

BMI 713: Computational Statistics for Biomedical Sciences

Lab: Working with data files

October 14, 2010

Data frames

Data frames are generalizations of matrices. Unlike a matrix, which can only store data of a single type (e.g., strings, numbers, and booleans), data frames can store many types of data in the same object.

To create a new data frame, use the `data.frame` function. The function takes one or more vectors $v_1, v_2, \dots, v_n$ of the same length and creates a data frame whose columns are $v_1, v_2, \dots, v_n$. For example¹

```
> x
[1] 6 8 19 21 17
> y
[1] FALSE TRUE FALSE FALSE FALSE
> z
[1] "t" "l" "e" "n" "l"
> df <- data.frame(x, y, z, stringsAsFactors=FALSE)
> df
      x      y z
1  6 FALSE t
2  8  TRUE l
3 19 FALSE e
4 21 FALSE n
5 17 FALSE l
```

As we can see, `df` remembers the names of the variables that were used to create each column; `df`'s columns are thus named `x`, `y` and `z`. Different names can be specified when `data.frame` is called:

```
> df <- data.frame(Number=x, Truth=y, Code=z, stringsAsFactors=FALSE)
> df
  Number Truth Code
1      6 FALSE   t
2      8  TRUE   l
3     19 FALSE   e
4     21 FALSE   n
5     17 FALSE   l
```

Row and column names can be changed after the data frame is created by using `rownames` and `colnames`:

```
> colnames(df)
[1] "Number" "Truth"  "Code"
> colnames(df) <- c("A", "B", "C")
> df
   A      B C
1  6 FALSE t
2  8  TRUE l
3 19 FALSE e
4 21 FALSE n
5 17 FALSE l
```

¹Here we set the `stringsAsFactors` parameter to `FALSE`. If this is not set, then any column of strings will be transformed into a column of factors. Despite the similarity between the two types of data, factors and strings often behave very differently; accidentally using a factor in place of a string can produce very confusing results. We will always use this option when working with data frames.

Similar to a matrix, the `[]` operators can be used to select rows and columns from a data frame.

```
> df[1,]  
  A      B C  
1 6 FALSE t
```

Note that the result of selecting row 1 from `df` has produced *another* data frame—not a vector like one would expect from a matrix. On the other hand, selecting a column produces a vector:

```
> df[,1]  
[1] 6 8 19 21 17
```

In addition to the square brackets, a convenience operator `$` can be used to select columns from a data frame by name. For example,

```
# These two commands are equivalent  
> df$A          # Select the column named "A"  
[1] 6 8 19 21 17  
> df[,1]        # Select column number 1  
[1] 6 8 19 21 17
```

Reading and writing data

.RData files

.RData files are an R-specific storage format for storing R variables. Unlike other file reading functions we will see, `load` creates variables in the workspace *without* requiring the assignment operator (`<-`). For example, if we have vectors `x`, `y` and `z`:

```
> x  
[1] 6 8 19 21 17  
> y  
[1] FALSE TRUE FALSE FALSE FALSE  
> z  
[1] "t" "l" "e" "n" "l"
```

we can save them to an .RData file with

```
> save(x, y, z, file="xyz.RData")
```

Now the variables `x`, `y` and `z` can be loaded into a different R instance with `load`:

```
> ls()  
character(0)          # A new R instance has been started  
> load("xyz.RData")  
> ls()                # Now variables x, y and z exist...  
[1] "x" "y" "z"  
> x                   # ...and have the same values as before.  
[1] 6 8 19 21 17  
> y  
[1] FALSE TRUE FALSE FALSE FALSE  
> z  
[1] "t" "l" "e" "n" "l"
```

For example, when R saves your workspace, it creates an .RData file by invoking `save.image`. While these storage formats are very convenient for R users, they are certainly not ideal for general analysis.

Delimited text files

Delimited text files are a very common data storage format. Typically one or more data tables are stored one row per line in plain, human readable text. Each line is split into columns by a special symbol called the *delimiter*. One common format is the *comma-separated values* (*.csv*) format, in which the delimiter is the comma `,`.

To write a data frame or matrix in delimited, plain text format to a file, one can use `write.table`. This function will create a delimited text file whose format will vary depending on the parameters to `write.table`. To write the `df` data frame to file, we might use:

```
> write.table(df, file="df.txt")
```

The contents of the file `df.txt` are now

```
"x" "y" "z"
"1" 6 FALSE "t"
"2" 8 TRUE  "l"
"3" 19 FALSE "e"
"4" 21 FALSE "n"
"5" 17 FALSE "l"
```

Notice that the delimiter in this case is a space. Also notice that the row and column names of the data.frame have been written. We can, for example, opt to change the delimiter and exclude row names by using:

```
> write.table(df, file="df.txt", sep=",", row.names=FALSE)
```

The file `df.txt` now contains

```
"x","y","z"
6,FALSE,"t"
8,TRUE,"l"
19,FALSE,"e"
21,FALSE,"n"
17,FALSE,"l"
```

`write.table` supports a number of different parameters; see `?write.table` for reference.

To read data from a delimited text file into R, we may use `read.table`. Its parameters are very similar to `write.table`—see `?read.table` for reference.

```
> df2 <- read.table("df.txt", header=TRUE, sep=",")
> df2
      x      y z
1    6 FALSE t
2    8  TRUE l
3   19 FALSE e
4   21 FALSE n
5   17 FALSE l
```

To read data from `df.txt`, it is important to specify the correct delimiter (in this example, a comma). We also specified `header=TRUE` to force R to treat the first line of the file as column names *rather* than data in the table. The value returned by `read.table` is a data frame.