Introduction to Data Analysis and Reporting with R

J. Alexander Branham

June 2017

Course Information

- We'll cover tools that should be helpful in nearly any analysis

Course Information

- We'll cover tools that should be helpful in nearly any analysis
  - Graphing, data manipulation, etc

Course Information

- We'll cover tools that should be helpful in nearly any analysis
    - Graphing, data manipulation, etc
- We won't cover specialized, specific tools. But you should get a good enough understanding of how R works to be able to teach yourself these

Outline

1. What is R?

Outline

1. What is R?
2. Graphics

Outline

1. What is R?
2. Graphics
3. Basic R

Outline

1. What is R?
2. Graphics
3. Basic R
4. Data manipulation

Outline

1. What is R?
2. Graphics
3. Basic R
4. Data manipulation
5. Reporting (time permitting)

What is R?

What is R?

- This is a course about R… *mais qu'est-ce que c'est*?

What is R?

- This is a course about R… *mais qu'est-ce que c'est*?
- "R is a language and environment for statistical computing and graphics"

What is R?

- This is a course about R... *mais qu'est-ce que c'est*?
- "R is a language and environment for statistical computing and graphics"
- Derived from S, designed at Bell Laboratories

What is R?

- This is a course about R... *mais qu'est-ce que c'est*?
- "R is a language and environment for statistical computing and graphics"
- Derived from S, designed at Bell Laboratories
    - S first appeared in 1976!

What is R?

- This is a course about R… *mais qu’est-ce que c’est*?
- “R is a language and environment for statistical computing and graphics”
- Derived from S, designed at Bell Laboratories
    - S first appeared in 1976!
- *R is a language* … so be prepared for it to hurt a bit to learn!

Advantages of R

Advantages of R

- Free

Advantages of R

- Free
- Open-source

Advantages of R

- Free
- Open-source
- Available on nearly every platform

Advantages of R

- Free
- Open-source
- Available on nearly every platform
- Extensible via packages — CRAN has over 10,000

Advantages of R

- Free
- Open-source
- Available on nearly every platform
- Extensible via packages — CRAN has over 10,000
- Great community

Running code

- How to use this R thing?

- How to use this R thing?
- If you have R and Rstudio installed, open Rstudio.

- How to use this R thing?
- If you have R and Rstudio installed, open Rstudio.
- You should see three panes.

- How to use this R thing?
- If you have R and Rstudio installed, open Rstudio.
- You should see three panes.
- We'll focus for now on the console, which is on the left and should look something like this:

```
R version 3.4.0 (2017-06-15) -- "You Stupid Darkness"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

[ ... ]

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

R is a big giant calculator

- R can do math
- Really, really fancy math
- Try typing `3 + 3` in the console
- After pressing enter, R will return `6`
- R understands the order of operations
    - `3 + 3 * 9` is different from `(3 + 3) * 9`

A quiz

- Time for a quiz!

A quiz

- Time for a quiz!
- What's 7 times 149?

A quiz

- Time for a quiz!
- What’s 7 times 149?
- What’s the square root of the previous answer?

A quiz

- Time for a quiz!
- What's 7 times 149?
- What's the square root of the previous answer?
- Tip: You can hit the up arrow to get whatever you entered last

```
7 * 149

[1] 1043

(7 * 149) ^ (1 / 2)

[1] 32.29551
```

- At this point, please install a few packages. You'll need an internet connection.
- `install.packages(c("tidyverse", "gapminder"))`
    - If you have already installed some packages, make sure they're up-to-date:
        - `update.packages()`
- Tip: just type `ins` then hit TAB for tab-completion
- Don't worry about what is going on here, I'll explain it later.
- Depending on your exact setup, R may ask you a few questions about using a personal library. Do so.
- If you get an error, make sure you can access the internet (`https://cloud.r-project.org` in particular)

R scripts

- While those packages are installing, let's go ahead and open up an R script.
- Allows you to save code so it doesn't disappear into the ether
- If using Rstudio, File, new file, R script (or Ctrl+shift+n)
- Tip: can send a line from R script to console for evaluation using ctrl+enter
- Strongly recommend that you type into a script and use a keyboard shortcut to evaluate code
    - Easier to edit & rerun
    - Allows you to save code
    - You may make comments

```
## This adds 3 + 3
3 + 3
3 * 2 # same
```

Graphics in R

- We need some data to work with

- We need some data to work with
- We're going to use some data that comes with the `gapminder` package you just installed

- We need some data to work with
- We're going to use some data that comes with the `gapminder` package you just installed
- To access the data, you need to load it into memory:

```
library(gapminder)
```

- `gapminder` is a data.frame

- `gapminder` is a data.frame
- Can get a sense of what it looks like with some functions

Exploring our data

- `gapminder` is a data.frame
- Can get a sense of what it looks like with some functions
- Let's get a sense of what `gapminder` has:

```
View(gapminder)
```

- `gapminder` is a data.frame
- Can get a sense of what it looks like with some functions
- Let's get a sense of what `gapminder` has:

```
View(gapminder)
```

```
head(gapminder)
```

```
      country continent year lifeExp      pop gdpPercap
1 Afghanistan      Asia 1952  28.801  8425333  779.4453
2 Afghanistan      Asia 1957  30.332  9240934  820.8530
3 Afghanistan      Asia 1962  31.997 10267083  853.1007
4 Afghanistan      Asia 1967  34.020 11537966  836.1971
```

Descriptive statistics

- R has lots of built-in functions for getting a sense of the data.
- Try running `summary(gapminder)`
- What’s the average life expectancy?

Descriptive statistics

- R has lots of built-in functions for getting a sense of the data.
- Try running `summary(gapminder)`
- What's the average life expectancy?

```
summary(gapminder)
```

```
       country        continent          year         lifeExp
 Afghanistan:  12   Africa  :624   Min.   :1952   Min.   :23.60
 Albania    :  12   Americas:300   1st Qu.:1966   1st Qu.:48.20
 Algeria    :  12   Asia    :396   Median :1980   Median :60.71
 Angola     :  12   Europe  :360   Mean   :1980   Mean   :59.47
 Argentina  :  12   Oceania : 24   3rd Qu.:1993   3rd Qu.:70.85
 Australia  :  12                  Max.   :2007   Max.   :82.60
 (Other)    :1632
```

- Let's start making graphs
- This is the fun part!
- We're going to rely on the 'ggplot2' package, which we installed earlier (as a part of the tidyverse package)
- "The Grammar of Graphics"
- load it up with

```
library(ggplot2)
```

Our question

What's the relationship between wealth (gdp) and average life expectancy?

Our question

What's the relationship between wealth (gdp) and average life expectancy?

- Scatterplot is a good way to get started looking at data!

- Use the `ggplot()` function to start a plot.
- The first *argument* is to tell it the *data*
- Tip: use `?ggplot` to look at the help page, where you can see the names of the arguments

- Use the **`ggplot()`** function to start a plot.
- The first *argument* is to tell it the *data*
- Tip: use **`?ggplot`** to look at the help page, where you can see the names of the arguments

```
ggplot(data = gapminder) # Please use gapminder data
```

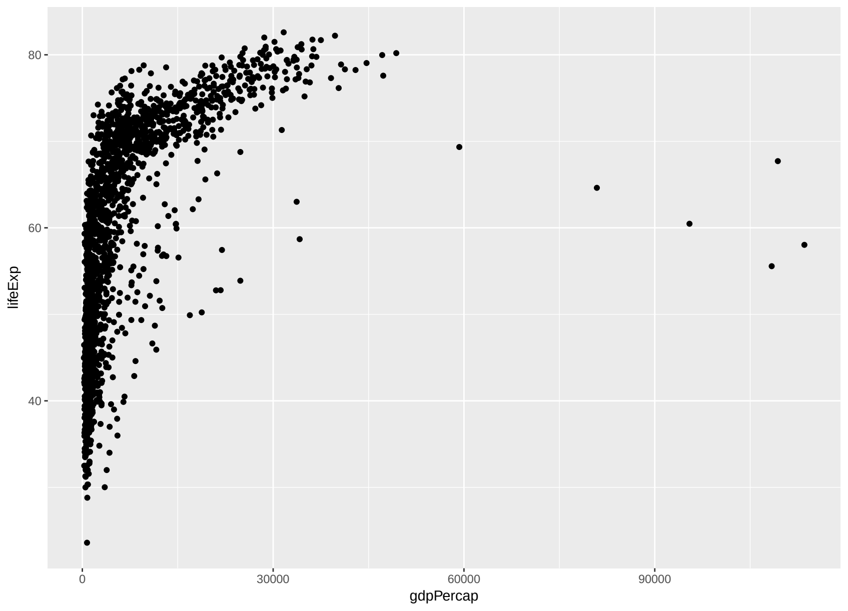
`geom_point`

- `ggplot()` by itself is pretty useless, it just starts a plot
- We then have to tell `ggplot` what to draw!
- Tip: `?geom_point`

geom_point

- `ggplot()` by itself is pretty useless, it just starts a plot
- We then have to tell `ggplot` what to draw!
- Tip: `?geom_point`

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = gdpPercap, # Put gdp on x axis
                           y = lifeExp))  # Put lifeExp on y
```


80
60
40
lifeExp
0
30000
60000
90000
gdpPercap

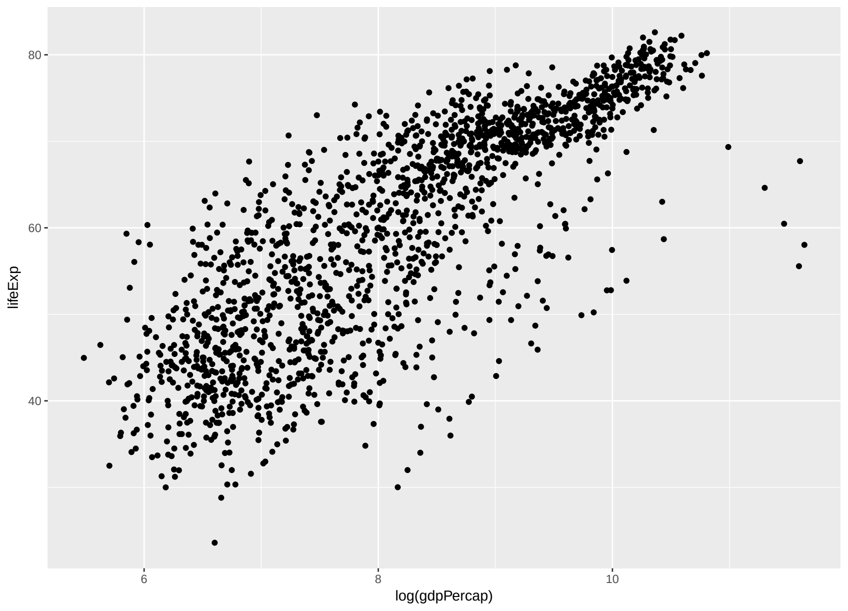
Fix that x axis!

