

# Data Visualization - 3.

# Make Some Graphs

Kieran Healy  
Code Horizons

January 2025

# Make Some Graphs

# Load our libraries

```
library(here)      # manage file paths
library(socviz)    # data and some useful functions
library(tidyverse) # your friend and mine

— Attaching core tidyverse packages ————— tidyverse 2.0.0 —
✓ dplyr     1.1.4    ✓ readr     2.1.5
✓forcats   1.0.0    ✓ stringr   1.5.1
✓ ggplot2   3.5.1    ✓ tibble    3.2.1
✓ lubridate 1.9.4    ✓ tidyrr    1.3.1
✓ purrr    1.0.2

— Conflicts ————— tidyverse_conflicts() —
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(gapminder) # some data
```

# Nearly done with the scaffolding

- ✓ Thought about elements of visualization
- ✓ Gotten oriented to R and RStudio
- ✓ Knitted a document
- ✓ Written a bit of `ggplot` code

# Nearly done with the scaffolding

- Thought about elements of visualization
- Gotten oriented to R and RStudio
- Knitted a document
- Written a bit of `ggplot` code
- Get my data in to R
- Make a plot with it



Feed ggplot **tidy** data

FEED ME

# What is tidy data?

gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

Tidy data

# What is tidy data?

gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

Tidy data is in *long* format

# Every column is a single variable

country	year	cases	population
Afghanistan	1999	745	10367071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	21166	128028583

variables

# Every row is a single observation

country	year	cases	population
Afghanistan	1995	745	1898790
Afghanistan	2000	2000	2000000
Burkina Faso	1995	87787	17200000
Burkina Faso	2000	86400	17400100
China	1995	212200	12720102
China	2000	218700	12804200

observations

# Every cell is a single value

country	year	cases	population
Afghanistan	1990	715	19981071
Afghanistan	2000	2666	20591360
Brazil	1990	37737	172001362
Brazil	2000	80488	174504898
China	1990	212253	1272919272
China	2000	216766	1280426583

values

# Get your data into long format

Very, *very* often, the solution to some data-wrangling or data visualization problem in a Tidyverse-focused workflow is:

# Get your data into long format

Very, *very* often, the solution to some data-wrangling or data visualization problem in a Tidyverse-focused workflow is:

**First, get the data into long format**

**Then do the thing you want.**

# Untidy data exists for good reasons

Storing and printing data in long format entails a lot of *repetition*:

```
library(palmerpenguins)
penguins %>
  group_by(species, island, year) %>
  summarize(bill = round(mean(bill_length_mm, na.rm = TRUE), 2)) %>
  tinytable::tt()
```

species	island	year	bill
Adelie	Biscoe	2007	38.32
Adelie	Biscoe	2008	38.70
Adelie	Biscoe	2009	39.69
Adelie	Dream	2007	39.10
Adelie	Dream	2008	38.19
Adelie	Dream	2009	38.15
Adelie	Torgersen	2007	38.80
Adelie	Torgersen	2008	38.77
Adelie	Torgersen	2009	39.31
Chinstrap	Dream	2007	48.72
Chinstrap	Dream	2008	48.70
Chinstrap	Dream	2009	49.05
Gentoo	Biscoe	2007	47.01
Gentoo	Biscoe	2008	46.94
Gentoo	Biscoe	2009	48.50

# Untidy data exists for good reasons

A wide format is *easier* and *more efficient* to read in print:

```
penguins %>%  
  group_by(species, island, year) %>%  
  summarize(bill = round(mean(bill_length_mm, na.rm = TRUE), 2)) %>%  
  pivot_wider(names_from = year, values_from = bill) %>%  
  tinytable::tt()
```

species	island	2007	2008	2009
Adelie	Biscoe	38.32	38.70	39.69
Adelie	Dream	39.10	38.19	38.15
Adelie	Torgersen	38.80	38.77	39.31
Chinstrap	Dream	48.72	48.70	49.05
Gentoo	Biscoe	47.01	46.94	48.50

# But also for less good reasons

State																		
A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q			
State	CD#	2018 Cook PVI Score	2018 Winner	Party	Dem Votes	GOP Votes	Other Votes	Dem %	GOP %	Other %	Dem Margin	2016 Clinton Margin	Swing vs. 2016 Prez	Raw Votes vs. 2016	Final?			
<b>New House Breakdown: 235D, 199R, 1 Not Certified</b>																		
Compiled by: David Wasserman & Ally Flinn, Cook Political Report. @Redistrict/@CookPolitical. <i>Italics</i> denotes freshman, <b>Bold</b> denotes party change.																		
Alabama	1	R+15	Bradley Byrne	R	89,226	153,228	163	36.8%	63.2%	0.1%	-26.4%	-29.2%	2.8%	79.3%	x			
Alabama	2	R+16	Martha Roby	R	86,931	138,879	420	38.4%	61.4%	0.2%	-23.0%	-31.7%	8.7%	78.7%	x			
Alabama	3	R+16	Mike Rogers	R	83,996	147,770	149	36.2%	63.7%	0.1%	-27.5%	-33.0%	5.5%	79.6%	x			
Alabama	4	R+30	Robert Aderholt	R	46,492	184,255	222	20.1%	79.8%	0.1%	-59.6%	-62.5%	2.9%	78.9%	x			
Alabama	5	R+18	Mo Brooks	R	101,388	159,063	222	38.9%	61.0%	0.1%	-22.1%	-32.9%	10.8%	82.8%	x			
Alabama	6	R+26	Gary Palmer	R	85,644	192,542	142	30.8%	69.2%	0.1%	-38.4%	-43.8%	5.4%	82.8%	x			
Alabama	7	D+20	Terri Sewell	D	185,010	0	4,153	97.8%	0.0%	2.2%	97.8%	41.2%	N/A	64.2%	x			
Alaska	AL	R+9	Don Young	R	131,199	149,779	1,188	46.5%	53.1%	0.4%	-6.6%	-14.7%	8.1%	88.6%	x			
Arizona	1	R+2	Tom O'Halleran	D	143,240	122,784	65	53.8%	46.1%	0.0%	7.7%	-1.1%	8.8%	92.0%	x			
Arizona	2	R+1	<i>Ann Kirkpatrick</i>	D	161,000	133,102	50	54.7%	45.2%	0.0%	9.5%	4.8%	4.7%	91.5%	x			
Arizona	3	D+13	Raul Grijalva	D	114,650	64,868	0	63.9%	36.1%	0.0%	27.7%	29.5%	-1.8%	84.8%	x			
Arizona	4	R+21	Paul Gosar	R	84,521	188,842	3,672	30.5%	68.2%	1.3%	-37.7%	-39.4%	1.7%	91.1%	x			
Arizona	5	R+15	Andy Biggs	R	127,027	186,037	0	40.6%	59.4%	0.0%	-18.8%	-20.5%	1.7%	91.7%	x			
Arizona	6	R+9	David Schweikert	R	140,559	173,140	0	44.8%	55.2%	0.0%	-10.4%	-9.8%	-0.6%	91.2%	x			
Arizona	7	D+23	Ruben Gallego	D	113,044	301	18,706	85.6%	0.2%	14.2%	85.4%	48.3%	N/A	79.0%	x			
Arizona	8	R+13	Debbie Lesko	R	135,569	168,835	13	44.5%	55.5%	0.0%	-10.9%	-20.8%	9.9%	91.5%	x			
Arizona	9	D+4	<i>Greg Stanton</i>	D	159,583	101,662	0	61.1%	38.9%	0.0%	22.2%	15.9%	6.3%	90.0%	x			
Arkansas	1	R+17	Rick Crawford	R	57,907	138,757	4,581	28.8%	68.9%	2.3%	-40.2%	-34.8%	-5.4%	77.2%	x			
Arkansas	2	R+7	French Hill	R	116,135	132,125	5,193	45.8%	52.1%	2.0%	-6.3%	-10.7%	4.4%	82.6%	x			
Arkansas	3	R+19	Steve Womack	R	74,952	148,717	6,039	32.6%	64.7%	2.6%	-32.1%	-31.4%	-0.7%	78.6%	x			
Arkansas	4	R+17	Bruce Westerman	R	63,984	136,740	4,168	31.2%	66.7%	2.0%	-35.5%	-32.8%	-2.7%	75.7%	x			
California	1	R+11	Doug LaMalfa	R	131,506	160,006	0	45.1%	54.9%	0.0%	-9.8%	-19.4%	9.6%	91.6%				
California	2	D+22	Jared Huffman	D	243,051	72,541	0	77.0%	23.0%	0.0%	54.0%	45.2%	8.8%	90.5%				
California	3	D+5	John Garamendi	D	132,983	96,106	0	58.0%	42.0%	0.0%	16.1%	12.5%	3.6%	86.8%				
California	4	R+10	Tom McClintock	R	156,253	184,401	0	45.9%	54.1%	0.0%	-8.3%	-14.5%	6.2%	94.6%				
California	5	D+21	Mike Thompson	D	203,012	0	53,836	79.0%	0.0%	21.0%	79.0%	44.6%	N/A	83.8%				
California	6	D+21	Doris Matsui	D	201,939	0	0	100.0%	0.0%	0.0%	100.0%	44.0%	N/A	81.4%				
California	7	D+3	Ami Bera	D	155,016	126,601	0	55.0%	45.0%	0.0%	10.1%	11.2%	-1.1%	91.0%				
California	8	R+9	Paul Cook	R	0	170,785	0	0.0%	100.0%	0.0%	-100.0%	-15.1%	N/A	73.3%				
California	9	D+8	Jerry McNerney	D	113,240	87,263	0	56.5%	43.5%	0.0%	13.0%	18.2%	-5.2%	82.4%				
California	10	EVEN	John Garamendi	D	115,666	105,810	0	55.0%	45.0%	0.0%	10.1%	11.2%	-1.1%	91.0%				

Spot the untidiness

# But also for less good reasons

State		2018 Cook PVI Score		2018 Winner		Party		Dem Votes	GOP Votes	Other Votes	Dem %	GOP %	Other %	Dem Margin	2016 Clinton Margin	Swing vs. 2016 Prez	Raw Votes vs. 2016	Final?	
State	CD#							D	60,619,428	50,896,244	1,978,795	53.4%	44.8%	1.7%	8.6%	2.1%	6.5%	83.3%	
<b>New House Breakdown: 235D, 199R, 1 Not Certified</b>																			
Compiled by: David Wasserman & Ally Flinn, Cook Political Report. @Redistrict/@CookPolitical. Italics denotes freshman. <b>Bold</b> denotes party change.																			
Alabama	1	R+15	Bradley Byrne	R	89,226	153,228	163	36.8%	63.2%	0.1%	-26.4%	-29.2%	2.8%	79.3%	x				
Alabama	2	R+16	Martha Roby	R	86,931	138,879	420	38.4%	61.4%	0.2%	-23.0%	-31.7%	8.7%	78.7%	x				
Alabama	3	R+16	Mike Rogers	R	83,996	147,770	149	36.2%	63.7%	0.1%	-27.5%	-33.0%	5.5%	79.6%	x				
Alabama	4	R+30	Robert Aderholt	R	46,492	184,255	222	20.1%	79.8%	0.1%	-59.6%	-62.5%	2.9%	78.9%	x				
Alabama	5	R+18	Mo Brooks	R	101,388	159,063	222	38.9%	61.0%	0.1%	-22.1%	-32.9%	10.8%	82.8%	x				
Alabama	6	R+26	Gary Palmer	R	85,644	192,542	142	30.8%	69.2%	0.1%	-38.4%	-43.8%	5.4%	82.8%	x				
Alabama	7	D+20	Terri Sewell	D	185,010	0	4,153	97.8%	0.0%	2.2%	97.8%	41.2%	N/A	64.2%	x				
Alaska	AL	R+9	Don Young	R	131,199	149,779	1,188	46.5%	53.1%	0.4%	-6.6%	-14.7%	8.1%	88.6%	x				
Arizona	1	R+2	Tom O'Halleran	D	143,240	122,784	65	53.8%	46.1%	0.0%	7.7%	-1.1%	8.8%	92.0%	x				
Arizona	2	R+1	<i>Ann Kirkpatrick</i>	D	161,000	133,102	50	54.7%	45.2%	0.0%	9.5%	4.8%	4.7%	91.5%	x				
Arizona	3	D+13	Raul Grijalva	D	114,650	64,868	0	63.9%	36.1%	0.0%	-27.7%	29.5%	-1.8%	84.8%	x				
Arizona	4	R+21	Paul Gosar	R	84,521	188,842	3,672	30.5%	68.2%	1.3%	-37.7%	-39.4%	1.7%	91.1%	x				
Arizona	5	R+15	Andy Biggs	R	127,027	186,037	0	40.6%	59.4%	0.0%	-18.8%	-20.5%	1.7%	91.7%	x				
Arizona	6	R+9	David Schweikert	R	140,559	173,140	0	44.8%	55.2%	0.0%	-10.4%	-9.8%	-0.6%	91.2%	x				
Arizona	7	D+23	Ruben Gallego	D	113,044	301	18,706	85.6%	0.2%	14.2%	85.4%	48.3%	N/A	79.0%	x				
Arizona	8	R+13	Debbie Lesko	R	135,569	168,835	13	44.5%	55.5%	0.0%	-10.9%	-20.8%	9.9%	91.5%	x				
Arizona	9	D+4	Greg Stanton	D	159,583	101,662	0	61.1%	38.9%	0.0%	22.2%	15.9%	6.3%	90.0%	x				
Arkansas	1	R+17	Rick Crawford	R	57,907	138,757	4,581	28.8%	68.9%	2.3%	-40.2%	-34.8%	-5.4%	77.2%	x				
Arkansas	2	R+7	French Hill	R	116,135	132,125	5,193	45.8%	52.1%	2.0%	-6.3%	-10.7%	4.4%	82.6%	x				
Arkansas	3	R+19	Steve Womack	R	74,952	148,717	6,039	32.6%	64.7%	2.6%	-32.1%	-31.4%	-0.7%	78.6%	x				
Arkansas	4	R+17	Bruce Westerman	R	63,984	136,740	4,168	31.2%	66.7%	2.0%	-35.5%	-32.8%	-2.7%	75.7%	x				
California	1	R+11	Doug LaMalfa	R	131,506	160,006	0	45.1%	54.9%	0.0%	-9.8%	-19.4%	9.6%	91.6%	x				
California	2	D+22	Jared Huffman	D	243,051	72,541	0	77.0%	23.0%	0.0%	54.0%	45.2%	8.8%	90.5%	x				
California	3	D+5	John Garamendi	D	132,983	96,106	0	58.0%	42.0%	0.0%	16.1%	12.5%	3.6%	86.8%	x				
California	4	R+10	Tom McClintock	R	156,253	184,401	0	45.9%	54.1%	0.0%	-8.3%	-14.5%	6.2%	94.6%	x				
California	5	D+21	Mike Thompson	D	203,012	0	53,836	79.0%	0.0%	21.0%	79.0%	44.6%	N/A	83.8%	x				
California	6	D+21	Doris Matsui	D	201,939	0	0	100.0%	0.0%	0.0%	100.0%	44.0%	N/A	81.4%	x				
California	7	D+3	Ami Bera	D	155,016	126,601	0	55.0%	45.0%	0.0%	10.1%	11.2%	-1.1%	91.0%	x				
California	8	R+9	Paul Cook	R	0	170,785	0	0.0%	100.0%	0.0%	-100.0%	-15.1%	N/A	73.3%	x				
California	9	D+8	Jerry McNerney	D	113,240	87,263	0	56.5%	43.5%	0.0%	13.0%	18.2%	-5.2%	82.4%	x				

