Appendix A
Vectors, Matrices, and Lists

A.1 Vectors

A.1.1 Creating a vector

Vectors can be created in several ways. We have already seen two methods, the combine function `c` and the colon operator `:` for creating vectors with given values. For example, to create a vector of integers 1 through 9 and assign it to `i`, we can use either of the following.

```
i = c(1,2,3,4,5,6,7,8,9)
i = 1:9
```

To create a vector without specifying any of its elements, we only need to specify the type of vector and its length. For example,

```
y = numeric(10)
```

creates a numeric vector `y` of length 100, and

```
a = character(4)
```

creates a vector of 4 empty character strings. The vectors created are

```
> y
 [1] 0 0 0 0 0 0 0 0 0 0
> a
[1] "" "" "" ""
```

A.1.2 Sequences

There are several functions in R that generate sequences of special types and patterns. In addition to `:`, there is a sequence function `seq` and a repeat `rep` function. We often use these functions to create vectors.

J. Albert and M. Rizzo, *R by Example*, Use R, DOI 10.1007/978-1-4614-1365-3,

© Springer Science+Business Media, LLC 2012

The `seq` function generates 'regular' sequences that are not necessarily integers. The basic syntax is `seq(from, to, by)`, where `by` is the size of the increment. Instead of `by`, we could specify the `length` of the sequence. To create a sequence of numbers $0, 0.1, 0.2, \ldots, 1$, which has 11 regularly spaced elements, either command below works.

```
> seq(0, 1, .1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(0, 1, length=11)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

The `rep` function generates vectors by repeating a given pattern a given number of times. The `rep` function is easily explained by the results of a few examples:

```
> rep(1, 5)
[1] 1 1 1 1 1

> rep(1:2, 5)
[1] 1 2 1 2 1 2 1 2 1 2

> rep(1:3, times=1:3)
[1] 1 2 2 3 3 3

> rep(1:2, each=2)
[1] 1 1 2 2

> rep(c("a", "b", "c"), 2)
[1] "a" "b" "c" "a" "b" "c"
```

A.1.3 Extracting and replacing elements of vectors

If `x` is a vector, `x[i]` is the i^{th} element of `x`. This syntax is used both for assignment and extracting values. If `i` happens to be a vector of positive integers $(i_1, \ldots, i_k)$, then `x[i]` is the vector containing $x_{i_1}, \ldots, x_{i_k}$, provided these are valid indices for `x`.

A few examples to illustrate extracting elements from a vector are:

```
> x = letters[1:8]   #letters of the alphabet a to h
> x[4]               #fourth element
[1] "d"

> x[4:5]             #elements 4 to 5
[1] "d" "e"

> i = c(1, 5, 7)
> x[i]               #elements 1, 5, 7
[1] "a" "e" "g"
```

Examples illustrating assignment are:

```
> x = seq(0, 20, 2)
> x[4] = NA          #assigns a missing value
> x
 [1]  0  2  4 NA  8 10 12 14 16 18 20

> x[4:5] = c(6, 7)    #assigns two values
> x
 [1]  0  2  4  6  7 10 12 14 16 18 20

> i = c(3, 5, 7)
> x[i] = 0       #assigns 3 zeros, at positions i
> x
 [1]  0  2  0  6  0 10  0 14 16 18 20
```

Sometimes it is easier to specify what to "leave out" rather than what to include. In this case, we can use negative indices. An expression `x[-2]`, for example, is evaluated as all elements of `x` *except* the second one. The negative sign can also be used with a vector of indices, as shown below.

```
> x = seq(0, 20, 5)
> x
[1]  0  5 10 15 20

> i = c(1, 5)
> y = x[-i]         #leave out the first and last element
> y
[1]  5 10 15
```

A.2 The `sort` and `order` functions

The `sort` function sorts the values of a vector in ascending order (by default) or descending order. At times, one wants to sort several variables according to the order of one of the variables. This can be accomplished with the `order` function.

Example A.1 (The `order` function). Suppose that we have the following data for temperatures and ozone levels

```
> temps = c(67, 72, 74, 62)
> ozone = c(41, 36, 12, 18)
```

and wish to sort the pairs (ozone, temps) in increasing order of ozone. The expression `order(ozone)` is a vector of indices that can be used to rearrange ozone into ascending order. This is the same order required for `temps`, so this order is what we need as an index vector for `temps`.

