Multivariate Stats, Week 5

Noah Silbert

September 22, 2017

To do

1. Review of homework
2. Principal Components Analysis

Functions in R

In last week's homework, I gave you a function that I wrote to carry out two tests of multivariate normality.

If you find yourself writing the same (or very similar) code over and over again, you should probably write a function. Here is the basic format for a function that tests to see if one input argument divided by a second input argument produces a remainder:

```
leftover = function(numerator, denominator=5){
  remainder = numerator %% denominator
  if(remainder > 0){
    return(remainder)
  }else{
    return("NOTHING!")
  }
}
```

Note that the first input argument does not have a default value, whereas the second does.

Using our function

```
leftover(10,5)
```

```
## [1] "NOTHING!"
```

```
leftover(11,5)
```

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

```
leftover(12,5)
```

```
## [1] 2
```

You can create "modules" by writing multiple functions into a `.R` file. Then you can make the functions available for use by calling that file with the function `source()` (which runs all the code in the file).

Principal Components Analysis

In Principal Components Analysis (PCA), we find linear combinations of variables that maximize the (within-group) variance of the linear combinations.

The first principal component of a data set is the linear combination of the variables along which the data have maximal variance.

The second principal component of that data set is linear combination of variables along which variance is maximal *and* the second dimension is orthogonal to the first.

The N^{th} principal component is the linear combination with maximal variance that is orthogonal to all $N-1$ prior principal components.

Principal component analysis can be used to describe and/or reduce the dimensionality of a data set, and the transformed data (i.e., the values on the principal components) can be used as inputs to other analyses (e.g., as predictors in a regression model).

PCA: The Math

First, we will assume that the data $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_p$ are centered (i.e., that the mean has been subtracted from each vector).

If there are correlations among the variables, a scatter plot of the data won't line up with the coordinate axes. We can think of PCA as a rotation so that the data *do* line up with a new set of axes.

Rotation is produced by an orthogonal matrix $\mathbf{A}$. The fact that $\mathbf{A}$ is orthogonal means that $\mathbf{A}^T\mathbf{A} = \mathbf{I}$. If we apply $\mathbf{A}$ to each vector $\mathbf{y}_i$, we get a new variable $\mathbf{z}_i = \mathbf{A}\mathbf{y}_i$.

If we want our new axes to be uncorrelated (i.e., each principal component to be orthogonal to all previous principal components), this is equivalent to stating that we want the covariance matrix of $\mathbf{Z}$ to be diagonal. Let $\mathbf{S}$ be the covariance matrix of $\mathbf{Y}$, then:

$$\mathbf{S}_z = \mathbf{A}\mathbf{S}\mathbf{A}^T = \begin{pmatrix} s_{z1}^2 & 0 & \ldots & 0 \\ 0 & s_{z2}^2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & s_{zp}^2 \end{pmatrix} \tag{1}$$

More PCA Math

The variances of the principal components s_{zi}^2 are the *eigenvalues* of $\mathbf{S}$, and the columns of $\mathbf{A}$ are the *eigenvectors* of $\mathbf{S}$. The principal components are the transformed variables $\mathbf{z}_i = \mathbf{Y}\mathbf{a}_i$, where $\mathbf{a}_i$ is the i^{th} column of $\mathbf{A}$.

An eigenvector $\mathbf{v}$ and eigenvalue λ (a.k.a. a characteristic vector and value) of a $N \times N$ matrix $\mathbf{T}$ are the vector and value such that $\mathbf{T}\mathbf{v} = \lambda\mathbf{v}$.

That is, if a linear transformation of a vector can be expressed as a scalar multiplication, then that vector is an eigenvector of the matrix implementing the linear transformation, and the scalar is the corresponding eigenvalue.

Eigenvector and eigenvalues come up *a lot* in multivariate statistics, so although they may not be particularly intuitive, they are very useful.

More PCA Math

If a set of variables are highly correlated, the first few eigenvalues of the covariance matrix **S** will be large, in which case we can represent the data set fairly well with just the first few principal components (i.e., we can reduce the dimensionality of the data).

