9.0 Data Frames in `Splus`

Notes based on those of Professor Joe Schafer

Main subjects covered today:

- Data Frames

9.1 Data Frames

Data frames are a relatively new part of the S language. They were introduced in 1991 and are described in the book by Chambers and Hastie (1992), "Statistical Models in S."

The data frame is based on the idea that many datasets in statistics can be arranged as rectangular arrays, with the rows representing individual sample units and the columns representing variables. So, in appearance, a data frame is similar to a matrix. But it is more general than a matrix in that the columns or variables can have different storage modes. For example, we can have a data frame in which one column contains numbers, another column contains logical values (T or F) and another column contains character strings. (For those of you who are familiar with SAS, a data frame is very much like a SAS dataset.)

If x1, x2, x3, ... are vectors of the same length representing different variables, we can glue them together into a data frame using the `data.frame()` function. The vectors need not have the same storage mode.

```
> x1 <- c(100,99,100,20)
> x2 <- c(20,19,19,10)
> x3 <- c("A","A","A","C")
```

```
> grades <- data.frame(x1,x2,x3)
> grades
   x1 x2 x3
1 100 20  A
2  99 19  A
3 100 19  A
4  20 10  C
```

9.2 Data frames as matrices

A data frame can be regarded as a cross between a matrix and a list. First, let's see how a data frame resembles a matrix.

Data frames, like matrices, have "dim", "nrow", "ncol", and "dimnames" attributes:

```
> dim(grades)
[1] 4 3
> nrow(grades)
[1] 4
> ncol(grades)
[1] 3
> dimnames(grades)
[[1]]:
[1] "1" "2" "3" "4"

[[2]]:
[1] "x1" "x2" "x3"
```

We can change the row and/or column labels in the same way that we did for matrices.

```
> dimnames(grades) <- list(c("Linda","John","Brian","Mikhail"),
+ c("homework","exam","final"))
> grades
          homework exam final
    Linda      100   20     A
```

```
     John       99   19     A
    Brian      100   19     A
  Mikhail       20   10     C
```

As with matrices, we can refer to subsets of the data frame with square brackets:

```
> grades[1:2,]
      homework exam final
Linda      100   20     A
 John       99   19     A
> grades[,1]
[1] 100  99 100  20
> grades["homework"]
[1] 100  99 100  20
> grades[,-1]
        exam final
  Linda   20     A
   John   19     A
  Brian   19     A
Mikhail   10     C
```

When we pull a single column out of a data frame, as in `grades[,1]`, the result is a vector. When we pull out a single row, however, as in `grades[1,]`, the result is a data frame with only one row. A single row is a data frame rather than a vector because it might have mixed storage modes.

The function `data.frame()` can also be used to turn a matrix into a data frame. If the matrix has no row labels, then `data.frame()` will label the rows as "1", "2", ... If the matrix has no column labels, then `data.frame()` will label the columns as "X1", "X2", etc.

```
> v <- matrix(rnorm(12),4,3)
> data.frame(v)
          X1         X2         X3
1  1.3289191  1.5826525  0.5284931
```

```
2 -0.5834504  0.7756624 -0.5783929
3  0.1294323 -0.1010787 -0.7104860
4  0.9805573 -0.8096563  0.1749112
```

9.3 Data frames as lists

A data frame is also a list. Each variable (column) is an element of the list. Like a list, we can refer to the variables by their positions using double square brackets ([[]]) or by their names using a dollar sign ($).

```
> grades[[1]]
[1] 100  99 100  20
> grades[[3]]
[1] A A A C
> grades$final
[1] A A A C
```

The attributes "length" and "names" are the same as they would be for a list.

```
> length(grades)
[1] 3
> names(grades)
[1] "homework" "exam"     "final"
```

A data frame is not exactly the same thing as a list, however. A data frame is a special kind of list in which the elements are all vectors of the same length.

9.4 Reading in data from external files

In lecture 5, we learned how to read in data from an external file using the `scan()` function. The `scan()` function by default returns the data set as a vector, which then can be reshaped into a matrix if needed. There is another command, `read.table()`, which can read a data matrix from a file and turn it into a data frame. The `read.table()` function is very "intelligent" because it can automatically handle numeric data and character data at the same time. It can also handle row labels and column labels automatically. By default, `read.table()` assumes that each record (i.e. each line of the data file) represents a single row of the data frame, and it assumes that the fields (i.e. variables) within a record are separated by blank spaces and/or tabbing characters.

