

# Data I/O + Subset

## Module 5

*Andrew Jaffe*

*June 16, 2015*

## Data Output

While its nice to be able to read in a variety of data formats, it's equally important to be able to output data somewhere.

`write.table()`: prints its required argument `x` (after converting it to a `data.frame` if it is not one nor a `matrix`) to a file or connection.

```
write.table(x,file = "", append = FALSE, quote = TRUE, sep = " ",
           eol = "\n", na = "NA", dec = ".", row.names = TRUE,
           col.names = TRUE, qmethod = c("escape", "double"),
           fileEncoding = "")
```

## Data Output

`x`: the R `data.frame` or `matrix` you want to write

`file`: the file name where you want to R object written. It can be an absolute path, or a filename (which writes the file to your working directory)

`sep`: what character separates the columns?

- “,” = .csv - Note there is also a `write.csv()` function
- “`\t`” = tab delimited

`row.names`: I like setting this to `FALSE` because I email these to collaborators who open them in Excel

## Data Output

For example, from the Homework 2 Dataset:

```
circ = read.csv("../data/Charm_City_Circulator_Ridership.csv", header=TRUE, as.is=TRUE)
circ2 = circ[,c("day","date", "orangeAverage","purpleAverage","greenAverage",
               "bannerAverage","daily")]
write.csv(circ2, file="../data/charmcitycirc_reduced.csv", row.names=FALSE)
```

Note that `row.names=TRUE` would make the first column contain the row names, here just the numbers `1:nrow(circ2)`, which is not very useful for Excel. Note that row names can be useful/informative in R if they contain information (but then they would just be a separate column).

## Data Input - Excel

Many data analysts collaborate with researchers who use Excel to enter and curate their data. Often times, this is the input data for an analysis. You therefore have two options for getting this data into R:

- Saving the Excel sheet as a .csv file, and using `read.csv()`
- Using an add-on package called `xlsx`

For single worksheet .xlsx files, I often just save the spreadsheet as a .csv file (because I often have to strip off additional summary data from the columns)

For an .xlsx file with multiple well-formatted worksheets, I use the `xlsx` package for reading in the data.

## More on Packages

Packages are add-ons that are commonly written by users comprised of functions, data, and vignettes

- Use `library()` or `require()` to load the package into memory so you can use its functions
- Install packages using `install.packages("PackageName")`
- Use `help(package="PackageName")` to see what contents the package has
- [http://cran.r-project.org/web/packages/available\\_packages\\_by\\_name.html](http://cran.r-project.org/web/packages/available_packages_by_name.html)

## More on Packages

Some useful data input/output packages

- `foreign` package - read data from Stata/SPSS/SAS
- `sas7bdat` - read SAS data
- `xlsx` - reads in XLS files

## Installing Packages

```
install.packages("xlsx") # OR:  
install.packages("xlsx",  
  repos="http://cran.us.r-project.org")  
library(xlsx) # or require(xlsx)
```

Note you will need a stand-alone version of Java to use this

## Saving R Data

It's very useful to be able to save collections of R objects for future analyses.

For example, if a task takes several hours(/days) to run, it might be nice to run it once and save the results for downstream analyses.

```
save(...,file="[name].rda")
```

where ... is as many R objects, referenced by unquoted variable names, as you want to save.

For example, from the homework:

```
save(circ,circ2,file="../data/charmcirc.rda")
```

## Saving R Data

You also probably have noticed the prompt when you close R about saving your workspace. The workspace is the collection of R objects and custom R functions in your current environment. You can check the workspace with `ls()` or view it in the “Workspace” tab:

```
ls()
```

```
## [1] "circ" "circ2" "file" "Index" "tmp" "x"
```

## Saving R Data

Saving the workspace will save all of these files in your current working directory as a hidden file called “.Rdata”. The function `save.image()` also saves the entire workspace, but you can give your desired file name as an input (which is nicer because the file is not hidden).

Note that R Studio should be able to open any .rda or .Rdata file. Opening one of these file types from Windows Explorer or OSX’s Finder loads all of the objects into your workspace and changes your working directory to wherever the file was located.

## Loading R Data

You can easily load any ‘.rda’ or ‘.Rdata’ file with the `load()` function:

```
tmp=load("../data/charmcirc.rda")
tmp
```

```
## [1] "circ" "circ2"
```

```
ls()
```

```
## [1] "circ" "circ2" "file" "Index" "tmp" "x"
```

Note that this saves the R object names as character strings in an object called ‘tmp’, which is nice if you already have a lot of items in your working directory, and/or you don’t know exactly which got loaded in

## Removing R Data

You can easily remove any R object(s) using the `rm()` or `remove()` functions, and they are no longer in your R environment (which you can confirm with running `ls()`)

You can also remove all of the objects you have added to your workplace with:

```
> rm(list = ls())
```

## Subsetting Data

Often you only want to look at subsets of a data set at any given time. As a review, elements of an R object are selected using the brackets.

Today we are going to look at more flexible ways of identifying which rows of a dataset to select.

