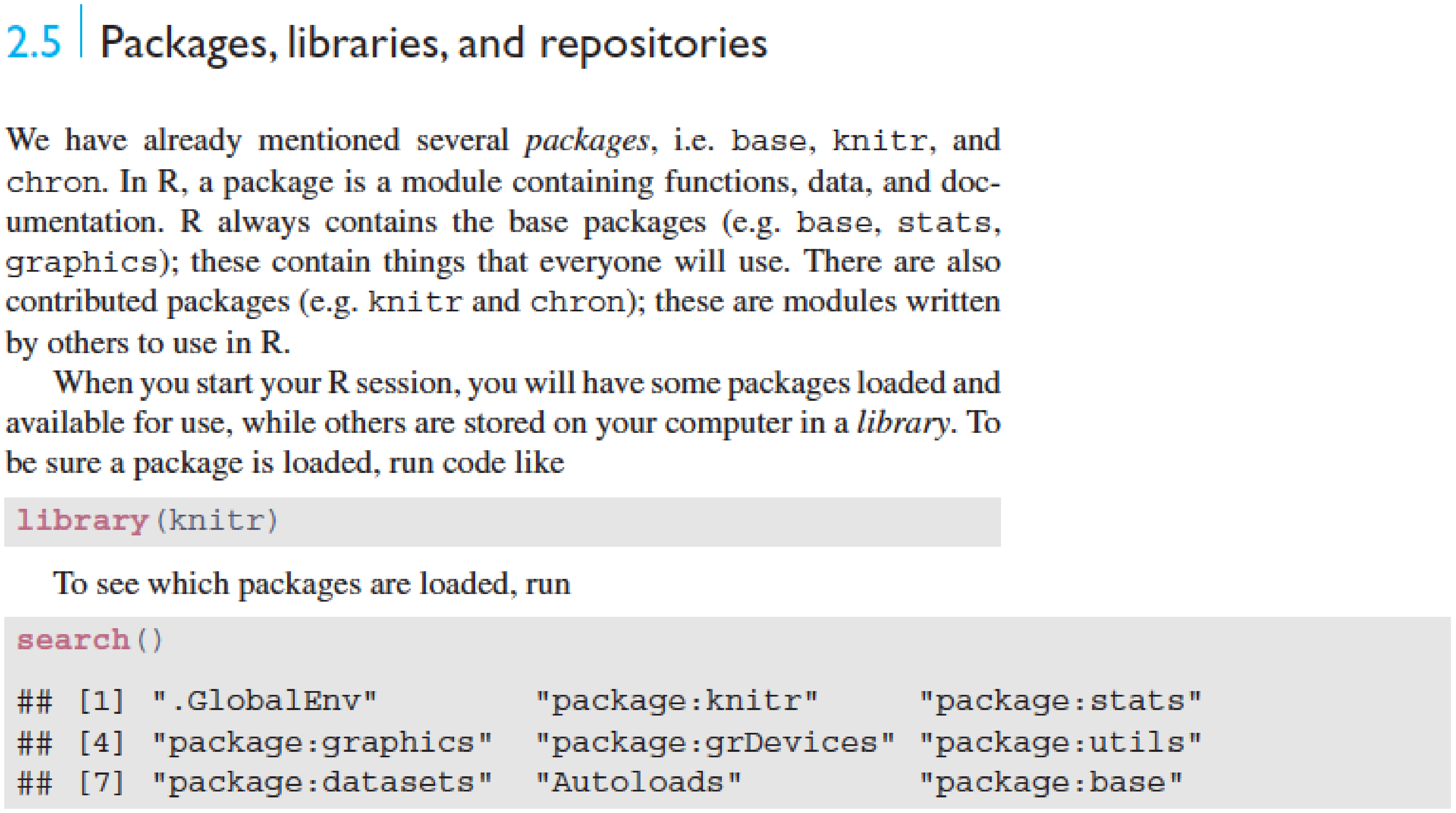
R – Statistics and Plotting

Lecture 4

- Data Manipulation
- Line plots, histogram

Load Require Packages



2.5 Packages, libraries, and repositories

We have already mentioned several *packages*, i.e. base, knitr, and chron. In R, a package is a module containing functions, data, and documentation. R always contains the base packages (e.g. base, stats, graphics); these contain things that everyone will use. There are also contributed packages (e.g. knitr and chron); these are modules written by others to use in R.

When you start your R session, you will have some packages loaded and available for use, while others are stored on your computer in a *library*. To be sure a package is loaded, run code like

```
library(knitr)
```

To see which packages are loaded, run

```
search()

## [1] ".GlobalEnv"        "package:knitr"     "package:stats"
## [4] "package:graphics"  "package:grDevices" "package:utils"
## [7] "package:datasets"  "Autoloads"         "package:base"
```

Source: A first course in statistical programming with R - W. John Braun and Duncan J. Murdoch, Cambridge, 2016.

Packages

```
stats::median(x)
```

Thousands of contributed packages are available, though you likely have only a few dozen installed on your computer. If you try to use one that isn't already there, you will receive an error message:

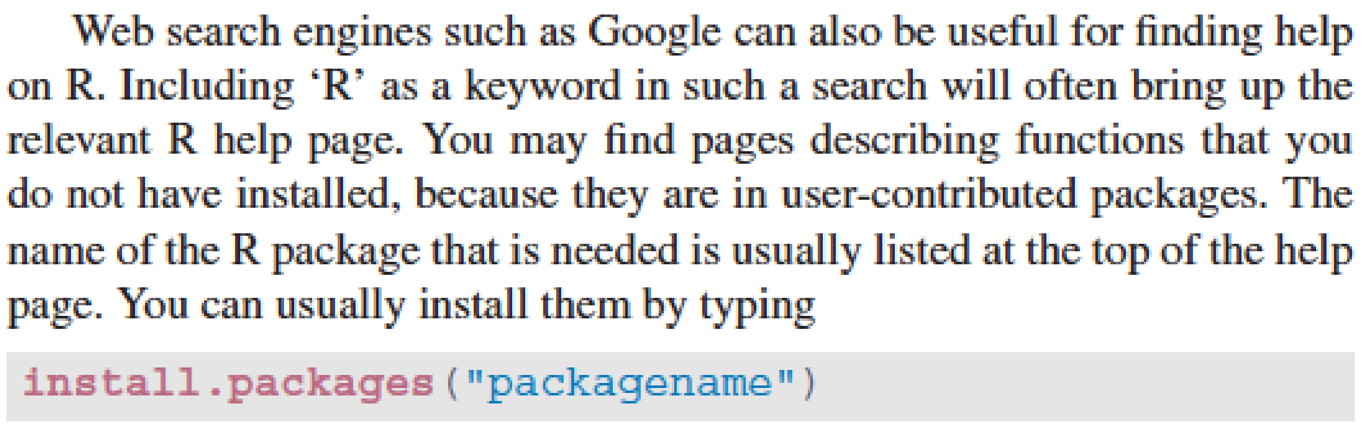
```
library(notInstalled)

## Error in library(notInstalled): there is no package called 'notInstalled'
```

