

Data manipulation and I/O

March 02, 2016

Data structures (cont.)

`data.frame` is the building block for most of what we will do in data analysis. Think about them as a `matrix` that can hold columns of different types and with column names.

```
states <- data.frame("code"      = c("CA", "NY", "NE", "AZ"),
                     "population" = c(38.8, 19.7, 2.1, 6.8),
                     "region"    = c("West", "Northeast", "Midwest", "West"),
                     "landlock"  = c(FALSE, FALSE, TRUE, TRUE))
```

We can access elements via indexing the same way as we would do in a `matrix` or we can access by name:

```
states[, 3]
states[, "region"]
states$region
```

We can also add new variables:

```
states$spanish <- c(28.5, 15.7, NA, 19.5)
```

We can edit values putting together a few things that we have seen so far:

```
states$population[states$code == "NE"] <- 1.8
```

Carefully examine what's happening in the previous line. `states$code` gets the column `code` in our dataset. `states$code == "NE"` returns a logical vector in which only the third observation will be `TRUE`. `states$population[states$code == "NE"]` access the population value for Nebraska. And then we assign the correct value 1.8 to it.

The same approach can be used for subsetting a dataset:

```
states[states$population > 10,] # Notice the comma!
subset(states, population > 10)
```

Transformation of variables is straightforward:

```
states$spanish <- states$spanish * states$population # But we could have used a new variable
```

But we will see soon that we don't need to do most of this transformations.

The dataset is also useful to think about variable types. Consider the variable `region`: it is a vector of characters, but we would like to consider it as a discrete variable in which we would represent it as a value (a number) with a label (the region name). That's a `factor` in R.

```
states$region <- factor(states$region)
states$region
levels(states$region)
```

Depending on who you talk to, they may hate them or love them.

Input/Output

We can write our dataset to disk in comma-separated format using the `write.csv` function.

```
write.csv(states, file="states.csv")
```

Note the named argument to indicate the filename. We could have specified any other folder/directory by passing a path. In the previous call, the file will be written to our current working directory. We can see which one it is with:

```
getwd()
```

```
## [1] "/Users/gonzalorivero/westat/westraining-R/intro-to-R/src"
```

and we can use the `setwd()` to set it.

Unsurprisingly, we can read our dataset back using `read.csv`.

```
states <- read.csv("states.csv")
states
```

What about other delimiters and even formats? For other delimiters, we can use the more general function `read.table` and `write.table` that allows us to specify which delimiter we want to use. Actually, `read.csv` is just `read.table` with a predefined delimiter.

```
states <- read.table("states.csv", sep=",")
```

The most common format for R uses the extension `.RData`, using the functions `save` and `load`.

```
save(states, file="states.RData")
load("states.RData")
states
```

But R can also read (and sometimes write) data in other binary formats: data coming from Stata, SAS, SPSS, or even Excel. The functions to handle these foreign formats are provided in the `foreign` package that comes with R. Take a look at [the documentation](#). We will see how to use additional packages soon enough.

Now that we can read and manipulate data, we can start doing some analysis. Let's the fun begin!