Chapter 2: Manipulating Your Data

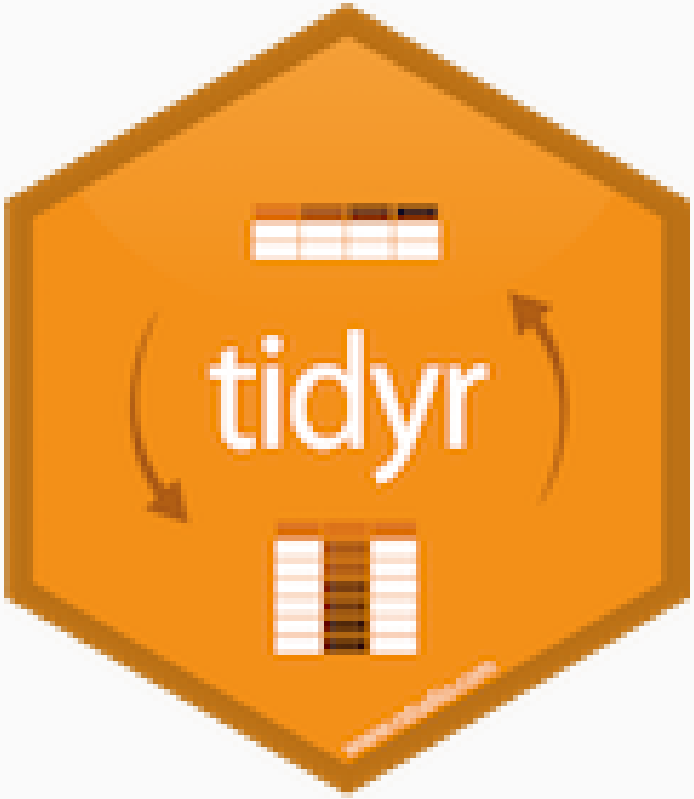
Tyson S. Barrett

Summer 2017

Utah State University

Introduction

The Newest and Brightest



The Newest and Brightest

Tidyverse

- In order to manipulate your data in the cleanest, most up-to-date manner, we are going to be using the "tidyverse" group of methods.
- The tidyverse[1] is a group of packages[2] that provide a simple syntax that can do many basic (and complex) data manipulating.
- The group of packages can be downloaded via:

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

After downloading it, simply use:

```
library(tidyverse)
```

Tidyverse

Note that when we loaded tidyverse it loaded 6 packages and told you of "conflicts". These conflicts are where two or more loaded packages have the same function in them. The last loaded package is the one that R will use by default. For example, if we loaded two packages–awesome and amazing–and both had the function–make_really_great and we loaded awesome and then amazing as so:

```
library(awesome)
library(amazing)
```

R will automatically use the function from amazing.

Conflicts

We can still access the `awesome` version of the function (because even though the name is the same, they won't necessarily do the same things for you). We can do this by:

```
awesome::make_really_great(arg)
```

That's a bit of an aside, but know that you can always get at a function even if it is "masked" from your current session.

Tidy Methods

The Tidy Data Way

I'm introducing this to you for a couple reasons.

1. It simplifies the code and makes the code more readable. As Mr. Wickham says, **there are always at least two collaborators on any project: you and future you.**
2. It is the cutting edge. The most influential individuals in the `R` world, including the makers and maintainers of `RStudio`, use these methods and syntax.

The majority of what you'll need to do with data as a researcher will be covered by these functions.

Methods for Tidying

There are several methods that help tidy up your data:

1. Piping
2. Selecting and Filtering
3. Grouping and Summarizing
4. Reshaping
5. Joining (merging)

To help illustrate each aspect, we are going to use real data from the National Health and Nutrition Examiniation Survey (NHANES). I've provided this data at https://tysonstanley.github.io/assets/Data/NHANES.zip. I've cleaned it up somewhat already.

A Walk-Through

Example: NHANES

Import

First, we will set our working directory with `setwd`. This tells `R` where to look for files, including your data files.

```
setwd("~/Dropbox/GitHub/blog_rstats/assets/Data/")
```

```
library(foreign)
dem_df <- read.xport("NHANES_demographics_11.xpt")
med_df <- read.xport("NHANES_MedHeath_11.xpt")
men_df <- read.xport("NHANES_MentHealth_11.xpt")
act_df <- read.xport("NHANES_PhysActivity_11.xpt")
```

Example: NHANES

Now we have four separate, but related, data sets in memory:

1. `dem_df` containing demographic information
2. `med_df` containing medical health information
3. `men_df` containing mental health information
4. `act_df` containing activity level information

Example: NHANES

Since all of them have all-cap variable names, we are going to quickly change this with a little trick:

```
names(dem_df) <- tolower(names(dem_df))
names(med_df) <- tolower(names(med_df))
names(men_df) <- tolower(names(men_df))
names(act_df) <- tolower(names(act_df))
```

This takes the names of the data frame (on the right hand side), changes them to lower case and then reassigns them to the names of the data frame.[3]

[3] Note that these are not particularly helpful names, but they are the names provided in the original data source. If you have questions about the data, visit http://wwwn.cdc.gov/Nchs/Nhanes/Search/Nhanes11_12.aspx.

