Reading Data in S-PLUS

Aidan Palmer

May 26, 1997

1 Introduction

This handout is intended for people who have some experience using S-PLUS and are familiar with common functions and commands. The following explains how to read data from text files into S-PLUS and how to manipulate the resulting data frames.

2 Using read.table

To follow this handout while running S-PLUS, go to the directory `/afs/andrew/stat/data/S-Tutorials` and copy examples 1 through 6 and the file `football.dat` to your home directory. Then go to your home directory and run S-PLUS.

There are many ways to read data, including `scan` and `read.table`. The function `scan("filename")` reads a file as a vector. This is useful if the data file consists of one vector, or the data have an unorthodox structure that cannot be read any other way.

If the data in the file are laid out neatly in a table, the best way read them into S-PLUS is with `read.table`. This function returns a data frame (an object which has several columns of data, each of which represent a different variable). The function `read.table` has several different options which allow you to read data from several different formats.

S-PLUS for Windows can read data from fancier formats, for instance Lotus and Excel files, when you choose "Import-External Files" from the menu.

The six example files contain ficticious data regarding the dose of a pain reliever that a patient took, how much water he or she took it with, and how much pain relief he or she experienced. The six files have only minor differences, to demonstrate the different formats from which `read.table` can read data.

2.1 Example 1

The first example looks like this:

```
.1                        10                7
.2                        10                15
.3                        10                19
.4                        10                23
.1                        20                15
.2                        20                13
.3                        20                26
.4                        20                38
.1                        30                21
.2                        30                28
.3                        30                31
.4                        30                47
```

Reading across, this says that the first patient took a dose of .1, with 10 units of water, and experienced 7 units of relief.

We can read this into S-PLUS (calling the data frame "ex1"):

```
> ex1 <- read.table("example1")
> ex1
    V1 V2 V3
 1 0.1 10  7
 2 0.2 10 15
 3 0.3 10 19
 4 0.4 10 23
 5 0.1 20 15
 6 0.2 20 13
 7 0.3 20 26
 8 0.4 20 38
 9 0.1 30 21
10 0.2 30 28
11 0.3 30 31
12 0.4 30 47
```

S-PLUS reads the data into a 12-row, 3-column matrix (just like the file) and labels the rows (the observations) 1 through 12 and the columns (the variables) V1, V2 and V3. These columns can be given more descriptive names by making an assignment to the `names` attribute of `ex1`.

```
> names(ex1) <- c("Dose","Water","Relief")
```

And to check that it worked:

```
> names(ex1)
[1] "Dose"   "Water"  "Relief"
> ex1
   Dose Water Relief
 1  0.1    10      7
 2  0.2    10     15
 3  0.3    10     19
 4  0.4    10     23
 5  0.1    20     15
 6  0.2    20     13
 7  0.3    20     26
 8  0.4    20     38
 9  0.1    30     21
10  0.2    30     28
11  0.3    30     31
12  0.4    30     47
```

We could have saved a step by specifying the column names in the `read.table` command:

```
> ex1 <- read.table("example1", col.names=c("Dose", "Water", "Relief"))
```

2.2 Example 2

In the second example, the data file already includes the variable names.

```
Dose                    Water           Relief
.1                      10              7
.2                      10              15
.3                      10              19
...
```

Unless instructed otherwise, S-PLUS will read the data into a 13-row, 3-column matrix with columns labelled V1, V2, and V3. The first observation in the V1 column would be "Dose", the first in the V2 column would be "Water", and the first in the V3 column would be "Relief". That isn't what we want, instead we want it to treat the first row (the "header") as variable names.

```
> ex2 <- read.table("example2", header=T)
```

The "header=T" argument tells S-PLUS that "example 2" has a header (T stands for "true"). The resulting data frame is the same as the one at the end of example 1.

2.3 Example 3

Example 3 is just like example 2, except now there is a line of text at the top of the file which says "THIS IS FICTICIOUS DATA!".

```
THIS IS FICTICIOUS DATA!

Dose                    Water           Relief
.1                      10              7
.2                      10              15
...
```

S-PLUS, if not warned, will interpret the first sentence as four columns: "THIS", "IS", "FICTICIOUS" and "DATA!". It will become confused when it reaches the data, which has three columns, not four. Tell S-PLUS to skip over the first two lines of the file (the text line and then the blank line that follows) so that the first line S-PLUS tries to read is the first line that we are actually interested in.

```
> ex3 <- read.table("example3", header=T, skip=2)
```

2.4 Example 4

Example 4 is just like example 2 except the data are separated by commas instead of by blank spaces.

```
Dose,Water,Relief
.1,10,7
.2,10,15
...
```

