Basic Course on R:
Entering and Importing Data Practical Answers

Karl Brand* and Elizabeth Ribble†

18-24 May 2017

Contents

*brandk@gmail.com

†emcclel3@msudenver.edu

1 Entering and Importing Data

1.1 Use R to do the following exercises on “mouse” data.

1.1.1 Enter the following into a data structure with the name `color`:

- purple
- yellow
- red
- brown
- green
- purple
- red
- purple

```
color <- c("purple", "yellow", "red", "brown", "green",
           "purple", "red", "purple")
```

1.1.2 Display the 2nd element of `color`.

```
color[2]

## [1] "yellow"
```

1.1.3 Enter the following into a data structure with the name `weight`:

- 23
- 21
- 18
- 26
- 25
- 22
- 26
- 19

```
weight <- c(23, 21, 18, 26, 25, 22, 26, 19)
```

1.1.4 What are the lengths of `color` and `weight`? Use a function to answer this.

```
length(color)

## [1] 8

length(weight)

## [1] 8
```

1.1.5 Join `color` and `weight` together in a data structure with 2 columns and 8 rows, and assign it to the object `mice`.

```
mice <- data.frame(color = color, weight = weight)
```

1.1.6 What is the data structure of `mice`? What are the dimensions?

```
mice

##    color weight
## 1 purple     23
## 2 yellow     21
## 3    red     18
## 4  brown     26
## 5  green     25
## 6 purple     22
## 7    red     26
## 8 purple     19

str(mice)

## 'data.frame': 8 obs. of  2 variables:
##  $ color : Factor w/ 5 levels "brown","green",..: 3 5 4 1 2 3 4 3
##  $ weight: num  23 21 18 26 25 22 26 19

dim(mice)

## [1] 8 2
```

Using `str` we see that `mice` is a data frame with dimensions 8 x 2, with variable `color` a factor with 5 levels and `weight` a numeric vector. The function `dim` can also be used to extract only the dimensions.

1.1.7 Display only the 3rd row of `mice`.

```
mice[3, ]
```

```
##   color weight
## 3   red     18
```

1.1.8 Display only the 2nd column (“weight”) of `mice`. Do so in two different ways.

Any two of the following are satisfactory:

```
mice[, 2]
mice[, -1]
mice[, "weight"]
```

and if `mice` is a data frame:

```
mice[[2]]
mice$weight
mice["weight"]
```

1.1.9 If the structure of `mice` is not a data frame, turn it into one and call it `micedf`. If `mice` is already a data frame, go ahead and rename it `micedf` anyway. Make sure that the weights are not factors!

```
micedf <- mice
```

If you created a matrix, e.g.

```
mice <- cbind(color, weight)
```

then do the following:

```
micedf <- data.frame(color, weight)
```

1.1.10 Display the dimensions of `micedf`.

```
dim(micedf)

## [1] 8 2
```

1.1.11 Assign the following strings to the row names of `micedf`:

- mouse1
- mouse2
- mouse3
- mouse4
- mouse5
- mouse6
- mouse7
- mouse8

Hint: try using `paste`.

```
row.names(micedf) <- paste("mouse", 1:8, sep="")
```

1.1.12 Create a list containing three elements and assign it to `mylist`:

- `micedf`
- A data frame of `micedf` with a new column called `double` that is 2 times the second column of `micedf` (`weight`). (Did you get an error? Make sure that the second column is numeric and if it isn't, change it!)
- The names of `micedf`.

```
mylist <- list(micedf,
               data.frame(micedf, double = 2 * micedf$weight),
               names(micedf))
```

1.1.13 Assign these names to `mylist`: `first`, `second`, `third`.

```
names(mylist) <- c("first", "second", "third")
```

1.1.14 Display `mylist`. What does it look like?

```
mylist

## $first
##          color weight
## mouse1 purple     23
## mouse2 yellow     21
## mouse3    red     18
## mouse4  brown     26
## mouse5  green     25
## mouse6 purple     22
## mouse7    red     26
## mouse8 purple     19
##
## $second
##          color weight double
## mouse1 purple     23     46
## mouse2 yellow     21     42
## mouse3    red     18     36
## mouse4  brown     26     52
## mouse5  green     25     50
## mouse6 purple     22     44
## mouse7    red     26     52
## mouse8 purple     19     38
##
## $third
## [1] "color"  "weight"
```

A list with three items with the given names preceded by `$`. The first two entries are data frames and have row names, but the third does not (it is a character vector).

1.1.15 View `mylist`. What does it look like (expand the window if necessary)?

```
View(mylist)
```

A matrix with each column representing a column from the columns of the two data frames, plus a final column where the two entries from the third element are repeated to fill up the rows of the matrix. Hence, `View` is not extremely useful for looking at lists.

1.1.16 Display only the second element of `mylist`. Do so in two different ways.

Any two of the following are satisfactory:

```
mylist[[2]] ## extract
mylist$second ## extract
mylist[2] ## subset
mylist[-c(1,3)] ## subset
mylist["second"] ## subset
```

1.1.17 Write `micedf` to a file called "micedf1.csv" in the course directory.

```
write.csv(micedf, "micedf1.csv")
```

1.1.18 Open "micedf1.csv" in Excel and describe what you see. Repeat the step above but do not include row names and call the file "micedf2.csv". How is the output different from "micedf1.csv"?

