Data Frames and Clean Data Files

Levente Orbán

2018-07-26

Data Frames

Until now, we loaded data into vectors using the `r, eval=F, echo=T c()` function. However, this is quite limited. Most of our data will have several variables or factors that we need to load into a single data set for analysis. Welcome data frames.

A data frame is used for storing tables of values. For example, consider the following vectors below. Now we can combine them into a single data frame:

```
employmentStatus <- c(TRUE,FALSE,FALSE,TRUE,TRUE,TRUE,FALSE)
siblings <- c(1,3,3,1,2,2,1)
mydata <- data.frame(employment = employmentStatus, sibs = siblings)
mydata
```

```
##   employment sibs
## 1       TRUE    1
## 2      FALSE    3
## 3      FALSE    3
## 4       TRUE    1
## 5       TRUE    2
## 6       TRUE    2
## 7      FALSE    1
```

Another similar function is `r, eval=F, echo=T list()`. However, unlike `r, echo=T, eval=F data.frame()` in which each column of data must have the same length (i.e., sample size), `r, eval=F, echo=T list()` is more flexible in allowing you to have different variables, each with different lengths. For example:

```
employmentStatus <- c(TRUE,FALSE,FALSE,TRUE,TRUE,TRUE,FALSE)
siblings <- c(1,3,3,1,2,2,1,1,3,3,1,2,2,1)
mydata <- list(sibs = siblings, emp = employmentStatus)
head(mydata)
```

```
## $sibs
##  [1] 1 3 3 1 2 2 1 1 3 3 1 2 2 1
##
## $emp
## [1]  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE
```

Importing Data

However, this data was hand-coded into R, which is a very time consuming process, especially if you have hundreds, thousands, or worse, hundreds of thousands of records. R can handle importing data from a variety of formats including the friendly csv (**c**omma **s**eparated **v**alues) format, or less ideal Excel or SPSS formats.

Data can come from a variety of sources. One increasingly more common source is the Internet. To import data directly from the web:

```
myData <- read.csv(url("http://llorban.com/psyc2300/1100_exam_grades.csv"))
```

If we wanted to take a look at this data, we could simply type the name of the variable and hit enter:

```
myData
```

```
##    student_id exam_2 exam_3
## 1           1  49.62  43.13
## 2           2   0.00   0.00
## 3           3  84.62  82.50
## 4           4  64.23  68.13
## 5           5  66.15   0.00
## 6           6  49.81  36.13
## 7           7  44.42  52.50
## 8           8  45.96   0.00
## 9           9  54.62  26.25
## 10         10  86.92  73.75
## 11         11  40.77  50.00
## 12         12  50.58  60.00
## 13         13  75.58  79.38
## 14         14   0.00   0.00
## 15         15  42.12   0.00
## 16         16  43.65   0.00
## 17         17  72.31  71.88
## 18         18   0.00   0.00
## 19         19  80.77  74.38
## 20         20  63.01  42.57
## 21         21  62.66  54.05
## 22         22  52.32  39.20
## 23         23  42.94  35.80
## 24         24  84.94  91.22
## 25         25  59.45  65.67
## 26         26  61.94  58.11
## 27         27  41.18  34.46
## 28         28  58.02  65.55
## 29         29  67.83  49.32
## 30         30  51.39  52.70
## 31         31  51.50  43.24
## 32         32  88.59  91.90
## 33         33  48.73  34.46
## 34         34  38.24  49.32
## 35         35  81.82  87.16
## 36         36  60.25  53.39
## 37         37  52.50  43.24
## 38         38  68.18  68.24
## 39         39  64.71  52.03
## 40         40  58.81  47.96
## 41         41  52.14  47.95
## 42         42   0.00   0.00
## 43         43  70.32  50.66
## 44         44  64.44  52.04
## 45         45  70.14  66.24
## 46         46  77.90  77.04
## 47         47  60.96  60.82
```

```
## 48         48  67.38  50.68
## 49         49  60.52  54.74
## 50         50  56.77  45.26
```

As you can see, this produces a long and fairly useless output. If we are just interested in a quick overview with the variables and types of values they take, we can run the *head()* function:

```
head(myData)
```

```
##   student_id exam_2 exam_3
## 1          1  49.62  43.13
## 2          2   0.00   0.00
## 3          3  84.62  82.50
## 4          4  64.23  68.13
## 5          5  66.15   0.00
## 6          6  49.81  36.13
```

Much better. `r, eval=F, echo=T head()` reveals three variables inside the data frame, one of which takes an integer, and the other two taking on float values.

