POLS 309 – Working with data

Casey Crisman-Cox[1]

Spring 2022

[1]Much of this content is adapted from Brenton Kenkel's e-book Practical Data Analysis for Political Scientists https://bkenkel.com/pdaps/

Welcome back

Reminder office hours are

Me (Zoom)

- Monday 4-5pm

Mingsi (Zoom or in person)

- Thursday 4-5pm

Sam (Zoom)

- Wednesday 11am-Noon

Agenda

- REVIEW: What are data?
- Types of data
- Presenting data
- Using graphs to present data

What do we know?

Last time we talk about random variables. Observational data come from random variable.

For instance, suppose that we are working quality control at a brewery. Each individual beer may be good or skunked. We observe

- 10 beers in a day (data)
- from an unknown process that determines if they're skunked or not (Random variable)

Difference between data and the data generating process

The **data generating process** (DGP) is a **model** of how observed data are created

- The DGP is a random variable
- The data are realizations from that random variable

In the beer example, we can imagine the DGP as Nature/God/Universe/Fates/Loki/etc flipping a (probably unfair) coin that determines whether any given beer is skunked.

Difference between data and the data generating process

The **data generating process** (DGP) is a **model** of how observed data are created

- The DGP is a random variable
- The data are realizations from that random variable

In the beer example, we can imagine the DGP as Nature/God/Universe/Fates/Loki/etc flipping a (probably unfair) coin that determines whether any given beer is skunked.

In other cases we might think about more complicated models that reflect how things like an individual's vote choice is determined which may include a combination of observed individual traits plus unknown factors that we can chalk up as idiosyncrasies or acts of Nature.

Difference between data and the data generating process

The **data generating process** (DGP) is a **model** of how observed data are created

- The DGP is a random variable
- The data are realizations from that random variable

In the beer example, we can imagine the DGP as Nature/God/Universe/Fates/Loki/etc flipping a (probably unfair) coin that determines whether any given beer is skunked.

In other cases we might think about more complicated models that reflect how things like an individual's vote choice is determined which may include a combination of observed individual traits plus unknown factors that we can chalk up as idiosyncrasies or acts of Nature.

Data is what we observe, the DGP is our model for how those observables come to be.

Types of data

In the brewery case, we have an example of **discrete data**.

- Discrete data take on a set of finite or fixed values

Types of data

In the brewery case, we have an example of **discrete data**.

- Discrete data take on a set of finite or fixed values
- Binary (unordered) $\{$not skunked, skunked$\} \rightarrow \{0, 1\}$

Types of data

In the brewery case, we have an example of **discrete data**.

- Discrete data take on a set of finite or fixed values
- Binary (unordered) $\{$not skunked, skunked$\} \rightarrow \{0, 1\}$
- Count (ordered, finite or infinite) $\{$Now many are skunked$\}$ *to* $\{0, 1, 2, \ldots\}$

Types of data

In the brewery case, we have an example of **discrete data**.

- Discrete data take on a set of finite or fixed values
- Binary (unordered) $\{$not skunked, skunked$\} \rightarrow \{0, 1\}$
- Count (ordered, finite or infinite) $\{$Now many are skunked$\}$ *to* $\{0, 1, 2, \ldots\}$
- Categorical (ordered or unordered)

Types of data

In the brewery case, we have an example of **discrete data**.

- Discrete data take on a set of finite or fixed values
- Binary (unordered) $\{$not skunked, skunked$\} \rightarrow \{0, 1\}$
- Count (ordered, finite or infinite) $\{$Now many are skunked$\}$ *to* $\{0, 1, 2, \ldots\}$
- Categorical (ordered or unordered)
- What would an example of an ordered categorical? An unordered? (Not necessary beer related)

Types of data

In the brewery case, we have an example of **discrete data**.

- Discrete data take on a set of finite or fixed values
- Binary (unordered) {not skunked, skunked} $\rightarrow$ {0, 1}
- Count (ordered, finite or infinite) {Now many are skunked} *to* {0, 1, 2, ...}
- Categorical (ordered or unordered)
- What would an example of an ordered categorical? An unordered? (Not necessary beer related)
- Ordered: How much you like something {A little, Neutral, A lot}
- Unordered: Race {White, Black or African American, Asian American, American Indian/Alaska Native, and Native Hawaiian/Pacific Islander}

Types of data

Alternatively we sometimes have **continuous data** that can take on any number of possible values, such as

- Strictly positive/negative: Height (always greater than 0)
- Weakly positive/negative: Income (at least 0)
- Bounded: Presidential approval over time (0-100) or Net approval (-100-100)

Other examples that fit into these boxes?

What is a data frame?

In a particular study, we are interested in answering a question using a data frame

A data frame is an $N \times M$ matrix where each cell contains a piece of data

In R we will use `data.frame` objects to represent this.

The Tidyverse

One very set of tools for working with data in R is called the "Tidyverse."

- It is a set of functions designed to make working with data easier to program and read
- It has quickly become the standard for working with and manipulating data
- There is a spectrum from true believers to light users. So the things I teach may not match exactly with what your future co-workers or even your TA's prefer (but it should be close)

What is in the tidyverse

There are **a lot** of tidy packages in R. Today we want three of them

```
library(readr) # for importing/exporting data
library(dplyr) #for working with data
library(tidyr) #also for working with data
```

These three packages will do a lot of work for us going forward. Your TAs may also introduce you to other helpful packages along the way. We'll also use:

```
library(knitr) #making tables
```

What does it mean to be "tidy"

According to the main authors behind the tidyverse the main traits that define tidy data

1. Each variable is a column
2. Each observation is a row

Note that this requires us to define what an observation is ahead of time. Common units of observation for social science include

1. Individuals
2. Individual–time
3. Countries
4. Country–years
5. Dyad–years (pairs of countries over time)
6. U.S. states
7. U.S. state–years
8. And so on...

Let's see an example

Reading data

The main way that we'll read data into R this semester is with the `read_csv` function from the `readr` package. Note that this only works for files that are CSV files. There are other functions for excel files (.xls, .xlsx), Stata files (.dta), and others your TA will review these in lab.

```
cw.data <- read_csv("civilwardata.csv", show_col_types = FALSE)
head(cw.data) #display first 6 rows
```

```
## # A tibble: 6 x 8
##   country  year ccode onset    pop gdpen   Oil
##   <chr>   <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1 USA      1945     2     0 140969  7.63     0
## 2 USA      1946     2     0 141936  7.65     0
## 3 USA      1947     2     0 142713  8.02     0
## 4 USA      1948     2     0 145326  8.27     0
## 5 USA      1949     2     0 147987  8.04     0
## 6 USA      1950     2     0 152273  8.77     0
## # ... with 1 more variable: ethfrac <dbl>
```

What's the unit of observation?