```
> oo = order(ozone)
> oo
[1] 3 4 2 1

> Ozone = sort(ozone)   #same as ozone[oo]
```

```
> Temps = temps[oo]

> Ozone
[1] 12 18 36 41
> Temps
[1] 74 62 72 67
```

R$_x$ A.1 *In Example A.1 we used* `sort` *to sort the values of* `ozone`*; however, it is not really necessary to sort (again) because* `order(ozone)` *contains the information for sorting* `ozone`*. In this example,* `sort(ozone)` *is equivalent to* `ozone[oo]`*.*

See Section 2.7.3 for an example of sorting a data frame.

A.3 Matrices

A.3.1 Creating a matrix

A matrix can be created using the `matrix` function. The basic syntax is `matrix(x, nrow, ncol)`, where `x` is a constant or a vector of values, `nrow` is the number of rows and `ncol` is the number of columns. For example, to create a 4 by 4 matrix of 0's, we use

```
> matrix(0, 4, 4)
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
[4,]    0    0    0    0
```

To create a matrix with specified elements, supply those elements in a vector as the first argument to `matrix`.

```
> X = matrix(1:16, 4, 4)
> X
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

The number of rows, number of columns, and dimension of a matrix that is already in the R workspace is returned by the functions `nrow` (or `NROW`), `ncol`, and `dim`, respectively.

```
> nrow(X)
[1] 4
> NROW(X)
[1] 4
> ncol(X)
```

```
[1] 4
> dim(X)
[1] 4 4
```

R_x A.2 (NROW, nrow, length) *It is helpful to know when to use* ***NROW****,* ***nrow****, or* ***length****.* ***length*** *gives the length of a vector, and* ***nrow*** *gives the number of rows of a matrix. But* ***length*** *applied to a matrix does not return the number of rows, but rather, the number of entries in the matrix. An advantage of* ***NROW*** *is that it computes* ***length*** *for vectors and* ***nrow*** *for matrices. This is helpful when the object could be either a vector or a matrix.*

Notice that when we supply the vector `x=1:16` as the entries of the matrix `X`, the matrix was filled in column by column. We can change this pattern with the optional argument `byrow=TRUE`.

```
> matrix(1:16, 4, 4)
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16

> A = matrix(1:16, 4, 4, byrow=TRUE)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
```

The row and column labels of the matrix also indicate how to extract each row or column from the matrix. To extract all of row 2 we use `A[2,]`. To extract all of column 4 we use `A[,4]`. To extract the element in row 2, column 4, we use `A[2, 4]`.

```
> A[2,]
[1] 5 6 7 8
> A[,4]
[1]  4  8 12 16
> A[2,4]
[1] 8
```

A submatrix can be extracted by specifying a vector of row indices and/or a vector of column indices. For example, to extract the submatrix with the last two rows and columns of `A` we use

```
> A[3:4, 3:4]
     [,1] [,2]
[1,]   11   12
[2,]   15   16
```

To omit a few rows or columns we can use negative indices, in the same way that we did for vectors. To omit just the third row, we would use

```
> A[-3, ]
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]   13   14   15   16
```

The rows and columns of matrices can be named using the `rownames` or `colnames`. For example, we can create names for `A` as follows.

```
> rownames(A) = letters[1:4]
> colnames(A) = c("FR", "SO", "JR", "SR")
> A
  FR SO JR SR
a  1  2  3  4
b  5  6  7  8
c  9 10 11 12
d 13 14 15 16
```

Now one can optionally extract elements by name. To extract the column labeled "JR" we could use `A[, 3]` or

```
> A[, "JR"]
 a  b  c  d
 3  7 11 15
```

A.3.2 Arithmetic on matrices

The basic arithmetic operators (`+ - * / ^`) on matrices apply the operations elementwise, analogous to vectorized operations. This means that if $A = (a_{ij})$ and $B = (b_{ij})$ are matrices with the same dimension, then `A*B` is a matrix of the products $a_{ij}b_{ij}$. Multiplying a matrix A above with itself using the `*` operator squares every element of the matrix.

```
> A = matrix(1:16, 4, 4, byrow=TRUE)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
> A * A
     [,1] [,2] [,3] [,4]
[1,]    1    4    9   16
[2,]   25   36   49   64
[3,]   81  100  121  144
[4,]  169  196  225  256
```

