# Tutorial 2

Quentin Bouet

2024-08-07

## Lecture: PCA Theory

During the lecture this week, I skipped over the theory behind PCA. It is however crucial to grasp how PCA works. Since students in MA3405 have varying levels of mathematical expertise, we will take a step-by-step approach to cover the theory of PCA thoroughly.

We will use the NCI60 data in ISLR2 package for demonstration. We will start by loading the dataset on the working directory,

```
#install.packages('ISLR2')
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.3.3
```

```
nci.labs <- NCI60$labs
table(nci.labs)
```

```
## nci.labs
##      BREAST         CNS       COLON K562A-repro K562B-repro    LEUKEMIA
##           7           5           7           1           1           6
## MCF7A-repro MCF7D-repro    MELANOMA       NSCLC     OVARIAN    PROSTATE
##           1           1           8           9           6           2
##       RENAL     UNKNOWN
##           9           1
```

```
nci.data <- NCI60$data
dim(nci.data)
```

```
## [1]   64 6830
```

NCI60 data contain gene expression level of 6830 genes from 64 cancer cell lines. The format is a list containing two elements: `data` and `labs`. `data` is a 64 by 6830 matrix of the expression values while `labs` is a vector listing the cancer types for the 64 cell lines.

Principal components are orthogonal vectors which explains variation of the data. The goal of a PCA is to find a new set of variables (i.e. smaller than original number of variables) that best describe the variation in the dataset. This is achieved via eigendecomposition of the covariance matrix. Therefore, the first step is to find the covariance matrix of centred nci.data with cov funciton. TO save computation time, we will only use the expression of first 500 genes,

```
nci.sub<- nci.data[, 1:500] #use first 500 genes
nci.sub.scale<-scale(nci.sub, scale = TRUE) # centering
var.cov<-cov(nci.sub) # covariance
```

The next step is to eigenden composition of the variance-covariance matrix. To do this in R, we need to use eigen function in matlib library,

```
library(matlib)
```

```
## Warning: package 'matlib' was built under R version 4.3.3
```

```
e_decom<-eigen(var.cov)
#e_decom
```

The function returns two elements, the first element is the eigenvalue of var-cov matris, while the second element is the eigenvector. We can see amount of variation explained by dividing each eigenvalue by the number of sample -1,

```
e_var<-e_decom$values/(nrow(nci.sub.scale)-1)
#We can convert this into percentage
e_var_per<-e_var/sum(e_var)
```

We can use cumsum function to see cumulative variance explained by each eigenvector,

```
cumsum(e_var_per)
```

```
##   [1] 0.2873299 0.3530888 0.4071624 0.4556800 0.4951736 0.5282026 0.5566873
##   [8] 0.5825078 0.6053133 0.6259047 0.6446653 0.6625738 0.6797258 0.6959969
##  [15] 0.7111110 0.7252893 0.7380529 0.7506153 0.7627741 0.7744136 0.7854348
##  [22] 0.7962173 0.8068821 0.8166017 0.8259323 0.8349158 0.8436573 0.8519819
##  [29] 0.8599666 0.8675347 0.8748886 0.8820126 0.8889987 0.8955669 0.9020307
##  [36] 0.9079449 0.9136513 0.9190380 0.9242756 0.9292143 0.9340148 0.9386865
##  [43] 0.9431378 0.9473779 0.9515888 0.9555294 0.9593546 0.9630147 0.9664972
##  [50] 0.9698545 0.9731686 0.9762408 0.9792284 0.9820359 0.9846295 0.9871846
##  [57] 0.9896477 0.9919106 0.9940943 0.9960666 0.9978008 0.9990194 1.0000000
##  [64] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [71] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [78] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [85] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [92] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [99] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [106] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [113] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [120] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [127] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [134] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [141] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [148] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [155] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [162] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [169] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
```

```
## [176] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [183] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [190] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [197] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [204] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [211] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [218] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [225] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [232] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [239] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [246] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [253] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [260] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [267] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [274] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [281] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [288] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [295] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [302] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [309] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [316] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [323] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [330] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [337] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [344] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [351] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [358] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [365] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [372] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [379] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [386] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [393] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [400] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [407] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [414] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [421] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [428] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [435] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [442] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [449] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [456] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [463] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [470] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [477] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [484] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [491] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [498] 1.0000000 1.0000000 1.0000000
```

The first 35 eigenvectors explained around 90% variation, and the 62 eigenvectors explained 99% of vairation. Let's produce a scree plot,

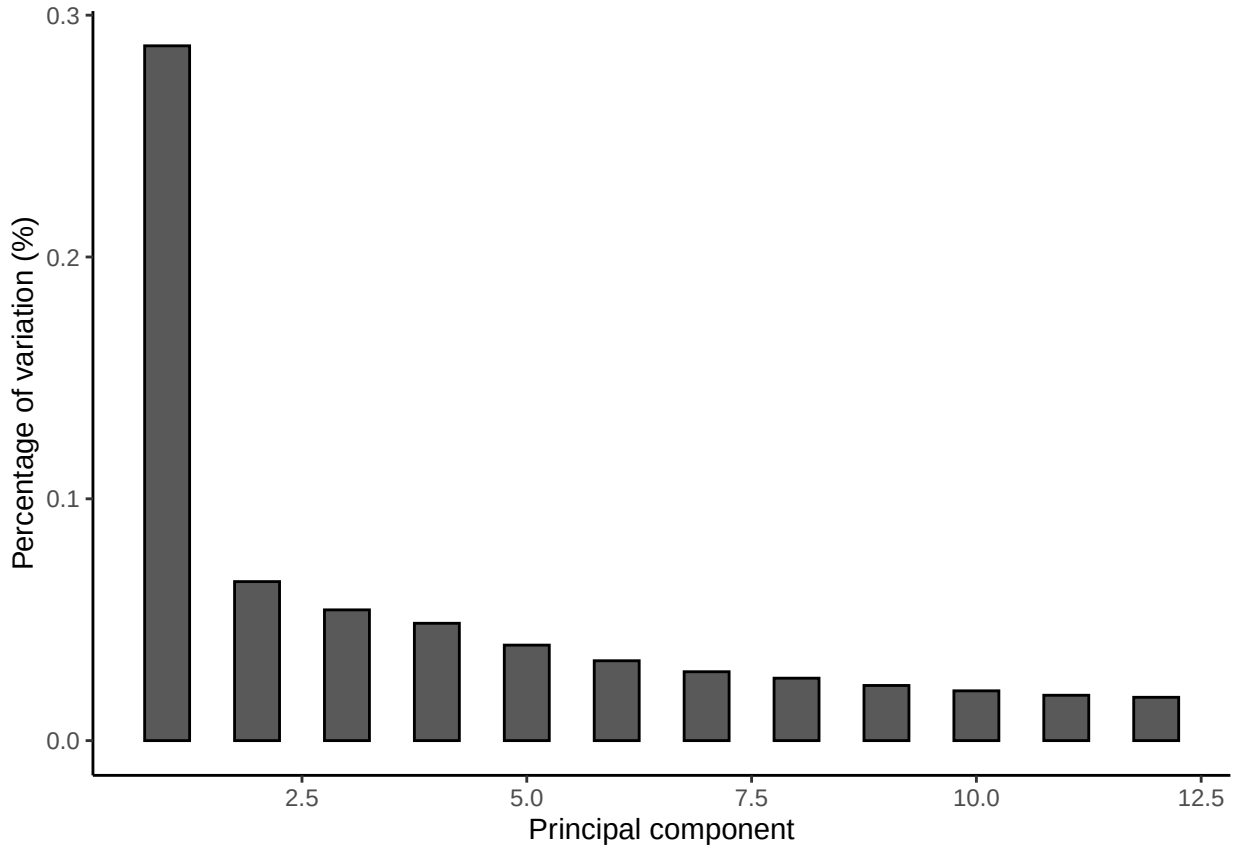
```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```r
pp<-data.frame("PC"=c(1:12), PER=e_var_per[1:12])
pp
```

```
##    PC        PER
## 1   1 0.28732989
## 2   2 0.06575895
## 3   3 0.05407357
## 4   4 0.04851763
## 5   5 0.03949360
## 6   6 0.03302898
## 7   7 0.02848468
## 8   8 0.02582047
## 9   9 0.02280553
## 10 10 0.02059140
## 11 11 0.01876055
## 12 12 0.01790858
```

```r
ggplot(pp, aes(x = PC, y = PER)) +
geom_col(width = 0.5, color = "black") +
xlab("Principal component") +
ylab("Percentage of variation (%)") +
theme_classic()
```