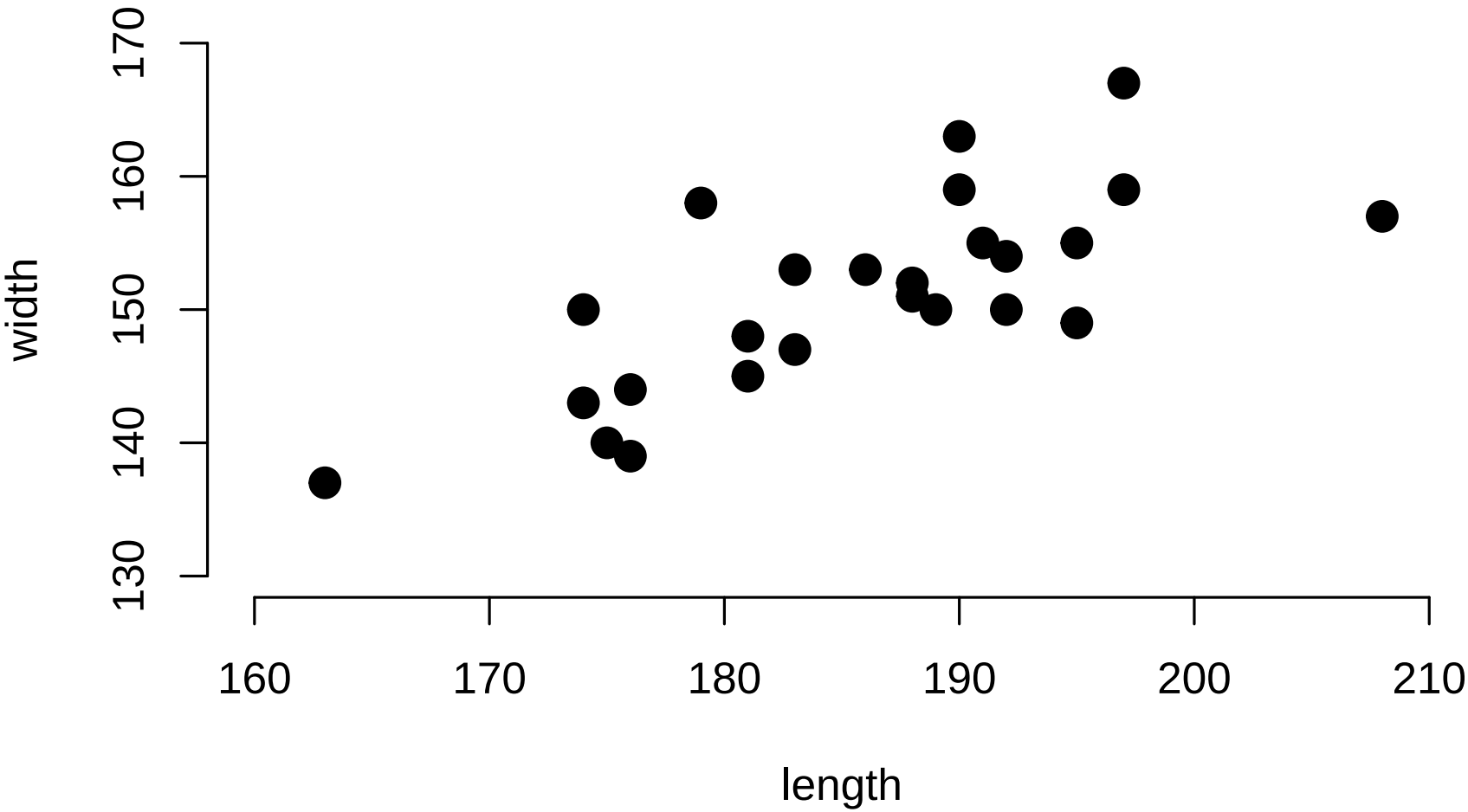
The amount of variance "explained" by the first k principal components can be calculated as $\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{p} \lambda_j}$, where $\lambda_i = s_{zi}^2$ (i.e., the λs are the eigenvalues of **S**).

If the variances are very different for different variables, or if the variables are on incomensurate scales, you can standardize a data set prior to performing PCA to put all of the variables on a common scale. In this case, you just find the eigenvalue and eigenvectors of the correlation matrix **R** rather than the covariance matrix **S**.

An example

Here's a set of head length ($\mathbf{y}_1$) and head width ($\mathbf{y}_2$) data from a set of first-born sons (i.e., the first two columns of the data in `T3_8_SONS.DAT`):

```
sons = read.table("T3_8_SONS.DAT")
colnames(sons) = c("length.1","width.1","length.2","width.2")
```



An example continued

We can get the covariance matrix for the first two columns using `cov()`:

```
S = cov(sons[,c("length.1","width.1")])
print(round(S,2))
```

```
##           length.1 width.1
## length.1     95.29   52.87
## width.1      52.87   54.36
```

We can get the eigenvalues and eigenvectors of **S** with the function `eigen()`, which returns a list containing the variables `values` and `vectors`:

```
eig = eigen(S)
```

Further continuation of the example

Recall that you can access elements of a list using $:

```
eig$values
```

```
## [1] 131.5183  18.1350
```

```
eig$values/sum(eig$values)
```

```
## [1] 0.8788199 0.1211801
```

Recall that the eigenvalues are the variances of the principal components. Dividing the eigenvalues by the sum of the eigenvalues gives us the proportion of variance accounted for by each principal component.

