

## Dealing With Data Frames in R

- A. What Is a Data Frame?
- B. How To Create a Data Frame
- C. How To Get a Data Frame Into R
- D. How To Look Inside a Data Frame
- E. How To Work With Variables Inside a Data Frame

### A. What Is a Data Frame?

Data frames are the primary, i.e., most important, data structure in R. They are tables of data in which every case (for now think of a case as being the same as a subject) gets its own row and every variable gets its own column. This is very similar to the way SPSS, SAS, and other statistical packages *require* you to enter data. R does not require that your data be in the form of a data frame, but data frames are very common and probably the most useful form in which to enter data sets of any substantial size.

Let's say we've collected data on one response variable or DV from 15 subjects, who were divided into three experimental groups called control ("contr"), treatment one ("treat1"), and treatment two ("treat2"). We might be tempted to table the data as follows...

contr	treat1	treat2
22	32	30
18	35	28
25	30	25
25	42	22
20	31	33

While this is a perfectly acceptable table, it is NOT a data frame, because values on our one response variable have been divided into three columns (and so have values on the grouping or independent variable). A data frame has the name of the variable at the top of the column (not the "name" of a group), and values of that variable in the column under the variable name, as follows...

scores	groups
22	contr
18	contr
25	contr
25	contr
20	contr
32	treat1
35	treat1
30	treat1
42	treat1
31	treat1
30	treat2
28	treat2
25	treat2
22	treat2
33	treat2

This is a proper data frame (and leave out the dashed lines, although in fact R could read this table just as you see it here). It does not matter in what order you type the columns, as long as each column contains values of one variable, and every recorded value of that variable is in that column.

## B. How To Create a Data Frame

Small data frames can be created right in the R Console at the command line, but data frames of any substantial size are best created using spreadsheet software. This is described in a separate handout. Note: in that handout it was recommended that every data frame created for use in R have a column containing a subject identifier. The example above does not. Since every line in the above data frame represents a different subject, technically a subject identifier is not required. However, it is a good habit to include a column in every data frame that is a subject ID. If you do not, R will create one for you when it reads in the data.

## C. How To Get a Data Frame Into R

R comes with many "built-in" datasets for educational purposes, and many of these are in the form of data frames. For our first lesson in data frames, we'll make use of one of those, called "ToothGrowth", which contains data on the effect of vitamin C supplements on tooth growth in guinea pigs. Since the data frame is "built-in" and, hence, already present in R, all we have to do is copy it into the workspace.

```
> data(ToothGrowth)      # copy the ToothGrowth data frame into the workspace
> ls()
[1] "ToothGrowth"
```

Notice once again that R did not confirm that the data frame had been copied successfully. It just did what it was told to do silently. To confirm that the data frame is in the workspace, we had to ask.

If you have created a data file (or someone else has) using an external software package such as a spreadsheet and you now want to read it into R from the working directory, you need to know a little bit about the structure of this file. Recall the "ccudrivers.csv" data file we created in the previous handout. I've dropped a copy into my working directory, which is...

```
> getwd()
[1] "C:/Documents and Settings/kingw/My Documents"
```

And I can confirm that it's there using...

```
> dir()
[1] "ccudrivers.csv"           "Curriculum Vitae.doc"
[3] "Cyberlink"               "DESKTOP.INI"
[5] "documents"               "downloads"
[remainder of output suppressed]
```

Since I created this file, I know it has a subject identifier in the first column, so I am going to use that first column as row names when I read the data into R.

```
> Drive = read.csv(file="ccudrivers.csv", row.names=1)
> ls()
[1] "Drive"           "ToothGrowth"
```

Notice that the data frame had to be assigned to a data object, in this case called "Drive". If we had failed to do so, the data inside the file would simply have been printed to the R Console and that would have been the end of it. If you see that happening (it happens to me sometimes), you'll know you've forgotten to do the assignment, and you'll have to try again.