The exponentiation operator is also applied to each entry of a matrix. The R expression `A^2` is evaluated as the matrix of squared elements a_{ij}^2, not the matrix product AA.

```
> A^2      #not the matrix product
     [,1] [,2] [,3] [,4]
[1,]    1    4    9   16
[2,]   25   36   49   64
[3,]   81  100  121  144
[4,]  169  196  225  256
```

Matrix multiplication is obtained by the operator `%*%`. To obtain the square of matrix `A` (using matrix multiplication) we need `A %*% A`.

```
> A %*% A  #the matrix product
     [,1] [,2] [,3] [,4]
[1,]   90  100  110  120
[2,]  202  228  254  280
[3,]  314  356  398  440
[4,]  426  484  542  600
```

Many of the one-variable functions in R will be applied to individual elements of a matrix, also. For example, `log(A)` returns a matrix with the natural logarithm $log(a_{ij})$ as its entries.

```
> log(A)
         [,1]      [,2]     [,3]     [,4]
[1,] 0.000000 0.6931472 1.098612 1.386294
[2,] 1.609438 1.7917595 1.945910 2.079442
[3,] 2.197225 2.3025851 2.397895 2.484907
[4,] 2.564949 2.6390573 2.708050 2.772589
```

The `apply` function can be used to *apply* a function to rows or columns of a matrix. For example, we obtain the vector of column minimums and the vector of column maximums of `A` by

```
> apply(A, MARGIN=1, FUN="min")
[1]  1  5  9 13

> apply(A, MARGIN=2, FUN="max")
[1] 13 14 15 16
```

Row means and column means can be computed using `apply` or by

```
> rowMeans(A)
[1]  2.5  6.5 10.5 14.5

> colMeans(A)
[1]  7  8  9 10
```

The `sweep` function can be used to *sweep* out a statistic from a matrix. For example, to subtract the minimum of the matrix and divide the result by its maximum we can use

```
> m = min(A)
> A1 = sweep(A, MARGIN=1:2, STATS=m, FUN="-")  #subtract min

> M = max(A1)
> A2 = sweep(A1, 1:2, M, "/")  #divide by max
```

```
> A2
          [,1]       [,2]      [,3]      [,4]
[1,] 0.0000000 0.06666667 0.1333333 0.2000000
[2,] 0.2666667 0.33333333 0.4000000 0.4666667
[3,] 0.5333333 0.60000000 0.6666667 0.7333333
[4,] 0.8000000 0.86666667 0.9333333 1.0000000
```

Here we specified `MARGIN=1:2` indicating all entries of the matrix. The default function is subtraction, so the `"-"` argument could have been omitted in the first application of `sweep`.

The column mean can be subtracted from each column by

```
> sweep(A, 2, colMeans(A))
     [,1] [,2] [,3] [,4]
[1,]   -6   -6   -6   -6
[2,]   -2   -2   -2   -2
[3,]    2    2    2    2
[4,]    6    6    6    6
```

Example 1.8 in Section 1.3 illustrates matrix operations.

A.4 Lists

One limitation of vectors, matrices, and arrays is that each of these types of objects may only contain one type of data. For example, a vector may contain all numeric data or all character data. A list is a special type of object that can contain data of multiple types.

Objects in a list can have names; if they do, they can be referenced using the `$` operator. Objects can also be referenced using double square brackets `[[ ]]` by name or by position.

