Chapter 4
Simple Functions

In previous chapters, we demonstrated how to enter data; read data from a spreadsheet, ascii file, or a database; and extract subsets of data. In this chapter, we discuss applying some simple functions to the data, such as the mean or the mean of a single data subset. These are functions that may be useful; however, they are not the tools that will convince you to become an R user. Use them when it is convenient. Upon first reading of the book, you may skip this chapter.

4.1 The `tapply` Function

R provides functions for calculating the mean, length, standard deviation, minimum, maximum, variance, and any other function of a single variable, multiple variables, or on subsets of observations. For illustration, we use a vegetation dataset. Sikkink et al. (2007) analysed grassland data from a monitoring program conducted in two temperate communities, Yellowstone National Park and the National Bison Range, USA. The aim of the study was to determine whether the biodiversity of these bunchgrass communities changed over time, and, if so, whether the changes in biodiversity related to particular environmental factors. For our purposes we use only the Yellowstone National Park data. To quantify biodiversity, the researchers calculated species richness, defined as the number of *different* species per site. The study identified about 90 species. The data were measured in 8 transects, with each transect being assessed at intervals of 4–10 years, for a total of 58 observations.

The following code can be used to import the data and gain basic information on the variables.

```
> setwd("C:/RBook/")
> Veg <- read.table(file="Vegetation2.txt",
                      header= TRUE)
> names(Veg)
```

```
 [1] "TransectName" "Samples"       "Transect"
 [4] "Time"         "R"             "ROCK"
 [7] "LITTER"       "ML"            "BARESOIL"
[10] "FallPrec"     "SprPrec"       "SumPrec"
[13] "WinPrec"      "FallTmax"      "SprTmax"
[16] "SumTmax"      "WinTmax"       "FallTmin"
[19] "SprTmin"      "SumTmin"       "WinTmin"
[22] "PCTSAND"      "PCTSILT"       "PCTOrgC"
```

```
> str(Veg)
```

```
'data.frame':   58 obs. of 24 variables:
 $ TransectName: Factor w/ 58 levels ...
 $ Samples     : int 1 2 3 4 5 6 7 8 9 10 ...
 $ Transect    : int 1 1 1 1 1 1 1 2 2 2 ...
 $ Time        : int 1958 1962 1967 1974 1981 1994...
 $ R           : int 8 6 8 8 10 7 6 5 8 6 ...
 $ ROCK        : num 27 26 30 18 23 26 39 25 24 21 ...
 $ LITTER      : num 30 20 24 35 22 26 19 26 24 16 ...
                     <Cut to reduce space>
```

The data are stored in the ascii file "Vegetation2.txt." Once the `read.table` function has been executed, we need to ensure that richness is indeed a numerical vector or integer. If, for some reason, R imports richness as a factor (e.g., because there is an alphanumerical in the column, or there are problems with the decimal separator), functions such as `mean`, `sd`, and the like will give an error message[1].

4.1.1 Calculating the Mean Per Transect

One of the first things we would like to know is whether the mean richness per transect differs. The code below calculates the mean richness, as well as mean richness for each transect (see Chapter 3 for selecting subsets of data):

```
> m <- mean(Veg$R)
> m1<- mean(Veg$R[Veg$Transect == 1])
> m2<- mean(Veg$R[Veg$Transect == 2])
> m3<- mean(Veg$R[Veg$Transect == 3])
```

[1] If you import data that are decimal comma separated with the default settings (i.e., with decimal point separation) or vice versa, R will see all variables as factors, and the `str`command will tell you so. Hence, to check that data were imported correctly, we recommend always using the `str` command on the imported data frame immediately after importing the data.

```
> m4<- mean(Veg$R[Veg$Transect == 4])
> m5<- mean(Veg$R[Veg$Transect == 5])
> m6<- mean(Veg$R[Veg$Transect == 6])
> m7<- mean(Veg$R[Veg$Transect == 7])
> m8<- mean(Veg$R[Veg$Transect == 8])
> c(m, m1, m2, m3, m4, m5, m6, m7, m8)

[1]  9.965517  7.571429  6.142857 10.375000 9.250000
[6] 12.375000 11.500000 10.500000 11.833333
```

The variable `m` contains the mean richness of all 8 transects, and `m1` through `m8` show the mean richness values per transect. Note that the `mean` command is applied to `Veg $R`, which is a vector of data. It is not a matrix; hence there is no need for a comma between the square brackets.