## Subsetting Data

You can put a `-` before integers inside brackets to remove these indices from the data.

```
x = c(1,3,77,54,23,7,76,5)
x[1:3] # first 3
```

```
## [1] 1 3 77
```

```
x[-2] # all but the second
```

```
## [1] 1 77 54 23 7 76 5
```

## Subsetting Data

Note that you have to be careful with this syntax when dropping more than 1 element:

```
x[-c(1,2,3)] # drop first 3
```

```
## [1] 54 23 7 76 5
```

```
# x[-1:3] # shorthand. R sees as -1 to 3
x[-(1:3)] # needs parentheses
```

```
## [1] 54 23 7 76 5
```

## Selecting on multiple queries

What about selecting rows based on the values of two variables? We can ‘chain’ together logical statements using the following:

- `&` : AND
- `|` : OR

```
# which Mondays had more than 3000 average riders?
which(circ$day == "Monday" & circ$daily > 3000)[1:20]
```

```
## [1] 148 155 162 169 176 183 190 197 204 211 218 225 232 239 246 253 260
## [18] 267 274 281
```

## AND

Which days had more than 10000 riders overall and more than 3000 riders on the purple line?

```
Index=which(circ$daily > 10000 & circ$purpleAverage > 3000)
length(Index) # the number of days
```

```
## [1] 280
```

```
head(circ[Index,],2) # first 2 rows
```

```
##      day      date orangeBoardings orangeAlightings orangeAverage
## 551  Friday 07/15/2011           4616           4794           4705
## 552  Saturday 07/16/2011          4540           4708           4624
##      purpleBoardings purpleAlightings purpleAverage greenBoardings
## 551              6464              6122              6293              NA
## 552              7797              7447              7622              NA
##      greenAlightings greenAverage bannerBoardings bannerAlightings
## 551              NA              NA              NA              NA
## 552              NA              NA              NA              NA
##      bannerAverage daily
## 551              NA 10998
## 552              NA 12246
```

## OR

Which days had more than 10000 riders overall or more than 3000 riders on the purple line?

```
Index=which(circ$daily > 10000 | circ$purpleAverage > 3000)
length(Index) # the number of days
```

```
## [1] 693
```

```
head(circ[Index,],2) # first 2 rows
```

```
##      day      date orangeBoardings orangeAlightings orangeAverage
## 180  Friday 07/09/2010           2807           2887           2847
## 188  Saturday 07/17/2010          1528           1498           1513
##      purpleBoardings purpleAlightings purpleAverage greenBoardings
## 180              3228              2960              3094.0              NA
## 188              3726              3399              3562.5              NA
##      greenAlightings greenAverage bannerBoardings bannerAlightings
## 180              NA              NA              NA              NA
## 188              NA              NA              NA              NA
##      bannerAverage daily
## 180              NA 5941.0
## 188              NA 5075.5
```

## Subsetting with missing data

Note that logical statements cannot evaluate missing values, and therefore returns an NA:

```
circ$purpleAverage[1:10] > 0
```

```
## [1] NA NA NA NA NA NA NA NA NA NA
```

```
which(circ$purpleAverage > 0)[1:10]
```

```
## [1] 148 149 150 151 152 153 154 155 156 157
```

## Subsetting with missing data

You can use the `complete.cases()` function on a data frame, matrix, or vector, which returns a logical vector indicating which cases are complete, i.e., they have no missing values.

## Selecting on multiple categories

You can select rows where a value is allowed to be several categories. In the homework, we had to subset the Charm City Circulator dataset by each day. How can we select rows that are 1 of 2 days?

The `%in%` operator proves useful: “`%in%` is a more intuitive interface as a binary operator, which returns a logical vector indicating if there is a match or not for its left operand.” It also returns `FALSE` for `NA`s

```
(circ$day %in% c("Monday", "Tuesday"))[1:20] # select entries that are monday or tuesday
```

```
## [1] TRUE TRUE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE FALSE  
## [12] FALSE FALSE FALSE TRUE TRUE FALSE FALSE FALSE FALSE
```

```
which(circ$day %in% c("Monday", "Tuesday"))[1:20] # which indices are true?
```

```
## [1] 1 2 8 9 15 16 22 23 29 30 36 37 43 44 50 51 57 58 64 65
```

## Subsetting columns

We touched on this last class. You can select columns using the variable/column names or column index

```
circ[1:3, c("purpleAverage", "orangeAverage")]
```

```
##   purpleAverage orangeAverage  
## 1           NA           952.0  
## 2           NA           796.0  
## 3           NA          1211.5
```

```
circ[1:3, c(7,5)]
```

```
##   purpleAlightings orangeAverage  
## 1           NA           952.0  
## 2           NA           796.0  
## 3           NA          1211.5
```

## Subsetting columns

You can also remove a column by setting its value to NULL

```
tmp = circ2
tmp$daily=NULL
tmp[1:3,]
```

```
##      day      date orangeAverage purpleAverage greenAverage
## 1  Monday 01/11/2010          952.0             NA           NA
## 2  Tuesday 01/12/2010          796.0             NA           NA
## 3 Wednesday 01/13/2010        1211.5             NA           NA
## bannerAverage
## 1           NA
## 2           NA
## 3           NA
```