Objects and Data Types in S-PLUS*

Aidan Palmer

May 26, 1997

1 Introduction

S-PLUS classifies objects into many different types, such as vectors and data frames. Some S-PLUS functions treat these different types differently. For example, the function `plot` will do a scatterplot if it receives a numeric vector, a boxplot if it receives a factor, or a series of plots if it receives regression output (complicated function output, such as the output from regression, is generally in the form of a list). This flexibility is called "object oriented programming".

All objects have *values* and *attributes*. An object's value is what you see when you type the name of the object, and attributes are things like length (for vectors), dimensions (for a matrix), or row and column labels (for a data frame). The most common objects and their attributes are described below.

2 Scalars and Vectors

The simplest object in S-PLUS is a scalar. Scalars are single numbers (type "numeric") or words (type "character"). Characters must be enclosed in quotes: `"x"` means the letter x, `x` means the object x. Similarly, `1` means the number one and `"1"` means the symbol used for the number one.

A vector is a set of scalars arranged in a one dimensional array (or alternatively, scalars are vectors of length 1). Vectors can only contain one type of scalar: either numbers or characters but not both. If S-PLUS sees both in the same vector it will change the numeric arguments to characters (by enclosing them in quotes).

```
> blah <- c(1,2,3,"d")
> blah
[1] "1" "2" "3" "d"
```

In the example above, S-PLUS converted the numbers 1,2 and 3 to their respective symbols so that all four elements in `blah` would be characters.

Use subscripts to see or change the value of a particular element or elements of a vector (this is covered in "Introduction to S-PLUS").

```
> blah[1]
[1] "1"
> blah[c(1,2,3)]
[1] "1" "2" "3"
> blah[4]
[1] "d"
> blah[4] <- "zip"
> blah
[1] "1"   "2"   "3"   "zip"
```

*In writing this handout, I was helped a great deal by the S-PLUS Help Archive maintained by the University of Toronto's Department of Statistics.

Vectors have a length attribute, which can be viewed or changed by using the function `length`.

```
> length(blah)
[1] 4
> length(blah) <- 5
> blah
[1] "1" "2" "3" "zip" ""
```

Note that when `blah` was changed to length five, the blank character (`""`) was added to the end of the vector as the fifth element. Had `blah` been a numeric vector, the new element would have been the blank numeric element ("NA", for Not Available).

Question: What happens when you reduce the length of a vector?

3 Checking and Changing Types

If you are unsure of the type of an object, check it by using the set of functions that begin with `is`. There is one for almost every type of object, and they return "T" or "F" (true or false) depending on whether the object in question is of that type.

```
> is.vector(blah)
[1] T
> is.character(blah)
[1] T
> is.numeric(blah)
[1] F
```

In the above example, S-PLUS was asked whether `blah` was a vector (true), a character (true, a character vector), or a numeric (false).

S-PLUS can change objects from one type to another, but only if there is an obvious way to do so.

```
> as.numeric("1")
[1] 1
> as.numeric("hello")
[1] NA
Warning messages:
  1 missing value  generated coercing from character to numeric in: as.numeri\
        c("hello")
```

In the example above, S-PLUS knew how to change the symbol "1" into the number 1, but not how to change the word "hello" into a number. Some uses of the `as` functions will be shown in subsequent sections.

4 Factors

Factors are a very useful tool in S-PLUS. A factor is a vector where the various elements represent different levels of a variable and the relative effects of those levels are unknown.

Suppose an experiment were performed on ten patients, using three different treatments: placebo, medicine A, and medicine B. Suppose the first three received the placebo, the next four received medicine A, and the final three received medicine B. The response variable is degree of improvement. The data will consist of two vectors: which treatment a patient received and the extent to which he or she recovered.