This can be read with `read.table`, after specifying that the separation character in this file is a comma.

```
> ex4 <- read.table("example4", header=T, sep=",")
```

2.5 Example 5

Example 5 is just like example 2 again, except that now we have a fourth variable representing the names of the patients.

```
Dose                    Water           Relief  Patient
.1                      10              7       "Homer"
.2                      10              15      "Marge"
...
```

If we try to read this in, S-PLUS will use the patients' names to label the rows (in place of the numbers one through twelve that it used in previous examples) instead of treating them as a variable. S-PLUS will try to do this any time the data array contains a column of text data with no elements repeated.

```
> ex5 <- read.table("example5", header=T)
> ex5
             Dose Water Relief
       Homer  0.1    10      7
       Marge  0.2    10     15
        Bart  0.3    10     19
        Lisa  0.4    10     23
      Maggie  0.1    20     15
       Burns  0.2    20     13
    Smithers  0.3    20     26
     Skinner  0.4    20     38
   Krabapple  0.1    30     21
      Krusty  0.2    30     28
    Flanders  0.3    30     31
Sideshow Bob  0.4    30     47
```

If you want S-PLUS instead to treat the patients' names as a vector, you must specify that S-PLUS should use the null row names (the numbers one through the length of the table, as in previous examples).

```
> ex5 <- read.table("example5", header=T, row.names=NULL)
> ex5
   Dose Water Relief      Patient
 1  0.1    10      7        Homer
 2  0.2    10     15        Marge
 3  0.3    10     19         Bart
 4  0.4    10     23         Lisa
 5  0.1    20     15       Maggie
 6  0.2    20     13        Burns
 7  0.3    20     26     Smithers
 8  0.4    20     38      Skinner
 9  0.1    30     21    Krabapple
10  0.2    30     28       Krusty
11  0.3    30     31     Flanders
12  0.4    30     47 Sideshow Bob
```

If you do want row names but there are none in the file (like in previous examples), or if you want S-PLUS to treat the text variables as vectors and use something else for row names, you can use the `row.names` argument to `read.table` which works like the `col.names` argument demonstrated in example 1.

> **Question:** How do you read example 6? It is exactly like example 4 in that the data are separated by commas but this time there are no commas between the variable names.

At this point you should remove the variables created above from your `.Data` area, because you will never need them again. Type `objects()` to see what variables you have created, and then use `rm` to remove the ones you don't want. For instance, type `rm(ex1)` to remove the data frame created in example 1.

3 Referencing Data in a Data Frame

We could have stored the example data in three different columns, instead of a single data frame. However, that would have created three times as many variables, and we would have no record of which go together. If you use data frames carefully, there is no danger of confusing `Dose` from the pain relief problem with a Dose vector from some other problem.

Read example two into S-PLUS again, this time calling it `pain.relief`.

```
> pain.relief <- read.table("example2", header=T)
> pain.relief
   Dose Water Relief
 1  0.1    10      7
 2  0.2    10     15
 3  0.3    10     19
 4  0.4    10     23
 5  0.1    20     15
 6  0.2    20     13
 7  0.3    20     26
 8  0.4    20     38
 9  0.1    30     21
10  0.2    30     28
11  0.3    30     31
12  0.4    30     47
```

This data frame consists of 12 observations in three variables. To reference a variable in the `pain.relief` data frame, put the name of a variable after the name of the data frame, separated by a $ sign.

```
> pain.relief$Dose
 [1] 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4
> pain.relief$Water
 [1] 10 10 10 10 20 20 20 20 30 30 30 30
> pain.relief$Relief
 [1]  7 15 19 23 15 13 26 38 21 28 31 47
```

So to do a summary of the variable `Relief`, the command is:

```
> summary(pain.relief$Relief)
 Min. 1st Qu. Median  Mean 3rd Qu. Max.
    7      15     22 23.58   28.75   47
```

4 Attaching a Data Frame

Typing the name of the data frame and a $ every time could grow tiresome very quickly. However, if you only type the name of the variable S-PLUS will not know where to find it. To solve this problem, put the data frame on your search path.