Sometimes we are simply interested in just the names of the variables:

```
names(myData)
```

```
## [1] "student_id" "exam_2"     "exam_3"
```

Or a little summary:

```
summary(myData)
```

```
##    student_id        exam_2          exam_3
##  Min.   : 1.00   Min.   : 0.00   Min.   : 0.00
##  1st Qu.:13.25   1st Qu.:48.95   1st Qu.:36.90
##  Median :25.50   Median :59.13   Median :50.67
##  Mean   :25.50   Mean   :55.83   Mean   :47.66
##  3rd Qu.:37.75   3rd Qu.:67.72   3rd Qu.:65.64
##  Max.   :50.00   Max.   :88.59   Max.   :91.90
```

In some cases, the data will reside on your computer. In this case, we need to set the path to the file on the computer. This will get tricky now, because each of our computers could have different paths. In the case of my computer, I can load a data set I just downloaded like so:

```
dataFromFile <- read.csv("~/Downloads/levente_weight.csv")
head(dataFromFile)
```

```
##                  Date Weight Fat.mass Bone.mass Muscle.mass Hydration
## 1 2017-12-29 10:00:18  169.2     38.8        NA          NA        NA
## 2 2017-12-28 11:06:28  170.8     38.1        NA          NA        NA
## 3 2017-12-24 10:28:29  168.8     35.4        NA          NA        NA
## 4 2017-12-21 09:08:12  168.9     36.6        NA          NA        NA
## 5 2017-12-20 09:14:08  168.1     36.3        NA          NA        NA
## 6 2017-12-19 10:45:56  168.6     38.6        NA          NA        NA
##   Comments
## 1       NA
## 2       NA
## 3       NA
## 4       NA
## 5       NA
## 6       NA
```

Let's dig into this data. We need to revisit the idea of subsetting from yesterday, and the idea of referring to column names. If we wanted to better understand Lev's weight trends over the years, we would need to be able to tease out data according to the date.

We can restrict our analysis to just one of the variables:

```
head(dataFromFile$Weight)
```

```
## [1] 169.2 170.8 168.8 168.9 168.1 168.6
```

We can even calculate the mean just for the weight:

```
mean(dataFromFile$Weight)
```

```
## [1] 165.0289
```

If you have a directory that you would like to use, and don't want to type in the whole path each time, R can help. getwd() will tell you where R will look by default.

```
getwd()
```

```
## [1] "/Users/llorban/Documents/Service to the Profession/2018 R Workshop/git_repo/rworkshop/day2"
```

So if I were to type in just the file name, `r, eval=F, echo=T read.csv("levente_weight.csv")` without the path, R would assume `r, eval=F, echo=T read.csv("/Users/llorban/levente_weight.csv")`. In my particular case, because the file is in the Downloads folder, typing in just the file name would result in an error. Go ahead, try it.

It is possible to change the directory R searches by default with the `r, eval=F, echo=T setwd()` command.

```
setwd("/Users/llorban/Downloads")
```

Once this command is executed, I can simply use the `r, eval=F, echo=T read.csv()` with just the file name.

Importing Excel files is also possible with addons. We will install an additional package:

```
install.packages("readxl")
```

And import files like so:

```
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 3.4.4
```

```
setwd("~/Downloads")
myData <- read_xlsx("Grossman and Kross 2014 Study 2.xlsx")
head(myData)
```

```
## # A tibble: 6 x 57
##   `Grossman and K~ X__1  X__2  X__3  X__4  X__5  X__6  X__7  X__8  X__9
##   <chr>            <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 ID               COND~ code  gend~ start end   feel~ sam.~ sam.~ happ~
## 2 1                3     3     -1    1/23~ 1/23~ 1     4     8     2
## 3 6                4     4     -1    1/23~ 1/23~ 1     3     2     2
## 4 8                4     4     -1    1/23~ 1/23~ 1     4     8     3
## 5 9                4     4     -1    1/23~ 1/23~ 1     3     8     2
## 6 10               2     2     -1    1/23~ 1/23~ 1     4     2     2
## # ... with 47 more variables: X__10 <chr>, X__11 <chr>, X__12 <chr>,
## #   X__13 <chr>, X__14 <chr>, X__15 <chr>, X__16 <chr>, X__17 <chr>,
## #   X__18 <chr>, X__19 <chr>, X__20 <chr>, X__21 <chr>, X__22 <chr>,
## #   X__23 <chr>, X__24 <chr>, X__25 <chr>, X__26 <chr>, X__27 <chr>,
```

```
## #   X__28 <chr>, X__29 <chr>, X__30 <chr>, X__31 <chr>, X__32 <chr>,
## #   X__33 <chr>, X__34 <chr>, X__35 <chr>, X__36 <chr>, X__37 <chr>,
## #   X__38 <chr>, X__39 <chr>, X__40 <chr>, X__41 <chr>, X__42 <chr>,
## #   X__43 <chr>, X__44 <chr>, X__45 <chr>, X__46 <chr>, X__47 <chr>,
## #   X__48 <chr>, X__49 <chr>, X__50 <chr>, X__51 <chr>, X__52 <chr>,
## #   X__53 <chr>, X__54 <chr>, X__55 <chr>, X__56 <chr>
```