For this data (sub)set, the first principal component accounts for about 88% of the variance in the data.

A continued example, continued again

The eigen*vectors* contain the coefficients that transform the original variables into the principal components. These are also sometimes called *loadings*.

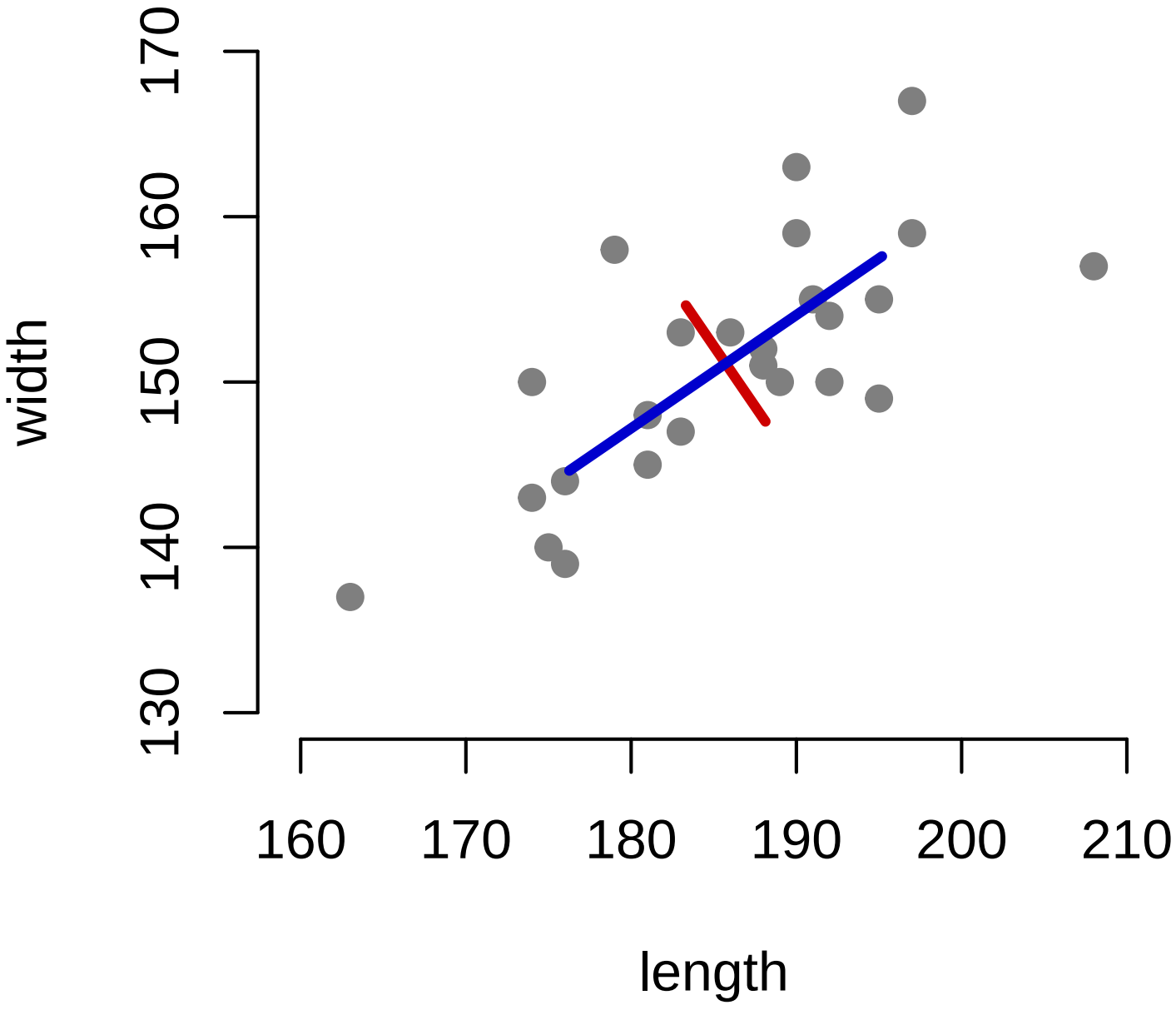
```
round(eig$vectors,2)
```

```
##       [,1]  [,2]
## [1,] -0.82  0.57
## [2,] -0.57 -0.82
```

The numbers in the first column correspond to the first eigenvalue and the first principal component, and the numbers in the second column to the second eigenvalue/principal component.

Note that you can multiply an eigenvector by any scalar value c and it would still be an eigenvector of the matrix in question.