You can check your search path by typing `search()`:

```
> search()
[1] "/GRAFT-U2/ap3b/.Data.hp"
[2] "/usr/statlocal/lib/Splus3.3/splus/.Functions"
[3] "/usr/statlocal/lib/Splus3.3/stat/.Functions"
[4] "/usr/statlocal/lib/Splus3.3/s/.Functions"
[5] "/usr/statlocal/lib/Splus3.3/s/.Datasets"
[6] "/usr/statlocal/lib/Splus3.3/stat/.Datasets"
[7] "/usr/statlocal/lib/Splus3.3/splus/.Datasets"
```

This means that if I type something, S-PLUS first looks for a function or variable of that name in my `.Data` directory, and then in the directories of functions and data sets that come with S-PLUS. You may have different directories on your search path, but this should give you the basic idea.

Next attach the `pain.relief` data frame to your search path in the second position (you can specify a different position, but two is the default). This means that if you type the name of a variable or function, S-PLUS will first look for it in your `.Data` directory, then in the `pain.relief` data frame, and then in the S-PLUS directories. This gives precedence to any existing variables over the ones in the data frame.

```
> attach(pain.relief)
> search()
[1] "/GRAFT-U2/ap3b/.Data.hp"
[2] "pain.relief"
[3] "/usr/statlocal/lib/Splus3.3/splus/.Functions"
[4] "/usr/statlocal/lib/Splus3.3/stat/.Functions"
[5] "/usr/statlocal/lib/Splus3.3/s/.Functions"
[6] "/usr/statlocal/lib/Splus3.3/s/.Datasets"
[7] "/usr/statlocal/lib/Splus3.3/stat/.Datasets"
[8] "/usr/statlocal/lib/Splus3.3/splus/.Datasets"
```

When you are done with the problem, use the `detach` function to remove the data frame from your search path. If you type `detach()` with no arguments, S-PLUS will detach the second thing from your search path. If this is not what you want, you can specify by number which data frame or directory you want detached. As long as you use `attach(pain.relief)` without specifying a position, you can use plain `detach()`.

When a data frame is attached to your search path, you can manipulate the variables in the data frame directly.

```
> Dose
   1   2   3   4   5   6   7   8   9  10  11  12
 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4
> Water
  1  2  3  4  5  6  7  8  9 10 11 12
 10 10 10 10 20 20 20 20 30 30 30 30
> Relief
 1  2  3  4  5  6  7  8  9 10 11 12
 7 15 19 23 15 13 26 38 21 28 31 47
```

Assignments to these variables will not affect the data in the data frame. Rather, they will result in the creation of a variable of the same name in your `.Data` area. For example, suppose `Dose` needs to be replaced by `Dose*10`.

```
> pain.relief <- read.table("example2", header=T)
> attach(pain.relief)
> pain.relief$Dose
 [1] 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4
> Dose
   1   2   3   4   5   6   7   8   9  10  11  12
 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4
> Dose <- Dose*10
> Dose
 1 2 3 4 5 6 7 8 9 10 11 12
 1 2 3 4 1 2 3 4 1  2  3  4
> pain.relief$Dose
 [1] 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4
```

Making an assigninment to Dose created a new variable (called Dose) in the .Data area, which takes precedence over the original Dose (still available as pain.relief$Dose).

Another approach (after removing the new Dose from the .Data area) would be to change the original Dose vector. However, Dose still refers to the old pain.relief$Dose. The changes do not take effect until the next time the pain.relief data frame is attached to the search path.

```
> pain.relief <- read.table("example2",header=T)
> attach(pain.relief)
> pain.relief$Dose
 [1] 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4
> pain.relief$Dose <- Dose*10
> pain.relief$Dose
 1 2 3 4 5 6 7 8 9 10 11 12
 1 2 3 4 1 2 3 4 1  2  3  4
> Dose
   1   2   3   4   5   6   7   8   9  10  11  12
 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4
> detach()
> attach(pain.relief)
> Dose
 1 2 3 4 5 6 7 8 9 10 11 12
 1 2 3 4 1 2 3 4 1  2  3  4
```

A third alternative would be to change Dose before attaching to the data frame. In this case, the right hand side of the assignment must use pain.relief$Dose (since pain.relief is not on the search path, and Dose is not in the .Data area).

```
> pain.relief <- read.table("example2", header=T)
> pain.relief$Dose <- pain.relief$Dose * 10
> attach(pain.relief)
> Dose
 1 2 3 4 5 6 7 8 9 10 11 12
 1 2 3 4 1 2 3 4 1  2  3  4
```

The point of this confusing example is to demonstrate some complications that arise when using data frames. I prefer the second or third approach, in order to keep the data frame up-to-date and to avoid cluttering the .Data area.

Please note that the rest of this handout uses the original pain.relief data frame.

