Data Wrangling

Leili Mortazavi

8/3/2020

Contents

Many people have contributed to developing and revising the tutorial material over the years:

Anna Khazenzon Cayce Hook Paul Thibodeau Mike Frank Benoit Monin Ewart Thomas Michael Waskom Steph Gagnon Dan Birman Natalia Velez, Kara Weisman Andrew Lampinen Joshua Morris Yochai Shavit Jackie Schwartz Russ Poldrack

Note on relative paths vs. absolute paths

Before we dig into working with data in R, let's talk about some good practices about how to write down your paths.

If you look at the top of your console panel, you can see your path. (i.e., where you are in the eyes of RStudio).

If you open up an R script in a new session of RStudio, the path will automatically be set to where your R script is saved.

If you already have other R scripts open in RStudio and want to work on a different script, you may need to change your path to where that script is saved.

To do this run:

```
setwd("~/PATH/TO/CURRENT/SCRIPT")
```

Why bother??

If you can keep a tidy directory structure, then you can write a **relative path** for importing and exporting data and plots. Not only will this make your coding easier, it will make your script reproducible (for your future self and for others). So if you move your project directory, or if someone else wants to work on it on their own device, the script will work smoothly, without the need to change all the paths every time.

So now that we've started working on a new script, which is saved in a different folder, let's use relative path to go there:

```
setwd("../Part2")
```

As you see now, all of us can now use this relative path within our R_basic folder, regardless of having saved the bootcamp files in different locations.

Just to convince you a little more, if I wanted to use absolute path this is how it would look like: `setwd("~/STANFORD/teaching/summerStats_2020/modules/R_basic/Part2")`

Ugly huh?! Plus, yours would probably be completely different up until the "modules" part.

Importing data

Now, let's use relative path to import the albums dataset, sitting in you data folder.

```
data <- read.csv("../data/albums.csv")
```

Inspecting data

```
View(data)
# note that View has to be capitalized!

head(data)
```

```
##                         album               artist year sales_millions  solo
## 1 Speakerboxx/The Love Below              Outkast 2003           11.4 FALSE
## 2           Life After Death The Notorious B.I.G. 1997           10.2  TRUE
## 3             All Eyez on Me                2Pac 1996            9.0  TRUE
## 4            Licensed to Ill         Beastie Boys 1986            9.0 FALSE
## 5                  Stankonia              Outkast 2000            4.0 FALSE
## 6               Ready to Die The Notorious B.I.G. 1994            4.0  TRUE
##      sales language years_since_release decade  feud born_yet TRUE. FALSE.
## 1 11400000  English                   16    00s FALSE     TRUE  TRUE  FALSE
## 2 10200000  English                   22    90s  TRUE     TRUE  TRUE  FALSE
## 3  9000000  English                   23    90s  TRUE     TRUE  TRUE  FALSE
## 4  9000000  English                   33    80s FALSE    FALSE  TRUE  FALSE
## 5  4000000  English                   19    00s FALSE     TRUE  TRUE  FALSE
## 6  4000000  English                   25    90s  TRUE     TRUE  TRUE  FALSE
```

This type of data is called a dataframe.

Let's formally check its type to see what R thinks of it:

```
class(data)
```

```
## [1] "data.frame"
```

Get a list of all the column names:

```
colnames(data)
```

```
##  [1] "album"               "artist"              "year"
##  [4] "sales_millions"      "solo"                "sales"
##  [7] "language"            "years_since_release" "decade"
## [10] "feud"                "born_yet"            "TRUE."
## [13] "FALSE."
```

Remember we said R tries to detect data types? Even though the whole dataset is a data frame, each column may be numeric, character, etc. Let's check:

```
# use your indexing skills
# select a column based on its index (i.e., first column):
class(data[,1])
```

```
## [1] "factor"
```

```
# select a column based on its name, using `$` as follows:
class(data$album)
```

```
## [1] "factor"
```

Note: The TAB key is your friend! Use it to make your life easier, AND to decrease the chances of making a typo, which will result in an annoying hard-to-detect error.