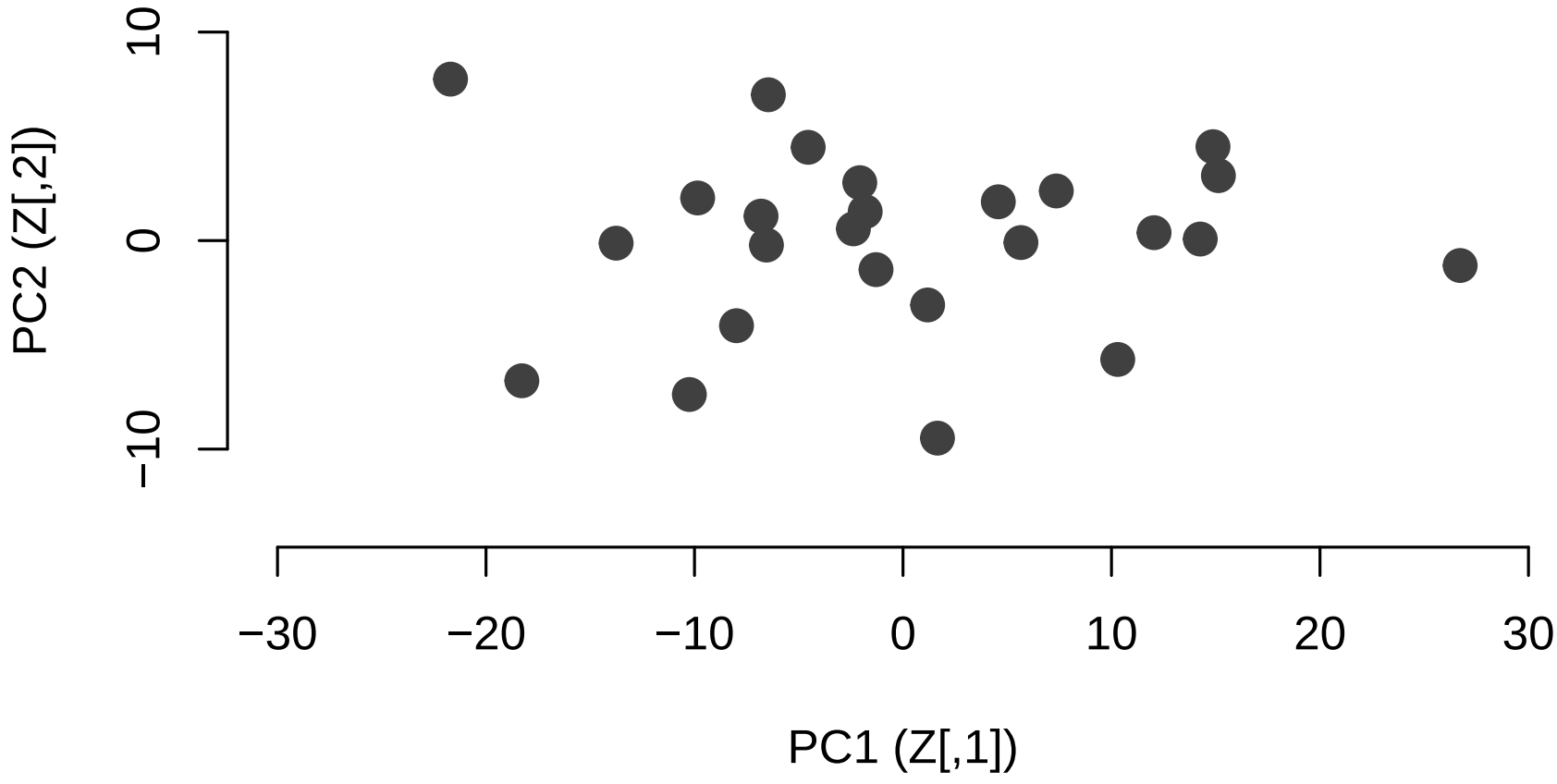
Illustration of principal components



Still more continued continuation

We can transform the (centered) data with the eigenvectors:

```
A = eig$vectors
Y = cbind(sons[,1]-mean(sons[,1]),sons[,2]-mean(sons[,2]))
Z = Y %*% A
```