Looking at data

```r
summary(cw.data)
```

```
##    country                year          ccode      
##  Length:6610        Min.   :1945   Min.   :  2.0  
##  Class :character   1st Qu.:1964   1st Qu.:230.0  
##  Mode  :character   Median :1977   Median :451.0  
##                     Mean   :1976   Mean   :450.6  
##                     3rd Qu.:1989   3rd Qu.:663.0  
##                     Max.   :1999   Max.   :950.0  
##                                                   
##      onset              pop         
##  Min.   :0.00000   Min.   :    222  
##  1st Qu.:0.00000   1st Qu.:   3217  
##  Median :0.00000   Median :   8137  
##  Mean   :0.01679   Mean   :  31787  
##  3rd Qu.:0.00000   3rd Qu.:  20601  
##  Max.   :1.00000   Max.   :1238599  
##                    NA's   :177      
##      gdpen             Oil             ethfrac      
##  Min.   : 0.048   Min.   :0.0000   Min.   :0.0010  
##  1st Qu.: 0.943   1st Qu.:0.0000   1st Qu.:0.1073  
##  Median : 2.028   Median :0.0000   Median :0.3255  
##  Mean   : 3.694   Mean   :0.1295   Mean   :0.3854  
##  3rd Qu.: 4.552   3rd Qu.:0.0000   3rd Qu.:0.6637  
##  Max.   :66.735   Max.   :1.0000   Max.   :0.9250  
##  NA's   :227                                       
```

The pipe operator

The tidyverse is heavily characterized by its use of the pipe operator %>%. This is a tool designed to make code easier to read and write. Consider a situation where we want to apply functions f, g, and h to some variable x. We could think of it as

- $f(g(h(x)))$

OR

- $x \to h() \to g() \to f()$

The pipe notation is the latter, it allows us to write in the order we want to do things rather than lots of difficult to read nesting.

Making new variables

Often times we want to make adjusts to variables or make new ones. In our civil war data, for example population is measured in 100s of people, maybe we want it to be 100,000s. How do we do this

Making new variables

Often times we want to make adjusts to variables or make new ones. In our civil war data, for example population is measured in 100s of people, maybe we want it to be 100,000s. How do we do this (divide by 1,000)

```
summary(cw.data$pop)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.     Max. 
##      222    3217    8137   31787   20601 1238599 
##     NA's 
##      177
```

```
cw.data <- cw.data %>%
  mutate(pop = pop/1000)
summary(cw.data$pop)
```

```
##      Min.  1st Qu.   Median     Mean  3rd Qu. 
##     0.222    3.217    8.137   31.787   20.601 
##      Max.     NA's 
## 1238.599      177
```

Long and wide data

Sometime the data we have is not tidy. Common examples include "wide" data.

- Tidy data is sometimes called long data
- Wide data can be something weird like countries on rows, years on columns, and a single variable filling the table.

Let's look at an example

Wide data (Free press)

Wide data is not tidy. For example

```
press.dat <- read_csv("FreedomHouse_Pressdata.csv",
                      show_col_types = FALSE)
head(press.dat)[,1:5]
```

```
## # A tibble: 6 x 5
##   country                ccode X1979 X1980 X1981
##   <chr>                  <dbl> <chr> <chr> <chr>
## 1 Afghanistan              700 NF    NF    NF
## 2 Albania                  339 NF    NF    NF
## 3 Algeria                  615 NF    NF    NF
## 4 Andorra                  232 <NA>  <NA>  <NA>
## 5 Angola                   540 NF    NF    NF
## 6 Antigua and Barbuda       58 <NA>  <NA>  F
```

Reshaping data

How can we make this data long? The `tidyr` package has the tool for us

```
press.long <- pivot_longer(press.dat,
                           # cols asks what var we want
                           # to swing around
                           # starts_with helps us here
                           cols=starts_with("X"),
                           #new name for old column name
                           names_to = "year",
                           #new var name
                           values_to = "free.press")
```

Did it work?

```r
head(press.long)
```

```
## # A tibble: 6 x 4
##   country     ccode year  free.press
##   <chr>       <dbl> <chr> <chr>
## 1 Afghanistan   700 X1979 NF
## 2 Afghanistan   700 X1980 NF
## 3 Afghanistan   700 X1981 NF
## 4 Afghanistan   700 X1982 NF
## 5 Afghanistan   700 X1983 NF
## 6 Afghanistan   700 X1984 NF
```

Googling things

Note that we have an “X” in front of our years, that may cause problems if we want to use the years for anything (the reason is that R won’t let variable names start with a number). How can we remove this? This is a prime time to try Googling for an answer

GOOGLE: Remove first letter from variable R

Let's do it

```r
press.long <- press.long %>%
  mutate(year=sub("X", "", year))
head(press.long)
```

```
## # A tibble: 6 x 4
##   country     ccode year  free.press
##   <chr>       <dbl> <chr> <chr>
## 1 Afghanistan   700 1979  NF
## 2 Afghanistan   700 1980  NF
## 3 Afghanistan   700 1981  NF
## 4 Afghanistan   700 1982  NF
## 5 Afghanistan   700 1983  NF
## 6 Afghanistan   700 1984  NF
```

Just one more thing (Do you guys know the show Colombo?)

Note that year is still not quite right. It's stll listed as a character not a number

```
press.long <- press.long %>%
  mutate(year = as.integer(year))
head(press.long)
```

```
## # A tibble: 6 x 4
##   country     ccode  year free.press
##   <chr>       <dbl> <int> <chr>
## 1 Afghanistan   700  1979 NF
## 2 Afghanistan   700  1980 NF
## 3 Afghanistan   700  1981 NF
## 4 Afghanistan   700  1982 NF
## 5 Afghanistan   700  1983 NF
## 6 Afghanistan   700  1984 NF
```

What can we do with this now

```r
summary(press.long)
```

```
##    country                ccode             year     
##  Length:6798        Min.   :  2.0   Min.   :1979  
##  Class :character   1st Qu.:290.0   1st Qu.:1987  
##  Mode  :character   Median :438.5   Median :1995  
##                     Mean   :464.1   Mean   :1995  
##                     3rd Qu.:678.0   3rd Qu.:2003  
##                     Max.   :990.0   Max.   :2011  
##                     NA's   :132                   
##   free.press       
##  Length:6798       
##  Class :character  
##  Mode  :character  
##                    
##                    
##                    
##                    
```

Character and factor

Is that an informative summary for free press?

Character and factor

Is that an informative summary for free press? Not really.

- Sometimes we want to make a character/string variable in a categorical variable
- In R, these are call factors

Character and factor

Is that an informative summary for free press? Not really.

- Sometimes we want to make a character/string variable in a categorical variable
- In R, these are call factors

```
press.long <- press.long %>%
  mutate(free.press = factor(free.press,
                             levels=c("NF", "PF", "F")))
summary(press.long$free.press)
```

```
##   NF   PF    F NA's
## 2095 1626 2087  990
```

- The `levels` option tells it the ordering of the categories (default is alphabetical)

Merging data

We now have two different data sets, we may want to combine them. Here is where we use the `merge` function

```
data.merged <- merge(x=cw.data, y=press.long,
                     #variable names to match on
                     by=c("ccode", "year"))
dim(data.merged)
```

```
## [1] 3032   10
```

```
dim(cw.data)
```

```
## [1] 6610    8
```

```
dim(press.long)
```

```
## [1] 6798    4
```

Merging data

Why do these have difference sizes?