If the medicine vector were coded as numbers, it would imply an ordering and a scale of the effects of the three treatments. For instance, coding placebo as 0, medicine A as 1 and medicine B as 2 would imply that medicine B is twice is good (or twice as bad) as medicine A, and that the difference between medicine B and medicine A is the same as the difference between medicine A and the placebo. In this experiment, we will not know the relative values of the three treatments until after the experiment is complete, so we cannot code the treatments as numbers. Instead, enter a vector of characters and convert it to a factor.

```
> medicine <- c(rep("Placebo", 3), rep("A", 4), rep("B", 3))
> medicine
 [1] "Placebo" "Placebo" "Placebo" "A"       "A"       "A"       "A"
 [8] "B"       "B"       "B"
> medicine <- as.factor(medicine)
> medicine
 [1] Placebo Placebo Placebo A       A       A       A       B       B
[10] B
```

(The function `rep(what, n)` is a shortcut which repeats the first argument `n` times.)

```
> levels(medicine)
[1] "A"       "B"       "Placebo"
```

The vector `medicine` is a factor with three levels, but no particular order to the levels. If a factor has a definite order to its levels, you can specify that by declaring it to be an "ordered factor". For example, suppose a marketing firm were trying to guage the appeal of a new movie. The data could consist of a factor for viewers' responses, and perhaps several vectors of demographic data on each viewer.

```
> responses
 [1] Great Good  Good  Poor  Great Fair  Awful Fair  Good  Fair  Poor  Great
[13] Fair  Good  Poor  Fair  Fair  Great
> levels(responses)
[1] "Awful" "Fair"  "Good"  "Great" "Poor"
```

The factor `responses` has five levels, but they should not be in alphabetical order. Use the `ordered` function to convert `responses` to an ordered factor.

```
> responses <- ordered(responses, levels=c("Awful", "Poor", "Fair", "Good", "Great"))
> responses
 [1] Great Good  Good  Poor  Great Fair  Awful Fair  Good  Fair  Poor  Great
[13] Fair  Good  Poor  Fair  Fair  Great

 Awful < Poor < Fair < Good < Great
```

Now S-PLUS understands that there is an ordering to the data, with "Awful" being the lowest level and "Great" being the highest.

WARNING: Do not try to change the order of the data by making an assignment to the `levels` attribute. Always change the order of the data by using the `ordered` function.

The only reason to make an assignment to the `levels` attribute is to change the labels on the levels.

```
> medicine
 [1] Placebo Placebo Placebo A       A       A       A       B       B
[10] B
> levels(medicine)
```

```
[1] "A"       "B"       "Placebo"
> levels(medicine) <- c("Medicine A", "Medicine B", "Placebo")
> medicine
 [1] Placebo    Placebo    Placebo    Medicine A Medicine A Medicine A
 [7] Medicine A Medicine B Medicine B Medicine B
```

> **Question:** Generate a vector, `moreresponses`, which consists of "Best Movie Ever" repeated twice and "Worst Movie Ever" repeated three times. Make it an ordered factor, so that "Worst Movie Ever" is the lower of the two levels. Append `moreresponses` to `responses`. What does this new vector look like? Do some editing to make "Worst Movie Ever" the lowest level (below "Awful") and "Best Movie Ever" the highest (above "Great"). Change the labels on the levels, if necessary.

5 Matrices

A matrix is like a vector, but in two dimensions. To enter a matrix in S-PLUS, enter a vector of data and the dimensions of the matrix. For example:

```
> matrix(c(1:9), nrow=3,ncol=3,byrow=T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

In the above example, the "by row" argument tells S-PLUS to read the data in by row. The default is to read it by column, which would have resulted in:

```
> matrix(c(1:9),nrow=3,ncol=3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Vectors can be converted into matrices, by using the `as.matrix` function. This function changes a vector of length n into an n by 1 matrix. The `as.vector` function can be used to change a matrix into a vector (by reading down the columns, from left to right).

Matrices have two attributes: the dimensions of the matrix, and the names for the rows and columns.

```
> bulls <- matrix(c("Ron", "Michael", "Scottie", "Dennis", "Luc", "Steve", "Toni",
"Jud", "John", "Bill"), nrow=2, ncol=5, byrow=T)
> bulls
        [,1]      [,2]      [,3]      [,4]     [,5]
[1,] "Ron"   "Michael" "Scottie" "Dennis" "Luc"
[2,] "Steve" "Toni"    "Jud"     "John"   "Bill"
```

To check the dimensions of a matrix, use the function `dim`.

```
> dim(bulls)
[1] 2 5
```