Spot the untidiness

😡 More than one header row

😡 Mixed data types in some columns

💀 Color and typography used to encode variables and their values

# Fix it **before** you import it

Prevention is better than cure!

An excellent article by Karl Broman and Kara Woo:

Broman KW, Woo KH (2018) “Data Organization in Spreadsheets.” *The American Statistician* 78:2–10

THE AMERICAN STATISTICIAN  
2018, VOL. 72, NO. 1, 2–10  
<https://doi.org/10.1080/00031305.2017.1375989>

 Taylor & Francis  
Taylor & Francis Group

 OPEN ACCESS  Check for updates

## Data Organization in Spreadsheets

Karl W. Broman<sup>a</sup> and Kara H. Woo<sup>b</sup>

<sup>a</sup>Department of Biostatistics & Medical Informatics, University of Wisconsin-Madison, Madison, WI; <sup>b</sup>Information School, University of Washington, Seattle, WA

### ABSTRACT

Spreadsheets are widely used software tools for data entry, storage, analysis, and visualization. Focusing on the data entry and storage aspects, this article offers practical recommendations for organizing spreadsheet data to reduce errors and ease later analyses. The basic principles are: be consistent, write dates like YYYY-MM-DD, do not leave any cells empty, put just one thing in a cell, organize the data as a single rectangle (with subjects as rows and variables as columns, and with a single header row), create a data dictionary, do not include calculations in the raw data files, do not use font color or highlighting as data, choose good names for things, make backups, use data validation to avoid data entry errors, and save the data in plain text files.

### ARTICLE HISTORY

Received June 2017  
Revised August 2017

### KEYWORDS

Data management; Data organization; Microsoft Excel; Spreadsheets

Data organization in spreadsheets

# The most common `tidyverse` operation

*Pivoting* from wide to long:

```
edu
```

```
# A tibble: 366 × 11
  age    sex   year total elem4 elem8   hs3   hs4 coll3 coll4 median
  <chr> <chr> <int> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl>
1 25-34 Male   2016 21845    116    468 1427  6386  6015  7432     NA
2 25-34 Male   2015 21427    166    488 1584  6198  5920  7071     NA
3 25-34 Male   2014 21217    151    512 1611  6323  5910  6710     NA
4 25-34 Male   2013 20816    161    582 1747  6058  5749  6519     NA
5 25-34 Male   2012 20464    161    579 1707  6127  5619  6270     NA
6 25-34 Male   2011 20985    190    657 1791  6444  5750  6151     NA
7 25-34 Male   2010 20689    186    641 1866  6458  5587  5951     NA
8 25-34 Male   2009 20440    184    695 1806  6495  5508  5752     NA
9 25-34 Male   2008 20210    172    714 1874  6356  5277  5816     NA
10 25-34 Male  2007 20024    246    757 1930  6361  5137  5593    NA
# i 356 more rows
```

Here, a “Level of Schooling Attained” variable is spread across the columns, from `elem4` to `coll4`. We need a *key* column called “education” with the various levels of schooling, and a corresponding *value* column containing the counts.

# Wide to long with `pivot_longer()`

We're going to put the columns `elem4:coll4` into a new column, creating a new categorical measure named `education`. The numbers currently under each column will become a new `value` column corresponding to that level of education.

```
edu >  
pivot_longer(elem4:coll4, names_to = "education")
```

```
# A tibble: 2,196 × 7  
  age   sex    year total median education value  
  <chr> <chr> <int> <int>  <dbl> <chr>     <dbl>  
1 25-34 Male    2016 21845     NA elem4      116  
2 25-34 Male    2016 21845     NA elem8      468  
3 25-34 Male    2016 21845     NA hs3       1427  
4 25-34 Male    2016 21845     NA hs4       6386  
5 25-34 Male    2016 21845     NA coll3     6015  
6 25-34 Male    2016 21845     NA coll4     7432  
7 25-34 Male    2015 21427     NA elem4      166  
8 25-34 Male    2015 21427     NA elem8      488  
9 25-34 Male    2015 21427     NA hs3       1584  
10 25-34 Male   2015 21427     NA hs4       6198  
# i 2,186 more rows
```

# Wide to long with `pivot_longer()`

We can name the value column to whatever we like. Here it's a number of people.

```
edu >
pivot_longer(elem4:coll4,
             names_to = "education",
             values_to = "n")

# A tibble: 2,196 × 7
  age   sex   year total median education     n
  <chr> <chr> <int> <int>  <dbl> <chr>     <dbl>
1 25-34 Male    2016 21845     NA elem4      116
2 25-34 Male    2016 21845     NA elem8      468
3 25-34 Male    2016 21845     NA hs3       1427
4 25-34 Male    2016 21845     NA hs4       6386
5 25-34 Male    2016 21845     NA coll3     6015
6 25-34 Male    2016 21845     NA coll4     7432
7 25-34 Male    2015 21427     NA elem4      166
8 25-34 Male    2015 21427     NA elem8      488
9 25-34 Male    2015 21427     NA hs3       1584
10 25-34 Male   2015 21427     NA hs4       6198
# i 2,186 more rows
```

# How to get your own data into R

# Reading in CSV files

Base R has `read.csv()`

Corresponding tidyverse “underscored” version: `read_csv()`.

It is pickier and more talkative than the Base R version. Use it instead.

# Where's my data? Using `here()`

If we're loading a file, it's coming from *somewhere*.

If it's a file on our hard drive somewhere, we will need to interact with the file system. We should try to do this in a way that avoids *absolute* file paths.

```
# This is not portable!
df ← read_csv("/Users/kjhealy/Documents/data/misc/project/data/mydata.csv")
```

We should also do it in a way that is *platform independent*.

This makes it easier to share your work, move it around, etc. Projects should be self-contained.

# Where's my data? Using `here()`

The `here` package, and `here()` function builds paths relative to the top level of your R project.

```
here() # this path will be different for you  
[1] "/Users/kjhealy/Documents/courses/data_visualization"
```

# Where's the data? Using `here()`

This seminar's files all live in an RStudio project. It looks like this:

```
/Users/kjhealy/Documents/courses/data_visualization
├── 00_dummy_files
├── LICENSE
├── Makefile
├── README.md
├── README.qmd
├── _extensions
├── _freeze
├── _quarto.yml
├── _site
├── _targets
└── _targets.R
├── _variables.yml
└── code
    └── course_notes.qmd
├── data
├── data_visualization.Rproj
├── deploy.sh
├── figures
├── html
├── index.html
├── index.qmd
└── pdf_slides
├── site_libs
└── slides
```

I want to load files from the `data` folder, but I also want *you* to be able to load them. I'm writing this from somewhere deep in the `slides` folder, but you won't be there. Also, I'm on a Mac, but you may not be.

# Where's the data? Using `here()`

So:

```
## Load the file relative to the path from the top of the project, without separators, etc
organs ← read_csv(file = here("data", "organdonation.csv"))
```

# Where's the data? Using `here()`

```
organs
```

```
# A tibble: 238 × 21
  country year donors   pop  pop.dens    gdp gdp.lag health health.lag pubhealth
  <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Austra...    NA    NA  17065  0.220 16774  16591  1300  1224    4.8
2 Austra...  1991  12.1 17284  0.223 17171  16774  1379  1300    5.4
3 Austra...  1992  12.4 17495  0.226 17914  17171  1455  1379    5.4
4 Austra...  1993  12.5 17667  0.228 18883  17914  1540  1455    5.4
5 Austra...  1994  10.2 17855  0.231 19849  18883  1626  1540    5.4
6 Austra...  1995  10.2 18072  0.233 21079  19849  1737  1626    5.5
7 Austra...  1996  10.6 18311  0.237 21923  21079  1846  1737    5.6
8 Austra...  1997  10.3 18518  0.239 22961  21923  1948  1846    5.7
9 Austra...  1998  10.5 18711  0.242 24148  22961  2077  1948    5.9
10 Austra... 1999   8.67 18926  0.244 25445  24148  2231  2077    6.1
# i 228 more rows
# i 11 more variables: roads <dbl>, cerebvas <dbl>, assault <dbl>,
# external <dbl>, txp.pop <dbl>, world <chr>, opt <chr>, consent.law <chr>,
# consent.practice <chr>, consistent <chr>, ccode <chr>
```

And there it is.

# **read\_csv()** has variants

**read\_csv()** Field separator is a comma: ,

```
organs ← read_csv(file = here("data", "organdonation.csv"))
```

**read\_csv2()** Field separator is a semicolon: ;

```
# Example only  
my_data ← read_csv2(file = here("data", "my_euro_file.csv"))
```

Both are special cases of **read\_delim()**

# Other species are also catered to

`read_tsv()` Tab separated.

`read_fwf()` Fixed-width files.

`read_log()` Log files (i.e. computer log files).

`read_lines()` Just read in lines, without trying to parse them.

# Also often useful ...

`read_table()`

For data that's separated by one (or more) columns of space.

# And for foreign file formats ...

The `haven` package provides

`read_dta()` Stata

`read_spss()` SPSS

`read_sas()` SAS

`read_xpt()` SAS Transport

Make these functions available with `library(haven)`

# You can read files remotely, too

You can give these functions local files, or they can also be pointed at URLs.

Compressed files (`.zip`, `.tar.gz`) will be automatically uncompressed.