- By default `merge` only keeps observations where `x` and `y` overlap. You can adjust this with
- `all` (keep all obs from both)
- `all.x` (keep all obs from `x` discard obs that only `y` has)
- `all.y` (keep all obs from `y` discard obs that only `x` has)

New example

```
dat1 <- data.frame(statecode =rep(1:4, each=2),
                   year=rep(c(1990, 1991),2),
                   treated = c(0,0,0,1,0,0,0,1))
dat1
```

```
##   statecode year treated
## 1         1 1990       0
## 2         1 1991       0
## 3         2 1990       0
## 4         2 1991       1
## 5         3 1990       0
## 6         3 1991       0
## 7         4 1990       0
## 8         4 1991       1
```

Throughout we want to maintain exactly these 8 observations

Merging pitfall: 1 using alls

```r
dat2 <- data.frame(state = c(1,1,2,3),
                   year = c(1990, 1990, 1990,1990),
                   gov1990 = c("Dem", "GOP", "Dem", "GOP"))
merge(dat1, dat2,
      by.x=c("statecode", "year"),
      by.y=c("state", "year"))
```

```
##   statecode year treated gov1990
## 1         1 1990       0     Dem
## 2         1 1990       0     GOP
## 3         2 1990       0     Dem
## 4         3 1990       0     GOP
```

What `all` option do we need?

Merging pitfall: 1 using alls

```r
dat2 <- data.frame(state = c(1,1,2,3),
                   year = c(1990, 1990, 1990,1990),
                   gov1990 = c("Dem", "GOP", "Dem", "GOP"))
merge(dat1, dat2,
      by.x=c("statecode", "year"),
      by.y=c("state", "year"))
```

```
##   statecode year treated gov1990
## 1         1 1990       0     Dem
## 2         1 1990       0     GOP
## 3         2 1990       0     Dem
## 4         3 1990       0     GOP
```

What `all` option do we need? `all.x`

Merging pitfall 2: duplicates

```
merge(dat1, dat2,
      by.x=c("statecode", "year"),
      by.y=c("state", "year"),
      all.x=TRUE)
```

```
##   statecode year treated gov1990
## 1         1 1990       0     Dem
## 2         1 1990       0     GOP
## 3         1 1991       0    <NA>
## 4         2 1990       0     Dem
## 5         2 1991       1    <NA>
## 6         3 1990       0     GOP
## 7         3 1991       0    <NA>
## 8         4 1990       0    <NA>
## 9         4 1991       1    <NA>
```

Now what's wrong?

Merging pitfall 2: duplicates

```r
merge(dat1, dat2,
      by.x=c("statecode", "year"),
      by.y=c("state", "year"),
      all.x=TRUE)
```

```
##   statecode year treated gov1990
## 1         1 1990       0     Dem
## 2         1 1990       0     GOP
## 3         1 1991       0    <NA>
## 4         2 1990       0     Dem
## 5         2 1991       1    <NA>
## 6         3 1990       0     GOP
## 7         3 1991       0    <NA>
## 8         4 1990       0    <NA>
## 9         4 1991       1    <NA>
```

Now what's wrong? With duplicates you need to figure out why they occur.

Merging pitfall 2: duplicates

```
merge(dat1, dat2,
      by.x=c("statecode", "year"),
      by.y=c("state", "year"),
      all.x=TRUE)
```

```
##   statecode year treated gov1990
## 1         1 1990       0     Dem
## 2         1 1990       0     GOP
## 3         1 1991       0    <NA>
## 4         2 1990       0     Dem
## 5         2 1991       1    <NA>
## 6         3 1990       0     GOP
## 7         3 1991       0    <NA>
## 8         4 1990       0    <NA>
## 9         4 1991       1    <NA>
```

Now what's wrong? With duplicates you need to figure out why they occur. Typos in the data? Midyear changes (which do you want?)? Something else?

Edit before

Suppose it was a midyear change (special election) and we just want the first one. We can just delete it first.

```
dat2 <- dat2 %>% filter(! (state==1 & gov1990=="GOP" ))
merge(dat1, dat2, by.x=c("statecode", "year"),
      by.y=c("state", "year"),
      all.x=TRUE)
```

```
##   statecode year treated gov1990
## 1         1 1990       0     Dem
## 2         1 1991       0    <NA>
## 3         2 1990       0     Dem
## 4         2 1991       1    <NA>
## 5         3 1990       0     GOP
## 6         3 1991       0    <NA>
## 7         4 1990       0    <NA>
## 8         4 1991       1    <NA>
```

One more thought

What if we didn't want those NAs? Suppose we just wanted `gov` to reflect the governor at the start of the study? (i.e., put the 1990 value in for both years). How might we do that with merge?

One more thought

What if we didn't want those NAs? Suppose we just wanted gov to reflect the governor at the start of the study? (i.e., put the 1990 value in for both years). How might we do that with merge? Just merge on states not years

```
dat2 <- dat2 %>% select(!year)
# remove year from dat2 so we don't have two year vars
merge(dat1, dat2, by.x=c("statecode"), by.y=c("state"),
      all.x=TRUE)
```

```
##   statecode year treated gov1990
## 1         1 1990       0     Dem
## 2         1 1991       0     Dem
## 3         2 1990       0     Dem
## 4         2 1991       1     Dem
## 5         3 1990       0     GOP
## 6         3 1991       0     GOP
## 7         4 1990       0    <NA>
## 8         4 1991       1    <NA>
```

Summaries: Descriptive statistics

We saw before that the `summary` command is great for a first cut, but it's often not in a form we want to show a client. For that we are going to use `summarise`, but for this we need to specify what we want to see.

What are some attributes of the data that might be interesting?

Summaries: Descriptive statistics

We saw before that the `summary` command is great for a first cut, but it's often not in a form we want to show a client. For that we are going to use `summarise`, but for this we need to specify what we want to see.

What are some attributes of the data that might be interesting?

Sample

- min
- max
- mean
- variance or standard deviation
- others

```
my.summary <- function(x){
  return(c(min(x, na.rm=TRUE), mean(x, na.rm=TRUE),
           sd(x, na.rm=TRUE), max(x, na.rm=TRUE)))
}
```

Summaries: Descriptive statistics

```
summary.stats <- data.merged %>%
  select(c("onset", "pop", "gdpen", "Oil", "ethfrac")) %>%
  summarize(across(.fns=my.summary))
summary.stats
```

```
##       onset        pop      gdpen       Oil
## 1 0.0000000    0.29300  0.1962578 0.0000000
## 2 0.0171504   34.27207  4.4956290 0.1576517
## 3 0.1298531  117.62707  4.6483540 0.3644742
## 4 1.0000000 1238.59938 31.9689999 1.0000000
##     ethfrac
## 1 0.0010000
## 2 0.4101338
## 3 0.2847634
## 4 0.9250348
```

what's missing?