Dimensionality reduction via PCA

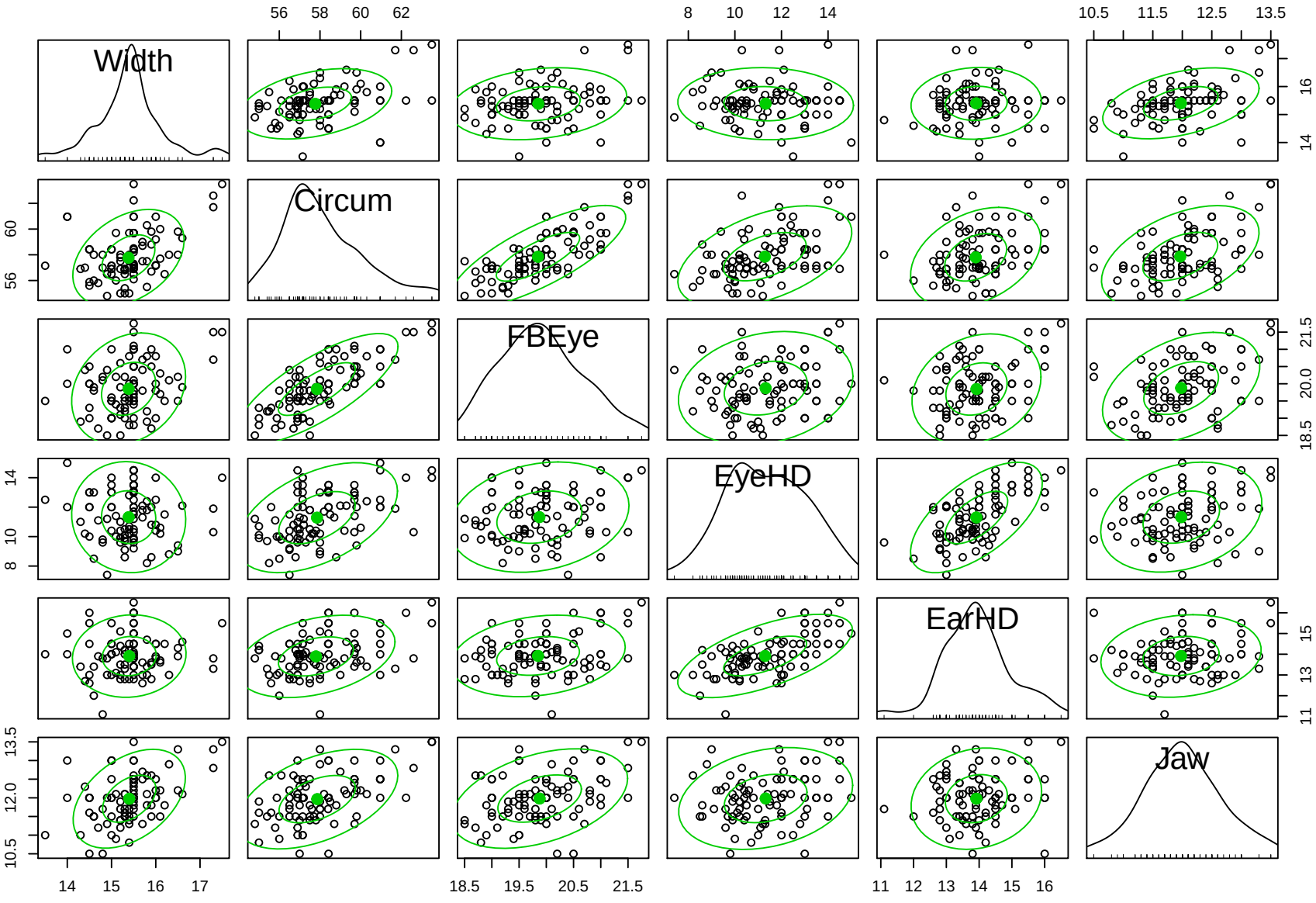
Here's another, higher-dimensional data set (Table 8.3 in the textbook). This is also measurements of people's heads for three groups of people from a study of football helmet design and neck injuries. The three groups are high school football players (1), college football players (2), and non-football players (3).

Using groups 2 and 3 from this data, we can illustrate dimensionality reduction using PCA.

```
fb.all = read.table("T8_3_FOOTBALL.DAT")
colnames(fb.all) = c("Group","Width","Circum","FBEye",
                     "EyeHD","EarHD","Jaw")
round(cor(fb.all[fb.all$Group %in% c(2,3),2:7]),2)
```

```
##         Width Circum FBEye EyeHD EarHD   Jaw
## Width    1.00   0.61  0.36  0.06  0.25  0.60
## Circum   0.61   1.00  0.73  0.34  0.09  0.41
## FBEye    0.36   0.73  1.00  0.01 -0.03  0.31
## EyeHD    0.06   0.34  0.01  1.00  0.30 -0.08
## EarHD    0.25   0.09 -0.03  0.30  1.00 -0.09
## Jaw      0.60   0.41  0.31 -0.08 -0.09  1.00
```