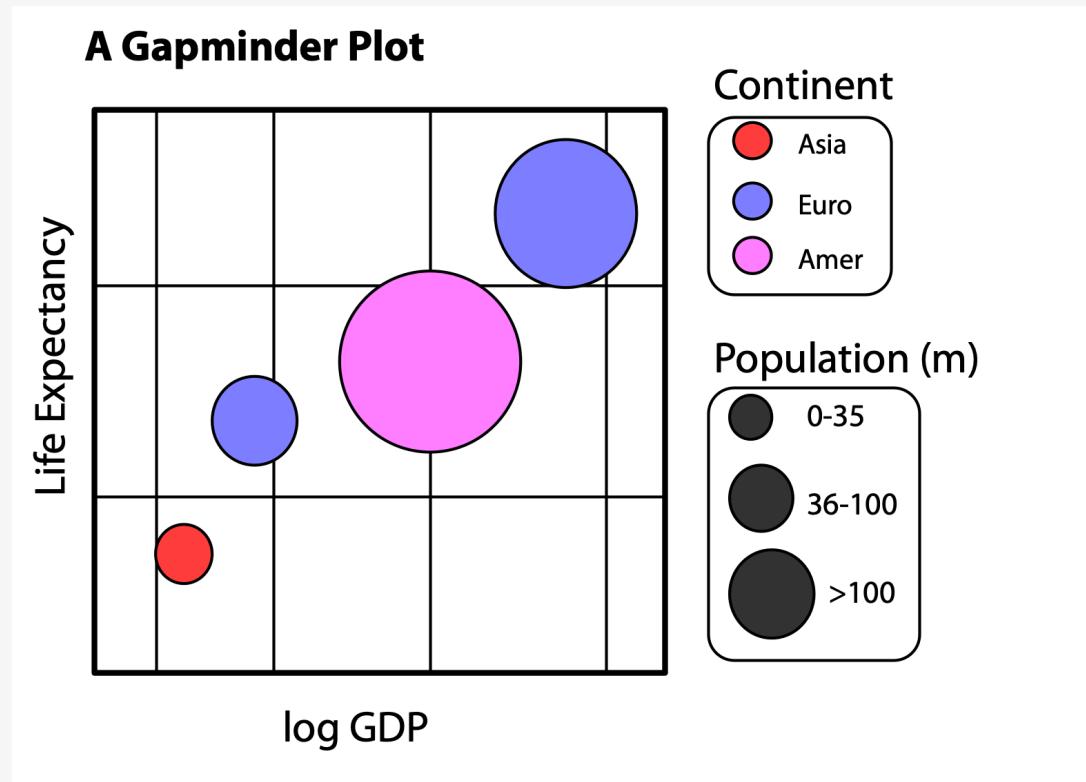
(Be careful what you download from remote locations!)

```
organ_remote ← read_csv("http://kjhealy.co/organdonation.csv")
organ_remote

# A tibble: 238 × 21
  country   year donors    pop pop.dens     gdp gdp.lag health health.lag pubhealth
  <chr>     <dbl> <dbl>    <dbl> <dbl>     <dbl> <dbl>     <dbl>     <dbl>
1 Austra...     NA    NA    17065  0.220  16774  16591  1300      1224     4.8
2 Austra...    1991  12.1   17284  0.223  17171  16774  1379      1300     5.4
3 Austra...    1992  12.4   17495  0.226  17914  17171  1455      1379     5.4
4 Austra...    1993  12.5   17667  0.228  18883  17914  1540      1455     5.4
5 Austra...    1994  10.2   17855  0.231  19849  18883  1626      1540     5.4
6 Austra...    1995  10.2   18072  0.233  21079  19849  1737      1626     5.5
7 Austra...    1996  10.6   18311  0.237  21923  21079  1846      1737     5.6
8 Austra...    1997  10.3   18518  0.239  22961  21923  1948      1846     5.7
9 Austra...    1998  10.5   18711  0.242  24148  22961  2077      1948     5.9
10 Austra...   1999   8.67  18926  0.244  25445  24148  2231      2077     6.1
# ℹ 228 more rows
# ℹ 11 more variables: roads <dbl>, cerebvas <dbl>, assault <dbl>,
#   external <dbl>, txp.pop <dbl>, world <chr>, opt <chr>, consent.law <chr>,
#   consent.practice <chr>, consistent <chr>, ccode <chr>
```

# A Plot's Components

# What we need our code to make



Data **represented** by visual elements;

like ***position*, *length*, *color*, and *size***;

Each measured on some **scale**;

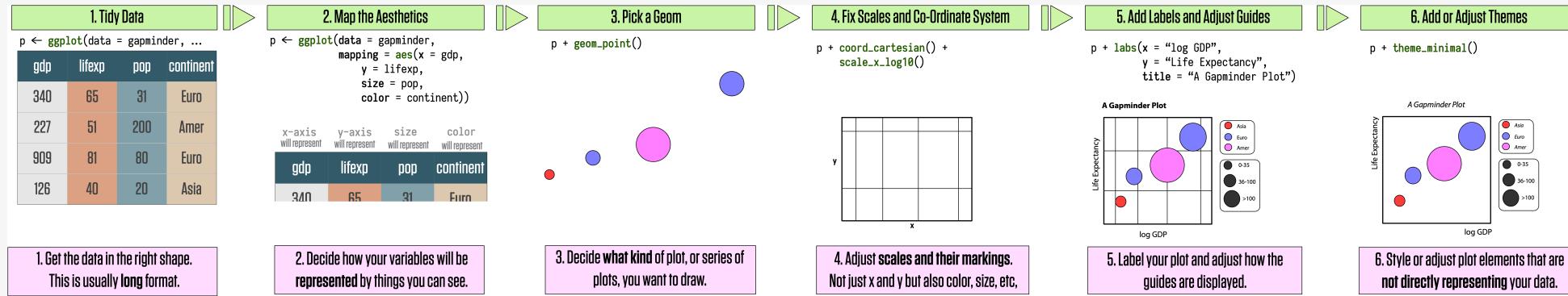
Each scale with a labeled **guide**;

With the plot itself also **titled** and labeled.

How does  
ggplot  
do this?

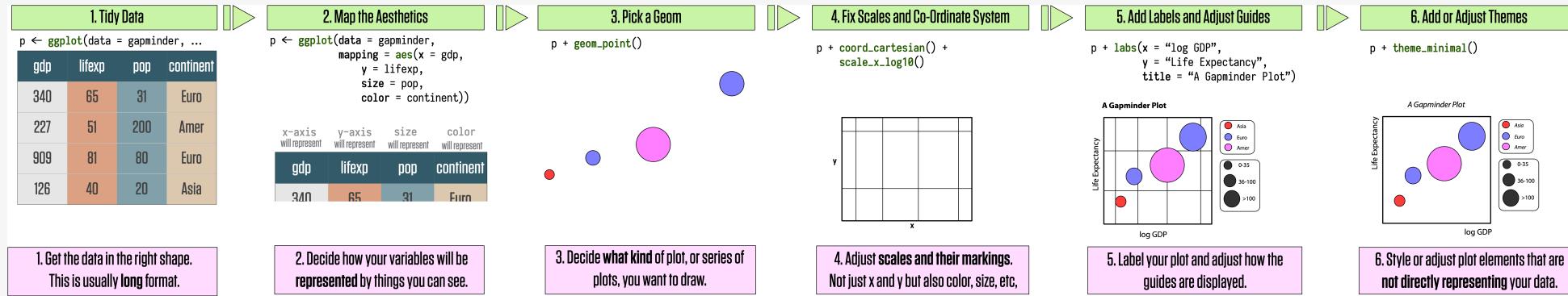
# ggplot's flow of action

# Here's the whole thing, start to finish



Flow of action

# We'll go through it step by step



Flow of action

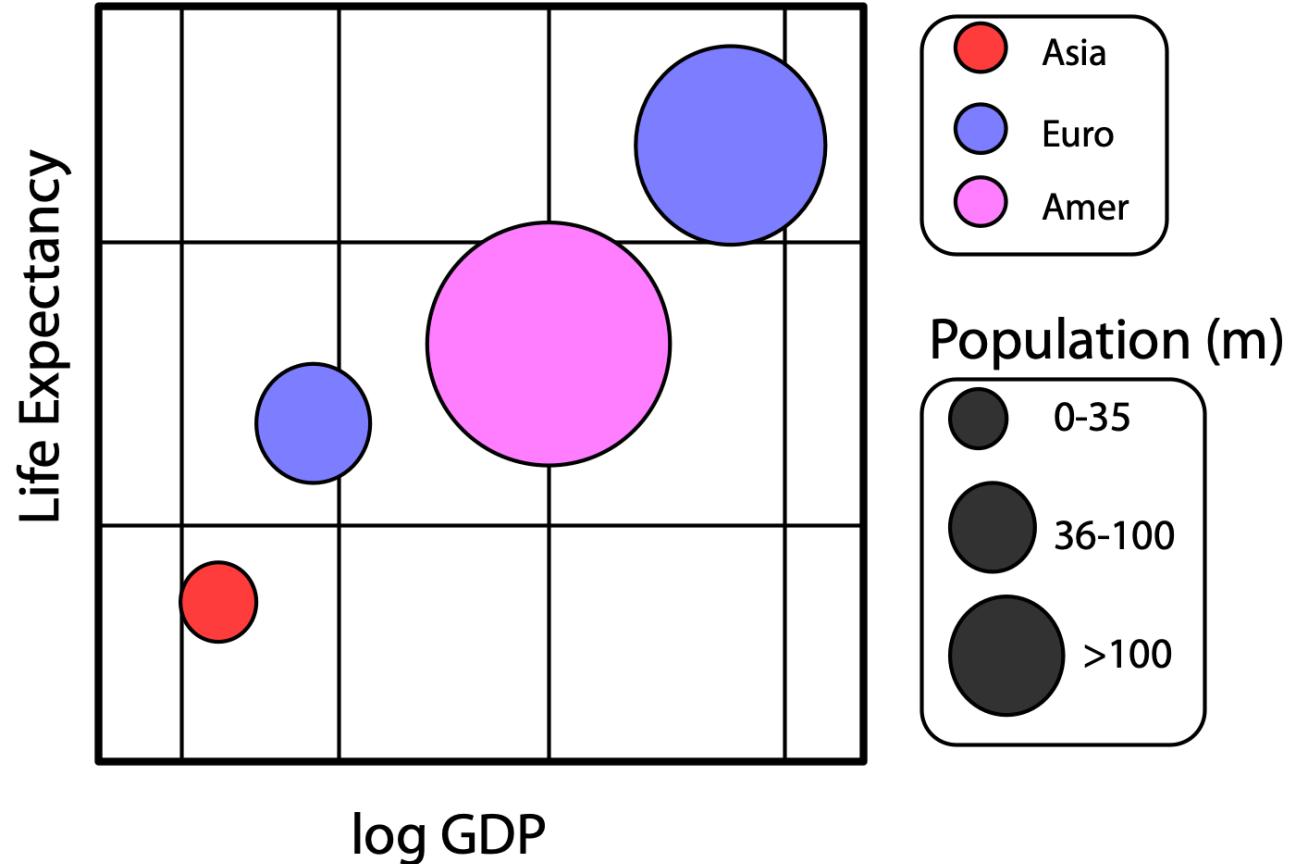
# ggplot's flow of action

gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

What we start with

# ggplot's flow of action

A Gapminder Plot



Where we're going

# ggplot's flow of action

1. Tidy Data

```
p <- ggplot(data = gapminder, ...)
```

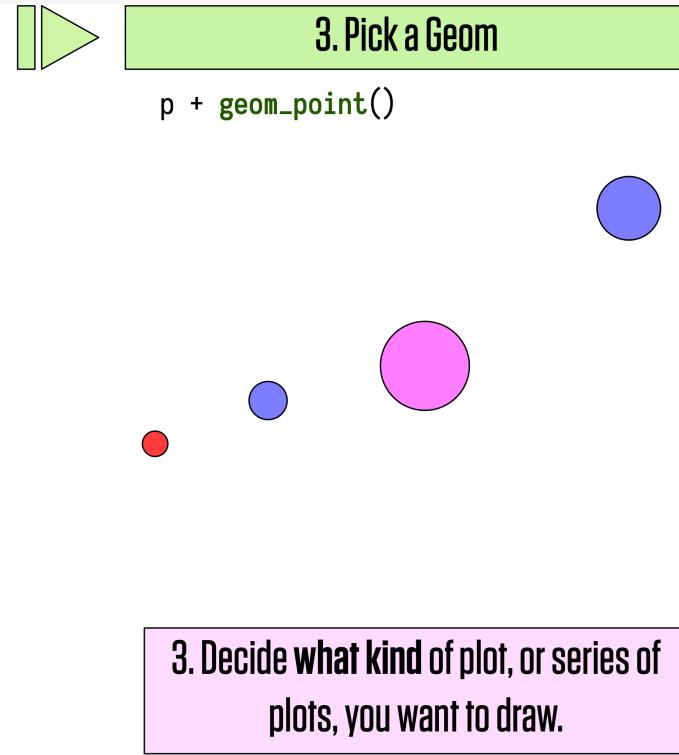
gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

2. Map the Aesthetics

```
p <- ggplot(data = gapminder,  
mapping = aes(x = gdp,  
y = lifexp,  
size = pop,  
color = continent))
```

x-axis will represent  
y-axis will represent  
size will represent  
color will represent

gdp	lifexp	pop	continent
340	65	31	Euro



1. Get the data in the right shape.  
This is usually **long** format.

2. Decide how your variables will be represented by things you can see.

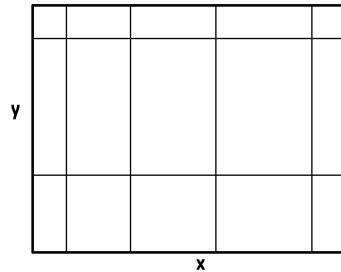
3. Decide what kind of plot, or series of plots, you want to draw.