Summaries: Descriptive statistics

```
summary.stats <- summary.stats %>%
  mutate(stat=c("min", "mean", "std. dev", "max"),
         .before=1) #.before tells it where to put the new col
summary.stats
```

```
##       stat     onset        pop      gdpen
## 1      min 0.0000000    0.29300  0.1962578
## 2     mean 0.0171504   34.27207  4.4956290
## 3 std. dev 0.1298531  117.62707  4.6483540
## 4      max 1.0000000 1238.59938 31.9689999
##          Oil   ethfrac
## 1 0.0000000 0.0010000
## 2 0.1576517 0.4101338
## 3 0.3644742 0.2847634
## 4 1.0000000 0.9250348
```

Summaries: Descriptive statistics by group

```
summary.stats.by.press <- data.merged %>%
  select(!c(year, starts_with("country"), ccode)) %>%
  group_by(free.press) %>%
  summarise(across(.fns=mean, na.rm=TRUE))
summary.stats.by.press
```

```
## # A tibble: 4 x 6
##   free.press   onset   pop gdpen    Oil ethfrac
##   <fct>        <dbl> <dbl> <dbl>  <dbl>   <dbl>
## 1 NF         0.0225   33.8  2.59 0.231    0.462
## 2 PF         0.0214   31.7  2.90 0.137    0.452
## 3 F          0.00426  38.4  8.04 0.0883   0.301
## 4 <NA>       0.0430   22.2  4.21 0.140    0.490
```

What happened there?

Glad you asked. Let's look at that line

```
summary.stats.by.press <- data.merged %>%
  select(!c(year, starts_with("country"), ccode)) %>%
  group_by(free.press) %>%
  summarise(across(.fns=mean, na.rm=TRUE))
```

and break it down

- `select` we know this, we're going to subset the data

What happened there?

Glad you asked. Let's look at that line

```
summary.stats.by.press <- data.merged %>%
  select(!c(year, starts_with("country"), ccode)) %>%
  group_by(free.press) %>%
  summarise(across(.fns=mean, na.rm=TRUE))
```

and break it down

- `select` we know this, we're going to subset the data
- `!c(year, starts_with("country"), ccode)` these columns
- `!` NOT
- `year`
- `starts_with("country")` Any column that starts with "country"
- `ccode`

What happened there?

Glad you asked. Let's look at that line

```
summary.stats.by.press <- data.merged %>%
  select(!c(year, starts_with("country"), ccode)) %>%
  group_by(free.press) %>%
  summarise(across(.fns=mean, na.rm=TRUE))
```

and break it down

- `select` we know this, we're going to subset the data
- `!c(year, starts_with("country"), ccode)` these columns
- `!` NOT
- `year`
- `starts_with("country")` Any column that starts with "country"
- `ccode`
- `group_by` whatever happens next happens to each group

What happened there?

Glad you asked. Let's look at that line

```
summary.stats.by.press <- data.merged %>%
  select(!c(year, starts_with("country"), ccode)) %>%
  group_by(free.press) %>%
  summarise(across(.fns=mean, na.rm=TRUE))
```

and break it down

- `select` we know this, we're going to subset the data
- `!c(year, starts_with("country"), ccode)` these columns
- `!` NOT
- `year`
- `starts_with("country")` Any column that starts with "country"
- `ccode`
- `group_by` whatever happens next happens to each group
- `summarise` we know this, we're going to summarize some data

What happened there?

Glad you asked. Let's look at that line

```
summary.stats.by.press <- data.merged %>%
  select(!c(year, starts_with("country"), ccode)) %>%
  group_by(free.press) %>%
  summarise(across(.fns=mean, na.rm=TRUE))
```

and break it down

- `select` we know this, we're going to subset the data
- `!c(year, starts_with("country"), ccode)` these columns
- `!` NOT
- `year`
- `starts_with("country")` Any column that starts with "country"
- `ccode`
- `group_by` whatever happens next happens to each group
- `summarise` we know this, we're going to summarize some data
- `across` We're going to do the same function to all the columns

What happened there?

Glad you asked. Let's look at that line

```
summary.stats.by.press <- data.merged %>%
  select(!c(year, starts_with("country"), ccode)) %>%
  group_by(free.press) %>%
  summarise(across(.fns=mean, na.rm=TRUE))
```

and break it down

- `select` we know this, we're going to subset the data
- `!c(year, starts_with("country"), ccode)` these columns
- `!` NOT
- `year`
- `starts_with("country")` Any column that starts with "country"
- `ccode`
- `group_by` whatever happens next happens to each group
- `summarise` we know this, we're going to summarize some data
- `across` We're going to do the same function to all the columns
- `mean` function to apply
- `na.rm=TRUE` option for mean

Presenting descriptive statistics

Nobody, **but nobody** wants to see ugly computer monospaced text. When you're preparing analysis for other people to read, make it look good and readable, for example

Presenting descriptive statistics

Nobody, **but nobody** wants to see ugly computer monospaced text. When you're preparing analysis for other people to read, make it look good and readable, for example

```
kable(summary.stats, digits=2,
      caption="Summary statistics",
      col.names =  c("", "Onset", "Pop.",
                     "GDP per cap.",
                     "Oil",
                     "Eth. Frac."))
```

Table 1: Summary statistics

	Onset	Pop.	GDP per cap.	Oil	Eth. Frac.
min	0.00	0.29	0.20	0.00	0.00
mean	0.02	34.27	4.50	0.16	0.41
std. dev	0.13	117.63	4.65	0.36	0.28
max	1.00	1238.60	31.97	1.00	0.93

Summaries: Tabulations

Sometimes with binary and categorical variables it makes as much or more sense to present tabulations or frequencies

```
table(data.merged$Oil)
```

```
##
##    0    1
## 2554  478
```

```
table(data.merged$free.press)
```

```
##
##   NF   PF    F
## 1157  842  940
```

To give your reader a sense as to distribution of the categories

Crosstabs

You can also use crosstabs to make a point about an interesting trend in your data.

Are civil conflicts more likely in freer states?

```
table(Onset=data.merged$onset,
      Press=data.merged$free.press)
```

```
##       Press
## Onset    NF   PF    F
##     0 1131  824  936
##     1   26   18    4
```

Crosstabs

Make them look good for papers. Remember you need to impress clients and they need to understand what you're doing.

Are civil conflicts more likely in freer states?

```
output <- table(Onset=data.merged$onset,
                Press=data.merged$free.press)
row.names(output) <- c("Onset", "No onset")
kable(output,
      caption="Cvil conflict onset and press freedom",
      col.names=c("Not free", "Partially Free",
                  "Free"))
```

Table 2: Cvil conflict onset and press freedom

	Not free	Partially Free	Free
Onset	1131	824	936
No onset	26	18	4

Saving data

You can save your csv files, too.

