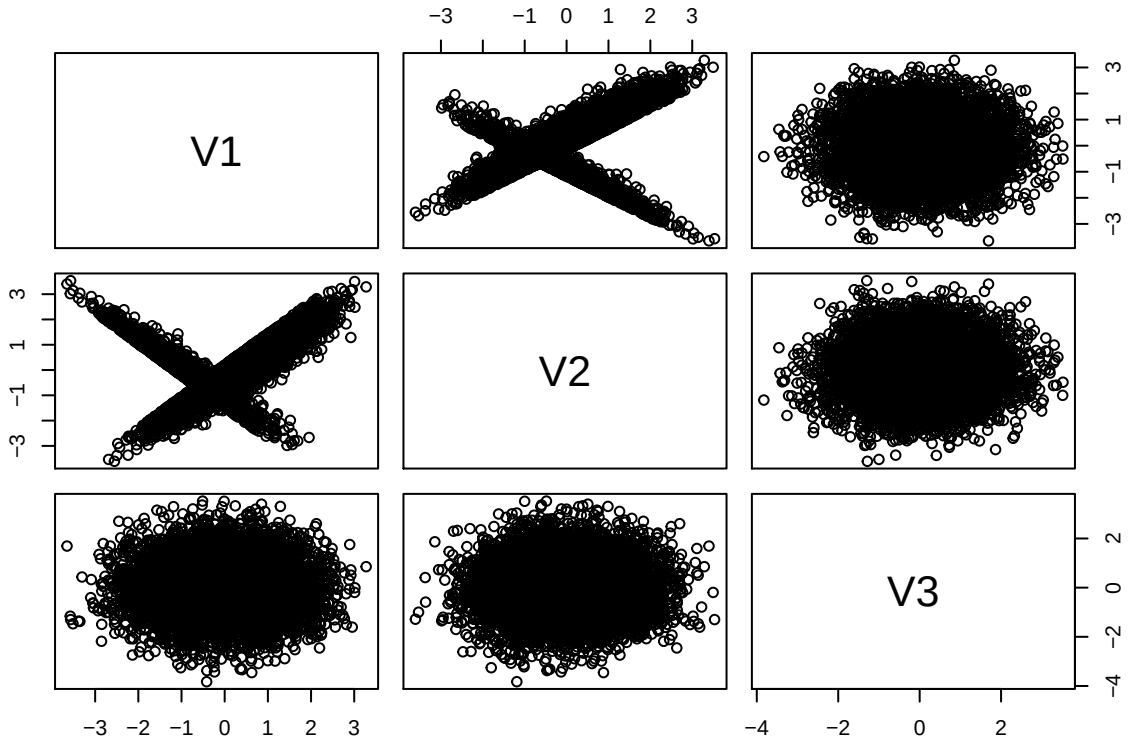
```
print(paste0("VARIANCE: x = ",round(var(XRs[,1]),3),
             "       y = ",round(var(XRs[,2]),3),
             "       z = ",round(var(XRs[,3]),3)));
```

```
## [1] "VARIANCE: x = 1       y = 1       z = 1"
```

```
scatterplot3d(XRs, xlim=xyzr.lim, ylim=xyzr.lim, zlim=xyzr.lim, highlight.3d=FALSE, main="XR - 3D Scatt
```




```
# this is interactive, and will open in its own window
plot3d(XRs, xlim=xyzr.lim, ylim=xyzr.lim, zlim=xyzr.lim, main="XR - 3D plot", col="orange");

graphics::plot( as.data.frame(XRs) );
```

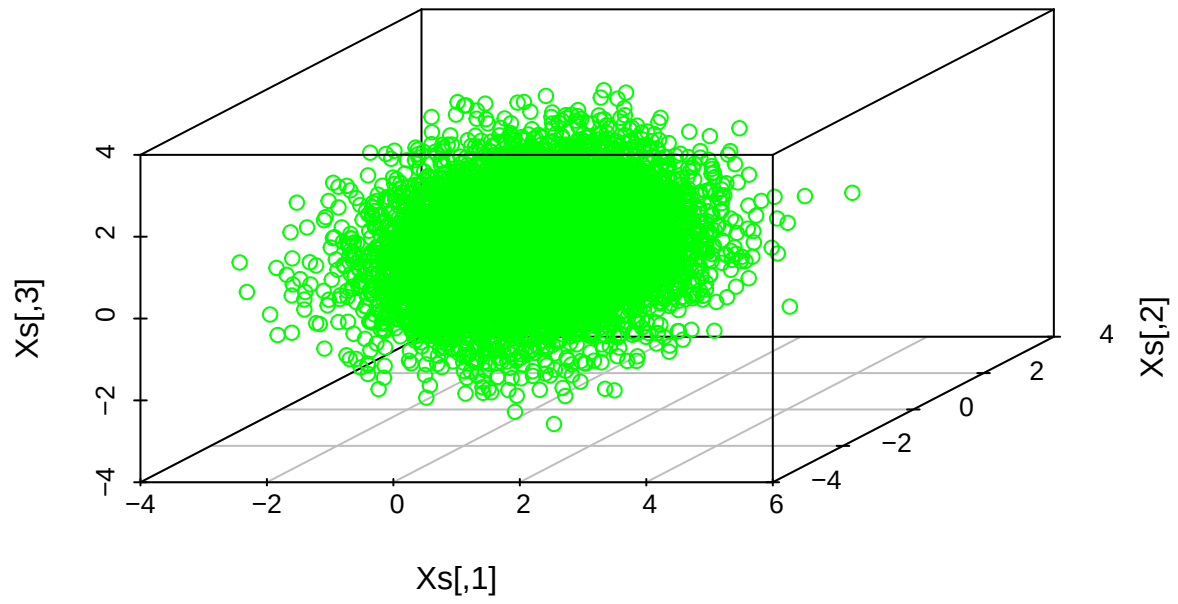


Translations, scalings, and rotations are not changing the overall basis.

Principle Components Analysis of Xs

If we don't scale, the one dimension will outweigh another dimension. This will create uninterpretable results.

```
scatterplot3d(Xs, highlight.3d=FALSE, main="Xs - 3D Scatterplot", color="green" );
```




```
Xs.PCA = princomp(Xs);

summary(Xs.PCA);
```

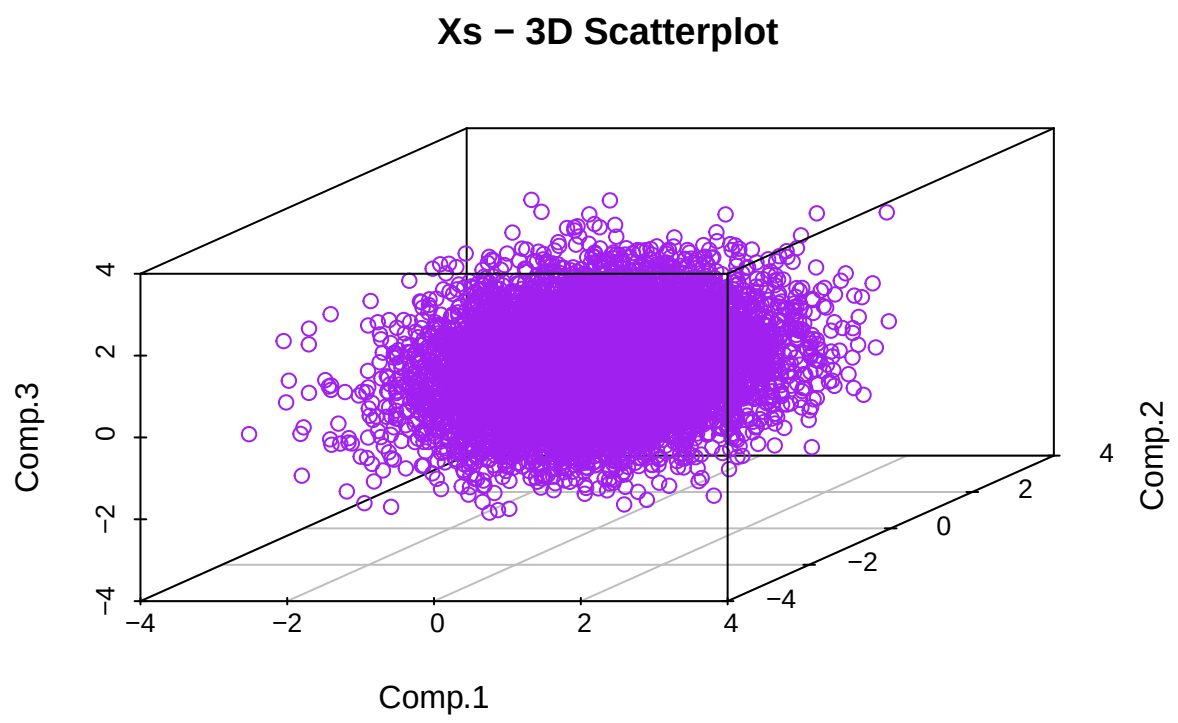
```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3
## Standard deviation     1.0069151 1.0003637 0.9925025
## Proportion of Variance 0.3379969 0.3336129 0.3283902
## Cumulative Proportion  0.3379969 0.6716098 1.0000000
```

```
str(Xs.PCA);
```

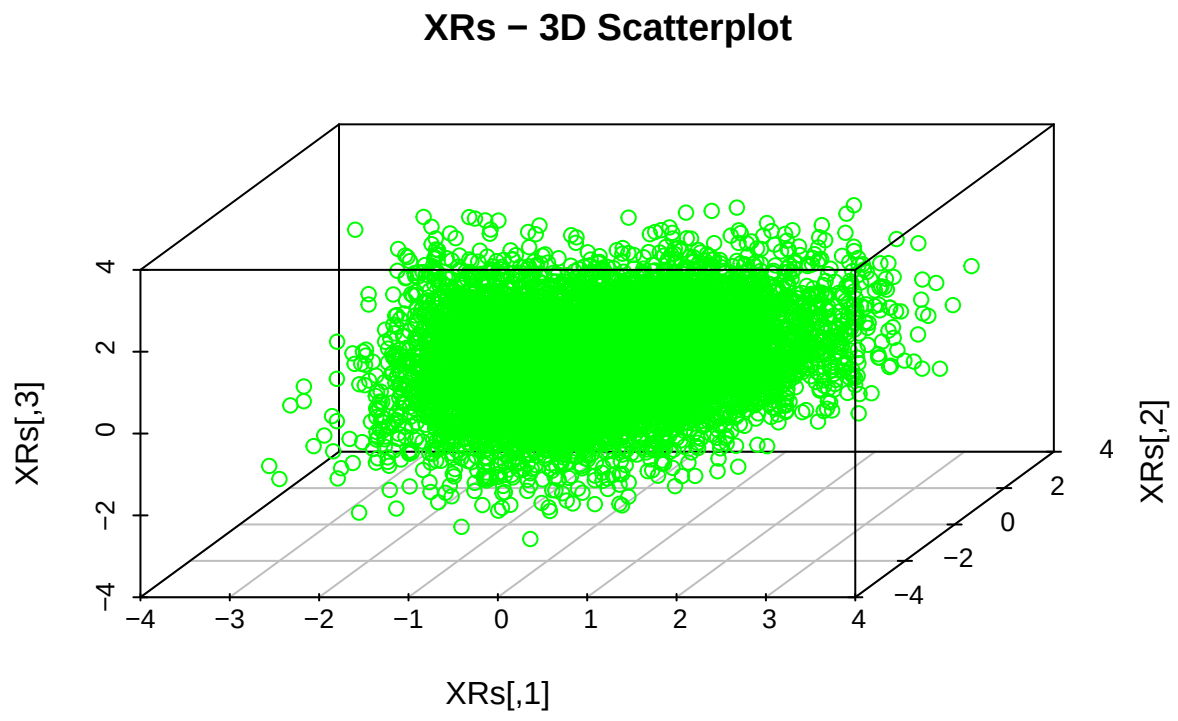
```
## List of 7
##  $ sdev    : Named num [1:3] 1.007 1 0.993
##   ..- attr(*, "names")= chr [1:3] "Comp.1" "Comp.2" "Comp.3"
##  $ loadings: 'loadings' num [1:3, 1:3] 0.7167 0.3038 -0.6277 0.0389 0.8813 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "Comp.1" "Comp.2" "Comp.3"
##  $ center  : num [1:3] 0.000000000000000712 0.000000000000000191 -0.000000000000000486
##  $ scale   : num [1:3] 1 1 1
##  $ n.obs   : int 9000
##  $ scores  : num [1:9000, 1:3] 1.069 -0.247 1.002 -0.542 0.342 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "Comp.1" "Comp.2" "Comp.3"
##  $ call    : language princomp(x = Xs)
```

```
##  - attr(*, "class")= chr "princomp"
```

```
Xs.PCA.scores = Xs.PCA$scores;

scatterplot3d(Xs.PCA.scores, highlight.3d=FALSE, main="Xs - 3D Scatterplot", color="purple" );
```




```
scatterplot3d(XRs, highlight.3d=FALSE, main="XRs - 3D Scatterplot", color="green" );
```




```
XRs.PCA = princomp(XRs);

summary(XRs.PCA);
```