- Is there a better way to show this relationship?

Fix that x axis!

- Is there a better way to show this relationship?

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = log(gdpPercap), # Log x-axis
                           y = lifeExp))
```


80
60
40
lifeExp
6
8
10
log(gdpPercap)

Aesthetics

- `ggplot()` creates a coordinate system

Aesthetics

- `ggplot()` creates a coordinate system
- You can then add one or more layers to this to create a plot

Aesthetics

- `ggplot()` creates a coordinate system
- You can then add one or more layers to this to create a plot
- We just added the `geom_point()` layer, which used the `x` and `y` aesthetics (`aes`) to add a layer of points to our plot

Aesthetics

- `ggplot()` creates a coordinate system
- You can then add one or more layers to this to create a plot
- We just added the `geom_point()` layer, which used the `x` and `y` aesthetics (`aes`) to add a layer of points to our plot
- We can add more information to the aesthetics to convey more information like color, shape, and size.

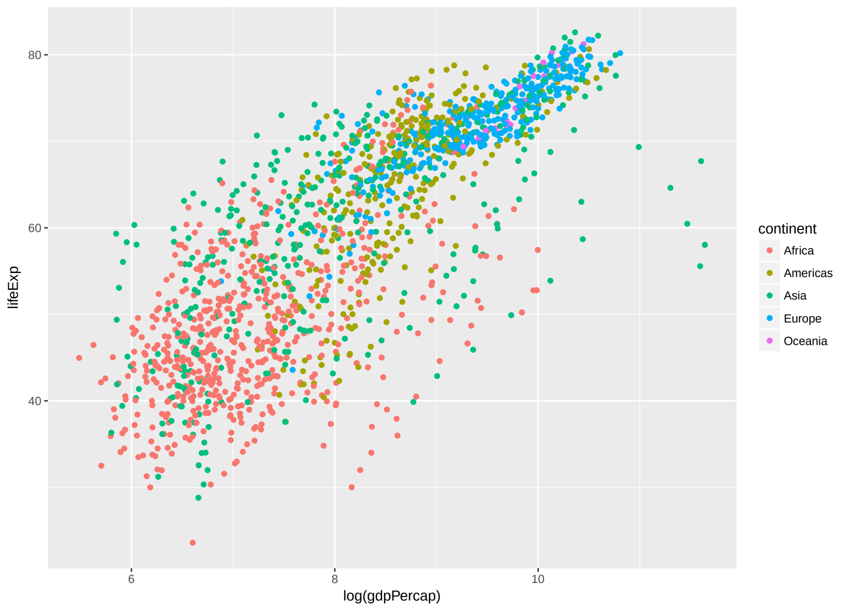
Aesthetics

- `ggplot()` creates a coordinate system
- You can then add one or more layers to this to create a plot
- We just added the `geom_point()` layer, which used the `x` and `y` aesthetics (`aes`) to add a layer of points to our plot
- We can add more information to the aesthetics to convey more information like color, shape, and size.
- Example: What if we want to convey info about relationship between wealth and life expectancy by continent?

Aesthetics

- `ggplot()` creates a coordinate system
- You can then add one or more layers to this to create a plot
- We just added the `geom_point()` layer, which used the `x` and `y` aesthetics (`aes`) to add a layer of points to our plot
- We can add more information to the aesthetics to convey more information like color, shape, and size.
- Example: What if we want to convey info about relationship between wealth and life expectancy by continent?
- One solution: add color by continent

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = log(gdpPercap),
                           y = lifeExp,
                           ## colour for the Brits
                           color = continent))
```

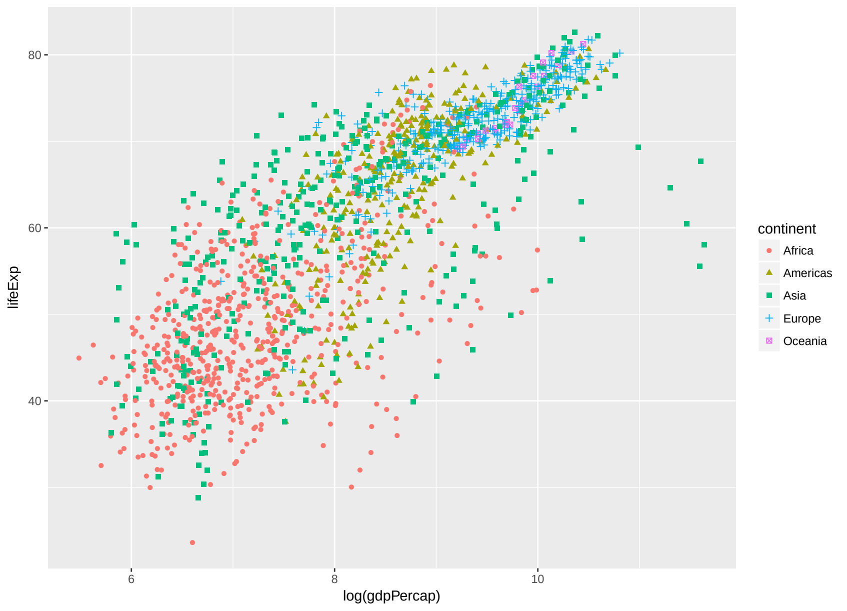

80
60
40
lifeExp
6
8
10
log(gdpPercap)
continent
Africa
Americas
Asia
Europe
Oceania

Multiple aesthetics - color & shape

- Of course, some people are colorblind, and others don't print things in color, so may be nice to use something like shape in addition:

- Of course, some people are colorblind, and others don't print things in color, so may be nice to use something like shape in addition:

```
ggplot(gapminder) +
  geom_point(aes(x = log(gdpPercap),
                 y = lifeExp,
                 color = continent,
                 shape = continent))
```


80
60
40
lifeExp
6
8
10
log(gdpPercap)
continent
Africa
Americas
Asia
Europe
Oceania

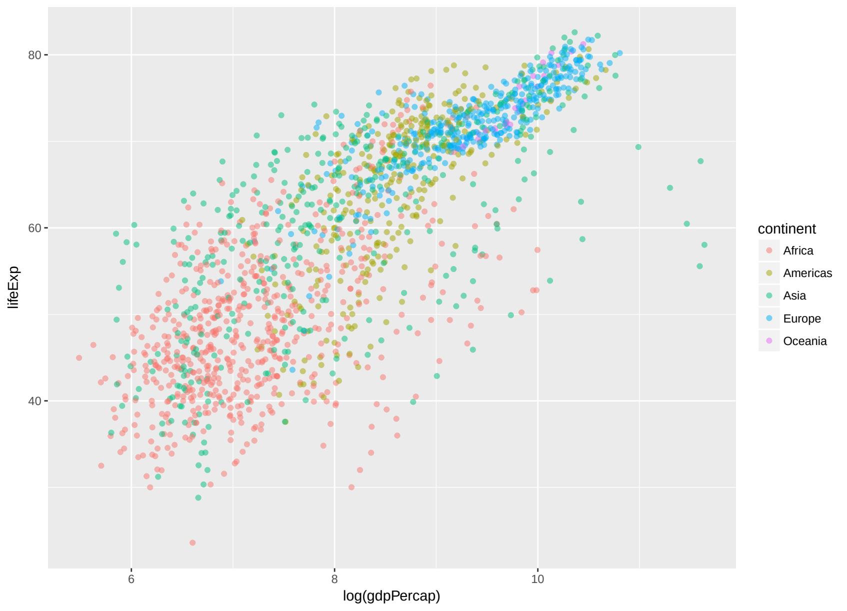
- There are more aesthetic mappings
- Try `size`, and `alpha` (transparency) for yourself

- There are more aesthetic mappings
- Try `size`, and `alpha` (transparency) for yourself

- You can set aesthetics directly by mapping the aesthetic to a value *outside* the call to `aes()`
- For example, we may want to make the dots slightly transparent to avoid overplotting

Aesthetics not mapped to variable

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = log(gdpPercap),
                           y = lifeExp,
                           color = continent),
             alpha = 0.5)
```

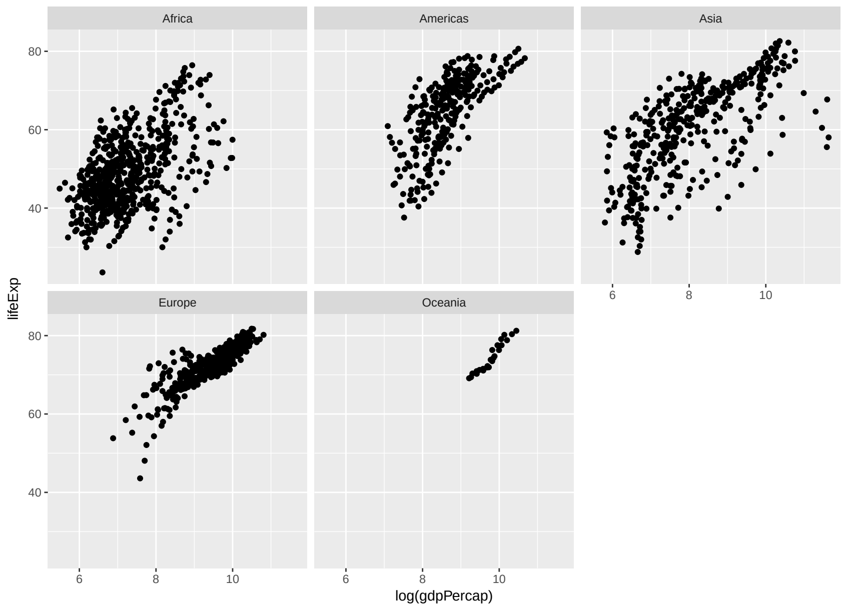

80
60
40
lifeExp
6
8
10
log(gdpPercap)
continent
Africa
Americas
Asia
Europe
Oceania

Facets

- So we can use aesthetics to add variables to our graph like `color`.
- We might also want to add variables by splitting up the graph based on values of another variables — e.g. subfigures
- If we want to use just one variable, use `facet_wrap()`

- So we can use aesthetics to add variables to our graph like `color`.
- We might also want to add variables by splitting up the graph based on values of another variables — e.g. subfigures
- If we want to use just one variable, use `facet_wrap()`

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = log(gdpPercap),
                           y = lifeExp)) +
  facet_wrap( ~ continent, nrow = 2)
```