- **NEVER** overwrite your original data. You never know when you'll find a mistake. Keep original data safe at all times

```
write_csv(data.merged, file="myNewCWdata.csv")
```

DAY 2: What do we know?

So far we've

- Reading data into R
- Making new variables
- Summarizing data
- Summary statistics and tables

We're going to do some of the same things today, but with a focus on visualizing data

Packages

As with anytime we work with data, we'll want

```r
library(dplyr)
library(readr)
library(tidyr)
```

Today we'll add in

```r
library(stringr)
library(ggplot2)
```

Setup

The `ggplot2` package is another component of the tidyverse, it opens the door to a number of plots.

- We'll start by looking at some COVID data in 2021. Our goal will be to summarize deaths (continuous) by
- State (unit of observation)
- Trump vote share (continuous variable)
- State income levels (discrete, ordered)
- Region (discrete, unordered)

Setup

Data from cdc.gov

```r
covid <- read_csv("covid_data.csv", show_col_types = FALSE)
head(covid)
```

```
## # A tibble: 6 x 15
##   submission_date state tot_cases conf_cases
##   <chr>           <chr>     <dbl>      <dbl>
## 1 12/01/2021      ND       163565     135705
## 2 09/01/2021      ND       118491     107475
## 3 08/08/2021      MD       473969         NA
## 4 05/13/2020      VT          855         NA
## 5 02/02/2021      IL      1130917    1130917
## 6 06/10/2020      VT         1009         NA
## # ... with 11 more variables: prob_cases <dbl>,
## #   new_case <dbl>, pnew_case <dbl>,
## #   tot_death <dbl>, conf_death <dbl>,
## #   prob_death <dbl>, new_death <dbl>,
## #   pnew_death <dbl>, created_at <chr>,
```

Aggregating

There's a lot going on here. We'll need to aggregate to the state-YTD observation

```
covid <- covid %>%
  filter(str_detect(submission_date, "2021")) %>% #new
  select(c("state", "new_death")) %>%
  group_by(state) %>%
  summarize(deaths=sum(new_death, na.rm=T))
```

Line by line this:

Aggregating

There's a lot going on here. We'll need to aggregate to the state-YTD observation

```
covid <- covid %>%
  filter(str_detect(submission_date, "2021")) %>% #new
  select(c("state", "new_death")) %>%
  group_by(state) %>%
  summarize(deaths=sum(new_death, na.rm=T))
```

Line by line this:

1. covid data we're using

Aggregating

There's a lot going on here. We'll need to aggregate to the state-YTD observation

```
covid <- covid %>%
  filter(str_detect(submission_date, "2021")) %>% #new
  select(c("state", "new_death")) %>%
  group_by(state) %>%
  summarize(deaths=sum(new_death, na.rm=T))
```

Line by line this:

1. `covid` data we're using
2. `filter` subset by rows

Aggregating

There's a lot going on here. We'll need to aggregate to the state-YTD observation

```
covid <- covid %>%
  filter(str_detect(submission_date, "2021")) %>% #new
  select(c("state", "new_death")) %>%
  group_by(state) %>%
  summarize(deaths=sum(new_death, na.rm=T))
```

Line by line this:

1. `covid` data we're using
2. `filter` subset by rows
3. `str_detect` does the string/character variable `submission_date` contain `"2021"`

Aggregating

There's a lot going on here. We'll need to aggregate to the state-YTD observation

```
covid <- covid %>%
  filter(str_detect(submission_date, "2021")) %>% #new
  select(c("state", "new_death")) %>%
  group_by(state) %>%
  summarize(deaths=sum(new_death, na.rm=T))
```

Line by line this:

1. `covid` data we're using
2. `filter` subset by rows
3. `str_detect` does the string/character variable `submission_date` contain `"2021"`
4. `select` subset by columns

Aggregating

There's a lot going on here. We'll need to aggregate to the state-YTD observation

```
covid <- covid %>%
  filter(str_detect(submission_date, "2021")) %>% #new
  select(c("state", "new_death")) %>%
  group_by(state) %>%
  summarize(deaths=sum(new_death, na.rm=T))
```

Line by line this:

1. `covid` data we're using
2. `filter` subset by rows
3. `str_detect` does the string/character variable `submission_date` contain `"2021"`
4. `select` subset by columns
5. `group_by` aggregate by group

Aggregating

There's a lot going on here. We'll need to aggregate to the state-YTD observation

```
covid <- covid %>%
  filter(str_detect(submission_date, "2021")) %>% #new
  select(c("state", "new_death")) %>%
  group_by(state) %>%
  summarize(deaths=sum(new_death, na.rm=T))
```

Line by line this:

1. `covid` data we're using
2. `filter` subset by rows
3. `str_detect` does the string/character variable `submission_date` contain `"2021"`
4. `select` subset by columns
5. `group_by` aggregate by group
6. `summarize` aggregate the data by creating a variable `deaths` by summing `new_death` within each state

Did it work

```r
head(covid)
```

```
## # A tibble: 6 x 2
##   state deaths
##   <chr>  <dbl>
## 1 AK       585
## 2 AL      9490
## 3 AR      5192
## 4 AS         0
## 5 AZ     15365
## 6 CA     50473
```

What's next

To get to deaths per 1000 people we need a population value census.gov provides such data.

```
state.pop <- read_csv("state_pop.csv", show_col_types = FAl
head(state.pop)
```

```
## # A tibble: 6 x 5
##   GEO_ID     NAME   POP_BASE2020  POP_2020  POP_2021
##   <chr>      <chr>  <chr>         <chr>     <chr>
## 1 id         Geogr~ Estimates Ba~ Populati~ Populat~
## 2 0100000US  Unite~ 331449281     331501080 3318937~
## 3 0200000US1 North~ 57609148      57525633  57159838
## 4 0200000US2 Midwe~ 68985454      68935174  68841444
## 5 0200000US3 South~ 126266107     126409007 1272253~
## 6 0200000US4 West ~ 78588572      78631266  78667134
```

Now because the census hates us, they have double variable names and no state abbreviations

Clean up

```
state.info <- read_csv("state_info.csv",
                        show_col_types = FALSE)
state.pop <- state.pop %>%
  slice(-1) %>% #slice let's you select rows by number
  select(c("NAME", "POP_2021")) %>%
  mutate(POP_2021 = as.numeric(POP_2021)) %>%
  merge(state.abrv, by.x="NAME", by.y="name", all.y=TRUE)
```

- `state.abb` and `state.name` are built in data that help us out with US states
- How many rows and what columns do we have now?

results

```r
head(state.pop)
```

```
##         NAME POP_2021 state region
## 1    Alabama  5039877    AL  South
## 2     Alaska   732673    AK   West
## 3    Arizona  7276316    AZ   West
## 4   Arkansas  3025891    AR  South
## 5 California 39237836    CA   West
## 6   Colorado  5812069    CO   West
##             division
## 1 East South Central
## 2            Pacific
## 3           Mountain
## 4 West South Central
## 5            Pacific
## 6           Mountain
```

```r
dim(state.pop)
```

```
## [1] 51  5
```

Merge

```
covid <- merge(covid, state.pop,  by="state")
head(covid)
```

```
##   state deaths       NAME POP_2021 region
## 1    AK    585     Alaska   732673   West
## 2    AL   9490    Alabama  5039877  South
## 3    AR   5192   Arkansas  3025891  South
## 4    AZ  15365    Arizona  7276316   West
## 5    CA  50473 California 39237836   West
## 6    CO   5457   Colorado  5812069   West
##             division
## 1            Pacific
## 2 East South Central
## 3 West South Central
## 4           Mountain
## 5            Pacific
## 6           Mountain
```

By population

How do we make deaths per 1000 people then? `mutate`? `summarize`? something else?

