PCA (Via the statistical approach)

Brandon Kozak and Margot Henry

18/11/2019

```
library(tidyverse)
library(BiocManager)
library(here)
library(lattice)
library(ggcorrplot)
library(GGally)
library(ggfortify)
library(factoextra)
library(ggrepel)
library(ade4)
library(pracma)
library(MASS)
```

1.0 How did we get to PCA?

1.0.1 SVD, eigenvalues, and eigenvectors

As we saw in the textbook, the basis for all things PCA was dependent on using a matrix decomposition known as Singular Value Decomposition (SVD). However, as much as the book tried, it never gave a clear cut answer as to why SVD was important, or even how we could interpret SVD in general. This only makes things worse once you realize that you are in a stats course but you have yet to see the statistical drive to PCA.

In this set of notes we aim to show PCA from a different point of view, and discuss why SVD might be used over this method in general.

1.0.2 The covariance Matrix

One thing that the book as yet to introduce is the covarince matrix, or covariance at all for that matter.

Covarince is a measurement of the relation ship between two variables. This is directly related to correlation, but is not standardized. This means that (similar to variance) it is hard to interpret what a value means without looking at the data itself. More over, the covarinace only gives us the direction of the relationship, where as the correlation gives us both the direction and the strength.

It can be shown that $Cor(x, y) = \frac{Cov(x,y)}{\sqrt{Var(x) \times Var(y)}}$

Simply put, the covariance matrix is a n-dimensional extension of variance.

That is, each ij^{th} entry of this matrix is the covarince of the i^{th} and j^{th} variable or covariate.

Note that the diagonal entries give us $Cov(x_i, x_i)$, one can show that this is equivalent to $Var(x_i)$

Furthermore $Cov(x, y) = Cov(y, x)$ this implies that the covarince matrix will always be symmetric.

Ex.) Let's take a look at the turtle data!

```r
turtles = read.table("../data/PaintedTurtles.txt", header = TRUE)

# To obtain the covariance matrix in R, we use cov()
trutles_cov = cov(turtles[,-1])

trutles_cov
```

```
##          length    width    height
## length 419.4960 253.9907 165.82979
## width  253.9907 160.6769 102.19149
## height 165.8298 102.1915  70.43972
```

```r
# Verify that the varainces match up.
var(turtles[,2])
```

```
## [1] 419.496
```

```r
var(turtles[,3])
```

```
## [1] 160.6769
```

```r
var(turtles[,4])
```

```
## [1] 70.43972
```

What do we see from this covarinace matrix exactly? Well we for sure know that there is some positive (direct) relationship between all three variables. However, it's hard to say how strong of a relationship since each variable has different ranges. For example, length and width have larger covariance because they take on larger values.

1.1 Thinking like a statistician

Now that we have the notion of a covarince matrix under our belt, lets try to build the workflow of PCA again.

1.1.1 The intuition behind PCA

Recall that PCA is a dimension reduction method. Our goal is to perform some linear transformations that takes our p-dimensional data and gives us k-dimensional data, where k is much smaller than p.

Beyond this, we also wish to keep as much information as possible after we reduce the dimension. Specifically, we wish that each new column (variable) we create to have maximal variance, and that each new column (variable) we create is uncorrelated with one another.

We will see soon that these new columns (variables) are called principle components (PC's)

1.1.2 The math that gets us there

First, since the covaraince matrix gives us information about the variance and covarince between all of our variables, you can probably guess that we will be using it to obtain our PC's.

That is, after some "fun" math, it can be shown that we can obtain the PC's by finding the eigenvectors of the covariance matrix of your data and then multiply them by our data.

Furthermore, the corresponding eigenvalue will tell us the amount of explained variability for that PC.

Now you might be thinking "How and where the heck did these eigenvalues and vectors even come from!?"

To be completely honest, this comes from that fun math I was talking about, in particular (if you are feeling up for it) look up Lagrange multipliers.

And much like SVD, it is hard to interpret eigenvalues and vectors in general, but in this context we get some nice results!

P.S If you would like a crash course on how to find eigenvalues and vectors of a matrix by hand (Don't recommend it), then check out the following link!

https://lpsa.swarthmore.edu/MtrxVibe/EigMat/MatrixEigen.html

1.1.3 A quick example

let's return back to the turtle data.