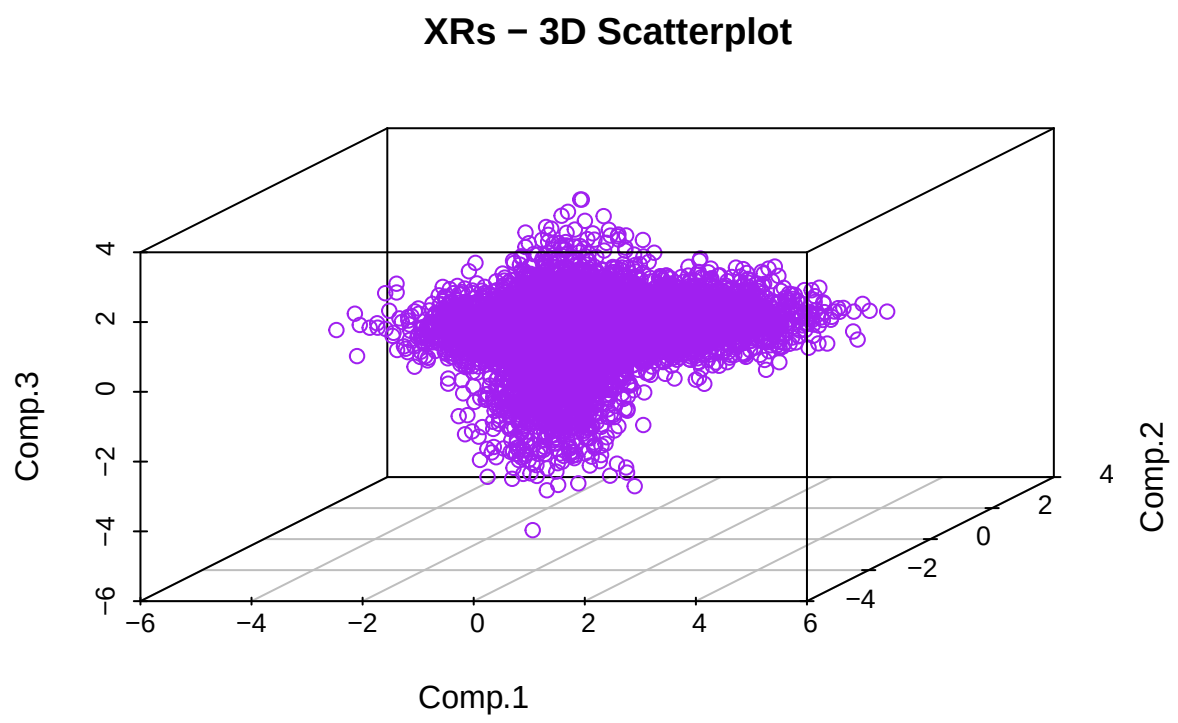
```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3
## Standard deviation     1.1890519 0.9999577 0.7654455
## Proportion of Variance 0.4713338 0.3333422 0.1953240
## Cumulative Proportion  0.4713338 0.8046760 1.0000000
```

```
str(XRs.PCA);
```

```
## List of 7
##  $ sdev    : Named num [1:3] 1.189 1 0.765
##   ..- attr(*, "names")= chr [1:3] "Comp.1" "Comp.2" "Comp.3"
##  $ loadings: 'loadings' num [1:3, 1:3] 0.70712 0.70706 -0.00761 0.00243 -0.01319 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "Comp.1" "Comp.2" "Comp.3"
##  $ center  : num [1:3] 0.000000000000000739 0.000000000000000191 -0.000000000000000486
##  $ scale   : num [1:3] 1 1 1
##  $ n.obs   : int 9000
##  $ scores  : num [1:9000, 1:3] 1.4111 -0.1523 0.0802 -0.842 -0.0758 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "Comp.1" "Comp.2" "Comp.3"
##  $ call    : language princomp(x = XRs)
```

```
##  - attr(*, "class")= chr "princomp"
```

```
XRs.PCA.scores = XRs.PCA$scores;

scatterplot3d(XRs.PCA.scores, highlight.3d=FALSE, main="XRs - 3D Scatterplot", color="purple" );
```



n-Dimensions

We could continue this for many dimensions. After 3-D, we have an issue with plotting them all at the same time.

```
graphics::plot( as.data.frame(XRs) );
```

We can do pairwise plots with the above.

A Data Example

This includes athletic records for 55 countries in each of 8 track and field events. This was likely in the 1980s, but is still a nice learning example.

Change path to dataset

```
## DROPBOX ... __student_access__ ...
## You need to change this ...
path.dataset = "C:/Users/Galac/Desktop/git419/Stats419_FALL2020/datasets/Multivariate-2009/datasets/";

file.running = paste0(path.dataset,"RECORDS.csv");
```

```
myData = read.csv(file.running,header=FALSE);
  colnames(myData)=c("Country","100m","200m","400m",
                                "800m","1500m","5000m",
                               "10000m","marathon");

myData;
```

```
##        Country  100m  200m  400m 800m 1500m 5000m 10000m marathon
## 1    Argentina 10.39 20.81 46.84 1.81  3.70 14.04  29.36   137.72
## 2    Australia 10.31 20.06 44.84 1.74  3.57 13.28  27.66   128.30
## 3      Austria 10.44 20.81 46.82 1.79  3.60 13.26  27.72   135.90
## 4      Belgium 10.34 20.68 45.04 1.73  3.60 13.22  27.45   129.95
## 5      Bermuda 10.28 20.58 45.91 1.80  3.75 14.68  30.55   146.62
## 6       Brazil 10.22 20.43 45.21 1.73  3.66 13.62  28.62   133.13
## 7        Burma 10.64 21.52 48.30 1.80  3.85 14.45  30.28   139.95
## 8       Canada 10.17 20.22 45.68 1.76  3.63 13.55  28.09   130.15
## 9        Chile 10.34 20.80 46.20 1.79  3.71 13.62  29.30   134.03
## 10       China 10.51 21.04 47.30 1.81  3.73 13.90  29.13   133.53
## 11    Columbia 10.43 21.05 46.10 1.82  3.74 13.49  27.88   131.35
## 12      CookIs 12.18 23.20 52.94 2.02  4.24 16.70  35.38   164.70
## 13   CostaRica 10.94 21.90 48.66 1.87  3.84 14.03  28.81   136.58
## 14       Czech 10.35 20.65 45.64 1.76  3.58 13.42  28.19   134.32
## 15     Denmark 10.56 20.52 45.89 1.78  3.61 13.50  28.11   130.78
## 16      DomRep 10.14 20.65 46.80 1.82  3.82 14.91  31.45   154.12
## 17     Finland 10.43 20.69 45.49 1.74  3.61 13.27  27.52   130.87
## 18      France 10.11 20.38 45.28 1.73  3.57 13.34  27.97   132.30
## 19         GDR 10.12 20.33 44.87 1.73  3.56 13.17  27.42   129.92
## 20         FRG 10.16 20.37 44.50 1.73  3.53 13.21  27.61   132.23
## 21          GB 10.11 20.21 44.93 1.70  3.51 13.01  27.51   129.13
## 22      Greece 10.22 20.71 46.56 1.78  3.64 14.59  28.45   134.60
## 23   Guatemala 10.98 21.82 48.40 1.89  3.80 14.16  30.11   139.33
## 24     Hungary 10.26 20.62 46.02 1.77  3.62 13.49  28.44   132.58
## 25       India 10.60 21.42 45.73 1.76  3.73 13.77  28.81   131.98
## 26   Indonesia 10.59 21.49 47.80 1.84  3.92 14.73  30.79   148.83
## 27     Ireland 10.61 20.96 46.30 1.79  3.56 13.32  27.81   132.35
## 28      Israel 10.71 21.00 47.80 1.77  3.72 13.66  28.93   137.55
## 29       Italy 10.01 19.72 45.26 1.73  3.60 13.23  27.52   131.08
## 30       Japan 10.34 20.81 45.86 1.79  3.64 13.41  27.72   128.63
## 31       Kenya 10.46 20.66 44.92 1.73  3.55 13.10  27.80   129.75
## 32       Korea 10.34 20.89 46.90 1.79  3.77 13.96  29.23   136.25
## 33      PKorea 10.91 21.94 47.30 1.85  3.77 14.13  29.67   130.87
## 34   Luxemburg 10.35 20.77 47.40 1.82  3.67 13.64  29.08   141.27
## 35    Malaysia 10.40 20.92 46.30 1.82  3.80 14.64  31.01   154.10
## 36    Mauritus 11.19 22.45 47.70 1.88  3.83 15.06  31.77   152.23
## 37      Mexico 10.42 21.30 46.10 1.80  3.65 13.46  27.95   129.20
## 38 Netherlands 10.52 29.95 45.10 1.74  3.62 13.36  27.61   129.02
## 39          NZ 10.51 20.88 46.10 1.74  3.54 13.21  27.70   128.98
## 40      Norway 10.55 21.16 46.71 1.76  3.62 13.34  27.69   131.48
## 41         Png 10.96 21.78 47.90 1.90  4.01 14.72  31.36   148.22
## 42 Philippines 10.78 21.64 46.24 1.81  3.83 13.74  30.64   145.27
## 43      Poland 10.16 20.24 45.36 1.76  3.60 13.29  27.89   131.58
## 44    Portugal 10.53 21.17 46.70 1.79  3.62 13.13  27.38   128.65
## 45     Rumania 10.41 20.98 45.87 1.76  3.64 13.25  27.67   132.50
## 46   Singapore 10.38 21.28 47.40 1.88  3.89 15.11  31.32   157.77
```

```
## 47       Spain 10.42 20.77 45.98 1.76  3.55 13.31  27.73   131.57
## 48      Sweden 10.25 20.61 45.63 1.77  3.61 13.29  27.94   130.63
## 49 Switzerland 10.37 20.45 45.78 1.78  3.55 13.22  27.91   131.20
## 50      Taipei 10.59 21.29 46.80 1.79  3.77 14.07  30.07   139.27
## 51    Thailand 10.39 21.09 47.91 1.83  3.84 15.23  32.56   149.90
## 52      Turkey 10.71 21.43 47.60 1.79  3.67 13.56  28.58   131.50
## 53         USA  9.93 19.75 43.86 1.73  3.53 13.20  27.43   128.22
## 54        USSR 10.07 20.00 44.60 1.75  3.59 13.20  27.53   130.55
## 55      WSamoa 10.82 21.86 49.00 2.02  4.24 16.28  34.71   161.83
```

```r
X = (myData[,-1]);
Xs = scale(X);

rownames(X) = rownames(Xs) = myData[,1]; # We still have the country names, important when we get to bi

# Xs;


performKMOTest(Xs);
```

```
## $KMO
## [1] 0.871196
##
## $msg
## [1] "meritorious!"
```

```r
performBartlettSphericityTest(Xs);
```

```
## $pvalue
## [1] 0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
##
## $msg
## [1] "Bartlett's test of sphericity ... pvalue < alpha ... 0.00000000000000000000000000000000000000000
```

Matrix Maths

```r
source_url( paste0(path.github, "humanVerseWSU/R/functions-str.R") );  # printMatrix
```

```
## SHA-1 hash of file is 5f57d466d946285783f15bb3b4d97ca48a951d0b
```

```r
n = nrow(Xs); n;  # number of observations
```

```
## [1] 55
```

```r
m = ncol(Xs); m;  # number of features
```

```
## [1] 8
```

```r
# let's use this syntax ... https://en.wikipedia.org/wiki/Principal_component_analysis


# why does covariance equal correlation
# it's scaled
# isClose deals with rounding/floating-point issues
print("Comparing covariance and correlation of Xs");
```

```
## [1] "Comparing covariance and correlation of Xs"
```

```
isClose( as.numeric( cov(Xs) ), as.numeric( cor(Xs) ) );
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [61] TRUE TRUE TRUE TRUE
```

```
# cor(Xs) or cor(X)

S = ( transposeMatrix(as.matrix(Xs)) %*% as.matrix(Xs)) / (n-1);
dim(S);
```

```
## [1] 8 8
```

```
print("Comparing S and the covariance of Xs");
```

```
## [1] "Comparing S and the covariance of Xs"
```

```
isClose( as.numeric( S ), as.numeric( cov(Xs) ) );
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [61] TRUE TRUE TRUE TRUE
```

```
S.eigen = eigen(S); # eigenvalues again
str(S.eigen);
```

```
## List of 2
##  $ values : num [1:8] 6.004 1.039 0.551 0.138 0.109 ...
##  $ vectors: num [1:8, 1:8] -0.323 -0.146 -0.369 -0.387 -0.393 ...
##  - attr(*, "class")= chr "eigen"
```

```
Lambda = S.eigen$values;
length(Lambda);   # notice it is of length "m"
```

```
## [1] 8
```

```
# S - Lambda * I, its determinant
round( det(S - diag(Lambda)), digits=4);
```