We can use the summary function of a data frame to get some summary values on each column:

```r
summary(data)
```

```
##                           album                    artist       year
##  All Eyez on Me              :1   2Pac                 :2   Min.   :1986
##  Ill Communication           :1   Beastie Boys         :2   1st Qu.:1994
##  Licensed to Ill             :1   Outkast              :2   Median :1996
##  Life After Death            :1   The Notorious B.I.G.:2   Mean   :1996
##  R U Still Down? Remember Me:1                             3rd Qu.:1998
##  Ready to Die                :1                            Max.   :2003
##  (Other)                     :2
##  sales_millions      solo             sales             language
##  Min.   : 3.000   Mode :logical   Min.   : 3000000   English:8
##  1st Qu.: 4.000   FALSE:4         1st Qu.: 4000000
##  Median : 6.500   TRUE :4         Median : 6500000
##  Mean   : 6.825                   Mean   : 6825000
##  3rd Qu.: 9.300                   3rd Qu.: 9300000
##  Max.   :11.400                   Max.   :11400000
##
##  years_since_release decade     feud          born_yet        TRUE.
##  Min.   :16.00       00s:2   Mode :logical   Mode :logical   Mode:logical
##  1st Qu.:21.25       80s:1   FALSE:4         FALSE:1         TRUE:8
##  Median :22.50       90s:5   TRUE :4         TRUE :7
##  Mean   :23.12
##  3rd Qu.:25.00
##  Max.   :33.00
##
##    FALSE.
##  Mode :logical
##  FALSE:8
##
##
##
##
##
```

```r
# on the entire data frame
is.na(data)
```

```
##      album artist  year sales_millions  solo sales language years_since_release
## [1,] FALSE  FALSE FALSE          FALSE FALSE FALSE    FALSE                FALSE
## [2,] FALSE  FALSE FALSE          FALSE FALSE FALSE    FALSE                FALSE
## [3,] FALSE  FALSE FALSE          FALSE FALSE FALSE    FALSE                FALSE
## [4,] FALSE  FALSE FALSE          FALSE FALSE FALSE    FALSE                FALSE
## [5,] FALSE  FALSE FALSE          FALSE FALSE FALSE    FALSE                FALSE
## [6,] FALSE  FALSE FALSE          FALSE FALSE FALSE    FALSE                FALSE
## [7,] FALSE  FALSE FALSE          FALSE FALSE FALSE    FALSE                FALSE
## [8,] FALSE  FALSE FALSE          FALSE FALSE FALSE    FALSE                FALSE
##      decade  feud born_yet TRUE. FALSE.
## [1,]  FALSE FALSE    FALSE FALSE  FALSE
## [2,]  FALSE FALSE    FALSE FALSE  FALSE
## [3,]  FALSE FALSE    FALSE FALSE  FALSE
## [4,]  FALSE FALSE    FALSE FALSE  FALSE
## [5,]  FALSE FALSE    FALSE FALSE  FALSE
```

```
## [6,]  FALSE FALSE    FALSE FALSE  FALSE
## [7,]  FALSE FALSE    FALSE FALSE  FALSE
## [8,]  FALSE FALSE    FALSE FALSE  FALSE
```

```
# on a specific column
# TRY FOR YOURSELF!
# Hint: use your indexing skills.



# is.na(data$year)
```

Manipulating data

When we analyze and visualize the data, we often need to manipulate data in some ways. For instance, we may need to select only 2 of the variables, or we may need to take the mean and standard deviation of a subset of our observations, e.g., separately for males and females.