To obtain the eigenvalues and vectors, we perform eigen value decomposition (also known as spectral decomposition).

In R we have a handy function called eigen()

```
# eigen decomp of the covariance matrix of trutles

decomp_turtles = eigen(cov(turtles[,-1]))

# eigen values = explained variablilty for the corresponding PC
decomp_turtles$values
```

```
## [1] 641.577533   5.204385   3.830670
```

```
# eigen vectors = the loadings for the PC
decomp_turtles$vectors
```

```
##           [,1]       [,2]        [,3]
## [1,] 0.8068646  0.5865919 -0.06985282
## [2,] 0.4947448 -0.7356259 -0.46269006
## [3,] 0.3227958 -0.3387689  0.88376382
```

```
# the PC's, each column is a turtle, and each row within each column is the PC value.
decomp_turtles$vectors %*% t(turtles[,-1])
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 123.93227 129.72637 130.62014 132.37358 136.49481 149.71816 151.75735
## [2,] -28.68294 -28.41609 -31.73810 -29.82323 -31.16627 -29.95849 -30.31460
## [3,]  37.77673  38.37440  41.23192  40.10999  44.25869  52.72533  48.17397
```

```
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## [1,] 161.82310 163.58288 163.58288 163.42611 166.14317 164.46397 165.85742
## [2,] -30.62311 -32.82999 -32.82999 -29.47592 -30.42037 -27.90850 -28.14939
## [3,]  54.46567  53.44937  53.44937  51.79841  52.65023  56.09563  56.07965
##          [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
## [1,] 171.65786 177.97941 179.14626 182.30386 188.12136 189.50410 190.61180
## [2,] -32.00429 -33.09346 -30.44296 -28.92667 -37.06102 -37.14420 -35.11410
## [3,]  56.78295  61.23848  60.45531  62.63026  66.75204  63.42322  66.83667
##          [,22]     [,23]     [,24]     [,25]     [,26]     [,27]     [,28]
## [1,] 193.10859 199.18844 215.56503 115.86166 119.15459 121.94151 128.04279
## [2,] -37.28892 -39.29306 -40.53304 -25.54459 -27.06697 -27.54873 -29.86827
## [3,]  67.02692  64.19517  71.62954  37.65037  34.85056  34.81862  38.61258
##          [,29]     [,30]     [,31]     [,32]     [,33]     [,34]     [,35]
## [1,] 129.50610 128.03645 129.87679 131.49052 131.78064 139.78140 140.00168
## [2,] -29.64646 -25.74652 -27.64841 -26.65892 -24.96586 -28.56690 -27.33653
## [3,]  37.71284  38.50695  39.91973  40.56532  40.34312  41.35325  42.01481
##          [,36]     [,37]     [,38]     [,39]     [,40]     [,41]     [,42]
## [1,] 139.63536 143.38590 144.33247 144.91906 147.70597 146.23632 148.30328
## [2,] -25.37053 -28.71162 -27.29149 -28.02712 -28.50888 -24.60894 -29.40221
## [3,]  43.01515  44.95695  43.51222  43.17345  43.14151  43.93562  46.11559
##          [,43]     [,44]     [,45]     [,46]     [,47]     [,48]
## [1,] 150.42303 152.26775 155.64125 155.86153 158.21227 167.82238
## [2,] -29.45333 -27.39117 -28.60856 -27.37819 -26.35665 -32.93224
## [3,]  43.99333  48.61334  48.24262  48.90419  50.75634  49.20483
```

```
# proportion of exmapled variablility per PC
decomp_turtles$values / sum(decomp_turtles$values)
```

```
## [1] 0.986113003 0.007999208 0.005887789
```

```
# cummuliative proportion of exmapled variablility per PC
cumsum(decomp_turtles$values / sum(decomp_turtles$values))
```

```
## [1] 0.9861130 0.9941122 1.0000000
```

In other words, we see that the 1st PC explains over 98% of the variability in all three covariates.

We will talk about how we can view the loadings in a bit.