This means that the package doesn't exist on your computer, but it might be available in a *repository* online. The biggest repository of R packages is known as CRAN. To install a package from CRAN, you can run a command like

```
install.packages("knitr")
```

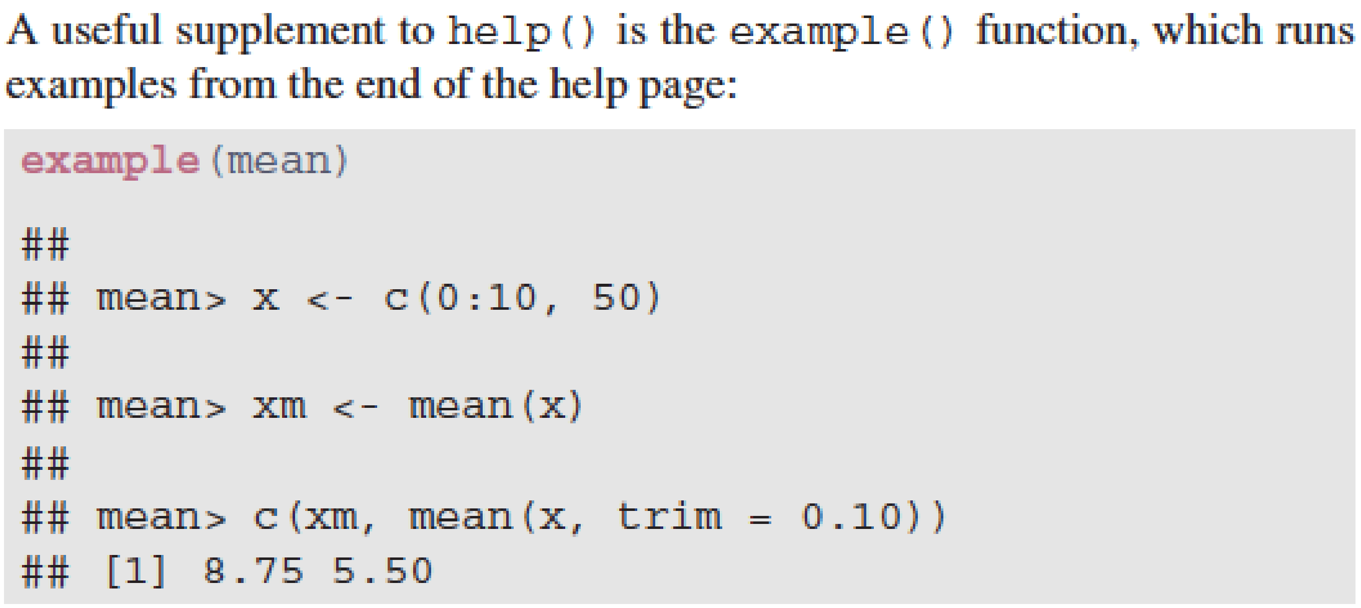
or, within RStudio, click on the `Packages` tab in the Output Pane, choose `Install`, and enter the name in the resulting dialog box.

Built-in Examples



Web search engines such as Google can also be useful for finding help on R. Including ‘R’ as a keyword in such a search will often bring up the relevant R help page. You may find pages describing functions that you do not have installed, because they are in user-contributed packages. The name of the R package that is needed is usually listed at the top of the help page. You can usually install them by typing

```
install.packages("packagename")
```

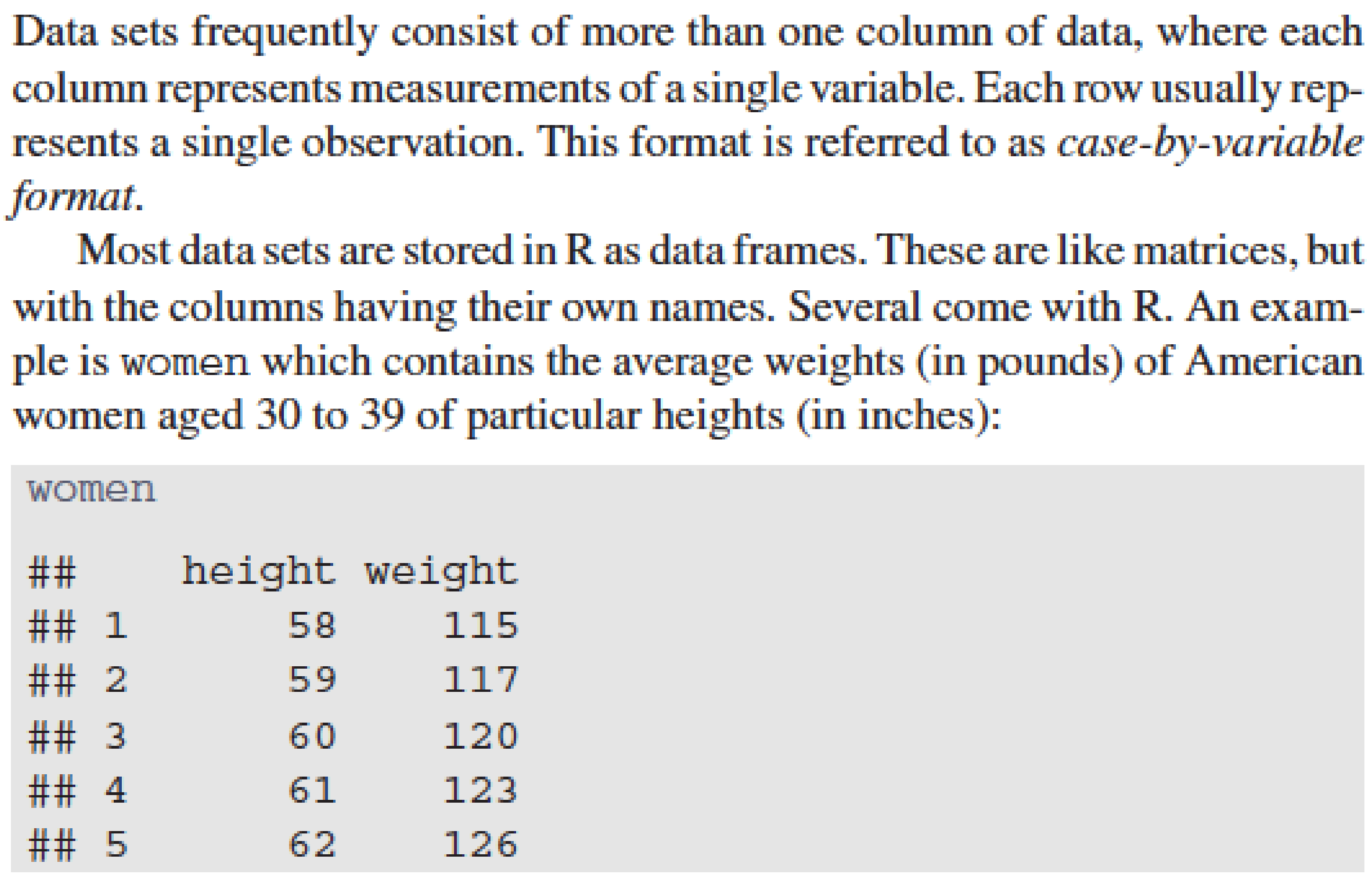


A useful supplement to `help()` is the `example()` function, which runs examples from the end of the help page:

```
example(mean)

##
## mean> x <- c(0:10, 50)
##
## mean> xm <- mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.75 5.50
```

Source: A f

Data Frame and Lists



Data sets frequently consist of more than one column of data, where each column represents measurements of a single variable. Each row usually represents a single observation. This format is referred to as *case-by-variable format*.

