Chapter 5

Data Frames

Of the types of objects known to R, we have covered two in detail (vectors and matrices) and one in passing (lists). This chapter is devoted to the most flexible and arguably most used object type, data frames. Whenever you encounter datasets they will usually be stored in `data.frame` objects. In this chapter we will cover how to load and save datasets; how to manipulate, and edit them; and how to compute some data summaries.

5.1 Loading and Saving Datasets

Let's start with loading and saving datasets. The easiest way to load data is with the `data()` function. If entered without arguments, it will bring up a list of all datasets that come bundled with R.

```
> data()
>
```

Many R packages you may download also bundle their own datasets. To access all datasets known to R type:

```
> data(package = .packages(all.available = TRUE))
>
```

To load a package, simply add its name as the argument of the `data()`. Let's load a package that comes bundled with the ggplot2 package. To load it, we first need to load the ggplot2 library.

```
> rm(list = ls()) # remove all objects in active memory
> ls()
character(0)
> library(ggplot2)
> data(diamonds)
> ls()
[1] "diamonds"
```

We can verify that it is indeed a data frame with the `class()` function:

```
> class(diamonds)
[1] "data.frame"
```

As with lists, we can find names of components with the `names()` function:

```
> names(diamonds)
[1] "carat"    "cut"      "color"    "clarity" "depth"
[6] "table"    "price"    "x"        "y"        "z"
```

. . . and some other characteristics – `dim()`, `nrow()`, and `ncol()`:

```
> dim(diamonds) # the dimensions of the data frame
[1] 53940    10

> nrow(diamonds) # the number of observations
[1] 53940

> ncol(diamonds) # the number of variables
[1] 10
```

To save a copy of the dataset to your hard drive you can use the `save()` function we discussed before:

```
> save(diamonds, file = "z:/diamonds.Rdata")
>
```

5.1.1 Other Formats

In virtually all cases, you will not find the data you want or need bundled in R. Frequently, you will have created or downloaded data in some other program (e.g. Excel, or Stata, etc). Below we will go over loading and saving datasets with different formats and from various sources.

Comma Separated Values

The easiest way to import data is from files in the comma separated values `.csv` format.If you have created such a file, for example using Excel or a text editor, you can load it with the `read.csv()` function. A typical `.csv` file will look something like this

```
"Ozone","Solar.R","Wind","Temp","Month","Day"
41,190,7.4,67,5,1
36,118,8,72,5,2
12,149,12.6,74,5,3
18,313,11.5,62,5,4
NA,NA,14.3,56,5,5
28,NA,14.9,66,5,6
23,299,8.6,65,5,7
19,99,13.8,59,5,8
...
```

If you have a file in the `.csv` format you can load it into **R** via the `read.csv()` function. The `read.csv()` function takes the following arguments:

→	`file = " "`	to specify the file name and location. this can be on your hard drive or a website
→	`sep = ","`	to specify the character used as the delimiter (the default is a comma)
→	`header = TRUE`	to specify whether your data has a variable names (the default is yes)
→	`quote = "\""`	to specify what quotes look like in your dataset (the default is ")[1]
→	`dec = "."`	to specify what decimals look like (the default is `.`)[2]

Let's load a dataset from my website. Since the dataset is formated nicely, we can just use the defaults (i.e. we don't have to specify all the above arguments).

```
> FE2013 <- read.csv(file = "http://www.peterhaschke.com/
   Teaching/R-Course/FE2013.csv")
>
```

[1]You may be confused by the `"\""` notation here. Whenever **R** encounters a delimiter of any kind (e.g. `{`, `(`, `[`, or `"`) It will wait for its counterpart to complete the operation. So if we had typed `quote = """`, **R** would be waiting for you to close the third quotation mark and produce an error. To tell **R** not treat the second `"` as a delimiter but a character, you will have to use **R**'s escape character, which happens to be `\`. We will come across the escape character again later on.

[2]This is useful when you are dealing with some European datasets that contain commas to instead of periods to identify decimals. Often they also use tabs, or semi-colons as delimiters. So pay attention to how your data is formatted.

You can also tell R to download the file for you so that you have a physical copy on your disk. You can then load it with the same function:

```
> download.file(url = "http://www.peterhaschke.com/Teaching/R
  -Course/FE2013.csv", destfile = "z:/FE2013.csv")
> FE2013 <- read.csv(file = "z:/FE2013.csv")
>
```

To save your dataset you can use the save() function we used before or the write.csv() function:

```
> save(FE2013, file = "z:/Dataset.Rdata")
> # or
> write.csv(FE2013, file = "z:/Dataset.csv")
>
```

Stata Data `.dta`

Importing datasets created in different formats is relatively straightforward. All you have to do is install the `foreign` package. This should be installed already in the THE STAR LAB but you will have to load it before being able to access its functions. The `foreign` package extends R's read() and write() functions. You now have access to read.dta() and write.dta() to read and write Stata files, and many others (e.g. read.spss(), etc).[3]

```
> library(foreign)
> Students <- read.dta("http://www.peterhaschke.com/Teaching/
  R-Course/Students.dta")
>
```

5.2 Manipulating Data Frames

Once you have your datasets loaded fun ensues. At this point we have three datasets in R's active memory, the diamonds dataset, the one on fuel economy and one on current Political Science Ph.D. students at the University of Rochester. Let's verify just to make sure:

```
> ls()
[1] "diamonds"  "FE2013"  "Students"
```

[3]Use the help.start() function to find out more about this package.