1.3 So many choices

As you may have noticed, the work flow for PCA contains alot of choices that you (the analyst) will have to make. In this section we will give some guidelines on how to make these choices

1.3.1 To scale or not to scale.

As we discussed before, the covaraince of two variables is hard to interpret, and it typically not comparable. Because of this, if our data contains lots of measures of different units or ranges, it may be best to scale our data first. The easiest way to do this, is to find the eigenvalues and vectors on the correlation matrix, rather than the covariance matrix.

For example, if we go back to the turtle data.

```r
# eigen decomp of the correlation matrix of trutles
decomp_turtles = eigen(cor(turtles[,-1]))

# eigen values = explained variablilty for the corresponding scaled PC
decomp_turtles$values
```

```
## [1] 2.93573765 0.04284387 0.02141848
```

```r
# eigen vectors = the loadings for the scaled PC
decomp_turtles$vectors
```

```
##           [,1]       [,2]        [,3]
## [1,] 0.5787981 -0.3250273  0.74789704
## [2,] 0.5779840 -0.4834699 -0.65741263
## [3,] 0.5752628  0.8127817 -0.09197088
```

```r
# the scaled PC's, each column is a turtle, and each row within each column is the scaled PC value.
decomp_turtles$vectors %*% t(scale(turtles[,-1]))
```

```
##            [,1]      [,2]        [,3]       [,4]        [,5]        [,6]
## [1,] -1.1265688 -1.062196 -0.75703376 -0.8787376 -0.46053753  0.3671965
## [2,]  0.4503009  0.476976  0.08737309  0.3004728  0.02374857 -0.2037215
## [3,] -1.5839883 -1.251193 -1.16678451 -1.0886944 -0.89193910 -0.3079912
##              [,7]       [,8]       [,9]      [,10]       [,11]      [,12]
## [1,] -0.066173399  0.5594116  0.4824871  0.4824871  0.29469545  0.3890427
## [2,] -0.004823908 -0.2668425 -0.3812657 -0.3812657 -0.04177327 -0.1399463
## [3,] -0.071796473  0.4107625  0.6031242  0.6031242  0.53584463  0.7093011
##           [,13]      [,14]      [,15]      [,16]      [,17]      [,18]
## [1,]  0.6980907  0.7007087  0.8098607  1.2589381  1.1628757  1.3650248
## [2,] -0.1158228 -0.1257441 -0.4265919 -0.6850179 -0.4337771 -0.3992286
## [3,]  0.4589892  0.5511965  0.9982637  1.3153132  1.3292828  1.4306717
##          [,19]     [,20]     [,21]     [,22]     [,23]     [,24]
## [1,]  1.840191  1.521574  1.835858  1.876304  1.629011  2.382438
## [2,] -1.196229 -1.037521 -1.033240 -1.197774 -1.185301 -1.537115
## [3,]  1.923102  2.084218  2.018321  2.227811  2.718711  3.587228
##           [,25]     [,26]      [,27]      [,28]      [,29]      [,30]
## [1,] -1.1774868 -1.430016 -1.4247802 -1.0296037 -1.1160971 -1.0743831
## [2,]  0.6545201  0.686836  0.6669932  0.3422065  0.4106153  0.6697294
## [3,] -2.1623079 -1.855822 -1.6714077 -1.3183245 -1.2151588 -1.4325960
##           [,31]      [,32]      [,33]      [,34]      [,35]      [,36]
## [1,] -0.9191839 -0.8626651 -0.8978755 -0.7578462 -0.7039453 -0.6244029
## [2,]  0.4650065  0.5214459  0.6661368  0.3835873  0.4499480  0.5544498
## [3,] -1.2981846 -1.2420110 -1.2670865 -0.6997251 -0.7357588 -0.8359131
##           [,37]      [,38]      [,39]      [,40]      [,41]      [,42]
## [1,] -0.4031159 -0.5530792 -0.5787207 -0.5734848 -0.5317708 -0.2778914
## [2,]  0.2233343  0.4082144  0.3700733  0.3502306  0.6093447  0.1434596
## [3,] -0.5561321 -0.5061288 -0.4420082 -0.2575935 -0.4750307 -0.2623815
##            [,43]       [,44]       [,45]       [,46]      [,47]       [,48]
## [1,] -0.47913763 -0.04748294 -0.06788857 -0.01398766 0.1599019  0.07999461
## [2,]  0.25205756  0.20622775  0.14824390  0.21460463 0.2209334 -0.16406976
## [3,] -0.08413705 -0.13290571  0.11562956  0.07959581 0.1528980  0.95961301
```

```r
# proportion of explained variablility per PC
decomp_turtles$values / sum(decomp_turtles$values)
```

```
## [1] 0.978579218 0.014281289 0.007139494
```

```r
# cummuliative proportion of exmapled variablility per PC
cumsum(decomp_turtles$values / sum(decomp_turtles$values))
```

```
## [1] 0.9785792 0.9928605 1.0000000
```