For example, suppose we have a file called "grades.dat":

```
Linda      100   20     A
John        99   19     A
Brian      100   19     A
Mikhail     20   10     C
```

See what happens when we apply `read.table()` to this file:

```
> grades <- read.table("grades.dat")
> grades
         V2 V3 V4
  Linda 100 20  A
   John  99 19  A
  Brian 100 19  A
Mikhail  20 10  C
```

Splus automatically interprets the first non-numeric field as a row label. If you don't want that to happen, you can use the option

row.names=NULL:

```
> read.table("grades.dat",row.names=NULL)
       V1  V2 V3 V4
1   Linda 100 20  A
2    John  99 19  A
3   Brian 100 19  A
4 Mikhail  20 10  C
```

Suppose that your file also has column names in it, like this:

```
    HW  Exam  Final
Linda      100   20     A
John        99   19     A
Brian      100   19     A
Mikhail     20   10     C
```

The option header=T will tell `read.table()` to interpret the first record as column labels:

```
> read.table("grades.dat",header=T)
         HW Exam Final
  Linda 100   20     A
   John  99   19     A
  Brian 100   19     A
Mikhail  20   10     C
```

The `read.table()` function is quite versatile. It has an optional argument called "sep" that allows you to read in data in fixed format, much as you would in SAS or Fortran. See help(read.table) for details.

There is also a function that can convert a SAS dataset into an Splus dataframe. The function is called `sas.get()`. See help(sas.get) for details.

9.5 Data frames as databases

We have already seen that a data frame acts like a matrix and a list. It can also act like a database. Recall that a database is a directory (such as .Data) that contains objects. We can add databases to the search list using the `attach()` function. It turns out that the `attach()` function can also attach a data frame to the search list.

Before going on, let's see why it might be advantageous to attach a data frame to the search list. Let's work with one of the sample datasets that comes with Splus. The dataset is called auto.stats, and it is located in one of the databases on your search list. If you type

```
> auto.stats <- auto.stats
```

then Splus will make a copy of this dataset and put it in .Data. It is a matrix with 74 rows and 12 columns:

```
> dim(auto.stats)
[1] 74 12
```

This dataset happens to be a matrix, not a data frame:

```
> is.data.frame(auto.stats)
[1] F
> is.matrix(auto.stats)
[1] T
```

So let's convert it to a data frame:

```
> auto.stats <- data.frame(auto.stats)
```

The variable names are:

```
> dimnames(auto.stats)[[2]]
[1] "Price"    "Miles.per.gallon" "Repair..1978." "Repair..1977."
```

```
[5] "Headroom" "Rear.Seat"        "Trunk"         "Weight"
[9] "Length"   "Turning.Circle"   "Displacement"  "Gear.Ratio"
```

Suppose that we want to make a scatterplot of "Weight" versus "Miles.per.gallon". We would do it as follows:

```
> motif()
> plot(auto.stats$Weight, auto.stats$Miles.per.gallon)
```

If we are going to analyze this dataset, it will become rather tedious to type "auto.stats" every time we want to refer to a variable. To get around this problem, we can attach the auto.stats data frame to the search list using `attach()`. We will attach it in position 1:

```
> attach(auto.stats,pos=1)
> search()
[1] "auto.stats"
[2] ".Data"
[3] "/usr/local/splus-3.3/splus/.Functions"
[4] "/usr/local/splus-3.3/stat/.Functions"
[5] "/usr/local/splus-3.3/s/.Functions"
[6] "/usr/local/splus-3.3/s/.Datasets"
[7] "/usr/local/splus-3.3/stat/.Datasets"
[8] "/usr/local/splus-3.3/splus/.Datasets"
```

Now if we type "`ls()`", we will see the individual variables in this data frame:

```
> ls()
 [1] ".Last.value"    "Displacement"     "Gear.Ratio"   "Headroom"
 [5] "Length"         "Miles.per.gallon" "Price"        "Rear.Seat"
 [9] "Repair..1977."  "Repair..1978."    "Trunk"        "Turning.Circle"
[13] "Weight"
```

We can now refer to these variables by name. They will be regarded as individual vectors. For example, we can re-create the scatterplot:

```
> plot(Weight,Miles.per.gallon)
```

Suppose we want to create new variables by transforming the old ones. For example:

```
> Gallons.per.mile <- 1/Miles.per.gallon
> plot(Weight,Gallons.per.mile)
```

After we are finished, we can detach the data frame. If we use the save option when we detach it, then any changes that we made to the variables in the data frame and any new variables that we created will be saved as well:

```
> detach(1,save="auto.stats")
```

Now if we type `ls()`, we see a list of the objects stored in .Data, including the updated auto.stats data frame.

One word of caution: A data frame is a special kind of database in which all the objects must be vectors of the same length. If we had created other types of objects (e.g. scalars) while the data frame was attached, these would not have been saved when we detached it.