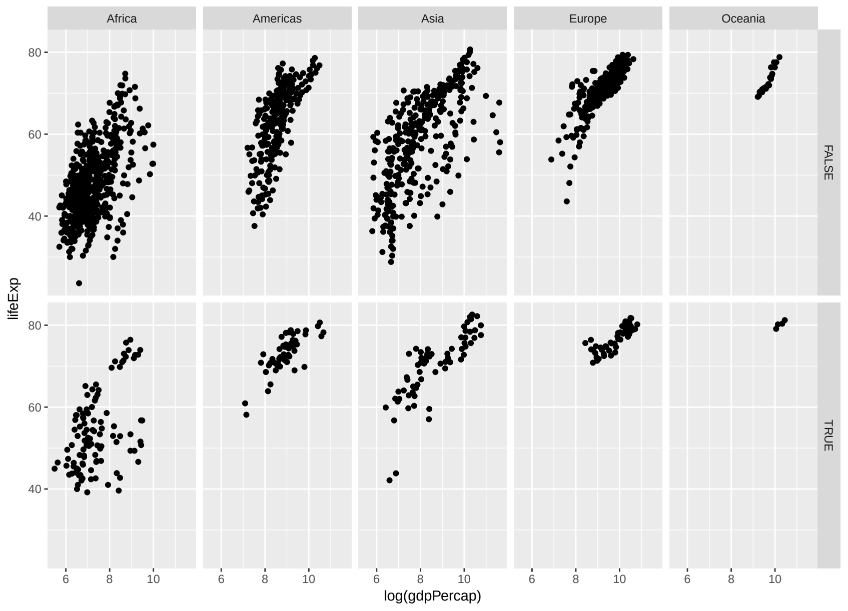
Africa
Americas
Asia
Europe
Oceania
lifeExp
80
60
40
6
8
10
log(gdpPercap)

- ggplot can facet with two variables with one by row and the other by column
- Use `facet_grid(row ~ column)` to do so
- Our `gapminder` data aren't very well suited for this, but you could do something like:

Facets with two variables

- ggplot can facet with two variables with one by row and the other by column
- Use `facet_grid(row ~ column)` to do so
- Our `gapminder` data aren't very well suited for this, but you could do something like:

```
ggplot(data = gapminder) +
  geom_point(mapping = aes(x = log(gdpPercap),
                           y = lifeExp)) +
  ## year >= 2000 will be TRUE or FALSE;
  ## we'll learn more about logical statements later on:
  facet_grid(year >= 2000 ~ continent)
```


Africa
Americas
Asia
Europe
Oceania
FALSE
TRUE
lifeExp
80
60
40
6
8
10
log(gdpPercap)

GGPLOT

- Review of what we've learned so far:
    - `ggplot()` creates a blank coordinate system

- Review of what we've learned so far:
    - `ggplot()` creates a blank coordinate system

- `aes()` helps us map variables to visual properties (x/y location, color, shape, etc)

- Review of what we've learned so far:
    - `ggplot()` creates a blank coordinate system

- `aes()` helps us map variables to visual properties (x/y location, color, shape, etc)

- `facet_wrap()` and `facet_grid()` help us convey variables via subfigures

- Review of what we've learned so far:
    - `ggplot()` creates a blank coordinate system
- `aes()` helps us map variables to visual properties (x/y location, color, shape, etc)
- `facet_wrap()` and `facet_grid()` help us convey variables via subfigures
- But what about plots other than the scatterplot?

- A `geom` (geometrical object) is `ggplot`'s way of representing data

- A `geom` (geometrical object) is `ggplot`'s way of representing data
- We've been using `geom_point()` to represent data as points, e.g. a scatterplot
- A `geom` is (usually) the thing we call the plot - line plots, bar plots, boxplots, etc

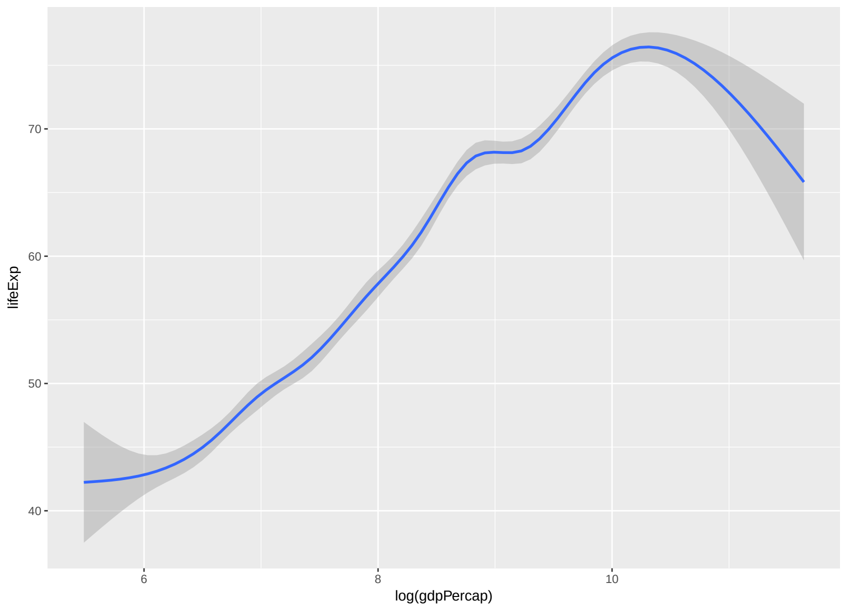
- A `geom` (geometrical object) is `ggplot`'s way of representing data
- We've been using `geom_point()` to represent data as points, e.g. a scatterplot
- A `geom` is (usually) the thing we call the plot - line plots, bar plots, boxplots, etc
- Let's plot the same relationship between wealth and life expectancy but using `geom_smooth()` rather than `geom_point()`:

- A `geom` (geometrical object) is `ggplot`'s way of representing data

- We've been using `geom_point()` to represent data as points, e.g. a scatterplot
- A `geom` is (usually) the thing we call the plot - line plots, bar plots, boxplots, etc

- Let's plot the same relationship between wealth and life expectancy but using `geom_smooth()` rather than `geom_point()`:

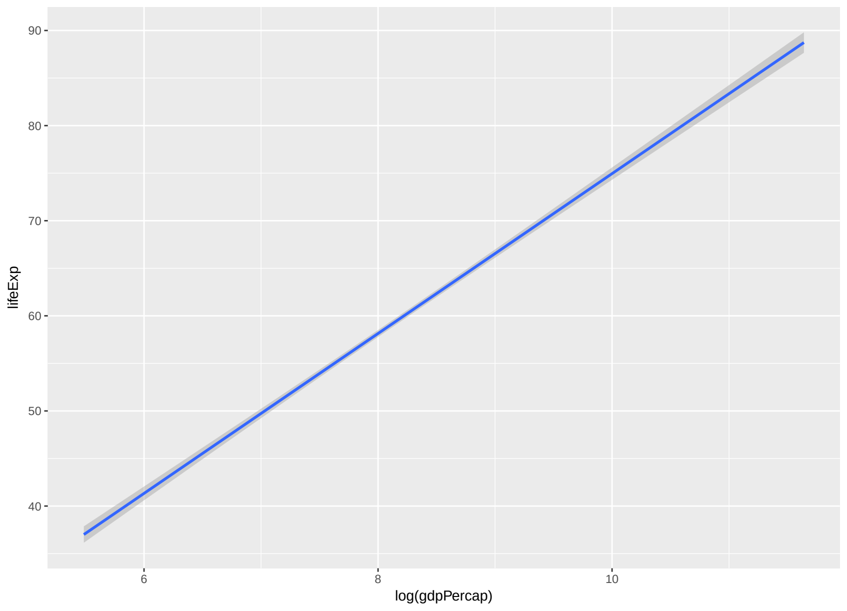
```
ggplot(data = gapminder) +
  geom_smooth(mapping = aes(x = log(gdpPercap),
                            y = lifeExp))
```


70
60
50
40
lifeExp
6
8
10
log(gdpPercap)

- Hey, that last plot looked pretty linear
- We can use OLS instead:

- Hey, that last plot looked pretty linear
- We can use OLS instead:

```
ggplot(data = gapminder) +
  geom_smooth(mapping = aes(x = log(gdpPercap),
                            y = lifeExp),
              method = "lm")
```


90
80
70
60
50
40
lifeExp
6
8
10
log(gdpPercap)

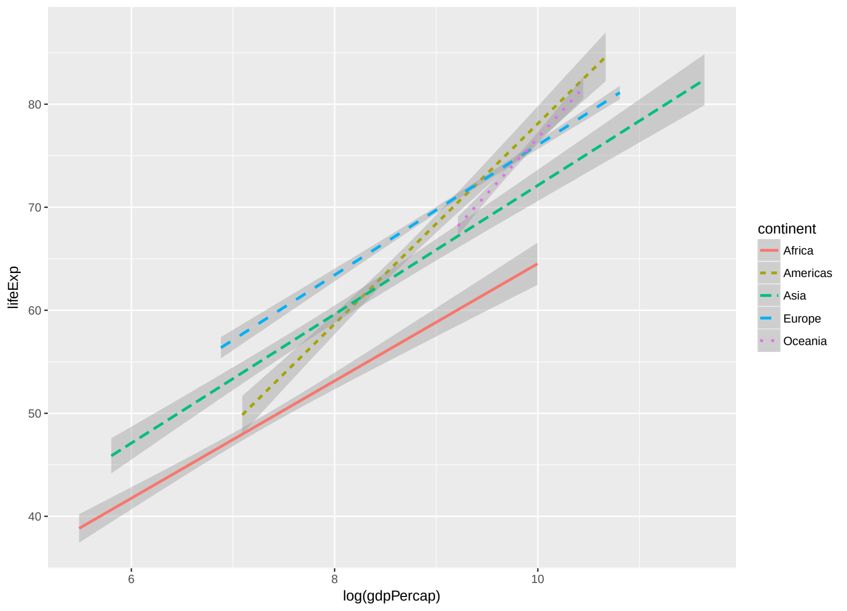
GEOMS AND AESTHETICS

- Note that different aesthetics are available for different geoms

- Note that different aesthetics are available for different geoms
- So while `linetype` didn't really make sense for our scatterplot, it makes total sense for a line:

- Note that different aesthetics are available for different geoms
- So while `linetype` didn't really make sense for our scatterplot, it makes total sense for a line:

```
ggplot(data = gapminder) +
  geom_smooth(mapping = aes(x = log(gdpPercap),
                            y = lifeExp,
                            color = continent,
                            linetype = continent),
            method = "lm")
```



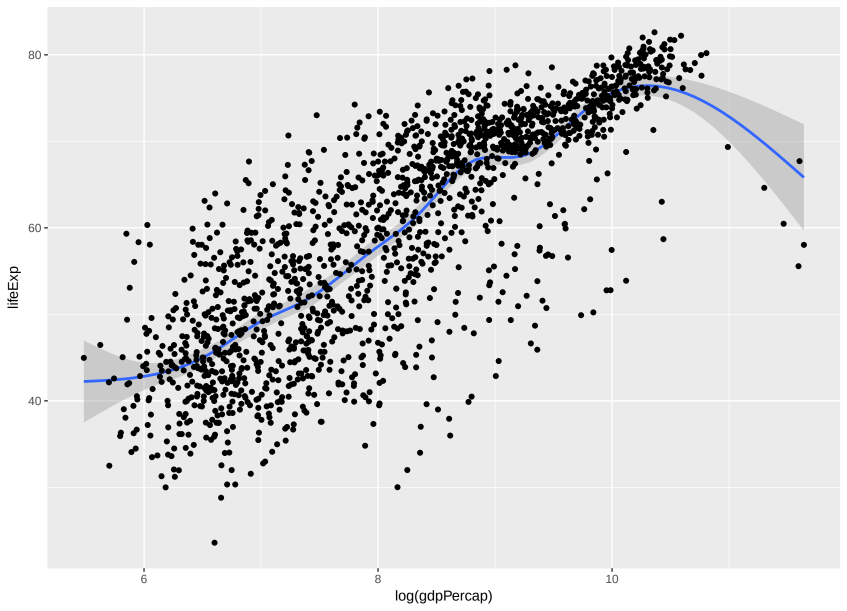
80
70
60
50
40
lifeExp
6
8
10
log(gdpPercap)
continent
Africa
Americas
Asia
Europe
Oceania