However, we see that scaling may have not been to much of an issue here, as we get close to the same results!

1.3.2 Picking the number of PC's

There are generally two strategies for picking the number of PC's

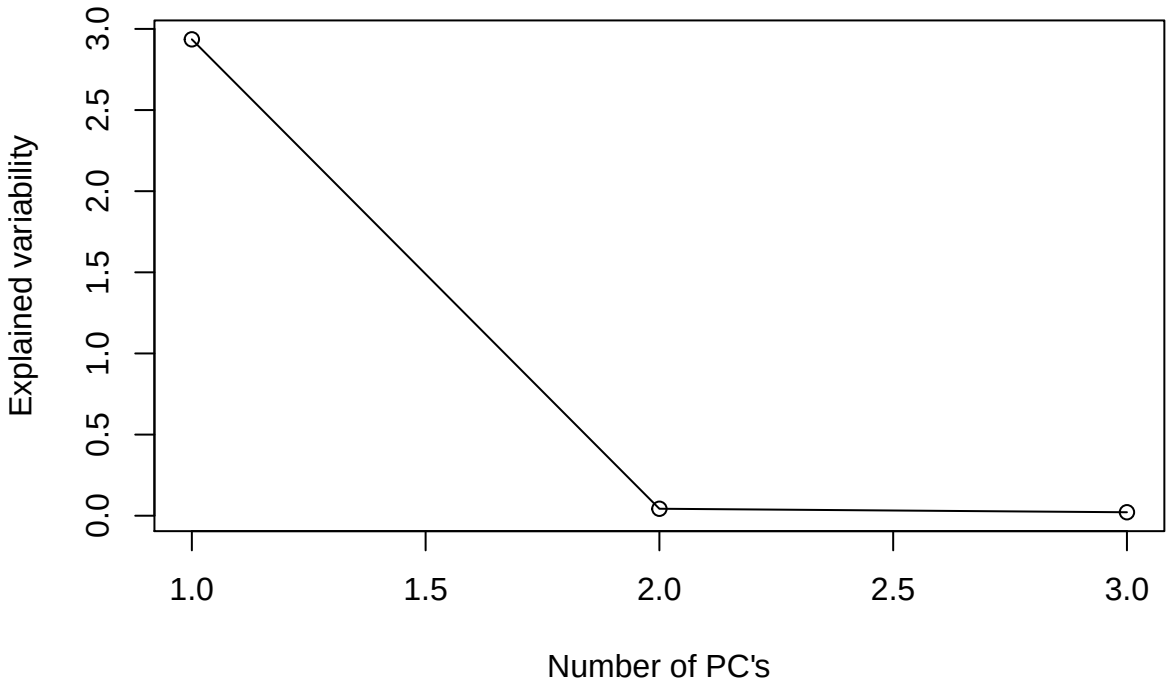
The first is more concrete. Pick some constant threshold c, the number of PC's we keep is equal to p where p is the PC who's cumulative explained variance is equal to or greater than c

For example, say we pick $c = .95$, then in the turtle example, one PC would be good enough.

The second involves a plot of the number of PC's v.s. the explained variability. This plot is known as a *scree plot*. With this method we look at the scree plot and pick a point before a sharp drop occurs. Of course, this may not always be obvious.

For the turtle data we would see this

```r
plot(1:3,decomp_turtles$values, type="l", xlab="Number of PC's", ylab="Explained variability")
points(decomp_turtles$values)
```



and again, pick 1 PC.

