# Visualization of PCA: a dimension-reducing method

*Cheng-Yueh Lu*

*December 25, 2017*

**PCA** (Principal Component Analysis) is a statistical procedure that converts a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The more factors or variables describing the data, the more unlikely for us to see the panorama. Therefore, to decrease the factor/variables, so-called dimensions, allows us to straightforwardly obsereve the data. For more details on PCA, please see **Making sense of principal component analysis, eigenvectors & eigenvalues**

Here, I use the endogenous `data (iris)` to provide a graphical illustration on how PCA reconstructs data.

In this illustration, I call three packages.

```
library (scatterplot3d)  # construct data points in 3D
library (scales)  # change the animation transparency
library (pca3d)  # draw a 2D PCA plot
```

Now, I would like to use a 3-dimensional data to present how to reduce the dimensions into 2-dimensional ones. First, I choose the first three column (factors) to be the three variables.

```
data (iris)
iris.original <- iris[,1:3]
```

I apply the endogenous function `prcomp` to do PCA, and it will return several objects.

```
pca <- prcomp (iris.original)
str (pca)
```

```
## List of 5
##  $ sdev    : num [1:3] 1.921 0.491 0.244
##  $ rotation: num [1:3, 1:3] 0.39 -0.091 0.916 -0.639 -0.743 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "Sepal.Length" "Sepal.Width" "Petal.Length"
##   .. ..$ : chr [1:3] "PC1" "PC2" "PC3"
##  $ center  : Named num [1:3] 5.84 3.06 3.76
##   ..- attr(*, "names")= chr [1:3] "Sepal.Length" "Sepal.Width" "Petal.Length"
##  $ scale   : logi FALSE
##  $ x       : num [1:150, 1:3] -2.49 -2.52 -2.71 -2.56 -2.54 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:3] "PC1" "PC2" "PC3"
##  - attr(*, "class")= chr "prcomp"
```

In this case, `x` and `rotation` are useful. `x` returns the coordinates of the individuals (observations) on the principal components.

```
pca$x
```

`rotation` returns the matrix of variable loadings (columns are eigenvectors).

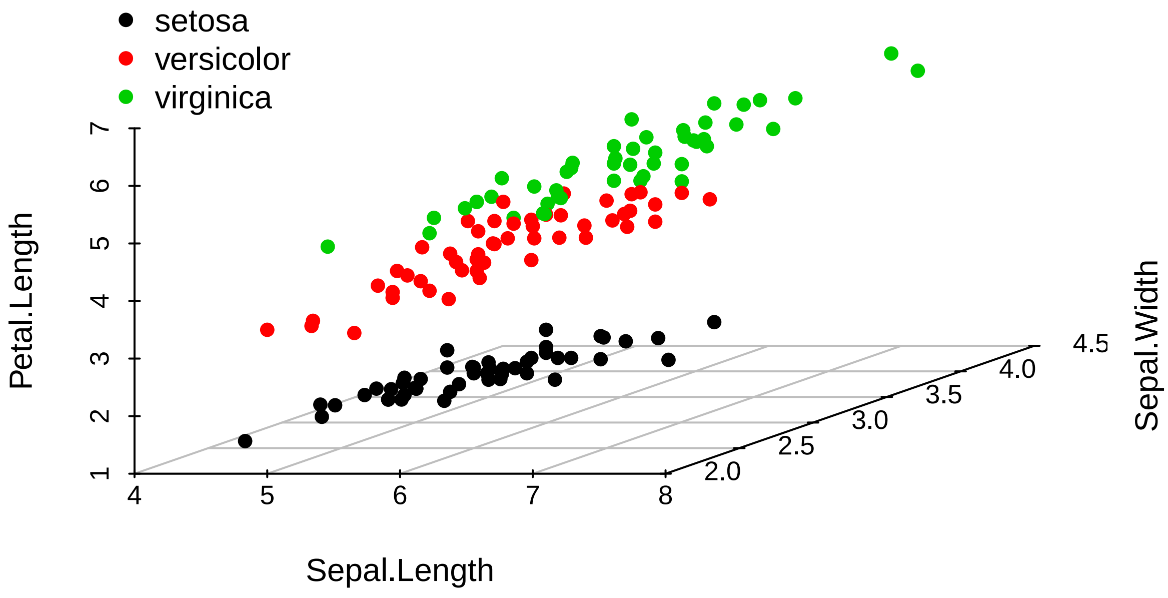
```
pca$rotation
```

```
##                      PC1        PC2        PC3
## Sepal.Length  0.38983343 -0.6392233 -0.6628903
## Sepal.Width  -0.09100801 -0.7430587  0.6630093
## Petal.Length  0.91637735  0.1981349  0.3478435
```