Let's use princomp function in R to run PCA,

```r
pca<-prcomp(nci.sub, center = TRUE)
summary(pca)
```

```
## Importance of components:
##                             PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     11.2764 5.39460 4.89186 4.63373 4.18066 3.82322 3.55048
## Proportion of Variance  0.2873 0.06576 0.05407 0.04852 0.03949 0.03303 0.02848
## Cumulative Proportion   0.2873 0.35309 0.40716 0.45568 0.49517 0.52820 0.55669
##                             PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     3.38036 3.17688 3.01873 2.88140 2.81522 2.75511 2.68342
## Proportion of Variance 0.02582 0.02281 0.02059 0.01876 0.01791 0.01715 0.01627
## Cumulative Proportion  0.58251 0.60531 0.62590 0.64467 0.66257 0.67973 0.69600
##                            PC15    PC16    PC17    PC18    PC19    PC20    PC21
## Standard deviation     2.58627 2.50492 2.37666 2.35787 2.31967 2.26959 2.20850
## Proportion of Variance 0.01511 0.01418 0.01276 0.01256 0.01216 0.01164 0.01102
## Cumulative Proportion  0.71111 0.72529 0.73805 0.75062 0.76277 0.77441 0.78543
##                            PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     2.18444 2.17249 2.07398 2.03206 1.99391 1.96686 1.91939
## Proportion of Variance 0.01078 0.01066 0.00972 0.00933 0.00898 0.00874 0.00832
## Cumulative Proportion  0.79622 0.80688 0.81660 0.82593 0.83492 0.84366 0.85198
##                            PC29    PC30    PC31    PC32    PC33    PC34    PC35
## Standard deviation     1.87980 1.83010 1.80402 1.77559 1.75833 1.70492 1.69132
## Proportion of Variance 0.00798 0.00757 0.00735 0.00712 0.00699 0.00657 0.00646
## Cumulative Proportion  0.85997 0.86753 0.87489 0.88201 0.88900 0.89557 0.90203
##                            PC36    PC37    PC38    PC39    PC40   PC41    PC42
## Standard deviation     1.61781 1.58915 1.54397 1.52247 1.47839 1.4576 1.43787
## Proportion of Variance 0.00591 0.00571 0.00539 0.00524 0.00494 0.0048 0.00467
## Cumulative Proportion  0.90794 0.91365 0.91904 0.92428 0.92921 0.9340 0.93869
##                            PC43    PC44    PC45    PC46    PC47    PC48    PC49
## Standard deviation     1.40354 1.36984 1.36511 1.32058 1.30109 1.27270 1.24144
## Proportion of Variance 0.00445 0.00424 0.00421 0.00394 0.00383 0.00366 0.00348
## Cumulative Proportion  0.94314 0.94738 0.95159 0.95553 0.95935 0.96301 0.96650
##                            PC50    PC51    PC52    PC53    PC54    PC55    PC56
## Standard deviation     1.21892 1.21106 1.16602 1.14984 1.11467 1.07134 1.06337
## Proportion of Variance 0.00336 0.00331 0.00307 0.00299 0.00281 0.00259 0.00256
## Cumulative Proportion  0.96985 0.97317 0.97624 0.97923 0.98204 0.98463 0.98718
##                            PC57    PC58    PC59    PC60    PC61    PC62    PC63
## Standard deviation     1.04406 1.00072 0.98305 0.93428 0.87604 0.73438 0.65875
## Proportion of Variance 0.00246 0.00226 0.00218 0.00197 0.00173 0.00122 0.00098
## Cumulative Proportion  0.98965 0.99191 0.99409 0.99607 0.99780 0.99902 1.00000
##                            PC64
## Standard deviation     5.008e-15
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

summary shows amount of variance explained by each component, these are the same as the values we got using eigen decomposition (pp). There is however some variation in the factor loading, this is because princomp uses Single value decomposition instead of eigen decomposition.

# Labs

## 12.5.1 Principal Components Analysis

In this lab, we perform PCA on the USArrests data set, which is part of the base R package. The rows of the data set contain the 50 states, in alphabetical order.

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 4.3.3
```

```
##
## Attaching package: 'ISLR'
```

```
## The following objects are masked from 'package:ISLR2':
##
##     Auto, Credit
```

```
data("USArrests")
states <- row.names(USArrests)
states
```

```
##  [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"
##  [5] "California"     "Colorado"       "Connecticut"    "Delaware"
##  [9] "Florida"        "Georgia"        "Hawaii"         "Idaho"
## [13] "Illinois"       "Indiana"        "Iowa"           "Kansas"
## [17] "Kentucky"       "Louisiana"      "Maine"          "Maryland"
## [21] "Massachusetts"  "Michigan"       "Minnesota"      "Mississippi"
## [25] "Missouri"       "Montana"        "Nebraska"       "Nevada"
## [29] "New Hampshire"  "New Jersey"     "New Mexico"     "New York"
## [33] "North Carolina" "North Dakota"   "Ohio"           "Oklahoma"
## [37] "Oregon"         "Pennsylvania"   "Rhode Island"   "South Carolina"
## [41] "South Dakota"   "Tennessee"      "Texas"          "Utah"
## [45] "Vermont"        "Virginia"       "Washington"     "West Virginia"
## [49] "Wisconsin"      "Wyoming"
```

The columns of the data set contain the four variables.

```
names(USArrests)
```

```
## [1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

We first briefly examine the data. We notice that the variables have vastly different means.

```
apply(USArrests, 2, mean)
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

Note that the `apply()` function allows us to apply a function—in this case, the `mean()` function to each row or column of the data set. The second input here denotes whether we wish to compute the mean of the rows, 1, or the columns, 2. We see that there are on average three times as many rapes as murders, and more than eight times as many assaults as rapes. We can also examine the variances of the four variables using the `apply()` function.

```
apply(USArrests, 2, var)
```

```
##     Murder    Assault   UrbanPop       Rape
##   18.97047 6945.16571  209.51878   87.72916
```

Not surprisingly, the variables also have vastly different variances: the UrbanPop variable measures the percentage of the population in each state living in an urban area, which is not a comparable number to the number of rapes in each state per 100,000 individuals. If we failed to scale the variables before performing PCA, then most of the principal components that we observed would be driven by the Assault variable, since it has by far the largest mean and variance. Thus, it is important to standardize the variables to have mean zero and standard deviation one before performing PCA.

We now perform principal components analysis using the `prcomp()` function, which is one of several functions in R that perform PCA.

```
pr.out <- prcomp(USArrests, scale = TRUE)
```

By default, the prcomp() function centers the variables to have mean zero. By using the option scale = TRUE, we scale the variables to have standard deviation one. The output from `prcomp()` contains a number of useful quan- tities.

```
names(pr.out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

The center and scale components correspond to the means and standard deviations of the variables that were used for scaling prior to implementing PCA.

```
pr.out$center
```

```
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
```

```
pr.out$scale
```

```
##    Murder   Assault  UrbanPop      Rape
##  4.355510 83.337661 14.474763  9.366385
```

The rotation matrix provides the principal component loadings; each col- umn of pr.out$rotation contains the corresponding principal component loading vector.

```
# This function names it the rotation matrix, because when we matrix-multiply the
# X matrix by pr.out$rotation, it gives us the coordinates of the data in the rotated
# coordinate system. These coordinates are the principal component scores.
pr.out$rotation
```

```
##                PC1        PC2        PC3         PC4
## Murder   -0.5358995 -0.4181809  0.3412327  0.64922780
## Assault  -0.5831836 -0.1879856  0.2681484 -0.74340748
## UrbanPop -0.2781909  0.8728062  0.3780158  0.13387773
## Rape     -0.5434321  0.1673186 -0.8177779  0.08902432
```

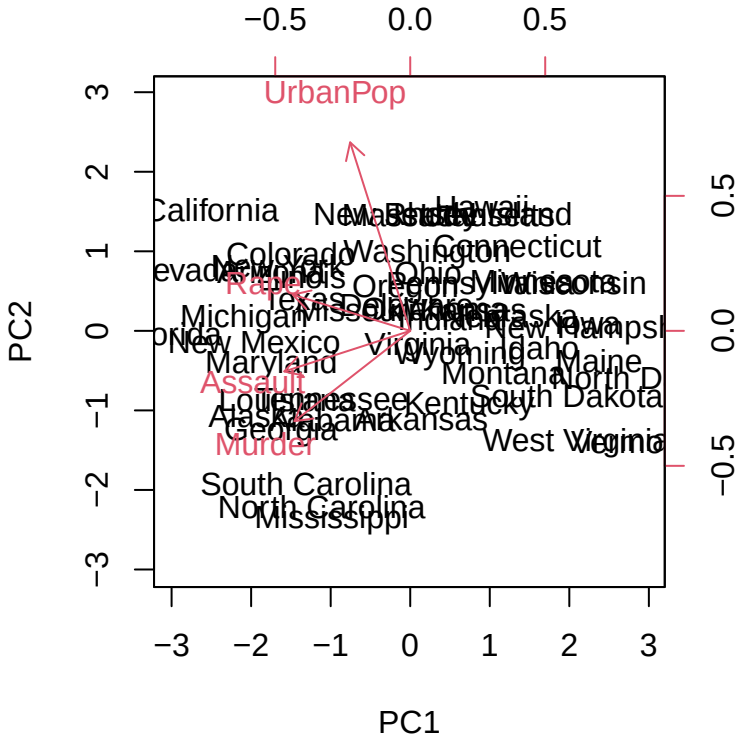
We see that there are four distinct principal components. This is to be expected because there are in general min(n - 1, p) informative principal components in a data set with n observations and p variables. Using the `prcomp()` function, we do not need to explicitly multiply the data by the principal component loading vectors in order to obtain the principal component score vectors. Rather the 50 × 4 matrix x has as its columns the principal component score vectors. That is, the kth column is the kth principal component score vector.