1.4 Visualization and interpretations

Now that we have a good understanding of the work flow of PCA, let take a look on how to visualize the projections and what they tell us.

1.4.1 Visualization of the projection

Throughout this section I will be using the function prcomp() to perform PCA. Note that prcomp() uses svd, but we can still interpret the PC's and loadings as if it was done using eigenvalues and vectors.

Let's start off with the turtle data again.

Even though we said one PC would be good enough, we mostly like to view a 2d projection, thus we normally take the first 2 PC's

```
# perform PCA, we achive scaleing by setting scale=T
pca_turtles = prcomp(turtles[,-1],scale = T)

# explained variance
pca_turtles$sdev
```

```
## [1] 1.7133994 0.2069876 0.1463505
```

```r
# loadings
pca_turtles$rotation
```

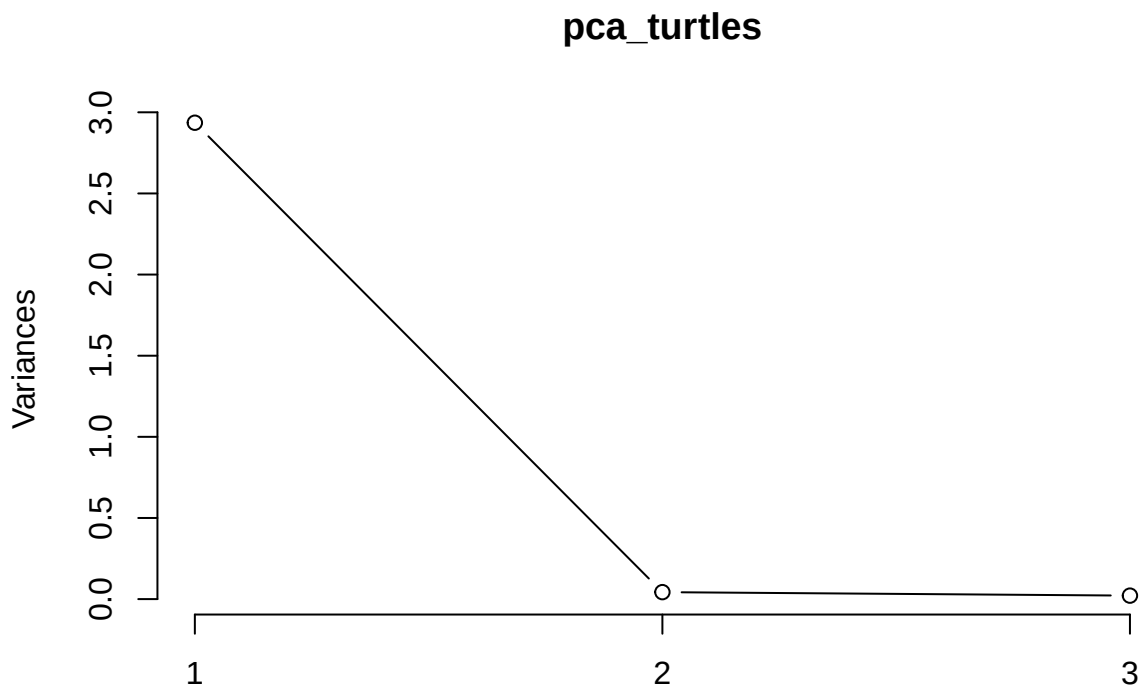
```
##               PC1        PC2         PC3
## length 0.5787981 -0.3250273 -0.74789704
## width  0.5779840 -0.4834699  0.65741263
## height 0.5752628  0.8127817  0.09197088
```

```r
# the PC's
pca_turtles$x
```

```
##                  PC1          PC2          PC3
##  [1,] -1.9836684602  0.167152488  0.134411527
##  [2,] -1.7055794726 -0.026616857  0.107424095
##  [3,] -1.3402164350  0.284470249  0.254984022
##  [4,] -1.4207818191  0.059047170  0.160036396
##  [5,] -0.9423811150  0.306657345  0.161534105
##  [6,]  0.0468925757  0.512977683 -0.076480320
##  [7,] -0.0904839545  0.011185276  0.035276987
##  [8,]  0.7172099610  0.184140239 -0.067633478
##  [9,]  0.8540018654  0.069717059  0.087956862
## [10,]  0.8540018654  0.069717059  0.087956862
## [11,]  0.5854403531 -0.160396974 -0.085160361
## [12,]  0.8016958980 -0.171575254 -0.043506318
## [13,]  0.7846503260  0.158804367 -0.265559143
## [14,]  0.8585070441  