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

```
D = diag(Lambda);  # Lambda * I
printMatrix(D, 5);
```

```
##      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] "6.00426" "0"       "0"       "0"       "0"       "0"       "0"
## [2,] "0"       "1.03903" "0"       "0"       "0"       "0"       "0"
## [3,] "0"       "0"       "0.55148" "0"       "0"       "0"       "0"
## [4,] "0"       "0"       "0"       "0.13785" "0"       "0"       "0"
## [5,] "0"       "0"       "0"       "0"       "0.10872" "0"       "0"
## [6,] "0"       "0"       "0"       "0"       "0"       "0.07696" "0"
## [7,] "0"       "0"       "0"       "0"       "0"       "0"       "0.05712"
## [8,] "0"       "0"       "0"       "0"       "0"       "0"       "0"
##      [,8]
## [1,] "0"
```

```
## [2,] "0"
## [3,] "0"
## [4,] "0"
## [5,] "0"
## [6,] "0"
## [7,] "0"
## [8,] "0.02457"
```

```
D.sqrt = diag(sqrt(Lambda));
D.sqrt;
```

```
##          [,1]    [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 2.450359 0.00000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [2,] 0.000000 1.01933 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [3,] 0.000000 0.00000 0.7426153 0.0000000 0.0000000 0.0000000 0.0000000
## [4,] 0.000000 0.00000 0.0000000 0.3712804 0.0000000 0.0000000 0.0000000
## [5,] 0.000000 0.00000 0.0000000 0.0000000 0.3297324 0.0000000 0.0000000
## [6,] 0.000000 0.00000 0.0000000 0.0000000 0.0000000 0.2774204 0.0000000
## [7,] 0.000000 0.00000 0.0000000 0.0000000 0.0000000 0.0000000 0.2390073
## [8,] 0.000000 0.00000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##           [,8]
## [1,] 0.0000000
## [2,] 0.0000000
## [3,] 0.0000000
## [4,] 0.0000000
## [5,] 0.0000000
## [6,] 0.0000000
## [7,] 0.0000000
## [8,] 0.1567533
```

```
printMatrix(D.sqrt, 5);
```

```
##      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] "2.45036" "0"       "0"       "0"       "0"       "0"       "0"
## [2,] "0"       "1.01933" "0"       "0"       "0"       "0"       "0"
## [3,] "0"       "0"       "0.74262" "0"       "0"       "0"       "0"
## [4,] "0"       "0"       "0"       "0.37128" "0"       "0"       "0"
## [5,] "0"       "0"       "0"       "0"       "0.32973" "0"       "0"
## [6,] "0"       "0"       "0"       "0"       "0"       "0.27742" "0"
## [7,] "0"       "0"       "0"       "0"       "0"       "0"       "0.23901"
## [8,] "0"       "0"       "0"       "0"       "0"       "0"       "0"
##      [,8]
## [1,] "0"
## [2,] "0"
## [3,] "0"
## [4,] "0"
## [5,] "0"
## [6,] "0"
## [7,] "0"
## [8,] "0.15675"
```

```
W = S.eigen$vectors;  # orthogonal unit vectors
dim(W);
```

```
## [1] 8 8
```

```
T = Xs %*% W;
dim(T);
```

```
## [1] 55  8
```

```
# The transpose of W is sometimes called the whitening or sphering transformation.
#round( transposeMatrix(W), digits=2);  # https://en.wikipedia.org/wiki/Whitening_transformation
printMatrix( transposeMatrix(W) , 5);
```

```
##      [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## [1,] "-0.32267" "-0.14563" "-0.3691"  "-0.38652" "-0.3933"  "-0.3862"
## [2,] "-0.382"   "-0.83737" "-0.08773" "0.02629"  "0.06835"  "0.17924"
## [3,] "-0.55604" "0.5075"   "-0.43354" "-0.15423" "0.07048"  "0.23249"
## [4,] "-0.48351" "0.06708"  "0.06157"  "0.63796"  "0.35283"  "-0.05184"
## [5,] "0.42333"  "-0.10204" "-0.7895"  "0.30287"  "0.22402"  "-0.18809"
## [6,] "0.04919"  "-0.05831" "-0.05031" "-0.49338" "0.51316"  "0.3067"
## [7,] "-0.07764" "0.00583"  "0.16748"  "-0.26452" "0.58119"  "-0.66265"
## [8,] "-0.13905" "0.04119"  "0.10215"  "0.11835"  "-0.24474" "-0.43981"
##      [,7]       [,8]
## [1,] "-0.38934" "-0.36555"
## [2,] "0.18655"  "0.26999"
## [3,] "0.21056"  "0.34376"
## [4,] "-0.23638" "-0.40973"
## [5,] "0.09881"  "-0.00364"
## [6,] "0.23851"  "-0.57788"
## [7,] "-0.0791"  "0.33582"
## [8,] "0.80031"  "-0.24637"
```

```
VAF = round(Lambda / traceMatrix(S), digits=4);
VAF.cumsum = cumsum(VAF);

# loadings  ??
F = as.matrix(W) %*% D.sqrt;
dim(F);
```

```
## [1] 8 8
```

```
Z = ( W %*% D %*% transposeMatrix(W) );  # (n-1) is built into each component ...
dim(Z);
```

```
## [1] 8 8
```

```
isClose(S, Z);
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [61] TRUE TRUE TRUE TRUE
```

Summary of Some Maths

```
Answer = round( rbind(F, rep(NA,times=8), Lambda, VAF, VAF.cumsum)
            ,digits=4);
  rownames(Answer) =
                c("100m","200m","400m","800m",
                  "1500m","5000m","10000m","marathon",
```

```
                  "", "EIGEN","% VAF", "C. % VAF");
#Answer;

printMatrix(Answer, 4);
```

```
##          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## 100m     "-0.7907" "-0.3894" "-0.4129" "-0.1795" "0.1396"  "0.0136"  "-0.0186"
## 200m     "-0.3569" "-0.8536" "0.3769"  "0.0249"  "-0.0336" "-0.0162" "0.0014"
## 400m     "-0.9044" "-0.0894" "-0.322"  "0.0229"  "-0.2603" "-0.014"  "0.04"
## 800m     "-0.9471" "0.0268"  "-0.1145" "0.2369"  "0.0999"  "-0.1369" "-0.0632"
## 1500m    "-0.9637" "0.0697"  "0.0523"  "0.131"   "0.0739"  "0.1424"  "0.1389"
## 5000m    "-0.9463" "0.1827"  "0.1727"  "-0.0192" "-0.062"  "0.0851"  "-0.1584"
## 10000m   "-0.954"  "0.1902"  "0.1564"  "-0.0878" "0.0326"  "0.0662"  "-0.0189"
## marathon "-0.8957" "0.2752"  "0.2553"  "-0.1521" "-0.0012" "-0.1603" "0.0803"
##          NA        NA        NA        NA        NA        NA        NA
## EIGEN    "6.0043"  "1.039"   "0.5515"  "0.1378"  "0.1087"  "0.077"   "0.0571"
## % VAF    "0.7505"  "0.1299"  "0.0689"  "0.0172"  "0.0136"  "0.0096"  "0.0071"
## C. % VAF "0.7505"  "0.8804"  "0.9493"  "0.9665"  "0.9801"  "0.9897"  "0.9968"
##          [,8]
## 100m     "-0.0218"
## 200m     "0.0065"
## 400m     "0.016"
## 800m     "0.0186"
## 1500m    "-0.0384"
## 5000m    "-0.0689"
## 10000m   "0.1255"
## marathon "-0.0386"
##          NA
## EIGEN    "0.0246"
## % VAF    "0.0031"
## C. % VAF "0.9999"
```

Run PCA

```
Xs.princomp = stats::princomp(Xs);
  summary(Xs.princomp);   # , loadings=TRUE);
```

```
## Importance of components:
##                            Comp.1    Comp.2     Comp.3     Comp.4     Comp.5
## Standard deviation     2.4279804 1.0100212 0.73583333 0.36788968 0.32672113
## Proportion of Variance 0.7505322 0.1298793 0.06893469 0.01723115 0.01359044
## Cumulative Proportion  0.7505322 0.8804115 0.94934615 0.96657729 0.98016773
##                             Comp.6      Comp.7      Comp.8
## Standard deviation     0.274886821 0.236824556 0.155321763
## Proportion of Variance 0.009620259 0.007140562 0.003071451
## Cumulative Proportion  0.989787987 0.996928549 1.000000000
```

```
Xs.prcomp = stats::prcomp(Xs);
  summary(Xs.prcomp);
```

```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.4504 1.0193 0.74262 0.37128 0.32973 0.27742 0.23901
## Proportion of Variance 0.7505 0.1299 0.06893 0.01723 0.01359 0.00962 0.00714
```

```
## Cumulative Proportion  0.7505 0.8804 0.94935 0.96658 0.98017 0.98979 0.99693
##                            PC8
## Standard deviation     0.15675
## Proportion of Variance 0.00307
## Cumulative Proportion  1.00000
```

```
Xs.prcomp.E = zeroIsh ( cov(Xs.prcomp$x) );  # better than zapsmall

printMatrix(Xs.prcomp.E, 4);
```

```
##     PC1      PC2     PC3      PC4      PC5      PC6     PC7      PC8
## PC1 "6.0043" "0"     "0"      "0"      "0"      "0"     "0"      "0"
## PC2 "0"      "1.039" "0"      "0"      "0"      "0"     "0"      "0"
## PC3 "0"      "0"     "0.5515" "0"      "0"      "0"     "0"      "0"
## PC4 "0"      "0"     "0"      "0.1378" "0"      "0"     "0"      "0"
## PC5 "0"      "0"     "0"      "0"      "0.1087" "0"     "0"      "0"
## PC6 "0"      "0"     "0"      "0"      "0"      "0.077" "0"      "0"
## PC7 "0"      "0"     "0"      "0"      "0"      "0"     "0.0571" "0"
## PC8 "0"      "0"     "0"      "0"      "0"      "0"     "0"      "0.0246"
```

```
  #Xs.prcomp.E;


Xs.prcomp.lambda = diag(Xs.prcomp.E);
Xs.prcomp.lambda;
```

```
##        PC1        PC2        PC3        PC4        PC5        PC6        PC7
## 6.00425735 1.03903427 0.55147755 0.13784916 0.10872349 0.07696208 0.05712450
##        PC8
## 0.02457161
```

```
# From matrix "maths"  ... Equal except for some rounding errors ...
Lambda;
```

```
## [1] 6.00425735 1.03903427 0.55147755 0.13784916 0.10872349 0.07696208 0.05712450
## [8] 0.02457161
```

```
zeroIsh( Lambda - Xs.prcomp.lambda );
```

```
## PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8
##   0   0   0   0   0   0   0   0
```

```
# isClose is a much more precise comparison ... based on floating-point issues
# zeroIsh is just dropping values very close to zero as a form of 1/10^digits of accuracy
```

Variance Accounted For (VAF)

The first dimension is selected to maximize explaining the data. The proportion or percentage of variance explained for that dimension is reported (`% VAF`).

The second dimension is selected to be orthogonal to the first dimension. The proportion or percentage of variance explained for that dimension is reported (`% VAF`). And the cumulative proportion or percentage of variance is also recorded (`C. % VAF`).

The third dimension is selected to be orthogonal to the first and second dimension. The proportion or percentage of variance explained for that dimension is reported (`% VAF`). And the cumulative proportion or percentage of variance is also recorded (`C. % VAF`).

And, so on.

Dimension Selection

So how many dimensions are "good enough"? There are a few common "rules of thumb".

Kaiser rule

Choose `Lambda` values that are greater than one; that is eigenvalues greater than one.

"The advantage of the rule is that it is easy to calculate, especially if you live in the 1950s, and don't have access to a fast computer." https://stats.stackexchange.com/questions/253535/

Common "G" factor

When 80% of the variance is accounted for. That is the (`C. % VAF`) reaches `0.800` or higher.