Core steps

# ggplot's flow of action

## 4. Fix Scales and Co-Ordinate System

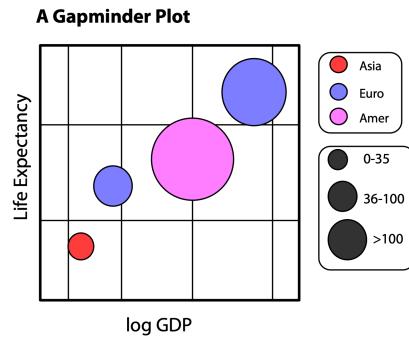
```
p + coord_cartesian() +  
  scale_x_log10()
```



**4. Adjust scales and their markings.**  
Not just x and y but also color, size, etc,

## 5. Add Labels and Adjust Guides

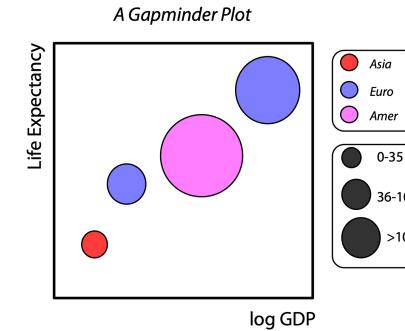
```
p + labs(x = "log GDP",  
         y = "Life Expectancy",  
         title = "A Gapminder Plot")
```



**5. Label your plot and adjust how the guides are displayed.**

## 6. Add or Adjust Themes

```
p + theme_minimal()
```



**6. Style or adjust plot elements that are not directly representing your data.**

Optional steps

# ggplot's flow of action: required

## 1. Tidy Data

```
p <- ggplot(data = gapminder, ...)
```

gdp	lifexp	pop	continent
340	65	31	Euro
227	51	200	Amer
909	81	80	Euro
126	40	20	Asia

1. Get the data in the right shape.  
This is usually **long** format.

Tidy data

# ggplot's flow of action: required

## 2. Map the Aesthetics

```
p <- ggplot(data = gapminder,  
             mapping = aes(x = gdp,  
                            y = lifexp,  
                            size = pop,  
                            color = continent))
```

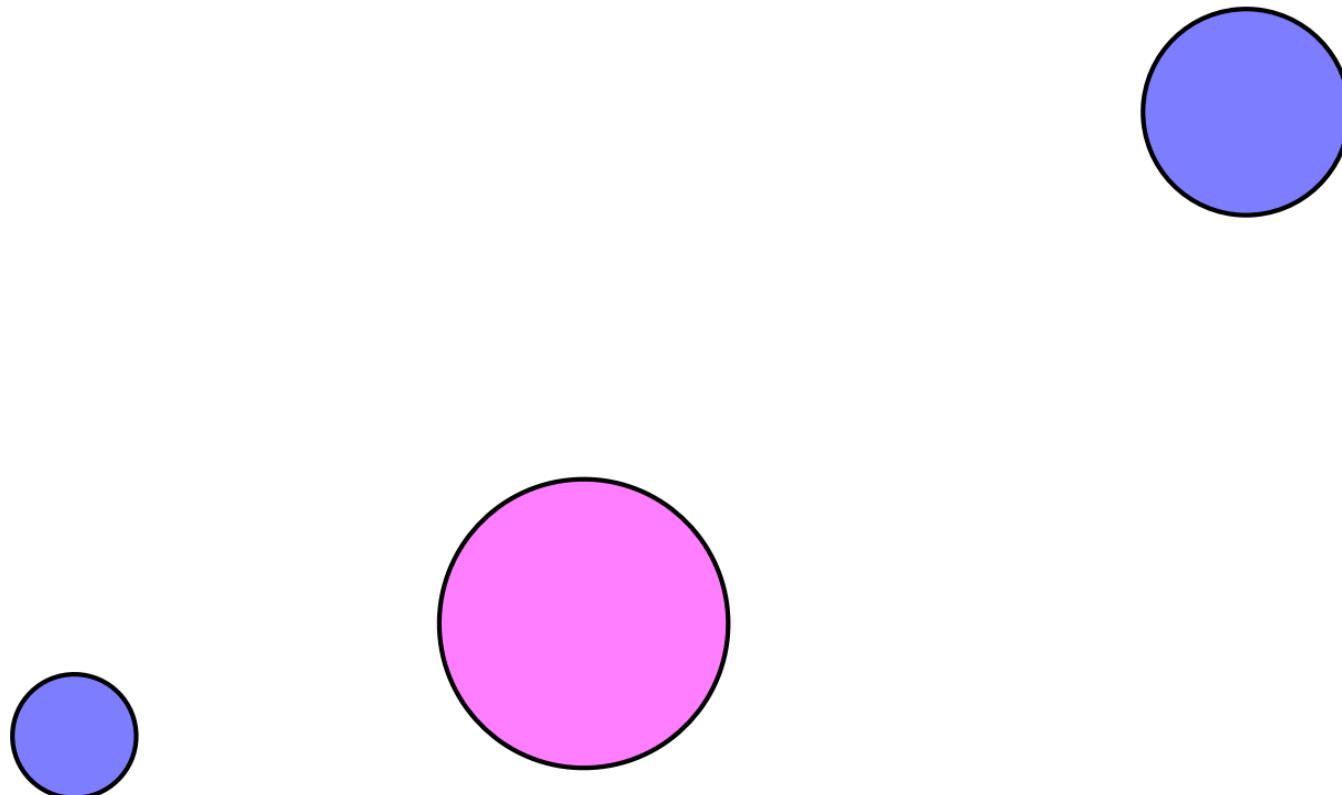
x-axis will represent	y-axis will represent	size will represent	color will represent
gdp	lifexp	pop	continent
340	65	31	Euro

2. Decide how your variables will be represented by things you can see.

# ggplot's flow of action: **required**

### 3. Pick a Geom

```
p + geom_point()
```



# Start with the data

```
gapminder
```

```
# A tibble: 1,704 × 6
  country   continent year lifeExp      pop gdpPercap
  <fct>     <fct>    <int>   <dbl>    <int>     <dbl>
1 Afghanistan Asia      1952    28.8  8425333    779.
2 Afghanistan Asia      1957    30.3  9240934    821.
3 Afghanistan Asia      1962    32.0 10267083    853.
4 Afghanistan Asia      1967    34.0 11537966    836.
5 Afghanistan Asia      1972    36.1 13079460    740.
6 Afghanistan Asia      1977    38.4 14880372    786.
7 Afghanistan Asia      1982    39.9 12881816    978.
8 Afghanistan Asia      1987    40.8 13867957    852.
9 Afghanistan Asia      1992    41.7 16317921    649.
10 Afghanistan Asia     1997    41.8 22227415    635.
# i 1,694 more rows
```

```
dim(gapminder)
```

```
[1] 1704    6
```