## D. How To Look Inside a Data Frame

We now have two data frames in our workspace to play with. Let's look a little bit at ToothGrowth. To see the entire data frame, all you have to do is type its name at the command prompt.

```
> ToothGrowth      # output not shown
```

The output is not shown because it is fairly long, 61 lines of it to be exact. We could have found that out in advance by asking for the size (dimensions) of the data frame.

```
> dim(ToothGrowth)
[1] 60  3
```

ToothGrowth contains 60 rows of data (and a header row, or row of variable names) arranged in 3 columns. We now know enough about ToothGrowth to know that simply printing it will cause the top of it to scroll off the Console window. (You can always scroll back to see it.) It might be wiser to ask to see just the top, or head, of the data frame.

```
> head(ToothGrowth)      # see the first six lines of data and the headers
  len supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5
4  5.8   VC  0.5
5  6.4   VC  0.5
6 10.0   VC  0.5
```

From this we get an idea of what the data look like. There is a header line that gives the names of the variables in the columns: len (tooth length), supp (type of supplement given), and dose (dose of supplement given). The first column contains row names, which R has created in this case, since they were not in the original data file. In that case, R simply numbers the lines from 1 to whatever. These "numbers" are NOT row numbers, however. (Remember this!) They are names. It's as if the rows were named "one", "two", "three", etc. DO NOT MISTAKE ROW NAMES FOR ROW NUMBERS! Notice also that the row names do not count in the dimensions of the data frame.

We can also look at the bottom, or tail, of the data frame.

```
> tail(ToothGrowth)     # see the last six lines of data and the headers
  len supp dose
55 24.8   OJ   2
56 30.9   OJ   2
57 26.4   OJ   2
58 27.3   OJ   2
59 29.4   OJ   2
60 23.0   OJ   2
```

Another useful function for seeing what's in a data frame is `summary()`.

```
> summary(ToothGrowth)
      len      supp      dose
Min.   : 4.20   OJ:30   Min.   :0.500
1st Qu.:13.07   VC:30   1st Qu.:0.500
Median :19.25                Median :1.000
Mean   :18.81                Mean   :1.167
3rd Qu.:25.27                3rd Qu.:2.000
Max.   :33.90                Max.   :2.000
```

This produces descriptive statistics on each variable inside the data frame. If the variable is numerical, such as `len` and `dose`, then the minimum value, first quartile, median, mean, third quartile, and maximum value are reported. If the variable is categorical, such as `supp`, then a frequency table is reported.

One more function that I use a lot simply prints the names of the variables in the data frame.

```
> names(ToothGrowth)
[1] "len" "supp" "dose"
```

You can also use `dimnames()`, which will report the row names as well.

```
> dimnames(ToothGrowth)
[[1]]
 [1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" "13" "14" "15"
[16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
[31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42" "43" "44" "45"
[46] "46" "47" "48" "49" "50" "51" "52" "53" "54" "55" "56" "57" "58" "59" "60"

[[2]]
[1] "len" "supp" "dose"
```

## E. How To Work With Variables Inside a Data Frame

When you read in a data file, R does NOT put the resulting data frame in the search path.

```
> search()
[1] ".GlobalEnv"          "package:stats"      "package:graphics"
[4] "package:grDevices"  "package:utils"     "package:datasets"
[7] "package:methods"    "Autoloads"         "package:base"
```

This means you cannot simply refer to variables inside the data frame by name and expect R to know what you're talking about.

```
> mean(len)          # asking for mean tooth length from the ToothGrowth data frame
Error in mean(len) : object 'len' not found
```

There is a good reason for this. R allows you to work with more than one data frame at a time. (Most stat packages do not.) Thus, R is very careful about making you specify where your variables are coming from. There are three ways to do this, two of which I'll show you now.

The first is to use the `$` notation (technically called list indexing). This involves typing the name of the

data frame, followed by a \$, followed by the name of the variable inside the data frame. There may or may not be spaces around the \$ as you please.

```
> mean(ToothGrowth$len)      # use list indexing to refer to variables in a d.f.
[1] 18.81333
> mean(ToothGrowth $ len)    # as usual, spacing is okay or optional
[1] 18.81333
```

The second is to use a function called `with()`, which is illustrated below.

```
> with(ToothGrowth, mean(len))
[1] 18.81333
```

Inside the `with()` function, type the data frame name, a comma, and then type the operation you wish to carry out. Don't forget to close the parentheses on the `with()` function.

