

# Wrangling Data in R

## Intermediate Data Analytics - Dat 280

Jan 29, 2024

## New homework policy (for future homeworks): Upload your script to Canvas!

- ▶ Turns out I can't easily download all your scripts at once on the cloud.
- ▶ Upload your script to Canvas by 10AM each Friday when homework is due.
- ▶ Click the checkbox next to your script name in the bottom right > click the gear > click export

## Another new homework policy: test all lines for errors!

- ▶ Before you submit, clear your global environment (click the broom) and run every line.
- ▶ If there are any errors:
  - ▶ fix them
  - ▶ if you can't fix, ask me about it
  - ▶ if you can't ask me about it, comment it out by putting a # at the start of the line.
- ▶ Moving forward:
  - ▶ You will lose points if I have to rewrite your code to get it to run.
  - ▶ Every line must run for it to be tested by the grading script.
  - ▶ Wrong answer is not the same as it not running at all.

## Best practices for class, moving forward

- ▶ Don't get distracted troubleshooting code while I'm lecturing.
  - ▶ Wait until I cut you loose to work on exercises, then ask me for help.
  - ▶ I need you to pay attention so you don't fall behind.
- ▶ I will work in more opportunities for working on your own and with others during lectures.
  - ▶ That way we keep the talk to a minimum while I'm lecturing.
  - ▶ More opportunities for me to help you troubleshoot and for you to write original code.
- ▶ Save the discussion/troubleshooting with your neighbor for when I cut you loose.

# Today's gameplan

- ▶ We are going to work through the most common wrangling needs/uses in R for data analytics.
  - ▶ Wrangling is the process of reshaping our data so we can use to answer the question we want to answer.
- ▶ I will cut you loose for some practice exercises. You can help your neighbor, get help from me, etc.

# The power of the tidyverse



- ▶ Tidyverse comes equipped with several packages that include functions that are very useful to us, dplyr and ggplot2 in particular
  - ▶ summarize, group\_by, mutate, filter, select, and arrange are some of the most powerful
  - ▶ We used a couple of these on Friday.

## Another package: reshape2

- ▶ Reshape2 is equipped for helping us convert data from wide to long.
- ▶ Long, or “tidy”, is often preferable. Some data comes as wide, and we must convert it.

**Wide Format**

Team	Points	Assists	Rebounds
A	88	12	22
B	91	17	28
C	99	24	30
D	94	28	31

**Long Format**

Team	Variable	Value
A	Points	88
A	Assists	12
A	Rebounds	22
B	Points	91
B	Assists	17
B	Rebounds	28
C	Points	99
C	Assists	24
C	Rebounds	30
D	Points	94
D	Assists	28
D	Rebounds	31

# The guided exercise

- ▶ Let's go step-by-step in a typical clean/wrangle of an interesting dataset.
- ▶ We may not need all of them, but get in the habit of loading and learning:
  - ▶ tidyverse, reshape2, janitor, and lubridate

```
library(tidyverse)
library(reshape2)
library(janitor)
library(lubridate)
```



## Step 1: Load in the data and determine how it is organized.

- ▶ Our data is citywide arrests in Philadelphia from 2010 to now.

```
arrests <- read.csv("arrests.csv")
```

- ▶ EXERCISES:
  - ▶ Is this data in wide or long format (with respect to offenses)?
  - ▶ Can you describe this data in one sentence?

## Step 1 cont.: How would I describe this data?

- ▶ Counts of individual offense types for arrests grouped by race and date.