```
dim(pr.out$x)
```

```
## [1] 50  4
```

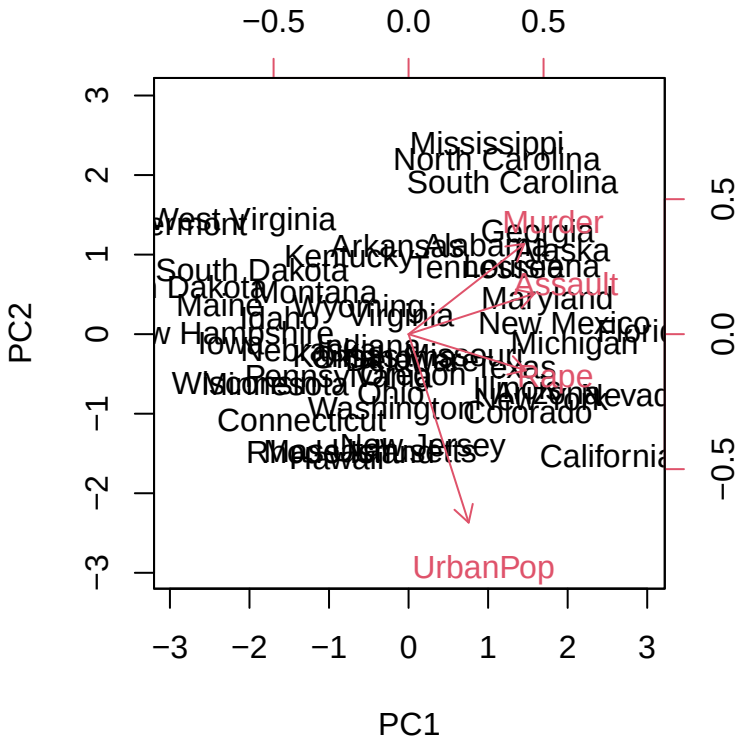
We can plot the first two principal components as follows:

```
biplot(pr.out, scale = 0)
```




The scale = 0 argument to `biplot()` ensures that the arrows are scaled to `biplot()` represent the loadings; other values for scale give slightly different biplots with different interpretations. Notice that this figure is a mirror image of Figure 12.1. Recall that the principal components are only unique up to a sign change, so we can reproduce Figure 12.1 by making a few small changes:

```
pr.out$rotation = -pr.out$rotation
pr.out$x = -pr.out$x
biplot(pr.out, scale = 0)
```




The `prcomp()` function also outputs the standard deviation of each principal component. For instance, on the USArrests data set, we can access these standard deviations as follows:

```
pr.out$sdev
```

```
## [1] 1.5748783 0.9948694 0.5971291 0.4164494
```

The variance explained by each principal component is obtained by squaring these:

```
pr.var <- pr.out$sdev^2
pr.var
```

```
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
```

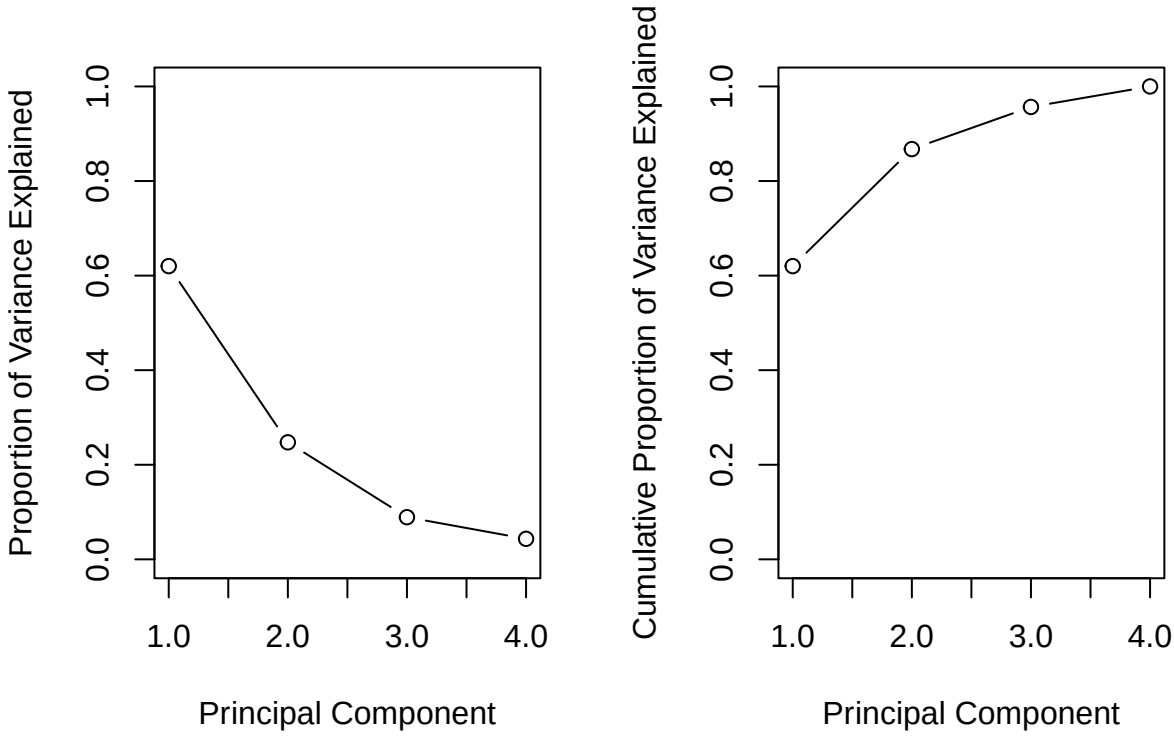
To compute the proportion of variance explained by each principal component, we simply divide the variance explained by each principal component by the total variance explained by all four principal components:

```
pve <- pr.var / sum(pr.var)
pve
```

```
## [1] 0.62006039 0.24744129 0.08914080 0.04335752
```

We see that the first principal component explains 62.0 % of the variance in the data, the next principal component explains 24.7 % of the variance, and so forth. We can plot the PVE explained by each component, as well as the cumulative PVE, as follows:

```
par(mfrow = c(1, 2))
plot(pve, xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     ylim = c(0, 1),
     type = "b")
plot(cumsum(pve), xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     ylim = c(0, 1),
     type = "b")
```




The result is shown in Figure 12.3. Note that the function `cumsum()` computes the cumulative sum of the elements of a numeric vector. For instance:

```
a <- c(1, 2, 8, -3)
cumsum(a)
```

```
## [1]  1  3 11  8
```

### 12.5.4 NCI60 Data Example

Unsupervised techniques are often used in the analysis of genomic data. In particular, PCA and hierarchical clustering are popular tools. We illus- trate these techniques on the NCI60 cancer cell line microarray data, which consists of 6,830 gene expression measurements on 64 cancer cell lines.

```
library(ISLR2)
nci.labs <- NCI60$labs
nci.data <- NCI60$data
```

Each cell line is labeled with a cancer type, given in nci.labs. We do not make use of the cancer types in performing PCA and clustering, as these are unsupervised techniques. But after performing PCA and clustering, we will check to see the extent to which these cancer types agree with the results of these unsupervised techniques.

The data has 64 rows and 6,830 columns.

```
dim(nci.data)
```

```
## [1]   64 6830
```

We begin by examining the cancer types for the cell lines.

```
nci.labs [1:4]
```

```
## [1] "CNS"   "CNS"   "CNS"   "RENAL"
```

```
table(nci.labs)
```

```
## nci.labs
##      BREAST         CNS       COLON K562A-repro K562B-repro    LEUKEMIA
##           7           5           7           1           1           6
## MCF7A-repro MCF7D-repro    MELANOMA       NSCLC     OVARIAN    PROSTATE
##           1           1           8           9           6           2
##       RENAL     UNKNOWN
##           9           1
```

#### PCA on the NCI60 Data

We first perform PCA on the data after scaling the variables (genes) to have standard deviation one, although one could reasonably argue that it is better not to scale the genes.

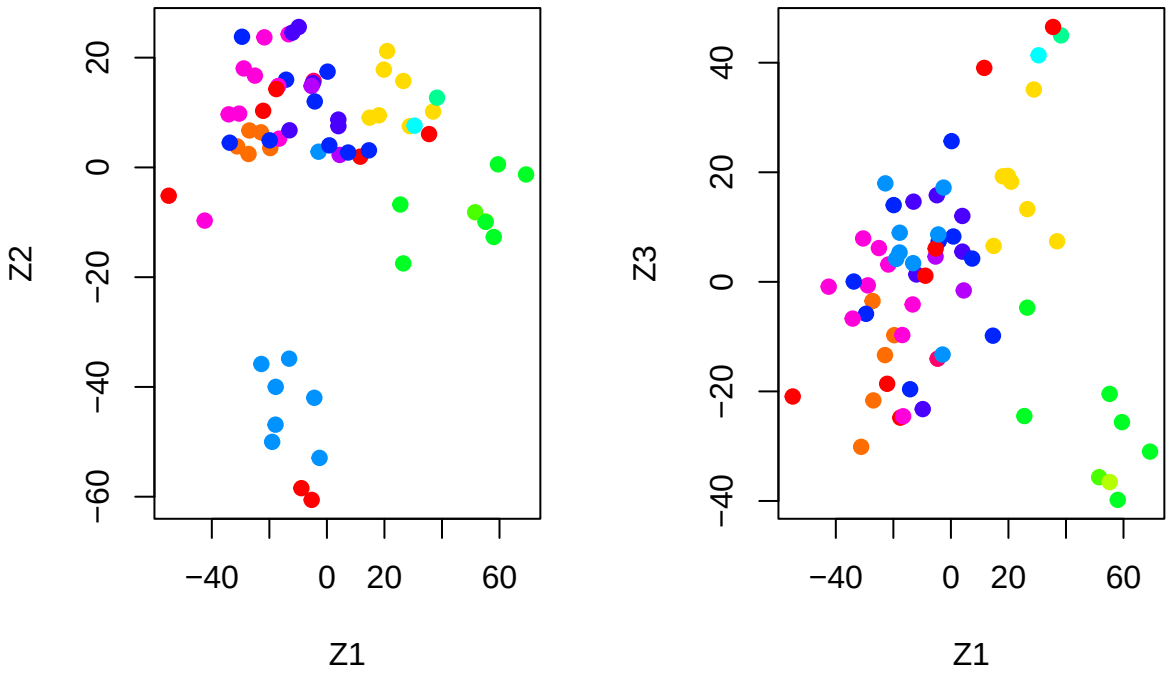
```
pr.out <- prcomp(nci.data , scale = TRUE)
```

We now plot the first few principal component score vectors, in order to visualize the data. The observations (cell lines) corresponding to a given cancer type will be plotted in the same color, so that we can see to what extent the observations within a cancer type are similar to each other. We first create a simple function that assigns a distinct color to each element of a numeric vector. The function will be used to assign a color to each of the 64 cell lines, based on the cancer type to which it corresponds.