MULTIPLE GEOMS

- To add multiple geoms, just add them one after the other:

- To add multiple geoms, just add them one after the other:

```
ggplot(data = gapminder) +
  geom_smooth(mapping = aes(x = log(gdpPercap),
                            y = lifeExp)) +
  geom_point(mapping = aes(x = log(gdpPercap),
                           y = lifeExp))
```


80
60
40
lifeExp
6
8
10
log(gdpPercap)

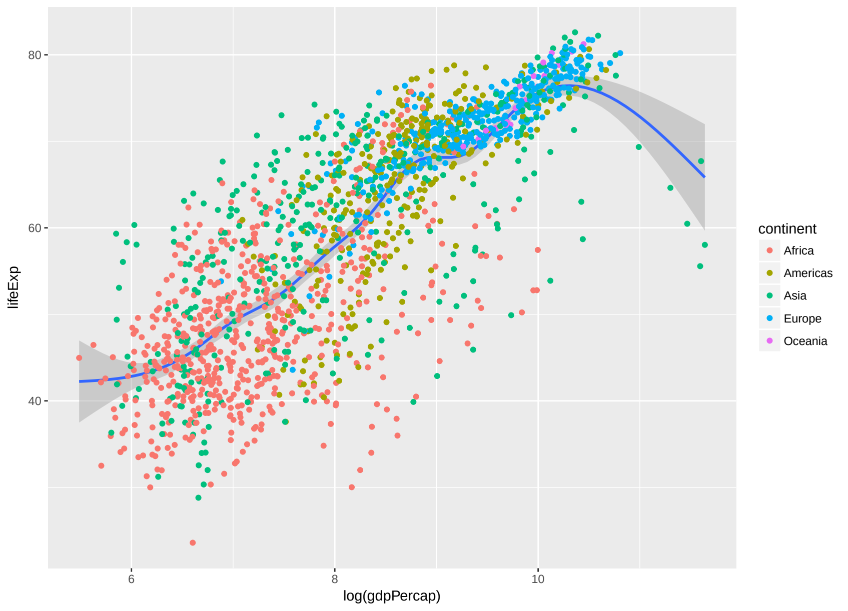
INHERIT AES

- Instead of retyping the `aes` mapping, we can specify a set of defaults in the `ggplot()` call, and overwrite (or add) then in each `geom` call:

INHERIT AES

- Instead of retyping the `aes` mapping, we can specify a set of defaults in the `ggplot()` call, and overwrite (or add) then in each `geom` call:

```
ggplot(data = gapminder,
       mapping = aes(x = log(gdpPercap),
                     y = lifeExp)) +
  geom_smooth() +
  geom_point(mapping = aes(color = continent))
```



80
60
40
lifeExp
6
8
10
log(gdpPercap)
continent
Africa
Americas
Asia
Europe
Oceania

Review

- ggplot2 provides a very flexible way to make high-quality graphics

Review

- ggplot2 provides a very flexible way to make high-quality graphics
- stuff we didn't look at:

- ggplot2 provides a very flexible way to make high-quality graphics
- stuff we didn't look at:
    - Lots of different geoms

Review

- ggplot2 provides a very flexible way to make high-quality graphics
- stuff we didn't look at:
    - Lots of different geoms
    - Changing scales

Review

- ggplot2 provides a very flexible way to make high-quality graphics
- stuff we didn't look at:
    - Lots of different geoms
    - Changing scales
    - Position

Review

- ggplot2 provides a very flexible way to make high-quality graphics
- stuff we didn't look at:
    - Lots of different geoms
    - Changing scales
    - Position
    - How to save to include in your paper (later, I promise!)

Basic R

Basics

- We skipped all of this because plotting is more fun & I wanted to start with something fun

Basics

- We skipped all of this because plotting is more fun & I wanted to start with something fun
- Let’s talk about basic R

Calculator

- Remember R can be a calculator:

```
3 * 3 + 29 ^ 4 + 7
```

```
[1] 707297
```

Calculator

- Remember R can be a calculator:

```
3 * 3 + 29 ^ 4 + 7
```

```
[1] 707297
```

- But R doesn't "remember" the answer to that anywhere

- Remember R can be a calculator:

```
3 * 3 + 29 ^ 4 + 7
```

```
[1] 707297
```

- But R doesn’t “remember” the answer to that anywhere
- You must *assign* the output to an object in order for R to remember it:

- Remember R can be a calculator:

```
3 * 3 + 29 ^ 4 + 7
```

```
[1] 707297
```

- But R doesn't "remember" the answer to that anywhere
- You must *assign* the output to an object in order for R to remember it:

```
x <- 3 * 3 + 29 ^ 4 + 7
my_name <- "Alex Branham"
```

- Remember R can be a calculator:

```
3 * 3 + 29 ^ 4 + 7
```

```
[1] 707297
```

- But R doesn't "remember" the answer to that anywhere
- You must *assign* the output to an object in order for R to remember it:

```
x <- 3 * 3 + 29 ^ 4 + 7
my_name <- "Alex Branham"
```

- Tip: In Rstudio, use alt+- (option+-) to get `<-`

Wait, what?

- Yeah, I just assigned letters to an object

Wait, what?

- Yeah, I just assigned letters to an object
- We can inspect the contents of an object by typing it into the R console:

```
x
```

```
[1] 707297
```

- Yeah, I just assigned letters to an object
- We can inspect the contents of an object by typing it into the R console:

```
x
```

```
[1] 707297
```

- Here, type `my_` then hit tab to have autocompletion

```
my_name
```

```
[1] "Alex Branham"
```

+

- If you forgot the closing `"` — `my_name <- "Alex Branham`
- The R prompt will change from `>` to `+`
- This indicates that R is waiting for you.
- Cancel by mashing `ESC`

R is pedantic

R is pedantic

- You have to be really specific with R:

```
x
```

```
[1] 707297
```

R is pedantic

- You have to be really specific with R:

```
x
```

```
[1] 707297
```

```
X
```

```
Error: object 'X' not found
```

- You have to be really specific with R:

```
x
```

```
[1] 707297
```

```
X
```

```
Error: object 'X' not found
```

```
my_nam
```

```
Error: object 'my_nam' not found
```

Things don't happen magically

```
x
```

```
[1] 707297
```

```
x / 1000
```

```
[1] 707.297
```

```
x
```

```
[1] 707297
```

```
x / 1000
```

```
[1] 707.297
```

```
x
```

```
[1] 707297
```

Missing values

- Missing data is represented by `NA` in R
- R thinks about this as “something that’s there, but whose value we do not know”
- Missingness propagates

Missing values

- Missing data is represented by `NA` in R
- R thinks about this as "something that's there, but whose value we do not know"
- Missingness propagates

```
mean(c(1, 2, NA))
```

```
[1] NA
```

- What will be the result?
- We'll learn more about logical statements in a bit, this asks "Is 3 equal to NA"?

```
3 == NA
NA == NA
```

Missingness quiz answer

```
3 == NA
```

```
[1] NA
```

```
NA == NA
```

```
[1] NA
```

Functions

- Functions in R can take zero or more arguments

```
function(arg1 = object1, arg2 = object2, arg3 = object3)
```

- Functions in R can take zero or more arguments

```
function(arg1 = object1, arg2 = object2, arg3 = object3)
```

```
my_vector <- seq(from = 1, to = 10, by = 1)
my_vector
```

```
[1]  1  2  3  4  5  6  7  8  9 10
```

- Functions in R can take zero or more arguments

```
function(arg1 = object1, arg2 = object2, arg3 = object3)
```

```
my_vector <- seq(from = 1, to = 10, by = 1)
my_vector
```

```
[1]  1  2  3  4  5  6  7  8  9 10
```

```
mean(x = my_vector)
```

```
[1] 5.5
```

```
my_vector <- c(1, 2, 3, NA, NA, NA, 3, 2, 1)
mean(x = my_vector)
```

```
[1] NA
```

```
my_vector <- c(1, 2, 3, NA, NA, NA, 3, 2, 1)
mean(x = my_vector)
```

```
[1] NA
```

```
mean(x = my_vector, na.rm = TRUE)
```

```
[1] 2
```

- You don't have to specify argument names if you type them in order.
- Since `x` is the first argument of `mean()`, no need to type `mean(x = my_vector)`
- Instead, can just type `mean(my_vector)`
- This cuts down on the amount you have to type

- OK, so now we know how to assign stuff and functions

Data

- OK, so now we know how to assign stuff and functions
- Let’s learn about how R thinks about data

Data

- OK, so now we know how to assign stuff and functions
- Let's learn about how R thinks about data
    - "data" here doesn't have to mean data from e.g. a survey

- OK, so now we know how to assign stuff and functions
- Let's learn about how R thinks about data
    - "data" here doesn't have to mean data from e.g. a survey
- R cares about the class (type) of data and its dimension(s)

- We'll discuss the four most common data types:
    - Numeric
    - Logical
    - Character
    - Factor
- We'll also cover `NA`

Numeric

- Numeric is how R thinks about numbers!
- These can also be called “integer” (if round numbers) or “double”

```
class(c(1, 2, 3))
```

```
[1] "numeric"
```

- Numeric is how R thinks about numbers!
- These can also be called "integer" (if round numbers) or "double"

```
class(c(1, 2, 3))
```

```
[1] "numeric"
```

```
sum(c(1, 2, 3))
```

```
[1] 6
```

Numeric

- Numeric is how R thinks about numbers!
- These can also be called “integer” (if round numbers) or “double”

```
class(c(1, 2, 3))
```

```
[1] "numeric"
```

```
sum(c(1, 2, 3))
```

```
[1] 6
```

```
class(sum(c(1, 2, 3)))
```

```
[1] "numeric"
```

Logical

- Logical can take two values — `TRUE` or `FALSE`

- Logical can take two values — `TRUE` or `FALSE`
- This is useful for dummy variables and tests

- Logical can take two values — `TRUE` or `FALSE`
- This is useful for dummy variables and tests

```
1:10 > 5
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Character

- Characters represent text
- Sometimes these are called “strings”

```
c("This", "vector", "is", "of", "length", "what?")
c("How about this one?")
```

Factor

- Factors are how R thinks about categorical variables

Factor

- Factors are how R thinks about categorical variables
- We already worked with these when we used the `continent` variable from `gapminder`

Factor

- Factors are how R thinks about categorical variables
- We already worked with these when we used the `continent` variable from `gapminder`

```
head(gapminder$continent)
```

```
[1] Asia Asia Asia Asia Asia Asia
Levels: Africa Americas Asia Europe Oceania
```

What type of data are the following?