Use the function `dimnames` to see or change the row and column names. Note that dimension names must be entered as a list (see next section).

```
> dimnames(bulls)
NULL
> dimnames(bulls) <- list(c("Starters","Bench"),c("PG","SG","SF","PF","C"))
> bulls
              PG       SG        SF        PF       C
Starters "Ron"   "Michael" "Scottie" "Dennis" "Luc"
   Bench "Steve" "Toni"    "Jud"     "John"   "Bill"
> dimnames(bulls)
[[1]]:
[1] "Starters" "Bench"

[[2]]:
[1] "PG" "SG" "SF" "PF" "C"
```

6 Lists

Lists are more powerful than vectors, because they can hold multiple types of things. A vector is a one-dimensional array of scalars, while a list is a one-dimensional array of *objects*. For example, this list contains a character vector, a numeric matrix, and a numeric vector:

```
> blah <- list(c("1","two","three","4"),matrix(c(1,2,3,4,5,6),nrow=2,ncol=3),c(1,7,17))
> blah
[[1]]:
[1] "1"     "two"   "three" "4"

[[2]]:
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

[[3]]:
[1]  1  7 17
```

Selecting elements from a list requires two separate subscripts, unlike vectors (which need one) or matrices (which need a pair). The first subscript (in double brackets) tells which part of the list to select, and the second (in single brackets) tells which element of that part to select.

```
> blah[[1]]
"1"     "two"   "three" "4"
> blah[[1]][3]
[1] "three"
> blah[[2]][1:2,2]
[1] 3 4
```

Lists have a names attribute:

```
> names(blah) <- c("some.characters","a.matrix","some.numbers")
> blah
$some.characters:
[1] "1"     "two"   "three" "4"

$a.matrix:
     [,1] [,2] [,3]
[1,]    1    3    5
```

```
[2,]    2    4    6

$some.numbers:
[1]  1  7 17
```

Once you assign names to a list, you can select parts of the list using either the name or double brackets.

```
> blah$some.characters[4]
[1] "4"
> blah[[1]][4]
[1] "4"
```

7 Data Frames

At this point you probably recognize the many similarities between data frames and the other types of objects in S-PLUS. Data frames have dimension, row names, and names attributes. Blanks in data frames are represented by NA, just like blank entries in numeric vectors (and factors).

Matrices may be converted to data frames by using the `data.frame` command (there's no `as.frame`). Data frames may be converted to matrices with `as.matrix`, although keep in mind that matrices, like vectors, can only contain one type of data while data frames can contain several. If there are any characters in the data frame, converting it to a matrix will result in all of the elements being converted to characters.

8 An Example, Using Data Frames and Factors

The following data are from the New York Choral Society of 1979. Each singer reported his or her height to the nearest inch. The objective is to examine how height varies across voice range. The voice parts, from lowest to highest, are Bass, Tenor, Alto and Soprano. The lowest two parts are sung by men, the highest two by women. (These data are available in the file `/afs/andrew/stat/data/S-Tutorials/singers.dat`).

```
Soprano Alto    Tenor   Bass
64      65      69      72
62      62      72      70
66      68      71      72
65      67      66      69
60      67      76      73
61      63      74      71
65      67      71      72
66      66      66      68
65      63      68      68
63      72      67      71
67      62      70      66
65      61      65      68
62      66      72      71
65      64      70      73
68      60      68      73
65      61      73      70
63      66      66      68
65      66      68      70
62      66      67      75
65      62      64      68
```

```
66      70          71
62      65          70
65      64          74
63      63          70
65      65          75
66      69          75
65      61          69
62      66          72
65      65          71
66      61          70
65      63          71
61      64          68
65      67          70
66      66          75
65      68          72
62                  66
                    72
                    70
                    69
```