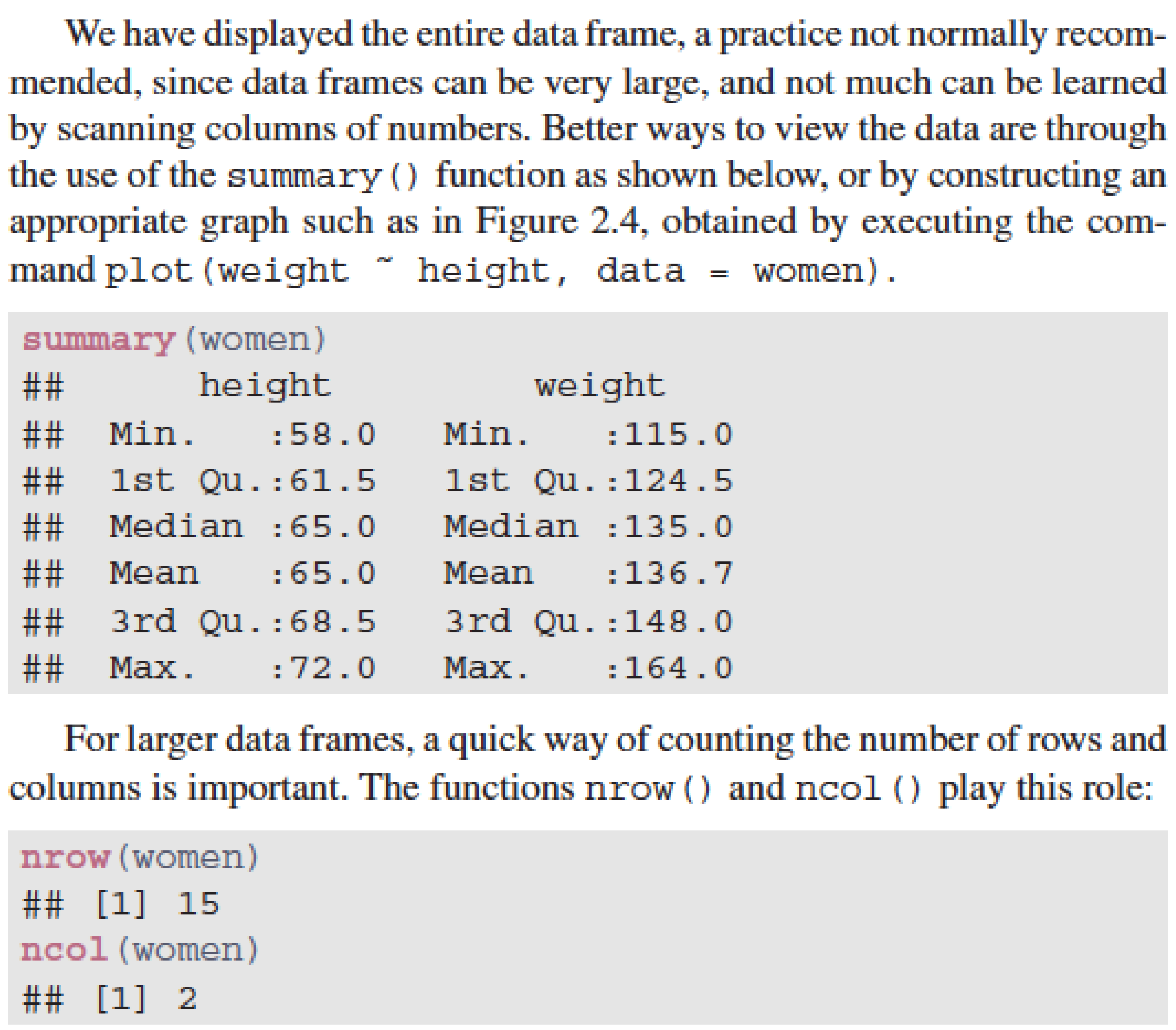
Most data sets are stored in R as data frames. These are like matrices, but with the columns having their own names. Several come with R. An example is women which contains the average weights (in pounds) of American women aged 30 to 39 of particular heights (in inches):

```
women

##     height weight
## 1       58    115
## 2       59    117
## 3       60    120
## 4       61    123
## 5       62    126
```

Try summary(women) in lab

Source: A first course in statistical programming with R - W. John Braun and Duncan J. Murdoch, Cambridge, 2016.

Managing Data Frames



We have displayed the entire data frame, a practice not normally recommended, since data frames can be very large, and not much can be learned by scanning columns of numbers. Better ways to view the data are through the use of the `summary()` function as shown below, or by constructing an appropriate graph such as in Figure 2.4, obtained by executing the command `plot(weight ~ height, data = women)`.

```
summary(women)
##      height          weight     
##  Min.   :58.0   Min.   :115.0  
##  1st Qu.:61.5   1st Qu.:124.5  
##  Median :65.0   Median :135.0  
##  Mean   :65.0   Mean   :136.7  
##  3rd Qu.:68.5   3rd Qu.:148.0  
##  Max.   :72.0   Max.   :164.0  
```

For larger data frames, a quick way of counting the number of rows and columns is important. The functions `nrow()` and `ncol()` play this role:

```
nrow(women)
## [1] 15
ncol(women)
## [1] 2
```

Source: A first course in statistical programming with R - W. John Braun and Duncan J. Murdoch, Cambridge, 2016.

