

Intro to Tidyverse

Amanda and Pilar

09/03/2020

Intro to Tidyverse

A combination of packages that are useful for: importing, cleaning and transforming, processing and analyzing, visualizing.



Intro to tidyverse

When you install (and load) tidyverse, you're installing (and loading) all of these packages. You can, instead, load the individual packages you'll use.

Install and load tidyverse on your computer.

Notice the message you get:

```
> library(tidyverse)
-- Attaching packages ----- tidyverse 1.3.0 --
v ggplot2 3.3.2      v purrr  0.3.4
v tibble  3.0.3      v dplyr  1.0.2
v tidyr   1.1.0      v stringr 1.4.0
v readr   1.3.1      v forcats 0.5.0
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
> |
```

Tidy Data

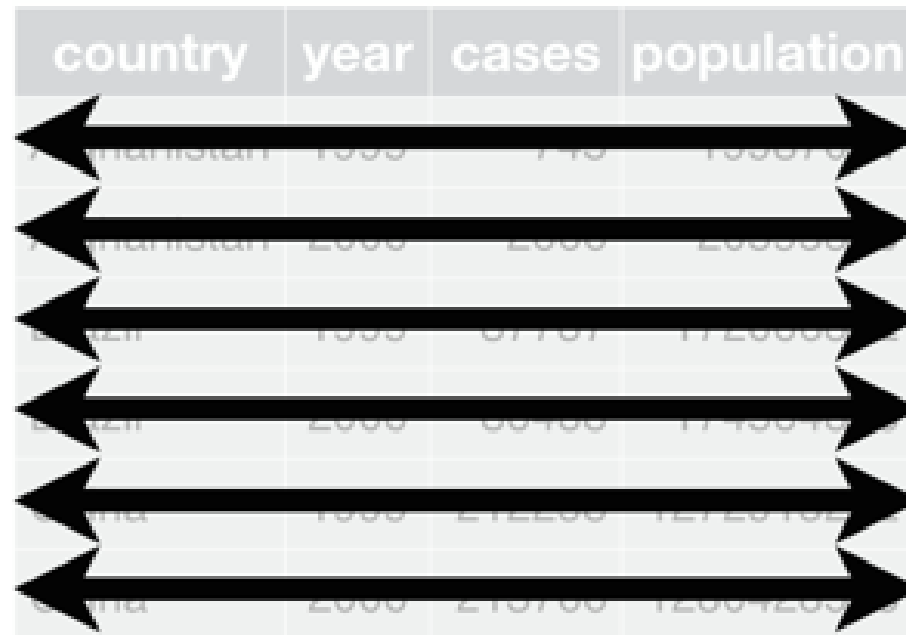
There are three interrelated rules which make a dataset tidy:

- 1) Each variable must have its own column

Tidy Data

There are three interrelated rules which make a dataset tidy:

2) Each observation must have its own row.



country	year	cases	population
Tanzania	1999	743	100079
Tanzania	2000	2000	200000
Tanzania	1999	87707	1720000
Tanzania	2000	88400	1740040
Tanzania	1999	212200	12720400
Tanzania	2000	210700	12004200

observations

Tidy Data

There are three interrelated rules which make a dataset tidy:

3) Each value must have its own cell.

Dplyr

Dplyr is a package for *data manipulation*.

It's intuitive. Check out it's basic commands and guess what each does:

- 1) mutate
- 2) select
- 3) filter
- 4) summarize/summarise
- 5) group_by
- 6) rename



Dplyr In Practice: exploring dataset

Let's use the data on Chicago neighborhoods.

```
chicago <- read.csv("data/Census_Data_selected_2008-2012.csv")
```

Check out the `glimpse` command. How does it compare to the Base R commands we learned last time? (`dim()`, `head()`, and `names()` ?