Data visualization



PCA for dimensionality reduction

We select the Group 2 and 3 data, get the covariance matrix, get the eigenvalues, and check the variances of the principal components to decide how many components to keep (i.e, how many dimensions the reduced space should have - we'll return to this shortly):

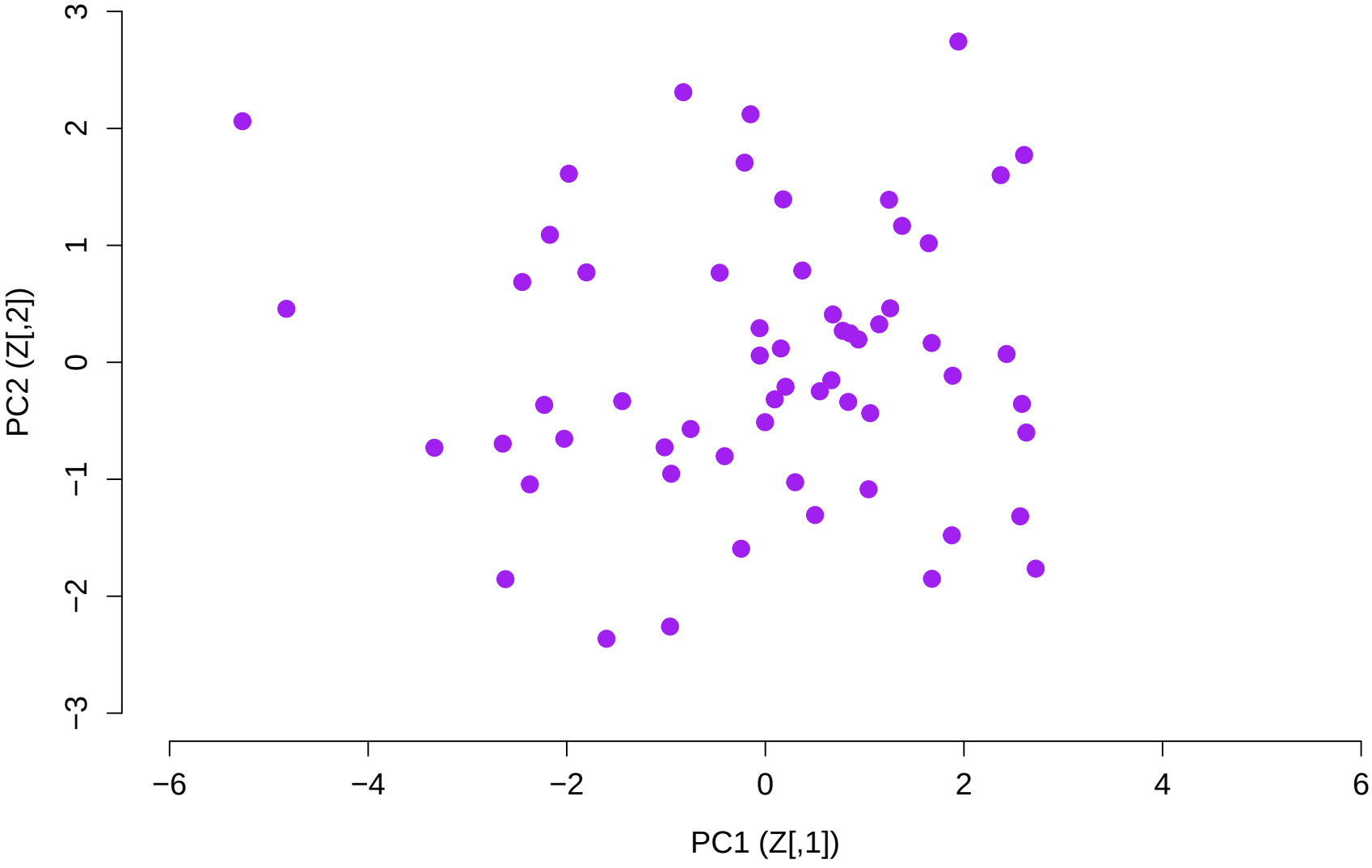
```
fb = fb.all[fb.all$Group %in% c(2,3),2:7]
S = cov(fb)
eig = eigen(S)
lam = eig$values
round(lam/sum(lam),2)
```

```
## [1] 0.58 0.24 0.08 0.06 0.03 0.02
```

In this case, the first two components account for 82% of the variance, so we can reduce the dimensionality to 2, and discard the third through seventh components:

```
A = eig$vectors
Y = as.matrix(fb)
for(i in 1:ncol(fb)){Y[,i] = Y[,i]-mean(Y[,i])}
Z = Y%*%A[,1:2]
```

Plotting reduced dimension data



Interpreting the principal components

The elements of the eigenvectors (i.e., the loadings) give us information that can be useful for interpreting the principal components. The higher a loading, the more strongly the corresponding original variable contributes to a component.

```
rownames(A) = colnames(fb)
round(A[,1:2],2)
```

```
##          [,1]  [,2]
## Width   -0.21  0.14
## Circum  -0.87  0.22
## FBEye   -0.26  0.23
## EyeHD   -0.33 -0.89
## EarHD   -0.07 -0.22
## Jaw     -0.13  0.19
```

Because we did PCA on the covariance (rather than correlation) matrix, the loadings are more difficult to interpret. We can draw ordinal inferences easily, but stronger (interval or ratio level) inferences are less readily available to us.