Managing Data Frames

We can extract elements from data frames using similar syntax to what was used with matrices. Consider the following examples:

```
women[7, 2]

## [1] 132

women[3, ]

##   height weight
## 3     60    120

women[4:7, 1]

## [1] 61 62 63 64
```

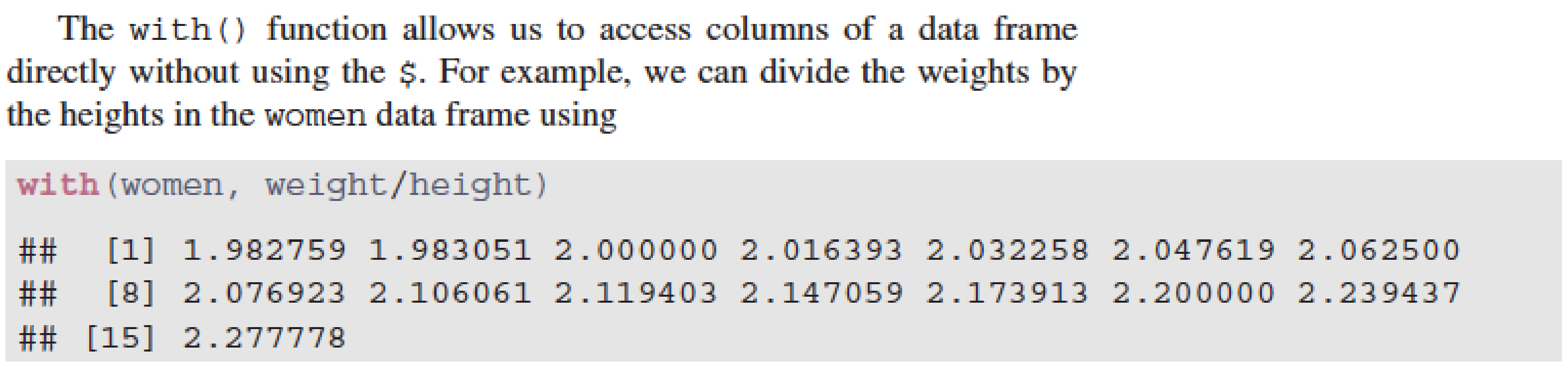
Data frame columns can also be addressed using their names using the $ operator. For example, the weight column can be extracted as follows:

```
women$weight

##  [1] 115 117 120 123 126 129 132 135 139 142 146 150 154 159 164
```

Thus, we can extract all heights for which the weights exceed 140 using

```
women$height[women$weight > 140]

## [1] 67 68 69 70 71 72
```

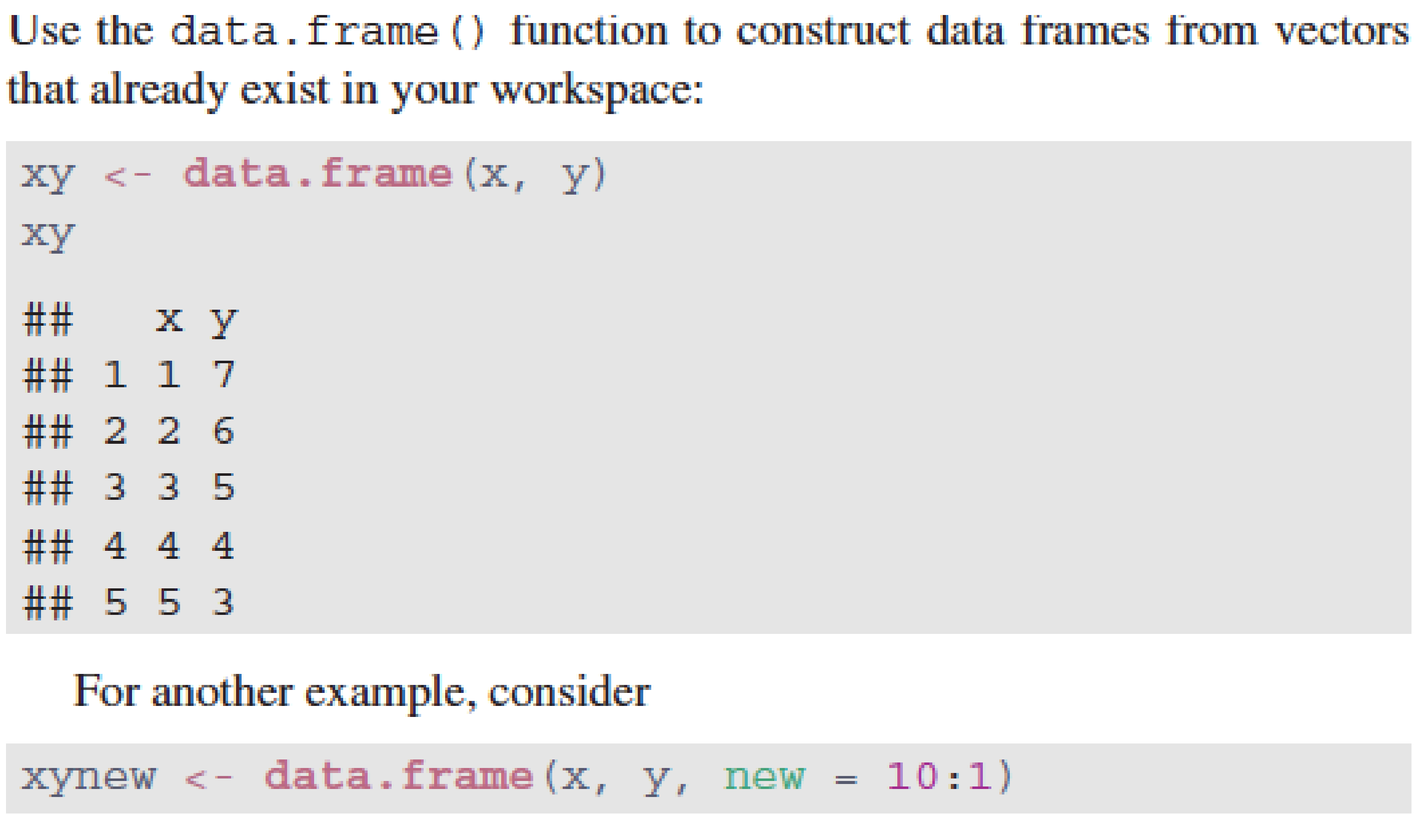
Managing Data Frames



The `with()` function allows us to access columns of a data frame directly without using the `$`. For example, we can divide the weights by the heights in the women data frame using

```
with(women, weight/height)

##  [1] 1.982759 1.983051 2.000000 2.016393 2.032258 2.047619 2.062500
##  [8] 2.076923 2.106061 2.119403 2.147059 2.173913 2.200000 2.239437
## [15] 2.277778
```

Construct Data Frames



Use the data.frame() function to construct data frames from vectors that already exist in your workspace:

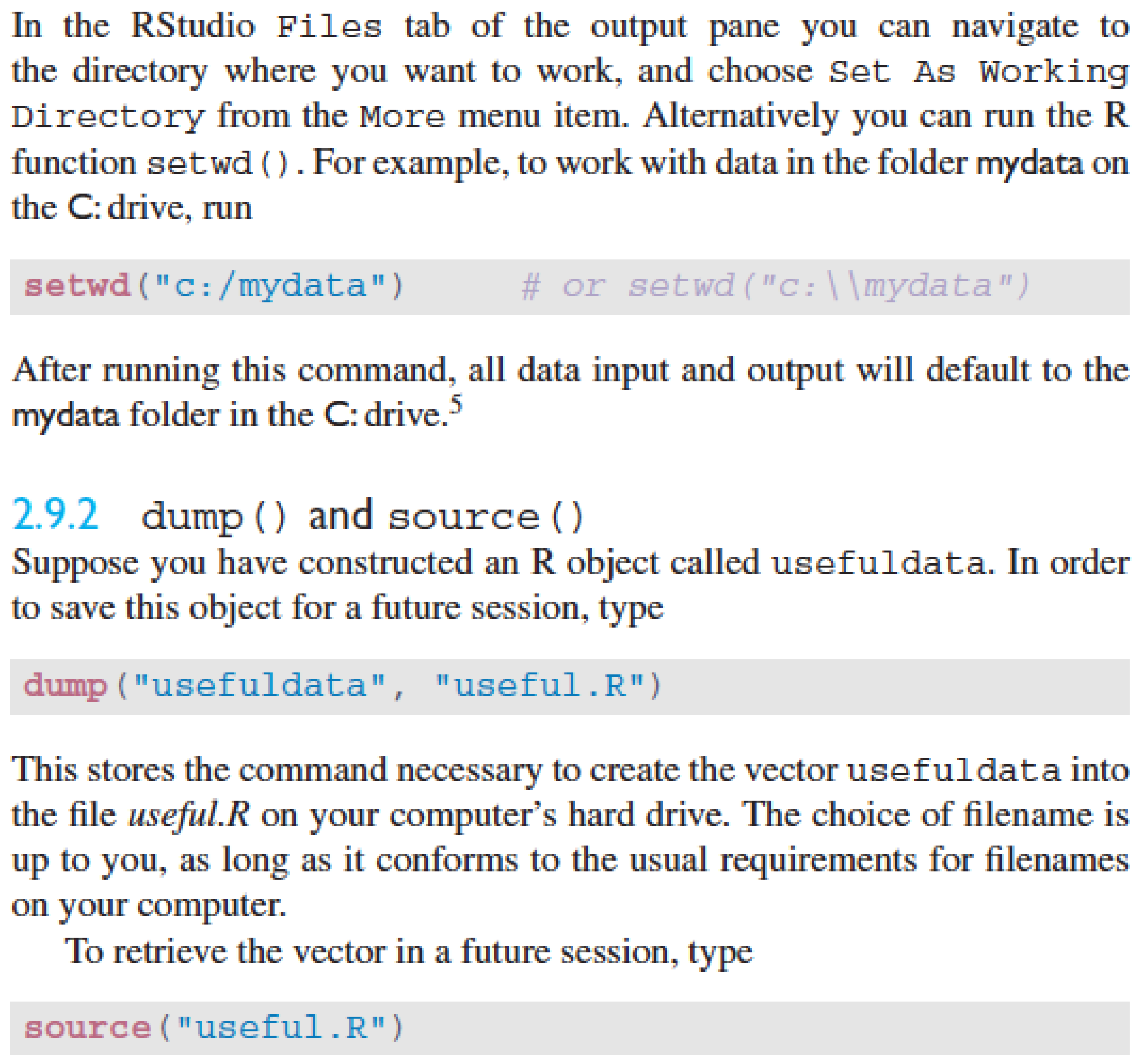
```
xy <- data.frame(x, y)
xy

##   x y
## 1 1 7
## 2 2 6
## 3 3 5
## 4 4 4
## 5 5 3
```

For another example, consider

```
xynew <- data.frame(x, y, new = 10:1)
```

Source: A first course in statistical programming with R - W. John Braun and Duncan J. Murdoch, Cambridge, 2016.

Change Directories



In the RStudio `Files` tab of the output pane you can navigate to the directory where you want to work, and choose `Set As Working Directory` from the `More` menu item. Alternatively you can run the R function `setwd()`. For example, to work with data in the folder **mydata** on the **C:** drive, run

```r
setwd("c:/mydata")        # or setwd("c:\\mydata")
```

After running this command, all data input and output will default to the **mydata** folder in the **C:** drive.[5]

2.9.2 `dump()` and `source()`

Suppose you have constructed an R object called `usefuldata`. In order to save this object for a future session, type

```r
dump("usefuldata", "useful.R")
```

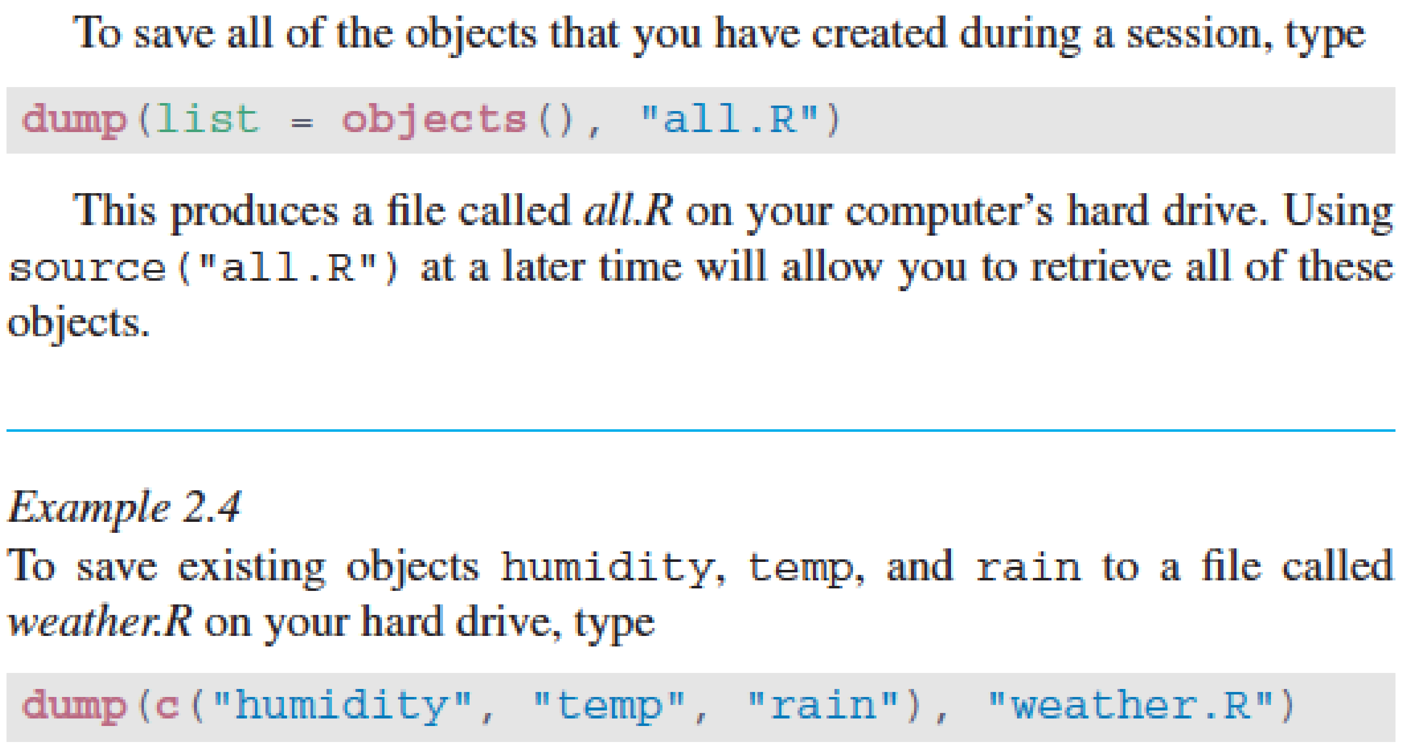
This stores the command necessary to create the vector `usefuldata` into the file *useful.R* on your computer's hard drive. The choice of filename is up to you, as long as it conforms to the usual requirements for filenames on your computer.