Scree Plotting (e.g., similar to Exploratory Factor Analysis)

```
source_url( paste0(path.github, "humanVerseWSU/R/functions-EDA.R") );   # how many factors
```

```
## SHA-1 hash of file is 62ba3333da32792e57c410e3f02a443a4c7f4985
```

```
## Warning: package 'pvclust' was built under R version 4.0.3
```

```
## Warning: package 'factoextra' was built under R version 4.0.3
```

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

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

```
## Warning: package 'psych' was built under R version 4.0.3
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
Xs.how.many = howManyFactorsToSelect(Xs);
```

```
## [1] "  Paralell Analysis"
```

```
## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.
```

```
## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully
```

```
## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.
```

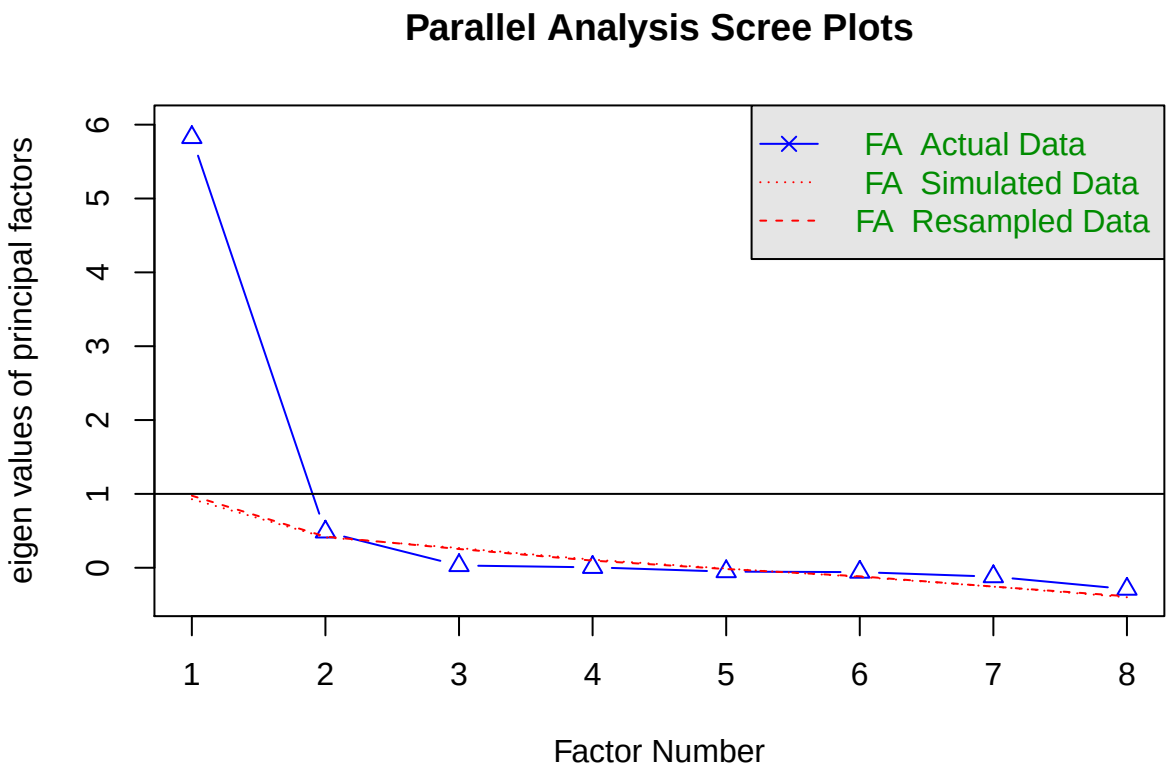
```
## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully
```

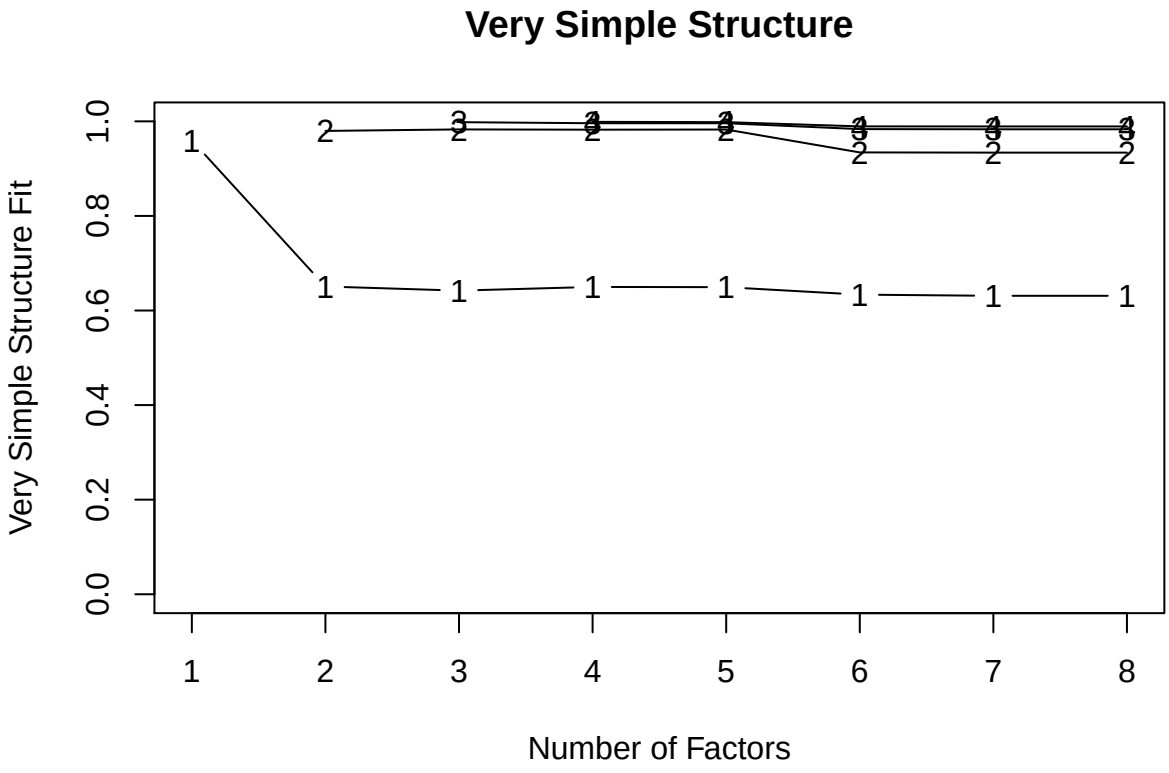
```
## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.
```

```
## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
```

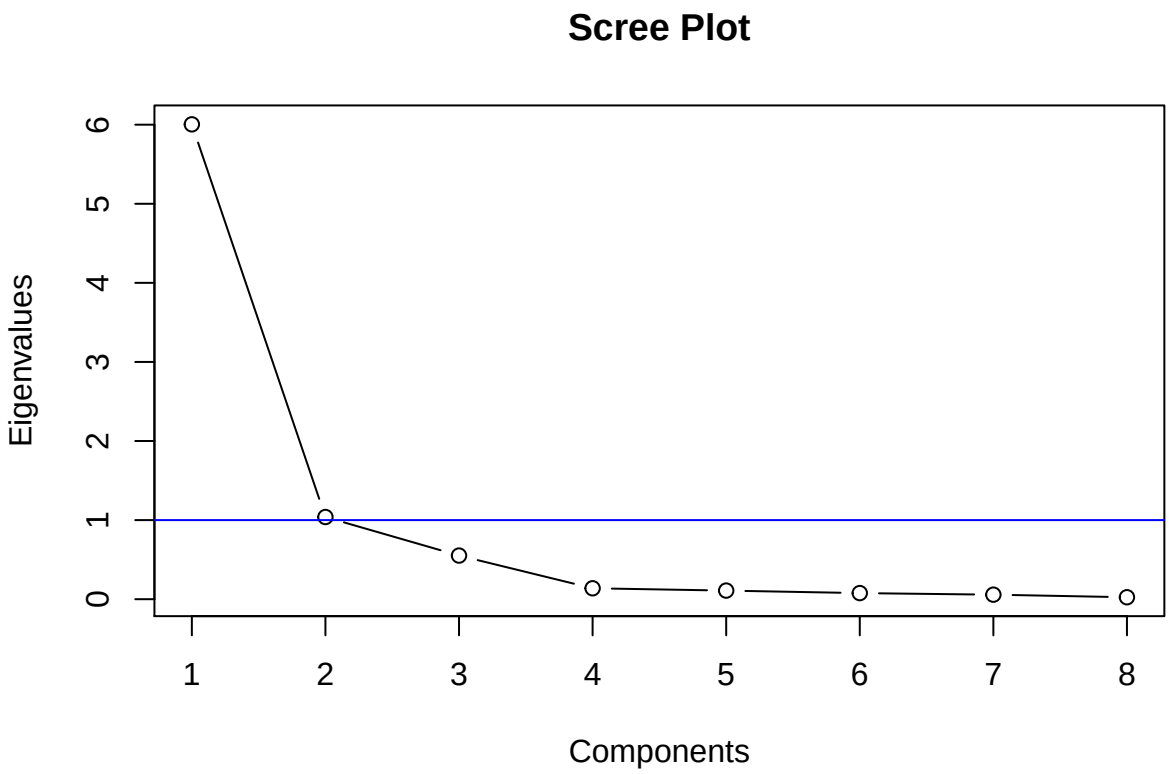
```
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.
```

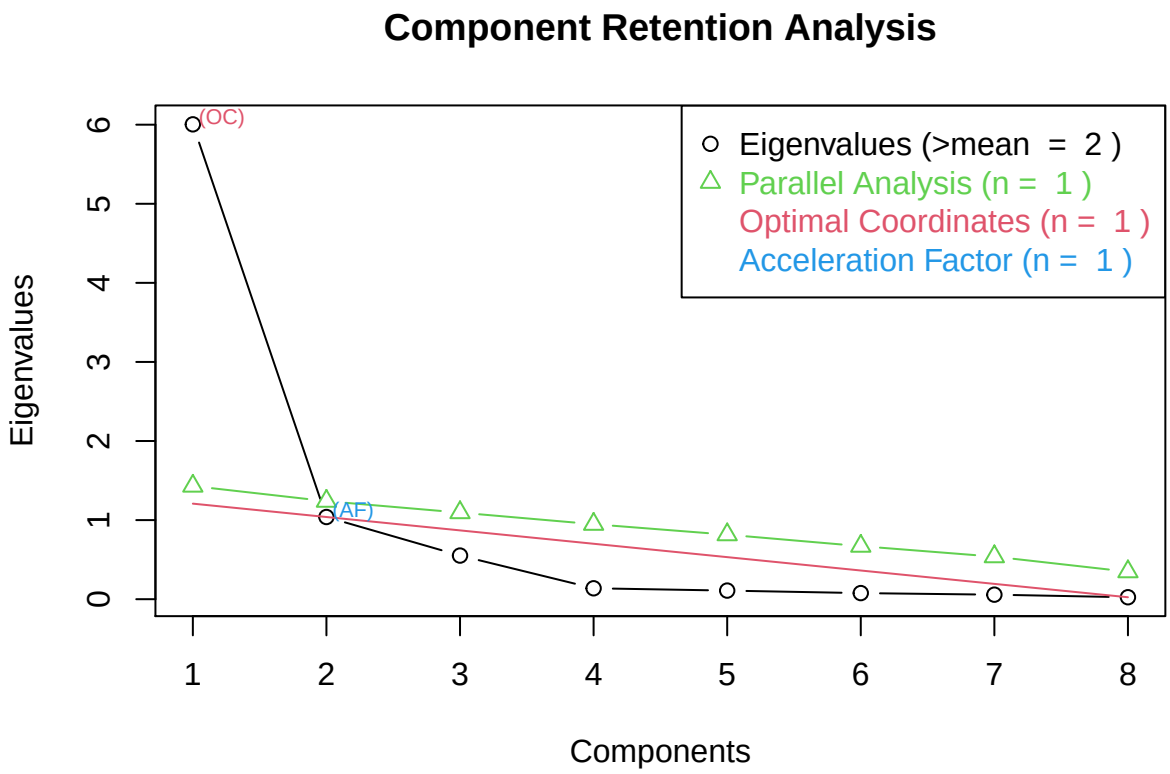
```
## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fac(r = r, nfactors = nfactors, n.obs = n.obs, rotate = rotate, : An
## ultra-Heywood case was detected. Examine the results carefully

## Parallel analysis suggests that the number of factors =  1  and the number of components =  NA
## [1] "============================================="
## [1] "  VSS Analysis"

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## The estimated weights for the factor scores are probably incorrect. Try a
## different factor score estimation method.

## Warning in fa.stats(r = r, f = f, phi = phi, n.obs = n.obs, np.obs = np.obs, :
## An ultra-Heywood case was detected. Examine the results carefully
```


Parallel Analysis Scree Plots
eigen values of principal factors
0
1
2
3
4
5
6
1
2
3
4
5
6
7
8
Factor Number
FA Actual Data
FA Simulated Data
FA Resampled Data




```
## [1] "************************"
## [1] "  Eigenvalues >= 1   ...  [ n = 2 ]"
## [1] 6.004257 1.039034
## [1] "************************"
```


Scree Plot
Eigenvalues
0
1
2
3
4
5
6
1
2
3
4
5
6
7
8
Components




```
## [1] "A 1-Factor solution has the most votes!"
## [1] ""
## [1] "Due to Optimal Coordinantes and Parallel Analysis Agreement,"
## [1] "A 1-Factor solution is *strongly* recommended!"
## [1] "************************"
## [1] "  Final Analysis of VSS, Eigen, nFactors"
##   Factor vote.count
## 1      1          4
## 2      2          2
## [1] ""
```

More Maths

The nature of PCA is similar to SVD. https://math.stackexchange.com/questions/3869/what-is-the-intuitive-relationship-between-svd-and-pca

Computationally, computing the S matrix (covariance) is expensive. SVD is less expensive, so for larger matrices, it is a best approach.

$$S = \frac{1}{n-1} X \cdot X^T$$

where X^T is the transpose of the data matrix. I am speaking generically or mathematically, which in this applied data case we are calling it the scaled `Xs`.