```
182
c("My name is Alex")
"TRUE"
FALSE
c(1, 2, 3)
c(1, "Alex", TRUE)
```

What's the difference?

```
[1] 1 2 3 4 5 6
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Data dimensions

What's the difference?

```
[1] 1 2 3 4 5 6
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

- Data can have dimensions
- Numeric, logical, character, and factors are single dimensions (so are lists)

Data dimensions

What's the difference?

```
[1] 1 2 3 4 5 6
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

- Data can have dimensions
- Numeric, logical, character, and factors are single dimensions (so are lists)

- That matrix is a 3 by 2 matrix
- Why might we want to have two-dimensional data?

- Matrices must have the same type, but we can mix and match types with a `data.frame`
- Remember `gapminder` from earlier?
- We used a `data.frame` to store columns with different data types

- Matrices must have the same type, but we can mix and match types with a `data.frame`
- Remember `gapminder` from earlier?
- We used a `data.frame` to store columns with different data types

- We can access (index, subset) `data.frame` objects using notation similar to matrix notation:

```
gapminder[2, 1] # get whatever is in the second row, 1st col
gapminder[1, ]  # get the first row (all)
gapminder[, 1]  # get the first col (all)
gapminder[, "country"] # select by name
gapminder$country # slightly different
```

Review

- What we learned

Review

- What we learned
- Missingness propagates

Review

- What we learned
- Missingness propagates
- Functions & arguments

- What we learned
- Missingness propagates
- Functions & arguments
- Basic vectors: numeric, logical, character, factor

Review

- What we learned
- Missingness propagates
- Functions & arguments
- Basic vectors: numeric, logical, character, factor
- Dimensions & the data.frame

Data import & manipulation

- Importing data in R is either trivially easy (usually) or super specific and difficult (rarely), so we won’t actually be doing this

- Importing data in R is either trivially easy (usually) or super specific and difficult (rarely), so we won't actually be doing this

- R has a lot of build in functions: `read.csv()`, `read.table()`, etc
- Packages provide still more: `readr::read_csv()`, `haven::read_dta()`, etc

- Importing data in R is either trivially easy (usually) or super specific and difficult (rarely), so we won't actually be doing this

- R has a lot of build in functions: `read.csv()`, `read.table()`, etc
- Packages provide still more: `readr::read_csv()`, `haven::read_dta()`, etc

- I prefer the `rio` package because I don't have to think
- Always gives you a `data.frame`:

```
library(rio)
csv_data <- import("file.csv")
stata_data <- import("file.dta")
```

Working directories & project structure

- R has the concept of a "working directory"
- You can see where this is by typing `getwd()` into the console
- I like to store data and code in separate folders:
- Tip: Rstudio can manage "projects" that take care of a lot of this

Simple project structure

```
my-paper-project/
|--- code/
|     |--- my-script.R
|     |--- my-alt-script.R
|--- data/
|     |--- awesome-data.csv
|--- output/
|     |--- figure1.eps
|     |--- figure2.eps
|     |--- table1.tex
|     |--- table2.tex
|--- my-paper.tex
```

Relative paths

- If you have code like that, you need to know what a relative path is so that code in your `code/` directory can load data in your `data/` directory!

Relative paths

- If you have code like that, you need to know what a relative path is so that code in your `code/` directory can load data in your `data/` directory!
- So if we're running a file from `code/` (that's the working directory), we can load data by doing:

```
my_awesome_data <- import("../data/awesome_data.csv")
```

- If you have code like that, you need to know what a relative path is so that code in your `code/` directory can load data in your `data/` directory!
- So if we're running a file from `code/` (that's the working directory), we can load data by doing:

```
my_awesome_data <- import("../data/awesome_data.csv")
```

- Two dots `..` says "go up one directory", we could chain them to go up two: `../..`

- We are going to use dplyr, another package you've installed, to help us transform data

[1] Technically, a `tibble`, but the difference isn't very much, so we'll ignore that

- We are going to use dplyr, another package you've installed, to help us transform data

- `filter()` drops rows based on columns
- `select()` selects columns

[1] Technically, a `tibble`, but the difference isn't very much, so we'll ignore that

- We are going to use dplyr, another package you've installed, to help us transform data

- `filter()` drops rows based on columns
- `select()` selects columns

- `mutate()` creates new variables
- `summarize()` return statistics

[1]Technically, a `tibble`, but the difference isn't very much, so we'll ignore that

- We are going to use dplyr, another package you've installed, to help us transform data

- `filter()` drops rows based on columns
- `select()` selects columns

- `mutate()` creates new variables
- `summarize()` return statistics

- `group_by()` allows us to do the above by groups

[1] Technically, a `tibble`, but the difference isn't very much, so we'll ignore that

- We are going to use dplyr, another package you've installed, to help us transform data

- `filter()` drops rows based on columns
- `select()` selects columns

- `mutate()` creates new variables
- `summarize()` return statistics

- `group_by()` allows us to do the above by groups

-These functions take data as the first argument and always return a data.frame[1]

```
library(dplyr)
```

[1] Technically, a `tibble`, but the difference isn't very much, so we'll ignore that

`filter`

- `filter()` uses logical statements (that are TRUE) to return rows:

```
filter(gapminder, continent == "Asia")
filter(gapminder, continent == "Asia" & year >= 2000)
filter(gapminder, continent == "Asia" & year != 2000)
filter(gapminder, continent == "Asia" | year == 2000)
```

Quiz

- Use filter to return all the rows containing observations from Asia or Africa

Quiz

- Use filter to return all the rows containing observations from Asia or Africa

```
filter(gapminder, continent == "Asia" | continent == "Africa")
filter(gapminder, continent %in% c("Asia", "Africa"))
```

`select`

- The `select` function selects one or more columns:

```
select(gapminder, country)
select(gapminder, country, year, continent)
select(gapminder, -continent)
```

- several helper functions (e.g. `starts_with`), see `?select` for examples

mutate

- Mutate creates new variables:

```
mutate(gapminder, gdp = pop * gdpPercap)
```

```
# A tibble: 1,704 x 7
       country continent  year lifeExp      pop gdpPercap        
        <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>        <
 1 Afghanistan      Asia  1952  28.801  8425333  779.4453  656708
 2 Afghanistan      Asia  1957  30.332  9240934  820.8530  758544
 3 Afghanistan      Asia  1962  31.997 10267083  853.1007  875885
 4 Afghanistan      Asia  1967  34.020 11537966  836.1971  964801
 5 Afghanistan      Asia  1972  36.088 13079460  739.9811  967855
 6 Afghanistan      Asia  1977  38.438 14880372  786.1134 1169765
 7 Afghanistan      Asia  1982  39.854 12881816  978.0114 1259856
```

summarize

- summarize (or summarise if you prefer) creates summary statistics:

```
summarize(gapminder, mean_life = mean(lifeExp))
```

```
# A tibble: 1 x 1
  mean_life
      <dbl>
1  59.47444
```

summarize

- `summarize` (or `summarise` if you prefer) creates summary statistics:

```
summarize(gapminder, mean_life = mean(lifeExp))
```

```
# A tibble: 1 x 1
  mean_life
      <dbl>
1  59.47444
```

- Though whoop-de-doo, we could've just done `mean(gapminder$lifeExp)` to get that!

summarize

- `summarize` (or `summarise` if you prefer) creates summary statistics:

```
summarize(gapminder, mean_life = mean(lifeExp))
```

```
# A tibble: 1 x 1
  mean_life
      <dbl>
1  59.47444
```

- Though whoop-de-doo, we could've just done `mean(gapminder$lifeExp)` to get that!
- Much more useful if we do this by groups

group_by

- All the functions we just learned can be performed by groups!
- This is really exciting and makes life much easier
- Calculate mean life expectancy by year:

group_by

- All the functions we just learned can be performed by groups!
- This is really exciting and makes life much easier
- Calculate mean life expectancy by year:

```
summarize(group_by(gapminder, year), mean_life = mean(lifeExp))
## Or, to add it to the data:
mutate(group_by(gapminder, year), year_mean_life = mean(lifeExp))
```

```
# A tibble: 12 x 2
    year mean_life
   <int>     <dbl>
 1  1952  49.05762
 2  1957  51.50740
 3  1962  53.60925
```

`group_by`, CONTINUED

- Calculate change in life expectancy by country:

group_by, CONTINUED

- Calculate change in life expectancy by country:

```
mutate(group_by(gapminder, country),
       life_change = lifeExp - lag(lifeExp))
```

```
# A tibble: 1,704 x 7
# Groups:   country [142]
       country continent  year lifeExp      pop gdpPercap life_ch
        <fctr>    <fctr> <int>   <dbl>    <int>     <dbl>       <
 1 Afghanistan      Asia  1952  28.801  8425333  779.4453
 2 Afghanistan      Asia  1957  30.332  9240934  820.8530       1
 3 Afghanistan      Asia  1962  31.997 10267083  853.1007       1
 4 Afghanistan      Asia  1967  34.020 11537966  836.1971       2
 5 Afghanistan      Asia  1972  36.088 13079460  739.9811       2
```

`group_by`, continued

- You can group by multiple variables

```
summarize(group_by(gapminder, continent, year),
          mean_life = mean(lifeExp))
```

```
# A tibble: 60 x 3
# Groups:   continent [?]
   continent  year mean_life
      <fctr> <int>     <dbl>
 1    Africa  1952  39.13550
 2    Africa  1957  41.26635
 3    Africa  1962  43.31944
 4    Africa  1967  45.33454
 5    Africa  1972  47.45094
```

- What if we want to select all countries in Africa and calculate mean life expectancy by year?

- What if we want to select all countries in Africa and calculate mean life expectancy by year?