# Create a plot object

Data is the `gapminder` tibble.

```
p ← ggplot(data = gapminder)
```

# Map variables to aesthetics

Tell `ggplot` the variables you want represented by visual elements on the plot

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp))
```

# Map variables to aesthetics

The `mapping = aes( ... )` call links variables to things you will see on the plot.

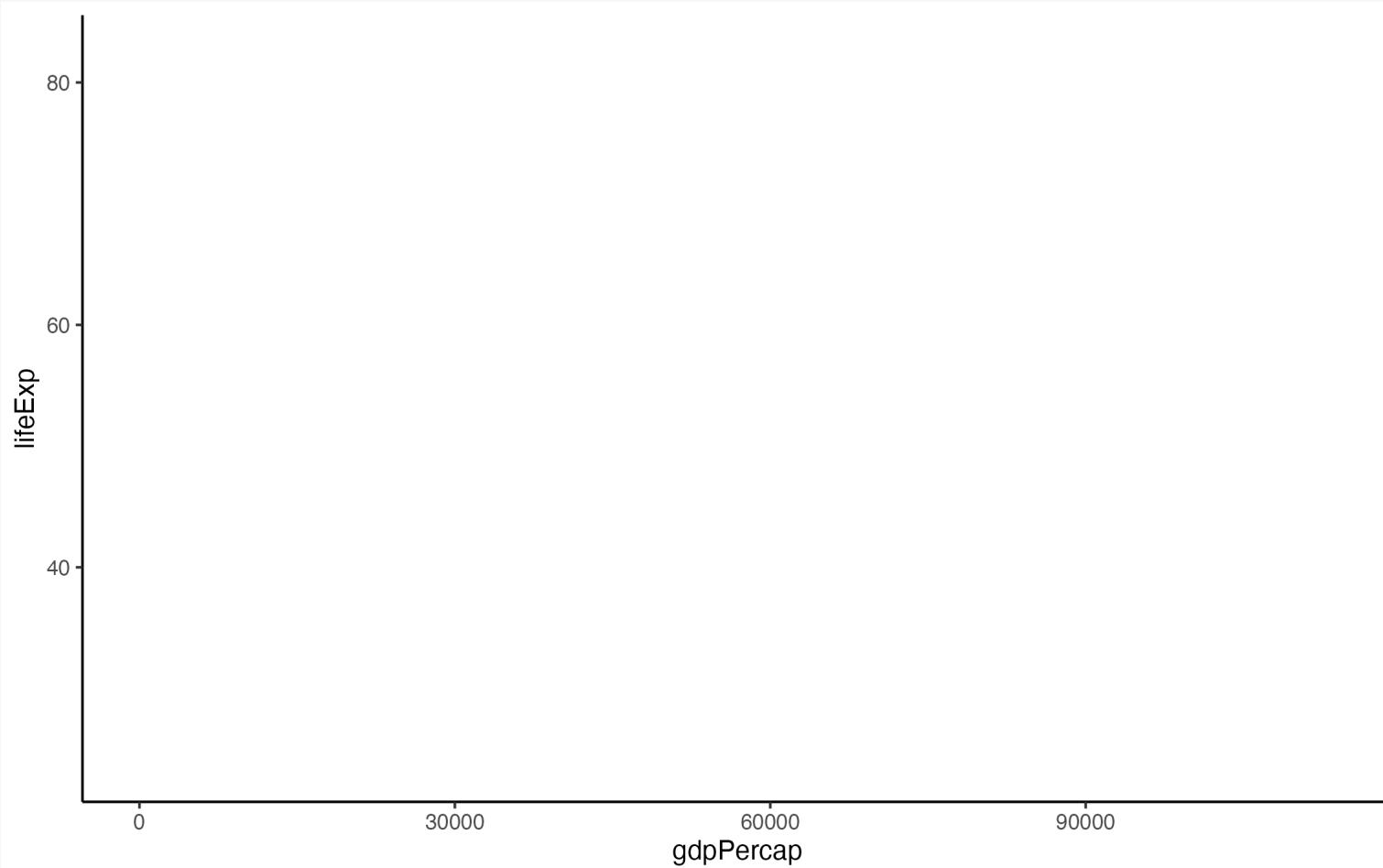
`x` and `y` represent the quantities determining position on the x and y axes.

Other aesthetic mappings can include, e.g., `color`, `shape`, `size`, and `fill`.

**Mappings** do not *directly* specify the particular, e.g., colors, shapes, or line styles that will appear on the plot. Rather, they establish ***which variables*** in the data will be represented by ***which visible elements*** on the plot.

# p has data and mappings but no geom

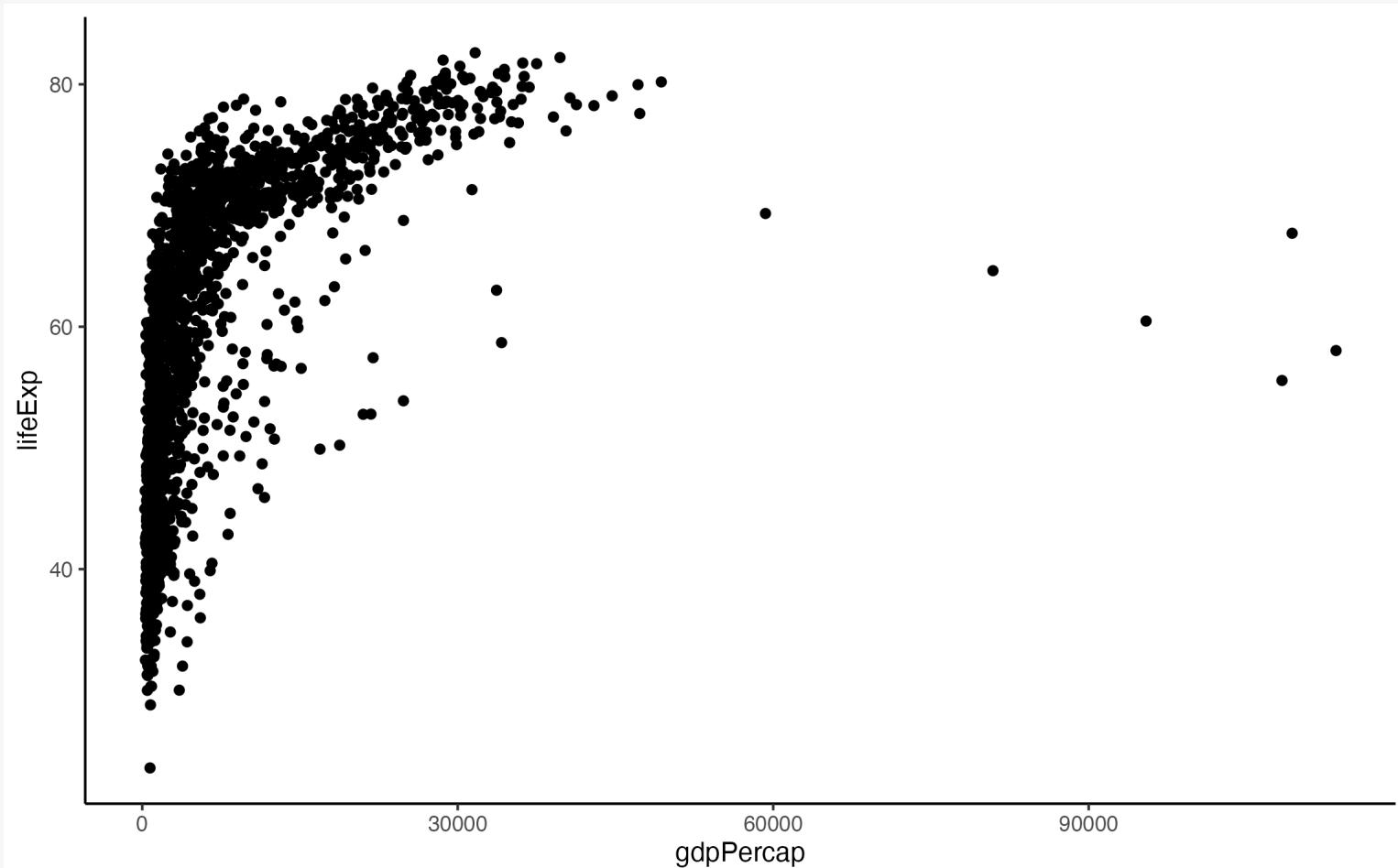
p



This empty plot has no geoms.

# Add a geom

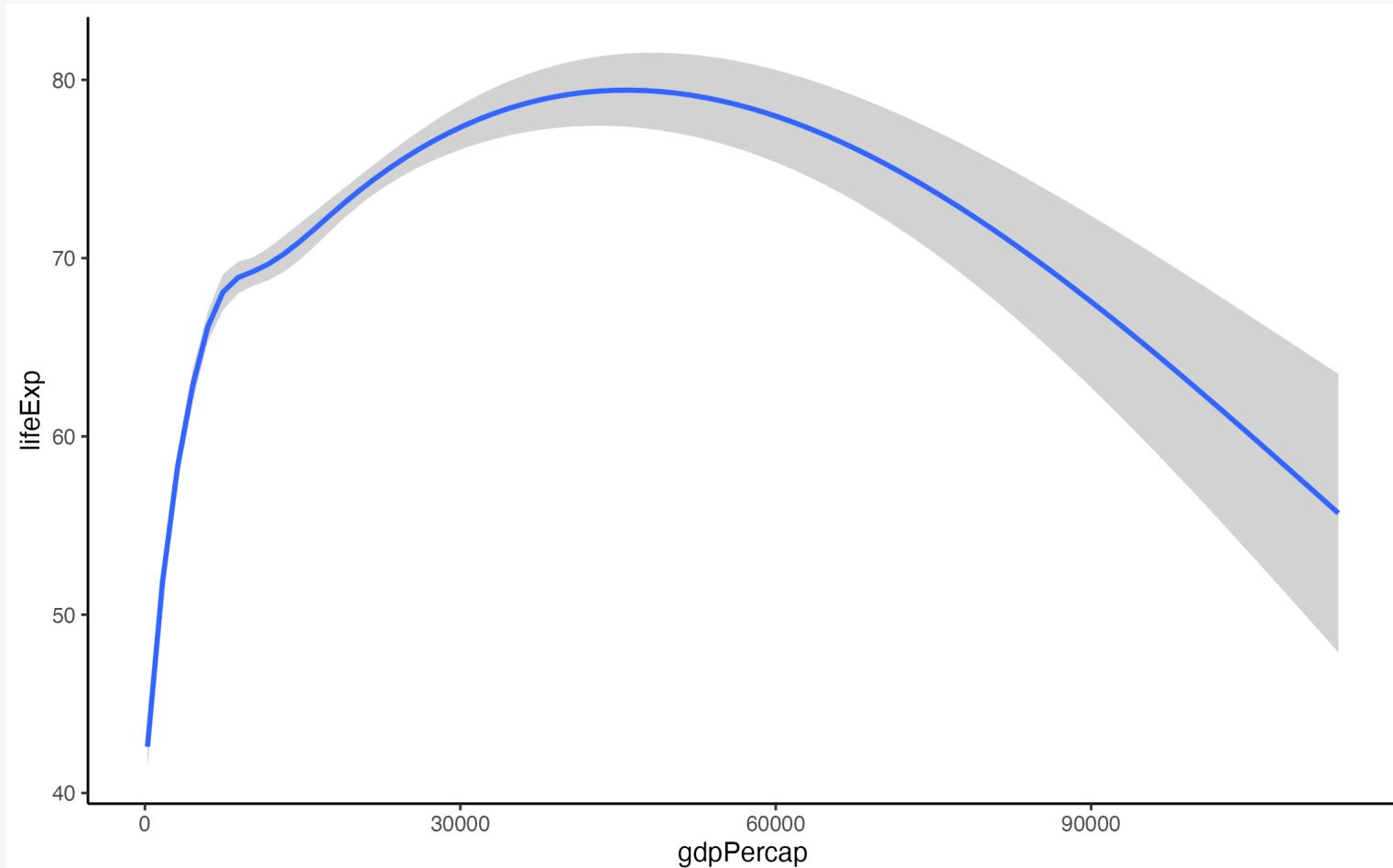
```
p + geom_point()
```



A scatterplot of Life Expectancy vs GDP

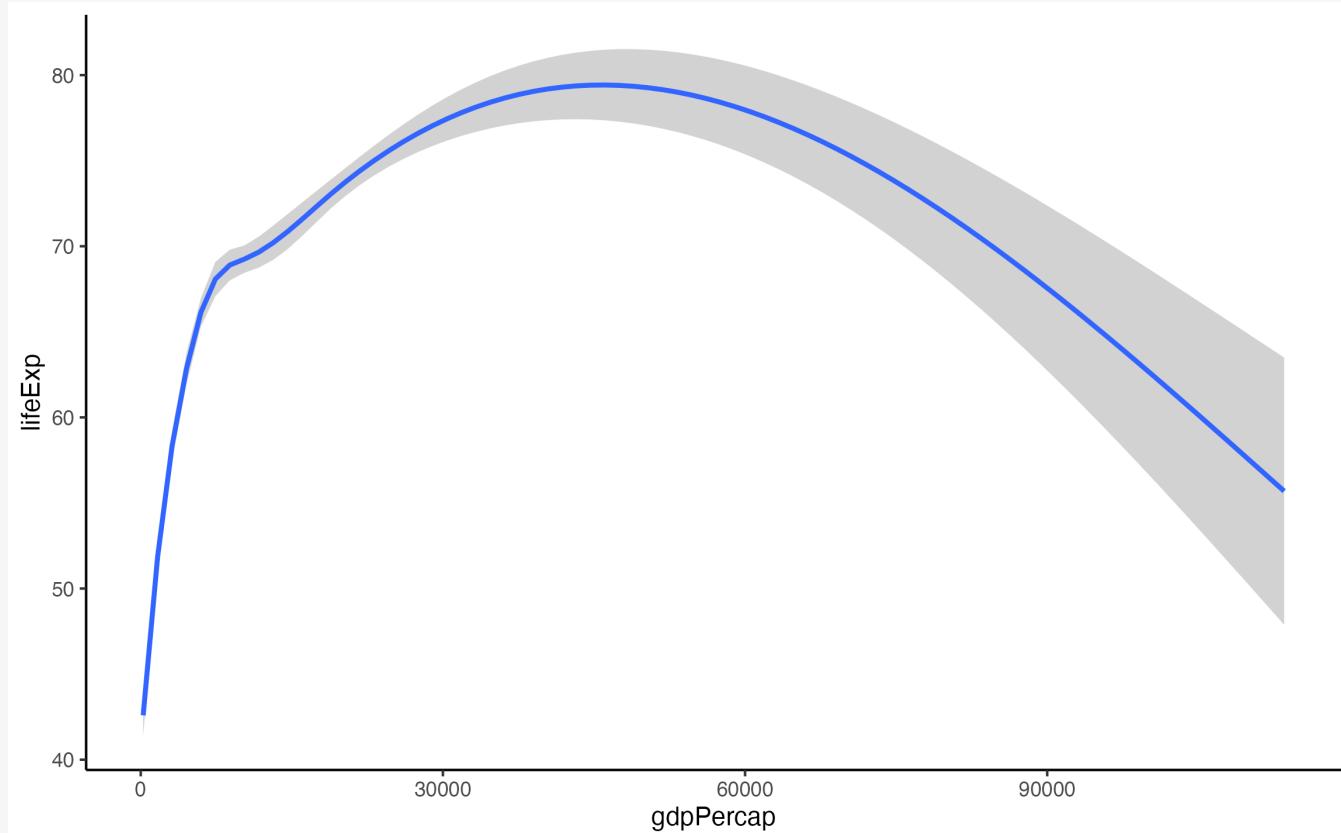
# Try a different geom

```
p + geom_smooth()
```



# Build your plots layer by layer

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



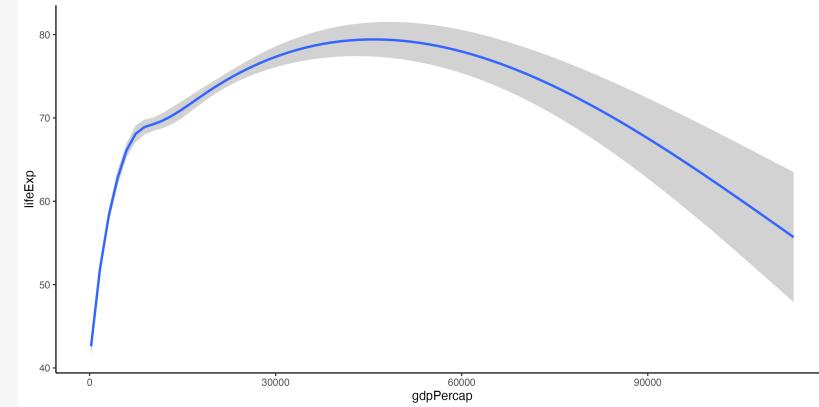
Life Expectancy vs GDP, using a smoother.

# This process is additive

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y=lifeExp))
```

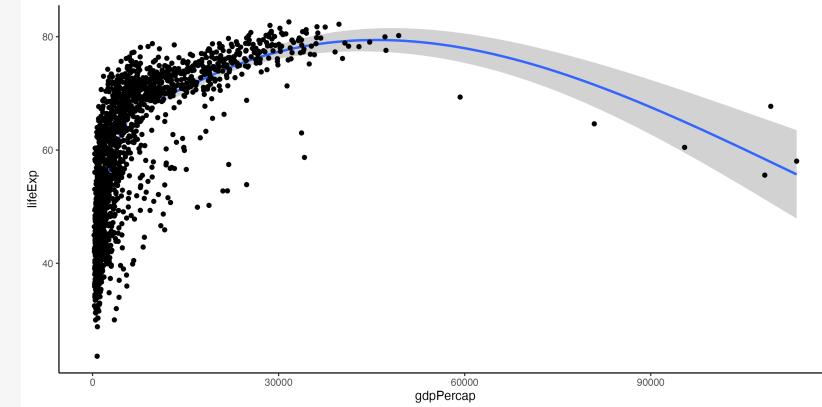
# This process is additive

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



# This process is additive

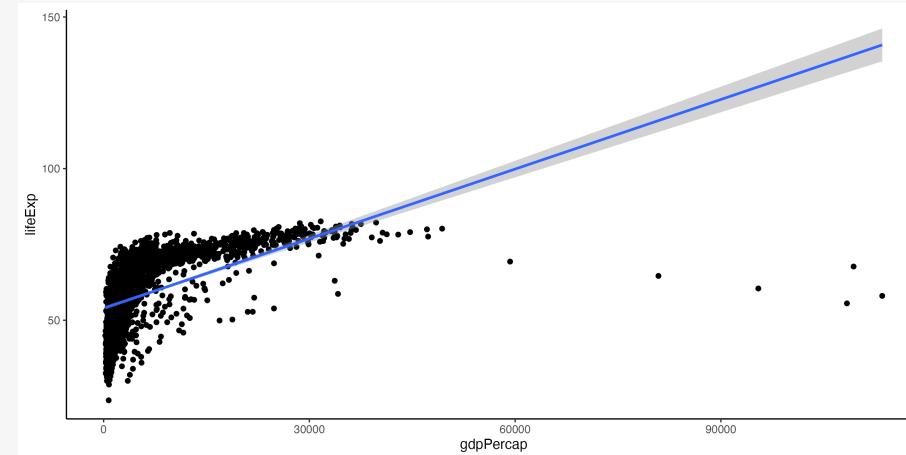
```
p <- ggplot(data = gapminder,  
             mapping = aes(x = gdpPercap,  
                            y=lifeExp))  
p + geom_smooth() +  
    geom_point()
```



# Every geom is a function

Functions take arguments

```
p <- ggplot(data = gapminder,  
             mapping = aes(x = gdpPercap,  
                           y = lifeExp))  
p + geom_point() +  
  geom_smooth(method = "lm")
```