0.104794074 -0.250211250
## [15,]  1.3539533202  0.022024628 -0.026660717
## [16,]  1.9344701590  0.199755276 -0.046330670
## [17,]  1.8083074735  0.012473257 -0.193141742
## [18,]  1.9898872486  0.045838632 -0.328245701
## [19,]  2.8909792543  0.386867832  0.090338572
## [20,]  2.7765475313  0.020058792  0.161190690
## [21,]  2.9072153955  0.242417826 -0.030166350
## [22,]  3.1408088253  0.208967720  0.099866694
## [23,]  3.3620866667 -0.261170953  0.279584195
## [24,]  4.5620089799 -0.223284088  0.212508003
## [25,] -2.5126887623  0.416643767 -0.057013087
## [26,] -2.4391243571  0.054525680  0.092008629
## [27,] -2.2914109209 -0.053494906  0.122704414
## [28,] -1.6935561972  0.101963915  0.191413462
## [29,] -1.6882415877 -0.048888684  0.195803096
## [30,] -1.9109134857 -0.009035984 -0.059124503
## [31,] -1.6543752488  0.092497277  0.030003352
## [32,] -1.5978564155  0.060758811 -0.043027757
## [33,] -1.6837364090 -0.013811669 -0.142365016
## [34,] -1.0861739982 -0.166460641  0.060017856
## [35,] -1.1035118831 -0.144188814 -0.028361145
## [36,] -1.1664470694 -0.083775927 -0.168603593
## [37,] -0.7219127043  0.022448287 -0.001306140
## [38,] -0.8307375049 -0.187105560 -0.059738211
## [39,] -0.7851402035 -0.225246620 -0.007874765
## [40,] -0.6374267672 -0.333267206  0.022821020
## [41,] -0.8600986652 -0.293414505 -0.232106579
```

```
## [42,] -0.4035410247 -0.058609519  0.019180241
## [43,] -0.4211712224 -0.344445485  0.064475063
## [44,] -0.1937018329 -0.041113377 -0.152439273
## [45,] -0.0003910952 -0.187275023 -0.069880042
## [46,] -0.0177289800 -0.165003196 -0.158259043
## [47,]  0.1355913785 -0.115768588 -0.256847448
## [48,]  0.8187414700 -0.501954874  0.178546507
```

```r
# to get a screeplot I like to use screeplot()
screeplot(pca_turtles, type="lines")
```