```
Cols <- function(vec){
                      cols <- rainbow(length(unique(vec)))
                      return(cols[as.numeric(as.factor(vec))])
                     }
```

Note that the `rainbow()` function takes as its argument a positive integer, and returns a vector containing that number of distinct colors. We now can plot the principal component score vectors.

```
par(mfrow = c(1, 2))
plot(pr.out$x[, 1:2],
     col = Cols(nci.labs),
     pch = 19,
     xlab = "Z1",
     ylab = "Z2")
plot(pr.out$x[, c(1, 3)],
     col = Cols(nci.labs),
     pch = 19,
     xlab = "Z1",
     ylab = "Z3")
```




The resulting plots are shown in Figure 12.17. On the whole, cell lines corresponding to a single cancer type do tend to have similar values on the first few principal component score vectors. This indicates that cell lines from the same cancer type tend to have pretty similar gene expression levels.

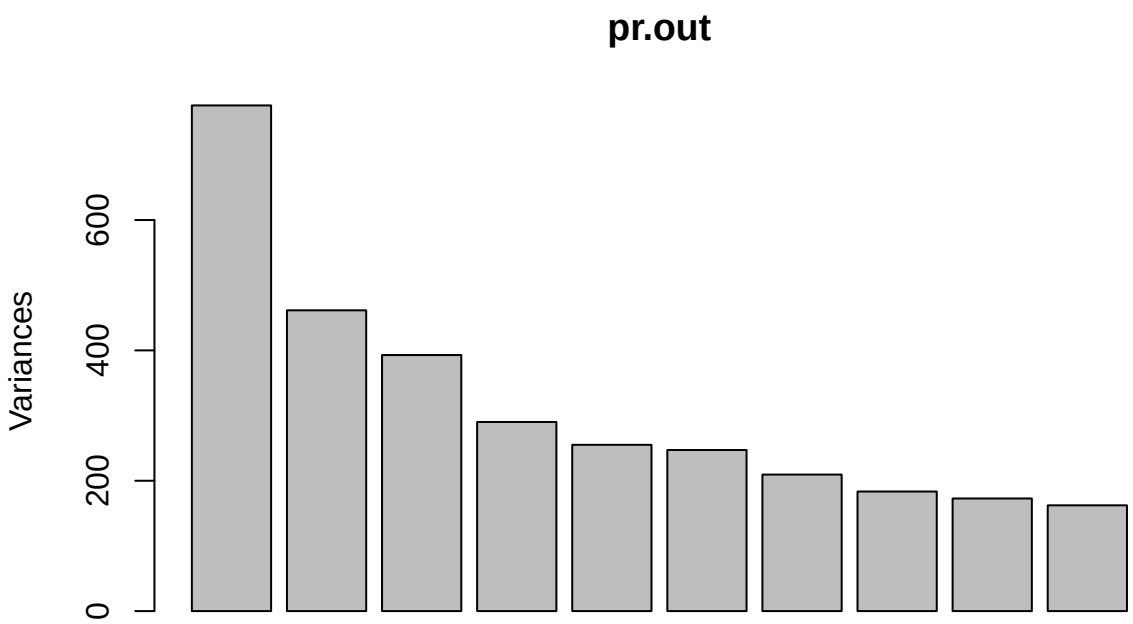
We can obtain a summary of the proportion of variance explained (PVE) of the first few principal components using the `summary()` method for a prcomp object (we have truncated the printout):

```
summary(pr.out)
```

```
## Importance of components:
##                             PC1      PC2      PC3      PC4      PC5      PC6
## Standard deviation     27.8535 21.48136 19.82046 17.03256 15.97181 15.72108
## Proportion of Variance  0.1136  0.06756  0.05752  0.04248  0.03735  0.03619
## Cumulative Proportion   0.1136  0.18115  0.23867  0.28115  0.31850  0.35468
##                             PC7      PC8      PC9     PC10     PC11     PC12
## Standard deviation     14.47145 13.54427 13.14400 12.73860 12.68672 12.15769
## Proportion of Variance  0.03066  0.02686  0.02529  0.02376  0.02357  0.02164
## Cumulative Proportion   0.38534  0.41220  0.43750  0.46126  0.48482  0.50646
##                            PC13     PC14     PC15     PC16     PC17     PC18
## Standard deviation     11.83019 11.62554 11.43779 11.00051 10.65666 10.48880
## Proportion of Variance  0.02049  0.01979  0.01915  0.01772  0.01663  0.01611
## Cumulative Proportion   0.52695  0.54674  0.56590  0.58361  0.60024  0.61635
##                            PC19    PC20     PC21    PC22    PC23    PC24
## Standard deviation     10.43518 10.3219 10.14608 10.0544 9.90265 9.64766
## Proportion of Variance  0.01594  0.0156  0.01507  0.0148 0.01436 0.01363
## Cumulative Proportion   0.63229  0.6479  0.66296  0.6778 0.69212 0.70575
##                            PC25    PC26    PC27   PC28    PC29    PC30    PC31
## Standard deviation     9.50764 9.33253 9.27320 9.0900 8.98117 8.75003 8.59962
## Proportion of Variance 0.01324 0.01275 0.01259 0.0121 0.01181 0.01121 0.01083
## Cumulative Proportion  0.71899 0.73174 0.74433 0.7564 0.76824 0.77945 0.79027
##                            PC32    PC33    PC34    PC35    PC36    PC37    PC38
## Standard deviation     8.44738 8.37305 8.21579 8.15731 7.97465 7.90446 7.82127
## Proportion of Variance 0.01045 0.01026 0.00988 0.00974 0.00931 0.00915 0.00896
## Cumulative Proportion  0.80072 0.81099 0.82087 0.83061 0.83992 0.84907 0.85803
##                            PC39    PC40    PC41   PC42    PC43   PC44    PC45
## Standard deviation     7.72156 7.58603 7.45619 7.3444 7.10449 7.0131 6.95839
## Proportion of Variance 0.00873 0.00843 0.00814 0.0079 0.00739 0.0072 0.00709
## Cumulative Proportion  0.86676 0.87518 0.88332 0.8912 0.89861 0.9058 0.91290
##                           PC46    PC47    PC48    PC49    PC50    PC51    PC52
## Standard deviation     6.8663 6.80744 6.64763 6.61607 6.40793 6.21984 6.20326
## Proportion of Variance 0.0069 0.00678 0.00647 0.00641 0.00601 0.00566 0.00563
## Cumulative Proportion  0.9198 0.92659 0.93306 0.93947 0.94548 0.95114 0.95678
##                            PC53    PC54    PC55    PC56    PC57   PC58    PC59
## Standard deviation     6.06706 5.91805 5.91233 5.73539 5.47261 5.2921 5.02117
## Proportion of Variance 0.00539 0.00513 0.00512 0.00482 0.00438 0.0041 0.00369
## Cumulative Proportion  0.96216 0.96729 0.97241 0.97723 0.98161 0.9857 0.98940
##                            PC60    PC61    PC62    PC63      PC64
## Standard deviation     4.68398 4.17567 4.08212 4.04124 1.951e-14
## Proportion of Variance 0.00321 0.00255 0.00244 0.00239 0.000e+00
## Cumulative Proportion  0.99262 0.99517 0.99761 1.00000 1.000e+00
```

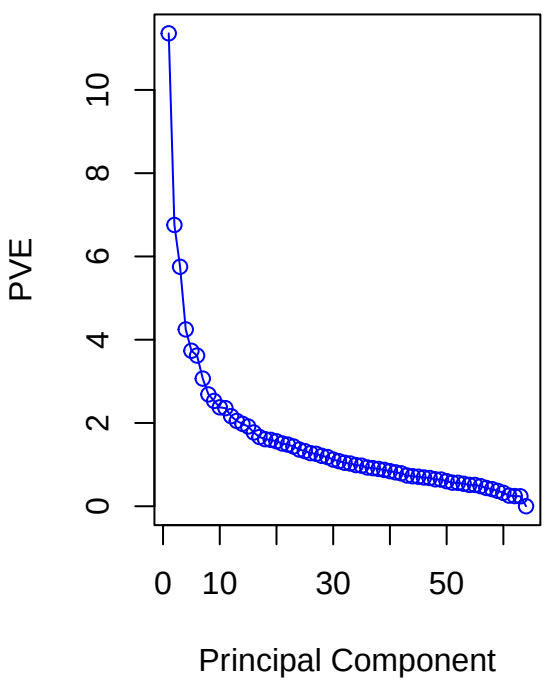
Using the `plot()` function, we can also plot the variance explained by the first few principal components.

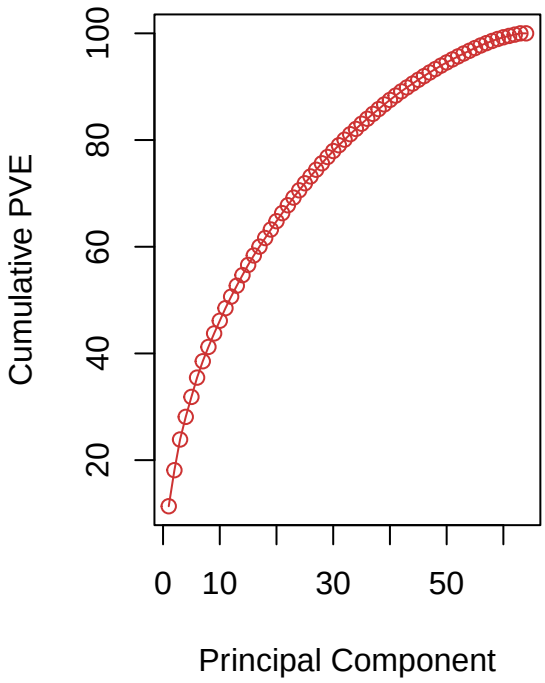
```
plot(pr.out)
```




Note that the height of each bar in the bar plot is given by squaring the corresponding element of `pr.out$sdev`. However, it is more informative to plot the PVE of each principal component (i.e. a scree plot) and the cu- mulative PVE of each principal component. This can be done with just a little work.