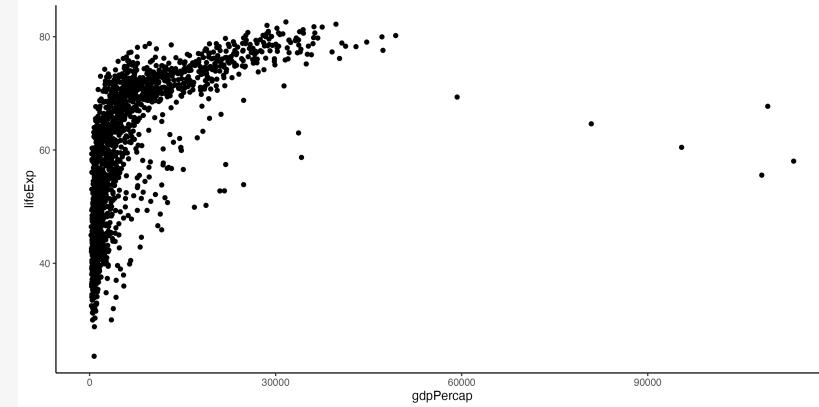


# Keep Layering

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y=lifeExp))
```

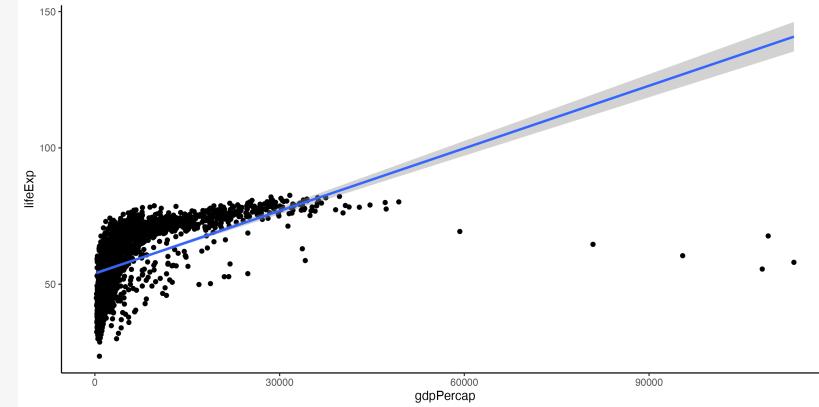
# Keep Layering

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



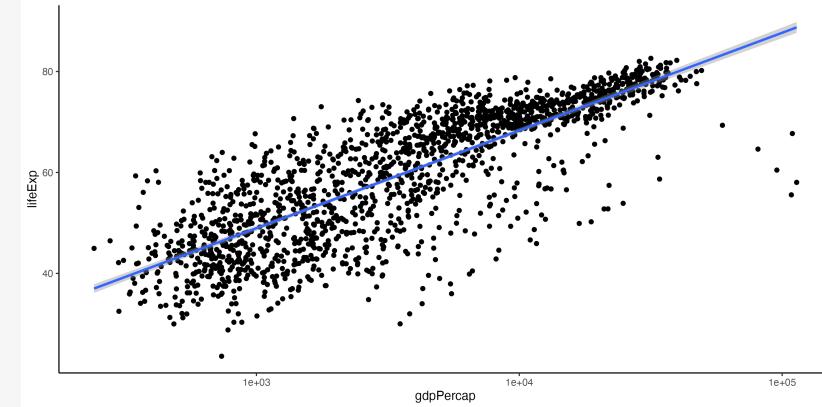
# Keep Layering

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



# Keep Layering

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y=lifeExp))  
p + geom_point() +  
  geom_smooth(method = "lm") +  
  scale_x_log10()
```

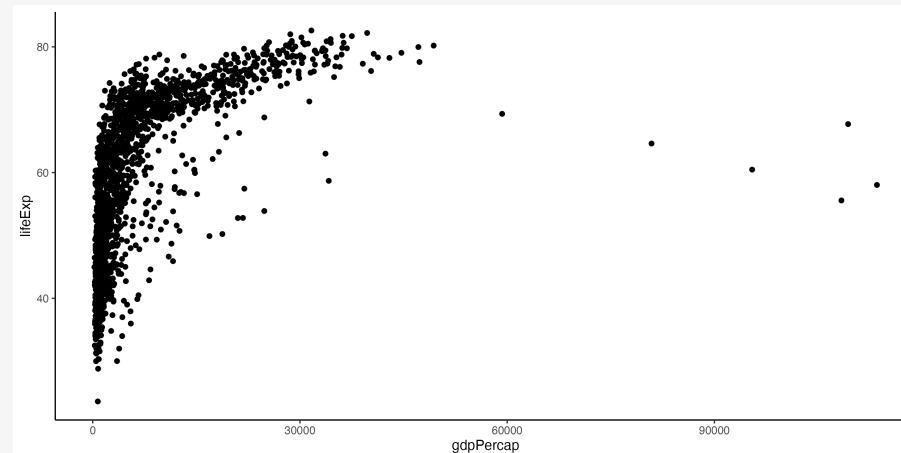


# Fix the labels

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y=lifeExp))
```

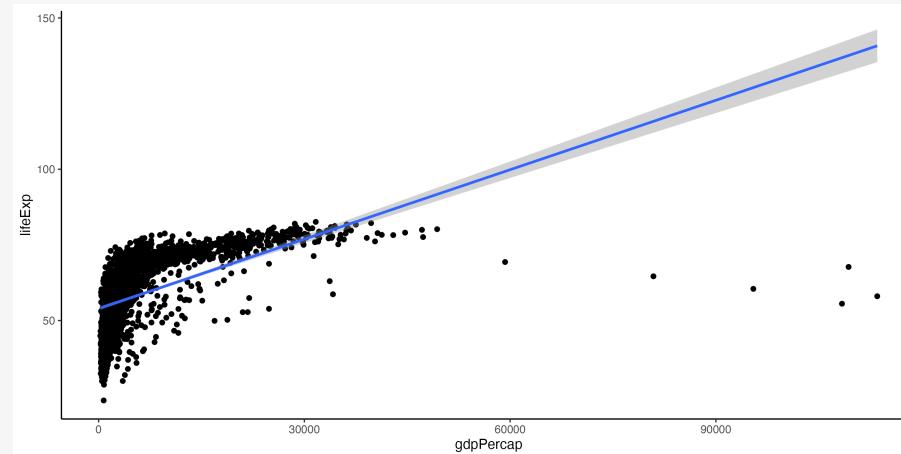
# Fix the labels

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



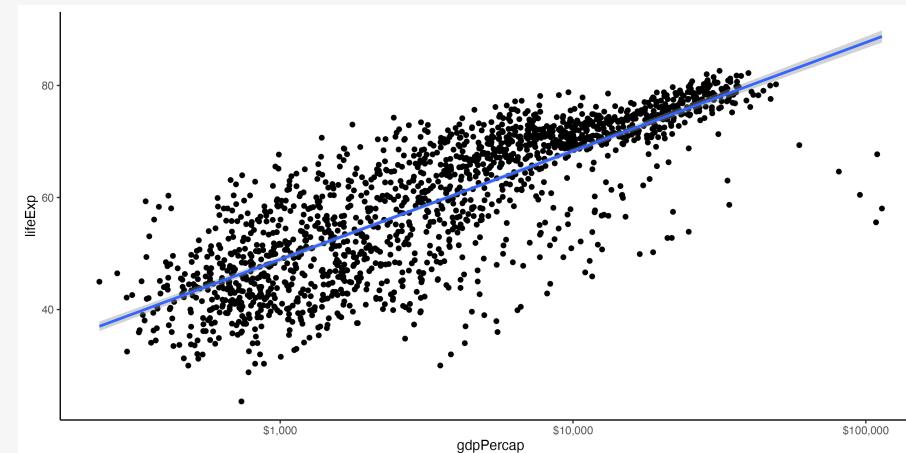
# Fix the labels

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



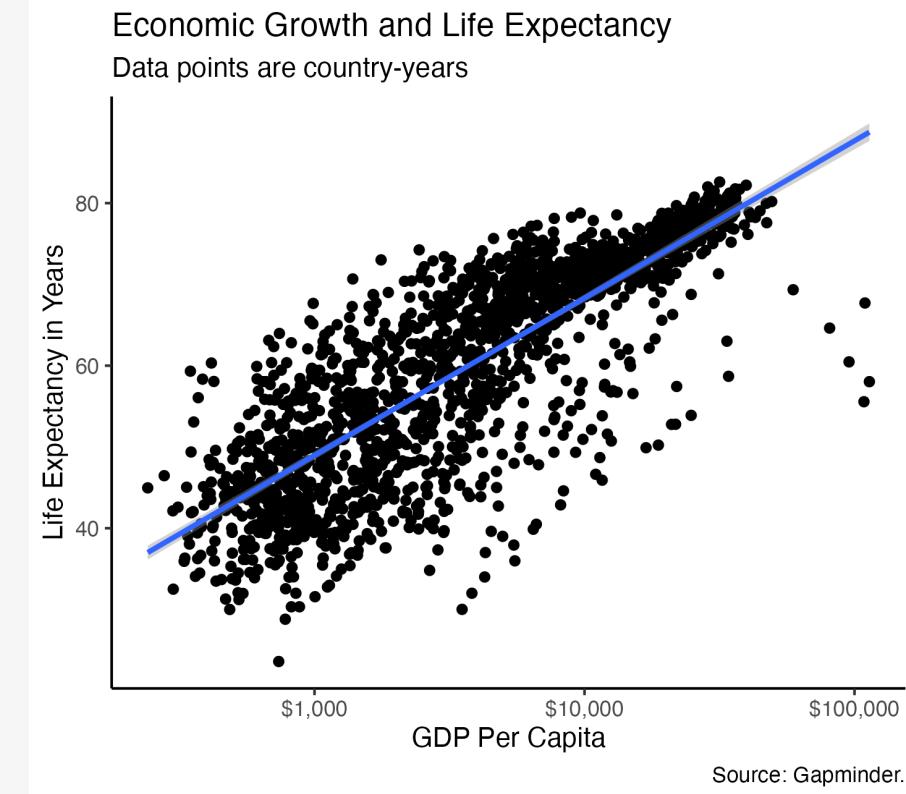
# Fix the labels

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y=lifeExp))  
p + geom_point() +  
  geom_smooth(method = "lm") +  
  scale_x_log10(labels = scales::label_dollar())
```



# Add labels, title, and caption

```
p <- ggplot(data = gapminder,  
             mapping = aes(x = gdpPercap,  
                            y = lifeExp))  
  
p + geom_point() +  
  geom_smooth(method = "lm") +  
  scale_x_log10(labels = scales::label_dollar()) +  
  labs(x = "GDP Per Capita",  
       y = "Life Expectancy in Years",  
       title = "Economic Growth and Life Expectancy",  
       subtitle = "Data points are country-years",  
       caption = "Source: Gapminder.")
```



# Mapping vs Setting your plot's aesthetics

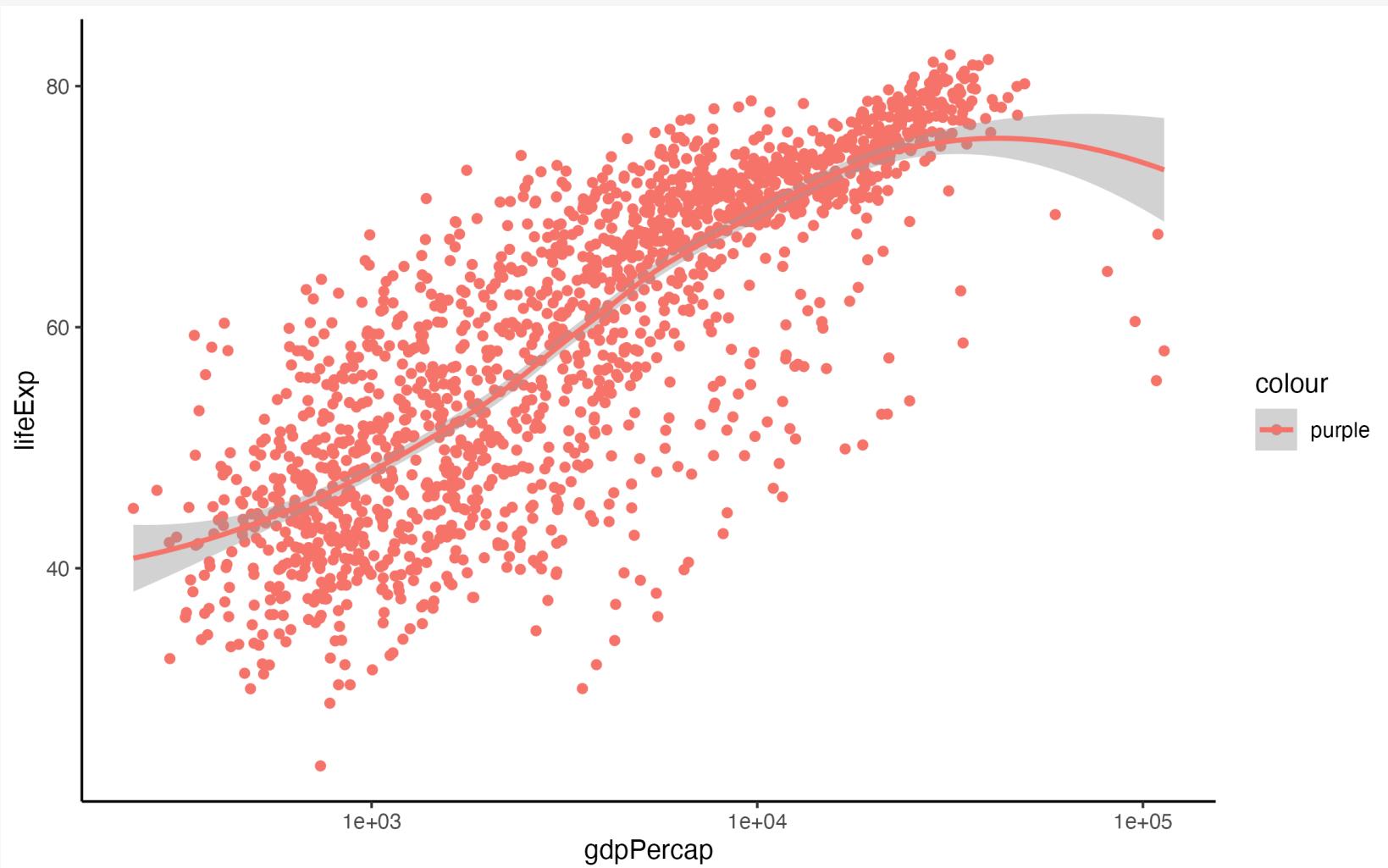
# “Can I change the color of the points?”

```
p ← ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                           y = lifeExp,
                           color = "purple"))

## Put in an object for convenience
p_out ← p + geom_point() +
  geom_smooth(method = "loess") +
  scale_x_log10()
```