Example A.2 (Creating a list). If the results of Example 1.3 (fatalities due to horsekicks) are not in the current workspace, run the script "horsekicks.R" discussed in Section 1.1.3. Now suppose that we would like to store the data (`k` and `x`), sample mean `r` and sample variance `v`, all in one object. The data are vectors `x` and `k`, both of length 5, but the mean and variance are each length 1. This data cannot be combined in a matrix, but it can be combined in a list. This type of list can be created as

```
> mylist = list(k=k, count=x, mean=r, var=v)
```

The contents of the list can be displayed like any other object, by typing the name of the object or by the `print` function.

```
> mylist
$k
[1] 0 1 2 3 4

$count
[1] 109  65  22   3   1
```

```
$mean
[1] 0.61

$var
[1] 0.6109548
```

Names of list components can be displayed by the `names` function.

```
> names(mylist)
[1] "k"     "count" "mean"  "var"
```

All of the components of this list have names, so they can be referenced by either name or position. Examples follow.

```
> mylist$count   #by name
[1] 109  65  22   3   1

> mylist[[2]]    #by position
[1] 109  65  22   3   1

> mylist["mean"] #by name
$mean
[1] 0.61
```

A compact way to describe the contents of a list is to use the `str` function. It is a general way to describe any object (`str` is an abbreviation for structure).

```
> str(mylist)
List of 4
 $ k    : num [1:5] 0 1 2 3 4
 $ count: num [1:5] 109 65 22 3 1
 $ mean : num 0.61
 $ var  : num 0.611
```

The `str` function is particularly convenient when the list is too large to read on one screen or the object is a large data set.

Many functions in R return values that are lists. An interesting example is the `hist` function, which displays a histogram; its return value (a list) is discussed in the next example.

Example A.3 (The histogram object (a list)). One of the many data sets included in the R installation is `faithful`. This data records waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. We are interested in the second variable `waiting`, the waiting time in minutes to the next eruption.

We construct a frequency histogram of the times between eruptions using the `hist` function, displayed in Figure A.1. The heights of the bars correspond to the counts for each of the bins. The histogram has an interesting shape. It is clearly not close to a normal distribution, and in fact it has two modes; one near 50-55, and the other near 80.

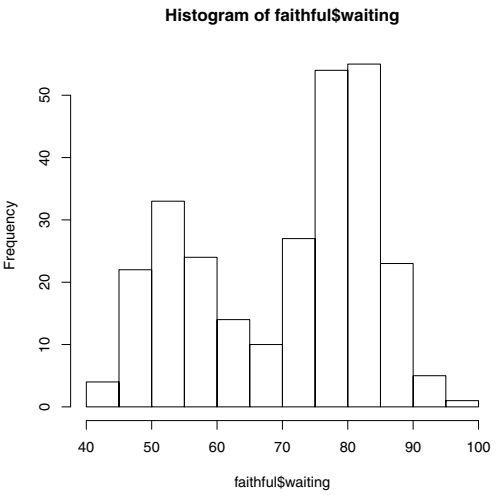

Fig. A.1 Histogram of waiting times between Old Faithful eruptions in Example A.3.

Typically one is only interested in the graphical output of the `hist` function, but some useful information is returned by the function. That information can be saved if we assign the value of the function to an object. Here we saved the result of `hist` in an object `H`. The object `H` is actually a list; it stores a variety of information about the histogram such as the bin counts, breaks (endpoints), midpoints of intervals, etc.

```
> H = hist(faithful$waiting)

> names(H)
[1] "breaks"      "counts"      "intensities" "density"
[5] "mids"        "xname"       "equidist"
```

The endpoints of the intervals (the *breaks*) are

```
> H$breaks
 [1]  40  45  50  55  60  65  70  75  80  85  90  95 100
```

The frequencies in each interval (the bin *counts*) are

```
H$counts
 [1]  4 22 33 24 14 10 27 54 55 23  5  1
```

The help topic for `hist` describes each of the components of the list `H` in the *Value* section.

To create a list, use the `list` function; a simple example is at the beginning of this section (Example A.2). There are several other examples throughout this book that illustrate how to create lists.