5 Selecting Parts of a Data Frame

The data in a data frame are a matrix, so you can select observations out of it using matrix notation. For instance, the element in the fifth row, second column of pain.relief is

```
> pain.relief[5,2]
[1] 20
```

The elements in the first, second and fifth rows, first and second columns are

```
> pain.relief[c(1,2,5),c(1,2)]
  Dose Water
1  0.1    10
2  0.2    10
5  0.1    20
```

This works just like selection from vectors, only now in two dimensions.

Sometimes it is important to exclude part of a data frame. For instance, suppose you discover that the third observation in the `pain.relief` data frame is an outlier. To see the `Dose` variable with the third observation removed, do the following:

```
> Dose[-3]
   1   2   4   5   6   7   8   9  10  11  12
 0.1 0.2 0.4 0.1 0.2 0.3 0.4 0.1 0.2 0.3 0.4
```

`Dose` is a one-dimensional vector, so it only needs one subscript. The `pain.relief` data frame has two dimensions, so S-PLUS will not know whether `pain.relief[-3]` means "without row 3" or "without column 3". The correct way to do it is to put a comma between the two subscripts (and no subscript means "use all rows" or "use all columns". To see `pain.relief` without row 3 but with all columns, type:

```
> pain.relief[-3,]
   Dose Water Relief
 1  0.1    10      7
 2  0.2    10     15
 4  0.4    10     23
 5  0.1    20     15
 6  0.2    20     13
 7  0.3    20     26
 8  0.4    20     38
 9  0.1    30     21
10  0.2    30     28
11  0.3    30     31
12  0.4    30     47
```

Similarly if you wanted to see all rows but without the second column you could type `pain.relief[,-2]`.

Question: How do you exclude *multiple* rows?

6 A Statistical Example with Data Frames

Ohio State University did an experiment comparing air-filled and helium-footballs. A novice punter kicked each football 39 times (alternating between the air-filled and the helium-filled ball) and the experiment recorded the distance (in yards) travelled by each ball. These data are in the file football.dat. Read that file into a data frame and attach to it.

Next open a graphics window by typing `motif()` or `x11()` on a UNIX machine or `win.start()` on a Windows machine.

A good place to start might be with summary statistics for helium footballs and regular footballs.

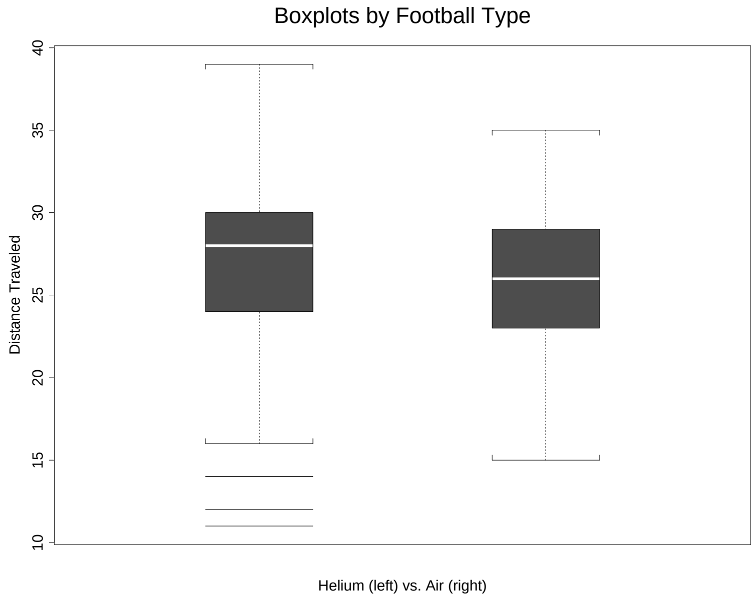
```
> summary(Helium)
 Min. 1st Qu. Median  Mean 3rd Qu. Max.
   11    24.5     28 26.38      30   39
> summary(Air)
 Min. 1st Qu. Median  Mean 3rd Qu. Max.
   15    23.5     26 25.92    28.5   35
```

It looks like the helium footballs go a little further on average. Next we could look at box plots:

```
> boxplot(Helium, Air)
```