Tidyverse

To do these, and many more operations, there's a set of packages that come in very handy. That is, tidyverse (by Hadley Wikham): a collection of packages that make
transforming data a lot easier!

```
# install.packages("tidyverse")

# load it
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Note: when we write scripts, it is conventional to load all the required packages on top of the script.

Selecting and subsetting

```
select(.data = data, year, artist)
```

```
##   year               artist
## 1 2003              Outkast
## 2 1997 The Notorious B.I.G.
## 3 1996                2Pac
## 4 1986         Beastie Boys
## 5 2000              Outkast
## 6 1994 The Notorious B.I.G.
## 7 1997                2Pac
## 8 1994         Beastie Boys
```

```
data_1 <- select(.data = data, year, artist)
head(data_1)
```

```
##   year               artist
## 1 2003              Outkast
## 2 1997 The Notorious B.I.G.
## 3 1996                2Pac
## 4 1986         Beastie Boys
## 5 2000              Outkast
## 6 1994 The Notorious B.I.G.
```

```
data_2 <- subset(data_1, year > 1995)
head(data_2)
```

```
##   year               artist
## 1 2003              Outkast
## 2 1997 The Notorious B.I.G.
## 3 1996                2Pac
## 5 2000              Outkast
## 7 1997                2Pac
```

Piping

It's quite inefficient to create new variables every time we want to perform some operation. What if we want to perform several steps (which we often do)?

Let's learn a really nice feature of the tidyverse package that will make coding in R much easier.

This is called ** piping**. `%>%` Piping will allow us to do several operations in one big chain.

```
# select and subset in one go

data_select <- data %>%
  # means: take the dataframe and feed it into the next line
  select(year, artist) %>%
  # means: select these two columns from the "data" dataframe,
  # and feed that into the next line
  subset(year > 1995)
  # means: subset some of the rows and save the result in the "data_select" dataframe
head(data_select)
```

```
##   year               artist
```

```
## 1 2003              Outkast
## 2 1997 The Notorious B.I.G.
## 3 1996                 2Pac
## 5 2000              Outkast
## 7 1997                 2Pac
```

Making new variables

```
data %>%
  mutate(sales = sales_millions * 1000000)
```

```
##                         album               artist year sales_millions  solo
## 1  Speakerboxx/The Love Below              Outkast 2003           11.4 FALSE
## 2            Life After Death The Notorious B.I.G. 1997           10.2  TRUE
## 3              All Eyez on Me                2Pac 1996            9.0  TRUE
## 4             Licensed to Ill         Beastie Boys 1986            9.0 FALSE
## 5                   Stankonia              Outkast 2000            4.0 FALSE
## 6                Ready to Die The Notorious B.I.G. 1994            4.0  TRUE
## 7 R U Still Down? Remember Me                2Pac 1997            4.0  TRUE
## 8           Ill Communication         Beastie Boys 1994            3.0 FALSE
##      sales language years_since_release decade  feud born_yet TRUE. FALSE.
## 1 11400000  English                   16    00s FALSE     TRUE  TRUE  FALSE
## 2 10200000  English                   22    90s  TRUE     TRUE  TRUE  FALSE
## 3  9000000  English                   23    90s  TRUE     TRUE  TRUE  FALSE
## 4  9000000  English                   33    80s FALSE    FALSE  TRUE  FALSE
## 5  4000000  English                   19    00s FALSE     TRUE  TRUE  FALSE
## 6  4000000  English                   25    90s  TRUE     TRUE  TRUE  FALSE
## 7  4000000  English                   22    90s  TRUE     TRUE  TRUE  FALSE
## 8  3000000  English                   25    90s FALSE     TRUE  TRUE  FALSE
```

We learned 3 important and very common functions of tidyverse: `select()`, `subset()`, and `mutate()`

Let's practice now.

Grouping and summarizing

dplyr (part of tidyvers) can also be used to quickly summarize data by different grouping variables.

In this dataset, who's the bestselling artist?

To find out, we group by `artist`, calculate each artist's total album sales with `summarize()`, and save the result in `result`:

```
result <- data %>%
  # group the data by `artist`
  group_by(artist) %>%
  # add up the `sales` for each artist and save in `total_sales`
  summarize(total_sales = sum(sales))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
head(result)
```