If there are multiple sheets, you can specify the sheet number:

```
library(readxl)
setwd("~/Downloads")
myData <- read_xlsx("Grossman and Kross 2014 Study 2.xlsx", sheet = 1)
head(myData)
```

```
## # A tibble: 6 x 57
##   `Grossman and K~ X__1  X__2  X__3  X__4  X__5  X__6  X__7  X__8  X__9
##   <chr>            <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 ID               COND~ code  gend~ start end   feel~ sam.~ sam.~ happ~
## 2 1                3     3     -1    1/23~ 1/23~ 1     4     8     2
## 3 6                4     4     -1    1/23~ 1/23~ 1     3     2     2
## 4 8                4     4     -1    1/23~ 1/23~ 1     4     8     3
## 5 9                4     4     -1    1/23~ 1/23~ 1     3     8     2
## 6 10               2     2     -1    1/23~ 1/23~ 1     4     2     2
## # ... with 47 more variables: X__10 <chr>, X__11 <chr>, X__12 <chr>,
## #   X__13 <chr>, X__14 <chr>, X__15 <chr>, X__16 <chr>, X__17 <chr>,
## #   X__18 <chr>, X__19 <chr>, X__20 <chr>, X__21 <chr>, X__22 <chr>,
## #   X__23 <chr>, X__24 <chr>, X__25 <chr>, X__26 <chr>, X__27 <chr>,
## #   X__28 <chr>, X__29 <chr>, X__30 <chr>, X__31 <chr>, X__32 <chr>,
## #   X__33 <chr>, X__34 <chr>, X__35 <chr>, X__36 <chr>, X__37 <chr>,
## #   X__38 <chr>, X__39 <chr>, X__40 <chr>, X__41 <chr>, X__42 <chr>,
## #   X__43 <chr>, X__44 <chr>, X__45 <chr>, X__46 <chr>, X__47 <chr>,
## #   X__48 <chr>, X__49 <chr>, X__50 <chr>, X__51 <chr>, X__52 <chr>,
## #   X__53 <chr>, X__54 <chr>, X__55 <chr>, X__56 <chr>
```

Indexing

Consider this data:

```
label <- c("Experimental", "Experimental", "Experimental", "Experimental", "Experimental", "Experimental
data <- c(12, 13, 14, 13, 12, 14, 15, 16, 14, 15, 16, 14, 13)

dataset <- data.frame(condition = label, values = data)
dataset[dataset$condition == "Control",]
```

```
##    condition values
## 7    Control     15
## 8    Control     16
## 9    Control     14
## 10   Control     15
## 11   Control     16
## 12   Control     14
## 13   Control     13
```

Subsetting is very powerful that allows you to filter the data in many different ways:

```r
dataset[dataset$values < 15,][dataset$label]
```

```
## data frame with 0 columns and 9 rows
```

Exercise 1

About the data you will work on: A survey randomly selected a group of Hong Kong residents for a telephone questionnaire. The survey was conducted during the H1N1 influenza epidemic. Survey questions polled subjects about their anxiety, risk perception, knowledge on modes of transmission, and preventative behaviours. I have selected and cleaned up a version of this questionnaire that has two columns of data: Column 1: questions asked before the H1N1 pandemic, and questions asked during the pandemic. Column 2: Answer to the following question: What do you think are your chances of getting swine flu (H1N1 influenza A) over the next 1 month compared to others outside your family? 1-Not at all; 2-Much less; 3- Less; 4-Evens; 5-More; 6-Much more; 7-Certain. Note that each row represents a different subject in this data set.

- Original Publication (for your reference): https://academic.oup.com/jid/article/202/6/867/936219
- Original Data (for your reference): https://datadryad.org/resource/doi:10.5061/dryad.1485f

Download the cleaned up data from our github repository under the **day2** folder. The name of the file is *2-1 fludata_chances.csv*

1. Import the following data set into R. Try running a web import, and an import from your local machine.
2. Find out the following about this data set: names of all the variables, and the sample size
3. Check for normality and homogeneity
4. Compute the mean of each variable
5. Use subselect for the "before" and "during" group, and compute the mean and standard deviation of each group

Exercise 2

Import this data either using the web importer or from your local machine: https://stats.idre.ucla.edu/stat/data/hsb2_small.csv. You will find this file in the git repository too, as 2-1 hsb2_small.csv under **day2**.

1. Find out the variable names and the data type for each variable
2. Select records that have a science score of over 60.
3. Compute the mean and standard deviation of "write" and "science" score of those subjects that have a science score of 60 or more