# What has gone wrong here?

p\_out

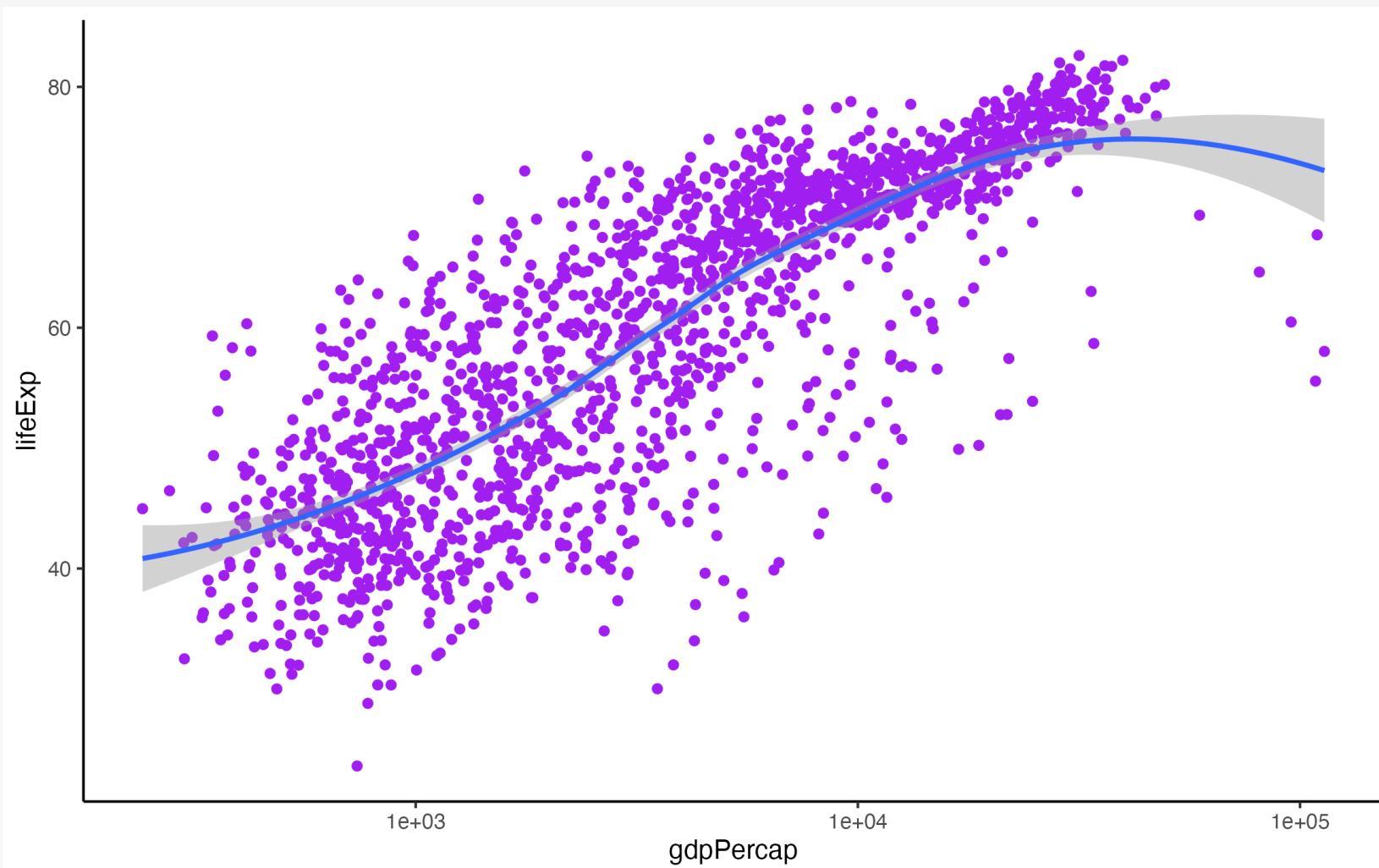


# Try again

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp))  
  
## Put in an object for convenience  
p_out ← p + geom_point(color = "purple") +  
       geom_smooth(method = "loess") +  
       scale_x_log10()
```

# Try again

p\_out



# Geoms can take many arguments

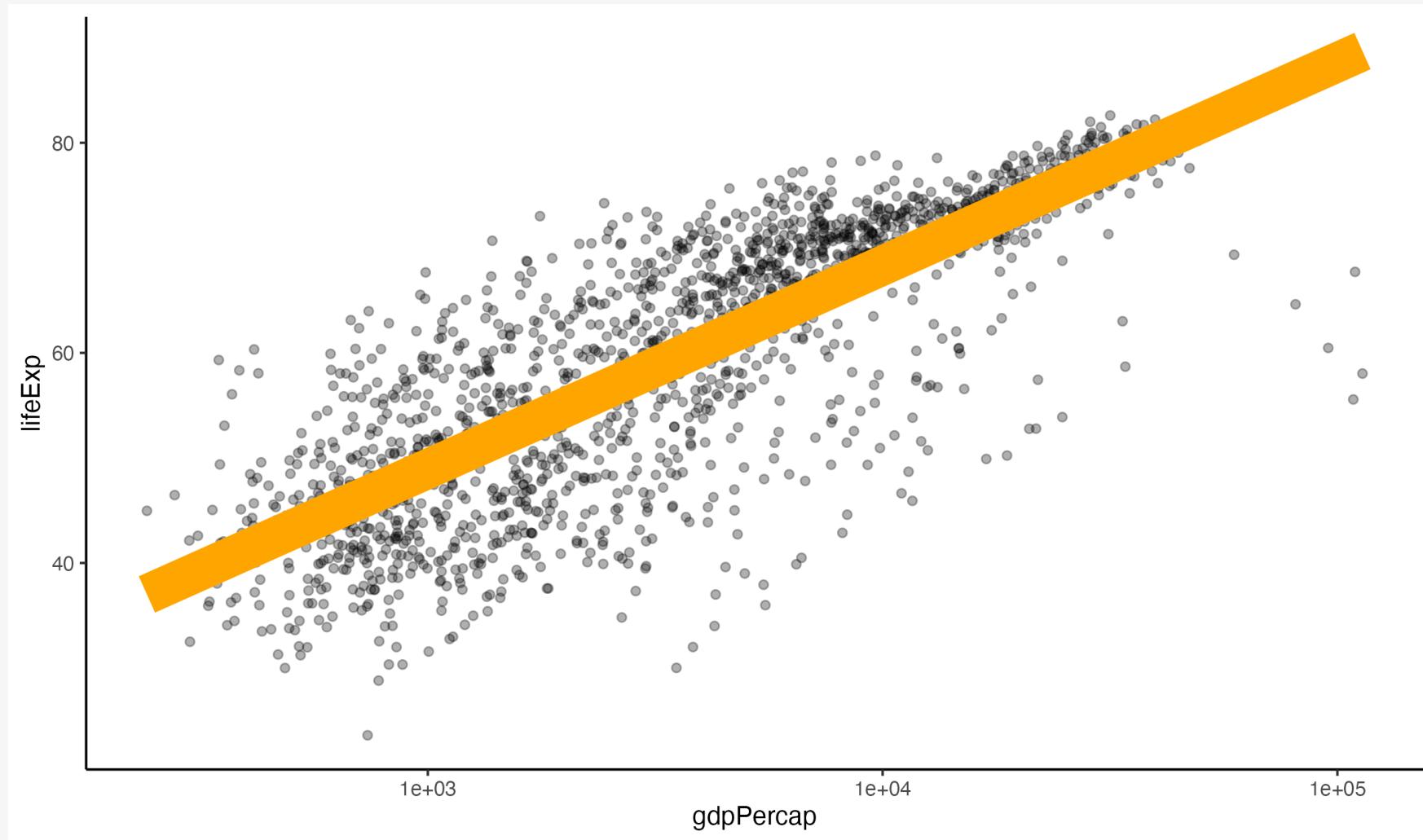
Here we set `color`, `size`, and `alpha`. Meanwhile `x` and `y` are mapped.

We also give non-default values to some other arguments

```
p ← ggplot(data = gapminder,
            mapping = aes(x = gdpPercap,
                           y = lifeExp))
p_out ← p + geom_point(alpha = 0.3) +
  geom_smooth(color = "orange",
              se = FALSE,
              linewidth = 8,
              method = "lm") +
  scale_x_log10()
```

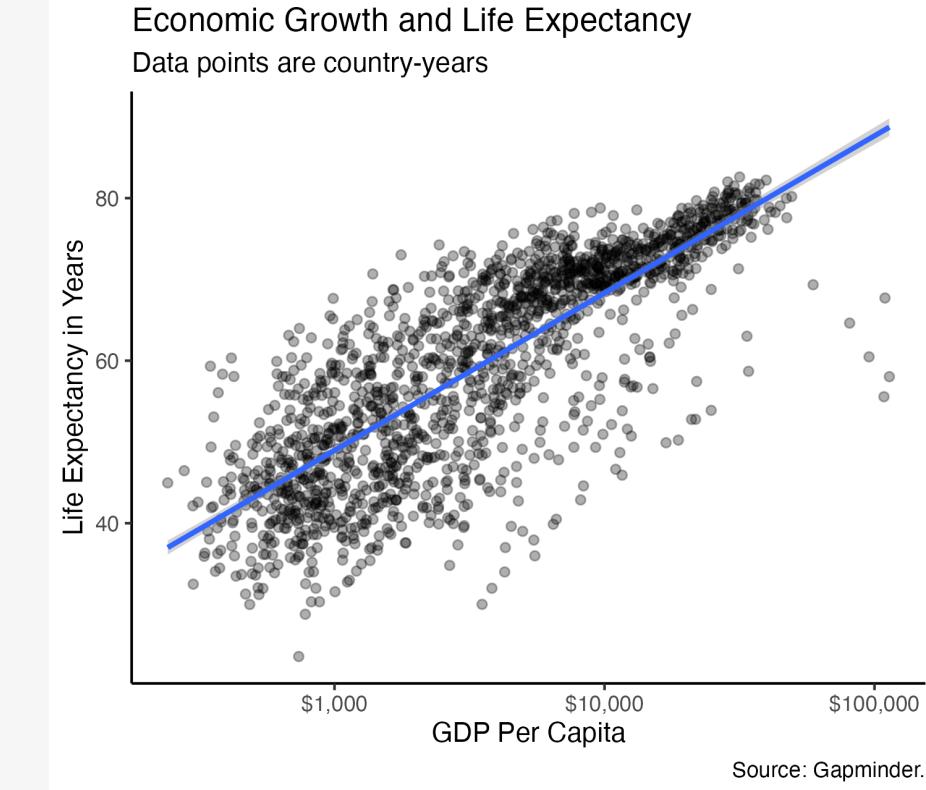
# Geoms can take many arguments

p\_out



# alpha for overplotting

```
p <- ggplot(data = gapminder,  
             mapping = aes(x = gdpPercap,  
                           y = lifeExp))  
p + geom_point(alpha = 0.3) +  
  geom_smooth(method = "lm") +  
  scale_x_log10(labels = scales::label_dollar()) +  
  labs(x = "GDP Per Capita",  
       y = "Life Expectancy in Years",  
       title = "Economic Growth and Life Expectancy",  
       subtitle = "Data points are country-years",  
       caption = "Source: Gapminder.")
```



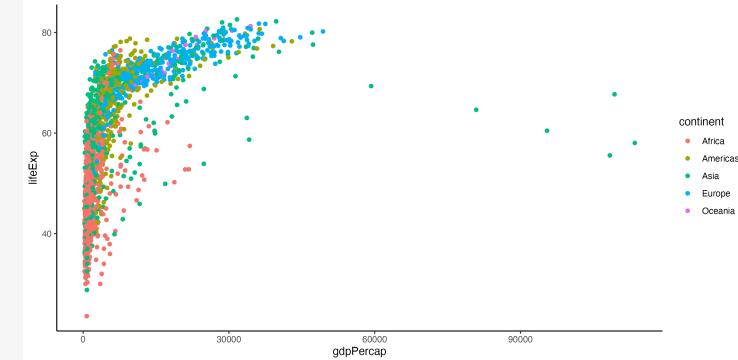
Map or Set values  
per geom

# Geoms can take their own mappings

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp,  
                           color = continent,  
                           fill = continent))
```