Example: NHANES

We will now go through each aspect of the tidy way of working with data using these four data sets.

Example: NHANES

Piping

Example: NHANES

Piping

%>% is the pipe "operator". It takes what is on the left hand side and puts it in the right hand side's function.

```
dem_df %>% summary
```

So the above code takes the data frame `df` and puts it into the `summary` function. This does the same thing as `summary(dem_df)`.

In this simple case, it doesn't really make the code more readable, but in more complex situations it can really help.

Example: NHANES

Select and Filter

Example: NHANES

Select and Filter

In situations where you want or need to subset your data, two main forms exist:

1. Selecting Variables
2. Filtering Rows

The following slides show the base R way and the tidyverse way of subsetting.

Example: NHANES

Selecting Variables

```
df[, c("var1", "var2", etc.)]
df %>%
  select(var1, var2, etc.)
```

Here both do the same thing. The first, using [, is the "base R" way of selecting variables. The second, using the pipe, is the tidyverse way. Both work great so the choice is yours.

Example: NHANES

Filtering Rows

```
df[df$var1 == 1, ]
df %>%
  filter(var1 == 1)
```

Again, both do the same thing. The first, using `[`, is the "base R" way of filtering rows so that you only keep the ones where "var1" in `df` is equal to `1`. Again, the second is the tidyverse way. Whichever you like you should use.

Example: NHANES

Grouping and Summarizing

A major aspect of analysis is comparing groups. Lucky for us, this is very simple in R. I call it the three step summary:

1. Data
2. Group by
3. Summarize

Example: NHANES

Grouping and Summarizing

```
dem_df$citizen <- factor(dem_df$dmdcitzn)
dem_df %>%                                  ## 1. Data
  group_by(citizen) %>%                     ## 2. Group by
  summarize(N = n())                        ## 3. Summarize
```

```
# A tibble: 4 × 2
  citizen     N
   <fctr> <int>
1      1  8685
2      2  1040
3      7    26
4     NA     5
```

Example: NHANES

Grouping and Summarizing

On the previous slide:

- The first column is the grouping variable and the second is the N (number of individuals) by group.
- We can quickly see that there are four levels, currently, to the citizen variable.
    - After some reading of the documentation we see that `1 = Citizen` and `2 = Not a Citizen`.
    - A value of `7` it turns out is a placeholder value for missing.
    - And finally we have an NA category.
        - It's unlikely that we want those to be included in any analyses, unless we are particularly interested in the missingness on this variable.
        - So let's do some simple cleaning to get this where we want it. To do this, we will use the `furniture` package.

Example: NHANES

Grouping and Summarizing

```
install.packages("furniture")
```

```
library(furniture)
## Changes all 7's to NA's
dem_df$citizen <- washer(dem_df$citizen, 7)
## Changes all 2's to 0's
dem_df$citizen <- washer(dem_df$citizen, 2, value=0)
```

Now, our citizen variable is cleaned, with 0 meaning not a citizen and 1 meaning citizen. Let's rerun the code from above with the three step summary:

Example: NHANES

Grouping and Summarizing

```
## Three step summary:
dem_df %>%                              ## 1. Data
  group_by(citizen) %>%                 ## 2. Group by
  summarize(N = n())                    ## 3. Summarize
```

```
# A tibble: 3 × 2
  citizen     N
    <chr> <int>
1       0  1040
2       1  8685
3    <NA>    31
```

Its clear that the majority of the subjects are citizens.

Example: NHANES

Grouping and Summarizing

Check multiple variables at the same time:

```
## Three step summary:
dem_df %>%                                  ## 1. Data
  group_by(citizen) %>%                     ## 2. Group by
  summarize(N = n(),                        ## 3. Summarize
            Age = mean(ridageyr, na.rm=TRUE))
```

```
# A tibble: 3 × 3
  citizen     N      Age
    <chr> <int>    <dbl>
1       0  1040 37.31635
2       1  8685 30.66252
3    <NA>    31 40.35484
```

Example: NHANES

Grouping and Summarizing

On previous slide:

- The `n()` function gives us counts
- The `mean()` function which, shockingly, gives us the mean.
  - Note that if there are `NA`'s in the variable, the mean (and most other functions like it) will give the result `NA`.
  - To have `R` ignore these, we tell the `mean` function to remove the `NA`'s when you compute this using `na.rm=TRUE`.

Example: NHANES

The Grouping and Summarizing Steps

This pattern of grouping and summarizing is something that will follow us throughout the book.

It's a great way to get to know your data well and to make decisions on what to do next with your data.