The "loadings" are a function of W (the "eigen" vectors): `W = S.eigen$vectors`

The "weights" of each component is based on its "eigen" values λ_is. Review what `Lambda / sum(Lambda)` would generate.

$$D = \lambda I$$

a matrix with the diagonal elements containing the λ_is.

We have $m = 1, 2, 3, ...8$ maximum components because that's the maximum number of dimensions we can rotate if we started with 8 features ...

This is a nice video explaining the importance of the vector maths. https://www.youtube.com/watch?v=mBcLRGuAFUk

Orthogonal Nature of W

Consider the matrix W.

We computed $T = X \cdot W$. https://en.wikipedia.org/wiki/Principal_component_analysis

```
dim(Xs);  # data
```

```
## [1] 55  8
```

```
dim(S);   # sample variance (correlation if Xs)
```

```
## [1] 8 8
```

```
dim(W);   # the eigenvectors associated with
```

```
## [1] 8 8
```

```
length(Lambda);
```

```
## [1] 8
```

```
n;
```

```
## [1] 55
```

```
m;
```

```
## [1] 8
```

```
dim(T);
```

```
## [1] 55  8
```

```
# what do you notice ??? ... orthogonality
zapsmall( W %*% transposeMatrix(W), digits=2);
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    0    0    0    0    0    0    0
## [2,]    0    1    0    0    0    0    0    0
## [3,]    0    0    1    0    0    0    0    0
## [4,]    0    0    0    1    0    0    0    0
## [5,]    0    0    0    0    1    0    0    0
## [6,]    0    0    0    0    0    1    0    0
## [7,]    0    0    0    0    0    0    1    0
## [8,]    0    0    0    0    0    0    0    1
```

```
zapsmall( transposeMatrix(W) %*% W, digits=2);
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    0    0    0    0    0    0    0
## [2,]    0    1    0    0    0    0    0    0
## [3,]    0    0    1    0    0    0    0    0
## [4,]    0    0    0    1    0    0    0    0
## [5,]    0    0    0    0    1    0    0    0
## [6,]    0    0    0    0    0    1    0    0
## [7,]    0    0    0    0    0    0    1    0
## [8,]    0    0    0    0    0    0    0    1
```

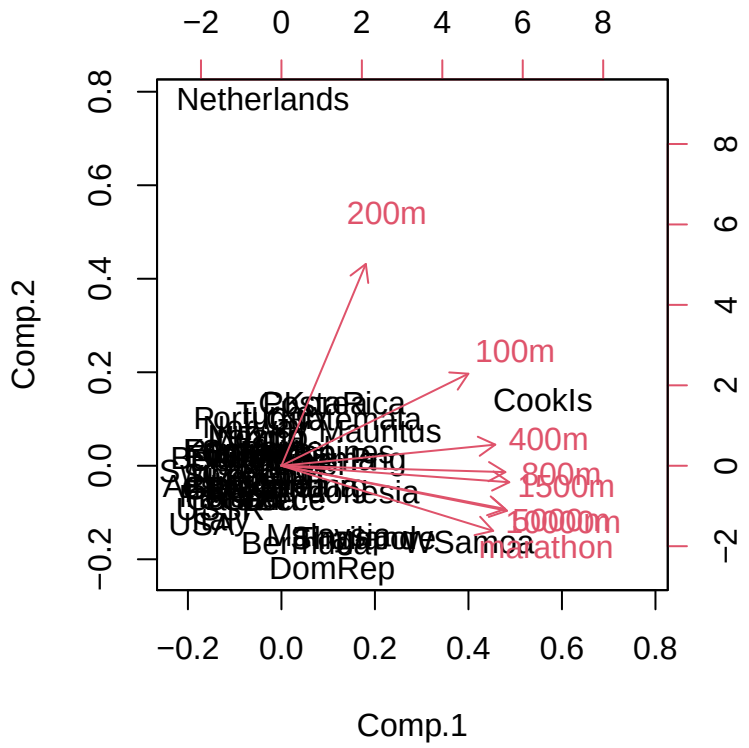
```
zeroIsh( W %*% transposeMatrix(W) );
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    0    0    0    0    0    0    0
## [2,]    0    1    0    0    0    0    0    0
## [3,]    0    0    1    0    0    0    0    0
## [4,]    0    0    0    1    0    0    0    0
## [5,]    0    0    0    0    1    0    0    0
## [6,]    0    0    0    0    0    1    0    0
## [7,]    0    0    0    0    0    0    1    0
## [8,]    0    0    0    0    0    0    0    1
```

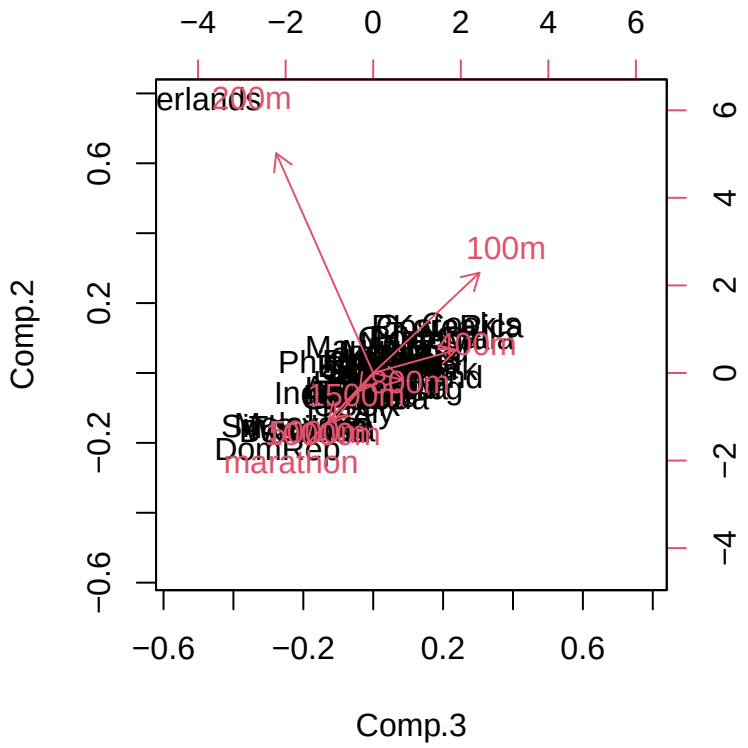
```
zeroIsh( transposeMatrix(W) %*% W );
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    0    0    0    0    0    0    0
## [2,]    0    1    0    0    0    0    0    0
## [3,]    0    0    1    0    0    0    0    0
## [4,]    0    0    0    1    0    0    0    0
## [5,]    0    0    0    0    1    0    0    0
## [6,]    0    0    0    0    0    1    0    0
## [7,]    0    0    0    0    0    0    1    0
## [8,]    0    0    0    0    0    0    0    1
```

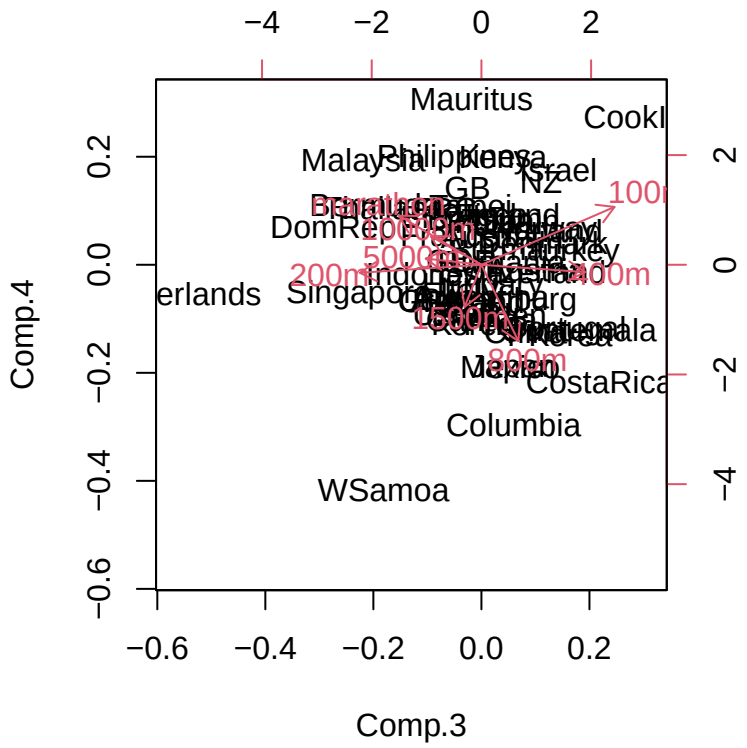
```
biplot(Xs.princomp);  # equivalent to # biplot(Xs.prcomp);
biplot(Xs.princomp, 1:2);
```



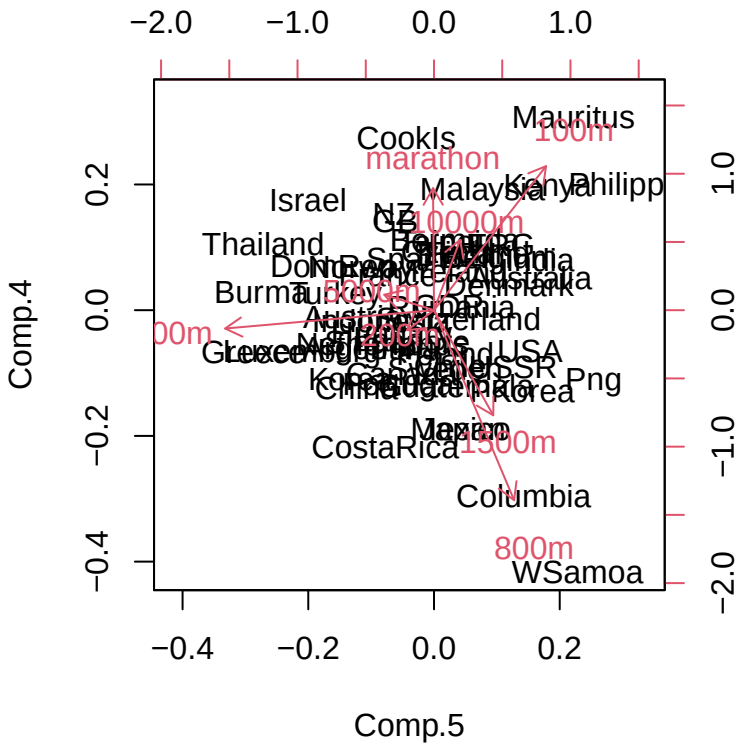

```
biplot(Xs.princomp, 3:2);  # Netherlands and something with 200 meters
```



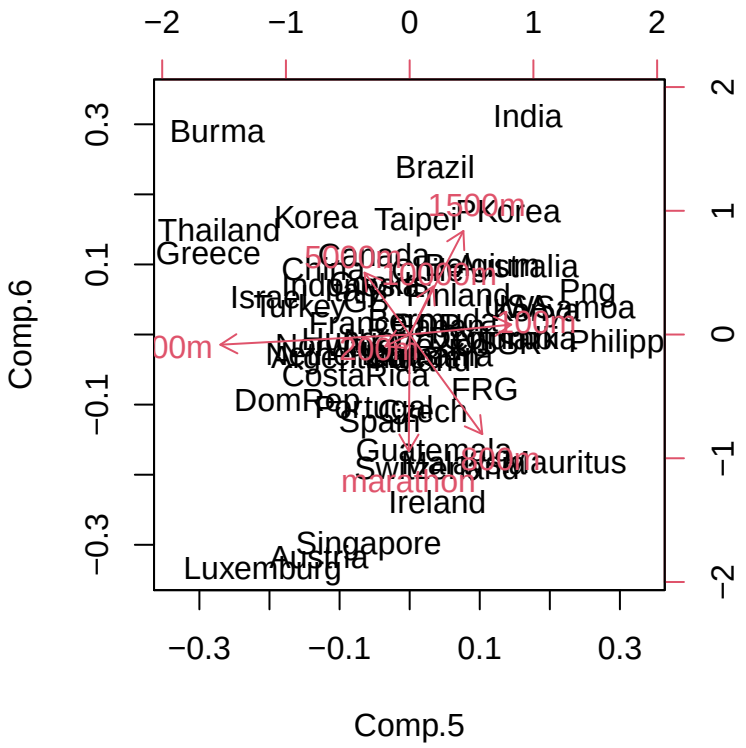

```
biplot(Xs.princomp, 3:4);
```



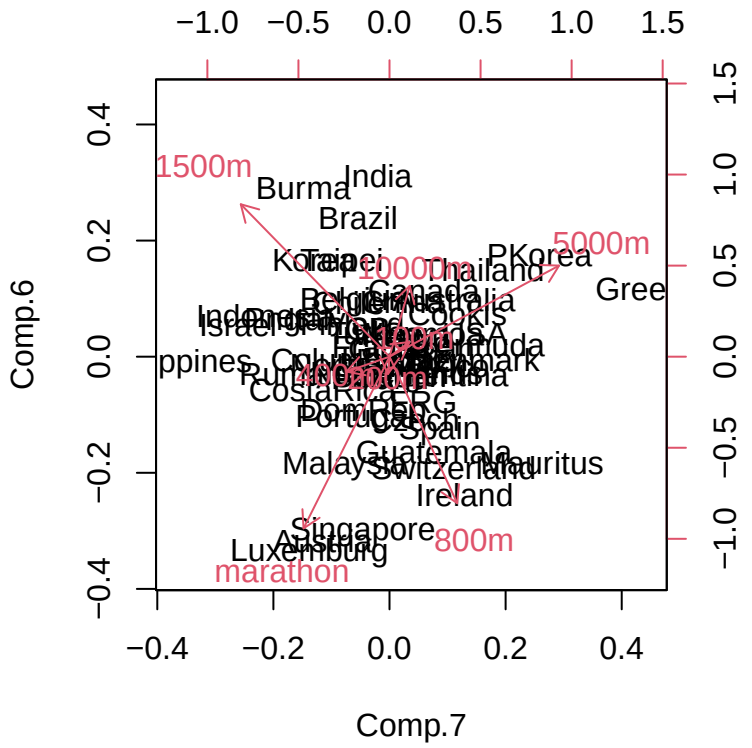

```
biplot(Xs.princomp, 5:4);  # Cook & marathon, Mauritius and 100m, W. Samoa and 800m
```