By population

How do we make deaths per 1000 people then? `mutate`? `summarize`? something else?

```
covid <- covid %>%
  mutate(pop1000= POP_2021/1000,
         deaths.per1000 = deaths/pop1000)
```

Single variable visualization

Let's time out for now and look at what we have.

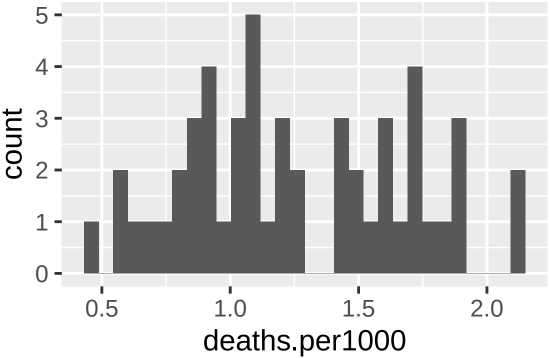
We can look at single continuous and their distribution using a **histogram**

- A histogram creates bins and sorts the data into those bins to give us a graphical representation of the sample
- This is really only informative for continuous or count variables. For categorical variables we would look at a bar chart.

Hisograms

To create a histogram of the population variable we would write

```
ggplot(covid)+
  geom_histogram(aes(x= deaths.per1000))
```



What's happening here

Let's do this piece by piece

- `ggplot(covid)` initializes a plot and tells `ggplot` we're using the `covid` data frame

What's happening here

Let's do this piece by piece

- `ggplot(covid)` initializes a plot and tells `ggplot` we're using the `covid` data frame
- `geom_histogram` We want a histogram
- `aes` Stands for "aesthetic." Different plots have different aesthetics. Because histograms only use one variable and it's along the x-axis, we specify it as `x=deaths.per1000`.

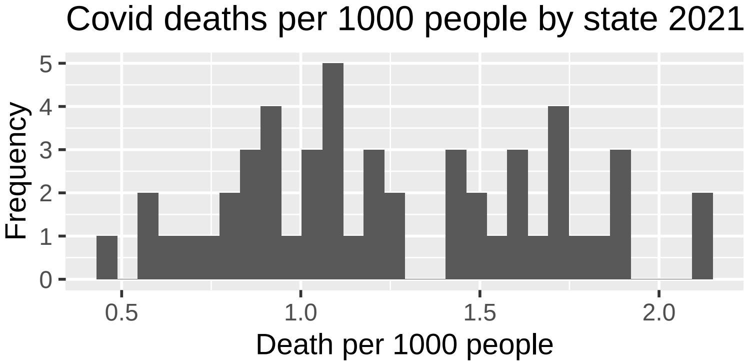
Making it better

Now if this is just for your own exploratory use then we can stop. But if it's for a problem set or report (or a real client one day), then you'll want to make it more informative

```
ggplot(covid)+
  geom_histogram(aes(x= deaths.per1000))+
  xlab("Death per 1000 people")+
  ylab("Frequency")+
  ggtitle("Covid deaths per 1000 people by state 2021")
```

Making it better

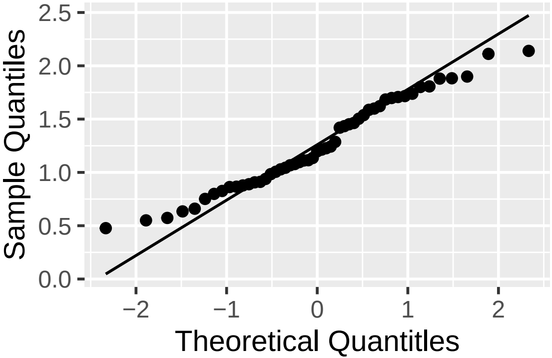
Now if this is just for your own exploratory use then we can stop. But if it's for a problem set or report (or a real client one day), then you'll want to make it more informative



Detecting a normal DGP

We used QQ plots and lines to diagnose if data are consistent with a Normal DGP: here's how

```
ggplot(covid)+
  geom_qq(aes(sample= deaths.per1000))+ #different aes
  geom_qq_line(aes(sample= deaths.per1000))+
  xlab("Theoretical Quantitles")+
  ylab("Sample Quantiles")
```



Let's build

Let's take a look at state income, maybe richer states will have fewer deaths (data from bea.gov)

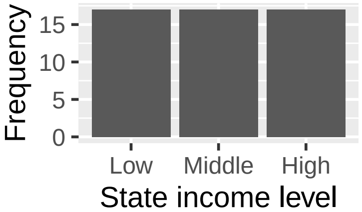
```
income <- read_csv("income_groups.csv",
                   show_col_types = FALSE)
head(income)
```

```
## # A tibble: 6 x 2
##   state income.group
##   <chr> <chr>
## 1 AK    High
## 2 AL    Low
## 3 AR    Low
## 4 AZ    Low
## 5 CA    High
## 6 CO    High
```

```
covid <- merge(covid, income, by="state") %>%
  mutate(income.group=factor(income.group,
                             levels=c("Low", "Middle", "High")))
```

Bar graphs (discrete single variable)

```
ggplot(covid)+
  geom_bar(aes(x= income.group))+
  xlab("State income level")+
  ylab("Frequency")
```



A discrete alternative to the histogram is a classic bar graph

This tells us something about how these are measured... in thirds. Exploratory plots can be helpful when documentation isn't clear.

Two way graphs: box plot (continuous by discrete)

One way we can look for trends is to compare a continuous variable across discrete categories. One option here is a box plot by income

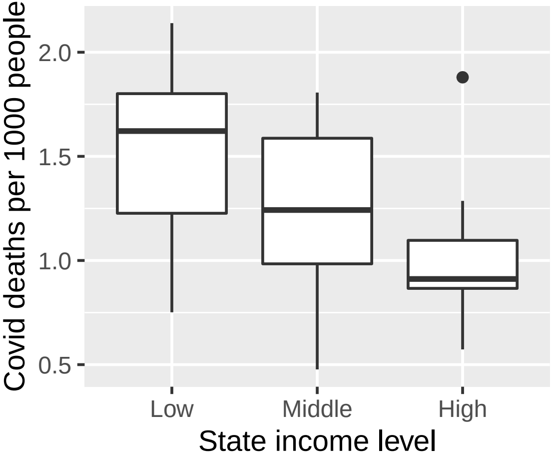
```
ggplot(covid)+
  geom_boxplot(aes(x = income.group, y = deaths.per1000))+
  ylab("Covid deaths per 1000 people")+
  xlab("State income level")
```

Two way graphs: box plot (continuous by discrete)

One way we can look for trends is to compare a continuous variable across discrete categories. One option here is a box plot by income

Two way graphs: box plot (continuous by discrete)

One way we can look for trends is to compare a continuous variable across discrete categories. One option here is a box plot by income



Two way graphs: box plot (continuous by discrete)

What goes into a box plot?