Chaining

- What if we want to select all countries in Africa and calculate mean life expectancy by year?
- This is easy to do because the dplyr functions always take the data as their first argument and always return a data.frame

Chaining, continued

- One option:

```
summarize(group_by(filter(gapminder,
                          continent == "Africa"),
                   year),
          mean_life = mean(lifeExp))
```

- One option:

```
summarize(group_by(filter(gapminder,
                          continent == "Africa"),
                   year),
          mean_life = mean(lifeExp))
```

- Or we could assign to objects along the way

```
just_africa <- filter(gapminder,continent == "Africa"),
africa_by_year <- group_by(just_africa, year)
summarize(africa_by_year, mean_life = mean(lifeExp))
```

- Both of those have downsides, though

Piping

- Both of those have downsides, though
- We'll use the *pipe* **%>%** to "pipe" the thing on the left into the thing on the right:
- Tip: In Rstudio, use Ctrl+shift+m (Cmd+shift+m) to get **%>%**

- Both of those have downsides, though
- We'll use the *pipe* **%>%** to "pipe" the thing on the left into the thing on the right:
- Tip: In Rstudio, use Ctrl+shift+m (Cmd+shift+m) to get **%>%**

```
gapminder %>%
  filter(continent == "Africa") %>%
```

Piping

- Both of those have downsides, though
- We'll use the *pipe* **%>%** to "pipe" the thing on the left into the thing on the right:
- Tip: In Rstudio, use Ctrl+shift+m (Cmd+shift+m) to get **%>%**

```
gapminder %>%
  filter(continent == "Africa") %>%

group_by(year) %>%
```

- Both of those have downsides, though
- We'll use the *pipe* **%>%** to "pipe" the thing on the left into the thing on the right:
- Tip: In Rstudio, use Ctrl+shift+m (Cmd+shift+m) to get **%>%**

```
gapminder %>%
  filter(continent == "Africa") %>%

group_by(year) %>%

summarize(meanlife = mean(lifeExp))
```

Quiz

- Create a data.frame containing the continent, year, avg life expectancy, and change in avg life expectancy

```
gapminder %>%
  group_by(continent, year) %>%
  summarize(avg_life = mean(lifeExp)) %>%
  mutate(change_life = avg_life - lag(avg_life))
```

```
# A tibble: 60 x 4
# Groups:   continent [5]
   continent  year avg_life change_life
      <fctr> <int>    <dbl>       <dbl>
 1    Africa  1952 39.13550          NA
 2    Africa  1957 41.26635  2.13084615
 3    Africa  1962 43.31944  2.05309615
 4    Africa  1967 45.33454  2.01509615
 5    Africa  1972 47.45094  2.11640385
```

Ungrouping

- Note that our answer had “continent” as a group
- It’s easy to forget about this, so if you’re saving the object for use later, you may want to run `ungroup()` to undo the grouping on the data.frame.

- Those commands take care of the most common data manipulation tasks

Other data manipulation

- Those commands take care of the most common data manipulation tasks
- There's tons more but we don't have the time to go over them all

- Those commands take care of the most common data manipulation tasks
- There's tons more but we don't have the time to go over them all
- Search engines and R's help are your friend

- We learned how to use some of the most common `dplyr` functions to manipulate data (filter, select, mutate, summarize)
- `group_by` makes doing this by groups super easy
- Piping can make it easier to read code

Diamonds

- We just learned a lot, let's apply some of it to a new dataset

- We just learned a lot, let's apply some of it to a new dataset
- I'm also going to switch from this powerpoint to a "live demo!"

Reporting from R

- We’ve learned most of what you need to do data analysis!
- Now let’s do a new analysis on how to report, so we’ll learn
    - How to report
    - Review much of what we learned
    - Learn a few more tricks and tips
- Right now is a good time to “restart” R and to make a project
- I put mine in `~/research/awesome-paper/` but you can put yours wherever!

- Let's change the dataset we're using, just for something new:
- We'll use the `midwest` dataset from `ggplot2`, which has info on some U.S. midwest counties:

```
library(tidyverse)
midwest
```

- Let's look at the relationship between college education and the percent living in poverty. And maybe this looks different in metro areas, so let's keep that in mind too.

Descriptive statistics

- Let's look at the relationship between college education and the percent living in poverty. And maybe this looks different in metro areas, so let's keep that in mind too.

- I always like to show some descriptive statistics
- Find the mean and standard deviation of our three variables!

- Let’s look at the relationship between college education and the percent living in poverty. And maybe this looks different in metro areas, so let’s keep that in mind too.

- I always like to show some descriptive statistics
- Find the mean and standard deviation of our three variables!

```
midwest %>%
  select(percbelowpoverty, percollege, inmetro) %>%
  summarize_all(funs(mean, sd))
```

Functions

- What if we want another function other than mean, sd, etc?

Functions

- What if we want another function other than mean, sd, etc?
- Very likely that it's either in base R or someone has written it

- What if we want another function other than mean, sd, etc?
- Very likely that it's either in base R or someone has written it
- Or you can write a function yourself!

Functions

- What if we want another function other than mean, sd, etc?
- Very likely that it's either in base R or someone has written it
- Or you can write a function yourself!
- This is actually really easy in R

- Let's pretend R didn't have a `mean` function
- How would we write it?
- What do we need to find?

- Let's pretend R didn't have a `mean` function
- How would we write it?
- What do we need to find?

$$\frac{1}{n}\sum x$$

- Let's pretend R didn't have a `mean` function
- How would we write it?
- What do we need to find?

$$\frac{1}{n}\sum x$$

```
sum(x) / length(x)
```

```
my_mean <- function(x){
  sum(x) / length(x)
}
my_mean(-1:10)
```

```
[1] 4.5
```

```
my_mean <- function(x){
  sum(x) / length(x)
}
my_mean(-1:10)
```

```
[1] 4.5
```

But what about NA???

- An `if` statement allows us to conditionally execute code

```
my_name <- "Alex"
if (my_name == "Alex"){
  print("I'm Alex!!!")
} else{
  print("You aren't Alex!!!")
}
```

How to modify our function???

```
my_mean <- function(x){
  sum(x) / length(x)
}
```

How to modify our function???

```
my_mean <- function(x){
  sum(x) / length(x)
}
```

- Solution: Use an if statement! But we gotta let the user tell us whether to remove NA...

```
my_mean <- function(x, na.rm = FALSE){
  if(na.rm){ x <- x[!is.na(x)]}
  sum(x) / length(x)
}
```

Always test a function to make sure it works!

```
my_mean(c(NA, 0, 1), TRUE)
my_mean(c(NA, 0, 1), FALSE)
```

```
midwest %>%
  select(percbelowpoverty, percollege, inmetro) %>%
  summarize_all(funs(mean, sd))
```

Back to our regularly scheduled program...

```
midwest %>%
  select(percbelowpoverty, percollege, inmetro) %>%
  summarize_all(funs(mean, sd))
```

- But what if we want to show that in our paper?

- There are several packages that let you easily make LaTeX tables, let's use `stargazer`:

```
library(stargazer)
```

- Can handle Word too, need to do an html dance. See package docs.

Descriptive stats, latex table:

```
midwest %>%
  select(percbelowpoverty, percollege, inmetro) %>%
  ## stargazer is picky about tibbles vs data.frames
  as.data.frame %>%
  stargazer(out = "../output/desc-stats.tex",
            title = "Descriptive Statistics")
```

- use `\input{output/desc-stats.tex}` to import the table into your paper

Table 1: Descriptive Statistics

Statistic	N	Mean	St. Dev.	Min	Max
percbelowpoverty	437	12.511	5.150	2.180	48.691
percollege	437	18.273	6.262	7.336	48.079
inmetro	437	0.343	0.475	0	1

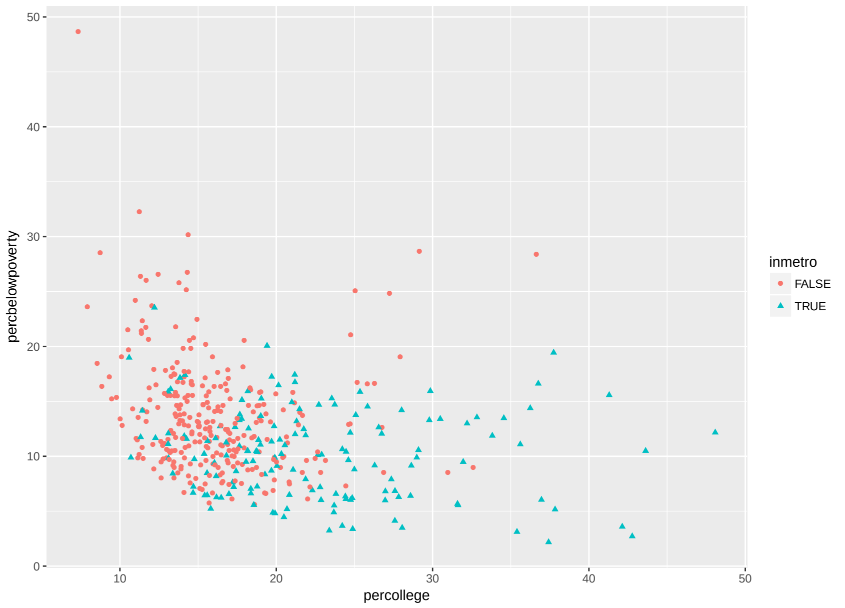
Plot 1

- Let’s make a scatterplot!

Plot 1

- Let's make a scatterplot!
- Make a scatterplot with `percbelowpoverty` on the y-axis and include info on `percollege` and `inmetro`

```
g <- midwest %>%
  ## inmetro is a number but needs to be discrete.
  ## as.logical will convert so that a 0 is FALSE
  mutate(inmetro = as.logical(inmetro)) %>%
  ggplot(aes(percollege, percbelowpoverty,
             color = inmetro,
             shape = inmetro)) +
  geom_point()
```

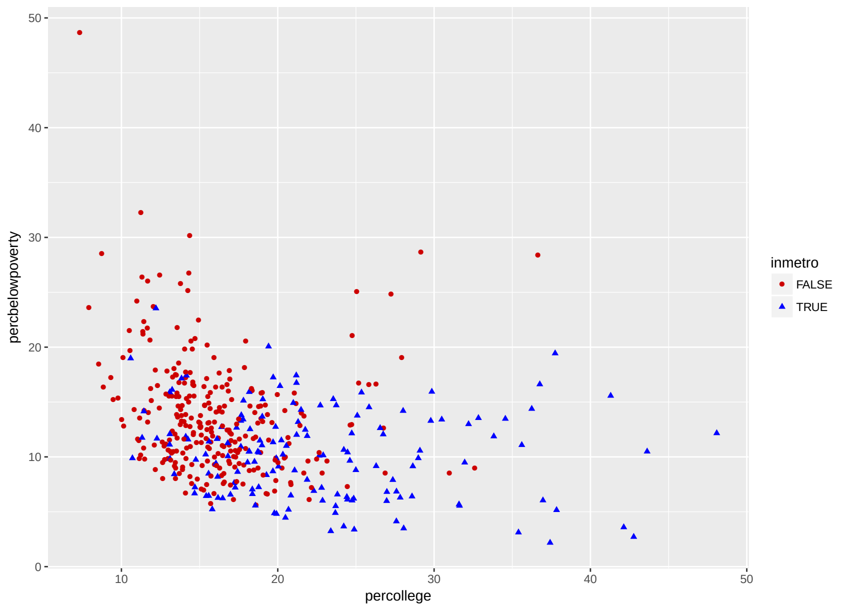


percbelowpoverty
percollege
0
10
20
30
40
50
inmetro
FALSE
TRUE

- Note that I assigned the plot to an object `g`
- We might want to change some more stuff about the graph (legends, assign colors, etc)
- This way I don't have to re-run the same code