To retrieve the vector in a future session, type

```r
source("useful.R")
```

Source: A first course in statistical programming with R - W. John Braun and Duncan J. Murdoch, Cambridge, 2016.

Storing Objects



To save all of the objects that you have created during a session, type

```
dump(list = objects(), "all.R")
```

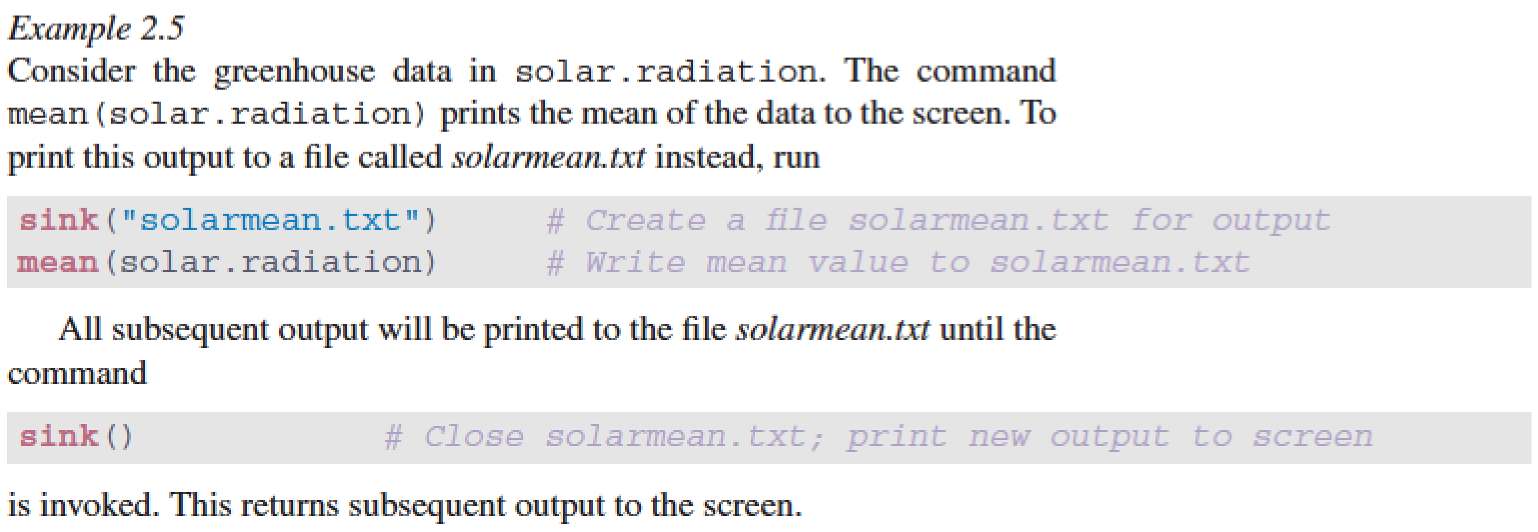
This produces a file called *all.R* on your computer's hard drive. Using `source("all.R")` at a later time will allow you to retrieve all of these objects.

Example 2.4
To save existing objects `humidity`, `temp`, and `rain` to a file called *weather.R* on your hard drive, type

```
dump(c("humidity", "temp", "rain"), "weather.R")
```

Source: A first course in statistical programming with R - W. John Braun and Duncan J. Murdoch, Cambridge, 2016.

Storing Objects



Example 2.5
Consider the greenhouse data in `solar.radiation`. The command `mean(solar.radiation)` prints the mean of the data to the screen. To print this output to a file called *solarmean.txt* instead, run

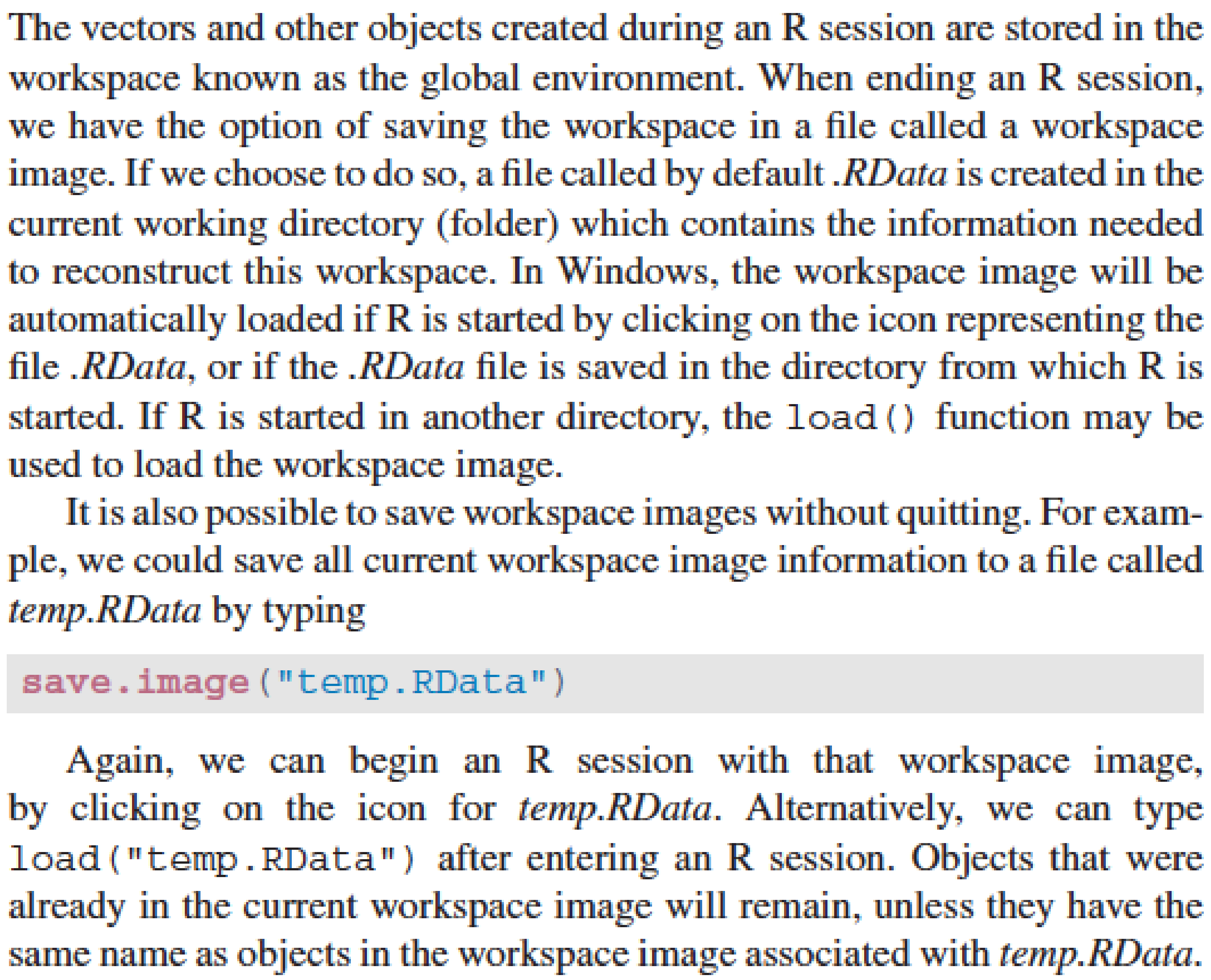
```r
sink("solarmean.txt")      # Create a file solarmean.txt for output
mean(solar.radiation)      # Write mean value to solarmean.txt
```