A.5 Sampling from a data frame

To draw a random sample of observations from a data frame, use the `sample` function to sample the row indices or the row labels, and extract these rows. Refer to the `USArrests` data introduced in Example 1.9 on page 20. To obtain a random sample of five states,

```
> i = sample(1:50, size=5, replace=FALSE)
> i
[1] 30  1  6 50  3
> USArrests[i, ]
           Murder Assault UrbanPop Rape
New Jersey    7.4     159       89 18.8
Alabama      13.2     236       58 21.2
Colorado      7.9     204       78 38.7
Wyoming       6.8     161       60 15.6
Arizona       8.1     294       80 31.0
```

Alternately, we could have sampled the row labels:

```
> samplerows = sample(rownames(USArrests), size=5, replace=FALSE)
> samplerows
[1] "North Dakota"  "West Virginia" "Montana"       "Idaho"
[5] "Virginia"
> USArrests[samplerows, ]
              Murder Assault UrbanPop Rape
North Dakota     0.8      45       44  7.3
West Virginia    5.7      81       39  9.3
Montana          6.0     109       53 16.4
Idaho            2.6     120       54 14.2
Virginia         8.5     156       63 20.7
```

References

1. Albert, J. (2009), *Bayesian Computation with R*, Springer, New York.
2. Agresti, A. and Coull, B. (1998), "Approximate is better than 'exact' for interval estimation of binomial proportions," *The American Statistician*, 52, 119-126.
3. Ashenfelter, O. and Krueger, A. (1994), "Estimates of the economic return to schooling from a new sample of twins," *The American Economic Review*, 84, 1157–1173.
4. Bolstad, W. (2007), *Introduction to Bayesian Statistics*, Wiley-Interscience.
5. Box, G. E. P., Hunter W. G. and Hunter J. S. (1978), *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*, Wiley, New York.
6. Cameron E. and Pauling L. (1978), "Experimental studies designed to evaluate the management of patients with incurable cancer," *Proc. Natl. Acad. Sci. U. S. A.*, 75, 4538–4542.
7. Canty, A. and Ripley. B. (2010), *boot: Bootstrap R Functions,* R package version 1.2-43.
8. Casella, G., and George, E. (1992), "Explaining the Gibbs sampler," *The American Statistician*, 46, 167–174.
9. Chib, S., and Greenberg, E. (1995), "Understanding the Metropolis-Hastings algorithm," *The American Statistician*, 49, 327–335.
10. Cleveland, W. (1979), "Robust locally weighted regression and smoothing scatterplots," *Journal of the American Statistical Association*, 74, 829–83.
11. Carmer, S. G. and Swanson, M. R. (1973), "Evaluation of ten pairwise multiple comparison proceedures by Monte Carlo methods," *Journal of the American Statistical Association*, 68, 66–74.
12. DASL: The Data and Story Library, http://lib.stat.cmu.edu/DASL/.
13. Davison, A. C. and Hinkley, D. V. (1997), *Bootstrap Methods and Their Applications*. Cambridge University Press, Cambridge.
14. Efron, B. and Tibshirani, R. J. (1993), *An Introduction to the Bootstrap*, Chapman & Hall/CRC, Boca Raton, FL.
15. Everitt, B. S., Landau, S., And Leese, M. (2001), *Cluster Analysis*, 4th edition, Oxford University Press, Inc. New York.
16. Faraway, J. (2002), "Practical regression and ANOVA Using R," Contributed documentation in PDF format available at http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf.
17. Friendly, M. (2002), "A brief history of the mosaic display," *Journal of Computational and Graphical Statistics*, 11, 89-107.
18. Fienberg, S. E. (1971), "Randomization and social affairs: The 1970 draft lottery," *Science*, 171, 255–261.