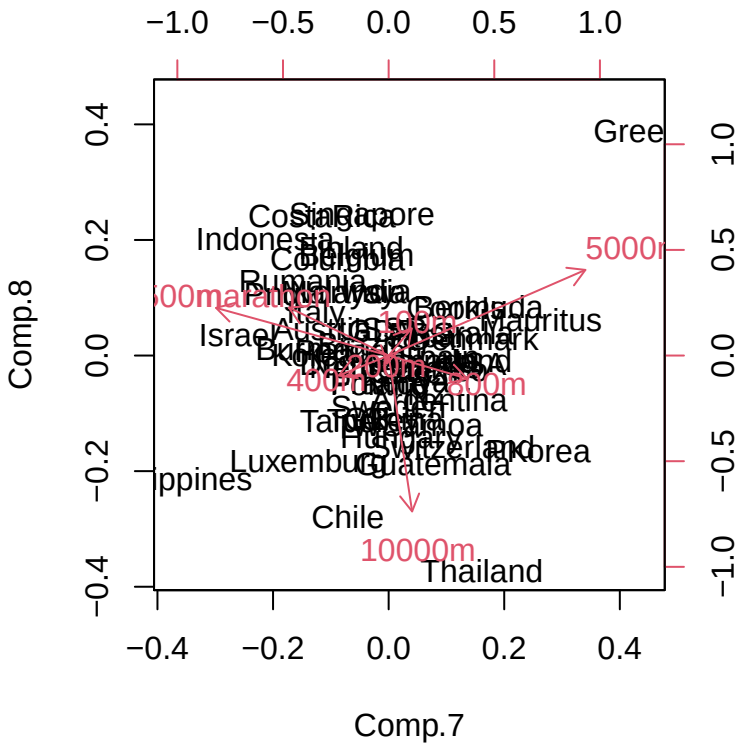

```
biplot(Xs.princomp, 5:6);
```




```
biplot(Xs.princomp, 7:6);
```




```
biplot(Xs.princomp, 7:8);
```




```
# notice the ordering
# the next will have the same component on the same dimension

# There are meaning in the higher dimensions, although it does not explain a lot of variance in all of

# Netherlands   200m
# Cook          marathon
# Mauritius     100m
# W. Samoa      800m
```

Think about what this is showing. It is mapping **both** the rows (countries) and columns (races) on the same component reduction map (showing any two components at a time).

Conclusion

https://en.wikipedia.org/wiki/200_metres#Men_(outdoor)

Who the outliers from these countries may be very difficult to discover because the Soviet era doesn't seem to have a lot of balanced records.

https://en.wikipedia.org/wiki/Men%27s_200_metres_world_record_progression

Maybe Muriaroa Ngaro (1980) was the runner associated with the marathon from the Cook Islands https://en.wikipedia.org/wiki/List_of_Cook_Islands_records_in_athletics#Men

In 1980, he ran the marathon in 2:51:26 ... The winning time was 2:11:03 run by German Waldemar Cierpinski (East Germany). https://en.wikipedia.org/wiki/Athletics_at_the_1980_Summer_Olympics_%E2%80%93_Men%27s_marathon

So maybe Muriaroa Ngaro was an extreme outlier because he was so slow.

```
X[c(12,34,36,38,51,55,53,54),];
```

```
##              100m  200m  400m 800m 1500m 5000m 10000m marathon
## CookIs      12.18 23.20 52.94 2.02  4.24 16.70  35.38   164.70
## Luxemburg   10.35 20.77 47.40 1.82  3.67 13.64  29.08   141.27
## Mauritus    11.19 22.45 47.70 1.88  3.83 15.06  31.77   152.23
## Netherlands 10.52 29.95 45.10 1.74  3.62 13.36  27.61   129.02
## Thailand    10.39 21.09 47.91 1.83  3.84 15.23  32.56   149.90
## WSamoa      10.82 21.86 49.00 2.02  4.24 16.28  34.71   161.83
## USA          9.93 19.75 43.86 1.73  3.53 13.20  27.43   128.22
## USSR        10.07 20.00 44.60 1.75  3.59 13.20  27.53   130.55
```

```
source_url( paste0(path.github, "humanVerseWSU/R/functions-stats.R") );  # updated findOutliersUsingIQR
```

```
## SHA-1 hash of file is a171cba2df689bfb36d7fa592a8be84f8d5eabdc
```

```
## so let's examine
Countries = rownames(X);
Events = colnames(X);

for(i in 1:8)
  {
  event = Events[i];
  print("############################");
  print(paste0( " Event :: ", event ) );  # print(paste0 ... should be a helper function ... use it all
  print("############################");

  outliers = findOutliersUsingIQR(X[,i]);
    very.slow = outliers$outer.upper;  # outer fence ...
    slow = setdiff(outliers$inner.upper, very.slow);  # "unique" inner fence ...

    very.fast = outliers$outer.lower;  # outer fence ...
    fast = setdiff(outliers$inner.lower,very.fast);  # inner fence ...


  if(length(very.slow) > 0)
    {
    print("-------------- very.slow --------------");
    print( Countries[ very.slow ] );
    print("---------------------------------------");
    }
  if(length(slow) > 0)
    {
    print("---------------- slow -----------------");
    print( Countries[ slow ] );
    print("---------------------------------------");
    }


  if(length(fast) > 0)
    {
    print("---------------- fast -----------------");
    print( Countries[ fast ] );
    print("---------------------------------------");
```

```
    }

  if(length(very.fast) > 0)
    {
    print("-------------- very.fast --------------");
    print( Countries[ very.fast ] );
    print("---------------------------------------");
    }



  }
```

```
## [1] "############################"
## [1] " Event :: 100m"
## [1] "############################"
## [1] "-------------- very.slow --------------"
## [1] "CookIs"
## [1] "---------------------------------------"
## [1] "---------------- slow -----------------"
## [1] "Mauritus"
## [1] "---------------------------------------"
## [1] "############################"
## [1] " Event :: 200m"
## [1] "############################"
## [1] "-------------- very.slow --------------"
## [1] "Netherlands"
## [1] "---------------------------------------"
## [1] "---------------- slow -----------------"
## [1] "CookIs"   "Mauritus"
## [1] "---------------------------------------"
## [1] "############################"
## [1] " Event :: 400m"
## [1] "############################"
## [1] "-------------- very.slow --------------"
## [1] "CookIs"
## [1] "---------------------------------------"
## [1] "############################"
## [1] " Event :: 800m"
## [1] "############################"
## [1] "-------------- very.slow --------------"
## [1] "CookIs" "WSamoa"
## [1] "---------------------------------------"
## [1] "############################"
## [1] " Event :: 1500m"
## [1] "############################"
## [1] "---------------- slow -----------------"
## [1] "CookIs" "WSamoa"
## [1] "---------------------------------------"
## [1] "############################"
## [1] " Event :: 5000m"
## [1] "############################"
## [1] "-------------- very.slow --------------"
## [1] "CookIs"
```

```
## [1] "----------------------------------------"
## [1] "---------------- slow -----------------"
## [1] "WSamoa"
## [1] "----------------------------------------"
## [1] "############################"
## [1] " Event :: 10000m"
## [1] "############################"
## [1] "---------------- slow -----------------"
## [1] "CookIs" "WSamoa"
## [1] "----------------------------------------"
## [1] "############################"
## [1] " Event :: marathon"
## [1] "############################"
## [1] "---------------- slow -----------------"
## [1] "CookIs"    "DomRep"    "Malaysia"  "Mauritus"  "Singapore" "WSamoa"
## [1] "----------------------------------------"
```

Another EDA analysis: hclust/pvclust

Xs.hclust.8

```
source_url( paste0(path.github, "humanVerseWSU/R/functions-EDA.R") );
```

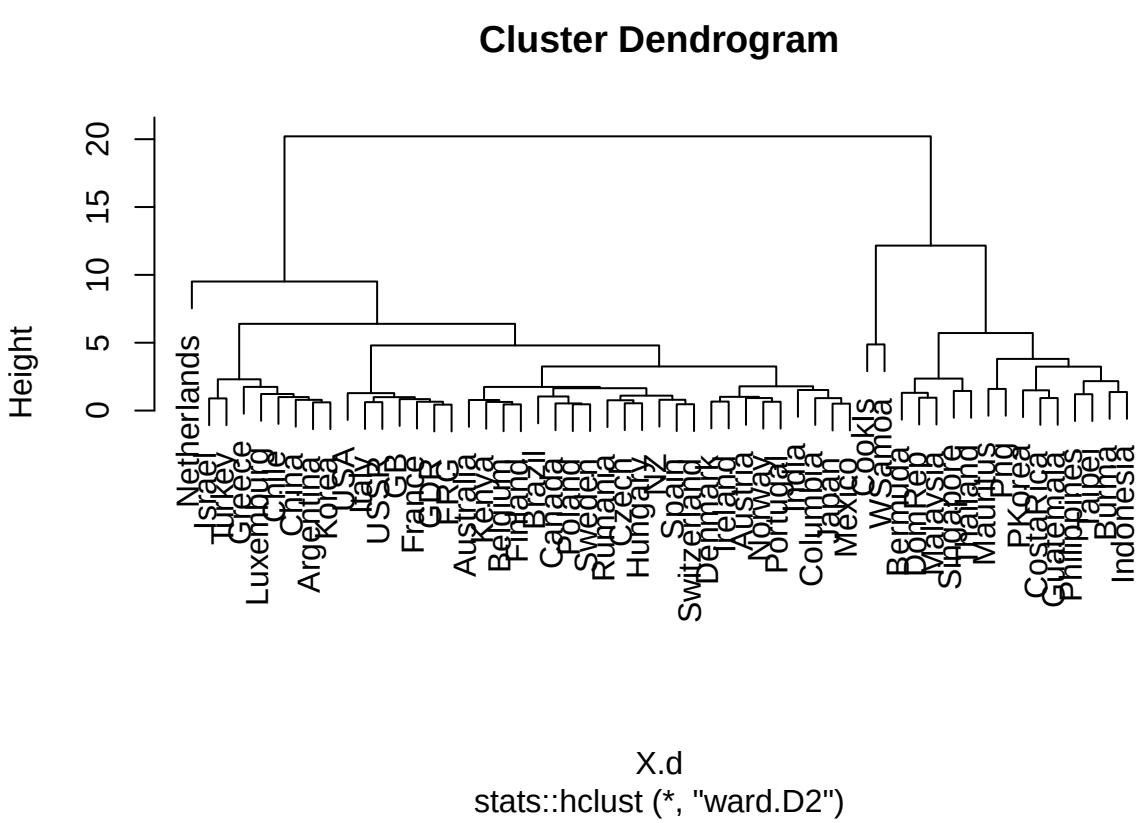
```
## SHA-1 hash of file is 62ba3333da32792e57c410e3f02a443a4c7f4985
```

```
# Xs;
dim(Xs);
```