Dplyr: renaming

Notice the variable names are quite messy. Let's fix this with `rename`.

```
rename(chicago, neighborhood = COMMUNITY.AREA.NAME)
```

Let's check if this worked:

```
names(chicago)
```

```
## [1] "Community.Area.Number"  
## [2] "COMMUNITY.AREA.NAME"  
## [3] "PERCENT.OF.HOUSING.CROWDED"  
## [4] "PERCENT.HOUSEHOLDS.BELOW.POVERTY"  
## [5] "PERCENT.AGED.16..UNEMPLOYED"  
## [6] "PERCENT.AGED.25..WITHOUT.HIGH.SCHOOL.DIPLOMA"  
## [7] "PERCENT.AGED.UNDER.18.OR.OVER.64"  
## [8] "PER.CAPITA.INCOME"  
## [9] "HARDSHIP.INDEX"  
## [10] "ZONE"
```

How come?

PRACTICE

Rename the following variables:

- 1) PERCENT.HOUSEHOLDS.BELOW.POVERTY into "poverty"
- 2) PERCENT.AGED.25..WITHOUT.HIGH.SCHOOL.DIPLOMA into "no_diploma".

Dplyr: summariz(s)ing

summarise reduces observations to a single value based on functions

```
summarise(chicago, mean_pov = mean(poverty))
```

```
##    mean_pov  
## 1 21.73974
```

Let's add other descriptive statistics:

```
summarise(chicago, mean_pov = mean(poverty), sd_pov = sd(poverty), max_pov = max(poverty))
```

```
##    mean_pov    sd_pov max_pov  
## 1 21.73974 11.45723    56.5
```

Dplyr: mutating

The `mutate` function allows us to create new variables based on some transformation of an existing variable.

Let's say we want to convert the poverty numbers into a proportion.

```
chicago <- mutate(chicago, pov_prop = poverty/100)

summary(chicago$pov_prop)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.0330  0.1335  0.1905  0.2174  0.2915  0.5650
```

Notice how we first assign a new variable name and then tell R how we want that variable to be created.

PRACTICE

Imagine that you are creating a simple additive index of 3 indicators: per capita income, percent of housing crowded, and percent without diploma.

Create a new variable with the sum of these values and divide it by 3.

Dplyr: filtering

Say we want to view values or perform operations only on a subset of cases. We can use `filter` to create conditions different conditions.

For instance, let's filter to get neighborhoods that have poverty rates above 20%.

If we run this command, it will simply show us the dataset with the filter.

```
filter(chicago, poverty > 20)
```

We can save this new dataset as a new object:

```
high_pov <- filter(chicago, poverty > 20)
```

You can combine multiple conditions too:

```
filter(chicago, poverty > 20 & diploma > 40 & neighborhood != "Hermosa")
```

PRACTICE

- 1) Create a new dataset that only has North Side neighborhoods.
- 2) Create a new dataset that only has North Side neighborhoods and where per capita income is above \$40,000.
- 3) Which neighborhoods are left?

Dplyr: selecting

Sometimes you will be working with large datasets that have hundreds of variables. But what if you want to work with only some of these variables?

We can create a dataset that has only the variables we want by using the `select` function:

```
small_chicago <- select(chicago, poverty, diploma, neighborhood)
```

```
glimpse(small_chicago)
```

```
## Rows: 78  
## Columns: 3  
## $ poverty      <dbl> 23.6, 17.2, 24.0, 10.9, 7.5, 11.4, 12.3, 12.9, 3.3, 5....  
## $ diploma      <dbl> 18.2, 20.8, 11.8, 13.4, 4.5, 2.6, 3.6, 2.5, 7.4, 11.5,...  
## $ neighborhood <chr> "Rogers Park", "West Ridge", "Uptown", "Lincoln Square..."
```


Dplyr: selecting

- select by name:
 - `select(chicago, poverty, diploma)`
- select by position:
 - `select(chicago, c(1, 3, 10))`
- select by range:
 - `select(chicago, neighborhood:poverty)` or `select(chicago, 1:3)`
- drop variables with `-`
 - `select(chicago, -diploma)`
- "select helpers" that make subsetting variables very easy
 - `one_of()`, `contains()`, `starts_with()`, `ends_with()`, `matches()`

Dplyr: grouping

Many times you'll want to analyze data across groups (regions, gender, age group, political party, etc.). We can use the `group_by` function for this.

```
grouping <- group_by(chicago, ZONE)
```

And now instead of summarizing over "chicago", we summarize over the new object "grouping".

```
summarise(grouping, mean_pov = mean(poverty))
```

```
## # A tibble: 7 x 2
##   ZONE          mean_pov
##   <chr>         <dbl>
## 1 ""           16.1
## 2 "Far South"   25.7
## 3 "North"      13.5
## 4 "Northwest"  15.0
## 5 "South"      31.9
## 6 "Southwest"  19.0
## 7 "West"       31.3
```

Dplyr: Pipes %>%

There's a wonderful tool in dplyr called a **pipe**, which will allow us to do things much more efficiently.