## Step 2: Convert the data to long (also called “tidy”) if it is wide.

- ▶ We will use `melt()` from the package `reshape2`.
- ▶ Specify `id.vars`: the variables to keep.
  - ▶ Everything else gets put under a categorical variable with an accompanying count variable.
  - ▶ the `c()` is important because it allows us to keep multiple variables intact

```
arrests_long <- melt(arrests, id.vars = c("defendant_race",  
    "day"))  
head(arrests_long, 2)
```

```
##   defendant_race          day          variable  
## 1           Asian 2010-01-01 05:00:00+00 Aggravated.Assault  
## 2           Asian 2010-01-04 05:00:00+00 Aggravated.Assault
```

## Step 3: Fix names and formats of variables, if necessary.

- ▶ Try `as.Date`, if that fails, then we may need to use lubridate commands instead.

```
as.Date(arrests_long$day)
```

- ▶ I hid the results to save space. Hopefully you see that it reads the dates just fine.
- ▶ Let's make a variable for it in the dataframe.

```
arrests_long$date <- as.Date(arrests_long$day)
```

## Step 3 cont.: Renaming our “melted” variables.

- ▶ Melt command will default your new category and counts to “variable” and “value”.

```
arrests_long <- rename(arrests_long,  
  crime = "variable", arrests = "value")
```

## Step 4: Aggregate the data at the level you desire.

- ▶ This is where dplyr AND lubridate come in handy.
  - ▶ lubridate has a function called floor\_date()
  - ▶ syntax: floor\_date(datevarname, "unit")
  - ▶ floor\_date() groups data by your unit of choice and displays it by the first day of that month.

```
arrests_monthly <- arrests_long %>%  
  group_by(floor_date(date, "month"),  
           crime) %>%  
  summarize(arrests = sum(arrests)) %>%  
  rename(month = "floor_date(date, \"month\")")
```

- ▶ I use rename at the end because our date will end up being named floor\_date(date, "month")
  - ▶ This is a very tedious name for a variable and requires single quotes around it since there are quotes in it already.

## Step 5: Visualize the data.

- If data is the first argument, you don't need to write "data=", I just did that to remind you.

```
ggplot(data = arrests_monthly, mapping = aes(month,
arrests)) + geom_line(aes(color = crime))
```



That last plot is overwhelming. Let's cut down on the categories.

- ▶ We will **replace** our data argument in ggplot with a subset of the data with whichever crimes we want.
- ▶ `grep1()` allows us to find only those crimes that contain a certain string.
- ▶ syntax for subset: `subset(data, condition)`
- ▶ syntax for `grep1`: `grep1("string", variablename)`
  - ▶ `c()` doesn't work for `grep1()`, separate strings without quotes and use a `|` symbol.



```
ggplot(data = subset(arrests_monthly,  
  grepl("Assault|Rob|Homicide|Violent",  
    arrests_monthly$crime)), mapping = aes(month,  
arrests)) + geom_line(aes(color = crime)) +  
labs(title = "Arrests by Month",  
  x = "Month", y = "Arrests")
```



## Exercise: Further clean the data and improve the graph.

1. Remove January 2024, as it is incomplete, so we get that weird dip at the end.
2. Add nice labels and a title to the graph.
3. Replicate the (nicer) graph with theft and burglary.

```
arrests_monthly_pre2024 <- subset(arrests_monthly,  
  year(arrests_monthly$month) < 2024)
```

EXERCISE: Read in tom.csv and convert it from wide to long.

	date_value	dispoType	variable	value
1	2014-01-02	Dismissed/Withdrawn/Etc	Homicide..Shooting	0
2	2014-01-02	Diversion	Homicide..Shooting	0
3	2014-01-02	Exonerated/Won on Appeal	Homicide..Shooting	0
4	2014-01-02	Guilty	Homicide..Shooting	0
5	2014-01-02	Guilty Plea/Nolo	Homicide..Shooting	0
6	2014-01-02	Not Guilty/Acquittal	Homicide..Shooting	0
7	2014-01-02	Total	Homicide..Shooting	0
8	2014-01-03	Dismissed/Withdrawn/Etc	Homicide..Shooting	0
9	2014-01-03	Diversion	Homicide..Shooting	0
10	2014-01-03	Exonerated/Won on Appeal	Homicide..Shooting	0

Figure 1: It should look like this if you did it correctly.

EXERCISE: Sum crimes by month and outcome in your tom dataframe