For more details, please see **Principal Component Methods in R: Practical Guide**

Let's see the original data points in 3D.

```
iris.color <- as.numeric (iris$Species)
iris.legend <- levels (iris$Species)
scatterplot3d (iris.original, pch = 16, color = iris.color, box = FALSE)
legend ("topleft", legend = iris.legend, col = 1:3, pch = 16, bty = "n")
```

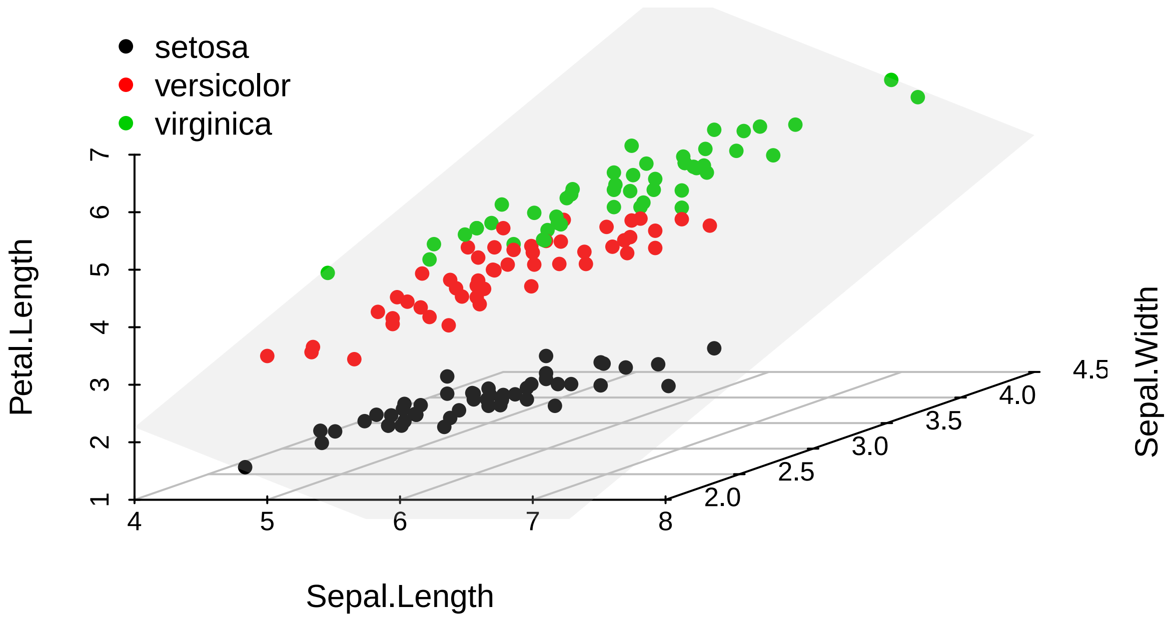


Pricipal components are not produced by a new factor, they are generated from different proportions of the existing factors. Thus, one can imagine it's just like to rotate the original data points to find an appropriate view on them. In this process, the reconstructed data should maintain most of the variations of the original ones, that is, the reconstructed ones still have the representitiveness. At the end of this rotation, original data points all project onto that 2D-plane making a 2D-plot which is easier to visualize.

To create that 2D-plane with PCA, `pca$rotation` is needed. More details can be seen here: **How to reverse PCA and reconstruct original variables from several principal components?**

```
mean <- colMeans (iris.original)

plane.coordinates <- cbind (c(0,0,0), pca$rotation[,1:2])
# imagine we have three points: (0,0,0), (PC1 composition), (PC2 composition)

actual.plane.coord <- t (plane.coordinates + mean)
# transpose matrix to shift those points with a mean value
```

```r
# make regression plane with lm
pca.plane <- lm (actual.plane.coord[,3] ~
                 actual.plane.coord[,1] + actual.plane.coord[,2])

iris.3D <- scatterplot3d (iris.original, pch = 16, color = iris.color, box = FALSE)
legend ("topleft", legend = iris.legend, col = 1:3, pch = 16, bty = "n")

iris.3D$plane3d (pca.plane, draw_lines = F, draw_polygon = T,
                 polygon_args = list (border = NA, col = alpha ("grey", 0.2)))
```