All subsequent output will be printed to the file *solarmean.txt* until the command

```r
sink()              # Close solarmean.txt; print new output to screen
```

is invoked. This returns subsequent output to the screen.

Saving and Retrieving Image Files



The vectors and other objects created during an R session are stored in the workspace known as the global environment. When ending an R session, we have the option of saving the workspace in a file called a workspace image. If we choose to do so, a file called by default *.RData* is created in the current working directory (folder) which contains the information needed to reconstruct this workspace. In Windows, the workspace image will be automatically loaded if R is started by clicking on the icon representing the file *.RData*, or if the *.RData* file is saved in the directory from which R is started. If R is started in another directory, the `load()` function may be used to load the workspace image.

It is also possible to save workspace images without quitting. For example, we could save all current workspace image information to a file called *temp.RData* by typing

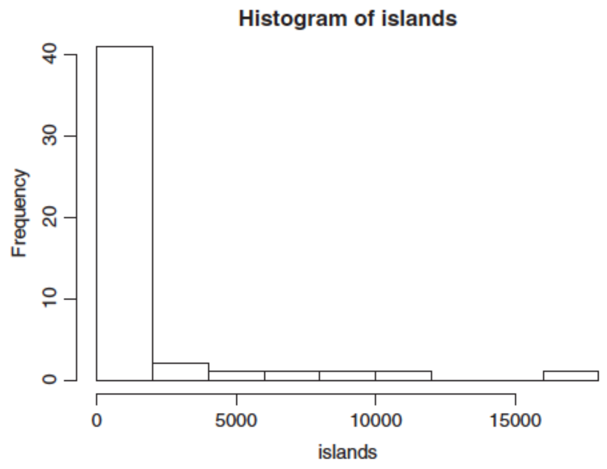
```
save.image("temp.RData")
```

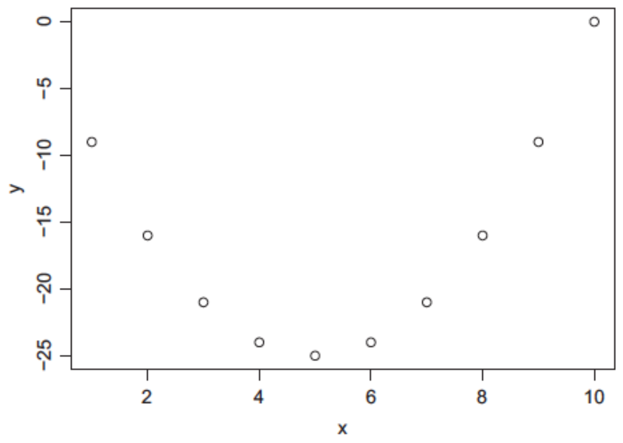
Again, we can begin an R session with that workspace image, by clicking on the icon for *temp.RData*. Alternatively, we can type `load("temp.RData")` after entering an R session. Objects that were already in the current workspace image will remain, unless they have the same name as objects in the workspace image associated with *temp.RData*.

Plotting

Two basic plots are the histogram and the scatterplot. The codes below were used to produce the graphs that appear in Figures 2.1 and 2.2:

```r
hist(islands)
x <- seq(1, 10)
y <- x^2 - 10 * x
plot(x, y)
```



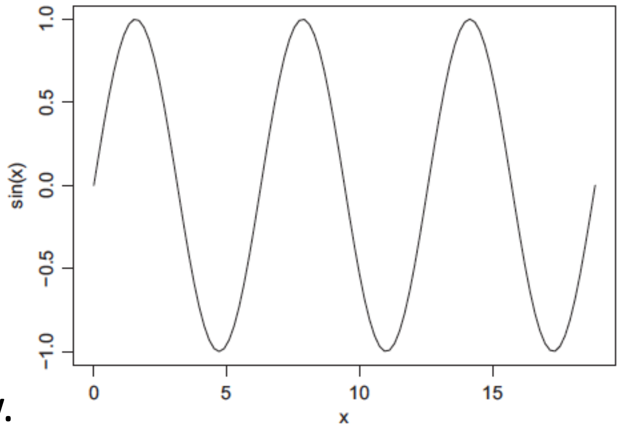


Plotting

Another useful plotting function is the `curve()` function for plotting the graph of a univariate mathematical function on an interval. The left and right endpoints of the interval are specified by `from` and `to` arguments, respectively.

A simple example involves plotting the sine function on the interval $[0, 6\pi]$:

```
curve(expr = sin, from = 0, to = 6 * pi)
```



Try the command below.

```
curve(x^2 - 10 * x, from = 1, to = 10)
```

Plotting

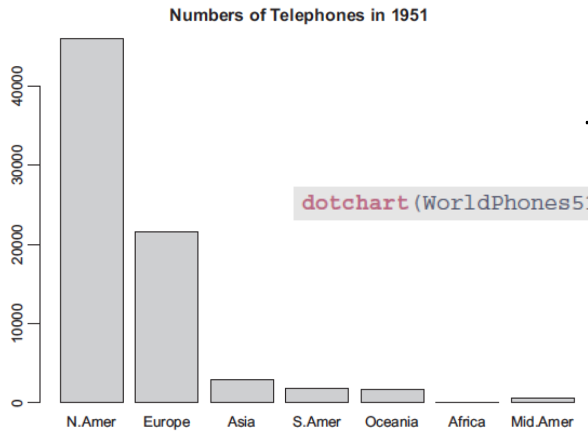
We could plot the bar chart using the barplot() function as

```
barplot(WorldPhones51)
```

```
barplot(WorldPhones51, cex.names = 0.75, cex.axis = 0.75,
            main = "Numbers of Telephones in 1951")
```

Understanding the code

The cex.names = 0.75 argument reduced the size of the region names to 0.75 of their former size, and the cex.axis = 0.75 argument reduced the labels on the vertical axis by the same amount. The main argument sets the main title for the plot.