- You may want to change the color, label legends, etc
- use `scale_aes_type` to do so
- So, for example, we can do `scale_color_manual` to change the properties of the color scale.
- Let's change it so that metro areas are blue and rural areas are red:

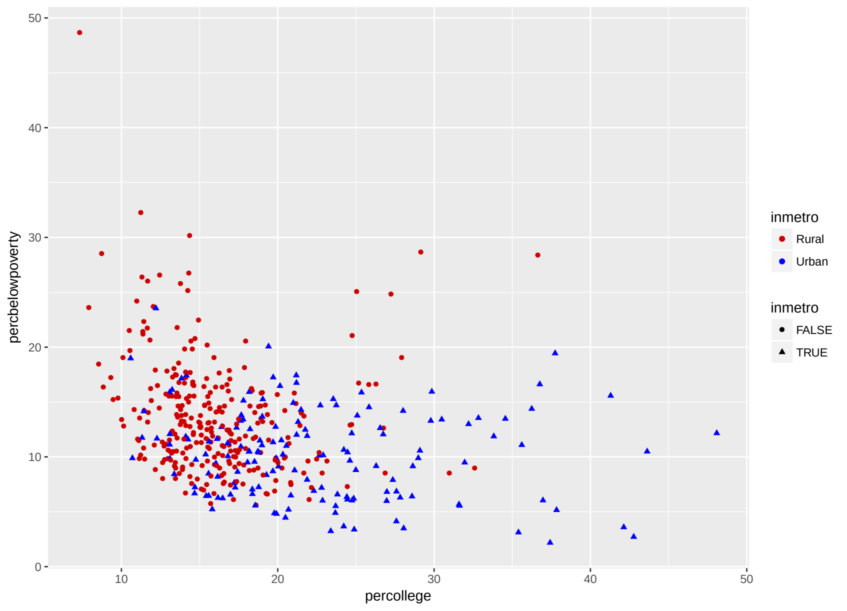
```
## Plain red is super harsh, let's scale it back a bit:
g + scale_color_manual(values = c("red3", "blue"))
```



50
40
30
20
10
0
percbelowpoverty
10
20
30
40
50
percollege
inmetro
FALSE
TRUE

- Of course, `FALSE` and `TRUE` are not good legend labels. We can change those too with the `scale_color_manual` command:

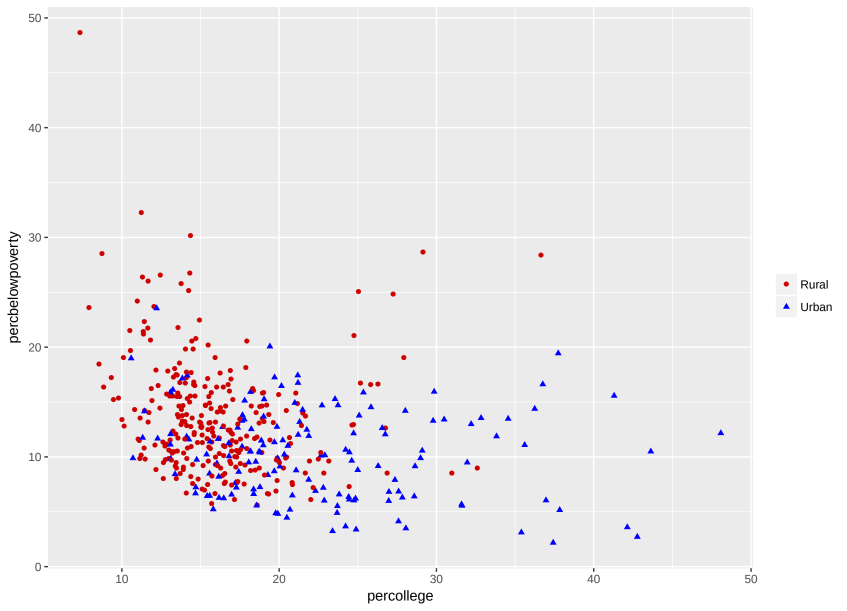
```
g + scale_color_manual(values = c("red3", "blue"),
                       labels = c("Rural", "Urban"))
```


50
40
30
20
10
0
10
20
30
40
50
percbelowpoverty
percollege
inmetro
Rural
Urban
inmetro
FALSE
TRUE

UGHHHHHHHHH

- Now the legends are separate, though. Need to tell the shape aesthetic to use the same labels!
- While we're at it, let's remove the legend title (name):
- Since we're done changing the scales, let's reassign `g`

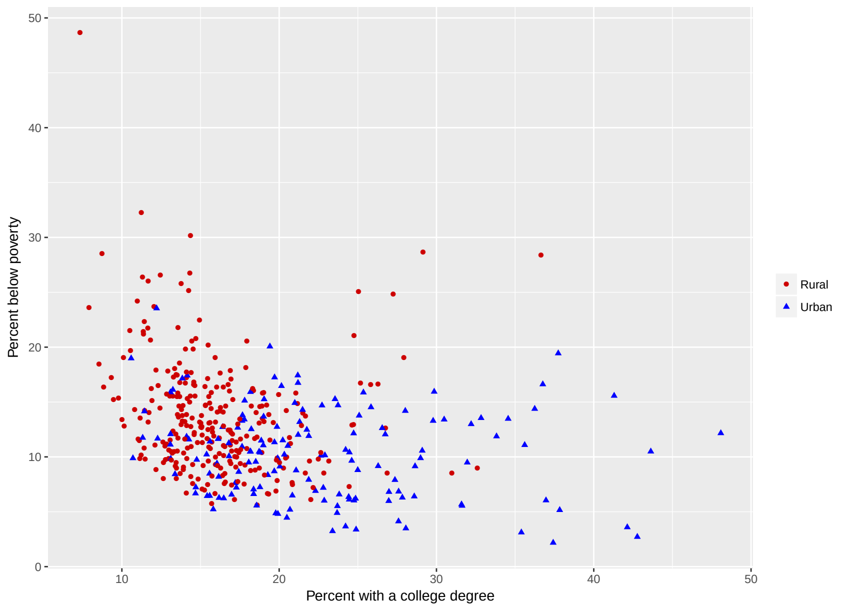
```
g <- g + scale_color_manual(values = c("red3", "blue"),
                            labels = c("Rural", "Urban"),
                            name = "") +
  scale_shape_discrete(labels = c("Rural", "Urban"),
                       name = "")
```



50
40
30
20
10
0
percbelowpoverty
10
20
30
40
50
percollege
Rural
Urban

- We should probably fix up our axis labels
- Note that if you want to give the plot a title, subtitle, or caption, you may do so here

```
g <- g + labs(y = "Percent below poverty",
              x = "Percent with a college degree")
```



50
40
30
20
10
0
Percent below poverty
10
20
30
40
50
Percent with a college degree
Rural
Urban

Background and themes

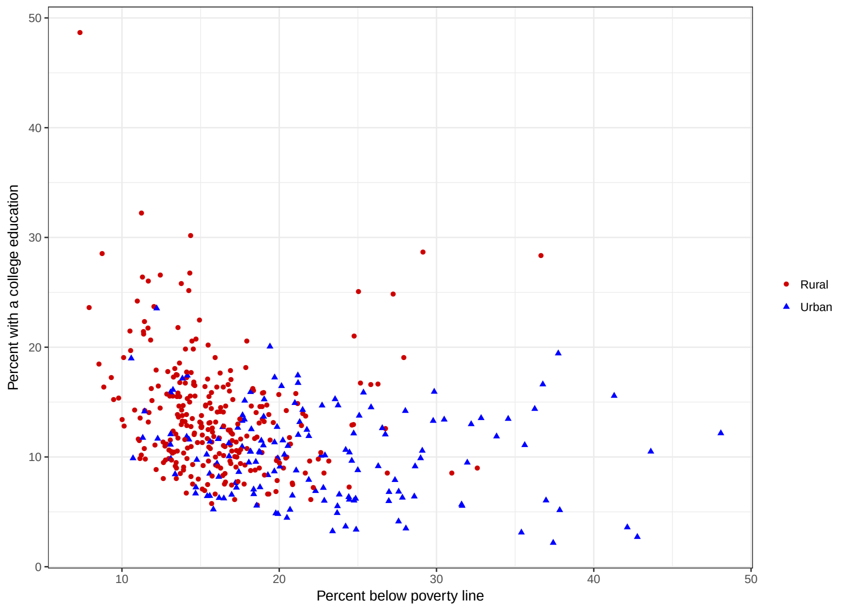
- I'm not a fan of the default grey background.
- You can adjust everything yourself, but there are several themes that come built-in
- The package `ggthemes` has many other themes
- You can make it look like you're graphing for the economist. Or from Stata

Much themes, wow

```
g + theme_grey()
g + theme_gray()
g + theme_bw()
g + theme_linedraw()
g + theme_light()
g + theme_dark()
g + theme_minimal()
g + theme_classic()
g + theme_void()
```

Plot 1, full

```
midwest %>%
  mutate(inmetro = as.logical(inmetro)) %>%
  ggplot(aes(percollege, percbelowpoverty,
             color = inmetro,
             shape = inmetro)) +
  geom_point() +
  scale_color_manual(values = c("red3", "blue"), labels = c("Rura
                     name = "") +
  scale_shape_discrete(labels = c("Rural", "Urban"), name = "") +
  labs(x = "Percent below poverty line",
       y = "Percent with a college education") +
  theme_bw()
```


Percent with a college education
Percent below poverty line
0
10
20
30
40
50
10
20
30
40
50
Rural
Urban

How to save ggplots

- The `ggsave` function saves a plot (by default, the last one you plotted)
- It's important to specify the width and height

```
ggsave("../output/my-scatterplot.eps",
       ## Important to specify!!!
       width = 9, height = 6.5)
```

- Let's run a linear predicting poverty with education and include an interaction term for inmetro
  - Yes, I'm ignoring all kinds of issues with this particular model

```
my_reg <- lm(percbelowpoverty ~ percollege * inmetro,
             data = midwest)
summary(my_reg)
```

```
stargazer(my_reg,
          out = "../output/my-reg.tex")
```

```
stargazer(my_reg,
          out = "../output/my-reg.tex")
```

- Use `\input{output/my-reg.tex}` in your LaTeX document to import the table!