```
## # A tibble: 4 x 2
##   artist               total_sales
##   <fct>                      <int>
## 1 2Pac                    13000000
## 2 Beastie Boys            12000000
## 3 Outkast                 15400000
## 4 The Notorious B.I.G.    14200000
```

```r
# setting eval to false in the brackets tells R not to run this code when knitting
# APPLY YOUR KNOWLEDGE:
# How would you change this code if we instead wanted to compare
# the total sales, in millions, of solo artists vs. group artists?

result <- data %>%
  mutate(sales = sales_millions * 1000000) %>%
  group_by(artist) %>%
  summarize(total_sales = sum(sales)) %>%
  arrange(desc(total_sales))









# ANSWER:
# result <- albums %>%
#  group_by(solo) %>%
#  summarize(total_sales = sum(sales_millions))

# you can also group by multiple variables
result <- albums %>%
  group_by(artist, solo) %>%
  summarize(total_sales = sum(sales_millions)) %>%
  arrange(desc(total_sales))
```

Tidy data

Tidy data is long data, where every row is one observation. If there are multiple observations per subject, then we have multiple rows per subject, but then have another variable (i.e., column) denoting some aspect of those observations.

Let's work through an example.

Import a dataset from your data/ folder. The file is called "prepost.csv".

```
# practice importing csv files









# ANSWER:
data_wide <- read_csv("../data/prepost.csv")
```

```
## Parsed with column specification:
## cols(
##   subject = col_double(),
##   gender = col_character(),
##   age = col_double(),
##   pretest = col_double(),
##   posttest = col_double(),
##   condition = col_character(),
##   diff = col_double()
## )
```

```
# first in matrix view:

# top 5 rows:

# names of columns:









# ANSWER:
# first in matrix view:
# View(data_wide)

# top 5 rows:
# head(data_wide, 5)

# names of columns:
# colnames(data_wide)
```

As you can see, this dataset is not in tidy format, because we have two observations per row (i.e., pretest and posstes).

Remember that tidy data is long data, where every row is one observation. If there are multiple observations per subject, then we have multiple rows per subject, but then have another variable (i.e., column) denoting some aspect of those observations.

Here, we want to make a variable that only denotes whether the score was from pretest or from posttest. Let's call this variable "prepost".

Then we want to put the score into another column. Let's call this variable "score".

So how do we do that?

Here's a function in tidyverse that takes care of this operation very gracefully: `pivot_longer()`

```
data_long <- data_wide %>%
  pivot_longer(cols = c(pretest, posttest),
               # means: select the columns that include the scores
               names_to = "prepost",
               # means: take the name of these two columns and put
               # them into a new variable called "prepost"
               values_to = "score"
               # means: take the values of these two columns and put them into a new variable called "s
               )
```

Now let's check out what pivot_longer() did.

```
data_long %>%
  head()
```

```
## # A tibble: 6 x 7
##   subject gender   age condition  diff prepost  score
##     <dbl> <chr>  <dbl> <chr>     <dbl> <chr>    <dbl>
## 1       1 f         20 drug          2 pretest     45
## 2       1 f         20 drug          2 posttest    47
## 3       2 m         45 drug          4 pretest     65
## 4       2 m         45 drug          4 posttest    69
## 5       3 f         36 drug         10 pretest     45
## 6       3 f         36 drug         10 posttest    55
```

```
# How many rows did the wide data have?


# How many rows does the long data have?
```

As we work through the visualization modules and statistical analyses, you'll see the benefits of having tidy data.

Saving out wrangled data

After we manipulate out data in ways that's ready for analyses or visualizations, we may want to save it out in a csv file, so that we don't need to wrangle it again.

```
write_csv(x = data_long,
          path = "../data/prepost_tidy.csv")
```

For extra material on the following, see module **BasicR_Part2_dataWranglingExtra.Rmd**

- creating dataframes in R
- manipulating data using base R