19. Gelman, A., Carlin, J., Stern, H. and Rubin, D. (2003), *Bayesian Data Analysis*, second edition, Chapman and Hall, New York.
20. Gentle, J. E. (2003), *Random Number Generation and Monte Carlo Methods*, second edition, Springer, New York.
21. Hand D. J., Daly F., Lunn A. D., McConway K. J., Ostrowski E. (1994), *A Handbook of Small Data Sets*, Chapman and Hall, London.
22. Hoaglin, D., Mosteller, F., and Tukey, J. (2000), *Understanding Robust and Exploratory Data Analysis*, Wiley-Interscience.
23. Hoff, P. (2009), *A First Course in Bayesian Statistics*, Springer, New York.
24. Hogg, R. and Klugman, S. (1984), *Loss Distributions*, Wiley, New York.
25. Hollander. M. and Wolfe, D. (1999), *Nonparametric Statistical Methods*, second edition, Wiley, New York.
26. Hornik, K (2009). *R FAQ: Frequently Asked Questions on R*, Version 2.13.2011-04-07, http://cran.r-project.org/doc/FAQ/R-FAQ.html.
27. Hothorn, T., Hornik, K., van de Wiel, M. and Zeileis, A. (2008), "Implementing a class of permutation tests: The coin package," *Journal of Statistical Software*, 28, 1-23.
28. Koopmans, L. (1987), *Introduction to Contemporary Statistical Methods*, Duxbury Press.
29. Labby, Z. (2009), "Weldon's dice, automated" *Chance*, 22, 258-264.
30. Larsen, R. J. and Marx, M. L. (2006), *An Introduction to Mathematical Statistics and Its Applications*, 4th edition, Pearson Prentice Hall, Saddle River, New Jersey.
31. Leisch, F. (2007), *bootstrap: Functions for the Book "An Introduction to the Bootstrap,"* R package version 1.0-22.
32. Meyer, D., Zeileis, A., and Hornik, K. (2010), *vcd: Visualizing Categorical Data*, R package version 1.2-9.
33. Montgomery, D. G. (2001), *Design and Analysis of Experiments,* 5th edition., Wiley, New York.
34. Moore, G. (1965), "Cramming more components onto integrated circuits," *Electronics Magazine*, April, 114-117.
35. Mosteller, F. (2010), *The Pleasures of Statistics*, Springer, New York.
36. Mosteller, F. and Wallace, D. L. (1984), *Applied Bayesian and Classical Inference: The Case of the Federalist Papers*, Springer-Verlag, New York.
37. Murrell, P. (2006), *R Graphics*, Chapman and Hall, Boca Raton, Florida.
38. *OzDASL, Australian Data and Story Library*, http://www.statsci.org/data/index.html.
39. Pearson, K. (1900), "On the criterion that a given system of derivations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *Philosophical Magazine*, 5, 157-175.
40. R Development Core Team (2011), "R: A language and environment for statistical computing," R Foundation for Statistical Computing, Vienna, Austria, http://www.R-project.org/.
41. The R Development Core Team (2011), "R: A Language and Environment for Statistical Computing, Reference Index." Version 2.13.0 (2011-04-13)
42. Ross, S. (2007), *Introduction to Probability Models,* 9th edition, Academic Press, Burlington, MA.
43. Sarkar, D. (2008), *Lattice: Multivariate Data Visualization with R*, Springer, New York.
44. Smith, J. M. and Misiak, H. (1973), "The effect of iris color on critical flicker frequency (CFF)," *Journal of General Psychology* 89, 91–95.
45. Snell, L. (1988), *Introduction to Probability*, Random House, New York.

46. Starr, N. (1997), "Nonrandom risk: the 1970 draft lottery. *Journal of Statistics Education*," 5, http://www.amstat.org/publications/jse/v5n2/datasets.starr.html
47. Tukey, J. (1977), *Exploratory Data Analysis*, Addison-Wesley.
48. "Campaign 2004: Time-tested formulas suggest both Bush and Kerry will win on Nov. 2," USA Today, http://www.usatoday.com/news/politicselections/nation/president/2004-06-23-bush-kerry-cover_x.htm.
49. Venables, W. N., Smith, D. M. and the R Development Core Team (2011). "An Introduction to R", Version 2.13.0 (2011-04-13).
50. Venables, W. N. and Ripley, B. D. (2002), *Modern Applied Statistics with S*, fourth edition. Springer, New York.
51. Verzani, J. (2010), *UsingR: Data sets for the text Using R for Introductory Statistics*," R package version 0.1-13.
52. Wickham, H. (2009), *ggplot2: Elegant Graphics for Data Analysis*, Springer, New York.
53. *The Washington Post* (2007) "Heads and Shoulders above the Rest,", http://blog.washingtonpost.com/44/2007/10/11/head_and_shoulders_above.html.
54. Wikipedia, "Heights of Presidents of the United States and presidential candidates," retrieved June 28, 2011.
55. Wilkinson, L. (2005), *The Grammar of Graphics*, second edition, Springer, New York.
56. Willerman, L., Schultz, R., Rutledge, J. N., and Bigler, E. (1991), "In vivo brain size and intelligence," *Intelligence*, 15, 223–228.

Index