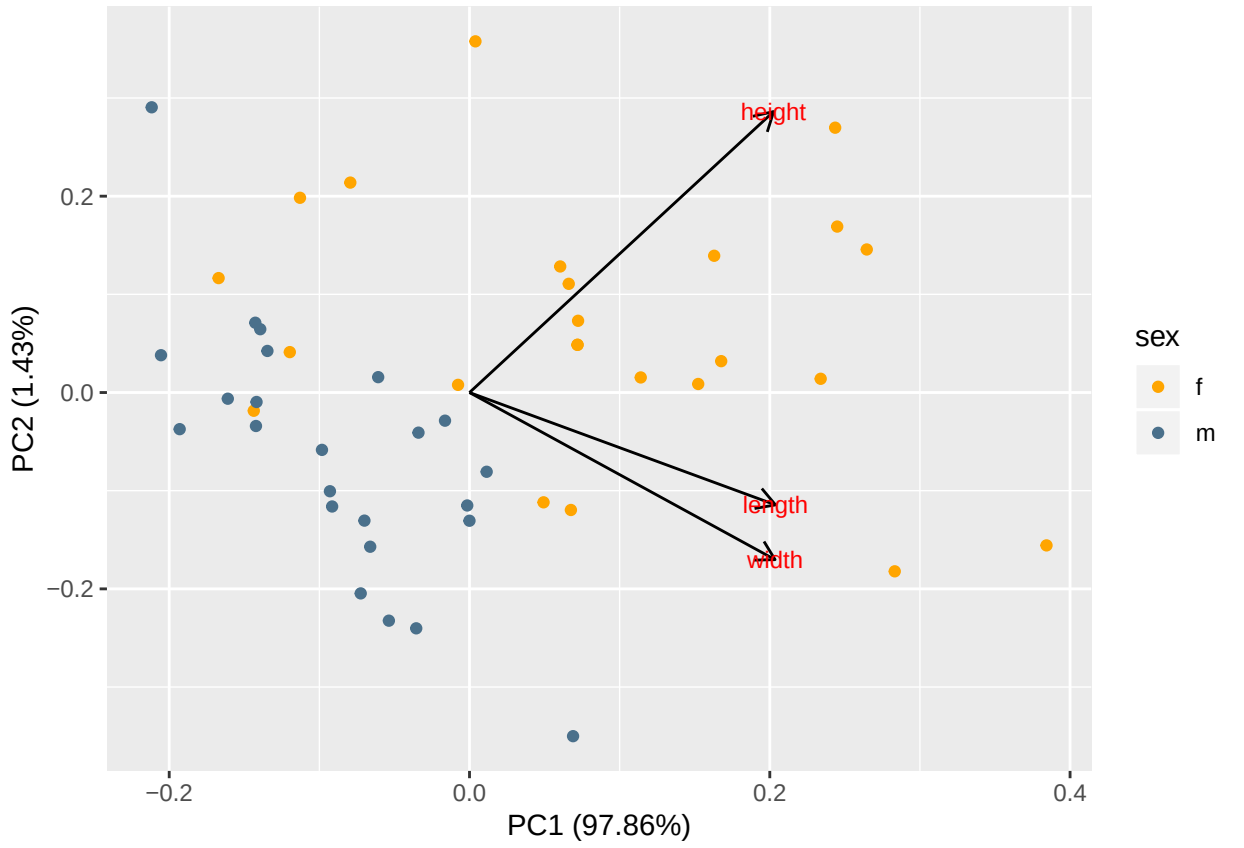


So using prcomp gives very similar results, despite using svd.

Now, if we want to look at the projection, my favorite method is to use autoplot from ggfortify, it is a extension for ggplot.

Note that a plot that contains both the PC's and the loading vectors is called a **Bi-plot**

```r
autoplot(pca_turtles,data=turtles,colour="sex",loadings = TRUE,
         loadings.colour = 'black',loadings.label = TRUE, loadings.label.size = 3) +
  scale_color_manual(values = c("orange", "skyblue4"))
```



1.4.2 Interpretation of the bi-plot

First, each point on this plot is the value of PC1 and PC2. The black arrows are the loadings or eigenvectors.

To interpret the loadings I like to look at the coefficients too. For PC1 we have:

```
pca_turtles$rotation[,1]
```

```
##    length     width    height
## 0.5787981 0.5779840 0.5752628
```

In other words, no one variable is dominating the other. It seems that PC1 is simply capturing the positive relationship between each variable. Recall that the PC is just a "formula" based on the data, that is, in this case, a longer, wider, and taller turtle will always have a larger first PC.

Also note how sex is clustered on either side of the x-axis. It would seem that PC1 might also be capturing some relationship about sex and the other variables.

This notion of interpreting the loadings onto latent variables is used often in factor analysis.

For PC2, we have

```
pca_turtles$rotation[,2]
```

```
##     length      width     height
## -0.3250273 -0.4834699  0.8127817
```

Clearly height is dominating here, and length and width have negative coefficients.

Interpreting this PC in particular is tricky (since it only explains 1.43% of the variability) but it may be capturing some underlying relationship between (height and sex), and (length and width). Notice that there is some vertical overlapping on the negative side of the bi-plot in terms of sex.

1.5 Why bother with SVD?

1.5.1 Numerical stability

It turns out performing eigenvalue decomposition on a computer may not always give precise answer, rather a loss of precision may occur. This is not a issues when we perform SVD, hence most function opt to use this decomposition in favor of eigenvalue decomposition.

I suppose the book was trying to tell us this all along!