8.1 Reading and Organizing the Data

The first difficulty is that the data file is not arranged in a table. It is in four columns (Soprano, Alto, Tenor, Bass), but the columns are not of equal length. This means that `read.table` will not be able to read the data. Instead, use `scan`, skipping over the column labels.

```
> height <- scan("singers.dat",skip=1)
> height
  [1] 64 65 69 72 62 62 72 70 66 68 71 72 65 67 66 69 60 67 76 73 61 63 74 71 65
 [26] 67 71 72 66 66 66 68 65 63 68 68 63 72 67 71 67 62 70 66 65 61 65 68 62 66
 [51] 72 71 65 64 70 73 68 60 68 73 65 61 73 70 63 66 66 68 65 66 68 70 62 66 67
 [76] 75 65 62 64 68 66 70 71 62 65 70 65 64 74 63 63 70 65 65 75 66 69 75 65 61
[101] 69 62 66 72 65 65 71 66 61 70 65 63 71 61 64 68 65 67 70 66 66 75 65 68 72
[126] 62 66 72 70 69
```

Now the data are in S-PLUS, but are not arranged in any useful way. Analyzing this data set would be made easier if it were represented as two vectors, one for height and one for corresponding singing part. Since the height vector is jumbled, this could be tricky but there are two approaches to the problem. The first is to sort the height vector so that the sopranos come first, with the altos, tenors, and basses to follow. The second is to leave the height vector as it is, and generate the singing part vector in a way that accounts for the ragged columns in the data.

The second way looks easier, so let's give that a shot. The first step is to generate a vector for singing part, which we can fill in as we go.

```
> part <- rep("", 130)
```

The data are in four columns for the first 20 lines, and the height vector was formed by reading across the data. This means that the first, fifth, ninth, etc. entries are sopranos, the second, sixth, tenth, etc. are altos and so on. In S-PLUS, "a:b" means "the integers from a to b", and we can use this shortcut to generate the indices which we need. The subscript 1 + 0:19*4 will pick out the first, fifth, ninth, etc. through 77th entries (which are sopranos).

```
> part[1 + 0:19*4] <- "soprano"
> part[2 + 0:19*4] <- "alto"
> part[3 + 0:19*4] <- "tenor"
```

```
> part[4 + 0:19*4] <- "bass"
> part
  [1] "soprano" "alto"    "tenor"   "bass"    "soprano" "alto"    "tenor"
  [8] "bass"    "soprano" "alto"    "tenor"   "bass"    "soprano" "alto"
 [15] "tenor"   "bass"    "soprano" "alto"    "tenor"   "bass"    "soprano"
 [22] "alto"    "tenor"   "bass"    "soprano" "alto"    "tenor"   "bass"
 [29] "soprano" "alto"    "tenor"   "bass"    "soprano" "alto"    "tenor"
 [36] "bass"    "soprano" "alto"    "tenor"   "bass"    "soprano" "alto"
 [43] "tenor"   "bass"    "soprano" "alto"    "tenor"   "bass"    "soprano"
 [50] "alto"    "tenor"   "bass"    "soprano" "alto"    "tenor"   "bass"
 [57] "soprano" "alto"    "tenor"   "bass"    "soprano" "alto"    "tenor"
 [64] "bass"    "soprano" "alto"    "tenor"   "bass"    "soprano" "alto"
 [71] "tenor"   "bass"    "soprano" "alto"    "tenor"   "bass"    "soprano"
 [78] "alto"    "tenor"   "bass"    ""        ""        ""        ""
 [85] ""        ""        ""        ""        ""        ""        ""
 [92] ""        ""        ""        ""        ""        ""        ""
 [99] ""        ""        ""        ""        ""        ""        ""
[106] ""        ""        ""        ""        ""        ""        ""
[113] ""        ""        ""        ""        ""        ""        ""
[120] ""        ""        ""        ""        ""        ""        ""
[127] ""        ""        ""        ""
```

The next 15 lines of the data file were in three columns, since there were only 20 tenors but at least 35 of every other group. The sopranos are numbers 81, 84, 87, etc.

```
> part[81 + 0:14*3] <- "soprano"
> part[82 + 0:14*3] <- "alto"
> part[83 + 0:14*3] <- "bass"
```

The next row of the data consists of one soprano and one bass, and the last three rows have basses only. That means the 126th is a soprano and the rest are basses.

```
> part[126] <- "soprano"
> part[127:130] <- "bass"
```

Now we have the two vectors that we need for the data frame: height and corresponding singing part. Let's put these data into a 130 by 2 matrix:

```
> temp.matrix <- matrix(c(height, part), nrow=130, ncol=2)
> rm(height)
> rm(part)
```

Note that in the matrix, height has been converted to characters because singing part was a character vector. This will have to be fixed.