```
pve <- 100 * pr.out$sdev^2 / sum(pr.out$sdev ^2)
par(mfrow = c(1, 2))
plot(pve,
     type = "o",
     ylab = "PVE",
     xlab = "Principal Component",
     col = "blue")
plot(cumsum(pve),
     type = "o",
     ylab = "Cumulative PVE",
     xlab = "Principal Component",
     col = "brown3")
```




(Note that the elements of pve can also be computed directly from the summary, `summary(pr.out)$importance[2, ]`, and the elements of `cumsum(pve)` are given by `summary(pr.out)$importance[3, ]`.) The resulting plots are shown in Figure 12.18. We see that together, the first seven principal components explain around 40 % of the variance in the data. This is not a huge amount of the variance. However, looking at the scree plot, we see that while each of the first seven principal components explain a substantial amount of variance, there is a marked decrease in the variance explained by further principal components. That is, there is an elbow in the plot after approximately the seventh principal component. This suggests that there may be little benefit to examining more than seven or so principal components (though even examining seven principal components may be difficult).

### Clustering the Observations of the NCI60 Data

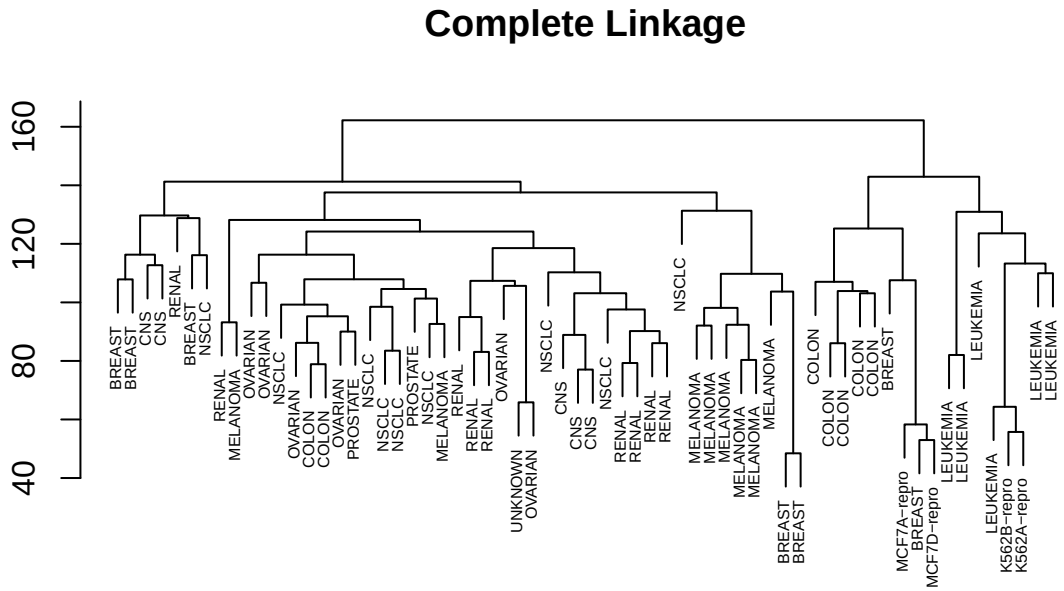
We now proceed to hierarchically cluster the cell lines in the NCI60 data, with the goal of finding out whether or not the observations cluster into distinct types of cancer. To begin, we standardize the variables to have mean zero and standard deviation one. As mentioned earlier, this step is optional and should be performed only if we want each gene to be on the same scale.

```
sd.data <- scale(nci.data)
```

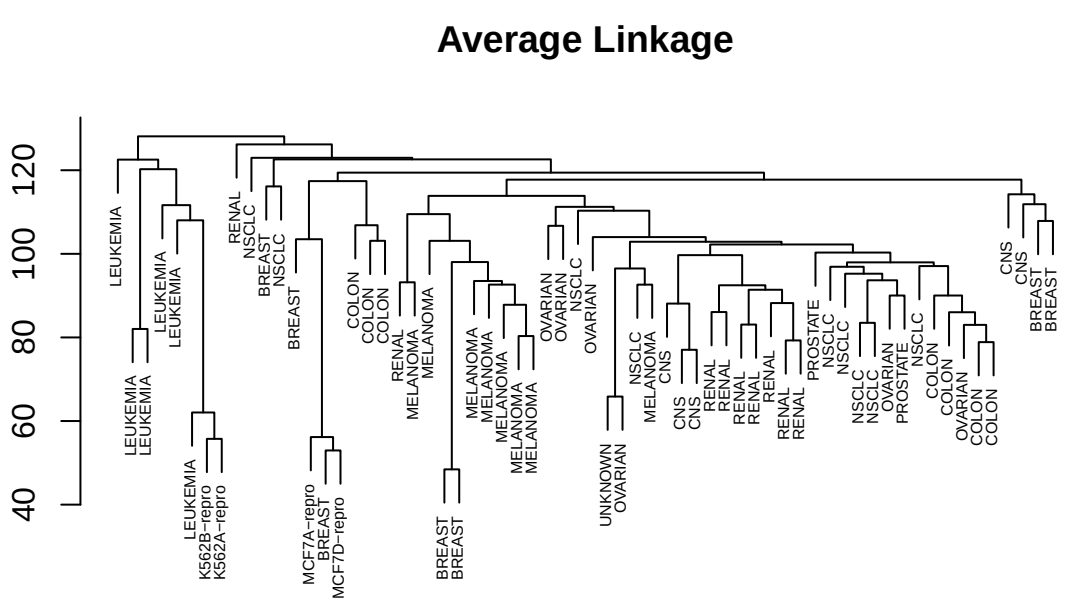
We now perform hierarchical clustering of the observations using complete, single, and average linkage. Euclidean distance is used as the dissimilarity measure.

```
#par(mfrow = c(1, 3))
data.dist <- dist(sd.data)
plot(hclust(data.dist), #complete linkage is default
```

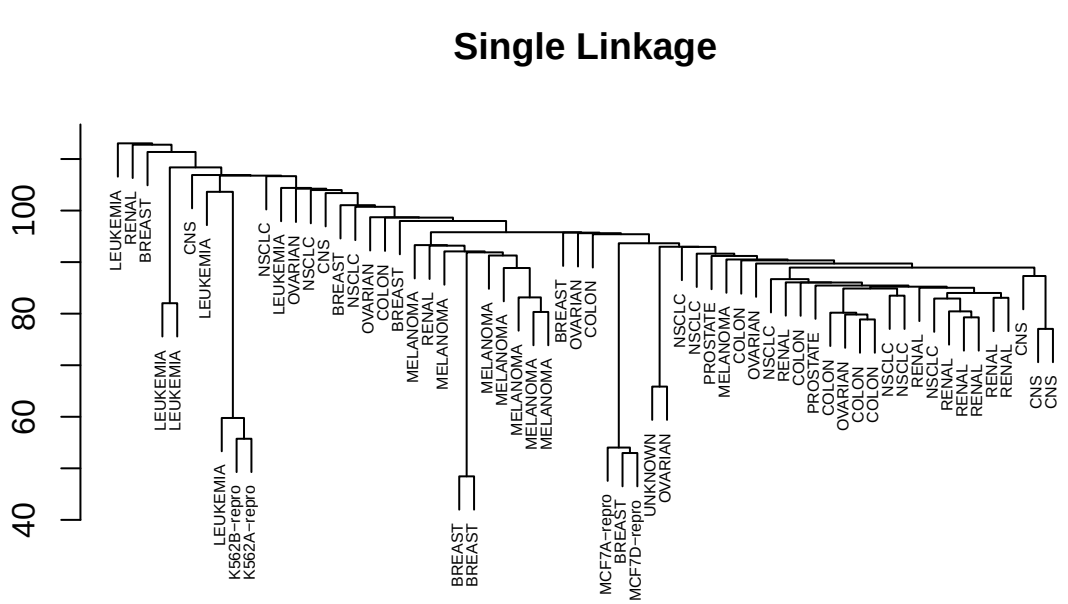
```
     xlab = "",
     sub = "",
     ylab = "",
     labels = nci.labs,
     main = "Complete Linkage",
     cex=0.5)
```




```
plot(hclust(data.dist, method = "average"),
             labels = nci.labs,
             main = "Average Linkage",
             xlab = "",
             sub = "",
             ylab = "",
             cex=0.5)
```