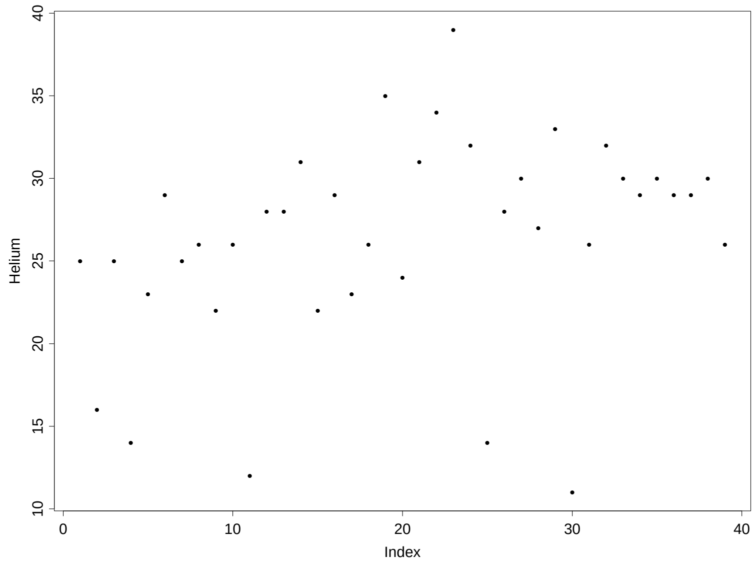
Note that when you do this, S-PLUS does not label anything. We can add some titles:

```
> title(ylab="Distance Traveled")
> title(xlab="Helium (left) vs. Air (right)")
> title("Boxplots by Football Type")
```



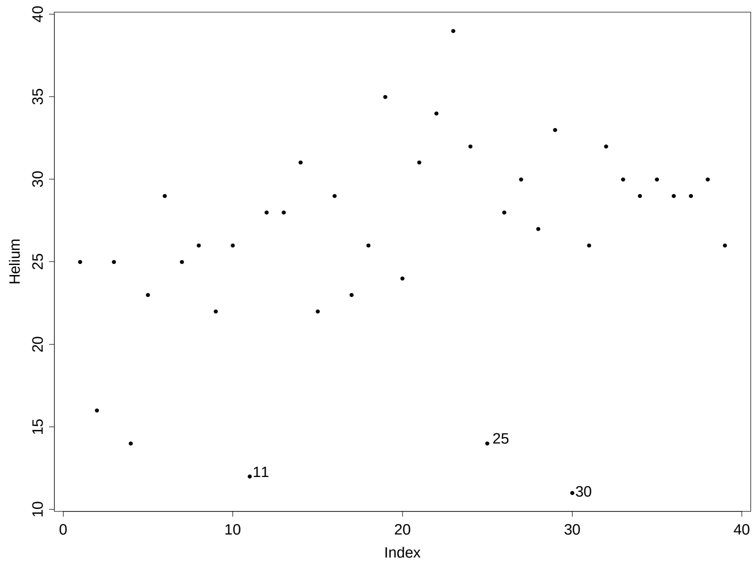
The box plot for helium-footballs shows a few outliers. Were these from when the kicker was beginning to kick the footballs and hadn't learned to kick them far yet, or were they accidental mis-kicks from later in the experiment? We can plot the helium distances in order and see what they show.

```
> plot(Helium)
```



This plot shows that there were a couple of relatively short kicks at the beginning, and then three shorts ones later which look to be outliers.

S-PLUS has a function which allows you to select points on a graph by using the mouse. Once you have plotted a graph, you can run `identify` on that graph. This allows you to identify the index of a point by clicking with the left mouse button and to stop identifying by clicking with the center mouse button (this is with a three-button mouse; if you have fewer buttons, play around and figure out how to use `identify` using your mouse). I typed `identify(Helium)` and clicked on each of the three points to get their index number.



The three outliers are observations number 11, 25, and 30. Now consider the distance the other helium footballs travelled:

```
> summary(Helium[-c(11,25,30)])
 Min. 1st Qu. Median  Mean 3rd Qu. Max.
   14      25     28 27.56      30   39
```

The median was unaffected by removing those three observations, but the mean increased slightly. It looks like helium-filled footballs travel about 1.6 yards further than regular footballs. The way to test whether this is significant is with a paired t-test (that is, the first air-kick compared to the first helium-kick, the second air-kick compared to the second helium-kick, etc.). We must exclude the 11th, 25th and 30th distances travelled by air-filled footballs in order to have matched pairs for all footballs.

```
> t.test(Helium[-c(11,25,30)],Air[-c(11,25,30)], paired=T)

         Paired t-Test

data:  Helium[ - c(11, 25, 30)] and Air[ - c(11, 25, 30)]
t = 1.4923, df = 35, p-value = 0.1446
alternative hypothesis: true mean of differences is not equal to 0
95 percent confidence interval:
 -0.5405538  3.5405538
sample estimates:
 mean of x - y
          1.5
```

The p-value is .14, so it appears that there is no significant difference in the distance travelled by helium-filled and air-filled footballs.

Now that this analysis is complete, detach the football data frame from your search path.