PCA with S vs R

The eigenvalues and eigenvectors of S and R need not be similar:

```
R = cov2cor(S)
eigR = eigen(R)
lamR = eigR$values
AR = eigR$vectors
round(rbind(lam,lamR),3)
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
## lam  3.323 1.374 0.476 0.325 0.156 0.088
## lamR 2.568 1.369 0.932 0.678 0.322 0.131
```

```
round(rbind(lam/sum(lam),lamR/sum(lamR)),3)
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
## [1,] 0.579 0.239 0.083 0.057 0.027 0.015
## [2,] 0.428 0.228 0.155 0.113 0.054 0.022
```

PCA with S vs R

Comparing loadings for PCA on **S** vs **R**:

```
round(A[,1:2],2)
```

```
##          [,1]  [,2]
## Width   -0.21  0.14
## Circum  -0.87  0.22
## FBEye   -0.26  0.23
## EyeHD   -0.33 -0.89
## EarHD   -0.07 -0.22
## Jaw     -0.13  0.19
```

```
round(AR[,1:2],2)
```

```
##          [,1]  [,2]
## Width   -0.51  0.01
## Circum  -0.56 -0.09
## FBEye   -0.46  0.15
## EyeHD   -0.14 -0.66
## EarHD   -0.11 -0.64
## Jaw     -0.42  0.34
```

Choosing how many dimensions to keep

Options are:

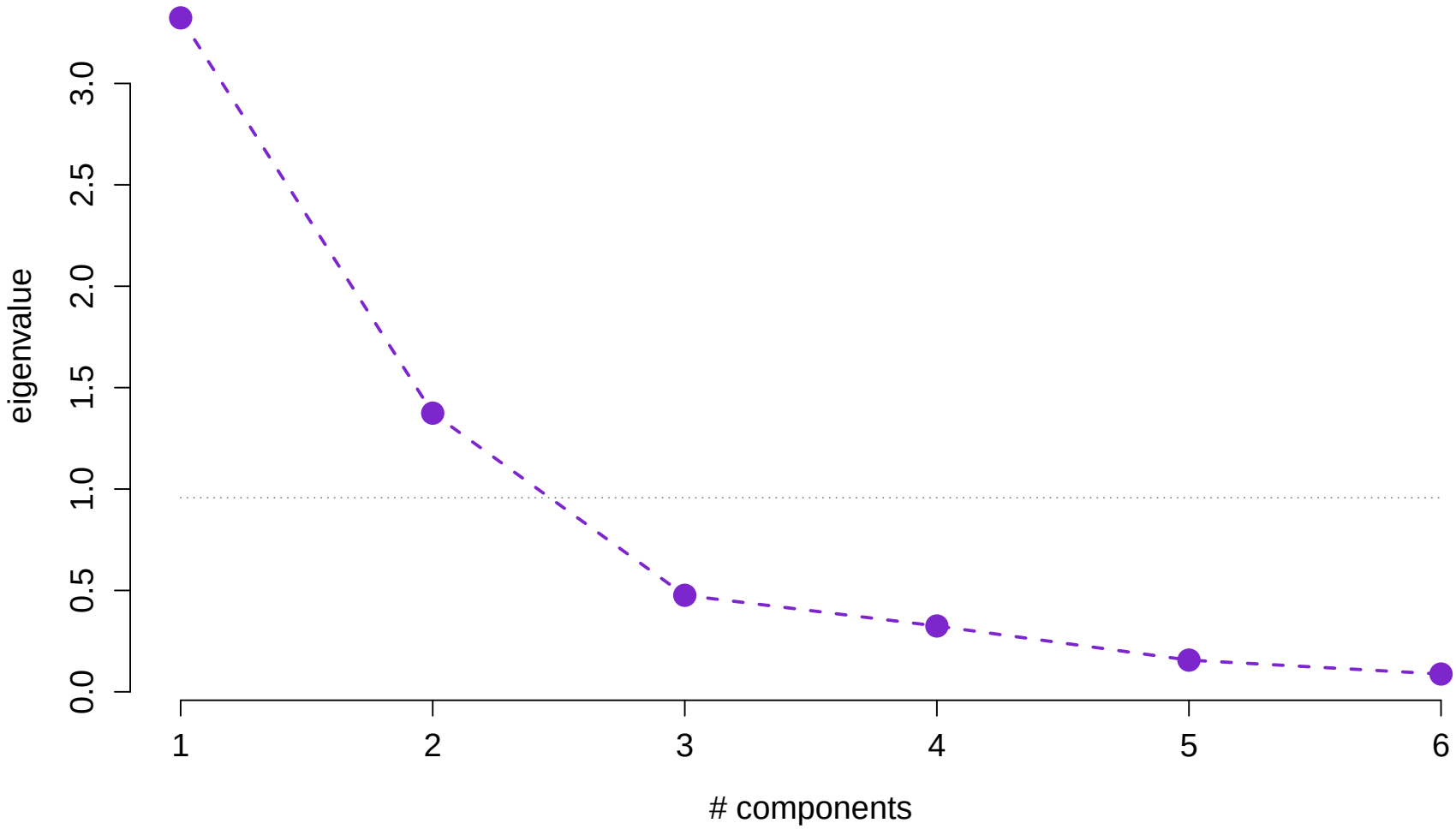
1. Retain dimensions sufficient to account for a particular amount of variance (e.g., 80%)
2. Retain components with eigenvalues greater than the average eigenvalue (with **R**, this criterion is 1).
3. Plot the eigenvalues and look for an "elbow", or a "natural" break between "large" and "small" eigenvalues
4. Test the significance of the components.