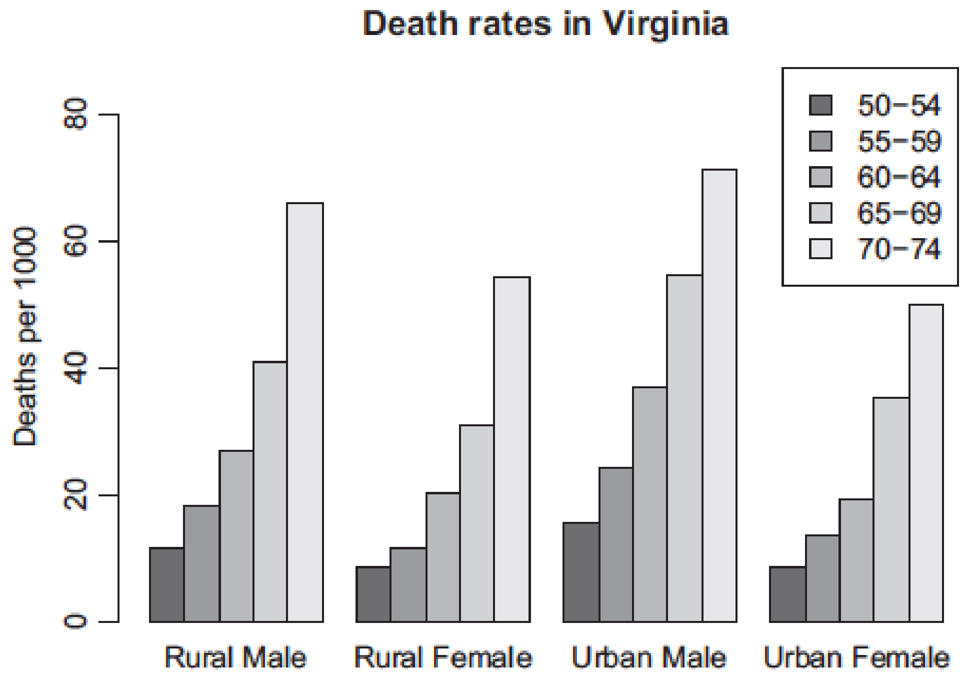
Try this in lab.

```
dotchart(WorldPhones51, xlab = "Numbers of Phones ('000s)")
```

Source: A first course in statistical programm Cambridge, 2016.

Plotting

Lets try this example in lab but using the Hope Discharge data



```
barplot(VADeaths, beside = TRUE, legend = TRUE, ylim = c(0, 90),
        ylab = "Deaths per 1000",
        main = "Death rates in Virginia")
```

Example 3.2
The `VADeaths` dataset in R contains death rates (number of deaths per 1000 population per year) in various subpopulations within the state of Virginia in 1940.

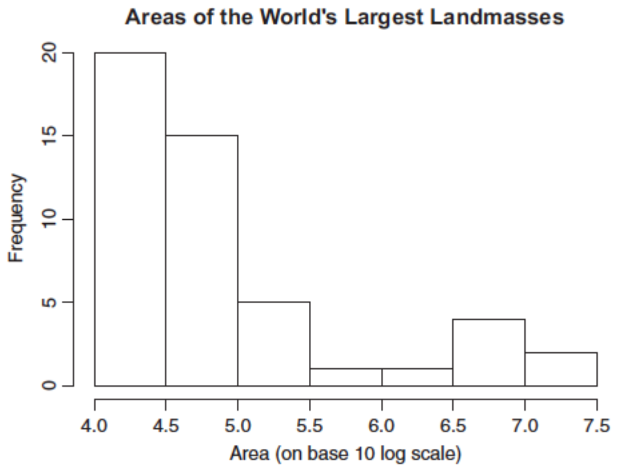
Source: A first course in statistical programming with R - W. John Braun and Duncan J. Murdoch, Cambridge, 2016.

Histrograms

A histogram is a special type of bar chart that is used to show the frequency distribution of a collection of numbers. Each bar represents the count of x values that fall in the range indicated by the base of the bar. Usually all bars should be the same width; this is the default in R. In this case the height of each bar is proportional to the number of observations in the corresponding interval. If bars have different widths, then the *area* of the bar should be proportional to the count; in this way the height represents the density (i.e. the frequency per unit of x).

In R, `hist(x, ...)` is the main way to plot histograms. Here `x` is a vector consisting of numeric observations, and optional parameters in `...` are used to control the details of the display.

```r
hist(log(1000*islands, 10),  xlab = "Area (on base 10 log scale)",
    main = "Areas of the World's Largest Landmasses")
```



Scatter Plot

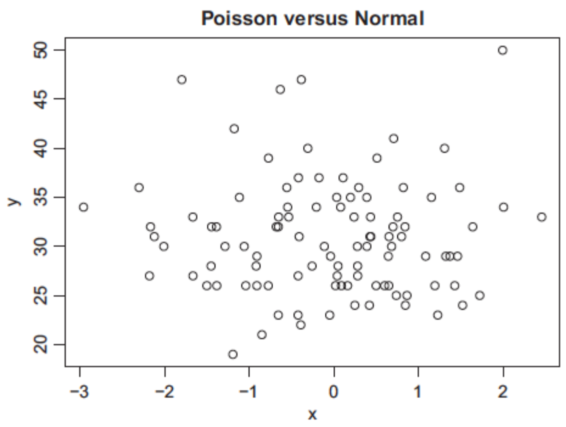
```
x <- rnorm(100)        # assigns 100 random normal observations to x
y <- rpois(100, 30)    # assigns 100 random Poisson observations
                       # to y; mean value is 30
mean(y)                # the resulting value should be near 30

## [1] 30.91
```

The `main` argument sets the main title for the plot. Figure 3.10 shows the result of

```
plot(x, y, main = "Poisson versus Normal")
```

Other possibilities you should try:

```
plot(x, y, pch = 16)       # changes the plot symbol to a solid dot
plot(x, y, type = 'l')     # plots a broken line (a dense tangle of line
                           # segments here)
plot(sort(x), sort(y), type = 'l')  # a plot of the sample "quantiles"
```



Source: A first course in statistical programming with dge, 2016.