- Center line:
- Top of the box:
- Bottom of the box:
- Top “whisker”:
- Bottom “whisker”:
- Other points:

Two way graphs: box plot (continuous by discrete)

What goes into a box plot?

- Center line: Median
- Top of the box:
- Bottom of the box:
- Top "whisker":
- Bottom "whisker":
- Other points:

Two way graphs: box plot (continuous by discrete)

What goes into a box plot?

- Center line: Median
- Top of the box: 75th percentile
- Bottom of the box: 25th percentile
- Top "whisker":
- Bottom "whisker":
- Other points:

Two way graphs: box plot (continuous by discrete)

What goes into a box plot?

- Center line: Median
- Top of the box: 75th percentile
- Bottom of the box: 25th percentile
- Top "whisker": Maximum observation that is **not** greater than 1.5 the IQR (height of the box)
- Bottom "whisker": Minimum observation that is **not** less than 1.5 the IQR (height of the box)
- Other points:

Two way graphs: box plot (continuous by discrete)

What goes into a box plot?

- Center line: Median
- Top of the box: 75th percentile
- Bottom of the box: 25th percentile
- Top "whisker": Maximum observation that is **not** greater than 1.5 the IQR (height of the box)
- Bottom "whisker": Minimum observation that is **not** less than 1.5 the IQR (height of the box)
- Other points: Outliers (beyond 1.5 IQR from the median)

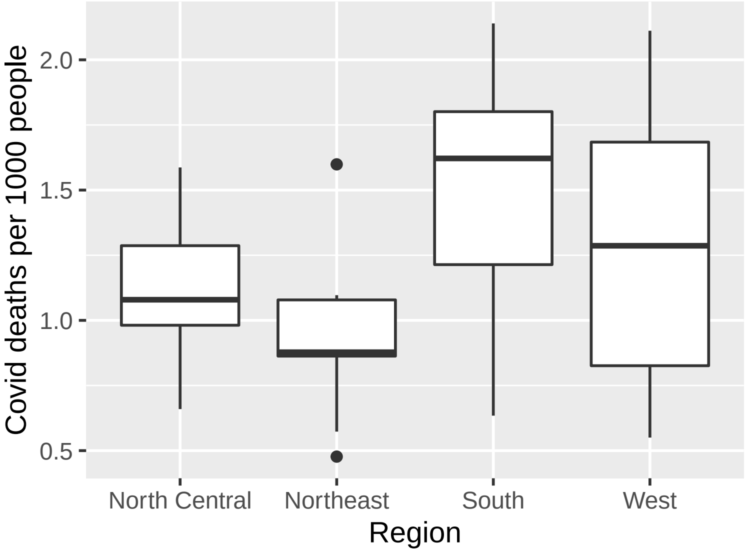
Two way graphs: box plot (continuous by discrete)

One way we can look for trends is to compare a continuous variable across discrete categories. One option here is a box plot by region

```
ggplot(covid)+
  geom_boxplot(aes(x = region, y = deaths.per1000))+
  ylab("Covid deaths per 1000 people")+
  xlab("Region")
```

Two way graphs: box plot (continuous by discrete)

One way we can look for trends is to compare a continuous variable across discrete categories. One option here is a box plot by region



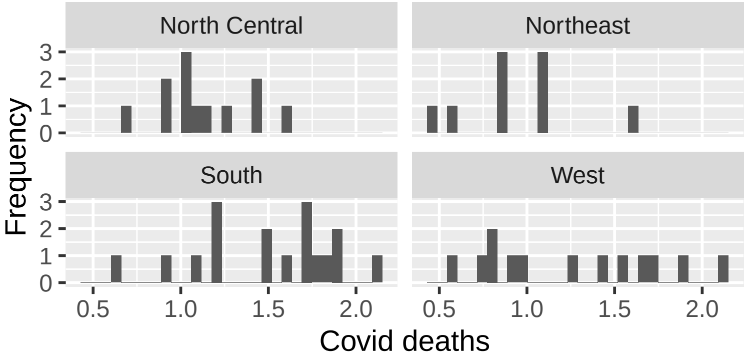
Multiple graphs: facetted plots (discrete by anything)

We can also do sub-group analysis by looking at facets

```
ggplot(covid, aes(x = deaths.per1000)) +
  geom_histogram() +
  facet_wrap("region")+ #note that this is quoted
  xlab("Covid deaths")+
  ylab("Frequency")
```

Multiple graphs: facetted plots (discrete by anything)

We can also do sub-group analysis by looking at facets



Two way graphs: Scatter plot (continuous by continuous)

For two continuous variables, a scatter plot is a very useful vizualization and is a great place to start any exploratory analysis with continuous variables. Let's go ahead and add in Trump vote share

Two way graphs: Scatter plot (continuous by continuous)

For two continuous variables, a scatter plot is a very useful vizualization and is a great place to start any exploratory analysis with continuous variables. Let's go ahead and add in Trump vote share

```r
vote.share <- read_csv("1976-2020-president.csv",
                       show_col_types = FALSE)
head(vote.share)
```

```
## # A tibble: 6 x 15
##    year state   state_po state_fips state_cen state_ic
##   <dbl> <chr>   <chr>         <dbl>     <dbl>    <dbl>
## 1  1976 ALABAMA AL                1        63       41
## 2  1976 ALABAMA AL                1        63       41
## 3  1976 ALABAMA AL                1        63       41
## 4  1976 ALABAMA AL                1        63       41
## 5  1976 ALABAMA AL                1        63       41
## 6  1976 ALABAMA AL                1        63       41
## # ... with 9 more variables: office <chr>,
```

Take what we need

```
vote.share <- vote.share %>%
  filter(office=="US PRESIDENT" & year==2020 &
           candidate %in% c("BIDEN, JOSEPH R. JR" ,
                            "TRUMP, DONALD J.")) %>%
  group_by(state_po) %>%
  summarize(vote.share=candidatevotes/sum(candidatevotes),
            candidate=candidate)%>%
  filter(candidate=="TRUMP, DONALD J.") %>%
  select(c("state_po","vote.share"))
covid <- merge(covid, vote.share,
               by.x="state", by.y="state_po")
```

Who wants to talk me through this one?