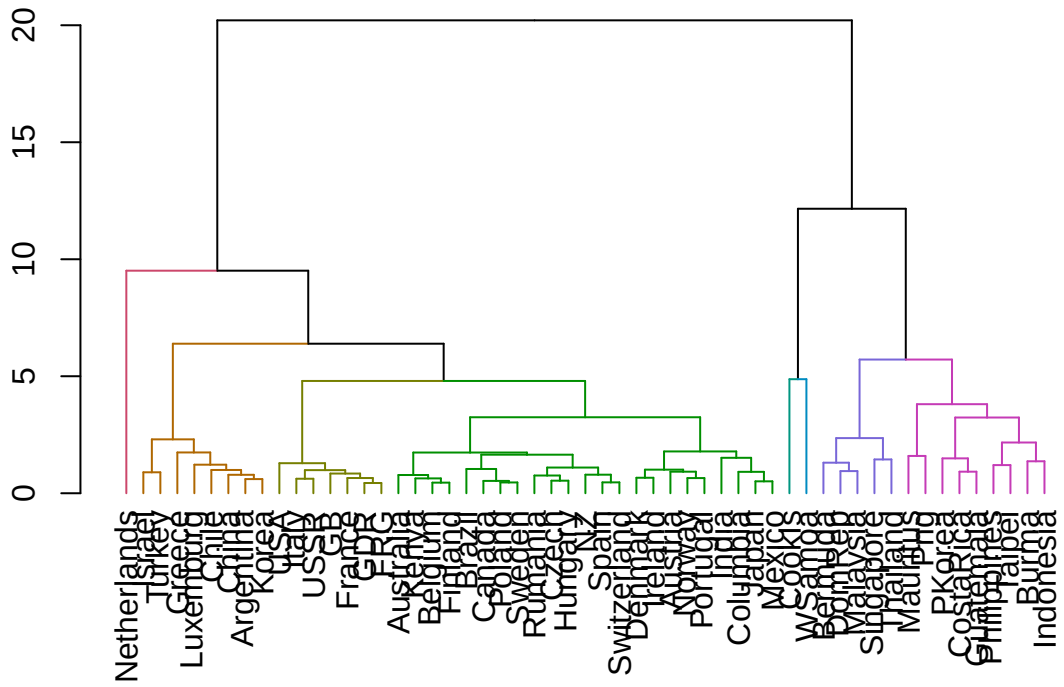
```
## [1] 55  8
```

```
Xs.hclust.8 = perform.hclust(Xs, n.groups = 8, pvclust.parallel = TRUE);
```

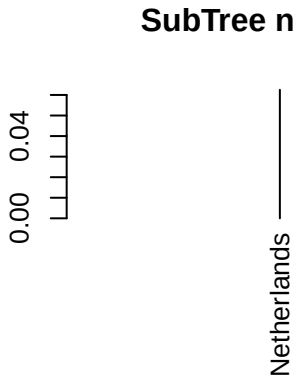
```
## Registered S3 method overwritten by 'dendextend':
##   method       from
##   text.pvclust pvclust
```

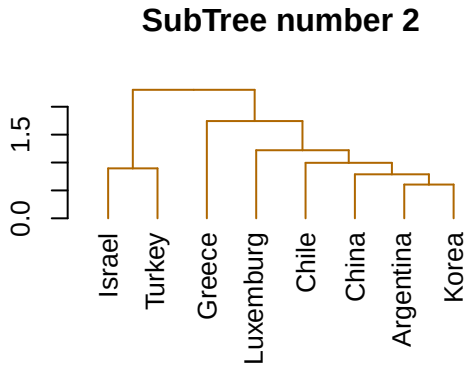


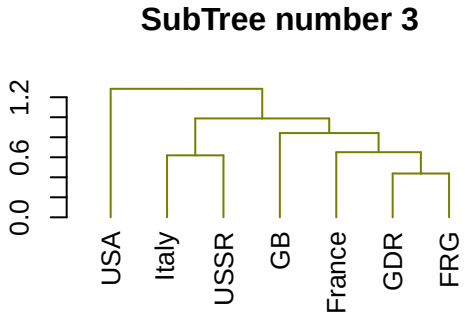
X.d
stats::hclust (*, "ward.D2")

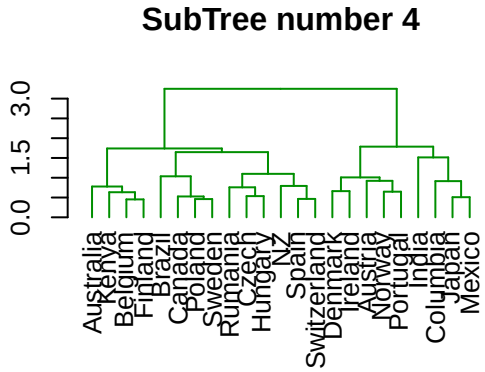


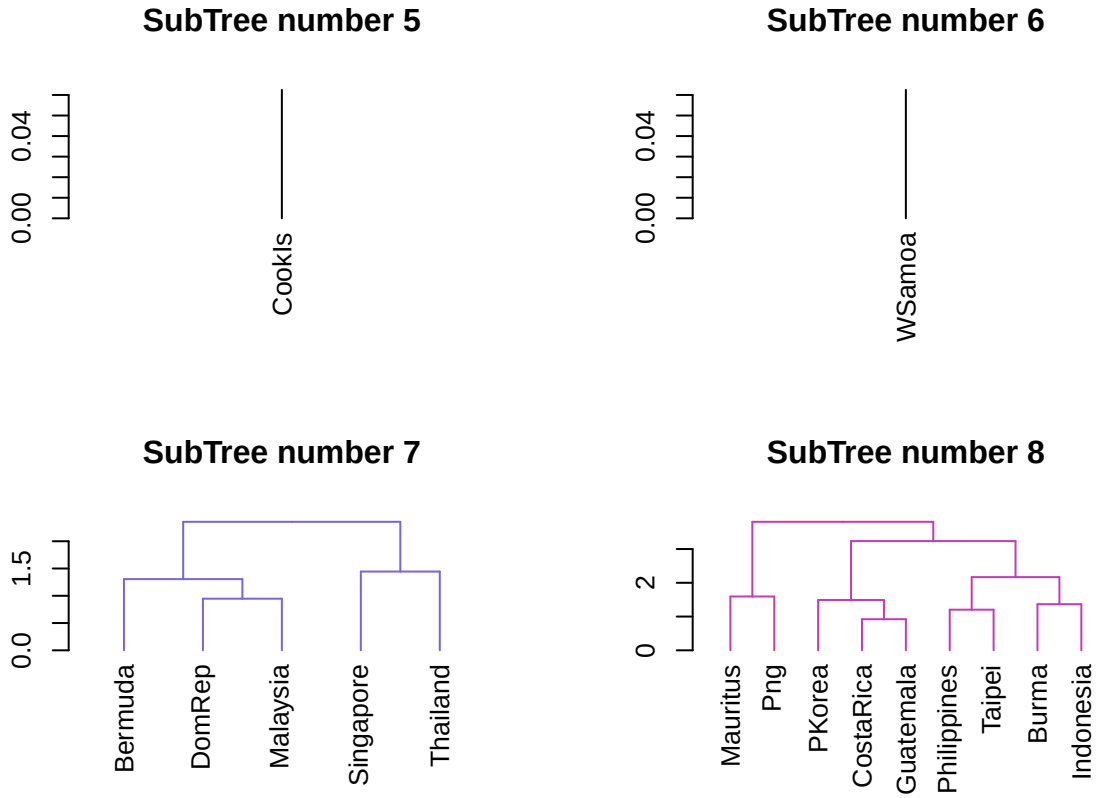

```
## [1] "Pruning 1 of 8"
## [1] "Pruning 2 of 8"
## [1] "Pruning 3 of 8"
## [1] "Pruning 4 of 8"
## [1] "Pruning 5 of 8"
## [1] "Pruning 6 of 8"
## [1] "Pruning 7 of 8"
## [1] "Pruning 8 of 8"
```


SubTree number 1
0.00
0.04
Netherlands


SubTree number 2
0.0
1.5
Israel
Turkey
Greece
Luxemburg
Chile
China
Argentina
Korea


SubTree number 3
0.0
0.6
1.2
USA
Italy
USSR
GB
France
GDR
FRG


SubTree number 4
0.0
1.5
3.0
Australia
Kenya
Belgium
Finland
Brazil
Canada
Poland
Sweden
Rumania
Czech
Hungary
NZ
Spain
Switzerland
Denmark
Ireland
Austria
Norway
Portugal
India
Columbia
Japan
Mexico



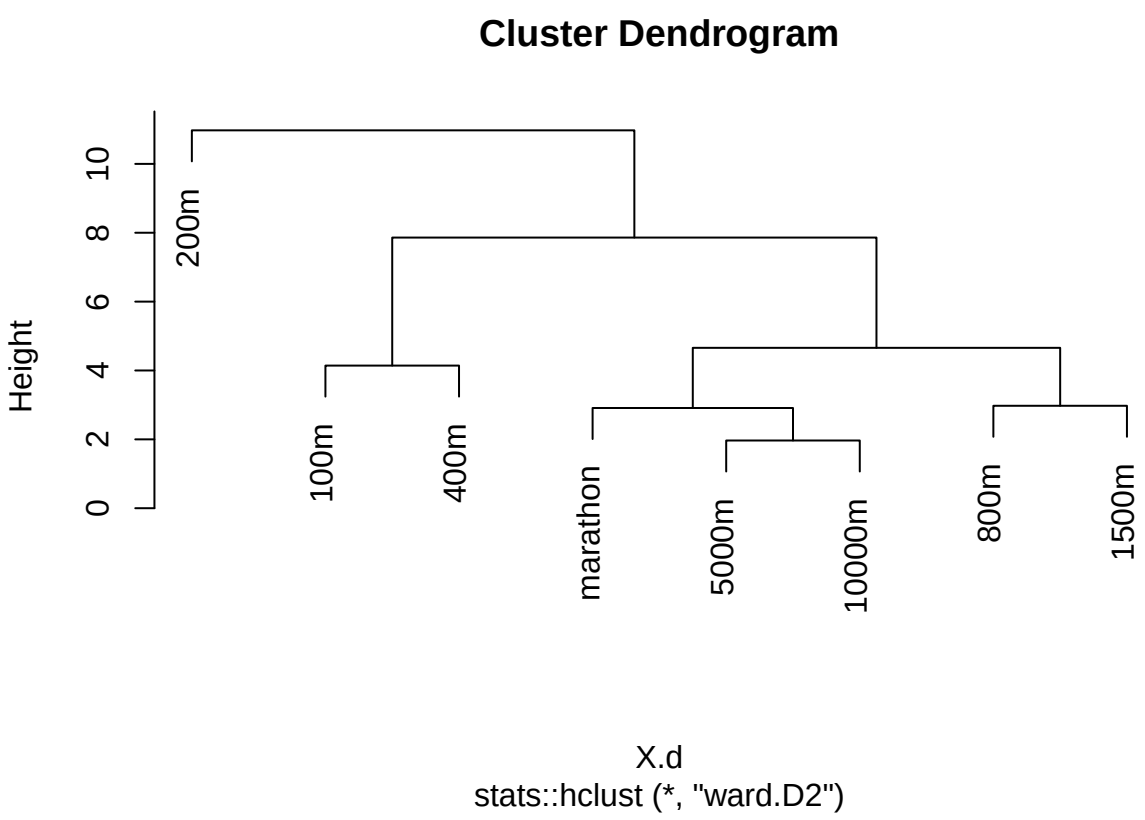

```
# if this doesn't work, turn off parallel ... # pvclust.parallel = FALSE

# if TRUE, (parallel::detectCores() - 1)
# you can pass a number like "ncores"
# pvclust.parallel = 4
```

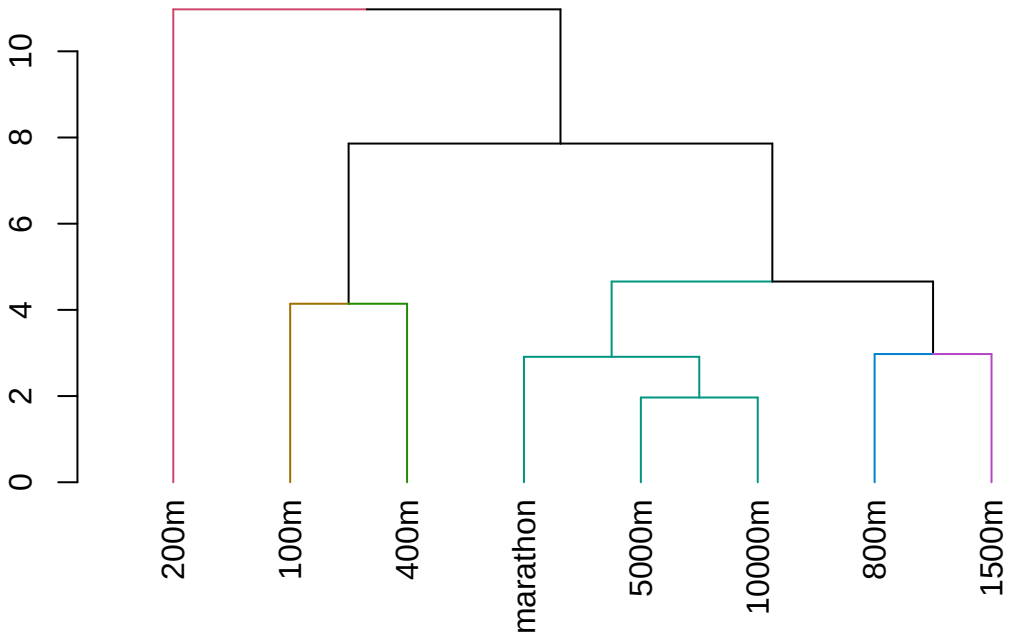
Xs.t.hclust.6

```
Xs.t = transposeMatrix(Xs);
#Xs.t;
dim(Xs.t);
```