```r
plot(hclust(data.dist, method = "single"),
             labels = nci.labs,
             main = "Single Linkage",
             xlab = "",
             sub = "",
             ylab = "",
             cex=0.5)
```




The results are shown in Figure 12.19. We see that the choice of linkage certainly does affect the results obtained. Typically, single linkage will tend to yield trailing clusters: very large clusters onto which individual observa- tions attach one-by-one. On the other hand, complete and average linkage tend to yield more balanced, attractive clusters. For this reason, complete and average linkage are generally preferred to single linkage. Clearly cell lines within a single cancer type do tend to cluster together, although the clustering is not perfect. We will use complete linkage hierarchical cluster- ing for the analysis that follows.

We can cut the dendrogram at the height that will yield a particular number of clusters, say four:

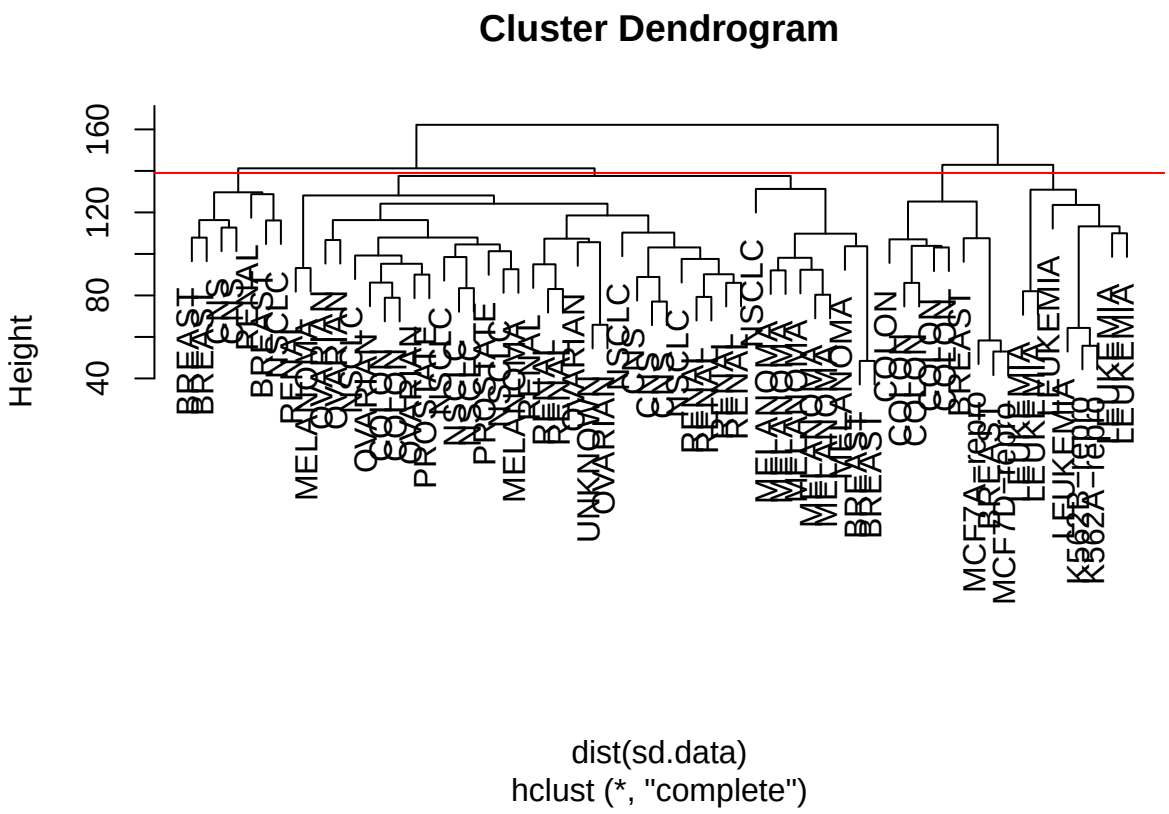
```
hc.out <- hclust(dist(sd.data))
hc.clusters <- cutree(hc.out , 4)
table(hc.clusters , nci.labs)
```

```
##             nci.labs
## hc.clusters BREAST CNS COLON K562A-repro K562B-repro LEUKEMIA MCF7A-repro
##            1      2   3     2           0           0        0           0
##            2      3   2     0           0           0        0           0
##            3      0   0     0           1           1        6           0
##            4      2   0     5           0           0        0           1
##             nci.labs
## hc.clusters MCF7D-repro MELANOMA NSCLC OVARIAN PROSTATE RENAL UNKNOWN
##            1           0        8     8       6        2     8       1
##            2           0        0     1       0        0     1       0
##            3           0        0     0       0        0     0       0
##            4           1        0     0       0        0     0       0
```

There are some clear patterns. All the leukemia cell lines fall in cluster 3, while the breast cancer cell lines

are spread out over three different clusters. We can plot the cut on the dendrogram that produces these four clusters:

```
par(mfrow = c(1, 1))
plot(hc.out , labels = nci.labs)
abline(h = 139, col = "red")
```




The `abline()` function draws a straight line on top of any existing plot in R. The argument h = 139 plots a horizontal line at height 139 on the dendrogram; this is the height that results in four distinct clusters. It is easy to verify that the resulting clusters are the same as the ones we obtained using cutree(hc.out, 4).

Printing the output of hclust gives a useful brief summary of the object:

```
hc.out
```

```
##
## Call:
## hclust(d = dist(sd.data))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 64
```

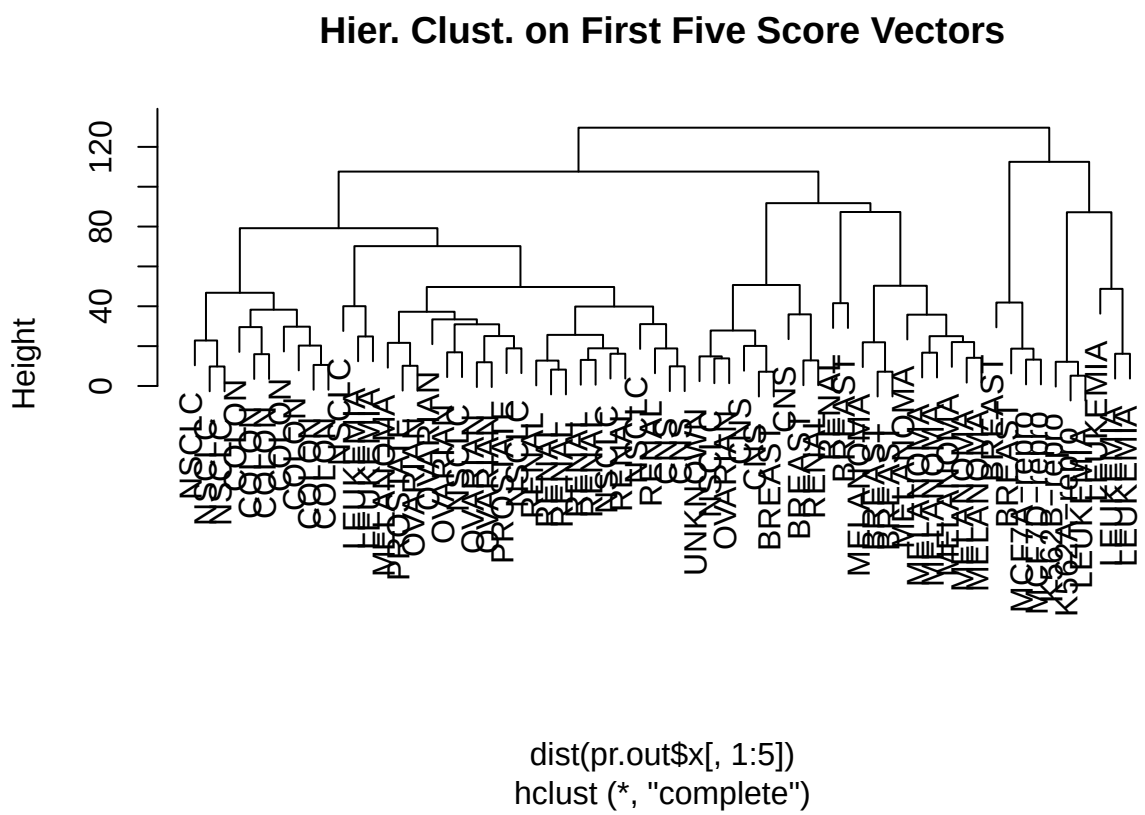
We claimed earlier in Section 12.4.2 that K-means clustering and hier- archical clustering with the dendrogram cut to obtain the same number of clusters can yield very different results. How do these NCI60 hierarchical clustering results compare to what we get if we perform K-means clustering with K = 4?

```r
set.seed (2)
km.out <- kmeans(sd.data , 4, nstart = 20)
km.clusters <- km.out$cluster
table(km.clusters , hc.clusters)
```

```
##             hc.clusters
## km.clusters  1  2  3  4
##            1 11  0  0  9
##            2 20  7  0  0
##            3  9  0  0  0
##            4  0  0  8  0
```

We see that the four clusters obtained using hierarchical clustering and K- means clustering are somewhat different. Cluster 4 in K-means clustering is identical to cluster 3 in hierarchical clustering. However, the other clusters differ: for instance, cluster 2 in K-means clustering contains a portion of the observations assigned to cluster 1 by hierarchical clustering, as well as all of the observations assigned to cluster 2 by hierarchical clustering. Rather than performing hierarchical clustering on the entire data matrix, we can simply perform hierarchical clustering on the first few principal component score vectors, as follows:

```r
hc.out <- hclust(dist(pr.out$x[, 1:5]))
plot(hc.out , labels = nci.labs ,
main = "Hier. Clust. on First Five Score Vectors")
```