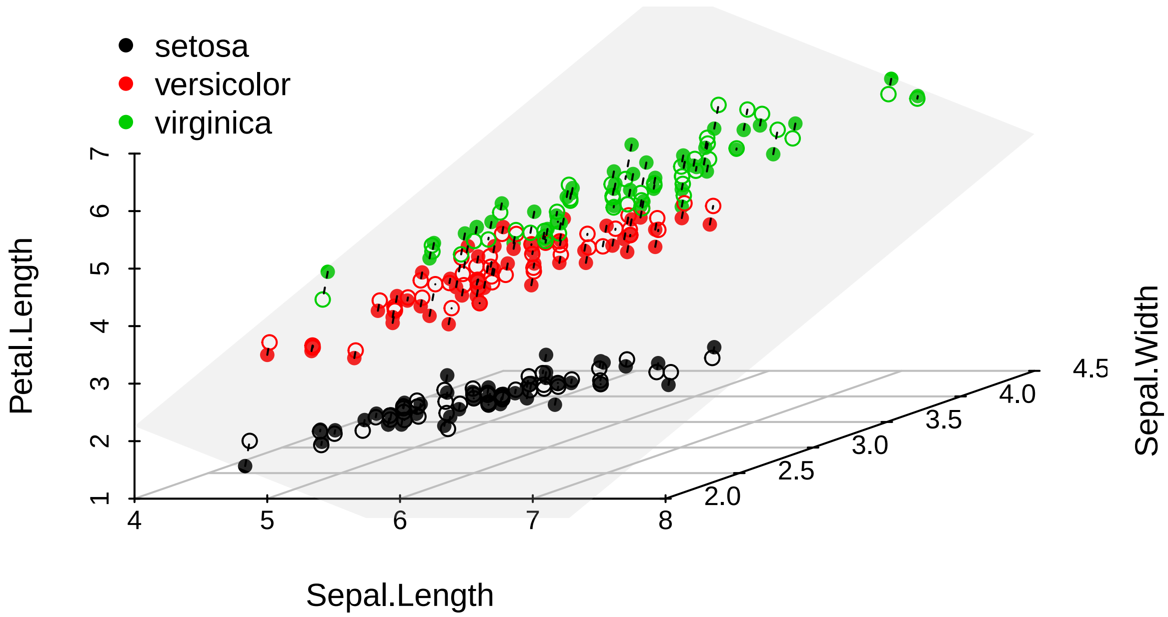


For more details on how I generate `plane.coordinates`, please see **Procrustes, PCA, and 3D coordinates**.

Now reconstruct the data on the 2D-plane and make the projection line from the original data points (solid circles) to the reconstructed ones (empty circles).

```r
pca.coordinates <- pca$x[,1:2] %*% t (pca$rotation[,1:2])
iris.2D <- scale (pca.coordinates, center = -mean, scale = F)

iris.3D <- scatterplot3d (iris.original, pch = 16, color = iris.color, box = FALSE)
legend ("topleft", legend = iris.legend, col = 1:3, pch = 16, bty = "n")

iris.3D$plane3d (pca.plane, draw_lines = F, draw_polygon = T,
                 polygon_args = list (border = NA, col = alpha ("grey", 0.2)))
iris.3D$points3d (iris.2D, pch = 21, col = iris.color)

# convert 3D coordinates into (x,y)
```

```
orig.convert <- iris.3D$xyz.convert (iris.original)
plane.convert <- iris.3D$xyz.convert (iris.2D)

# use converted coordinates to draw lines
segments (orig.convert$x, orig.convert$y, plane.convert$x, plane.convert$y,
          col = "black", lty = "dashed")
```



Details can also be seen in **How to reverse PCA and reconstruct original variables from several principal components?**

To visualize PCA more easily, I create a seies of .png file with the `for` loop fuction. First, `rename` function helps generate a series of filenames.

```
rename <- function(x)
{
  if (x < 10)
  {return (name <- paste ('000', i, 'plot.png', sep = ''))}
    if (x < 100 && i >= 10)
    {return (name <- paste ('00', i, 'plot.png', sep = ''))}
      if (x >= 100)
        {return (name <- paste ('0', i, 'plot.png', sep = ''))}
}
# start from 000i.plot.png to the end
```