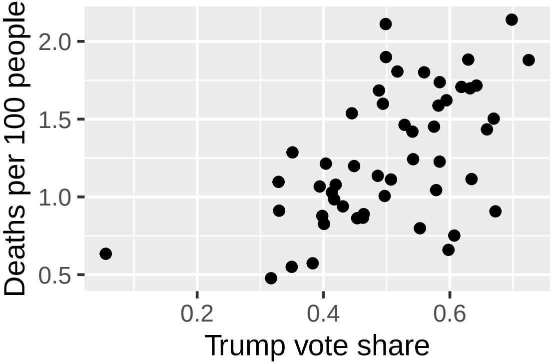
Two way graphs: Scatter plot (continuous by continuous)

For two continuous variables, a scatter plot is a very useful vizualization and is a great place to start any exploratory analysis with continuous variables. Let's go ahead and add in Trump vote share

```
ggplot(covid)+
  geom_point(aes(x = vote.share, y = deaths.per1000))+
  ylab("Deaths per 100 people")+
  xlab("Trump vote share")
```

Two way graphs: Scatter plot (continuous by continuous)

For two continuous variables, a scatter plot is a very useful vizualization and is a great place to start any exploratory analysis with continuous variables. Let's go ahead and add in Trump vote share



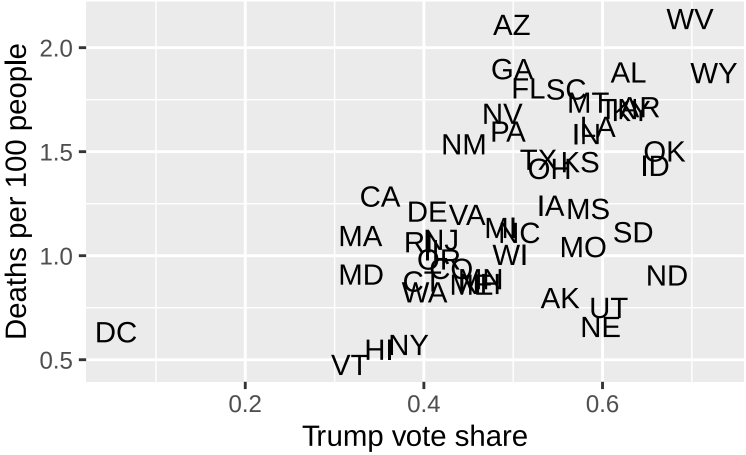
Two way graphs: Scatter plot (continuous by continuous)

This one way to present a possible trend, but people always want to know which dots are what. One neat trick is to use labels instead of points

```
ggplot(covid)+
  geom_text(aes(x = vote.share, y = deaths.per1000,
                label=state))+
  ylab("Deaths per 100 people")+
  xlab("Trump vote share")
```

Two way graphs: Scatter plot (continuous by continuous)

This one way to present a possible trend, but people always want to know which dots are what. One neat trick is to use labels instead of points



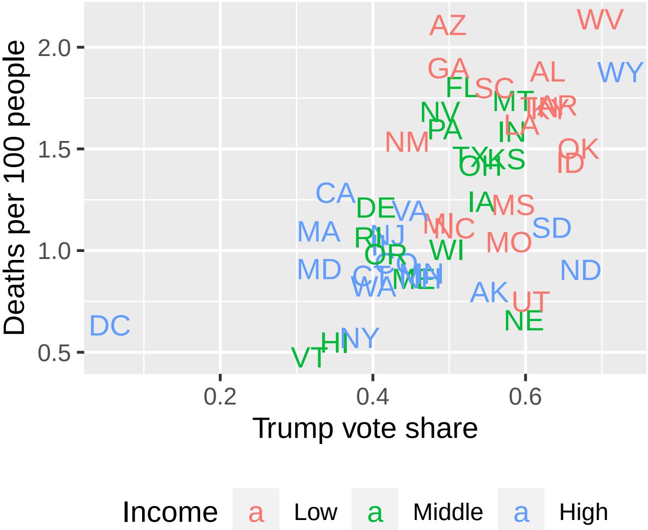
Using colors, shape, and size to tell a story

We can use the size, color, or shape of a point to include more information in a plot. Let's go back to the scatter plot and say we wanted to include information about region. We could facet or

```
ggplot(covid)+
  geom_text(aes(x = vote.share, y = deaths.per1000,
                label=state, color=income.group))+
  ylab("Deaths per 100 people")+
  xlab("Trump vote share")+
  guides(color=guide_legend(title="Income"))+
  theme(legend.position = "bottom")
```

Using colors, shape, and size to tell a story

We can use the size, color, or shape of a point to include more information in a plot. Let's go back to the scatter plot and say we wanted to include information about region. We could facet or



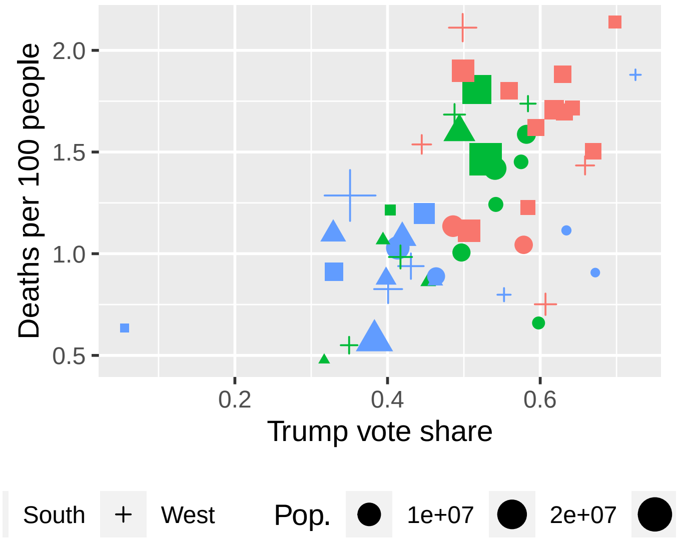
Using colors and size to tell a story

You could keep going with this, but there are problems with going too far for example

```
ggplot(covid)+
  geom_point(aes(x = vote.share, y = deaths.per1000,
                 color=income.group,
                 shape=region, size=POP_2021))+
  ylab("Deaths per 100 people")+
  xlab("Trump vote share")+
  guides(color=guide_legend(title="Income"),
         size=guide_legend(title="Pop."))+
  theme(legend.position = "bottom")
```

Using colors and size to tell a story

This is a mess and a half. Less is often more.



Saving plots

Problem set 0 will walk you through how to put graphs into your problem sets and write ups.

Saving plots

Problem set 0 will walk you through how to put graphs into your problem sets and write ups.

But sometimes you want to save a plot you've made to put into another paper (say a word document) or to send directly to someone to make a point or to put on your fridge

Here's how you would do that that

```
plot.out <- ggplot(covid)+
  geom_text(aes(x = vote.share, y = deaths.per1000,
              label=state, color=income.group))+
  ylab("Deaths per 100 people")+
  xlab("Trump vote share")+
  guides(color=guide_legend(title="Income"))+
  theme(legend.position = "bottom")
ggsave(plot.out, filename = "covid_by_state.png",
      height=3, width=4)
# you'll need to trial and error the
# height and width for your own use
```