```
table(cutree(hc.out , 4), nci.labs)
```

```
##    nci.labs
##     BREAST CNS COLON K562A-repro K562B-repro LEUKEMIA MCF7A-repro MCF7D-repro
##   1      0   2     7           0           0        2           0           0
##   2      5   3     0           0           0        0           0           0
##   3      0   0     0           1           1        4           0           0
##   4      2   0     0           0           0        0           1           1
##    nci.labs
##     MELANOMA NSCLC OVARIAN PROSTATE RENAL UNKNOWN
##   1        1     8       5        2     7       0
##   2        7     1       1        0     2       1
##   3        0     0       0        0     0       0
##   4        0     0       0        0     0       0
```

Not surprisingly, these results are different from the ones that we obtained when we performed hierarchical clustering on the full data set. Sometimes performing clustering on the first few principal component score vectors can give better results than performing clustering on the full data. In this situation, we might view the principal component step as one of denois- ing the data. We could also perform K-means clustering on the first few principal component score vectors rather than the full data set.

# Questions

## Exercise 8, Chapter 12

In Section 12.2.3, a formula for calculating the *proportion of variance explained* (PVE) was given in Equation 12.10. We also saw that the PVE can be obtained using the sdev output of the `prcomp()` function.

On the USArrests data, calculate PVE in two ways:

a. Using the sdev output of the `prcomp()` function, as was done in Section 12.2.3.

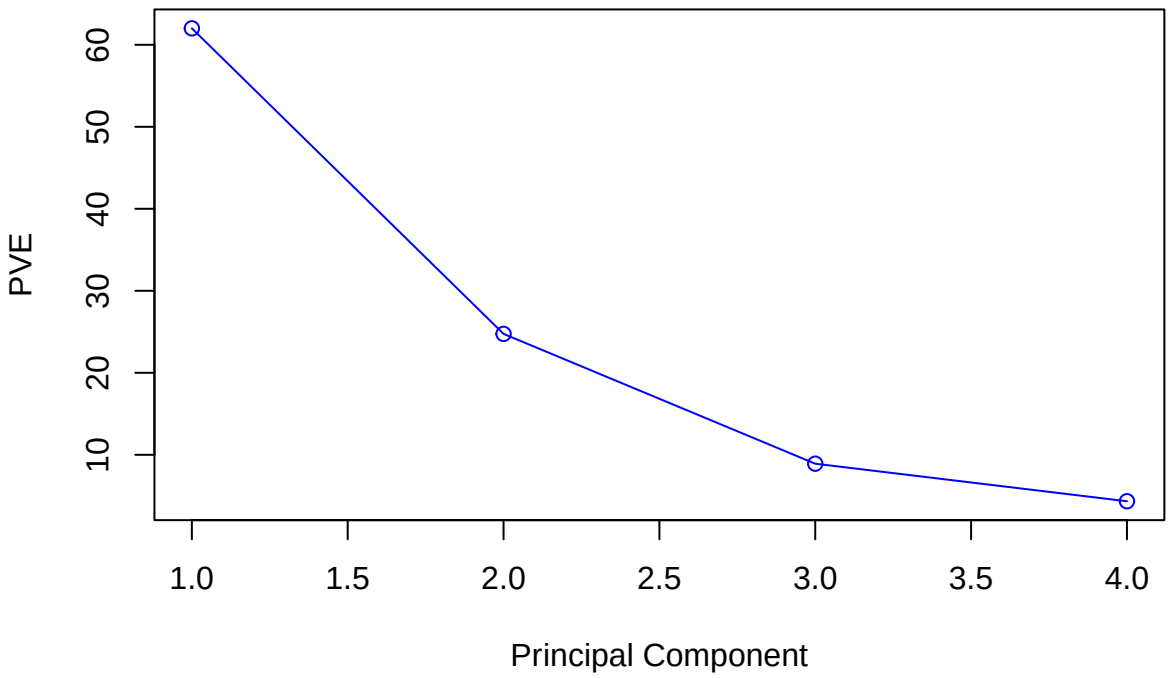
```
#load data
#install.packages('ISLR') #note: may need to install.packages in the console!
library(ISLR)
data("USArrests")

# Note: The variance explained by each principal component:
# pr.var = pr.out$sdev^2

pr.out = prcomp(USArrests, scale = TRUE)
pve =100*pr.out$sdev^2 / sum(pr.out$sdev^2)
pve
```

```
## [1] 62.006039 24.744129  8.914080  4.335752
```

```
plot(pve,
     type = "o",
     ylab = "PVE",
     xlab = "Principal Component",
     col = "blue")
```




b. By applying Equation 12.10 directly. That is, use the `prcomp()` function to compute the principal component loadings. Then, use those loadings in Equation 12.10 to obtain the PVE.

$$\frac{\sum_{i=1}^{n} z_{im}^2}{\sum_{j=1}^{p}\sum_{i=1}^{n} x_{ij}^2} = \frac{\sum_{i=1}^{n}\left(\sum_{j=1}^{p} \phi_{jm} x_{ij}\right)^2}{\sum_{j=1}^{p}\sum_{i=1}^{n} x_{ij}^2}. \tag{12.10}$$

```
# Obtain loadings from prcomp() function
# The loadings make up the principal component loading vector
loadings <- pr.out$rotation
# Scale dataset just to make sure the data we use is consistent
USArrests2 <- scale(USArrests)

# Convert dataset into matrix, square each value in matrix, and sum them up
# to get the denominator of the equation
sumvalue <- sum(as.matrix(USArrests2)^2)

# Multiply these two matrices and then square
# Note: - "%*%" is matrix multiplication
#       - xij is "as.matrix(USArrests2)"
#       - phijm is "loadings"
#       - only works as xij times phijm (not interchangeable,
#         results in non-conformable arguments error if written as
#         "loadings %*% as.matrix(USArrests2)")
```

```
num <- (as.matrix(USArrests2)%*%loadings)^2

#calculate the column value for num matrix
colvalue<-c()
for (i in 1:length(num[1,])){
  colvalue[i]<-sum(num[,i])
}
#calculate new pve
pve <- 100*colvalue/sumvalue
pve
```

```
## [1] 62.006039 24.744129  8.914080  4.335752
```

These two approaches should give the same results.

*Hint: You will only obtain the same results in (a) and (b) if the same data is used in both cases. For instance, if in (a) you performed* `prcomp()` *using centered and scaled variables, then you must center and scale the variables before applying Equation 12.10 in (b).*

## Sparrows

Download the `sparrow2.csv` data from LearnJCU. This dataset consists of seven morphological variables taken from 1026 sparrows. The seven variables were:

- wingcrd = wingcord
- flating = flattened wing
- tarsus = leg
- head = bill tip to back of skull
- culmen = beak length
- nalopsi = bill tip to nostril
- weight

Conduct exploratory data analysis to check if there is correlation between variables.

```
Sparrows2 <- read.csv("Sparrows2.csv")
dim(Sparrows2)
```

```
## [1] 1026    7
```

```
variables <- names(Sparrows2)
variables
```

```
## [1] "wingcrd"  "flatwing" "tarsus"   "head"     "culmen"   "nalospi"  "wt"
```

We first briefly examine the data. We notice that the variables have vastly different means.

```
apply(Sparrows2, 2, mean)
```

```
##   wingcrd  flatwing    tarsus      head    culmen   nalospi        wt
## 57.877485 58.947173 21.462573 32.032846 13.157505  9.662865 20.195322
```

We can also examine the variances of the four variables using the apply() function.

```
apply(Sparrows2, 2, var)
```

```
##   wingcrd  flatwing    tarsus      head    culmen   nalospi        wt
## 5.2453560 5.5470018 0.8460905 0.9139933 0.6012948 0.4615368 3.1134220
```

We now perform principal components analysis using the prcomp() function, which is one of several functions in R that perform PCA. It is important to standardize the variables to have mean zero and standard deviation one before performing PCA. By default, the prcomp() function centers the variables to have mean zero. By using the option scale = TRUE, we scale the variables to have standard deviation one. The output from prcomp() contains a number of useful quantities.

```
pr.out <- prcomp(Sparrows2, scale = TRUE)
names(pr.out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

The center and scale components correspond to the means and standard deviations of the variables that were used for scaling prior to implementing PCA.

```
pr.out$center
```

```
##   wingcrd  flatwing    tarsus      head    culmen   nalospi        wt
## 57.877485 58.947173 21.462573 32.032846 13.157505  9.662865 20.195322
```

```
pr.out$scale
```

```
##   wingcrd  flatwing    tarsus      head    culmen   nalospi        wt
## 2.2902742 2.3552074 0.9198318 0.9560299 0.7754320 0.6793650 1.7644892
```

\textcolor{red}{The rotation matrix provides the principal component loadings; each column of pr.out$rotation contains the corresponding principal component loading vector.}

```
# This function names it the rotation matrix, because when we matrix-multiply the
# X matrix by pr.out$rotation, it gives us the coordinates of the data in the rotated
# coordinate system. These coordinates are the principal component scores.
pr.out$rotation
```

```
##                 PC1        PC2         PC3         PC4         PC5         PC6
## wingcrd  -0.3791495  0.5173084  0.27205588 -0.08004070 -0.07302830 -0.03442239
## flatwing -0.3787484  0.5171277  0.27721466 -0.08131335 -0.08270043 -0.02509085
## tarsus   -0.3850118 -0.1182589 -0.56119623 -0.68895005 -0.10335807 -0.19364930
## head     -0.4031232 -0.2698651  0.02880689  0.01504806 -0.07720170  0.87040720
## culmen   -0.3615164 -0.3931759  0.39028956 -0.08732971  0.70572242 -0.23811257
## nalospi  -0.3551271 -0.4438757  0.21375739  0.30004726 -0.63536797 -0.37072959
## wt       -0.3811083  0.1629282 -0.57909276  0.64377649  0.26352974 -0.09454517
##                  PC7
## wingcrd   0.708306951
## flatwing -0.705832698
```

```
## tarsus   -0.002092217
## head      0.006283316
## culmen   -0.004948015
## nalospi   0.002365069
## wt       -0.005247099
```