Table 2:

	Dependent variable:		
	percbelowpoverty		
	(1)	(2)	(3)
percollege	−0.231***		−0.231***
	(0.038)		(0.069)
inmetro		−3.375***	−5.144***
		(0.494)	(1.707)
percollege:inmetro			0.144*
			(0.087)
Constant	16.740***	13.669***	17.383***
	(0.731)	(0.289)	(1.151)
Observations	437	437	437
R^2	0.079	0.097	0.125
Adjusted R^2	0.077	0.095	0.119
Residual Std. Error	4.948 (df = 435)	4.900 (df = 435)	4.835 (df = 433)
F Statistic	37.410*** (df = 1; 435)	46.744*** (df = 1; 435)	20.581*** (df = 3; 433)
Note:			*$p<0.1$; **$p<0.05$; ***$p<0.01$

Multiple models

- Oftentimes, we want multiple models

Multiple models

- Oftentimes, we want multiple models
- You can, of course, copy paste code, but that's error prone (typos, ugh), and difficult to change later on

Multiple models

- Oftentimes, we want multiple models
- You can, of course, copy paste code, but that's error prone (typos, ugh), and difficult to change later on
- There's an easy solution for this, but first let's talk about a data structure we haven't mentioned much yet:

- Oftentimes, we want multiple models
- You can, of course, copy paste code, but that's error prone (typos, ugh), and difficult to change later on
- There's an easy solution for this, but first let's talk about a data structure we haven't mentioned much yet:

- Oftentimes, we want multiple models
- You can, of course, copy paste code, but that's error prone (typos, ugh), and difficult to change later on
- There's an easy solution for this, but first let's talk about a data structure we haven't mentioned much yet:

THE LIST

Lists

- A list is one dimensions (like numeric, logical, character, factor)

Lists

- A list is one dimensions (like numeric, logical, character, factor)
- But each element can be of a different type

Lists

- A list is one dimensions (like numeric, logical, character, factor)
- But each element can be of a different type
- We can create lists with the `list` command
- Look at the difference:

```
c(3, TRUE, "Nancy")
```

```
[1] "3"     "TRUE"  "Nancy"
```

```
list(3, TRUE, "Nancy")
```

```
[[1]]
[1] 3

[[2]]
[1] TRUE

[[3]]
[1] "Nancy"
```

- Subsetting lists can be a little weird
- We use `[[` or `[` to subset
- First, create a list:

```
x <- list(c(1:10),
         c(TRUE, NA, TRUE),
         c("Bob", "Alice", "Nancy", "Drew"))
```

- What is the difference:

```
x[[1]]
x[1]
```

Named lists

- The double bracket contains the thing at the position,
- Single bracket returns a list of the thing at the position
- Elements of a list can have names:

```
names(x) <- c("nums", "logs", "chars")
## Can also specify at creation time e.g. list(nums = 1:10) etc
```

```
x

$nums
 [1]  1  2  3  4  5  6  7  8  9 10

$logs
[1] TRUE   NA TRUE

$chars
[1] "Bob"   "Alice" "Nancy" "Drew"
```

Subsetting named lists

- We can now access elements of the list by name instead of by position:

```
x$chars
```

```
[1] "Bob"   "Alice" "Nancy" "Drew"
```

Data frames are lists too!

- Remember we can use `dataframe$varname` to access variables from a data frame?

Data frames are lists too!

- Remember we can use `dataframe$varname` to access variables from a data frame?
- Does this look similar to what we just did with lists?

Data frames are lists too!

- Remember we can use `dataframe$varname` to access variables from a data frame?
- Does this look similar to what we just did with lists?
- That’s because data frames are secretly lists themselves!

- OK, why did we just learn about lists?

- OK, why did we just learn about lists?

Back to modeling

- OK, why did we just learn about lists?
- We were modeling percent below poverty with an interaction between college education and metro area status

- OK, why did we just learn about lists?
- We were modeling percent below poverty with an interaction between college education and metro area status

- OK, why did we just learn about lists?
- We were modeling percent below poverty with an interaction between college education and metro area status
- What if we want to “build the model” by including constituent variables one at a time?

- OK, why did we just learn about lists?
- We were modeling percent below poverty with an interaction between college education and metro area status
- What if we want to "build the model" by including constituent variables one at a time?
- One way:

```
model1 <- lm(percbelowpoverty ~ percollege, data = midwest)
model2 <- lm(percbelowpoverty ~ inmetro, data = midwest)
model3 <- lm(percbelowpoverty ~ percollege * inmetro, data = midw
```

- But if we do that, we now have three models just floating around.
- To get summary measures:

- But if we do that, we now have three models just floating around.
- To get summary measures:

```
summary(model1)
summary(model2)
summary(model3)
```

Y-hats:

```
predict(model1)
predict(model2)
predict(model3)
```

- That's just with three models!

The problem

- That's just with three models!
- Sometimes we run many more and the problem only gets worse!

The problem

- That's just with three models!
- Sometimes we run many more and the problem only gets worse!

The problem

- That’s just with three models!
- Sometimes we run many more and the problem only gets worse!

- Idea! let’s use the list to make life easier!

```
my_formulae <- list(model1 = percbelowpoverty ~ percollege,
                   model2 = percbelowpoverty ~ inmetro,
                   model3 = percbelowpoverty ~ percollege * inmet
```

Run the models!

- Base R provides `lapply` which iterates over lists

```
my_regs <- lapply(my_formulae,
                  function(l_ele){lm(l_ele, data = midwest)})
```

- First argument is a list, second is a function to apply to each element of the list
- We use an *anonymous function* - one that we create on the fly. You could've created a named function too like we did with `my_mean` previously

- I don't really like that syntax though so I use `map` from the `purrr` package.
- This will do the same thing; the tilde magically creates an anonymous function in the background

```
my_regs <- map(my_formulae, ~ lm(.x, data = midwest))
```

Summarize the models

```
map(my_regs, summary)
```

```
map(my_regs, predict)
```

```
map(my_regs, residuals)
```

The broom package has three functions that turns models into data.frames:

1. `glance()` returns a row with model quality/complexity
2. `tidy()` returns a row for each coefficient
3. `augment()` returns a row for every row in the data, adding some values (usually residuals and the like)

```
map(my_regs, broom::glance)
```

```
map(my_regs, broom::tidy)
```

```
map(my_regs, broom::augment)
```

- The stargazer function is smart enough to figure out multiple models:

```
stargazer(my_regs,
          out = "../output/my-reg.tex")
```

Table 3:

	Dependent variable:		
	percbelowpoverty		
	(1)	(2)	(3)
percollege	−0.231***		−0.231***
	(0.038)		(0.069)
inmetro		−3.375***	−5.144***
		(0.494)	(1.707)
percollege:inmetro			0.144*
			(0.087)
Constant	16.740***	13.669***	17.383***
	(0.731)	(0.289)	(1.151)
Observations	437	437	437
R^2	0.079	0.097	0.125
Adjusted R^2	0.077	0.095	0.119
Residual Std. Error	4.948 (df = 435)	4.900 (df = 435)	4.835 (df = 433)
F Statistic	37.410*** (df = 1; 435)	46.744*** (df = 1; 435)	20.581*** (df = 3; 433)
Note:			*p<0.1; **p<0.05; ***p<0.01

Plotting predicted values

- We usually want to plot the predicted values from our models

Plotting predicted values

- We usually want to plot the predicted values from our models
- We'll keep using a linear model, but this can really be anything

Plotting predicted values

- We usually want to plot the predicted values from our models
- We'll keep using a linear model, but this can really be anything
- Let's say we want to compare how adding the interaction term affects our predictions

- We usually want to plot the predicted values from our models
- We'll keep using a linear model, but this can really be anything
- Let's say we want to compare how adding the interaction term affects our predictions
- One way: Plot predicted values from our first and third regressions!

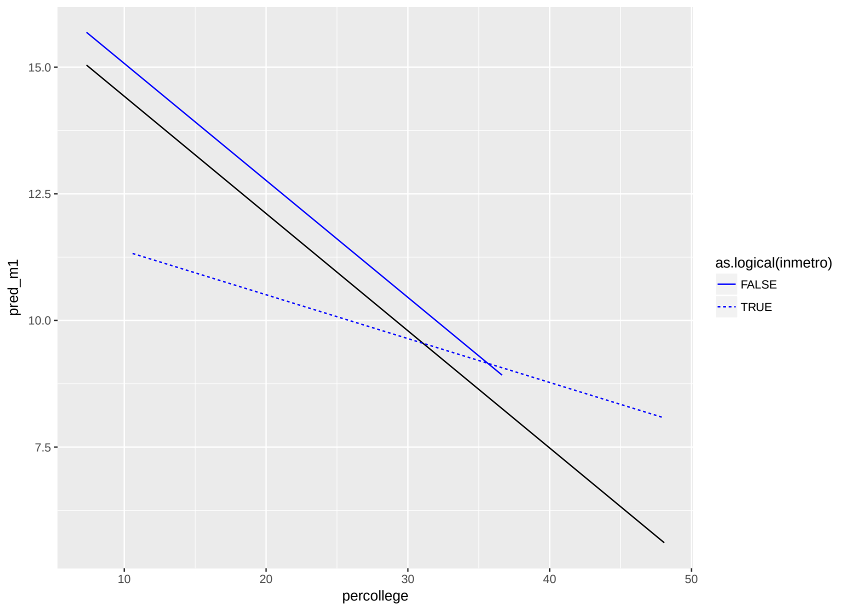
Plotting predicted values

- We usually want to plot the predicted values from our models
- We'll keep using a linear model, but this can really be anything
- Let's say we want to compare how adding the interaction term affects our predictions
- One way: Plot predicted values from our first and third regressions!
- Let's add them to our data.frame

Getting predicted values

```
my_midwest <- midwest
my_midwest$pred_m1 <- predict(my_regs$model1)
my_midwest$pred_m3 <- predict(my_regs$model3)
```

```
ggplot(my_midwest,
       aes(percollege)) +
  geom_line(aes(y = pred_m1)) +
  geom_line(aes(y = pred_m3,
                linetype = as.logical(inmetro)),
            color = "blue")
```



pred_m1
15.0
12.5
10.0
7.5
10
20
30
40
50
percollege
as.logical(inmetro)
FALSE
TRUE

Merging

- Let's talk about merging data!

- Let’s talk about merging data!
- Oftentimes we have different datasets that we need to merge together for whatever reason

Merging

- Let's talk about merging data!
- Oftentimes we have different datasets that we need to merge together for whatever reason
- dplyr refers to "merging" as "joining," which is language borrowed from SQL

- Let’s talk about merging data!
- Oftentimes we have different datasets that we need to merge together for whatever reason
- dplyr refers to “merging” as “joining,” which is language borrowed from SQL
- Let’s go to another “live demo”

Merging

- Let’s talk about merging data!
- Oftentimes we have different datasets that we need to merge together for whatever reason
- dplyr refers to “merging” as “joining,” which is language borrowed from SQL
- Let’s go to another “live demo”
- Let’s look at two toy datasets that come with dplyr

- Stack overflow (but have an MRE)

How to get help

- Stack overflow (but have an MRE)
- Twitter (#rstats)

How to get help

- Stack overflow (but have an MRE)
- Twitter (#rstats)
- Me! (branham@utexas.edu)

Other tools that work well with R

- Git

Other tools that work well with R

- Git
- LaTeX

Other tools that work well with R

- Git
- $\LaTeX$
- (r)markdown

Thanks for coming!