Its symbol is %>% (*tip: keyboard shortcut Ctrl+Shift+M*)

- It lets us perform multiple operations at once, without creating an object for each one of them.
- It sends the result of one function to another function
- Read %>% as "then"
- %>% increases the readability of your code

Dplyr: Pipes %>%

Let's take the example above (mean of poverty by area) and do it in one step. The first thing we do is tell it which dataset we want to run the commands in, and then we simply write the commands, without having to specify the dataset.

```
chicago %>% group_by(ZONE) %>% summarise(mean_pov = mean(poverty))
```

```
## # A tibble: 7 x 2
##   ZONE          mean_pov
##   <chr>         <dbl>
## 1 ""           16.1
## 2 "Far South"   25.7
## 3 "North"       13.5
## 4 "Northwest"  15.0
## 5 "South"       31.9
## 6 "Southwest"  19.0
## 7 "West"       31.3
```

This translates to: in the dataset 'chicago', group by 'zone' and give us the mean of poverty for each zone.

Dplyr: Pipes %>%

You can add as many commands as you want (but nothing in excess is good!).

```
chicago %>% select(poverty, ZONE) %>% filter(poverty > 30) %>%  
  group_by(ZONE) %>% summarise(mean_pov = mean(poverty))
```

```
## # A tibble: 4 x 2  
##   ZONE      mean_pov  
##   <chr>      <dbl>  
## 1 Far South    44.8  
## 2 South       39.2  
## 3 Southwest   40.5  
## 4 West       38.4
```

In this case, we are first selecting only two variables from the dataset, then filtering for neighborhoods with poverty rates over 30%, then grouping by zone, then obtaining the mean poverty rate for each zone.

PRACTICE

1) Combine these commands into a single one using %>%

```
chicago_subset <- select(chicago, neighborhood, ZONE, poverty, diploma, HARDSHIP.INDEX)

chicago_subset <- filter(chicago_subset, poverty < 50)

grouping <- group_by(chicago_subset, ZONE)

summarise(grouping, index_avg = mean(HARDSHIP.INDEX), mean(poverty), mean(diploma))
```

```
## # A tibble: 7 x 4
##   ZONE          index_avg `mean(poverty)` `mean(diploma)`
##   <chr>          <dbl>          <dbl>          <dbl>
## 1 ""              NA              16.1             10
## 2 "Far South"     57.4              22.9            18.6
## 3 "North"        21.9              13.5            12.3
## 4 "Northwest"    48                15.0            26.7
## 5 "South"        55.8              30.1            17.6
## 6 "Southwest"    59                19.0            26.1
## 7 "West"         68.6              31.3            27.9
```

PRACTICE

2) Notice how there is one Zone that is unidentified (i.e. it has no name).

Try filtering out the cases that aren't associated to any zone. In other words, filter out that missing Zone category.

MORE PRACTICE

Using the World Value Survey dataset from yesterday, let's explore people's liberal-conservative values across the world.

- 1) Create a subset with the following variables: Q106-Q109, Q240, Q184-Q186. (Refer to codebook to see what these variables are.)
- 2) Keep only observations from the following countries: US, Colombia, Kazakhstan, Japan, Zimbabwe
- 3) Rename the following variables:
 - Q106 to "income_eq"
 - Q240 to "left_right"
 - Q184 to "just_abort"
- 4) Find the mean of "left_right" for the total sample. Then find the mean of "left_right" in each country.
- 5) Now compare this with the mean of "income_eq" and "just_abort". (You can explore the other variables in the dataset as well.) What does this tell us about liberal-conservative values?