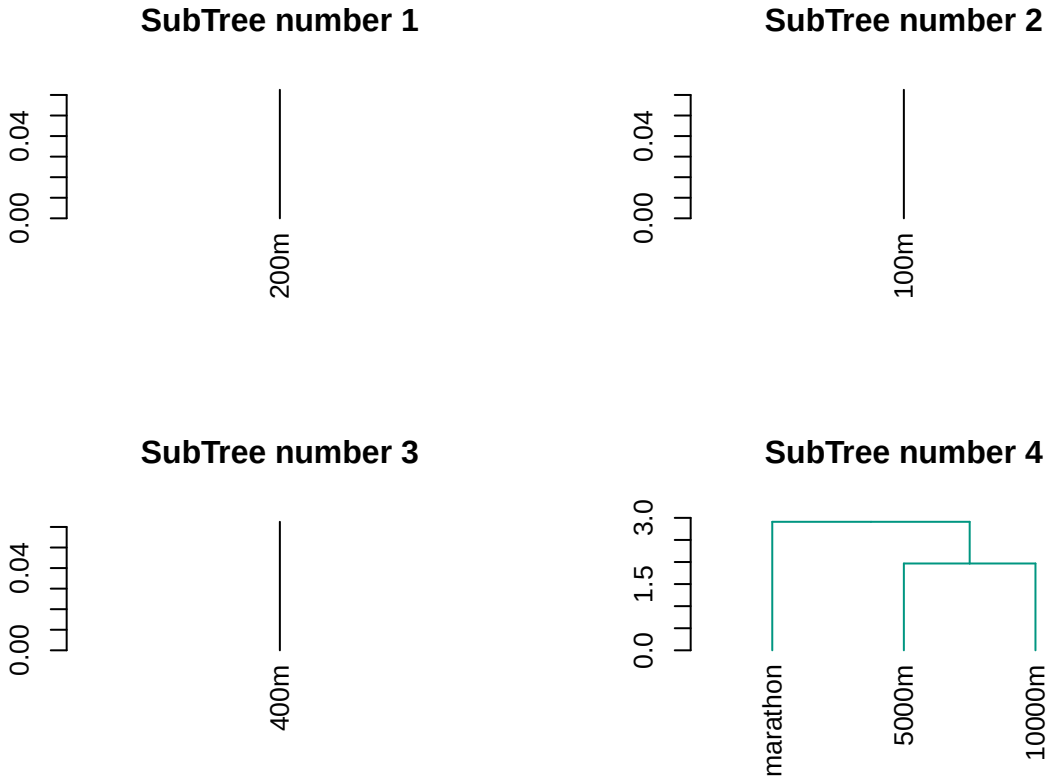
```
## [1]  8 55
```

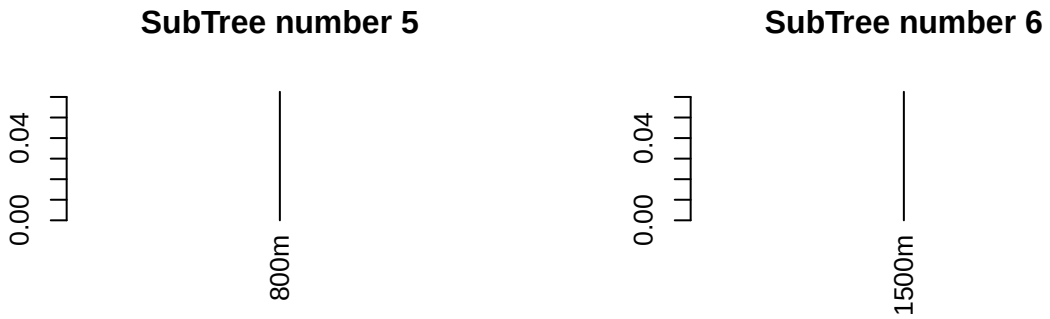
```
Xs.t.hclust.6 = perform.hclust(Xs.t, n.groups = 6, pvclust.parallel = TRUE);
```


Cluster Dendrogram
Height
0
2
4
6
8
10
200m
100m
400m
marathon
5000m
10000m
800m
1500m
X.d
stats::hclust (*, "ward.D2")



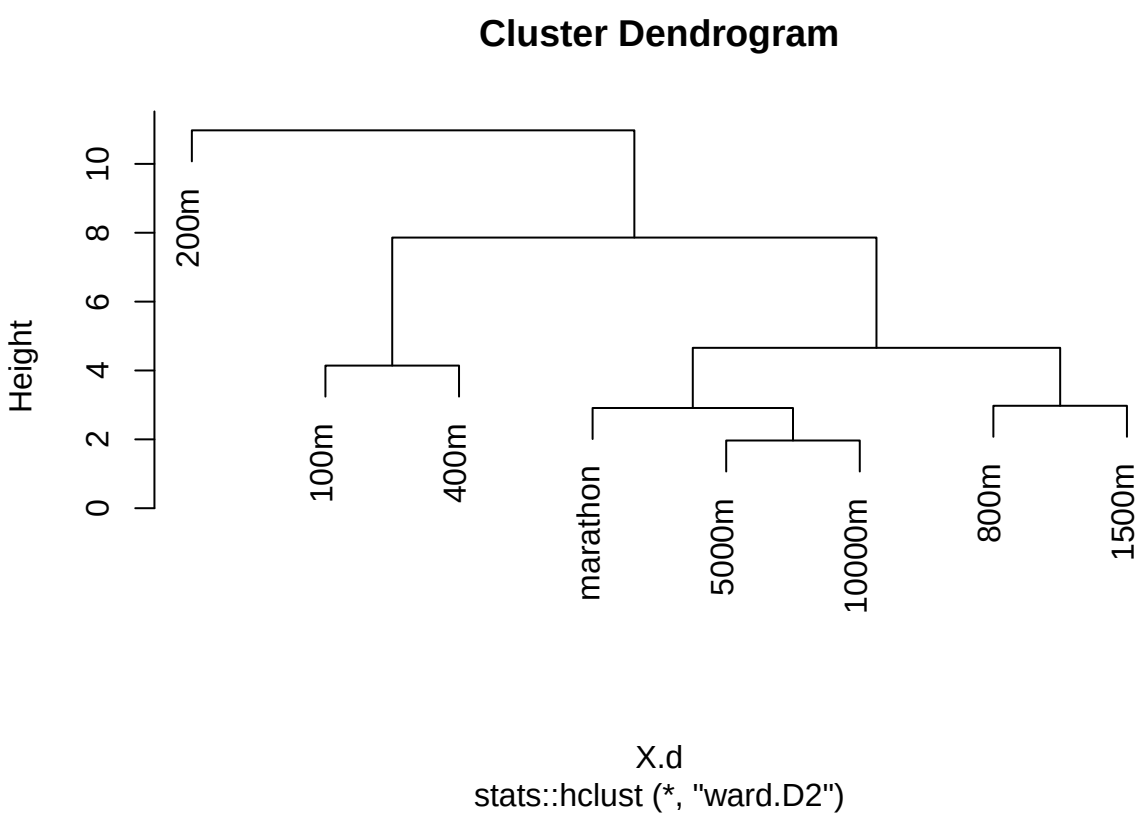

```
## [1] "Pruning 1 of 6"
## [1] "Pruning 2 of 6"
## [1] "Pruning 3 of 6"
## [1] "Pruning 4 of 6"
## [1] "Pruning 5 of 6"
## [1] "Pruning 6 of 6"
```

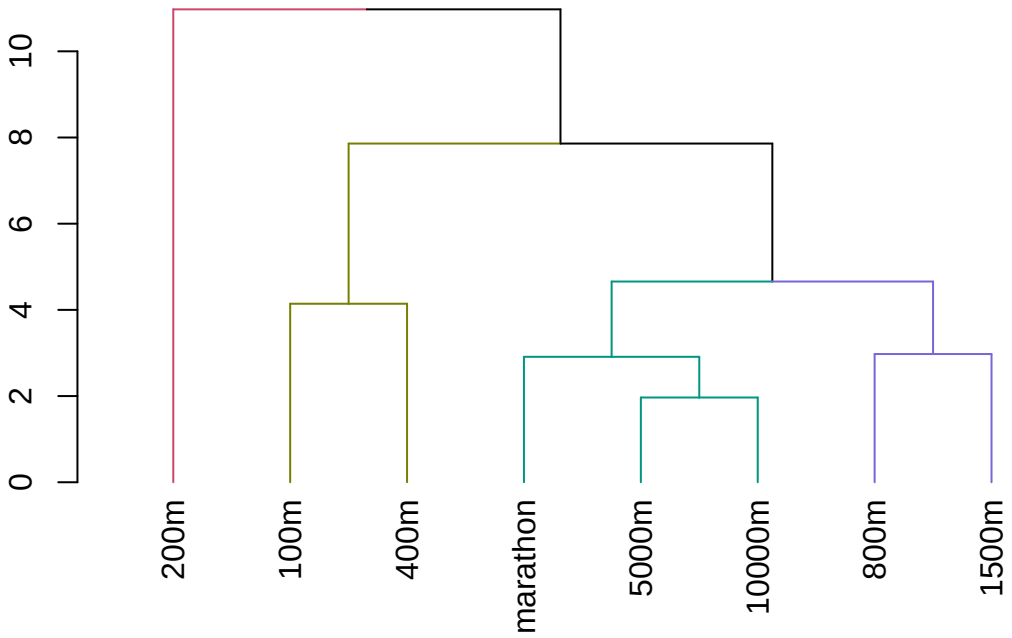

SubTree number 1
0.00
0.04
200m
SubTree number 2
0.00
0.04
100m
SubTree number 3
0.00
0.04
400m
SubTree number 4
0.0
1.5
3.0
marathon
5000m
10000m



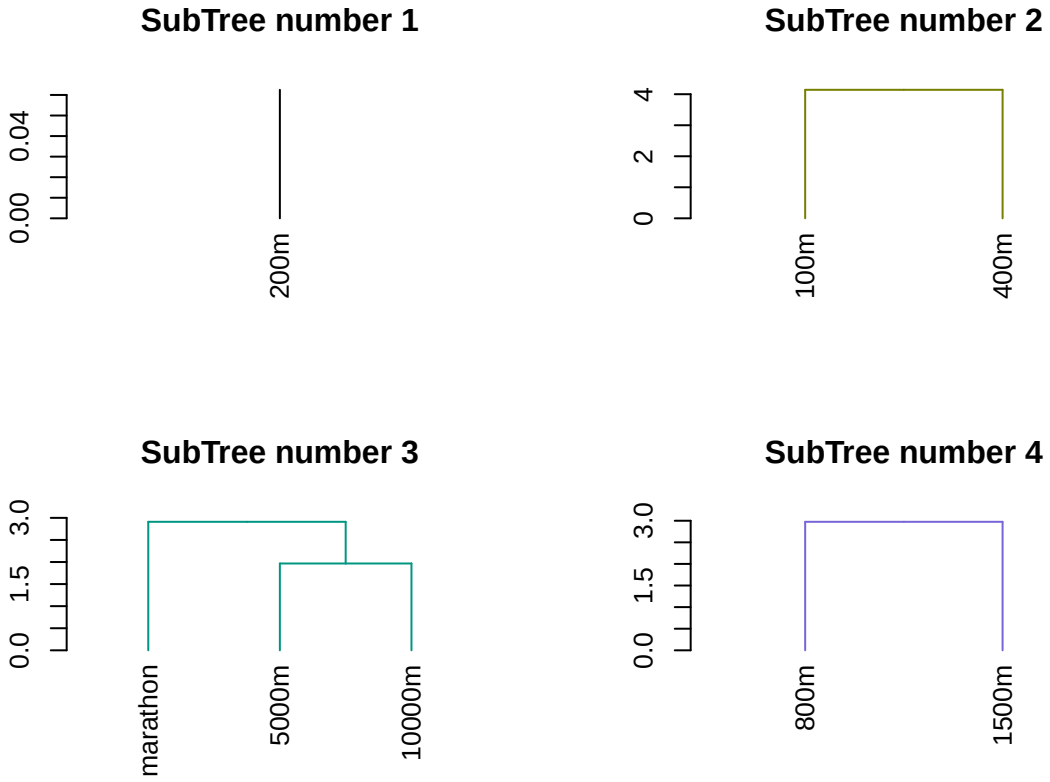
Xs.t.hclust.4

```
Xs.t.hclust.4 = perform.hclust(Xs.t, n.groups = 4, pvclust.parallel = TRUE);
```


Cluster Dendrogram
Height
0
2
4
6
8
10
200m
100m
400m
marathon
5000m
10000m
800m
1500m
X.d
stats::hclust (*, "ward.D2")




```
## [1] "Pruning 1 of 4"
## [1] "Pruning 2 of 4"
## [1] "Pruning 3 of 4"
## [1] "Pruning 4 of 4"
```


SubTree number 1
0.00
0.04
200m
SubTree number 2
0
2
4
100m
400m
SubTree number 3
0.0
1.5
3.0
marathon
5000m
10000m
SubTree number 4
0.0
1.5
3.0
800m
1500m