Both of these methods involve extra typing, and who wants that? So there are two ways around it, both of which can be messy. The first is to copy the variable of interest into the workspace and work with it there. This helps especially if you plan to do a lot of analysis with the variable.

```
> len = ToothGrowth$len
> ls()
[1] "Drive"      "len"        "ToothGrowth"
> mean(len)
[1] 18.81333
```

I do this sometimes. You have to be careful, however, because if you already had something called "len" in your workspace, it has now been overwritten. That is, you just erased the old "len" and replaced it with the new one.

The second is to attach the data frame to the search path using the `attach()` function. If you do this, make sure your workspace is clean. Otherwise, you may get a warning message like the following.

```
> attach(ToothGrowth)
The following object(s) are masked _by_ '.GlobalEnv':

    len
```

What the heck? What R is trying to tell you is, you already have something called "len" in your workspace (global environment). Since R looks in the workspace first for variables, THIS is the version of "len" you will be working with. The one inside `ToothGrowth` is masked (or shadowed in some versions of R). Thus, if the version of "len" in your workspace is different from the one inside `ToothGrowth`, you'll get the wrong answers!

My advice is back away! Detach `ToothGrowth` from your search path, clean up your workspace, and then attach `ToothGrowth`.

```
> detach(ToothGrowth)
> rm(len)
> attach(ToothGrowth)
```

No warnings this time. Everything should be fine. Now you can work directly with the variables inside `ToothGrowth` without having to name the data frame.

```
> mean(len)
[1] 18.81333
```

Two more warnings about using `attach()`. First, like most things in R, it works silently. Some people just cannot get out of the habit of expecting their computer software to be chatty. I've had students come to my office with a laptop and a puzzled look, and when I inspect their R search path, they have five copies of the same data frame attached! I suppose they kept expecting R to tell them it had been attached, and when R just quietly and obediently did what it was told, they thought nothing had happened and tried again. And again and again and again. If you want to see that the data frame has been attached to the search path, you have to ask.

```
> search()
[1] ".GlobalEnv"      "ToothGrowth"      "package:stats"
[4] "package:graphics" "package:grDevices" "package:utils"
[7] "package:datasets" "package:methods"  "Autoloads"
[10] "package:base"
```

There it is in place number two, which means if R does not find the variable you have requested to work with in the workspace (global environment), it will look next in `ToothGrowth`.

Second warning. NEVER leave things attached to the search path when you don't need them there anymore.

```
> detach(ToothGrowth)
```

Keeping your search path clean is as important as keeping your workspace clean. You don't need to detach stuff if you are just going to quit R, however. When you quit, stuff you've attached is automatically detached.

### Exercises Using "Drive"

1. How big is the `Drive` data frame? [Answer: 28 rows by 8 columns.]
2. What are the names of the variables in `Drive`?
3. How can you see the first six rows of data in `Drive`?
4. If you got no. 3, look at the last row of output. Why is it called row 7 when it is clearly row 6?
5. What is the mean age of subjects in this data set (all of whom were college students)? [Answer: 26.76 years.]
6. Of the 28 subjects in this data set (how do you know there are 28?), how many claim never to tailgate while driving? [Hint: `summary(Drive$tailgate)` is one way to get the answer. Can you think of others?]
7. Of the 28 subjects in this data set, how many claim never to have gotten a ticket?
8. If you answered no. 7 using the `summary` command, why do you think the reported categories are out of order? That is, why are they listed as "many", "never", "some", and not in some more sensible order? (The answer is not in this handout. So guess!)
9. Do this: `> xtabs(~cautious+seatbelt, data=Drive)`  
What is the result? What is the likelihood that someone who is always cautious also always wears a seatbelt (in this sample)? What about for someone who is sometimes cautious?