4.1.2 Calculating the Mean Per Transect More Efficiently

It is cumbersome to type eight commands to calculate the mean value per transect. The R function `tapply` performs the same operation as the code above (for `m1` through `m8`), but with a single line of code:

```
> tapply(Veg$R, Veg$Transect, mean)
        1          2          3          4          5
 7.571429   6.142857  10.375000   9.250000  12.375000
        6          7          8
11.500000  10.500000  11.833333
```

You can also run this code as

```
> tapply(X = Veg$R, INDEX = Veg$Transect, FUN = mean)
```

The `tapply` function splits the data of the first variable (`R`), based on the levels of the second variable (`Transect`). To each subgroup of data, it applies a function, in this case the mean, but we can also use the standard deviation (function `sd`), variance (function `var`), length (function `length`), and so on. The following lines of code calculate some of these functions for the vegetation data.

```
> Me <- tapply(Veg$R, Veg$Transect, mean)
> Sd <- tapply(Veg$R, Veg$Transect, sd)
> Le <- tapply(Veg$R, Veg$Transect, length)
> cbind(Me, Sd, Le)
         Me        Sd Le
1  7.571429 1.3972763  7
2  6.142857 0.8997354  7
3 10.375000 3.5831949  8
```

```
4  9.250000 2.3145502  8
5 12.375000 2.1339099  8
6 11.500000 2.2677868  8
7 10.500000 3.1464265  6
8 11.833333 2.7141604  6
```

Each row in the output gives the mean richness, standard deviation, and number of observations per transect. In a later chapter we discuss graphic tools to visualise these values.

4.2 The `sapply` and `lapply` Functions

To calculate the mean, minimum, maximum, standard deviation, and length of the full series, we still need to use `mean (Veg$R)`, `min (Veg$R)`, `max (Veg$R)`, `sd (Veg$R)`, and `length (Veg$R)`. This is laborious if we wish to calculate the mean of a large number of variables such as all the numerical variables of the vegetation data. We specifically say "numerical" as one cannot calculate the mean of a factor. There are 20 numerical variables in the vegetation dataset, columns 5–25 of the data frame `Veg`. However, we do not need to type in the `mean` command 20 times. R provides other functions similar to the `tapply` to address this situation: the `lapply` and the `sapply`. The use of `sapply` and its output is given below:

```
> sapply(Veg[, 5:9], FUN= mean)

        R      ROCK    LITTER        ML  BARESOIL
 9.965517 20.991379 22.853448 1.086207 17.594828
```

To save space, we only present the results of the first five variables. It is important to realise that `tapply` calculates the mean (or any other function) for subsets of observations of a variable, whereas `lapply` and `sapply` calculate the mean (or any other function) of one or more variables, using *all* observations.

The word FUN stands for function, and must be written in capitals. Instead of the mean, you can use any other function as an argument for `FUN`, and you can write your own functions. So what is the difference between `sapply` and `lapply`? The major differences lie in the presentation of output, as can be seen in the following example.

```
> lapply(Veg[, 5:9], FUN= mean)
$R
[1] 9.965517
$ROCK
[1] 20.99138
```

```
$LITTER
[1] 22.85345
$ML
[1] 1.086207
$BARESOIL
[1] 17.59483
```

The output of `lapply` is presented as a list, whereas `sapply` gives it as a vector. The choice depends on the format in which you would like the output.

The variable that contains the data in `lapply` and `sapply` needs to be a data frame. This will not work:

```
> sapply(cbind(Veg$R, Veg$ROCK, Veg$LITTER, Veg$ML,
                Veg$BARESOIL), FUN = mean)
```

It will produce one long vector of data, because the output of the `cbind` command is not a data frame. It can easily be changed to a data frame:

```
> sapply(data.frame(cbind(Veg$R, Veg$ROCK, Veg$LITTER,
         Veg$ML, Veg$BARESOIL)), FUN = mean)

       X1         X2         X3        X4         X5
 9.965517  20.991379  22.853448  1.086207  17.594828
```

Note that we have lost the variable labels. To avoid this, make a proper data frame (Chapter 2) before running the `sapply` function. Alternatively, use the `colnames` function after combining the data with the `cbind` function.

Do Exercise 1 in Section 4.6. This is an exercise in the use of the `tapply`, `sapply`, and `lapply` functions with a temperature dataset.

4.3 The summary Function

Another function that gives basic information on variables is the `summary` command. The argument can be a variable, the output from a `cbind` command, or a data frame. It is run by the following commands.

```
> Z <-cbind(Veg$R, Veg$ROCK, Veg$LITTER)
> colnames(Z) <- c("R", "ROCK", "LITTER")
> summary(Z)
```

```
      R                 ROCK             LITTER
Min.    : 5.000    Min.    : 0.00   Min.    : 5.00
1st Qu. : 8.000    1st Qu. : 7.25   1st Qu. :17.00
Median  :10.000    Median  :18.50   Median  :23.00
Mean    : 9.966    Mean    :20.99   Mean    :22.85
3rd Qu. :12.000    3rd Qu. :27.00   3rd Qu. :28.75
Max.    :18.000    Max.    :59.00   Max.    :51.00
```

The `summary` command gives the minimum, first quartile, median, mean, third quartile, and maximum value of the variable. An alternative R code gives the same result:

```
> summary(Veg[ , c("R","ROCK","LITTER")])
```

or

```
> summary(Veg[ , c(5, 6, 7)])
```

Output is not presented here.