The file "micedf1.csv" looks exactly like the data frame when displayed in R.

```
write.csv(micedf, "micedf2.csv", row.names = FALSE)
```

The file "micedf2.csv" looks just like the data frame above, but it is missing the row names.

1.1.19 Now read in "micedf1.csv" and "micedf2.csv" into R in two new objects (`newmice1` and `newmice2`, respectively). Describe any differences between the two objects. What are the dimensions of each object?

```
newmice1 <- read.csv("micedf1.csv")
newmice2 <- read.csv("micedf2.csv")
str(newmice1)

## 'data.frame': 8 obs. of  3 variables:
##  $ X     : Factor w/ 8 levels "mouse1","mouse2",..: 1 2 3 4 5 6 7 8
##  $ color : Factor w/ 5 levels "brown","green",..: 3 5 4 1 2 3 4 3
##  $ weight: int  23 21 18 26 25 22 26 19

str(newmice2)

## 'data.frame': 8 obs. of  2 variables:
##  $ color : Factor w/ 5 levels "brown","green",..: 3 5 4 1 2 3 4 3
##  $ weight: int  23 21 18 26 25 22 26 19
```

```
newmice1

##        X   color weight
## 1 mouse1 purple     23
## 2 mouse2 yellow     21
## 3 mouse3    red     18
## 4 mouse4  brown     26
## 5 mouse5  green     25
## 6 mouse6 purple     22
## 7 mouse7    red     26
## 8 mouse8 purple     19

newmice2

##    color weight
## 1 purple     23
## 2 yellow     21
## 3    red     18
## 4  brown     26
## 5  green     25
## 6 purple     22
## 7    red     26
## 8 purple     19
```

The first file was written with row names, but when we read it into R, the row names are now just a column in the data frame. Because there was an empty cell in the space where a column name should have been, R names it `X`. The second file had no row names, and, just like the first file, is given the default row names of 1 to the number of rows. The dimensions are 8 x 3 and 8 x 2, respectively (seen in the output of the `str` call).

1.1.20 Read in "micedf1.csv" into R (assign to object `newmice3`). Use the argument `row.names` to indicate that the first column should be row names and do not allow strings to be turned into factors. Display the object and the structure of the object and describe how it is different from `newmice1` and `micedf`. What are the dimensions of `newmice3`?

```
newmice3 <- read.csv(paste(mydir, "micedf1.csv", sep="/"),
                     row.names = 1, stringsAsFactors = F)
```

```
str(newmice3)

## 'data.frame': 8 obs. of  2 variables:
##  $ color : chr  "purple" "yellow" "red" "brown" ...
##  $ weight: int  23 21 18 26 25 22 26 19

newmice3

##          color weight
## mouse1 purple     23
## mouse2 yellow     21
## mouse3    red     18
## mouse4  brown     26
## mouse5  green     25
## mouse6 purple     22
## mouse7    red     26
## mouse8 purple     19
```

The new object `newmice3` now has row names where in `newmice1` they were treated as a variable in the data frame. The `color` variable was not converted to a factor, unlike in `newmice1`. The new object `micedf` is now (nearly) identical to the original `micedf`. It has the same row names and dimensions as `micedf`. The only difference is the class of `weight` is an integer instead of just "numeric" (more specific). The dimensions are 8 x 2.

1.2 Use R to do the following exercises on the `Puromycin` data.

1.2.1 Load the `Puromycin` data using the `data()` function.

```
data(Puromycin)
```

1.2.2 What is the data structure of `Puromycin`? What are the dimensions? Do not just display the data (this is not convenient for large datasets).

```
str(Puromycin)

## 'data.frame': 23 obs. of  3 variables:
##  $ conc : num  0.02 0.02 0.06 0.06 0.11 0.11 0.22 0.22 0.56 0.56 ...
##  $ rate : num  76 47 97 107 123 139 159 152 191 201 ...
##  $ state: Factor w/ 2 levels "treated","untreated": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "reference")= chr "A1.3, p. 269"
```

Using str we see that Puromycin is a data frame with dimensions 23 x 3, with variables conc and rate numeric vectors and state a factor with 2 levels.

1.2.3 What are the names of Puromycin? Use a function other than str.

```
names(Puromycin)

## [1] "conc"  "rate"  "state"
```

1.2.4 What are the levels of the state variable? Use a function other than str.

```
levels(Puromycin$state)

## [1] "treated"   "untreated"
```

1.2.5 Display the rate for all concentrations less than 0.10 in the treated group.

```
Puromycin[Puromycin$conc < .10 & Puromycin$state=="treated", "rate"]

## [1]  76  47  97 107

## or
Puromycin$rate[Puromycin$conc < .10 & Puromycin$state=="treated"]

## [1]  76  47  97 107

## or
Puromycin[["rate"]][Puromycin$conc < .10 & Puromycin$state=="treated"]

## [1]  76  47  97 107
```

1.2.6 What are the row indices for the concentrations of 0.22?

```
which(Puromycin$conc == 0.22)

## [1]  7  8 19 20
```

If you want to save your work: save your R session and/or source code!