To test the significance of the larger components, we test the hypothesis that the smaller components are equal: $H_0 : \lambda_{p-k+1} = \cdots = \lambda_p$. We do this by calculating the average of the last k eigenvalues, $\bar{\lambda} = \frac{1}{k} \sum_{i=p-k+1}^{p} \lambda_i$, and use the test statistic $u = \left(n - \frac{2p+11}{6}\right) \left(k \ln \bar{\lambda} - \sum_{i=p-k+1}^{p} \ln \lambda_i\right)$, which is approximately distributed as a χ^2 random variable with df $\frac{1}{2}(k-1)(k+2)$.

The football data again

```
round(lam/sum(lam),2)
```

```
## [1] 0.58 0.24 0.08 0.06 0.03 0.02
```



Statistical test of component retention

Here is a function that calculates the test statistic described above, taking as input a vector of eigenvalues, and numbers `k` and `n`, the number of eigenvalues hypothesized to be equal and the sample size, respectively, and returning the test statistic and p value:

```
pca_test = function(lambda,k,n){
  p = length(lambda)
  lbar = mean(lambda[(p-k+1):p])
  u = (n-(2*p+11)/6)*(k*log(lbar)-sum(log(lambda[(p-k+1):p])))
  return(list(u=u,p=pchisq(u,.5*(k-1)*(k+2),lower.tail=F)))
}
```

Statistical test of component retention with the football data

This code initializes a vector for storing p values, then it loops from 2 to 5, calling `pca_test()` to check if the last 2, 3, ... eigenvalues are equal:

```
uvals = vector(length=4)
pvals = vector(length=4)
for(i in 2:5){
  test.out = pca_test(lam,k=i,n=nrow(fb))
  pvals[i-1] = round(test.out$p,3)
  uvals[i-1] = round(test.out$u,3)
}
out = rbind(uvals,pvals)
colnames(out) = paste("k = ",2:5,sep="")
out
```

```
##       k = 2 k = 3  k = 4   k = 5
## uvals 4.618 23.84 44.102 123.926
## pvals 0.099  0.00  0.000   0.000
```

Note that this result suggests we should keep four components, whereas the skree plot suggested that we should only keep two. Isn't statistical analysis fun?

An absurdly brief overview of factor analysis (more later in the semester)

Factor analysis uses linear combinations of variables in much the same way that PCA does. However, the general goal of factor analysis is different than the general goal of PCA.

In PCA, we seek linear combinations of variables that maximize the variance of the transformed variables.

In factor analysis, we seek linear combinations of variables to account for patterns of covariance between the original variables.

The factors in factor analysis are much like principal components, except that they may be correlated with each other.

There are two types of factor analysis: exploratory and confirmatory. In exploratory factor analysis, the full set of loadings (i.e., coefficients weighting the variables) is estimated, and the factors are rotated according to one or another criterion. In confirmatory factor analysis, you decide ahead of time which loadings you want to estimate (i.e., which variables serve as indicators of which factors), which typically constrains the model quite a bit (the EFA model is very unconstrained, so various constraints have to be placed on it to estimate the parameters).