Second, `for` loop function produces a series of .png files with `frames` angles.

```
frames <- 360  # it depends on how many plots you want to produce in a series
for (i in 1:frames)  # generate `frames` plots
{
  name <- rename (i)
  png (name, height = 600, width = 600, res = 240, pointsize = 6)

  iris.3D <- scatterplot3d (iris.original, angle = i, main = "Data reconstruction",
                           pch = 16, cex.symbols = 0.75,
                           color = alpha (iris.color, (frames-i)/(frames+5*i)),
                           box = FALSE)
                           # `frames` plots with 1 to `frames` angles
                           # subset alpha in color to make transparency
  legend ("topleft", legend = iris.legend, col = 1:3, pch = 16, bty = "n")

  iris.3D$plane3d (pca.plane, draw_lines = F, draw_polygon = T,
                   polygon_args = list (border = NA,
                   col = alpha ("grey", 0.3*sqrt (i/frames))))

  iris.3D$points3d (iris.2D, pch = 21, cex = 0.75,
                    col = alpha (iris.color, sqrt (i/frames)))

  orig.convert <- iris.3D$xyz.convert (iris.original)
  plane.convert <- iris.3D$xyz.convert (iris.2D)
  segments (orig.convert$x, orig.convert$y, plane.convert$x, plane.convert$y,
            col = alpha ("grey", (frames-i)/(frames+2*i)), lty = "dashed", lwd = 0.8)

  dev.off ()
}
```

Then, install **ImageMagick** outside of R to help convert all the .png files into a .gif file.

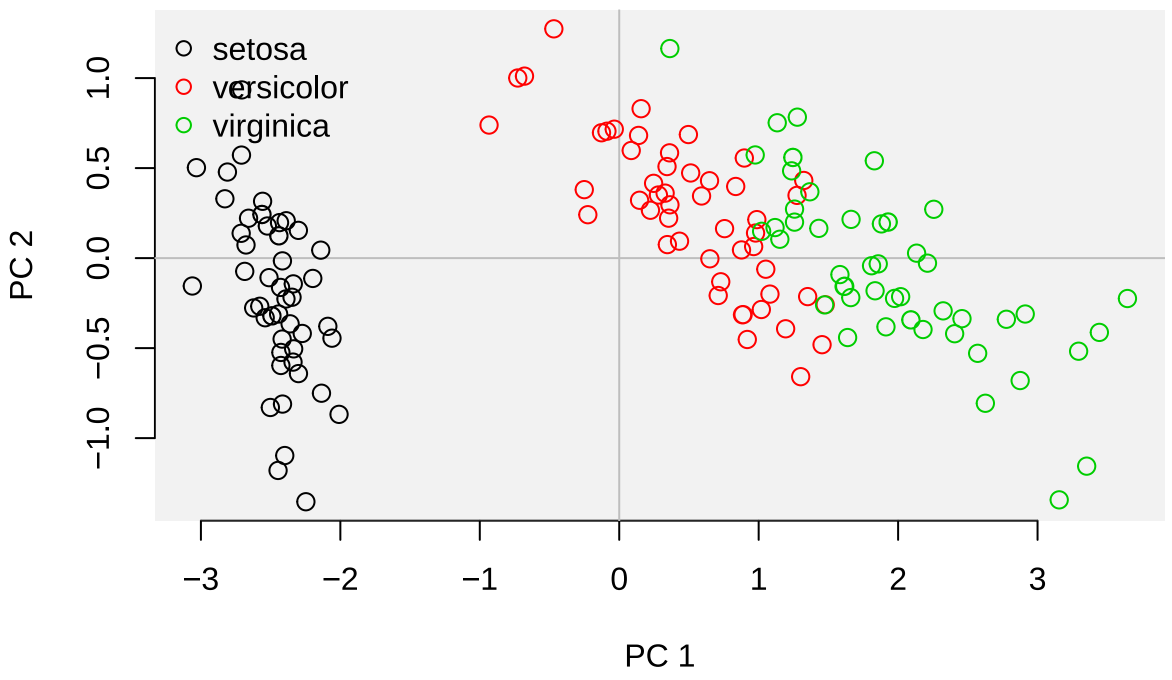
```
system ('"C:\\Program Files\\ImageMagick-7.0.7-Q16\\convert.exe"
        -delay 1x25 -loop 0 *.png pca.gif')
# the directory should be the address of ImageMagick folder
# -delay 1x25 equals 25 frames per second
# -loop 0 means looping forever

file.remove (list.files (pattern = ".png"))
```

The most important is that one should get the address of ImageMagick folder to execute this command. In addition, remove all the created .png files.

Check the .gif file **here**. For more details of ImageMigick, please see (or probably download) **ImageMagick**, or simply take a glance on how ImageMagick animate images: **Animate .gif images in R / ImageMagick**.

By the way, what will the traditional PCA plot be like? I call the `pca3d` package to draw it.

```
pca2d <- pca2d (pca, group = iris.legend, col = iris.color,
               bg = alpha ("grey", 0.2), shape = 21, radius = 1.2)
legend ("topleft", pca2d$groups, col = 1:3, pch = pca2d$pch, bty = "n")
```



Did you see the PC axes centering at 0? That's why we have to transform `pca$x` into the actual coordinates in the original 3D-plot with a mean to generate `iris.2D`.