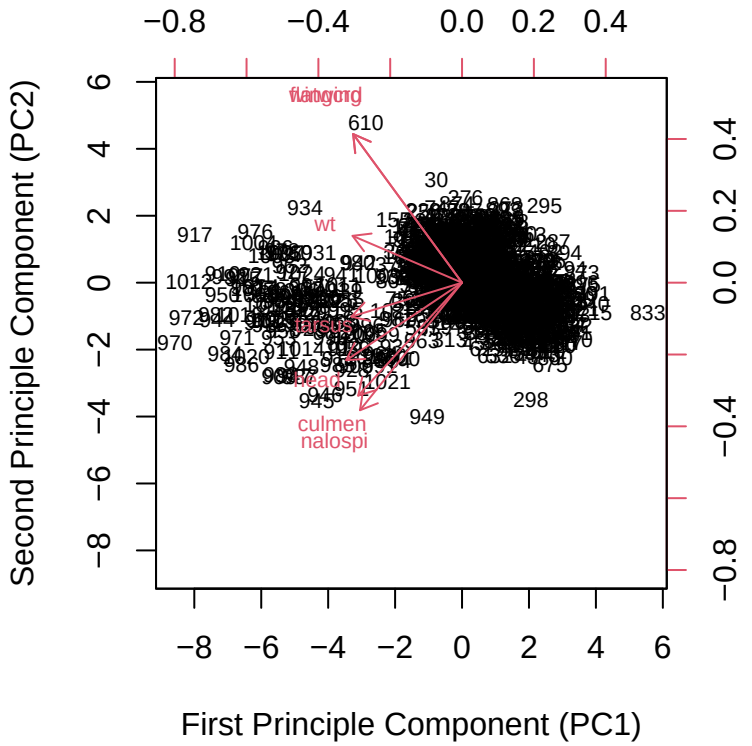
We see that there are seven distinct principal components. This is to be expected because there are in general min(n - 1, p) informative principal components in a data set with n observations and p variables. Using the prcomp() function, we do not need to explicitly multiply the data by the principal component loading vectors in order to obtain the principal component score vectors. Rather the 1026 × 7 matrix x has as its columns the principal component score vectors. That is, the kth column is the kth principal component score vector.

```
dim(pr.out$x)
```

```
## [1] 1026    7
```

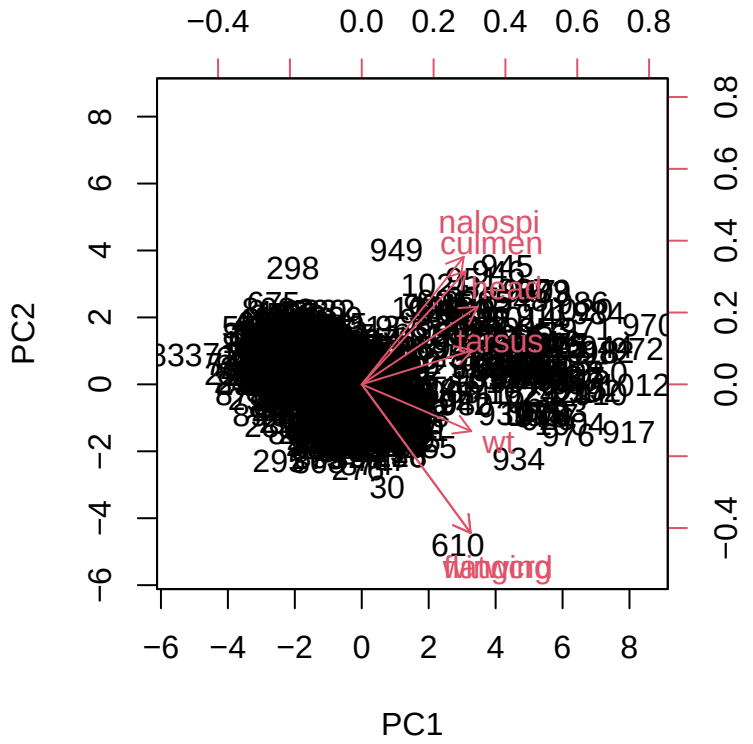
We can plot the first two principal components as follows with the biplot() function:

```
biplot(pr.out,
       scale = 0,
       cex = 0.7,
       xlab="First Principle Component (PC1)",
       ylab="Second Principle Component (PC2)")
```




We want the arrow things to be in the positive areas. SO let's redo it.

```
pr.out$rotation = -pr.out$rotation
pr.out$x = -pr.out$x
biplot(pr.out, scale = 0)
```




The black numbers represent the scores for the first two principal components. The orange arrows indicate the first two principal component loading vectors (with axes on the top and right).

Here we can compute the proportion of variance explained by each principal component:

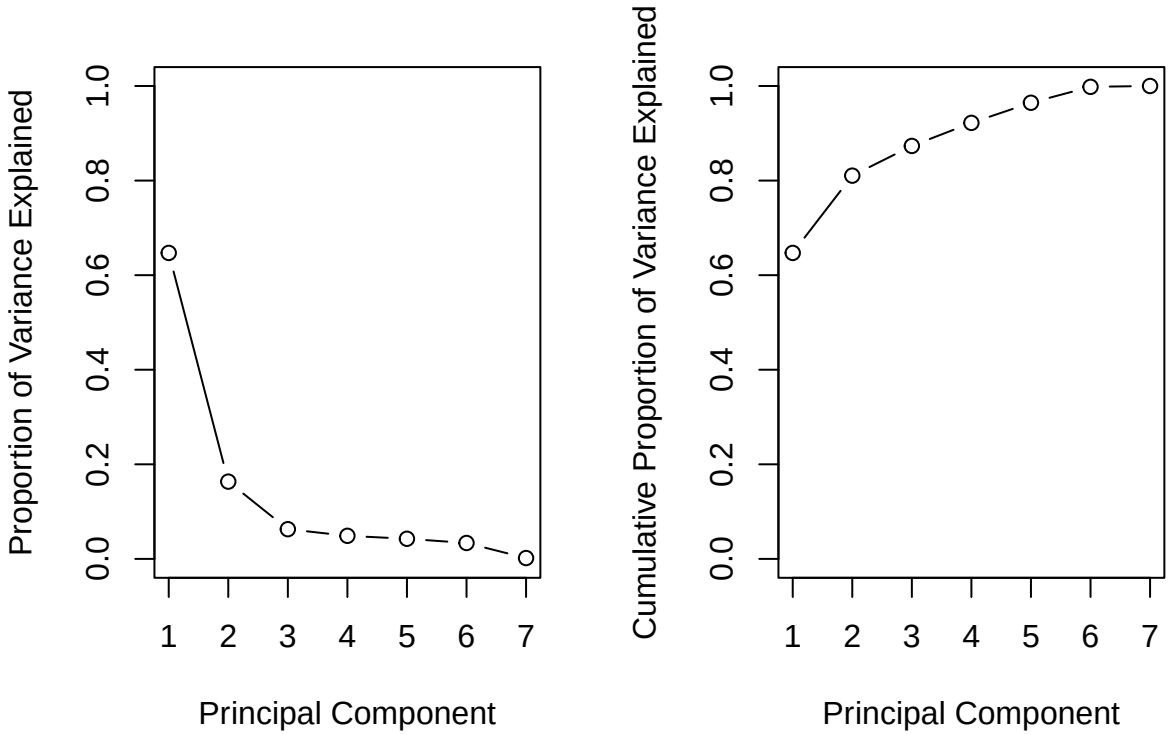
```
pr.var <- pr.out$sdev^2
pve <- pr.var / sum(pr.var)
pve
```

```
## [1] 0.647031871 0.163418844 0.062806010 0.048800235 0.042535482 0.033646681
## [7] 0.001760876
```

We see that the first principal component explains 64.7 as well as the cumulative PVE, as follows:

```
par(mfrow = c(1, 2))
plot(pve, xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     ylim = c(0, 1),
     type = "b")
plot(cumsum(pve), xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
```

```
     ylim = c(0, 1),
     type = "b")
```




However, looking at the scree plot, we see that while each of the first three principal components explain a substantial amount of variance, there is a marked decrease in the variance explained by further principal components. That is, there is an elbow in the plot after approximately the third principal component. This suggests that there may be little benefit to examining more than three or so principal components.

Perform PCA and answer the following questions:

1. How much variation is explained in the (i) first (ii) and (iii) principal component analysis?

\textcolor{red}{The first principal component explains 64.7 % of the variance in the data, the next principal component explains 16.3 % of the variance, and the third principal component explains 6.3% of the variance.}

2. How many principal components do you recommend using? Why?

Looking at the scree plot, we see that while each of the first three principal components explain a substantial amount of variance, there is a marked decrease in the variance explained by further principal components. That is, there is an elbow in the plot after approximately the third principal component. This suggests that there may be little benefit to examining more than three or so principal components.

3. Can you describe the first two principal components?

We see that the first loading vector places approximately equal weight on bill tip to back of skull, beak length and bill tip to nostril, but with much less weight on flattened wing and wingcord. Hence this component roughly corresponds to a measure of beak/bill-related variables. The second loading vector places most of its weight on the wing-related features much less weight on the other three features. Hence, this component roughly corresponds to measures of wing-related variables. Overall, we see that the wing-related variables are located close to each other, and that the beak/bill-related variables are far from the others.

4. Interpret any interesting features in the biplot.

The biplot (This figure is known as a biplot, because it displays both the principal component scores and the principal component loadings) indicates that the two wing-related variables, flattened wing and wingcord are strongly correlated with each other. On the other hand, the beak/bill-related variables (bill tip to back of skull, beak length and bill tip to nostril) are correlated with each other too.