Example: NHANES

Reshaping

This is a big part of working with data. Unfortunately, it is also a difficult topic to understand without much practice at it. In general, two data formats exist:

1. Wide form
2. Long form

Only when the data is cross-sectional and each individual is a row does this distinction not matter much. Otherwise, if there are multiple measures per individual, or there are multiple individuals per cluster, the distinction between wide and long is very important for modeling and visualization.

Example: NHANES

Wide Form

Wide form generally has one unit (i.e. individual) per row. This generally looks like:

	ID	Var_Time1	Var_Time2
1	1	1.138688557	0.67206981
2	2	-0.926541315	0.30853689
3	3	-0.007108554	0.55613005
4	4	0.533288410	0.23545637
5	5	-0.909166260	0.01326606
6	6	1.396866039	0.73015902
7	7	1.748336183	0.66249056
8	8	0.100194424	0.36643398
9	9	0.511294922	0.08342045
10	10	-0.585448865	0.56180077

Example: NHANES

Long Form

In contrast, long format has the lowest nested unit as a single row. This means that a single ID can span multiple rows, usually with a unique time point for each row as so:

	ID	Time	Var
1	1	1	0.4722128
2	1	2	0.1303989
3	1	3	0.7835221
4	1	4	0.4007190
5	2	1	0.1882725
6	2	2	0.8000024
7	3	1	0.7557883
8	3	2	0.1840514
9	3	3	0.9533038

Quick Sidetrack from NHANES: Reshaping

Wide to Long

With a fake data set, we'll go from wide to long...

```
df_wide <- data.frame("ID"=c(1:10),
                      "Var_Time1"=rnorm(10),
                      "Var_Time2"=runif(10))
df_long <- gather(df_wide, "var_label", "value", 2:3)
```

We provided the data, some variable names, and told it what columns contained the values.

Quick Sidetrack from NHANES: Reshaping

Long to Wide

Now we will go from long to wide using `spread()` from the same package.

```
df_long <- data.frame("ID"=c(1,1,1,1,2,2,3,3,3),
                      "Time"=c(1,2,3,4,1,2,1,2,3),
                      "Var"=runif(9))
df_wide <- spread(df_long, Time, Var)
```

Here, we provided the column name (`Time`) that had the value labels and (`Var`) that contained the values themselves.

With a little bit of code we can move data around without any copy-pasting that is so error-prone.

Example: NHANES

Joining (merging)

The final topic in the chapter is joining data sets.

We currently have 4 data sets that have mostly the same people in them but with different variables. One tells us about the demographics; another gives us information on mental health. We may have questions that ask whether a demographic characteristics is related to a mental health factor. This means we need to merge, or join, our data sets.[4]

[4] Note that this is different than adding new rows but not new variables. Merging requires that we have at least some overlap of individuals in both data sets.

Example: NHANES

Joining (merging)

When we merge a data set, we combine them based on some ID variable(s). Here, this is simple since each individual is given a unique identifier in the variable `seqn`. Within the `dplyr` package there are four main joining functions: `inner_join`, `left_join`, `right_join` and `full_join`. Each join combines the data in slightly different ways.

Example: NHANES

Joining (merging)

Let's first load `dplyr`:

```
library(dplyr)
```

Example: NHANES

Joining (merging)

Inner Join

Here, only those individuals that are in both data sets that you are combining will remain. So if person "A" is in data set 1 and not in data set 2 then he/she will not be included.

```
inner_join(df1, df2, by="IDvariable")
```

Example: NHANES

Joining (merging)

Left or Right Join

This is similar to inner join but now if the individual is in data set 1 then `left_join` will keep them even if they aren't in data set 2. `right_join` means if they are in data set 2 then they will be kept whether or not they are in data set 1.

```
left_join(df1, df2, by="IDvariable")   ## keeps all in df1
right_join(df1, df2, by="IDvariable")  ## keeps all in df2
```

Example: NHANES

Joining (merging)

Full Join

This one simply keeps all individuals that are in either data set 1 or data set 2.

```
full_join(df1, df2, by="IDvariable")
```

Each of the left, right and full joins will have missing values placed in the variables where that individual wasn't found. For example, if person "A" was not in df2, then in a full join they would have missing values in the df1 variables.

Example: NHANES

For our NHANES example, we will use `full_join` to get all the data sets together. Note that in the code below we do all the joining in the same overall step.

```
df <- dem_df %>%
  full_join(med_df, by="seqn") %>%
  full_join(men_df, by="seqn") %>%
  full_join(act_df, by="seqn")
```

So now `df` is the the joined data set of all four. We started with `dem_df` joined it with `med_df` by `seqn` then joined that joined data set with `men_df` by `seqn`, and so on.

Conclusions

In This Chapter:

- You have learned how to manipulate your data in several ways:
  - Summarizing
  - Reshaping
  - Joining

For analyses in the later chapters, we will use this new `df` *object that we concluded with containing NHANES data.*

Also, you'll see that many of these methods apply to more than just manipulating data. As you learn one method, you'll begin to see how easily you can use it in other situations.