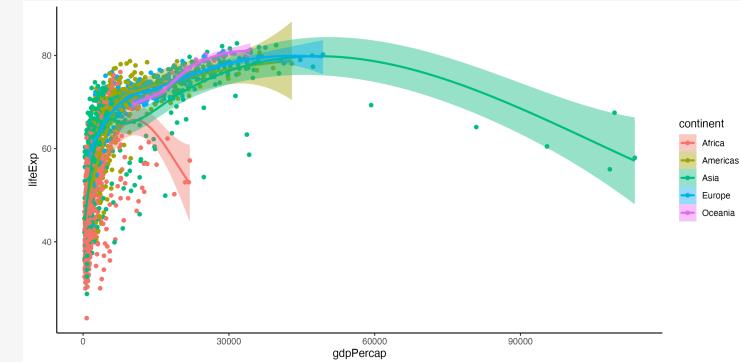
# Geoms can take their own mappings

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp,  
                           color = continent,  
                           fill = continent))  
p + geom_point()
```



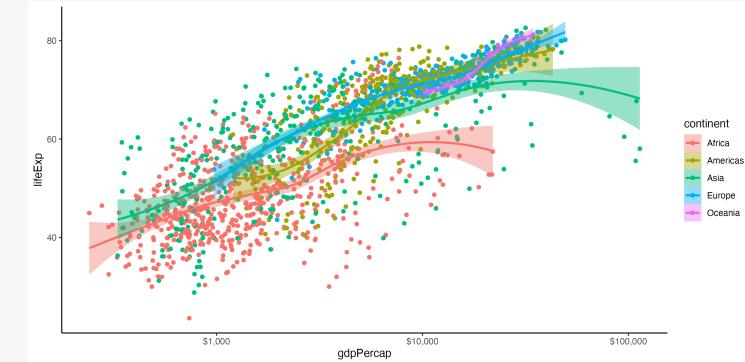
# Geoms can take their own mappings

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp,  
                           color = continent,  
                           fill = continent))  
p + geom_point() +  
  geom_smooth(method = "loess")
```



# Geoms can take their own mappings

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp,  
                           color = continent,  
                           fill = continent))  
p + geom_point() +  
  geom_smooth(method = "loess") +  
  scale_x_log10(labels = scales::label_dollar())
```

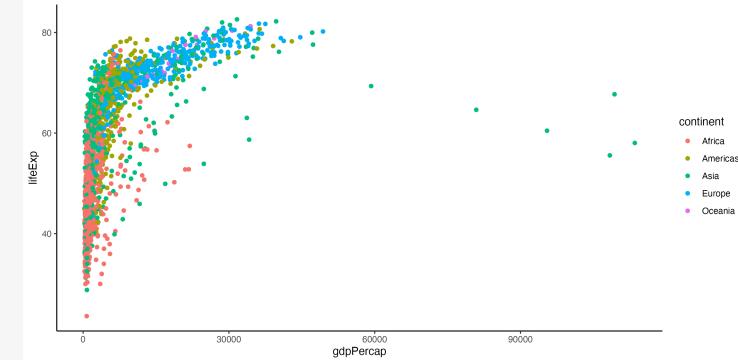


# Geoms can take their own mappings

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp))
```

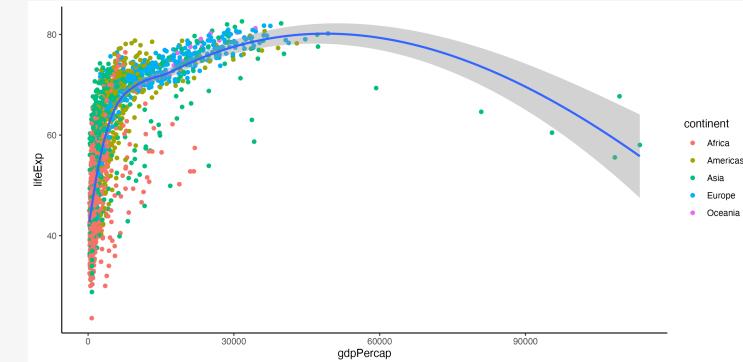
# Geoms can take their own mappings

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



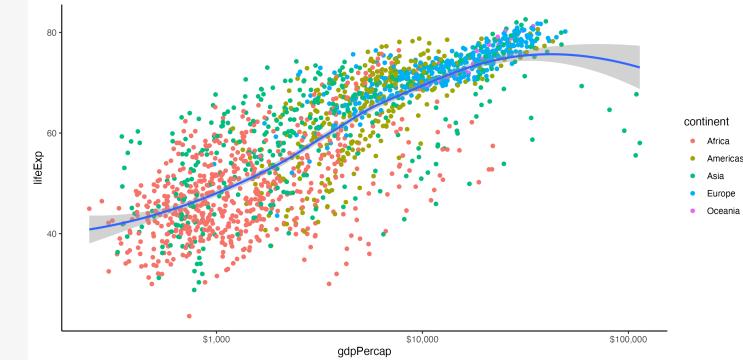
# Geoms can take their own mappings

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp))  
p + geom_point(mapping = aes(color = continent)) +  
  geom_smooth(method = "loess")
```



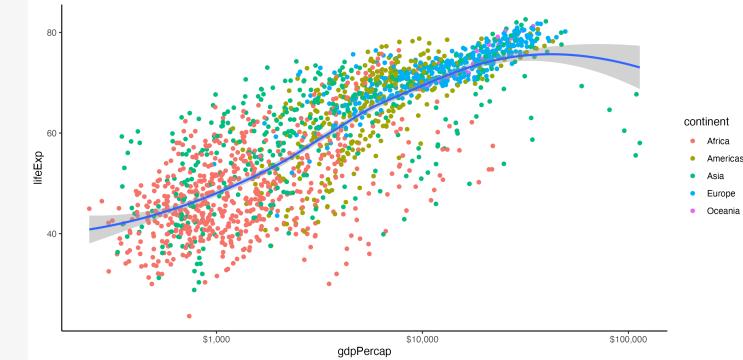
# Geoms can take their own mappings

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp))  
p + geom_point(mapping = aes(color = continent)) +  
  geom_smooth(method = "loess") +  
  scale_x_log10(labels = scales::label_dollar())
```



# Geoms can take their own mappings

```
p ← ggplot(data = gapminder,  
            mapping = aes(x = gdpPercap,  
                           y = lifeExp))  
p + geom_point(mapping = aes(color = continent)) +  
  geom_smooth(method = "loess") +  
  scale_x_log10(labels = scales::label_dollar())
```



Pay attention to  
which scales and  
guides are drawn,  
and why

# Guides and scales reflect `aes()` mappings

```
mapping = aes(color =  
continent, fill = continent)
```

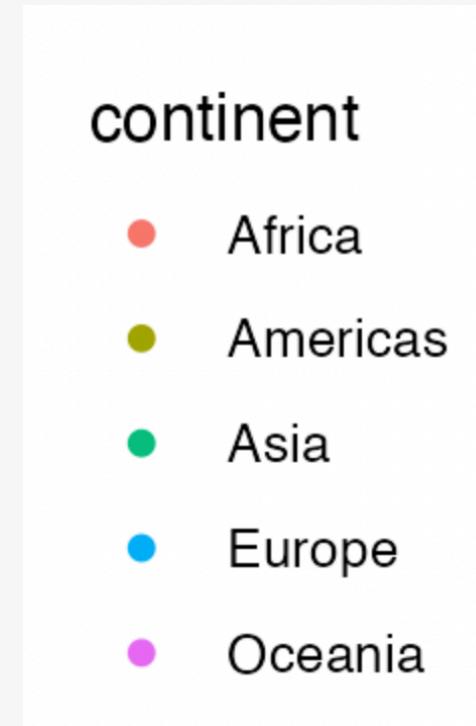


# Guides and scales reflect `aes()` mappings

```
mapping = aes(color =  
continent, fill = continent)
```



```
mapping = aes(color =  
continent)
```



**Remember: Every  
mapped variable  
has a scale**

# Saving your work

# Use ggsave()

```
## Save the most recent plot
ggsave(filename = "figures/my_figure.png")

## Use here() for more robust file paths
ggsave(filename = here("figures", "my_figure.png"))

## A plot object
p_out ← p + geom_point(mapping = aes(color = log(pop))) +
  scale_x_log10()

ggsave(filename = here("figures", "lifexp_vs_gdp_gradient.pdf"),
       plot = p_out)

ggsave(here("figures", "lifexp_vs_gdp_gradient.png"),
       plot = p_out,
       width = 8,
       height = 5)
```

# In code chunks

Set options in any chunk:

## RMarkdown Style

```
```{r, fig.height=8, fig.width=5, fig.show="hold", fig.cap="A caption"}  
gapminder %>  
  ggplot(mapping = aes(x = gdpPercap, y = lifeExp)) +  
  geom_point()  
```
```

## Quarto Style

```
```{r}  
#| fig-height: 8  
#| fig-width: 5  
#| fig-show: "hold"  
#| fig-cap: "A caption"  
  
gapminder %>  
  ggplot(mapping = aes(x = gdpPercap, y = lifeExp)) +  
  geom_point()  
```
```

# Or for the whole Rmd document:

```
knitr::opts_chunk$set(warning = TRUE,  
                      message = TRUE,  
                      fig.retina = 3,  
                      fig.align = "center",  
                      fig.asp = 0.7,  
                      dev = c("png", "pdf"))
```

# Getting Help

The name of the function, and the library it is in.

What it does.

More details on each named argument. This will tell you what class of thing each argument has to be—an object, a number, a data frame, a logical value, etc.

What the function returns—i.e., the result of whatever operation or calculation it performs. This can be a single number, as here, or a multi-part object such as a list, a data frame, a plot, or a model.

R Documentation

**Arithmetic Mean**

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)  
## Default S3 method:  
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

- x An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for `trim = 0`, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning. If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

[weighted.mean](#), [mean](#), [POSIXct](#), [colMeans](#) for row and column means.

Examples

```
x <- c(0:10, 50)  
xm <- mean(x)  
c(xm, mean(x, trim = 0.10))
```

Self-contained examples that you can run at the console. These may use built-in datasets or other R functions.

[Package `base` version 3.4.3 [Index](#)]

Other related functions

Visit the package's Index page to look for Demos and Vignettes detailing how it works.

## How to read an R Help page

The name of the function, and the library it is in.

mean {base}

R Documentation

## Arithmetic Mean

What it does.

### Description

Generic function for the (trimmed) arithmetic mean.

### Usage

mean(x, ...)

## Default S3 method:

mean(x, trim = 0, na.rm = FALSE, ...)

More details on each named argument. This

### Arguments

How to read an R Help page

**[** mean {base}

R Documentation

## Arithmetic Mean

### Description

**[** Generic function for the (trimmed) arithmetic mean.

### Usage

```
mean(x, ...)
```

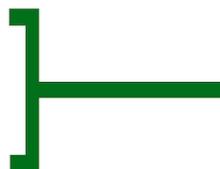
```
## Default S3 method:  
mean(x, trim = 0, na.rm = FALSE, ...)
```

### Arguments

**[** **x** An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for `trim = 0`, only.

**trim** the fraction (0 to 0.5) of observations to be trimmed from each end of **x** before the mean is computed. Values of **trim** outside that range are taken as the nearest endpoint.

**na.rm** a logical value indicating whether `NA` values should be stripped before the computation



The function's name, and in the parentheses the named arguments it expects, in the order it expects them. If an argument has a default value, it is shown. Arguments without default values (e.g. **x**) must be provided by you.

How to read an R Help page

| Description  |  |
|--|--|
| What it does.  | Generic function for the (trimmed) arithmetic mean.  |
| More details on each named argument. This will tell you what class of thing each argument has to be—an object, a number, a data frame, a logical value, etc. | <p><b>Usage</b></p> <pre>mean(x, ...)  ## Default S3 method: mean(x, trim = 0, na.rm = FALSE, ...)</pre> <p><b>Arguments</b></p> <ul style="list-style-type: none"> <li>x An R object. Currently there are methods for numeric/logical vectors and <a href="#">date</a>, <a href="#">date-time</a> and <a href="#">time interval</a> objects. Complex vectors are allowed for <code>trim = 0</code>, only.</li> <li>trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.</li> <li>na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.</li> <li>... further arguments passed to or from other methods.</li> </ul> |
| What the function returns—i.e., the result of whatever operation or calculation it performs. This can be a single number, as                                 | <p><b>Value</b></p> <p>If <code>trim</code> is zero (the default), the arithmetic mean of the values in <code>x</code> is computed, as a numeric or complex vector of length one. If <code>x</code> is not logical (coerced to numeric), numeric (including integer) or complex, <code>NA_real_</code> is returned, with a warning.</p> <p>If <code>trim</code> is non-zero, a symmetrically trimmed mean is computed with a fraction of <code>trim</code> observations deleted from each end before the mean is computed.</p>   |

How to read an R Help page

object such as a list, a data frame, a plot, or a model.

## References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

## See Also

[weighted.mean](#), [mean.POSIXct](#), [colMeans](#) for row and column means.

## Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

Other related functions

Self-contained examples that you can run at the console. These may use built-in datasets or other R functions.

[Package *base* version 3.4.3 [Index](#)]

Visit the package's Index page to look for Demos and Vignettes detailing how it works.

How to read an R Help page