4.4 The `table` Function

In Exercises 1 and 7 in Section 2.4, we introduced the deer data from Vicente et al. (2006). The data were from multiple farms, months, years, and sexes. One of the aims of the study was to find a relationship between length of the animal and the number of *E. cervi* parasites. It may be the case that this relationship changes with respect to sex, year, month, farm, or even year and month. To test this, one needs to include interactions in the statistical models. However, problems may be encountered if there are no sampled females in some years, or if some farms were not sampled in every year. The `table` function can be used to learn how many animals per farm were sampled, as well as the number of observations per sex and year. The following code imports the data, and shows the results.

```
> setwd("c:/RBook/")
> Deer <- read.table(file="Deer.txt", header= TRUE)
> names (Deer)
 [1] "Farm"   "Month"   "Year"      "Sex"      "clas1_4"
 [6] "LCT"    "KFI"     "Ecervi"    "Tb"
```

```
> str(Deer)
```

```
[1] "Farm"   "Month"   "Year"      "Sex"      "clas1_4"
[6] "LCT"    "KFI"     "Ecervi"    "Tb"
```

```
> str(Deer)
'data.frame' : 1182 obs. of 9 variables:
 $ Farm      : Factor w/ 27 levels"AL","AU","BA",..: 1...
 $ Month     : int 10 10 10 10 10 10 10 10 10 10 ...
 $ Year      : int 0 0 0 0 0 0 0 0 0 0 ...
 $ Sex       : int 1 1 1 1 1 1 1 1 1 1 ...
 $ clas1_4   : int 4 4 3 4 4 4 4 4 4 4 ...
 $ LCT       : num 191 180 192 196 204 190 196 200 19 ...
 $ KFI       : num 20.4 16.4 15.9 17.3 NA ...
 $ Ecervi    : num 0 0 2.38 0 0 0 1.21 0 0.8 0 ...
 $ Tb        : int 0 0 0 0 NA 0 NA 1 0 0 ...
```

Farm has been coded as AL, AU, and so on, and is automatically imported as a factor. The other variables are all vectors of numerical or integer values. The number of observations per farm is obtained by

```
> table(Deer$Farm)
 AL   AU   BA   BE   CB   CRC   HB   LCV   LN   MAN   MB
 15   37   98   19   93    16   35     2   34    76   41
 MO   NC   NV   PA   PN    QM   RF    RÑ   RO   SAL  SAU
278   32   35   11   45    75   34    25   44     1    3
 SE   TI   TN VISO   VY
 26   21   31   15   40
```

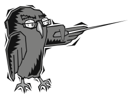
At one farm, 278 animals were sampled and, at others, only one. This dataset typically requires a mixed effects modelling[2] approach in which "farm" is used as a random effect (see Zuur et al., 2009). This method can cope with unbalanced designs. However, the inclusion of a sex/year interaction term[3] in such models for these data will give an error message. This is because both sexes were not measured in every year, as can be seen from the following contingency table. (The labels 0, 1, 2, 3, 4, 5, and 99 in the horizontal direction refer to 2000, 2001, 2002, 2003, 2004, 2005, and 1999, respectively. In the vertical direction 1 and 2 indicate sex).

```
> table(Deer$Sex, Deer$Year)
    0  1   2   3  4  5 99
1 115 85 154  75 78 34 21
2  76 40 197 123 60 35  0
```

In 1999, animals of only one sex were measured. We recommend always using the table command before including interactions between two categorical variables in regression type models.

[2] A mixed effects model is an extension of linear regression.

[3] A sex/year interaction term allows the effect of sex to differ over the years.


Do Exercise 2 in Section 4.6. This is an exercise in using the `table` function with a temperature dataset.

4.5 Which R Functions Did We Learn?

Table 4.1 shows the R functions that were introduced in this chapter.

Table 4.1 R functions introduced in this chapter

Function	Purpose	Example
`tapply`	Apply FUN on y for each level of x	`tapply (y, x, FUN = mean)`
`sapply`	Apply FUN on each variable in y	`sapply (y, FUN = mean)`
`lapply`	Apply FUN on each variable in y	`tapply (y, FUN = mean)`
`sd`	Calculate the standard deviation of y	`sd (y)`
`length`	Determine the length of y.	`length(y)`
`summary`	Calculate general information	`summary(y)`
`table`	Calculate a contingency table	`table(x, y)`

4.6 Exercises

Exercise 1. The use of the `tapply`, `sapply`, and `lapply` functions to calculate mean temperature per month.

The file *temperature.xls* contains temperature observations made at 31 locations along the Dutch coastline. The data were collected and provided by the Dutch institute RIKZ (under the monitoring program MWTL; Monitoring Waterstaatkundige Toestand des Lands). Sampling began in 1990, and the final measurements in the spreadsheet were taken in December 2005, a period of 16 years. Sampling frequency was 0–4 times per month, depending on the season.

Calculate a one-time series of monthly averages using data from all stations. The end result should be a variable of dimension 16×12. Also calculate the standard deviation and number of observations per month.

Exercise 2. The use of the `table` function for the temperature data.

Using the data in Exercise 1, determine the number of observations per station. How many observations were made per year? How many observations were made at each station per year?