Now that we have the data in a matrix, we can convert it to a data frame and put the two columns in the form that we will want:

```
> singers.frame <- data.frame(temp.matrix)
> rm(temp.matrix)
> names(singers.frame) <- c("Height","Part")
> singers.frame$Height <- as.numeric(singers.frame$Height)
> singers.frame$Part <- ordered(as.factor(part), levels=c("bass","tenor",
"alto","soprano"))
> singers.frame
   Height    Part
```

```
 1      64 soprano
 2      65    alto
 3      69   tenor
 4      72    bass
 5      62 soprano
 6      62    alto
 7      72   tenor
 8      70    bass
 9      66 soprano
10      68    alto
11      71   tenor
12      72    bass
13      65 soprano
14      67    alto
15      66   tenor
16      69    bass
17      60 soprano
18      67    alto
19      76   tenor
20      73    bass
...
```

We could also have made the vectors into a data frame via:

```
> singers.frame <- data.frame(list(height, part))
```

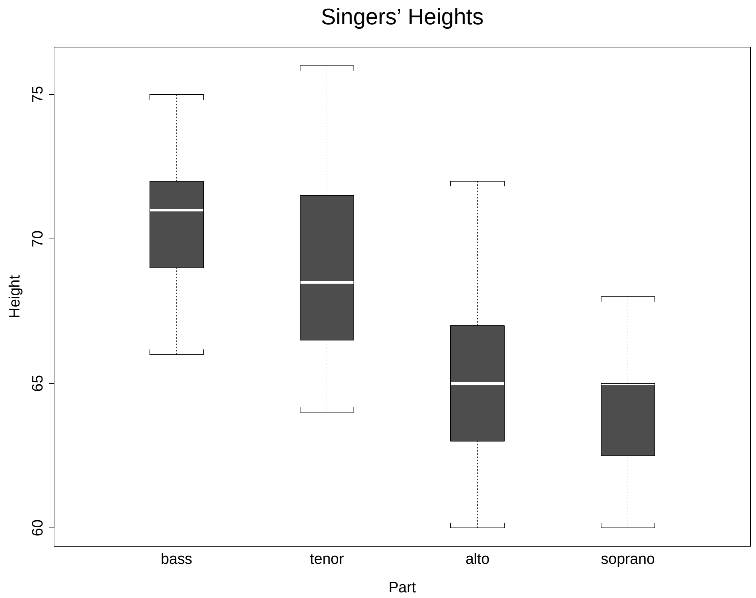
This would have preserved `height` as a numeric vector.

> **Question:** If you wanted to convert the raw data to a cleaner format in the first way suggested (by putting the `height` vector in order by singing part), how would you do it?

8.2 The Analysis

First, some boxplots.

```
> motif()
> attach(singers.frame)
> plot(Part, Height)
> title(main="Singers' Heights")
```



This shows that people who sing the lower parts are generally taller. Of course, part of this is the fact that men sing bass and tenor while women sing alto and soprano. So the interesting comparisons are between bass and tenor, and between alto and soprano.

Note that if `part` had not been coded as an ordered factor the boxplots would have been in alphabetical order. If it had not been coded as a factor at all, S-PLUS would have made dotplots instead of boxplots.

```
> t.test(Height[Part=="soprano"],Height[Part=="alto"])

        Standard Two-Sample t-Test

data:  Height[Part == "soprano"] and Height[Part == "alto"]
t = -1.1289, df = 69, p-value = 0.2628
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.7591030  0.4876745
sample estimates:
 mean of x mean of y
     64.25  64.88571

> t.test(Height[Part=="tenor"],Height[Part=="bass"])

        Standard Two-Sample t-Test

data:  Height[Part == "tenor"] and Height[Part == "bass"]
t = -2.1297, df = 57, p-value = 0.0375
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.04220979 -0.09368765
sample estimates:
 mean of x mean of y
     69.15  70.71795
```

These t-tests show that there is no significant difference in mean height between sopranos and altos, but that mean height for basses is significantly higher than mean height for tenors.